

**Palacký University Olomouc, Faculty of Science,
Department of Geoinformatics**

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

AUTOMATION OF PROCESSING GNSS TRACK RECORDS FOR DESIGNING INTENSITY MAPS

Diploma thesis

Author

Bc. Benjamín ŠRAMO

Supervisor (Palacký University Olomouc)

Mgr. Radek BARVÍŘ, Ph.D.

Co-supervisor (Paris Lodron University Salzburg)

Prof. Bernd RESCH, Ph.D.

Erasmus Mundus Joint Master Degree Programme

Copernicus Master in Digital Earth

Specialization Track Geovisualization & Geocommunication

Olomouc, Czech republic, 2023



Palacký University
Olomouc



With the support of the
Erasmus+ Programme
of the European Union

ANNOTATION

Automated processing of GNSS track data provides opportunities for effective quantitative visualisation of mobility data. Presented processing tool and its settings for matching GNSS track records to a road network for visualizing passage frequency. Optimal tool parameterization and correctness rate are purposed for three case studies in urban and rural environments. The published work aims to help cartographers to effectively manipulate GNSS data. The functionality of the tool is demonstrated by designing web and paper intensity maps, i.e. graduated-colour and graduated-symbol maps.

KEYWORDS

mobility, GPX format, parametrisation, map matching, graduated-colour map, graduated-symbol map

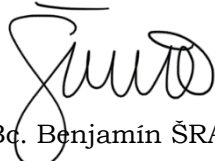
Number of pages: 45

Number of attachments: 9

This thesis has been composed by Benjamín Šramo for the Erasmus Mundus Joint Master's Degree Program in Copernicus Master in Digital Earth for the academic years 2021/2022 and 2022/2023 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.

In Olomouc, 23. 5. 2023


Bc. Benjamín ŠRAMO

ACKNOWLEDGEMENT

I would like to thank my supervisor Mgr. Radek Barvř, Ph.D., for finding an interesting topic for the thesis. I thank him for his sincere interest in actively supervising the thesis and improving its quality. I thank my fellow colleagues in Salzburg and Olomouc for their words of encouragement during my studies.

Last but not least, I thank God, my family, my girlfriend and my closest friends for supporting me in what I love to study, being there with me in my hardships and celebrating my successes, hopefully, another success is waiting around the corner.

ASSIGNMENT OF DIPLOMA THESIS

(project, art work, art performance)

Name and surname: Bc. Benjamín ŠRAMO
Personal number: R210705
Study programme: N0532A330010 Geoinformatics and Cartography
Work topic: Automation of Processing GNSS Track Records for Designing Intensity Maps
Assigning department: Department of Geoinformatics

Theses guidelines

The aim of the thesis is to find an automated way of processing GNSS track data into a linear layer suitable for multiple methods of quantitative visualization. A student will analyze various approaches and their settings for joining GNSS track records to a linear road network for visualizing passage frequency or other quantitative attributes. After examining the correctness rate of the process, the best workflow for inputting GPX files will be automated and published aiming to help cartographers with the effective processing of data. The functionality will be demonstrated by designing graduated-colour or graduated-symbol maps of the city of Olomouc or a similar area in either digital or analogue form.

The student will attach all created datasets and thesis outputs in digital form. The student will also 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.

Extent of work report: max 50 pages
Extent of graphics content: as needed
Form processing of diploma thesis: printed
Language of elaboration: English

Recommended resources:

- [1] Hashemi, M. (2017). A testbed for evaluating network construction algorithms from GPS traces. *Computers, Environment and Urban Systems*, 66, 96-109.
- [2] John, S., Hahmann, S., Rousell, A., Löwner, M. O., & Zipf, A. (2017). Deriving incline values for street networks from voluntarily collected GPS traces. *Cartography and Geographic Information Science*, 44(2), 152-169.
- [3] Jan, O., Horowitz, A. J., & Peng, Z. R. (2000). Using global positioning system data to understand variations in path choice. *Transportation Research Record*, 1725(1), 37-44.
- [4] Duncan, M. J., Badland, H. M., & Mummery, W. K. (2009). Applying GPS to enhance understanding of transport-related physical activity. *Journal of Science and Medicine in Sport*, 12(5), 549-556.
- [5] Ma, Q., & Kockelman, K. (2019). A low-cost GPS-data-enhanced approach for traffic network evaluations. *International Journal of Intelligent Transportation Systems Research*, 17(1), 9-17.

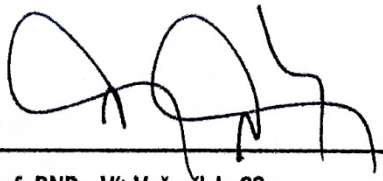
Supervisors of diploma thesis: Mgr. Radek Barvíř, Ph.D.
Department of Geoinformatics

Date of assignment of diploma thesis: **December 5, 2022**

Submission deadline of diploma thesis: **May 5, 2023**

L.S.

doc. RNDr. Martin Kubala, Ph.D.
Dean



prof. RNDr. Vít Voženilek, CSc.
Head of Department

Olomouc December 5, 2022

CONTENT

LIST OF ABBREVIATIONS.....	8
INTRODUCTION	9
1 OBJECTIVES.....	10
2 STATE OF ART.....	11
2.1 Global Navigation Satellite System	11
2.2 Map Matching.....	14
2.3 Geovisualization in Jupyter.....	16
3 METHODOLOGY.....	17
3.1 Study Area and Data.....	17
3.2 Automation Workflow.....	18
3.3 Jupyter Notebook	20
3.4 Python Packages.....	20
4 TOOL DEVELOPMENT	22
4.1 Design	22
4.1.1 Pre-Processing	24
4.1.2 Data Mining.....	26
4.1.3 Post-Processing.....	30
4.1.4 Outputs	31
4.2 Debugging and Exceptions.....	33
4.3 Documentation and Distribution.....	34
5 TESTING AND ASSESSMENT	36
5.1 Olomouc Case Study.....	36
5.2 Malá Fatra Case Study.....	38
5.3 Slovak Paradise Case Study	40
6 RESULTS	42
7 DISCUSSION	43
CONCLUSION	45
REFERENCES AND INFORMATION SOURCES	
ATTACHMENTS	

LIST OF ABBREVIATIONS

Abbreviation	Meaning
BSD	Berkeley Source Distribution
EO	Earth Observation
FOSS	Free and Open-Source Software
GeoJSON	Geographic JavaScript Object Notation
GI	Geographic Information
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPX	GPS Exchange Format
GUI	Graphical User Interface
HMM	Hidden Markov Model
HTML	Hypertext Markup Language
MIT	Massachusetts Institute of Technology
MM	Map matching
OSM	OpenStreetMap
RFID	Radio-frequency Identification
UX	User Experience
VGI	Volunteered Geographic Information
WLAN	Wireless Local Area Network

INTRODUCTION

Technologies for positioning services play a prominent role in infrastructure development and periodic “smart” monitoring. Nowadays policymakers oversee urban development through the lenses of smart city models. Raubal et al. (2017) consider the integration of GNSS data to be the game-changing aspect for increasing the sustainability of travel behaviour. Location-based sensors provide a network of data sources ready to analyse. The network may be rich in multi-temporal or multi-user layers. In this thesis, the focus is on the automated processing of trajectory data.

The trajectory data is a primary source for human mobility data mining (Crivellari et al., 2022). The multi-temporal layers of motion data are a solid base for further trajectory predictions (Chen & Liu & Yu, 2014), trajectory classification (Dabiri & Heaslip, 2018), motion flow modelling (Song et al., 2014) or activity recognition (Gao & Sun, 2012). Though Feng and Zhu (2016) explain that the first significant step is to store the vast-volume of data which can rapidly accumulate. Secondly, while integrating different data sources one needs to define a common metric resolution to compare data with different sampling frequencies. Thirdly, the spatio-temporal queries run a complex computation. Therefore, one must adapt appropriate techniques to perform such an analysis.

Map matching (MM) is a fundamental approach for GNSS data consolidation. The process assigns the position derived from the GNSS receiver to the target digital model of a street network utilizing polylines that approximate the edges of the network (Jensen & Tradišauskas, 2009). The MM approaches improve with the growing measuring precision of the receivers and differ in implementing deterministic or stochastic approaches. A stochastic method based on a Hidden Markov Model (HMM) with non-emitting states is used in the thesis. The model can deal with missing positions, and one can specify probability distributions of parameters related to signal reception (Meert & Verbeke, 2018).

The results of multi-temporal data mining are beneficial for individual citizens to inspect the spatial distribution of their movement behaviour over time (Feng & Zhu, 2016). Contrasting single-user data, multi-user data provide more objective information about citizen mobility (Ávila Callau et al., 2020). Such information has the potential to develop infrastructure sustainably.

Any GNSS data referred to a road network can be quantified and visualized in maps of intensity. Not only individual citizens but also individuals responsible for the local development of a particular mode of transport or recreational activity may benefit from the visual outcomes. Therefore, the thesis results of the automated processing of GNSS data is distributed in an open, user-friendly, self-explanatory environment and integrates with free and open-source solutions.

1 OBJECTIVES

The aim of the thesis is to find an automated way of processing GNSS track data into a linear georeferenced layer suitable for methods of quantitative visualization. The presented approach and its settings for matching GNSS track records to a road network for visualizing passage frequency. Examining the correctness rate of the process supports the best workflow for integrating GPX files for automated data processing. The work is published aiming to help cartographers with the effective manipulation of mobility data for quantitative visualisation. The functionality will be demonstrated by designing graduated-colour or graduated-symbol maps. The methodological goals for the thesis study are following:

- (1) automation of spatial analyses of GNSS trajectory data on a street network,
- (2) implementation of possible appropriate quantitative geovisualization,
- (3) assessment of results (1) and (2) and possibilities for further use.

2 STATE OF ART

This chapter outlines the relevant research frontiers concerning the thesis objectives. The first area is the global perspective on GNSS and its impact on large-scale spatial studies. Secondly, the development of map matching for feasible path approximations is discussed. Finally, the consecutive relationship of data mining and dynamic geovisualization is emphasized by the described Python packages.

2.1 Global Navigation Satellite System

Global Navigation Satellite System (GNSS) can be attributed to two conceptual systems, namely Radio Navigation Satellite Services (RNSS) representing the technical architecture (EU Agency for Space Programme, 2022) and Location-Based Services (LBS) representing the application infrastructure (Huang & Gao, 2018).

RNSS infrastructure provides satellite signals for positioning through GNSS (American *GPS*, Russian *GLONASS*, European *Galileo*, Chinese *BeiDou*, etc.), regional constellations (Japanese *QZSS*, Indian *NavIC*, etc.) and through Satellite-Based Augmentation Systems (SBAS) for accuracy improvement (*WAAS*, *EGNOS*, *MSAS*, etc.) (EU Agency for Space Programme, 2022). Figure 1 illustrates the GNSS constellations in their prescribed orbits, the majority at an altitude of approximately 20 000 km (Borealis Precision, 2023).

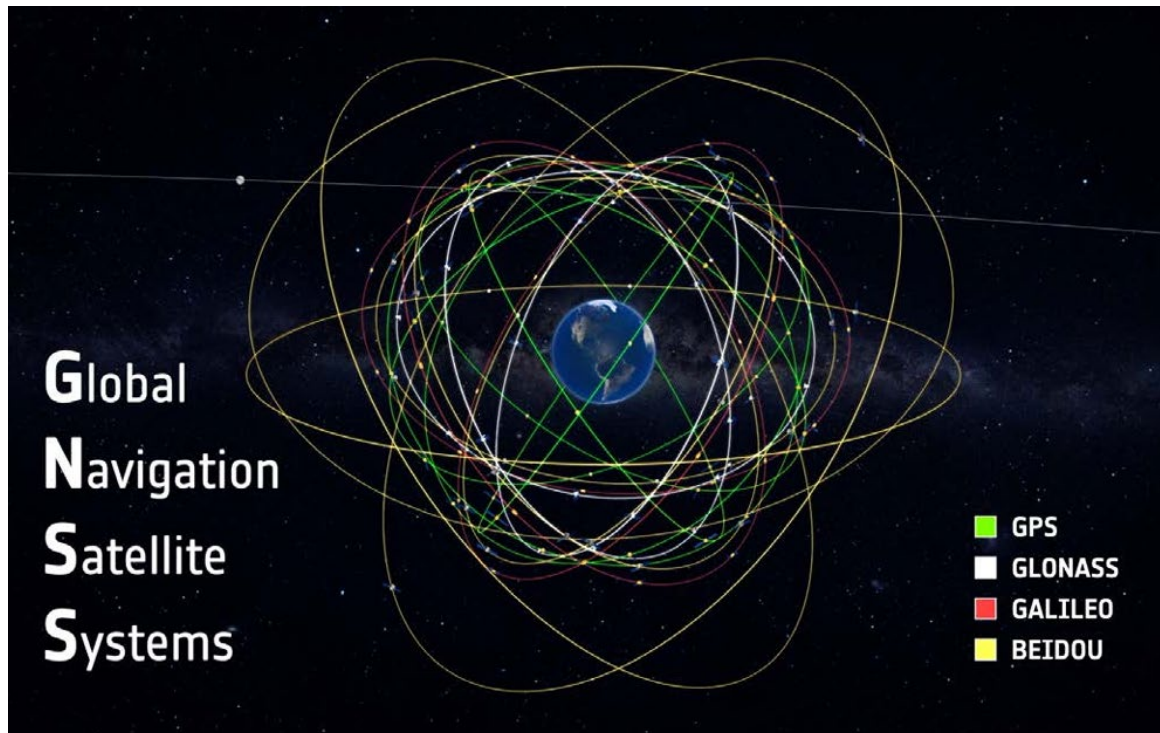


Figure 1: Satellite constellations of global GNSS, ©ESA.

GALILEO AND EGNOS

Galileo is the first GNSS that unlike other GNSS is not under military control (fig. 2). The European player provides European citizens independence and sovereignty from other global powers. The system provides global navigation, positioning, and timing information together with several high-performance services. This GNSS has been in operation since 2016 providing its services fully, freely and openly world-wide. Most positioning devices work with multiple GNSS constellations; thus, *Galileo* provides seamless interoperability with *GPS*, *GLONASS* and *BeiDou*. Such GNSS compatibility grants accurate operational domain. User applications can rely on the positioning data like never before and generate value for the economy and improve the quality of life (EU Agency for Space Programme, 2022).

EGNOS, European Geostationary Navigation Overlay Service, is not a global positioning service, but rather it is the augmentation service to improve the accuracy and reliability of the satellite positioning services in Europe. *EGNOS* has been fully operational since 2009. and openly serves the European public especially through safety of life service in civil aviation, maritime, rail and road transportation (EU Agency for Space Programme, 2022).



Figure 2: Logo of Galileo GNSS, ©ESA.

LOCATION-BASED SERVICES

LBS is a mobile application which interconnects through communication networks providing a service and content based on positioning and spatial modelling (Huang & Gao, 2018). In most cases, these user application services harness the current location of a user (Huang & Gao, 2018). However, location history may also serve as a data source for case-specific studies, such as mobility behaviour (Jonietz & Bucher, 2018). Huang and Gao (2018) differentiate LBS based on outdoor positioning – GNSS, cellular networks and Wi-Fi, and indoor – WLAN, Bluetooth, RFID. The services have a wide range of application from sport tracking, marketing, emergency management, healthcare, entertainment etc. (Huang & Gao, 2018). Furthermore, specific examples of LBS for mobility are traffic simulation and travel forecasting (Jan et al., 2000), as well as changes of mobility behaviour for sustainability (Jonietz & Bucher, 2018).

GNSS is the key component for positioning and its further services (fig. 3). The EU Agency for the Space Programme (2022) forecasted that road navigation together with customer solutions enabling LBS will dominate 90 % of the GNSS market for the period 2021–2031. Furthermore, it is estimated that the revenues from the added-value services utilising GNSS will account for 72 % of total GNSS revenues in 2031. Therefore, position-based solutions have a wide potential for expansion, especially in the segments of Road & Automotive and Customer Solutions, Tourism & Health (EU Agency for Space Programme, 2022).

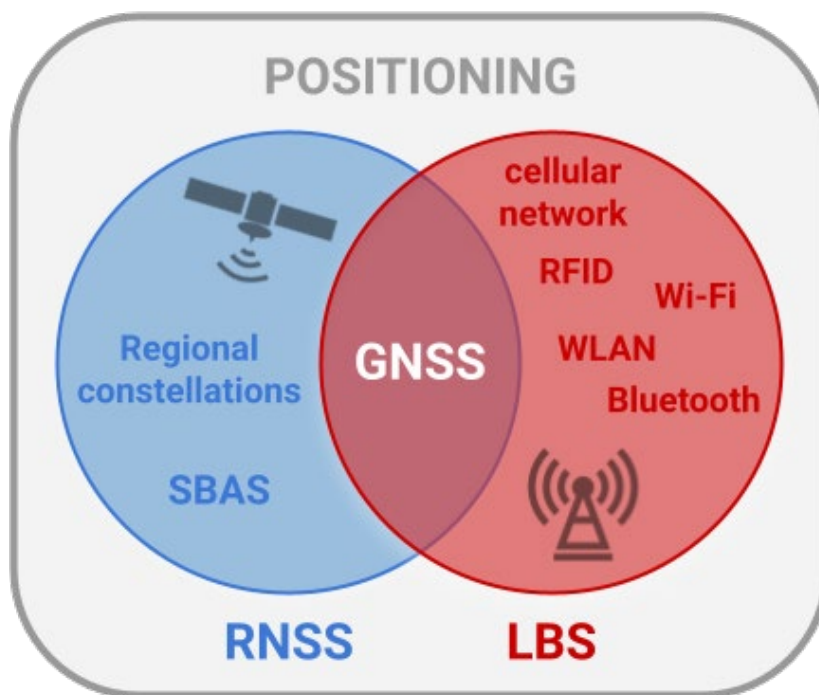


Figure 3: Components responsible for location recording – positioning.

POSITIONING ACCURACY

Nowadays, the utilizing of outdoor GNSS location data is affected by the quality of the signal coming from the satellites. Naturally, there are common geometric errors such as ionospheric and atmospheric delays, these can be corrected by the SBAS. Particularly in urbanised areas are common multipath effect errors or generally different delusions of precision due to the cold start problem (Àvila Callau et al., 2020).

Moreover, Àvila Callau et al. (2020) call attention to the user-made distortions in the case of VGI on citizen mobility. The study focuses on the detection and filtering of GNSS trace outliers whose geometric characteristics differ significantly from the characteristics of other coordinates that compose the complete trace. In conclusion, Àvila Callau et al. (2020) state that filtering allows the scientist to divide the users according to their level of reliability and principally the calculated error rate may indicate the quality of the VGI source. Another method to handle the errors and gaps in records due to various reasons is map matching (Jonietz & Bucher, 2018). Although, the performance of MM is highly dependent on its calculation technique and parametrization.

2.2 Map Matching

Map Matching is a process that assigns the coordinates of a trajectory to the target digital model of a street network utilizing polylines that approximate the edges of the network (Jensen & Tradišauskas, 2009). According to Feng and Zhu (2016), the trajectory of a moving object expressed by a tuple of geographic coordinates is called a geographical trajectory or a raw trajectory. In contrast, a semantic trajectory is a sequence of meaningful places with a geotagged location. The street network is a directed graph comprising nodes: road junctions and road ends, and edges: directed road segments (Feng & Zhu, 2016). The polylines also called routes or matched paths are sequences of edges in the street network where consecutive edges share a vertex (Feng & Zhu, 2016).

In general, there are two types of MM: online and offline (Jensen & Tradišauskas, 2009). In the case of online MM, the location on the street network is calculated in real-time, for instance in navigation systems. Whereas the offline MM works with static historic or modelled datasets. The methods and parametrisation are tailored for the use case, thus resulting in a better matching result. Such offline MM scenarios could be for insurance or road pricing in pay-per-use pricing (Jensen & Tradišauskas, 2009).

The challenge for the MM algorithm and heuristics is the trade-off between the possible roads by the location measurements and the feasibility of the path (Newson & Krumm, 2009). Simple matching to the nearest road produces unrealistic solutions as displayed in figure 4. Therefore, sophisticated algorithms employ the sequence of measurements to ensure not only path proximity to the measurement, but also path continuity, direction similarity (White et al., 2000), topology (Greenfeld, 2002). However, the geometric methods mentioned here are still not satisfactory for noisy data in the dense street network of an urbanised area.

