

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Traffic Modelling and Simulation Towards Digital Twins of Road Networks in Uninterrupted Flow Settings

Henrique Costa Sousa



Master in Informatics and Computing Engineering

Supervisor: Prof. Rosaldo José Fernandes Rossetti

July 26, 2024

Traffic Modelling and Simulation Towards Digital Twins of Road Networks in Uninterrupted Flow Settings

Henrique Costa Sousa

Master in Informatics and Computing Engineering

Approved by:

President: Prof. Daniel Augusto Gama de Castro Silva

Referee: Prof. Vitor Hugo Mendes da Costa Carvalho

Supervisor: Prof. Rosaldo José Fernandes Rossetti

July 26, 2024

Resumo

A gestão e o desenvolvimento da infraestrutura rodoviária são tarefas complexas. A simulação tem sido usada para abordar este problema. Simultaneamente, o crescimento da tecnologia de Gémeos Digitais (Digital Twins) tem permitido que sistemas reais sejam controlados em tempo real através de uma representação virtual do referido sistema, a qual é mantida atualizada com o uso de dados obtidos também em tempo real. A combinação das duas tecnologias resulta num Gémeo Digital de uma rede rodoviária que utiliza dados recolhidos a partir dos sensores colocados nas estradas para realizar uma simulação que funciona simultaneamente e replica o estado do sistema real.

Embora haja um extenso trabalho sobre a criação e simulação de redes rodoviárias, a maioria não considera dados em tempo real ou online. De facto, a calibração da demanda de tráfego parece ser a norma, o que predetermina os veículos que devem ser introduzidos na rede e as rotas que irão seguir. Abordagens que utilizam dados durante a simulação são mais raras.

Este trabalho propõe e implementa uma metodologia que preenche essa lacuna na literatura. Descreve o processo de criação, calibração e execução de uma simulação que pode aproveitar um fluxo constante de dados (em tempo real ou não) para modificar ativamente a simulação de modo a corresponder ao sistema real. O simulador *Simulation of Urban MObility* (SUMO) é usado como referência para a implementação.

A criação da rede é automatizada e depende do OpenStreetMap (OSM), assumindo que quaisquer sensores utilizados foram previamente conhecidos. Isto permite a partição da rede em sub-redes que podem ser simuladas independentemente, e o cálculo das dependências entre os fluxos das estradas dentro da rede através de sistemas de equações lineares.

A execução da simulação implica um controlo ativo dos veículos, redirecionando-os, e o ajuste do seu comportamento de condução, o que é feito por um modelo de *Machine Learning* (ML) que utiliza os dados reais e o estado da simulação como entrada.

Um estudo de caso foi realizado para validar a metodologia. A “Via de Cintura Interna” (VCI) no Porto, Portugal, foi simulada. A rede obtida, a sua partição, os sensores utilizados, o sistema de equações lineares deduzido relacionado com os fluxos rodoviários e o treino dos modelos foram analisados. A validação também é realizada comparando os resultados da simulação com os dados correspondentes que foram utilizados como sua entrada. Os modelos de Machine Learning foram bem-sucedidos em diminuir o erro associado à velocidade dos veículos.

Palavras-chave: Digital Twins, road networks, microscopic traffic simulation, simulation calibration, simulation validation, Simulation of Urban MObility, uninterrupted flow

Abstract

The management and development of road infrastructure is a complex task. Simulation has been used to tackle this problem. Simultaneously, the growth of Digital Twin technology has enabled real systems to be controlled in real-time by using a virtual representation of said system which is kept updated with the use of data collected in real-time. Combining the two technologies results in a Digital Twin of a road network that employs data collected from the sensors placed on the roads to achieve a simulation that runs simultaneously and replicates the state of the real system.

Though there is extensive work on the creation and simulation of road networks, most do not consider real-time or online data. In fact, the calibration of the traffic demand appears to be the norm, which predetermines the vehicles that should be introduced into the network and the routes that they are going to take. Approaches that make use of data during the simulation are rarer.

This work proposes and implements a methodology that fills that gap in literature. It describes the process of creation, calibration, and running of a simulation that can take advantage of a constant stream of data (in real-time or not) in order to actively modify the simulation to match the real system. The Simulation of Urban MObility (SUMO) simulator is used as a reference for the implementation.

The creation of the network is automated and relies on OpenStreetMap (OSM), assuming that any used sensors were known beforehand. This allows for the partition of the network into sub-networks which can be simulated independently, and the computation of the dependencies between the flows of the roads within the network through systems of linear equations. Running the simulation implies an active control of the vehicles, by rerouting them, and adjusting their driving behavior, which is done by a Machine Learning (ML) model that takes the real data and the state of the simulation as input.

A case study was conducted to validate the methodology. The “Via de Cintura Interna” (VCI) in Porto, Portugal, was simulated. The network obtained, its partition, the sensors used, the deduced system of linear equations related to road flows, and the training of the models were analysed. Validation is also performed by comparing the results of the simulation to the corresponding data which was used as its input. The Machine Learning models were successful in decreasing the error associated with the speed of the vehicles.

Keywords: Digital Twins, road networks, microscopic traffic simulation, simulation calibration, simulation validation, Simulation of Urban MObility, uninterrupted flow

Acknowledgements

First, I would like to thank Professor Rosaldo Rossetti, who accompanied me throughout the whole project with advice and orientation that made this project possible.

My gratitude to ARMIS who promoted this work and provided the necessary assets for its execution.

Recognition must also go to all of the other professors on the faculty who have shaped my journey in the institution and also the colleagues who have shared this experience with me.

On a more personal note, I must share my deep deep appreciation for all of my family and friends who have always been present and have been the cornerstone of these five years and before.

Henrique Sousa

“All models are wrong, but some are useful.”

George Box

Contents

Acknowledgements	iii
1 Introduction	1
1.1 Context and problem	1
1.2 Objectives and goals	1
1.3 Structure	3
2 Background	4
2.1 Traffic simulation	4
2.1.1 Granularity	4
2.1.2 Simulation	4
2.1.3 Traffic demand	4
2.2 Digital Twins	5
3 Literature review	6
3.1 Road network simulation	6
3.1.1 Road segment	6
3.1.2 Data collection	7
3.1.3 Modeling	13
3.1.4 Calibration	17
3.1.5 Validation	18
3.1.6 Gap analysis	19
3.2 Summary	19
4 Simulator specifications	20
4.1 Network	20
4.2 Simulation	20
4.3 Induction loop	21
4.4 Calibrator	21
4.5 Runtime APIs	22
5 Methodological approach	23
5.1 Problem formalisation	23
5.1.1 Network	23
5.1.2 Traffic	26
5.1.3 Objective	27
5.2 Materials	28
5.3 Methods	29
5.3.1 Network	29

