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 previously explored. 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 solutions which could be interesting not only to 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.

Broader context, som vadå?

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 pre-processing focuses 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 ordering 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 Binary Search

Binary search is a common algorithm for searching through a sequence for a target value. The algorithm utilises a binary tree for the search. The algorithm can be described by the following steps, from Knuth [5]:

1. If the tree has no root node, the search is unsuccessful. The target value is not in the tree, as the tree is empty.
2. If the root node has the target value, the search is successful. The position of the target value is found.
3. If the target value is smaller than the root value, the left side of the binary tree is explored. The algorithm returns to the first step of the procedure. This time the tree is the left half of the binary tree, with the new root node being the left child of the previous root node.
4. If the target value is greater than the root value, the right side of the binary tree is explored. The algorithm returns to the first step, but this time the tree is the right subtree of the binary tree. The new root node is the right child of the previous root node.

The procedure is illustrated in Figure 2.1. The algorithm has a complexity of $O(log_2 n)$, as the search space is in the binary case divided in two after each comparison. However, the algorithm requires a sorted sequence of values.

2.2 Gaussian Processes

Definition 1 is the formal definition of a Gaussian process (GP) given by Rasmussen et al. in [9].

Definition 1 *A Gaussian process is a collection of random variables, any infinite number of which have a joint Gaussian distribution.*

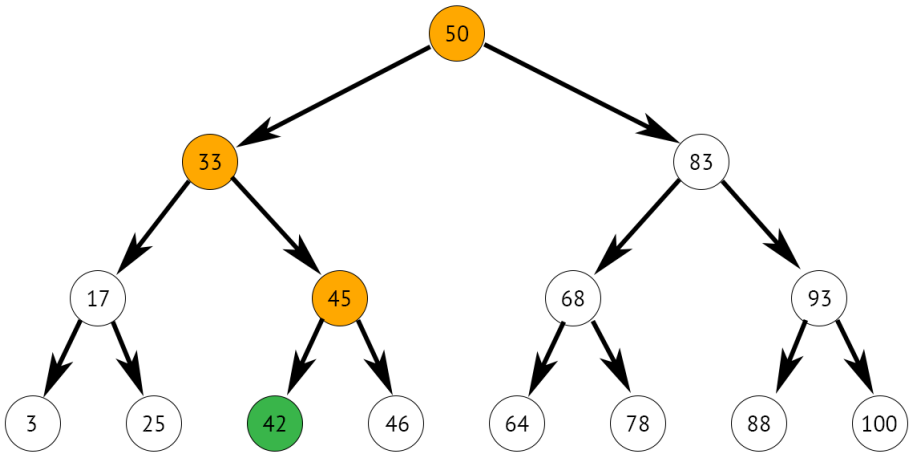


Figure 2.1: Example of a binary search algorithm run. The target value is 42. First the root node 50 is compared, but as $42 < 50$ the left subtree is explored. The new root node is 33 and $42 > 30$, so the right subtree is explored. The new root node is 45 and $42 < 45$. The algorithm should search the left subtree, which in this case only consists of one node, with the target value. The search is successful.

A GP is denoted by the equation

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')),$$

where

- $m(x) = \mathbb{E}[f(x)]$ is the mean function;
- $k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]$ is the covariance function. x and x' are two inputs.

A GP is thus completely described by its mean and covariance function, where the mean function is taken to be zero for notational simplicity [9]. It can be seen as a probability distribution over functions. The covariance function can be expressed in the form of a *kernel*, where kernels exhibit different characteristics. A comparison between common kernels is illustrated in Figure 2.2.

continue...

2.3 Combining GPs

...

2.4 Finite-State Machines

Finite-state machines (FSM) are well-defined mathematical models which can have numerous representations. The formal notation used here are from the excellent book "Automata and Computability" by Kozen [6]. A FSM is defined as states with potentially multiple transitions between each state. The FSM can only be in one given state at any time.

Formally a FSM is given by the structure

$$M = (Q, \Sigma, \delta, s, F),$$

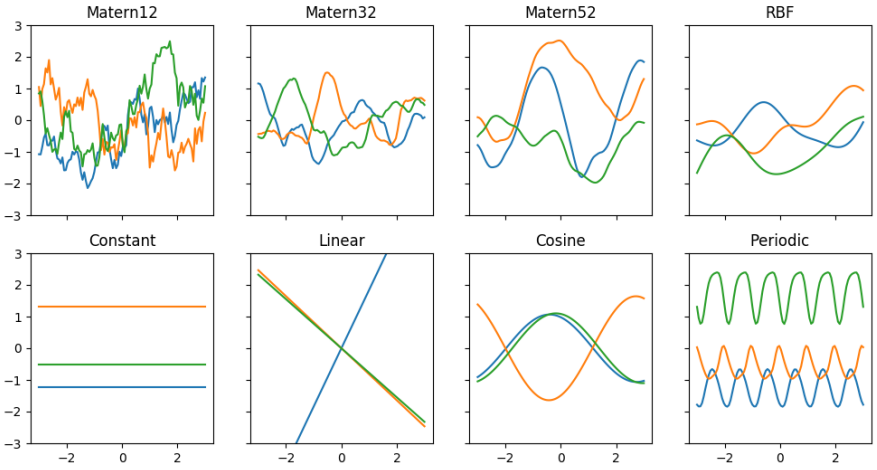


Figure 2.2: Samples from GP priors defined by common covariance functions. The functions exhibit different characteristics depending on the choice of kernel.

where

- Q is a finite set of states.
- Σ is the input alphabet of the FSM.
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function for the FSM. It can intuitively be seen as the function which tells the FSM which state to transition to in response to an input. For example, if M is in state q and receives input x it should move to state $\delta(q, x)$.
- $s \in Q$ is the start state.
- $F \subset Q$; where the elements of F are the final states of the FSM.

