GPS Probe Generation

This program reads a csv file which contains GPS points along the road. Sometimes, such data may be sparse and smooth visualization along the road may not be guaranteed. This program generates intermediate GPS points along the road using OpenStreetMap data. The sample csv file (gps_probe_Nepal.csv) is provided for demo purpose. The csv file assumes four fields ap_id (i.e. gPS_device ID), timestamp, lat (i.e. latitude) and lon (i.e. longitude). Based on the sequence of latitude and longitude points and the timestamps, intermediate points and corresponding timestamp are generated for visualization along the road.

The program works for generating routes that can be visualized in Mobmap. However, this code is an initial version which have to be improved for differnt aspects. This is published here in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Main Modules used:

- Postgresql (version 10.12) with PostGIS (version 2.4.8) extension
  - https://www.postgresql.org/download/
  - Copy Command

- shp2pgsql command
  - Windows: http://download.osgeo.org/postgis/windows
  - Ubuntu: https://manpages.ubuntu.com/manpages/bionic/man1/shp2pgsql.1.html

- Python 3
  - https://www.python.org/downloads/
- pyroutelib3 for generating OSM routes
  - https://pypi.org/project/pyroutelib3/
- pandas for working with data
  - https://pypi.org/project/pandas/
- psycopg2 for connecting with postgresql
  - https://pypi.org/project/psycopg2/
- tkinter for GUI
  - https://pypi.org/project/tkintertable/
- Shapely for Geometric Objects
  - https://pypi.org/project/Shapely/
- Faker for Data anonymization
  - https://pypi.org/project/Faker/
- GraphHopper Map-matching tool
  - https://github.com/graphhopper/map-matching

Note: Postgresql is not necessary if you do not wish to clip data to target geography.

1. Edit config.py

```
# EDit your Credentials to access Postgis Database

##-------------Database User Credentials
dbuser      = "postgres"    # Update with yoour database user name
dbpassword  = "postgres123" # Update with yoour database user passowrd
database    = "mobility"
host        = "localhost"


##------------- local OSM data file OR online
'''
OSM data is accessed online by pyrouteLib3. For local data option
comment this line and uncomment the following line.
'''
osm_data_source = ""

#osm_data_source = "pokhara_roads.osm"  # downloaded from OpenStreetMap
#osm data location For offline processing


##------------- Limit Geographic Boundary of Data
# if NOT needed then give empty values

shp_table_name      = 'gadm36_NPL_2'
column_name         = 'NAME_2'
column_name_value   = 'Gandaki'
'''
shp_table_name      = ''
column_name         = ''
column_name_value   = ''
'''
```

Figure 1: Config File

2. **C**reate database and enable POSTGIS extension. (**Optional**)

- Create Database ‘mobility’

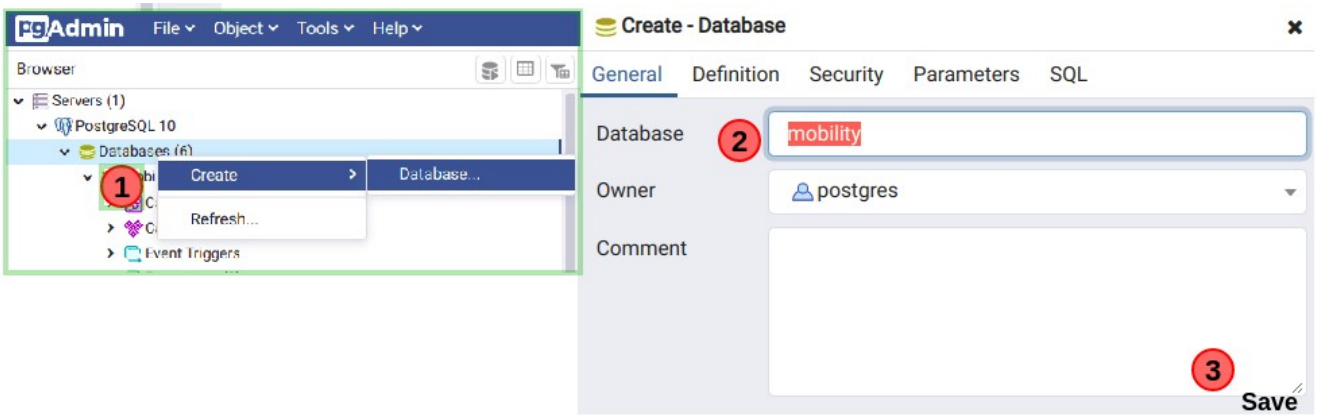

Figure 2: Create Database

- Enable PostGIS extension, by running the following queries:

```
CREATE EXTENSION postgis;
SELECT postgis_full_version();
```