5.3.2	Network information	32
5.3.3	Simulation	54
5.3.4	Modeling	67
5.4	Assessment	69
5.5	Summary	70
6	Experimental setup and results	72
6.1	Network	72
6.2	Network information	75
6.3	Dataset	77
6.4	Results baseline	85
6.5	Additional information	87
6.6	Training	87
6.7	Results and discussion	88
6.8	Summary	93
7	Conclusions	94
7.1	Objectives	94
7.2	Contributions and results	95
7.3	Future work	95
	References	98
A	Appendix	103
A.1	VCI independent systems	110
A.2	Data analysis	148
A.3	Experiment Validation	154
A.3.1	Validation of the simulation with no model for system 1	154
A.3.2	Validation of the simulation for model 1 for system 1	157
A.3.3	Validation of the simulation for model 2 for system 1	162
A.3.4	Validation of the simulation for model 3 for system 1	166
A.3.5	Validation of the simulation for model 4 for system 1	170
A.3.6	Validation of the simulation for model 5 for system 1	174
A.3.7	Validation of the simulation for model 6 for system 1	178
A.3.8	Validation of the simulation for model 7 for system 1	182

List of Figures

5.1	Example network	25
5.2	Example of a network with a continuous line on a split edge	46
5.3	Architecture of the simulation framework	55
5.4	Architecture of the machine learning model	69
6.1	Count of all samples by interval start	78
6.2	Traffic flow by sensor id	79
6.3	Comparison between average speeds	79
6.4	Distribution of flows by sensor id	80
6.5	Distribution of speed of intervals by sensor id	80
6.6	Average flow and interval speed in time	81
6.7	Average flow and vehicle speed in time	81
6.8	Average interval and vehicle speed in time	82
6.9	Average interval and vehicle speed in time for 17:00-19:00 period	83
6.10	Average interval and vehicle speed in time for 17:00-19:00 period	84
6.11	Absolute error for validation with no model in time	85
6.12	Comparison of real volume with simulated volume for validation with no model in time	86
6.13	Comparison of real speed with simulated speed for validation with no model in time	86
6.14	Comparison of real speed with simulated speed for validation for model 2 in time	88
6.15	Comparison of real speed with simulated speed for validation for model 3 in time	89
6.16	Absolute percentage error for validation for model 4 in time	89
6.17	Absolute percentage error for validation for model 6 in time	90
6.18	Absolute percentage error for validation for model 5 in time	90
6.19	Absolute percentage error for validation for model 7 in time	91
A.1	VCI's independent systems numbered from 1 to 24	110
A.2	Independent system 1	111
A.3	Independent system 2	115
A.4	Independent system 3	118
A.5	Independent system 4	119
A.6	Independent system 5	121
A.7	Independent system 6	121
A.8	Independent system 7	123
A.9	Independent system 8	125
A.10	Independent system 9	126
A.11	Independent system 10	128
A.12	Independent system 11	130

A.13 Independent system 12	131
A.14 Independent system 13	133
A.15 Independent system 14	135
A.16 Independent system 15	135
A.17 Independent system 16	136
A.18 Independent system 17	138
A.19 Independent system 18	139
A.20 Independent system 19	140
A.21 Independent system 20	141
A.22 Independent system 21	142
A.23 Independent system 22	143
A.24 Independent system 23	145
A.25 Independent system 24	146
A.26 Comparison between vehicle and interval average speeds in time for 17:00-19:00 period	148
A.27 Vehicle average speed by sample for 17:00-19:00 period	149
A.28 Interval average speed by sample for 17:00-19:00 period	149
A.29 Comparison between vehicle and interval average speeds by sensor for 17:00- 19:00 period	150
A.30 Vehicle average speed by sensor for 17:00-19:00 period	150
A.31 Interval average speed by sensor for 17:00-19:00 period	151
A.32 Interval speed distribution by sensor for 17:00-19:00 period	151
A.33 Traffic flow and average vehicle speed in time for 17:00-19:00 period	152
A.34 Traffic flow and average interval speed in time for 17:00-19:00 period	152
A.35 Traffic flow by sample for 17:00-19:00 period	153
A.36 Traffic flow by sensor for 17:00-19:00 period	153
A.37 Traffic flow distribution by sensor for 17:00-19:00 period	154
A.38 Comparison of real speed with simulated speed for validation with no model per sample	154
A.39 Comparison of real speed with simulated speed for validation with no model by sensor	155
A.40 Comparison of real volume with simulated volume for validation with no model per sample	155
A.41 Comparison of real volume with simulated volume for validation with no model by sensor	156
A.42 Absolute percentage error for validation with no model in time	156
A.43 Mean squared error for validation with no model in time	157
A.44 Comparison of real speed with simulated speed for validation for model 1 in time	157
A.45 Comparison of real speed with simulated speed for validation for model 1 per sample	158
A.46 Comparison of real speed with simulated speed for validation for model 1 by sensor	158
A.47 Comparison of real volume with simulated volume for validation for model 1 in time	159
A.48 Comparison of real volume with simulated volume for validation for model 1 per sample	159
A.49 Comparison of real volume with simulated volume for validation for model 1 by sensor	160
A.50 Absolute error for validation for model 1 in time	160
A.51 Absolute percentage error for validation for model 1 in time	161
A.52 Mean squared error for validation for model 1 in time	161

A.53 Comparison of real speed with simulated speed for validation for model 2 per sample	162
A.54 Comparison of real speed with simulated speed for validation for model 2 by sensor	162
A.55 Comparison of real volume with simulated volume for validation for model 2 in time	163
A.56 Comparison of real volume with simulated volume for validation for model 2 per sample	163
A.57 Comparison of real volume with simulated volume for validation for model 2 by sensor	164
A.58 Absolute error for validation for model 2 in time	164
A.59 Absolute percentage error for validation for model 2 in time	165
A.60 Mean squared error for validation for model 2 in time	165
A.61 Comparison of real speed with simulated speed for validation for model 3 per sample	166
A.62 Comparison of real speed with simulated speed for validation for model 3 by sensor	166
A.63 Comparison of real volume with simulated volume for validation for model 3 in time	167
A.64 Comparison of real volume with simulated volume for validation for model 3 per sample	167
A.65 Comparison of real volume with simulated volume for validation for model 3 by sensor	168
A.66 Absolute error for validation for model 3 in time	168
A.67 Absolute percentage error for validation for model 3 in time	169
A.68 Mean squared error for validation for model 3 in time	169
A.69 Comparison of real speed with simulated speed for validation for model 4 in time	170
A.70 Comparison of real speed with simulated speed for validation for model 4 per sample	170
A.71 Comparison of real speed with simulated speed for validation for model 4 by sensor	171
A.72 Comparison of real volume with simulated volume for validation for model 4 in time	171
A.73 Comparison of real volume with simulated volume for validation for model 4 per sample	172
A.74 Comparison of real volume with simulated volume for validation for model 4 by sensor	172
A.75 Absolute error for validation for model 4 in time	173
A.76 Mean squared error for validation for model 4 in time	173
A.77 Comparison of real speed with simulated speed for validation for model 5 in time	174
A.78 Comparison of real speed with simulated speed for validation for model 5 per sample	174
A.79 Comparison of real speed with simulated speed for validation for model 5 by sensor	175
A.80 Comparison of real volume with simulated volume for validation for model 5 in time	175
A.81 Comparison of real volume with simulated volume for validation for model 5 per sample	176
A.82 Comparison of real volume with simulated volume for validation for model 5 by sensor	176
A.83 Absolute error for validation for model 5 in time	177
A.84 Mean squared error for validation for model 5 in time	177
A.85 Comparison of real speed with simulated speed for validation for model 6 in time	178
A.86 Comparison of real speed with simulated speed for validation for model 6 per sample	178
A.87 Comparison of real speed with simulated speed for validation for model 6 by sensor	179
A.88 Comparison of real volume with simulated volume for validation for model 6 in time	179
A.89 Comparison of real volume with simulated volume for validation for model 6 per sample	180
A.90 Comparison of real volume with simulated volume for validation for model 6 by sensor	180

