Outlier Detection and Cleaning in Trajectories: A Benchmark of Existing Tools

Mariana M G Duarte[1], Mahmoud Sakr[1,2]

[1]*Université Libre de Bruxelles, Brussels, Belgium*
[2]*Ain Shams University, Cairo, Egypt*

Abstract
Outliers can affect trajectory analysis as they represent errors. There are two outlier detection categories, one focusing on a collection of trajectories, where one whole trajectory can be an outlier and another on points inside individual trajectories. In this paper, we focus on the latter. We benchmark existing open-source libraries and compare their efficiency and accuracy in cleaning outliers. To compare the accuracy, we present a method to build ground truth using the other sensor data in a multi-sensor environment.

Keywords
Outlier Detection, Trajectory cleaning, Trajectory Preprocessing, Benchmark, Programming Libraries

1. Introduction

Outlier or noise detection is an essential step in trajectory cleaning, as it helps to identify points in the data that deviate significantly from the expected pattern. Moving object trajectory data comes with many outliers due to sensor and connectivity problems. Outliers in trajectory data can lead to misleading analysis results and inaccurate decision-making, which can have significant consequences for businesses, governments, and individuals. This type of data problem affects the accuracy of analytic functions, e.g., trajectory similarity [1, 2]. As such, outlier mining is an essential function in mobility data management systems [3, 4].

There are two outlier detection categories, one focusing on a collection of trajectories, where a trajectory can be an outlier and another on points inside individual trajectories. In this paper, we focus on the latter.

We benchmark open-source libraries and compare their efficiency and accuracy in detecting and cleaning outliers. The main contributions of this work are:

- A automated method for generating ground truth in multi-sensor data.
- A benchmark of libraries, including movetk [5], movingpandas[6], scikit-mobility[7], Ptrail [8], PyMove [9], Argosfilter[10] and Stmove [11] focusing on outlier detection. The benchmark considers the user perspective. That is, the context is to compare the offering of the existing libraries for end users rather than comparing their algorithms and implementation aspects.

Outline. The rest of the paper is structured as follows. In Section 2 we present essential concepts for outlier detection. Section 3 surveys the state of art. A benchmark using real data is the subject of Section 4. In Section 5, we discuss our findings and we conclude.

2. Outlier Detection

There are a variety of outlier detection techniques. In [12], the authors divide the methods into three categories: mean/median, Kalman and Particle Filters, and, Heuristics-Based Outlier Detection.

Distance-based algorithms compare each point in the trajectory to its neighbors and select the points that are significantly further away than expected. Methods based on the mean or Median Filters replace points compared to the measurements done at preceding points in time. These algorithms are simple and practical for detecting single outliers. Nonetheless, these techniques depend on the number of predecessors compared to the mean. Multiple successive outlier points can affect the accuracy of the outcome trajectory. These algorithms are not always effective at detecting multiple successive outlier points, which can affect the accuracy of the output trajectory. More advanced algorithms or methods may be needed to detect and correct outliers in these cases.

Statistic-based detection and correction methods include the Kalman filter (KF). It is a well established method [13] used to smooth point series. This algorithm estimates missing points based on previously observed values that might have measurement errors. In [13], the

Proceedings of the Workshop on Big Mobility Data Analytics (BMDA) co-located with EDBT/ICDT 2023 Joint Conference (March 28-31, 2023), Ioannina, Greece
mariana.machado.garcez.duarte@ulb.be (M. M. G. Duarte); mahmoud.sakr@ulb.be (M. Sakr)

authors mention the advantages of the KF and its derivatives is its recursive aspect, which can be used in real-time. It is also widely used due to its simplicity and capability to provide accurate estimations and prediction results.

Particle filters (PF) uses a set of randomly generated particles to represent the possible states of the system and update the particles based on observed data. In contrast to KF, they are not restricted to Gaussian distribution of errors, which makes them applicable in a wider range of noisy data. PF can however be computationally intensive, and thus not commonly implemented in trajectory libraries. Additionally, like KF, PF are sensitive to the first measurement in the trajectory, and their accuracy can be reduced if the first point is an anomaly[12, 14, 15].

The Hampel Filter (HF) detects and replaces outliers in trajectories with estimates via the Hampel identifier. The HF expresses a conventional heuristic that almost all values lie within three standard deviations of the mean. For each trajectory, the method calculates the median of a sliding window and adjacent points on each side of the trajectory. The HF estimates the standard deviation of each point about its window median using the median absolute deviation. If a measurement differs from the median by more than the threshold, the filter replaces the sample with the median.

