ST_VISIONS: A Python Library for Interactive Visualization of Spatio-temporal Data

Andreas Tritsarolis[1], Christos Doulkeridis[2], Nikos Pelekis[3], and Yannis Theodoridis[1]
[1]*Department of Informatics*, [2]*Department of Digital Systems*, [3]*Department of Statistics & Insurance Science*
University of Piraeus, Piraeus, Greece
{andrewt,cdoulk,npelekis,ytheod}@unipi.gr

***Abstract*—In this demo paper we present ST_VISIONS, an easy-to-use Python library for interactive visualizations of spatial and spatio-temporal datasets. By automating the low-level details of the underlying visualization library (Bokeh), ST_VISIONS allows data scientists to create interactive, map-based visualizations, by writing Python code at a higher level of abstraction. Consequently, we accelerate the task of visualization from different sources, while we support interactive filtering, colorization, as well as multiple graphs, for various types of spatial and spatio-temporal data.**

I. INTRODUCTION

Modern applications generate massive volumes of spatio-temporal trajectory data daily that need to be collected, processed and analyzed, in order to extract useful knowledge in terms of mobility patterns. The spectrum of applications is wide [1], [2]: fleet monitoring systems, vessel and aircraft tracking services, ride-sharing apps, traffic control management, and so on. In all these domains, a common need is to provide easy support for visualizations in order to quickly perform visual analytics (VA) [3], so that data analysts and domain experts can easily obtain an overview.

However, the task of visualizing a spatio-temporal dataset is still far from trivial nowadays. Existing general-purpose visual analysis tools (such as Tableau) do not provide full support for spatio-temporal or mobility data. GIS software or specialized tools, such as V-Analytics, provide a rich palette of functions, however they cannot be easily integrated with *(iPython) Notebooks*, which has become the de-facto standard for ad-hoc data analytics. On the other hand, data scientists that typically work with Python need to learn and use visualization libraries (e.g., Bokeh) that although they provide rich visualizations, they require a steep learning curve. In consequence, this introduces obstacles and delays the data analysis process.

Motivated by this limitation, we propose a simple, easy-to-learn and use Python library – called ST_VISIONS – for interactive visualization of spatio-temporal datasets. The main objective of ST_VISIONS is to empower data analysts by offering advanced visualizations with only a few lines of code. Perhaps more importantly, the necessary code consists of methods that encapsulate the entire visualization logic and respective elements, thus simplifying the task of visual analysis. Internally, ST_VISIONS uses an underlying library (in our case, Bokeh), but essentially provides wrapper functionality that facilitates and speeds up development. Compared to other Python libraries that focus on management of trajectory data and their visualization, such as MovingPandas [4], ST_VISIONS is more generally applicable to spatial and spatio-temporal data of different types (points, polygons, and polylines).

