Boarding and Alighting Passengers in SimBus Pro

Determining realistic stop times by modelling and simulating the boarding and alighting of passengers in more detail

Goudappel
MOBILITEIT BEWEEGT ONS

UNIVERSITY OF TWENTE.

ET – Civil Engineering

Committee:
E.C. van Berkum
R. Kromanis
G.J. Wolters
F. Verhoof

Final Version
6-7-2022

Frazer, R.J.H. (Rick)

Preface

This report marks the end of my Bachelor Civil Engineering at the University of Twente. The thesis *'Boarding and Alighting Passengers in SimBus Pro: Determining realistic stop time by modelling and simulating the boarding and alighting of passengers in more detail'* was carried out at the engineering and consultancy firm Goudappel. In this research, the reliability of the data used as input is validated, a theoretical foundation is laid for boarding and alighting, and the resulting theory is implemented. I hope the outcomes are valuable and the resulting model can be further developed and used in projects.

The past 10 weeks have flown by. Next to completing a great project, I had a lot of fun getting to know Goudappel and learning more about myself, about working at an engineering firm and about conducting research and traffic simulation too. Even though COVID is not completely gone, I was very glad to be able to work at least a few days in the week at the office. This meant I got to talk to and know a lot more people than my supervisors and see everyone in person. This also gave a great morale boost while working on my project. I feel I have become stronger as an individual as well, knowing better how I work and perform in the environment of an office.

Even though the thesis is an individual assignment, this project could not be completed entirely on my own. I would like to thank my external main supervisor at Goudappel, Geert-Jan Wolters, for answering every question I ever asked him, guiding me at Goudappel, and critically thinking with me about all kinds of solutions. I would also like to thank my second supervisor at Goudappel, Freek Verhoof, for helping me with all my questions about GOVI data and timetables and retrieving all necessary data. I want to give my thanks to Erik Oerlemans from Goudappel and Marcel van der Holst from Arriva as well for providing me with answers during the interviews on GOVI data. Finally, I would like to thank my internal supervisor Eric van Berkum for his all critical thinking, insights and guidance during this project.

I had a lot of fun conducting this project and hope you will enjoy reading it too.

Rick Frazer,

Enschede, July 6, 2022

Summary

The goal of this project was to investigate the reliability of GOVI data, come up with a theoretical basis for the processes of boarding and alighting, and implement this theory in SimBus Pro. From this, conclusions can be drawn about the validation and implementation of data and code.

For the engineering and consultancy firm Goudappel it was desired to know how the stop times can be properly set in Vissim, so that boarding and alighting passengers are taken into account. The resulting figures should be reliable and realistic as well. Afterwards, this functionality has to be implemented and coded in SimBus Pro, creating a model that reflects reality more closely. This improved model can be validated and used in projects.

Firstly, the validation of the GOVI data was executed. This data is the major input for the timetables that are used in the model, and as such has a major impact on the model and possibly the implementation of passengers. The validation has two approaches: a purely time-based approach and a geographic approach. Afterwards, interviews were held with two experts.

From the validation, it was shown that the data is not as reliable as perhaps thought. Between observed and recorded times quite a gap can be found due to the precision of the used instruments in measuring the times. Geographically, correlations can be found between the geographic location of a bus at the moment it is recorded as departed.

The executed observation study had two major focal points: Gathering the stop times to validate GOVI data, and counting the amount of passengers that board and alight, including the time it took for them to do so. From the observation study, a few different relations have been fitted for the number of passengers. for alighting, a function could be fitted as relation. For boarding and the combination of boarding and alighting, a function could not be fitted. Instead, two exponential distributions have been fitted, and the functions are used as theoretical maximum for the number of passengers. These form the theoretical foundation for the implementation.

During the implementation phase, testing in Vissim was done, and solutions brought forward for boarding and alighting and for pedestrian routing. During the testing, a list with requirements for the model was set up. For alighting only, a new function is set up to accommodate the calculations. In the correct situation, it will be called when a bus addresses a decision point. The dwell time will be determined and then the function called. The calculation for boarding is done in different stages of the simulation. The volume of pedestrians and generation times are determined during the preparation of the model. Here, the correct times pedestrians should be generated are determined dynamically for the model. For the combination of boarding and alighting, the two separate processes come together.

Finally, the model can be validated. For the validation, three indicators were chosen to validate the model with. These are visual validation, hindrance of buses through pedestrians, and groups of alighting passengers. To able to validate the model, the station of Deventer has been built in Vissim. Both a complete Vissim-network and a timetable have been set up. Visually, the model works as intended. The functionality built for keeping track of pedestrian hindrance works correctly as well, with 40 seconds of total hindrance from 10 simulation runs. The function used for alighting was validated with an additional validation dataset. For this validation, the R^2 between the function and the validation dataset was calculated. For the calibration dataset, the R^2 was 0.88, and for the validation dataset this is 0.94. This is a higher R^2, and thus the conclusion can be drawn that the calibrated function is a better fit than thought.

Table of Contents

Table of Figures

1. Introduction

In the last couple of decades, scientific and engineering models have gotten greater importance and more detailed, but also expanded, with the rise of more powerful computers. They can simulate and forecast reality better and better using improved technologies and increased data recording.

A major branch of engineering that benefits a lot from computer models is traffic engineering. In traffic models, the effects of behavior and road design on traffic can be analyzed in-depth before any implementations are made in the real world. As major benefit, road design can be optimized in a way that is not possible without models. In traffic engineering, every road and intersection is a completely different project and thus benefit a lot from modelling.

An area that can benefit from more attention is that of buses. Most traffic models are aimed at cars and trucks and how for example road design affects the travel times but buses are rarely taken into account or studied in detail.

For the purpose of studying buses in bus stations and the effects of timetables and other factors, Goudappel is programming this ability into SimBus Pro. This model is programmed into the traffic simulation program Vissim. Within this program however, there is not a lot of knowledge on the boarding and alighting of passengers. These processes however, have a lot of influence on the times that buses need to stop at stations, and are thus provide a valuable area of research.

This report will begin by explaining the context of the research, followed by the theoretical framework. Afterwards, it will delve into the validation of the data input, before arriving at the main design cycle of the project. The design cycle contains the research questions, prototyping, and validation. Finally, the discussion, conclusion and recommendations will be discussed.

2. Research Context

2.1. The Program

SimBus is a program to simulate bus stations or terminals. In this program, however, it is very difficult or even impossible to add new functionalities. On top of this, the software used was very old and not supported anymore. The visualization possibilities are very limited as well. When viewing the output of the model, which is shown below in Figure 1, it is hard to tell what the output is.

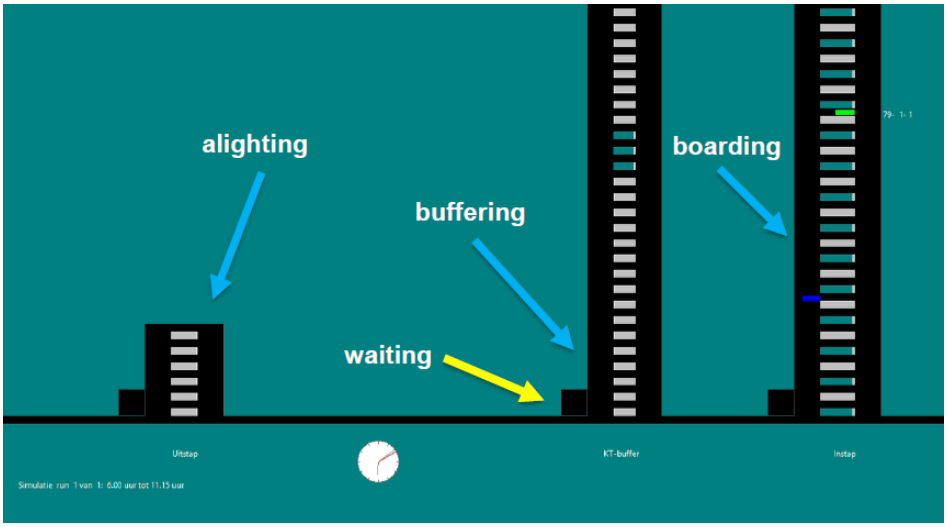

Figure 1: Visualization of SimBus (Source: Goudappel)

Because the old SimBus was not up to par anymore with modern software standards, it was decided a new version is to be built: SimBus Pro. In SimBus Pro, the simulation of bus stations should be improved. New visualization opportunities arise as well, on top of long awaited opportunities for new functionalities. At the end of March 2022 a first version of the new program was finished, but work continues to improve the program.

In essence, SimBus Pro is an extension to the traffic simulation program of Vissim. The scripts, programmed by Goudappel, add additional functionality to the simulation program that support buses in a realistic way. To add onto the advantages of using Vissim to program SimBus Pro: in Vissim projects can support more than the bus station, for example the surrounding network. The network surrounding the bus station can be simulated to produce a multimodal model. The programming is object-oriented and done in Python. In this way, buses can be programmed as objects and each instance of a bus can carry information with them about for example timetables, passengers, and routes.

2.2. Involved Parties

The party that commissioned this project is Goudappel. With the arrival of SimBus Pro Goudappel intends to provide clients, such as municipalities, provinces and concession holders, improved and more realistic answers to their problems. On top of this, with the new program better visuals are also on the table. Especially the new and improved visuals will help clients in thinking and visualizing proposed solutions and advice on top of the raw data that already existed. Having an improved simulation application will help clients come to better informed decisions and greater understanding.

Goudappel is an engineering and consultancy firm in the Netherlands, and focuses its projects mainly on the Netherlands itself, but has a vision to improve cities worldwide. Goudappel is one of the leading firms in mobility engineering in the Netherlands and has a lot of expertise in designing and modelling all kinds of solutions in network management, public transport, parking, pedestrians and cycling.

2.3. My contribution

One of the functions that must be improved is the simulation of passengers that are boarding and/or alighting. At present, the stop time is assumed to be normally distributed, where mean and standard deviation are determined from GOVI data.

GOVI (Grenzeloze Openbaar Vervoer Informatie in Dutch) data contains the times buses arrive and depart from stations. This is for example used for dynamic information screens to let passengers know when the next bus will arrive (InTraffic, sd). From this data, the arrival and departure times can be retrieved as well, which are used to determine the stop times in SimBus Pro. It is unknown however, how reliable this data is. Vissim offers the possibility to determine the stop times based on the number of people getting on and off. Unfortunately, Goudappel lacks the knowledge to apply this functionality properly and reliably. Goudappel would therefore like to know how the stop times can be properly set in Vissim, so that boarding and alighting passengers are taken into account. The resulting figures should be reliable and realistic as well. Afterwards, this functionality has to be implemented and coded in SimBus Pro, creating a model that reflects reality more closely. This improved model can be validated and used in projects.

Besides the fact that the boarding and alighting process can be simulated more reliably as a result of this new functionality, secondary effects can be visualized as well. Pedestrians can possibly hinder arriving and departing buses and cause small delays. This analysis is one of the important reasons to model passengers but is outside the scope of this project. To add onto the positive effects, adding models of people into the simulation as passengers, provides improved visualizations and adds a visual check to see if everything is working correctly.

Before implementing anything in the model or Vissim, some data will need to be gathered as well. Through an observational study real world data will be gathered on the time it takes passengers to get on and off a bus. This data will be used as input for the model instead of GOVI data and on top of this used to validate the GOVI data. A very interesting aspect of this project for Goudappel is to look at the reliability of GOVI data.

It is important to note that within a bus station, the boarding and alighting of passengers are mostly seen as two different processes. When a bus come into a station, it lets all passengers get off, and afterwards either goes to a buffering stop to wait for its next service or stays at the same spot to wait for its next service. When the new service starts, new passengers are allowed to board. In the modelling process, the boarding and alighting will thus be two separate processes, and treated as such when possible. Only in the case a bus needs to board and alight passengers at the same time, the processes will be combined.

2.4. Research objective

The research objective is to determine realistic stop times in SimBus Pro by modelling and simulating the boarding and alighting of passengers in more detail with the use of observational data.

3. Theoretical Framework

In this chapter the theoretical or research framework will be discussed. In Figure 2 below the complete framework is shown. Broadly speaking, the whole project is a modelling project and will follow a design cycle. The final result will be the implementation in the model itself. This will be done through two research objects: Boarding and Alighting. The third object connected to this is Visualization. This will be a connecting factor between the simulation of the processes and the visual aspect that is seen within Vissim. This requires the input of both the boarding and the alighting processes to function correctly.

On both the boarding and alighting of passengers a literature study can be done. It may be possible sources have both of these processes in the same paper but it can be reviewed whether a lot of or any theoretical thinking has been done in the past on these subjects. This is discussed under Past Research.

Next to the literature review, the observation data gathered from the observational study will be a major input for these whole project. From this data, the stop time of the bus can be determined related to the number of passengers getting in and out. In addition, this data can be used to validate the GOVI data used at the moment to find out how reliable it is. Both of these applications will be further explained in the Design Cycle. The validation of GOVI data and observation data together serve as input for the model processes.

The final step is validation of the boarding and alighting behavior in the model at the end of the process. Using a validation dataset this will be statistically checked. After this, improvements can be made to the model where necessary. This will also make the design process iterative, typical of a design cycle. If the model is not good enough, one can go back in the design process and improve the necessary areas.

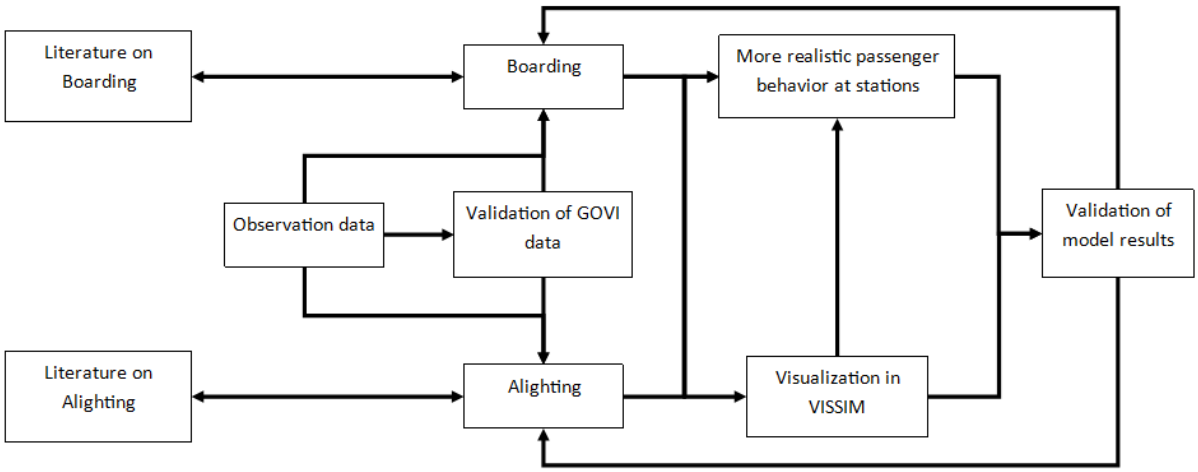

Figure 2: Theoretical Framework

4. Design Cycle

In the modelling of the processes, a design cycle will be used. This design cycle will ensure the modelling process will be a complete and logical process. An example of a design cycle and its 4 steps is shown below in Figure 3. In this section, the research part of the thesis will be discussed. The problem was defined in the context under My Contribution.

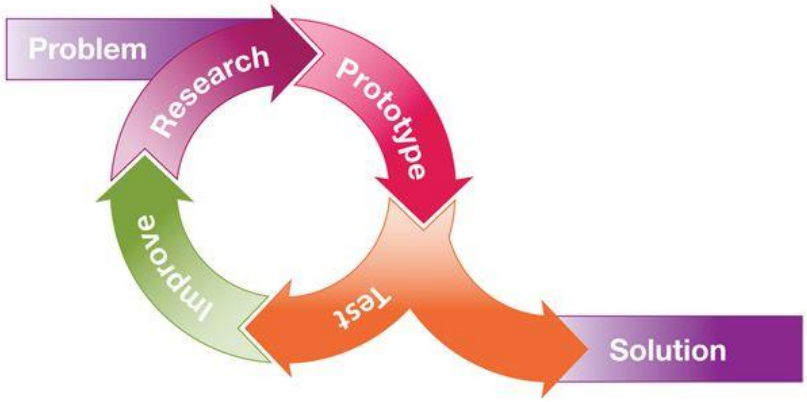

Figure 3: Design Cycle (Eddleman, 2016)

4.1. Research questions

From the context and contextual framework three main phases to this thesis can be identified. These are the observation study, validation of GOVI data and last but not least the implementation in SimBus Pro. For this reason, there are three main research questions, each aimed it its own phase respectively.

What relation fits between the observed stop times and number of passengers boarding/alighting?

One of the key points that should be determined from the observation study is the relation between the stop times that were observed and the amount of passengers that were either alighting or boarding.

How does GOVI data compare to data from observations?

In this phase, the key point is the comparison of GOVI data to the observation data to find out how reliable GOVI data is. The answer can be used in the model to make it more generic and realistic, as described in the Research Methods.

How will the boarding and alighting of passengers be implemented in SimBus Pro?

The third and also biggest phase of the project is the implementation. The main question is how to implement the processes in full completeness, but several sub-questions should be answered before this:

How can the calculation of stop times be correctly implemented?

How can the relation between stop times and passenger counts be correctly implemented?

How can Vissim correctly visualize the processes?

How are other factors taken into account and modelled?

After answering these sub-questions, the main question can be answered and thus the entire implementation can be programmed. This section is very oriented towards practicality, because in the end here part of a program should be delivered. The answers to these questions can also develop during the different iterations of the model.

4.1.1. Observation Study

The first phase of the project is an observation study. In this study, the behavior of passengers in bus stations and bus arrival and departure times will be observed in order to record the necessary data.

During this study, a number of variables is accounted for that can be considered for the validation of GOVI data. These are the type of bus station, the transport companies, and whether the station is canopied or not. All of these factors can influence for example the GPS-systems and thus the exact times that are saved in the data. The study will be done in three locations: Arnhem Centraal station, Deventer station and Enschede station. The first two stations are showcased below in Figure 4 and Figure 5. Arnhem Centraal station is a roofed bus station with a separate boarding and alighting platform and buffer stops. The boarding platforms are in the shape of a herring bone. In Figure 5 on the right it can be seen from the picture that the bus station is the ground floor of a big office building. Deventer is not a roofed bus station and has the boarding platforms in the form of an island. In both stations multiple transport companies have concessions. On top of the two mentioned stations, the bus station of Enschede will be used as well to gather data points about passengers boarding and alighting.

Figure 4: Aerial picture of Deventer station (Source: Cyclomedia)

Figure 5: Aerial picture of Arnhem Centraal Station (left; Source: Cyclomedia) and a street-picture of the station from Stationsplein (right; Source: Google)

The observational study will be executed for about 3 hours. In these 3 hours, both peak and off-peak hours will be taken into account. Within these 3 hours on the busy stations, there will be enough buses coming through to generate sufficient data points needed. On Arnhem Centraal, strategic locations will be found to do so, as the platforms are in the shape of a herring bone and only a part can be captured. On top of this, Arnhem has a few specific platforms for alighting only. The other platforms are for boarding only. For the session in Arnhem, more time will be taken to measure at two different spots, to capture both boarding and alighting. Only having a few alighting platforms does mean a lot of buses will come through, giving the same amount of data points in less time. For all stations, one session will be executed to get the calibration datasets. In addition, one more session will be executed to gather the validation dataset at one of the stations.

In the execution, the observations will be recorded as well. For one man it is too much to keep track of all factors at the same time and write everything down correctly. During the observation study, both the arrival and departure times will be tracked by hand. For all other data, the bus station will be recorded. All other factors will be analyzed from the recordings afterwards. The recording will help with the addition of more detail and double-checking as well, as the recordings can be accessed at a later time and reviewed again. More importantly, they will prevent too much needing to be written down during the execution. All passenger related data will thus be analyzed in the recordings.