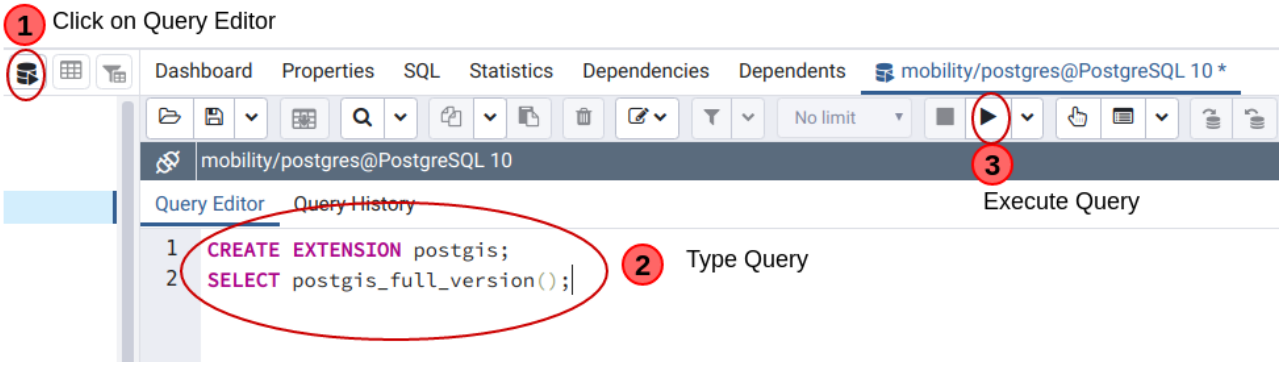

Figure 3: Enable postgis extension

3. Align GPS probe data along the road network

1. If Java and Maven are not already installed then install them. Java 8 and Maven >=3.3.
   - **JDK** (https://www.java.com/en/download/)
   - **OpenJDK** (https://jdk.java.net/archive/)
   - https://mkyong.com/maven/how-to-install-maven-in-windows/

NOTE: Install JDK or OpenJDK, any of them works.

2. Download and install GraphHopper Map-matching tool

2.1 Download map-matching to **GPSProbeGeneration** directory (with option 1 **OR** option 2)

- **Option 1:** Download: https://github.com/graphhopper/map-matching and copy the code in the appropirate directory as shown in the image below.

