Palacký University Olomouc, Faculty of Science,
Department of Geoinformatics

Paris Lodron University Salzburg, Faculty of Natural Sciences,
Department of Geoinformatics

DEVELOPMENT OF A GEOREFERENCED EYE-MOVEMENT DATA CREATION TOOL FOR INTERACTIVE WEB MAPS

Diploma thesis

Minha Noor SULTAN (B.E.)

Supervisor (Palacký University Olomouc)
RNDr. Stanislav POPELKA, Ph.D.

Co-supervisor (Paris Lodron University Salzburg)
Prof. Dr. Josef STROBL

Erasmus Mundus Joint Master Degree Programme
Copernicus Master in Digital Earth
Specialization Track Geovisualization & Geocommunication
Olomouc, Czech Republic, 2021

ANNOTATION

This thesis study focuses on the development of a utility tool for the analysis of eye-tracking data recorded on interactive web maps. The tool simplifies the labour-intensive task of frame-by-frame analysis of screen recordings with overlaid eye-tracking data in the current eye-tracking eco-systems. The tool's main functionality is to convert the screen coordinates of participant's gaze to real world coordinates and allow exports in commonly used spatial data formats. This study explores the existing state-of-art in an eye-tracking analysis of dynamic cartographic products as well as the research and technology aiming at improving the analysis techniques. The product of this thesis, called ET2Spatial, is tested in depth in terms of performance and accuracy. Several use-case scenarios of the tool are demonstrated in the evaluation section, and the capabilities of GIS software for visualizing and analysing eye-tracking data are investigated. The tool and associated pilot studies aim to enhance the research capabilities in the field of eye-tracking in Geovisualization.

KEYWORDS

Utility; eye-tracking; georeferencing; interactivity; user-logging; GIS

Number of pages: 73

Number of appendixes: 4

This thesis has been composed by *Minha Noor SULTAN* for the Erasmus Mundus Joint Master's Degree Program in Copernicus Master in Digital Earth for the academic year 2019/2020 at the Department of Geoinformatics, Faculty of Natural Sciences, Paris Lodron University Salzburg, and Department of Geoinformatics, Faculty of Science, Palacký University Olomouc.

Hereby, I declare that this piece of work is entirely my own, the references cited have been acknowledged and the thesis has not been previously submitted to the fulfilment of the higher degree.

May 20, 2021 Olomouc. Minha Noor SULTAN

I would like to thank my supervisor Dr. Popelka for his timely support and advice throughout my work and my co-supervisor Prof. Strobl for his insightful consultation. I would also like to extend a thank you to the participants who took part in the experiments which helped me evaluate the usability of the tool. In addition, I would like to thank Ondřej Růžička for allowing me to extend his past diploma work as an asset in my current study. Lastly but most importantly I would like to thank my family for supporting my ambition of higher education.

UNIVERZITA PALACKÉHO V OLOMOUCI

Přírodovědecká fakulta

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Sultan MINHA NOOR**
Osobní číslo:	**R200676**
Studijní program:	**N0532A330010 Geoinformatics and Cartography**
Studijní obor:	**Geoinformatics and Cartography**
Téma práce:	**Development of georeferenced eye-movement data creation tool for interactive web maps**
Zadávající katedra:	**Katedra geoinformatiky**

Zásady pro vypracování

The aim of the thesis is to develop a tool that will allow recording eye-movement data during observation of an interactive web map.
The tool will convert the eye-tracker’s coordinates into real-world coordinates. It will allow geolocating the gaze position and ease the analysis of eye-tracking data, which has to be done manually so far.
The functionality of the tool will be proven on the example of several case studies. E.g. free-viewing, task solving with the map, digitalization using the sight etc.
The output of the thesis will be the developed tool as well as the results of case studies.
The student will attach all the collected datasets and all the animations to the thesis in digital form. The student will create a website about the thesis following the rules available on the department's website and a poster about the diploma thesis in A2 format. The student will submit the entire text (text, attachments, poster, outputs, input and output data) in digital form on a storage medium and the text of the thesis in two bound copies to the secretary of the department.

Rozsah pracovní zprávy:	**max. 50 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**tištěná**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., & Van de Weijer, J. (2011). Eye tracking: A comprehensive guide to methods and measures. Oxford: Oxford University Press.

Bojko, A. (2013). Eye tracking the user experience: A practical guide to research. Brooklyn: Rosenfeld Media.

Ooms, K., Coltekin, A., De Maeyer, P., Dupont, L., Fabrikant, S., Incoul, A., . . . Van der Haegen, L. (2015). Combining user logging with eye tracking for interactive and dynamic applications. Behavior research methods, 47(4), 977-993.

Herman, L., Popelka, S., & Hejlova, V. (2017). Eye-tracking Analysis of Interactive 3D Geovisualization. Journal of Eye Movement Research, 10(3), 1-15. doi:10.16910/jemr.10.3.2

Göbel, F., Kiefer, P., & Raubal, M. (2017, May). FeaturEyeTrack: a vector tile-based eye tracking framework for interactive maps. In Societal Geo-Innovation: Short Papers, Posters and Poster Abstracts of the 20th AGILE Conference on Geographic Information Science. Wageningen University and Research (pp. 9-12).

Vedoucí diplomové práce: **RNDr. Stanislav Popelka, Ph.D.**
Katedra geoinformatiky

Datum zadání diplomové práce: **9. listopadu 2020**
Termín odevzdání diplomové práce: **6. května 2021**

L.S.

doc. RNDr. Martin Kubala, Ph.D.
děkan

prof. RNDr. Vít Voženílek, CSc.
vedoucí katedry

V Olomouci dne 14. prosince 2020

CONTENT

LIST OF ABBREVIATIONS

Abbreviation	Meaning
ET	Eye-Tracking
API	Application Programming Interface
CRS	Coordinate Reference System
GCS	Geographic Coordinate System
PCS	Projected Coordinate System
GUI	Graphical User Interface
JSON	Java Script Object Notation
SHP	Shapefile
CSV	Comma Separated Values
EXE	Executable file
IPYNB	IPython Notebook
XML	Extensible Markup Language
CMD	Command Line
IDE	Integrated Development Environment
DF	Data Frame
MT	MapTrack
DBF	Data Base File
SHX	Shape Index Format Extension

INTRODUCTION

Eye-tracking in the field of cartography has been active in the last few decades particularly as the main pillar of cognition in geographic visualization. Usability analysts employ gaze locations as a mechanism to derive insights into where the user focuses more while reading a stimulus. This stimulus tends to be mostly static maps when it comes to cartography. With the evolution of techniques in geovisualization and technology itself, however, these stimuli have also evolved over time from analog to digital and from static to dynamic. The interactive stimuli hence pose their own challenges when it comes to evaluating their usability through eye-tracking techniques.

Interactive online maps have proliferated in recent decades to a very high extent of usage frequency. Not only do these correspond to standout navigation platforms such as Google Maps or Mapy.cz but also to numerous everyday web applications making use of web mapping services and APIs. As such, evaluation of these interactive stimuli in an easy and efficient way, from a human cognition perspective becomes an important topic to explore.

Unlike static maps, the depicted content of the map in an interactive environment is dependent on various factors which change drastically as soon as interaction is performed, such as pan or zoom. Although the present-day eye-tracking systems provide tools for the evaluation of an animated medium, it is not efficient for the evaluation of a web map for a variety of reasons. First, the output reflects the screen coordinates through a recording of the user's interaction with the map. And to do a detailed analysis of the features and variables on the map at different scale levels would be very labor-intensive as it would require frame-by-frame evaluation. In addition, these screen coordinates do not have the capability to provide intuition from a spatial perspective. Secondly, in most usability studies, more than one participant is involved in experimentation since the map-reading task can vary between different individuals. Comparison of data from different users' eye-tracking experiments is only possible through exhaustive manual efforts in traditional eye-tracking software.

1 OBJECTIVES

The aim of the thesis is to develop a tool that will allow recording the eye-movement data observed during interaction with a web map and store them as spatial data. The main functionality of the tool will be to convert user's gaze positions on a map to real-world coordinates. The tool will allow easier analysis of eye-tracking data in the realm of interactive online maps. Through individual spatial data exports from the tool, the visualization and comparison of multiple user's data simultaneously on a basemap will be made more feasible. In summary, the project aims to simplify the task of eye-tracking evaluation techniques on web maps by creating an application that takes factors like zoom level and pan operations on the map into account.

The ambition is to build a tool that can extend previous research and development at the Department of Geoinformatics, Palacký University Olomouc, and which suits the existing setup of the Eye-tracking lab but at the same time enable room for additional functionalities and allow independent cost-free usage by researchers under a public license.

A part of this study is to explore existing solutions pertaining to eye-tracking and usability testing for interactive web maps, particularly automated solutions, and their distinction in comparison to the current study. The study will touch on different approaches for coordinate conversions as well as methods used for accuracy assessment.

The tool will be evaluated in terms of functionality as well as the quality of output. The usability of the tool will be tested through the example of multiple visualizations and case studies. The study will also tackle a non-conventional approach of inspecting eye-movement data as spatial features, subjecting them to traditional spatial operations and comparing them to standard techniques provided by eye-tracking software and hence the applicability of spatial functions on the eye-tracking data will be gauged. The final output will be an open-source desktop application with a simple GUI. A guide will be compiled on the usage of the tool as well as on its scalability in terms of functionality and code.

2 METHODOLOGY

The solution to any problem starts with the identification and detailed research of the subject matter. For the case of this diploma thesis, the discovery process of the problem involved the usage of current software and methodologies available in the eye-tracking laboratory. After the assessment of the techniques in place for the evaluation of web maps, the problem was identified formally, and the overall concept of the study was formulated. The introduction section explains the context in detail. The next step was to recognize the group, concerned with the subject at hand, as end users and enlist their requirements. This is where the specifics of the solution were consulted with the supervisor. Decisions regarding the type of the end product, the inputs, and the outputs were made. Although unconventional, the research on the technicalities of existing automated solution(s) such as applications was done after setting up a framework for this study which resulted in a benefit that no direct inspiration was taken from it rather it was used to provide insight on a similar subject with different approach.

2.1 Solution Process

An executable file with a graphical user interface was chosen for the end product instead of a command-line interface application. Considering the scope and designated time for the thesis, the tool was planned to be compatible with the well-known technologies first, for inputs and outputs. In order to start programming the tool, sample data was needed to work with. The next section on Data Collection elaborates on how this sample data was collected.

A scripting environment was then set up by installing the appropriate technologies on the local system. The used technologies are further mentioned in section 2.3. Based on the designed architecture of the system, the first task was to synchronize all the collected datasets and determine the most efficient methodology for that. Before that, each of the datasets were pre-processed and organized through the code. The eye-tracking data, after synchronization with the user interaction data, was compiled in one place. Several logics and formulas were separately tested in a script for the conversion of screen coordinates to geographical coordinates. On each of these approaches, testing was performed to check which one had the highest accuracy and which was later adopted for the tool itself. The next task focused on determining suitable methods for the conversion of the new data files to the desired output file formats.

Once the script was functional and giving expected output, a GUI was designed. The tool was given the name *'ET2Spatial'* accompanied by the design of a logo. The GUI was programmed and stitched to the main code and compiled afterward as a stand-alone exe file.

The final objective, which was to demonstrate use cases, was done by collecting more participants' data in the eye-tracking lab and processing this data in GIS software.

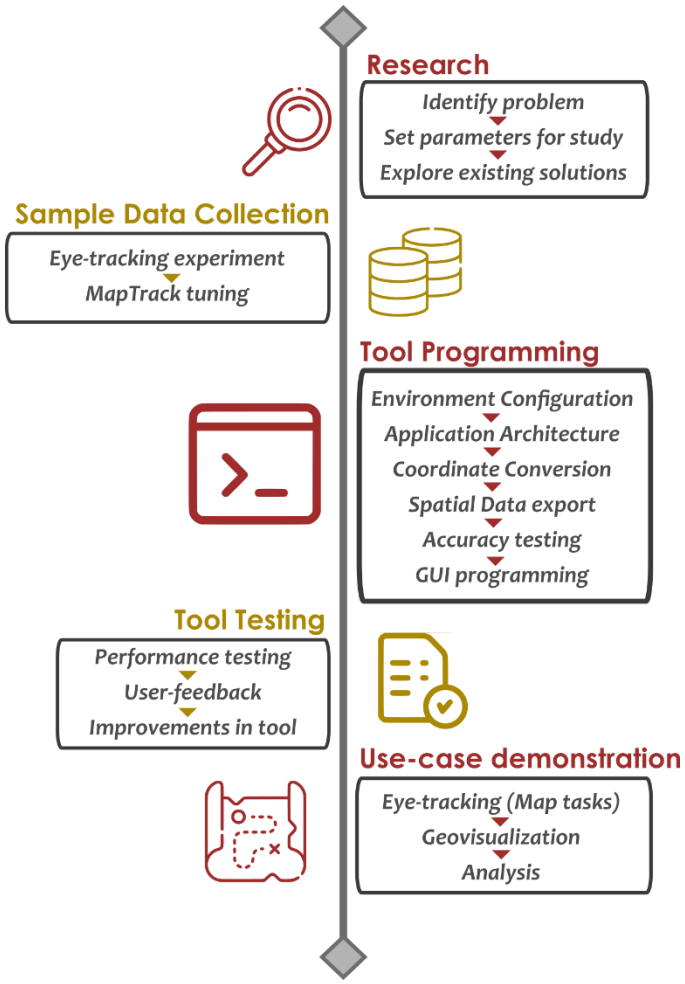


Figure 1 Process Timeline.

2.2 Data Collection

The programming of the tool required initial data to work with, which consisted of eye-tracking data and the user interaction data. This sample data was collected for one participant only.

MapTrack

User interaction refers to the tactile input by the user, such as key presses, mouse clicks, zoom, or pan. The user interaction data was exported through MapTrack. MapTrack is an online application developed at the Palacký University by Ruzicka (2012). The application logs basic map actions such as map center coordinates, zoom level, and time.

Eye-tracking

The initial experiment was done in the eye-tracking lab at the Department of Geoinformatics, Palacký University. The data was recorded through SMI RED 250 eye-tracker and was exported through the SMI BeGaze software. The data mainly consisted of two exports pertaining to the raw data points, and the fixation data points.

2.3 Used technologies

For the creation of the tool, multiple technologies were used. ET2Spatial was made in python along with several supporting modules as enlisted below:

Python

Python is a high-level interpreted programming language. This language was selected as a programming medium because of familiarity, its simpler learning curve, and its large pool of third-party packages for all kinds of disciplines. Python version 3.8 was installed on the local system.

Pip

Pip is a tool that helps in the easy installation of python packages. It usually comes with the python language installation and is called through cmd. All the python packages mentioned in the technologies were installed through pip.

Pandas

Pandas is a python package that allows for efficient data manipulation and analysis. Since this study was focused on manipulation of eye-tracking and user interaction data, Pandas was an effective resource in building of the tool.

NumPy

NumPy is a python library that brings high-level functionality for arrays and mathematical functions in the python code. For the thesis, NumPy was employed to execute mathematical formulas in the code.

xmltodict

xmltodict is another module in python that simplifies parsing and working with xml data structures. The version installed by pip and used by the tool was xmltodict 0.12.0.

PyShp

PyShp is the python shapefile library that allows for ESRI shapefile reading and writing. The PyShp version 2.1.3 was used. It was employed for the file format conversion and export.

geojson

Geojson is a python library that helps encode and decode GeoJSON data. Geojson version 2.5.0 was used for this project and was also employed in one of the file exports for the ET2Spatial tool.

PyQT5

PyQT5 is a python binding for QT v5, a comprehensive set of C++ libraries that enable modern mobile and desktop interactive functionalities. In this study, PyQT5 was used to develop a GUI. PyQT5 version 5.15.2 was installed through pip and used in code.

PyInstaller

PyInstaller is a python module that packages the python script into an executable file by reading all the dependencies. This module was used for generating an exe file after the code was finalized. It was also installed through pip.

Others

Some other modules that were used in the code namely; *math, os, xml.etree* did not need installation as they come inbuilt with python language.

Jupyter Notebook

Jupyter Notebook is an interactive open-source application that allows researchers to create, share and visualize code and data among other things. It runs in the browser and provides a simple lightweight yet powerful platform for scripting. For the current study, Jupyter Notebook was used to create and test different parts of the code as ipynb.

SPYDER

Unlike Jupyter notebook, which can support multiple programming languages, SPYDER is an IDE specifically for python. It can be downloaded for any operating system and usually has well-known inbuilt python packages as well. For this thesis, SPYDER was used to debug and compile the code after scripting and testing in Jupyter notebook. It was particularly used for building a GUI.

QT Designer

QT Designer is a tool that implements PyQT5 framework with a GUI interface and allows users to design layouts in an easy and fast manner. Contrary to coding a layout and visualizing the results each time on compiling the code, QT Designer offers drag and drop capabilities to build a layout structure. For this study, QT Designer was used to conceptualize and implement the GUI of the ET2Spatial tool.

For the second part of this study, the use cases and visualizations, **QGIS version 3.12** and **ArcGIS Pro** were used.

3 STATE OF ART

The number of users and researchers who use eye-tracking systems is growing tremendously. The current technological development in eye-tracking software and hardware is pursuing a trend whereby no advanced technical skills are needed to adopt these systems. As such, the user base is expected to grow even further (Holmqvist et al., 2011). This, combined with the frequency of data generation, requires efficient and faster ways for data analysis. Cartography has been no alien to employing eye-tracking in the usability study of maps. One of the earliest examples dating back to the evaluation of simple drawn maps and aerial images (Enoch, 1959).

The maps as visual stimuli have changed since then, from static to dynamic. Eye-tracking and analysis of eye-movement data on interactive stimuli are costly not only in terms of time but also for data and storage since the standard mechanisms produce video recordings with overlaid gaze points (Pfeiffer, 2012).

Another drawback of traditional eye-tracking mechanisms for interactive mediums is the level of effort required in creation of areas of interest (AOI) during analysis. As opposed to static maps, where the researcher can create polygons on stimuli for easier statistical evaluation. Dynamic AOIs are a possible approach suggested by Holmqvist et al. (2011), where the AOI can be established for non-static stimuli such as screen recording of a map. A tool developed by Papenmeier and Huff (2010) facilitates drawing these dynamic AOIs. However, the rapidly changing content of the interactive web mediums where users perform pan zoom and clicks makes it nearly impractical to manually annotate the areas of interests and makes it very labor-intensive because the video for each participant is different.

Although research in the intersectional domain of eye-tracking and interactive web maps is nowhere near saturated, it is not non-existent. Efforts have been made to understand cognitive processes involving interactive dynamic screen maps, but very little knowledge has been gathered until the last decade. In 2012 a study carried by Ooms et al. deployed a technique for understanding user behavior with interactive maps. This technique involved standard eye-tracking apparatus combined with a joystick for tactile input. The dynamic map, however, was pre-recorded with its pan actions. The recordings were used as stimuli in the eye-tracking experiment, and users logged a signal as soon as they identified the subject. The study combined recordings from both input mediums to measure response times of different user groups and conclude results about interaction. The experiments pertain to larger research aiming at insight into users' cognitive processes while reading interactive maps. Content-dependent analysis, contrary to content-independent analysis for eye-tracking in cartography, needs sophisticated methods for evaluation (Ooms et al., 2012). The study was pioneering for evaluating a dynamic medium; however, a dedicated framework to assess the content was not used.

Gaze coordinates that can be transformed into geo-coordinates can provide more information and feasible solution to the existing issues with interactive web maps (Giannopoulos, Kiefer and Raubal, 2012; Ooms et al., 2015).

Gaze Map Matching was introduced, taking into account the aforementioned issue of content-dependent analysis. The gaze metrics such as gaze sequences and gaze fixation points are studied with respect to the underlying vector data to inspect the geographic features as subjects (Kiefer & Giannopoulos, 2012).

When it comes to non-conventional usage of eye-tracking data in the field of cartography and GIS, a handful of studies have been carried out so far. Considering eye-tracking data as spatial features allows dedicated spatial functions to be applied to the eye-tracking metrics such as scan paths, fixation, and raw gaze points. The traditional methods and tools offered by eye-tracking software, however, lack the capability of studying the gaze data metrics from a spatio-temporal perspective. The dataset has a very similar structure to movements datasets of real-world features in geographic spaces. Hence techniques designed for the evaluation of spatial movement data can also be resourceful in the evaluation of gaze points data (Adrienko et al., 2012).

While these attempts at understanding the content of the interactive mediums have been less, the research on open-source solutions for eye-tracking mechanisms on the dynamic maps has been relatively scarce. One such attempt at an evaluation mechanism of an interactive medium has been for 3D models. 3DgazeR helps in a less cumbersome analysis of interactive 3D models. The tool works by calculating 3D coordinates (x,y,z) for each point of view in a 3D scene. These coordinates are derived from the orientation and location of virtual cameras as well as the screen coordinates of eye movements. The output generates gaze points referenced to the 3D model, which can be visualized in QGIS (Herman, Popelka & Hejlova, 2017). The tool addresses the problem of eye-tracking in an interactive medium but is constrained to a 3D environment.

A Framework suggested by Ooms et al. (2015) captures the essence of the problem in the best way. The study aims to build an application that is compliant with interactive map mediums and logs user data and raw gaze samples. Several approaches are tested in the study to determine which one fares best and can be used independently with a variety of software. The study explores both desktop-based and online-based user data logging solutions and settles on the desktop-based user-logging approach using PyHook as the triumphant one because of its ability to work with a wide array of applications. This approach logged all user mouse clicks and key presses on web pages which were eventually synchronized with eye-tracking data through an imposed mouse-click command. The conversion of the ET screen coordinates to geographic coordinates is done by calculating the map extent, distances, and direction of the succeeding user-interaction movement. The principle that the scale remains constant during pan operation helps in estimating the new center coordinates of the viewing window in reference to the previous one. The mathematical conversion itself is done through forward and inverse map projection formulas. The use-cases focus on applying the methodology in other domains such as marketing, psychology, and traffic science. Although the study is very similar to the current thesis being developed, the proposed methodology differs from ET2Spatial in various ways. Dedicated Map APIs are not taken into account, and the entire referencing procedure is based on detailed user-logging actions such as pan, zoom, scroll, and click, which are

unneeded and far too complex for the scope of this thesis. However, some smaller parts of the proposed technique have been adopted in this tool, such as the imposition of tactile user input for easier synchronization of data. The suggested approach is very effective for a generalized system working with interactive mediums, such as web pages and even static photographs and maps. However, it does not fit the niche objectives of this diploma thesis. In addition, the focus for the use-case of the ET2Spatial tool is for cartographic evaluation and GIS analysis only, which requires a simpler lightweight approach than the one suggested by the study.

FeatureEyeTrack, a tool developed at ETH Zürich, measures the real-world coordinates from a user's screen coordinates for interactive web maps. It is the closest contender to the tool developed in this thesis and shares the same goal of easing analysis of eye-tracking data for dynamic online mediums. The framework involves an eye-tracker, a logger, and a web map service. The logger records all the mouse clicks and user inputs, the extent of the map, and the zoom level are fetched and stored in an SQLite database through a web page featuring an interactive web map. The tool uses Mapbox API, and the main program written in Java which receives the gaze data stream, which is then combined with the user logged data, converted, and stored in the database (Göbel, Kiefer & Raubal, 2017). The concept of FeatureEyeTrack is similar to *ET2Spatial*, developed in this study, but the framework, approach, and used technologies are different in practice (More in section 4). Moreover, FeatureEyeTrack is not available under a public license which created room for similar yet niche tools to be developed by institutions. *ET2Spatial* mainly works in post-processing mode and is independent of database requirements, system settings, and any installations. It builds on free, open-source technologies and components and extends on existing works of students at the department.

4 TOOL CREATION

The architecture of the solution comprises four main modules; eye-tracking module, map interaction module, connection module, and conversion module. Figure 2 provides an illustrative view of the solution.

The eye-tracking module consists of an eye-tracking device connected to a 1920×1200 px screen. This module is responsible for sensing, recording, and exporting gaze locations of the user. The map interaction module consists of a framework for extracting the user's interaction with the web map. In this study, MapTrack (Ruzicka, 2012) is used. However, an application-independent approach is also discussed in section 4.1. The map interaction module outputs the active map coordinates as latitude and longitude. The connection module, which is one of the main components of the tool, is responsible for data synchronization between the ET data and the user interaction data. Post connection module, data gets passed on to the conversion module. The conversion module accounts for the calculation of geographic coordinates from the screen coordinates.

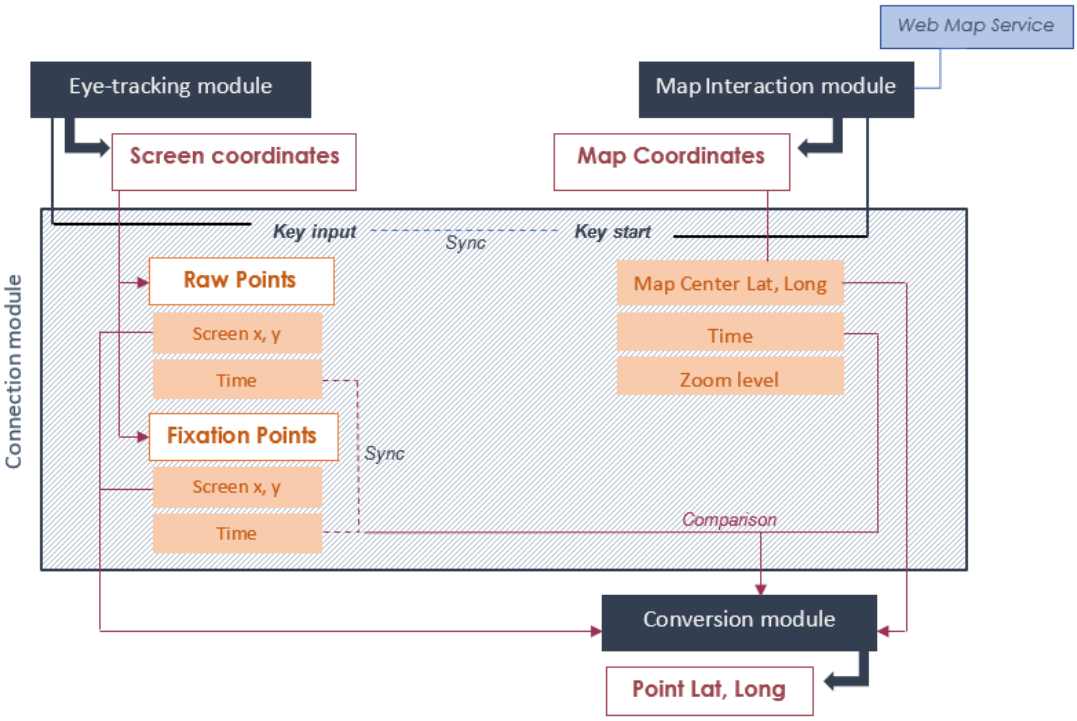


Figure 2 Schema of ET2Spatial tool.