A FSM can be modelled with various representations [6], e.g, in tabular form, as a transition diagram or as a regular expression.

2.5 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a broad concept containing various techniques [1, 3, 4, 10, 11] 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 2.3 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 [11].

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, 10, 11]. 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 [10], 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 [10]. The

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

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

Figure 2.3: 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*.

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 [10].

In [11], 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) [11]. Re-expression is the notion of applying mathematical functions to the data. Re-expressing the data can help with the data analysis [4, 11]. 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, 11]. For example, the winsorised mean estimator would be less sensitive to localised misbehaviour than the sample mean estimator [10].

Smoothing data is important for many different applications [2, 7, 8, 11]. 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 [11]. Another approach is the *running weighted average* [11]. 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 [11]. 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 [11].

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

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. A pre-study of the dataset is conducted after the basic characteristics of the dataset have been described. The dataset pre-study analyses the data in-depth and identifies problems which occur because of the basic characteristics and their limitations. The problems are illustrated by real-world examples. The pre-processing section covers methods employed to transform the data into usable features. The pre-processing section also mentions solutions to some of the problems identified in the dataset pre-study.

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.

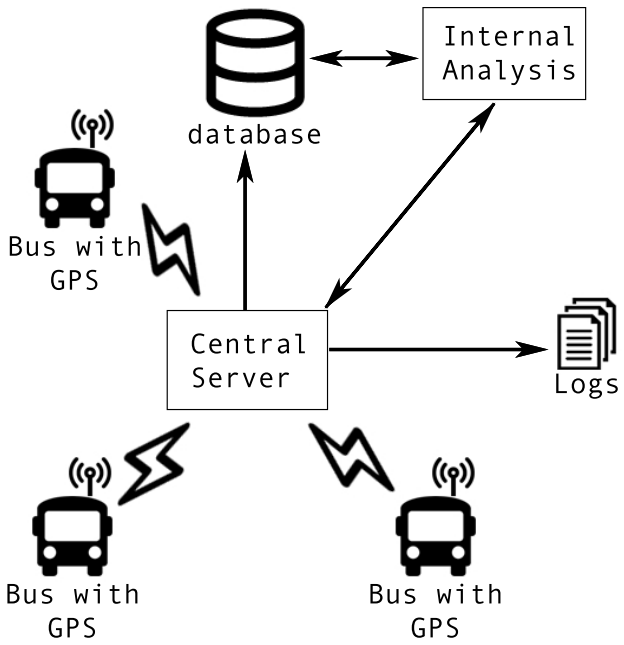


Figure 3.1: Simplified graph illustrating the data gathering process. Each bus is equipped with a GPS sensor and transmits its position to a central server, which is propagated to a database. The dataset used in this thesis project is a collection of logs, where each log is a collection of events. The logs serve as a historical timeline of all events occurring at the central server. Examples of events occurring are the central server receiving the GPS position of a bus or events produced by the "Internal Analysis" component.

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 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 parts: 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 of the type `ObservedPositionEvent`. The header is defined as all the parameters before the `|||` separator. All the parameters after the separator are 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`).

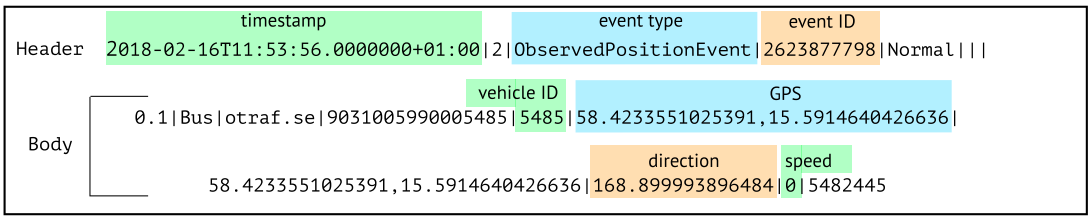


Figure 3.2: Example of a raw `ObservedPositionEvent` entry. The header and the body are separated by `|||`. Each parameter in the header and body is separated by a single `|`. Key parameters for the `ObservedPositionEvent` event type is highlighted.

- *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.
- *Vehicle ID:* Unique ID for the bus transmitting its position.
- *GPS:* The GPS position of the bus in latitude and longitude.
- *Direction:* The angle of rotation of the bus, in degrees.
- *Speed:* The current speed of the bus, in metres per second.

Event Types

The dataset contains 22 unique event types. Figure 3.3 visualises the distribution of event types for an arbitrary day in the dataset. Only event types from buses are visualised, events from trains are discarded. 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 dataset? Will take a lot of time to create!

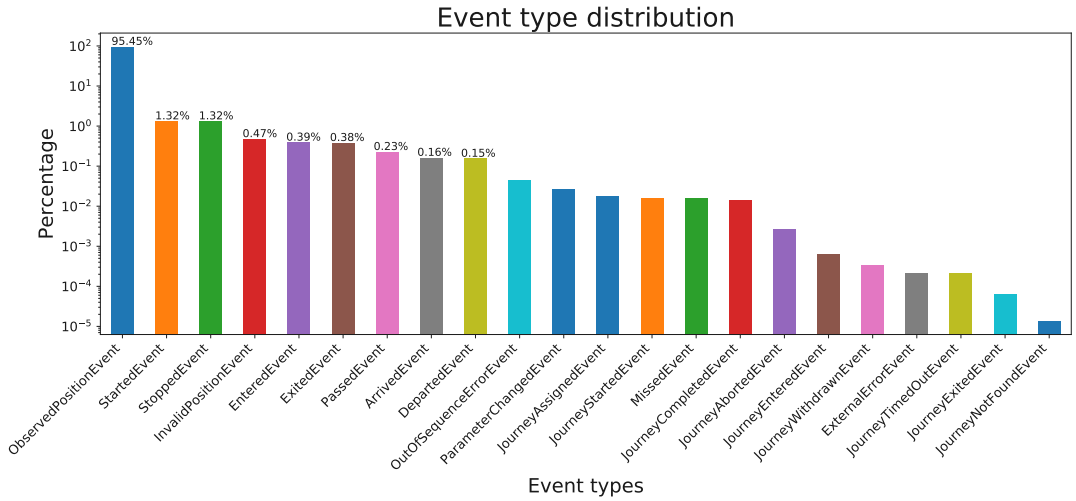


Figure 3.3: The distribution of event types for an arbitrary day in the dataset.