Finally, heuristic-based techniques focus more on detection, rather than correction. For instance, [12] does not replace outlier points with estimated values, but instead removes them from the trajectory. Common heuristics are based on the speed. The idea is that if the speed/speed change rate is significantly higher than a given threshold and a proportion of the points in the entire trajectory, the point is removed. This approach has the advantage of not introducing any estimated values into the trajectory, but it can lead to significant data loss.

In [5], a new method category based on physical movement properties is introduced, such as speed (Optimal Speed-bounded) and acceleration (Optimal Acceleration-bounded). This method defines limits on the minimum and maximum allowed values for these properties, and uses them to determine whether a point in the trajectory is consistent with the model. The limits are minimum v- and maximum speed v+, minimum a-, and maximum acceleration a+. It follows the definition that from one point to its successor, there should always be inside [v, v+] and [a, a+]. In addition, a trajectory T = ⟨p1, . . . , pn ⟩ is consistent with the model if and only if there exists at least one point in a path such that the measurement coincides with the point and the speed and acceleration are inside speed and acceleration bounds. The method defines a reachable region as a cone, i.e. given the physical boundaries, it is possible to reach the cone from point pi to pi+1. Additionally, If a trajectory T is consistent, then is so any subtrajectory. In opposition, it is not necessarily possible to construct a trajectory from the concatenation of two consistent subtrajectories: the concatenation ⟨p1, . . . , pn = q1, . . . , qm⟩ of two consistent subsequences T = ⟨p1, . . . , pn ⟩ and U = ⟨q1, . . . , qm⟩ with pn = q1 is not necessarily consistent. Joining these subtrajectories can reproduce inconsistent points—especially when considering an acceleration-bound model. The speed of two points can infer two accelerations for the same position. The model is called concatenable if it is possible to join both sub-trajectories respecting the bounds.

In the next Session, we relate each of these methods to state-of-the-art libraries.

3. State of Technology

In this section of the paper, we will benchmark the available libraries that offer trajectory outlier detection and correction. We also highlight the algorithms and methods used by each library.

MovingPandas[1] [6] is a Phyton library for trajectories of moving objects. Data can be represented in Pandas [16], GeoPandas [17], HoloViz [18], CSV, GIS file formats, JSON, and geoJSON. MovingPandas implements structures for movement data in Python for interaction and analysis of movement. This library has many trajectory manipulation functions. Focusing on the outlier detection, this library implements KF. For outlier detection, MovingPandas uses the KF algorithm, which is implemented using the Stone Soup software [19].

Scikit-mobility[2] [7] is a Python library. It extends Pandas [16]. Scikit-mobility offers functions for prepossessing and cleaning trajectory and analysis. The library chosen method for outlier detection is heuristic filtering, based on the speed and a given threshold. This approach can be effective for identifying points in the trajectory that deviate significantly from the expected pattern but may not be as accurate as other methods that use estimation to deal with outlier points.

Ptrail [3] [8]is a Python package that uses parallel computation and vectorization, making it suitable for large datasets. It offers several preprocessing steps, such as feature extraction, filtering, interpolation and outlier detection. Ptrail removes outliers using a Hampel filter [20] based on the distance and speed of the ships between consecutive points. For each trajectory, the method calculates the median of a sliding window and adjacent points on each side of the trajectory. The HF also estimates the standard deviation of each point about its window median using the median absolute deviation. If a measurement differs from the median by more than the

[1]https://github.com/anitagraser/movingpandas-examples
[2]https://github.com/scikit-mobility/scikit-mobility
[3]https://github.com/YakshHaranwala/PTRAIL

threshold multiplied by the standard deviation, the filter replaces the sample with the median.

PyMove [4] [9, 21] is a Python library. It offers a range of operations for data preprocessing and pattern mining. PyMove also provides tools for data visualization, allowing users to explore and understand their data through various techniques and channels. PyMove detects outlier points considering the distance traveled, minimum and maximum speed.