4.1.2. Validation of GOVI Data

GOVI data is used in everyday life to share travel information to passengers in both information screens on the stations and with other parties that give this information to passengers, such as travel apps like 9292. The time until the next bus arrives can be estimated and dynamically changed on the screens accordingly. In this dataset the exact times buses arrive and leave from stations is recorded as well, which is the interest of the validation. (InTraffic, sd)

As described in the observation study, the arrival and departure times of buses will be recorded, next to the passenger data. The GOVI data for the exact times the observation study is executed can be retrieved from the database. These arrival and departure times from GOVI are statistically

compared with the arrival and departure times from the observation study. From this, it can be concluded how reliable GOVI data is, or whether it may consistently over- or underestimates the times.

The GOVI data can only be retrieved about half a month after the month the observations were executed in. This means the validation can happen around half May. As a result, this phase is not the second phase in the planning of the project, but in the planning this is accorded for.

4.1.3. Modelling and implementation in SimBus Pro

The phase of modelling the boarding and alighting processes is an iterative process. In the first iteration, a model with base functionalities will be programmed. In other iterations, extra improvements, detail and bugfixes can be added into the model. What is not known is how reality compares to GOVI data. In the validation of GOVI data, the comparison between reality and GOVI data is already made. When looking at boarding and alighting, no external factors or exceptional situations will be modelled in the initial base version.

The first step in this phase is to set up a conceptual model, describing the processes and providing input for this phase. Another introductory step is to investigate how currently SimBus Pro is programmed and how it works together with Vissim to create one output in form of a simulation. Connected to this is the step to take a look at what needs to be done in modelling terms to get passengers working in the model. This is done by looking at the different settings in Vissim related to this and investigating what they do and how they can be useful. After these initial investigations the complete conceptual plan can be implemented in SimBus Pro and Vissim.

A different step compared to the others is the visualization. In principal, the theoretical and technical sides come first, as they are more important. Good visualization will help users a lot however. Not a big part of the iteration will be spent on this, as Vissim should automatically help a lot in providing this once the functionalities are implemented correctly, but enough time should be spent to make the model look realistic, on top of behaving realistically. Because the visualization will be with the help and style of Vissim, not much thought has to be given to how the visualization should look like for users to understand it correctly. For viewers watching the end result, it should be a representation of a real-world phenomenon, which Vissim can do well by itself.

To conclude, in the base version, observation data can be related to the number of boarding and alighting passengers, from which SimBus Pro and Vissim determine the correct number of passengers and stop times accordingly. Vissim also visualizes this correctly in the simulation.

Finally, it can be tested whether the implementation works correctly. The validation of the model is done using observation data as well, but from another dataset that was taken at a different time and/or day. If the model does not function correctly, in the next iteration, necessary parts can be improved. Other detail can be added as well to make the simulation more realistic in the time that is available.

4.2. Theoretical Base

In this phase of the project, the theoretical foundation will be laid for the implementation. This foundation will include finding relations between amounts of passengers and stop times, and validating the GOVI data used as input.

4.2.1. The reliability of GOVI data

The first phase of the prototype that will be discussed in this report is the validation of the GOVI data. This data is the major input for the timetables that are used in the model, and as such has a major impact on the model and possibly the implementation of passengers. For this reason, the validation of this data is discussed first. From the observation study, the complete list of observed buses can be found in Appendix A – Observed buses.

The validation has two approaches: a purely time-based approach and a geographic approach. In the time-based approach, the retrieved departure, arrival and total dwell times are compared against the observed departure, arrival and dwell times. In the geographic approach, the difference in geographic location is compared between the observed departure time (which is always at the stop itself at the moment a bus starts to drive) and the recorded departure time in the data.

As much data was used as possible to give the best picture possible, but not all data was usable. It can happen that vehicle numbers or departure and arrival times are not recorded correctly and instead show a value of 0. About 5% of the data was not usable due to this reason.

In the time-based analysis, the recorded arrival and departure times are directly compared to the observed arrival and departure times. In APPENDIX, the graphs are shown for both stations. From the graphs, it can be noticed that Arriva is constantly late recording departing buses, and constantly early recording arriving buses. Keolis is much closer to the actual departure and arrival times, though they are also regularly too late with recording arriving buses.

In the geographic analysis, a bus's geographic location at the moment it is recorded as departed is compared to the moment the bus actually departed. The goal of this analysis is to look for a geographic relation between the recorded departure times. It is important to note that this analysis is possible because the station was recorded on video from a strategic observation location. Again, in APPENDIX, the graphs showing this analysis can be found. From the figures, it can be seen that there definitely is a geographic relationship with the recorded geographic locations. For Keolis this is shortly outside the platform and for Arriva this is at the edge of the station.

From both approaches, assumptions can be made about the reasons why there is so much difference between observed and recorded times. The GPS systems or other factors could influence this to produce the recorded results. However, speculation does not suffice for strong conclusions. For this reason, two experts were interviewed. These were Erik Oerlemans, a public transport advisor from Goudappel, and Marcel van der Holst, a business analyst from transport company Arriva.

During the interview, questions were asked about the working of the systems supplying the data, factors influencing the data collection, and what they knew about the differences in recorded time versus observed time.

From the interviews a number of conclusions could be drawn. According to both Erik and Marcel, GPS-polygons are drawn around a stop or station to identify with GPS-systems whether a bus or

other vehicle has arrived. This is not the only way however, as arrival times can be determined with an odometer as well, which measures wheel rotations and is also used for measuring mileage for example. Between different bus types there is not a lot of difference. Within a company, all buses usually use the same expensive software on their board computers, and thus give the same results.

A major factor that plays a role in this is the surrounding area. If a station is close to a lot of high buildings or even underneath a building, such as Arnhem Centraal, the transport company can choose to enlarge the GPS-polygon that is used o measure arrival and departure. This is no more than a balancing act between the precision of the recording and the reliability of the system.

According to Marcel, Arriva knows of the deviation between recorded and real times. Internally, they know for each station they service what the polygon is that they made, the average time difference from the polygon to a stop and back, and the distance from the polygon to a stop and back. However, they specifically choose not to alter or clean their data in any way to keep the main objective of the data as reliable as possible: Passenger information.

An interesting discussion that can be talked about is the use of this data for measuring punctuality. With for example the GPS data of Arriva, they will arrive early at station more often and leave too late more often than is actually the case in reality. Because this data is also used in measuring punctuality, it gives rise to discussion about the punctuality as well.

For the implementation, this will mean a correction has to be made for the concluded deviation between recorded and real times. With the correction, the dwell time will approach reality more closely and is thus better for simulating the timetables realistically. The correction can be made in two ways: in a time-based solution and a geography-based solution.

For a time-based solution, the total dwell time from the GOVI data can be corrected. Depending on the software solution used by a transport company, an average time difference can be set up for a station. When you know the type of solution used by a company, an educated guess can be made for the average time or the time could be provided by the company for an existing station.

In a geography-based solution, the geographic location of buses can be used to correct the dwell time. Points in the network can be defined where a bus would enter a GPS-polygon. The exact driving time between the polygon and the stop in the simulation can be subtracted from the total dwell time. For leaving the polygon, the average time to the polygon can be subtracted. However, this solution will only work for GPS-systems, which is not the only system that can be used.

4.2.2. The relation between passenger volumes and stop times

The observation study had two major focal points: Gathering the stop times to validate GOVI data, and counting the amount of passengers that board and alight, including the time it took for them to do so. In this and the following chapter the results of both will be discussed and prepared as input for the model.

After the execution of the observation study, the results were digitized and further analyzed. From the boarding and alighting passengers, the total time taken and average time per passenger can be observed and calculated. For both boarding and alighting, this was done per group of passengers. Groups of less than 3 people were not considered big enough for this. When using 1 or 2 people to determine the necessary time, there are two consequences: without counting more precise than per second, the data is not precise enough to measure per passenger, and a single passenger has a lot more variability and uncertainty in time taken. It is also a lot more work to include all of those

groups, because those numbers occur more often. In total 32 data points are recorded for boarding and 39 for alighting.

For alighting, a power function was found to be the best fitting function, with an R^2 of 0.88 rounded off. Next to the good fit to the data, a big advantage of the power function is that it does not flatten quickly. It takes a very long time before it does so, and thus prevents a large difference in passenger numbers for only a small difference in time. It does this while keeping to the trend of the data, which is the slow flattening of the curve. The function fitted to the data is shown below:

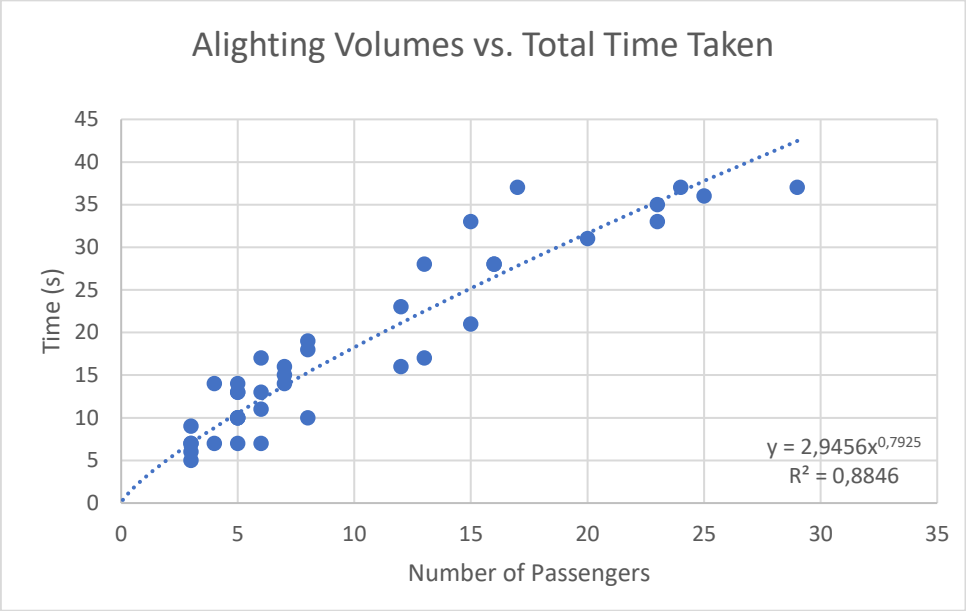

Figure 6: Function fitting for alighting

When rewritten, the function will be:

Equation 1: Function for alighting

$$n = (\frac{t}{2.95})^{\frac{1}{0.79}}$$

Where *t* is the time in seconds and *n* the number of passengers. This is the definitive function for calculating the number of alighting passengers from the dwell time.

For boarding passengers, a different approach is made. Ideally, this would be done by finding a relationship between the total dwell time and the total number of passengers boarding. In Figure 29 below these two are plotted against each other, plotted per station. It can be seen however, that there is no relationship at all to be discovered between the total dwell time and the number of passengers. There is a minimum or baseline to be discovered because passengers still need to board the bus. The baseline is based on the smallest gradient from (0,0) to a point in the figure. Apart from a minimum boarding time that logically increases when passenger numbers increase, these two variables are completely independent. Due to the design of the stations and timetables, the bus stations all behave a little different in this aspect. On Arnhem Centraal, buses wait in buffer spots before going to the platform more often than not, and on Deventer buses spent a lot of time standing on the platform instead of buffering. This difference is clearly found in the graph. From this however, it can be concluded that using this data no relationship can be established between the dwell time for boarding and the number of passengers boarding. As a conclusion, the number of passengers boarding cannot be determined reliably at all and is observed to be random.

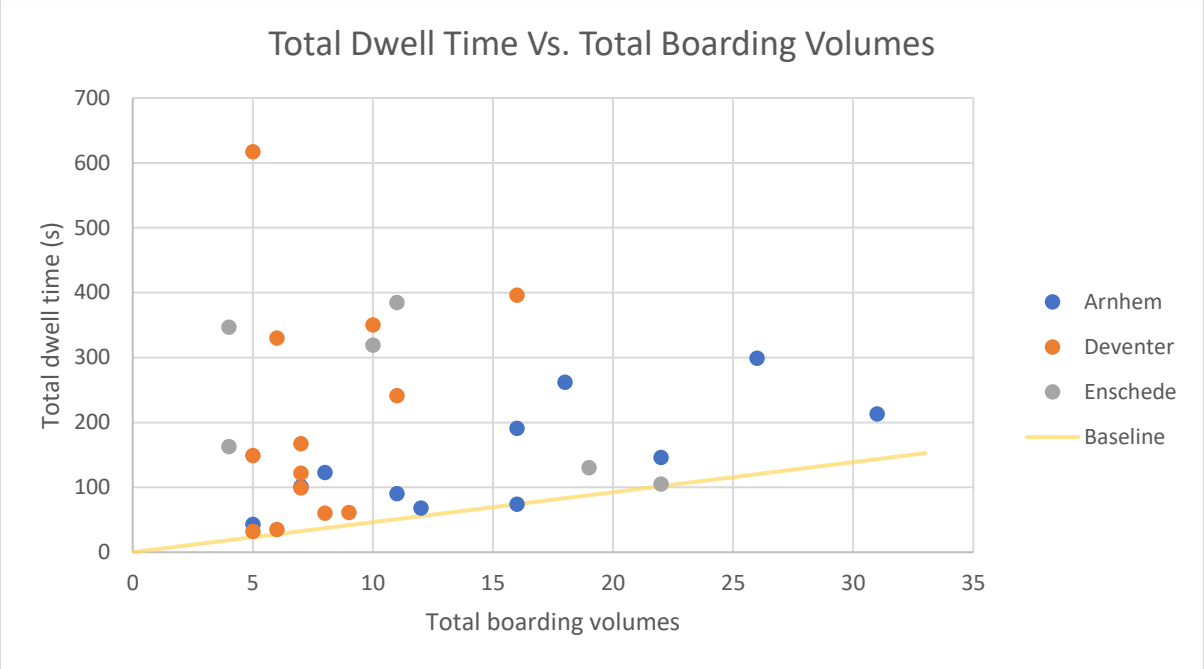

Figure 7: Total Dwell Time Vs. Total Boarding Volumes

SimBus Pro tries to reflect reality as good as possible, and one of the requirements for this project is to have people boarding buses in the model. However, the number of passengers cannot be determined reliably with a mathematical function. For this reason, the aim will be to program a simple function that forms a basis for further improvement and extension, is visually realistic, and performs well. Although there is no relation between total dwell time and total number of passengers, there is a relation between the time it takes for passengers to board and the number of them. This relation is shown below:

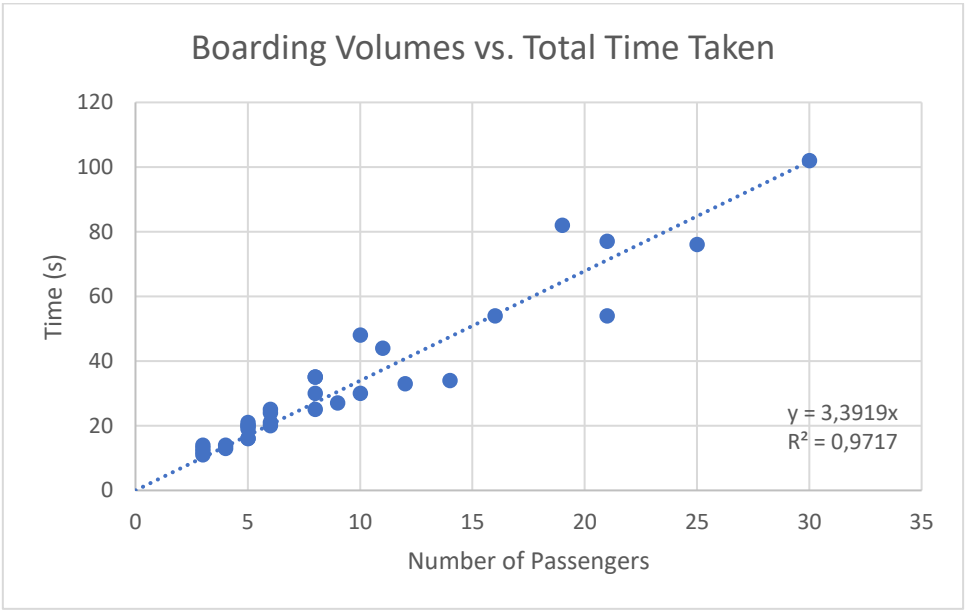

Figure 8: Function fitting for boarding

It is actually a strong relation with an R^2 of 0.97. This function will be used to determine the maximum number of passengers that can board within the dwell time and when rewritten to the form *n* = … gives the following function:

Equation 2: Function for boarding capacity

$$n = \frac{t}{3.39}$$

Where *t* is the time in seconds and *n* the number of passengers. However, not 100% of the time a bus is standing still people are boarding. This means Equation 2Equation 6 is the theoretical maximum amount of passengers that can board. Because the amount of passengers boarding is observed to be random, a solution is sought in the area of distributions.

Two exponential distributions are put forward and tested using the Chi square test: one distribution for boarding, with a λ of 0.16. The sum of the Chi Square values is 3.01 for this λ. The critical value is 14.07 for 95% confidence, and the sum of the values is lower than this. This gives the distribution:

Equation 3: Exponential Distribution for boarding

$$F(x;\lambda) = 1 - e^{-0.16x}$$

And a second exponential distribution for determining the amount of alighting passengers when alighting and boarding is combined. Using the data for alighting, we get an exponential distribution with an optimal λ of about 0.12. the sum of the Chi Square values is 7.41 which again is lower than the critical value of 14.07 for 95% confidence. To enforce the maximum amount of passengers that can alight in the bus's dwell time the same power function (Equation 1Equation 5) can be used as is used for alighting only. The functions are thus statistically valid. The second distribution thus is:

Equation 4: Exponential Distribution for alighting

$$F(x;\lambda) = 1 - e^{-0.12x}$$

To conclude, the 3 actions of boarding, alighting and the combination of both two all have a theoretical function or distribution that describes the amount of passengers boarding and alighting. Table 1 below summarizes which equations and/or distributions are used for which actions.

Table 1: Summary of used functions

Action	**Used function or distribution**
Alighting	Equation 1
Boarding	Equation 2 and Equation 3
Combination of boarding and alighting	Equation 1 (alighting), Equation 2 (boarding) and Equation 3 (boarding) and Equation 4 (alighting)

4.2.3. Conceptual Model

The completion of the previous chapter means the theoretical basis for passengers is complete. The following and final step before the implementation phase can begin is the conceptual model. The complete model for passengers is shown below in Figure 9. According to (Wand & Weber, 2002), the conceptual model provides input for the modelling process and helps in documenting and checking the original requirements of the model.

At the start of the simulation, the code iterates over all buses. It is checked whether boarding is an action that a bus will perform during its trip. This is done by iterating over every bus in the timetable. If so, the pedestrian volumes that should be generated are determined from Equation 2 and Equation 3 based on the average dwell time. The average dwell time from the timetable is used because a bus will only know its exact dwell time once it is in the network. In this way, passengers can also be generated before the bus has arrived at the station, which is more realistic. Before being given to the model, the passenger amounts are checked against the theoretical maximum amount of passengers that can board within the average dwell time (the fitted function) and against the capacity of the bus. This ensures groups of passengers do not take longer to board than the preferred dwell time.

The computation for alighting can be done during the simulation. Once the bus knows its next action and dwell time, which in the model is called the desired dwell time, the number of passengers alighting can be computed. Here one of two possibilities is possible for alighting: the bus has alighting as its only action, or the only action planned is boarding and alighting combined. This means the action will be a combination of alighting and boarding. Based on this, the number of passengers is calculated from Equation 1 in the first case, or determined using Equation 1, Equation 2, Equation 3 and Equation 4 in the second case. Again, the amounts are checked against the capacity of the bus and the maximum amount of passengers that can board in this dwell time.

After determining the amount of boarding or alighting passengers, the correct attributes in Vissim can be set, completing the computation and letting Vissim carry out the processes.

