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 Trajektorie Prognostisering

Linus Kortesalmi

Supervisor : Mattias Tiger
Examiner : Fredrik Heintz

External supervisor : Simon Johansson

LINKÖPING UNIVERSITY

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

Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Linus Kortesalmi

Abstract

Abstract.tex

Acknowledgments

Acknowledgments.tex

Contents

List of Figures

Todo list

1 Introduction

1.1 Motivation

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 is 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. They cover issues such as pre-processing, data noise and outliers. The questions can be seen as a pre-study to Machine Learning. 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?*
   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. It analyses the features of the dataset, e.g. different event types, event parameters, and event structure.

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.
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.
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.

Higher-Level Problem Questions:

1. *What are some of the higher-level problems which can be formulated using the available data?*
   These problems can utilise both the data available in the provided dataset and any complementary external data. The answer to this question will not be a complete list of all possible problems, but rather a short list of a few interesting examples. Each formulated problem will have its core problem explained.
2. *What are some of the solutions to the formulated problems?*
   The list of solutions for each formulated problem will not be a complete list of all possible solutions. The solutions will be tackling the core of each formulated problem. Each solution will explicitly state if there is a baseline available for comparison or if one could be easily created.
3. *Who could benefit from the solutions?*
   This question analyses the solutions in a broader context, e.g. from a societal or ethical point of view. For example, a solution could yield great results for the industry at the cost of consumer privacy.

Fråga: Ska jag lista de problem jag har redan här? Annars blir det svårt att fortsätta med Problem Specific Question.

Problem Specific Questions

1. *How can GPS variance estimation be solved with combined Gaussian Processes Regression using a local trajectory model?*
2. *How can the GPS variance estimation model be evaluated?*
   The method employed makes certain assumptions regarding the inherent noise of the data. The kernels applied to the model need to evaluated. The local trajectory model shall be compared with a global trajectory model.
3. *How can Trajectory Forecasting be realised with Gaussian Processes Regression using a local trajectory model?*
4. *How can the Trajectory Forecasting model be evaluated in the context of information gain?*
   The Trajectory Forecasting model could, for example, be evaluated by comparing the new forecasts with the existing forecasts from the baseline created by the internal system of Östgötatrafiken AB. This evaluation leads to information regarding which model to use to get more precise forecasts. The model could also be evaluated by looking at what kind of information and insights are available from the output of the model. A model which returns a probability distribution

This question is very explicit and stems from the results from asking the questions in the two previous groups. Is this okay, or do I need to add more information after each question in the previous groups? In one way this question kind of pops up from nowhere.

of arrival times, which are updated in real-time, should be evaluated in a broader context than precision of a single forecast.

Det här kanske inte hör hemma här. Det är väldigt klumpigt skrivet iaf.

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.

2 Theory

3 Data

This chapter describes the spatiotemporal 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