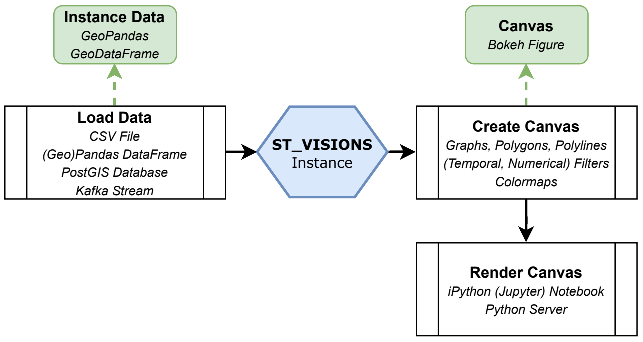


Fig. 1: ST_VISIONS architecture overview

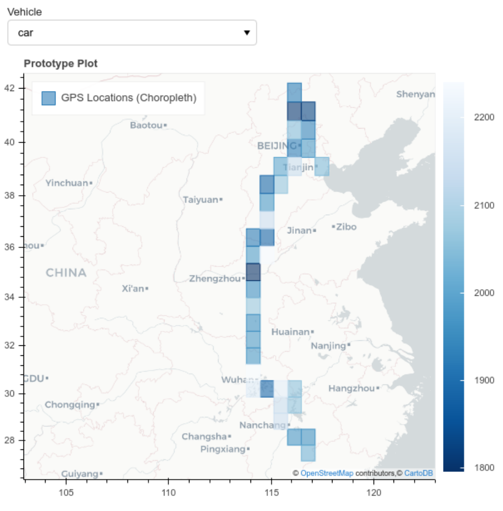
The rest of this paper is structured as follows: in Section II, we present the system design of ST_VISIONS, its functionality and we discuss some technical details. In Section III, we describe the demonstration scenarios that showcase the functionality of ST_VISIONS. Finally, we conclude the paper in Section IV.

II. THE ST-VISIONS LIBRARY

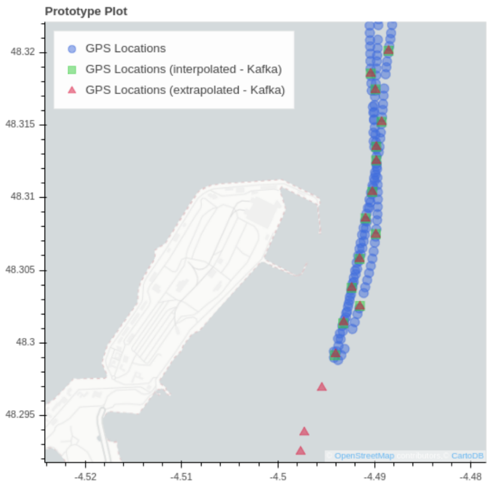
In this section, we present the system architecture of ST_VISIONS and its main functionality, so as to provide a comprehensive view of its offerings for developers and data analysts.

A. System Architecture

Fig. 1 presents the high-level architecture of our system prototype. Three main activities appear there: (a) data loading, (b) canvas creation, and (c) rendering. ST_VISIONS supports various data sources that are typically encountered in data analysis workflows, such as CSV files, relational database (PostGIS), as well as streams (Kafka), which can be easily loaded using a single command. The input data is of spatial or spatio-temporal nature and different types are supported, including points, polygons, and polylines. Also, other non-spatial attributes (object type, object ID, etc.), either categorical or numerical, are also supported.



(a) Choropleth map



(b) Multiple datasets on canvas

Fig. 2: Examples of visualizations obtained by ST_VISIONS

After data loading, the abstraction of a GeoPandas DataFrame is adopted for internal data representation. In turn, this offers a unified view on data, regardless of the original data source. Then, a canvas[1] object is created which is associated with an underlying visualization library (in our case Bokeh), and it is populated with data from the DataFrame. The canvas is subject to parameterization regarding the visual elements (e.g., glyphs, colours, etc.), and filters can be added to the fields of the DataFrame in order to provide interactive visualization. It should be noted that in this step ST_VISIONS exploits the features offered by Bokeh, by encapsulating its functionality in customized methods that have been implemented with main goal to expose to developers en easy-to-use and straightforward interface.

Finally, the rendering of the canvas takes place. ST_VISIONS supports rendering either via iPython (Jupyter) Notebooks or via Browser. The former is quite straightforward, as it involves setting a couple of parameters in code. The latter, however, is a more involved process, as a Python webserver (via the "python -m bokeh serve" command) must be created, thus deploying an application, which can be consumed by desktop and mobile devices as well. In both cases, the rendering process must be quick, without any redundant overhead, in order to avoid any performance throttling phenomena, as we discussed in Section II-C.

[1]bokeh.plotting.figure, Bokeh Documentation

B. Functionality

ST_VISIONS offers a wide palette of features to the data analyst. First, different types of data filtering (i.e., record selection) are supported, including temporal filtering and filtering of categorical/numerical attributes. Also, the visualized objects can be rendered in different colors based on the value of a categorical attribute. For numerical attributes, a heatmap is used to associate similar numerical values to nearby colors in the palette. Moreover, different records that correspond to the same object, such as sequences of positions of a moving object, can be grouped together and visualized as trajectories.

