Linköping University | Department of Computer and Information Science
Master thesis, 30 ECTS | Computer Science
202018 | LIU-IDA/LITH-EX-A--2018/XXX--SE

Gaussian Process Regression based GPS Variance Estimation and Trajectory Forecasting

Regression med Gaussiska Processer för Estimering av GPS Varians och Trajektoriebaserade Tidtabellsprognoser

Linus Kortesalmi

Supervisor : Mattias Tiger
Examiner : Fredrik Heintz

External supervisor : Simon Johansson

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

`Abstract.tex`

Acknowledgments

`Acknowledgments.tex`

Contents

List of Figures

Todo list

1 Introduction

1.1 Motivation

This question explores what kind of data the dataset contains. The dataset is a novel dataset which has not been worked on before. The provided documentation is minimal, which makes this question non-trivial and the insights from the question even more valuable.

Work In Progress

1.2 Aim

The aim of this thesis project is to explore a novel dataset and formulate problems and solution suggestions which could be interesting to not only the dataset provider but also in a broader context. The dataset is currently used to provide time forecasts on the public transportation available in Östergötland county.

1.3 Research Questions

The specific research questions treated in this thesis project are presented in this section. They are divided into three groups: *Metadata Questions*, *Higher-Level Problem Questions*, and *Problem Specific Questions*. The *Metadata Questions* investigate and explore the provided dataset in depth, using Exploratory Data Analysis (EDA). They cover issues such as pre-processing, data noise and outliers. These questions can be seen as a pre-study before machine learning techniques are applied. The insights from these questions are the basis for the *Higher-Level Problem Questions*. The *Higher-Level Problem Questions* formulate various high-level problems by using the information available both internally in the dataset and externally from sources outside the provided dataset. High-level problems are problems which are not inherent in the dataset but rather problems which can be solved by using the information available in the dataset. They also aim to suggest solutions for these formulated problems. The *Problem Specific Questions* are questions related to a specific problem and solution described by the *Higher-Level Problem Questions*.

Metadata Questions:

1. *What kind of information is available in the dataset provided by Östgötatrafiken AB?*
   What kind of data does the dataset contain? Which features exist in the dataset,

such as different event types, event parameters, and event structures? What does these features mean and how do they relate to each other? Are some features more important than others? Are there any features missing from the given dataset?

2. *What pre-processing needs to be done in order to solve higher-level problems?*
   The higher-level problems are here defined as problems which are not inherent inside the dataset but rather problems which can be solved by using the dataset. The pre-processing focuses instead of solving problems inherent in the dataset, such as processing events and extracting relevant information from them or building a knowledge base by looking at the order of events. What are some pre-processing methods that could be applied to this dataset? How can the raw data in the dataset be transformed to useful features? How does the ordering of the raw data affect the solutions for higher-level problems? How can the raw data be transformed into trajectories? How can information from different event types be added to the trajectories?

3. *How can noisy measurements be detected?*
   Noisy measurements can affect the solutions for higher-level problems negatively. Various anomaly detection algorithms can be applied to detect such cases, but they could also be solved using dataset-tailored algorithms. Which manual inspection methods could be used to detect noisy measurements? Which general anomaly detection algorithms could be applied? How could a dataset-tailored algorithm be implemented? How does the general anomaly detection algorithm differ from the dataset-tailored algorithm?

4. *How is the provided dataset related to external data sources and how can they be combined?*
   There are external data sources available which could complement the data in the provided dataset. This question explores if these sources could be combined in order to solve problems on a higher level. Which external sources exist and what further information could be gathered from the external data sources? How compatible is the data from the external sources with the data in the given dataset? Does the external sources contain any of the features that were deemed missing from the given dataset (in *Metadata Question* 1)?

Higher-Level Problem Questions:

1. *What interesting higher-level problems can be investigated and solved based on the available data?*
   The higher-level problems can utilise both the data available in the provided dataset and any complementary external data. What problems span over the areas of GPS Positioning and Trajectory Forecasting? What are the core problems for each area? How could the existing system that generated the dataset be improved?

2. *How can these problems be solved?*
   The solutions will tackle the core of each problem. Each solution will explicitly state if there is a baseline available for comparison or if one could be easily created. How can problems with GPS Positioning and Trajectory Forecasting be solved?

3. *How does solutions to these problems benefit society and the industry?*
   This question analyses the solutions in a broader context, e.g. from a societal or ethical point of view. What value do the solutions offer to the industry? How is consumer privacy affected by the solutions?

Problem Specific Questions

1. *How can the spatio-temporal varying GPS variance be estimated from sets of observed trajectories over extended periods of time?* Can recent Gaussian Processes Regression (GPR) based trajectory modelling approaches be used to solve this problem? How

can the GPR-based approach scale with multiple trajectory models? How can potential periodicity of the data be handled in the approach? How can the model be trained on extended periods of time?