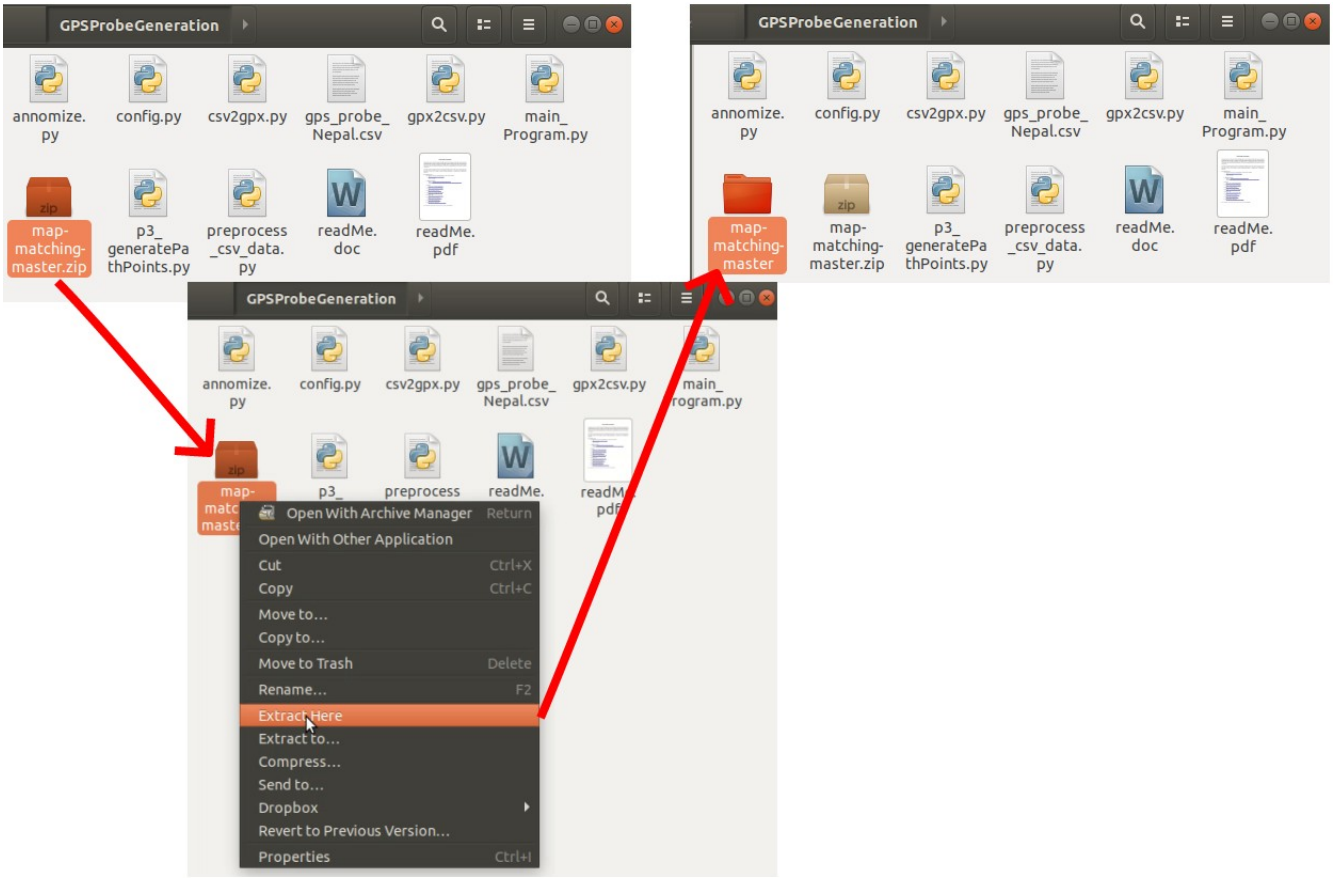

Fig 4: download map-matching

- **Option 2:** Use git clone comman:
  - Change directory to <GPSProbeGeneration>
  - git clone https://github.com/graphhopper/map-matching.git

Fig 5: Git clone command to download map-matching

(Make sure **map-matching-master** *is inside* **GPSProbeGeneration** *as shown in the image)*