From a user perspective, a typical workflow activity would involve conducting ET experiments on a web map such as Google Maps displayed through the MapTrack application on the screen of an ET setup. The data from both, ET device and the MapTrack application would be fed to the ET2Spatial tool. The tool then converts the points in the datasets to spatial features that can be imported into any GIS software and overlaid for multiple participants on a cartographic basemap. Figure 3 illustrates the said workflow.

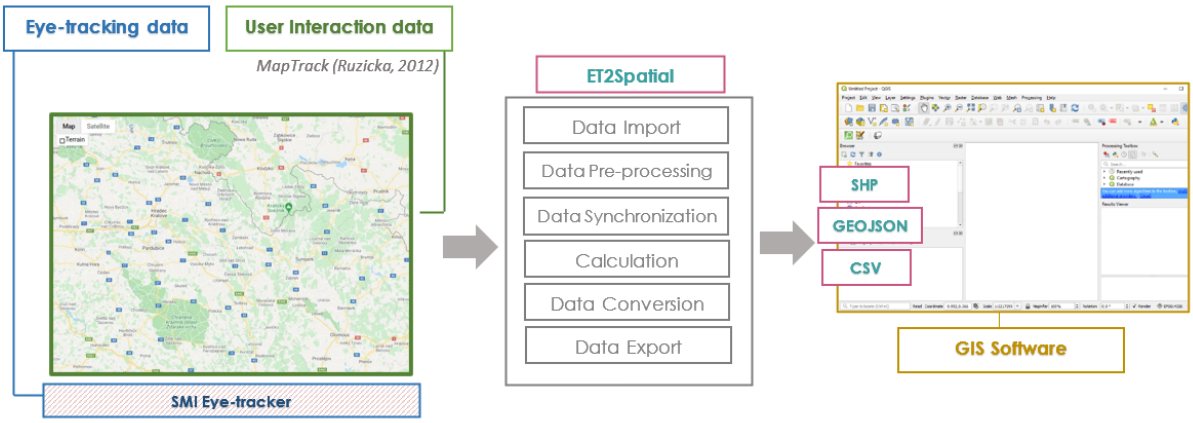


Figure 3 User activity Diagram.

4.1 User Interaction data

User Interaction data implies the logged interaction of a user with a web application or a website. These interactions can be registered by deploying custom JavaScript code on a proxy server to detect different mouse and keyboard events on the client-side. Several dedicated applications and tools are also available for doing these tasks. However, one downside of using this approach is that some web map applications block certain user input events from being registered, for instance, a mouse down event (Ooms et al., 2015).

On the other hand, Web Map APIs can potentially overcome this as they offer inbuilt functions for registering user map events. Since, for the scope of this thesis, the concern is only with the content of the web map itself and not with the layout of the webpage as a whole, it was more practical to use an application that employs web map APIs for registering events. The primary data needed from a user interaction framework, recorded during an ET experiment, was the geographic coordinates of the current center of the web map with an associated timestamp, the current zoom level, and a map pan event registration.

MapTrack (Ruzicka, 2012) was chosen as the apt application for this thesis primarily because of its ability to record the aforementioned web map events on the client side. In addition, MapTrack, being open-source and a product developed at the Department of Geoinformatics, Palacký University Olomouc, allowed access to the source code and hence the possibility of small additions or tweaks if needed to create a harmonized framework with the tool from this diploma thesis.

As briefly discussed in section 2.2, MapTrack is an online application hosted at the domain of the department (http://eyetracking.upol.cz/maptrack/), which creates a registry of user activity with the web map. As of now, the time of development of this thesis study, MapTrack is configured to work with Google Maps only. Figure 4 shows the landing page of the application, which requires a user id. This id is not restrained to a specific

alphabetical or numerical format. However, for the sake of conformity to ET experiments, the id entered in MapTrack was identical to the one used in ET software for each participant.

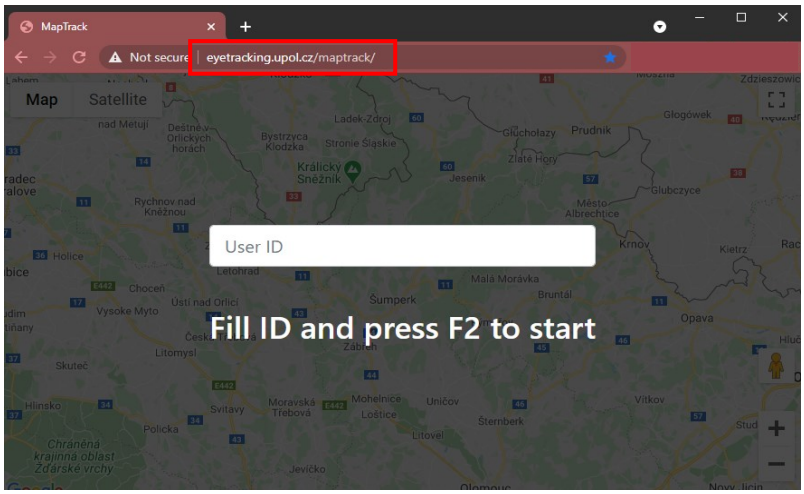


Figure 4 MapTrack Interface.

An initial experiment was done to collect sample data to work with. The MapTrack application was displayed on the screen connected to the ET device and used as a (screen-recording) stimulus. The sample map task involved finding the location of the Department of Geoinformatics, Palacký University Olomouc, on the web map. Every time the user scrolls or pans, the zoom level and map center are changed and registered. Once the interaction of zoom, scroll and pan was recorded. For the sample experiment, it was stored into an xml file that was available to download through the same web domain. Figure 5 shows the result directory as well as the structure of the xml data file. The experiment was done in full-screen mode, which removed the need to filter out ET points outside the web map. The *mapsize*, *var* which refers to participant id, and the *start* time stamp were stored as global variables in the code.

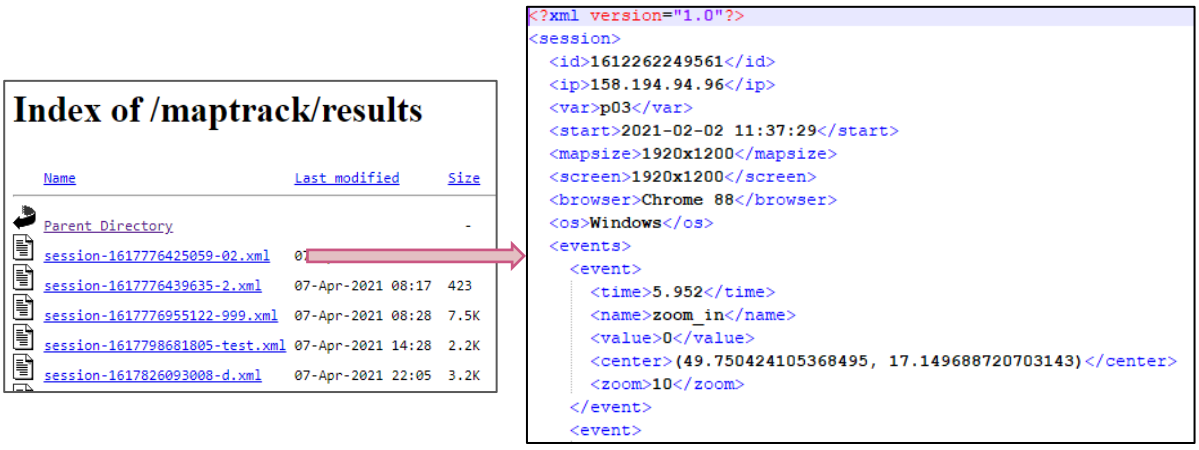


Figure 5 xml file structure.

Through a loop, the xml file was parsed with the help of xml.etree library. Every child element of the *event* was stored into a series which was later organized as pandas dataframe. The columns were renamed for convenience. The *center* element stored as a string, was split for separate lat, long values, and the data format was also changed to float. Time was converted to milliseconds with the correct decimal places. Figure 6 shows the resulting header and sample user interaction data after pre-processing.

	MapTrack_RT	Action	MapCenter	Zoom_level	MapCenter_lat	MapCenter_long
0	5952.000000	zoom_in	(49.750424105368495, 17.149688720703143)	10.0	49.750423	17.149689
1	7566.000000	move	(49.53967593673293, 17.333709716796893)	10.0	49.539677	17.333710
2	8201.000000	zoom_in	(49.53885049359118, 17.317230224609393)	11.0	49.538849	17.317230
3	8451.000000	zoom_in	(49.54241070589379, 17.309000532438578)	12.0	49.542412	17.309000
4	9438.000000	move	(49.559116123519374, 17.339212934782328)	12.0	49.559116	17.339212
5	9940.000000	zoom_in	(49.55265737281604, 17.323591749479593)	13.0	49.552658	17.323591
6	12150.000000	move	(49.57915508376768, 17.267801801960828)	13.0	49.579155	17.267801

Figure 6 Pre-processed user interaction data-header.

A downside of using the mapping APIs for collecting user interaction data is the reliability on vendor-specific functions and hence creating a separate mechanism for every web map such as OSM, Google Maps, or MapBox. Adjacent to the MapTrack solution, for storing user interaction data, an application-independent solution was also researched. Typically web map applications have map center coordinates in the URL, which tends to change every time the user commits an action such as pan or zoom. Additionally, the URL also contains information about zoom levels. This assumption can be made about any web map platform or web map API implemented in REST. The web browser history contains a registry for URLs that can harvest information about the map center coordinates and zoom level at every time stamp.

Although browser extensions can bulk export the history as CSV, an automated solution is through a python module called browserhistory, which fetches the said data for Chrome, Safari, and Firefox in a CSV format. For web browsers, the timestamps are usually noted up to seconds. Depending on the use case that may suffice, but for this thesis, the accuracy was of prime importance; hence the timestamps needed to be in milliseconds. This was one of the main reasons MapTrack was employed instead. However, the potential for using browserhistory module for an application or API independent framework is significant and worth detailed dissection.

4.2 Eye-tracking data

The ET experiment for collecting the sample data involved an SMI RED 250 eye-tracker with a sampling frequency of 250Hz. The data was exported through SMI BeGaze software. For this study, raw gaze points were used as well as identified fixations. The raw

points data stream exported by the eye tracker comprises of triples in the form of screen x, y & t. These triples are typically aggregated spatio-temporally in reference to the fixations. A fixation is registered when the eye rests on a screen location for a certain amount of time compared to saccades which denote the quick eye movements between fixations. The number of properties, as columns, for both raw points and fixations was kept to basic minimum i.e. surplus information such as age, gender etc. was omitted temporarily during export.

4.2.1 Raw gaze points

During the export of raw gaze points file from SMI BeGaze software, timestamp of day, recording time in milliseconds, point of regard x and y, and participant information were deemed necessary to work with. The file was exported from the software as a text file and read through a CSV python library within the script. The file was also converted to a pandas data frame that allowed for easier visualization of data sheets and offered efficient methods to manipulate rows and columns in the data. The pre-processing steps in the script for the raw data points involved:

- Data slicing vertically; this implies column selection. Although the user can select only required columns during ET data export from an ET software, specific columns mentioned above were extracted explicitly through script regardless to reduce active data size in case of a large number of columns in the original export.
- Data slicing horizontally; this implies row selection. The row selection was an important step which played a part in further data synchronization (section 4.3). ET systems have the ability to record key inputs in addition to the main eye-tracking. One advantage of MapTrack initiation and termination mechanism is the 'F2' key press which also gets registered by the eye-tracker in the file output. This input is used to crop the raw points dataset such that only the points recorded during the usage of the web map are considered.
- Renaming; columns were renamed for the sake of convenience. The term *RT* in the script refers to Recording time.
- Cleaning: the text file contained special characters for missing values. The rows with these characters for any one of the columns were removed to avoid errors in later calculations.
- Indexing: many functions in organized data structures such as pandas df rely on proper indexing. The index for the pre-processed raw data was reset after subjection to slicing and deletion operations.
- Time column formatting: After the raw points data was synchronized with the user interaction data through key input, the recording time needed adjustment as well. To achieve that, time length *length1* was calculated by subtracting the initial time stamp in the unsliced data from the timestamp at index 0 in the newly sliced dataset. This length was then subtracted from each entry in the *RT* column to create a formatted time column *Format_RT*.

	TOD	RT	Screen_x	Screen_y	Participant	Format_RT
0	11:37:29:456	8564251.3	1005.9	539.2	P03	1.1
1	11:37:29:460	8564255.3	1006.3	536.9	P03	5.1
2	11:37:29:464	8564259.3	1005.6	536.2	P03	9.1
3	11:37:29:468	8564263.3	1004.1	535.7	P03	13.1
4	11:37:29:473	8564267.7	1001.2	535.6	P03	17.5
...	...	...	...	...	...	...

Figure 7 Pre-processed raw points data-header.

4.2.2 Fixation points

The raw gaze points file is sufficient on its own for an independent export if needed, but the fixations file depends on the time calculations from the raw points file for synchronization to the user interaction data. To prepare the fixations points data following steps were pursued in the code:

- Data slicing vertically: Similar to the procedure in raw points pre-processing, the columns from the original text file were selected to only the necessary ones for processing i.e., Fixation start time, Duration, Fixation position x, y, and participant number.
- Renaming
- Cleaning: Same procedure to remove entries with non-numeric values was applied.
- Indexing: the dataset was assigned new index values using dedicated pandas function
- Time column formatting: The fixation start time in the original fixations file were the relevant time stamps needed for calculations; however, these times were not synchronized to the time window for the actual user web interaction. Only the fixations that occurred during the participant's usage of the web map were needed. It meant that the fixations occurring during the prompt slides and instructions reading had to be excluded. As mentioned, fixations file exported from SMI BeGaze does not include a record of the user's key inputs. Hence the method followed for the raw points file could not be applied here. The time length calculated for the raw points was also utilized here by subtracting it from the fixation start times. Consequently, the negative values from the results were filtered out as it implied those fixations happened before the user initiated MapTrack. These results were stored in a new column *Sync_time.* The data was now synchronized at the same starting time as the user interaction. To remove the fixations that happed after the MapTrack application was closed, another length *length2* was calculated. This value was

computed by subtracting the first and last timestamps of raw gaze points which yielded the total gaze time on a web map. All the values in *Sync_time* that were greater than length2 were filtered out.

	Participant	Fix_start_time	Duration	Fix_x	Fix_y	Sync_time
0	P03	13472.0	294.5	843.8	669.8	349.9
1	P03	13811.1	190.0	699.7	489.6	689.0
2	P03	14068.8	198.9	640.6	388.7	946.7
3	P03	14321.2	218.6	1208.3	739.4	1199.1
4	P03	14604.1	203.1	1079.2	676.3	1482.0
5	P03	14849.1	260.0	900.3	524.6	1727.0
6	P03	15144.7	218.5	845.3	490.1	2022.6

Figure 8 Pre-processed fixations points data-header.

4.3 Data Synchronization (Stitching)

The concept of synchronization is important for this thesis study because it contributes to the temporal and spatial accuracy of the output of the tool. It implies that all the input datasets have the same starting and ending point in terms of time. As mentioned earlier, an imposed tactile user input from the web map recorded by the ET can significantly help. In the raw data file, F2 was used to synchronize it with MapTrack data. The fixations data was synchronized to the MapTrack data through variables in the raw data file. Once these three datasets were pre-processed and synced, the next step was to combine map interaction properties such as map center Lat Long and zoom level to raw data and fixations data individually. This step would stitch the datasets based on time comparisons and be necessary for the coordinate calculation later (section 4.4).

Data stitching required iterating over each row in both the ET raw dataframe(df) and the MT df. Pandas provide efficient data handling functions especially accessing df values through multiple options. However, the iteration over rows on both tables through loops proved to be complicated using in a pandas df. The iterrows() function gives a series in return for every row it is iterated over and which is susceptible to data type changes. In addition, iteration over df is tricky, especially if the values are being modified because the iterrows() returns a copy instead of a view (pandas documentation). Hence, the data frames were converted to dictionary data structures to ease the process of looping and modifying values.

```
subset_et_raw=subset_et_raw.to_dict(orient='records')
```

The dictionary was oriented by ‘records’ which meant that every item would have a column name and its associated value for the specific row. It is a list-like structure

[{column->value}]. Figure 9 shows the dictionary structure of pre-processed ET raw points data.


```
[{'TOD': '11:37:29:456',
  'RT': 8564251.3,
  'Screen_x': '1005.9',
  'Screen_y': '539.2',
  'Participant': 'P03',
  'Format_RT': 1.1000000014901161},
 {'TOD': '11:37:29:460',
  'RT': 8564255.3,
  'Screen_x': '1006.3',
  'Screen_y': '536.9',
  'Participant': 'P03',
  'Format_RT': 5.100000001490116},
 {'TOD': '11:37:29:464',
  'RT': 8564259.3,
```


Figure 9 Dictionary structure

Figure 10 shows the logic for the stitching operation. Every row in the time column of the fixations data table is compared to the corresponding row in the time column of the MapTrack data table. The *sync_time* (see Fig 8) values are compared to the MapTrack_RT (see Fig 6). If the time in row i of fixations data is greater than or equal to the time in row i of user interaction data but less than the time of the succeeding row in the user interaction data table, the Map center coordinates and zoom level from the MT table get appended to the fixations table for that row.

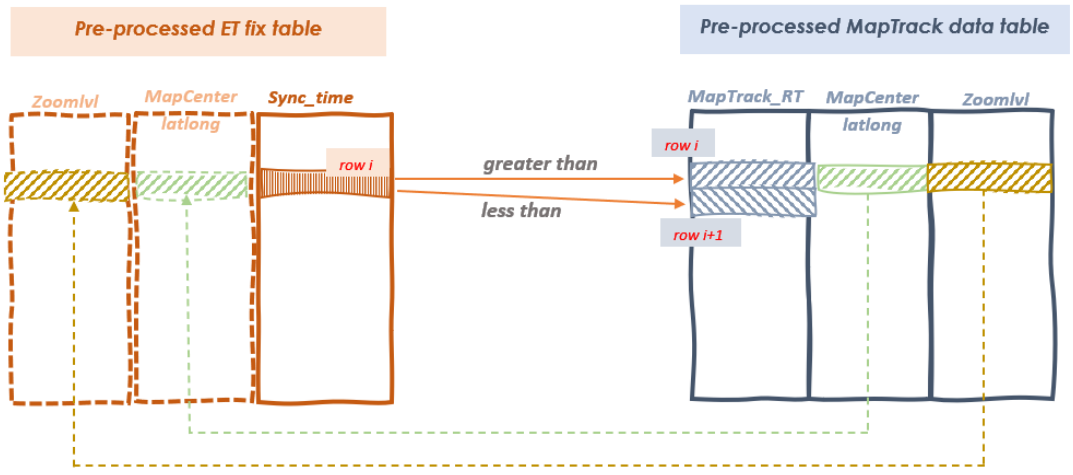


Figure 10 Data stitching schema

The logic is implemented in code as follows:

```
combined1 = []
idx = 0
for r1 in subsetxml_mt:
    r1_maptrack = float(r1['MapTrack_RT'])
    for idx2, r2 in enumerate(subset_et_raw[idx:], start=idx):
        r2_sync = float(r2['Format_RT'])
```

```
        if r2_sync > r1_maptrack:
            idx = idx2
            break
        combined1.append({ **r1, **r2})
```

An empty table is declared for adding map center Lat Long for every time stamp in raw table. The first loop is iterating over the MT data table, which starts from an index 1 initially set to 0. The next loop is also starting from index 1 and using is the index values of the raw table as index 2 for iteration, but it breaks as soon as the condition is satisfied, and the index moves to the next value in the MT data table. The values are appended to the empty table at the end of second loop. The resulting dictionary is converted back to a data frame, and the columns are formatted for the appropriate data type. This process was done for the fixations data as well.

4.4 Data Conversion

Once the raw points data table and the fixation points data table were populated with a zoom level and map center coordinates for every time stamp, the next step was to calculate the real-world point coordinates. Several approaches were researched and tested for the said calculations, out of which only the ones with higher accuracy and relevancy have been discussed here.

4.4.1 Approach 1

The first approach revolved around finding the screen distance between the gaze point and the screen center. The screen center was calculated through screen size stored during the MT data import. The screen distance was then multiplied by distance per pixel to get the distance in meters. Distance per pixel was calculated using existing formulae found online. One of the sources was Open Street Map documentation. The meters per pixel formula takes into account the latitude at which the point is located. Since the latitude of the ET gaze point was the subject in question and unknown, the known map center latitude was substituted in the formula instead. After calculating distance in meters, the output was converted to decimal degrees and added to the map center latitude and longitude.

The resulting coordinates obtained through this method were off by one degree. In practice, especially at smaller zoom levels, this would lead to inaccuracies in visualizations and analyses. This inaccuracy in calculation can be accounted to the fact that the latitude of the desired location is needed to calculate meters per pixel; since that is unknown and the whole subject of this study, the map's center latitude is used, which can be significantly different from the latitude at gaze location on smaller zoom levels.

Some mapping libraries provide inbuilt functionalities for getting distance per pixel, such as the *this.getResolution* function in OpenLayers.

```
dist_x_px=abs(screen_x-center_x)
dist_y_px=abs(screen_y-center_y)
```

```
metersPx = 156543.03392 * np.cos(np.deg2rad(mapcenter_lat)) / pow(2,
                          zoomlvl) #formula by Google
metersPx2 = 40075016.686 * np.cos(np.deg2rad(mapcenter_lat)) / pow(2,
                          zoomlvl+8)#formula by OSM

dist_x_m=dist_x_px * metersPx
dist_y_m=dist_y_px * metersPx

Dist_dd_x=dist_x_m/(111.32*1000*np.cos(np.deg2rad(mapcenter_lat)))
Dist_dd_y=dist_y_m/(111.32*1000*np.cos(np.deg2rad(mapcenter_lat)))

point_long=mapcenter_long-Dist_dd_x
point_lat=mapcenter_lat+Dist_dd_y
```

4.4.2 Approach 2

The second approach used map extent coordinates and screen coordinates of points as inputs. The idea behind this approach was the transformation of the points from one coordinate system to another coordinate system, i.e. from the screen coordinate system to the geographical coordinate system. The formula remaps values from one range to another. The maximum and minimum values of both the screen and the map were used in addition to the screen point coordinate.

The output was then latitude for the y screen coordinate and longitude for x screen coordinate as input. The y coordinate had to be adjusted since the origin of a traditional screen coordinate system is on the top left, and hence the y values progress in a direction opposite to a geographic coordinate or cartesian coordinate system. The accuracy of this approach was 5-6 seconds which was very good for the scope of this study.

```
screen_y_adjusted=mapHeight-screen_y
point_long= Long_min+(Long_max-Long_min)*((screen_x-0)/(1920-0))
point_lat= Lat_min+(Lat_max-Lat_min)*((screen_y_adjusted-0)/(1200-0))
```

Since the map bounding box or the map extent coordinates were not a direct input i.e. the values were not directly exported by MT; this approach was not prioritized. In order to pursue this approach, two possibilities were considered; the first was to derive map extent from map center coordinates, the second was to modify the MT code to return map extent values. The former option was researched and tested, but it yielded very inaccurate map extent values using map center coordinates. The latter option was reserved as a backup option in case approach 3 were to fail. In addition, using only map center coordinates for calculations meant lesser reliability on web map-specific libraries, and from a scalability perspective was considered a better option, such as if the browserhistory module was to be pursued later.

4.4.3 Approach 3

The third approach was the one taken forward to implement in the tool for this diploma thesis. The inputs were the same as the first approach; zoom level, map center coordinates, and screen coordinates. These inputs were used in the Web Mercator projection formula shown in Fig 11.

Web Mercator is a variation of spherical projection, which is the de facto standard used by web mapping platforms such as Google Maps, OpenStreetMaps, and Mapbox. It is slightly different from the Mercator projection because it employs spherical formula at all scales, unlike the Mercator maps with the ellipsoidal version of projection at a large scale (Fig 11).

Spherical Forward--earth to map

- $x = R\ (\lambda - \lambda_{\circ})$
- $y = R\ \ln(\tan(\pi/4 + \Phi/2))$

Spherical Inverse--map to earth

- $\Phi = \pi/2 - 2\ \arctan(e^{-y/R})$
- $\lambda = x/R + \lambda_{\circ}$

Ellipsoid Forward--earth to map

- $x := a * (\lambda - \lambda o)$
- $y := 0.5 * a * \ln((1 + \sin(\phi)) / (1 - \sin(\phi)) * [(1 - e * \sin(\phi)) / (1 + e * \sin(\phi))]^{e})$

Ellipsoid Inverse--map to earth

- $\Phi = [\pi/2] - 2 * \text{ArcTan}([\exp(-y / a)] * [(1 - e * \sin(\phi)) / (1 + e * \sin(\phi))]^{0.5 * e})$
- $\lambda = \lambda o + x / a;$

Figure 11 Mercator Projection formulae (*Source: USNA, 2012*).