Movetk[5] [5] is C++ library. It offers tools for constructing, cleaning and analyzing trajectory data. One of the key features of Movetk is its implementation of the Optimal Speed-bounded and Optimal Acceleration-bounded algorithms for outlier detection. Movetk implements a series of outlier detection methods based on Optimal Speed-bounded algorithm and the Optimal Acceleration-bounded algorithm. This can help to identify potential outlier points and can provide a more accurate and reliable way of detecting anomalies in the data. In addition to these algorithms, movetk implements a range of other methods for outlier detection, including greedy and smart greedy approaches. These methods build on the basic speed and acceleration-bounded algorithms and incorporate additional strategies and techniques to improve their performance and accuracy. For the greedy approach, movetk greedily builds a consistent subsequence by testing if the new mesurament is consistent with the last in the subsequence under the speed-bounded model (GSB) or acceleration-bounded model (GAB). In addition, the authors implement a Smart Greedy Speed/Acceleration-bounded methods (SGSB/SGAB). With SGSB and SGAB, multiple subsequences are tracked simultaneously. The next mesurament is added to all subsequeces that end in a consistent measurement; if there is no such subsequence, a new subsequence starting with the measurement is created. In the end, the longest subsequence is returned. Also, as a baseline, the library implements their interpretation of the method described in [12] as Local Greedy Speed-bounded (LGSB). In LGSB, a graph is constructed with a vertex per measurement. Two vertices are connected if their timestamps are successive in the original trajectory and they are consistent to the speed-bound. A measurement is added to the output, if and only if its vertex is in a connected component of a given size. LGSB does not guarantee that the complete output is consistent according to the speed bound [12].

Argosfilter [6] [10] is an R package that offers a set of functions for working with trajectory data. The outlier detection in Argosfilter uses two different methods: one based on speed and the other based on location. The speed-based method is similar to the one provided in [12], but the location-based method is based on the algorithm described in the paper [22]. This method uses a set of spatial constraints, such as a minimum distance between two points or a maximum distance from a reference point to identify and remove outliers from a trajectory.

Stmove[7] [11] is an R package library. It provides construction functions, filter, and outlier detection functions. For the outlier filter, the KF is applied.

In the next Session, we present experiments performed in the libraries presented above.

4. Experiments

In this Section, we present our method for constructing ground truth in Subsection 4.1. We present an analysis of the benchmark results in Subsection 4.2.

4.1. Constructing Ground-truth

Constructing a ground-truth for trajectory data analysis is a task that presents several challenges. The difficulties include the cost, accuracy, and scale of the ground-truth data that is required. Additionally, cleaning the data to remove errors and noise can be time-consuming and may not always produce reliable results.

We present a method that can be applied to the data originating from multi-sensor tracking technologies. Our proposed method offers a different approach to traditional methods for constructing ground-truth, such as manual annotation or data cleaning techniques like those described in [23, 24, 25]. It also allows for data integration from multiple sensors, increasing the overall accuracy of the ground-truth. The Algorithm compares the calculated speed and bearing with the recorded values present in the data. The input consist of speed and heading thresholds (tS and tH) and the file path. In line 6, for each point in the file, firstly, we check if the IDs are the same, i.e. the points belong to the same trajectory. Secondly, we calculate speed and bearing between the point and its successor. Later, we compare the newly calculate speed and bearing with the files' input in line 11. If the difference in speed or heading are bigger than the inputted thresholds, the point is considered an outlier and added to the output array.

Our method involves cross-referencing the data from multiple sensors in order to generate a more accurate representation of the true trajectory. We apply this method in data from modern multi-sensor trackings, such as GPS, AIS, ADS-B, Mode S, TCAS and FLARM sensors. By combining the data from multiple sources, we can increase the overall accuracy of the ground-truth in the presence of errors or noise in individual sensors. When comparing data from different sources, we can differentiate the data to see if there are any discrepancies or inconsistencies

[4]https://github.com/InsightLab/PyMove
[5]https://github.com/movetk/movetk
[6]https://cran.r-project.org/web/packages/argosfilter/argosfilter.pdf
[7]https://tinyurl.com/stmove

Algorithm 1: Ground-truth

```
Input: thresoldSpeed $tS$, thresoldHeading $tH$,FilePath $file$
Output: Outliers
Outliers ⟵ {};
file ⟵ ReadFile (FilePath);
file ⟵ OrderByIDandTimeStamp (file);
speed ⟵ 0;
bearing ⟵ 0;
foreach $point \in file.lenght$ do
    if file[point].id == file[point+1].id then
        speed ⟵ CalculateSpeed (point, point+1);
        bearing ⟵ CalculateBearing (point, point+1);
        if (abs (speed - file[point].sensorRecordedSpeed) > tS) OR
        (abs (bearing - file[point].sensorRecordedBearing) > tH) then
            Outliers.append(point);
        end
    end
end
return Outliers;
```

between multiple sources, such as different sensors or instruments. In addition, statistical tests can determine whether the data is consistent with a particular hypothesis or model. For example, we can check the calculated speed with the speed given in the data.