Of the 20 event types available in the dataset, this thesis project only used 12. The events to use were chosen by analysing the log for a single day in great detail. Event types which

occurred rarely and seemingly at random were discarded, as no pattern could be determined for them. For example, the `InvalidPositionEvent` type only contains the GPS position of the vehicle. The `ObservedPositionEvent` type also contains the *Speed* and *Direction* parameters. All of the `InvalidPositionEvent`s events had 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 of a bus stop. The `ExitedEvent` is similarly produced when the vehicle leaves the predefined distance of 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. For example, 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 of a bus line. The `JourneyCompletedEvent` event type is produced when the bus has reached the final bus stop of a bus line.

- `JourneyAssignedEvent`: This event type is in most cases accompanied by a `ParameterChangedEvent` to denote when a bus changes its journey.

3.3 Dataset Pre-Study

The dataset pre-study was conducted after the basic characteristics were identified. It analysed the dataset in greater depth, identifying interesting scenarios that occurred in the data. The analysis was conducted by manually parsing the data. This manual task used EDA

Figure 3.4: 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.

methods to simplify the process; the focus was put on visualising the data in order to detect patterns and outliers. All events during an arbitrary day were initially used for this task. Anomalies that occurred were isolated and categorised by applying the rationale behind the binary search algorithm. The log was divided in two equal halves in order to see which half inhibited the anomalies. The procedure was repeated as long as the anomalies were present. This virtually halved the search space at each iteration. If an anomaly was missing from both halves it was an indication that the anomaly occurred by combining the events around the split.

The search was more complex than one simple binary search algorithm run, as anomalies could occur in both splits. If anomalies were present in both splits, the split intervals were saved and then the search proceeded in one of the splits. When the search exhausted the arbitrarily chosen split, it started searching the other split using the saved interval. The method can thus be seen as a binary search algorithm which branches and starts another binary search processes if anomalies are detected in both splits.

Human Error: Early Stopping

Figure 3.4 and 3.5 illustrate two scenarios when the `JourneyCompletedEvent` type for a started journey would be missing. In Figure 3.4, the bus driver is supposed to visit the three markers in order to complete the journey for a particular line. In Figure 3.5, 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

Figure 3.5: 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 an erroneous ordering of contextual events (`JourneyCompletedEvent` never received).

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 3.6. 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 roughly 45 meters before the final bus stop. The bus stops there for 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 **??**.

3.4 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 objects with attributes. Figure 3.2 visualises which attributes

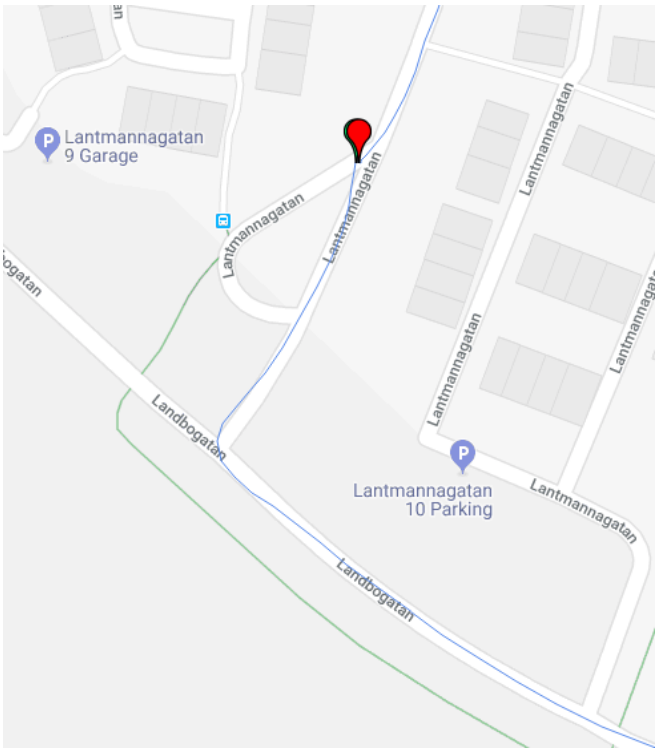


Figure 3.6: Real-world example of a bus driver stopping roughly 45 meters before reaching the final bus stop. The red marker is the GPS position where the bus stopped and the bus icon on the map is the pre-determined position of the bus stop. The bus stops there for a few minutes before it drives off to the first bus stop for a new journey. The bus stop detection algorithm systematically does not identify these cases.

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 3.7.

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 FSM.

Context-Providing Finite-State Machine

The context-providing FSM constructed in this thesis project is shown in Figure 3.8. 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 3.9 for a real-world example. However, this is not always the case when working with real-world data, as shown in Figures 3.10 and 3.11. 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.