The Web Mercator variation adjusts the world coordinates before applying the zoom. The origin of the coordinate system is on the top left, same as that of a display screen, and hence take tiles and pixels into account (Fig 12). OpenStreetMap provides good comprehensive documentation online on this subject, such as the technique behind the tiling of slippy maps (2019).

$$x = \left\lfloor \frac{256}{2\pi} 2^{\text{zoom level}} (\lambda + \pi) \right\rfloor \text{ pixels}$$

$$y = \left\lfloor \frac{256}{2\pi} 2^{\text{zoom level}} \left(\pi - \ln\left[\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right]\right) \right\rfloor \text{ pixels}$$

Figure 12 Web Mercator projection formula *(Source:Wiki Web Mercator).*

The map center coordinates, latitude and longitude are first converted to projected coordinates x and y through a forward spherical Mercator projection but with the Web Mercator variation. The latitude (φ) and the longitude (λ) are converted to radians and multiplied to the constant terms. Since the zoom level is the same for both the latitude and longitude, it was grouped together with the constant terms and calculated separately in the code. $256/2\pi * 2^{\text{zoomlevel}}$ refers to the constant terms that are termed as unit distance in the code shown below. After calculating map center x and y through the coordinates and forward projection formula, the distances in x and y direction are calculated as *xDist* and *yDist* columns through the screen distances from the ET screen coordinates of the points. This yields the x and y locations of the points in the projected coordinate system. The final latitude and longitude of these points are computed using the inverse Mercator projection

formula as shown above. The implementation of this in code with the pandas df columns for fixations points data is shown below. In addition, the resulting header of this calculation is shown in Fig 13.

```
#Approach 3
et_fix_combined['unit_distance']=(256/(2*np.pi))*
                                  2**et_fix_combined['Zoom_level']

#forward projection

et_fix_combined['mapcenter_x']=et_fix_combined['unit_distance']*
                           (np.deg2rad(et_fix_combined['MapCenter_long'])
                            +np.pi)

et_fix_combined['mapcenter_y']=et_fix_combined['unit_distance']*
                            (np.pi-np.log(np.tan((np.pi/4)+
                       np.deg2rad(et_fix_combined['MapCenter_lat'])/2)))

et_fix_combined['xDist']=et_fix_combined['mapcenter_x']-(mapWidth/2-
                          et_fix_combined['Fix_x'])
et_fix_combined['yDist']=et_fix_combined['mapcenter_y']-(mapHeight/2-
                          et_fix_combined['Fix_y'])

#inverse projection formula for point to latlong
et_fix_combined['XPoint']=(et_fix_combined['xDist']/
                            et_fix_combined['unit_distance'])-np.pi

et_fix_combined['YPoint']=-(et_fix_combined['yDist']/
                            et_fix_combined['unit_distance']) + np.pi

et_fix_combined['Longitude']=np.rad2deg(et_fix_combined['XPoint'])
et_fix_combined['Latitude']=np.rad2deg((np.arctan(np.exp
                              (et_fix_combined['YPoint']))-(np.pi/4))*2)
```

	Zoom_level	MapCenter_lat	MapCenter_long	Fix_start_time	Duration	Fix_x	Fix_y	Sync_time	ind	unit_distance
1	10.0	49.750423	17.149689	13811.1	190.0	699.700012	489.600006	689.000000	1	4.172151e+04
2	10.0	49.750423	17.149689	14068.8	198.9	640.599976	388.700012	946.700012	2	4.172151e+04
3	10.0	49.750423	17.149689	14321.2	218.6	1208.300049	739.400024	1199.099976	3	4.172151e+04
4	10.0	49.750423	17.149689	14604.1	203.1	1079.199951	676.299988	1482.000000	4	4.172151e+04
5	10.0	49.750423	17.149689	14849.1	260.0	900.299988	524.599976	1727.000000	5	4.172151e+04
6	10.0	49.750423	17.149689	15144.7	218.5	845.299988	490.100006	2022.599976	6	4.172151e+04
7	10.0	49.750423	17.149689	15436.4	180.0	1514.300049	750.400024	2314.300049	7	4.172151e+04

	mapcenter_x	mapcenter_y	xDist	yDist	XPoint	YPoint	Longitude	Latitude
	1.435600e+05	8.918676e+04	1.434438e+05	8.925656e+04	0.296533	1.002251	16.990112	49.688450
	1.435600e+05	8.918676e+04	1.432997e+05	8.907636e+04	0.293080	1.006570	16.792221	49.848283
•••	1.435600e+05	8.918676e+04	1.432406e+05	8.897546e+04	0.291663	1.008989	16.711060	49.937549
	1.435600e+05	8.918676e+04	1.438083e+05	8.932616e+04	0.305270	1.000583	17.490677	49.626575
	1.435600e+05	8.918676e+04	1.436792e+05	8.926306e+04	0.302176	1.002095	17.313385	49.682675
	1.435600e+05	8.918676e+04	1.435003e+05	8.911136e+04	0.297888	1.005731	17.067703	49.817280
	1.435600e+05	8.918676e+04	1.434453e+05	8.907686e+04	0.296569	1.006558	16.992172	49.847841
	1.435600e+05	8.918676e+04	1.441143e+05	8.933716e+04	0.312604	1.000319	17.910904	49.616789

Figure 13 Final table-header

4.5 Data Exports

Once the script was functional for data conversion i.e., the points were converted from screen coordinates to geographic coordinates; the next step was file conversion and export. The main file formats considered for exports in the ET2Spatial tool were Geojson and Shapefile. The reason for selecting these file formats was their widespread usage and popularity amongst the GIS community and their compatibility with most GIS software such as QGIS and ArcGIS. Geojson can also be easily visualized with mapping libraries and APIs such as leaflet, Mapbox, and Google Maps.

The script generated a separate file for every participant with the label mirroring the participant id that was stored at the beginning of the script. The converted raw gaze points file and the converted fixations points file were given as an output for each participant. The shapefile function generated a CSV file as a byproduct which, if needed, could also be used in GIS software as an alternative.

When it comes to the structure of the files being exported, only the necessary columns were exported, namely; Latitude, Longitude, Zoom level, Time, and Id. Where Latitude and Longitude are in decimal degrees and Time is in millisecond stamps. The fixations file has an extra duration column, which also gives the amount of time the user fixated gaze on that point in milliseconds. The Id is the explicit id from the data frame assigned at the time of data pre-processing and creation of formatted time columns and is in the temporal order of the points. It can be used to re-order points in the shapefile easily.

4.5.1 GeoJSON

GeoJSON is an extension of JSON file format that is designed for the representation of spatial features along with their non-spatial attributes. GeoJSON is an open standard GIS file format that encodes geographical features as either points, lines, or polygons and can have additional attributes as properties. For this study, the python module geojson was used which has the ability to encode and decode geojson files as well as classes for individual GeoJSON objects.

```
#GeoJSON fixations file
featuresf = et_fix_combined.apply(
            lambda row: Feature(geometry=Point((float(row['Longitude']),
float(row['Latitude'])))),
            axis=1).tolist()

# all the other columns used as properties
propertiesf=et_fix_combined[['Zoom_level',
                             'Duration',
                              'Sync_time']].to_dict('records')

# whole geojson object
feature_collection=FeatureCollection(features=featuresf,
                   properties=propertiesf)

#taking user selection path
folderpath= os.path.normpath
            (QtWidgets.QFileDialog.getExistingDirectory
            (self, 'Select Folder for fixation points GeoJSON file'))

filename=p_num+r'_fixationpoints.geojson'
fi=os.path.join(folderpath, filename)

with open(fi, 'w', encoding='utf-8') as f:
            json.dump(feature_collection, f, ensure_ascii=False)
```

In the above code, the features are defined using the latitude and longitude columns computed from previous calculations. The features defined the geometry of the points and were stored in a list. The attributes selected for export were the zoom level, duration, and time. The respected columns were selected from the data frame and stored in the properties dictionary. To compile a GeoJSON object which is a feature collection, the inbuilt function from the module was used. Both the features list and the properties dictionary were sent as inputs to the function. The result was dumped to a user-defined folder path using another pre-defined module function. Fig 14 shows a sample GeoJSON output.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "geometry": {
        "coordinates": [
          16.990112,
          49.68845
        ],
        "type": "Point"
      },
      "properties": {
        "Duration": 294.5,
        "Time": 349.8999938964844,
        "Zoom": 10
      },
      "type": "Feature"
    },
```


Figure 14 Sample Output GeoJSON

4.5.2 Shapefile

The main output of the ET2Spatial tool was the ESRI Shapefile of the converted ET data points. A separate shapefile was generated for the fixations data frame and for the raw points data frame. Each of these shapefiles had attributes as described in section 4.5. Python module pyshp provides functionalities to read and write shapefiles. It generates dbf and shx file extensions in addition to the shp file format. Pyshp module is compatible with python version 2.7 and above. In the code, pyshp is imported as *shapefile*.

Since a part of the shapefile creation process was the import of the data points in CSV format, the CSV file was exported to a folder location instead of only keeping it in the temporary memory, which could potentially be used as an alternative in the visualization process. The header file was defined with the same columns as in the GeoJSON export, but an additional index column was added as well. The index column was created during the pre-processing stage, where the implicit index values were assigned as an explicit index to indicate a temporal order of values. Although CSV export usually outputs the implicit index of the pandas data frame as a separate unnamed column, the defined index column was separately exported nevertheless to keep the original indices associated with the ET point data.

```
#csv
header = ["ind","Zoom_level", "Format_RT", "Longitude", "Latitude"]
et_raw_combined.to_csv(f'/path/{p_num}_rawpoints.csv', columns = header)
```

In addition to the shapefile creation, a projection file was also generated, the purpose of which was to keep a record of the projection system in which the converted ET spatial

points were. A blog tutorial by Geospatiality (2015) was used as a guide to create a function for this purpose. It involved passing the EPSG code in the function arguments. The EPSG code was passed as a parameter in the URL of spatialreference.org, a site that has a comprehensive database of all geographic projections. The result was exported in Well-Known-Text format after formatting spaces and newline characters. A sample export is shown below.

```
def getPROJ (self,epsg_code):
     from urllib.request import urlopen
     with urlopen("http://spatialreference.org/
        ref/epsg/{0}/prettywkt/".format(epsg_code)) as wkt:
            format_spaces = wkt.read().decode('utf-8').replace(" ","")
            result = format_spaces.replace("\n", "")
     return result
```

Export:

```
PROJCS["NAD83(NSRS2007)/MichiganCentral",GEOGCS["NAD83(NSRS2007)",DATUM[
"NAD83_National_Spatial_Reference_System_2007",SPHEROID["GRS1980",637813
7,298.257222101,AUTHORITY["EPSG","7019"]],TOWGS84[0,0,0,0,0,0,0],AUTHORI
TY["EPSG","6759"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["
degree",0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","
4759"]],UNIT["metre",1,AUTHORITY["EPSG","9001"]],PROJECTION["Lambert_Con
formal_Conic_2SP"],PARAMETER["standard_parallel_1",45.7],PARAMETER["stan
dard_parallel_2",44.18333333333333],PARAMETER["latitude_of_origin",43.31
666666666667],PARAMETER["central_meridian",-
84.36666666666666],PARAMETER["false_easting",6000000],PARAMETER["false_n
orthing",0],AUTHORITY["EPSG","3587"],AXIS["X",EAST],AXIS["Y",NORTH]]
```

After the CSV export, a shapefile writer class object was initiated. To export attributes of the points in the shapefile, new fields were created through the inbuilt shapefile class function and with a data type declared for each attribute. N refers to double integer, the length of which is limited to 18 characters. F refers to floating point numbers and with the same length restriction as an integer. It was important that the newly created fields did not have any attribute that was not present in the CSV file header because the fields would be populated by iterating through rows of the CSV file for each of the mentioned columns. In the *for* loop, the values for each row were separately stored in the corresponding variables. Out of these, the latitude and longitude variables were used to create the geometry of the points, and the rest were added as attributes or *records*. The counter was used to enumerate the total number of features and which was also a part of the output written to the shapefile. Finally, a projection file was written along with the shapefile, using the *getPROJ* function described above. The output and visualizations of these functions are discussed in section 5.3.

```
# create a point shapefile
import shapefile as shp
```

```
raw_shp=shp.Writer('/path/')s
#ensure geometry and attributes match
raw_shp.autoBalance = 1

# create the field names and data type for each.
raw_shp.field("ind", "N")
raw_shp.field("Zoom_level", "N")
raw_shp.field("Format_RT", "F")
#count the features
counter = 1
# access the CSV file
with open('/path/', 'rt') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    # skip the header
    next(reader, None)
    #iterate through every row and assign attributes to variables
    for row in reader:
        ind=row[1]
        zoom_level = row[2]
        time = row[3]
        longitude = row[4]
        latitude = row[5]
        # create the point geometry
        raw_shp.point(float(longitude),float(latitude))
        # add attribute data
        raw_shp.record(ind,zoom_level, time)
        #print "Feature " + str(counter) + " added to Shapefile."
        counter = counter + 1
raw_shp.close()

# create a projection file
prj = open(f"/path/{p_num}raw", "w")
epsg = getPROJ("3587")
prj.write(epsg)
prj.close()
```

4.6 Graphical User Interface

After creating and testing the functionality of all the relevant functions in the python script using Jupyter Notebook, the next step was to create a user interface for the script. The python script, which was in ipython notebook format was first migrated to a compiler environment. The script was imported as a python file in SPYDER IDE.

4.6.1 Designing Layout

The conceptual layout of the user interface was kept very simple with a button for each; main imports, function, and exports. The size of the window was kept small because there were no visual aspects to the processing. In addition, basic level function completion notifications were aimed for during the design process. QT Designer simplified the execution of the user interface of ET2Spatial from conceptual layout to a concrete code. QT Designer is a tool that helps build quick graphical user interfaces by drag and drop mechanism. It allows users to design the layout with slots, buttons, and widgets. QT Designer is based on the QT GUI framework written in C++. The output from the QT Designer can be easily integrated into any code.

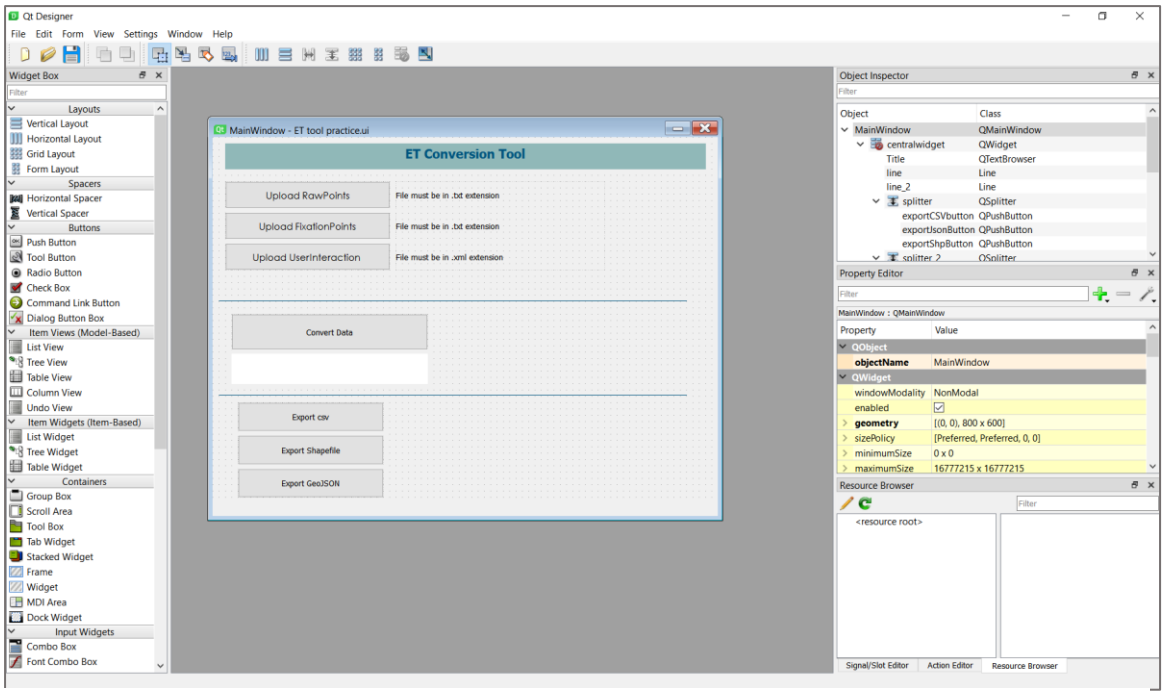


Figure 15 Prototype GUI in QT Designer

The main components of the layout of ET2Spatial were organized through QT Designer. In addition to the buttons and title of the window, a progress bar was added. The font size, type, and color were changed in the software. Files created in QT Designer are usually saved in .ui format. The user interaction layout code created in this manner was, however, a skeleton structure without any functionality associated with the buttons. Figure 15 shows the initial layout of ET2Spatial that was composed using QT Designer software.

The rest of the tweaks in layout and the stitching of buttons to functions were done in SPYDER IDE using PyQt5. As described in section 2.3, PyQt5 is a set of comprehensive python libraries, originally written in C++, that enable interactive functionalities such as forms and graphical user interfaces. Riverbank Computing, which is the developer and distributor of PyQt, has extensive documentation on PyQt5 and which was used frequently

during the next steps of this thesis study. To make the ui file accessible and for editing features through programming, the file format was converted from .ui to .py.

```
Pyuic5 -x ETtool.ui -o ETtool.py
```

The pyuic5 command, along with the input ui file and desired output .py file, was specified in the command line. The python file that was generated as a result contained all the relevant libraries needed for integration into the main code.

A logo was also created as a part of the layout designing procedure of GUI; although not necessary for this thesis study, it was done to give the tool more user-friendliness and quick impression. The idea behind the logo was to keep it very illustrative and simple. Since the two main aspects of the tool were eye-tracking and spatial locations, an eye shape with a map marker in the center was used to convey the main concept. Figure 16 shows the final logo used in the tool; this logo was meant to be utilized as an icon for the layout window as well.

Figure 16 ET2Spatial icon.

4.6.2 Connecting functions

The python script tested on sample data so far was structured procedurally with line-by-line execution of the program. This code, however, needed to be restructured into functions in a modular manner to be able to connect to buttons and called with single commands. Three import functions were set up for the raw data points, fixation data points, and user interaction data. The general logic behind the raw data points and the fixation data points was the same.

In the code shown below, the *setupUi(self,MainWindow)* function enables the initialization as well as formatting of different widgets and buttons, which can also call functions from other parts of the code as inputs. One example of the raw data points, namely *RawFiles(self)* function, is depicted. This function is connected to the **Upload RawPoints** button through the `self.RawButton.clicked.connect(self.RawFiles)` command. Within the *RawFiles(self)* function, a global variable is first declared, then a user file input function is called through the PyQt library which outputs a path in return from the user's selection. This path is formatted and fed to the read CSV function of pandas, which reads the input file and stores it into a data frame in the variable declared before. Upon completing this function, a notification is given to the user on the user interface by

replacing the default text label next to the button with the name of the uploaded file in bold green text (see section 6).

```
   class Ui_MainWindow(QDialog):
       def setupUi(self, MainWindow):
           self.RawButton.setFont(font)
           self.RawButton.setObjectName("RawButton")
           self.retranslateUi(MainWindow)
           self.RawButton.clicked.connect(self.RawFiles)

       def RawFiles(self):
           #IMPORT RAW POINTS FILE AND STORE IN DF
           global et_raw
           global path_r
           filetype = 'Text file(*.txt)'
           rname=QFileDialog.getOpenFileName(self,'Open
                   File',filter=filetype)
           path_r=os.path.normpath(rname[0])
           et_raw = pd.read_csv(path_r,delimiter="\t")
           if (path_r is not None):
               self.labelR.setText(os.path.basename(path_r))
               self.labelR.setStyleSheet("color:green;font-weight: bold;
                                          font-size: 10pt")
```

The *Ui_MainWindow* class (shown above) is initialized in the *main* function.

```
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Similarly, the button for **Upload FixationPoints** in the GUI was connected to the FixFiles(self) function in the code. Figures 17 and 18 outlines the function behind each component of the ET2Spatial GUI.

Finally, one button called **Reset** was added to provide convenience to the user. The tool does not have a capability for multiple imports and conversion simultaneously. In many cases, including this thesis study, there is more than one participant for which the data needs to be converted. Restarting the application again and again would be very ineffective and although the user could upload data for the new participant and start over, the

configuration of the user interface could be confusing. To simplify this problem, *ResetProgress(self)* function was defined in the *MainWindow* class. The only purpose was to delete the existing main variables from memory and reset the labels next to the upload buttons. This would signify a restart of the tool and the progress.

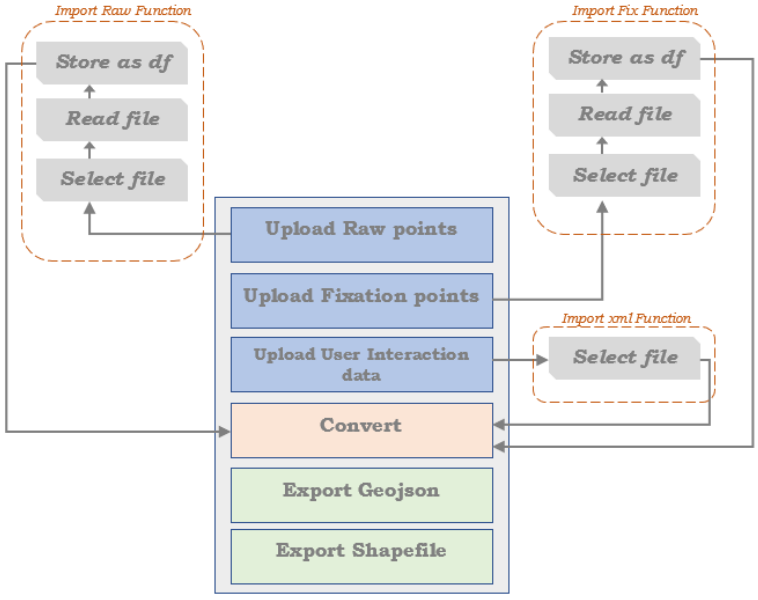


Figure 17 GUI widget connections imports.

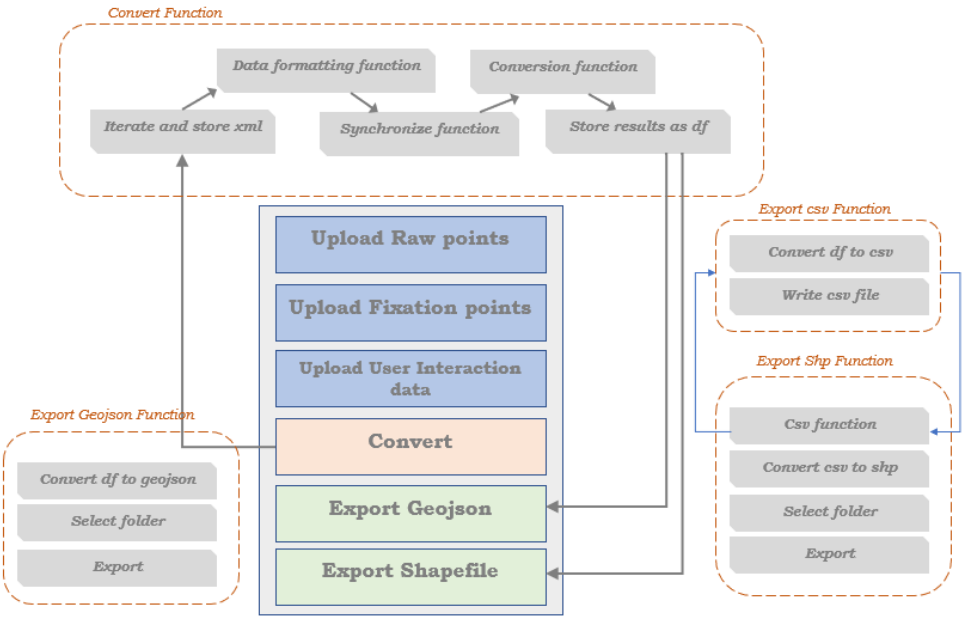


Figure 18 GUI widget connections convert & exports.

The code was also optimized for error handling during the imports. If the user forgets to specify one of the three files, a warning window was configured to show up when clicking the **Convert** button.

Since one of the objectives of this study was to create a standalone desktop application, an exe file was generated from the python file. This was done using PyInstaller module. Cx_Freeze and Pytoexe modules have a similar purpose and were tested as well, but they resulted in unresolvable dependencies with the python environment on the local machine. PyInstaller gives the option to create a folder with all the necessary modules, libraries, and dependencies along with the exe or create a single standalone executable file packaged with all the dependencies implicitly. The latter option was pursued. The code below was executed through a command prompt. The logo designed previously was also specified to be used as an icon for the tool. `-F` refers to one file output in contrast to multiple files.

```
pyinstaller -w -F -i "D:\icon256.ico" ET.py
```

One con of generating a single exe file was the comparatively larger size of the executable file and the time needed for its initialization. To give notification on the initialization progress, a command line interface was added. This command line window pops up as soon as the executable file is clicked. It generates notifications on the compilation of the tool and after completion, opens the GUI window of ET2Spatial. Figure 19 shows the final interface window of ET2Spatial.

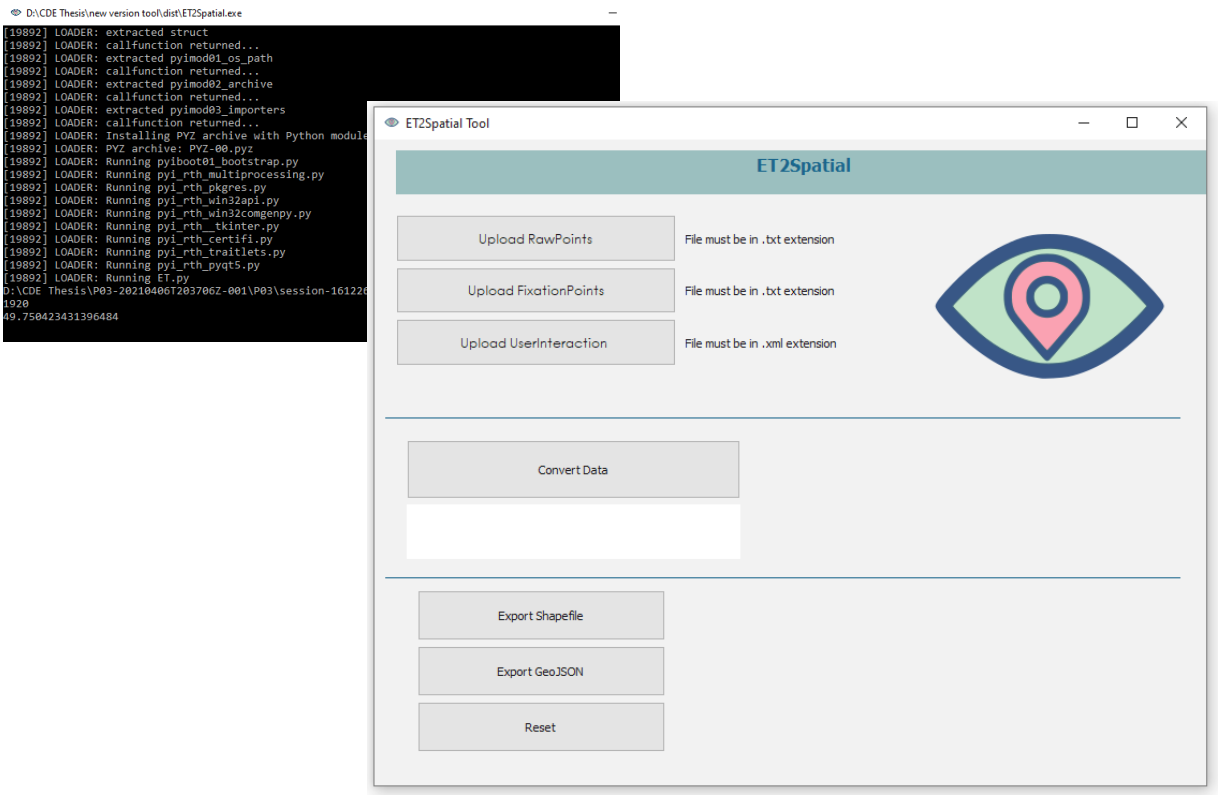


Figure 19 ET2Spatial GUI and initialization

5 TOOL EVALUATION

Quality control is a part of the lifecycle in product development of any kind, particularly in software development software quality concept implies that the software functions according to expectations. For the evaluation of ET2Spatial tool, a similar concept was adopted. The evaluation benchmarks were categorized into output accuracy, tool performance, and use case scenarios. The evaluation was an important part of the thesis study as it gave insights into how effective and useful the developed tool was. It also allowed analyzing the strengths and weaknesses of the thesis study in detail.