Our datasets for this benchmark consist of multi-sensor trajectory data. The first source is AIS data[8]: AIS is the location tracking system for sea vessels. Additionally, the AIS data is collected from a variety of different ships, providing a diverse set of trajectories to work with. This can be useful for testing the robustness of different algorithms and methods, and for evaluating their performance on different types of data. In this paper, we utilize total of 4.3 GB. The Fig. 1 shows the raw data and Fig. 2 shows filtering of outliers.

The raw data collected in the OpenSky Network [9] is stored in a historical database and used by researchers to study and improve air traffic control technologies and processes. The Fig. 3 shows the raw data.

We utilize two datasets from [26] [10] consists of GPS trajectory datasets of Southeast Asia from Singapore and Jakarta. Grab-Posisi covers over 1 million kilometers and contains more than 88 million points. The Images 4 and 5 show the raw data.

Table 1 shows the total number of records, as well as the number of outliers detected in the cross-check, the correct points, i.e., points that are not considered outliers, and the percentage of outliers over total points. The cross-check method was used to detect the outliers.

OpenSky data does not have a significant amount of outliers. One possible reason for the low number of outliers in the OpenSky data could be the accuracy of the sensors used to collect the data. If the sensors are highly accurate, there may be fewer errors or discrepancies in the data, resulting in fewer outliers. Additionally, it is possible that the data has been adjusted or corrected in some way to account for any errors or noise, further reducing the number of outliers. It is also worth considering the nature of the data itself.

[8]https://dma.dk/safety-at-sea/navigational-information/ais-data
[9]https://opensky-network.org
[10]https://engineering.grab.com/grab-posisi

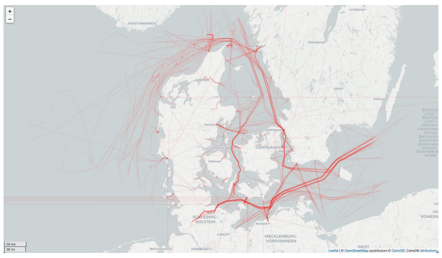
Figure 1: AIS raw dataset

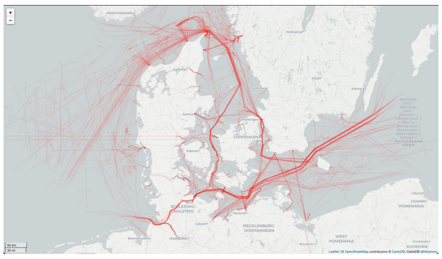
Figure 2: AIS with Ground-truth filtering

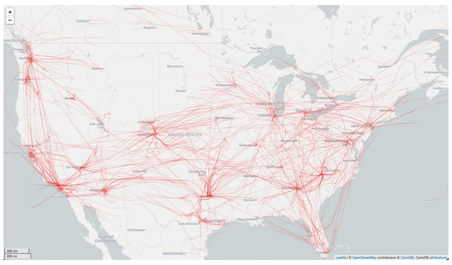
Figure 3: Open Sky raw dataset

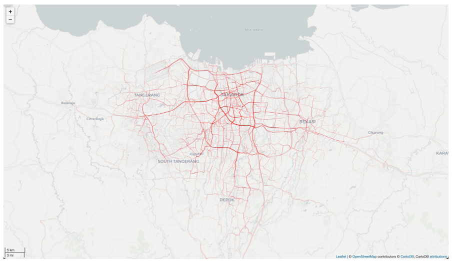
Figure 4: Jakarta raw dataset

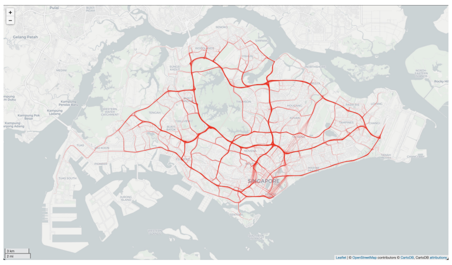

Figure 5: Singapore raw dataset

Table 1
Datasets total points, outliers, correct points, and percentage of outlier over total points

DataSet	Total Points	Outliers	Normal Points	%
city=Jakarta				
Part 0	5597262	28326	5568936	0.506%
Part 1	5602260	28591	5573669	0.510%
Part 2	5599282	28358	5570924	0.506%
Part 3	5598078	28287	5569791	0.505%
city=Singapore			0	
Part 0	3034553	22764	3011789	0.750%
Part 1	3029553	22640	3006913	0.747%
Part 2	3032527	22798	3009729	0.752%
OpenSky			0	
2020-05-25-00	723098	2	723096	0.000%
2022-02-28-22	2609917	0	2609917	0.000%
2022-06-27-01	2084075	0	2084075	0.000%
AIS			0	
aisdk_20110301	7778783	343	7778440	0.004%
aisdk-2022-11-17	8926371	224	8926147	0.003%
aisdk-2022-11-18	9009140	390	9008750	0.004%

4.2. Benchmark

The benchmark was run in seven libraries composed by Ptrail, MovingPandas, Scikit-Mobility,Pymove, Stmove, Argosfilter and MoveTk. These were described in Session 3. Fig. 6. The data used in the benchmark was decribed in Session 4.1. The benchmark code is publicly available[11].