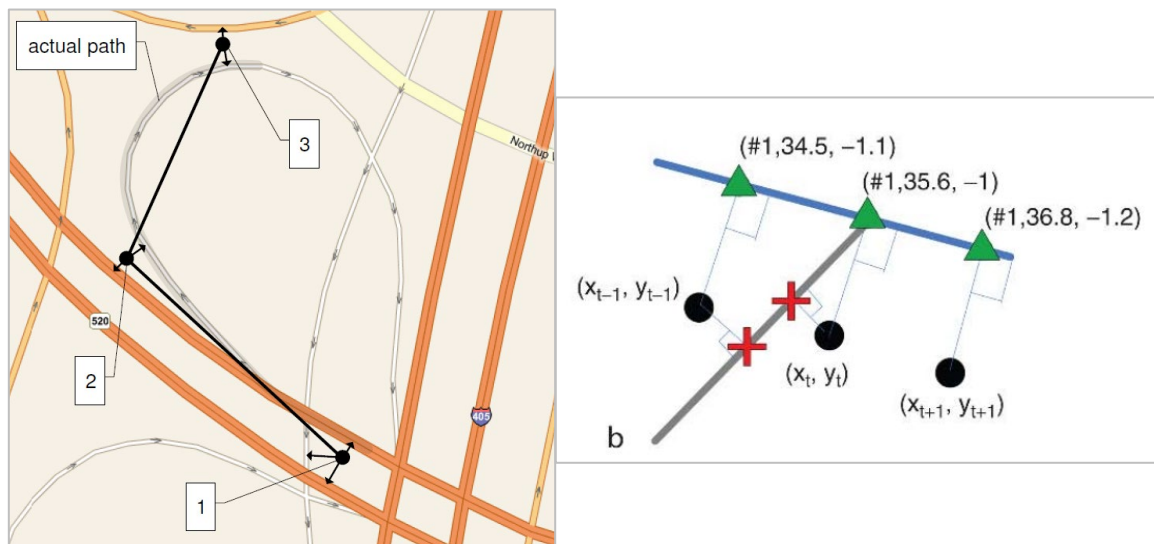


Figure 4: Map matching of measured locations (black dots) to the road network. Merely matching to the nearest road is prone to make mistakes (left: Newson & Krumm, 2009; right: Jensen & Tradišauskas, 2009).

Newson & Krumm (2009) propose a method to deal with multiple possible path hypotheses through simultaneous probabilistic modelling – Hidden Markov Models (HMM). The models can approximate the possible states and assign them probabilities. The states are represented in the street segments and the transitions between the states are governed by the connectivity of the street network (Newson & Krumm, 2009). The MM algorithm in this thesis tool follows the HMM theory to ensure a higher performance in various environments. The authors of the algorithm refer to non-emitting states challenge, segments with missing measurement positions, that can be resolved by the HMM (Meert & Verbeke, 2018).

Map Matching is the critical part of query processing in the trajectory data mining framework (fig. 5). The framework explains the phases for handling trajectory data, e.g. GPX records, to create a value-added product to understand the data holistically. Feng and Zhu (2016) describe the workflow as follows, firstly, in pre-processing phase one aims to improve the quality of the raw data through calibration, sampling, cleaning, segmentation etc. The following data management step handles the volume of data in an efficient and scalable manner. Query processing retrieves new appropriate results such as a new map-matched path from a GPX dataset. Lastly, the trajectory data mining tasks summarize knowledge in spatial analyses. Through all these phases one needs to protect the privacy of sensitive information represented in the data (Feng & Zhu, 2016).

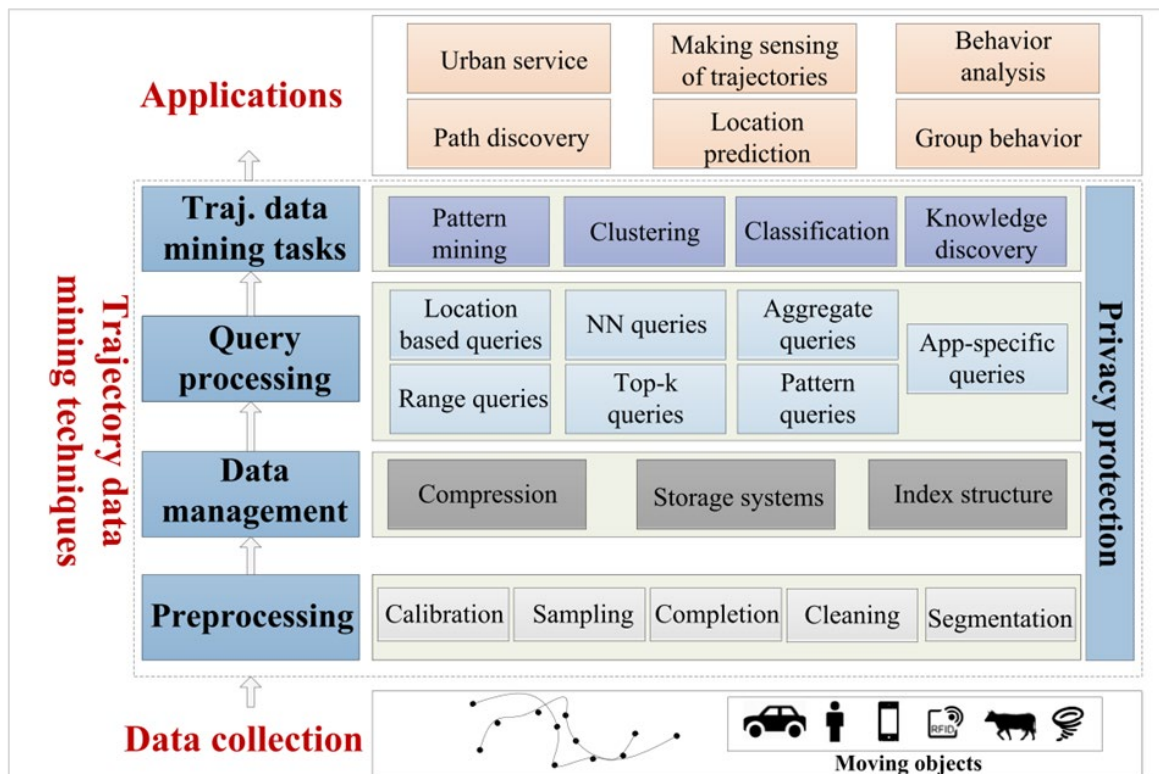


Figure 5: Framework of trajectory data mining (Feng & Zhu, 2016).

2.3 Geovisualization in Jupyter

The interactive aspect of *Jupyter Notebooks* is developed through the implementation of widgets. These elements enable bidirectional communication between the front-end and backend, the widgets link the capabilities of JavaScript mapping libraries to Jupyter (Corlay & Renou, 2019). The interactive geovisualization of the results close the circle of GIS functionality in *Jupyter*.

The *Leafmap* package (Wu, 2021) for interactive visualization is built upon FOSS packages i.e., *ipyleaflet* responsible for interactive mapping (Renou et al., 2021), *folium* displays data enabling *LeafletJS* extensions (Filipe et al., 2021), *WhiteboxTools* for geoprocessing toolbox (Lindsay, 2018) and *ipywidgets* responsible for map interactivity (Grout et al., 2021). Therefore, the added value of the *Leafmap* package is the integration of the most popular visualization packages together in a manner that requires minimal coding skills in the *Jupyter Notebook* (Wu, 2021). The tool developed for the thesis implements the *foliummap* module of the *Leafmap* package (Wu, 2021). The package displays the preliminary result in map preview and facilitates the decisions for changes in parametrization; or allows the final web map to be saved as a HTML file for the simple presentation of the results.

3 METHODOLOGY

The potential for GNSS track records is vast. However, the data mining of positioning data creates a set of case-specific problems. Geospatial awareness and knowledge are the solid base for a robust and actionable solution. The following subchapters describe the principles and methods of how the objectives of the thesis have been accomplished.

3.1 Study Area and Data

Settlements dispose of an increasing number of mobility data provided by sensors, especially when dealing with citizen volunteered geographic information – VGI (Callau et al., 2020). Therefore, the purpose and potential applications of the automation of mobility data are in the sphere of infrastructure development.

The automation tool aims to work in any type of study area. Primarily the area is determined by the input data. The tool has been tested in three different environments (Chapter 5). Firstly, an urbanised area, Olomouc – a city in the eastern Czech Republic. The city has the sixth largest population in Czechia. However, its cultural heritage, quality higher education and well-known events attract tourists and students from abroad to explore the city's riches. The area is characterised by a dense road infrastructure. The second case study area is situated in the mountain range Malá Fatra, Slovakia. The area covers a variety of landscapes – from rural settlements to mountainous environments. Most of the region is located in the national park, so the region is rich in natural beauty and is a popular site for hiking expeditions. The road network density is diverse across the region, but it is rather sparse. The third case study area is the National Park Slovak Paradise. The area is known for its rugged terrain, gorges, and cliffs. Together with the dense vegetation the area is an interesting testing example due to the high GNSS signal distortions. The area network density is very low overall. All three study areas are known for their potential to attract visitors to explore the region, therefore, the data-backed planning is vital for their sustainable infrastructure development.

The data provider is Mgr. Radek Barvíř, Ph. D., the supervisor of the thesis. The positioning data track mobility, in particular running, cycling and hiking. The data is supposed to represent VGI, therefore, the data source is diverse in the accuracy rates and undocumented. The GNSS position has been received by the wearable devices or a smartphone application listed below:

- Garmin Forerunner 255 (Galileo-enabled device),
- Garmin Forerunner 35 Optic,
- Lenovo K10 Note Dual SIM via application *Strava*,
- Lenovo A6 Note Dual SIM via application *Strava* (Galileo-enabled device),
- Lenovo A7000 Black via application *Strava*.

Strava application is a mobile tracking application for over 30 types of activities. Its key feature is to collect and store positioning data of an activity. The *Strava* community has over 100 million sport enthusiasts in 195 countries (Strava, 2023).

The provided data are in GPX format, which is an open standard defined in XML schema. The de-facto XML standard version 1.1 has stayed here since 2004 (Foster, n.d.). Generally, the measurements are recorded every 1–5 seconds, i.e. a very frequent sampling rate. Therefore, the automation workflow offers an option for appropriate generalization and compression of the datasets (Attachment 3).

3.2 Automation Workflow

The tool for automation of GPX data to the linear layer of passage frequencies consists of the following parts: parametrization and data acquisition, map matching, post-processing, result visualization and storage. In figure 6 is a general overview of the automation workflow showing preliminary processing steps resulting in two main outputs – web map and geodata.

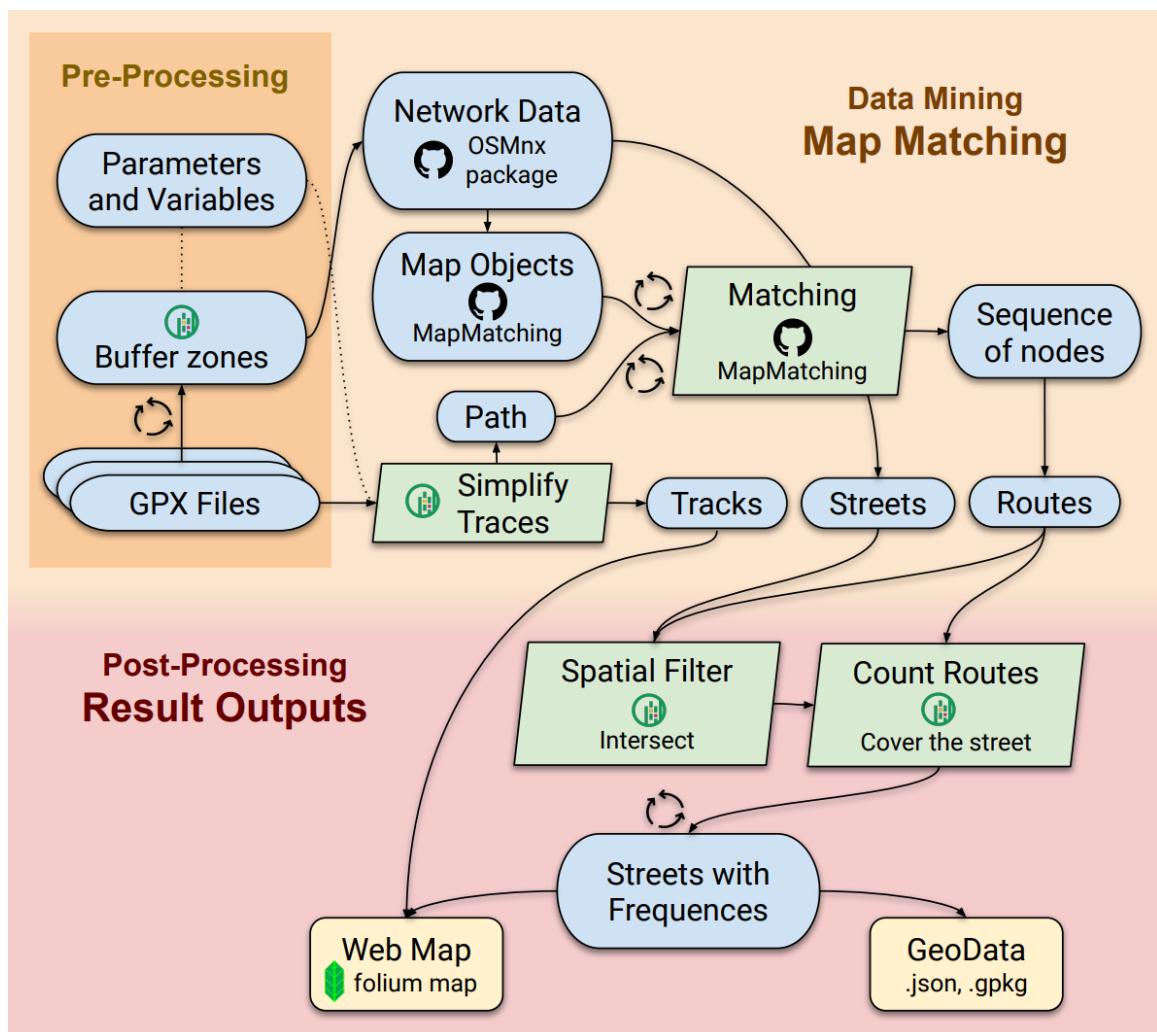


Figure 6: Workflow diagram and Python packages within the workflow.

The parametrization and data upload parts are the only parts expecting a direct user interaction of the workflow. The user sets 11 parameters that have a direct impact on the final product (fig. 7). The parameters consist of 2 directories for data input and output, a parameter for street network download, 9 map matching parameters such as level of generalization, GNSS measurement noise, thresholds etc. The detailed explanation of parametrization is explained in [Chapter 4.1.1](#). The data upload refers to the link of GPX data either to the local file or to the given sample data web repository. Access to the set of GPX files is mandatory for the automation process to execute.

```
1  ## Set the directory location for data and output files
2  DATA_FOLDER = 'data' # path to your data repository
3  OUTPUT_FOLDER = 'output' # directory save the results
4
5  ## Study area for OpenStreetMap Street Network Download [1]
6  BUFFER_DIST = 500 # area around the GPX records; in meters
7
8  ## Map Matching parameters. [2]
9  TOLERANCE = 5 # threshold for GPX track simplification; in meters
10 MAX_DIST = 100 # break for zero measurement (match) probability; in meters
11 OBS_NOISE = 20 # expected GNSS noise, in meters
12 OBS_NOISE_NE = 60 # expected GNSS noise for non-emitting states (the value should be larger than OBS
13 DIST_NOISE = 5 # difference between distance between matched route and distance between tracks, in m
14 DIST_NOISE_NE = 15 # difference between distances for non-emitting states, in meters
15 MAX_LATTICE_WIDTH = 7 # search the route with this number of possible paths at every step
16 INCREASE_MAX_LATTICE_WIDTH = 2 # if no solution is found, the width can be incremented by the value
17 MIN_PROB_NORM = 0.005 # eliminate the lattice where it drops below normalized probability
```

Figure 7: View on the parameters and variables in Google Colab.

The core part of the data mining is the map matching. MM follows the structure of a Python package responsible for matching the GNSS measurements to the street network. The matching is enriched with the road network download (OSMnx) and track generalization. The data mining starts with the download of the street network from OSM, the extent is defined by the input data and the parameter for buffering. The MM works on the probabilistic calculation of possible paths, therefore, the parametrization is crucial for the result. The output of the analysis is a sequence of passed nodes on the street network. The nodes construct matched routes to the network. In the same step, the traces of mobility are concatenated into so-called tracks that serve as a reference to the matched result. For more details see [Chapter 4.1.2](#).

Post-processing has two main goals for automation. Firstly, the variables are transformed into data structures ready for data geospatial visualization and storage. Secondly, the matched routes are overlaid with a blank street network on which are calculated frequencies i.e., the number of matched passages through the street, for more details see [Chapter 4.1.3](#).

The output map visualization serves the user as a preview of the result ([Chapter 4.1.4](#)). The asset supports the user with a simple visualization in the form of an interactive web map. Based on the visualization one can readjust the parametrization that would fit the characteristics of the study area and the input data. The outputs can be saved or downloaded as two types of web maps in HTML format or as GIS-ready data in GeoJSON, GeoPackage formats. More sophisticated and quality maps can be made in GIS by integrating the output geodata ([Attachment 4–6](#)).

3.3 Jupyter Notebook

Since 2014 Jupyter has been growing to a popular spin-off project of *IPython*, Interactive Python. FOSS principles ensure web computing independent of any (programming) language. The *Jupyter Notebook* creates a web application allowing users an easy manipulation with the code. In the notebook one can combine snippets of computer code, computational outputs, and explanatory texts. “For data scientists, *Jupyter* has emerged as a de-facto standard” (Perkel, 2018, p. 145).

Granger and Peréz (2021) present how *Jupyter* pioneered the data science with conception of thinking and storytelling with code and data. The shift towards interactive computation runs the code with “a human in the loop”. The human may modify the code snippets and see the effects immediately. Thus, the execution is no longer working in linear fashion, but rather the user is empowered to explore the calculation more deeply and experimentally (Perkel, 2018).

Secondly, Granger and Peréz (2021) emphasize the notebook's ability to talk in the universal language – storytelling. Such computational narrative leads the audience to understanding and effective learning. *Jupyter Notebooks* bring options to add textual descriptions, markdown with LaTeX support, multi-media content, etc.

The nature of the notebook supports the sharing of identical development environments. The dissemination of the integrated computing environment increases the further usability across target groups. This benefit is supported with notebooks that run in the cloud (Perkel, 2018). Such cloud collaboration is becoming popular across different platforms e.g., *Google Colab*, *Binder*, *Code Ocean*.

Google Colab is a freemium tool created by *Google Research* (Google, 2023). The environment is fully cloud-based, i.e. local hardware independent, cloud back-up, cross-platform. Moreover, it allows multi-user access to the notebook (Sherrer, 2022). Therefore, the Python code developed for the thesis has been integrated in the *Google Colab* environment.

3.4 Python Packages

Python packages promote modularity in programming. The packages consist of modules ordered in hierarchical structure. The idea behind is to break down the large tasks into small manageable modules.

Firstly, the reusability of packages improves the functionality of a wide range of programs. Moreover, the code appears simple and organised, the package helps avoid collisions between objects through separate namespaces. Therefore, it makes coding, maintenance and debugging easier (Sturtz, 2022). Considering the open ability to create and access free and open-source repositories, such as *GitHub*, the distribution and popularity of Python packages is growing (fig. 8).

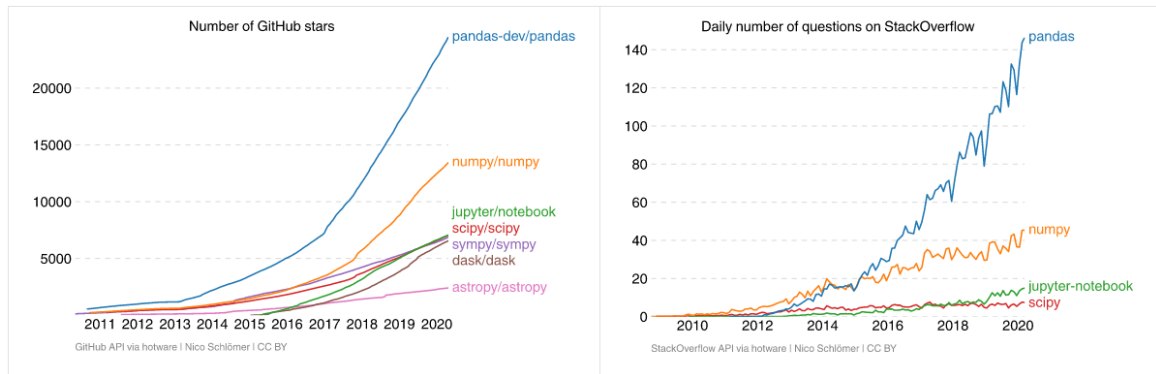


Figure 8: Growing popularity of the “Scientific Python” (Schlömer, 2020).

The central package for the automation is *Leuven.MapMatching* package (Meert, 2021a). It has been developed by Belgian scientists from *KU Leuven*, department of *Declarative Languages and Artificial Intelligence* and the lab *Sirris – Elucidata Group*. The package is deployed under an open Apache licence, Version 2.0. The added functionality is responsible for the alignment of a GNSS trace to a map or road segments (Meert, 2021a).

The second package enables the automation to work freely in any part of the world where street networks are mapped by *OpenStreetMap* contributors. *OSMNx* facilitates the download and basic data management steps for the street network of custom choice (Boeing, 2017). The author of the package is Geoff Being, the professor of urban planning and spatial analysis at University of South Carolina, and the distribution is under MIT licence.

The major package for data processing is *GeoPandas* (Jordahl et al., 2020). It is a geospatial processing package following *Pandas* structure for possible non-spatial interoperability. BSD licence distributes the package functionality freely and openly (Jordahl et al., 2020).

The last important package is responsible for data geovisualization in the *Jupyter* environment. It is the *Leafmap* package (Wu, 2021), which is built on *folium* and *ipyleaflet* packages substantial for interactive map creation. Its key advantage is the capability to tie functionalities of different packages together with minimal coding requirements on the user (Wu, 2021). However, the initiative is novel and needs better user support for all the functionality integration.