2. *How can the approach to estimate the spatio-temporal varying GPS variance be evaluated?*
   What assumptions are made regarding the inherent noise of the data? How can kernels be evaluated if a GPR-based approach is applied? How can trajectories be evaluated? Which evaluation criteria can be applied to evaluate the approach?
3. *How can Trajectory Forecasting be implemented from sets of observed trajectories?*
   Can GRP-based trajectory modelling approaches be used to solve this problem? Which modifications need to be done to the approach in order to produce trajectory forecasts? How can features from observed trajectories over extended periods of time be used to improve the forecasts?
4. *How can the Trajectory Forecasting model be evaluated?*
   Can the existing forecasts from Östgötatrafiken AB be used to create an evaluation baseline? How can new forecasts be compared with the baseline? Which insights could be made from the information available in the output of the Trajectory Forecasting model? How can the model respond to immediate real-time changes? How can the performance of a single forecast be evaluated? How can the overall performance of the model be evaluated? Which evaluation criteria can be applied to evaluate the forecasts? Can the new model be combined with the existing model in order to produce more precise forecasts? Are there any benefits to return a probability distribution over arrival times compared to returning the expected arrival time?

1.4 Delimitations

The dataset is provided by Östgötatrafiken AB and is not available for public use. This thesis project will only focus on the bus data in the dataset, data from public transportation trains will be ignored. In order to support manual inspection of the data in the dataset, the data is filtered to only contain data from a certain geographical area.

1.5 Structure

Analysis of Opportunities Enabled by the Data in chapters...

2 Background

2.1 Gaussian Processes

In [7] everything we need to know about Gaussian Processes (GP) is written, good huh?

Obviously edit this and fill out the section!

2.2 Combining GPs

...

2.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a broad concept containing various techniques [1, 3, 4, 8, 9] for exploring and analysing data. Early popular techniques include *box plots* and *stem-and-leaf* displays. A stem-and-leaf plot takes numbers and splits them into two groups. The first group contains the leading digit(s) and the second group contains the trailing digit(s). Figure 2.1 is an example of a stem-and-leaf plot with one leading and one trailing digit. The grouping helps when sorting batches of data and visualising important features, without losing the information of every single data point used [9].

15, 17, 19, 21, 22, 24, 30

Stem	Leaf
1	5 7 9
2	1 2 4
3	0

Figure 2.1: Example of a stem-and-leaf plot. The numbers above the plot is the input. The first digit of the number is the *stem*, the following digits are the *leafs*.

EDA can be seen as applying tools and statistics to analyse data in a meaningful way, e.g., it could be applied to the detection of outliers, smoothing the data, and performing a variance analysis [1, 4, 8, 9]. EDA can also reveal subtle practical problems with the chosen model that can easily be missed when performing statistical theory analysis of the model [3].

In [8], Tukey describes how EDA can be used to answer research questions such as "What is the age distribution for the Vietnamese population?" and "Are there differences in the annual household per capita expenditures between the rural and urban populations in Vietnam?". Tukey uses plots to compare different groups and estimators to quantify the difference. For example, the sample mean estimator, or the *winsorised* mean can be used [8]. The winsorised mean handles the case where tails of a distribution dominates the value space. This would cause the sample mean estimator to poorly reflect on the "typical" data point, as it is skewed by the small tail population [8].

In [9], Velleman et al. present different EDA techniques and highlights four key areas of EDA: displays (plots), residuals, re-expressions and resistance. Residuals is what remains after data analysis is performed. Residuals could, for example, be what remains after fitting data to a model (the error of the fit) [9]. Re-expression is the notion of applying mathematical functions to the data. Re-expressing the data can help with the data analysis [4, 9]. Examples of mathematical functions that can be applied are: logarithm, square root, reciprocal square function or generally raising the data to some power p. Resistance is the concept that outliers should not disproportionately affect the data analysis [4, 9]. For example, the winsorised mean estimator would be less sensitive to localised misbehaviour than the sample mean estimator [8].

Smoothing data is important for many different applications [2, 5, 6, 9]. This can, for example, be done by applying *running median smoothers*. The running median smoothers go through all the data points in sequence and calculate only the median for the n closest values near each point [9]. Another approach is the *running weighted average* [9]. Instead of taking the median of the n values, the average is calculated. The average can also be weighted with different functions, like hanning smoothing [9]. The hanning smoothing for three data points is shown in Eq. 2.1. It is worth noting that a single outlier will heavily affect the hanning smoothing and that in practice it is common to first apply a running median smoothing to remove outliers [9].

$$\hat{y}_t = \frac{1}{4}y_{t-1} + \frac{1}{2}y_t + \frac{1}{4}y_{t+1} \tag{2.1}$$

3 Data

This chapter describes the spatio-temporal dataset used in the thesis project. The dataset provider is briefly mentioned alongside the data gathering process, followed by the structure of the data. The structure of the data describes the different event types in the dataset and how they were used in the thesis project. After the basic characteristics of the dataset has been described the pre-processing steps applied are described and motivated. The pre-processing section also covers problems inherent in the dataset and the solutions employed to remedy them. The problems are visualised by real-world examples.