A.91 Absolute error for validation for model 6 in time	181
A.92 Mean squared error for validation for model 6 in time	181
A.93 Comparison of real speed with simulated speed for validation for model 7 in time	182
A.94 Comparison of real speed with simulated speed for validation for model 7 per sample	182
A.95 Comparison of real speed with simulated speed for validation for model 7 by sensor	183
A.96 Comparison of real volume with simulated volume for validation for model 7 in time	183
A.97 Comparison of real volume with simulated volume for validation for model 7 per sample	184
A.98 Comparison of real volume with simulated volume for validation for model 7 by sensor	184
A.99 Absolute error for validation for model 7 in time	185
A.100 Mean squared error for validation for model 7 in time	185

List of Tables

3.1	Road segment	6
3.2	Infrastructure: Data sources	7
3.3	Infrastructure: Information	8
3.4	Infrastructure: Data processing	9
3.5	Traffic: Data sources	10
3.6	Traffic: Data information	11
3.7	Infrastructure: Data processing	12
3.8	Modeling: Software	13
3.9	Modeling: Complementary software	13
3.10	Modeling: Space Granularity	14
3.11	Modeling: Input parameters	15
3.12	Modeling: Output parameters	16
3.13	Calibration	17
3.14	Validation: Methodology	18
3.15	Validation: Parameters	18
5.1	Definable entities for the example network	26
5.2	Reachable edges for network in Figure 5.2	49
5.3	Default values for used parameters of vehicle type	64
6.1	Depth for each edge type for filtering of the experiment network	73
6.2	Number of lanes per edge for experiment network	75
6.3	Network details of independent systems	76
6.4	Training configurations of the machine learning models	87
6.5	All metrics for the validated models. Averaging metrics of every data point	91
6.6	All metrics for the validated models. Averaging metrics by interval start	92

Listings

4.1	Example induction loop definition	21
4.2	Example calibrator definition	21
5.1	Definition of a SUMO edge's induction loop	34
5.2	Definition of a SUMO's edge route	34
5.3	Definition of a SUMO edge's entry calibrator	34
5.4	Implementation of sensor position calculating algorithm	35
5.5	Implementation of sensor position calculating algorithm for bidirectional sensor devices	36
5.6	Definition of a SUMO sensor's induction loop	38
5.7	Definition of a SUMO sensor's calibrators	38
5.8	Algorithm to calculate every path	40
5.9	Algorithm to initialize every route object	41
5.10	Algorithm to cross reference to calculate the next route objects	42
5.11	Algorithm to ensure order of the next route objects	42
5.12	Definition of a specific SUMO route	43
5.13	Algorithm to calculate rerouter routes and next routes	44
5.14	Algorithm to map edges to routes where it is featured	44
5.15	Algorithm to determine which lanes are reachable from a lane within an edge	47
5.16	Algorithm to calculate reachable edges information	48
5.17	Calculating independent system edges	50
5.18	SUMO configuration file	54
5.19	Function call to set flow of the sensor vehicle type calibrator	57
5.20	Calculate sensor flow calibrator route id	57
5.21	Running loop with vehicle rerouting	59
5.22	Adjustable Rerouter class	60
5.23	Adjustable Rerouter's algorithm to choose the next route	62
5.24	Function call to set flow of the SUMO edge's entry calibrator	66
5.25	Function call to set flow of the sensor flow calibrator	66
6.1	Algorithm for computing adjacent edges according to type and debth	73
A.1	Example sections of an .osm file	103
A.2	Configuration file for the conversion of the .osm files to a SUMO network file	103
A.3	Sumo configuration file for running a simulation	104
A.4	Changes to sumolib/net/__init__.py	105
A.5	Changes to sumolib/net/lane.py	106
A.6	Rerouter and AdjustableRerouter considering route compatibility	107

Acronyms and Symbols

ATCS	Adaptive Traffic Control System
API	Application Programming Interface
DM	Digital Model
DS	Digital Shadow
DT	Digital Twin
HVAC	Heating, ventilation, and air conditioning
IoV	Internet of Vehicles
IoT	Internet of Things
MAE	Mean absolute error
MAPE	Mean absolute percentage error
ML	Machine Learning
MSE	Mean squared error
OD	Origin-Destination
OSM	OpenStreetMap
RSE	Relative squared error
SUMO	Simulation of Urban MObility
VCI	"Via de Cintura Interna"
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Context and problem

Road infrastructure planning and operation is a difficult task. As for planning, not mentioning the legal and bureaucratic work of procuring land, having engineers design the road according to physical requirements, planning the road location, path, and what purpose it serves is of utmost importance. Furthermore, the operation of a road network is not trivial as it entails coordinating multiple and finite resources for tasks such as handling accidents, temporary obstacles, road works, cleaning, etc. To facilitate these tasks, simulation can be used to inform decisions. In this regard, simulation can be used in a descriptive manner to represent the system's current status or in a speculative way to, for example, predict traffic behavior if the conditions are changed.

Since traffic itself is complex and has multiple interconnected variables, one of which being unpredictable drivers, the simulation of a road network becomes an equally complex task. Therefore, how can one make such a simulation reflect reality? Not only does one need to develop a model that could represent an arbitrary but plausible road network, but we also need to calibrate it to the traffic scenario and validate its performance in that same scenario in real life.

While that alone is arduous, synchronizing the simulation with reality so that its results can have immediate value for traffic operators is another hurdle. The solution can be described as a Digital Twin (DT) (see Chapter 2). Achieving it would imply that the model is fed with real-time data, all computations are done at the same pace as reality, and the knowledge provided by the simulation can be used by the operators to enact change in the actual system.

1.2 Objectives and goals

In order to solve this problem, several interconnected and incremental steps must be taken. These steps can be formulated as the research questions that shall guide our methodology:

1. How to create a simulation?
 - (a) What tools should be used to run the simulation?

- (b) What data are necessary to create a road network suitable for simulation?
 - (c) How to create a road network for the simulation?
- 2. How to run an online simulation?
 - (a) What data are necessary to run an online simulation?
- 3. How to calibrate the simulation?
 - (a) What data are necessary to calibrate a simulation?
 - (b) How to automate the calibration of the simulation?
- 4. How to validate the simulation?

For the present work, we will attempt to answer these questions and present a methodology, which shall be used and applied to a specific scenario. The criteria for said task to have been successful implies the fulfillment of the following:

- Identify a tool to use for the simulation;
- Specify the data that shall be obtained to create a road network that is compatible with the tool;
- Specify what processing, modifications, and transformations should be applied to the data to create the road network;
- Specify what data is necessary to run an online simulation;
- Devise a methodology and framework that enables the running of an online simulation;
- Specify what data is needed to calibrate a simulation;
- Devise a methodology and framework that allows the calibration of an online simulation;
- Devise a methodology to validate a simulation;
- Describe if and how the aforementioned steps can be automated;
- Implementation of an online simulation in an automated manner through a case study;
- Validation of the case study according to the methodology;

The specific scenario chosen is the simulation of the "Via de Cintura Interna" (VCI) inner ring in Porto, Portugal. The characteristics of this network influenced the implementation and the methodology used. Additionally, cooperation with the ARMIS company will allow for the usage of data on traffic, mainly traffic counts from inductive-loop traffic detectors.