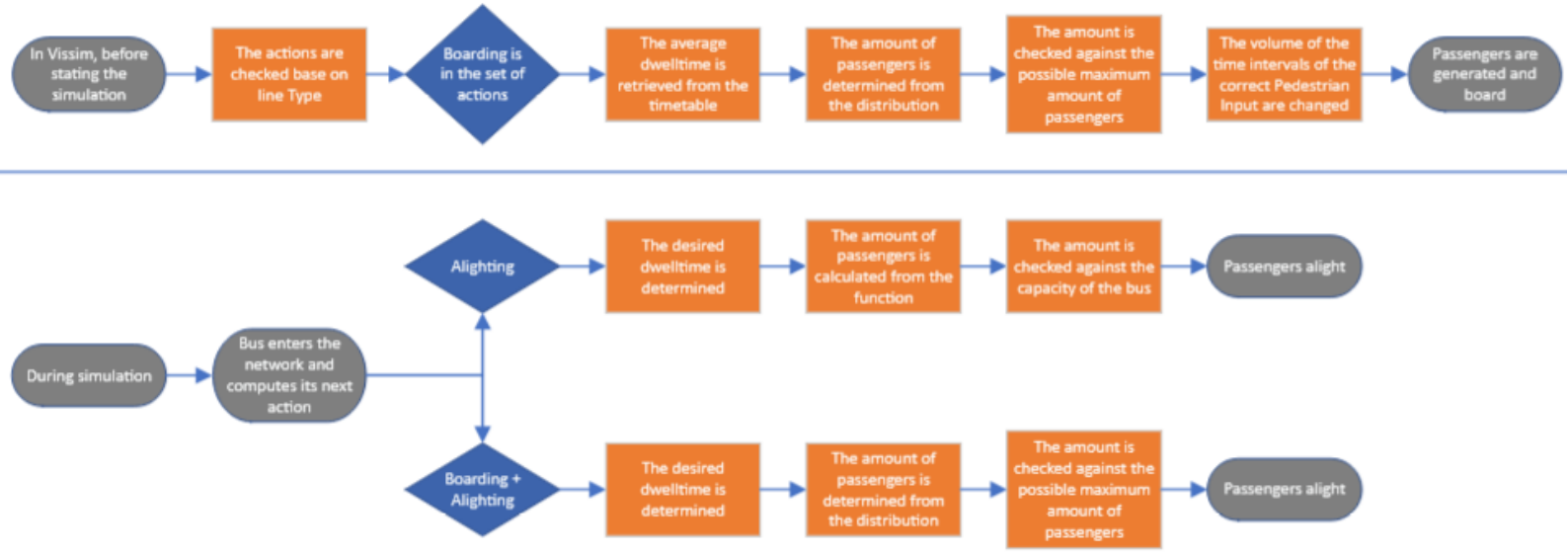

Figure 9: Conceptual Model

4.2.4. Bus Capacity

In the conceptual model, bus capacity is mentioned as a limiting factor for the maximum amount of passengers. A capacity needs to be defined in the model as a small input. In the current version of SimBus Pro 3 standard bus types are modelled: a 12 meter bus, 13 meter bus and 18 meter bus.

After researching this topic, bus capacity turns out to be completely vehicle-specific. The amount of passengers a bus is allowed to carry is determined by the manufacturers of the buses and can differ a lot for buses of the same length. Looking at the vehicle types that transport company RET uses, the capacities for a 12 meter bus range from 80 people to 98 people. For this reason, a list with average capacity values will be used in the model. Derived from the vehicles used by RET, a 12 meter bus carries on average 88 people, a 13 meter bus will use 88 people as capacity as well, and an 18 meter bus carries 126 people. These values will be used in the model as capacity values. (RET, 2021)

4.3. Prototype

In the prototype stage, the research done will be converted into a model, or in this case, expansion of an existing model. This stage will consist of the testing and programming in Vissim and SimBus Pro.

4.3.1. Testing

With the conceptual model in place, the modelling in Vissim can commence. The first major point is to test what related variables can be used and influenced to enable or influence passengers in Vissim. From this, a list of requirements for a Vissim network can be set up.

From initial tests in Vissim, it turned out to be quite simple to enable passenger flows in the Vissim networks, both with and without the use of SimBus Pro. The tests gave a small list of requirements for the network to enable passenger flows. In this list, it is assumed that all alighting passengers, once alighted, will always go to a set exit of the model and do not change from one bus to the other. The list is shown below:

- Each stop needs a platform on its right hand side (When vehicles drive on the right side of the road)
- Each stop needs a waiting area for the passengers to wait for the bus. This area is coupled to the stop.
- At least 1 exit and/or entrance for the network has to be implemented to generate both start and end points for the routes of the passengers.
- There needs to be a valid connection between the platforms and the exits/entrances. This can be done via pedestrian areas and other means, such as escalators.
- There need to be pedestrian routes connecting the entrance with the waiting areas and routes connecting the platforms with the exit.
- Make sure the alighting location distribution is correct for every partial PT line and boarding location distribution for every waiting area.

Below in Figure 10 a 3D picture is shown of Vissim network, including the scripts of SimBus Pro, in its simplest form, with a bus boarding passengers. The network is also in its simplest form for SimBus Pro to work: A loop with one entrance, and one exit. This network was used for the initial testing. The model includes the platform of the stop in purple, waiting area in blue, entrance/exit in green and walkable areas connecting the mentioned areas in grey.

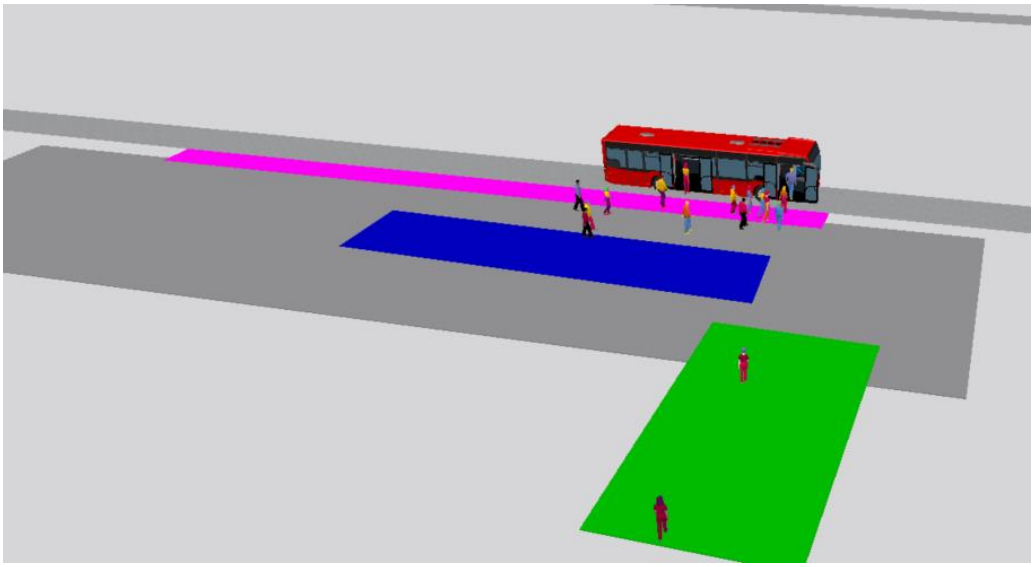

Figure 10: Vissim test network with passengers boarding a bus

Some effects from Vissim that may have impact on further implementation (without influences of SimBus Pro) were also noticed:

- The standard bus in Vissim waits automatically on any alighting passengers, even past the scheduled departure time.
- After the doors are closed, Vissim does not wait on any boarding passengers.
- After alighting all passengers, the bus in Vissim will also wait on its scheduled departure time, if this has not been reached.

A bus station does not exist of one platform only, it usually has a lot more. In the definition of the network, alighting passengers did not need any other changes to the network to work correctly. For the boarding passengers one problem arose that had to be solved: Boarding passengers need a way of knowing which platform they need to go to. This problem and its solution is further discussed in chapter 4.3.5. Pedestrian Routing Figure 11 shows a station with two platforms, where passengers are alighting on the second platform and passengers are boarding on the first platform.

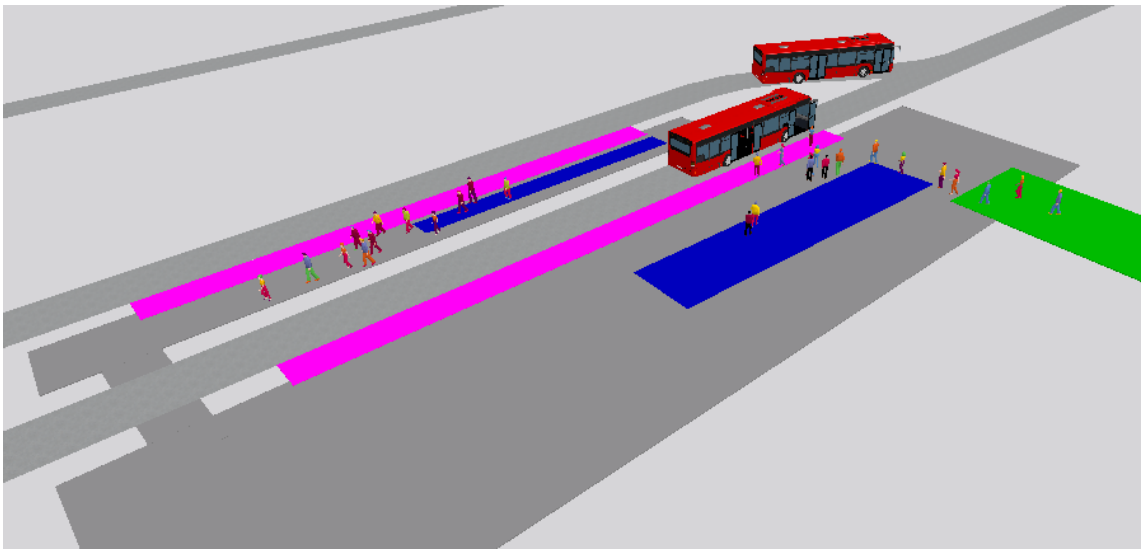

Figure 11: Passengers boarding and alighting on multiple stops

4.3.2. Variables

In the end, a number of variables from Vissim were used to come to the current implementation. In the table below, a summary of the currently used variables is used. The table also includes interesting variables that could be used and changed, but are not used at this moment. All variables refer to the processes connected to passengers. Occupancy and Volume work as inputs, while AlightPerc, AlightTm en BoardTm directly influence passengers. DoorClosDur en DoorLockBeforeDep directly influence the vehicle. RelFlow influences the routing of the pedestrians.

Table 2: Used and possible variables in Vissim

Variable	Falls under object	Description	Used/changed
Occupancy	Vehicle	The number of people in a vehicle	Yes
Volume	Pedestrian Input	The volume of pedestrians that will spawn	Yes
AlightPerc	PTLineStop	The percentage of passengers alighting	Yes, set to 100%
AlightTm	VehicleType	The time it takes 1 passenger to alight	No
BoardTm	VehicleType	The time it takes 1 passenger to board	No
DoorClosDur	VehicleType	Door closure duration	No
DoorLockDurBefDep	PTLineStop	Door lock duration before departure	No
RelFlow	PedestrianRoutingDecision	The relative flow of the route to a destination	No
DwellTm	Pedestrian	Dwell time of the pedestrian	No

4.3.3. Assumptions

In the model a few small assumptions have to be made to not overcomplicate this project. One black box there is no sight on is the total occupancy of the bus on lines that do not start or end at the station. It is assumed that all passengers alight at a bus station, as it is impossible to know from the available data how many passengers stay in the bus on lines that do not terminate at the station. To compensate for this, the difference between the initial occupancy and the occupancy after exiting the network will be recorded in the results of the model. This leaves the exact occupancy out of the view, and also leaves the opportunity open to implement this in the future.

Another black box is the direction passengers travel when coming in or out of the network. On top of this, people sometimes transfer to another bus line. Implementing this in the time and data available for this project is impossible. For this reason, one or a small amount of central points will be chosen where most people come from and/or go to. These points will be the entrance and/or exit of the network for passengers. This does mean, to make realistic networks, a little knowledge of the passenger flows in the ‘to be implemented’ area needs to be known.

There are also some variables that can be influenced and have effect on the stop times. These variables are the door opening and closing times, delay between door lock and driving and the

location distributions. The location distributions describes the distribution of which door passengers board or alight from. Without enough time, these values can be kept on their standard values from Vissim. These values are also mentioned in Table 2.

4.3.4. Boarding and Alighting

With the network ready to handle passengers, the functions for boarding and alighting can be set up. In the conceptual model the functioning of boarding and alighting are already discussed. In this chapter, the coding and some of its challenges will be discussed more in-depth.

During the coding, a high standard is aimed for. All code is strived to be as general as possible, so it could be used with any bus station modeled. On top of this, it is made as structured, concise and well-documented as possible, to ensure readability and workability with the code. Functions are made to be built upon and have the opportunity to be expanded in the future.

For ease of turning passengers on and off, there is a global variable in the main scripts affecting the preparation and simulation scripts that can turn all code passenger-related on or off. This makes it very easy to switch passengers on and off in the model. Both with the passenger code turned on and off the model behaves seamlessly as intended.

In the timetable, the dwell time distributions are already defined. In the current version of SimBus Pro, the dwell time is determined in the model at the bus stop itself, the timestep after the bus is standing still. If alighting passengers are calculated and set in this timestep, it is 'too late' for Vissim to recognize there are passengers that want to alight from the bus. For this reason, the only major change to the implementation of SimBus Pro itself had to be made. SimBus Pro makes use of decision points in the model where, amongst other things, the route and action(s) of the bus are determined before sending the bus along a route. Instead of defining the dwell time at the stop itself, the total dwell time is defined at each of these decision points. As a result, the calculations for alighting can be done at this decision point as well.

For alighting only, a new function is set up to accommodate the calculations. In the correct situation, it will be called when a bus addresses a decision point. The dwell time will be determined and then the function called. As described in the conceptual model, it should calculate the number of passengers and ensure the theoretical maximum is not exceeded. Afterwards, it sets the occupancy of the bus correctly.

The calculation for boarding is done in different stages of the simulation. Pedestrians could be generated once a bus knows its action and bus stop, but this is too late to be realistic. This only gives a small amount of time to generate all pedestrians that should be boarding and can thus cause major groups to suddenly spawn in this short time. For this reason, the volume of pedestrians and generation times should be determined in a different stage of the simulation. In the timetable, two columns are added for the start and end times of when the generation should happen. During the preparation of the model, the correct times pedestrians should be generated are determined dynamically for the model.

As explained in the theoretical foundation, an exponential distribution needs to be used as well when alighting is done at the same time as boarding. For the combination of boarding and alighting, the two separate processes come together. However, they do not have to be in the same function in the code. Boarding is already defined in the previous paragraphs, and alighting too. To make the combination, the alighting function is extended. Within the function, both versions of alighting can

be called when needed. A case is added where the number of alighting passengers can be determined from the exponential distribution instead of the function. Even though boarding and alighting are separated in the code, they can work together perfectly in Vissim.

For reproducibility of simulation runs, seed values are a great and common solution. The used distributions for boarding and alighting have the ability as well to use seed values. The volumes that should be generated thus need calculated at the start of the simulation, where the seed value of the simulation run can be used.

4.3.5. Pedestrian Routing

Boarding and alighting are part of the actions a bus takes when it uses a bus station. Next to the bus, passengers also have actions in a bus station, and have to come from and go to somewhere. One of the main side effects that can be analyzed with pedestrians in the station is the hindrance and delay of buses at pedestrian crossings. To enable this analysis, pedestrians need to have their entrance or exit to the network outside of the station. Pedestrians need to function correctly in the model as well, and as side-effect of including pedestrians in the model, they need to be routed to their correct platforms.

When a passenger alights from a bus, the process is quite simple: the passenger walks to the exit of the network. For pedestrians coming to the station to board a bus, a number of situations can be identified that influence where pedestrians go:

- The bus arrives at the correct platform and people can board
- The pedestrian is too late for the bus and has to return to the exit of the network
- The bus arrives at a different platform than where the pedestrian is (going)

After several successive attempts in the modeling phase, expanding the possibilities each time, SimBus Pro is fully able to route pedestrians the way they need to go. The elaboration of this solution can be found in Appendix D – Pedestrian Routing Solution.

4.4. Validation

The final stage after implementing the complete conceptual model is the validation of the model. For the validation, three indicators were chosen to validate the model with. These are visual validation, hindrance of buses through pedestrians, and groups of alighting passengers. To able to validate the model, the station of Deventer has been built in Vissim. Both a complete Vissim-network and a timetable have been set up.

In the visual validation, the representation of reality and behavior of the model are checked by directly looking at the simulation running. While looking at the model, no peculiarities could be found between the implementation, the conceptual model and the coding. On top of this, the passengers and buses behave as in reality. There are only a few issues that are not the fault of the implementation but of other factors. These known issues are listed below:

Table 3: Known Issues

No	Issue	Problem of:
1	Passengers cannot choose a bus if there is more than 1 at a stop. They will always go for the front bus.	Vissim
2	Pedestrians do not stand still in waiting areas, even though they should.	Vissim
3	Buses cannot drive backwards to get around a bus sitting in front of them. This means they have to wait until the bus in front leaves the platform.	Vissim
4	The time passengers take in VisWalk to board and alight are short and not influenceable; The variables BoardTm and AlightTm do not work with VisWalk.	Vissim
5	Bus routes cannot become very complex; at a decision point, a route to every bus stop needs to be defined and routes cannot go to another decision point.	SimBus Pro itself

Figure 12: Model of Deventer station during simulation

As mentioned in the Research Context, a major secondary effect that can be studied is hindrance or delay time due to pedestrians. In the model, a solution has been found to measure and quantify for each bus the delay they experience. The elaboration of this solution can be found in Appendix E – Pedestrian Hindrance Solution. The crossing for pedestrians does not have a crosswalk, and thus road traffic has priority in theory. From observations however, it was noticed that sometimes buses have to wait. This could be because the bus driver himself lets the pedestrian(s) have priority or because people are not aware of the coming bus and cross. For this reason, the buses are not strictly given priority in the model, only from a small distance the pedestrians will wait for a coming bus.

To determine the average hindrance time for a simulation run, 10 runs are executed. In one run, a timetable with 78 buses is simulated. Out of all 78 buses, on average 2 buses are hindered by pedestrians each run. In total over the 10 runs buses experience about 40 seconds of hindrance due to pedestrians. Counting all hindrance (including pedestrian hindrance) buses experience 1089 seconds of hindrance. This means about 4% of the total hindrance experienced is because of pedestrians. It is important to keep in mind that the used network was designed to give buses priority most of the time. The total amount of hindrance is quite high however. After checking this in the model, it was found that it was a lot higher because of known issue #3, with buses sometimes needing to wait a long time before the buses in front of them have left. The pedestrian hindrance per run is shown below in Figure 13. From this we can draw the conclusions that this added functionality works as intended.

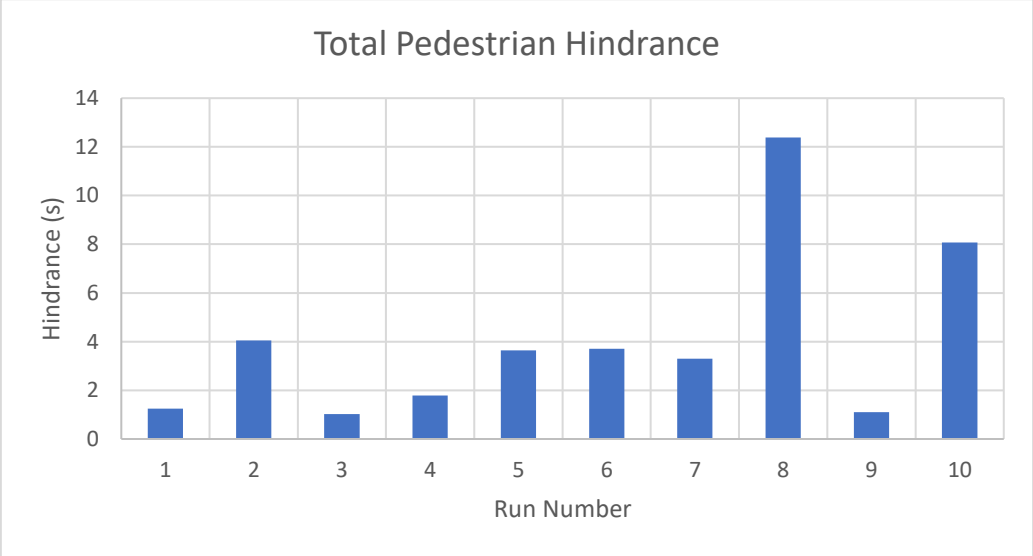

Figure 13: Total Pedestrian Hindrance per run

The third and biggest focal point of the validation is the alighting passengers. In this validation, only the alighting passengers are considered. When passengers board or when those actions are combined the number of passengers turned out to be random, which is why only alighting is taken into account. In the validation of these passengers, the function used is statistically compared with groups of passengers that were recorded in the validation dataset.