5.1 Output Accuracy

Output accuracy refers to the positional accuracy of the exported spatial data from the tool i.e., whether the geographical coordinates of the eye-tracking points were at the exact locations they were supposed to be. To determine the accuracy of the exports, three mechanisms were established which held more credibility when used in combination than individually.

The first evaluation mechanism was based on the concept of reverse conversion whereby the known geographic coordinates of a point were fed into an algorithm to output screen coordinates. This was done through the transformation formula mentioned in section 4.4.2. The transformation formula converts a point from one coordinate system to another using bounding box coordinates. Since the approach was not utilized to calculate coordinates in the tool itself, it was used for evaluation. The approach, however, did not have to be programmed into code for usage because one such application[1] was already developed as a part of one of the semester projects in the Masters study. The said application was written in Java. It took geographic coordinates from a database of user tweets and converted the points to screen coordinates to display them in a desktop window with a basemap. This application was tweaked to take user input for geographic coordinates instead of fetching from an existing database. The main inputs of the application were x minimum and x maximum as longitudes, y minimum and y maximum as latitudes, and the point latitude and longitude. Additionally, the output coordinate system bounding coordinates were also specified i.e., the screen width and height. To find the bounding box of the point on full-screen mode and at a fixed zoom level, an online tool bboxfinder[2] was used. The point and its bounding box coordinates and the screen size (the same output from MapTrack) were entered into the application. As a result, the screen coordinates of the point were obtained.

The resultant screen coordinate was then used as a control as it had known geographic coordinates. The same screen coordinates for the said point were passed through the conversion framework of this study or through ET2Spatial to check how much the output

[1] Twitter time series Visualizer, a semester project developed in alliance with three group members. Done as a requirement for Software Development course during third semester.
[2] http://bboxfinder.com/

deviated from the actual geographic coordinates. Figure 20 depicts the graphic explanation of the evaluation mechanism carried out for this study. One thing to note is that better accuracy was observed for map exploration at a higher zoom level than at a small scale. As mentioned in section 4.4.3, the web Mercator projection formula is used for common web maps for transformation such as Google Maps, Bing Maps, Open Street Maps etc. Hence, the output of the tool correlates well in terms of accuracy when overlaid over the web maps using the same projection.

In addition to the structured methodology for accuracy evaluation, a rough interpretation of the output was also made through visual analysis. The geographic coordinates of the points, as exported by the tool, were overlaid on a basemap. These points were visually compared to the point positions on a single frame in the video recording exported through the eye-tracking software.

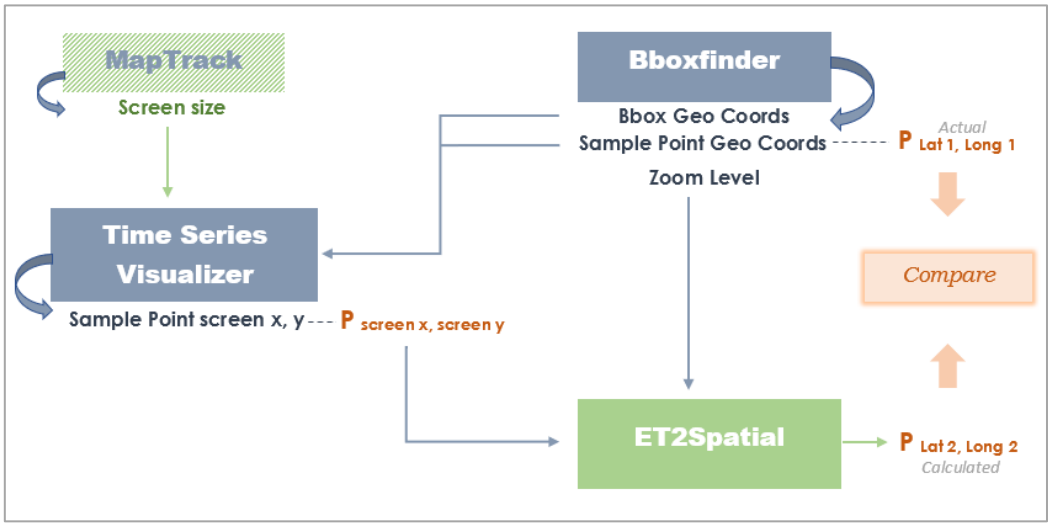


Figure 20 Accuracy evaluation mechanism.

5.2 Tool performance

The second part of the tool assessment was the functional quality i.e., how smooth the tool functions when executed on different systems. While packaging the tool in a single executable file, all the dependencies were included. These packaged python dependencies would allow users to run the tool without installing any modules or language by themselves and would also prevent any system environment clashes. To test this however, the tool was run on four different laptops with a Windows operating system. No noticeable problem was observed during these trials.

One small weakness of the tool was its compilation or initialization time. The common initialization time was 40-50 seconds. However, as mentioned earlier, a command-line interface was added to portray the initialization process before the final graphical user interface shows up. This was added to lower the inconvenience during usage.

While testing the tool on different systems, user feedback was also administered, and improvements were made in the graphical user interface, such as displaying notifications on file uploads, the transparency of the window, and so on.

Finally, the tool was tested for the performance of its imports, conversion, and export functions. Thirty-four datasets were converted through the tool, out of which only two failed to give correct outputs. The reason for these two failed attempts was the non-compliance with the pre-requisites of raw data files such as the F2 key inputs. The tool has a very basic level error handling. But to manage unexpected scenarios, a README file was prepared to give the users guidance and information on which columns are necessary during export from the eye-tracking software and how to use the tool.

5.3 Use-case demonstration

The final evaluation criteria for the diploma thesis was the proof of concept. Several use case scenarios were established to show the possible capabilities of the exports from the tool.

One of the main driving forces behind this thesis study was the inability of the traditional ET software to overlay multiple participant data for interactive media on one file. Figure 21 shows the main problem the ET2Spatial tool addresses. In addition to the main visualization capabilities, this section enlists and explores multiple scenarios where the exported ET coordinates can be treated in a spatial context and subjected to dedicated GIS functions in the software. The results will not be judged based on the accuracy of the tasks but the merits and demerits of using GIS software and spatial function to analyze and visualize the ET data. The goal of these pilot studies is to provide and test possible application areas of ET2Spatial.

Concerning the visualizations, precision is only possible if the correct basemaps are loaded that were used at the time of experimentation. Particularly for the usability analyses, the tool can only be meaningful if the exact cartographic renderer is accessed through the GIS software. In QGIS, this process was very straightforward and was done using XYZ tiles functionality whereby the desired basemap was fetched through web url and added as a layer to the project. The process for ArcGIS was not a one-step procedure as it required the use of ArcGIS Online to add the basemap as a web layer and then accessing this content through the online portal in ArcGIS Pro. For map tasks demonstrated in the upcoming sections, mainly Google Maps Roadmap (Czech place labels) and Satellite map were used. The ET spatial data was imported into the GIS software and projected in the WGS 84 Geographic Coordinate System.

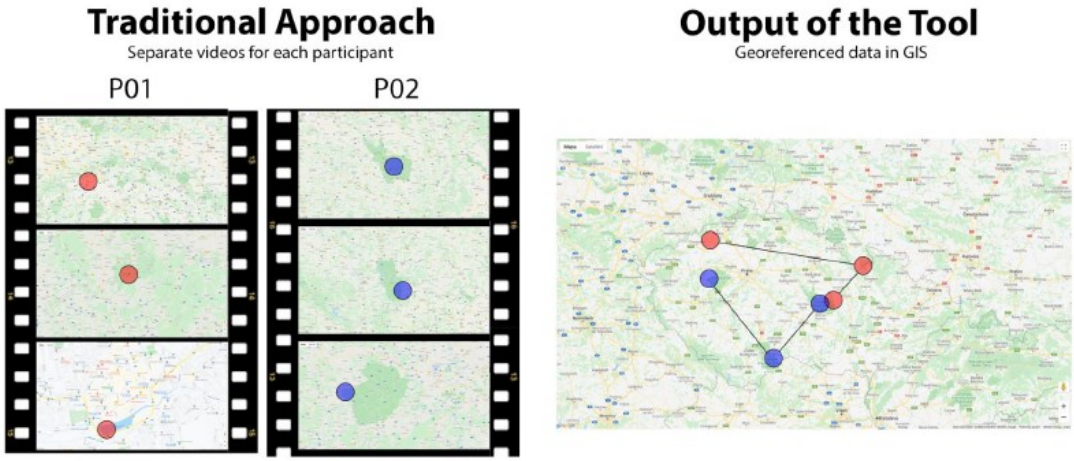


Figure 21 Main use-case scenario.

5.3.1 Experiments Setup

The study environment for collecting the data consisted of an eye-tracker i.e., SMI RED 250 for recording gaze points, a web application MapTrack for the web map interactivity data, SMI Experiment Center for setting up the map tasks and instructions, and SMI BeGaze for exporting the ET data. For post-processing of the results, ArcGIS Pro and QGIS were used. There were eight participants who undertook the experiment and were asked to solve several map tasks (Section 5.3.3-5.3.6). The participants were aged 25-35 and consisted of local as well as foreign students. All the participants included in the study had previous experience with web map usage.

5.3.2 Visualizations

The visualization aspect of the use-cases focuses on the different techniques that can be applied for displaying the eye-tracking data through manipulation of symbology, labeling, and custom rulesets.

Multiple participant data overlay

The first visualization tested for use-cases was the actualization of the main problem scenario. Figure 22 shows the eye-tracking fixation points of multiple participants displayed on the same basemap in ArcGIS.

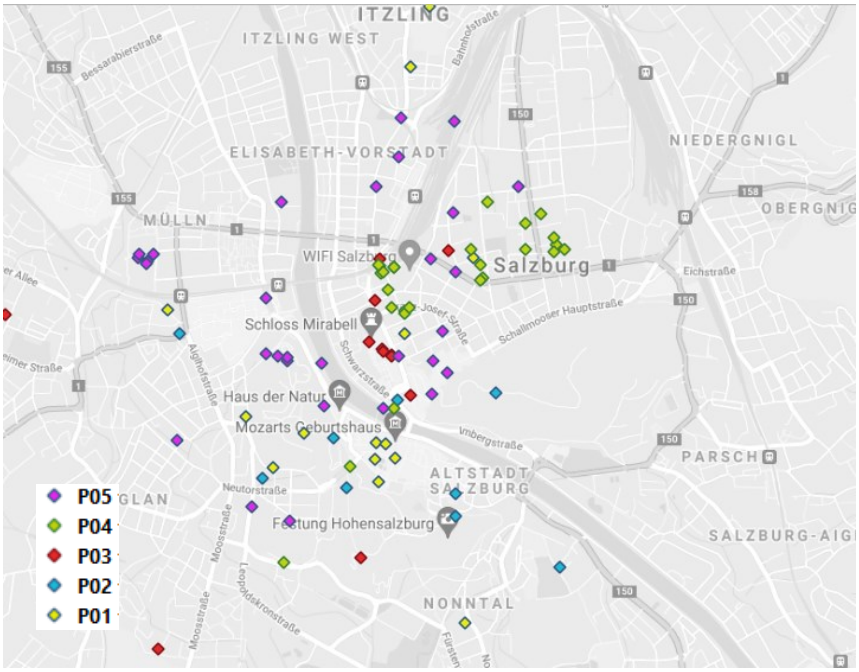


Figure 22 Multiple Participant Visualization.

Unlike the traditional eye-tracking software, the symbology for each of the participants can be manipulated as desired. The basemap itself has a few optimization options, such as color toning and transparency. In this case, the basemap was changed to grayscale to better distinguish the point symbology from the map features. A convenient way to have a structured symbology for all the points is by merging the point datasets for all the participants into one layer.

Scale-based rendering

The eye-tracking points were recorded with different variables during the user's interaction with the map i.e., every point was recorded at a certain zoom level and at a certain timestamp. Different zoom levels imply that the underlying content of the map can be different for each value. An accurate position of the ET point is not meaningful unless displayed at the correct scale with the original basemap to analyse the content user was looking at during the experiment. As done with the multiple participant visualization, it is possible to symbolize the ET data based on zoom level and to generate a category for each with different colors and shapes. Labeling can also help in this regard. Figure 23 shows the fixation points of a single user labeled by zoom levels and categorized as well in terms of colors.

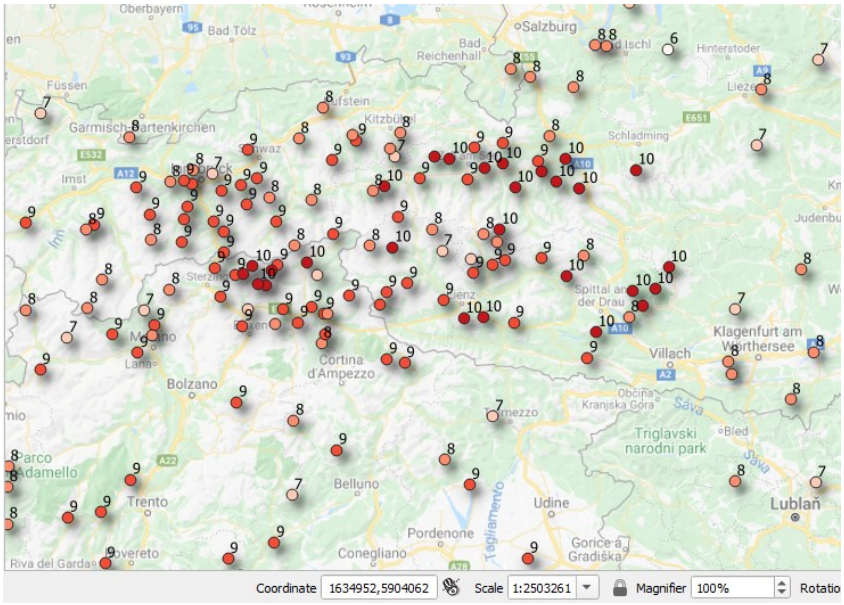


Figure 23 Categorization by zoom level.

This approach, although aesthetic, can be quite cumbersome while doing analysis or visual inspection. The reason being that the analyst would need to manually zoom to the right scale for every point to see what content lies on the cartographic renderer, and secondly, most GIS software such as ArcGIS or QGIS does not have a 'zoom level' displayed in the windows rather a map scale. This map scale is usually formalized in easy terms as a zoom level for online web maps and can be used in scale-based rendering. Scale-based rendering means that the symbology or labels on the map appear at different scales. This option is usually a part of most sophisticated GIS software.

Figure 24 shows the example of scale-based rendering which was set up as a part of the visualization use-case series for this study. In QGIS, the points were symbolized using the rule-based symbology, which categorizes points into different groups, in this case, the zoom levels, and then a minimum and maximum scale for each category can be defined. Resulting points in one category, such as zoom level 8 appear only when the user scrolls to a scale of 1:2311162 and beyond. The information on the scale value range associated with every zoom level was fetched through documentation online.

Scale-based rendering is a very convenient feature of the GIS software and is even more apt for this study, where the points are dependent on different zoom levels. It rids the analyst of labor-intensive manual scrolling for each point and for each zoom level during analysis and automates the process by only displaying relevant points at every scale.

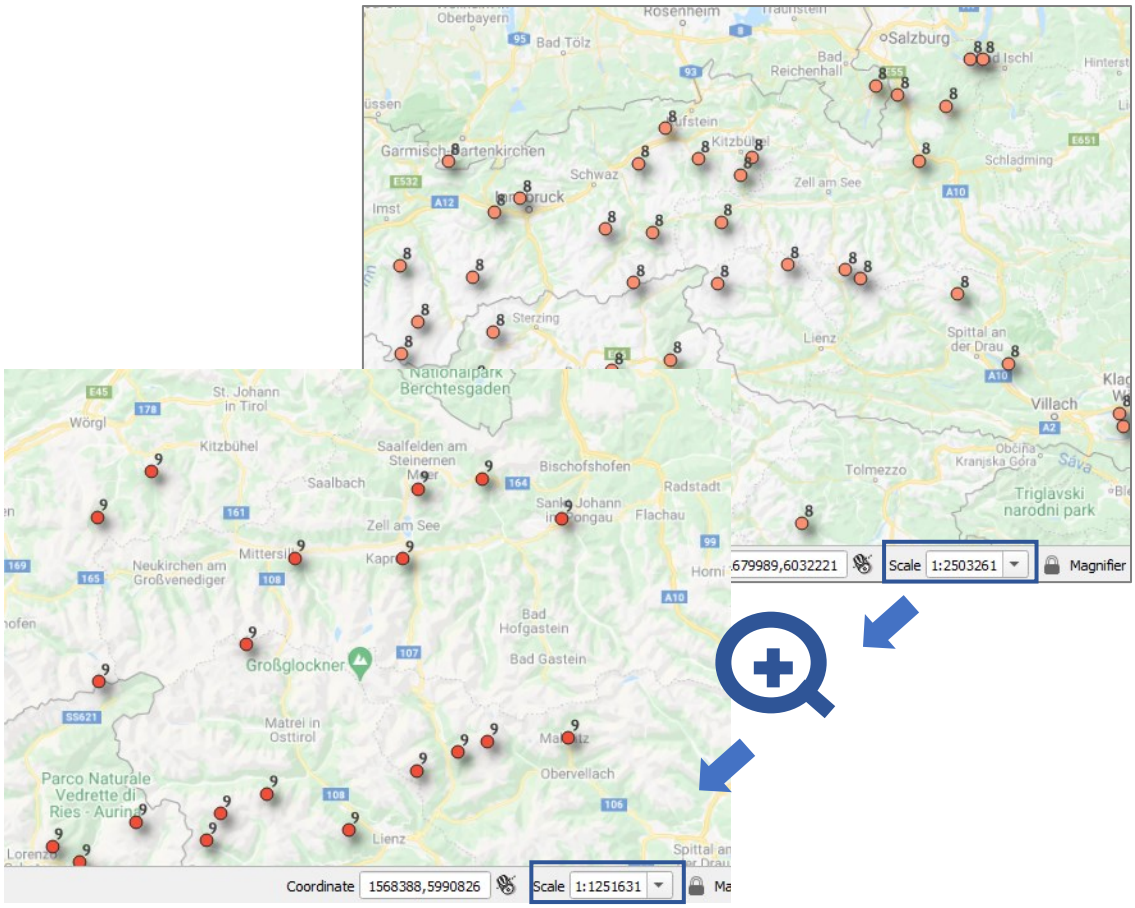
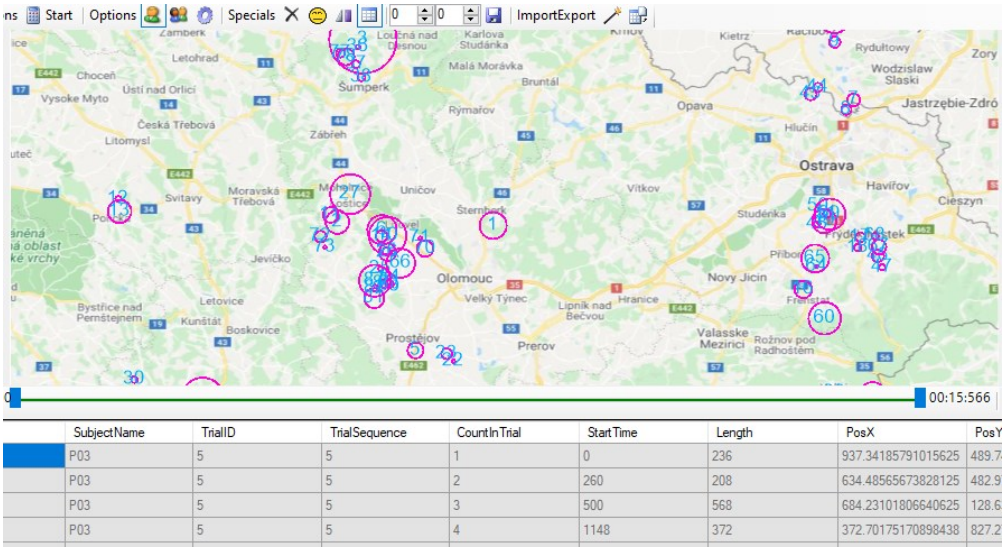


Figure 24 Scale-based rendering of ET points.

Attribute based visualization

The symbols can be varied based on their attributes. Similar to the examples shown before where the point symbols were categorized as unique symbols with varying colors, the graduated or proportional symbology for point datasets gives unique visualization capabilities. For the eye-tracking fixation points, the fixation duration can be used as a parameter for the graduated symbology or proportional symbology varying by size. This technique of visualization is somewhat similar to the standard ET software. Figure 25 shows example fixation points displayed over a static map image in the software Ogama.

Ogama is open-source software that allows analysis of eye-tracking data. It allows visualization of ET data in the form of circles representing fixations, which vary in size by their duration, scan paths, attention maps, and spotlights. The colors for the circle symbology can be customized. The labels, however only represent the sequence of fixations.



	SubjectName	TrialID	TrialSequence	CountInTrial	StartTime	Length	PosX	PosY
	P03	5	5	1	0	236	937.34185791015625	489.7
	P03	5	5	2	260	208	634.48565673828125	482.9
	P03	5	5	3	500	568	684.23101806640625	128.6
	P03	5	5	4	1148	372	372.70175170898438	827.2

Figure 25 Example of fixations on the map image in Ogama.

In comparison, the GIS software have much more options regarding symbology and labeling. Figure 26 Fixation points with graduated symbologyshows the fixations points displayed in ArcGIS Pro as graduated symbols with time duration as labels and a variable. Unlike the ET software where the symbol for the fixations can only be circle, GIS software allow customization of symbols in terms of shapes, colors and combination of both. The maximum and minimum symbol range can also be adjusted to fit the purpose of analysis.

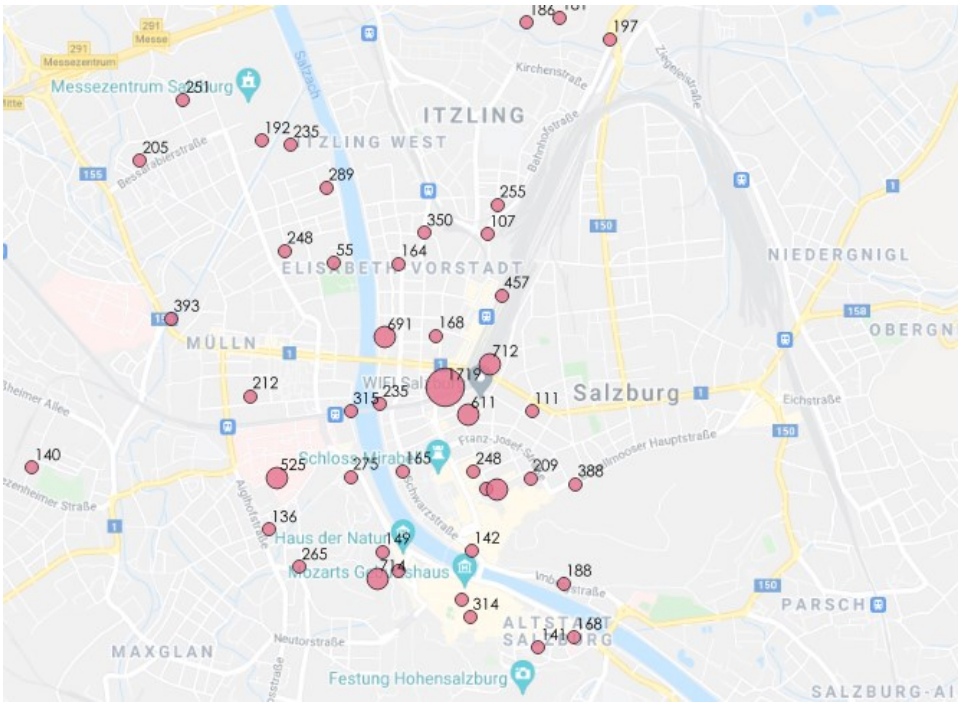


Figure 26 Fixation points with graduated symbology.

Custom scanpaths

Scanpath, in eye-tracking, is the sequence of locations in user's gaze i.e., the areas the user viewed on screen one after another. Scanpaths are one of the basic visual analytics mechanisms in eye-tracking and almost every ET software has features for consructing and displaying them. Figure 27 shows the default scanpaths built by Ogama software for three participants. Like many other ET software there are options to customize the colors and sizes of the scanpaths, although ET software such as SMI BeGaze offer more sophisticated options for customizations. These scanpaths however are over non-interactive media and are non-interactive themselves. In addition, as mentioned before, the labeling capabilities are very constrained.

These scanpaths can be recreated in the GIS software with multiple possibilities. Figure 28 shows the scanpaths created in ArcGIS Pro. Using point to line tool, a new feature dataset was created, this dataset was displayed in addition to the fixation point data layer. The order field takes input for the order of points and the index column was specified here. The symbology for both the point and line dataset can be manipulated separately.