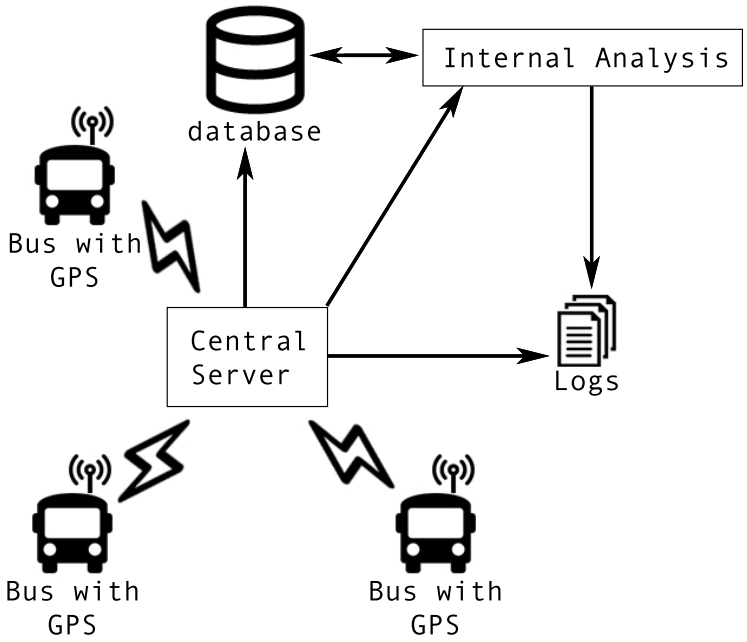

Figure 3.1: Simplified graph illustrating the data gathering process. Each bus is equipped with a GPS sensor and transmits its position to a central server/database. The dataset used in this thesis project is the log, which is a collection of documents. Each document contains the GPS data sent from all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

Analysis" component in the system. The "Internal Analysis" component is simplified and is beyond the scope of this thesis project.

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.

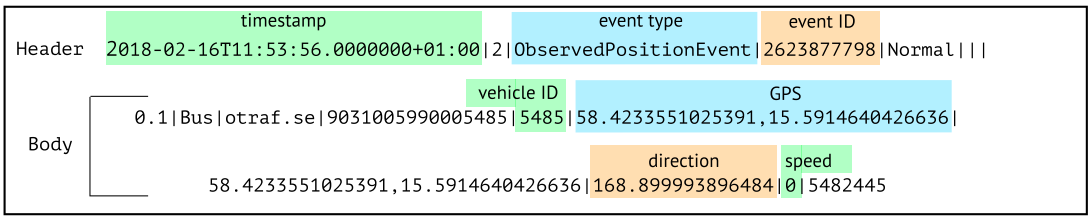

Figure 3.2: Example of a raw `ObservedPositionEvent` entry. The header and the body is separated by `|||`. Each parameter in the header and body is separated by a single `|`. Key parameters for the `ObservedPositionEvent` event type is highlighted.

- *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 a random 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 dataset? Will take a lot of time to create!

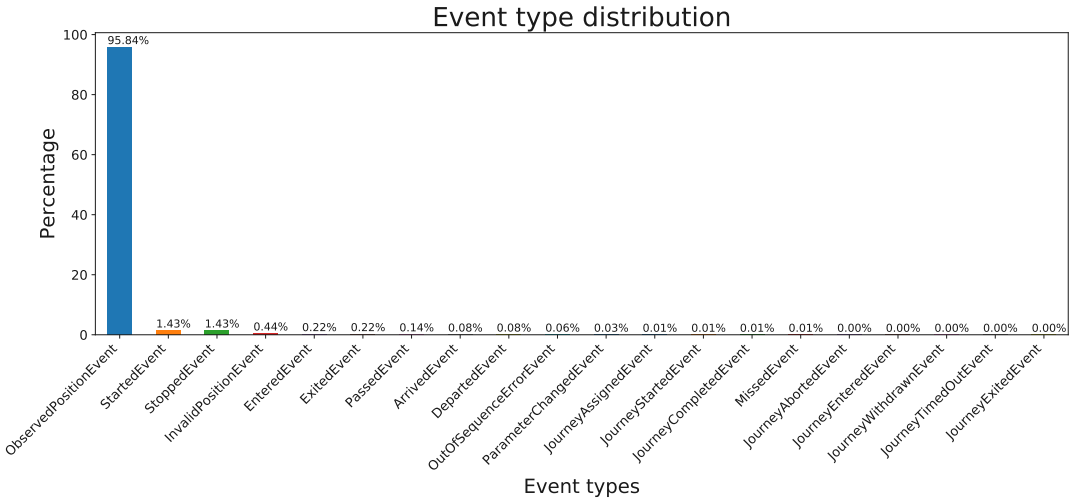

Figure 3.3: The distribution of event types for a random day in the dataset.

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.

Wasnt it something like this Östgötatrafiken did? @Mattias

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.