During the observation for the validation dataset, 20 data points were recorded for groups of alighting passengers. These points are plotted together with the calibration dataset and trendline in Figure 14 below. From eyesight, the data points are close to the trendline and within the range of the calibrated dataset, except for one small outlier. Unfortunately, it was not a very busy day when

the validation dataset was observed, so 11 is the maximum amount of passengers observed. To statistically come to a conclusion the R^2 or coefficient of determination is calculated between the validation data and the calibrated function. This is a statistical measure that determines how well data fits a regression model. In other words, it is a goodness-of-fit test (CFI Team, 2022). For the calibration dataset, the R^2 was 0.88, and for the validation dataset this is 0.94. This is a higher R^2, and thus the conclusion can be drawn that the calibrated function is a better fit than thought.

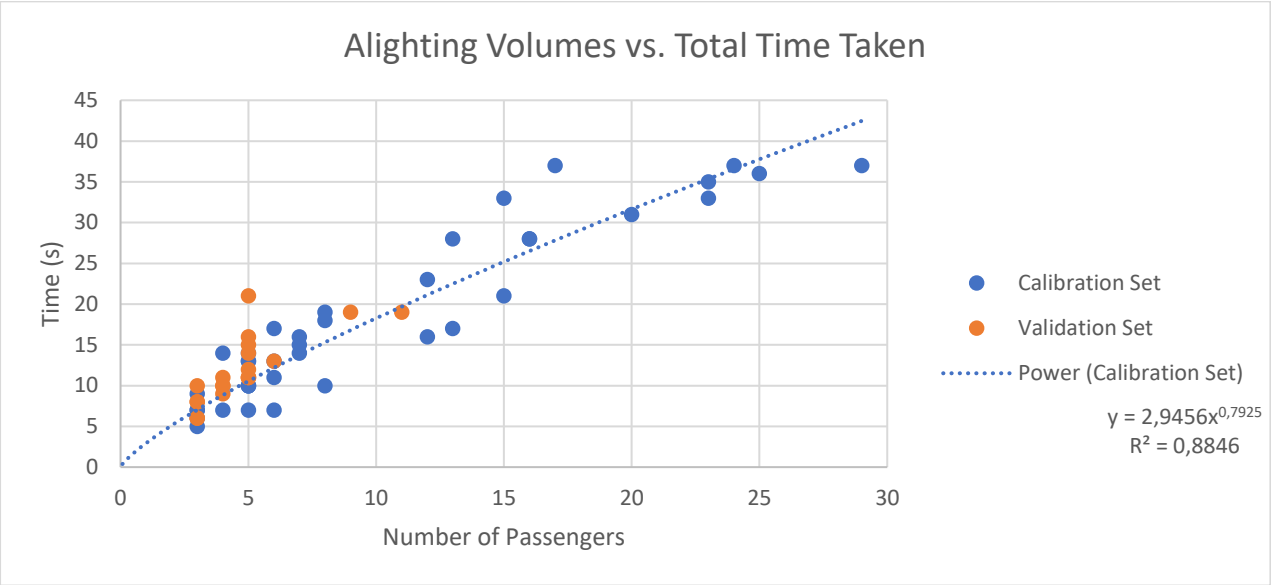

Figure 14: Validation of alighting function

5. Discussion

With the validation of the model, the project is complete. It would be awesome if every project went perfectly and gave the exact results you hoped for. Unfortunately this does not happen a lot or at least not without its difficulties. This project had some of those difficulties too.

The observation study went quite well. The first observation at Arnhem had some practical struggles of where to position myself, but after those struggles that happen when you do something for the first time, it went swimmingly. One technical issue I found out when watching the recordings was that the first 30min of footage from my calibration observations at Deventer were unusable, because the camera did not focus itself correctly. Luckily I thought beforehand that something like this might occur, and stopped and started the recording multiple times during the observation, preventing the entire recording from being unusable.

I really like modelling, programming and playing with Vissim. Sometimes this tended to take the better of me in this project. In some stages, when working on the theoretical background, I already worked on the coding of it as well. Some parts of the theoretical foundation changed and developed throughout the project. This meant some of the bits of code already written had to be changed or completely rewritten, using up time I did not really have and making the theory and coding more of a parallel process, rather than a sequential one. A good preparation can really save time!

During the project, the reliability of the GOVI data has been researched and has come to good conclusions. However, no time was available in this project to implement a solution in the model. Compared to the boarding, alighting and pedestrian routing, implementing a solution for the unreliability of GOVI is a complete different avenue of implementation. Instead of the passengers, this solution deals with the buses and dwell times themselves. Time is unfortunately always a constraint to be kept in mind.

The programming itself went pretty well. I already had some experience in both thinking as a modeler and coding in Python, which really helped a lot. I was quickly able to understand both the code that my supervisor Geert-Jan had written, and the way Vissim implements scripting with the COM interface. This made the coding really enjoyable, gradually making good results. It was also fun to show Geert-Jan things about Vissim that he did not know yet himself, giving me the opportunity to learn him things as well.

Some of the conclusions of the study were also not what was initially thought or hoped. Especially during the fitting of the functions, it was not hoped that determining the amount of passengers boarding and alighting when boarding is involved was impossible to do. The solution put forward with the exponential distributions is a great solution for this project and works fine in the simulation, but can benefit from more research or additional data sources, such as data about passenger numbers for a line.

6. Conclusion & Recommendations

The goal of this project was to investigate the reliability of GOVI data, come up with a theoretical basis for the processes of boarding and alighting, and implement this theory in SimBus Pro. From this, conclusions can be drawn about the validation and implementation of data and code. In this project, the following research questions were answered:

What relation fits between the observed stop times and number of passengers boarding/alighting?

From the observation study, a few different relations have been fitted for the number of passengers. for alighting, a function could be fitted as relation. For boarding and the combination of boarding and alighting, a function could not be fitted. Instead, two exponential distributions have been fitted, and the functions are used as theoretical maximum for the number of passengers.

How does GOVI data compare to data from observations?

From the validation, it was shown that the data is not as reliable as perhaps thought. Between observed and recorded times quite a gap can be found due to the precision of the used instruments in measuring the times. Geographically, correlations can be found between the geographic location of a bus at the moment it is recorded as departed.

How will the boarding and alighting of passengers be implemented in SimBus Pro?

By answering the sub-questions, the main question will be answered. All sub-questions describe a part of the main question, forming a complete answer in the end.

How can the calculation of stop times be correctly implemented?

The dwell times themselves were already implemented correctly. To accommodate the calculations for passengers, the determination of these times was moved to another point in the network.

How can the relation between stop times and passenger counts be correctly implemented?

A few solutions were used for the implementation of this. For alighting passengers, the calculations are done during the simulation, when a bus approaches a decision point. For boarding, the calculation of time intervals is done in the preparation of the network and the calculation of the number of passengers in the initialization of the simulation run.

How can Vissim correctly visualize the processes?

Luckily, Vissim already does this by itself. When the network and models are defined correctly, Vissim executes the visualization itself. In the 3D visualization, a few changes are made to make it look good and realistic, but other than that, it is all done by Vissim.

How are other factors taken into account and modelled?

In the modelling, the only other factor that has been modelled is the routing of pedestrians in the network. Using a holding area and code that iterates over and checks every pedestrian, pedestrians can be routed correctly to their buses, the exit, or other platforms.

Next to conclusions, a few recommendations can be made as well:

For the number of boarding passengers and alighting passengers when combined with boarding, exponential distributions were brought forward as solution. More effort could be put, however, in replacing these with better relations. This can be done by putting more research into analyzing the data, but this can also be done with better data sources about passenger numbers, such as OV-Chipkaart data.

For the GOVI data, a correction has to be made to get more realistic dwell times. This could not be implemented into the model yet, but can definitely be done. A good recommendation is to talk at least once to each major transport company about the type of system they use. From this information, a realistic correction can be made.

In the model, some improvements can still be made. Performance-wise, it can be investigated whether the code can optimized to have both less simulation time and increased reliability, if possible. Some of the issues that are known can be investigated for a possible solution to fix them.

7. Bibliography

CFI Team. (2022, May 7). *R-Squared*. From Corporate Finance Institute: https://corporatefinanceinstitute.com/resources/knowledge/other/r-squared/

Eddleman, S. W. (2016, September). *How to Plan an Engineering Design Challenge.* From Carolina: https://www.carolina.com/teacher-resources/Interactive/how-to-plan-an-engineering-design-challenge/tr39004.tr

Glen, S. (2022). *Chi-Square Statistic: How to calculate It / Distribution*. From StatisticsHowTo.com: https://www.statisticshowto.com/probability-and-statistics/chi-square/

InTraffic. (n.d.). *Grenzeloze Openbaar Vervoer Informatie (GOVI)*. From InTraffic: https://www.intraffic.nl/case/grenzeloze-openbaar-vervoer-informatie/

RET. (2021, October). *Bus*. From RET: https://corporate.ret.nl/over-ret/materieel/bus

Wand, Y., & Weber, R. (2002). Research Commentary: Information Systems and Conceptual Modeling - A Research Agenda. *Information Systems Research*, 363-376.

Appendix A – Observed buses

In this appendix, a list with all observed buses is kept. It would be too much to make figures of all other files that were used for and with the observation, function fitting and making of the timetable. For reproducibility of the results in the GOVI data, all observed buses on Arnhem Centraal and the first observation trip to Deventer are noted in the table below, sorted by line. The buses used only for alighting and boarding data points are not noted, because their arrival and departure times were not used or noted. The observation at Arnhem Centraal was done on the 25th of April 2022 and done at Deventer on the 29th of April 2022.

Table 4: Observed departed buses on Arnhem Centraal

Line nr.	Destination	Departure time		Line nr.	Destination	Departure time
1	Velp	08:10:00		43	Apeldoorn via Dieren	09:49:00
1	Velp	08:25:00		43	Apeldoorn via Dieren	10:19:00
1	Velp	08:40:00		51	Wageningen	08:14:00
1	Velp	08:55:00		51	Wageningen	08:46:00
1	Velp	09:10:00		51	Wageningen	09:17:00
1	Velp	09:25:00		56	Heteren	08:20:00
3	Het Duifje	08:23:00		56	Heteren	08:50:00
3	Het Duifje	08:26:00		56	Heteren	09:20:00
3	Het Duifje	08:38:00		60	Tolkamer	08:19:00
3	Het Duifje	08:42:00		60	Tolkamer	09:22:00
3	Het Duifje	08:53:00		62	Duiven	08:35:00
3	Het Duifje	08:57:00		62	Duiven	09:05:00
3	Het Duifje	09:08:00		62	Duiven	09:35:00
3	Burger's Zoo	08:26:00		105	Barneveld	09:56:00
3	Burger's Zoo	08:38:00		231	Apeldoorn via De Maten	09:47:00
3	Burger's Zoo	08:42:00		231	Apeldoorn via De Maten	10:17:00
3	Burger's Zoo	08:53:00		231	Apeldoorn via De Maten	10:47:00
3	Burger's Zoo	08:57:00		300	Nijmegen via Bemmel	10:04:00
3	Burger's Zoo	09:08:00		331	Velp Zuid	08:13:00
8	Arnhem Dennenweg	08:20:00		331	Velp Zuid	08:28:00
9	Schaarsbergen IPC	08:20:00		331	Velp Zuid	08:43:00
9	Schaarsbergen IPC	08:50:00		331	Velp Zuid	09:13:00
9	Schaarsbergen IPC	09:20:00		331	Velp Zuid	09:28:00
10	Papendal	09:44:00		331	Nijmegen	10:00:00
10	Papendal	10:14:00		352	Wageningen	08:14:00
11	Station Zuid	09:46:00		352	Wageningen	08:29:00
11	Station Zuid	10:16:00		352	Wageningen	08:46:00
11	Station Zuid	10:46:00		352	Wageningen	09:03:00
12	IJsseloord 2	08:30:00		352	Wageningen	09:18:00
12	IJsseloord 2	09:00:00		352	Wageningen	09:33:00
12	IJsseloord 2	09:30:00		352	Wageningen	10:03:00
14	Nijmegen Brakkenstein	10:02:00		352	Wageningen	10:18:00
26	Dieren	10:07:00				
27	Doetinchem	10:38:00				
33	Nijmegen CS	09:44:00				
33	Nijmegen CS	10:14:00				
33	Nijmegen CS	10:44:00				

Table 5: Observed departed buses on Deventer

Line nr.	Destination	Departure time		Line nr.	Destination	Departure time
1	De Vijfhoek via Blauwenoord	14:35:00		**6**	Colmschate	15:18:00
1	De Vijfhoek via Blauwenoord	15:05:00		**6**	Colmschate	15:48:00
1	De Vijfhoek via Blauwenoord	15:28:00		**6**	Colmschate	16:18:00
1	De Vijfhoek via Blauwenoord	15:43:00		**6**	Colmschate	16:48:00
1	De Vijfhoek via Blauwenoord	15:58:00		7	Kloosterlanden	14:39:00
1	De Vijfhoek via Blauwenoord	16:13:00		7	Kloosterlanden	15:09:00
1	De Vijfhoek via Blauwenoord	16:28:00		7	Kloosterlanden	16:09:00
1	De Vijfhoek via Blauwenoord	16:43:00		7	Kloosterlanden	16:39:00
2	Platvoet via Keizerslanden	14:21:05		8	Bedrijvenpark A1	14:33:00
2	Platvoet via Keizerslanden	14:53:00		8	Bedrijvenpark A1	15:03:00
2	Platvoet via Keizerslanden	15:10:24		8	Bedrijvenpark A1	15:33:00
2	Platvoet via Keizerslanden	15:44:43		8	Bedrijvenpark A1	16:03:00
2	Platvoet via Keizerslanden	16:11:49		8	Bedrijvenpark A1	16:33:00
2	Platvoet via Keizerslanden	16:43:06		57	Laren	14:21:00
3	Platvoet via Zandweerd	14:35:00		57	Laren	15:21:00
3	Platvoet via Zandweerd	15:05:00		57	Laren	16:21:00
3	Platvoet via Zandweerd	15:35:00		81	Zutphen	14:55:00
3	Platvoet via Zandweerd	16:05:00		81	Zutphen	15:55:00
3	Platvoet via Zandweerd	16:35:00		81	Zutphen	16:55:00
4	Schalkhaar	14:35:00		160	Bathmen	14:20:00
4	Schalkhaar	15:05:00		160	Bathmen	14:50:00
4	Schalkhaar	15:35:00		160	Bathmen	15:20:00
4	Schalkhaar	16:05:00		160	Bathmen	16:20:00
4	Schalkhaar	16:35:00		160	Bathmen	16:50:00
5	De Vijfhoek via Ziekenhuis	14:20:00		161	Zwolle Hessenpoort	14:42:00
5	De Vijfhoek via Ziekenhuis	14:50:00		161	Zwolle Hessenpoort	15:42:00
5	De Vijfhoek via Ziekenhuis	15:20:00		161	Zwolle Hessenpoort	16:42:00
5	De Vijfhoek via Ziekenhuis	15:35:00		165	Raalte	14:39:00
5	De Vijfhoek via Ziekenhuis	15:50:00		165	Raalte	15:09:00
5	De Vijfhoek via Ziekenhuis	16:05:00		165	Raalte	15:39:00
5	De Vijfhoek via Ziekenhuis	16:20:00		165	Raalte	16:09:00
5	De Vijfhoek via Ziekenhuis	16:35:00		165	Raalte	16:39:00
5	De Vijfhoek via Ziekenhuis	16:50:00				
6	Colmschate	14:18:00				
6	Colmschate	14:48:00				

Appendix B – Graphs for validation of GOVI data

In this appendix, all graphs for the validation of the GOVI data are shown. Firstly, the time-based analysis is shown. Afterwards, the geography-based solution is shown.

For Deventer, the differences in departure and arrival times are shown below:

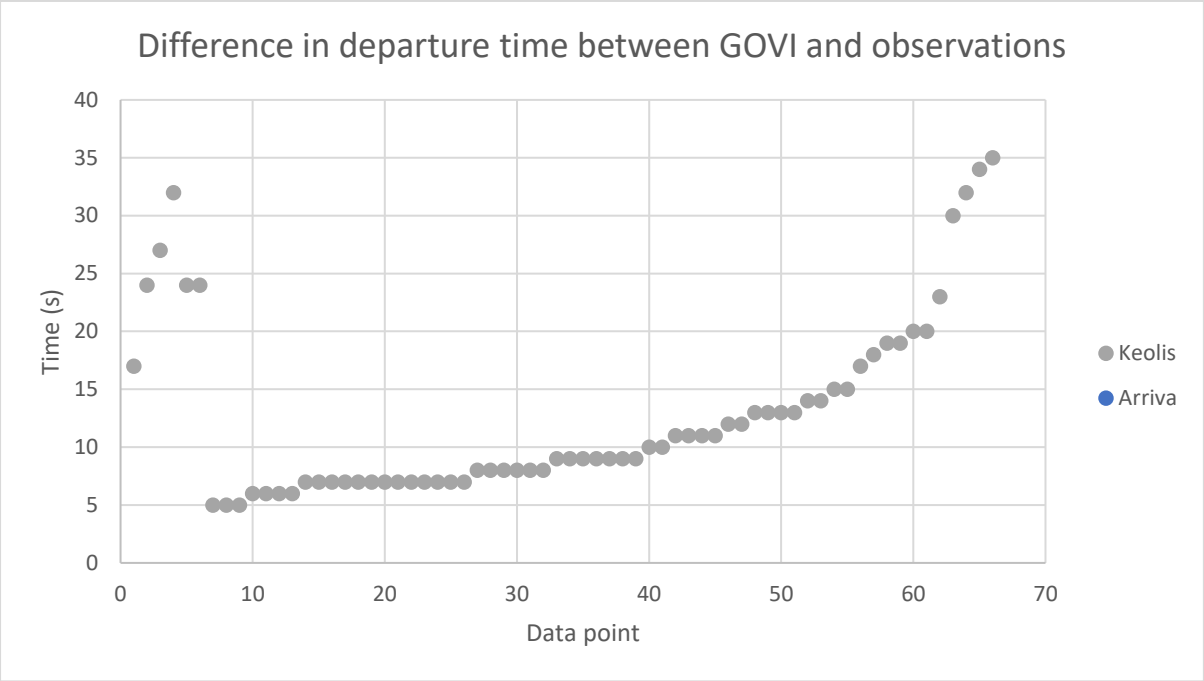

Figure 15: Difference in departure time between GOVI and observations

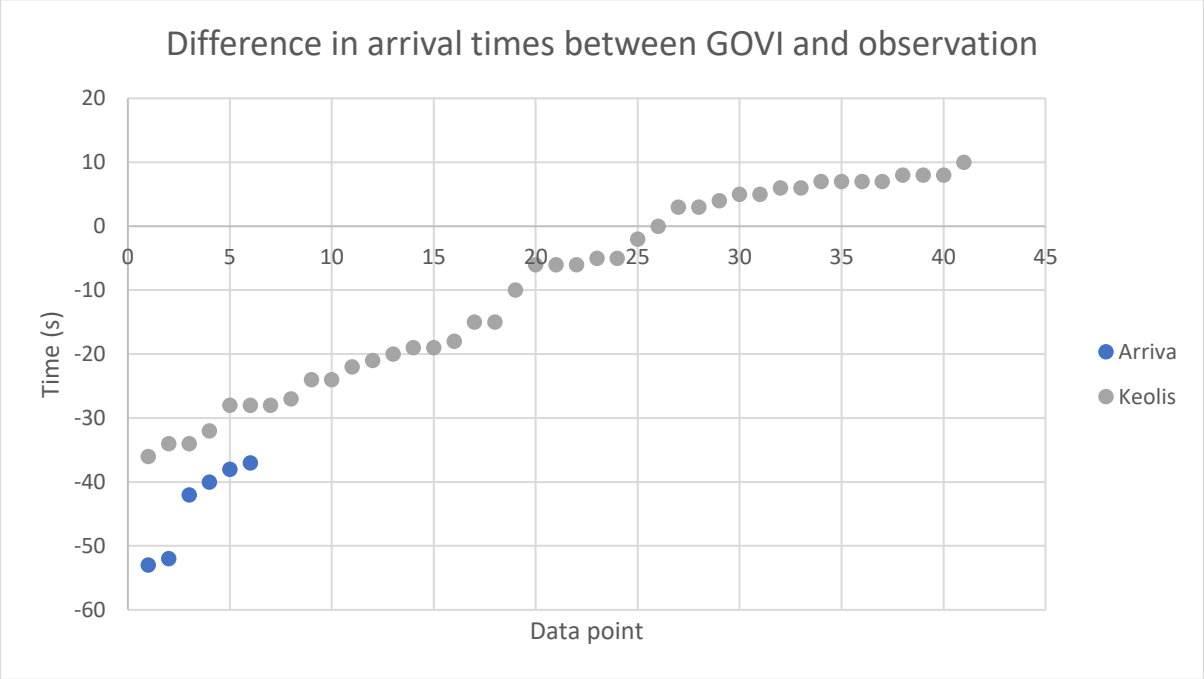

Figure 16: Difference in arrival times between GOVI and observation

For Arnhem, the differences in departure and arrival times are shown below. On top of all differences in departure time, only the stops mentioned in the GOVI data as 'INTERMEDIATE' are used as well, to see whether they are more reliable. As can be seen by the yellow points of the trolley buses, they are not.