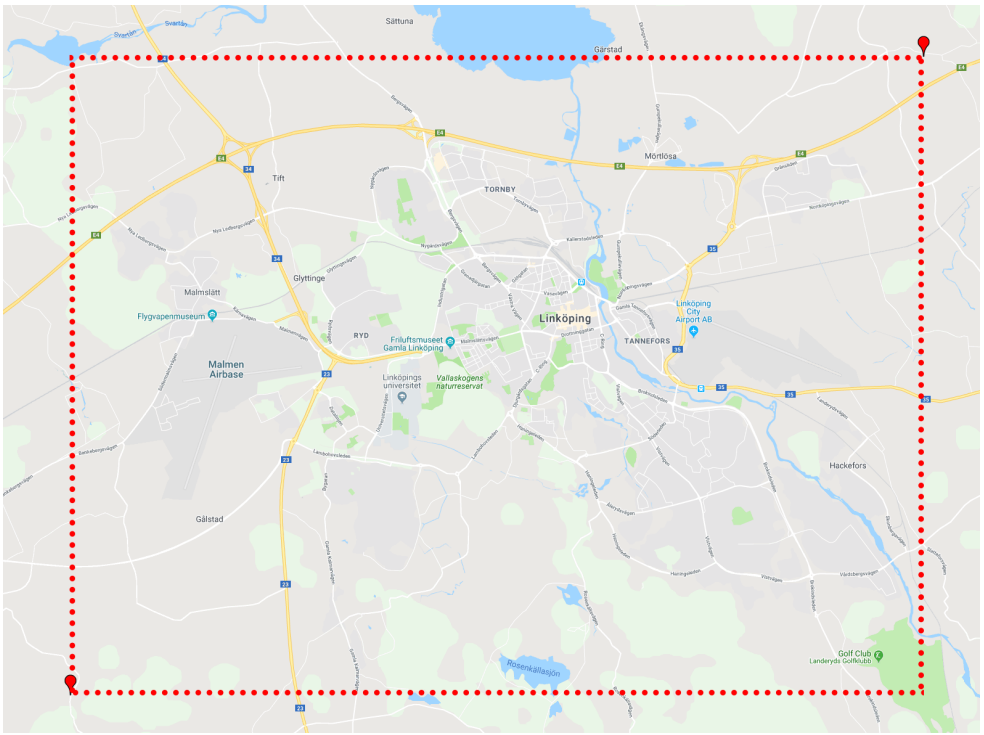


Figure 3.7: A geo-fence is constructed to filter out events occurring outside the virtual perimeter. The two red markers create a rectangular boundary, which is illustrated with the red-dotted line. The geographical area is the city of Linköping.

The problem is partly solved by 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. Unfortunately, this approach would not work when dealing with streamed real-time data, as the ordering cannot be changed without introducing a buffer. The context-providing FSM was slightly altered in order to support the processing of streamed real-time data. Another motivation behind the change was to remove the assumption that the sequence of *Event ID*s is always correct. The `ObservedPositionEvent`s are instead placed in a temporary buffer once an anomaly is detected in the event ordering. For example, if a new `JourneyAssignedEvent` is received before the supposed `JourneyCompletedEvent` and the system is in the "Started" state. Once the `JourneyCompletedEvent` type is received, after some delay, the data in the buffer is retroactively added to the "Started" state. The "Started" state then correctly contains all `ObservedPositionEvent`s received from the starting bus stop to the final bus stop. However, 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. These scenarios can easily be detected by utilising the context-providing FSM with the added buffer extension.

Bus Stops

Using the FSM 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 FSM solution described does not handle events about a bus arriving, departing or passing a bus stop on the journey. Handling this type of data

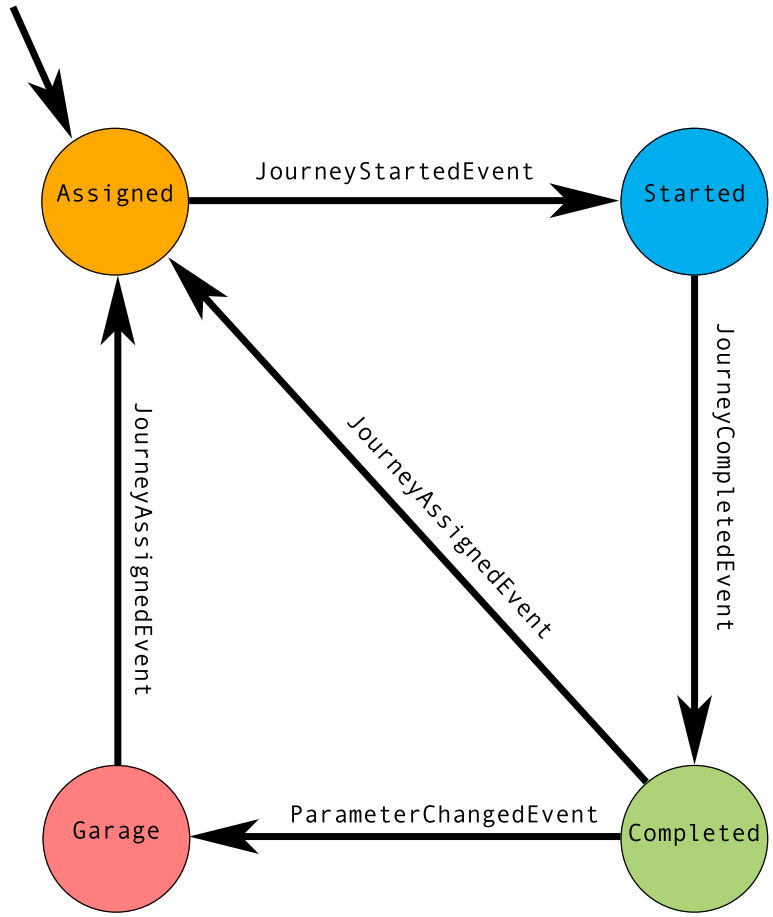


Figure 3.8: Finite-state machine providing context to `ObservedPositionEvent`s. The constructed FSM is simplified to illustrate the best-case scenario. The "Assigned" state is the starting state. `ObservedPositionEvent`s are assigned to the current FSM state.

