IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 September 2022, accepted 14 October 2022, date of publication 25 October 2022, date of current version 1 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3216565

APPLIED RESEARCH

PyTrack: A Map-Matching-Based Python Toolbox for Vehicle Trajectory Reconstruction

MATTEO TORTORA, (Student Member, IEEE), ERMANNO CORDELLI, AND PAOLO SODA, (Member, IEEE)

Department of Engineering, Unit of Computer Systems and Bioinformatics, University Campus Bio-Medico of Rome, 00128 Rome, Italy

Corresponding author: Matteo Tortora (m.tortora@unicampus.it)

This work was supported in part by the "University-Industry Educational Centre in Advanced Biomedical and Medical Informatics (CEBMI)" (Educational, Audiovisual and Culture Executive Agency of the European Union) under Grant 612462-EPP-1-2019-1-SK-EPPKA2-KA. The work of Matteo Tortora was supported by FSTechnology SpA, through the Ph.D. Grant.

ABSTRACT The exponential growth of IoT devices, smartphones, smartwatches, and vehicles equipped with positioning technology, such as Global Positioning System (GPS) modules, has boosted the development of location-based services for several applications in Intelligent Transportation Systems. However, the inherent error of location-based technologies makes it necessary to align the positioning trajectories to the actual underlying road network, a process known as map-matching. To the best of our knowledge, there are no comprehensive tools that allow us to model street networks, conduct topological and spatial analyses of the underlying street graph, perform map-matching processes on GPS point trajectories, and deeply analyse and elaborate these reconstructed trajectories. To address this issue, we present PyTrack, an open-source map-matching-based Python toolbox designed for academics, researchers and practitioners that integrate the recorded GPS coordinates with data provided by the OpenStreetMap, an open-source geographic information system. This manuscript overviews the architecture of the library offering a detailed description of its capabilities and modules. Besides, we provide an introductory guide to getting started with PyTrack covering the most fundamental steps of our framework. For more information on PyTrack, users are encouraged to visit the official repository at https://github.com/cosbidev/PyTrack or the official documentation at https://pytrack-lib.readthedocs.io.

INDEX TERMS Intelligent transportation systems, map-matching, OpenStreetMap, Python, library, hidden Markov model, global navigation satellite system, trajectory.

I. INTRODUCTION

The exponential growth of IoT devices and Big Data technologies we are witnessing in recent years has favoured the rise of location-based services, which made available trajectory data that in the context of transportation systems are used to track the continuous movement of objects, such as cars, bikes, pedestrians, etc. Among these services, the Global Navigation Satellite System (GNSS), such as the Global Positioning System (GPS), has emerged as one of the leading location-based technologies, assisting and serving intelligent transportation in several applications [1]. Intelligent Transportation Systems (ITSs) have been developed since the second half of the 20th century. Both increasing urbanisation and the latest advances in technology have made it a timely and extremely relevant topic [2], [3]. ITSs aim to improve transport systems in all aspects and concern the design, analysis and control of information technology by integrating data from different sources (e.g. GNSS receivers, cameras, LIDAR systems and so on) [2].

Due to both the inherent error of location-based systems (e.g. satellite positioning error, transmission errors, etc.) and the environment-related issues (e.g. signal outages, multipath problems, poor sky visibility, etc.), the acquired trajectories diverge from the actual ones. Consequently, to guarantee high-quality trajectory data to ITS applications, it is first required to align the GPS sampling data to the actual underlying driving routes network layer, a process known as

The associate editor coordinating the review of this manuscript and approving it for publication was Venkata Ratnam Devanaboyina.

map-matching [4]. Indeed, the main objective of map-matching algorithms is to reconstruct the correct sequence of road segments the vehicle is travelling on by integrating the positioning data with the spatial road network data.

Although many routing engines integrate basic map-matching features, to the best of our knowledge, there are no comprehensive tools that allow us to model street networks, conduct topological and spatial analyses of the street graph, reconstruct trajectories by performing map-matching processes on GPS coordinates, and deeply analyse and elaborate these reconstructed trajectories. Furthermore, these routing engines are designed for the industries by offering ready-to-use systems running on application servers and are all non-Python based. Python [5] is a high-level, cross-platform, general-purpose interpreted programming language that, due to its supportive community, readability and learning curve, is now one of the most popular programming languages in the research sector. To address this absence of frameworks for vehicle trajectory reconstruction, analysis and visualisation, we propose PyTrack. PyTrack is an open-source Python toolbox designed for academics, researchers and practitioners, even from the industry, that integrates the recorded GPS coordinates with data provided by the open-source OpenStreetMap (OSM) [6], a collaborative project offering a free editable geographic database of the world, making available the geodata underlying the maps. Our toolbox can serve the intelligent transport research in different contexts, such as improving the computation of the distance travelled by a driver to calculate the fare of a ride, analysing traffic flows, developing virtual simulation environments for the first training phase in self-driving vehicle applications, reconstructing the video of a vehicle's route by exploiting available data and without equipping it with camera sensors, to update the urban road network, and so on.