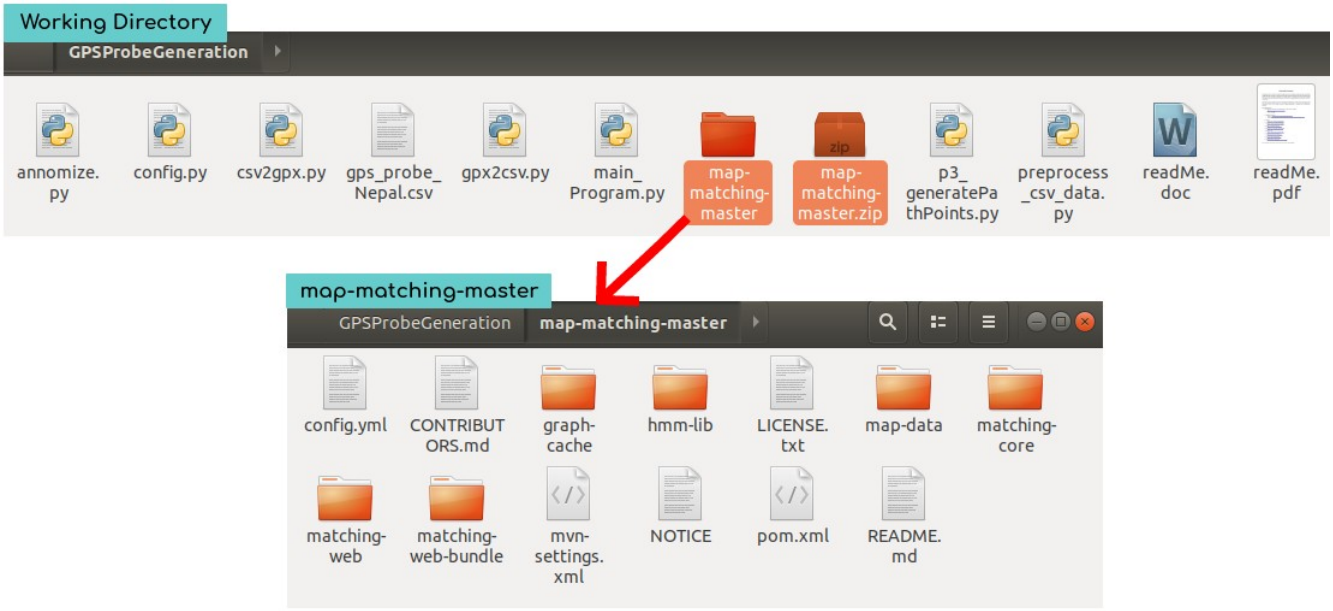

Fig 6: Directory path and contents after download

2.2 Build Graphhopper

- Change directory to <map-matching-master>
- Command: `mvn package -DskipTests`

```
GPSProbeGeneration/map-matching-master$ mvn package -DskipTests
```

Fig 7: Build Command

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for GraphHopper Map Matching Parent Project 1.0-SNAPSHOT:
[INFO]
[INFO] GraphHopper Map Matching Parent Project ..... SUCCESS [  0.002 s]
[INFO] hmm-lib ..................................... SUCCESS [  3.658 s]
[INFO] GraphHopper Map Matching .................... SUCCESS [  1.099 s]
[INFO] GraphHopper Map Matching Dropwizard Bundle ..... SUCCESS [  1.533 s]
[INFO] GraphHopper Map Matching Web ................... SUCCESS [  5.984 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  12.858 s
[INFO] Finished at: 2020-06-15T17:25:18+05:45
[INFO] ------------------------------------------------------------------------