3.1 Background

The dataset was provided by Östgötatrafiken AB and contains GB of data. Östgötatrafiken AB is owned by Östergötland County and is responsible for the public transportation in the county. This thesis project only analysed the bus data available in the given data set. The dataset is a collection of documents, where one document represents a full day of data. A typical day has a document size of around GB.

300?

2.5?

Data Gathering

The process of gathering the data used in this thesis project can be generally described by the following simplified procedure:

1. Each Östgötatrafiken AB bus is running a system collecting data from sensors installed inside the bus.
2. The system collects the sensor data and transmits it to a central server or database.
3. A log containing all events for a full day is created and stored as a document in a collection.
4. The central server processes and analyses the data. The results from the data analysis is stored in the log.

Figure 3.1 illustrates the procedure. The collection of logs is the dataset used in this thesis project. The logs contain the GPS data from the buses and also events created by the "Internal

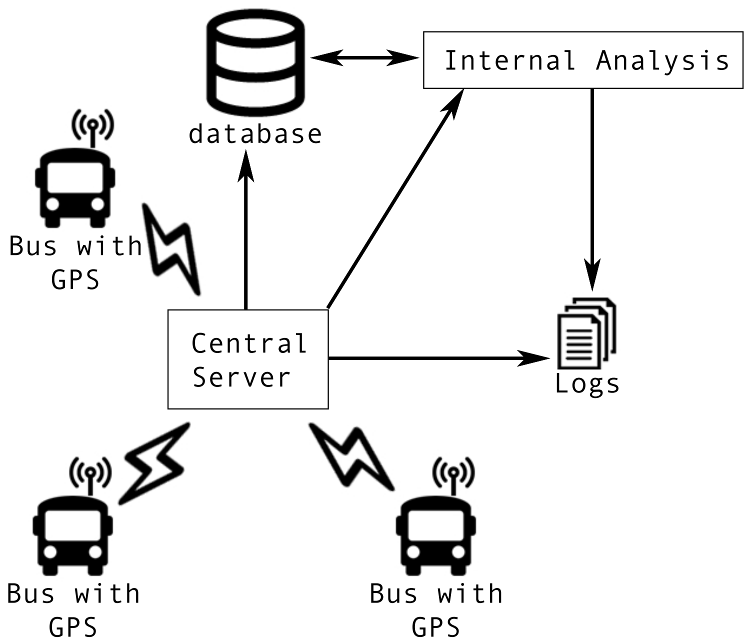

Figure 3.1: Simplified graph illustrating the data gathering process. Each bus is equipped with a GPS sensor and transmits its position to a central server/database. The dataset used in this thesis project is the log, which is a collection of documents. Each document contains the GPS data sent from all buses during a single day, together with data from the "Internal Analysis" component of the server.

Analysis" component in the system. The precise implementation of the "Internal Analysis" component is unknown.

3.2 Structure

A document in the dataset is made up of a large number of events representing a single day. A single day typically contains roughly 21 million events. Each event is represented by a single line of text. An event can be split into two groups: a header and a body. There are different types of events reported during the span of a single day. Each type has its own header and body structure.

Is it interesting to plot this exact number and its variance?

Event Example

Figure 3.2 illustrates an event with the event type `ObservedPositionEvent`. The header is defined as all the parameters before the `|||` separator. All the parameters after the separator is defined as the body of the event. In this example the header and body contain seven key parameters:

- *Timestamp:* A timestamp (`2018-02-16T11:53:56.0000000+01:00`), which is the timestamp from the system running on the bus.
- *Event Type:* The event type (`ObservedPositionEvent`).
- *Event ID:* The event id (`2623877798`). This is a number set by the system responsible for collecting the data from all buses. It is incremented for every event added to the log by either the database system or the "Internal Analysis" component in Figure 3.1.

```
                    timestamp                             event type             event ID
Header  2018-02-16T11:53:56.0000000+01:00|2|ObservedPositionEvent|2623877798|Normal|||

                                          vehicle ID                GPS
Body       0.1|Bus|otraf.se|9031005990005485|5485|58.4233551025391,15.5914640426636|

                                                       direction          speed
              58.4233551025391,15.5914640426636|168.899993896484|0|5482445
```

Figure 3.2: Example of a raw `ObservedPositionEvent` entry. The header and the body is separated by `|||`. Each parameter in the header and body is separated by a single `|`. Key parameters for the `ObservedPositionEvent` event type is highlighted.

- *Vehicle ID:* Unique ID for the bus transmitting its position.
- *GPS:* The GPS position of the bus in latitude and longitude.
- *Direction:* The direction of the bus.
- *Speed:* The current speed of the bus.