To summarise, there are several key features characterising the PyTrack toolbox, making it unique. They are:

- A library written in pure Python language, so that it can be easily used in the academic, research and industrial sectors;
- Open-source software with a liberal license based on the open-source OpenStreetMap data;
- An automatic and smart processing of OpenStreetMap data based on regions of interest and buffer areas making efficient use of RAM memory;
- A trajectory reconstruction module enabling snapping, data cleaning, interpolation and map-matching based on Hidden Markov Model (HMM) and Viterbi algorithm.
- Core analytic modules including visualisation and video extraction capabilities based on open-source frameworks and Google Street View API.
- A plugin-based analytic module, enabling the user to customise and add functionality based on personal development requirements.
- All PyTrack services are easily available either via the Python library interface or via the command-line interface (CLI) without running any servers.

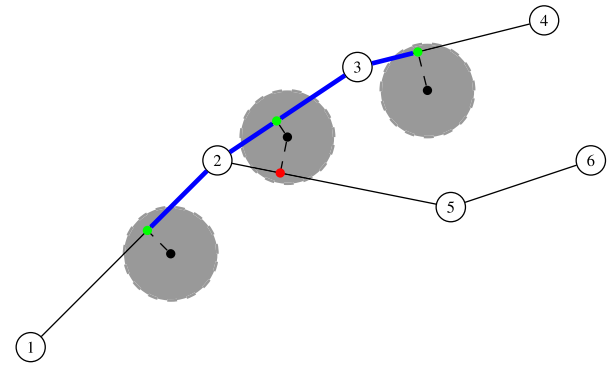

FIGURE 1. Road network graph with an example of map-matching application. The filled circle areas represent the confidence regions around the recorded geographical coordinates, the blue edge path is the reconstructed trajectory, and, finally, the black, red and green dots are the sampled GPS points, the rejected candidate and the matched points, respectively.

Furthermore, our toolbox is well-documented with a getting start guide to help the user familiarise themselves with our framework, and a large number of toy examples are available.[1]

The rest of this manuscript is organised as follows: in section II we present a brief overview of map-matching since it is a central issue of this project. Then, in section III, we present existing frameworks dealing with the map-matching problem to better highlight the purpose of the proposed tool. Section IV outlines the framework architecture, offering a detailed description of the individual modules. Then, in section V we provide an introductory guide to PyTrack covering the most essential steps of our toolbox. Finally, concluding remarks are presented in section VI.

II. MAP-MATCHING: A QUICK OVERVIEW

In this section, we provide a general overview of map-matching algorithms to better contextualise our work, offering a brief formulation of the concepts behind the method and a concise taxonomy of the topic.

Map-matching algorithms involve aligning recorded geographic coordinates (e.g. GPS data) to the underlying spatial road network to identify the correct road path a vehicle was on. As shown in Figure 1, if a road network is represented as a directed topological road network graph $G = (V, E)$, where each node $v \in V$ is a road junction and each edge $e \in E$ is a road segment, the map-matching algorithms attempt to find the sequence of edges $\{e_1, e_2, \ldots, e_n\}$ that a vehicle is driving along. It is noteworthy that the map-matching process is not error-free. Complex road topology, errors in maps and data sparseness can produce different levels of ambiguity.

Given the growth of map-matching techniques applied in different scenarios, different taxonomies have been proposed to categorise them in accordance with the different properties to be highlighted. In [4], the authors proposed a frequency-based taxonomy that categorises map-matching

[1]The official library documentation can be reached via https://pytrack-lib.readthedocs.io

TABLE 1. Comparison of related map-matching frameworks. [1] Analytics capabilities are limited to the street network graph, without any trajectory analytics.

Name	License	Python	Routing Engine	Road Graph Modelling	Map Matching	Analytics
OSRM	BSD-2	✗	✓	✗	✓	✓[1]
Roads API	Commercial	✗	✗	✗	✓	✗
Valhalla	MIT	✗	✓	✗	✓	✓[1]
GraphHopper	Apache-2.0	✗	✓	✗	✓	✓[1]
OSMnx	MIT	✓	✗	✓	✗	✓[1]
Kinetica	Commercial	✗	✗	✓	✓	✓[1]
FMM	Apache-2.0	✗	✗	✗	✓	✗
MMA	GPL-3.0	✓	✗	✗	✓	✗
PyTrack	BSD-3	✓	✗	✓	✓	✓

techniques into three different groups based on the sampling frequency of the recorded geographical coordinates:

- **High-frequency sampling:** The GPS data point are sampled with a sampling time in the range $1 - 0.1$ Hz.
- **Low-frequency sampling**: The GPS data point are sampled with a sampling time in the range $11 - 120$ s.
- **Ultra-low-frequency sampling**: The GPS data point are sampled with a sampling time of more than 2 min.