With regards to the input geometries, ST_VISIONS provides methods that automatically extract their corresponding coordinates in order to be suitable for the canvas' plot, allowing the visualization of not only simple, but also quite complex geometries, like polygons with holes. The aforementioned module also provides methods for data transformation, for instance, creating trajectories from point geometries, and classifying proximity, with respect to another (polygon geometry) dataset.

Thanks to the aforementioned methods, more complex visualizations are easily supported. For instance, Fig. 2a illustrates the creation of an interactive Choropleth map, in combination with a categorical filtering based on the vehicle type. Also, multiple datasets can be visualized on the same canvas, which allows comparative inspection. A nice example of this functionality is the visualization of raw and compressed trajectories of moving objects on the same canvas (Fig. 2b). Furthermore, we can arrange multiple canvases in a grid, for comparative inspection and interactive visualization of two (or more) datasets.

C. Technical Aspects

```
1  # Create an ST_VISIONS instance
2  plot = st_visions()
3  # Load a csv dataset
4  plot.get_data_csv(...)
5  # Plot points on the map
6  viz_express.plot_points_on_map(plot)
7  # Add some data filters
8  plot.add_temporal_filter(...)
9  plot.add_numerical_filter(...)
10 # Render geometries on iPython notebook
11 plot.show_figures()
```

The above code snippet shows the basic use of ST_VISIONS. First, an instance of the library is created (line

2), and it is used to load the data (line 4). In the case of point data, the appropriate method is invoked for plotting, using a module, called *viz_express* that encapsulates initialized parameters for ease (line 6). In addition, interactive filters are added, in our example, a temporal and a numerical filter (lines 8 and 9, respectively). Finally, the visualization is generated (line 11).

While the underlying API (Bokeh) is quite popular for interactive dashboards and visualizations, an issue arises when multiple filters (i.e., widgets such as Sliders, DropDown menus, etc.) are introduced. Because each filter is essentially autonomous, they cannot be synchronized automatically, thus may result in data loss, if they are not controlled properly.

A baseline workaround for that problem, is using a shared callback for all introduced widgets, that will take into account the value of each filter simultaneously. While this solution is quite useful, it cannot be generalized for multiple dynamically introduced widgets. Thus, in order to address this issue, we propose a solution that can effectively account for multiple widgets while avoiding performance throttling phenomena.

Algorithm 1: FILTER CALLBACK. The core structure of the filters' callback.

```
Input: Callback Policy $attr$, Old Value $old$, New Value $new$
1 $callback\_filter\_data(filter.id)$
2 $filtered\_data = get\_data()$
3 $filtered\_data = widget\_filter\_data(filtered\_data, new)$
4 $callback\_prepare\_data(filtered\_data, filter.id == this.lock)$
```

Algorithm 1 describes the core structure of a filter's callback method in order to co-exist harmoniously with the previously introduced widgets. More specifically, given a change in the value of a filter, its respective callback method must:

1) Iterate all other introduced widgets and execute their callbacks in order to filter the data (line 1);
2) Fetch the newly filtered data and apply the widgets' corresponding filter method (lines 2–3);
3) Finally, pass the data to the instance's ColumnDataSource (CDS)[2] and render the remaining geometries on the canvas (line 4).

Algorithm 2, which is in charge for the first step of Algorithm 1, iteratively traverses the instance's introduced widgets and triggers their respective callback methods, in order to apply their respective filter to the dataset at hand. To avoid any deadlocks (i.e., recursive calls to the same callback function), a (private) lock attribute is introduced, which (if None) is assigned with the widgets identifier[3].

After our dataset is properly (i.e., according to specification) filtered, it is passed as input to Algorithm 3. This algorithm is in charge of preparing the input data for output (i.e., passing the data to the instance's CDS). Because the aforementioned method exists in each callback method (as part of its core structure), in order to avoid flickering, as well as performance throttling phenomena, rendering to the canvas is done *only* by the widget the lock is assigned to. Finally, our data is rendered, and the lock is released (i.e., reverted to *None*).