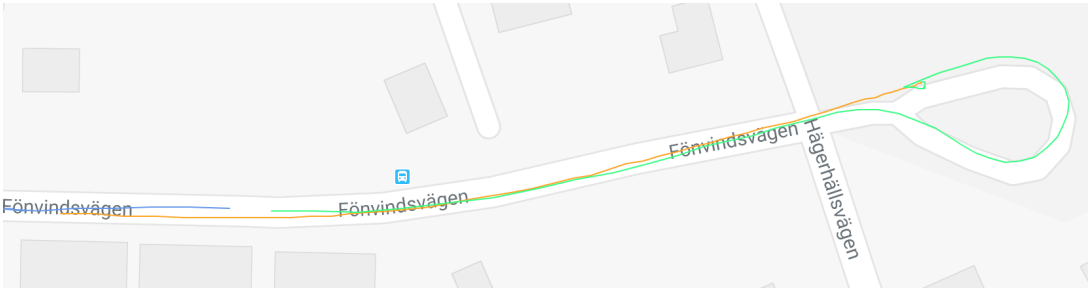


Figure 3.9: Real-world scenario illustrating when events occur in a logical ordering. The blue line to the left is `ObservedPositionEvent`s in the "Started" state. Upon reaching the final bus stop for the line the state changes to "Completed". The `ObservedPositionEvent`s for this state is drawn with 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 `ObservedPositionEvent`s 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 3.10: Example illustrating when the 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".

is a critical step in detecting imprecisions in the bus stop detection algorithm or early stopping due to human error. The "Started" state in the FSM could be extended to not only include `ObservedPositionEvent` types, but also `ArrivedEvent`, `DepartedEvent`, and `PassedEvent` types. A missed final bus stop could then, for example, 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. Extending the FSM to handle the bus stop data would cause the complexity of the FSM to increase.

An alternative approach was implemented instead, as the existing context-providing FSM could easily be verified to produce correct output. The approach extracted all events of the types `StartedEvent`, `StoppedEvent`, `PassedEvent`, `ArrivedEvent`, and `DepartedEvent` for each specific *Vehicle ID*. These events were then paired with the journeys for each respective *Vehicle ID*. Any events of the type `StartedEvent` or `StoppedEvent` occurring before the first `ObservedPositionEvent` of the journey were discarded. Events of the type `PassedEvent`, `ArrivedEvent` or `DepartedEvent` were included in the journey if they occurred within a specific period of time of the first `ObservedPositionEvent`. When all `ObservedPositionEvent`s of a journey were processed, the procedure continued with the next journey. The set of extracted events thus naturally reduced in size for each journey processed. Events occurring in-between journeys were discarded. When adding a bus stop the journey the name of the bus stop was also added to a separate array, in order to simplify the process of detecting faulty journeys.

Results

The result 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 FSM was in the "Started" state and all the bus stops the bus arrived at, departed from or passed by. It also contains all the stops and starts made by the bus during the journey. Erroneous event type

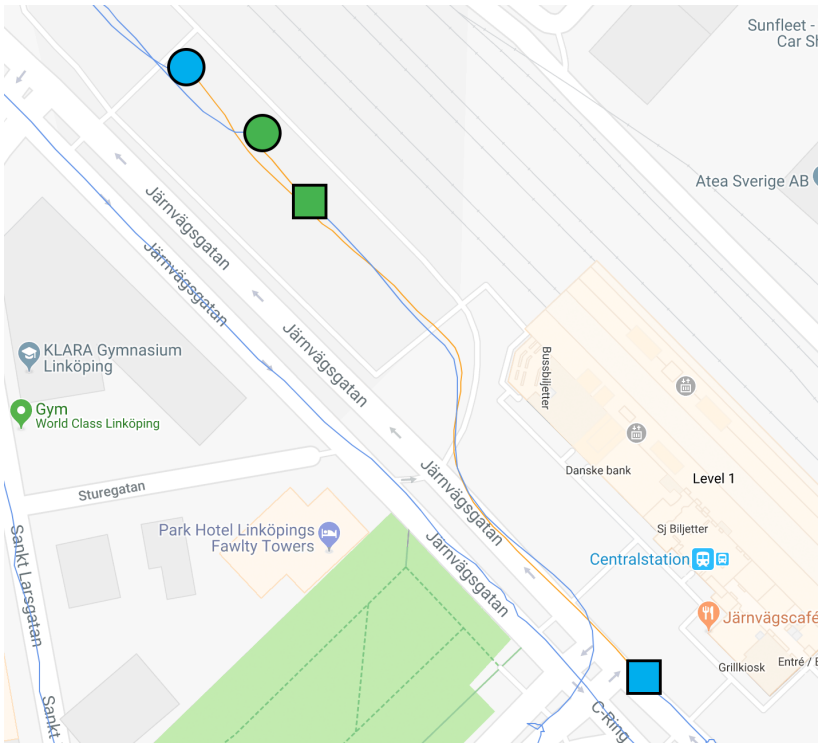


Figure 3.11: Another example illustrating two cases where a bus is assigned a new bus line before completing the started one. The two circles (green and blue) denote the final bus stops for the respective bus lines. One of the buses is assigned a new bus line at the blue rectangle in the bottom-right 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.

ordering was solved by sorting the events based on *Event ID* and using a buffer to retroactively add `ObservedPositionEvent` in the "Assigned" state to the "Started" state in the case of an 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 result 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.

3.5 Future Work

There are possible improvements to make when handling the data. This section mentions some areas were improvements could be made.