In [1] the authors proposed a data information-based taxonomy categorising the map-matching approaches into four groups based on the information used to perform the matching. These are the following:

- **Geometric:** Geometric principle-based map-matching algorithms take into account only the geometric information of the spatial road network data without considering the way edges are connected. In general, these methods find the optimal path analysing the relationship between geometric entities using distance, similarity and other factors.
- **Topological:** Topology-based techniques take into account the topological relationships (i.e. adjacency, connectivity, containment) among the road network graph entities in combination with the local search algorithm to find the optimal path.
- **Probabilistic:** Probability-based algorithms involve finding a matching path with the maximum probability using confidence regions around the recorded geographical coordinates.
- **Advanced techniques:** Advanced techniques concern the use of novel and more refined concepts such as a Kalman Filter, Dempster–Shafer's mathematical theory of evidence, fuzzy logic methods, neural networks, reinforcement learning, genetic algorithms and so on, to find an optimal matching path.

Finally, map-matching techniques can be divided into *incremental* and *global* methods according to the number of sampling points considered during the matching process [4]. The former sequentially processes the points one-by-one, whilst the latter takes into account the whole trajectory of the sampled data point to determine the matched one.

III. RELATED WORK

In the recent years, various frameworks with capabilities to serve the ITS paradigm have been developed in different programming languages. However, none of these completely covers the problem of vehicle path reconstruction.

An overview of the most relevant frameworks is listed in Table 1, which summarises those that have an active community and are more in line with the ITS paradigm. Now, we briefly describe each one.

The Open Source Routing Machine (OSRM) [7] is a high-performance routing engine based on the open and free road network data of the OpenStreetMap project [6]. It is a C++ cross-platform framework fully functional on Linux, FreeBSD, Windows, and Mac OS X operating systems. It also provides a matching sub-module available via HTTP API, C++ library interface and NodeJs wrapper to snap noisy GPS traces to the road network in the most plausible way.

The Roads API [8] is a fee-based service of the Google Maps Platform [9] available via a simple HTTPS interface that allows the user to map GPS coordinates to the geometry of the road. A major limitation of this service is that it handles a maximum of 100 GPS points collected along a route.

Valhalla [10] is an open-source OpenStreetMap-based routing engine. It is a C++ cross-platform framework fully functional on many Linux and Mac OS distributions. Valhalla also includes several library modules each of which is responsible for a different function, including Meili, a library to perform map-matching.

GraphHopper [11] is an open-source OpenStreetMap-based routing library associated with a server and written in Java providing also a web interface called GraphHopper Maps. It is a cross-platform framework officially running on Linux, Mac OS X and Windows. It also provides a sub-module to snap GPX coordinates to the underlying spatial road network.

OSMnx [12] is an open-source Python library built on top of OpenStreetMap's APIs. It allows you to download and model road networks or other infrastructures, filtering road networks by car, bicycle, public transport, and foot accessibility, and allows you to simplify network topologies by cleaning nodes and consolidating intersections. Although it is a well-maintained library, its purpose is limited to analysing the topologies of road networks modelled through graphs, e.g. calculating street bearings, orientations or shortest-path routes. In fact, it does not provide trajectory elaboration and map-matching functionalities.

Kinetica [13] is a scalable and distributed memory-first OLAP database written in C++ developed by Kinetica DB Inc. under a proprietary license. It enables real-time analysis on large geospatial and temporal datasets with integrated graph, SQL, and AI/ML capabilities. It also includes map-matching capabilities allowing the user to map GPS coordinates to road geometry.

Fast Map Matching (FMM) [14] is a high-performance open source map-matching framework solving the problem of matching noisy GPS data to a road network. It is C++ cross-platform library fully functional on Unix, Mac and Windows operating systems, including a Python API wrapping C++ code. Although it is a well-documented library, its purpose is

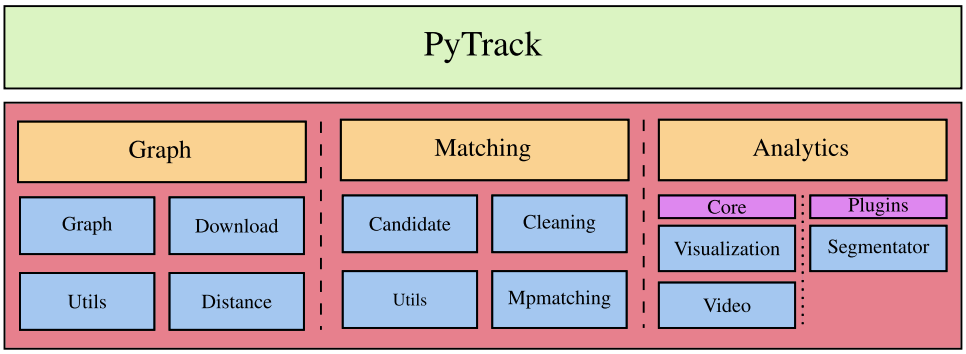

FIGURE 2. Architecture of PyTrack.

limited to map-matching functionalities and it has not been updated for about two years.