Event Types

The dataset contains 20 unique event types. Figure 3.3 visualises the distribution of event types for an arbitrary day in the dataset. The figure only gives an indication of what the true distribution could be, as it is computationally expensive to calculate the true distribution for the given dataset, due to its size. Knowledge about the true distribution is not required in order to reason about the event types. As the figure shows, the majority of events that occur are of the type `ObservedPositionEvent`, which is the event containing an updated GPS position for a vehicle.

Should we instead do a boxplot of the whole data-set? Will take a lot of time to create!

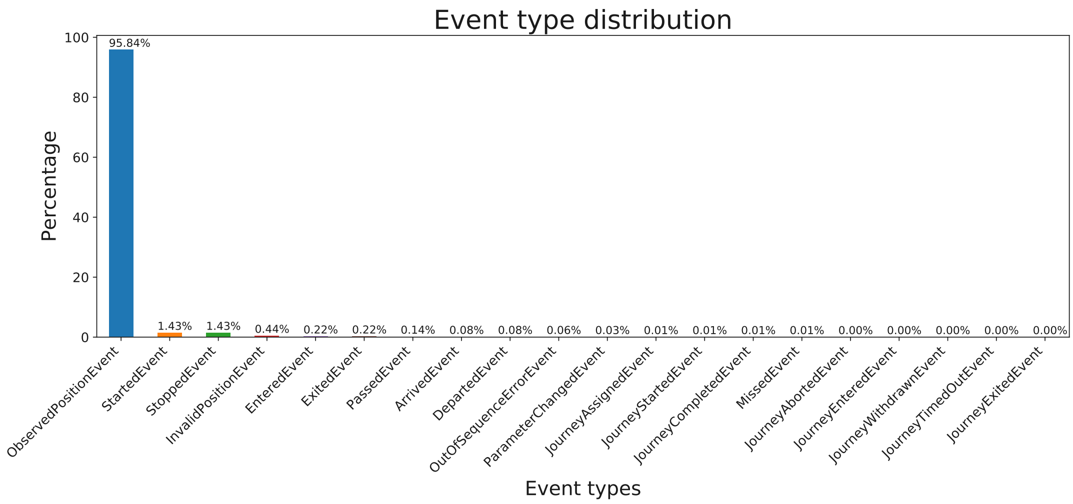

Figure 3.3: The distribution of event types for a random day in the dataset.

Of the 20 event types available in the dataset, this thesis project only used 12 of them. The events to use were chosen by analysing the log for a single day in great detail. Event types which occurred rarely and seemingly random were discarded, as no pattern could be determined for them. The `InvalidPositionEvent` type only contains the GPS position of the vehicle. The valid `ObservedPositionEvent` type also contains the *Speed* and *Direction* parameters. The GPS position of the `InvalidPositionEvent` events were always the same coordinates. In this thesis project the `InvalidPositionEvent` type was discarded, due to the missing parameters and static GPS position.

The 12 event types used in this thesis project were:

- *ObservedPositionEvent*: This event type contains the information highlighted in Figure 3.2. It is the most prevalent event type in the provided dataset. This event type is contextless, as it contains no information about which public transportation line the vehicle is currently serving, if any.

- *StartedEvent* and *StoppedEvent*: These two event types provide context to a sequence of observed position events. They denote when the vehicle has started or stopped moving, respectively. For example, they can be used to identify road intersections, bus stops, traffic or driver breaks.

- *EnteredEvent* and *ExitedEvent*: These two event types are used by the "Internal Analysis" component to identify bus stops. The *EnteredEvent* is produced by the system when the vehicle is within a certain predefined distance to a bus stop. The *ExitedEvent* is similarly produced when the vehicle leaves the predefined distance to the bus stop. These event types could, for example, be used in an algorithm which improves the bus stop detection.

- *PassedEvent*, *ArrivedEvent*, and *DepartedEvent*: These three event types are used to provide information regarding which bus stop a particular bus is at. The *PassedEvent* type denotes when a particular bus, serving a specific public transportation line, passed a bus line stop. It contains information about the predefined position of the stop, the public transportation line the particular bus is currently serving and the time of the passing. Similarly, the *ArrivedEvent* and *DepartedEvent* types denote when a particular bus arrived at or departed from a particular bus stop. These event types provide context to the observed positions. In this thesis project they are used to group a sequence of observed positions into a segment between two stops for a particular bus line.

- *ParameterChangedEvent*: The *ParameterChangedEvent* type is the most dynamic event type in the data set, e.g. it can be used to inform the system when the doors on a particular bus open or close or when a bus changes journeys. In this thesis project it is only used to identify bus lines and give context to observed positions.

- *JourneyStartedEvent* and *JourneyCompletedEvent*: The *JourneyStartedEvent* type is produced by the "Internal Analysis" component when a bus has reached the starting bus stop for a bus line. The *JourneyCompletedEvent* event type is produced when the bus has reached the final bus stop for a bus line.

- *JourneyAssignedEvent*: This event type is accompanied by a *ParameterChangedEvent* to denote when a bus changes its journey.