Discarded Event Types

The discarded event types should be investigated in-depth with regard to the whole dataset and not only for an arbitrary day. Some of the events could perhaps be used to easier detect or handle the faulty journeys. During the initial manual processing of the data, no reasonable pattern was discovered for any of the excluded event types. This may be an indication that the rarer event types are wrongly produced by the "Internal Analysis" component. A more probable explanation would be that the method of manually investigating the context different event types occur in is subpar and prone to human errors. An automated approach tailored to focus on rare event types would probably have yielded better results and pattern recognition.

4 Bus Stop Detection

The first high-level problem is the problem with inaccurate bus stop detections. This causes the system to occasionally miss bus stops, which results in faulty or incomplete journeys. Incomplete journeys require more pre-processing to be done in order to use the data. Another problem with the existing bus stop detection algorithm is that bus stop coordinates are pre-determined, meaning that they do not always represent the actual bus stop. This introduces an overhead to the system, where bus stop coordinates need to be updated in order to continue to provide accurate detection of bus stops.

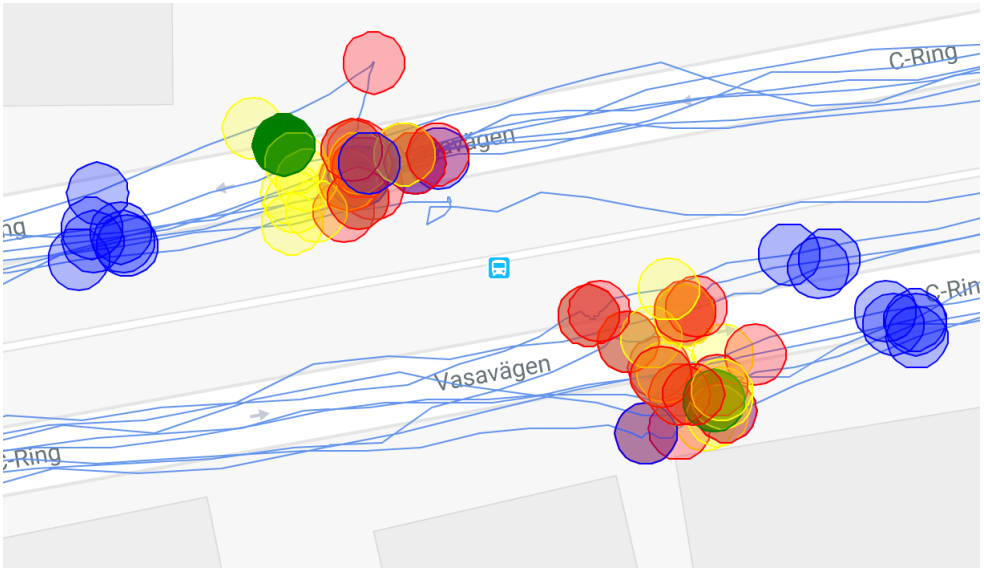


Figure 4.1: Output from the Bus Stop Detection Algorithm from the "Internal Analysis" component. The green circles are the pre-determined coordinates for the bus stop. The bus stop shown has two modes, depending on if the bus travels west (top road) or east (bottom road). The red and yellow clusters are the GPS positions of the bus when it arrives to the bus stop. The blue clusters are the GPS positions of the buses departing from the bus stop.

Figure 4.1 shows the results from the existing Bus Stop Detection Algorithm of the "Internal Analysis" component. Each bus produces seven GPS coordinates for each bus stop: one blue, two red, two yellow, and one green. The green GPS coordinate is always the same between all buses for each bus stop; it is the pre-determined coordinate for the bus stop. The blue GPS coordinate is the position of the bus when it departs from the bus stop (the last positional update within a pre-determined radius of the bus stop). The red and yellow coordinates are the positions of the buses when the system determines that they have arrived (stopped) at the bus stop. Due to lacking documentation, the difference between the red and yellow coordinates are unclear as they are occasionally identical, but not always (as shown in the figure). The existing bus stop detection algorithm thus checks within a pre-determined radius around the green GPS coordinate for updates from buses. When the first positional update from the bus is received, the system tracks the bus and waits for it to stop. Upon stopping, the bus is determined to have arrived at the bus stop. When the bus starts again, the system continues to track it and reports when the bus has left the radius of the bus stop. The blue GPS coordinate shows where this happens.

5 GPS Variation Estimation

The main component of the dataset is the GPS positional update events from the vehicles. They contribute to roughly 96% of the events in the data, as shown by Figure 3.3 in Section 3.2. An interesting high-level problem is thus the estimation of the variance in the GPS positions. The hypotheses are that the GPS positions vary depending on:

1. *Environment (Spatial Locality):* The surrounding environment impacts the visibility of GPS satellites. For example, in an environment with a lot of reflections and obstacles, the precision of the GPS position will decrease. On the other hand, in open fields the precision will increase, as there are fewer obstacles blocking the GPS satellite signals. The environment can vary between different road segments, but also inside one road segment, e.g., a road can be partly covered by houses or trees. The variance could either be seen as continuous, where each point on the journey has its own variance, or as road segments, where each road segment has its own variance. In the continuous case, neighbouring pairs of points would be connected and thus exhibit more similar variance, depending on the distance between the points (granularity) in a journey. In the case of road segments, different segments would exhibit a, potentially, larger difference in variance.

   source?

   Road segments could either be created geographically, e.g., a segment is the road between two bus stops or between crossings, or contextually from the output of the continuous case. A geographical road segmentation would most likely get a higher average variance, as the environment could change within the segment. The contextual segmentation would look at sequences of points in the journey, and group the sequential points which exhibit similar variance. The result would potentially be segments where the inherent variance of each segment is lower than between different segments.

   Figure 5.1 illustrates how the GPS variance can differ for different road segments. In the shown scenario, the difference between geographical segmentation between crossings and contextual segmentation would be minor, as the variance seems to be similar for a given road segment. However, the difference would be major if the geographical segmentation was done between bus stops. The number of road segments would be higher in the crossing-geographical approach than in the contextual approach, as, for example, "Järnvägsgatan" has many crossings but exhibit similar variance.