```

Fig 8: Successful Built

3. Import OSM map for Nepal

- Download: https://download.geofabrik.de/asia/nepal-latest.osm.pbf
- Copy to map-data directory.
- Update config.py with the name of the map file **target_osm_pbf = nepal-latest.osm.pbf**
- NOTE: make sure the GPS Probe is within the map region, otherwise map-matching may encounter exceptions.

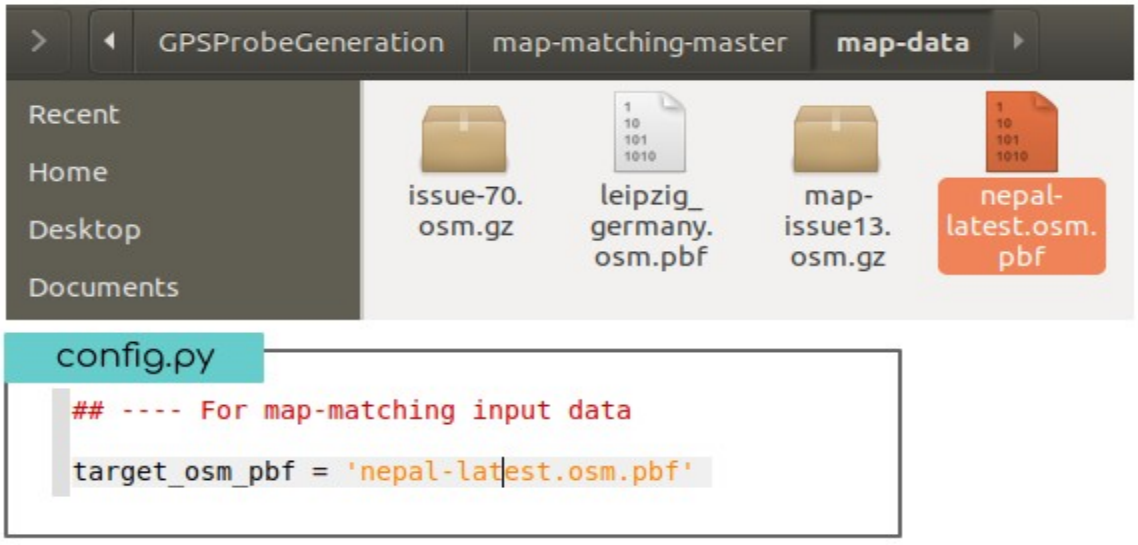

Fig 9: Download and Save nepal-latest.osm.pbf

4. Run main_program.py (Online OSM data download)

(`python3 main_program.py` OR `python main_program.py`)

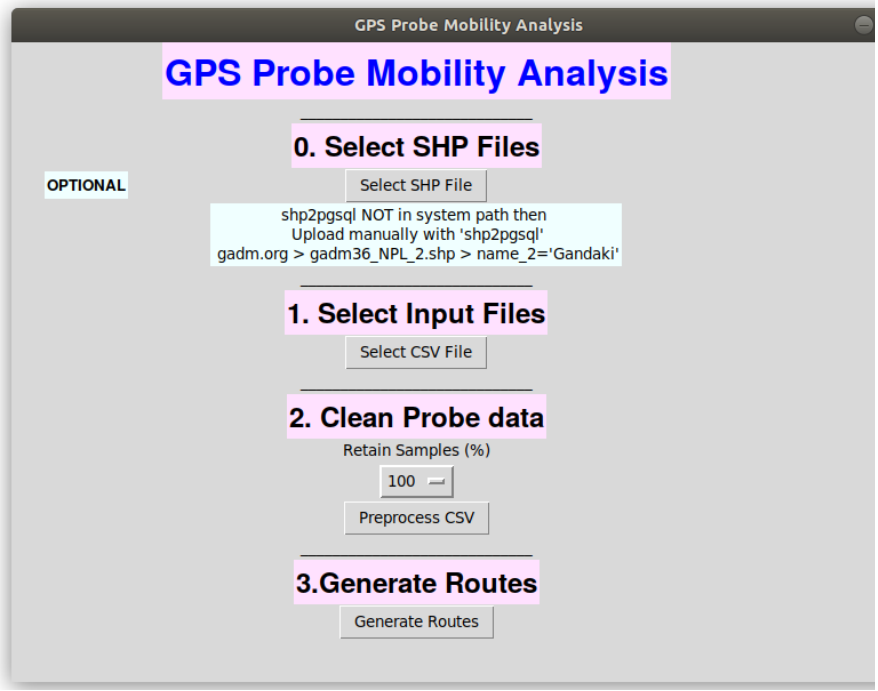

Figure 10: Program Interface (at start)

4.1 Select Input Files:

- Shapefile Upload (**Optional**)
  - Get Nepal shapefile from <https://biogeo.ucdavis.edu/data/gadm3.6/shp/gadm36_NPL_shp.zip> and select 'gadm36_NPL_3.shp'
  - If needed use shp2pgsql program to upload it to database. Command line can be used as follows:

```
shp2pgsql -I -s 4326 gadm36_NPL_3.shp  gadm36_NPL_3 |
PGPASSWORD=postgres123 psql -d mobility -h localhost -U postgres
```

  - Make sure the database table names and values match the values in config.py . Leave them blank if you do not need to clip the probe data with shapefile.

```
##------------ Limit Geographic Boundary of Data
# if NOT needed then give empty values
'''
shp_table_name      = 'gadm36_NPL_3'
column_name         = 'NAME_3'
column_name_value   = 'Kaski'
'''
shp_table_name      = ''
column_name         = ''
column_name_value   = ''
```

Figure 11: Config File

- Select ‘gps_probe_nepal.csv’

	ap_id	timestamp	latitude	longitude
2	4545	2019-07-01 00:02:30	28.201628	83.969565
3	4545	2019-07-01 00:03:05	28.201328	83.967702
4	4545	2019-07-01 00:04:43	28.203124	83.966503
5	4545	2019-07-01 00:05:32	28.204191	83.964941

Figure 12: Input File Fields

4.2 Clean Probe Data:

- Select sampling percentage. For eg. if you select 10, then the program will use 10% of the ap_id data.
- Removing duplicates.
- Correcting duplicate timestamp at different location.
- Clipping the Probe Data within the selected (e.g. Kaski) region only (if shapefile is provided).

4.3 Generate Routes:

- The **output/** directory will contain the output files.
- ‘*combined_csv.csv*’ contains the generated routes for all ap_id.