3.3 Pre-processing Events

The first step deemed necessary in order to use the data in the provided dataset was to transform it from strings to object with attributes. Figure 3.2 visualises which attributes an *ObservedPositionEvent* object contains. Similar structures were created for each of the mentioned event types.

During this pre-process step all the events from a vehicle type other than "Bus" were ignored. A *geo-fence* was applied in order to facilitate and support manual inspection and visualisation of the data in the dataset. A geo-fence is a virtual polygon which establishes a virtual perimeter for a real-world geographical area. The geo-fence applied in this thesis project is shown in Figure 3.4.

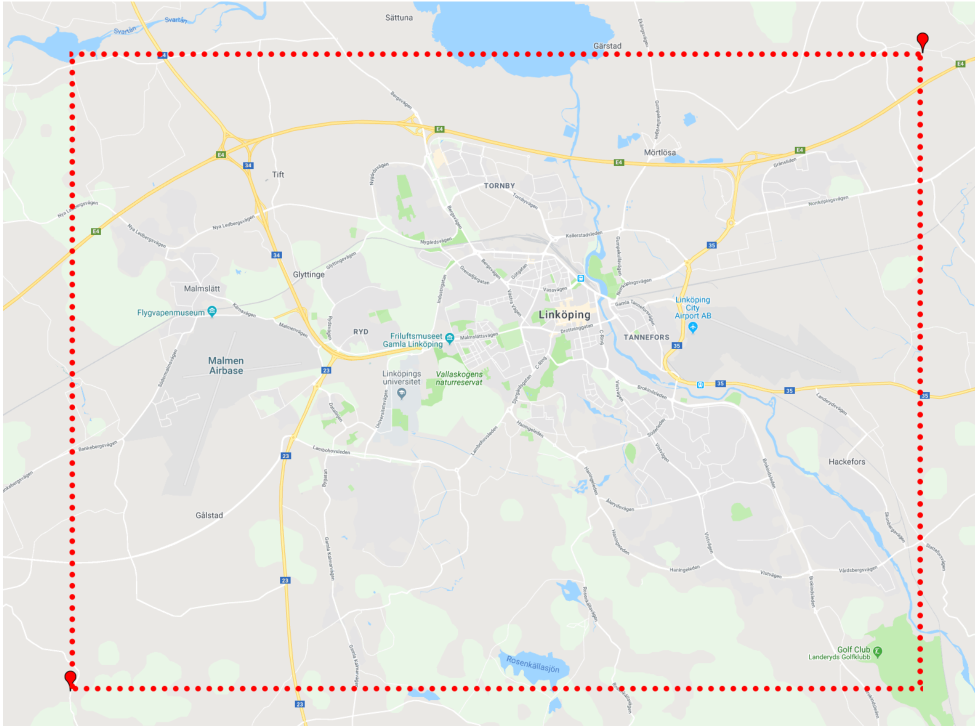

Figure 3.4: A geo-fence is constructed to filter out events occurring outside the virtual perimeter. The two red markers create a rectangular boundary, which is illustrated with the red-dotted line. The geographical area is the city of Linköping.

After the parsing and filtering step the idea was to provide context to *ObservedPositionEvents*. Only using this one contextless event type greatly reduces the span of potential problems one could solve with the provided dataset. Context was provided by constructing a finite-state machine.

Context-Providing Finite-State Machine

Finite-state machines are well-defined and can be modelled with simple algorithmic structures . The context-providing finite-state machine constructed in this thesis project is shown in Figure 3.5. The shown state machine is illustrating the best-case scenario, when the actual order of events is equal to the logical ordering of events, see Figure 3.6 for a real-world example. However, this is not always the case when working with real world data, as shown in Figures 3.7 and 3.8. Occasionally the timing of events gets mixed up, e.g. a bus in the "Started" state receives a *JourneyAssignedEvent* before it receives a *JourneyCompletedEvent*. This ordering breaks the logical ordering of events: a journey needs to be completed before a new one can assigned.

what is the formal definition?

source?

The problem is solved by partly changing the ordering of events from *Timestamp* to *Event ID*. This is a feasible approach when the data is batched into separate files, where one file is a full day of events. When processing data in real time the approach would have to be slightly altered. The *ObservedPositionEvents* would have to be placed in a temporary buffer once an anomaly is detected in the event ordering, e.g. *JourneyAssignedEvent* is received before *JourneyCompletedEvent* and the system is in the "Started" state. Once the *JourneyCompletedEvent* type is finally received, the data in the buffer could be retroactively added to the "Started" state. The "Started" state would then correctly contain all *ObservedPositionEvents* received from the starting bus stop to the final bus stop.