4 TOOL DEVELOPMENT

The first objective of the thesis is to develop a tool for automated processing of GNSS track records. Consolidation of GNSS records adds value for further understanding of the data especially about the mobility of individuals. The whole development started with the review of current approaches dealing with the issue ([Chapter 2](#)). As different approaches were identified, the designing process of the tool's functionality started ([Chapter 4.1](#)). Iteratively optimizations have been implemented to fit the methods to the overall intent. Debugging and raising the exceptions support the smooth execution of the code ([Chapter 4.2](#)). Lastly, the documentation and distribution are discussed in [Chapter 4.3](#).

4.1 Design


In the Methodology ([Chapter 3](#)), one finds a general introduction to the framework of the thesis. The Design chapter refers to the specifics of the tool development in *Jupyter Notebook*, Python 3 kernel. The vital step of the design process is the examination of the appropriate Python packages, which would support functionality between each other regardless of the different version compatibility.

The final code has undergone many modifications to design the best possible solution. For instance, in the beginning, the code worked with a geolocation parameter to define the study extent. Later, the tool calculates the study area by itself from the input data and the buffer parameter. Therefore, the upgrade brought the need to enable computation with multiple study areas at once ([Chapter 4.1.2](#)). The second modification, the author of the map-matching package was requested to add additional functionality. The function deals with the cases where a road network is missing ([Chapter 4.1.2](#)). Furthermore, the tool and output testing uncovered hidden computational errors in frequency calculation ([Chapter 4.1.3](#)). Additionally, different erroneous datasets have been tested to ensure the proper raising of exceptions ([Chapter 4.2](#)). Minor additional adjustments had to be implemented due to the release of new package versions.

In the following sub-chapters, the code is explained in detail. The tool is distributed through two main channels: the standalone *Jupyter Notebook* and the *Voilà* web application (QuantStack, 2019) running on the *Jupyter Notebook*. Therefore, the subtle differences of code design are outlined.

In the notebook tool, the code is written in cells, code snippets, that are superimposed by the explanatory text. The code cells can be run sequentially or how the user desires. During the processing the user is informed with the well-designed preliminary messages. The advantage of the environment is that it provides the open-source code, the user can modify the code as desired. The notebook is structured into

sections therefore one can navigate through the table of contents. In figure 9 the contents and structure of the notebook can be inspected.



AUTOMATION OF PROCESSING GNSS TRACK RECORDS FOR DESIGNING INTENSITY MAPS
Install and Import Dependences
1. Pre-Processing
Set Variables and Parameters
From GPX Files to Study Area
2. Data Mining: Map Matching
A. Import OSM Street Network
B. Parametrised Map Matching
3. Post-processing
2D Array into GeoDataFrame
Calculation of Frequencies
Visualization
Save the Results

Figure 9: Preview of the tool contents in Google Colab.

In case of a notebook enabled for *Voilà*, the core functionality resembles the one in the notebook version. However, significant changes in the design of the code had to be implemented, see the structure and its interaction flows in figure 10. The code loads at the start of the application, although, the functions execute only after clicking on “run button”. The processing steps are enclosed in functions that are called as the user clicks on the button. All input and output variables can be accessed in user-friendly widgets i.e., buttons, sliders, etc. The user does not interact with the code itself, rather with the GUI through the Markdown content and the widgets. The preliminary and final results are being shown through the *output* widget.

Limitations arise in the application customization beyond the tool parametrisation provided in the widgets. *Voilà* installs and imports the only packages defined inside the configuration file, the code is not modifiable from the GUI. For such special customization needs, the standalone *Jupyter Notebook* is a more suitable environment.

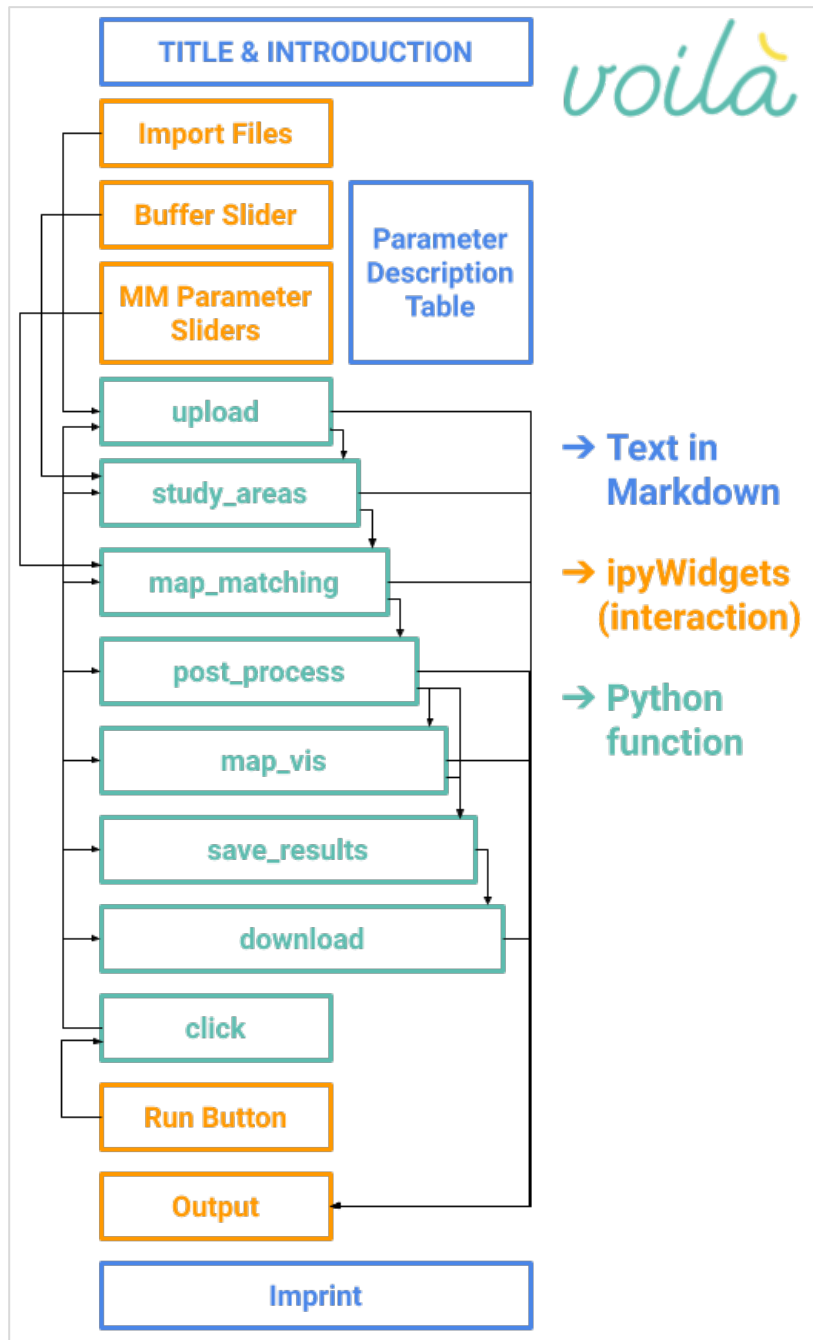


Figure 10: Structure and interaction-flow design of Jupyter for Voilà.

4.1.1 Pre-Processing

Before any pre-processing steps, all dependencies should be installed and imported into *Jupyter*. The dependencies are specified in the header of the notebook or in the configuration file.

The pre-processing section is divided into two parts. The first part deals with the declaration of variables and parameterisation, see the following code snippet. The user is required to specify the location of the data repository containing the GPX files and the output folder for the results. The notebook also offers the download of ready-to-use example GPX files. The parameterisation deals with the study area extent

[line 6] and the MM essentials [lines 9–17]. These numerical variables have their default values, nevertheless, it is essential to adapt the parameter values to the study area and the input track records. Chapter 5 deals with the testing and optimal parameterisation proposal.

The map-matching algorithm uses the *KU Leuven* solution (Meert & Verbeke, 2018), which can handle low positioning accuracy and sparse sampling frequencies considering the HMM (Chapter 2.2). The matching is an approximation; therefore, the algorithm requires several probabilistic parameters determining the transition probability. The parameters are described as follows. *Minimum normalised probability of observations* eliminates the lattice where the probability falls below a certain confidence level [line 10]. *Maximum distance* from the track determines a break from any further matching [line 11]. *Maximum lattice width* restricts the lattice in search for a suitable route [line 12] and the *increase of the lattice width* increments the width if no solution is found [line 13]. *Observation noise* is the standard deviation of expected GNSS noise [line 14], the *noise for non-emitting states* expects a higher noise level [line 15]. *Distance noise* for emitting and non-emitting states is a standard deviation of the difference between the distance between states and the distance between observations (Meert, 2021b). In line 9 of the code snippet, the user can set the *tolerance* in meters for record simplification (Douglas & Peucker, 1973). The simplification flattens out the differences in the spatio-temporal precision of the records.

```
1  ## Set the directory location for data and output files
2  DATA_FOLDER = 'data' # path to your data repository
3  OUTPUT_FOLDER = 'output' # directory save the results
4
5  ## Study area for OpenStreetMap street network download
6  BUFFER_DIST = 100 # area around the GPX records; in meters
7
8  ## Map Matching parameters
9  TOLERANCE = 4 # threshold for GPX track simplification; in meters
10 MIN_PROB_NORM = 0.002 # eliminate the lattice where it drops below normalized probability
11 MAX_DIST = 100 # break for any further matching; in meters
12 MAX_LATTICE_WIDTH = 7 # search the route with this number of possible paths at every step
13 INCREASE_MAX_LATTICE_WIDTH = 4 # if no solution found, the width increments by the value
14 OBS_NOISE = 20 # expected GNSS noise, in meters
15 OBS_NOISE_NE = 60 # expected GNSS noise for non-emitting states , in meters
16 DIST_NOISE = 5 # difference between distance between matched route and distance [...]
17 DIST_NOISE_NE = 15 # difference between distances for non-emitting states, in meters
```

Secondly, the next part of the pre-processing calculates the appropriate study areas based on the input data, the GPX files, and the *buffer distance* variable. The following code snippet iterates over the files in the data repository. From the valid GPX files, the script extracts the points with their position record [line 11]. The metric buffer distance is converted to the degrees in line 21 and the buffer areas are calculated [line 22]. The *areas* polygons are the basis for further analysis i.e., downloading the OSM road network.

Inside the code, one can find exceptions for unexpected errors. The script can skip over an invalid GPX file. The execution will fail with an exception in the case of empty data and in the case of the data being located in the polar regions i.e., above 80° or below -80° of latitude. The code snippet ends with an output statement about the number of study areas ready for analysis.

```

1 gpx_combined = gpd.GeoDataFrame(columns = ['latitude', 'longitude'])
2 for filename in listdir(DATA_FOLDER):
3     if filename.endswith(".gpx"):
4         try:
5             gpx_file = Converter(input_file = DATA_FOLDER + "/" + filename).gpx_to_dataframe()
6         except:
7             print(filename, "is NOT valid, the tool removed the file from repository.")
8             remove(join(DATA_FOLDER, filename))
9             continue
10
11     gpx_point = gpd.GeoDataFrame(gpx_file, geometry=gpd.points_from_xy(gpx_file.longitude,
12     gpx_file.latitude)).set_crs('epsg:4326')
13     gpx_combined = pd.concat([gpx_combined, gpx_point])
14
15 if gpx_combined.empty:
16     raise Exception("Uploaded data is empty.")
17
18 if (gpx_combined["latitude"] >= 80).any() or (gpx_combined["latitude"] <= -80).any():
19     raise Exception("Execution failed, uploaded data contains in polar region.")
20
21 latitude = gpx_combined["latitude"].iloc[0]
22 distance_deg = BUFFER_DIST / (111319.488 * np.cos(np.radians(latitude)))
23 gpx_buffer = gpx_combined["geometry"].buffer(distance_deg).set_crs(4326)
24 areas = gpx_buffer.unary_union
25 areas = gpd.GeoDataFrame(index=[0], crs='epsg:4326', geometry=[areas])
26 if areas.type[0] == "MultiPolygon":
27     areas = areas.explode()
28
29 print("From the GPX files have been calculated", len(areas.index), "area(s) for graph(s).")

```

4.1.2 Data Mining

The core of the whole automation is in the data-mining part. It brings together the data from the pre-processing stage and calculates the most likely routes from the track records. The tool can work with multiple study areas at once therefore the whole datamining-computation is designed in two nested cycles. The first cycle iterates over the *areas* and downloads the *OpenStreetMap* road network for the area. The second cycle iterates over the GPX files within the study area and computes the routes on the graph – the road network.

In the first code snippet represented below, one can see the declaration for the expected variables: *street_all* [line 1] is a container for all road networks used in the analysis, *track_df* [line 2] collects the coordinates of the tracks before matching, *route_df* [line 3] is the store for all matched routes on the road network. In the first cycle, the OSM road network is downloaded based on a network type and study area. The network type is pre-selected network by the mode of transport. If the selection is insufficient, one can create a custom filter based on the “highway” attribute of a road. In case the network is not present in the area, the code skips the area with an exception [line 14–16]. Finally, the script creates the in-memory representation of a map through adding graph nodes and graph edges to *InMemMap* object – *map_con* [lines 22–26]. The structure of the object represents the connectivity of the graph to suggest the most possible transitions (Meert, 2021b).

```

1 street_all = gpd.GeoDataFrame(columns = ['Latitude', 'Longitude'])
2 track_df = pd.DataFrame(columns = ['Latitude', 'Longitude', 'id'])
3 route_df = pd.DataFrame(columns = ['Latitude', 'Longitude', 'id'])
4 id = 1
5 for _, row in areas.iterrows():
6     ## A. Download graph based on network type
7     area_gdf = gpd.GeoDataFrame(index=[0], crs='epsg:4326', geometry=[row["geometry"]])
8     print("Building a graph for a specific area...")
9
10    try:
11        ### NETWORK TYPES {"all_private", "all", "bike", "drive", "drive_service", "walk"}
12        ### or TYPE YOUR OPTIONAL CUSTOM "HIGHWAY" FILTER HERE
13        graph = ox.graph_from_polygon(row["geometry"], network_type = 'all', simplify = False)
14    except:
15        print("WARNING: One area has no street network mapped.")
16        continue
17
18    print("===", graph, "\n") # number of nodes and edges
19    street_lines = ox.graph_to_gdfs(graph, nodes = False)
20    street_all = pd.concat([street_lines, street_all])
21    ### Map Object
22    map_con = InMemMap("road_network",
23                      use_latlon = True,
24                      use_rtree = True,
25                      index_edges = True)
26    for node in graph.nodes:
27        lat = graph.nodes[node]['y']
28        lon = graph.nodes[node]['x']
29        map_con.add_node(node, (lat, lon))
30
31    for edge in graph.edges:
32        node_a, node_b = edge[0], edge[1]
33        map_con.add_edge(node_a, node_b)
34        map_con.add_edge(node_b, node_a)
35
36    map_con.purge() # remove nodes without location or edges
37

```

The second nested cycle prepares the GPX data for map matching and runs the matching for each file in the study area. In the first part of the cycle, the track positions are connected into a line trace, which is subsequently simplified based on the *TOLERANCE* parameter [lines 61–62]. A *path* variable is a list of coordinate-pair tuples [line 70] that feeds the map-matching algorithm. A *track* variable is a two-dimensional list of coordinate-pairs [line 71], the concatenation of all tracks, *track_df*, serves as the reference layer for the matched routes.

```

38 ## B. Map Matching
39 for filename in listdir(DATA_FOLDER):
40     if not filename.endswith(".gpx"):
41         continue
42
43     gpx_df = Converter(input_file = DATA_FOLDER + "/" + filename).gpx_to_dataframe()
44     try:
45         gpx_point = gpd.GeoDataFrame(gpx_df, geometry = gpd.points_from_xy(gpx_df.longitude,
46                                 gpx_df.latitude)).set_crs('epsg:4326')
47     except:
48         print("WARNING: The file has no record of position, the matching stopped.")
49         continue
50     gpx_point['id'] = 1
51     area_check = gpx_point.within(area_gdf)

```

```

51     if not area_check.iloc[0]:
52         continue # skip the GPX file outside of the graph
53     print("Map matching of " + filename + " started...")
54     try:
55         gpx_line = gpx_point.groupby(['id']) ['geometry'].apply(lambda x: LineString(x.tolist()))
56     except:
57         print("WARNING: The file has only one record of position, the matching stopped.")
58         continue
59
60     line_gdf = gpd.GeoDataFrame(gpx_line, geometry = 'geometry').set_crs('epsg:4326')
61     tolerance_deg = TOLERANCE / (111319.488 * np.cos(np.radians(latitude)))
62     line_gdf['geometry'] = line_gdf['geometry'].simplify(tolerance_deg) # simplify
63     gpx_coords = line_gdf.apply(lambda row: list((row.geometry).coords), axis=1)
64     passage = list(gpx_coords[1])
65     track = []
66     path = []
67     for i in range(len(passage)):
68         lat = passage[i][1]
69         lon = passage[i][0]
70         path.append((lat, lon))
71         track.append([lat, lon])
72
73     track = np.array(track)
74     df = pd.DataFrame(track, columns = ['Latitude', 'Longitude'])
75     df['id'] = id # id for grouping into one line
76     track_df = pd.concat([track_df, df])

```

The matching algorithm itself is executed in lines 77–96. The *DistanceMatcher* [line 77–87] differs from a simple matcher in that it takes into account the similarity of the distances between the track observations and the matched states (Meert, 2021b). Apart from the parameters defined at the beginning, the *matcher* is defined by: the network graph – *map_con*, possible non-emitting states restriction – *restrained_ne*. The *match* method matches the path to the graph with node repetition allowed [line 89]. If the matcher fails to match the entire path, the early stop index is returned (Meert, 2021b). Therefore, in case of an interrupted matching, a new matching is performed, allowing jumps to a nearby edge [lines 90–94]. The reasons for the possible mismatch can be on the side of the road-network data or due to an inappropriate MM parameterisation. Finally, the matching returns a list of OSM node IDs through which the matched route passes [line 96].

```

77     matcher = DistanceMatcher(map_con,
78                             max_dist = MAX_DIST,
79                             min_prob_norm = MIN_PROB_NORM,
80                             max_lattice_width = MAX_LATTICE_WIDTH,
81                             increase_max_lattice_width = INCREASE_MAX_LATTICE_WIDTH,
82                             obs_noise = OBS_NOISE,
83                             obs_noise_ne = OBS_NOISE_NE,
84                             dist_noise = DIST_NOISE,
85                             dist_noise_ne = DIST_NOISE_NE,
86                             non_emitting_edgeid = False,
87                             restrained_ne = False)
88
89     matcher.match(path, unique = False) # retain not only unique nodes in the sequence
90     if matcher.early_stop_idx is not None:
91         print("WARNING: Parts of the path were omitted from matching due to the road mismatch.")
92         from_matches = matcher.best_last_matches(k = MAX_LATTICE_WIDTH)
93         matcher.continue_with_distance(from_matches=from_matches, max_dist = MAX_DIST)
94         matcher.match(path, expand = True)
95
96     node_id = matcher.path_pred_olynodes_withjumps # node_ids the route passes through
97

```

Originally, the map-matching package could not jump over areas with no road network, i.e. with incomplete data or typically with no path to be mapped. The matching would be interrupted with an early stop and the rest of the track left unmatched. Figure 11 illustrates the forced interruption due to the topologically incomplete data. Collaboration with the developer community is essential, the author of the package was able to add functionality that allows matching to continue with a given distance. Essentially, the lines 90–94 integrate the newly added functionality to fix the bug in the calculation. Moreover, the interaction resulted in both-sided code promotion.

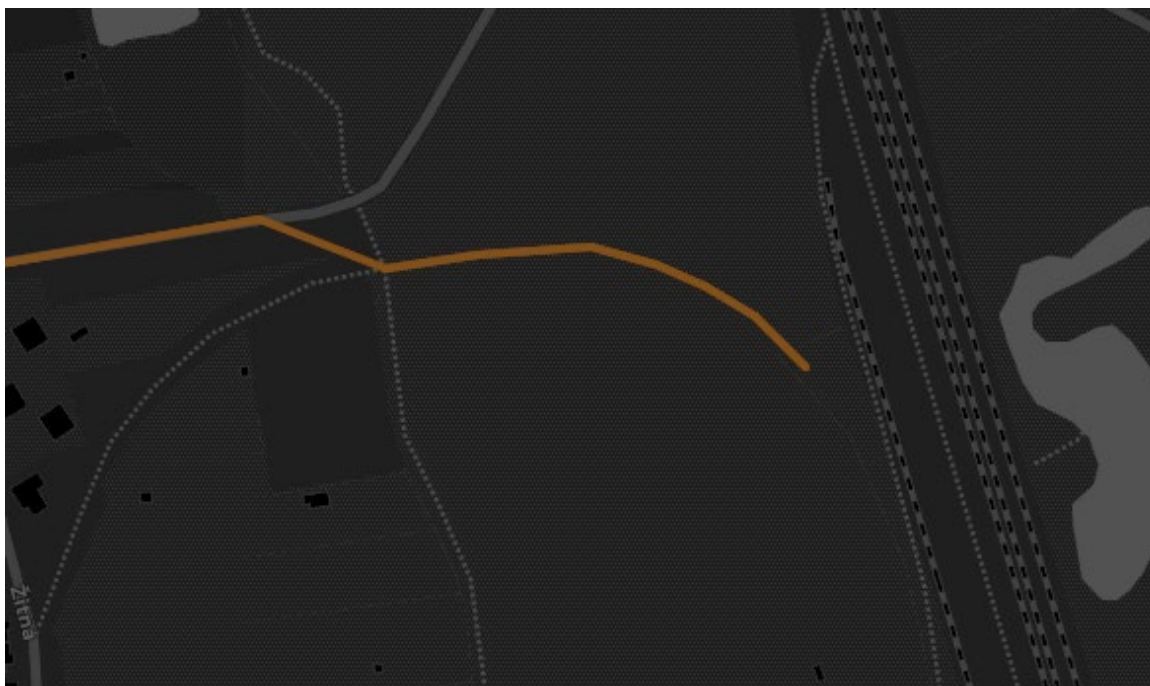


Figure 11: Interrupted track matching in an area with no road network (Šramo, 2023).