The present work aims to develop a methodology that can be applied in the simulation field, particularly microscopic traffic simulation. It draws from previous works on Digital Twins for road networks. It is expected to contribute to the field by employing Machine Learning (ML) algorithms to leverage the abundance of the data collected by sensors to calibrate the simulation.

1.3 Structure

In Chapter 2 we review some overall concepts and methods. In Chapter 3, we analyze current work done relating to road network simulations and the overall patterns. In Chapter 5, we explain in full the methodology used to develop the simulation. In Chapter 6, we analyze the network and data for the specific scenario used to validate the methodology. Finally, in Chapter 7, we reflect on the results of the work.

Chapter 2

Background

2.1 Traffic simulation

2.1.1 Granularity

The granularity of a simulation can be classified as either *macroscopic*, *mesoscopic*, and *microscopic*, from a lower to a higher level of detail [1]. For macroscopic simulation, traffic flows and speeds are what is simulated. For microscopic, the simulation happens at the vehicle level, with as many elements as possible of both the road and the driving being implemented. Traffic lights, road accesses, speed limits, and intersections can be modeled for the road; and driver behavior, overtaking, and route choosing are contemplated for traffic. Lane changing and vehicle following behaviors are embedded into the models, which dictate the more minute movement of a vehicle in relation to other vehicles on the road [2]. Mesoscopic simulation takes characteristics from both macroscopic and microscopic simulation. Each vehicle is simulated, but its movement and position are dictated by macroscopic phenomena [2], [3]. Some work also further identifies sub-microscopic models that take into consideration vehicle-specific characteristics like "engine characteristics, fuel map consumption, gears, battery or hybrid systems to compute EVs energy demand" [3].

2.1.2 Simulation

Traffic simulation can be done either through a model or a model-less approach. The latter may be implemented through Neural Networks [4], [5], [6]. In this work, however, we will only take into consideration traffic simulation utilizing a model. Besides the already established simulators like SUMO [7] and VISSIM [8], there is continuous work in developing and proposing new microscopic [9], mesoscopic [10], [11], and macroscopic [12] simulators.

2.1.3 Traffic demand

Part of the microscopic simulations analyzed make use of traffic demand in order to create traffic (see Chapter 3). This method is based on first defining a set of locations that can serve as either

origin or destination. Afterward, the number of trips between each origin-destination (OD) pair is calculated. Methodologies for the development of demand models are ubiquitous in the literature, and not only for vehicle-only transportation [13], [14], [15], [16], [17].

2.2 Digital Twins

The term Digital Twin is traced back to Michael Grieves, who introduced it in 2003 for his Product Life Management course [18]. NASA endorsed the concept in 2010 in [19] defining it as "an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin". Other work also present an application-specific or rather restrictive definition. For example, in [20] it is defined as "a virtual model designed to accurately reflect a physical object", and in [21] as "digital duplicates of CPPS [Cyber-Physical Production System] that represent, control, and monitor their physical counterpart". Given the high variety of definitions, there have been efforts to consolidate its usage. In [22] a general definition of a digital twin as "a virtual representation of a physical system (and its associated environment and processes) that is updated through the exchange of information between the physical and virtual systems" is proposed after reviewing multiple definitions across the literature.

In recent years, Digital Twins have been applied to various areas. For example rail management [23], manufacturing [24], [25], smart buildings [26], industry [27], [28], HVAC systems [29], [30], and critical infrastructure [31], [32]. Additionally, there has also been work done for road networks [33], [34], [35].

Besides *Digital Twin* other terminology has arisen to better distinguish between implementations with slightly different characteristics. In [36], the additional concepts of *Digital Model* and *Digital Shadow* are explained. They differ in the degree of automation of the data flows between the virtual and the physical and vice-versa. When there is no automation, it is named a *Digital Model*. When there is only an automated data flow from the physical to the virtual system it is considered a *Digital Shadow*. Only when the data flow is automated from the virtual system to the physical system would it be considered a *Digital Twin*. Despite the previously stated differences in definitions of Digital Twin, these distinctions seem to have gained some support [37], [38], [39].

Chapter 3

Literature review

In this chapter, we will review past work done in DTs regarding road networks. We will go point by point on the areas that will be considered essential for the analysis and elicitation of a methodology applicable to a DT of a road network.

3.1 Road network simulation

In this section, we provide an overview of each of the outlined steps. The studies have varied purposes and, therefore, different requirements and methodologies applied. As expected, the processes of each publication will not always be compatible or suitable for integration with one another. A work-by-work review discriminating each of the steps is not performed.

3.1.1 Road segment

Table 3.1: Road segment

Road segment	References
Urban	[40], [35], [41], [42], [43], [1], [44], [45], [46]
City-wide	[47], [35], [41], [42], [43], [1], [48], [49], [50]
Intersection	[44], [46]
Roundabout	[45]
Highway	[40], [47], [51], [50]
Highway interchange	[47], [33], [33]
On/Off-ramp	[40], [47], [34], [51], [50]
Other	[52]
Infrastructure only	[53]

Across the literature, studies in urban environments seem to be prevalent. It makes sense as that is where traffic accumulates, causing congestion, and whose study would cause a bigger impact. Additionally, locations where points of conflict, mainly crossing vehicles or changes of direction, exist are studied (on/off-ramps, interchanges, intersections and roundabouts). These locations are where traffic is also expected to be greater, and their study could be beneficial. Not only that, but the upgrade of the infrastructure, namely with the implementation of adaptive traffic control systems (ATCS) is intuitively suited for those scenarios. On the other hand, there is a wide range of city-wide studies which are not necessarily DTs. There appears to be a consensus that modeling and analyzing traffic yields good results. These results would take the form of insights into the dynamics of traffic in the city that can help city planners with their decisions. In [52] there is only a reference found to being "not a city center with huge traffic flow". Finally, in [53] an accurate and precise virtual representation of the road and pavement itself is attempted, and falls outside the scope of the work. It is listed here to promote the thought that, when the work into traffic simulation advances sufficiently that traffic is simulated to an acceptable degree of realism, the road itself might be an element that might be worth considering as it too affects traffic.

3.1.2 Data collection

We shall now overview the methods used for data collection and their integration into the DT.

Infrastructure: data sources

In a DT, we need to represent not only the dynamic entities, namely vehicles but also the static entities that those objects are going to interact with. We are henceforth going to refer to it as "Infrastructure". It is paramount that the infrastructure is well represented.

Table 3.2: Infrastructure: Data sources

Infrastructure data sources	References
Imagery	[53], [35], [42], [43]
Digital surface model	[53]
Digital elevation model	
Topography	[53]
Maps	[40], [47], [35], [41], [42]
Design drawings	[33]
Field data	[33], [42]
Other	[42], [43], [1]

The work using imagery uses Satellite, street-level, or aerial imagery. OpenStreetMap (OSM)[54] is widely used. Google Maps [55], HERE maps [56], and other high-definition maps are also utilized.