Algorithm 2: CALLBACK_FILTER_DATA. Callback corpus for synchronized data filtering.

```
Input: Widget Identifier $id$
1 if $this.lock = None$ then
2     $this.lock \leftarrow id$
3     foreach $widget \in this.widgets$ do
4         if $widget.id \neq id$ then
5             $widget.trigger\_callback(attr = “value”, old = None, new = widget.value)$
6         end
7     end
```

Algorithm 3: CALLBACK_PREPARE_DATA. Preparing filtered dataset for rendering.

```
Input: Filtered Dataset $data$, Output Flag $flag$
1 $this.CDS\_data \leftarrow data$
  /* Q: Am I Ready to Render? */
2 if $flag$ then
3     $this.renderToCanvas(this.CDS\_data)$
4     $this.lock \leftarrow None$        // Releasing Lock...
5     $this.CDS\_data \leftarrow None$
        // Emptying Intermediate Storage...
```

All the above algorithms are integrated into the *BokehFilters* class located at the *callbacks* module of ST_VISIONS, allowing for custom synchronized callbacks. In order to create a custom callback, after inheriting the *BokehFilters* class, one must implement, as baseline, the *callback* method, according to Algorithm 1.

III. DEMONSTRATION SCENARIO

For the demonstration scenarios we envisage, we use iPython (Jupyter) Notebooks for easier interaction. During the demo session, the participants will be able to perform the following steps on two different datasets from the urban and maritime domain, namely, the “GeoLife”[4] [5]–[7] and the “Piraeus” Dataset [8], respectively.

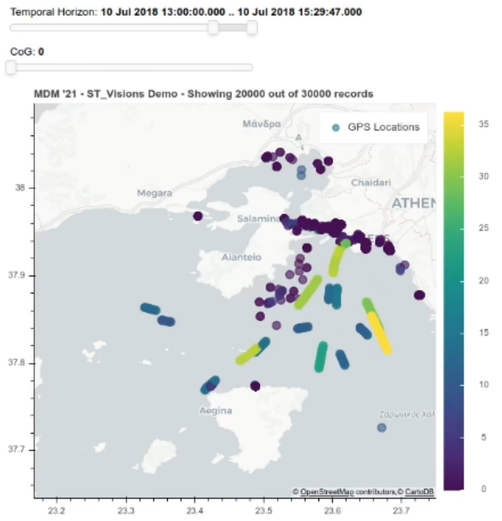
The accompanying video shows the following functionality.

- *Data loading*: The user can load data from a locally stored CSV file, by specifying a set of tooltips that associate field names with descriptive labels, and produce the visualization by a simple method (*plot_points_on_map*) invocation. Additional features, such as selecting a subset

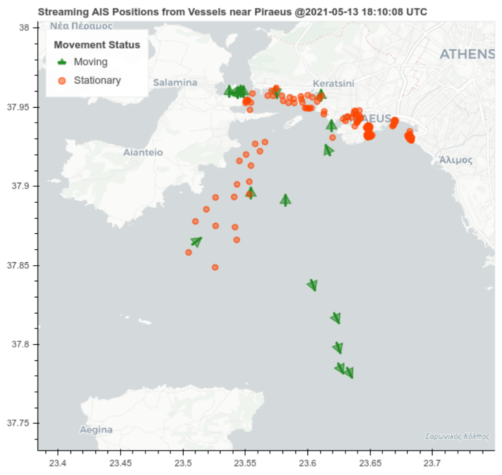
[2]ColumnDataSource, Bokeh Documentation

[3]According to Bokeh Documentation each model/widget/filter is assigned with its own unique identifier.

