Module Interface Specification for Software Engineering

Team 17, Track a Trace
Zabrain Ali
Linqi Jiang
Jasper Leung
Mike Li
Mengtong Shi
Hongzhao Tan

January 18, 2023

1 Revision History

Date	Version	Notes
January 16, 2023	1.0	Edited Abbreviations and Acronyms, Introduction and Notation
January 17, 2023	1.1	Edited Module Decomposition
January 18, 2023	1.2	Edited MIS of all Modules

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/paezha/PyERT-BLACK/tree/main/docs/SRS.

symbol	description
EPSG	European Petroleum Survey Group
CRS	Coordinate-Reference System
CSV	Comma-Separated Values
MG	Module Guide
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
OSM	OpenStreetMap: a free, open geographic database
PBF format	Protocolbuffer Binary Format
GDB file	Geodatabase: a collection of files in a folder on disc that hold related geospatial data
2D	two-dimensional
SHP	shapefile: a geospatial vector data format for GIS software
URL	Uniform Resource Locator

Contents

3 Introduction

The following document details the Module Interface Specifications for the project PyERT. The project aims to re-implement the functionalities of GERT (Dalumpines and Scott, 2018) which uses ArcGIS Pro packages, with open-source packages and libraries, and remove any use of ArcGIS in GERT. The same as the original GERT toolkit, the purpose of the product system is to match GPS trajectories to transportation network for further analysis to the GPS data.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/paezha/PyERT-BLACK.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	Notation	Description
character	char	A single symbol or digit
integer	$\mathbb{Z}$	A number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	A number without a fractional component in [1, ∞)
real	$\mathbb{R}$	Any number in (-∞, ∞)
DataFrame	df	Tuple that the value for each field is a sequence. Sequences of differet fields could be of different data types but their lengths will be the same.
GeoDataFrame	gdf	A special type of df where there will be always a field of 'geometry' which is a sequence of geometric data type (e.g. Point, LineString, etc.)
Point	point	A geometric data type that describes a point, which will be a sequence of $\mathbb{R}$, the length of the sequence will be always 2
LineString	ls	A geometric data type that describes a line segment, which will be a sequence of sequences of $\mathbb{R}$, the length of the inner sequences will be always 2
Polygon	polygon	A geometric data type that describes a shape, will be a sequence of of sequences of $\mathbb{R}$, where the first and last inner sequences are the same (connecting to make a closed shape), the length of the inner sequences will be always 2

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	GPS Data Preprocessing Module GPS Data Mode Detection Module Trip Segments and Activity Locations Extraction Module Route Choice Set Generator Module Route Choice Analysis Variables Generator Module Activity Locations Identification Module Main Function Module
Software Decision Module	Dataframe Data Structure Module GeoDataframe Data Structure Module Geometric Object Analysis and Manipulation Module Network Analysis Module OSM Network Dataset Reader Module Plotting Module

Table 1: Module Hierarchy

6 MIS of GPS Data Preprocessing Module

6.1 Module

GPSPreprocess

6.2 Uses

N/A

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
GPSPreprocess	data: df of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, latitude: seq of $\mathbb{R}$, longitude: seq of $\mathbb{R}$, Speed_kmh: seq of $\mathbb{R}$)		
getData		processedData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of point)	

Routine name	In	Out	Exceptions
filterData	data: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of point)		
smoothData	data: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of point)		

6.4 Semantics

6.4.1 State Variables

processedData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of point)

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

GPSPreprocess(data):

- transition: filters and smooths given DataFrame and converts it to a GeoDataFrame
- output: $out := self$
- exception: None

getData():

- output: $out := preprocessedData$
- exception: None

filterData(data):

- transition: $data := (\forall x, y : data | x \neq y)$
- exception: None

smoothData(data):

- transition: $data := (\forall row : data | row.speed < 50)$
- exception: None

6.4.5 Local Functions

None

7 MIS of GPS Data Mode Detection Module

7.1 Module

ModeDetection

7.2 Uses

N/A

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
detectModes	processedData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of point)		
getEpisodeData		episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, Mode: seq of String, geometry: seq of point)	

7.4 Semantics

7.4.1 State Variables

episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, Mode: seq of String, geometry: seq of point

7.4.2 Environment Variables

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

detectModes(processedData):