The work in [53], mentioned in Section 3.1.1 as regarding only the infrastructure, makes use of multiple types of data from Digimap [57] and also cites Digital elevation models as a possible source. Field data in [33] comes from a road friction tester and in [42] represents "on-site inspections". Additionally, in [42], GPS traces and data from the public transit company (on bus routes) have been used to fix and complement the data. In [43] data from the Austrian project GIP [58] is used. [1] mentions a "transportation planning network". In general, machine-readable data is used for larger road segments. Complementary data may be used for fine-tuning.

Overall, there is both the case of just one data source being used [1], [41], [52], [40], but it tends to be informationally dense enough that it suffices for the applications. The use of additional sources seems to be a recurrent practice.

Infrastructure: information

From the data collected, different papers focus on different parts of the infrastructure to model.

Table 3.3: Infrastructure: Information

Infrastructure information	References
Road markings	[53], [52], [35], [42]
Road limits	[33], [53], [52], [42]
Road design	[33], [53]
Road type	[35], [42]
Signals and rules	[52], [35], [41], [42]
Traffic lights	[35], [42], [1], [46]
Physical properties	[33]
Other	[52], [42]
Road network	[40], [47], [35], [41], [42], [43], [1]

Road markings are mainly concerned with identifying the lanes present in the road, as it is one of the fundamental characteristics of a road. Road design refers to the vertical and horizontal alignment and the cross-section, that is, the shape and position of the road in space. In [53], this is the main object of the study (see Section 3.1.1). However, in [33], this aspect is important because the risk calculation of the short road segment is heavily dependent on the shape of the road. Road limits refer to the edges of pavement in [53] and boundaries in [33]. In [42], the large-scale model is concerned with the width of the road. Though unclear, the road type appears

to influence the road network, particularly the layout of intersections. In [35] the priority of roads is specifically state as one of the features collected, whereas, in [42] there is also a mention of a priority level ranging from "1 (footpath) up to 13 (national motorway)." Signals and rules include properties such as speed limits, access rights, and traffic signs. Regarding traffic lights, not only their position is registered, but often also their programming [35],[1], [46]. In [33], the friction coefficient is measured for the purposes of the risk research. In [52], information about public transit and on-street parking is retrieved. In [42], there are mentions of other traffic facilities that are not clarified. On the work that contemplates a city-wide study (see Section 3.1.1), all of the above elements are joined together to assemble a road network that is supported by the software used for the study (see Table 3.8).

Infrastructure: data processing

The methods used to convert the data collected to the information needed are varied.

In [53], the process of extracting all the relevant information is described in detail. In [40], the software *osmconvert* and *osmosis* are used to handle the *.osm.pbf* data, whereas SUMO's *osmNetconvert* is later used; SUMO's *NETEDIT* is also used for manual tweaks. In [33], the "Path and road surface module" in the *Carsim* and *Trucksim* software converts the road geometry to a model.

Table 3.4: Infrastructure: Data processing

Infrastructure data processing	References
Modeling software add-ons	[33], [40], [35], [42], [43]
Other Software	[40], [35]
Custom method	[53], [41], [42], [43]

Software add-ons seems to be the most common way to convert a large-scale area into the specified network. In fact, [40], [47], [35], and [42] all use OSM and rely on SUMO's *netconvert* and *netedit* for processing. [43] and [33] also employ TRANSIMS's *TransimsNet* and *Carsim/Trucksim*'s "Path and road surface", respectively, to expedite work. The custom methods used in [41], [42], and [43] use additional processes to fix mistakes or shortcomings of the automated software. [43] and [42] make use of the complementary information besides the map itself that is referred to in Table 3.2.

Traffic: data sources

Table 3.5: Traffic: Data sources

Traffic data sources	References
Live	[33], [48], [51], [40], [47], [52], [1], [59], [49][50]
Recorded	[34], [44], [45], [35], [41] [42], [43], [1], [49], [46]
Static camera footage	[45], [48], [52], [59], [46]
Drone footage	[34], [33]
Internet of Vehicles	[44], [51], [59]
Internet of Things	[48]
Traffic counters	[40], [47], [41], [1], [49], [46], [50]
Distance sensors	[52]
Demographic data	[42], [49]
Surveys	[35] [43]
Personal device data	[40], [47], [35], [42]
Entities	[41], [42], [1]
Other	[51], [41]

The *Live* and *Recorded* rows in Table 3.5 show if the live data can be used during the simulation. Even though [1] is prepared to run with an online data stream, lack of performance prevents it from being a truly live simulation. Other publications, even if they do not implement the proposed methodology in a live environment, are designed with that in mind, so they are included. An exception to the rule is [49], which uses both recorded data as an initial input, on top of which live data is also used.

There are a lot of different sources of data when it comes to traffic in different studies according to their purpose. The entity in [41] and [42] is a public transit company. In [1], state entities, a highway agency, taxi fleets, and a GPS company all aid in providing data. The Internet of Vehicles (IoV), the Internet of Things (IoT), and distance sensors are used in studies that do not present a standard simulating software and that develop their own methodologies to either aid [51], [44] in DTs development, or have an approach that, if implemented, would rely on custom software [48], [59], [52]. Traffic counters are widely used. This seems to be a good tool as the data coming from these sensors is easily digestible by the existing simulators and does not require the complex processing that would be required if the same information were to be obtained by

cameras and computer vision. The examples citing drone footage do employ such methods [34], [33], particularly SIFT and Yolov5. It is observable that the examples of static camera footage also do so, but in general, their methodology is vague. The footage in [45], however, does not employ computer vision at all (see Table 3.7). The remaining data sources are similar as they mostly attempt to gather insights into an individual's habits and location history. These sources are mostly to inform demand (see Table 3.6).

Traffic: data information

Table 3.6: Traffic: Data information

Traffic data information	References
Traffic behaviour	[34], [45], [1]
Trajectory	[34], [33], [1], [46]
Velocity	[34], [33], [41]
Position	[44]
Vehicle type	[34], [48], [40], [41], [42]
Driving Behaviour	[34]
Traffic flow	[40], [47], [1]
Traffic demand	[35], [41], [42], [43], [1], [49], [46], [50]
Other	[48], [51], [42]

Trajectory, velocity, and position are all metrics that correlate with vehicles on the road. It makes sense that they would be used to aid in DT development. Traffic flows are obtained through traffic counters. It would be expected that the other work referred in Table 3.5 would also use this information. It is likely that that is indeed the case, but no explicit textual reference to it was found, hence its non-inclusion. Traffic demand is a major and complex information that is used mainly for city-wide research. Different papers present different methodologies to come up with this demand, often in the form of an Origin-Destination (OD) matrix, and that is why there are multiple complementary data sources in the publications that use this information (see Table 3.5). Vehicle type is also useful as not only different vehicles have widely different characteristics. Particularly, they are relevant for emissions [41] and multi-modal simulations [42]. Traffic behavior in [45] is the only information used and represents the travel time between points, the maximum queue entrance, and the number of immobilization at the entrance. In [34], it represents the type of road segment as either acceleration, deceleration, or constant speed road segment. Finally, in [1] gridlock are detected. In [48], temperature, humidity, and air quality are expected to be recorded. Additionally, the system aims to store data frames associated with each vehicle. In [51], navigation

data from the vehicles is transformed into a weaving or non-weaving intention for the work in lane-changing assistance. [42] also uses public transit information. Drive behavior in [34] represents lane changing and car following parameters.