Map-Matching Algorithm (MMA) [15] is an open-source Python implementation of [16] with functionalities limited to map-matching. It is built on top of OpenStreetMap's APIs and allows you to download and model road networks via Redis, an open-source in-memory database. Note that it has not been updated for about three years.

Although many existing frameworks offer map-matching features, on the one hand, they are mainly all non-Python all-in-one software and, on the other hand, they are designed for the industry, requiring these ready-to-use frameworks to be run on private servers, distributed custom map applications, and so on. Hence, in this work, we propose an open-source map-matching-based pure Python toolbox to reconstruct a vehicle trajectory by integrating the OpenStreetMap data with the recorded GPS coordinates. In addition, unlike all the frameworks seen above, this toolbox implements a plugin-based analytic module that allows the analysis, visualisation and elaboration of the reconstructed trajectories. The analytical module implements core functionalities, including visualisation features and a module that uses the optimally matched trajectory to reconstruct a video of the whole route a vehicle has travelled by querying the Google Street View API.

IV. ARCHITECTURE

Library architecture is necessary to keep the code well organised and well maintained. On the one hand, this facilitates the testing process, helps to discover and fix bugs, allows flexibility and extendibility by adding new modules and facilitates code scaling. On the other hand, it increases the usability of the library itself by enabling developers and users to get an overview of existing packages, modules, classes and methods. Figure 2 shows an overview of the architecture of PyTrack. The various modules, classes and methods are organised in three packages, namely graph, matching and analytics. Below there is a description of these three core PyTrack packages.

1) **Graph**: as an underlying road graph is required to enable many library services, this package permits you to build, analyse and manage it. It is built on the OpenStreetMap API and arranges downloaded geospatial data in graph form. Hence, PyTrack provides a set of modules and methods to build and analyse a graph based on the NetworkX library [17], the same library used by OSMnx; this guarantees cross-compatibility between libraries, allowing us to simplify and lighten our library.
2) **Matching**: this provides a set of algorithms and data-structures for trajectory reconstruction matching a sequence of noisy GPS points to the underlying street network assembled with the previous package.
PyTrack uses a Hidden Markov Model-based approach proposed in [18] to solve the map-matching problem. Please refer to Figure 1 to see how the map-matching problem can be modelled. Given a sequence of noisy GPS measurements (observations in terms of HMMs) correlated with a set of potential candidate road points (hidden states in terms of HMMs), the problem is to find the most likely sequence of candidates and hence reconstruct the road path. This task of determining which sequence of variables is the underlying source of some sequence of observations is called the decoding task. However, the most common decoding algorithm for HMMs is a dynamic programming algorithm known as the Viterbi algorithm [19].
3) **Analytics**: this provides a set of methods and classes that enable the analysis, visualisation and processing of reconstructed trajectories that can help understand the data and support ITS with analysis capabilities. Since classes represent one of the library's fundamental data types, the package implements a class for visualising and analysing maps. This class inherits methods from Folium [20], a Python library that wraps Leaflet [21], a Javascript library implemented to provide interactive maps. In addition, the package is organised in such a way that third-party methods can easily be used on the data extracted via PyTrack.

V. GETTING STARTED

The following section provides a starter's guide to PyTrack v2.0.6. It covers the key steps to getting started with our library, beginning with the installation process, followed by the initialisation of a road graph, and ending with the execution of a map-matching process and the visualisation of its results. For the sake of brevity, this guide is only a brief overview of PyTrack's functionality. Therefore, readers interested in delving into these issues or exploring further practical