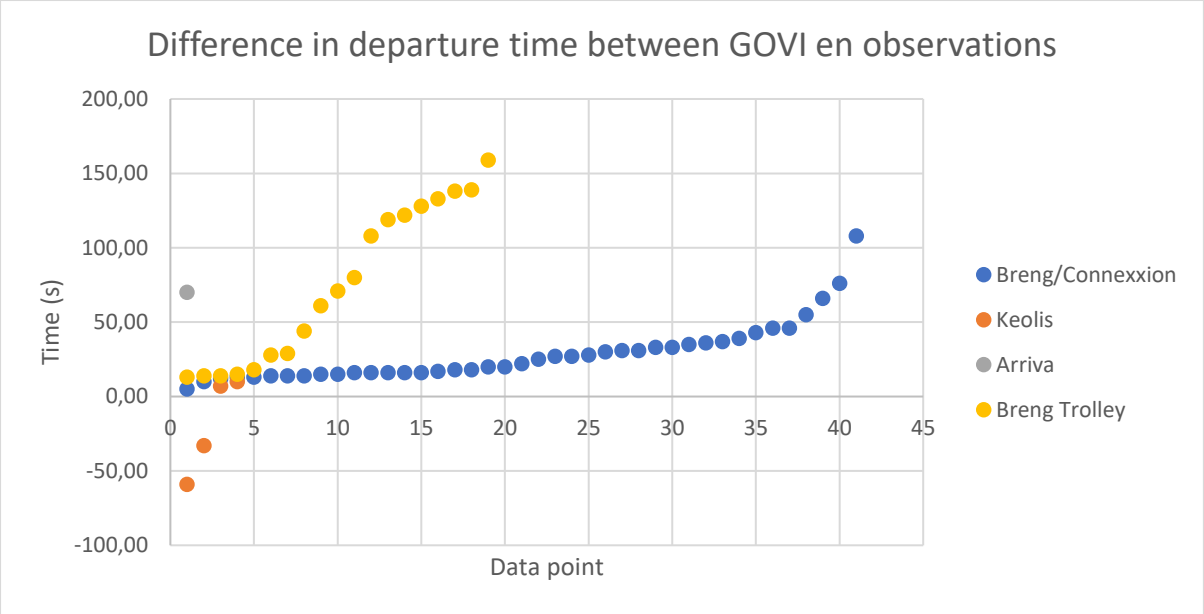

Figure 17: Difference in departure time between GOVI en observations

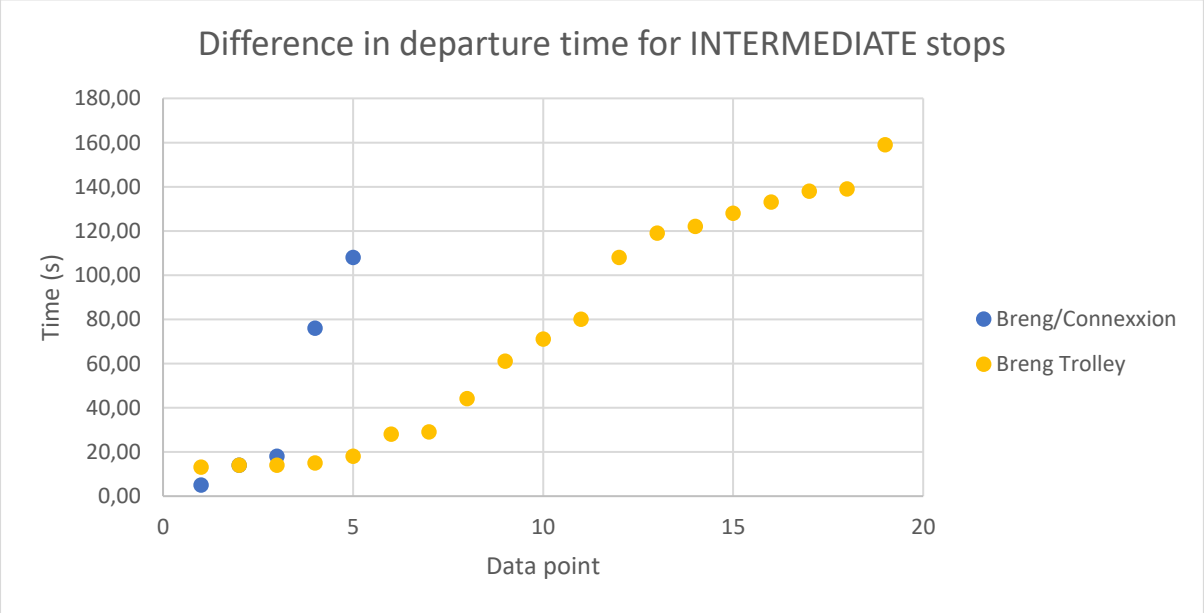

Figure 18: Difference in departure time for INTERMEDIATE stops

In the following graph, a lot of the massive differences in time can be explained due to the fact that buses can buffer before arriving at the platform for boarding passengers. This can create those massive differences, starting at more than 3 minutes.

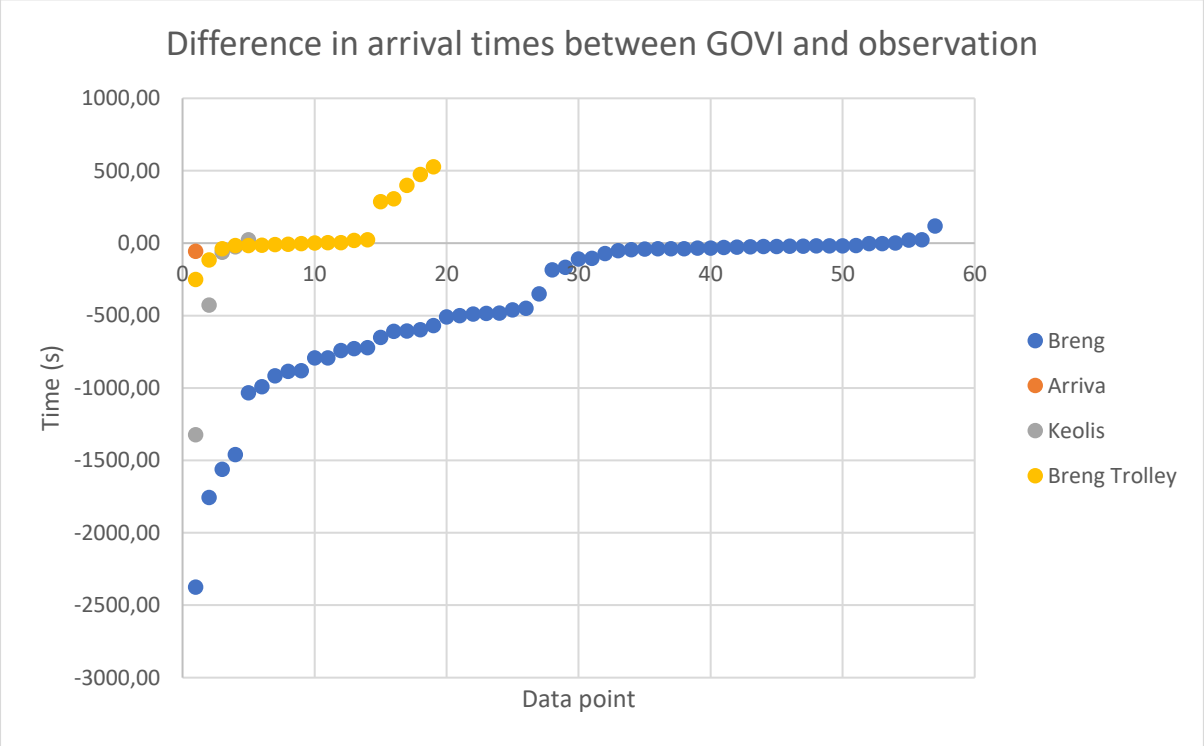

Figure 19: Difference in arrival times between GOVI and observation

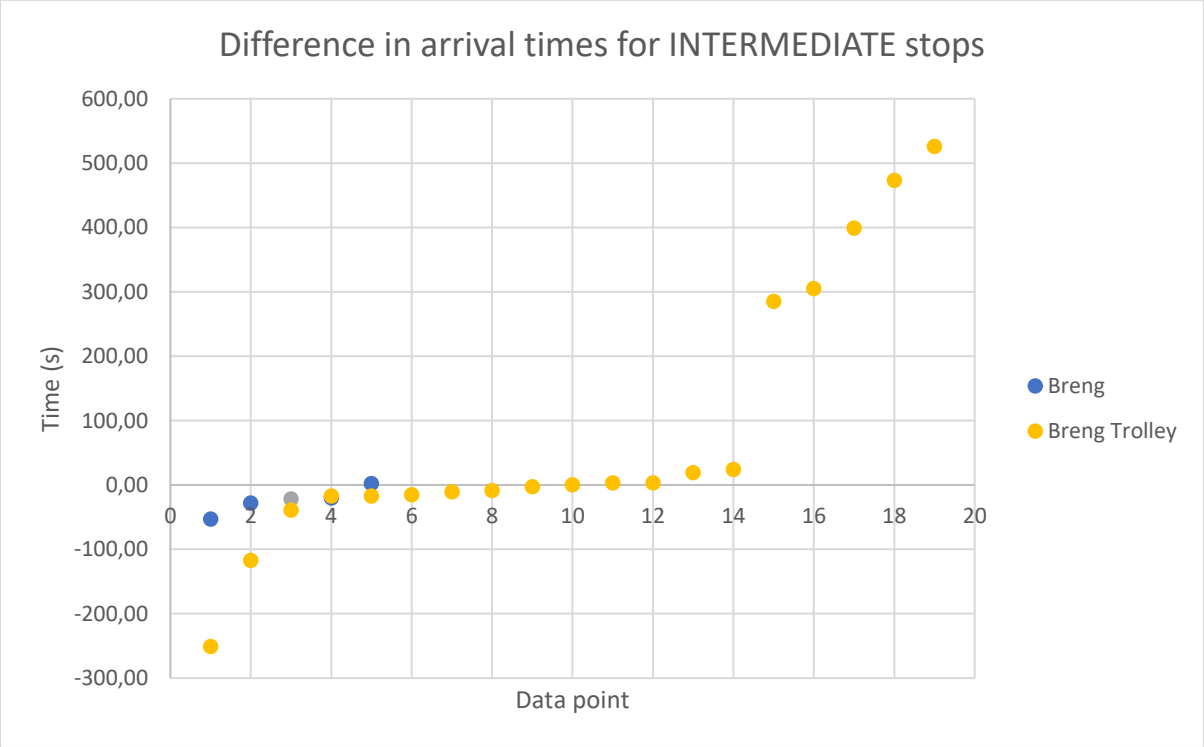

Figure 20: Difference in arrival times for INTERMEDIATE stops

An interesting consequence is to see what the actual impact is on the dwell times. When comparing the observed and recorded dwell times, the following graph is the result:

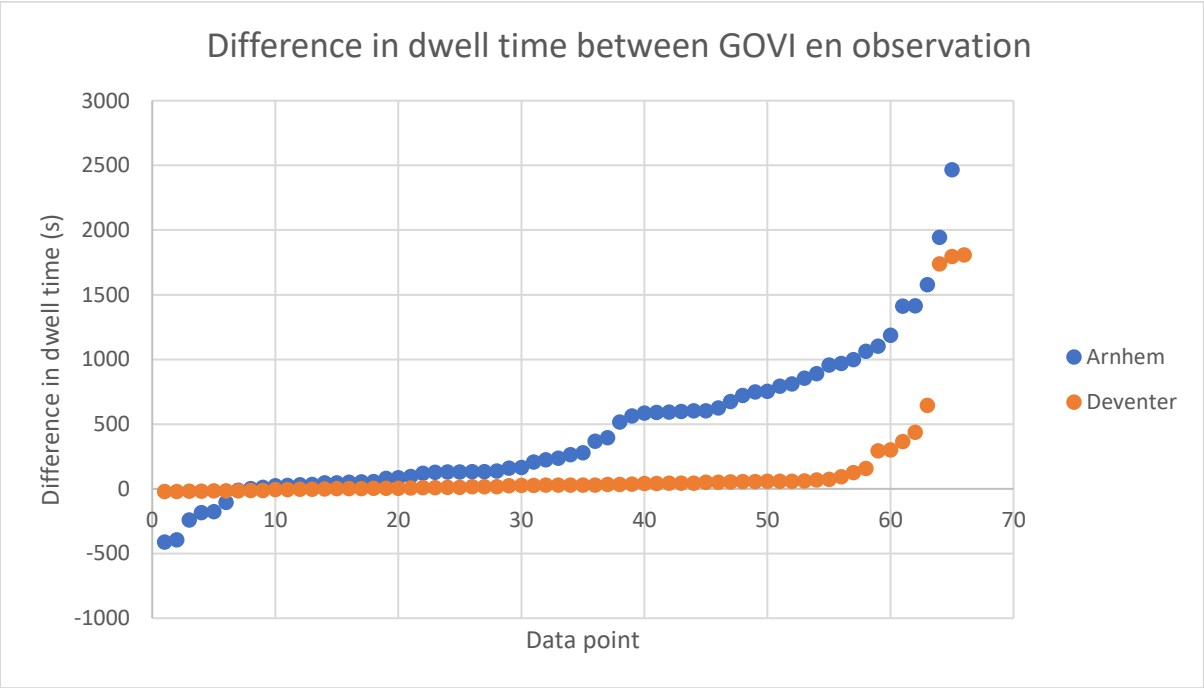

Figure 21: Difference in dwell time between GOVI en observation

A lot of the major differences of more than 2 minutes are because those buses have buffered at the station and thus arrived a lot earlier. To give a better idea of the smaller scale differences the graph down below zooms in on Deventer where the differences of more than 2 minutes have been cut away:

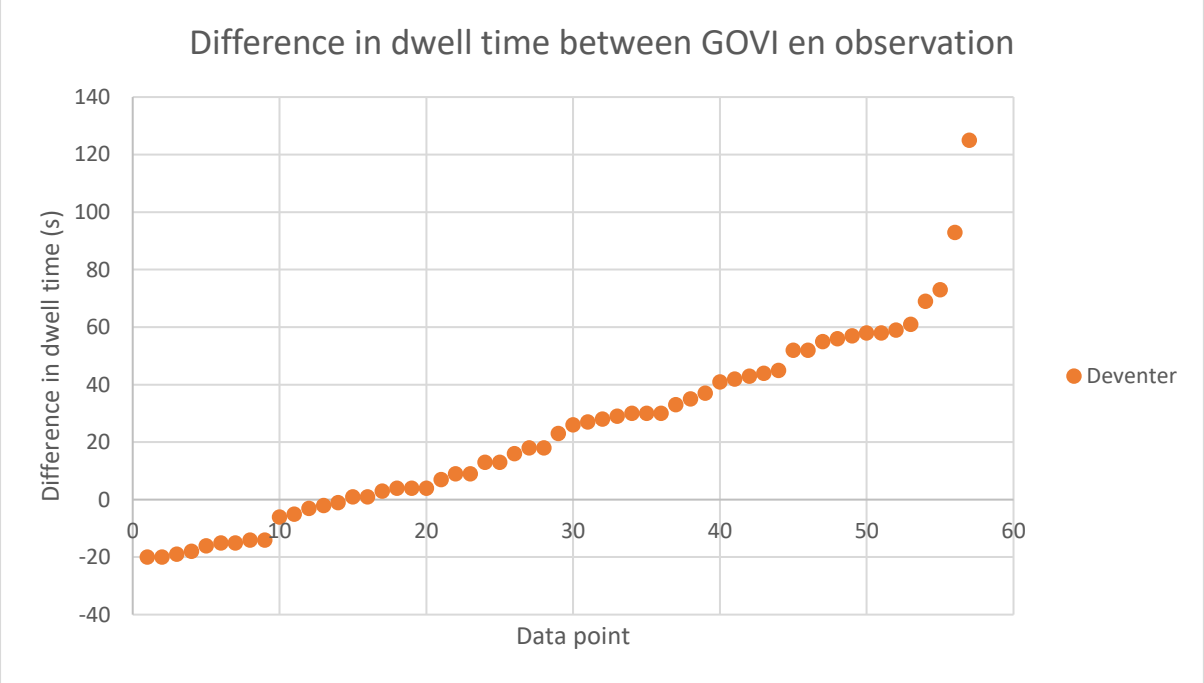

Figure 22: Difference in dwell time between GOVI en observation

Next to the time-based analysis, a geographic analysis has been conducted as well. For 3 platforms in Deventer, serviced by both Keolis and Arriva, this analysis has been executed. For platform A, the results are shown below. Every colored point represents a unique bus. The location of the point on the platform itself indicates whether a bus departed from the front or back. A correlation can be seen: all buses are recorded as departed when driving on the Stationsstraat. Whether a bus departs from the front or back of the platform does not matter.

Figure 23: Geographic analysis of platform A (Keolis)

The same correlation is found when analyzing platform F. Again, just past the platform, whether a bus may be driving forwards or backwards, the bus is recorded as departed.

Figure 24: Geographic analysis of platform F (Keolis)

When analyzing platform E, which is serviced by Arriva, a different correlation can be found. This is the same correlation as found in the time-based analysis, where Arriva buses arrive earlier than in reality and depart later than in reality in the data. The recordings are centered around the same geographic location however. The arrow indicates whether a bus was arriving or departing.

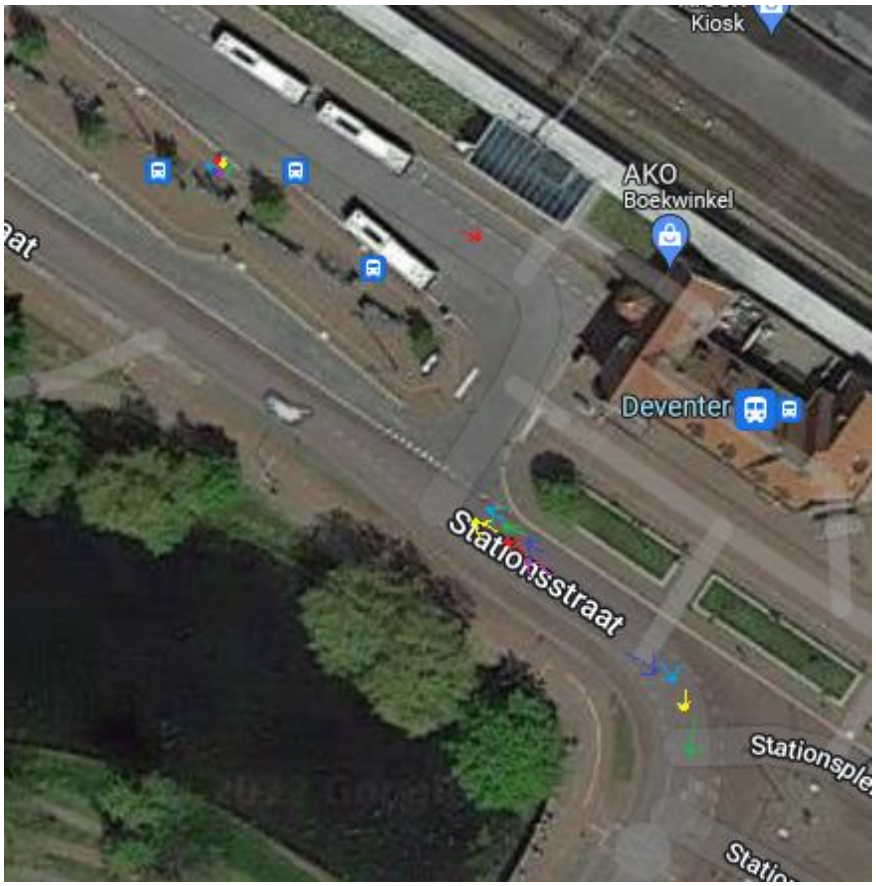

Figure 25: Geographic analysis of platform E (Arriva)

Appendix C – Function and Distribution Fitting

This appendix delves deeper into the fitting of the functions and distributions used in the theoretical foundation. The used methods and reasoning will be explained in more detail.