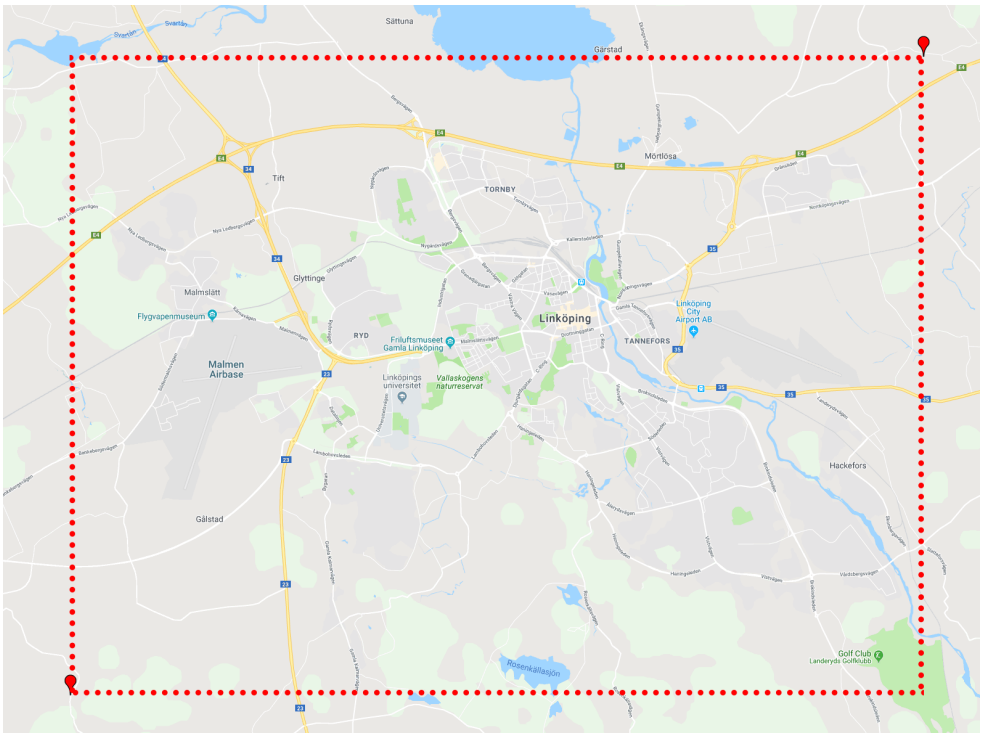

Figure 3.4: A geo-fence is constructed to filter out events occurring outside the virtual perimeter. The two red markers create a rectangular boundary, which is illustrated with the red-dotted line. The geographical area is the city of Linköping.

After the parsing and filtering step the idea was to provide context to *ObservedPositionEvent*s. 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 *ObservedPositionEvent*s 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 *ObservedPositionEvent*s received from the starting bus stop to the final bus stop.