The end of the code snippet is devoted to the processing of the preliminary result, the sequence of node IDs, to create a set of matched roads [lines 98–116]. Based on the ID the code can access to the geometry of the nodes and retrieve their coordinates. Each road of a route is a connection of two consecutive nodes of the graph. These individual roads are concatenated to form a route – *route_df*. At the end of the whole computation, two variables are deleted [line 123]. This step unloads the cache memory and speeds up the further analyses in the post-processing phase.

```

98     id_route = 1
99     for i in range(len(node_id)-1):
100         route_node = []
101         lat = graph.nodes[node_id[i]]['y']
102         lon = graph.nodes[node_id[i]]['x']
103         latlon = [lat, lon]
104         route_node.append(latlon)
105         lat2 = graph.nodes[node_id[i + 1]]['y']
106         lon2 = graph.nodes[node_id[i + 1]]['x']
107         latlon2 = [lat2, lon2]
108         route_node.append(latlon2)
109
110     route_node = np.array(route_node)
111     df = pd.DataFrame(route_node, columns = ['Latitude', 'Longitude'])
112     df['id'] = id_route # the same id for one line (street)

```

```

113     route_df = pd.concat([route_df, df])
114     id_route += 1
115
116     id += 1
117
118     print("Matching of " + filename + " finished successfully.\n")
119
120 if route_df.empty:
121     raise Exception("The map matching has no result.")
122
123 del graph, map_con

```

The user is notified with the execution proceedings regularly. In the case of an exception [line 121] or warnings [lines 15, 47, 57, 91], the necessary information is provided. Details about the debugging process and exceptions for the tool are discussed in [Chapter 4.2](#).

4.1.3 Post-Processing

The last processing phase deals with two main goals. Firstly, the lines 2–7 convert the *track_df* and *route_df* variables into a geometry-enabled structure – *GeoDataFrame*. Secondly, the calculation of frequencies is carried out on the streets. The reference street network is cleaned from unnecessary attributes [line 9]. Then it is spatially filtered where the network intersects only the matched routes [line 10], the redundant duplicates are pruned from the *street_freq* [line 11]. A proper tool output check uncovered a redundancy issue in the OSM data. After these important steps the code can loop over the streets to calculate route frequencies on them [lines 14–16]. On each street the algorithm calculates the number of routes that are identical in the geometry attribute with the street – *covers* method (GeoPandas developers, 2022). The load of the calculation increases exponentially with the size of the study area. After the cycle the frequencies are added as a new attribute to the result variable *street_freq*. The variable is *GeoDataFrame* with the following attributes: *osmid*, *length*, *frequency* and *geometry*.

```

1 # Arrays of tracks and matched routes to GeoDataFrame
2 track_point = gpd.GeoDataFrame(track_df, geometry = gpd.points_from_xy(
3     track_df.Longitude, track_df.Latitude))
4 track_lines = track_point.groupby(['id'])['geometry'].apply(lambda x: LineString(x.tolist()))
5 tracks_gdf = gpd.GeoDataFrame(track_lines, geometry = 'geometry').set_crs('epsg:4326')
6 route_point = gpd.GeoDataFrame(route_df, geometry = gpd.points_from_xy(route_df.Longitude,
7     route_df.Latitude))
8 route_line = route_point.groupby(['id'])['geometry'].apply(lambda x: LineString(x.tolist()))
9 route_gdf = gpd.GeoDataFrame(route_line, geometry = 'geometry').set_crs('epsg:4326')
10 # Calculation of Frequencies
11 street_all = street_all.loc[:, ['osmid', 'length', 'geometry']]
12 street_freq = street_all.overlay(route_gdf, how = 'intersection') # drop streets out of the routes
13 street_freq = street_freq.drop_duplicates(subset=['osmid', 'length'])
14 frequency = []
15 print("Calculating passage frequencies on streets...")
16 for _, row in street_freq.iterrows():
17     bool_series = route_gdf.covers(row["geometry"])
18     frequency.append(bool_series.values.sum())
19
20 street_freq["frequency"] = frequency
21 print("The length of matched roads is", round(street_freq["length"].sum()), "meters.")

```

4.1.4 Outputs

The output preview dynamically presents the results to the user in web maps. There is no need to export the data first, but it supports the idea of allowing the user to interactively modify the results as needed within the *Jupyter* environment. By the repetitive optimisation of the parameters, the user gains the knowledge of how the algorithm works in a particular area. In the case of such a parameter-dependent tool, the output preview is a crucial element for practical use.

The final output, streets with frequencies, is displayed in a map preview using the *Leafmap* Python package (Wu, 2021). The map preview consists of functionality such as map zoom, full-screen view, geolocation, distance and area measuring tool, link to the tool documentation and the layer switcher (fig. 12 & 13). The interface allows the user to interact with two linear layers: the reference layer of GNSS track records [lines 6–9] and the road passages, which are styled in graduated colours [lines 10–21]. The symbology is designed for the light (fig. 12) and the dark basemaps (fig. 13).

```
1 m_light = leafmap.Map(width = "100%",
2                       height = "480",
3                       draw_control = False,
4                       attribution_control = True,
5                       tiles = "CartoDB positron")
6 m_light.add_gdf(tracks_gdf,
7                layer_name = "GPX tracks",
8                info_mode = None,
9                style = {'color':'blue', 'weight':0.5, 'opacity': 0.5})
10 m_light.add_data(street_freq,
11                 "frequency",
12                 cmap = "Wistia", # color ramp
13                 scheme = "Quantiles", # classification scheme
14                 k = 5, # max. number of classes (saved in attribute)
15                 add_legend = True,
16                 legend_title = "Number of passages",
17                 legend_position = "bottomright",
18                 layer_name = "road passages",
19                 style_function = lambda feat: {"color": feat["properties"]["color"],
20                                             "weight": 4,
21                                             'opacity': 0.9})
22 m_light.add_text("INTENSITY OF MOBILITY ON ROAD NETWORK", fontsize = 22, fontcolor='#404040',
23                 bold = True, padding = '0px', background = True, bg_color = 'white',
24                 border_radius = '5px', position = 'topright')
25 m_light.add_text("<a href = 'https://github.com/bsramo144/Thesis-Jupyter' target = '_blank'>
26                 <img width = '250' alt = 'Asset 3logo' src = 'https://user-
27                 images.githubusercontent.com/47752920/234973760-c8157fdd-a3cf-43cf-88b0-
28                 4dc8096cfe7c.png'></a>", background = False, position = 'topright')
```

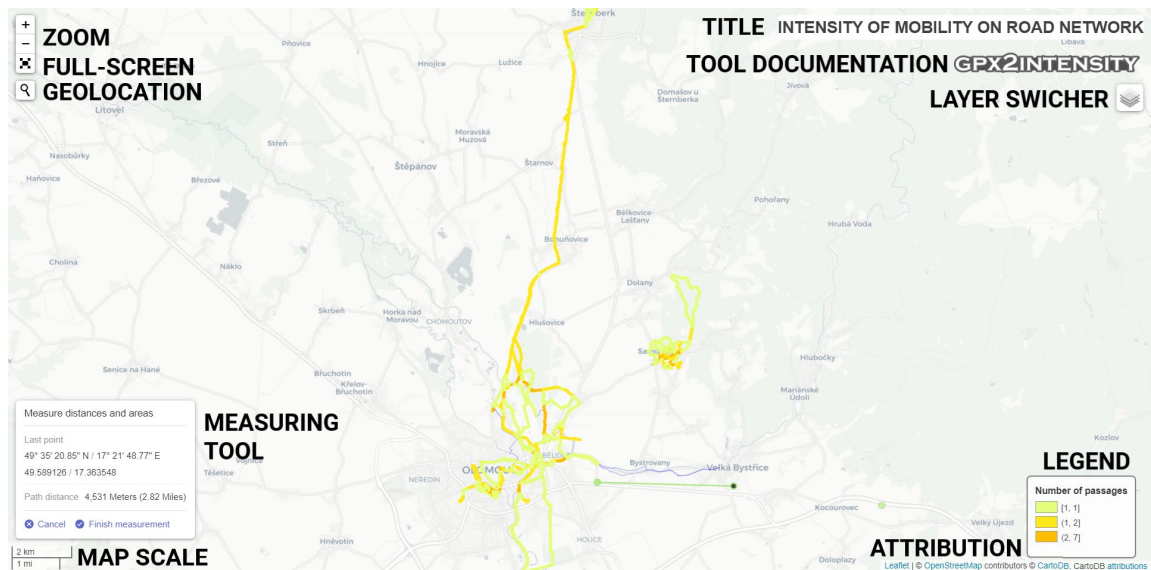


Figure 12: Light version of output preview with a map layout description.

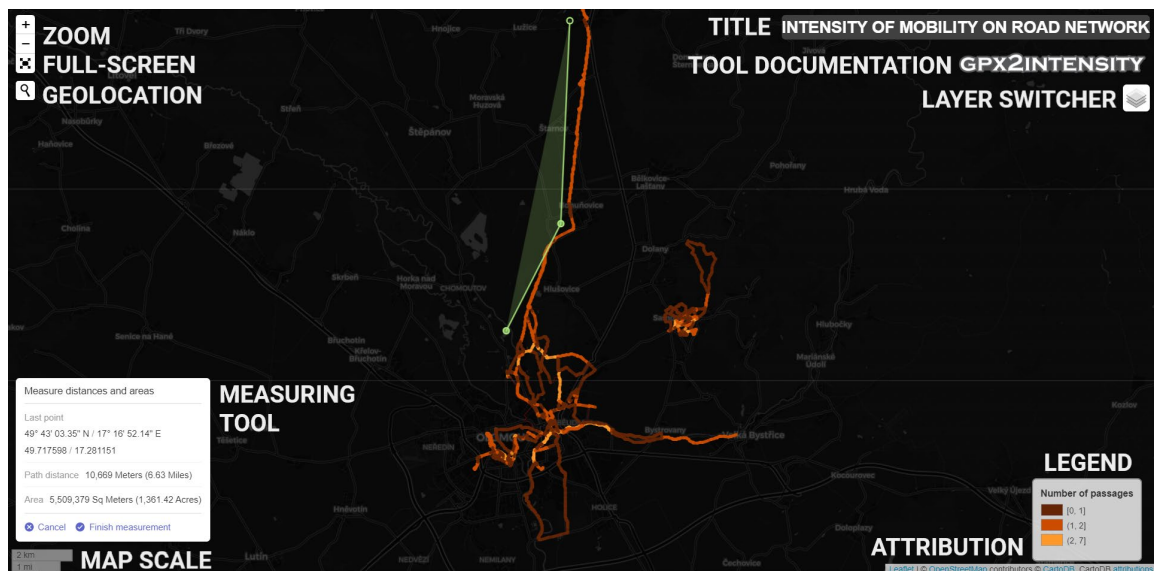


Figure 13: Dark version of output preview with a map layout description.

At the end of the iterative optimisation process, the user can save the output data or download it from the cloud. The script allows saving the linear data in two GIS-ready formats, GeoJSON and GeoPackage [lines 2, 3]. Secondly, the user can choose to download the light or dark version of the *Leaflet* map in HTML [lines 5, 6]. These web maps serve as a simple data preview for users with no further experience in GIS and cartography.

```

1 # Save the final linear layer with frequencies to GeoJSON and GeoPackage file
2 street_freq.to_file(OUTPUT_FOLDER + "/lines_freq.json", driver="GeoJSON")
3 street_freq.to_file(OUTPUT_FOLDER + "/lines_freq.gpkg", driver="GPKG")
4 # Save the interactive map to html
5 m_light.to_html(OUTPUT_FOLDER + "/light_map.html")
6 m_dark.to_html(OUTPUT_FOLDER + "/dark_map.html")

```


See Attachments 4–6 for examples of how the cartographic output of the data can be designed in linear graduated-symbol maps. The visualisation follows the frequency classification previously stored in the attributes in the tool. The class value in the attribute greatly simplifies the symbolization process in GIS. The background imagery for the print maps is the Sentinel-2 cloud-free composite by *EOX IT Services GmbH* (2020). Sentinel-2 satellites are part of European EO programme *Copernicus*. In the case of Olomouc, a different high-resolution imagery is used for the city mapping. The author designed the maps in *ArcGIS Pro 3.0.2* and the layout was finalised in *Adobe Illustrator 27.5*.

4.2 Debugging and Exceptions

According to Heusser (2022), debugging involves several steps which include identifying the problem, isolating its source, and finally, either fixing the issue or finding a workaround. The final step is to test the solution to ensure that it effectively resolves the problem. These bugs occur due to coding errors or when the program is not maintained properly, a collision of new package versions may cause the code to fail.

The notebook's advantage is that it runs the code snippets independently; therefore, the debugging is easier compared to one block of script. Nevertheless, in some cases, the use of debugging modules or packages is inevitable. The tool has been debugged in the most naïve way – logging. Logging is an intuitive way to track the source of the error through *print* statements. No installation is required, but the frequent prints may appear spammy (Trablesi, 2020).

The tool has been tested with different erroneous datasets to ensure the proper raising of exceptions. The error test GPX files were the following: files with only one or no point location, a data file within a polar region and a GPX file with an incorrect XML schema. For each case, there are designed suitable ways either to continue running the code or stop the execution by raising an exception, see the examples in [Chapter 4.1](#).

4.3 Documentation and Distribution

Documentation and distribution are the most important aspects for the further use of the tool. The overall presentation has to communicate the tool's capabilities. The level of detail in documentation varies based on the platform and the target user group. The most basic commentaries are in the *Voilà* web application interface, one can proceed to detailed documentation through a hyperlink. The *GitHub* start page introduces the workflow and the solutions in more general terms. The *Jupyter Notebook* together with the thesis provides the most detailed documentation and instructions for experienced users.

Each type of distribution is an attachment to the thesis. The solutions are tied together with the name “*gpx2intensity*” and its logo (fig. 14). The tool is built on open-licensed packages, so the entire use of the tool follows the FOSS principles. *GitHub* is a repository for the scripts and is a portal to subordinate applications like *Google Colab* and *Voilà*. The tool code solutions are presented in three *Jupyter Notebooks*:

1. *gpx2intensity_tool.ipynb* tool aimed at users with coding skills ([Attachment 1](#)),
2. *gpx2intensity.ipynb* – modification of the notebook for the *Voilà* application ([Attachment 2](#)),
3. *gpx_compression.py* is a Python script for GPX file compression ([Attachment 3](#)).



Figure 14: Logo of the tool “*gpx2intensity*” (source: author).

The web application runs on a free cloud server – *Binder*. It is a way to share reproducible and interactive environments, for example, *JupyterHub*, from online repositories (Ragan-Kelley et al., 2018). The scripting difference between the tool and the tool modification for *Voilà* is explained at the beginning of [Chapter 4.1](#). The structure subordinates to the application UX. The user runs the whole sequence of functions with one click on the “Run Button” ([fig. 10](#)). Download buttons support user-friendly interaction with the outputs of the tool.

Figure 15 is the preview of the start page of the web application. The tool requires the upload of the local GPX files, for instance, the test data from the *GitHub* repository ([Attachment 9](#)). The default values of parameters can be changed. After clicking on the “Run the Tool” button the preliminary messages are printed. Finally, the runtime and the results are displayed with the options to download the result data or the web maps. In the imprint section, the user can find further links to the documentation or licencing.

GPX2INTENSITY

TITLE & INTRO

AUTOMATION OF PROCESSING GPX TRACK RECORDS FOR DESIGNING INTENSITY MAPS

The tool matches GNSS track records to a road network and calculates frequencies on it. The map matching runs on probabilistic model handling states with missing observation i.e., non-emitting states. For more information about the map-matching algorithm see [Leuven.MapMatching](#) documentation. For more details regarding the tool see the [GitHub](#) documentation.

Set Parameters

Import Files

Note: maximum upload size is 10 MB.

Parameter	Description
Buffer	expand the study area from the GPX records; in meters
Tolerance	GPX track simplification threshold; in meters
Min. Proba.	stop matching below normalized probability
Max. Distance	break for zero match probability; in meters
Max Lattice	search the route with the number of possible paths at every step
Increase Latt.	if no solution is found, increase the lattice by the value
Obs. Noise	standard deviation of measurement noise, in meters
Obs. Noise NE	standard deviation of measurement noise for non-emitting states, in meter (the value should be larger than Obs. Noise)
Dist. Noise	difference between distance between matched route and distance between tracks, in meters
Dist. Noise Ne	difference between the distances for non-emitting states, in meters (the value should be larger than Dist. Noise)

Parameter Description Table

▼ Environment Parameters

Buffer

100

Tolerance

4

Min Proba.

0.002

▼ Thresholding of Matching

Max. Distance

100

Max Lattice

7

Increase Latt.

5

▼ Measurement Noise

Obs. Noise

24

Obs.Noise NE

50

Dist. Noise

5

Dist.Noise NE

10

MM Parameter Sliders

Run Button

The tool is Attachment 2 to Master Thesis.

**COPERNICUS MASTER
IN DIGITAL EARTH**

With the support of the
Erasmus+ Programme
of the European Union

Imprint

License: CC BY 4.0

Benjamin Šramo (2023)
Palacky University in Olomouc, University of Salzburg

Figure 15: gpx2intensity web-application start page with a layout annotation.

5 TESTING AND ASSESSMENT

The possibilities of cloud-based tools are wide. Therefore, it is essential to determine the scope and limitations of the tool for possible use cases. The testing and assessment part consists of three case studies: in urban ([Chapter 5.1](#)), mixed rural and mountainous ([Chapter 5.2](#)), and rugged natural ([Chapter 5.3](#)) environments. Each case study aims to find suitable parameterisation for the area. The parameters are set to critical break values to determine overfitting or underfitting by visual comparison of the reference tracks with the new matched routes in the output web map. Secondly, the best representative output result has been assessed in the frequency correctness rate. Reasonings and advice about further applications are suggested in the following sub-chapters.

5.1 Olomouc Case Study

As mentioned earlier in [Chapter 3.1](#), Olomouc is a Czech city with an interesting cultural heritage pulling visitors to come (fig. 16). Well-maintained infrastructure will make the life experience for the locals and visitors more enjoyable and make it more attractive for new ones to arrive. The case study aims to create an exemplary intensity map of sample mobility data provided by Mgr. Radek Barvíř, Ph. D.

The positioning data have been processed through the web application “gpx2intensity” multiple times to find the most suitable parametrisation. The street network of the area is very dense in built-up areas, it creates larger possibilities for routing. Therefore, it is important to keep the detail of the positioning and the optimal track simplification tolerance set to 2 metres (tab. 1). Table 1 presents the list of testing runs, for each run a parameter is altered to find the optimal solution. Three pairs of parameters are altered altogether. The first pair is the maximum lattice width with its increase because they are mutually correlated by design. The same reason for pairing the parameters is for the observation noise and its non-emitting-state alternative, and for the distance noise and its non-emitting-state alternative. The most optimal fitting result is in the ninth testing run with the execution time of 293 seconds and 107,2 km of matched routes (tab. 1). From 165,2 km of input tracks, the routes with frequencies decreased the length by 58 km.

In the second part of the assessment, 20 control points have been identified from the expert perspective to evaluate the frequency precision. The location of the control points can be inspected in the final map presentation ([Attachment 4](#)). At each control point the reference value is compared to the result of the tool calculation. Control point 4 shows incorrect frequency due to the wrong route choice of the map matching. In contrast, the wrong value at point 10 is due to the incorrect choice of roads on one street. The overall precision at the control points is 90 %. The sensitivity rate of 93,9 % indicates the number of passages assigned correctly (tab. 1).

Table 1: Parameterization matrix of test runs in Olomouc and frequency precision assessment. The parameters are set to critical break values to determine overfitting/underfitting by comparing the reference tracks with the new matched routes in the interactive web map. The optimal solution is run number 9. It has returned 90 % of correct measurements at control points, the sensitivity of the correct matching is at 93,9 % based on the control samples.

Test Run	PARAMETRIZATION MATRIX										RESULTS		FREQUENCY PRECISION		
	BUFFER_D IST	TOLERANCE	MIN_PROB_ NORM	MAX_ DIST	MAX_LATTICE_ WIDTH	INCREASE_MAX_ LATTICE_WIDTH	OBS_N OISE	OBS_NOISE_ NE	DIST_N OISE	DIST_NOISE_ NE	Time [s]	Length [m]	Control Point	Reference Value	Result from Calculation
1	100	2	0,002	100	7	5	16	30	3	10	223	104 639	1	2	2
2	100	2	0,002	100	7	5	16	30	5	16	252	105 236	2	2	2
3	100	2	0,002	100	7	5	16	30	7	20	245	105 553	3	0	0
4	140	2	0,002	100	7	5	16	30	5	16	244	105 421	4	2	1
5	180	2	0,002	100	7	5	16	30	5	16	285	105 278	5	3	3
6	140	0	0,002	100	7	5	16	30	5	16	405	101 387	6	6	6
7	140	4	0,002	100	7	5	16	30	5	16	434	104 876	7	2	2
8	140	2	0,002	140	7	5	16	30	5	16	326	107 243	8	2	2
9	140	2	0,002	120	7	5	16	30	5	16	293	107 243	9	2	2
10	140	2	0,002	120	7	5	12	30	5	16	404	107 673	10	3	1
11	140	2	0,002	120	7	5	20	40	5	16	235	107 250	11	6	6
12	140	2	0,002	120	5	4	16	30	5	16	209	108 894	12	3	3
13	140	2	0,002	120	9	6	16	30	5	16	257	107 017	13	2	2
14	140	2	0,001	120	7	5	16	30	5	16	237	107 243	14	3	3
15	140	2	0,003	120	7	5	16	30	5	16	234	107 243	15	1	1
16													16	1	1
17													17	3	3
18													18	2	2
19													19	2	2
20													20	2	2

90 % measurements at control points are correct
93,9 % of passages have been assigned correctly (sensitivity)

overfittingoptimalunderfitting



Figure 16: City of Olomouc in November (Fidler, 2013).