The index column, as exported from ET2Spatial tool, allows sequential temporal ordering of points. The index label is hence used to specify the gaze sequence of the participants as scanpaths. The additional zoom level labels can help in understanding the trend of using zoom levels by the users while doing map tasks. In the referenced Figure 28, the red labels represent the zoom level associated with each fixation point. The interactivity of scroll and pan in the GIS software also allows the labels to be explored more clearly when zoomed in to a smaller area which otherwise might appear clustered in static images in frame-by-frame analysis through ET software. One thing to note is that in Figure 28, the scanpaths have been generated for the whole experiment and for all users combined, which is not a straightforward possibility in ET software such as Ogama.

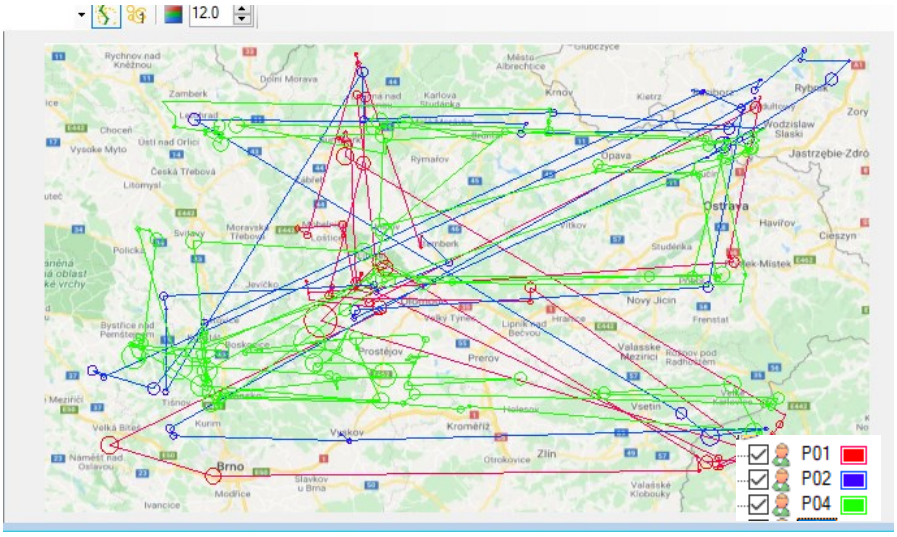


Figure 27 Example scanpaths Ogama.

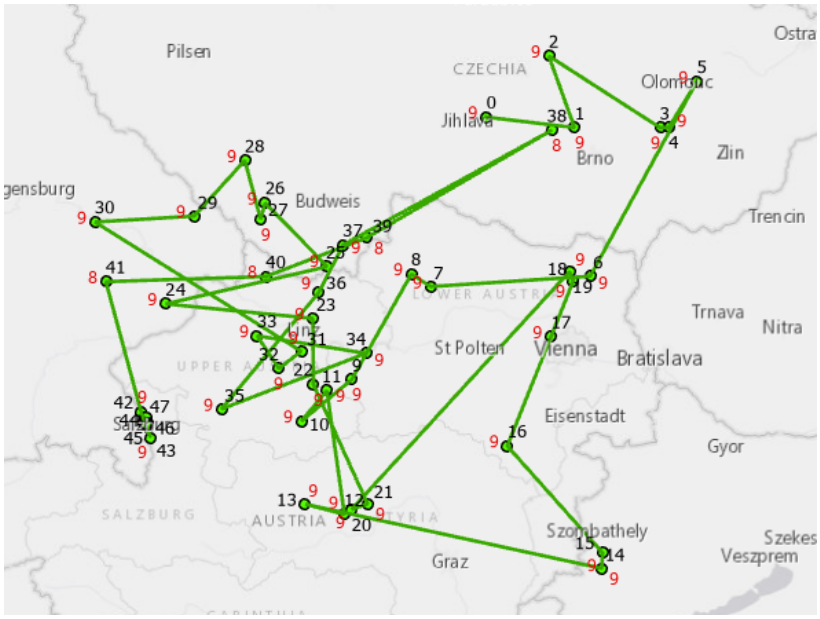


Figure 28 Scanpaths in ArcGIS Pro with zoom level labels in red.

Many interesting avenues are opened to explore users' cognitive behavior with interactive maps when the scanpaths are aggregated not just by their temporal sequence but with participant and zoom level individually. To take the multiple labelling ability one step further. In Figure 29 the left image shows the scanpaths categorized by zoom levels. These lines were aggregated on the zoom column and depict the temporal sequence of fixations at every zoom level. So, for instance, the yellow line shows the sequence of fixations by all users at zoom level 15. A convenient addition to this would be the scale-based rendering for every category. In the right figure, the lines were aggregated for every participant; this scenario is the closest to the one shown for ET software in Figure 27. For the sequence labels in these cases, the index would need to be reset for each category, something that is done automatically in the ET software for every participant.

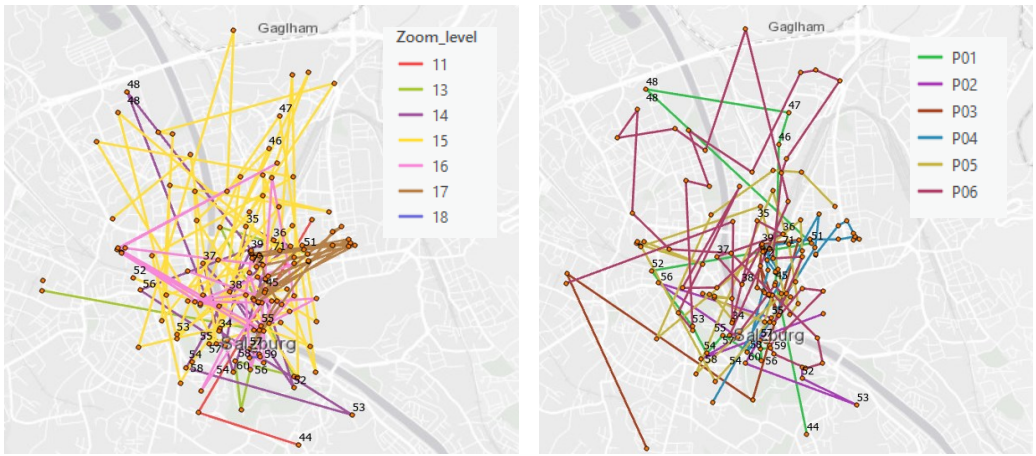


Figure 29 Scanpaths aggregated by zoom level vs participant.

5.3.3 Task 01

The visualizations demonstrated the capabilities of GIS software using the basic symbology manipulation options. The next few tasks exhibit the usage of ET data in a spatial context and how different GIS functions may or may not improve the analysis capabilities.

Task details: The first map task was to find the city of Salzburg, particularly to identify the city center. The initial web map window was set to the bounding region of Olomouc, where the Department is located. There were eight participants in total, each of whom was asked to find their way to Salzburg through interaction with the web map.

Analysis: The focus for analysis on the first task was to be able to answer simple questions such as which participants were able to fulfil the task, what was the average duration of fixations within Salzburg, what were the most common zoom levels used during the analysis etc.

To start off, the raw data from SMI BeGaze was exported and then converted through ET2Spatial. This converted spatial data was imported into ArcGIS Pro for every participant and projected into WGS 84 using the define projection tool. All the participant data points were merged into one file, and the source information was retained in the new merged file to have a separate participant column for further analysis. For this analysis, only fixation points were used because they give more insight into the user's focus rather than raw ET gaze points. Figure 30 shows the data for all participants after import, project, and merge functions.

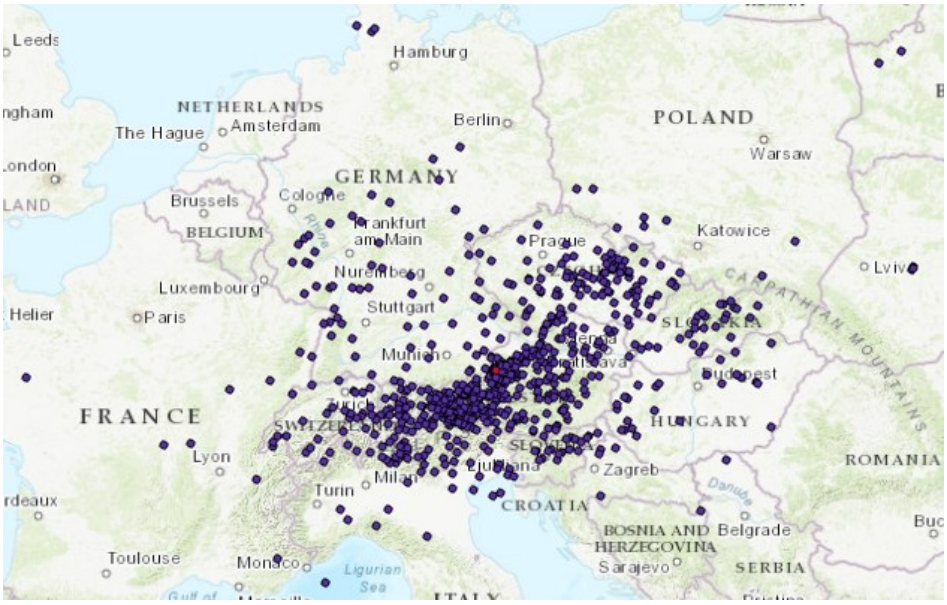
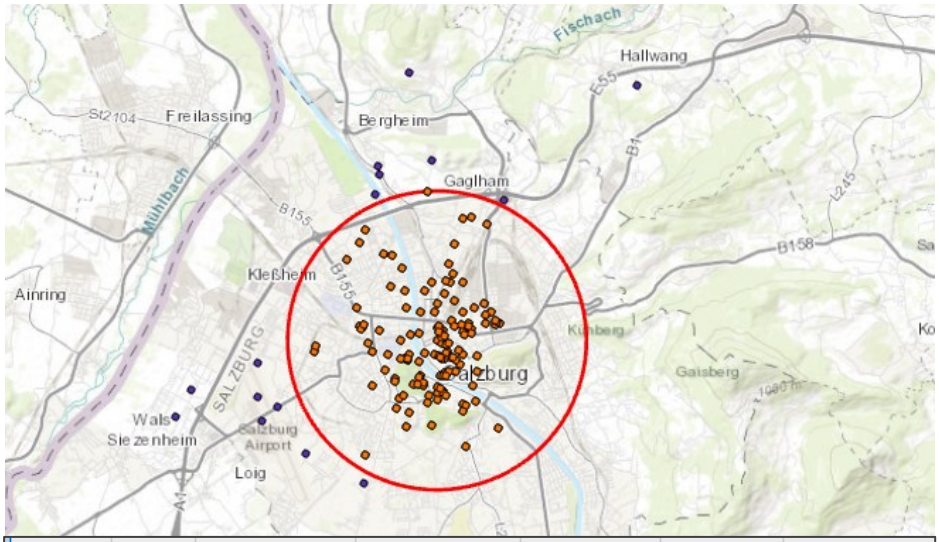


Figure 30 Merged fixation points in ArcGIS Pro for task 01.

The red point shows the location of the center of Salzburg, a feature that was manually created to give a visual reference to the rest of the points. The usage of supplementary

spatial data is elaborated more in Task 02. The Salzburg point was used to create a buffer of ~4 km from the city center using the Modify Features toolset.



OBJECTID	COUNT	MIN_Zoom_level	MAX_Zoom_level	MEAN_FirstFix	MIN_FirstFix	MEAN_Duration
1	138	10	18	15038.202899	9034	312.471014

Figure 31 Fixation points within buffer zone.

Spatial overlay operations such as union, intersection, difference etc. can be used to analyze the ET features further. Figure 31 shows the resulting fixation points after intersect tool was applied on the points with city buffer. Basic statistics were generated in the attribute table of the buffer layer as well as showing the minimum and maximum zoom levels used by participants when in the vicinity of Salzburg, which turned out to be between the range of 10 and 18. The mean of the first fixation represents the average time it took for all participants to fixate on location in the vicinity of Salzburg for the first time i.e., the average time for the first fixation within the buffer. This meant that, on average, it took 15s for participants to locate Salzburg on the map. The first individual fixation was at 9034ms.

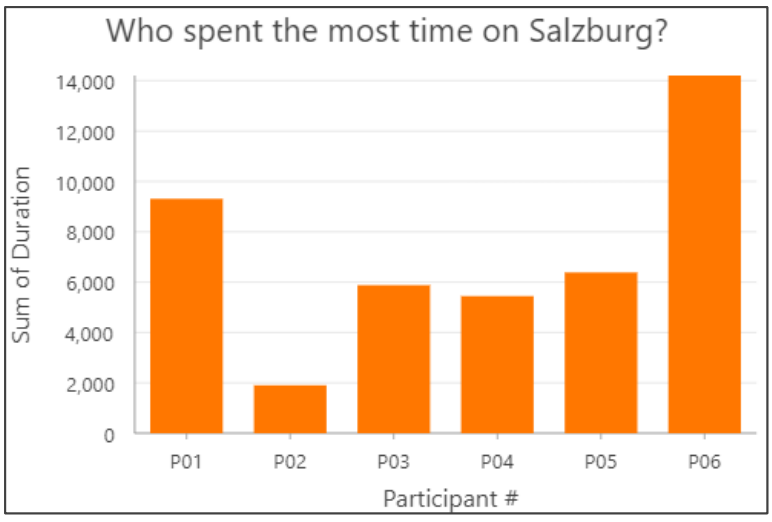


Figure 32 Chart 1: Duration vs Participants.

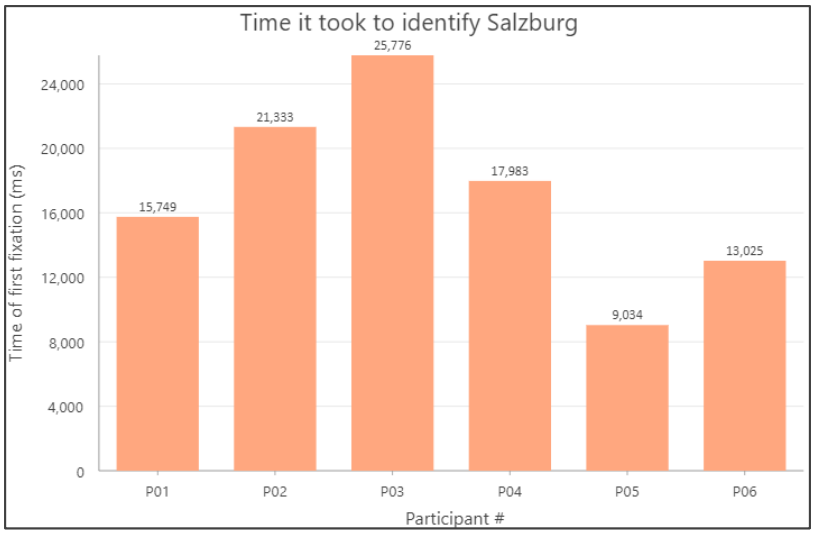


Figure 33 Chart 2: First fixation vs. Participants.

ArcGIS offers some basic exploratory data analytics through charts, plots, and histograms. This allows the investigation of non-spatial attribute data without having to migrate to different software. Figure 32 and Figure 33 show answers to interesting questions by querying the attributes of spatial data falling within the Salzburg buffer. One thing to note is that although the ET experiment for this task had eight participants, only six participants had fixations that fell within the AOI namely, the Salzburg buffer zone.

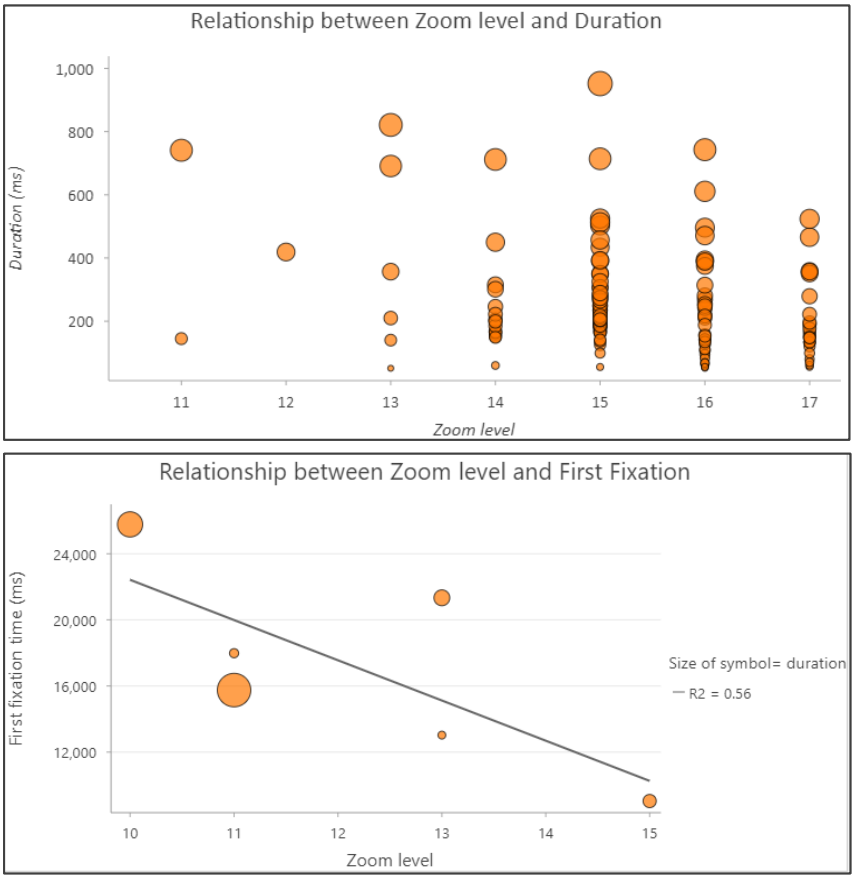


Figure 34 Chart 3 (top): Duration vs Zoom, Chart 4 (bottom): Fixation vs Zoom.

The charts in Figure 34 depict more relations between the users and map usage during this task. For instance, zoom level 15 had the most fixation duration time, from which a conclusion can be made that it was used the most for identifying and exploring Salzburg. This zoom level also had the first fixation at 9034ms, a figure mentioned in the first summary table of this task. The relationship between time for the first fixation for each participant and the corresponding zoom level was negative, as unraveled by chart 4. The earliest first fixation happened at a higher zoom level than the first fixation at the highest timestamp. Chart 3 tells that this earliest first fixation was for P05, whereas the latest one was for P03. Another intriguing observation was that even though the label for Salzburg appears at minimum zoom level 6 on Google Maps, the first fixations on the city were made on zoom level 10 onwards.

When the data inside the Salzburg buffer is compared to the overall dataset, it can be observed that the zoom level range used by participants varied more for the whole task. Interestingly, the mean fixation duration for the whole task was smaller compared to the mean fixation duration within Salzburg.

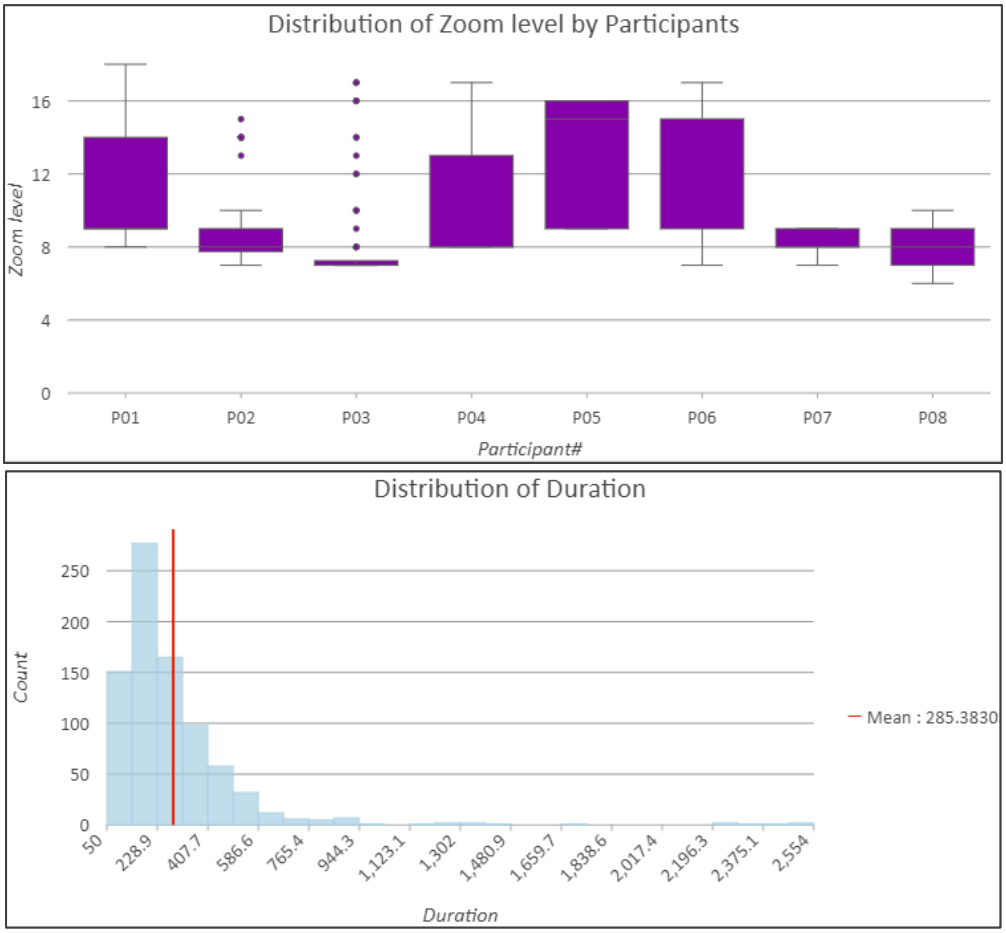


Figure 35 Chart 5 (top): Distribution of zoom level task01, Chart 6 (bottom): Distribution of Duration task 01.

5.3.4 Task 02

Task details: The second map task was to find the department of Geoinformatics, Palacký University Olomouc, particularly to identify its building (KGI). The initial web map window was set to the bounding region of the Czech Republic at a smaller scale. There were seven participants for this task, each of whom was asked to locate the building on the map. The Google basemap was set to the roadmap, similar to the previous task.

Analysis: The focus of analysis for this task was to do proximity analysis using spatial techniques. Since all the participants were from the department itself, it was expected that all participants would have similar behavior and success while using the interactive map. This usage trend was the subject of this analysis. However, to introduce some variation in the task, the participants were divided into two subgroups i.e., the local participants, which were staff members or students native to the Czech Republic, and the foreign participants, which were international students and were temporary residents in the city. In the analysis, any similarities or differences in the map usage for this task were explored.

The first step was to import all fixation point shapefiles for all the participants. The shapefiles were then merged into one main shapefile and projected into WGS 84. Figure 36 shows the extent of the initial web map window and the resulting fixation points by the participants merged to one shapefile.

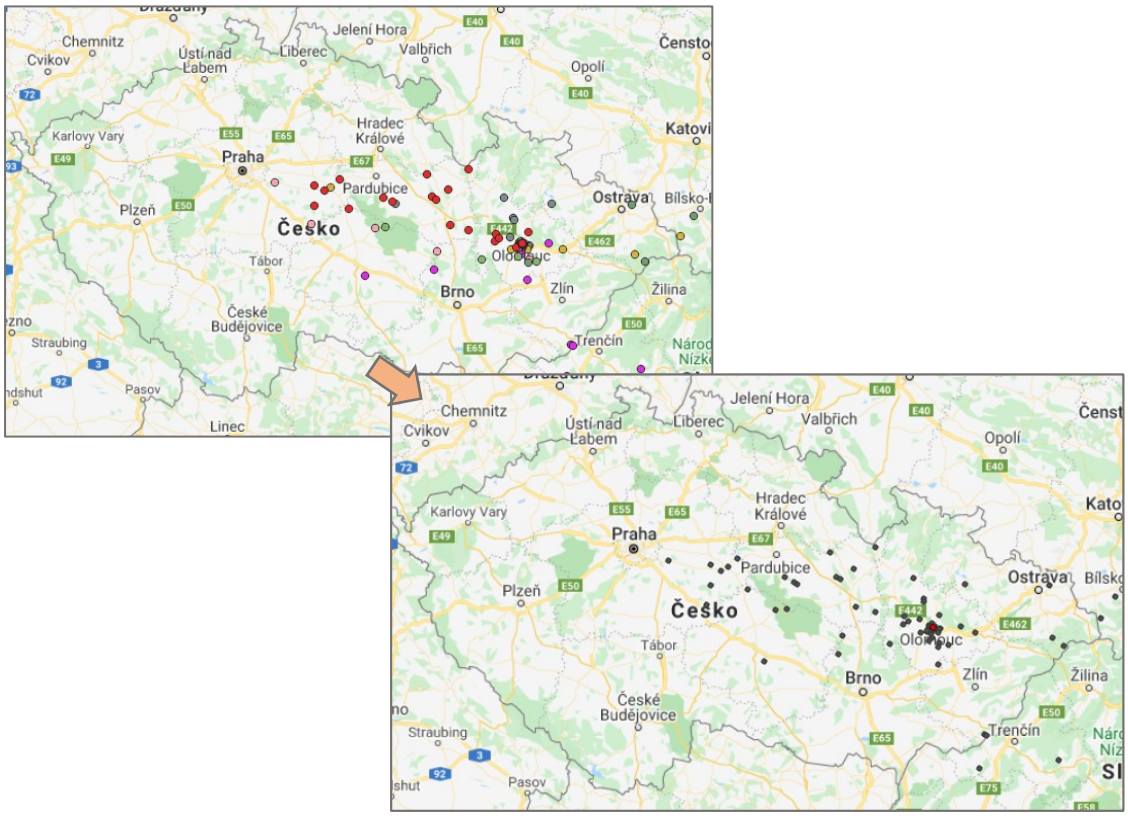


Figure 36 Merged fixation points in ArcGIS Pro for task 02.

The first investigation was to find the trend of usage of zoom level during this task for all participants. A line chart was prepared in ArcGIS for showing the zoom levels for each participant and the associated counts. Figure 37 shows how the most used range for all participants was zoom 16-18, with the highest usage of level 17 shown by participant 04 and the most diverse range of usage shown by participant 06 up to zoom level 21.