- ‘ap_id’ values in ‘*combined_csv.csv*’ are anonymized with Numerical values and saved as ‘*final_csv_4_mobmap_big.csv*’ .
- MobMap visualization can be done using
  - ‘*final_csv_4_mobmap.csv*’ (generated **quickly** without Internet requests with OSMAPI)
  - ‘*final_csv_4_mobmap_big.csv*’ (Takes more time for generation because it make multiple request with OSMAPI over Internet)
- Filenames starting with **‘log_’** are log files.
- res_csv and temp_csv contains temporary files.
- Make sure the **output/** folder is removed before starting this program.

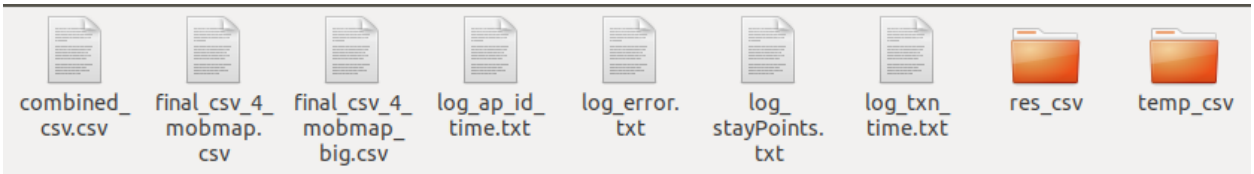

Figure 13: Output Files

4.4 Intermediate Files:

- The input/ directory contains original as well as intermediate probe data.
  - Original.csv : raw data uploaded to the system at Step 1.
  - preprocessed.csv : Preprocessed file (remove duplicates, corrected same timestamp and multiple location issue) for all the data.
  - csv/ folder : input separated into individual .csv file for each ap_id (i.e. each car)

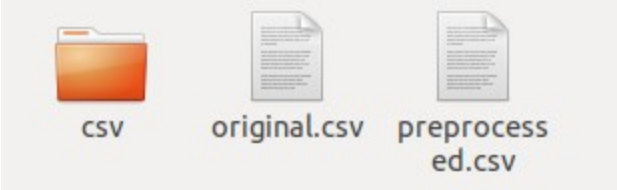

Figure 14: Input Folder with Intermediate files

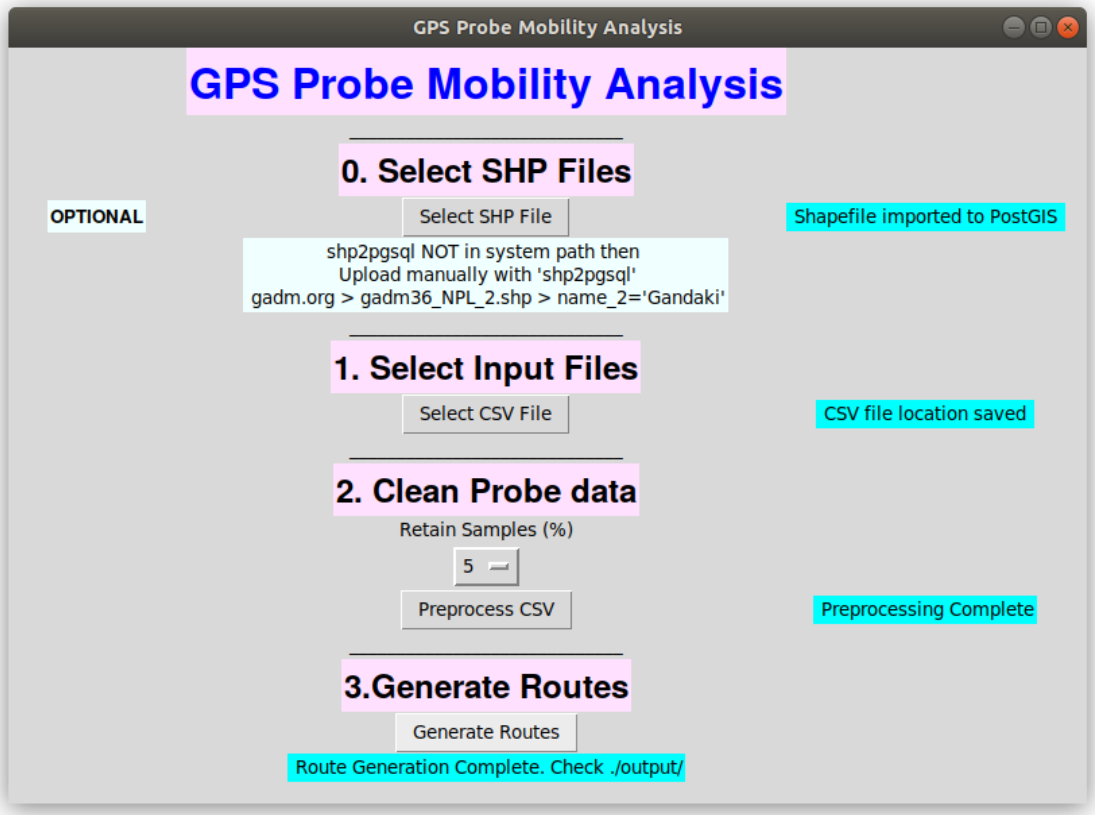

Figure 15: Program Interface (after successful execution)

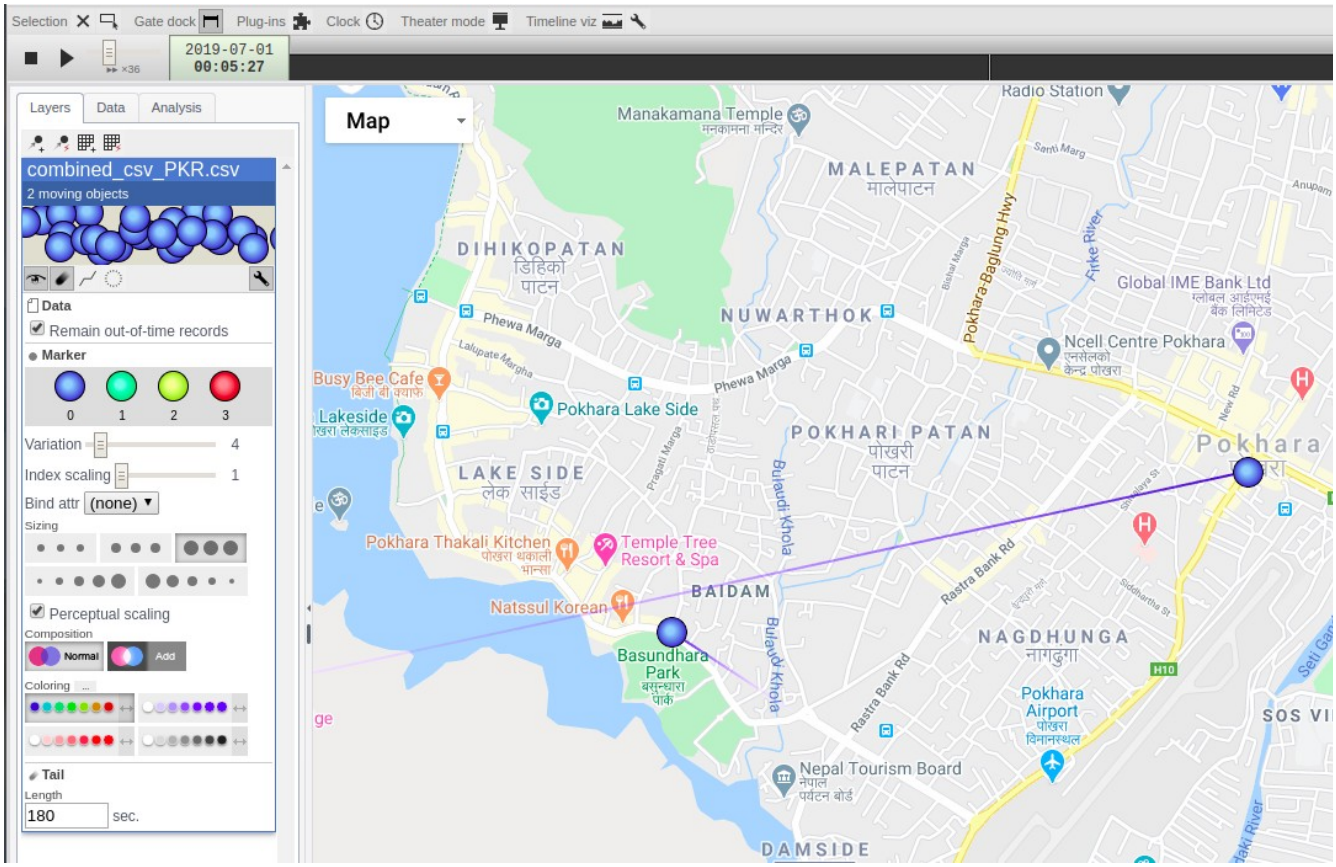

Figure 16: Visualization in MobMap

5. Run main_program.py (Local data source)

5.1 Specify the OSM data file location in config.py