2. *GPS Sensor Calibration:* Each bus has its own GPS sensor, where different buses could have different models with their own calibrations. In the general case the different GPS sensors show small variance due to different calibrations. However, occasionally they can vary, e.g., by some constant factor, as shown in Figure 5.2. The single orange-coloured bus journey seems to have a constant offset compared to the rest of the journeys for that particular bus line. The shown scenario is quite extreme, as the offset is large enough to cause problems with the Bus Stop Detection Algorithm in the "Internal Analysis" component. No bus stops are detected in this particular scenario. These extreme situations can thus be detected by combining the positional events with the bus stop events. However, differences in calibrations which yield smaller offsets cannot be detected with this combining approach.

3. *Time:* The timestamp of the position events affects the precision of the GPS sensor. At a single given point of the journey the precision could vary depending on the positions of satellites and the amount of satellites visible. The hypothesis is that there is a constant bias to the GPS positions depending on the timestamp. The positions of the satellites thus cause a small offset to occur in the GPS position. This parameter probably has some periodicity, which could be fitted using a Gaussian Process with a periodical kernel.

Figure 5.1: Visualisation of multiple journeys by different buses in the same area. Each blue line is a journey in the "Started" state. The variance differs between different road segments, as shown around "World Class Linköping" compared to "Järnvägsgatan".

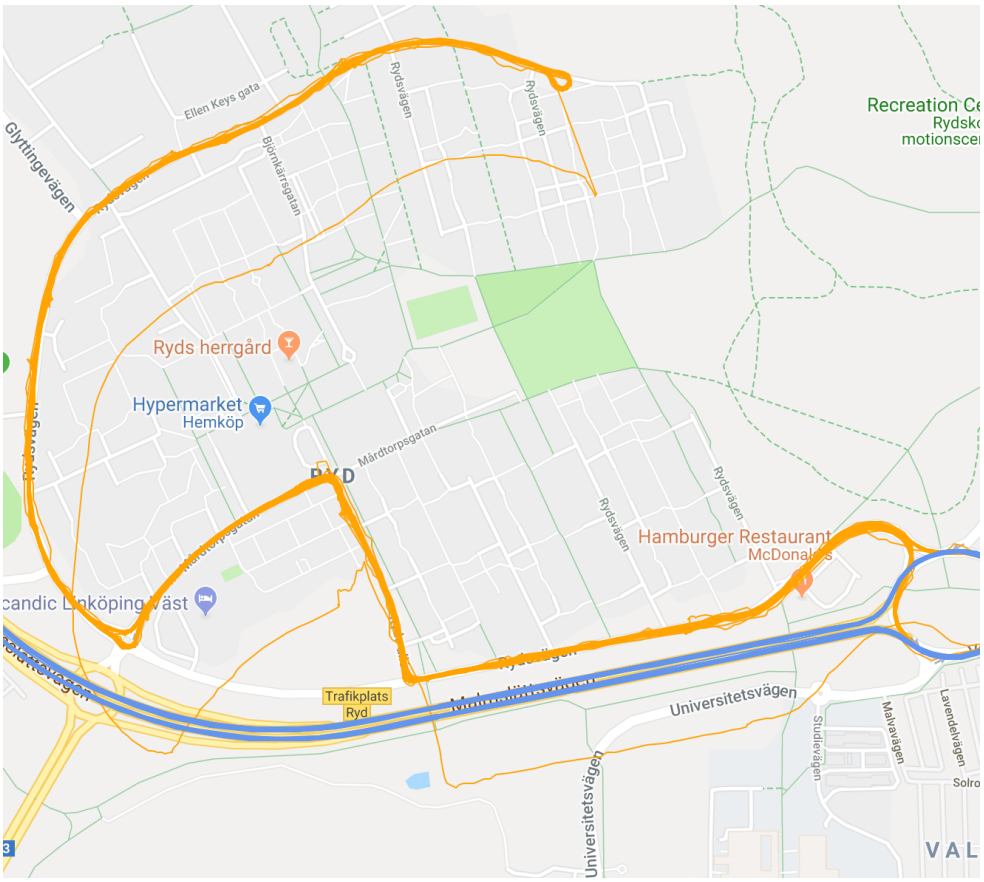


Figure 5.2: Illustrates the case of a poorly calibrated GPS sensor on an individual bus. The orange lines creating the thicker line are all the other journeys for the same bus line, but done by other buses. The thin orange line is one bus with an extremely poorly calibrated GPS sensor. It seems to have a constant offset compared to the rest. This real-world scenario shows that the calibration of GPS sensors is important in order to produce useful data.

Road Segment Data

Google Maps provides a Roads API[1] to map GPS coordinates to the closest road segment. The road segments could, for example, be used to create a model calculating the orthogonal variance of GPS coordinates with respect to the road segments. However, the variance of GPS positions does not only occur orthogonal to a given road segment.

Bild?

Manual inspection of journeys done in the pre-processing step highlights scenarios with discrepancies between roads in Google Maps and actual roads the buses drove. The Google Maps roads are in these scenarios usually incorrectly drawn or has missing roads. Figure 5.3 shows such a real-world example. The discrepancies would cause problems if used naïvely together with the Google Maps Roads API. These journeys would have to be detected and either handled or ignored. The detection could be done by looking at the distances to the closest Google Maps Roads segments. The distances would be roughly similar for all the journeys, as in Figure 5.3. Since the distances would be roughly similar they could be averaged and interpreted as a bias to the roads. Applying the bias-interpretation methodology would result in a more generalised model using the Google Maps Roads API. Ignoring the journeys with discrepancies could also be seen as feasible, given that the occurrence is rare.