5.2 Malá Fatra Case Study

Malá Fatra is a mountain range that is part of the Western Carpathians in Slovakia (fig. 17). The study area extends to the rural areas of Terchová in the north and Strečno in the west. Therefore, the road network is not dense in most cases, however, the settlements increase the diversity of the network. The highest elevation point of the records is at Veľký Kriváň – 1 709 m above sea level, the lowest is at Nezbudská Lúčka 356 m above sea level. Another factor influencing the GNSS signal quality is the dense forest vegetation. Therefore, generally the positioning above the tree line provides records with a low delusion of precision, thus better map-matching results.

In table 2, the optimal parameterisation for the best result fits the test run number 9. As the study area is more distributed compared to the previous study, hence the time of tool computation increased rapidly to 862 seconds. The total length of the matched routes with frequencies is 132,9 km, which is 66,4 km less than the input single track records.

The precision assessment has used 15 control points expertly sampled on critical spots. See the distribution of the points in the map, which is [Attachment 5](#) to the thesis. All incorrect frequencies captured in the control points are due to the low positioning precision. In case of the control point 4, the spatial delusion is too large to match the graph. On the other hand, control point 14 is overestimated due to the noisy, oscillating track record. Many other smaller oscillating records have been smoothed by the simplification function – the *TOLERANCE* parameter.

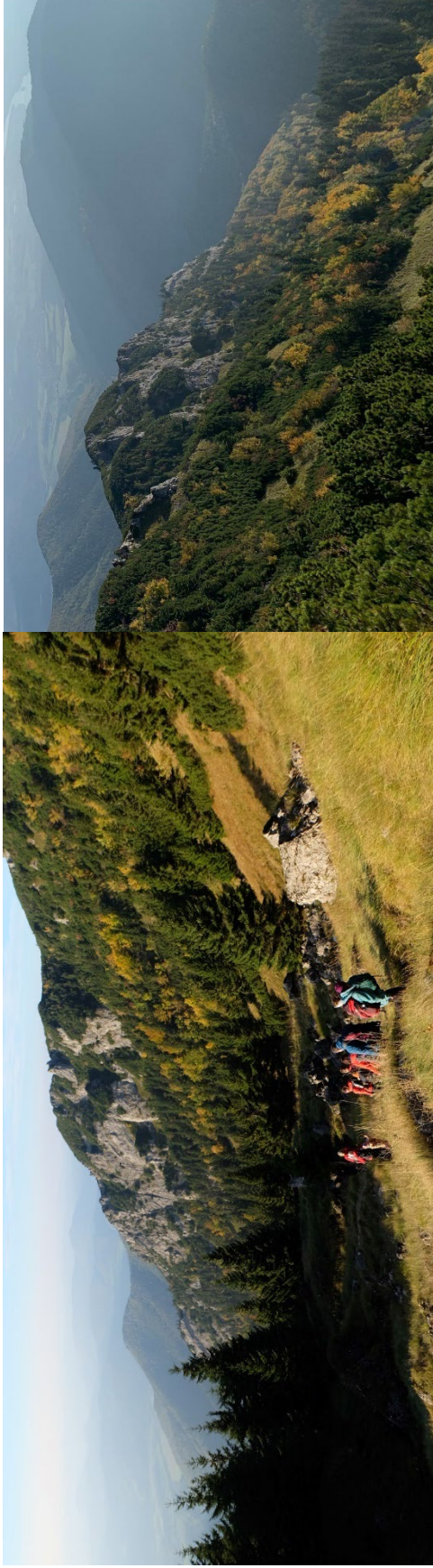


Figure 17: Malá Fatra mountain ridge covered with limestone nappes (source: author).

Table 2: Parameterization matrix of test runs in Malá Fatra and frequency precision assessment.

Test Run	PARAMETERIZATION MATRIX										RESULTS			FREQUENCY PRECISION	
	BUFFER_DIST	TOLERANCE	MIN_PROB_NORM	MAX_DIST	MAX_LATTICE_WIDTH	INCREASE_MAX_LATTICE_WIDTH	OBS_NOISE_NE	OBS_NOISE_NE	DIST_NOISE_NE	DIST_NOISE_NE	Time [s]	Length [m]	Control Point	Reference Value	Result from Calculation
1	100	6	0,006	260	5	4	20	100	8	40	513	128 447	1	3	3
2	100	6	0,006	280	5	4	20	100	8	40	662	128 463	2	2	2
3	100	6	0,006	240	5	4	20	100	8	40	670	128 384	3	2	2
4	100	6	0,006	260	7	5	20	100	8	40	949	132 573	4	3	2
5	100	6	0,006	260	9	6	20	100	8	40	970	132 522	5	4	4
6	100	6	0,006	260	7	5	16	70	6	30	924	132 835	6	2	2
7	100	6	0,006	260	7	5	16	50	6	24	714	128 996	7	4	4
8	100	6	0,006	260	7	5	12	70	4	30	947	121 173	8	0	0
9	120	6	0,006	260	7	5	16	70	6	30	862	132 874	9	1	1
10	140	6	0,006	260	7	5	16	70	6	30	775	121 322	10	2	2
11	120	7	0,006	260	7	5	16	70	6	30	861	132 508	11	1	1
12	120	5	0,006	260	7	5	16	70	6	30	972	133 063	12	1	1
13	120	6	0,008	260	7	5	16	70	6	30	1148	132 874	13	4	4
14	120	6	0,004	260	7	5	16	70	6	30	862	129 091	14	1	3
15													15	2	2

86,7 % measurements at control points are correct
 90,6 % of passages have been assigned correctly

overfitting optimal underfitting

5.3 Slovak Paradise Case Study

Slovak Paradise National Park is known for its attractive rugged relief, i.e. gorges, canyons, karst landforms etc. (fig. 18). This interesting environment generates challenges in terms of GNSS signal quality. The records are noisy in a very sparse road network.

The parameterisation for this study differentiates from the previous two significantly. The network is sparse and generally homogenous over the study area. The optimal parameterisation returns 91,8 km long routes with frequencies in 325 seconds (tab. 3). The validation of the result is done through 10 control points, see their distribution in [Attachment 6](#). At every control point, the calculated frequencies corresponded to the reference values (tab. 3). The precision based on the control points is 100 %. However, the points did not capture the overestimations due to the noisy and oscillating records. The errors due to the wrong road choice do not arise, because the road possibilities are much lower than in the urbanised area.



Figure 18: Hiking in the Slovak Paradise, Suchá Belá (source: author).

Table 3: Parameterization matrix of test runs in Slovak Paradise and frequency precision assessment.

Test Run	PARAMETRIZATION MATRIX										RESULTS		FREQUENCY PRECISION		
	BUFFER_DIST	TOLERANCE	MIN_PROB_NORM	MAX_DIST	MAX_LATTICE_WIDTH	INCREASE_MAX_LATTICE_WIDTH	OBS_N_OISE	OBS_NOISE_NE	DIST_N_OISE	DIST_NOISE_NE	Time [s]	Length [m]	Control Point	Reference Value	Result from Calculation
1	260	5	0,000	200	5	4	20	60	6	24	182	91 518	1	2	2
2	260	5	0,000	240	5	4	20	60	6	24	209	91 872	2	2	2
3	260	5	0,000	280	5	4	20	60	6	24	203	91 833	3	1	1
4	260	5	0,000	240	9	6	20	60	6	24	323	90 910	4	2	2
5	260	5	0,000	240	13	8	20	60	6	24	498	90 759	5	2	2
6	260	5	0,000	240	9	6	20	60	5	20	285	90 958	6	1	1
7	260	5	0,000	240	9	6	20	60	7	28	296	90 944	7	1	1
8	260	5	0,000	240	9	6	8	25	6	24	302	91 818	8	2	2
9	260	5	0,000	240	9	6	6	20	6	24	305	91 828	9	3	3
10	220	5	0,000	240	9	6	8	25	6	24	325	91 818	10	2	2
11	180	5	0,000	240	9	6	8	25	6	24	244	77 392			
12	220	7	0,000	240	9	6	8	25	6	24	271	91 425			
13	220	3	0,000	240	9	6	8	25	6	24	353	91 894			
14	220	5	0,001	240	9	6	8	25	6	24	276	91 688			

100 % correctness rate for the control points

overfitting optimal underfitting

6 RESULTS

The thesis consists of two main parts: the tool development ([Chapter 4](#)) and the testing and assessment ([Chapter 5](#)). The development follows the previous research of processing GNSS track records. The automated approach is unique to consolidate the positioning data in one linear layer with passage frequencies. Testing purposes the parameterisation methodology for optimal results. And the assessment part proves the result for a particular level of accuracy.

The *Jupyter Notebooks* for the direct interaction ([Attachment 1](#)) or the indirect through the web application ([Attachment 2](#)), allow the user to upload the GPX files to process them under specific parametrisation. A simple tool for compression of the GPX data has been developed to handle the large input datasets in a systematic manner ([Attachment 3](#)). After the calculation of passage frequencies, the user can inspect the processing outputs in two modes of web maps. The final outputs can be saved in two data formats and two web maps. The cloud-based tool is open to an expert user group throughout the world. The reference network data come from the global database of *OpenStreetMap contributors*. However, the ability of the result to convey a strong message originates from the credibility and objectivity of the input GNSS track records.

Testing and assessment in three different case studies support the importance of human knowledge in the loop of automation. Optimal parametrisation plays the most important role in the outcome. Generally, the frequency precision above 90 % grants the *gpx2intensity* tool to provide an innovative source of information for road infrastructure development, especially for active transport, i.e. hiking, running, cycling etc. [Attachments 4–6](#) are demonstrations of how the intensity maps for mobility can be designed in graduated-symbol symbology. The presented maps visualize the records of a single user; therefore, their primary purpose is to present the visual design. The content itself, passage frequencies, is not credible for further distribution.

7 DISCUSSION

The thesis investigates the possibilities of utilising a *Jupyter Notebook* for workflow automation of positioning data. The notebook runs on Python kernel, the processing of the automation is not as fast and efficient as it would be if run in other programming language. The notebook's advantage is its wide user group. The explanatory text superimposed in the code increases the chances for multiple users to manipulate the data-mining tool appropriately.

The community developing Python packages in the data science field, including GI science, is growing. Accordingly, all the development accounts for the growing popularity of *Jupyter Notebooks*. The notebooks are effective in delivering the essential knowledge to guide the end-user to successful results. Furthermore, the *Voilà* extends the code abstraction to the web application enabling *ipyWidgets* in GUI. Such web application demands computing power on the cloud. A recent initiative *Binder* pioneered to provide free cloud servers for virtual environments (Ragan-Kelley et al., 2018). The solution has become popular for collaboration among data scientists. However, the *Binder* sever can overload easily and the web application depends on the server availability at the time. The alternative approach is the collaborative notebook at *Google Colab* by *Google Research* (Google, 2023). It runs on the freemium business model. The basic computing power is free with further paid alternatives to increase the services and capabilities. Unfortunately, *Google Research* does not offer a free cloud server to run a web application like *Voilà*.

The next core idea, the integration of VGI together with scientific methods is a current trend in GI science (Àvila Callau, 2020). The VGI diversity is a benefit and a drawback at the same time. The diversity of data sources, crowd-sourcing, covers the environment in such a way that from the individual records, one can zoom out to the general, abstracted character of the region. The more data providers in an area, the better and more objective is the understanding of the study area. For instance, a case study for urban development can examine the behaviour of e-scooter riders within a city. The service providers dispose of the positioning data of their scooters, such VGI provides a multi-user sample of scooter driver traces. After the data mining processes, the most frequent passages could be subjected to a development test. It is a data-oriented method to address sustainable mobility development in urban areas. The thesis objective is not to generate similar actionable outcomes but rather to design a tool that is easy to implement in any environment. The tool is tested on diverse positioning data from different devices in different study areas. The datasets come from a single data provider; therefore, the results are highly subjective.

As mentioned earlier, the VGI bring drawbacks by its nature. Compared to data collected according to scientific principles, the general public does not follow the principles for data collection, the quality of VGI is variable and undocumented due to the different devices or methodologies, the contributors are not experts and have different interests while collecting data (Àvila Callau, 2020). Nonetheless, the advancing

technology capabilities and its democratisation allow users to contribute to “citizen science” with their data, such as GNSS track records of mobility.

Lastly, the *gpx2intensity* tool resolves the challenge of unifying positioning data of different levels of detail. The most possible approximation to reference street network, a route, creates the common ground to combine the different VGI sources together. The map-matching algorithm enables HMM to calculate routing for track parts with non-emitting states as well (Newson & Krumm, 2009). The tool parameterisation and computation are heavy. The processing of VGI requires expert knowledge, thus the very necessary information for the user is given within the tool. The result of automation summarises the total passage frequencies of the input positioning data. The notebook is distributed on online platforms, from which the user can run the tool on the cloud, or one can download the notebook and run it locally. Primarily, the tool pre-processes the geographic data for the cartographers to design graduated-symbol or graduated-colour maps. In the case of non-cartographers, the secondary output is a web map in two colour modes generated automatically. The continuation of the tool development can lead to data processing for other methods of geovisualization. Apart from the mobility behaviour, one can investigate the origin-destination relationships. In most cases, the primary purpose for mobility is the transport from point “A” to point “B”. Therefore, in the future the tool could be improved to visualize the GNSS track records in the edge-path bundling presented by Wallinger et al. (2022).

CONCLUSION

The “*Automation of Processing GNSS Track Records for Designing Intensity Maps*” compiles the solid foundation of the state-of-the-art and the growing importance of GNSS, data mining of track records and the geovisualization in Jupyter Notebook ([Chapter 2](#)). Based on the previous research done in the field, [Chapter 3](#) describes the methodology to accomplish the first two sub-goals: (1) automation of spatial analyses of GNSS trajectory data on street network, (2) implementation of possible appropriate quantitative geovisualization. The tool development ([Chapter 4](#)) discusses the design, debugging and exceptions, and documentation and distribution of the product. Lastly, the third sub-goal has been addressed in the chapters of Testing and Assessment ([Chapter 5](#)), Results ([Chapter 6](#)) and Discussion ([Chapter 7](#)). The aim is to assess the results and outline the possibilities for further use.

The thesis overall provides a scientific approach to solving the challenges of geovisualization of positioning data. The work profits from the community publishing reproducible, free and open-source research. New ideas for processing big geographic data, coming from satellites, sensors, wearable devices etc., have a vast potential to keep the development processes sustainable. Such approaches lead to an increased quality of life and prosperity in the form of economic assets. The geoprocessing tool is distributed in cloud platforms of two types. The published work anticipates contributing to the GI community with a new method to process GNSS track records and providing the groundwork for future developments either in the infrastructure or geovisualization perspective.

REFERENCES AND INFORMATION SOURCES

- ÁVILA CALLAU, Aitor, Yolanda PÉREZ-ALBERT, and David SERRANO GINÉ. *Quality of GNSS Traces from VGI: A Data Cleaning Method Based on Activity Type and User Experience*. Online. ISPRS International Journal of Geo-Information, vol. 9 (December 2020), no. 12, p. 727. ISSN 2220-9964. Available from: <https://doi.org/10.3390/ijgi9120727>. [viewed 2023-05-16].
- BOEING, Geoff. *OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks*. Online. Computers, Environment and Urban Systems, vol. 65 (September 2017), pp. 126–139. ISSN 0198-9715. Available from: <https://doi.org/10.1016/j.compenvurbsys.2017.05.004>. [viewed 2023-05-16].
- BOREALIS PRECISION. *GNSS Knowledge - Step 1 - Satellites*. Online. Borealis Precision - Industry Leading Representative. 2023. Available from: <https://www.gnss.ca/gnss/1291-step-1-satellites>. [viewed 2023-05-19].
- CHEN, Meng, Yang LIU, and Xiaohui YU. *Nlpmm: A next location predictor with markov modeling*. In: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 186–197. Tainan, Taiwan, 2014-05-13 – 2014-05-16. [n.d.].
- CORLAY, Sylvain, and Martin RENO. *Interactive GIS in Jupyter with ipyleaflet*. Online. Medium. 2019. Available from: <https://blog.jupyter.org/interactive-gis-in-jupyter-with-ipyleaflet-52f9657fa7a>. [viewed 2023-04-17].
- CRIVELLARI, Alessandro, Bernd RESCH, and Yuhui SHI. *TraceBERT—A Feasibility Study on Reconstructing Spatial–Temporal Gaps from Incomplete Motion Trajectories via BERT Training Process on Discrete Location Sequences*. Online. Sensors, vol. 22 (February 2022), no. 4, p. 1682. ISSN 1424-8220. Available from: <https://doi.org/10.3390/s22041682>. [viewed 2023-05-16].
- DABIRI, Sina, and Kevin HEASLIP. *Inferring transportation modes from GPS trajectories using a convolutional neural network*. Online. Transportation Research Part C: Emerging Technologies, vol. 86 (2018), pp. 360–371. ISSN 0968-090X. Available from: <https://doi.org/10.1016/j.trc.2017.11.021>. [viewed 2023-04-10].
- DOUGLAS, DAVID H., and THOMAS K. PEUCKER. *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*. Online. Cartographica: The International Journal for Geographic Information and Geovisualization, vol. 10 (1973), no. 2, pp. 112–122. ISSN 1911-9925. Available from: <https://doi.org/10.3138/fm57-6770-u75u-7727>. [viewed 2023-05-16].
- EOX IT SERVICES GMBH. *The global and cloudless Sentinel-2 map by EOX*. Online. Sentinel-2 cloudless map of the world by EOX. 2020. Available from: <https://s2maps.eu/>. [viewed 2023-04-17].
- EU AGENCY FOR SPACE PROGRAMME. *Market Overview*. In: EUSPA EO and GNSS Market Report, pp. 9–29. 2022.

- FENG, Zhenni, and Yanmin ZHU. *A Survey on Trajectory Data Mining: Techniques and Applications*. Online. IEEE Access, vol. 4 (2016), pp. 2056–2067. ISSN 2169-3536. Available from: <https://doi.org/10.1109/access.2016.2553681>. [viewed 2023-05-16].
- FIDLER, Jan. *City of Olomouc in November*. Image. 2013. Available from: https://live.staticflickr.com/2828/10931928614_de9502cd7a_b.jpg. [viewed 2023-05-19].
- FILIFE, Frank ANEMA, Rob STORY, James GARDINER, Martin JOURNOIS, et al. *python-visualization/folium: v0.14.0*. Online. Zenodo. 2022-12-12. Available from: <https://zenodo.org/record/7430093#.ZGPC2nb7RhF>. [viewed 2023-04-16].
- FOSTER, Dan. *GPX: the GPS Exchange Format*. Online. TopoGrafix. [n.d.]. Available from: <https://www.topografix.com/gpx.asp>. [viewed 2023-04-11].
- GAO, Qing-Bin and Shi-Liang SUN. *Trajectory-based human activity recognition using Hidden Conditional Random Fields*. Online. In: 2012 International Conference on Machine Learning and Cybernetics (ICMLC 2012). Xian, 2012-07-15 – 2012-07-17. IEEE, 2012. ISBN 9781467314879. Available from: <https://doi.org/10.1109/icmlc.2012.6359507>. [viewed 2023-04-16].
- GEPANDAS DEVELOPERS. *geopandas.GeoSeries.covers*. Online. GeoPandas 0.13.0 documentation. 2022. Available from: <https://geopandas.org/en/stable/docs/reference/api/geopandas.GeoSeries.covers.html>. [viewed 2023-02-07].
- GOOGLE. *Google Colab*. Online. Google Research. 2023. Available from: <https://research.google.com/colaboratory/faq.html>. [viewed 2023-05-03].
- GREENFELD, Joshua S. *Matching GPS Observations to Locations on a Digital Map*. In: 81th Annual Meeting of the Transportation Research Board. Washington, DC, USA. 2002.
- GROUT, Jason. *ipywidgets: Interactive Widgets for the Jupyter Notebook*. Online. GitHub. 2021. Available from: <https://github.com/jupyter-widgets/ipywidgets>. [viewed 2023-05-16].
- HEUSSER, Matt. *What is debugging?* Online. Software Quality. 2022. Available from: <https://www.techtarget.com/searchsoftwarequality/definition/debugging>. [viewed 2023-05-03].
- HUANG, Haosheng, and Song GAO. *Location-Based Services*. Online. Geographic Information Science & Technology Body of Knowledge, vol. 2018 (January 2018), Q1. Available from: <https://doi.org/10.22224/gistbok/2018.1.14>. [viewed 2023-05-16].
- JAN, Oliver, Alan J. HOROWITZ, and Zhong-Ren PENG. *Using Global Positioning System Data to Understand Variations in Path Choice*. Online. Transportation Research Record: Journal of the Transportation Research Board, vol. 1725 (2000), no. 1, pp. 37–44. ISSN 2169-4052. Available from: <https://doi.org/10.3141/1725-06>. [viewed 2023-05-16].