After the execution of the observation study, the results were digitized and further analyzed. From the boarding and alighting passengers, the total time taken and average time per passenger can be observed and calculated. For both boarding and alighting, this was done per group of passengers. Groups of less than 3 people were not considered big enough for this. When using 1 or 2 people to determine the necessary time, there are two consequences: without counting more precise than per second, the data is not precise enough to measure per passenger, and a single passenger has a lot more variability and uncertainty in time taken. It is also a lot more work to include all of those groups, because those numbers occur more often. Below in Figure 26 and Figure 27 are plots showing the total time taken plotted against the group size. In total 32 data points are recorded for boarding and 39 for alighting.

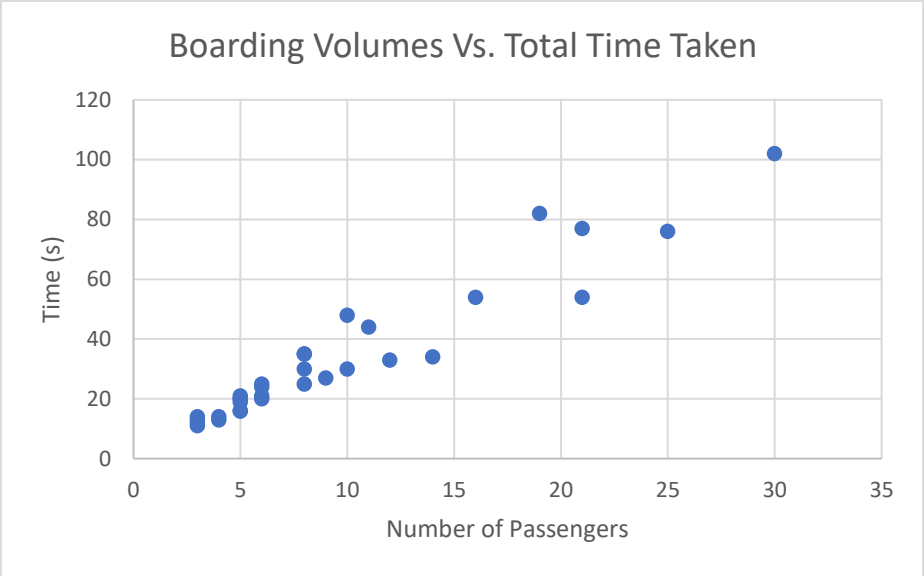

Figure 26: Boarding Volumes Vs. Total Time Taken

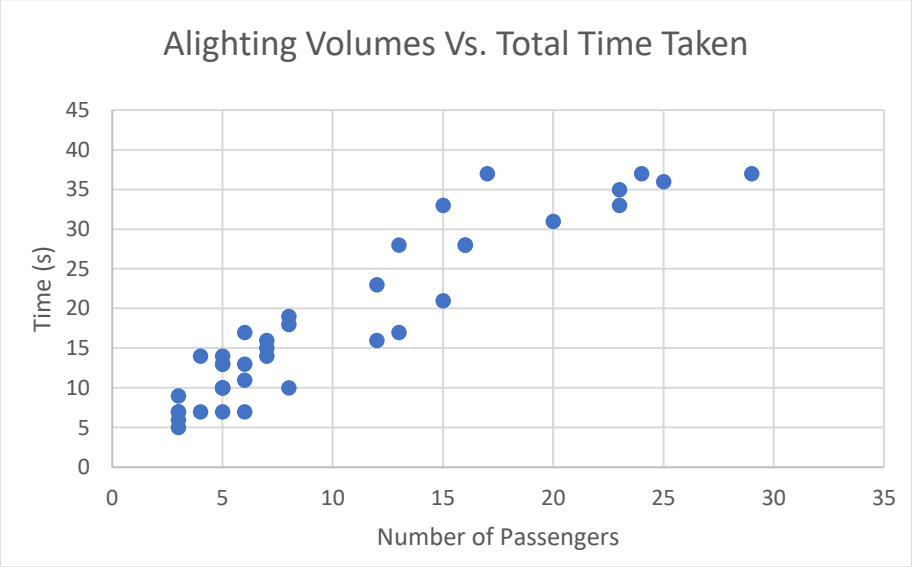

Figure 27: Alighting Volumes Vs. Total Time Taken

Both graphs show a correlation between the volume of passengers and the total time it takes a group of passengers to either board or alight. Looking at the graph, the boarding passengers show a quite linear trend but for alighting this is a little different. Larger groups of people take relatively less time per passenger to alight and thus have a smaller total alighting time. This may be due to the fact that large groups can keep the card scanners in the bus constantly busy and constantly have people coming through the doors and are thus much more efficient in alighting.

The step that follows from the observations is turning this into an input for the model. For alighting only, it is assumed that a bus will let passengers get off the bus and immediately afterwards it continues to its next action. This means fitting a function through the observed data points is the course of action, serving as basis for a function in SimBus Pro.

The best fitting function for alighting was a 2nd degree function with an R^2 of 0.89, but this function is not applicable in this situation as a parabola has an extreme value at the top and will descend afterwards. Using this type of function is not logical. The next best function is a power function, with an R^2 of 0.88 rounded off. Next to the good fit to the data, a big advantage of the power function is that it does not flatten quickly. It takes a very long time before it does so, and thus prevents a large difference in passenger numbers for only a small difference in time. It does this while keeping to the trend of the data, which is the slow flattening of the curve. The function fitted to the data is shown below:

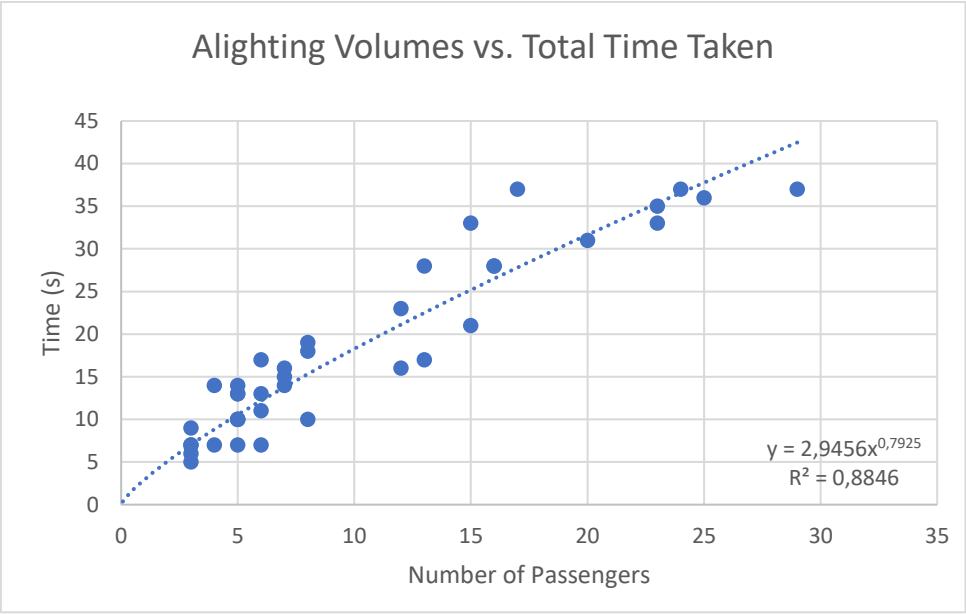

Figure 28: Function fitting for alighting

This means the function will be:

$$t = 2.95n^{0.79}$$

Where *t* is the time in seconds and *n* the number of passengers. The next step is to rewrite the function to calculate the number of passengers from the dwell time instead of the other way around, meaning the function has to be written in the form *n* = … . This means we get the following function:

Equation 5: Function for alighting

$$n = (\frac{t}{2.95})^{\frac{1}{0.79}}$$

Which is the definitive function for calculating the number of passengers from the dwell time.

For boarding passengers, a different approach is made. Ideally, this would be done by finding a relationship between the total dwell time and the total number of passengers boarding. In Figure 29 below these two are plotted against each other, plotted per station. It can be seen however, that there is no relationship at all to be discovered between the total dwell time and the number of passengers. There is a minimum or baseline to be discovered because passengers still need to board the bus. The baseline is based on the smallest gradient from (0,0) to a point in the figure. Apart from a minimum boarding time that logically increases when passenger numbers increase, these two variables are completely independent. Due to the design of the stations and timetables, the bus stations all behave a little different in this aspect. On Arnhem Centraal, buses wait in buffer spots before going to the platform more often than not, and on Deventer buses spent a lot of time standing on the platform instead of buffering. This difference is clearly found in the graph. From this however, it can be concluded that using this data no relationship can be established between the dwell time for boarding and the number of passengers boarding. As a conclusion, the number of passengers boarding cannot be determined reliably at all and is observed to be random.

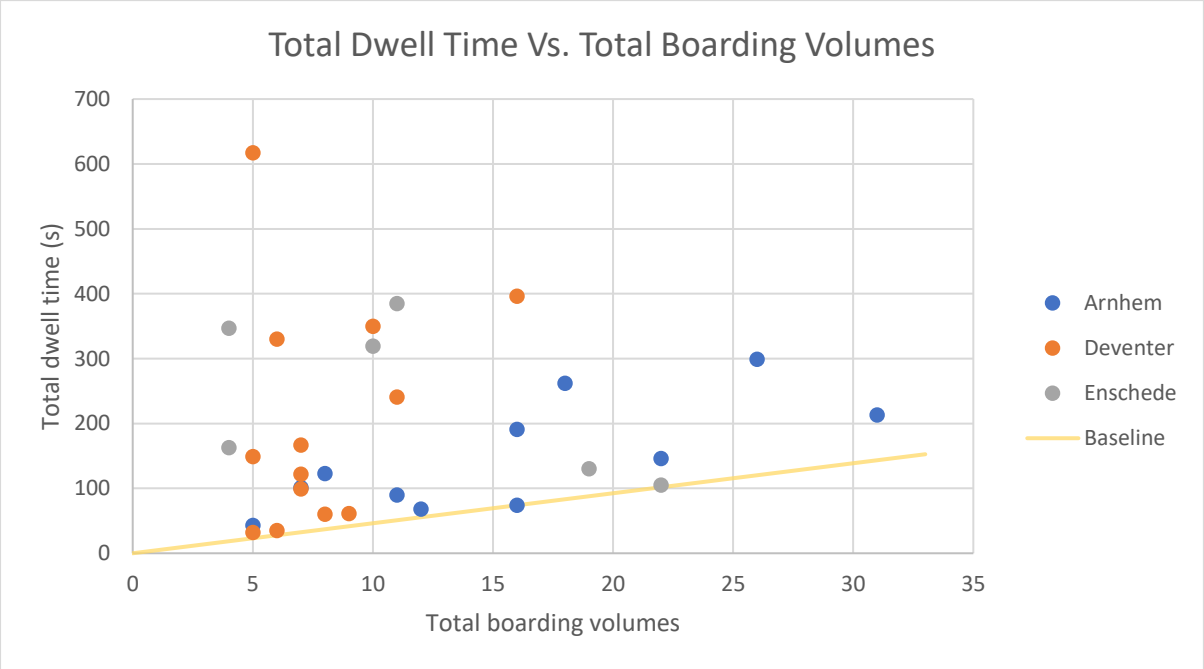

Figure 29: Total Dwell Time Vs. Total Boarding Volumes

SimBus Pro tries to reflect reality as good as possible, and one of the requirements for this project is to have people boarding buses in the model. However, the number of passengers cannot be determined reliably with a mathematical function. For this reason, the aim will be to program a simple function that forms a basis for further improvement and extension, is visually realistic, and performs well. Although there is no relation between total dwell time and total number of passengers, there is a relation between the time it takes for passengers to board and the number of them. This relation is shown below:

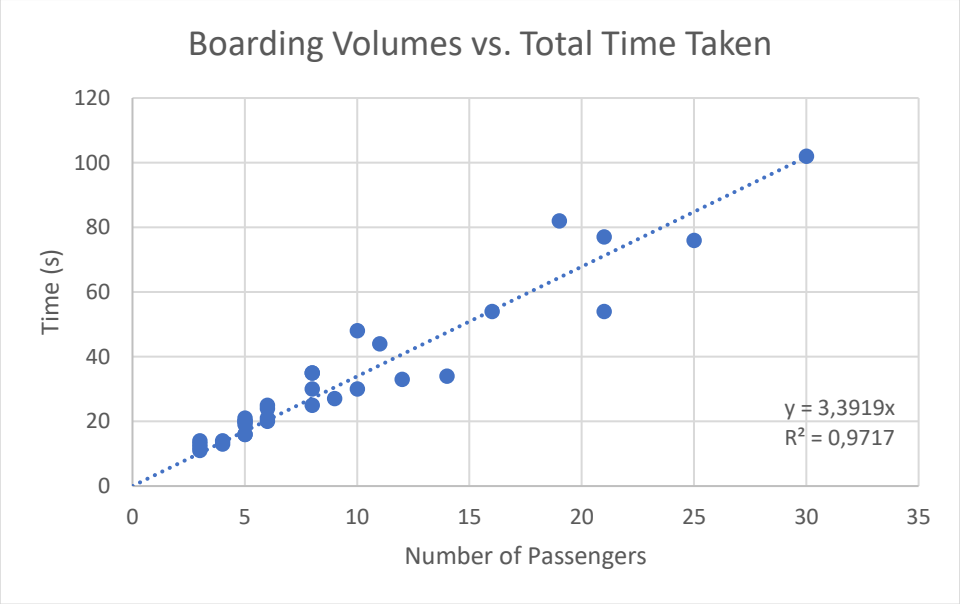

Figure 30: Function fitting for boarding

It is actually a strong relation with an R^2 of 0.97. This function will be used to determine the maximum number of passengers that can board within the dwell time and when rewritten to the form *n* = … gives the following function:

Equation 6: Function for boarding capacity

$$n = \frac{t}{3.39}$$

Where *t* is the time in seconds and *n* the number of passengers. However, not 100% of the time a bus is standing still people are boarding. This means Equation 6 is the theoretical maximum amount of passengers that can board. Because the amount of passengers boarding is observed to be random, a solution is sought in the area of distributions.

When grouping the frequency of each group size in bins of size 4 and plotting this in a histogram, there is a correlation to be seen. The frequency of the observed passenger groups seems to follow the form of an exponential distribution. The data points are plotted in Figure 31 below. This means that a distribution can be fitted from which SimBus Pro draws a number. This number is compared to the capacity of Equation 6 and used if it is lower or equal to the theoretical maximum. This gives us a number of passengers in the simulation that looks realistic and does not lengthen the determined dwell times. For this graph more data could be used. For the functions mentioned above, no group of people smaller than 3 was used. For the distribution fitting all passengers that boarded can be used. This means all groups of 0-2 passengers can be included as well.

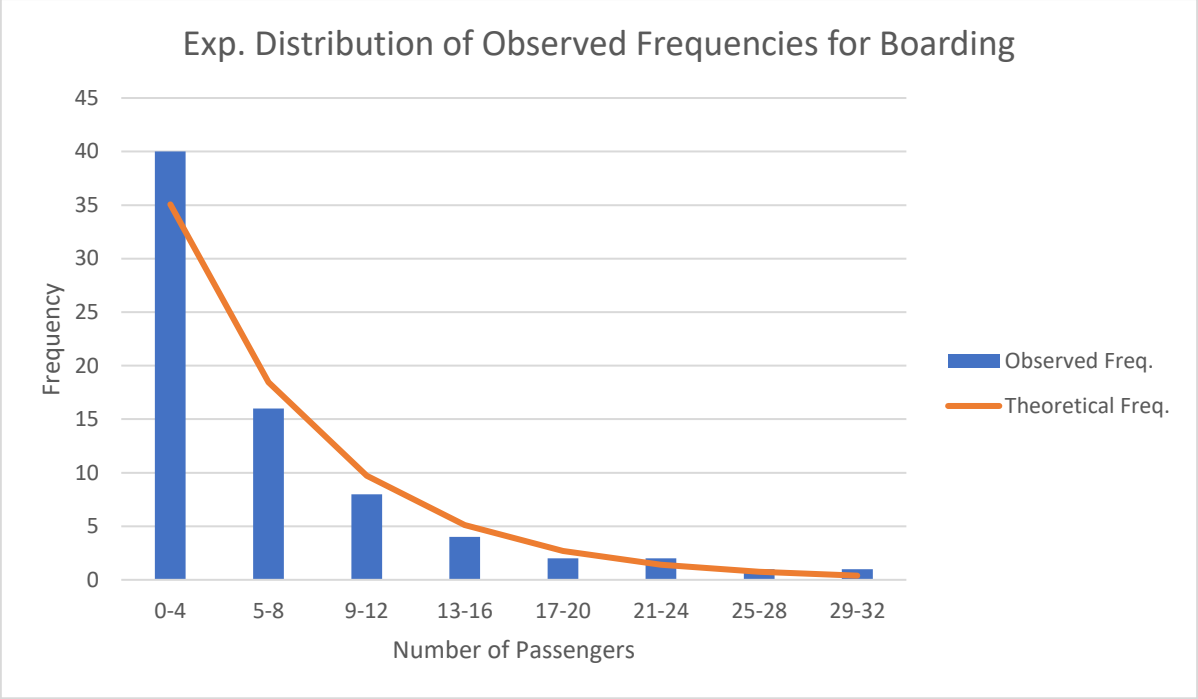

Figure 31: Exponential Distribution of Observed Frequencies for Boarding

The cumulative distribution function of an exponential distribution has the form:

$$F(x;\lambda) = 1 - e^{-\lambda x}$$

When multiplying the difference in frequency per bin of this function with the total sum of the frequencies, we get the theoretical frequency, as shown in Figure 31 above.

For comparing the observed frequencies with the calculated frequencies, the Chi square test can be used. The Chi square test can show a relationship between two variables and the chi-squared statistic tells how much difference exists between the observed counts and the theoretical counts (Glen, 2022). Using the Chi Square test and a solver for λ, the optimal λ is found to be about 0.16. The sum of the Chi Square values is 3.01 for this λ. The critical value is 14.07 for 95% confidence, and the sum of the values is lower than this. The function is thus statistically valid. The resulting distribution is:

Equation 7: Exponential Distribution for boarding

$$F(x;\lambda) = 1 - e^{-0.16x}$$

The same can be done for the combination of boarding and alighting. Not all buses do boarding and alighting purely as separate processes. Depending on station and timetable, buses can also execute 1 stop in the station for both boarding and alighting. Not all lines have to terminate at the stations in question, for example.

As with boarding, it is impossible to reliably estimate how much of the time standing still is spent on alighting passengers. Using the exact same method as with boarding we get the graph of Figure 32. Using the data for alighting, we get an exponential distribution with an optimal λ of about 0.12. the sum of the Chi Square values is 7.41 which again is lower than the critical value of 14.07 for 95% confidence. To enforce the maximum amount of passengers that can alight in the bus's dwell time the same power function (Equation 5) can be used as is used for alighting only.