- transition: Read a processedData GeoDataFrame produced by the GPSPreprocsser module, then partition the given points into segments, then classify these episodes by mode (walk, car, bus, etc.). Store these episodes into the episodeData GeoDataFrame (a state variable).
- exception: None

getEpisodeData():

- output: A GeoDataFrame object containing the classified GPS Episodes. It consists of recordID (represented by a sequence of integers), serialID (represented by a sequence of integers), LocalTime (represented by a sequence of strings), Speed_kmh (represented by a sequence of real numbers), Mode(represented by a sequence of Strings), and geometry (represented by a sequence of Points).
- exception: None

7.4.5 Local Functions

None

8 MIS of Trip Segments and Activity Locations Extraction Module

8.1 Module

Extractor

8.2 Uses

N/A

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
extractTripSegments	episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of String, geometry: seq of point)		
getTripSegments		tripSegmentsData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of point)	

Routine name	In	Out	Exceptions
extractActivityLocations	episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, Mode: seq of String, geometry: seq of point)		
getActivityLocations		extractedData : gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of point)	

8.4 Semantics

8.4.1 State Variables

tripSegments: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of point
activityLocations: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of point)

8.4.2 Environment Variables

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

extractTripSegments(episodeData):

- transition: Read the episodeData GeoDataFrame generated by the ModeDetection module, and extract trip segments (sequences of data points in a travel episode) from this GeoDataFrame. Store these trip segments into the tripSegments GeoDataFrame (a state variable).
- exception: None

getTripSegments():

- output: A GeoDataFrame object containing the trip segments data. It consists of recordID (represented by a sequence of integers), serialID (represented by a sequence of integers), LocalTime (represented by a sequence of strings), Speed_kmh (represented by a sequence of real numbers), Mode(represented by a sequence of Strings), and geometry (represented by a sequence of Points).

- exception: None

extractActivityLocations(episodeData):

- transition: Read the episodeData GeoDataFrame generated by the ModeDetection module, and extract activity locations (stops or endpoints of trip segments) from this GeoDataFrame. Store these trip segments into the tripSegments GeoDataFrame (a state variable).

- exception: None

getActivityLocations():

- output: A GeoDataFrame object containing the activity location data. It consists of recordID (represented by a sequence of integers), serialID (represented by a sequence of integers), LocalTime (represented by a sequence of strings), Speed_kmh (represented by a sequence of real numbers), Mode(represented by a sequence of Strings), and geometry (represented by a sequence of Points).

- exception: None

8.4.5 Local Functions

None

9 MIS of Route Choice Set Generator Module

9.1 Module

RCSGenerator

9.2 Uses

N/A

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
mapPointToNetwork	networkData: gdf of (streetName: sequence of string, geometry: sequence of ls) points: sequence of point	pointOnNet: gdf of (streetGeometry: sequence of ls, streetName: sequence of string, geometry: sequence of point)	-
detectGap	points: sequence of point	gapPoints: sequence of sequence of point	-
fillInGap	gapPoints: sequence of sequence of point	filledGap: sequence of ls	-
findStreetIntersect	allExistStreets: sequence of ls	intersections: gdf of (street1Geo: sequence of ls, street2Geo: sequence of ls, geometry: sequence of point)	-

Name	In	Out	Exceptions
connectPoints	pointOnNet: gdf of (streetGeometry: sequence of ls, streetName: sequence of string, geometry: sequence of point]) intersections: gdf of (street1Geo: sequence of ls, street2Geo: sequence of ls, geometry: sequence of point) filledGap: sequence of ls	route: ls	-
RouteChoiceGen	trip: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{N}$, TimeStart: sequence of string, Mode: sequence of string, geometry: sequence of point) networkData: gdf of (streetName: sequence of string, geometry: sequence of ls)	routeChoice: gdf of (SerialID: sequence of $\mathbb{N}$, geometry: sequence of ls)	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

mapPointToNetwork(networkData, points):

- output: a GeoDataframe object with the input points mapped onto the input network. It consists of streetGeometry (represented by a sequence of LineString), streetName (represented by a sequence of string), and geometry (represented by a sequence of Point).
- exception: none

detectGap(points):

- output: a sequence of all the gap points among the given points, which is a sequence of sequences [2] of point objects.
- exception: none

fillInGap(gapPoints):

- output: a sequence of LineString objects that represent all the filled gaps given the gap points.
- exception: none

findStreetIntersect(allExistStreets):