The European aviation industry [27] developed methods to reduce carbon emissions to meet climate targets. An aircraft can fly an optimal flight path and use various technologies and infrastructure to minimize fuel consumption and carbon emissions. This might include using modern flight planning software and meteorological data to plan for a minimal amount of fuel, using green energy at the airport to power the aircraft on the ground, and using electric taxi solutions to minimize ground-based emissions. The aircraft would also fly an optimal climb phase, follow a fuel-efficient cruising level, and use idle thrust descent to minimize fuel consumption during descent, i.e., the aircraft should change its altitude as rarely as possible. Due to this standard, we consider the OpenSky dataset with only 2D rather than 3D data. However, it is important to note that the assumption may not hold true in all circumstances. It may be necessary to use 3D data or incorporate altitude data in some cases to analyze flight patterns accurately.

[11]https://github.com/marianaGarcez/OutlierDetectionLibraries

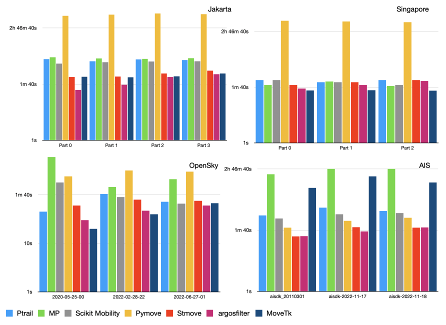

Figure 6: Run time

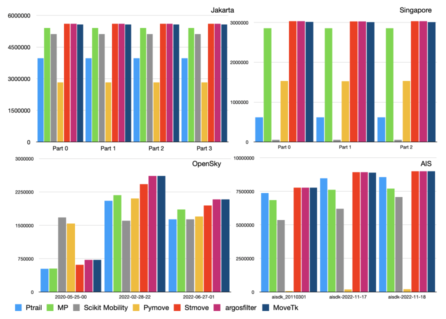

Figure 7: Amount of points after correction

Fig. 6 relates the run time of libraries. The top left graph corresponds to Jakarta datasets, divided by each file part 0, 1, 2, and 3. The top right graph represents the Singapore dataset and corresponding files. OpenSky dataset run time is at the bottom left and AIS is at the bottom right. It is possible to see that the first four libraries, i.e., Ptrail, Moving Pandas, Scikit Mobility and Pymove takes the longest to finish as they are implemented in Python. For Jakarta and Singapore datasets, Pymove presents the highest run time in the biggest datasets. Both R libraries have similar performance. Also, C++ has a similar performance to the R libraries.

For the OpenSky dataset, Pymove and Moving Pandas have similar performance in terms of run time, while MoveTk is slightly faster but has similar performance to Stmove and argosfilter.

The AIS data exhibits an exception in the run time of MoveTk, which may be due to the implemented process of reading the data. Its performance is comparable to that of Pymove. The run time of the various libraries varies depending on the dataset being used and the specific implementation of the library. It is important to consider the run time of the different options when selecting a library for detecting and erasing trajectory data.

