yoGERT GIS Library

Capstone 4G06

System Design Document

Team 19,
Smita Singh, Abeer Alyasiri, Niyatha Rangarajan,
Moksha Srinivasan, Nicholas Lobo, Longwei Ye

April 5, 2023

1 Revision History

Date	Version	Notes
January 10th, 2023	REV0	Updated Sections 2-11, Moksha, Smita, Niyatha
January 18th, 2023	REV0	Revised based on MIS, Moksha, Niyatha
April 3rd, 2023	REV1	Revised based on feedback, Smita

2 Reference Material

2.1 Abbreviations and Acronyms

symbol	description
ALs	Activity Locations are possible stop locations during an episode
CLI	Command Line Interface
CSV	Comma Separated Values is a file type with data separated by commas.
EBA	Episode Behaviour Analysis: Analyzes an episode with the following attributes: transportation mode, speed, direction, duration, distance, and trip trajectory.
GERT	GIS-based episode reconstruction toolkit
GIS	Geographical Information Systems
MD	Mode Detection: Detection of type of transportation being used

Contents

List of Tables

List of Figures

3 Introduction

The following document is a high level system design document for the yoGERT software library. The yoGERT library is a re-implementation of the existing GERT toolbox that aims to aid geographers and the general public in processing and deriving insights from GPS data. For more information regarding this project please refer to the following:

Link to Software Requirements Specification
Link to Hazard Analysis
Link to V&V

4 Purpose

The primary purpose of the system design document is to provide a high level overview of the library's user interface and testing design decisions. Furthermore, it will include a thorough timeline for implementation. For more detailed, lower level design decisions, please refer to the documents below:

Link to Module Guide
Link to Module Interface Specification

5 Scope

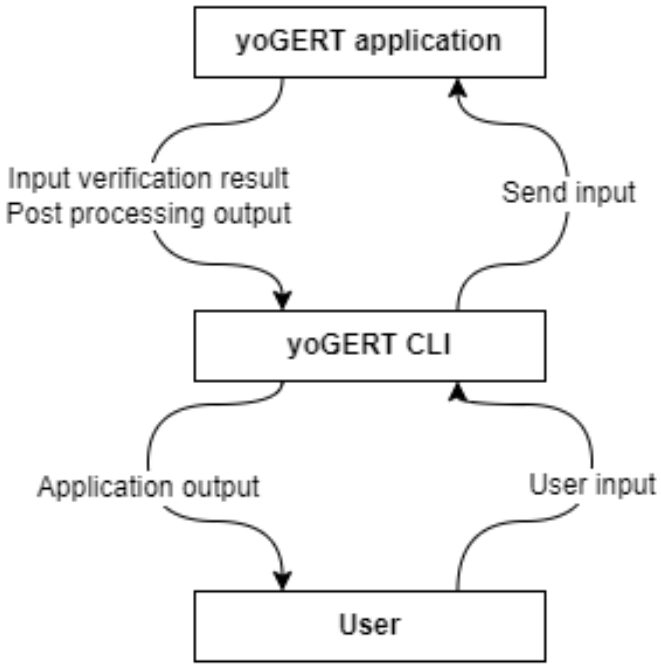

Figure 1: System Context Diagram

6 Project Overview

6.1 Normal Behaviour

Using their command line interface, users will clone the yoGERT repo on their local machine. Then they can navigate to the repository and use the command "pip install . " to install the yoGERT library on their machine. Then users can import their library in any project. For example, if the user would like to generate a travel episode, they can use following import statement in their project: "from yoGERT.EpisodeGeneration import episodeGenerator" Users can call functions from specific modules and pass in the desired input and output paths. They can view output data in two forms, both of which are saved locally on their machine. The first form is csv files that are generated when most modules are run. The second form is html files that are generated after calling the mapping module.

6.2 Undesired Event Handling

A full analysis of the undesired event handling in the system can be found in the Hazard Analysis document.

The main methods to handle undesired events are:

- Robust documentation to help the user avoid any unauthorized or undefined actions
- ~~A startup script to standardize the user's environment before running the toolbox~~
- Descriptive warnings and exceptions provided to the user in the case of invalid commands or inputs
- Saving system state to avoid losing user data

If unexpected behaviour occurs, users will see error messages with recommended courses of action on their CLI.