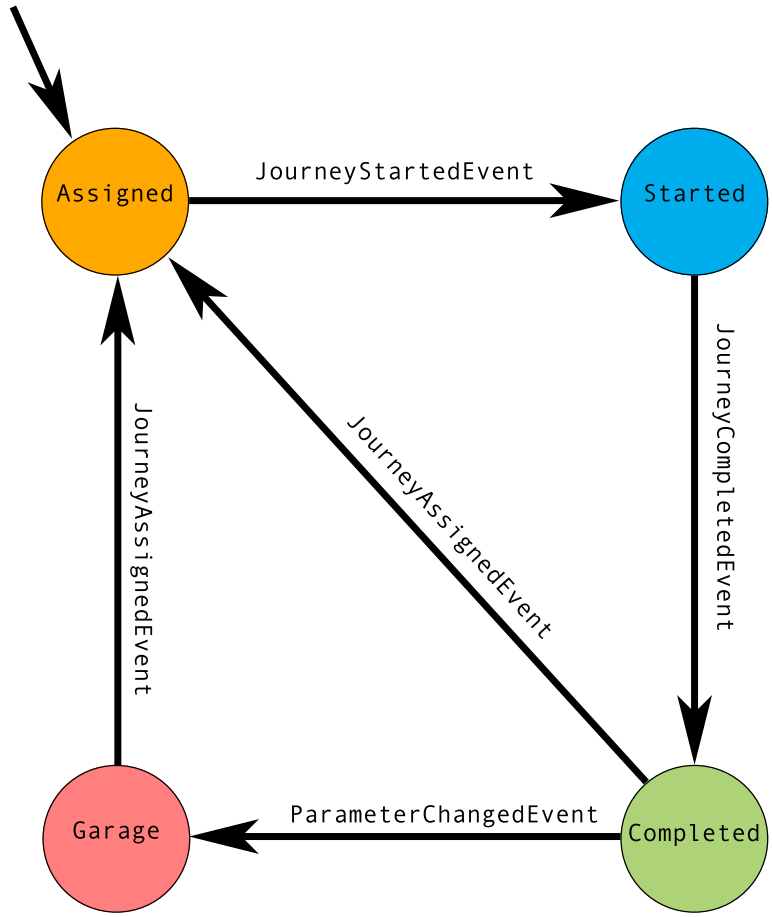

Figure 3.5: Finite-state machine providing context to *ObservedPositionEvent*s. The constructed finite-state machine is simplified to illustrate the best-case scenario. The "Assigned" state is the starting state. *ObservedPositionEvent*s are assigned to the current state the state machine is in.

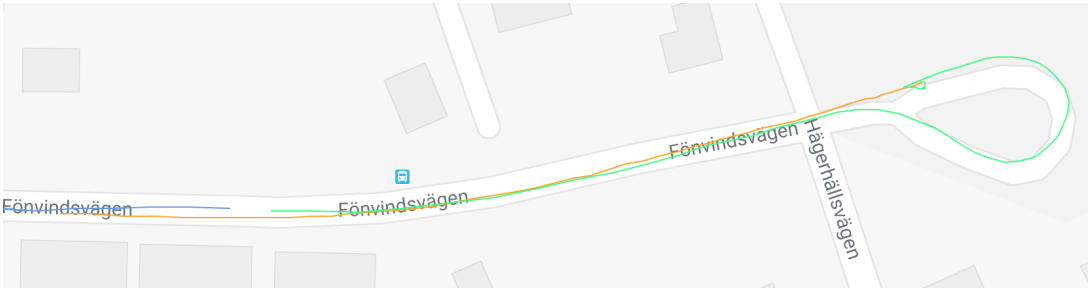

Figure 3.6: Real-world scenario illustrating when events occur in a correct, logical ordering. The blue line to the left is *ObservedPositionEvent*s in the "Started" state. Upon reaching the final bus stop for the line the state changes to "Completed". The *ObservedPositionEvent*s for this state is drawn 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 3.7: 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 (at the end of "Björkliden"). The last part of the journey is thus assigned to a new state "Assigned", instead of the actual, logical state "Started".