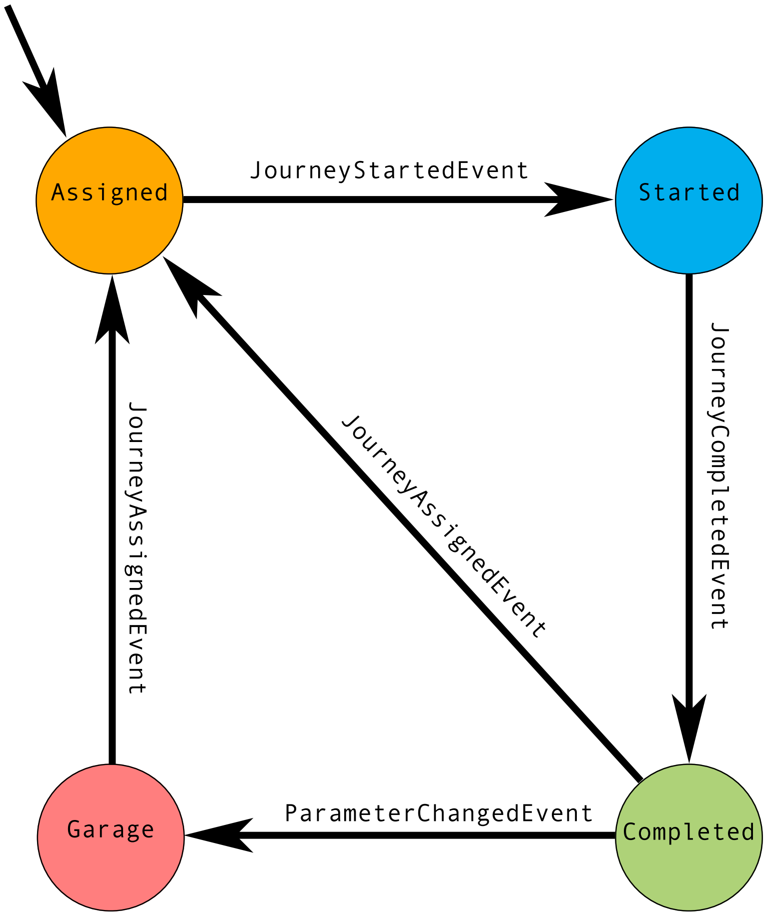

Figure 3.5: Finite-state machine providing context to *ObservedPositionEvents*. The constructed finite-state machine is simplified to illustrate the best-case scenario. The "Assigned" state is the starting state. *ObservedPositionEvents* are assigned to the current state the state machine is in.

Figure 3.6: Real-world scenario illustrating when events occur in a correct, logical ordering. The blue line to the left is *ObservedPositionEvents* in the "Started" state. Upon reaching the final bus stop for the line the state changes to "Completed". The *ObservedPositionEvents* for this state is drawn with a green line to denote the "Completed" state. The bus turns around and stops for a period of time until a new bus line is assigned to it. In this particular scenario, the bus is assigned the same bus line number, but in the opposite direction. The orange line denotes the *ObservedPositionEvents* in the "Assigned" state. Shortly after passing the first bus stop the orange line changes to blue (not shown in the image), which denotes the "Started" state.

Figure 3.7: Example illustrating when the real ordering of events breaks the logical ordering. The bus is assigned a new bus line long before reaching the final bus stop. The final bus stop is marked with a circle. The rectangle marks the position of the bus when it is assigned a new bus line. The last part of the journey (the path between the rectangle and the circle) is thus assigned to a new state "Assigned", instead of the actual, logical state "Started".

Unfortunately, it is not always the case that the *JourneyCompletedEvent* type is in an erroneous ordering. Occasionally the type is missing from the sequence of events due to either human errors or imprecise algorithms in the "Internal Analysis" component.

Human Error: Early Stopping

Figure 3.9 and 3.10 illustrate two scenarios when the *JourneyCompletedEvent* type for a started journey would be missing. In Figure 3.9, the bus driver is supposed to visit the three markers in order to complete the journey for a particular line. In Figure 3.10, the dashed line highlights the route of the bus line, while the blue line is the actual route the bus drove. In both these real-world examples, the bus drivers ignore the final stops of the journeys.

The "Internal Analysis" component never deems the journey as completed in these scenarios, which results in the *JourneyCompletedEvent* type never being produced. This scenario is easy to detect in historical data, but in real-time certain assumptions need to be made. For example, if the journey is compared with an average journey, the anomaly could be detected early. However, it would be uncertain if the anomaly is due to a journey being stopped early or if the bus driver took a wrong turn on the highway. A time constraint threshold would have to be introduced to the system in order to separate these two cases. In the case of a wrong turn, the bus would eventually converge to the average journey, while in the early-stopping scenario the journey would most likely never converge.