6.3 Component Diagram

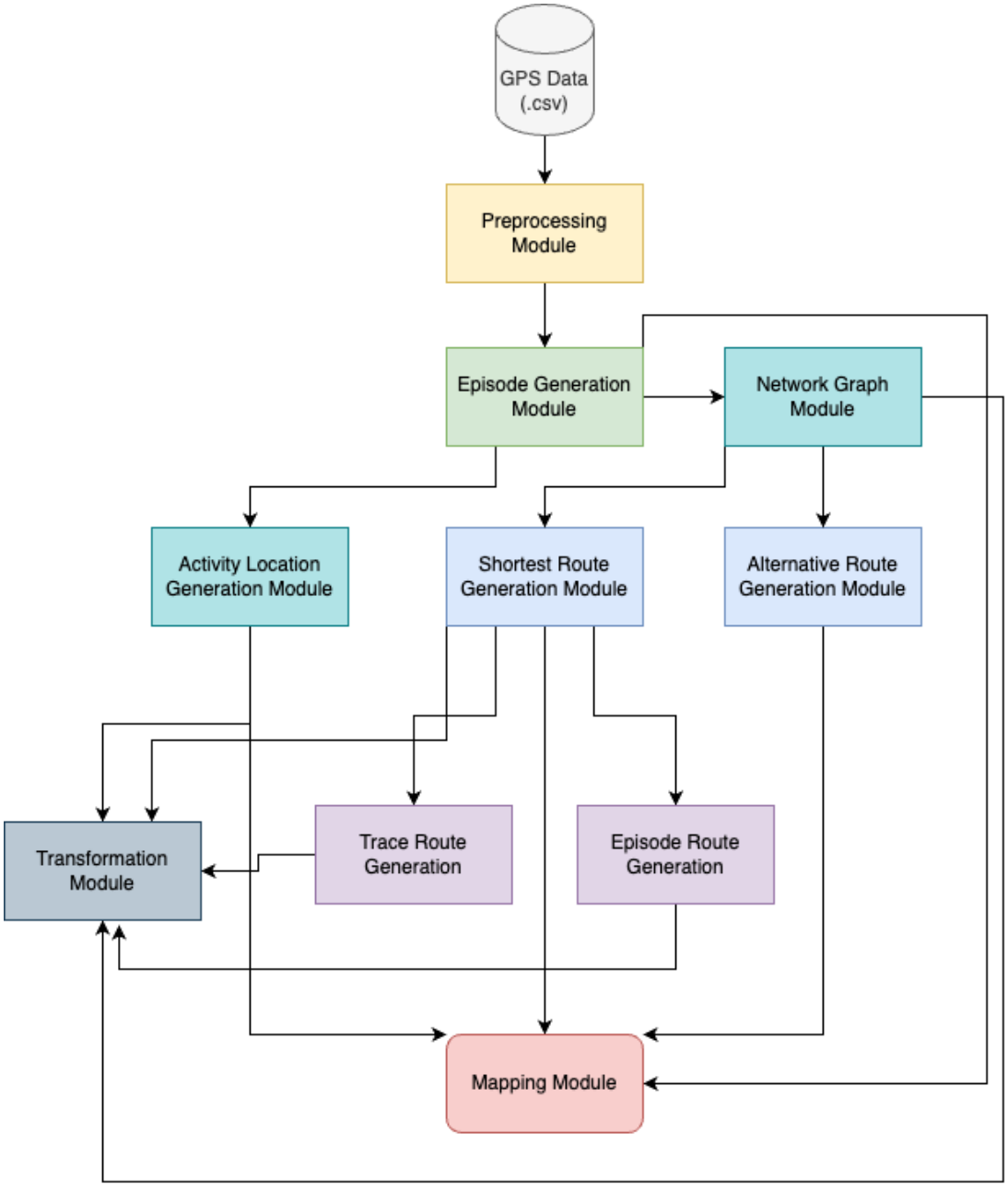

Figure 2: System Component Diagram

6.4 Connection Between Requirements and Design

R1: The system shall allow users to upload GPS data in ~~a standard format~~ CSV file format - We are using a CSV format.

R2: The system shall process GPS data points of multiple traces. - we are creating episode segments that process GPS features through latitude and longitude for each trace in the data

~~R4: The system shall produce output in a standard transferable format.- We are using a CSV format between processing per file.~~

R3: The system shall produce output in a CSV file format and HTML file format for interactive maps - output for each module will be in the form of csv files or html map files

~~R5~~ R4: The system shall use longitude, latitude, and time variables from GPS data points. - we are using a CSV with those exact columns.

~~R6~~ R5: The system shall extract episode attributes including speed, duration, direction, distance,~~ change in direction, acceleration,~~ and status points from GPS data points for each trace - we have an episode generation function to achieve exactly this.

~~R8~~ R6: The system shall classify extracted trace into different types of episodes including stop, ~~car~~ drive, and walk ~~, bus, and other travel episodes~~. - specified an enum type for detecting modes based on the velocity recorded.