Vart vill jag komma med det här?

[1]https://developers.google.com/maps/documentation/roads/intro

Before using the simpler approach of ignoring journeys with discrepancies, a deeper analysis needs to be conducted.

Figure 5.3: Real-world example of multiple buses driving on a road not correctly modelled by Google Maps. For example, at "Sjukhusvägen" the buses are driving through a building according to Google Maps. In this particular scenario the area has received major road restructuring and renovation. The Google Maps representation is thus likely outdated.

5.1 Methodology

The method used in this thesis project to solve the higher-level GPS Variation Estimation problem is based on combined Gaussian Processes (GPs). The method is described by 5 steps:

1. *Stop Compression:* Each stop present in the journey causes an imbalance of points being clustered around each stop. The longer the stop is, the more points are present. This results in problems with the GP, as certain coordinates would be denser than others. Positions sent while a bus has stopped are averaged into a single coordinate. The time of the stop is also adjusted. The result is a journey consisting of a sequence of coordinates where stops are invisible time-wise.

2. *Segment Journeys:* Each bus journey in the dataset is fitted with its own GPs. Journeys are analysed in order to detect self-overlapping, which would cause problems for the fitted

GPs. Journeys with self-overlap are split at the overlapping points into two segments. The process recursively analyses each resulting segment and proceeds with splitting the segments as long as self-overlapping is present. If a journey has multiple segments, each segment is fitted with its own GPs.

The first GP fits x, y (longitude, latitude) coordinates to a new time parameter τ:

$$f_1 : x, y \longmapsto \tau \tag{5.1}$$

τ is the relative time of the journey, denoting the progress of the bus in completing the journey. The second and third GP fits τ to a new x and y, respectively:

$$f_2 : \tau \longmapsto x \tag{5.2}$$

$$f_3 : \tau \longmapsto y \tag{5.3}$$

3. *Point-Wise Combining of GP:* The GPs f_2 (Eq. 5.2) and f_3 (Eq. 5.3) each describe the variance (σ_2^2 and σ_3^2, respectively) at each point in the fitted segment or journey. The GPs describing the variances are point-wise combined using Equation 5.4 and 5.5, creating the GPs f_4 (Eq. 5.6) and f_5 (Eq. 5.7), respectively.

$$\mu(x^*) = \frac{\sum_{j=1}^{J} N_j \mu_j(x^*)}{\sum_{j=1}^{J} N_j}, \tag{5.4}$$

$$\sigma^2(x^*) = \frac{\sum_{j=1}^{J} N_j(\sigma_j^2(x^*) + \mu_j^2(x^*))}{\sum_{j=1}^{J} N_j} - \mu(x^*)^2, \tag{5.5}$$

$$f_4 : \tau \longmapsto \sigma_4^2 \tag{5.6}$$

$$f_5 : \tau \longmapsto \sigma_5^2 \tag{5.7}$$

where σ_4^2 is the variation in longitude and σ_5^2 the variation in latitude.

4. *Visualisation:* The resulting f_4 and f_5 GPs are then visualised. hur?

5. *Comparison:* The visualised GPs are compared and different routes are compared at the same spatial area. The comparison analyses if there is a periodicity to the variation or if the variation is purely based on spatial locality.

6 Arrival Time Prediction

Trajectory Forecasting is the process of predetermining when a bus will arrive at a particular bus stop. The existing Trajectory Forecasting is based purely on long-time historical data. It provides a single timestamp when it predicts the bus to arrive at a certain bus stop.

Arrival Time Prediction -> ...

6.1 Methodology

This thesis project proposes a Trajectory Forecasting model based on multiple GPs. It provides time to arrival distributions to all the remaining bus stops, meaning that it can answer queries such as:

- *"What is the most likely time of arrival at bus stop p?"*
- *"What is the probability of the bus arriving at the earliest/latest possible time?"*
- *Which stop impacts the arrival time of the bus the most?*

The approach has three steps:

1. *Stop Model and Compression:* Stops will be modelled as both bus stops and red lights, i.e., common stops during a bus journey. GPS positions of stopped buses need to be compressed in order to support more robust GPs . The result of this step is a timeline for each bus journey with marked common stops and compressed GPS positions during stops.
2. *Segment Stops (With Model Overlap):* The next step is to process the timeline and segment it between stops. The segments need to be checked for self-overlaps, in order to avoid the problems described in Section . Each segment will have a small model overlap, which means that neighbouring segments will share a few data points. This improves the robustness of the model.
3. *GP Trajectory Models:* The final step is to fit multiple GPs to each trajectory segment. The first GP fits x, y (longitude, latitude) coordinates to a time τ:

$$f_1 : x, y \longmapsto \tau \tag{6.1}$$

source: Gaussian Process Based Motion Pattern Recognition with Sequential Local Models

refsec:problems-trajectory-forecasting

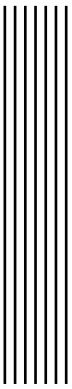
τ is the relative time of the journey, denoting the progress of the bus in completing the journey. The second and third GP fits τ to a new x and y, respectively:

$$f_2 : \tau \longmapsto x \tag{6.2}$$

$$f_3 : \tau \longmapsto y \tag{6.3}$$

7 Discussion

8 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] D. E. Knuth. 'Optimum binary search trees'. In: *Acta Informatica* 1.1 (Mar. 1971), pp. 14–25. ISSN: 1432-0525. DOI: 10.1007/BF00264289. URL: https://doi.org/10.1007/BF00264289.

[6] Dexter C. Kozen. *Automata and Computability*. 1st. 233 Spring Street, New York, NY 10013, USA: Springer Science+Business Media, Inc., 1997. ISBN: 0-387-94907-0.

[7] 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.

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

[9] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. ISBN: 026218253X.

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

[11] 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.