- JENSEN, Christian S., and Nerius TRADIŠAUSKAS. *Map Matching*. Online. In: Encyclopedia of Database Systems, pp. 1692–1696. Boston, MA: Springer US, 2009. ISBN 9780387355443. Available from: https://doi.org/10.1007/978-0-387-39940-9_215. [viewed 2023-02-07].
- JONIETZ, David, and Dominik BUCHER. *Continuous trajectory pattern mining for mobility behaviour change detection*. In: 14th International Conference on Location Based Services, pp. 211–230. Springer International Publishing, 2018.
- JORDAHL, Kelsey, Joris VAN DEN BOSSCHE, Martin FLEISCHMANN, Jacob WASSERMAN, James MCBRIDE, et al. *geopandas/geopandas: v0.8.1*. Online. Zendo. 2020. Available from: <https://doi.org/10.5281/zenodo.3946761>. [viewed 2023-01-16].
- LINDSAY, John. *WhiteboxTools User Manual*. Online. John Lindsay | Home. 2018. Available from: https://jblindsay.github.io/wbt_book. [viewed 2023-04-16].
- MEERT, Wannes, and Mathias VERBEKE. *HMM with non-emitting states for Map Matching*. In: European Conference on Data Analysis. Paderborn, Germany, 2018-07-04 – 2018-07-06. 2018.
- MEERT, Wannes. *Leuven.MapMatching toolbox for aligning GPS measurements to locations on a map*. Online. GitHub. 2021a. Available from: <https://github.com/wannesm/LeuvenMapMatching>. [viewed 2023-02-13].
- MEERT, Wannes. *LeuvenMapMatching/distance.py*. Online. GitHub. 2021b. Available from: <https://github.com/wannesm/LeuvenMapMatching/blob/master/leuvenmapmatching/matcher/distance.py>. [viewed 2023-01-10].
- PERKEL, Jeffrey M. *Why Jupyter is data scientists' computational notebook of choice*. Online. Nature, vol. 563 (October 2018), no. 7729, pp. 145–146. ISSN 1476-4687. Available from: <https://doi.org/10.1038/d41586-018-07196-1>. [viewed 2023-05-16].
- QUANTSTACK. *And voilà!* Online. Medium. 2019. Available from: <https://blog.jupyter.org/and-voilà-f6a2c08a4a93>. [viewed 2023-04-16].
- RAGAN-KELLEY, B., C. WILING, F. AKICI, D. LIPPA, D. NIEDERHUT, et al. *Binder 2.0- Reproducible, interactive, sharable environments for science at scale*. In: Proceedings of the 17th python in science conference, pp. 113–120. 2018.
- RAUBAL, Martin, David JONIETZ, Francesco CIARI, Konstantinos BOULOUCHOS, Stefan HIRSCHBERG, et al. *Towards an Energy Efficient and Climate Compatible Future Swiss Transportation System*. Online. ETH Zurich Research Collection, 2017. Available from: <https://doi.org/10.3929/ethz-b-000201484>.
- RENOU, Martin, Sylvain CORLAY, David BROCHART, Jason GROUT, Brian E. GRANGER, et al. *jupyter-widgets/ipyleaflet: A Jupyter - Leaflet.js bridge*. Online. GitHub. 2021. Available from: <https://github.com/jupyter-widgets/ipyleaflet>. [viewed 2023-04-16].

- SCHLÖMER, N. *Scientific Python*. Image. 2020. Available from: <https://user-images.githubusercontent.com/181628/80079616-cfe52400-8550-11ea-95a6-cdcab06530c1.png>. [viewed 2023-04-16].
- SHERRER, Kara. *Google Colab vs Jupyter Notebook: Compare data science software*. Online. TechRepublic. 2022. Available from: <https://www.techrepublic.com/article/google-colab-vs-jupyter-notebook/>. [viewed 2023-04-17].
- SONG, Xuan, Quanshi ZHANG, Yoshihide SEKIMOTO, and Ryosuke SHIBASAKI. *Prediction of human emergency behavior and their mobility following large-scale disaster*. Online. In: KDD '14: The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, NY, USA: ACM, 2014. ISBN 9781450329569. Available from: <https://doi.org/10.1145/2623330.2623628>. [viewed 2023-05-16].
- STRAVA. *About Us*. Online. Strava | Running, Cycling & Hiking App. 2023. Available from: <https://www.strava.com/about>. [viewed 2023-05-20].
- STURTZ, John. *Python Modules and Packages – An Introduction*. Online. Python Tutorials – Real Python. 2022. Available from: <https://realpython.com/python-modules-packages/>. [viewed 2023-04-16].
- ŠRAMO, Benjamín. *How to match GPX data that pass through an area with no road network?* Image. 2023. Available from: <https://github.com/wannesm/LeuvenMapMatching/issues/42>. [viewed 2023-05-19].
- TRABELSI, Eyal. *Debugging Jupyter Notebooks Will Boost Your Productivity*. Online. Medium. 2020. Available from: <https://towardsdatascience.com/debugging-jupyter-notebooks-will-boost-your-productivity-a33387f4fa62>. [viewed 2023-05-02].
- WALLINGER, Markus, Daniel ARCHAMBAULT, David AUBER, Martin NOLLENBURG, and Jaakko PELTONEN. *Edge-Path Bundling: A Less Ambiguous Edge Bundling Approach*. Online. IEEE Transactions on Visualization and Computer Graphics, vol. 28 (January 2022), no. 1, pp. 313–323. ISSN 1941-0506. Available from: <https://doi.org/10.1109/tvcg.2021.3114795>. [viewed 2023-05-18].
- WHITE, Christopher E., David BERNSTEIN, and Alain L. KORNHAUSER. *Some map matching algorithms for personal navigation assistants*. Online. Transportation Research Part C: Emerging Technologies, vol. 8 (2000), no. 1-6, pp. 91–108. ISSN 0968-090X. Available from: [https://doi.org/10.1016/s0968-090x\(00\)00026-7](https://doi.org/10.1016/s0968-090x(00)00026-7). [viewed 2023-05-16].
- WU, Qiusheng. *Leafmap: A Python package for interactive mapping and geospatial analysis with minimal coding in a Jupyter environment*. Online. Journal of Open Source Software, vol. 6 (2021), no. 63, p. 3414. ISSN 2475-9066. Available from: <https://doi.org/10.21105/joss.03414>. [viewed 2023-04-16].

ATTACHMENTS

LIST OF ATTACHMENTS

Bound Attachments:

- Attachment 1 gpx2intensity_tool.ipynb *(preview with the link to the source code)*
- Attachment 2 gpx2intensity.ipynb *(preview with the link to the source code)*
- Attachment 3 gpx_compression.py *(preview with the link to the source code)*
- Attachment 4 Map of Intensity: Olomouc, Case Study *(A3 format)*
- Attachment 5 Map of Intensity: Malá Fatra, Case Study *(A3 format)*
- Attachment 6 Map of Intensity: Slovak Paradise, Case Study *(A3 format)*

Free Attachments:

- Attachment 7 Thesis Website
- Attachment 8 Poster *(A2 format)*
- Attachment 9 Sample GNSS Track Records *(32 GPX files)*

Attachment 1 `gpx2intensity_tool.ipynb` (original location at [Google Colab](#))

GPX2INTENSITY

AUTOMATION OF PROCESSING GNSS TRACK RECORDS FOR DESIGNING INTENSITY

MAPS

The tool is an attachment 1 to the Master Thesis

"Automation of Processing GNSS Track Records for Designing the Intensity Maps"

submitted in Palacký University in Olomouc and University of Salzburg, 2023.



COPERNICUS MASTER
IN DIGITAL EARTH



With the support of the
Erasmus+ Programme
of the European Union

Install and Import Dependences

```
1 !pip list -v # check all preinstalled packages and their versions
```

```
1 !pip install numpy==1.23.0
2 #!pip install -U numpy
3 !pip install pyproj
4 !pip install gpx-converter
5 !pip install geopandas # geoprocessing
6 !pip install git+https://github.com/wannesm/LeuvenMapMatching.git # map matching
7 !pip install osmnx # OpenStreetMap street network data access
8 !apt-get install -y libspatialindex-dev # map visualization
9 !pip install fiona # map visualization
10 !pip install rtree # map visualization
11 !pip install mapclassify # map visualization
12 !pip install leafmap # map visualization
```

```
1 from os.path import exists, join, basename
2 from os import listdir, remove
3 from urllib.request import urlretrieve
4 import osmnx as ox
5 from shapely.geometry import Polygon, Point, LineString, MultiPolygon
6 from leuvenmapmatching.map.inmem import InMemMap
7 import numpy as np
8 import pandas as pd
9 import geopandas as gpd
10 from gpx_converter import Converter
11 from leuvenmapmatching.matcher.distance import DistanceMatcher
12 import leafmap.foliumap as leafmap
13 import mapclassify
```

1. Pre-Processing

Set Variables and Parameters

The Map Matching algorithm runs on probabilistic model handling states with lacking observation. For more information see [documentation](#) for the MM.

```

1  ## Set the directory location for data and output files
2  DATA_FOLDER = 'data' # path to your data repository
3  OUTPUT_FOLDER = 'output' # directory save the results
4
5  ## Study area for OpenStreetMap street network download
6  BUFFER_DIST = 100 # area around the GPX records; in meters
7
8  ## Map Matching parameters
9  TOLERANCE = 4 # threshold for GPX track simplification; in meters
10 MIN_PROB_NORM = 0.002 # eliminate the lattice where it drops below normalized probability
11 MAX_DIST = 100 # break for any further matching; in meters
12 MAX_LATTICE_WIDTH = 7 # search the route with this number of possible paths at every step
13 INCREASE_MAX_LATTICE_WIDTH = 4 # if no solution found, the width increments by the value
14 OBS_NOISE = 20 # expected GNSS noise, in meters
15 OBS_NOISE_NE = 60 # expected GNSS noise for non-emitting states , in meters
16 DIST_NOISE = 5 # difference between distance between matched route and distance [...]
17 DIST_NOISE_NE = 15 # difference between distances for non-emitting states, in meters

```

▼ From GPX Files to Study Area

In case of empty data folder, feed the repository with the GPX data.

Optionally, download the test GPX data from GitHub repository by running the code below.

```

1  ## Downloading sample data from GitHub
2  data_repo_url = 'https://raw.githubusercontent.com/...'
3  files = ['filename.gpx',
4          'filename_XY.gpx']
5
6  def download(url):
7      filename = join(DATA_FOLDER, basename(url))
8      if not exists(filename):
9          local, _ = urlretrieve(url, filename)
10         print('Downloaded ' + local)
11
12  for data in files:
13      download(data_repo_url + data)

```

```

1  gpx_combined = gpd.GeoDataFrame(columns = ['latitude', 'longitude'])
2  for filename in listdir(DATA_FOLDER):
3      if filename.endswith(".gpx"):
4          try:
5              gpx_file = Converter(input_file = DATA_FOLDER + "/" + filename).gpx_to_dataframe()
6          except:
7              print(filename, "is NOT valid, the tool removed the file from repository.")
8              remove(join(DATA_FOLDER, filename))
9              continue
10
11         gpx_point = gpd.GeoDataFrame(gpx_file, geometry=gpd.points_from_xy(gpx_file.longitude,
12             gpx_file.latitude)).set_crs('epsg:4326')
13         gpx_combined = pd.concat([gpx_combined, gpx_point])
14
15  if gpx_combined.empty:
16         raise Exception("Uploaded data is empty.")
17
18  if (gpx_combined["latitude"] >= 80).any() or (gpx_combined["latitude"] <= -80).any():
19         raise Exception("Execution failed, uploaded data contains in polar region.")
20
21  latitude = gpx_combined["latitude"].iloc[0]
22  distance_deg = BUFFER_DIST / (111319.488 * np.cos(np.radians(latitude)))
23  gpx_buffer = gpx_combined["geometry"].buffer(distance_deg).set_crs(4326)
24  areas = gpx_buffer.unary_union
25  areas = gpd.GeoDataFrame(index=[0], crs='epsg:4326', geometry=[areas])
26  if areas.type[0] == "MultiPolygon":
27         areas = areas.explode()
28
29  print("From the GPX files have been calculated", len(areas.index), "area(s) for graph(s).")

```

▼ 2. Data Mining: Map Matching

A. OSM Street Network Download

1. The code iterates over the study areas one by one.
2. Optionally set your custom filter based on "highway" attribute. Defaultly all routes mapped in OSM are part of the analysis.

```
FILTER = '['highway' ~ "footway|track|pedestrian|cycleway|path|unclassified|residential|service|tertiary"]'
```

3. Download graph for the area of interest based on network type or based on OSM *highway* attribute.
4. Create the in-memory representation of a map: translate graph(nodes, edges) to *InMemMap* object. The structure represents the connectivity of the graph suggesting the most possible transitions.

▼ B. Parameterised Map Matching

1. Iterate over the GPX files in the data folder.
2. Check if the file locates inside the study area, if not move to the next file.
3. Retrieve coordinates from GNSS measurements and create algorithm-readable object.
4. Save the street network of the area for post-processing.
5. Run the matching over the path.
6. Create single streets from the nodes that the matched path passes through and merge the pair of nodes (streets) into one DataFrame.

```
1 street_all = gpd.GeoDataFrame(columns = ['Latitude', 'Longitude'])
2 track_df = pd.DataFrame(columns = ['Latitude', 'Longitude', 'id'])
3 route_df = pd.DataFrame(columns = ['Latitude', 'Longitude', 'id'])
4 id = 1
5 for _, row in areas.iterrows():
6     ## A. Download graph based on network type
7     area_gdf = gpd.GeoDataFrame(index=[0], crs='epsg:4326', geometry=[row["geometry"]])
8     print("Building a graph for a specific area...")
9
10    try:
11        ### NETWORK TYPES {"all_private", "all", "bike", "drive", "drive_service", "walk"}
12        ### or TYPE YOUR OPTIONAL CUSTOM "HIGHWAY" FILTER HERE
13        graph = ox.graph_from_polygon(row["geometry"], network_type = 'all', simplify = False)
14    except:
15        print("WARNING: One area has no street network mapped.")
16        continue
17
18    print("===", graph, "\n") # number of nodes and edges
19    street_lines = ox.graph_to_gdfs(graph, nodes = False)
20    street_all = pd.concat([street_lines, street_all])
21    ### Map Object
22    map_con = InMemMap("road_network",
23                      use_latlon = True,
24                      use_rtree = True,
25                      index_edges = True)
26    for node in graph.nodes:
27        lat = graph.nodes[node]['y']
28        lon = graph.nodes[node]['x']
29        map_con.add_node(node, (lat, lon))
30
31    for edge in graph.edges:
32        node_a, node_b = edge[0], edge[1]
33        map_con.add_edge(node_a, node_b)
34        map_con.add_edge(node_b, node_a)
35
36    map_con.purge() # remove nodes without location or edges
37
```

```

38 ## B. Map Matching
39 for filename in listdir(DATA_FOLDER):
40     if not filename.endswith(".gpx"):
41         continue
42
43     gpx_df = Converter(input_file = DATA_FOLDER + "/" + filename).gpx_to_dataframe()
44     try:
45         gpx_point = gpd.GeoDataFrame(gpx_df, geometry = gpd.points_from_xy(gpx_df.longitude,
46             gpx_df.latitude)).set_crs('epsg:4326')
47     except:
48         print("WARNING: The file has no record of position, the matching stopped.")
49         continue
50     gpx_point['id'] = 1
51     area_check = gpx_point.within(area_gdf)
52     if not area_check.iloc[0]:
53         continue # skip the GPX file outside of the graph
54     print("Map matching of " + filename + " started...")
55     try:
56         gpx_line = gpx_point.groupby(['id']) ['geometry'].apply(lambda x: LineString(x.tolist()))
57     except:
58         print("WARNING: The file has only one record of position, the matching stopped.")
59         continue
60
61     line_gdf = gpd.GeoDataFrame(gpx_line, geometry = 'geometry').set_crs('epsg:4326')
62     tolerance_deg = TOLERANCE / (111319.488 * np.cos(np.radians(latitude)))
63     line_gdf['geometry'] = line_gdf['geometry'].simplify(tolerance_deg) # simplify
64     gpx_coords = line_gdf.apply(lambda row: list((row.geometry).coords), axis=1)
65     passage = list(gpx_coords[1])
66     track = []
67     path = []
68     for i in range(len(passage)):
69         lat = passage[i][1]
70         lon = passage[i][0]
71         path.append((lat, lon))
72         track.append([lat, lon])
73
74     track = np.array(track)
75     df = pd.DataFrame(track, columns = ['Latitude', 'Longitude'])
76     df['id'] = id # id for grouping into one line
77     track_df = pd.concat([track_df, df])
78     matcher = DistanceMatcher(map_con,
79         max_dist = MAX_DIST,
80         min_prob_norm = MIN_PROB_NORM,
81         max_lattice_width = MAX_LATTICE_WIDTH,
82         increase_max_lattice_width = INCREASE_MAX_LATTICE_WIDTH,
83         obs_noise = OBS_NOISE,
84         obs_noise_ne = OBS_NOISE_NE,
85         dist_noise = DIST_NOISE,
86         dist_noise_ne = DIST_NOISE_NE,
87         non_emitting_edgeid = False,
88         restrained_ne = False)
89
90     matcher.match(path, unique = False) # retain not only unique nodes in the sequence
91     if matcher.early_stop_idx is not None:
92         print("WARNING: Parts of the path were omitted from matching due to the road mismatch.")
93         from_matches = matcher.best_last_matches(k = MAX_LATTICE_WIDTH)
94         matcher.continue_with_distance(from_matches=from_matches, max_dist = MAX_DIST)
95         matcher.match(path, expand = True)
96
97     node_id = matcher.path_pred_olynodes_withjumps # node_ids the route passes through
98
99     id_route = 1
100     for i in range(len(node_id)-1):
101         route_node = []
102         lat = graph.nodes[node_id[i]]['y']

```

```

102     lon = graph.nodes[node_id[i]]['x']
103     latlon = [lat, lon]
104     route_node.append(latlon)
105     lat2 = graph.nodes[node_id[i + 1]]['y']
106     lon2 = graph.nodes[node_id[i + 1]]['x']
107     latlon2 = [lat2, lon2]
108     route_node.append(latlon2)
109
110     route_node = np.array(route_node)
111     df = pd.DataFrame(route_node, columns = ['Latitude', 'Longitude'])
112     df['id'] = id_route # the same id for one line (street)
113     route_df = pd.concat([route_df, df])
114     id_route += 1
115
116     id += 1
117
118     print("Matching of " + filename + " finished successfully.\n")
119
120 if route_df.empty:
121     raise Exception("The map matching has no result.")
122
123 del graph, map_con

```

```

Building a graph for a specific area...
=== MultiDiGraph with 15261 nodes and 31538 edges

Map matching of ol_7.gpx started...
Matching of ol_7.gpx finished successfully.

Map matching of ol_12.gpx started...
Matching of ol_12.gpx finished successfully.

Map matching of ol_2.gpx started...
Matching of ol_2.gpx finished successfully.

Map matching of ol_14.gpx started...
Matching of ol_14.gpx finished successfully.

Map matching of ol_15.gpx started...
WARNING: Parts of the path were omitted from matching due to the road mismatch.
Matching of ol_15.gpx finished successfully.

Map matching of ol_1.gpx started...
Matching of ol_1.gpx finished successfully.

Map matching of ol_5.gpx started...
Matching of ol_5.gpx finished successfully.

Map matching of ol_9.gpx started...
Matching of ol_9.gpx finished successfully.

Map matching of ol_10.gpx started...
Matching of ol_10.gpx finished successfully.

Map matching of ol_3.gpx started...
WARNING: Parts of the path were omitted from matching due to the road mismatch.
Matching of ol_3.gpx finished successfully.

Map matching of ol_6.gpx started...
WARNING: Parts of the path were omitted from matching due to the road mismatch.
Matching of ol_6.gpx finished successfully.

Map matching of ol_13.gpx started...
WARNING: Parts of the path were omitted from matching due to the road mismatch.
Matching of ol_13.gpx finished successfully.

Building a graph for a specific area...
=== MultiDiGraph with 1237 nodes and 2604 edges

Map matching of ol_11.gpx started...
WARNING: Parts of the path were omitted from matching due to the road mismatch.
Matching of ol_11.gpx finished successfully.

Map matching of ol_8.gpx started...
WARNING: Parts of the path were omitted from matching due to the road mismatch.
Matching of ol_8.gpx finished successfully.

Map matching of ol_4.gpx started...
Matching of ol_4.gpx finished successfully.

```


▼ 3. Post-processing

2D Array into GeoDataFrame

From 2D array of coordinates returns a set of LineStrings of tracks and matched routes differentiated by the IDs.

▼ Calculation of Frequencies

Clean the street network from unnecessary attributes and duplicates. Spatial filtering performs an intersection of street network and routes, and frequency incrementation based on covers function.