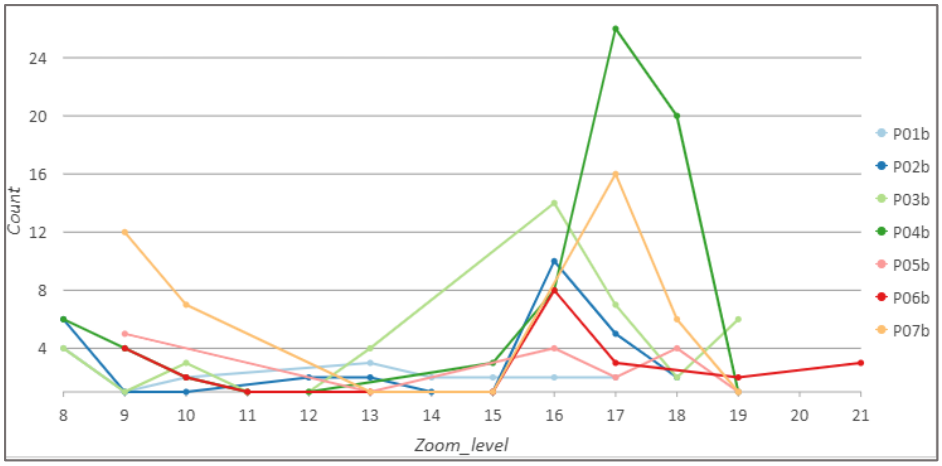


Figure 37 Chart 7 Zoom level distribution by participants.

A point feature dataset containing a single point placed at the location of KGI was created and overlaid on the point dataset. This point was then used to create distances to all other fixation points. Using create origin to destination links tool in ArcGIS, lines with distance labels in feet were generated. The output from this tool gave an idea of the distance variation between the farthest and the closest points.

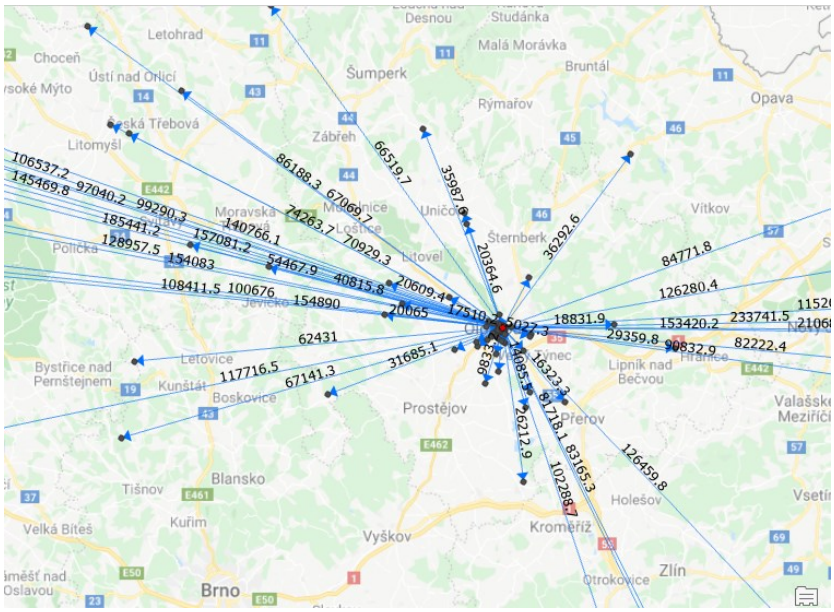
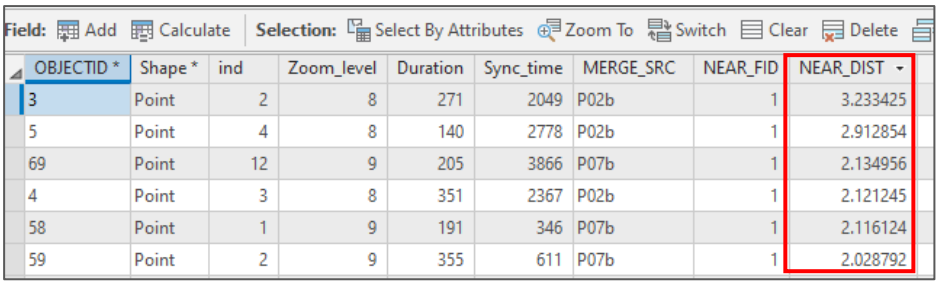


Figure 38 Distance links from KGI.

The distance links are helpful in visual inspection. However, for analysis, a different approach was needed to populate the attribute table with the closest features from the KGI point and the distances to them. For this, the Near tool was used, which in legacy toolset in ArcGIS Pro was point distance. The near distances were generated without any radius specified, to give an idea about distance distribution and the standard deviation through charts. The distance in decimal degrees is shown as an added column in Figure 39.



Field: Add Calculate | Selection: Select By Attributes Zoom To Switch Clear Delete

OBJECTID *	Shape *	ind	Zoom_level	Duration	Sync_time	MERGE_SRC	NEAR_FID	NEAR_DIST
3	Point	2	8	271	2049	P02b	1	3.233425
5	Point	4	8	140	2778	P02b	1	2.912854
69	Point	12	9	205	3866	P07b	1	2.134956
4	Point	3	8	351	2367	P02b	1	2.121245
58	Point	1	9	191	346	P07b	1	2.116124
59	Point	2	9	355	611	P07b	1	2.028792

Figure 39 Distance calculation of nearest fixation points to KGI.

The Near tool was used to create NEAR_DIST column for both the local subgroup and foreign subgroup datasets separately. Distance charts were then created to see the distribution of fixation point distances from the actual target point. Figure 40 and Figure 41 show these distribution charts. The point highlighted in the map shows the cluster of features with the closest distances or the highest bar in the chart. This was one method through which the fixations closest to the subject could be sorted.

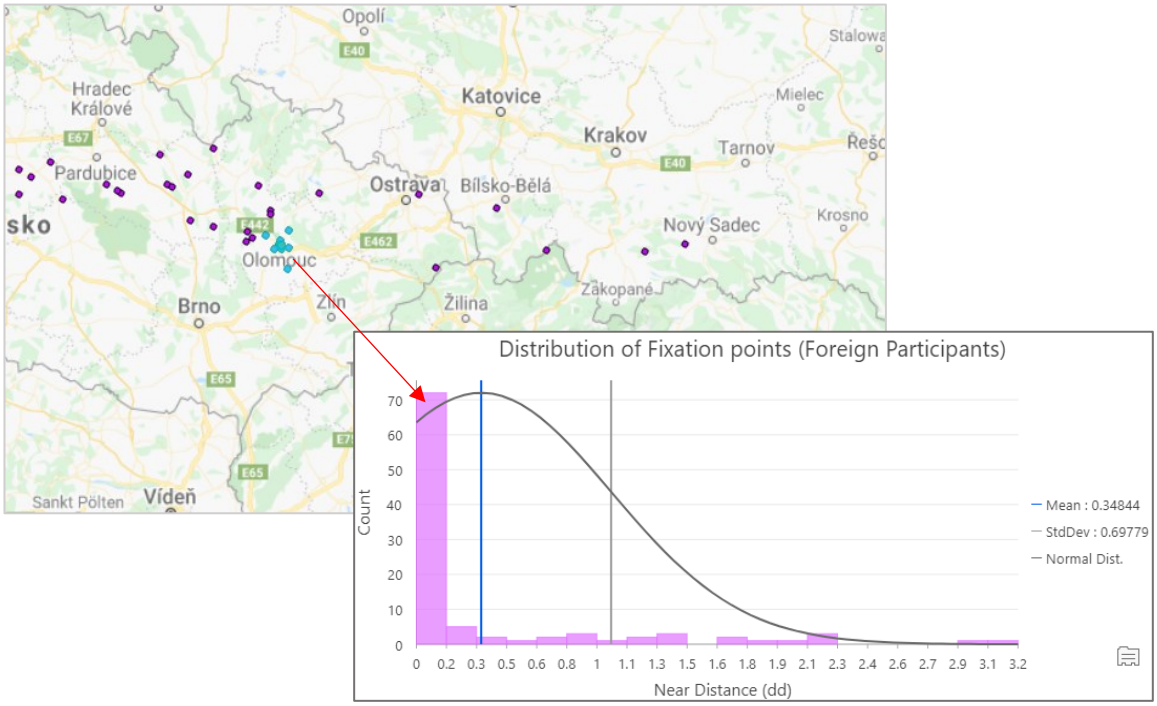


Figure 40 Chart 8 Near distance distribution of Foreign Participant's fixations.

A thorough look at the charts tells that the standard deviation for the foreign participant subgroup was 0.67 which was more than the local participants value of 0.43. The mean for the local participants was 0.15, which was lower than 0.34 for the foreign participants. The conclusion that the local participants were visually closer to the KGI department on the map during this map task was in line with the assumption that the local group would perform better at this map task due to familiarity.

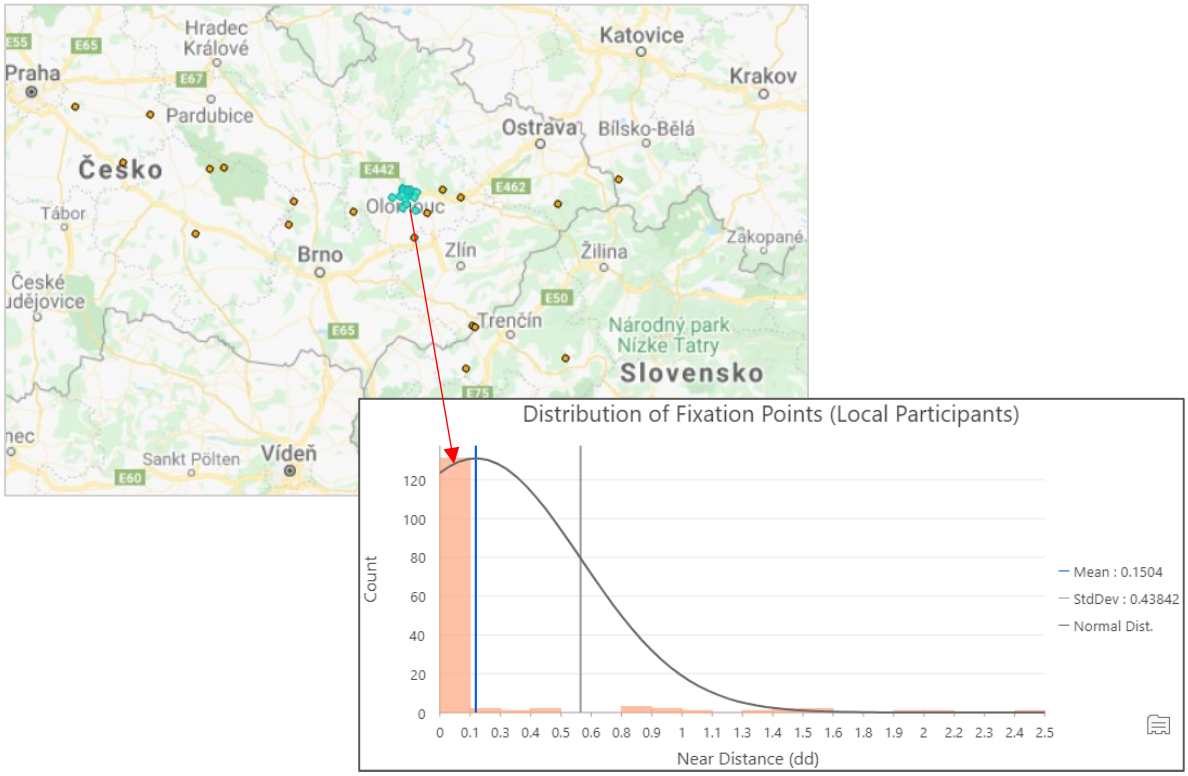


Figure 41 Chart 9 Near distance distribution of Local Participant's fixations.

The Near tool was one way of determining the nearest features, Spatial join tool was also used to demonstrate its capability in analysis regard. Figure 42 shows points within 100m zone of the KGI department and which were further analyzed.

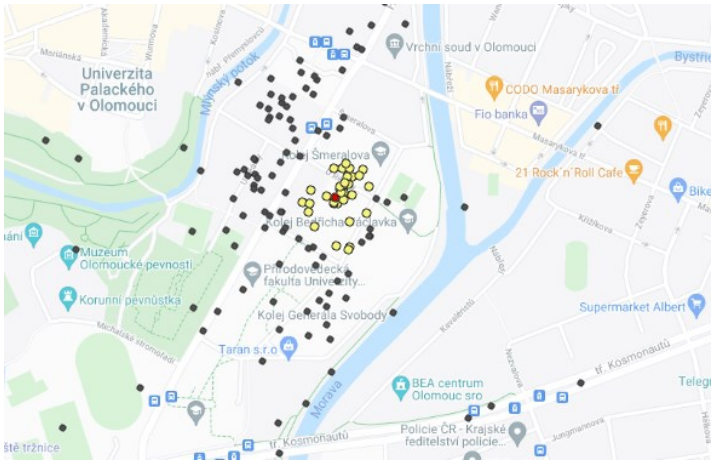


Figure 42 Nearest neighbors within 100m.

When the dataset for the whole task is plotted to find any correlation between zoom level and the duration of the fixation points, no correlation was found. The R2 value indicates zero meaning the duration users spend of a certain fixation is irrespective of the zoom level, hence irrespective of the level of information displayed on the base map, particularly for this task. The reasoning behind this can be owed to the fact that the participants used memory and familiarity more than the zoom dependent content for this task.

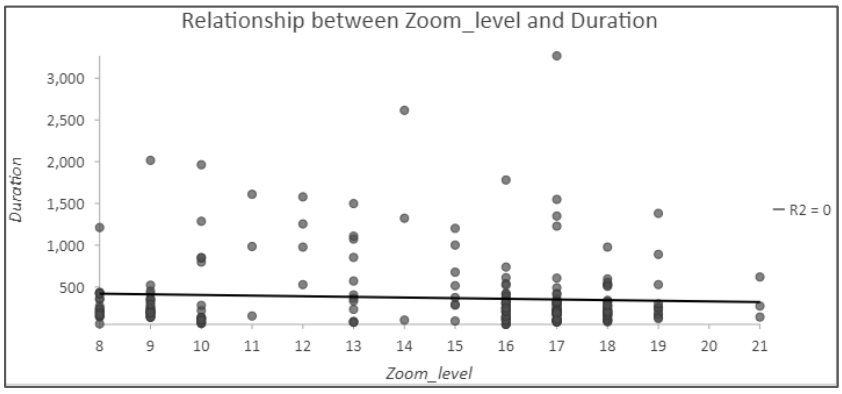


Figure 43 Chart 10 Scatterplot zoom vs. duration task 02.

Some sophisticated visualization techniques can help interpret quick information about the data without doing detailed analysis, such as creating charts and slicing data. One interesting symbology technique is bivariate symbology, where two variables can be represented with different colors. The values of color assigned to each variable depend on the division of data into high-low groups by means of frequency distribution. Figure 44 shows the bivariate symbology applied to the nearest neighbors found through the spatial join.

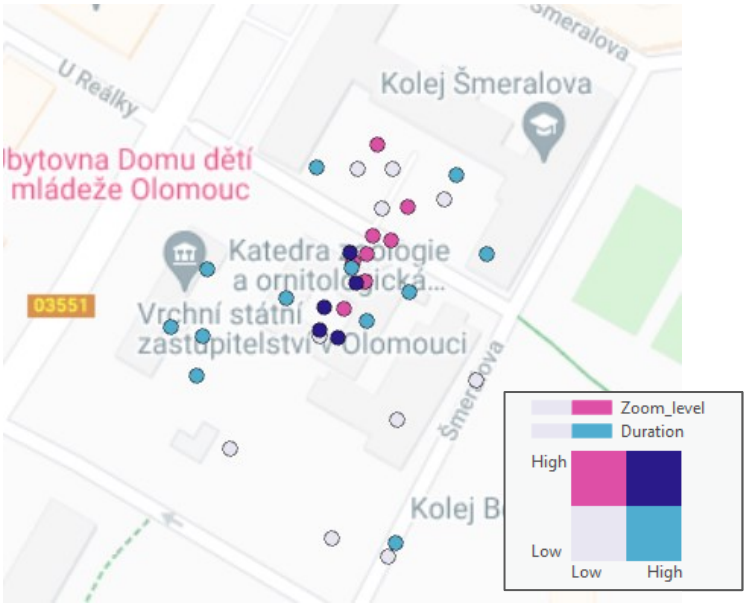


Figure 44 Bivariate analysis of nearest points.

5.3.5 Task 03

Task details: The third map task was to find the Department of Geoinformatics, KGI (Katedra Geoinformatiky, KGI), similar to the last task. The initial web map window was again set to the extent of the Czech Republic. The only difference in this task was the Google basemap which was set to satellite map rather than a labeled roadmap.

Analysis: The analysis in this task was aimed one step above the previous ones in levels of complexity. The focus was to demonstrate the use of additional spatial data for eye-tracking points and how it could elevate the analysis capabilities. Attention was on questions like what are the major differences in user interaction with a base map without labels and a basemap with labels and symbols. And how much extra area was observed on the satellite map in comparison to the roadmap.

To start off, the same routine of importing, merging, and projecting fixation points was applied. Near tool was used to populate near distances in decimal degrees, like in task 02. Bar charts were again created to be able to compare the standard deviation of fixation point distances between the satellite base map and roadmap for local and foreign participants each.

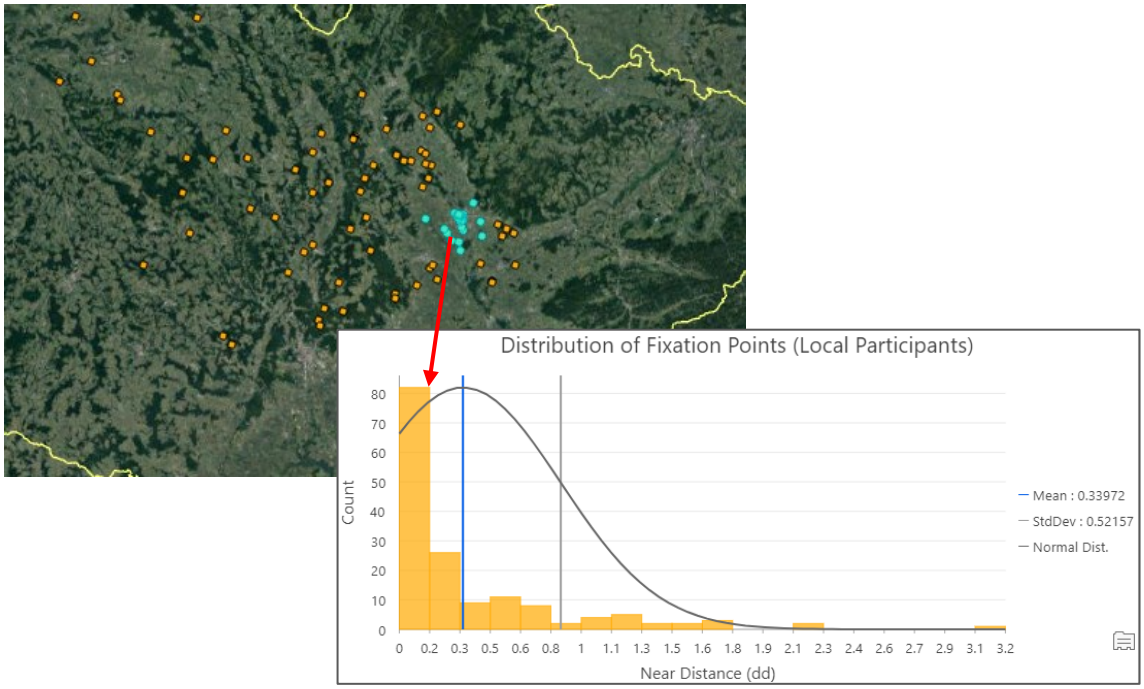


Figure 45 Chart 11 Distribution of near distances of Local Participants.

Figure 45 and Figure 46 show the mean near distance and the standard deviation in the near distances for the local participants and the foreign participants. The mean displacement of the fixations from KGI for the local participants on the satellite basemap was 0.33 dd (decimal degrees) compared to 0.93 for the foreign participants.

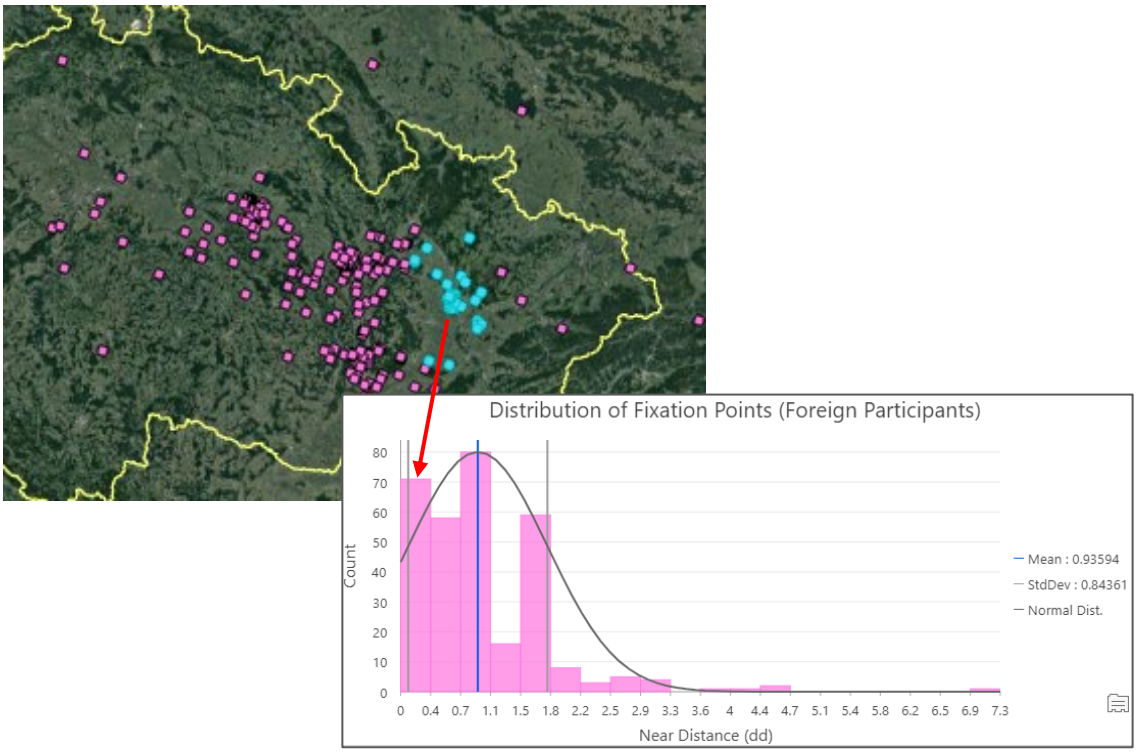


Figure 46 Chart 12 Distribution of near distances Foreign Participants.

Although the local subgroup performed better again than the foreign group in this task, the performance was lower when compared to task 02, where the local participants had a mean displacement of 0.15. The difference for the foreign group in mean and standard deviation was even more drastic, as shown in chart 12. The higher displacements have higher occurrences, implying that the participants had more fixations away from the desired location, i.e., less accurate fixations and the map reading was more complex than the roadmap.

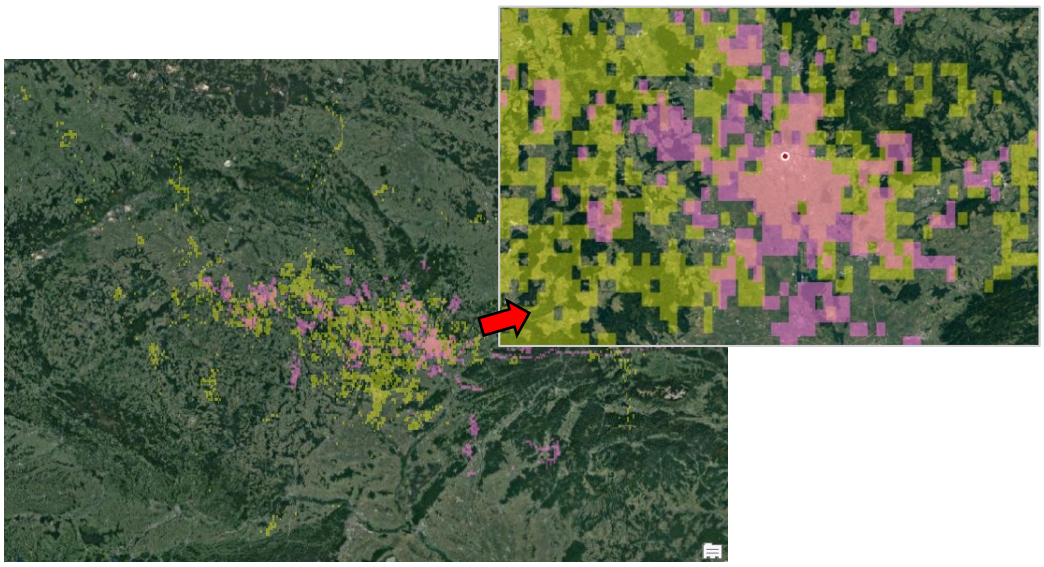


Figure 47 Areas with visual density; yellow: Satellite basemap, pink: Road Basemap

The next analysis was to determine how much extra area was observed without the use of labels on the basemap. For this analysis, raw eye-tracking points were used instead of fixations. Both the raw points shapefiles for the road basemap and the satellite basemap were imported and merged.

Resultingly, there were two feature datasets of points; one containing the points of all participants that were recorded on the road basemap during task 02 and the other containing points of all participants that were recorded on the satellite basemap during task 03. The points were converted to a raster to represent a continuous surface of areas with high gaze point density.

Figure 47 shows the raster pixels in yellow showing the gaze density of points for the satellite map, whereas the pink areas show the areas of visual attention derived from points for the roadmap. At first sight, it can be evaluated that the visual area covered by the satellite task is more than the roadmap one.