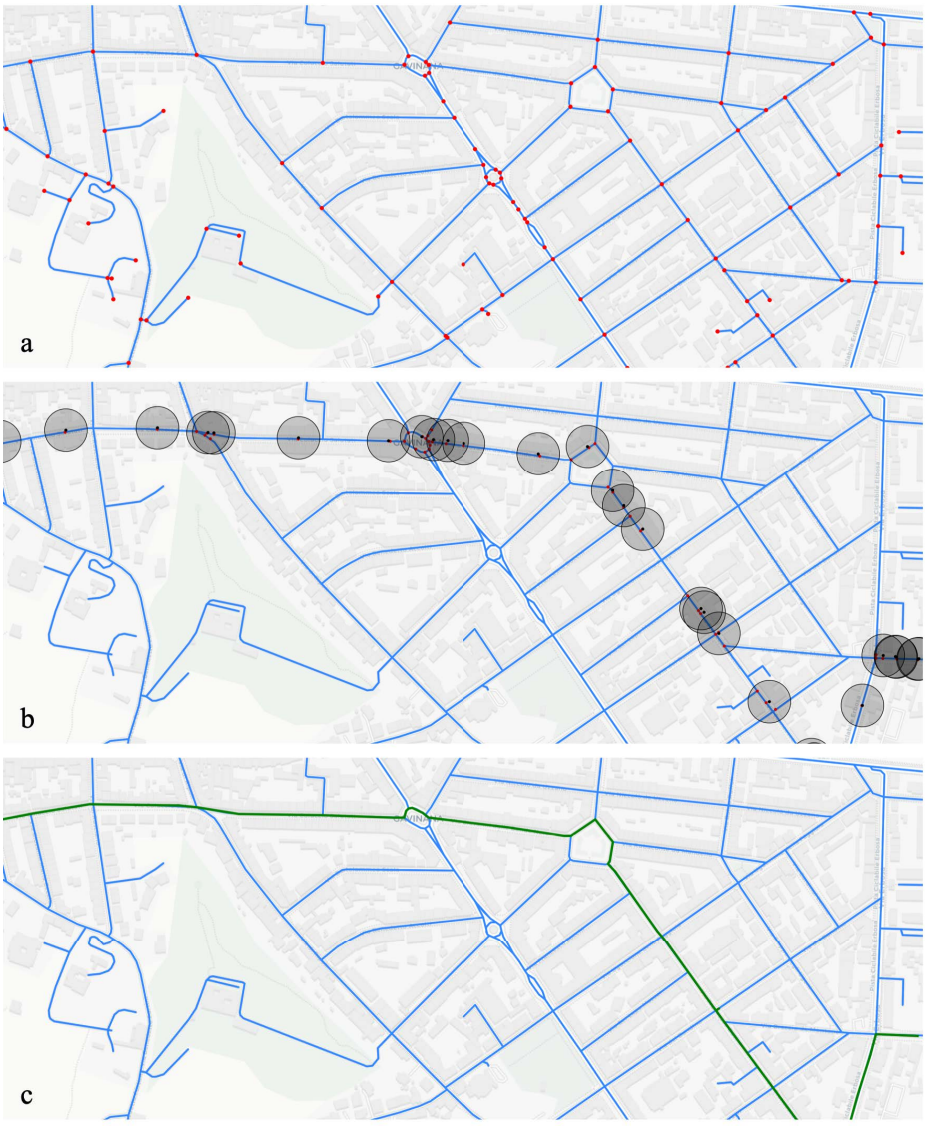

FIGURE 3. Different visualisation methods implemented in the PyTrack's `Map` class. Panel (a) shows an example of a road network graph by using PyTrack's `graph_from_bbox` method. Panel (b) shows an example of computed using `get_candidates` method. Panel (c) in green shows an example of the matched path obtained by the `viterbi_search` method.

examples are encouraged to visit the official documentation at https://pytrack-lib.readthedocs.io

A. INSTALLATION

PyTrack is available both on Python Package Index (PyPI) [22] and Anaconda repository [23]. PyPI is the central repository for the Python programming language to make software easily portable and accessible. On the other hand, Anaconda collects binary packages, so no compiler is needed to install them. Therefore, our toolbox can be installed by using the Python Packaging Authority's recommended package manager:

```
$ install pytrack-lib
```

Or by using the official Anaconda package and environment manager:

```
$ a install pytrack
```

B. GET THE ROAD NETWORK GRAPH

A fundamental step enabling us to deal with GPS data points is to model street networks. Next, we initialise a road graph using only two lines of code:

```
from pytrack.graph import graph
from pytrack.graph import distance

bbox = distance.enlarge_bbox(north,
    south,
west, east, dist)
G = graph.graph_from_bbox(*bbox,
simplify=True,
network_type='drive')
```

The `enlarge_bbox` method returns a bounding box of the region of interest and takes as input variables *north* and

south, indicating respectively the northern and the southern latitude of the bounding box, the *west* and *east* variables, indicating respectively the western and the eastern longitude of the bounding box, and, finally, a variable *dist*, in metres, indicating how much to expand the bounding box. The `graph_from_bbox` method creates a multi-directed graph querying the OpenStreetMap service. The method uses the NetworkX library to create the graph, so the graph object is compatible with NetworkX's methods. It is also compatible with the methods provided by the OSMNx library, since the latter also constructs road graphs using NetworkX. Figure 3a shows an example of the network graph created using the `graph_from_bbox` method and rendered using our `Map` class. We will further explore the latter when we present the analytics package in subsection V-D.

C. INTO THE MAP-MATCHING PROCESS

In the following, we will look at the core methods implemented in the matching package.