- output: a GeoDataframe object consists of street1Geo (represented by a sequence of LineString), street2Geo (represented by a sequence of LineString), and geometry (represented by a sequence of Point), where the Point in geometry is an intersection point between the LineStrings in street1Geo and street2Geo.
- exception: none

connectPoints(pointOnNet, intersections, filledGap):

- output: a LineString object which connects all of the Point objects in pointOnNet, any Point object in intersections that involves, and the LineString objects in filledGap.
- exception: none

RouteChoiceGen(trip, networkData):

- output: a GeoDataframe object consists of SerialID (represented by a sequence of $\mathbb{N}$), and geometry (represented by a sequence of LineString). The GeoDataframe object will be generated by using mapPointToNetwork, detectGap, fillInGap, findStreetIntersect and connectPoints fucntions
- exception: none

9.4.5 Local Functions

None

10 MIS of Route Choice Analysis Variables Generator Module

10.1 Module

RCAVarGenerator

10.2 Uses

N/A

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
routelength	route: sequence of sequence[2] of $\mathbb{R}$	length: $\mathbb{R}$	-
countTurns	route: sequence of sequence[2] of $\mathbb{R}$	numOfTurnsByType: tuple of (left : $\mathbb{N}$, right : $\mathbb{N}$,total : $\mathbb{N}$)	-
mapLegToStreet	route: sequence of sequence[2] of $\mathbb{R}$ networkData: gdf of (Name: sequence of *string*, geometry: sequence of ls)	streetOnRoute: gdf of (streetName: sequence of *string*, geometry: sequence of ls), numOfRoad : $\mathbb{N}$	-
longestLeg	streetOnRoute: gdf of (streetName: sequence of *string*, geometry: sequence of ls)	longestLegInfo:tuple of (legStreet : *string* , leglength : $\mathbb{R}$)	-

Name	In	Out	Exceptions
RCAVarGen	route: (SerialID: sequence of $\mathbb{N}$, geometry: sequence of ls) networkData: gdf of (streetName: sequence of string, geometry: sequence of ls)	RCA:(SerialID: sequence of $\mathbb{N}$, distancemeter: sequence of $\mathbb{R}$, numOfRoads: sequence of $\mathbb{N}$, numOflturns: sequence of $\mathbb{N}$, numOfrturns: sequence of $\mathbb{N}$, streetLongestLeg: sequence of string, lengthLongestLeg: sequence of $\mathbb{R}$, geometry: sequence of ls)	

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

routelength(route):

- output: An $\mathbb{N}$ that represents the length of the input route in meters.
- exception: none

countTurns(route):

- output: a tuple of (left : $\mathbb{N}$, right : $\mathbb{N}$,total : $\mathbb{N}$) that represents the number of left turns, right turns and total turns of the input route.
- exception: none

mapLegToStreet(networkData, route):

- output: streetOnRoute := a GeoDataframe object consists of streetName (represented by a sequence of string) and geometry (represented by a sequence of LineStrings. numOfRoad := An $\mathbb{N}$ that counts the number of unique steets the input route has been on.
- exception: none

longestLeg(streetOnRoute):

- output: a tuple of (legStreet : string , legLength : $\mathbb{R}$) that contains the streetName that the longest leg belongs to, length of the longest leg in meters.
- exception: none

RCAVarGen(networkData, route):

- output: a GeoDataframe object consists of SerialID (represented by a sequence of $\mathbb{N}$), and geometry (represented by a sequence of LineString), distance of the route (represented by sequence of $\mathbb{R}$), number of roads(represented by sequence of $\mathbb{R}$), number of turns (represented by sequence of $\mathbb{R}$), streetLongestLeg (represented by sequence of string), length of the longest leg (represented by $\mathbb{R}$). The GeoDataframe object will be generated by using routelength, countTurns, mapLegToStreet and longestLeg functions.