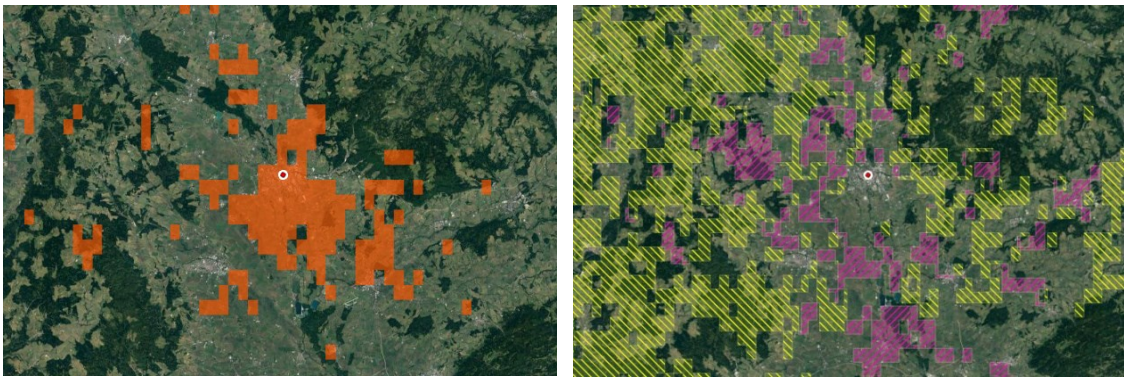


Figure 48 Common visual density areas in orange (left), Exclusive visual density areas in hatched symbology (right).

Attention maps are present in standard ET software as well, although as heatmap variations. Visual focus areas created through GIS software give more manipulation options, especially for analysis. The areas that were common for the two layers in Figure 47 were exported as a new layer using the Intersect tool. The orange areas (in Figure 48) represent the pixels that participants viewed during both tasks, i.e., when observing the basemap with labels and without labels and only imagery. Most of this layer covers the area of Olomouc city.

Using this common layer, exclusive visual density zones were created for both the satellite basemap and the road basemap. Erase tool was used to perform this difference operation. The yellow hatched layer shows the zones of visual density that were made over satellite basemap but were not present in the road basemap. The same was done for road basemap areas by erasing the common areas.

A summarized version of these layers in shown in Figure 49 where the exclusive focus areas for task 02 are represented in hatched pink symbol and for task 03 are represented in hatched yellow symbols.

The raster layers were converted to polygon for easier statistical analysis and without simplifying the geometry. The calculate field option in the attribute table was then used to generate the area for the whole layer. The area calculation was possible by keeping the layers in raster format as well. The area for the common visual focus zones was 0.1424 kilometers, for the imagery only i.e., the satellite basemap layer, it was 1.0310, and for the road basemap, visual focus zone was 0.3710. Hence **2.77** times more map area was explored while using the satellite basemap compared to the road basemap and performing the same task.

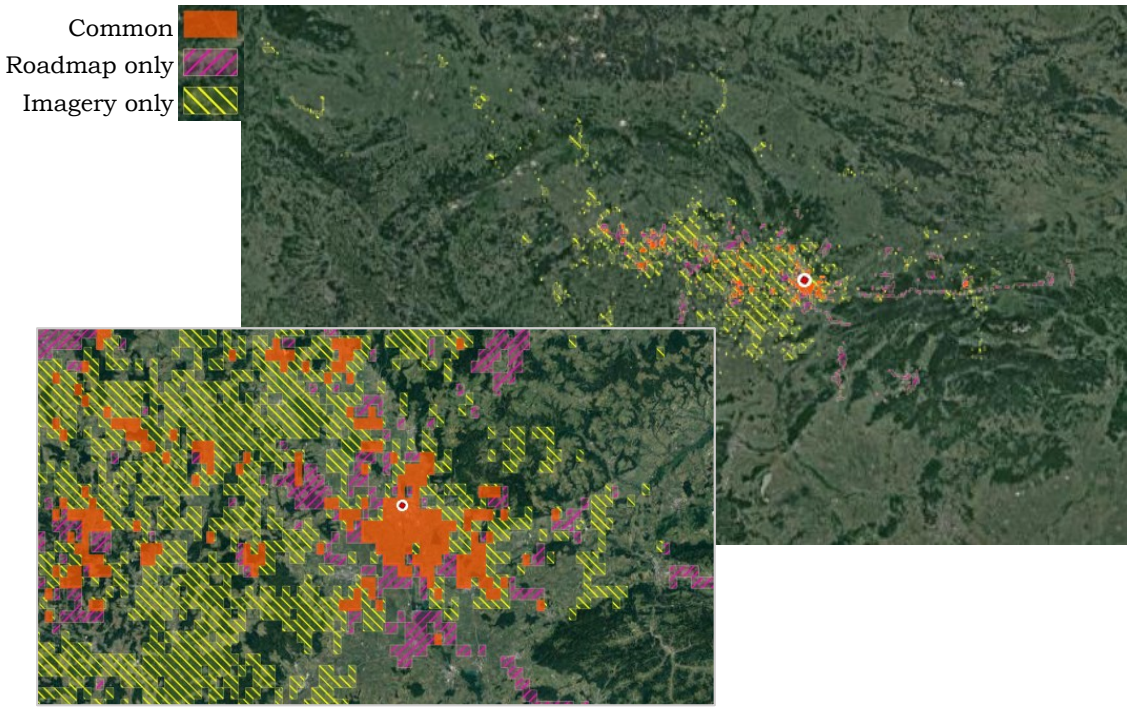


Figure 49 Visual Focus Zones task 02 v task 03

As mentioned in the prompt of this task's analysis, one objective was to demonstrate the usage of additional spatial datasets. The merged fixation points file for all participants in task 03 was again utilized. In addition, the shapefiles of the boundary of the Czech Republic, the boundary of Olomouc city, and the point shapefile of KGI were imported and used to determine which and how many fixations were closer to the country boundary than the city and which ones were closer to the department rather than the city boundary. It was an interesting technique to analyze how close the participants were while trying to find the department without any boundaries or labels on the basemap.

The Near tool was used to find the closest features from the points. Input features were the merged fixation points file. And for the Near features, all three supplementary spatial datasets were specified i.e., KGI point, the Czechia boundary, and the Olomouc boundary. The additional shapefiles were first converted from polygon to lines. However, the points that fall within the polygon would have a Near distance to the polygons as zero. In the

fixation point shapefile, new columns were added. NEAR_FC column specified the nearest feature for every point in the shapefile. Figure 50 shows the fixation points categorized as red, blue, or green based on the feature they are closest to. The symbolization of points in this regard gives a quick impression of the points with spatial proximity to areas of interest on different levels. Figure 51 is one basic statistics generated for this analysis which shows that most participants spent more time near KGI on the satellite map on average. This could be attributed to the attempt at identification of the Department building. The next highest duration on average was near Olomouc, which was perhaps to locate the correct city or town.

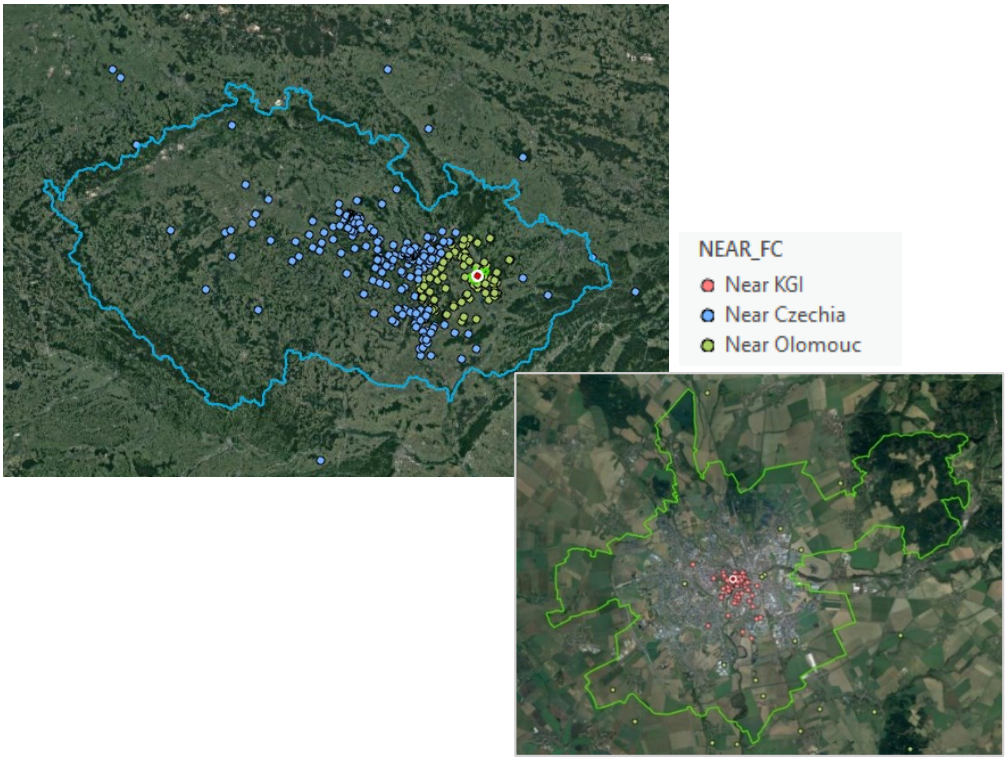


Figure 50 Proximity Analysis with additional spatial data.

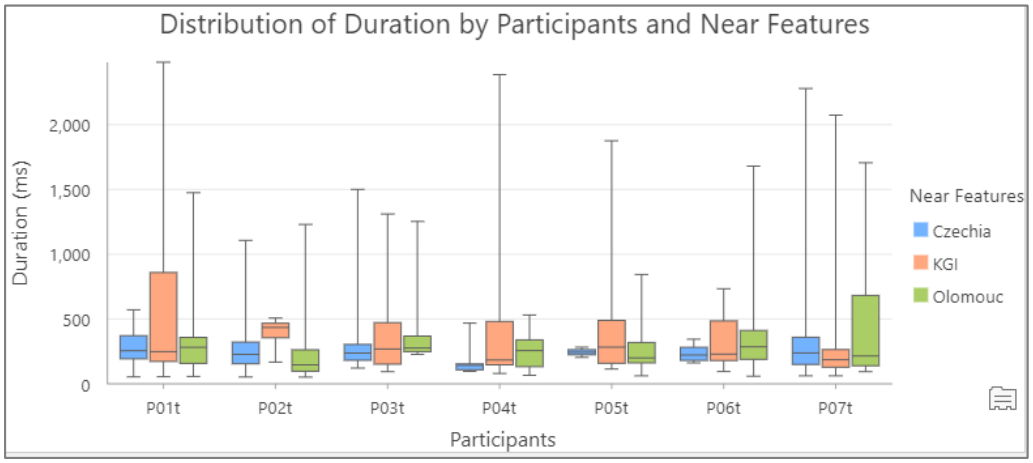


Figure 51 Chart 13 Duration distribution of Participants task 03

5.3.6 Task 04

Task details: The last task was a free viewing map task where the users were asked to freely explore Chicago's city for a fixed time of 50 seconds. The Google basemap in this task was set to the roadmap, and the extent bounds were set to the region of Illinois where Chicago was visible at the center of the map (Figure 53).

Analysis: The main questions addressed for the analysis of this task were, which districts were the participants focusing on the most. And if there was a trend or pattern to the fixation points during this map task. Although more sophisticated analysis could be done here, such as which features did users pay attention to such as parks, hospitals, universities etc. but it required a detailed spatial dataset of the city and filtering it for the needs of this study, all of which was beyond the scope of this diploma thesis where the focus was only to demonstrate the capabilities of the tool rather than a detailed analysis.

The points for all the participant's fixation points were merged in ArcGIS Pro and projected in WGS 84. To evaluate the trend or the point pattern, the Spatial Autocorrelation tool was used. This pattern analysis was just a sample study however. Spatial Autocorrelation generates Moran's I index values which tell how dispersed or clustered the data is and whether the null hypothesis can be rejected or not based on the z-score and p-values which are measures of spatial correlation.

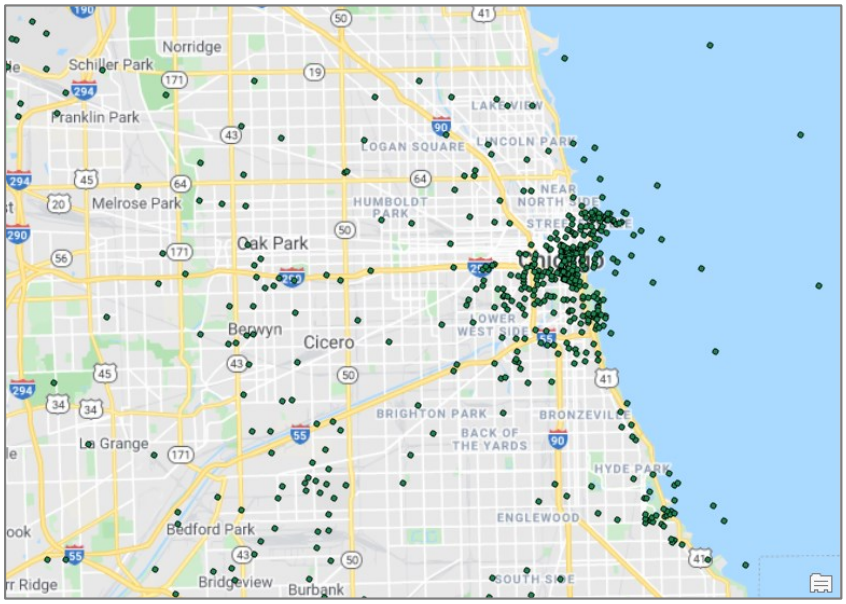


Figure 52 Merged fixation points task 04.

Since the input points for this study are eye-tracking points and not actual spatial features, the whole merged dataset cannot be used. For one the points are dependent on the zoom level, which means that for an accurate analysis the pattern must be evaluated for each zoom level separately. On screen, the distance between fixations remains constant at

every zoom level but when this data is converted to real-world coordinates, higher zoom level points tend to 'spatially cluster' together.

Two new shapefiles were created and merged based on zoom level. Figure 53 shows points for all participants at zoom 7 and zoom 16. These points were given as an input to the spatial autocorrelation tool with duration as the variable in order to judge whether there was clustering and whether at that zoom level the similar values of duration were clustered together.

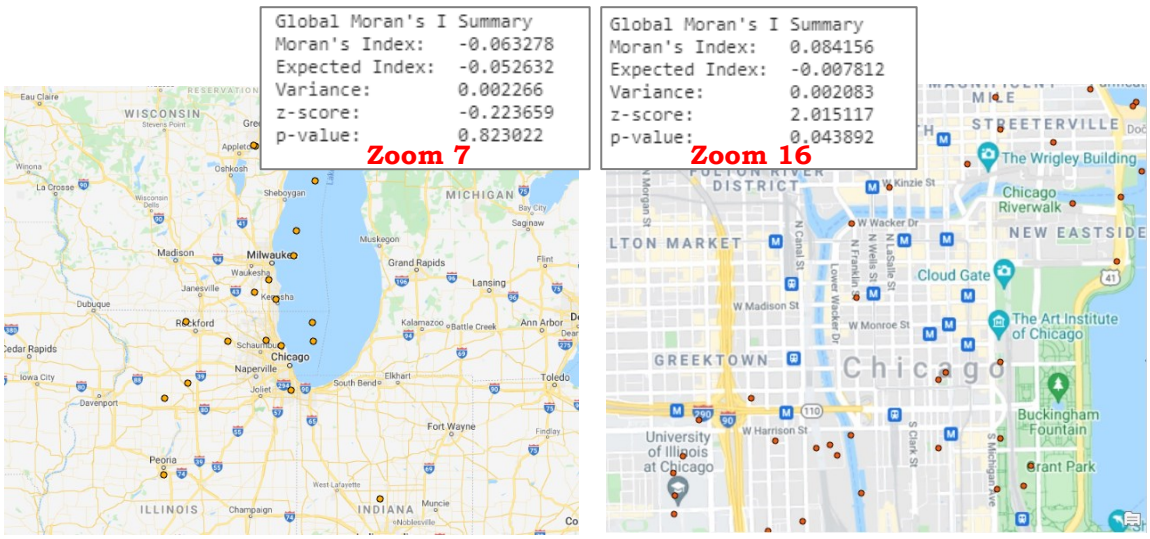


Figure 53 Moran's I indices at zoom 7 & 16.

The generated Moran's I value for points at zoom level 16 indicates a very slight clustering of like values for the duration. This might mean that larger fixations were made in areas closer together. The index can be interpreted for this case because the p-value is statistically significant (0.04). This statistic can be of importance to explore further such as which values are clustered and at what regions. On the other hand, at zoom 7, the negative Moran's Index indicates dispersion or spatial heterogeneity and the p-value is not statistically significant indicating no spatial autocorrelation in the data. Similarly other zoom levels can be investigated in the same manner to unravel point patterns and spatial autocorrelation to give insight into the datasets.

When it comes to visualization of the areas of high fixation density in terms of duration multiple ways can be used. The graduated symbology or proportional symbology is one way, as was demonstrated in previous sections. Kernel density or point density layers can also be generated however manipulating the input parameters to generate an appropriate layer can be confusing just for visualization purposes. The proportional symbology, on the other hand, can give an unpleasing or inconvenient appearance when lots of points are clustered together, and the symbols overlap. Proportional symbols work fine when the scale is flexible, and points are evenly spread out. But for visualization at a fixed scale or fixed zoom level, a tessellation layer can be generated. This method shows hexagons of a high or low density of points.

To achieve this visualization, a tessellation was first created with a basic unit of 1km. In the case of this task, 1km was considered sufficient to distinguish different zones of Chicago. This tessellation layer was combined with the fixation points layer using the Spatial Join tool. The count of points falling within each unit was written to the table and which could be categorized with either natural breaks or manual class size in the symbology tab. The transparency was decreased to create an overlay appearance. Figure 54 shows the fixation density of points for the whole dataset. Figure 55 shows the fixation density of points recorded at zoom level 16 only. This type of visualization can help determine which areas or which districts were mostly focused on by the participants. It is an effective way of quick visual inspection. For instance, at zoom 16, most fixations were made over the Navy Pier.

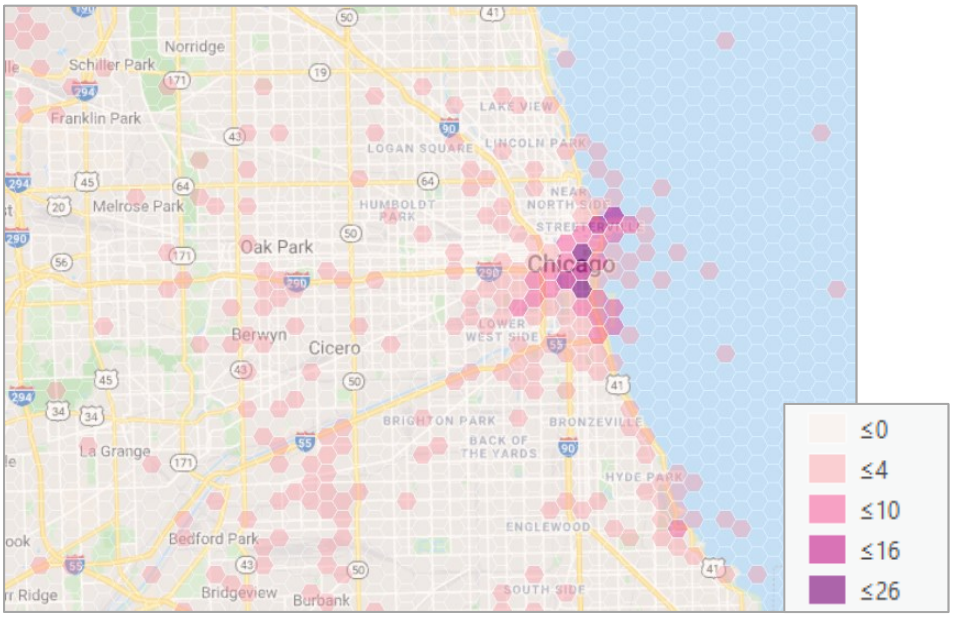


Figure 54 Fixation density overlay (all fixations).

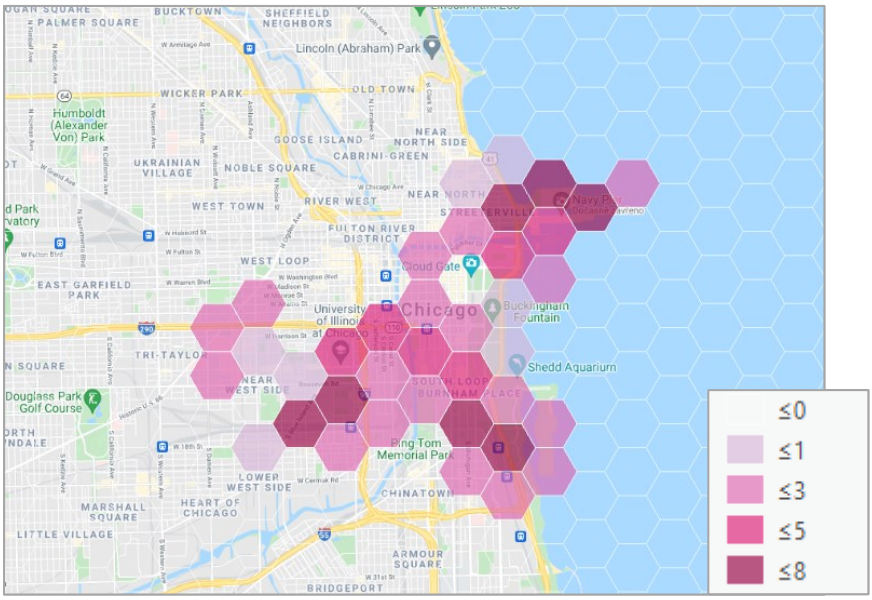


Figure 55 Fixation density overlay (fixations at zoom 16).

6 RESULTS

The results described in this chapter are a summarized conclusion of Chapter 4, which focused on creating a tool, and Chapter 5, which enlisted the possibilities of use-cases of the tool.

6.1 ET2Spatial

ET2Spatial is a desktop-based tool that allows the conversion of eye-tracking points, recorded on an interactive web map, to real-world coordinates. The tool is built on python and in the form of a stand-alone executable file. It is capable of running without any installations. The main inputs of the tool are:

- Raw eye-tracking gaze points, exported from the ET software in the form of a text file.
- Fixation eye-tracking points, exported from the ET software in the form of a text file.
- User interaction data, containing information about the user's logged inputs with the web map, in the form of xml file.

The output file contains the geographic coordinates and attribute values such as zoom level, sequence of points, and fixation duration. The tool gives outputs in three formats, ESRI Shapefile, GeoJSON, and CSV. A separate output is created for each participant. Additionally, a file containing information on the projection of converted points is also generated. The packaged file, along with the executable, contains a readme file that gives information on the input file structure requirements of the tool.

The tool is mainly configured to work with the SMI eye-tracking system; for eye-tracking data and MapTrack (Ruzicka, 2012); for the user-interaction data, both of which are commonly employed in the current research environment of the eye-tracking lab at the university. Hence, ET2Spatial is expected to contribute to the existing tools and framework that are already in place at the Department of Geoinformatics, Palacký University Olomouc. The tool, however, is created using open-source technologies and has the potential of being scaled up.

The conversion of points from screen coordinates to geographic coordinates is done primarily through forward and reverse Web Mercator projection formulae, used by the web map such as Google Maps, OSM etc. themselves. Regardless, the accuracy of the output from the tool has been verified through a structured mechanism (section 5.1).

The tool has been examined for performance on multiple participants, and in total, 34 datasets were put to the test for conversion. The tool allows the conversion of a new dataset without restarting the whole application using the Reset feature. Notifications are also displayed when different functions are performed (Figure 56).

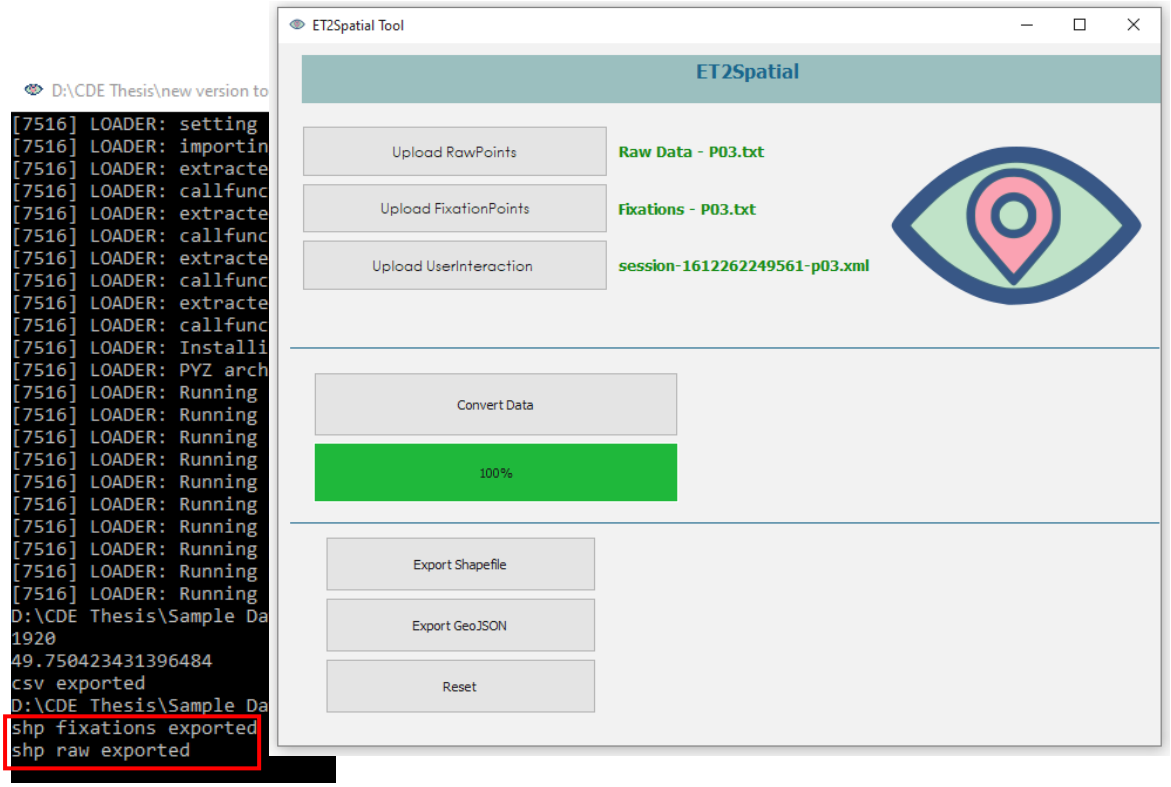


Figure 56 ET2Spatial Interface.