In general, the scale of the simulation is correlated to the scale of the information. Research into city-wide simulators makes use of traffic demand and traffic flow. Studies regarding a road segment do not need this information as they are primarily concerned with specific vehicles' position, velocity, or trajectory.

Traffic: data processing

Table 3.7: Infrastructure: Data processing

Infrastructure data sources	References
Computer Vision	[34], [33], [48], [46]
Manual tolling	[45]
Custom method	[48], [40], [47], [42], [59], [1], [49]

As referred in Table 3.5, computer vision is achieved in [34] and [33] through SIFT and Yolov6. [48] Uses Google API for plate detection and recognition. [46] uses a data source that appears to have used video processing that is assumed to have been through computer vision. Only [45] uses manual tolling of camera footage. The small scale of the work makes it viable, but that obviously would not be the case for a digital twin. There is a wide variety of custom methodologies. [48] proposes a data fusion algorithm for the various sources. [40] and [47] use a system of linear equations for predicting traffic flows in segments for which there is no data. In [59], the need for translation from the source format to a model format is mentioned. [42], [1], [49] relate to how they achieve the respective traffic demands.

3.1.3 Modeling

Software

Table 3.8: Modeling: Software

Modeling: Software	References
PTV Visum	[41]
PTV Vissim	[34], [45], [46]
MATLAB Simulink	[33]
Carsim/Trucksim	[33]
Prescan	[34]
SUMO	[51], [40], [47], [35], [42], [1], [49]
TRANSIMS	[43]
Other	[59]

From the analyzed literature, authors whose intent is to achieve a simulation of a DT of traffic use an already developed software with the exception of [59], who proposes a custom model based on Virtual Vehicle models, which are, in and of themselves, a DT. Amongst the existing software, SUMO appears to be the most prevalent.

Complementary software

Table 3.9: Modeling: Complementary software

Modeling: Complementary software	References
Tracking	[44]
Communication Network	[51]
Traffic management	[51], [40], [1], [50]
Driver Behaviour	[40], [47], [45], [35], [46]

In Table 3.9 are present not only add-ons of theoretical models that were explicitly referenced in the work analysed, but also software that was developed as the purpose of their respective work. For example, [51] developed a Lane-changing and Variable Speed Limit Control that is tested within the context of a microsimulation. Additionally, it uses a communication network software, OMNet++, to simulate the communication between vehicles in the IoV. Similarly, [44] develops a tracking algorithm from the position of vehicles connected to the IoV. [50] uses DynaMIT, a

Table 3.10: Modeling: Space Granularity

Modeling: Space granularity	References
Micro	[34], [33], [45] [51], [40], [47], [35], [42], [43], [1], [59], [49], [46]
Meso	
Macro	[41]

Dynamic Traffic Assignment (DTA) software to test its own methodology for DTA. Driver behavior relates to the models used for the drivers in the studie. [46] and [45] use Wiedemann's car following model. [35] uses Krauss's car-following and Erdmann's lane changing. [40] uses the enhanced intelligent driver model for car-following and LC2013 as its lane changing model. As for traffic management, both [40], [47], and [1] use SUMO's TraCI to interact with the simulation during its execution.

Space granularity

For the work presented in Table 3.8, the granularity is simply the granularity of the software. Therefore, they all present a microscopic granularity, but for [41] who developed a macroscopic model. [44] is also listed as a microscopic simulation as it heavily correlates to a simulation of such nature since it intends to track the movements of a particular vehicle.

Input parameters

Table 3.11: Modeling: Input parameters

Modeling: Input parameters	References
Simulation resolution	[45], [40], [35]
Drivers' behaviour	[34], [45], [51], [40], [35], [59], [46]
Vehicle type	[34], [40], [47], [35], [41], [42], [59], [46]
Position	[44], [59]
Speed	[34], [33], [40], [47], [41], [59]
Maximum speed	[45], [40], [47], [41]
Trajectory	[34], [33]
Intended route	[51]
Traffic flow	[40], [47], [49], [46]
Traffic demand	[45], [35], [41], [42], [43], [1], [49], [46], [50]
Other	[41], [59]

A large amount of the models listed use traffic demand. Only [45] is a special case, as the demand is the number of proceeding vehicles for the represented roundabout. For the rest, traffic demand is on a city-wide level and guides the simulation by informing the software to create the vehicles accordingly. For this city-level work, often, it is the only information needed, or at least specifically stated. Additional information, mainly the vehicle type, driving behavior (driving model), or speeds, adds to the richness and is an attempt to approach reality. [41] also uses public transport distribution and meteorological information. [59] uses communication network data and also weather information.

Output parameters

Each of the studies on DTs or other similar models results in certain outputs that we shall now enumerate.

Table 3.12: Modeling: Output parameters

Output parameters	References
Travel time	[45], [35], [42], [49], [46]
Fuel consumption	[45]
Emissions	[45], [41]
Risk	[34], [33]
Position	[51]
Speed	[41]
Trajectory	[34], [44]
Traffic flow	[45], [40], [47], [35], [41], [42], [43], [1], [46], [50]
External feedback	[51], [52], [59]
Other	[35], [59], [49]

External feedback comes in many forms, though it is either idealistic or only achieved in the simulation. In [59], each vehicle communicates traffic events to other connected vehicles in the IoV. [52] proposes signal lights, dynamic route scheduling, and vehicle cooperation be derived from the simulation. [51] talks about lane changing permission and position to be given to vehicles in the IoV. Traffic flow and travel times are the most usual output parameters. These are normally interpreted for the whole simulation and are used as a benchmark. As for risk, [34] examines the collision risk as a result of the trajectory output information. [33] calculates sideslip risk, collision risk, and trajectory deviation.

In general, the output parameters are aligned with the specific purpose of the work. Work that intends to analyze a particular phenomenon will have parameters such as risk, emissions, or fuel consumption. Work with a broader scale will resort to traffic flows, travel times, or speeds.

3.1.4 Calibration

Table 3.13: Calibration

Calibration	References
No calibration	[35], [42], [43]
Custom methodology	[42], [43], [1]
Iterative experimentation	[35], [49]
Neural Networks	[45], [51], [46]
Literature methodology	[51]
Improvement of Literature Methodology	[47], [35], [50]
Inferred from real data	[35], [42]

Calibration of parameters is done in varied ways. The main areas are represented in Table 3.13. As for the work that has parameters that are not calibrated, [35] uses the default car-following model. [42] assumes values for its utility function parameters. [43] performs no parameter calibration due to too expensive computations. Inferences from real data are possible, such as in [35], which takes car length from known data of the city simulated. Similarly, [42]’s availability of mode of transportation is based on statistical data. In [35] In-Simulation Adaptive Rerouting and Iterative Incremental Dynamic User Equilibrium are developed to improve Iterative Incremental Dynamic User Equilibrium and Dynamic User Assignment in their use for calculating routes chosen. [50]’s purpose is to improve the Extended Kalman Filter as applied for online calibration of demand. For the iterative approaches, [46] uses Monte Carlo to calibrate ten undisclosed parameters. In [35], the best different rerouting probabilities through roads with different priorities are tested by iterating to search the two-dimensional space of possible values. [45] uses Neural Networks as a black box in order to expedite the calibration of driver behavior’s parameters by functioning as a black box replacement of the simulation. The Variable Speed Limit control add-on in [51] is implemented through neural networks as well.