Fig. 8 and 9 illustrate the run time for the Python and

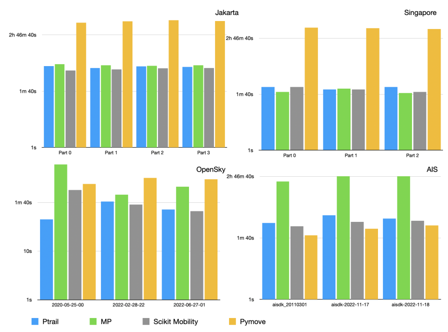

Figure 8: Run time for Python Libraries

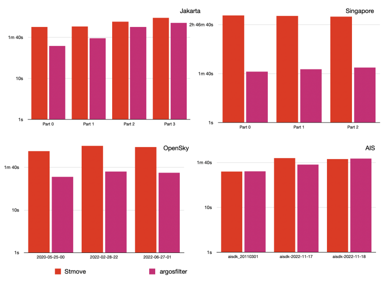

Figure 9: Run time for R libraries

R programming languages, respectively. For the Jakarta and Singapore datasets, the python libraries Ptrail, Moving Pandas and Scikit Mobility have a similar performance. It is possible to see a higher processing time for the Pymove library. For the OpenSky and AIS datasets, Ptrail and Scikit Mobility present a better performance. In contrast, Moving Pandas exhibit a higher processing time for both datasets. Pymove run time is comparable with Ptrail and Scikit Mobility only with AIS dataset. Overall, Pymove has a higher processing time compared to the other libraries. Both R libraries have a similar run time for the Jakarta and AIS datasets. Although, Argosfilter run time is consistently lower throughout all datasets. It is possible to see a considerable difference in performance for Singapore and OpenSky datasets.

As shown in Fig. 7, the number of outliers removed by each library varies depending on the dataset being used. These points consist of total points, i.e., outliers and normal points. The amount of points corresponds to the correct points detected by each library. For the Jakarta dataset, there is a consistent quantity of removed points, with Pymove removing the most points followed by Ptrail. The Singapore dataset is consistent between Stmove, argosfilter, MoveTk, and Moving Pandas, with Pymove, Ptrail, and Scikit Mobility removing points in increasing order. For the OpenSky dataset, Scikit Mobility and Ptrail remove the most points, while Moving Pandas, Stmove, argosfilters, and MoveTk have a consistent amount of removal. For the AIS dataset, all libraries seem consistent in the number of points removed, with the exception of Pymove, which removes a larger number of points. It is important to note that the number of removed points does not necessarily imply accuracy or precision. It is necessary to compare the actual outlier and normal points with the resulting trajectories in order to fully assess the accuracy and reliability of the trajectories cleaned by each library.

In order to analyze results, we utilize scores to compare the performance of the different libraries. Fig. 10, 11, 12, 13 show the Accuracy, precision, recall and F-1 scores of each library, respectively. The scores are based on comparison to the cross-check data and libraries output.

Fig. 10 shows the accuracy of each library. It is possible to observe that over all datasets, Movetk has the highest accuracy. For Jakarta dataset, Moving Pandas accuracy is comparable to MoveTk. As second, we observe Ptrail, argosfilter and Pymove. Scikit Mobility, has the lowest accurarcy in this dataset. For the Singapore dataset, Moving Pandas also has comparable accuracy to MoveTk. Libraries Scikit Mobility, Pymove, StMove, argosfilter have a comparable performance. Ptrail has the lowest accurarcy in this dataset. For OpenSky data, both argosfilter and Movetk have the highest accuracy. All other libraries have a comparable performance. With the exception of Ptrail which presents a low accuracy in the file '2020-02-25-00'. A high accuracy score means that the library is able to correctly identify a large proportion of the true outliers and normal points in these datasets. A high accuracy score indicates that the trajectory is likely to be accurate and useful for identifying patterns or anomalies in the data. While Ptrail and Scikit Mobility tend to have lower accuracy.

Precision measures the proportion of true positives among all positive predictions made by the library. A high precision score indicates that the library can correctly identify a large proportion of the true outliers. In contrast, a low precision score indicates that the library is prone to false positives. Looking at Fig. 11, we can see that the libraries Ptrail, Moving Pandas, Scikit Mobility, Pymove, and StMoe have a consistent level of precision across the different datasets. Except for StMove in the AIS dataset, the precision is the lowest among all libraries. Argos filter and MoveTk have lower precision scores throughout all datasets. Overall, these results suggest that Ptrail, Moving Pandas, Scikit Mobility, Pymove, and StMoe are relatively reliable in precision, with a low rate of false positives. On the other hand, Argos filter and MoveTk tend to have lower precision scores, indicating that they may be prone to false positives.

Recall is a measure of the proportion of true positives correctly identified by the library. A high recall score indicates that the library is able to correctly identify a