Imprecise Algorithms: Final Bus Stop Missed

This scenario occurs due to a mix of imprecision in the bus stop detection algorithm and human error. The scenario is illustrated in Figure 3.11. The bus driver completes the journey of a particular bus line and reaches the final bus stop. However, the "Internal Analysis" component does not detect that the bus stop has been reached. This is due to the bus driver stopping the bus slightly roughly 45 meters before the final bus stop. The bus stops there for

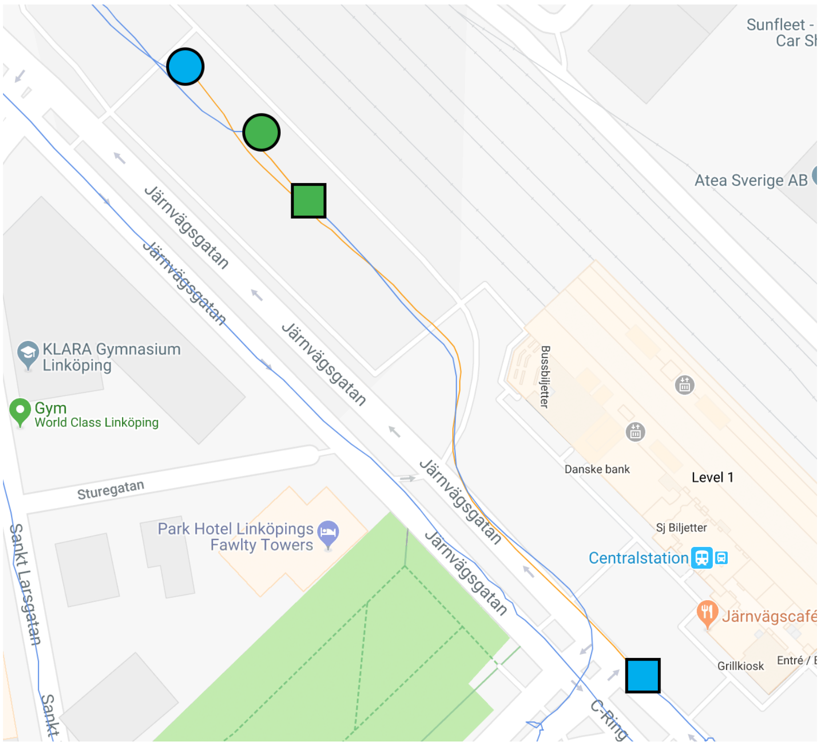

Figure 3.8: Another example illustrating two cases where a bus is assigned a new bus line before completing the started one. The two circles (green and blue) denote the final bus stops for the respective bus lines. One of the buses is assigned a new bus line at the blue rectangle in the bottom-right corner, long before reaching the final bus stop. The other bus is assigned a new bus line at the green rectangle, which is closer to the final bus stops. This example demonstrates that the assignment to a new bus line is independent of the distance to the final bus stop.

a few minutes, before it drives to the first bus stop for the new bus line number which was assigned.

This systematically occurs at certain bus stops, due to there being a "waiting" space commonly used by bus drivers while waiting for a new journey to be assigned. The scenario highlights a problem with the implemented bus stop detection algorithm. The bus stop detection algorithm can be improved to both handle these scenarios and yield more precise bus stop detection. An improved bus stop detection algorithm is proposed in Section **??**.

Bus Stops

Using the finite-state machine provides context to the *ObservedPositionEvent* types. The states introduced yield a simple way to visualise contextual paths, e.g. actual journeys for a particular bus line or the path a bus drives to start a journey under a new bus line number. However, the context-providing finite-state machine solution does not handle events about a bus arriving, departing or passing a bus stop on the journey. Handling this type of data could be a critical step in detecting imprecisions in the bus stop detection algorithm or early stopping due to human error. The "Started" state in the finite-state machine can easily be extended to not only include *ObservedPositionEvent* types, but also *ArrivedEvent*, *DepartedEvent*, and *PassedEvent* types. For example, a missed final bus stop could be identified by looking at all the bus stops added to the "Started" state for that journey and compare them to the bus stops in other journeys for that particular bus line.

Figure 3.9: Example of early stopping in a journey. The three markers are the final three stops of a particular bus line. Instead of following the pre-determined route of the bus line, the final three stops are skipped. This results in the journey never being deemed completed.

Figure 3.10: Another example of early stopping in a journey. The red dashed line is the planned route of the bus line. The blue line at "Nya Ledbergsvägen" is the actual route the bus drove. This results in the journey never being deemed completed, creating a erroneous ordering of contextual events.