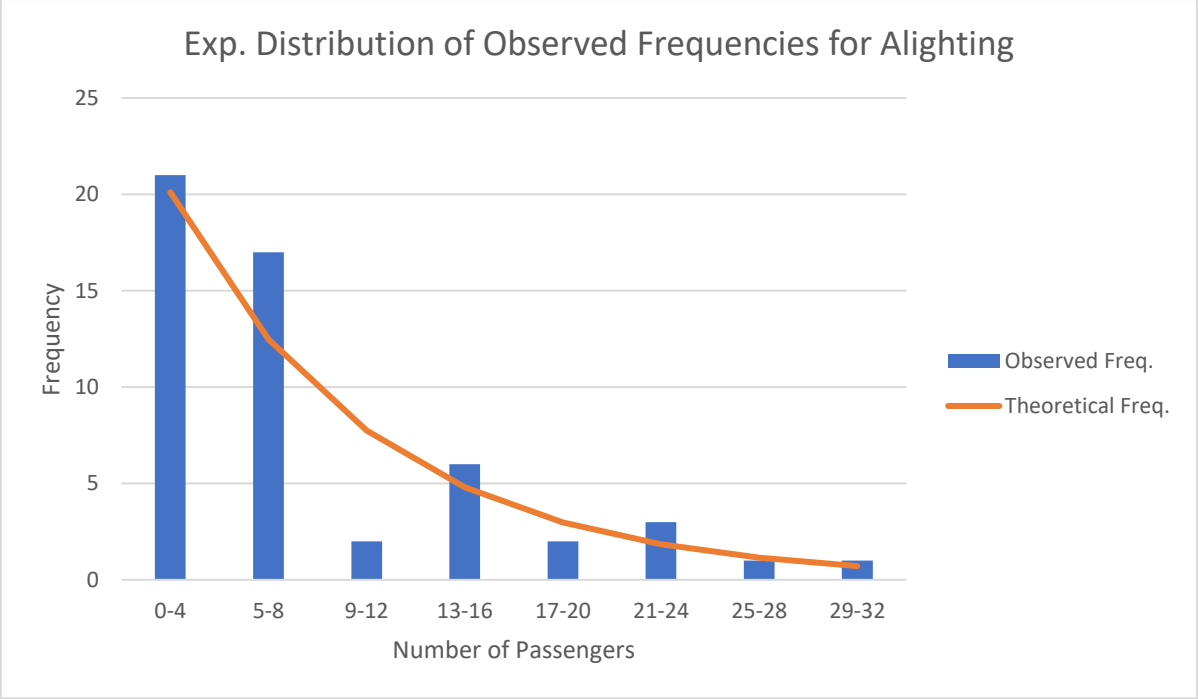

Figure 32: Exponential Distribution of Observed Frequencies for Alighting

The resulting distribution is:

Equation 8: Exponential Distribution for alighting

$$F(x;\lambda) = 1 - e^{-0.12x}$$

To conclude, the 3 actions of boarding, alighting and the combination of both two all have a theoretical function or distribution that describes the amount of passengers boarding and alighting. Table 6 below summarizes which equations and/or distributions are used for which actions.

Table 6: Summary of used functions

Action	**Used function or distribution**
Alighting	Equation 5
Boarding	Equation 6 and Equation 7
Combination of boarding and alighting	Equation 5 (alighting), Equation 6 (boarding), Equation 7 (boarding) and Equation 8 (alighting)

Appendix D – Pedestrian Routing Solution

This appendix describes the solution for pedestrian routing. This consists of 2 parts: preparing the network for all pedestrians and the code executed during the simulation. In the preparation, the intervals are set correctly and the correct pedestrian areas, inputs and routes are inserted. During the simulation, pedestrians are set on their correct routes when needed.

In the script that is executed before starting Vissim, the network is prepared for all pedestrians. This has multiple steps. Firstly, the necessary time intervals in which pedestrians need to be generated are calculated and inserted.

```
712 def make_vissim_for_peds(vissim_root, timetable: pd.DataFrame, starttime_seconds):
713     """Functie om alles wat benodigd is om passagiers te laten werken toe te voegen aan het Vissim-netwerk:
714     Hieronder vallen de tijdsintervallen en pedestrian volumes, inputs, routes en gebieden
715 
716     Args:
717         vissim_root (root): Ingelezen INPX-bestand
718         timetable (array): Array met de dienstregeling
719         starttime_seconds (int): de starttijd van de simulatie in seconden
720     """
721 
722     # Maak eerst de tijdsintervallen aan en vul ze met de juiste volumes
723     print('Voeg de benodigde intervallen toe en bepaal het aantal passagiers')
724 
725     # Bepaal de benodigde intervallen voor instappers
726     int_start = {}
727     int_end = {}
728     intervals = []
729     for i in range(len(timetable)):
730         if timetable.at[i, 'aanuit'] == 1 and timetable.at[i, 'type'] < 5: # Type 5, 6 en 7 hebben geen instap:
731             # Bepaal alle benodigde tijdsintervallen
732             spawntime_start = timetable.at[i, 'passstart'] # ... seconden voor een bus zal vertrekken beginnen met voetgangers genereren
733             spawntime_end = timetable.at[i, 'passeind'] # ... seconden voor een bus zal vertrekken stoppen met voetgangers genereren
734             start = timestring_to_seconds(timetable.at[i, 'vertrektijd']) - spawntime_start - starttime_seconds
735             end = timestring_to_seconds(timetable.at[i, 'vertrektijd']) - spawntime_end - starttime_seconds
736             int_start[timetable.at[i, 'vertrektijd']] = start
737             int_end[timetable.at[i, 'vertrektijd']] = end
738             if start not in intervals:
739                 intervals.append(start)
740             if end not in intervals:
741                 intervals.append(end)
742 
743     intervals.sort() # Ze moeten van laag naar hoog staan op een rij voor Vissim
745     # Voeg alle intervallen toe
746     timeIntervalSets_element = vissim_root.find('timeIntervalSets')
747     for timeIntervalSet in timeIntervalSets_element.findall('timeIntervalSet'):
748         if timeIntervalSet.attrib['no'] == "PEDESTRIANINPUT":
749             timeInts_element = timeIntervalSet.find('timeInts')
750             for Ints in intervals:
751                 timeInterval_element = ET.Element('timeInterval', {"start":str(Ints)})
752                 timeInts_element.append(timeInterval_element)
```

Figure 33: Calculation of intervals

The next step is to set up the all pedestrian areas, inputs, and routes. For each bus in the timetable that has boarding in its set of actions, a pedestrian area is made with corresponding pedestrian input where the pedestrians are generated. This input receives the correct time intervals previously calculated.

For the problem of deciding where passengers need to go, a holding area is created under the existing waiting areas for each stop. In this area, pedestrians are given a very large dwell time until it can be decided where they should go but are visually waiting at the bus stop. On top of this, routes need to be made from the inputs to the correct holding areas, and from the holding areas to the exit, waiting area, and all other holding rooms for switching between platforms. All stops are made compatible with passengers as well.

The entire block of code generating these areas, inputs and routes is shown below:

```
    print('Voeg de pedestrian inputs en routes toe')

    # Maak de gebieden voor haltes en routes naar de uitgangen
    areas_element = vissim_root.find('areas')
    area_number = get_no_of_last_element(vissim_root, 'areas', 'area')
    counter = 0
    edge_counter = 1
    area_dict = {}
    for area_d_elements in areas_element.findall('area'):
        # Zorg dat de platform edges ook werken op de haltes met H#_0
        if area_d_elements.attrib['name'].startswith('Platform edge'):
            for ptstop_element in vissim_root.find('ptStops').findall('ptStop'):
                if ptstop_element.attrib['name'] == "H" + str(edge_counter) + "_0":
                    ptstopRef = ptstop_element.attrib['no']
            ptStops_element = area_d_elements.find('ptStops')
            intObjectRef_element = ET.Element('intObjectRef', {'key':ptstopRef})
            ptStops_element.append(intObjectRef_element)
            edge_counter += 1

        if area_d_elements.attrib['name'].startswith('W'):
            area_number += 1
            counter += 1

            # Zorg ook dat de wachtruimtes ook werken op de haltes met H#_0
            for ptstop_element in vissim_root.find('ptStops').findall('ptStop'):
                if ptstop_element.attrib['name'] == "H" + str(counter) + "_0":
                    ptstopRef = ptstop_element.attrib['no']
                    boardVols_element = ET.Element('boardVols')
                    boardingVolume_element = ET.Element('boardingVolume',{"allPTLines":"true",
                                                                          "relVol":"1",
                                                                          "timeFrom":"0",
                                                                          "timeTo":"2147483647", # ??? Ik heb geen idee waar deze max vandaan komt (word als max bij de andere haltes gebruikt)
                                                                          "volume":"1"})
                    boardVols_element.append(boardingVolume_element)
                    ptstop_element.append(boardVols_element)
            ptStops_element = area_d_elements.find('ptStops')
            intObjectRef_element = ET.Element('intObjectRef', {'key':ptstopRef})
            ptStops_element.append(intObjectRef_element)

            # Vind de basic dwelltime distributie van 9999 sec
            for timeDistr in vissim_root.find('timeDistributions').findall('timeDistribution'):
                if timeDistr.attrib['name'] == "basic dwelltime":
                    basicdistr = timeDistr.attrib['no']

            # Maak de beslissingsgebieden aan
            area_dict[("D" + str(counter))] = str(area_number)
            area_element = ET.Element('area', {"alwUseAsDest":"false",
                                               "alwUseAsOrig":"false",
                                               "cellSize":"0.15",
                                               "desSpeedFact":"1",
                                               "displayType":"10",
                                               "isQueue":"false",
                                               "level":"1",
                                               "name":"D" + str(counter),
                                               "no":str(area_number),
                                               "obstDist":"0.5",
                                               "pedRecAct":"true",
                                               "ptUsage":"PTNONE",
                                               "ptWaitBehav":"WAITATFIXEDLOCATION",
                                               "queueApproachingDirectLineRadius":"2",
                                               "queueApproachingMethod":"STATICPOTENTIAL",
                                               "queueEvalAct":"false",
                                               "queueSpc":"1.5",
                                               "queueSpcChoice":"QUEUESPACINGCHOICEDEFAULT",
                                               "showClsfValues": "true",
                                               "showIndivPeds":"true",
                                               "thickness":"0.05",
                                               "tmDistr":str(basicdistr),
                                               "waitAreaMinSpc":"0",
                                               "waitPosApprMeth":"DIRECTLINE",
                                               "waitTmIsRelToSimStart":"false",
                                               "zOffset":"0.01"})
            geometry = area_d_elements.find('geometry')
            geometry_element = ET.Element('geometry')
            points = geometry.find('points')
            points_element = ET.Element('points')
            for point in points.findall('point'):
                x = point.attrib['x']
                y = point.attrib['y']
                point_element = ET.Element('point', {"x":x, "y":y})
                points_element.append(point_element)
```

```
            geometry_element.append(points_element)
            area_element.append(geometry_element)
            areas_element.append(area_element)

            for elements in areas_element.findall('area'):
                if elements.attrib['name'].startswith('U'):
                    stop_number = elements.attrib['no']
            make_ped_route(vissim_root, str(area_number), stop_number, ('U' + str(counter)), None) # Maak een route naar de uitgang
            make_ped_route(vissim_root, str(area_number), area_d_elements.attrib['no'], area_d_elements.attrib['name'], None) # Maak een route naar de wachtruimte

    # Maak de routes tussen haltes aan
    for i in area_dict:
        for j in area_dict:
            if i != j:
                make_ped_route(vissim_root, area_dict[i], area_dict[j], (i + j), None)

    # Maak de inputs, gebieden ervoor en routes naar de voorkeurshaltes aan
    pedestrianInputs_element = vissim_root.find('pedestrianInputs')
    pedestrianInput_number = get_no_of_last_element(vissim_root, 'pedestrianInputs', 'pedestrianInput')
    for area_elements in areas_element.findall('area'):
        if area_elements.attrib['name'] == 'U1':
            for i in range(len(timetable)):
                if timetable.at[i, 'aanuit'] == 1 and timetable.at[i, 'type'] < 5: # Type 5, 6 en 7 hebben geen instap
                    # Maak eerst het input gebied aan voor elke bus in de dienstregeling
                    area_number += 1
                    pedestrianInput_number += 1
                    timetable_name = str(timetable.at[i, 'simbuslijn']) + "_" + str(timetable.at[i, 'aankomsttijd']) # Is hetzelfde als de bus naam
                    area_element = ET.Element('area', {"alwUseAsDest":"false",
                                                       "alwUseAsOrig":"false",
                                                       "cellSize":"0.15",
                                                       "desSpeedFact":"1",
                                                       "displayType":"10",
                                                       "isQueue":"false",
                                                       "level":"1",
                                                       "name":"I_" + timetable_name,
                                                       "no":str(area_number),
                                                       "obstDist":"0.5",
                                                       "pedRecAct":"true",
                                                       "ptUsage":"PTNONE",
                                                       "ptWaitBehav":"WAITATFIXEDLOCATION",
                                                       "queueApproachingDirectLineRadius":"2",
                                                       "queueApproachingMethod":"STATICPOTENTIAL",
                                                       "queueEvalAct":"false",
                                                       "queueSpc":"1.5",
                                                       "queueSpcChoice":"QUEUESPACINGCHOICEDEFAULT",
                                                       "showClsfValues": "true",
                                                       "showIndivPeds":"true",
                                                       "thickness":"0.05",
                                                       "waitAreaMinSpc":"0",
                                                       "waitPosApprMeth":"DIRECTLINE",
                                                       "waitTmIsRelToSimStart":"false",
                                                       "zOffset":"0"})
                    geometry = area_elements.find('geometry')
                    geometry_element = ET.Element('geometry')
                    points = geometry.find('points')
                    points_element = ET.Element('points')
                    for point in points.findall('point'):
                        x = point.attrib['x']
                        y = point.attrib['y']
                        point_element = ET.Element('point', {"x":x, "y":y})
                        points_element.append(point_element)
                    geometry_element.append(points_element)
                    area_element.append(geometry_element)
                    areas_element.append(area_element)

                    # Maak dan de input aan voor die bus
                    pedestrianInput_element = ET.Element('pedestrianInput', {"name":"I_" + timetable_name, "no":str(pedestrianInput_number)})
                    location_element = ET.Element('location', {"area":str(area_number)})
                    posOnArea_element = ET.Element('posOnArea',{"x":"0", "y":"0"})
                    location_element.append(posOnArea_element)
                    timeIntPedVols_element = ET.Element('timeIntPedVols')

                    # Vind de begin en eind intervallen
                    start = 0
                    end = 0
                    for j in range(len(intervals)):
                        if intervals[j] == int_start[timetable.at[i, 'vertrektijd']]:
                            start = j
                        if intervals[j] == int_end[timetable.at[i, 'vertrektijd']]:
                            end = j

                    for k in range(start, end):

                        timeIntervalPedVolume_element = ET.Element('timeIntervalPedVolume', {"cont":"false",
                                                                                             "pedComp":"1",
                                                                                             "timeInt":"5 " + str(intervals[k]) + "000",
                                                                                             "volType":"STOCHASTIC",
                                                                                             "volume":"0"})
                        timeIntPedVols_element.append(timeIntervalPedVolume_element)
                    pedestrianInput_element.append(location_element)
                    pedestrianInput_element.append(timeIntPedVols_element)
                    pedestrianInputs_element.append(pedestrianInput_element)

                    # En als laatste maak de routes naar de voorkeurshalte voor deze input aan
                    for elements in areas_element.findall('area'):
                        if elements.attrib['name'] == ("D" + str(timetable.at[i, 'ivkp'])):
                            stop_number = elements.attrib['no']
                            stop_name = elements.attrib['name']
                        continue
                    make_ped_route(vissim_root, str(area_number), stop_number, stop_name, timetable_name) #Maak route naar de voorkeurshalte
```

Figure 35: Generation of areas, inputs and routes

Because routes have to be made quite a lot, a function has been set up called make_ped_route (it can be seen multiple times in the code above) to make the repetition of this process easier:

```
def make_ped_route(vissim_root, area1, area2, area2_name, timetable_name):
    """Functie om automatisch statische pedestrian routes te maken op basis van opgegeven begin- en eindgebied

    Args:
        vissim_root (root): Ingelezen INPX-bestand
        area1 (int): Startgebied voor de route
        area2 (int): Eindgebied voor de route
        area2_name (str): Naam van het eindgebied
        timetable_name (str): Naam van de simbuslijn en vertrektijd
    """
    route_number = get_no_of_last_element(vissim_root, 'pedestrianRoutingDecisionsStatic', 'pedestrianRoutingDecisionStatic')
    route_number += 1
    pedestrianRoutingDecisionsStatic_element = vissim_root.find('pedestrianRoutingDecisionsStatic')
    if timetable_name != None:
        # De bus/timetable namen worden gebruikt om voetgangers uit inputs te identificeren
        pedestrianRoutingDecisionStatic_element = ET.Element("pedestrianRoutingDecisionStatic", {"allPedTypes":"true",
                                                                                                 "name":timetable_name,
                                                                                                 "no":str(route_number),
                                                                                                 "pathChoiceMeth":"AREACENTER",
                                                                                                 "routeChoiceMeth":"STATIC"})
    else:
        pedestrianRoutingDecisionStatic_element = ET.Element("pedestrianRoutingDecisionStatic", {"allPedTypes":"true",
                                                                                                 "name":area2_name,
                                                                                                 "no":str(route_number),
                                                                                                 "pathChoiceMeth":"AREACENTER",
                                                                                                 "routeChoiceMeth":"STATIC"})
    location_element = ET.Element('location', {"area":area1})
    posOnArea_loc_element = ET.Element('posOnArea',{"x":"0", "y":"0"})
    location_element.append(posOnArea_loc_element)
    pedRoutSta_element = ET.Element('pedRoutSta')
    pedestrianRouteStatic_element = ET.Element('pedestrianRouteStatic', {"formula":"",
                                                                         "name":area2_name,
                                                                         "no":"1",
                                                                         "relFlow":""})
    pedRoutLoc_element = ET.Element('pedRoutLoc')
    pedestrianRouteLocation_element = ET.Element('pedestrianRouteLocation', {"area":area2,
                                                                             "banElevUse":"false",
                                                                             "calcInt":"1",
                                                                             "cellsize":"0.15",
                                                                             "g":"1.5",
                                                                             "h":"0.7",
                                                                             "impact":"1",
                                                                             "obstDist":"0.075",
                                                                             "pathChoiceMethod":"AREACENTER",
                                                                             "useDynPot":"false"})
    posOnArea_sta_element = ET.Element('posOnArea',{"x":"0", "y":"0"})
    pedestrianRouteLocation_element.append(posOnArea_sta_element)
    pedRoutLoc_element.append(pedestrianRouteLocation_element)
    pedestrianRouteStatic_element.append(pedRoutLoc_element)
    pedRoutSta_element.append(pedestrianRouteStatic_element)
    pedestrianRoutingDecisionStatic_element.append(location_element)
    pedestrianRoutingDecisionStatic_element.append(pedRoutSta_element)
    pedestrianRoutingDecisionsStatic_element.append(pedestrianRoutingDecisionStatic_element)
```

Figure 34: Function for making pedestrian routes

With this the preparation of the network is complete. The following step is during the initialization of the simulation run. Here, all holding areas and all routes found, stored in global dictionaries and the flows of the routes set to 0. The dictionary ped_route_aan will be explained in more detail in the next part.

```
        # Vind alle decision gebieden
        global D_areas
        global ped_route_aan
        D_areas = {}
        ped_route_aan = {}
        for area in Vissim.Net.Areas:
            if area.AttValue("Name").startswith('D'):
                D_areas[area.AttValue("No")] = area.AttValue("Name")
                ped_route_aan[area.AttValue("Name")] = 3

        # Vind alle static pedestrian routes en zet de juiste RelFlows op 0
        global PedRouteDecs
        global PedRouteStas
        PedRouteDecs = {}
        PedRouteStas = {}
        for pedroutedec in Vissim.Net.PedestrianRoutingDecisionsStatic:
            if len(pedroutedec.AttValue("Name")) > 0:
                PedRouteDecs[pedroutedec.AttValue("Name")] = pedroutedec.AttValue("No")
                PedRouteStas[pedroutedec.AttValue("No")] = pedroutedec.PedRoutSta.ItemByKey(1)
                if len(pedroutedec.AttValue("Name")) == 2:
                    pedroutedec.PedRoutSta.ItemByKey(1).SetAttValue('RelFlow(1)', 0)
```

Figure 36: Initialization of holding areas and pedestrian routes

Last but not least there is the implementation that runs during the simulation. When standing in the holding area, pedestrians can go in three directions: to the waiting area, to the exit, or to another platform. To regulate the process, each stop receives a timer. This is the dictionary ped_route_aan. Every timestep 1 is subtracted from the timer. If the timer is smaller than 1, the model is ready to assign a pedestrian to a route. When a pedestrian can go, the timer is set to 3, and the relative flow of the correct route is set to 1. Two timesteps later, when the timer is 1, the pedestrian is on its way and the flows of the routes can be set to 0 again. One timestep later, the model is ready to repeat the process. This ensures only a single pedestrian is set on a route, and not an entire group that may have different destinations. It makes the assignment of pedestrians limited in a time frame but this can be optimized by sending all pedestrians with the same destination to the same route at once.