[4]The dataset is publicly available at research.microsoft.com


Temporal Horizon: 10 Jul 2018 13:00:00.000 .. 10 Jul 2018 15:29:47.000
CoG: 0
MDM '21 - ST_Visions Demo - Showing 20000 out of 30000 records
GPS Locations
Megara
Salamina
Chaidari
ATHEN
Aianteio
Aegina
© OpenStreetMap contributors,© CartoDB

(a) Colorization and filtering


Streaming AIS Positions from Vessels near Piraeus @2021-05-13 18:10:08 UTC
Movement Status
Moving
Stationary
Salamina
Keratsini
ATHENS
Aianteio
Άλιμος
Aegina

(b) Streaming data

Fig. 3: ST_VISIONS demonstration scenarios

of the records for visualization, adding a title, etc., are also supported. Moreover, various data sources and types are supported.

- *Colorization and data filtering*: After loading a set of points on the map, we select an attribute (e.g., speed) for colorization. Furthermore, we equip the visualization with a slidebar enabling filtering by time. Moreover, we add a filter on another attribute (e.g., heading or acceleration) to filter trajectories using both filters interactively (Fig. 3a).
- *Streaming data*: Having a data stream that continuously stores current locations to a (PostGIS) database, we obtain the latest positions per moving object by getting a view from it at fixed intervals (e.g., every 5 sec). For instance, Fig. 3b depicts the latest vessel positions within the coverage of an AIS antenna located at the University of Piraeus[5], and they are colorized according to their mobility status (stopped – red, moving – green), as illustrated in the corresponding legend. The depicted information is updated based on the synchronization used by ST_VISIONS. Due to ST_VISIONS' synchronization mechanisms, the points' attributes are updated accordingly, with the legend always reflecting the current view.

The code of ST_VISIONS, as well as code for demonstrating more scenarios both on Python Notebook and Server are available at: https://github.com/DataStories-UniPi/ST-Visions.

IV. CONCLUSIONS

In this paper, we demonstrated ST_VISIONS, an easy-to-use Python library for interactive visualizations of spatial and spatio-temporal datasets. ST_VISIONS offers a developer-friendly way to visualize geographical data, by hiding many of the details of the underlying visualization library (in our case Bokeh). This speeds up a cumbersome task that many data analysts need to confront, namely the creation of interactive visualizations. ST_VISIONS has several salient features: support for complex geometry types, advanced filtering mechanisms using multiple filters, data acquisition from streams (Kafka), complex visualizations (such as Choropleth maps), and support for multiple datasets either on the same canvas or by arrangement in a grid-like structure.

ACKNOWLEDGMENT

This work was supported by EU project VesselAI (Grant Agreement No 957237), by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Innovation (GSRI) under Grant Agreement No 1667, by project i4Sea (grant T1EDK-03268) funded by the European Regional Development Fund of the EU and Greek national funds (through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call Research-Create-Innovate), and by the 2018 National Funds Programme of the GSRT.

REFERENCES

[1] G. A. Vouros *et al.*, Eds., *Big Data Analytics for Time-Critical Mobility Forecasting, From Raw Data to Trajectory-Oriented Mobility Analytics in the Aviation and Maritime Domains*. Springer, 2020.
[2] N. Pelekis and Y. Theodoridis, *Mobility Data Management and Exploration*. Springer, 2014.
[3] N. V. Andrienko and G. L. Andrienko, "Spatio-temporal visual analytics: A vision for 2020s," *J. Spatial Inf. Sci.*, vol. 20, no. 1, pp. 87–95, 2020.
[4] A. Graser, "MovingPandas: Efficient structures for movement data in python," *GI_Forum – Journal of Geographic Information Science*, vol. 7, no. 1, pp. 54–68, 2019.
[5] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma, "Understanding mobility based on GPS data," in *UbiComp*, vol. 344. ACM, 2008, pp. 312–321.
[6] Y. Zheng, L. Zhang, X. Xie, and W. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *WWW*. ACM, 2009, pp. 791–800.
[7] Y. Zheng, X. Xie, and W. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Engineering Bulletin*, vol. 33, no. 2, pp. 32–39, 2010.
[8] Y. Kontoulis, A. Tritsarolis, and Y. Theodoridis, "UniPi AIS data 2018," Feb. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4498410

[5]The stream is accessible at http://datastories.org/unipi-ais