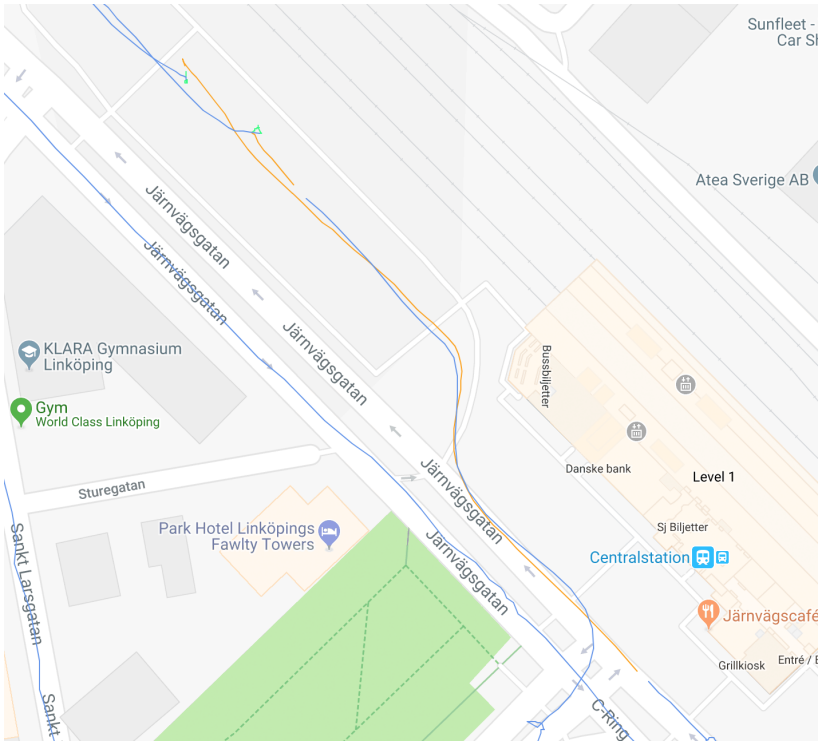

Figure 3.8: Another example illustrating two cases where a bus is assigned a new bus line before completing the started one. The bus at the crossing of "C-Ring" and "Järnvägsgatan", in the bottom-right corner, is assigned a new bus line long before reaching the final bus stop. The two small green clusters highlight when the *JourneyCompletedEvent* types were received.

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.

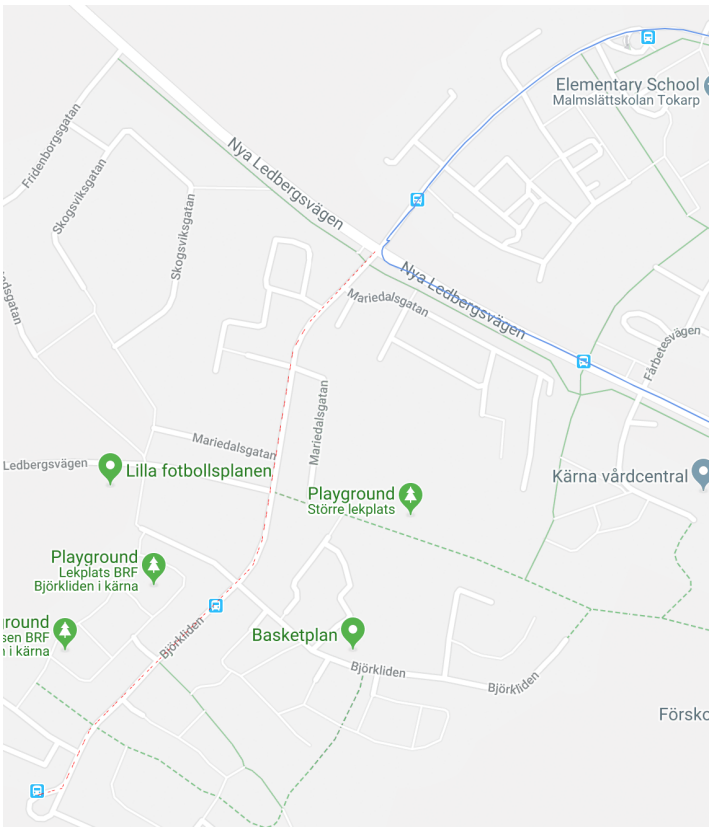

Figure 3.10: Another example of early stopping in a journey. The red dashed line is the planned route of the bus line. The blue line at "Nya Ledbergsvägen" is the actual route the bus drove. This results in the journey never being deemed completed, creating a erroneous ordering of contextual events.

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 th8e 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.