In conclusion, parameters inferred from real data are an advantage as they leave less margin for error. Choosing well which parameters will not be calibrated saves time in calibration and allows for the study of the particular method being developed.

3.1.5 Validation

Table 3.14: Validation: Methodology

Validation: Methodology	References
Comparison with real-world data	[40], [47], [35], [41], [42], [43], [49], [46], [50]
Comparison with base models	[44], [35], [59]
Comparison between developed models	[51], [35], [41], [50]
Comparison with other models	
Ablation study	[44], [51], [50]
Time performance	[44], [49], [50]

Table 3.15: Validation: Parameters

Validation: Parameters	References
Travel times	[45], [35], [46]
Traffic behaviour	[45]
Speed	[51]
Traffic flow	[40], [47], [35], [42], [43], [49], [46], [50]
Identity	[44]
Emissions	[41]

When it comes to validation, particularly in the context of DTs, time performance is of the utmost importance as it defines the viability of the implementation. Oddly enough, not every Digital Twin study uses this metric.

Validation against base models, between developed models and ablation studies, is a good way to provide evidence of causation and not only correlation. Though combining the three might be ideal, it might imply extra implementation effort that the authors might not have had available. Nonetheless, there is a lack of validation against other models. The fact that each work relies on different data and that replication might be troublesome without the insights of the authors themselves make this a hard thing to achieve. In the future, more effort should be made to compare different modeling approaches in order to solidify the field. Validation against real-world metrics will always be essential, as the comparison with other models (developed by the authors or others) can only be fruitful if in comparison against what the simulation intends to replicate. Most of the work analyzed does indeed do this, focusing mainly on traffic flow [50], [43], [42], [35], [40],

[49]. Travel times [46], [35] and speeds [49] are also used. [41] compares against emissions as that is the purpose of the study.

3.1.6 Gap analysis

By performing this analysis, it becomes evident that there is an unbalance in the number of studies regarding the methodologies used for the simulation.

First, there is an abundance of approaches based on traffic demand, particularly in the form of OD matrices, which focus on the calibration of traffic before the start of the simulation and, as such, lack the flexibility for real-time calibration. Moreover, seldom is the calibration of driving behavior considered; and rarely is Machine Learning used. Though not tackled in this work, the developed simulations hardly ever have their models applied in outside systems and have a more scientific scope. Another note is that when simulations regarding a smaller area are developed, they tend to study a key performance indicator of interest.

3.2 Summary

To summarize, though there are various publications on the simulation of road networks, most of them do not attempt to achieve a DT as they do not achieve automated data flow, either to the simulation or back to the real world. However, there is promising work that is already seeking to develop this technology by considering online data, even if they lack actual implementations, and some studies have achieved the usage of real-time data.

Most publications attempt to develop or implement methods to correctly simulate the traffic in an area by calibrating the traffic demand during the simulated time period. For this, the use of OD matrices is recurrent, and other data sources, such as GPS traces, can be used as a complement. SUMO is a widely used software for the simulation which will then be evaluated by comparing the traffic flows to real recorded traffic flows.

Chapter 4

Simulator specifications

SUMO is a state-of-the-art microsimulator and is taken as a reference throughout this work. At its core, all of the information is kept as XML (Extensible Markup Language) files and can then be imported, used, modified, ran, or saved. There are some key features that are important to bear in mind in order to better understand the developed methodology.

4.1 Network

SUMO's simulation uses a network with its own elements, which include junctions, edges, lanes, calibrators, and induction loops. Junctions are connected by edges which can be made of 1 or several lanes. Induction loops and calibrators are referred to as additional, which are extra elements responsible for the interaction or recording of traffic data.

The network and the additional are both XML files, the former with the extension *.net.xml*, and the latter with the extension *.add.xml*

4.2 Simulation

The simulation then happens on a step-by-step basis, where each step has a duration that is recommended to not exceed 1 second ¹. The simulation can be initialized through a configuration file ², which too is an XML file with the extension *.sumocfg*. It is used to define the network, additional, and routes by defining the files where said information is located. In [Listing A.3](#) there is an example configuration file. All of the files used in the configuration are XML files as described before. The routes file as the extension *.rou.xml*. Additionally, the begin and end steps can also be indicated.

¹https://sumo.dlr.de/docs/Simulation/Basic_Definition.html#defining_the_time_step_length

²<https://sumo.dlr.de/docs/sumo.html>

During run time, vehicles will attempt to follow their designated route, which is a sequence of edges. Each vehicle has a given vehicle type³ which influences the vehicles physical characteristics and driving behavior. Vehicles, by default, use a car-following model based on Krauss's⁴ and a custom lane-changing model⁵.

4.3 Induction loop

An induction loop⁶ is an additional that is placed on a lane and that gathers measurements on the traffic that passes through it in a given interval. The measurements gathered are the number of vehicles (interval flow), the hourly flow, the percentage of time it was occupied by a vehicle, the arithmetic and harmonic means of the speeds (in m/s , the arithmetic mean of the lengths of the vehicles and the number of vehicles that touched the induction loop (but that did not contribute to the measurements). The definition of an induction loop requires the id, the position in and the id of the lane where it is located, the periodicity with which it should record the data, and the file to which said data should be saved.

Below there is an example of a definition of an induction loop from the experiment (see Chapter 6):

```
1 <inductionLoop id="e_98835987_0" lane="98835987_0" pos="135.505" period="300.0"
  file="counters_output.xml"/>
```

Listing 4.1: Example induction loop definition

4.4 Calibrator

A calibrator⁷ is an additional that can either be placed on an edge or in a lane whose main purpose is to control the vehicles that pass through it during the simulation. It can define the flow (number of vehicles passing), the route, and the type of vehicles. The definition of a calibrator requires the id, the position, and the element where it is placed. Additionally, to be active, a calibrator must have at least an associated flow element that defines the period when it is active and what calibration it performs. A flow must have defined the interval (with a begin and an end times) and at least one of the attributes speed, vehsPerHour, and type. Below there is an example of a definition of a calibrator from the experiment (see Chapter 6):

```
1 <calibrator id="ca_98835987" edge="98835987" pos="0.0" output="
  entry_calibrators_output.xml">
```

³https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html#vehicle_types

⁴<https://sumo.dlr.de/docs/Car-Following-Models.html>

⁵https://elib.dlr.de/102254/1/Springer-SUMOs_Lane_changing_model.pdf

⁶https://sumo.dlr.de/docs/Simulation/Output/Induction_Loops_Detectors_%28E1%29.html

⁷<https://sumo.dlr.de/docs/Simulation/Calibrator.html>

```
2   <flow begin="0" end="3600" speed="22.22"/>
3 </calibrator>
```

Listing 4.2: Example calibrator definition

4.5 Runtime APIs

SUMO has two available APIs that allow for the interaction with the simulation in real time: the Traffic Control Interface (traci)⁸ and libsumo⁹. These allow, for example, to control the flow of the calibrators, the routes of the vehicles, the properties of the vehicle types, and even to fetch information about said elements. All of these functionalities are used in this work and can be found throughout Section 5.3. As recommended by the SUMO documentation¹⁰ traci was initially used for testing, and afterward sumolib to achieve better performance. Consider that any mention of traci is interchangeable with sumolib throughout this report.