```
##------------- local OSM data file OR online
'''
OSM data is accessed online by pyrouteLib3. For local data option
comment this line and uncomment the following line.
'''
osm_data_source = ""

#osm_data_source = "pokhara_roads.osm"  # downloaded from OpenStreetMap
#osm data location For offline processing
```

Figure 17: Local Data source for route generation where Internet is not available

5.2 OSM data

- Download defined region and regional data from: http://download.geofabrik.de/
- Select smaller or specific region as follows and save (e.g. *my_map.osm*)

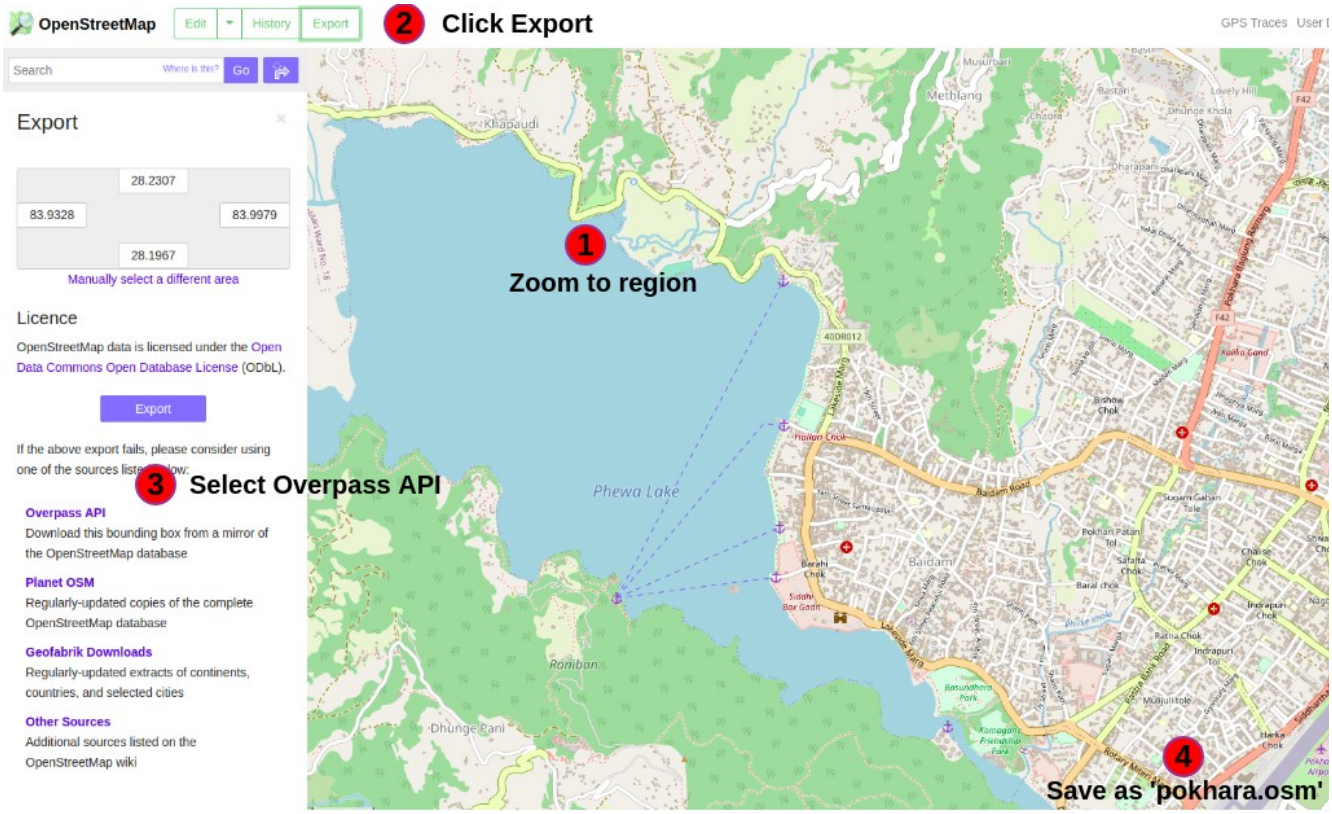

Figure 18: Extract OSM Data

- OSM data size effects the performance. Hence, consider fitering unnecessary data from it. An example is shown below:
  - Filter data from ‘`pokhara.osm`’ using osmfilter tool and save as ‘`pokhara_roads.osm`’.

```
osmfilter pokhara.osm --keep="highway=*" > pokhara_roads.osm
```