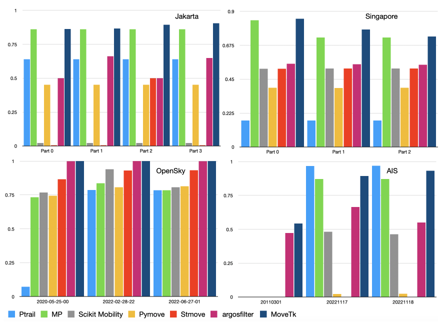

Figure 10: Accuracy

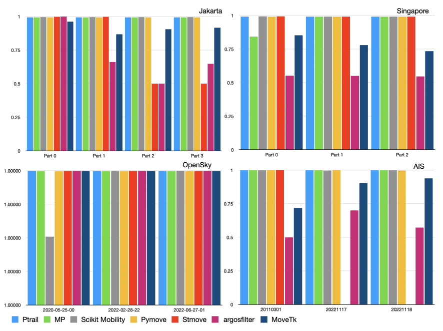

Figure 11: Precision

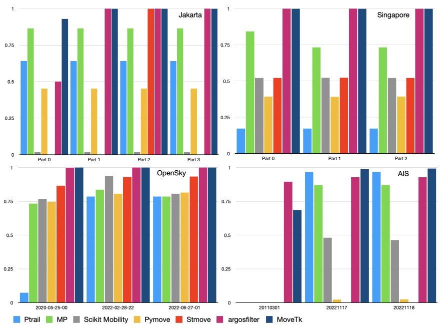

Figure 12: Recall

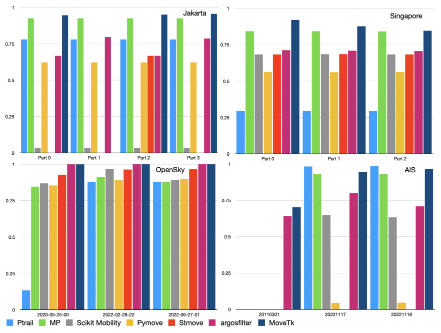

Figure 13: F-1 Score

large proportion of the true outliers in the dataset, while a low recall score indicates that the library is prone to false negatives. The Fig. 12 shows the recall. Movetk have the highest recall for almost all datasets and files. For the Jakarta dataset, Moving Pandas has the second highest recall, followed by Ptrail, argosfilter and Pymove. Scikit Mobility presents the lowest recall. The libraries MoveTk and argosfilter in Singapore dataset have the highest recall. Followed by Moving Pandas. Also, StMove and Scikit Mobility have comparable performance and, Ptrail presents the lowest performance. For the OpenSky dataset, the libraries have a similar performance. Still, MoveTk presents the highest recall and Ptrail the lowest. The AIS dataset has MoveTk as the highest recall. Ptrail, Moving pandas, Scikit Mobility, Pymove and Stmove have the lowest recall rate. These results suggest that MoveTk is the most reliable library regarding recall, with a low rate of false negatives. On the other hand, Ptrail, Moving pandas, Scikit Mobility, Pymove, and Stmove tend to have lower recall scores, indicating that they may be prone to false negatives. These results suggest that MoveTk is the most reliable library regarding the recall, with a low rate of missed true outliers. On the other hand, Ptrail, Moving Pandas, Scikit Mobility, Pymove, and Stmove tend to have lower recall scores, indicating that they may be prone to missing true outliers.

The F-1 score is a metric used for comparing the overall performance of the different libraries. The final metric we consider is the F-1 score, a combination of precision and recall, shown in Fig. 13. The MoveTk library has the biggest rate for the F1-score in all datasets. Followed by argosfilter library. In the Jakarta dataset, Moving Pandas has the second highest F-1 score. Followed by argosfilter and Pymove. In contrast, Stmove and Scikit Mobility have a low rate for the F-1 score. For the Singapore dataset, Moving Pandas has the third highest rate, followed by StMove. Ptrail has the lowest rate in this dataset. The OpenSky dataset, has a similar performance in all libraries. With the exception of Ptrail which has the lowest score in the first file. In AIS dataset, Movetk, Ptrail, Moving Pandas, argosfilter have the highest score. In contrast, Pymove and StMove have the lowest rate.

These results suggest that MoveTk [5] is the most reliable library for trajectory outlier removal, with a high level of accuracy, precision, and recall. Moving Pandas [6] also performs well in terms of F-1 score, particularly for the Jakarta and Singapore datasets. On the other hand, Ptrail, argosfilter, Pymove, Scikit Mobility, and Stmove tend to have lower F-1 scores, indicating that they may not be as reliable for removing outliers in these datasets. It is also important to consider the specific requirements of the application and the trade-offs between precision, recall and run time. Also, it is important to note that the libraries' performance varies based on the dataset. In addition, the ground-truth method can be applied in all of these datasets to assist in outlier detection.

5. Conclusion

In this paper, we presented an approach for constructing ground-truth that involves cross-referencing the data from multiple sensors. We applied this method to data from modern multi-sensor tracking technologies. We also evaluated the performance of several libraries for outlier removal in trajectory data, including Ptrail, Moving Pandas, Scikit Mobility, Pymove, Stmove, argosfilter, and MoveTk. Our results showed that MoveTk was the most reliable library, with a high level of accuracy, precision, and recall. Moving Pandas also performed well, particularly for the Jakarta and Singapore datasets. For future work, it would be interesting to explore other methods for constructing ground-truth in trajectory data, such as using machine learning techniques or integrating data from additional sources.