10.4.5 Local Functions

None

11 MIS of Activity Locations Identification Module

11.1 Module

ALIM

11.2 Uses

N/A

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
identifyLU	AL_gdf: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of point), LU_gdf: gdf of (house_number: sequence of $\mathbb{N}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of polygon, lu_code: sequence of $\mathbb{N}$, lu_classification: sequence of string)	ALlu_gdf: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of point, house_number: sequence of $\mathbb{N}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of polygon, lu_match: sequence of string, lu_code: sequence of $\mathbb{N}$, lu_classification: sequence of string)	

identifyPAL	AL_gdf: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of point), PAL_gdf: gdf of (house_number: sequence of $\mathbb{N}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of polygon, pal_id: sequence of $\mathbb{N}$, pal_classification: sequence of string)	ALpal_gdf: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of point, house_number: sequence of $\mathbb{N}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of polygon, pal_match: sequence of string, pal_id: sequence of $\mathbb{N}$, pal_classification: sequence of string)	

createActLocInfo	ALlu_gdf: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of point, house_number: sequence of $\mathbb{N}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of polygon, lu_match: sequence of string, lu_code: sequence of $\mathbb{N}$, lu_classification: sequence of string) ALpal_gdf: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of point, house_number: sequence of $\mathbb{N}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of polygon, pal_match: sequence of string, pal_id: sequence of $\mathbb{N}$, pal_classification: sequence of string)	ALinfo_gdf: gdf of (SerialID: sequence of $\mathbb{N}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of point, house_number: sequence of $\mathbb{N}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of polygon, lu_match: sequence of string, lu_code: sequence of $\mathbb{N}$, lu_classification: sequence of string, pal_match: sequence of string, pal_id: sequence of $\mathbb{N}$, pal_classification: sequence of string)	

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

identifyLU(AL_gdf, LU_gdf):

- output: A GeoDataFrame of Activity Locations with appending information to the object based on the inputted LU GeoDataFrame that corresponds with an existing SerialID of Activity Locations if it exists. This includes the lu_match which states if the SerialID matches the LU, lu_code and lu_classification:
- exception: None

identifyPAL(AL_gdf, PAL_gdf):

- output: A GeoDataFrame of Activity Locations with appending information to the object based on the inputted PAL GeoDataFrame that corresponds with an existing SerialID of Activity Locations if it exists. This includes pal_match which states if the SerialID matches the PAL, pal_id and pal_classification
- exception: None

createActLocInfo(ALlu_gdf, ALpal_gdf):

- output: A GeoDataFrame of Activity Location Information with appending information to the object based on the inputted identified Activity Location GeoDataFrame with LU additional information and identified Activity Location GeoDataFrame with PAL additional information
- exception: None

11.4.5 Local Functions

None

12 MIS of Main Function Module

12.1 Module

Main

12.2 Uses

GPSPreprocess (Section 6), ModeDetection (Section 7), Extractor (Section 8), RCSGenerator (Section 9), RCAVarGenerator (Section 10), ALIM (Section 11)

12.3 Syntax

12.3.1 Exported Constants

EPSGNumEarth: $\mathbb{N}$

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	FildNotFoundError, FileTypeError, InvalidEPSGNum

12.4 Semantics

12.4.1 State Variables

programProgress: $\mathbb{R}$

12.4.2 Environment Variables

consoleWin: 2D sequence of pixels displayed on the screen
gpsDataFile: A text file in CSV format
networkDatasetFile: A file/folder that contains the data for the transportation network dataset
outputFolder: A virtual location in computer for files or other folders/directories

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

main():

- transition: Modify consoleWin to display text prompts asking user to input the file paths for the gpsDataFile, networkDatasetFile and outputFolder, asking user to input the EPSG CRS number ($\mathbb{N}$), informing the user which stage of map matching the running program is currently on, showing error/warning to user if exception appears, display the current progress based on the programProgress state variable, and displaying URLs that user can copy to visualize the map-matching result on browser.
  Modify outputFolder to generate files that the program will output under the file path of outputFolder.

- output: None

- exception: exc := a file path gpsDataFile OR networkDatasetFile cannot be found $\Rightarrow$ FileNotFoundError
  exc := the format of gpsDataFile OR networkDatasetFile is incorrect $\Rightarrow$ FileTypeError
  exc := the EPSG CRS number from user input cannot be used $\Rightarrow$ InvalidEPSGNum

12.4.5 Local Functions

None

References

Ron Dalumpines and Darren M. Scott. Gis-based episode reconstruction toolkit (gert): A transferable, modular, and scalable framework for automated extraction of activity episodes from gps data. *Travel Behaviour and Society*, 11:121–130, 2018. ISSN 2214-367X. doi: https://doi.org/10.1016/j.tbs.2017.04.001. URL https://www.sciencedirect.com/science/article/pii/S2214367X17300406.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

13 Appendix

N/A