~~R9~~ R7: The system shall decompose episode trace into output segments of type stop and trip. - same as above.

~~R10: The system shall identify trip trajectory using extracted segments. - we have used the OSMNx library with Dijkstra's algorithm to match the points of interest with the shortest path taking into account the mode of travel.~~

~~R11~~ R8: The system shall identify activity locations **for each trace** based on the episode attributes ~~of the route to the stop~~ from the start point to the end point. - The episode generation module separates the stops for each traces and the activity location module identifies the activity location.

~~R16: The system shall store activity location identifications - mode generation handles this.~~
~~R18: The system shall allows requests for activity location description by filtering. - This description is possible using the OSMNx library.~~
~~R19: The system shall allow users to request trip segments' description for a given GPS data set. - same as above~~

~~R20: The system shall allow users to request episode descriptions of GPS inputs - same as above.~~

R9: The system shall generate EBA variables based on trip trajectory after the episode segment generation. - This is handled with Episode Generation Module.

R10: The system shall store EBA data set. - Data is stored locally on the users computer at a designated output location.

R11: The system shall automate routes from position A to position B based on EBA set for each extracted trace or episode along with input customization to handle input analysis. - This is handled through the Shortest Route and Mapping Module.

R12: The system shall output alternative routes to the user upon their request of bike automated route. System shall accept customized options that include shortest route by distance and shortest route by time. - This description is implemented handled with the Alternative Route module.

R13: The system shall store activity location descriptions. - The Fetch Activity Location Module stores the descriptions in the Activity Location objects.

R14: The system shall output files for activity location description to the user upon their request. - The Fetch Activity Location Module will generate CSVs that have all the attributes of the Activity Location including description, name, longitude and latitude.

R15: The system shall output csv file for trip segments' description for a given GPS data set to the user upon their request. - Handled by Episode Generation.

R16: The system shall output csv file for episode descriptions of GPS inputs to the user upon their request. - See above.

R17: The system shall identify different traces from GPS data set. - See above.

~~NFR2: The system shall display episodes through informative descriptions - same reasons for the OSMnx library.~~
~~NFR4: The system must be intuitive in terms of its design. - robust documentation in the set up script. As a user you only provide a CSV file and documentation explains how the CSV file will be processed.~~
~~NFR8: The system's functionality must allow the user to track their progress. - The CSV files generated help track the progress per step.~~
~~NFR9: The system must render the require information within 6000 seconds upon request. - the OSMNx library is slow to load but caching the response allows to achieve this constraint.~~

~~NFR10: The system must not make the user's location public. - It is a CLI that only uses local data.~~
~~NFR11: The system must render the route accurately matching the GPS data points provided. - we find the nearest node which validates the GPS trace on the shortest path.~~
~~NFR12: The system must be able to process 47.3 million points of GPS data. - this is possible through caching.~~
~~NFR15: The system shall be able to run on personal laptops and desktops that uses Linux, Windows, and MacOS operating system with Python preinstalled. - achieved since it is dependent on only python libraries.~~
~~NFR16: The system must be functional on Linux, Windows, and MacOS operating systems. - same as above.~~

NFR2: The system shall display informative descriptions on the interactive maps. - Mapping module ensures that all maps generated are interactive html files
NFR3: The system shall display text in font size not less than 10pt.- Mapping module sets the display text font.

NFR4: The system must be intuitive in terms of its design. - robust documentation in the set up script. As a user you only provide a CSV file and documentation explains how the CSV file will be processed.

NFR15: The system must be functional on Linux, Windows, and MacOS operating systems. - All the modules are designed to be implemented on Python which runs on all operating systems.

NFR16: The data they've uploaded system must allow users to have access to read and modify they've uploaded All outputs generated by each module is easily accessible to the user.

NFR17: The system shall allow access to all system services and data outputs. - See above.

7 System Variables

N/A

7.1 Monitored Variables

7.2 Controlled Variables

7.3 Constants Variables

8 User Interfaces

The user will ~~interface with the GERT toolbox through the use of their command line tool of choice (terminal, powershell, etc)~~. interact with the library inside their own proejects. This decision was partially driven by one of the main goals of the project, being able to process large inputs in a short amount of time. A library is more fitting as users have more flexibilty on how they want to process their data and which modules they want to use. Although a GUI would allow a wider range of users to interact with the system, the team simply does not have the time to implement a robust GUI. Lastly, conversing with the project supervisor, Dr. Paez, uncovered that many of the primary users (academics in the geography field) already have familiarity working with libraries.