Finally, the length of matched routes is printed.

```
1 # Arrays of tracks and matched routes to GeoDataFrame
2 track_point = gpd.GeoDataFrame(track_df, geometry = gpd.points_from_xy(
3     track_df.Longitude, track_df.Latitude))
4 track_lines = track_point.groupby(['id'])['geometry'].apply(lambda x: LineString(x.tolist()))
5 tracks_gdf = gpd.GeoDataFrame(track_lines, geometry = 'geometry').set_crs('epsg:4326')
6 route_point = gpd.GeoDataFrame(route_df, geometry = gpd.points_from_xy(route_df.Longitude,
7     route_df.Latitude))
8 route_line = route_point.groupby(['id'])['geometry'].apply(lambda x: LineString(x.tolist()))
9 route_gdf = gpd.GeoDataFrame(route_line, geometry = 'geometry').set_crs('epsg:4326')
10 # Calculation of Frequencies
11 street_all = street_all.loc[:, ['osmid', 'length', 'geometry']]
12 street_freq = street_all.overlay(route_gdf, how = 'intersection') # drop streets out of the routes
13 street_freq = street_freq.drop_duplicates(subset=['osmid', 'length'])
14 frequency = []
15 print("Calculating passage frequencies on streets...")
16 for _, row in street_freq.iterrows():
17     bool_series = route_gdf.covers(row["geometry"])
18     frequency.append(bool_series.values.sum())
19 street_freq["frequency"] = frequency
20 print("The length of matched roads is", round(street_freq["length"].sum()), "meters.")
```

```
Calculating passage frequencies on streets...
The length of matched roads is 102657 meters.
```

▼ Visualization

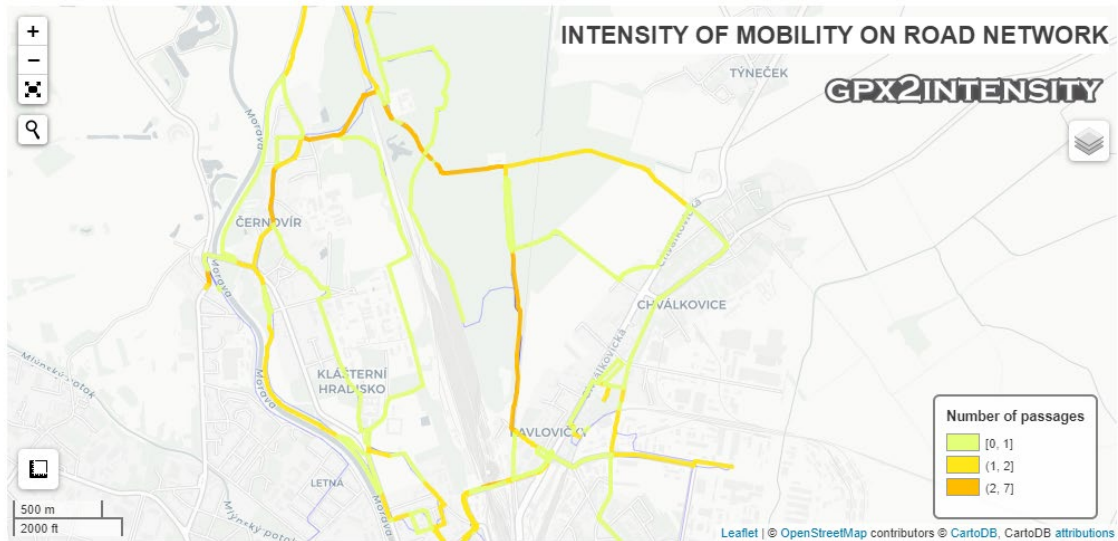
The leafmap package utilises the features of leaflet mapping library and folium python package.

```
1 m_light = leafmap.Map(width = "100%",
2     height = "480",
3     draw_control = False,
4     attribution_control = True,
5     tiles = "CartoDB positron")
6 m_light.add_gdf(tracks_gdf,
7     layer_name = "GPX tracks",
8     info_mode = None,
9     style = {'color':'blue', 'weight':0.5, 'opacity': 0.5})
10 m_light.add_data(street_freq,
11     "frequency",
12     cmap = "Wistia", # color ramp
13     scheme = "Quantiles", # classification scheme
14     k = 5, # max. number of classes (saved in attribute)
15     add_legend = True,
16     legend_title = "Number of passages",
17     legend_position = "bottomright",
18     layer_name = "road passages",
19     style_function = lambda feat: {"color": feat["properties"]["color"],
20     "weight": 4,
21     "opacity": 0.9})
22 m_light.add_text("INTENSITY OF MOBILITY ON ROAD NETWORK", fontsize = 22, fontcolor='#404040',
23     bold = True, padding = '0px', background = True, bg_color = 'white',
24     border_radius = '5px', position = 'topright')
```

```

m_light.add_text("<a href = 'https://github.com/bsramo144/Thesis-Jupyter' target = '_blank'>
23 <img width = '250' alt = 'Asset 3logo' src = 'https://user-
images.githubusercontent.com/47752920/234973760-c8157fdd-a3cf-88b0-
4dc8096cfe7c.png'></a>", background = False, position = 'topright')
24 m_light.zoom_to_gdf(street_freq)
25 m_light

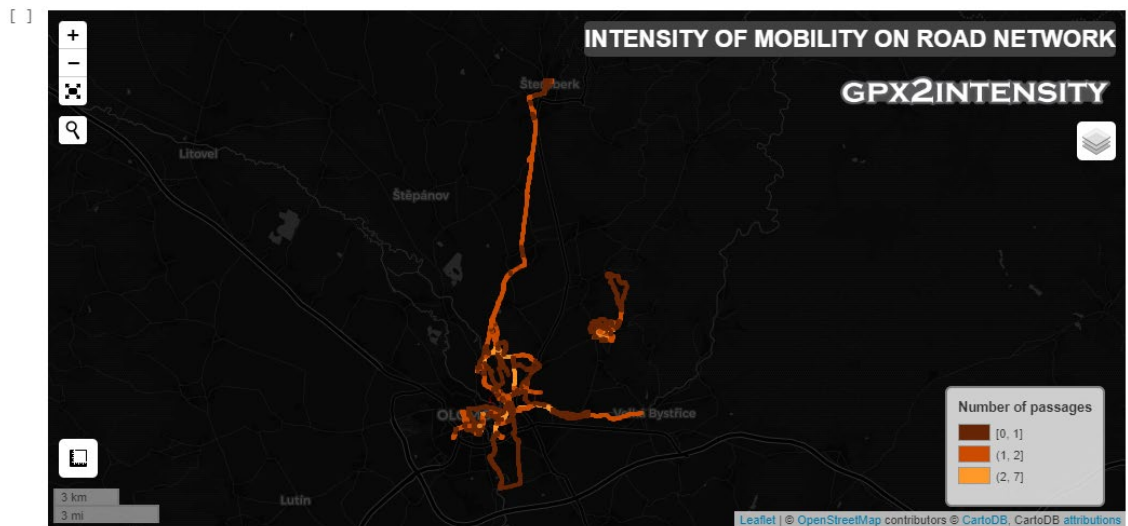
```



```

1 m_dark = leafmap.Map(width = "100%",
2 height = "480",
3 draw_control = False,
4 attribution_control = True,
5 tiles = "Cartodbdark_matter")
6 m_dark.add_gdf(tracks_gdf,
7 layer_name = "GPX tracks",
8 info_mode = None,
9 style = {'color':'red', 'weight':0.5, 'opacity': 0.5})
10 m_dark.add_data(street_freq,
11 "frequency",
12 cmap = "YlOrBr_r", # color ramp
13 scheme = "Quantiles", # classification scheme
14 k = 5, # max. number of classes (saved in attribute)
15 add_legend = True,
16 legend_title = "Number of passages",
17 legend_position = "bottomright",
18 layer_name = "road passages",
19 style_function = lambda feat: {"color": feat["properties"]["color"],
20 "weight": 4,
21 "opacity": 0.9})
22 m_dark.add_text("INTENSITY OF MOBILITY ON ROAD NETWORK", fontsize = 22, fontcolor='#404040',
bold = True, padding = '0px', background = True, bg_color = 'white',
border_radius = '5px', position = 'topright')
m_dark.add_text("<a href = 'https://github.com/bsramo144/Thesis-Jupyter' target = '_blank'>
23 <img width = '250' alt = 'Asset 3logo' src = 'https://user-
images.githubusercontent.com/47752920/234973760-c8157fdd-a3cf-88b0-
4dc8096cfe7c.png'></a>", background = False, position = 'topright')
24 m_dark.zoom_to_gdf(street_freq)
25 m_dark

```



▼ Save the Results

- Web data preview (light or dark basemaps)
- Spatial Data for GIS (GeoPackage, GeoJSON)

```

1 # Save the final linear layer with frequencies to GeoJSON and GeoPackage file
2 street_freq.to_file(OUTPUT_FOLDER + "/lines_freq.json", driver="GeoJSON")
3 street_freq.to_file(OUTPUT_FOLDER + "/lines_freq.gpkg", driver="GPKG")
4 # Save the interactive map to html
5 m_light.to_html(OUTPUT_FOLDER + "/light_map.html")
6 m_dark.to_html(OUTPUT_FOLDER + "/dark_map.html")

```

Attachment 2 `gpx2intensity.ipynb` (*[the original source code on GitHub](#)*)

```
%%html
<style>
  table {float:left}
  .folium-map leaflet-container leaflet-fade-anim leaflet-grab leaflet-touch-drag {
    display: inline-block;
  }
</style>
```

```
import ipywidgets as widget
from IPython.display import FileLink, HTML
from os.path import exists, join, basename
from os import listdir, remove
import osmnx as ox
from shapely.geometry import Polygon, Point, LineString, box
from leuvenmapmatching.map.inmem import InMemMap
import numpy as np
import pandas as pd
import geopandas as gpd
from gpx_converter import Converter
from leuvenmapmatching.matcher.distance import DistanceMatcher
import leafmap.foliummap as leafmap
import time
```

GPX2INTENSITY

AUTOMATION OF PROCESSING GPX TRACK RECORDS FOR DESIGNING INTENSITY MAPS

The tool matches GNSS track records to a road network and calculates frequencies on it. The map matching runs on probabilistic model handling states with missing observation i.e., non-emitting states. For more information about the map-matching algorithm see [Leuven.MapMatching documentation](#). For more details regarding the tool see the [GitHub documentation](#).

Set Parameters

```
upload_out = widget.Output()
upload_out
```

```
def clear_upload():
    with upload_out:
        upload_out.clear_output()
        DATA_UPLOAD = widget.FileUpload(
            accept='.gpx',
            multiple=True,
            description='Upload files')
        display(DATA_UPLOAD)
    return DATA_UPLOAD
```

```
DATA_UPLOAD = clear_upload()
```

Parameter	Description
Buffer	<i>expand the study area from the GPX records; in meters</i>
Tolerance	<i>GPX track simplification threshold; in meters</i>
Min. Proba.	<i>stop matching below normalized probability</i>
Max. Distance	<i>break for zero match probability; in meters</i>
Max Lattice	<i>search the route with the number of possible paths at every step</i>
Increase Latt.	<i>if no solution is found, increase the lattice by the value</i>
Obs. Noise	<i>standard deviation of measurement noise, in meters</i>
Obs. Noise NE	<i>standard deviation of measurement noise for non-emitting states, in meter (the value should be larger than Obs. Noise)</i>
Dist. Noise	<i>difference between distance between matched route and distance between tracks, in meters</i>
Dist. Noise Ne	<i>difference between the distances for non-emitting states, in meters (the value should be larger than Dist. Noise)</i>

```
BUFFER_DIST = widget.IntSlider(  
    value=140,  
    min=40,  
    max=300,  
    step=20,  
    description='Buffer',  
    continuous_update=False,  
    orientation='horizontal',  
    readout=True,  
    readout_format='d',  
)  
TOLERANCE = widget.IntSlider(  
    value=2,  
    min=0,  
    max=10,  
    step=1,  
    description='Tolerance',  
    continuous_update=False,  
    orientation='horizontal',  
    readout=True,  
    readout_format='d',  
)  
MAX_DIST = widget.IntSlider(  
    value=120,  
    min=20,  
    max=300,  
    step=20,  
    description='Max. Distance',  
    continuous_update=False,  
    orientation='horizontal',  
    readout=True,  
    readout_format='d'  
)  
OBS_NOISE = widget.IntSlider(  
    value=16,  
    min=2,  
    max=50,  
    step=2,  
    description='Obs. Noise',  
    disabled=False,  
    continuous_update=False,  
    orientation='horizontal',  
    readout=True,  
    readout_format='d'  
)  
OBS_NOISE_NE = widget.IntSlider(  
    value=30,  
    min=5,  
    max=150,  
    step=5,  
    description='Obs.Noise NE',  
    disabled=False,  
    continuous_update=False,  
    orientation='horizontal',  
    readout=True,  
    readout_format='d'  
)  
DIST_NOISE = widget.IntSlider(  
    value=5,  
    min=1,  
    max=50,  
    step=1,  
    description='Dist. Noise',  
    disabled=False,
```

```
        continuous_update=False,
        orientation='horizontal',
        readout=True,
        readout_format='d'
    )
DIST_NOISE_NE = widget.IntSlider(
    value=16,
    min=4,
    max=100,
    step=2,
    description='Dist.Noise NE',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='d'
)
MAX_LATTICE_WIDTH = widget.IntSlider(
    value=7,
    min=1,
    max=20,
    step=1,
    description='Max Lattice',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='d'
)
INCREASE_MAX_LATTICE_WIDTH = widget.IntSlider(
    value=5,
    min=1,
    max=10,
    step=1,
    description='Increase Latt.',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='d'
)
MIN_PROB_NORM = widget.FloatSlider(
    value=0.002,
    min=0,
    max=0.01,
    step=0.001,
    description='Min Proba.',
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.3f',
)
```

```
items = [widget.VBox([BUFFER_DIST, TOLERANCE, MIN_PROB_NORM]),
         widget.VBox([MAX_DIST, MAX_LATTICE_WIDTH, INCREASE_MAX_LATTICE_WIDTH]),
         widget.VBox([OBS_NOISE, OBS_NOISE_NE, DIST_NOISE, DIST_NOISE_NE])]
accordion = widget.Accordion(children=items)
accordion.set_title(0,"Environment Parameters")
accordion.set_title(1,"Thresholding of Matching")
accordion.set_title(2,"Measurement Noise")
accordion
```

```

def upload(DATA_UPLOAD):
    for filename in listdir("data_upload"):
        if filename.endswith(".gpx"):
            remove(join("data_upload", filename))

    if len(DATA_UPLOAD) == 0:
        with output:
            raise Exception("NO file has been uploaded. Check if the size of the files exceeds the
upload limit 10 MB.")
            raise Exception("Reload the page and upload new files again.")
        raise Exception(0)

    for elem in DATA_UPLOAD.items():
        name, file_info = elem
        data_path = join("data_upload", name)
        with open (data_path, 'wb') as file:
            file.write(file_info['content'])

```

```

def study_areas(BUFFER_DIST):
    gpx_combined = gpd.GeoDataFrame(columns = ['latitude', 'longitude'])
    for filename in listdir("data_upload"):
        if filename.endswith(".gpx"):
            try:
                gpx_file = Converter(input_file = "data_upload/"+ filename).gpx_to_dataframe()
            except:
                with output:
                    print(" WARNING:", filename, "is invalid GPX file, the tool will skip the
file.")
                    remove(join("data_upload", filename))
                continue

            gpx_point = gpd.GeoDataFrame(gpx_file, geometry=gpd.points_from_xy(gpx_file.longitude,
gpx_file.latitude)).set_crs('epsg:4326')
            gpx_combined = pd.concat([gpx_combined, gpx_point])

    if gpx_combined.empty:
        with output:
            raise Exception("Uploaded data is empty.")
        raise Exception(0)

    if (gpx_combined["latitude"] >= 80).any() or (gpx_combined["latitude"] <= -80).any():
        with output:
            raise Exception("Execution failed, uploaded data contains coordinates in polar area.
Reload the page and upload new files again.")
            raise Exception("Reload the page and upload new files again.")
        raise Exception(0)

    latitude = gpx_combined["latitude"].iloc[0]
    distance_deg = BUFFER_DIST / (111319.488 * np.cos(np.radians(latitude))) # metric system to
degree distance
    gpx_buffer = gpx_combined.buffer(distance_deg).set_crs(4326)
    areas = gpx_buffer.unary_union
    areas = gpd.GeoDataFrame(index=[0], crs='epsg:4326', geometry=[areas])
    if areas.type[0] == "MultiPolygon":
        areas = areas.explode()
    with output:
        print("From the uploaded GPX files have been calculated", len(areas.index), "area(s) for
graph(s). \n")
    return latitude, areas

```

```

def map_matching(latitude, areas, TOLERANCE, MAX_DIST, MIN_PROB_NORM, MAX_LATTICE_WIDTH,
INCREASE_MAX_LATTICE_WIDTH, OBS_NOISE, OBS_NOISE_NE, DIST_NOISE, DIST_NOISE_NE):
    street_all = gpd.GeoDataFrame(columns = ['Latitude', 'Longitude'])
    track_df = pd.DataFrame(columns = ['Latitude', 'Longitude', 'id'])
    route_df = pd.DataFrame(columns = ['Latitude', 'Longitude', 'id'])
    id = 1
    for _, row in areas.iterrows():
        ## Download graph based on network type
        ### possible network types {"all_private", "all", "bike", "drive", "drive_service", "walk"}
        with output:
            print("Building graph for a specific area...")
            area_gdf = gpd.GeoDataFrame(index=[0], crs='epsg:4326', geometry=[row["geometry"]])

        try:
            graph = ox.graph_from_polygon(row["geometry"], network_type = 'all', simplify = False)
        except:
            with output:
                print("In the area is no street network mapped. \n Reload the page and upload new
files again.")
                continue

            with output:
                print(graph, "\n") # number of nodes and edges
                street_lines = ox.graph_to_gdfs(graph, nodes = False)
                street_all = pd.concat([street_lines, street_all])
        ## Leuven Map Object
        map_con = InMemMap("road_network",
                            use_latlon = True,
                            use_rtree = True,
                            index_edges = True)

        for node in graph.nodes:
            lat = graph.nodes[node]['y']
            lon = graph.nodes[node]['x']
            map_con.add_node(node, (lat, lon))

        for edge in graph.edges:
            node_a, node_b = edge[0], edge[1]
            map_con.add_edge(node_a, node_b)
            map_con.add_edge(node_b, node_a)

        map_con.purge() # remove nodes without location or edges
        ## Map Matching
        for filename in listdir("data_upload"):
            if not filename.endswith(".gpx"):
                continue

            gpx_df = Converter(input_file = "data_upload/" + filename).gpx_to_dataframe()
            gpx_point = gpd.GeoDataFrame(gpx_df, geometry = gpd.points_from_xy(gpx_df.longitude,
gpx_df.latitude)).set_crs('epsg:4326')
            gpx_point['id'] = 1
            area_check = gpx_point.within(area_gdf)
            if not area_check.iloc[0]:
                continue # skip the GPX file outside of the graph

            with output:
                print(" Map matching of " + filename + " started...")
            try:
                gpx_line = gpx_point.groupby(['id']) ['geometry'].apply(lambda x:
LineString(x.tolist()))
            except:
                with output:
                    print(" WARNING: The file has only one record of position, therefore, the
matching stopped.")
                continue

```



```

line_gdf = gpd.GeoDataFrame(gpx_line, geometry = 'geometry').set_crs('epsg:4326')
tolerance_deg = TOLERANCE / (111319.488 * np.cos(np.radians(latitude))) # metric
system to degree distance
line_gdf['geometry'] = line_gdf['geometry'].simplify(tolerance_deg) # reducing line
vertices inside the tolerance
gpx_coords = line_gdf.apply(lambda row: list((row.geometry).coords), axis=1)
# for row in gpx_coords.items():
passage = list(gpx_coords[1])
track = []
path = []
for i in range(len(passage)):
    lat = passage[i][1]
    lon = passage[i][0]
    path.append((lat, lon))
    track.append([lat, lon])

track = np.array(track)
df = pd.DataFrame(track, columns = ['Latitude', 'Longitude'])
df['id'] = id # id for grouping into one line
track_df = pd.concat([track_df, df])
matcher = DistanceMatcher(map_con,
                           max_dist = MAX_DIST,
                           min_prob_norm = MIN_PROB_NORM,
                           max_lattice_width = MAX_LATTICE_WIDTH,
                           increase_max_lattice_width = INCREASE_MAX_LATTICE_WIDTH,
                           obs_noise = OBS_NOISE,
                           obs_noise_ne = OBS_NOISE_NE,
                           dist_noise = DIST_NOISE,
                           dist_noise_ne = DIST_NOISE_NE,
                           non_emitting_edgeid = False,
                           restrained_ne = False)

matcher.match(path, unique = False) # retain only unique nodes in the sequence (avoid
repetitions)
if matcher.early_stop_idx is not None:
    with output:
        print(" Parts of the path were omitted from matching due to the road
mismatch.")

    from_matches = matcher.best_last_matches(k = MAX_LATTICE_WIDTH)
    matcher.continue_with_distance(from_matches = from_matches, max_dist = MAX_DIST)
    matcher.match(path, expand = True)

node_id = matcher.path_pred_onlynodes_withjumps # retrieve the node_ids the route
passes through
id_route = 1
for i in range(len(node_id)-1):
    route_node = []
    lat = graph.nodes[node_id[i]]['y']
    lon = graph.nodes[node_id[i]]['x']
    latlon = [lat, lon]
    route_node.append(latlon)
    lat2 = graph.nodes[node_id[i + 1]]['y']
    lon2 = graph.nodes[node_id[i + 1]]['x']
    latlon2 = [lat2, lon2]
    route_node.append(latlon2)
    route_node = np.array(route_node)
    df = pd.DataFrame(route_node, columns = ['Latitude', 'Longitude'])
    df['id'] = id_route # the same id for one line (street)
    route_df = pd.concat([route_df, df])
    id_route += 1

id += 1
with output:

```

```

        print(" Matching of " + filename + " finished successfully.\n")

    if route_df.empty:
        with output:
            raise Exception("The map matching has no result, the execution has terminated.
Reload the page and upload new files again.")
            raise Exception(0)

    del graph, street_lines, map_con, node_id
    return street_all, track_df, route_df

```

```

def post_process(track_df, route_df, street_all, start_time):
    track_point = gpd.GeoDataFrame(track_df, geometry=gpd.points_from_xy(track_df.Longitude,
track_df.Latitude))
    track_lines = track_point.groupby(['id']) ['geometry'].apply(lambda x: LineString(x.tolist()))
    tracks_gdf = gpd.GeoDataFrame(track_lines, geometry='geometry').set_crs('epsg:4326')
    #print(tracks_gdf[:10])
    route_point = gpd.GeoDataFrame(route_df, geometry=gpd.points_from_xy(route_df.Longitude,
route_df.Latitude))
    route_line = route_point.groupby(['id']) ['geometry'].apply(lambda x: LineString(x.tolist()))
    route_gdf = gpd.GeoDataFrame(route_line, geometry='geometry').set_crs('epsg:4326')
    #print(route_gdf[:10])
    street_all = street_all.loc[:, ['osmid', 'length', 'geometry']] # drop unnecessary columns.
    with output:
        print("Calculating passage frequencies on streets, this part may take time...")
        street_freq = street_all.overlay(route_gdf, how='intersection') # drop geometries not part of
the routes
        street_freq = street_freq.drop_duplicates(subset=['osmid', 'length'])
        frequency = []
        for _, row in street_freq.iterrows():
            series = route_gdf.covers(row["geometry"])
            frequency.append(series.values.sum())
        street_freq["frequency"] = frequency
        #street_freq = street_freq.dissolve(by='osmid')
        output.clear_output()
        with output:
            print("The length of matched roads is", round(street_freq["length"].sum(), "meters.")
            print("The execution has finished in %s seconds." % (round(time.time() - start_time)))
            print("\n\n")

    return tracks_gdf, street_freq

```

```

def map_vis(tracks_gdf, street_freq):
    m_light = leafmap.Map(width="100%",
        height="380",
        draw_control=False,
        attribution_control=True,
        tiles="CartoDB positron")
    m_light.add_gdf(tracks_gdf,
        layer_name='tracks',
        info_mode=None,
        style={'color':'blue', 'weight':0.5, 'opacity': 0.5})
    m_light.add_data(street_freq,
        "frequency",
        cmap = "Wistia",
        scheme='Quantiles',
        k=5,
        add_legend=True,
        legend_title="Number of passages",
        legend_position="bottomright",
        layer_name="passages",
        style_function = lambda feat: {"color": feat["properties"]["color"],
            "weight": 4,
            'opacity': 0.9})
    m_light.add_text("INTENSITY OF MOBILITY ON ROAD NETWORK", fontsize=22, fontcolor='#404040',

```

```

bold=True, padding='0px', background=True, bg_color='white', border_radius='5px',
position='topright')
    m_light.add_text("<a href='https://github.com/bsramo144/Thesis-Jupyter' target='_blank'><img
width='250' alt='Asset 3logo' src='https://user-images.githubusercontent.com/47752920/234973760-
c8157fdd-a3cf-43cf-88b0-4dc8096cfe7c.png'></a>", background=False, position='topright')
    m_light.zoom_to_gdf(street_freq)

    m_dark = leafmap.Map(width="100%",
                        height="380",
                        draw_control=False,
                        attribution_control=True,
                        tiles="Cartodbdark_matter")
    m_dark.add_gdf(tracks_gdf,
                  layer_name='tracks',
                  info_mode=None,
                  style={'color':'red', 'weight':0.5, 'opacity': 0.5})
    m_dark.add_data(street_freq,
                    "frequency",
                    cmap = "YlOrBr_r",
                    scheme='Quantiles',
                    k=5,
                    add_legend=True,
                    legend_title="Number of passages",
                    legend_position="bottomright",
                    layer_name="passages",
                    style_function = lambda feat:{"color": feat["properties"]["color"],
                                                "weight": 4,
                                                'opacity': 0.8})

    m_dark.add_text("INTENSITY OF MOBILITY ON ROAD NETWORK", fontsize=22, fontcolor='white',
bold=True, padding='0px', background=True, bg_color='#404040', border_radius='5px',
position='topright')
    m_dark.add_text("<a href='https://github.com/bsramo144/Thesis-Jupyter' target='_blank'><img
width='250' alt='Asset 3logo' src='https://user-images.githubusercontent.com/47752920/234973760-
c8157fdd-a3cf-43cf-88b0-4dc8096cfe7c.png'></a>", background=False, position='topright')
    m_dark.zoom_to_gdf(street_freq)

    return m_light, m_dark

```

```

def save_results(street_freq, m_light, m_dark):
    street_freq.to_file("data_upload/lines.json", driver="GeoJSON")
    street_freq.to_file("data_upload/lines.gpkg", driver="GPKG")
    m_light.to_html("data_upload/light_map.html")
    m_dark.to_html("data_upload/dark_map.html")

```

```

def download_button(name, button_title):
    file_name = str(FileLink("data_upload/"+name)).rpartition('/')[2]
    html_voila = '<a style="color: white; border-radius: 3px;" class="lm-Widget p-Widget
jupyter-widgets jupyter-button widget-button mod-primary"
href="../../files/data_upload/'+file_name+"
download='"+file_name+" ">'+button_title+'</a>'
    html_jupyter = '<a style="color: white; border-radius: 3px;" class="lm-Widget p-Widget
jupyter-widgets jupyter-button widget-button mod-primary"
href="../../data_upload/'+file_name+" ">'+button_title+'</a>'
    display(HTML(html_voila))

```

```

def click(b):
    global DATA_UPLOAD
    start_time = time.time()
    output.clear_output()
    with output:
        print("CALCULATION STARTED \n")
        upload(DATA_UPLOAD.value)
    with output:
        print("(1/3) Creating study area zones around the uploaded data.")
        latitude, areas = study_areas(BUFFER_DIST.value)

```

```

with output:
    print("===")
    print("(2/3) Matching GPX files to the street graph of the study area.")
    street_all, track_df, route_df = map_matching(latitude, areas, TOLERANCE.value,
                                                MAX_DIST.value, MIN_PROB_NORM.value,
MAX_LATTICE_WIDTH.value, INCREASE_MAX_LATTICE_WIDTH.value, OBS_NOISE.value, OBS_NOISE_NE.value,
DIST_NOISE.value, DIST_NOISE_NE.value)

with output:
    print("===")
    print("(3/3) Post-processing of the results.")

tracks_gdf, street_freq = post_process(track_df, route_df, street_all, start_time)
m_light, m_dark = map_vis(tracks_gdf, street_freq)
with output:
    display(m_light, m_dark)

save_results(street_freq, m_light, m_dark)
with output:
    display(HTML("<b>Download outputs:</b>"))
    display(HTML("Linear Features (.json, .gpkg)"))
    download_button('lines.json', 'GeoJson')
    download_button('lines.gpkg', 'GeoPackage')
    display(HTML("Web Map (.html)"))
    download_button('light_map.html', 'Light Mode')
    download_button('dark_map.html', 'Dark Mode')

DATA_UPLOAD = clear_upload()
return DATA_UPLOAD

```

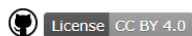
```

output = widget.Output()
RUN_BUTTON = widget.Button(description='Run the Tool', button_style='primary')
RUN_BUTTON.on_click(click)
RUN_BUTTON

```

```
output
```

The tool is Attachment 2 to Master Thesis.



Benjamin Šramo (2023)
Palacký University in Olomouc, University of Salzburg

Attachment 3 `gpx_compression.py` (*the original source code on GitHub*)

```
!pip install gpx-converter
!pip install geopandas

import pandas as pd
import geopandas as gpd
from os import listdir
from gpx_converter import Converter
from shapely.geometry import LineString
import numpy as np

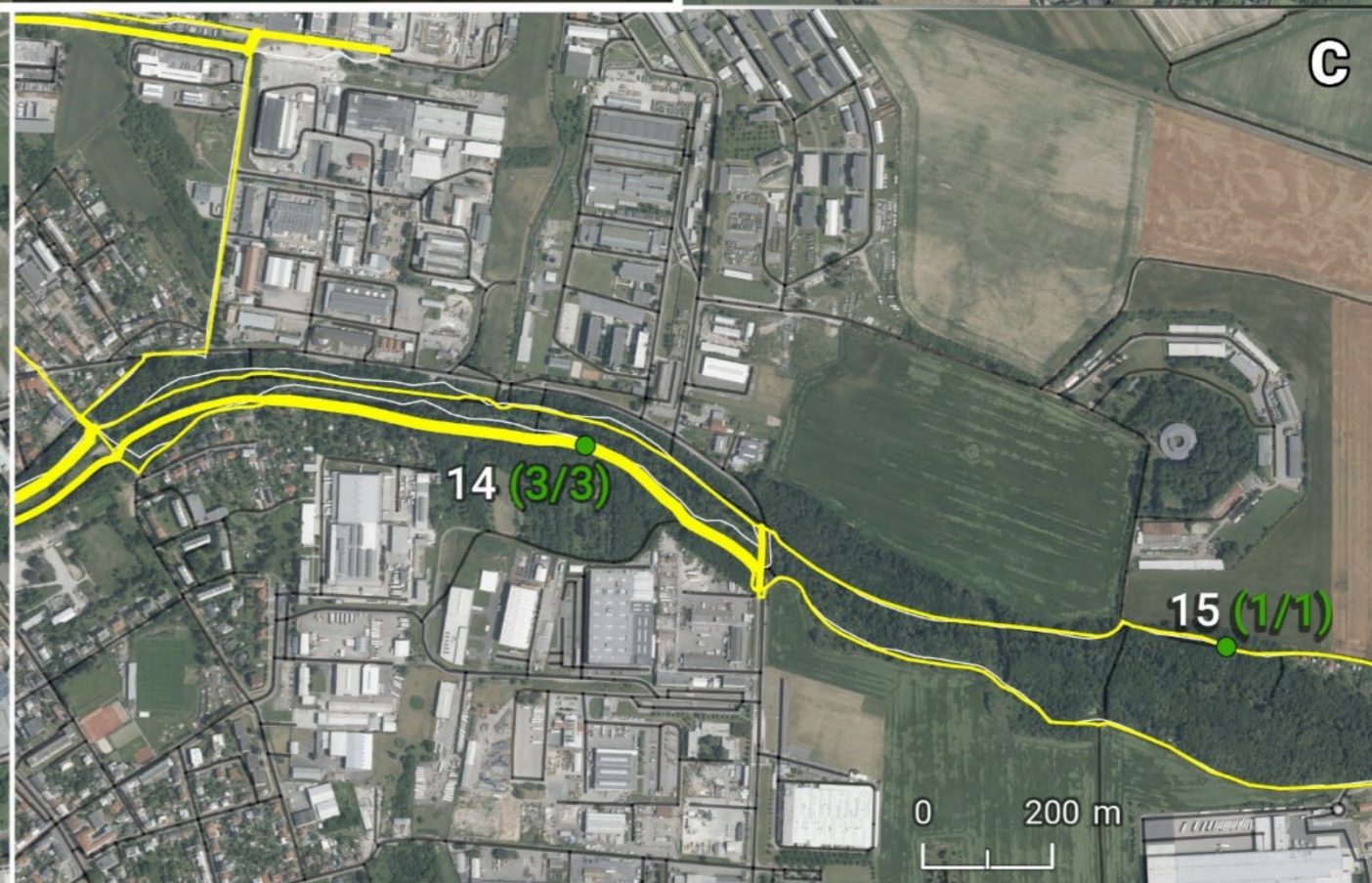
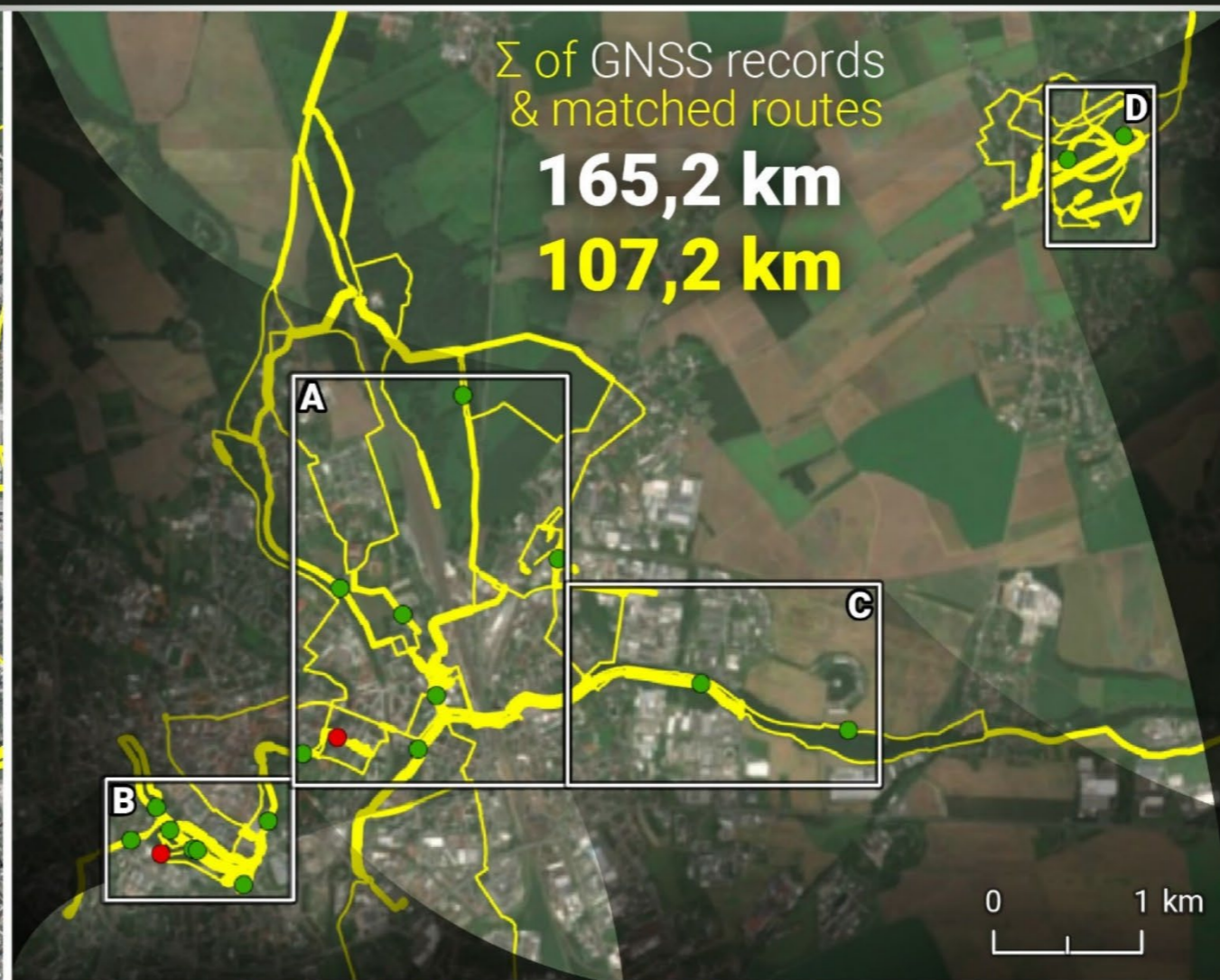
# NOTE: data repo named "data"
for filename in listdir("data"):
    if not filename.endswith(".gpx"):
        continue

    gpx_df = Converter(input_file = "data/" + filename).gpx_to_dataframe()
    try:
        gpx_point = gpd.GeoDataFrame(gpx_df, geometry = gpd.points_from_xy(gpx_df.longitude,
gpx_df.latitude)).set_crs('epsg:4326')
    except:
        print("WARNING: The file has no record of position, therefore, the matching stopped.")
        continue

    gpx_point['id'] = 1
    try:
        gpx_line = gpx_point.groupby(['id']) ['geometry'].apply(lambda x: LineString(x.tolist()))
    except:
        print("WARNING: The file has only one record of position, therefore, the matching stopped.")
        continue

    line_gdf = gpd.GeoDataFrame(gpx_line, geometry = 'geometry').set_crs('epsg:4326')
    latitude = gpx_point["latitude"].iloc[0]
    tolerance = 1 / (111319.488 * np.cos(latitude)) # 1 meter, metric system to degree distance
    line_gdf['geometry'] = line_gdf['geometry'].simplify(tolerance)

    geom = line_gdf.iloc[0,0]
    x,y = geom.coords.xy
    coords = pd.DataFrame({'LAT':y, 'LON':x})
    new_gpx = Converter.dataframe_to_gpx(coords, lats_colname='LAT', longs_colname='LON',
output_file='data/g_' + filename)
    print("New generalised file " + filename + " saved.")
```



CASE STUDY MALÁ FATRA

the attachment 5
to the Master Thesis



GPX2INTENSITY



With the support of the
Erasmus+ Programme
of the European Union



CASE STUDY SLOVAK PARADISE

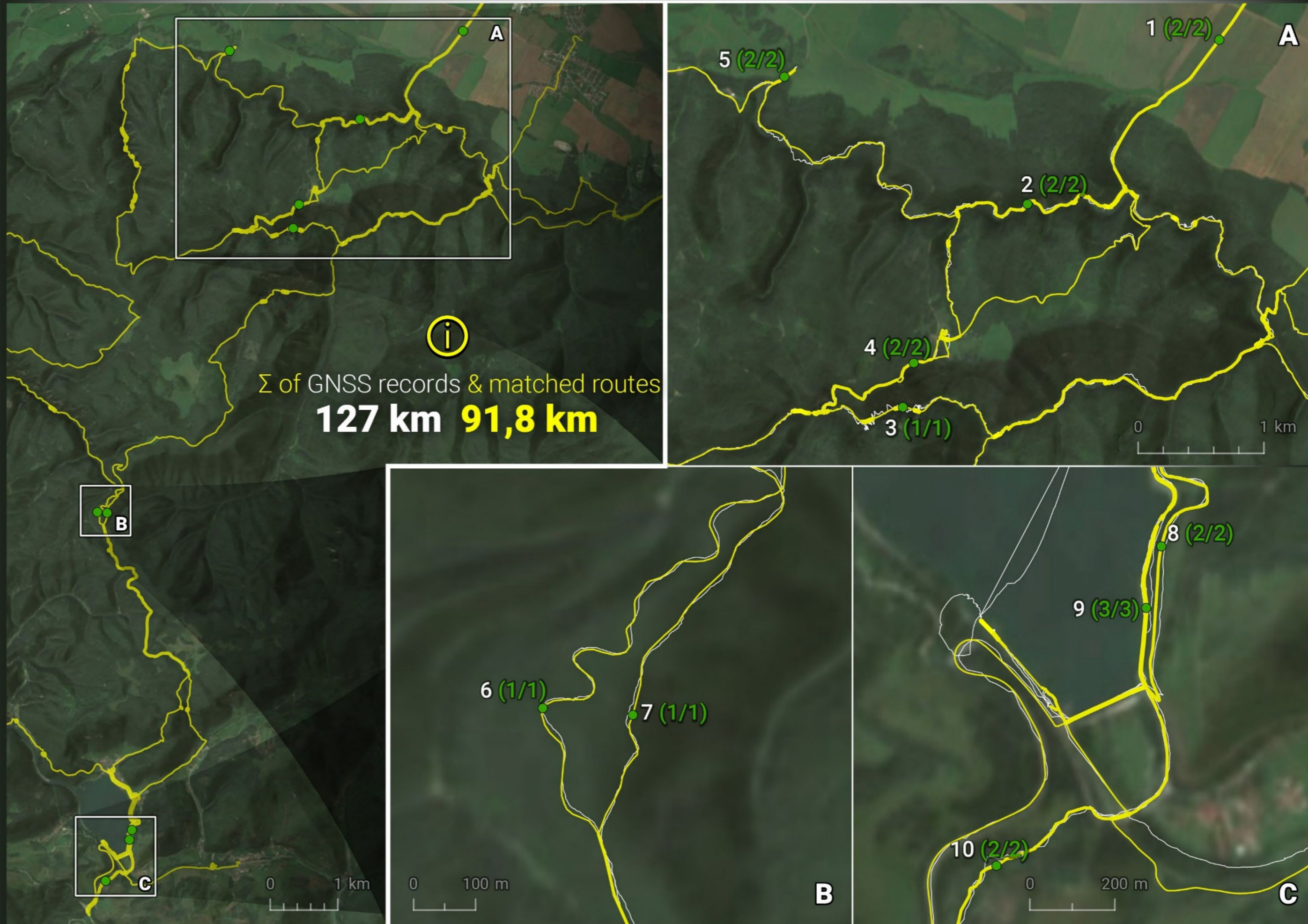
the attachment 6
to the Master Thesis



GPX2INTENSITY



With the support of the
Erasmus+ Programme
of the European Union



satellite imagery:
Sentinel-2 cloud-free
composite (EOX, 2020)
hillshade:
Esri, Intermap, NASA,
NGA, USGS
road network:
OSM contributors