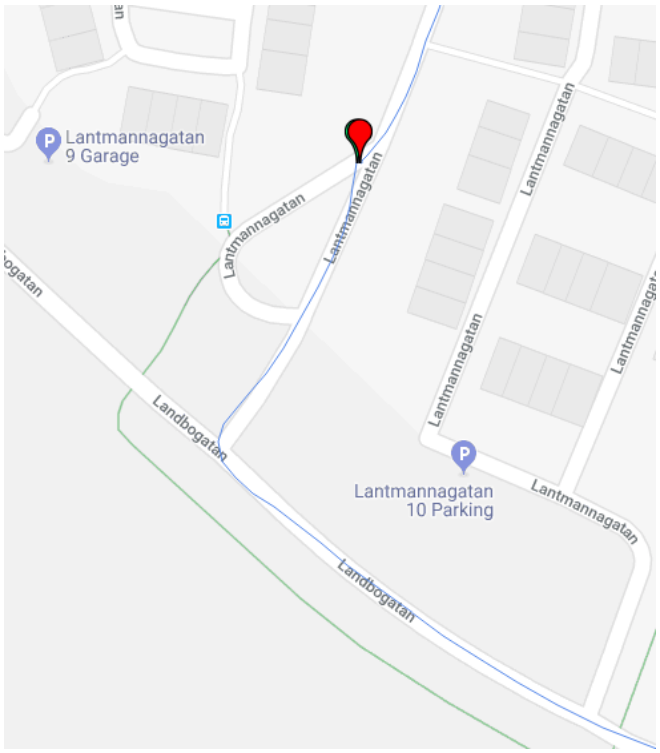

Figure 3.11: Real-world example of a bus driver stopping roughly 45 meters before reaching the final bus stop. The red marker is the GPS position where the bus stopped and the bus icon on the map is the pre-determined position of the bus stop. The bus stops there for a few minutes before it drives off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

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 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.

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 High-Level Problems

This chapter describes three high-level problems formulated with data from the provided dataset.

4.1 Improved Bus Stop Detection Algorithm

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 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 departures 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.

4.2 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.

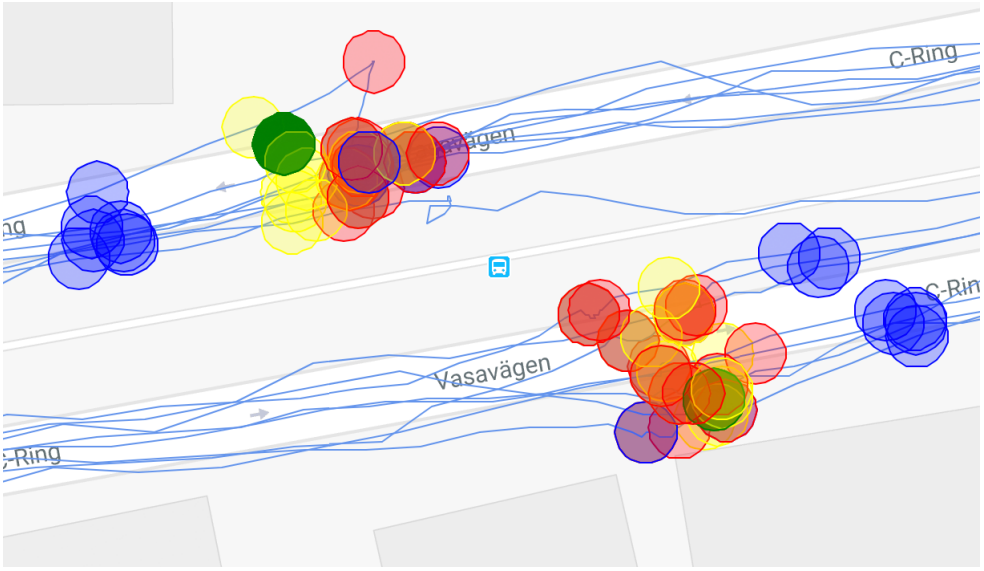

Figure 4.1: Output from the Bus Stop Detection Algorithm from the "Internal Analysis" component. The green circle are the pre-determined coordinates for the bus stop. The bus stop shown has two modes, depending on if the bus travels west (top road) or east (bottom road). The red and yellow clusters are 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.

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:* 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 4.2 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" have 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 4.3. 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 4.2: 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".