References

[1] A generic trajectory similarity operator in moving object databases, Egyptian Informatics Journal 18 (2017) 29–37.

[2] M. S. Bakli, M. A. Sakr, T. H. A. Soliman, A spatiotemporal algebra in hadoop for moving objects, Geo-spatial Information Science 21 (2018) 102–114.

[3] E. Zimányi, M. Sakr, A. Lesuisse, Mobilitydb: A mobility database based on postgresql and postgis, New York, NY, USA, 2020.

[4] E. Zimányi, M. Sakr, A. Lesuisse, M. Bakli, Mobilitydb: A mainstream moving object database system, 2019, pp. 206–209.

[5] B. Custers, M. V. D. Kerkhof, W. Meulemans, B. Speckmann, F. Staals, Maximum physically consistent trajectories, ACM Trans. Spatial Algorithms Syst. 7 (2021).

[6] A. Graser, Movingpandas: Efficient structures for movement data in python, GI Forum Volume 7 (2019) 54–68.

[7] L. Pappalardo, F. Simini, G. Barlacchi, R. Pellungrini, Scikit-mobility: a python library for the analysis, generation and risk assessment of mobility data, 2019.

[8] S. Haidri, Y. J. Haranwala, V. Bogorny, C. Renso, V. P. da Fonseca, A. Soares, Ptrail – a python package for parallel trajectory data preprocessing (2021).

[9] A. Sanches, Uma arquitetura e implementação do módulo de pré-processamento para biblioteca PyMove, Bachelor's thesis, UFC, 2019.

[10] C. Freitas, C. Lydersen, M. A. Fedak, K. M. Kovacs, A simple new algorithm to filter marine mammal argos locations, Marine Mammal Science (2008).

[11] D. P. Seidel, E. R. Dougherty, W. M. Getz, Exploratory movement analysis and report building with r package stmove, bioRxiv (2019).

[12] Y. Zheng, Trajectory data mining: An overview, ACM Trans. Intell. Syst. Technol. 6 (2015).

[13] C. Urrea, R. Agramonte, Kalman filter: Historical overview and review of its use in robotics 60 years after its creation (2021).

[14] S. H. Lee, M. West, Performance comparison of the distributed extended kalman filter and markov chain distributed particle filter, IFAC Proceedings (2010).

[15] J. Kotecha, P. Djuric, Gaussian particle filtering, IEEE Transactions on Signal Processing 51 (2003) 2592–2601.

[16] Wes McKinney, Data Structures for Statistical Computing in Python, in: Stéfan van der Walt, Jarrod Millman (Eds.), Proceedings of the 9th Python in Science Conference, 2010, pp. 56 – 61.

[17] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, Y. Huang, T-drive: Driving directions based on taxi trajectories, 2010.

[18] S. Yang, M. S. Madsen, J. A. Bednar, Holoviz: Visualization and interactive dashboards in python, KDD, 2022.

[19] P. Thomas, J. Barr, B. Balaji, K. White, An open source framework for tracking and state estimation, 2017.

[20] T. M, Hampel filter in python, 2021.

[21] A. F. D. Oliveira, Uma arquitetura e implementação do módulo de visualização para biblioteca PyMove, Bachelor's thesis, UFC, 2019.

[22] C. Freitas, C. Lydersen, M. A. Fedak, K. M. Kovacs, A simple new algorithm to filter marine mammal argos locations, Marine Mammal Science (2008).

[23] S. Wu, E. Zimanyi, M. Sakr, K. Torp, Semantic segmentation of ais trajectories for detecting complete fishing activities, IEEE Computer Society, 2022.

[24] X. Yang, L. Tang, Q. Li, A data cleaning method for big trace data using movement consistency, 2018.

[25] S. Moosavi, B. Omidvar-Tehrani, R. Ramnath, Trajectory annotation by discovering driving patterns, 2017.

[26] X. Huang, Y. Yin, S. Lim, G. Wang, B. Hu, J. Varadarajan, S. Zheng, A. Bulusu, R. Zimmermann, Grab-posisi: An extensive real-life gps trajectory dataset in southeast asia, in: SIGSPATIAL, New York, NY, USA, 2019.

[27] E. Control, The economics of aviation decarbonisation towards the 2030 green deal milestone (2022).