To ensure a seamless user experience, the team will document all possible user inputs with examples to follow along with.

9 Design of Hardware

N/A

10 Design of Electrical Components

N/A

11 Design of Communication Protocols

The nature of our project did not facilitate the creation or design of a communication protocol as it is mainly a data transformation project. However, we did make some choices with regard to communication to minimize the likelihood of unexpected behaviour. We chose to use a wrapper for the OSM API called "overpy" rather than the API itself because we only require GETting information and never *update* the open source database. This avoids

accidental updates of the OSM database (although there is a thorough review process), especially when users try to manipulate the code.

On the whole, our system follows a "pipe and filter" architecture style. This means that modules receive input from a data source, transform the data, and output the transformation. Although modules do interact, we ensured that large amounts of information are not being relayed through the modules. Instead, we store intermediate data sources for the user to peruse.

12 Timeline

Module	**Main Developer(s)**	**Due Date**
Input Pre-processing	Moksha Srinivasan	January 24th, 2023
Episode Generation	Nicholas Lobo	January 24th, 2023
Data Transformation	Niyatha Rangarajan	January 25th, 2023
Mapping Module	Abeer Alyasiri	January 28th, 2023
Alternative Route	Abeer Al-Yasiri	January 29th, 2023
Network Graph	Abeer Al-Yasiri	January 29th, 2023
Route Choice Generation	Abeer Al-Yasiri	January 29th, 2023
Activity Location Detection	Smita Singh	January 30th, 2023
Travel Mode Detection	Nicholas Lobo	January 30th, 2023
User Documentation	Longwei Ye	January 30th, 2023
Setup Script	Moksha Srinivasan	January 31st, 2023
~~Main Module~~	~~Moksha Srinivasan~~	~~January 31st, 2023~~
Final Peer Code Reviews*	All	January 31st, 2023
Integration Testing	Longwei Ye	February 1st, 2023
Benchmark/Stress Testing	Longwei Ye	February 2nd, 2023

Table 1: Module Timeline

Before merging code, all team members will have written unit tests for their respective modules with mocked versions of the input ensuring correct module functionality. All components should be completed and tested by 1 week before the Final Demo on February 9th and changes made afterward will be for cleanup.

*Note: Although code reviews have been taking place throughout the implementation process, the team plans to come together and review code once again before the final demonstra-

tion. This ensure that code and documentation is all up to date and in line with changing requirements.

A Interface

No additional information

B Mechanical Hardware

N/A

C Electrical Components

N/A

D Communication Protocols

No additional information

E Reflection

One limitation of yoGERT is the use of the OSM database to retrieve data regarding transportation networks, activity locations, and general up-to-date GPS data. Since the dataset is crowd-sourced, it is not updated as often as say, Google Maps and the toolbox with subsequently have inaccuracies. With unlimited resources, the yoGERT team would aggregate data from alternative sources such as the most recently updated local transportation maps to increase accuracy.

Furthermore, another limitation of the solution provided is the CLI interface. With more time and resources the team would be keen to create a front-end interface that is accessible to a wide variety of users. This is because a CLI tool can have a very high barrier to entry for non-technically oriented stakeholders. Furthermore, a GUI tool could more easily help students and those keen to learn about geoprocessing visualize the output.

Finally, one big limitation in our design of yoGERT is the inability to append to existing inputs and outputs. Since the system is batch sequential, in the current implementation it is not able to quickly update inputs and add new episodes. This functionality is achievable and very useful, but would require an overhaul of the current design to succeed.

Another design that the team explored was embedding the application into a Jupyter notebook. This aided in the mission of yoGERT being a learning tool, with full code transparency and markdown comments throughout. This allowed the user to understand exactly what is occuring as they run the code and gain an intuitive understanding of how to process and

manipulate GPS data. However, the main drawback of this approach is that Jupyter notebook can be slow to start up and take a long time to execute code. One of the main goals of the toolbox is the ability to quickly process large amounts of data which is difficult using Jupyter notebook. The developers chose to stick with a CLI interface as it is significantly faster and those who are interested can still look through the documentation and manipulate the code accordingly.

Lastly, the team also explored using R to implement the toolbox. This was an idea suggested by the team's supervisor due to the language being designed for large scale data manipulation. The benefits of R would have been the ability to complete deep statistical analysis which is very useful for route choice analysis and mode detection. The disadvantages are that R has significantly fewer libraries for our use case (geography) and visualizations tend to be more robust in python. On the contrary, python is less efficient for deep statistical analysis but has significantly more libraries for our use case, and it is easier to provide interactive data visualizations.