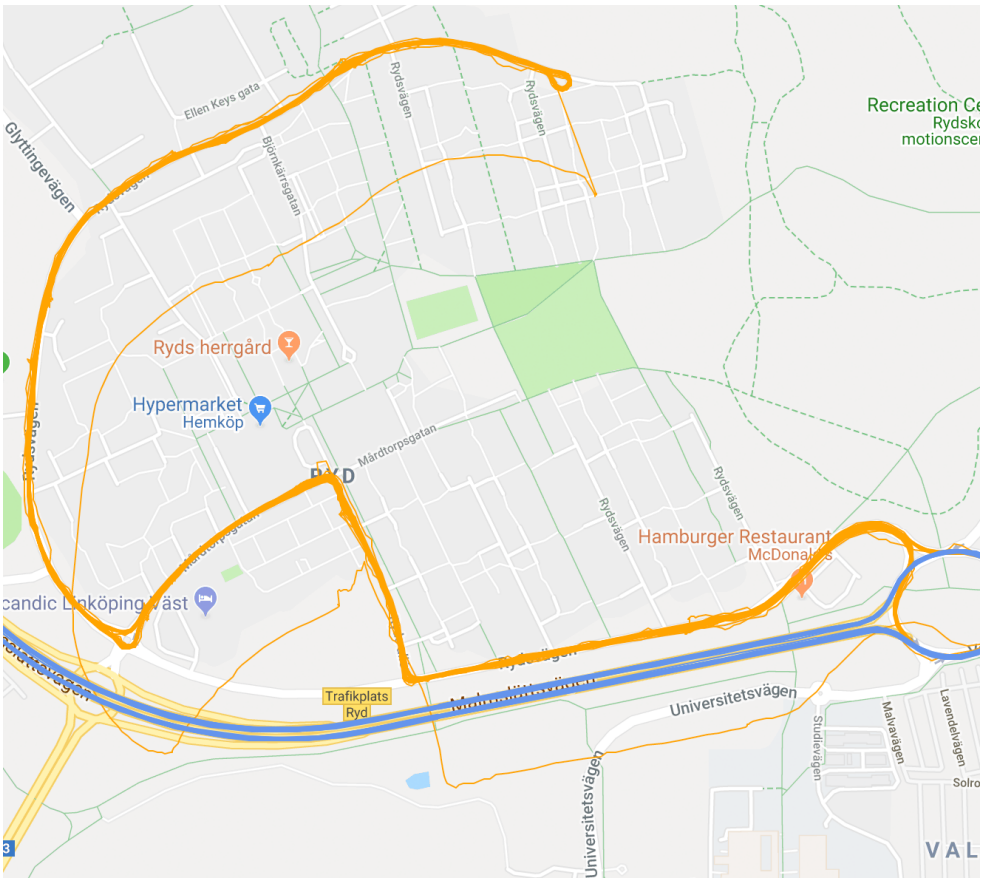

Figure 4.3: Illustrates the case of a poorly calibrated GPS sensor on an individual bus. The orange lines creating the thicker line are all the other journeys for the same bus line, but done by other buses. The thin orange line is one bus with an extremely poorly calibrated GPS sensor. It seems to have 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 or buildings. Figure 4.4 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 roughly be similar for all the journeys, as in Figure 4.4. 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

Vart vill jag komma med det här?

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

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

Figure 4.4: 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.

4.3 Trajectory Forecasting

Trajectory Forecasting is the process of determining when a bus arrives at a given 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.

5 Method

This chapter

5.1 GPS Variation Estimation

5.2 Trajectory Forecasting

This thesis project proposes an alternative Trajectory Forecasting model, based on multiple Gaussian Processes (GPs). It provides time to arrival distributions to all the remaining bus stops, meaning that it can answers queries such as: *"What is the most likely time of arrival at station p?"* and *"What is the probability of the bus arriving at the earliest/latest possible time?"*.

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 needs 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 4.3. 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 (latitude, longitude) coordinates to a time 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:

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

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

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

6 Discussion

7 Conclusion