To know where a pedestrian should go, a new variable acting as status of the bus is introduced for two cases. If the bus drives on the link that has the stop, its status is set to 'instap', and pedestrians with that bus as destination can be sent to the waiting area.

```
695                 if bus.current_action == 'instap' and passengers:
696                     bus.status = 'instap' # Verander status naar instap zodat passagiers naar de halte gaan lopen
```

Figure 37: Status change for boarding

If the bus is at the platform and about to leave, the status is set to 'weg', indicating the bus as left

```
718         if veh.AttValue('DwellTm') < 8 and (self.buses[bus]['status'] == 'haltering' or self.buses[bus]['status'] == 'tweede haltering') and bus.status == 'instap':
719             bus.status = 'weg' # Verander status naar weg zodat overgebleven passagiers naar de uitgang lopen; de 8 sec is zodat niemand overblijft omdat ze anders niet op tijd bij de bus zijn en Vissim niet kijkt naar mensen die er bijna zijn.
```

Figure 38: Status change for gone buses

and the pedestrian should return to the exit. When the bus has left the network, it's status cannot be checked anymore but by looping through the list with buses that have left the network it can be found whether the bus the bus pedestrian wanted to take is gone.

If the selected platform of the bus is not the same as the preferred platform, the pedestrians can be sent to the selected platform. The entire block of code is shown below:

```
if passengers and len(Vissim.Net.Pedestrians) > 0:
    # Ga door alle pedestrians in het netwerk, zet hun bestemming correct en pas routes aan waar nodig
    for ped in Vissim.Net.Pedestrians:
        if len(ped.AttValue("ToBus")) == 0 and ped.AttValue("StaRoutDecNo") != None:
            keys = [k for k, v in PedRouteDecs.items() if v == ped.AttValue("StaRoutDecNo")]
            if len(keys) > 0:
                ped.SetAttValue('ToBus', keys[0])
        ped_area = ped.AttValue("ConstrElNo")
        if ped_area in D_areas.keys() and ped_route_aan[D_areas[ped_area]] < 1:
            # Vind welke halte het om gaat
            listrank = list(D_areas.keys()).index(ped_area)

            # Vind de bus waar ze op wachten, check status van die bus en stuur de juiste routes aan waar nodig
            for bus in buses_in_network.values():
                if ped.AttValue("ToBus") in bus.name and bus.status == 'instap' and ped.AttValue('DwellTm') != None:
                    # Bus komt aan
                    # Verzet routes zodat hij naar de juiste wachtruimte loopt
                    if bus.selected_busstop != bus.timetable_data['ivkp'] and D_areas[ped_area] == ('D'+ str(bus.timetable_data['ivkp'])):
                        PedRouteStas[PedRouteDecs[('D'+ str(listrank+1) + 'D' + str(bus.selected_busstop))]].SetAttValue('RelFlow(1)', 1)
                    else:
                        PedRouteStas[PedRouteDecs[('W'+ str(listrank+1))]].SetAttValue('RelFlow(1)', 1)
                        #if bus.Vissim_reference.AttValue("Speed") < 1 and bus.Vissim_reference.AttValue('DwellTm') < 10:
                        #    bus.Vissim_reference.SetAttValue('DwellTm', 10) # Om ervoor te zorgen dat de bus op de evt. late passagier nog wacht
                    ped_route_aan[D_areas[ped_area]] = 3
                    ped.SetAttValue('DwellTm', 0)
                elif ped.AttValue("ToBus") in bus.name and bus.status == 'weg' and ped.AttValue('DwellTm') != None:
                    # Bus is van perron vertrokken
                    # Verzet routes zodat hij naar de uitgang loopt
                    PedRouteStas[PedRouteDecs[('U'+ str(listrank+1))]].SetAttValue('RelFlow(1)', 1)
                    ped_route_aan[D_areas[ped_area]] = 3
                    ped.SetAttValue('DwellTm', 0)

            for bus_name in buses_out_netwerk:
                if ped.AttValue("ToBus") in bus_name and ped.AttValue('DwellTm') != None:
                    # Bus is uit netwerk
                    # Verzet routes zodat hij naar de uitgang loopt
                    PedRouteStas[PedRouteDecs[('U'+ str(listrank+1))]].SetAttValue('RelFlow(1)', 1)
                    ped_route_aan[D_areas[ped_area]] = 3
                    ped.SetAttValue('DwellTm', 0)

    for i in ped_route_aan.keys():
        if ped_route_aan[i] == 1:
            # Vind welke halte het om gaat
            listrank = list(ped_route_aan.keys()).index(i)

            # Zet alle routes van deze halte uit
            PedRouteStas[PedRouteDecs[('W'+ str(listrank+1))]].SetAttValue('RelFlow(1)', 0)
            PedRouteStas[PedRouteDecs[('U'+ str(listrank+1))]].SetAttValue('RelFlow(1)', 0)
            for j in range(len(ped_route_aan)):
                if j != listrank:
                    PedRouteStas[PedRouteDecs[('D'+ str(listrank+1) + 'D' + str(j+1))]].SetAttValue('RelFlow(1)', 0)
            ped_route_aan[i] -= 1
```

Figure 39: Route choice implementation

Appendix E – Pedestrian Hindrance Solution

In this appendix, the solution to determining pedestrian hindrance is discussed.

During a simulation, a vehicle can be blocked by other traffic, or the rules of traffic. In Vissim, a vehicle knows the type and the number of the next object it could wait for or interact with. Using this train of thought, it can be checked whether the object referenced in this interaction is the pedestrian crossing. If so, the delay time of this bus can be checked.

During the initialization of the simulation run, a global dictionary is made for keeping track of all buses that experience hindrance, and the possible conflict areas are noted in a global list:

```
1005        # Definieer dictionary voor passagiershinder en conflict areas waar hinder kan gebeuren
1006        global bus_pass_hinder
1007        global pass_conflict_area
1008        bus_pass_hinder = {}
1009        pass_conflict_area = []
1010        for conflict_area in Vissim.Net.ConflictAreas:
1011            if (conflict_area.Link1.AttValue('IsPedArea') == True or conflict_area.Link2.AttValue('IsPedArea') == True):
1012                pass_conflict_area.append(conflict_area.AttValue('No'))
```

Figure 40: Initialization of pedestrian hindrance

During the simulation, over each bus is iterated to see whether they can possibly experience hinder from a pedestrian crossing. If so, they are added to the bus_pass_hinder dictionary and their current delay time is noted. After passing the crossing, the delay time of the bus is checked with the previously stored value in bus_pass_hinder and if the difference in the two is significant this is stored in the logging.

```
1189    # Ga links langs waar passagiershinder kan ontstaan
1190    if passengers and Vissim.Net.Vehicles.Count > 0:
1191        for veh in Vissim.Net.Vehicles.GetAll():
1192            busname = veh.AttValue('Name')
1193            # Als de bus voor de link van de passagiersoversteekplaats wacht, bereken dan de DelayTm
1194            if len(busname) > 0 and busname not in bus_pass_hinder.keys() and (veh.AttValue('InteractState') == "BRAKEAX" or veh.AttValue('InteractState') == "CLOSEUP") and veh.AttValue('InteractTargType') == "CONFLICTAREA":
1195                if veh.AttValue('InteractTargNo') in pass_conflict_area:
1196                    bus_pass_hinder[busname] = veh.AttValue('DelayTm') # De DelayTm voordat hij moet wachten op voetgangers wordt hiermee vastgelegd
1197            elif busname in bus_pass_hinder.keys() and (veh.AttValue('InteractTargType') != "CONFLICTAREA" or veh.AttValue('InteractTargNo') not in pass_conflict_area):
1198                totale_hinder = veh.AttValue('DelayTm') - bus_pass_hinder[busname] # Totale vertraging door voetgangers
1199                if totale_hinder > 1: # Alleen significante verschillen worden meegenomen, zodat het niet per ongeluk of op andere momenten gaat
1200                    add_to_logbook('bus', busname, 'hinder door voetgangers', totale_hinder)
1201                del bus_pass_hinder[busname]
```

Figure 41: Checking for pedestrian hindrance

Appendix F – All Other Code

In this chapter, any code that has been programmed by me but has not been mentioned in any other chapter yet will be shown here. Firstly, all items regarding the preparation of the network are explained, after which the items regarding the initialization of the network are explained. Finally, the items executed during the simulation are explained.

The first and only item for the preparation of the network is the insertion of free distributions. As mentioned in 4.3.4. Boarding and Alighting, the model makes use of decision points in the network. For the purpose of determining the dwell time earlier, vehicle attribute decisions will be used to determine the dwell time. These are placed at the decision points. Connected to vehicle attribute decisions in Vissim are free distributions. In the code of SimBus Pro, all time distributions are already retrieved from the timetable, and thus all that needs to be done is the addition of those time distributions to the free distributions:

```
97  def insert_free_distributions(vissim_root, free_distributions):
98      """Deze functie voegt elke unieke tijdsdistributie toe aan de free distributies van Vissim. Dit omdat de vehicle attribute decisions de free distributions gebruiken.
99          Het doet hetzelfde als insert_time_distributions, maar deze hebben net andere indexen voor 'no' of kunnen dit hebben.
100         Ook voegt deze functie de vehicle attribute decisions op de correcte locaties toe aan het netwerk.
101
102     Args:
103         vissim_root (root): Ingelezen inpx-bestand
104         time_distributions (dict): alle unieke tijdsdistributies
105     """
106
107     print("Voeg de free distributions en decisions toe aan Vissim")
108     # Voeg eerst de free distributions toe
109     freeDistribution_number = get_no_of_last_element(vissim_root, 'freeDistributions', 'freeDistribution')
110     freeDistributions_element = vissim_root.find('freeDistributions')
111     for free_distribution_key, free_distribution_value in free_distributions.items():
112         freeDistribution_number += 1
113         freeDistribution_element = ET.Element('freeDistribution', {'mean': str(free_distribution_value['mean_time']),
114                                                                    'name': str(free_distribution_key),
115                                                                    'no': str(freeDistribution_number),
116                                                                    'stdDev': str(free_distribution_value['standard_deviation']),
117                                                                    'type': 'NORMAL'})
118         freeDistributions_element.append(freeDistribution_element)
119         free_distribution_value['no'] = freeDistribution_number
120         free_distributions[free_distribution_key] = free_distribution_value
```

Figure 42: Insertion of free distributions

Afterwards, the vehicle attribute decisions can be made using the free distributions. To make the connection between the attribute decision and the dwell time of the bus, a custom attribute called SetDwellTime has been made. This attribute contains the dwell time that is given to the calculation for updating the dwell time.

```python
    # En dan de vehicle attribute decisions
    vehicleAttributeDecisions_element = vissim_root.find('vehicleAttributeDecisions')
    vehicleAttDec_number = get_no_of_last_element(vissim_root, 'vehicleAttributeDecisions', 'vehicleAttributeDecision')
    for freeDistr_element in vissim_root.find('freeDistributions').findall('freeDistribution'):
        for link_element in vissim_root.find('links').findall('link'):
            # Vind de sign-in links
            link_name = link_element.attrib['name']
            if len(link_name) == 0 or len(freeDistr_element.attrib['name']) < 4 or freeDistr_element.attrib['name'] == 'Dummy':
                continue
            if not '_' in link_name:
                continue
            link_element_split = link_name.split("_")
            if len(link_element_split) != 2:
                continue
            if not link_element_split[0].isnumeric():
                continue
            if link_element_split[1].lower() != 'in':
                continue
            vehicleAttDec_number += 1
            vehicleAttributeDecision_element = ET.Element('vehicleAttributeDecision', {"allVehTypes":"false",
                                                                                       "attr":"SETDWELLTIME",
                                                                                       "decType":"DISTRIBUTION",
                                                                                       "distr":str(freeDistr_element.attrib['no']),
                                                                                       "lane":str(link_element.attrib['no']) + " 1",
                                                                                       "name":str(freeDistr_element.attrib['name']),
                                                                                       "no":str(vehicleAttDec_number),
                                                                                       "pos":"4",
                                                                                       "timeFrom":"0",
                                                                                       "timeTo":"99999",
                                                                                       "value":"0"})
            vehClasses_element = ET.Element('vehClasses')
            intObjectRef_element = ET.Element('intObjectRef', {"key":"14"}) # Is een normale bus in jullie basis Vissim netwerk. Heb ik gekozen omdat ik een voertuigtype nodig had dat de decisi
            vehClasses_element.append(intObjectRef_element)
            vehicleAttributeDecision_element.append(vehClasses_element)
            vehicleAttributeDecisions_element.append(vehicleAttributeDecision_element)
```

Figure 45: Insertion of vehicle attribute decisions

During the initialization of the simulation run, the seed and bus capacity dictionary is defined, and the number of passengers boarding each bus is determined and set correctly in the pedestrian inputs:

```python
if passengers:
    random.seed(Vissim.Simulation.AttValue('RandSeed')) # Definieer de seed voor de distributies

    # Definieer buscapaciteit
    global buscapacity
    buscapacity = {12: 88, 13: 88, 18: 126} # 88 voor een 12 en 13 meter bus, 126 voor een 18 meter gelede bus
```

Figure 44: Definition of seed and bus capacity

```python
    # Bereken het aantal passagiers dat in moet stappen
    for i in range(len(timetable)):
        if timetable.at[i, 'aanuit'] == 1 and timetable.at[i, 'type'] < 5: # Type 5, 6 en 7 hebben geen instap
            ghal = timetable.at[i, 'ghal']
            spawntime_start = timetable.at[i, 'passstart'] # ... seconden voor een bus zal vertrekken beginnen met voetgangers genereren
            spawntime_end = timetable.at[i, 'passeind'] # ... seconden voor een bus zal vertrekken stoppen met voetgangers genereren
            spawntime_diff = spawntime_start - spawntime_end
            total_pass_in = random.expovariate(0.160548860738857) # Exponentiële distributie voor de passagiersgroepen
            max_pass_in = floor(ghal / 3.3919) # Maximale aantal passagiers dat kan instappen in de halteertijd
            if total_pass_in > max_pass_in:
                total_pass_in = max_pass_in # Om te zorgen dat er niet meer passagiers instappen dan de halteertijd van de bus
            elif total_pass_in > buscapacity[timetable.at[i, 'lengte']]:
                total_pass_in = buscapacity[timetable.at[i, 'lengte']] # Capaciteit van de bus
            ped_volume = floor(total_pass_in*(3600/spawntime_diff)) # Van aantal passagiers naar voetgangers per uur
            for ped_input in Vissim.Net.PedestrianInputs:
                if (str(timetable.at[i, 'simbuslijn']) + "_" + str(timetable.at[i, 'aankomsttijd'])) in ped_input.AttValue("Name"):
                    ped_input.TimeIntPedVols.SetAllAttValues("Volume", ped_volume)
```

Figure 43: Calculation of number of boarding passengers

In the preparation of the network, the free distributions and vehicle attribute decisions made in the preparation are initialized. They are stored in global dictionaries as well. On top of this, the number of the vehicle class belonging to the buses of SimBus is located, together with a standard class that can be used to make the attribute decisions non-active:

```
    # Vind alle free distributions en zet ze correct voor de vehicle attribute decisions
    global free_distributions
    free_distributions = {}
    for free_distribution in Vissim.Net.FreeDistributions:
        free_distribution_name = free_distribution.AttValue('Name')
        if len(free_distribution_name) > 0:
            free_distributions[free_distribution_name] = free_distribution.AttValue('No')

    # Vind het class-nummer van Vehicle Class OV
    global OVClass
    global BusClass
    for classes in Vissim.Net.VehicleClasses:
        if classes.AttValue("Name") == "OV":
            OVClass = classes.AttValue("No")
        # En ook een standaard class die negeert wordt door de rest van het verkeer
        if classes.AttValue("Name") == "Bus":
            BusClass = classes.AttValue("No")

    # Vind alle attribute decisions en zet ze correct voor de startwaarde
    global veh_att_dec
    veh_att_dec = {}
    for d_point in Vissim.Net.VehicleAttributeDecisions:
        veh_att_dec[(str(d_point.AttValue('No')) + " " + d_point.AttValue("Name"))] = d_point
        d_point.SetAttValue("VehClasses", BusClass)
```

Figure 48: Declaration of variables for free distributions, vehicle classes and vehicle attribute decisions

During the simulation the vehicle class that triggers the attribute decision has to be set to the correct class when the bus approaches the attribute decision and back to the standard class once has bus is gone:

```
        for d_point in veh_att_dec:
            # Zet de juiste vehicle attribute decision op actief
            if  str(bus.free_timedistribution) in veh_att_dec[d_point].AttValue('Name') and str(self.sign_in_link) == str(veh_att_dec[d_point].AttValue('Lane')).split("-")[0]:
                veh_att_dec[d_point].SetAttValue("VehClasses", OVClass)
```

Figure 47: Application of correct vehicle class

```
        for d_point in veh_att_dec:
            # Zet de vehicle attribute desicion weer op non-actief
            if str(veh_att_dec[d_point].AttValue('Name')) == str(bus.free_timedistribution) and str(self.sign_in_link) == str(veh_att_dec[d_point].AttValue('Lane')).split("-")[0]:
                veh_att_dec[d_point].SetAttValue("VehClasses", BusClass)
```

Figure 46: Application of standard vehicle class

Immediately after doing this, the function for determining the number of alighting passengers is called, if necessary:

```
        if action == 'uitstap' and bus.Vissim_reference.AttValue('SetDwellTime') > 0 and passengers:
            # Bepaal het aantal uitstappende passagiers
            total_pass_uit = bus.determine_passengers(bus.Vissim_reference.AttValue('SetDwellTime'), 1)
            bus.Vissim_reference.SetAttValue('Occup', total_pass_uit)
        elif action == 'instap' and bus.type != 3 and bus.first_action == 'instap' and passengers:
            total_pass_uit = bus.determine_passengers(bus.Vissim_reference.AttValue('SetDwellTime'), 2)
            bus.Vissim_reference.SetAttValue('Occup', total_pass_uit)
```

Figure 49: Call of function for alighting passengers

Finally, there is the function that determines the amount of passengers alighting at a stop. Both the power function and the exponential distribution are implemented here according to the conceptual model.

```
def determine_passengers(self, halteertijd, type):
    """Deze functie bepaalt het aantal passagiers dat bij de halte moet uitstappen, gebaseerd op de gewenste halteertijd en de bepaalde functie hiervoor.
        Deze functie wordt aangeroepen als de bus een halte heeft gekozen en er naartoe rijdt.

    Args:
        halteertijd (float): De halteertijd die getrokken is uit de standaardverdeling
        type (int): 1 = uitstap, 2 is combi uitstap/instap
    """
    if type == 1:
        total_pass_uit = floor((halteertijd/2.9456)**(1/0.7925))
        if total_pass_uit > buscapacity[self.timetable_data['lengte']]:
            total_pass_uit = buscapacity[self.timetable_data['lengte']] # Capaciteit van de bus
        return total_pass_uit
    elif type == 2:
        total_pass_uit = floor(random.expovariate(0.119273591926339)) # Exponentiële distributie voor de passagiersgroepen
        max_pass_uit = floor((halteertijd/2.9456)**(1/0.7925)) # Maximale aantal passagiers dat kan uitstappen in de halteertijd
        if total_pass_uit > max_pass_uit:
            total_pass_uit = max_pass_uit # Om te zorgen dat er niet meer passagiers uitstappen dan de halteertijd van de bus
```

Figure 50: Function for calculating the amount of alighting passengers