First, let us look at how to extract the candidate points. Let us recall that the candidate points are those points with a high probability of being the points that best match the actual GPS points. In PyTrack, they are extracted with the following line of code:

```
from pytrack.matching import candidate

G_interp, results = candidate.
    get_candidates(G,
points, interp_dist=5,
radius=30)
```

The `get_candidates` method returns two outputs, *G_interp* and *results*, respectively, an interpolated version of the graph computed in the previous section and a dictionary of candidate points. The interpolation of the graph is necessary to obtain a more accurate map-matching result. The method takes as input the original graph *G*, a list *points* of the coordinates (latitude/longitude) of the actual GPS points in the same coordinate reference system as the graph on which to perform the map-matching operation, the step distance *interp_dist* to interpolate the graph, and finally the *radius* in metres of the circle representing the confidence region around which to search for candidate points. Panel b of Figure 3 shows a visualisation of the results obtained by the `get_candidates` method.

Once we have identified the candidate points, we are able to construct the *trellis* diagram. It is a directed acyclic graph and is the central data type used to model Hidden Markov Models to perform the Viterbi algorithm. In our case, each node of this graph represents a candidate, namely an object of the class `Candidate` used to represent a candidate element, and each edge is a transition probability between two candidates. In PyTrack we can compute a trellis graph with only one line of code:

```
from pytrack.matching import
    mpmatching_utils
```

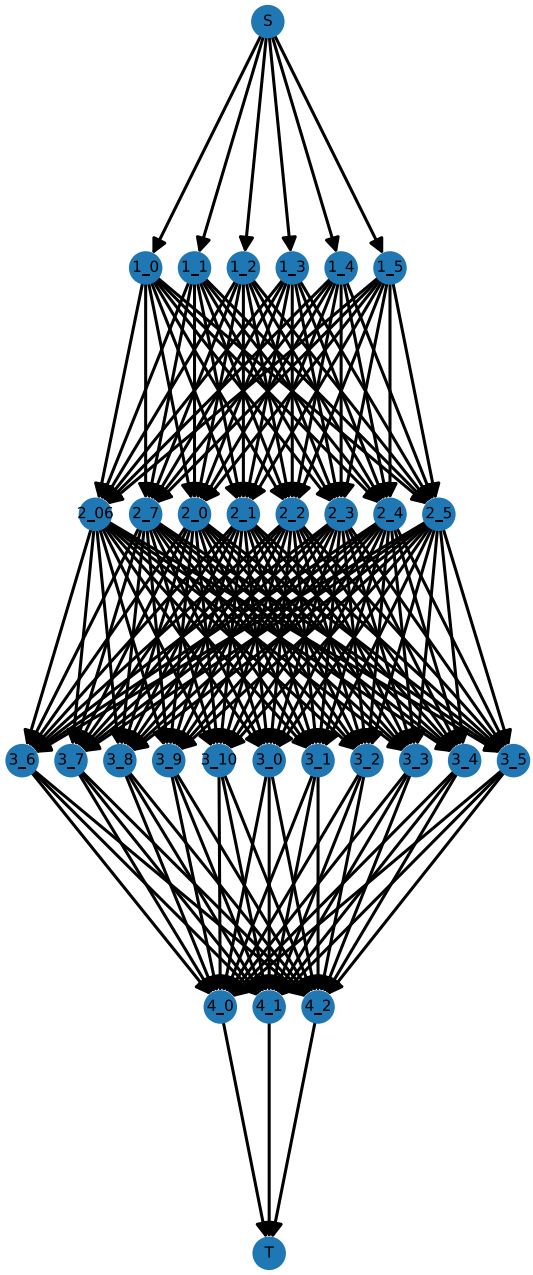

FIGURE 4. An example of a directed acyclic trellis graph representing a Hidden Markov Model modelling a map-matching problem. The figure is obtained using the PyTrack's `draw_trellis` method. The goal is to find the most likely sequence of candidates that best matches the actual GPS points from the start node to the target node.

```
trellis = mpmatching_utils.
    create_trellis(
    candidates)
```

The method returns a directed acyclic trellis graph and takes as input the output of the `get_candidates` method. Figure 4 shows a representation of the diagram obtained by the `create_trellis` method.

Finally, once the trellis diagram schematising the Hidden Markov Model of the map-matching problem is obtained, we are ready to perform the matching process. In PyTrack

we can perform the map-matching process with the following line of code:

```
from pytrack.matching import mpmatching

path_prob, pred = mpmatching.
    viterbi_search(
G_interp, trellis)
```

This method returns *path_prob* and *pred*, which are, the joint probability and the matched predecessor for each node of the trellis diagram, respectively. It also takes as input the interpolated version of the street graph *G_interp* and directed acyclic diagram *trellis*. Panel c of Figure 3 shows the path obtained by the `viterbi_search` method, i.e. the most likely sequence of candidates that best match the sequence of the actual GPS measurements.

D. VISUALIZING THE RESULTS

We now present the methods used for visualising the results of the methods introduced in section V-C.