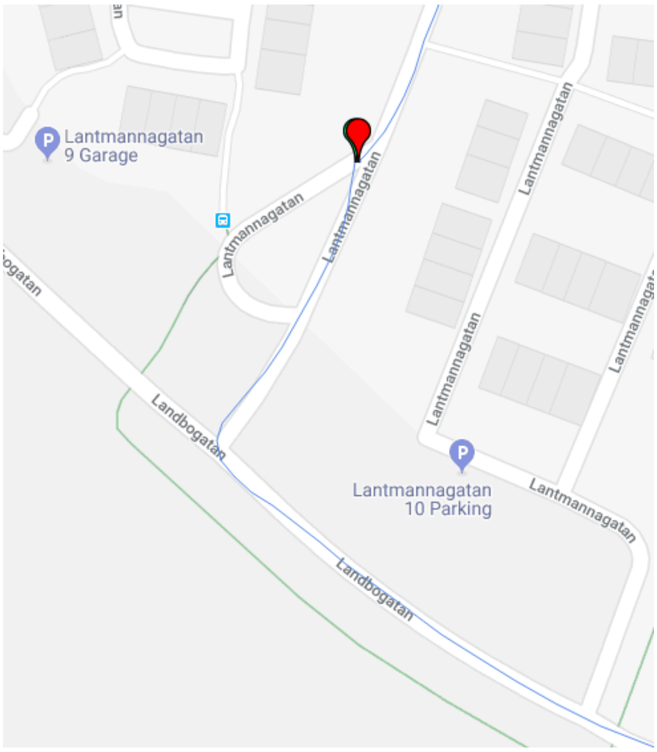

Figure 3.11: Real-world example of a bus driver stopping roughly 45 meters before reaching the final bus stop. The red marker is the GPS position where the bus stopped and the bus icon on the map is the pre-determined position of the bus stop. The bus stops there for a few minutes before it drives off to the first bus stop for a new journey. The bus detection algorithm systematically does not identify these cases.

Results

The results of the pre-processing step is a collection of journeys for each bus line. A journey consists of all the *ObservedPositionEvent*s sent by the bus while the finite-state machine was in the "Started" state and all the bus stops the bus arrived at, departed from or passed by. Erroneous event type ordering was solved by sorting the events based on *Event ID* and retroactively adding *ObservedPositionEvent* in the "Assigned" state to the "Started" state in the case of early journey assignment. Journeys with early stopping or missed final bus stops are still prevalent in the collection of journeys. These can be detected by analysing the bus stops registered during a journey. In this thesis, the faulty journeys will only be marked as anomalies and discarded. An index-tree is constructed to quickly access all journeys for a bus with a particular *Vehicle ID*. A number of high-level problems can now be formulated by using the two results from the pre-processing steps.

Discussion

The faulty journeys are only marked as anomalies and discarded in this thesis. The fraction of faulty journeys can be calculated and used as a baseline when comparing an improved bus stop detection algorithm with the existing one. Faulty journeys can be categorised based on the error in the journey, e.g. a missed starting bus stop is a different error compared with a missed final bus stop. The categorisation of faults could provide deeper insights into the dataset. For example, the insights could be used to improve the bus stop detection algorithm or identify journeys where a particular error occurs regularly.

4 GPS Variation Estimation

5 Discussion

6 Conclusion

Bibliography

[1] Luc Anselin. 'Interactive techniques and exploratory spatial data analysis'. In: *Geographical Information Systems: principles, techniques, management and applications* 1 (1999), pp. 251–264.

[2] Andrew P. Bradley. 'The use of the area under the ROC curve in the evaluation of machine learning algorithms'. In: *Pattern Recognition* 30.7 (1997), pp. 1145–1159. ISSN: 0031-3203. DOI: `https://doi.org/10.1016/S0031-3203(96)00142-2`. URL: `http://www.sciencedirect.com/science/article/pii/S0031320396001422`.

[3] Andrew Gelman. 'A Bayesian Formulation of Exploratory Data Analysis and Goodness-of-fit Testing*'. In: *International Statistical Review* 71.2 (2003), pp. 369–382. ISSN: 1751-5823. DOI: `10.1111/j.1751-5823.2003.tb00203.x`. URL: `http://dx.doi.org/10.1111/j.1751-5823.2003.tb00203.x`.

[4] David C. Hoaglin. 'John W. Tukey and Data Analysis'. In: *Statistical Science* 18.3 (2003), pp. 311–318. ISSN: 08834237. URL: `http://www.jstor.org/stable/3182748`.

[5] Bo Pang, Lillian Lee and Shivakumar Vaithyanathan. 'Thumbs Up?: Sentiment Classification Using Machine Learning Techniques'. In: *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*. EMNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 79–86. DOI: `10.3115/1118693.1118704`. URL: `https://doi.org/10.3115/1118693.1118704`.

[6] John R Quinlan et al. 'Learning with continuous classes'. In: *5th Australian joint conference on artificial intelligence*. Vol. 92. Singapore. 1992, pp. 343–348.

[7] Carl Edward Rasmussen. 'Gaussian Processes in Machine Learning'. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Ed. by Olivier Bousquet, Ulrike von Luxburg and Gunnar Rätsch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–71. ISBN: 978-3-540-28650-9. DOI: `10.1007/978-3-540-28650-9_4`. URL: `https://doi.org/10.1007/978-3-540-28650-9_4`.

[8] John W Tukey. *Exploratory data analysis*. Reading, Mass., 1977.

[9] Paul F Velleman and David C Hoaglin. *Applications, basics, and computing of exploratory data analysis*. Duxbury Press, 1981. ISBN: 0-87150-409-X. URL: `http://hdl.handle.net/1813/78`.