6.2 Sample Use-cases

The main goal of the tool was to ease the eye-tracking analysis of web maps, which was evaluated and demonstrated through multiple use-case scenarios (section 5.3.2). The eye-tracking data of multiple participants was converted through the tool and visualized over the same basemap. Different symbology techniques were applied to the eye-tracking data in GIS software to compare it to the options present in standard ET software. In addition, some traditional ET software features such as scanpaths and attention zones were recreated in the GIS software to prove the capability of visualizing spatial ET data in a GIS environment. Overall, the visualization of converted spatial data in ArcGIS and QGIS provided promising results with features such as scale-based rendering of points, multiple custom paths generation, attribute-based categorization, and custom labeling.

Apart from the successful data visualization capabilities, four test tasks were carried out to test the advanced spatial functionalities on eye-tracking points. The four ET experiments were taken by seven participants who were asked to use the web map through common interaction methods such as zoom, scroll etc. and solve the map tasks:

- Task 01: Find Salzburg city center on the map, starting from Olomouc as the extent window. Google roadmap (with labels) as basemap.
- Task 02: Find KGI, UPOL building on the map, starting from the Czech Republic as extent window, Google roadmap (with labels) as basemap.

~ Task 03: Find KGI, UPOL building, starting from the Czech Republic as an extent window. Google satellite map (without labels) as basemap.

~ Task 04: Explore the city of Chicago, starting from Chicago as an extent window. Google roadmap (with labels) as basemap.

Each task was evaluated through different spatial techniques such as buffer zone analysis for AOI, proximity analysis, nearest features analysis, determination of point density and spatial auto-correlation, general statistical analysis of attributes, and usage of supplementary spatial data with ET points. Figure 57 shows the summarized usage of zoom level on the interactive web environment by the participants in all the above-mentioned tasks as well as their mean fixation duration. All the analyses demonstrated capable results and possibilities of the usage of the tool in research.

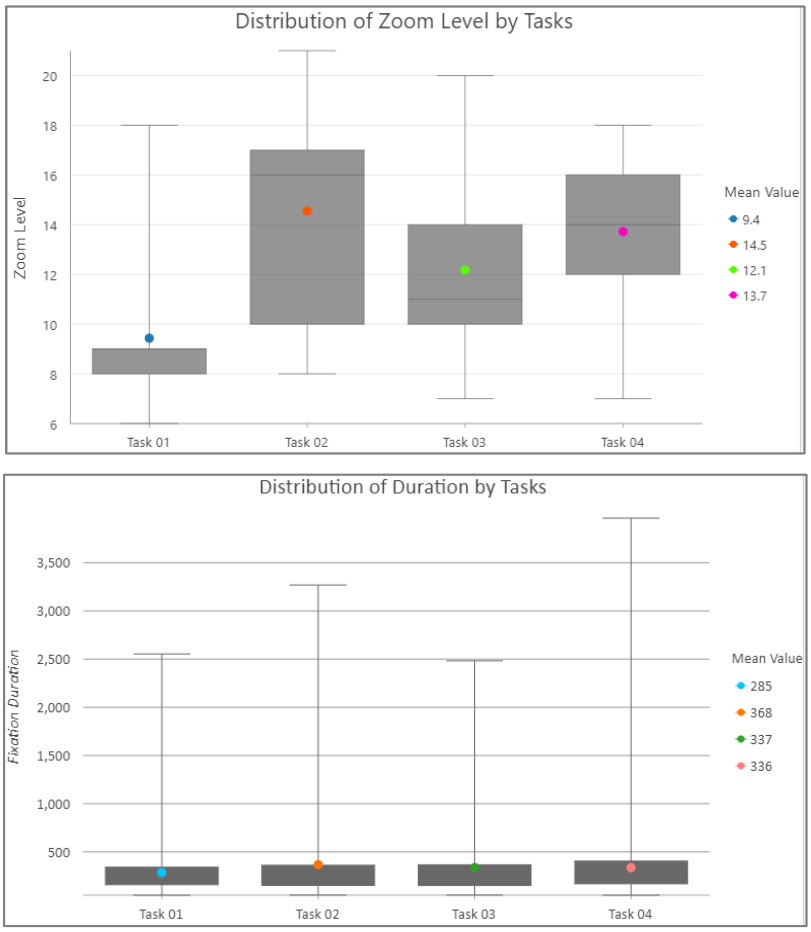


Figure 57 Chart 14 (top) Summarized zoom level usage in tasks, Chart 15 (bottom); Summarized fixation duration in tasks.

7 DISCUSSION

The driving force behind this thesis study was the cumbersome analysis techniques for eye-tracking data captured on interactive stimuli, particularly web maps. The tool developed as a result addressed these issues by converting the points from one coordinate system to another. The priority was to develop a solution that would, at a basic level, cater to the needs of the research environment at the department but would also be able to be used independently. The tool created, hence, is very niche and works with specific technologies as of now i.e., MapTrack and SMI RED ET system. MapTrack is open-source, and can be used by anyone through the internet with the results of interaction readily available in xml format. Thereby it rids the user of dependency on a third-party propriety system to be able to use ET2Spatial. On the other hand, although ET2Sspatial is configured with the output from SMI BeGaze, the tool can theoretically work with output from any ET system if naming conventions for the columns are followed. Apart from MapTrack, the python module browserhistory (section 4.1) can fetch user-interaction data without relying on any other software. The tool can be scaled to include this module if the circumstances favor its usage over MapTrack.

The tool was mainly developed on open-source technologies such as modules and libraries in python. The sufficiently available documentation and community support in this regard helped ease the development process of the tool. One thing to be pointed out is that in the code, Pandas module was chosen for manipulation of data considering eye-tracking data structuring and analysis requirements. Overall, this was a good choice, but it had certain limitations, e.g., in row iteration during data stitching from different data frames and sources, a problem that was eventually solved by using dictionaries instead of dataframes in specific parts of the code.

Notifications were added to the tool for user convenience on functions such as imports, exports, and conversion. An additional notification appears when the user forgets to input one or more files. However, the code is not intelligent enough to prompt the user if the files of participants are mixed up. To compensate for that, the file name of the uploaded files appears on the tool window to give the user an inclination of which participant data is being handled. Although the code for the tool has basic level error handling, there are many scenarios that rely on user vigilance instead of code intelligence. On testing, one such scenario was found to be the incorrect keyboard logs in input files.

The tool takes the input of one participant's files at a time, which in large experiments could be time-consuming. The tool can be scaled to have a bulk file upload feature, but to simplify the continuous conversion process for multiple participants, the reset feature was added and which proved to be helpful during the testing on 34 datasets used in the evaluation.

A fair advantage of transforming the eye-tracking points into the geographic space was the availability of spatial functions that could be applied to the eye-tracking data of several participants. These GIS functions could help in analyzing the ET data from a spatial perspective. For most of the use-cases and visualizations, ArcGIS was used because of the

range of spatial operations available in the software. The open-source alternative to it, QGIS, was also tested with data visualizations and basic level spatial functions. An important aspect of the analysis of ET points in GIS software is the usage of the cartographic renderer. These cartographic renderers need to be the same as the ones used during the eye-tracking experiment. Conveniently, Both ArcGIS and QGIS had ways to include custom basemaps such as Google satellite basemap or roadmap.

The evaluation of the tool was done through several use-case scenarios, some of which compared the attributes such as any correlation between zoom level and duration. When ET points are imported in the GIS software after conversion through the tool, they are displayed as spatial features, but these are mock spatial features in essence as they do not have an actual ground presence. This needs to be kept in mind when performing analysis and choosing spatial functions and attributes of features. An example is the correlation study between ET point distances and the zoom level. The correlation, in this case, would always exist because the distances being considered are spatial distances, and when points are projected from screen space to geographic space, points at higher zoom levels would tend to cluster together. A point pattern analysis for ET features with these attributes would hence hold no value.

The case studies presented in this thesis are all done on basemaps such as on Google Maps API. However, there is a possibility to overlay additional vector or raster datasets on the basemaps. One such example can be of the CLC (Corrine Landcover Classes) dataset from Copernicus where the usability of the products can be tested and compared against the underlying imagery basemap. There is also a vast potential of research in the field of remote sensing and eye-tracking where geo-referenced eye-tracking datasets can be used to impart human intelligence to the Machine Learning algorithms and for that purpose Sentinel datasets can be used as stimuli.

Since the participant group was small, consisting of only seven-eight people, the results for some of the analyses do not hold a conclusive value rather show a good range of the capabilities that GIS software and ET data could offer together. Overall, the tool can be a resourceful aid in future research carried out at the department and elsewhere. The pilot experiments with associated analysis shown in this study could be used as a guide in manipulating the output data from the tool.

8 CONCLUSION

Standard eye-tracking systems offer good evaluation techniques for static stimuli when it comes to geovisualization products. However, the current practices carried out for analysis with eye-tracking on interactive mediums such as web maps are very cumbersome and time-consuming. Research done in this regard has been scarce, and the availability of free, open-source tools addressing the issue is far lesser.

The goal of this study was to create a tool that would solve the problems of eye-tracking analysis on dynamic interactive web maps and supplement the research eco-system in the eye-tracking lab at UPOL. The tool developed as a result of this thesis, namely ET2Spatial, converts screen coordinates recorded during an eye-tracking experiment to real-world geographic coordinates.

ET2Spatial was developed in python using a variety of modules. The tool takes three input files; the raw ET points, fixation points, and the user interaction data. These input datasets are pre-processed, synchronized based on timestamps, and stitched together. The main conversion of points relies on the Web Mercator projection formulas. Eventually, the tool offers export in shapefile format along with a CSV and a geojson format.

The tool was evaluated in terms of accuracy of output, performance, and its general usability. Four eye-tracking experiments were carried out with a small group of participants who solved map tasks on interactive basemaps such as Google Maps. In total, 34 eye-tracking datasets were passed through the tool for conversion. Post-conversion, these ET datasets were imported in GIS software such as ArcGIS and QGIS to test the visualization capabilities as well as the evaluation of points with spatial techniques. The pilot studies demonstrated good results and multiple options for visualizing eye-tracking data in the GIS environment and the usage of spatial functions to amplify the scope of analysis.

Using GIS environments to analyze and visualize eye-tracking points proved to open many new potential avenues of research questions and capabilities. The custom symbology and labeling options in GIS software gave much more control over the visualization aspect of cartographic products compared to standard ET software. The tool also eased the problem of visualizing multiple participant's ET data on the same layer, which was one of the main driving forces behind tool creation. The manipulation of ET points from a spatial perspective gave additional investigation capabilities.

ET2Spatial builds on the existing technology developed at the department, i.e., MapTrack, and aims to provide a harmonized framework for carrying out future research. Despite being slightly technology-specific, the tool is open-source and can be used anywhere provided an eye-tracking system exists. The tool has the potential to be scaled up and can be employed for usability studies of interactive cartographic mediums as well as analysis of human interaction and cognition with web maps.

REFERENCES AND INFORMATION SOURCES

ANDRIENKO, G., ANDRIENKO, N., BURCH, M., & WEISKOPF, D. (2012). Visual analytics methodology for eye movement studies. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2889-2898.

ENOCH, J. M. (1959). Effect of the size of a complex display upon visual search. *JOSA, 49(3), 280-286.*

GÖBEL, F., KIEFER, P., & RAUBAL, M. (2017). FeaturEyeTrack: a vector tile-based eye tracking framework for interactive maps. In *Societal Geo-Innovation: Short Papers, Posters and Poster Abstracts of the 20th AGILE Conference on Geographic Information Science. Wageningen University and Research* (pp. 9-12).

HERMAN, L., POPELKA, S., & HEJLOVA, V. (2017). Eye-tracking analysis of interactive 3d geovisualization. *Journal of eye movement research*, *10*(3), 2.

KIEFER, P., GIANNOPOULOS, I., RAUBAL, M., & DUCHOWSKI, A. (2017). Eye tracking for spatial research: Cognition, computation, challenges. *Spatial Cognition & Computation*, *17*(1-2), 1-19.

KIEFER, P., & GIANNOPOULOS, I. (2012). Gaze map matching: mapping eye tracking data to geographic vector features. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems* (pp. 359-368).

OOMS, K., COLTEKIN, A., DE MAEYER, P., DUPONT, L., FABRIKANT, S., INCOUL, A., ... & VAN DER HAEGEN, L. (2015). Combining user logging with eye tracking for interactive and dynamic applications. *Behavior research methods*, *47*(4), 977-993.

OOMS, K., DE MAEYER, P., FACK, V., VAN ASSCHE, E., & WITLOX, F. (2012). Interpreting maps through the eyes of expert and novice users. *International Journal of Geographical Information Science*, *26*(10), 1773-1788.

PAPENMEIER, F., & HUFF, M. (2010). DynAOI: A tool for matching eye-movement data with dynamic areas of interest in animations and movies. *Behavior research methods*, *42*(1), 179-187.

PFEIFFER, T. (2012, March). Measuring and visualizing attention in space with 3D attention volumes. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (pp. 29-36).

RŮŽIČKA, O. (2012). Profil uživatele webových map. *Univerzita Palackého v Olomouci. Thesis.*

SMI (SensoMotoric Instruments). (2017). In *BeGaze Manual* (pp. 339–375). Accessed at: https://gazeintelligence.com/smi-product-manual

Internet Sources:

Generate a Projection File (.prj) using Python [Online]. 2015 [cit. 2021-04-02]. Geospatiality. Available Online: https://glenbambrick.com/2015/08/09/prj/

Geojson 2.5.0 [online]. [cit. 2021-04-13]. Python Software Foundation: Sean Gillies. Available Online: https://pypi.org/project/geojson/

How Spatial Autocorrelation (Global Moran's I) works [Online]. [cit. 2021-05-03]. ArcGIS Pro. Available Online: https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/h-how-spatial-autocorrelation-moran-s-i-spatial-st.htm

Jupyter [online]. [cit. 2021-01-20]. Project Jupyter. Available Online: https://jupyter.org/install.html

Map Projection [Online]. 2012 [cit. 2021-03-22]. USNA. Available Online: https://www.usna.edu/Users/oceano/pguth/md_help/html/mapb0iem.htm

NumPy v1.20 Manual [online]. 2008-2021 [cit. 2021-03-06]. The SciPy community. Available Online: https://numpy.org/doc/stable/

Pandas Documentation [online]. 2008-2021 [cit. 2021-03-03]. Pandas Development Team. Available Online: https://pandas.pydata.org/docs/

Pyinstaller 4.3 [online]. [cit. 2021-04-23]. Python Software Foundation. Available Online: https://pypi.org/project/pyinstaller/

Pip 21.1.1 [online]. [cit. 2021-02-30]. Python Software Foundation. Available Online: https://pypi.org/project/pip/

Pyshp 2.1.3 [online]. [cit. 2021-04-10]. Python Software Foundation: Joel Lawhead. Available Online: https://pypi.org/project/pyshp/

Python 3.8.10 Documentation [online]. 2001-2021 [cit. 2021-02-30]. Python Software Foundation. Available Online: https://docs.python.org/3.8/

PyQt5 5.15.4 [online]. [cit. 2021-03-20]. Python Software Foundation: Riverbank Computing Limited. Available Online: https://pypi.org/project/PyQt5/

Slippy map tilenames [Online]. 2019 [cit. 2021-03-25]. OpenStreetMap Wiki. Available Online: https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames

Spyder [online]. [cit. 2021-01-20]. Spyder IDE. Available Online: https://www.spyder-ide.org/

Web Mercator projection [Online]. 2021 [cit. 2021-03-27]. Wikipedia. Available Online: https://en.wikipedia.org/wiki/Web_Mercator_projection

Xmltodict 0.12.0 [online]. [cit. 2021-03-10]. Python Software Foundation. Available Online: https://pypi.org/project/xmltodict/

Zoom Levels [Online]. 2019 [cit. 2021-03-25]. OpenStreetMap Wiki. Available Online: https://wiki.openstreetmap.org/wiki/Zoom_levels

ATTACHMENTS

Code Snippets from ET.py (Full code on Pen Drive)

```
# -*- coding: utf-8 -*-

"""
Created 2021
Palacky University Olomouc
@author: Minha Noor Sultan
Copernicus Masters in Digital Earth
"""

class Ui_MainWindow(QDialog):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("ET Tool")
        MainWindow.setWindowIcon(QtGui.QIcon('icon256.png'))
        MainWindow.resize(800, 600)
        font = QtGui.QFont()
        self.retranslateUi(MainWindow)
        self.RawButton.clicked.connect(self.RawFiles)
        self.FixButton.clicked.connect(self.FixFiles)
        self.xmlButton.clicked.connect(self.XFiles)
        self.convertButton.clicked.connect(self.Convert)
        # self.exportCSVbutton.clicked.connect(self.expCSV)
        self.exportJsonButton.clicked.connect(self.expJSON)
        self.exportShpButton.clicked.connect(self.expSHP)
        self.Reset.clicked.connect(self.ResetProgress)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)


    def RawFiles(self):
        #IMPORT RAW POINTS FILE AND STORE IN DF
        global et_raw
        global path_r
        filetype = 'Text file(*.txt)'
        rname=QFileDialog.getOpenFileName(self,'Open
        File',filter=filetype)
        path_r=os.path.normpath(rname[0])
        et_raw = pd.read_csv(path_r,delimiter="\t")
        if (path_r is not None):
            self.labelR.setText(os.path.basename(path_r))
            self.labelR.setStyleSheet("color: green; font-weight: bold;
            font-size: 10pt")



    def Convert(self):
        #CONVERT THE COORDINATES
        try:
            et_raw
            et_fix
            path_x
        except NameError:
            msg= QMessageBox()
            msg.setIcon(QMessageBox.Critical)
            msg.setText('Upload all files!')
```

```
            msg.setWindowTitle('WARNING')
            msg.exec()

        else:
            self.progressBar.setValue(1)
            xtree = et.parse(path_x)
            xroot = xtree.getroot()
            #store Participant number
            global p_num
            p_num=xroot.find("var").text
            #store Map Size
            m_size=xroot.find("mapsize").text
            m_size=m_size.split('x')
            m_size=pd.to_numeric(m_size)
            mapWidth=int(m_size[0])
            mapHeight=int(m_size[1])
            print(mapWidth)
            #screen center
            center_x=m_size[0]/2
            center_y=m_size[1]/2
            #store Starting time
            TOD_start=xroot.find("start").text

            #convert xml to dataframe
            dfcols = ['MapTrack_RT', 'Action','MapCenter','Zoom_level']
            subsetxml_mt = pd.DataFrame(columns=dfcols)

            for i in xtree.iter(tag='event'):
                subsetxml_mt = subsetxml_mt.append(
                    pd.Series([i.find('time').text, i.find('name').text,
                        i.find('center').text, i.find('zoom').text],
                            index=dfcols), ignore_index=True)

            #format MapCenter into lat and long
            #split
            subsetxml_mt[['MapCenter_lat','MapCenter_long']]=
              subsetxml_mt.MapCenter.str.split(expand=True)
            #remove characters
            subsetxml_mt['MapCenter_lat'] =
              subsetxml_mt['MapCenter_lat'].str.strip('(,')
            subsetxml_mt['MapCenter_long'] =
              subsetxml_mt['MapCenter_long'].str.strip(')')
            #change data type
            subsetxml_mt['MapCenter_lat']=
            pd.to_numeric(subsetxml_mt.MapCenter_lat, downcast='float')
            subsetxml_mt['MapCenter_long']=
            pd.to_numeric(subsetxml_mt.MapCenter_long, downcast='float')
            subsetxml_mt['Zoom_level']=
            pd.to_numeric(subsetxml_mt.Zoom_level, downcast='float')
            subsetxml_mt['MapTrack_RT']=
            pd.to_numeric(subsetxml_mt.MapTrack_RT, downcast='float')
            subsetxml_mt['MapTrack_RT']=
            subsetxml_mt['MapTrack_RT']*1000 #convert to milliseconds
            #index refresh
            subsetxml_mt=subsetxml_mt.reset_index(drop=True)
            #convert to dict
            subsetxml_mt=subsetxml_mt.to_dict(orient='records')
            self.progressBar.setValue(10)
            #DATA FORMATTING RAW DATA
```

```
            #convert the column names to strings
            et_raw.columns = et_raw.columns.astype("str")
            et_raw.columns = et_raw.columns.str.strip()

            #slice rows by selection
            test_rows=et_raw.loc[et_raw['Content']=='F2']
            ind=test_rows.index
            subset_et_raw=et_raw.loc[ind[0]:ind[1]]

            #slice columns by selection
            subset_et_raw=
              subset_et_raw[['Time of Day [h:m:s:ms]',
              'RecordingTime [ms]','Point of Regard Right X [px]',
                'Point of Regard Right Y [px]','Participant',
                                        #'Gender','Age']]

            #rename columns
            subset_et_raw=subset_et_raw.rename
             (columns={'Time of Day [h:m:s:ms]': 'TOD',
                               'RecordingTime [ms]':'RT',
                               'Point of Regard Right X [px]'
                                   :'Screen_x',
                               'Point of Regard Right Y[px]'
                                   :'Screen_y' })
            #RAW DATA SYNCHRONIZATION

            #calculating length for synchronization
            length=subset_et_raw.loc[ind[0],'RT']-
                      et_raw.loc[0,'RecordingTime [ms]']

            #calculating length 2 for sync of fix data end time
            length2=subset_et_raw.loc[ind[1],'RT']-
                     subset_et_raw.loc[ind[0],'RT']

            #formatted Recording time, starting at f2
            subset_et_raw['Format_RT']=subset_et_raw['RT']-
                    subset_et_raw.loc[ind[0],'RT']

            #exclude the row with -
            subset_et_raw =
                 subset_et_raw.loc[(subset_et_raw["Screen_x"]!= '-')
                     |(subset_et_raw["Screen_y"]!= '-')]

            #drop index
            subset_et_raw=subset_et_raw.reset_index(drop=True)
            #add index column
            subset_et_raw['ind'] =subset_et_raw.index

            #convert from df to dict
            subset_et_raw=subset_et_raw.to_dict(orient='records')
            self.progressBar.setValue(15)


    def expCSV(self,raw,fix):
        #csv raw
        header2 = ["ind","Zoom_level", "Format_RT", "Longitude",
          "Latitude"]
        et_raw_combined.to_csv(raw, columns = header2)
        print('csv exported')

    def expJSON(self):
```

```
        #GeoJSON raw
        featuresr = []
        insert_featuresr = lambda X:

featuresr.append(geojson.Feature(geometry=geojson.Point((X["Longitude"],
                                                         X["Latitude"])),
                                 properties=dict(Zoom=X["Zoom_level"],
                                                 Time=X["Format_RT"])))
        et_raw_combined.apply(insert_featuresr, axis=1)

        filename=p_num+r'_rawpoints.geojson'
        r=os.path.join(folderpath, filename)
        with open(r, 'w', encoding='utf-8') as f:
            geojson.dump(geojson.FeatureCollection(featuresr), f,
sort_keys=True,
       ensure_ascii=False)

        print('json raw exported')
```

GUIDE.txt

Tool: ET2Spatial

This tool converts the screen coordinates of eye-tracking data recorded on a webmap, to real-world coordinates.

HOW TO USE:-

-Download the tool as a zip. After unzipping, double click the exe file. The tool takes maximum 1 minute to compile. The icon files should be in the same directory as the exe file

For Eye-tracking Data following convention should be followed:

-In the Raw gaze points: ['Time of Day [h:m:s:ms]','RecordingTime [ms]','Point of Regard Right X [px]','Point of Regard Right Y [px]','Participant','Gender','Age']

-In the Fixations gaze points following elements should exist: ['Event Start Trial Time [ms]','Event Duration [ms]','Fixation Position X [px]','Fixation Position Y [px]"Participant','Gender','Age']

-For the User Interaction file, The tool currently only works with MapTrack export (http://eyetracking.upol.cz/maptrack/results)

-On the initiation of the tool a terminal window also opens, The notifications for the export buttons are printed in this window.

SYSTEM SETUP:-

The tool is standalone and should not need additional installations. However in case of problems, these are the recommended steps:

1. Installation of Python v 3.5+ on the local system (https://www.python.org/downloads/release/python-387/), Make sure that you add python to PATH

2. Installation of the packages. In the requirements.txt all the necessary modules are listed, you can install them individually or through 'pip install -r requirements.txt' in shell.

EXPORTS:-

The exports are named after the participant number which was given in input files.

The tool exports 1) shapefile: Which will produce CSV and Shapefiles for both raw points and fixation points. The shapefiles are exported in the EPSG 3857 Projection System.

2) GeoJSON: Which will produce GeoJSON for both raw points and fixation points

SHAPEFILE:-

.shp, .shx, .dbf

The shapefile has the following contents:

--------Raw Points.shp

ind : index or serial number in temporal order

latitude: in decimal degrees

longitude: in decimal degrees

zoom level: the Google maps zoom factor at which each point was recorded

Format_RT: the time stamp in millisecond for each point

--------Fixation Points.shp

ind : index or serial number in temporal order

latitude: in decimal degrees

longitude: in decimal degrees

zoom level: the Google maps zoom factor at which each point was recorded

Sync_time: the time stamp in millisecond for each point

Duration: the amount of time of a fixation on screen

Along with the shapefile export there is a:

PROJECTION FILE:It contains the information on the projection of the exported points in WKT format

CSV FILES: all the above mentioned attributes of points in csv format.

GEOJSON:-

The geojson file contains a point geometry file with associated properties:

--------Raw Points.geojson

Geometry: latitude, longitude

Properties: zoom level, Format_RT

--------Fixation Points.geojson

Geometry: latitude, longitude

Properties: zoom level, Sync_time, Duration

GETTING DATA FROM MAPTRACK:

To use MapTrack data with Eye-tracker, it must be selected as a stimulus in presentation settings.

Enter the participant Id, Matching with the one entered in Eye-tracking system. Press F11 to enter fullscreen mode. Press F2 to start.

After the experiment is finished, Press F2 again to terminate the 'interaction mode'. The results can be accessed at: http://eyetracking.upol.cz/maptrack/results/

Download the xml file for the relevant participant ID and finished time.

LIST OF ATTACHMENTS

Bound Attachments

Attachment 1: Code Snippets of ET2Spatial.

Attachment 2: Guide text on how to run tool.

Free Attachments

Attachment 3: Poster

Attachment 4: USB Pen Drive/DVD

Structure of Pen Drive/DVD

Directories:

- Code
- ET2Spatial Tool
- Eye-tracking Project (SMI Experiment file & MapTrack files)
- Input Data (Data exports from SMI BeGaze & MapTrack for Task 1, 2, 3 & 4)
- Output Data & Analysis (File exports from ET2Spatial & ArcGIS/QGIS project files)
- Poster
- Text
- Website