Since our library is primarily concerned with reconstructing the vehicle's trajectory and the primary data type the library deals with is GPS coordinate, it is necessary to render the maps. PyTrack offers the `Map` class contained in the analytics package. This extends the `folium.Map` to add functionality useful to represent graphs and road paths. Folium is a library enabling the interactive mapping strengths of the Leaflet library on Python. Let us now delve into how we can use this class.

A base map must be created before you can operate and add elements to a map. In PyTrack you can create a base map with one line of code, passing your starting coordinates:

```
from pytrack.analytics import
    visualization

maps = visualization.Map(location=(lat,
    lon))
maps
```

In this way, you have initialised a map object with which you can interact. Now we can add elements to this base map: for instance we now add the street graph created with the `graph_from_bbox` method introduced in section V-B. This can be done in PyTrack, by the following code:

```
from pytrack.analytics import
    visualization
maps.add_graph(G)
maps
```

This takes as input the street network graph *G*; in this respect, panel (a) in Figure 3 shows an example of the graph rendered.

In addition, we can also display on the map the candidate nodes identified using the method `get_candidates` described in section V-C. In PyTrack, use the following method from the class `Map`:

```
from pytrack.analytics import
    visualization
maps.draw_candidates(candidates, radius)
maps
```

This takes as input *candidates* and *radius*: they are a Python dictionary of candidate nodes and the candidate search radius, respectively. Panel (b) of Figure 3 shows an example of the candidate nodes and the search region rendered using a filled circle.

Finally, we render the reconstructed path that best matches the actual GPS points obtained using the method `viterbi_search` (section V-C) and the following method code:

```
from pytrack.analytics import
    visualization
maps.draw_path(G_interp, trellis, pred)
maps
```

This takes as input the interpolated version of the original graph, the directed acyclic trellis graph and a dictionary collecting the matched predecessor for each node of the graph, respectively. All these inputs are computed using the methods seen in the previous section. Panel (c) of Figure 3 shows an example of the reconstructed path.

With the analytics package we can also render the trellis diagram computed using the `create_trellis` method, by typing:

```
from pytrack.analytics import
    visualization
graph = visualization.draw_trellis(
trellis, **kwargs)
graph
```

This returns a Matplotlib figure illustrating the trellis diagram used in the Hidden Markov Model process to find the path that best matches the actual GPS data. This method uses both the NetworkX and Matplotlib [24] library, for creating the diagram and rendering the figure, respectively. Consequently, it inherits all their methods and properties. The method takes as input *trellis* and *kwargs*, respectively, the directed acyclic trellis graph obtained using the `create_trellis` method and other optional keyword arguments allowing us to specify how the image should be rendered, such as the size and resolution of the figure, the size and font of the nodes, and so on. Figure 4 shows an example of the application of this method, where, for the sake of visualisation, only a few nodes of the trellis diagram are shown.

VI. CONCLUSION

This paper has introduced PyTrack, an open-source map-matching-based Python toolbox to reconstruct a vehicle trajectory that can support the ITS paradigm offering analytics capabilities. An introduction to the ITS paradigm and the map-matching problem was offered to better contextualise the scope of this library; we also analysed existing frameworks to explain why we developed our library.

The architecture of the framework consists of three principal packages: graph, matching and analysis, and we described each package in depth to illustrate its functionalities. Finally, we provided a brief step-by-step walkthrough from installation to the process of trajectory reconstruction and visualisation of results.

As presented in section IV, the library allows the use of a third-party plugin for semantic segmentation of street scenes, in the future, we plan to implement an in-house segmenter to increase synergy with the entire Pytrack ecosystem. Furthermore, still referring to the analytical package, we aim to implement an in-house plugin to perform road condition analysis using computer vision techniques applied directly to the extracted videos. This could be used to facilitate local authorities in the observation, analysis and maintenance of the road network. All these improvements aim to serve ITS with innovative functionalities to facilitate and improve the transportation sector with a positive return on community welfare.

In the tradition of open-source development and collaboration, any kind of contribution is welcome. Therefore, developers and researchers who would like to make code improvements, add new functionalities, and propose documentation improvements and ideas are invited to contact the authors. However, it is worth noting that although the library has been developed as bug-free as possible, issues may occur during the runtime. Therefore, in case of troubles, bugs and doubts during execution, please contact us directly on the official PyTrack repository.

REFERENCES

[1] M. A. Quddus, W. Y. Ochieng, and R. B. Noland, "Current map-matching algorithms for transport applications: State-of-the art and future research directions," *Transp. Res. C, Emerg. Technol.*, vol. 15, no. 5, pp. 312–328, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X07000265
[2] L. Zhu, F. R. Yu, Y. Wang, B. Ning, and T. Tang, "Big data analytics in intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 1, pp. 383–398, Jan. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8344848
[3] A. Haydari and Y. Yilmaz, "Deep reinforcement learning for intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 1, pp. 11–32, Jan. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9146378
[4] Z. Huang, S. Qiao, N. Han, C. Yuan, X. Song, and Y. Xiao, "Survey on vehicle map matching techniques," *CAAI Trans. Intell. Technol.*, vol. 6, no. 1, pp. 55–71, Mar. 2021. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12030
[5] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA, USA: CreateSpace, 2009.
[6] *OpenStreetMap*. Accessed: Oct. 26, 2022. [Online]. Available: https://www.openstreetmap.org/about
[7] D. Luxen and C. Vetter, "Real-time routing with OpenStreetMap data," in *Proc. 19th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS)*. New York, NY, USA: ACM, 2011, pp. 513–516, doi: 10.1145/2093973.2094062.
[8] *Roads API*. Accessed: Oct. 26, 2022. [Online]. Available: https://developers.google.com/maps/documentation/roads
[9] *Google Maps Platform*. Accessed: Oct. 26, 2022. [Online]. Available: https://developers.google.com/maps
[10] *Valhalla*. Accessed: Oct. 26, 2022. [Online]. Available: https://github.com/valhalla/valhalla
[11] *GraphHopper*. Accessed: Oct. 26, 2022. [Online]. Available: https://github.com/graphhopper/graphhopper
[12] G. Boeing, "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Comput. Environ. Urban Syst.*, vol. 65, pp. 126–139, Sep. 2016.
[13] *Kinetica*. Accessed: Oct. 26, 2022. [Online]. Available: https://www.kinetica.com
[14] C. Yang and G. Gidófalvi, "Fast map matching, an algorithm integrating hidden Markov model with precomputation," *Int. J. Geograph. Inf. Sci.*, vol. 32, no. 3, pp. 547–570, 2018.
[15] *Map-Matching Algorithm*. Accessed: Oct. 26, 2022. [Online]. Available: https://github.com/categulario/map_matching
[16] H. Koller, P. Widhalm, M. Dragaschnig, and A. Graser, "Fast hidden Markov model map-matching for sparse and noisy trajectories," in *Proc. IEEE 18th Int. Conf. Intell. Transp. Syst.*, Sep. 2015, pp. 2557–2561.
[17] A. Hagberg, P. Swart, and D. S. Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos Nat. Lab. (LANL), Los Alamos, NM, USA, Tech. Rep. LA-UR-08-05495, 2008.
[18] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in *Proc. 17th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2009, pp. 336–343.
[19] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2009.
[20] Python-Visualization. *Folium*. Accessed: Oct. 26, 2022. [Online]. Available: https://python-visualization.github.io/folium/
[21] *Leaflet*. Accessed: Oct. 26, 2022. [Online]. Available: https://github.com/Leaflet/Leaflet
[22] PSFoundation. *PyPI: The Python Package Index*. Accessed: Oct. 26, 2022. [Online]. Available: https://pypi.org
[23] I. Anaconda. *Anaconda*. Accessed: Oct. 26, 2022. [Online]. Available: https://www.anaconda.com
[24] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, May 2007.

MATTEO TORTORA (Student Member, IEEE) received the bachelor's degree in industrial engineering and the master's degree (Hons.) in biomedical engineering from the University Campus Bio-Medico, Rome, Italy, in 2017 and 2020, respectively, where he is currently pursuing the Ph.D. degree in biomedical engineering (artificial intelligence area) with the Computer Systems and Bioinformatics (CoSBi) Research Laboratory, Faculty of Engineering. His current research interests include reinforcement learning, machine learning, computer vision, and multimodal learning.

ERMANNO CORDELLI received the master's degree (Hons.) in biomedical engineering and the Ph.D. degree in biomedical engineering (computer science area) from the Computer Systems and Bioinformatics (CoSBi) Research Laboratory, Faculty of Engineering, University Campus Bio-Medico di Roma (UCBM), in 2014 and 2017, respectively. He worked for one year with a collaboration contract with the CoSBi Research Laboratory, Faculty of Engineering, UCBM. He is currently an Assistant Professor at the CoSBi Laboratory, UCBM. His main research interests include artificial intelligence and its applications in the health sector, federated learning, computer vision, radiomics, and the IoT working within a project on the creation of a intelligent pen for diabetes treatment.

PAOLO SODA (Member, IEEE) received the degree (Hons.) in biomedical engineering and the Ph.D. degree in biomedical engineering (computer science area) from the University Campus Bio-Medico (UCBM), Rome, in 2004 and 2008, respectively. He is currently a Full Professor of computer science and computer engineering at the UCBM. His research interests include artificial intelligence, pattern recognition, machine learning, big data analytics, and data mining applied to data, signal, 2D and 3D image and video processing and analysis.

Open Access funding provided by 'Consiglio Nazionale delle Ricerche-CARI-CARE' within the CRUI CARE Agreement