⁸<https://sumo.dlr.de/docs/TraCI.html>

⁹<https://sumo.dlr.de/docs/Libsumo.html>

¹⁰<https://sumo.dlr.de/docs/TraCI.html>

Chapter 5

Methodological approach

5.1 Problem formalisation

Idealistically, the objective of a Digital Twin is to replicate the state of a road network. The network is considered to be a highway ring road with only uninterrupted traffic, and the intersections between the roads consist of entry and exit ramps. The methodology and implementation might be applicable to other networks with no traffic lights and intersections are all of the types described in Section 5.1.1. Similarly, an implementation in SUMO is assumed, but other simulators can be used in an analogous manner.

In this work, we assume we have a limited set of data, and that exact replication is not possible. As such, the scope of the simulation shall be limited to approximating the data collected from the simulation to real-world data while inferring an accurate vehicle model, which includes driving behavior and physical properties. The simulation data to be compared can be either the data automatically collected by SUMO, or a specific implementation can be made to collect that data. In this work, only measurements that are either automatically outputted or accessible through traci are used. It is possible to modify the implementation to accommodate other measurements, for example, the number of vehicles in a certain range of another, but it will not be analyzed. The driving behavior refers to the parameters of SUMO that control vehicle-level decision-making, which characterizes SUMO as a microscopic simulator. The car-following, lane-changing and also inherent properties of the car, such as length and width, fall under this scope.

5.1.1 Network

It is important to describe a network abstraction that represents SUMO's implementation of a road network that is based on a graph. Some concepts that will help discuss the methodology will also be introduced.

Let N be a road network and G the underlying non-planar directional weakly-connected graph G where the maximum in-degree and out-degree is two, V its vertices (or nodes), and E its edges.

Edges:

Each edge $E_{i,j}$ has a source and a destination vertex V_i and V_j and represents a segment of or a full road.

An edge is characterized by a physical shape, the maximum speed, the type, and its lanes. An edge can have one or more lanes E_{i,j_0} to E_{i,j_n} and for which lane changing authorizations can be specified, which determine whether a vehicle can change to the left or right lanes.

Incoming and outgoing edges are also associated with each edge. Analogously for lanes.

Vertices:

Each vertex V_i can represent an intersection of roads, a change in the properties of a single road (maximum speed, type, number of lanes, lane changing authorizations), or be purely a conceptual division in the road.

Similarly to edges, vertices also have incoming and outgoing edges. The incoming edges of V_i are all $E_{j,i}, j \in V$.

The interactions between edges can create the following:

Connected (or adjacent) edges:

Two edges are connected if $E_{a,i}$ is one of $E_{i,b}$ incoming edges and $E_{i,b}$ is one of $E_{a,i}$ outgoing edges.

Merge:

There is a merge when $E_{a,i}$ and $E_{b,i}$ are connected to $E_{i,c}$.

$E_{i,c}$ is considered a merge edge and V_i a merge vertex.

Split:

There is a split when $E_{a,i}$ is connected to $E_{i,b}$ and $E_{i,c}$.

$E_{a,i}$ is considered a split edge and V_i a split vertex.

Dead-end:

There is a dead end when none of the incoming edges of V_i have as outgoing edges any of the outgoing edges of V_i . A dead-end is always either an entry, an exit, or both.

V_i is considered a dead-end vertex.

Entries

An entry is such that V_i is a dead-end vertex and V_i has outgoing edges.

V_i is an entry vertex and any $E_{i,j}, j \in V$ is an entry edge.

Exits

An exit is such that V_i is a dead-end vertex and V_i has incoming edges.

V_i is an exit vertex and any $E_{j,i}, j \in V$ is an exit edge.

Path

A path is a list of edges such that every two subsequent edges are connected, and none of the edges but the last can be a split edge (for simplicity, parts of a path, where all edges but the last are not split edges and are subsequently connected, are not considered paths; meaning every path is as big as possible without breaking any of the conditions).

As a corollary, the first edge of a path is always either an entry edge or an outgoing edge of a split; and the last edge is either an exit edge or a split edge.

Continuity:

There is continuity when $E_{a,i}$ and $E_{i,b}$ are connected and $E_{a,i}$ is the only incoming edge of V_i and $E_{i,b}$ is the only outgoing edge of V_i .

In that case, V_i is a continuous vertex; $E_{a,i}$ and $E_{i,b}$ are two continuous edges; $E_{a,i}$ is the previous continuous edge of $E_{i,b}$; and $E_{i,b}$ is the next continuous edge of $E_{a,i}$.

Continuous segment:

A continuous segment is a list of edges such that every two subsequent edges are continuous, the first edge has no previous continuous edge, and the last edge has no next continuous edge (for simplicity, parts of a continuous segment, where all edges are also subsequently continuous, are not considered continuous segments; meaning every continuous segment is as big as possible without breaking any of the conditions).

As a corollary, the first edge of a continuous segment is always either an entry edge, an outgoing edge of a split edge, or a merge edge, and the last edge is either an exit edge, a split edge, or an incoming edge of a merge edge.

In [Figure 5.1](#) an example network displays all of the defined concepts. The letters represent the vertices whereas the numbers represent the edges. The flow of traffic is from left to right. In [Table 5.1](#), the most relevant concepts are labeled.

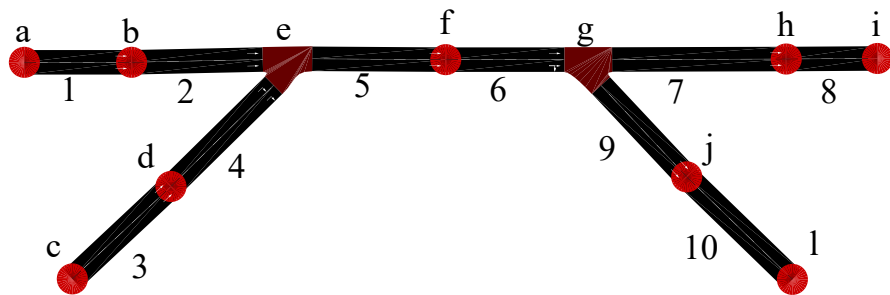


Figure 5.1: Example network

Table 5.1: Definable entities for the example network

Designation	Entities
Dead-ends	a, c, i, l
Entry vertices	a, c
Exit vertices	i, l
Entry edges	1, 3
Exit edges	8, 10
Merge vertices	e
Split vertices	g
Merge edges	5
Split edges	6
Paths	[1,2,3,4], [3,4,5,6], [7,8], [9,10]
Continuous segments	[1,2], [3,4], [5,6], [7,8], [9,10]

5.1.2 Traffic

Given the previously described network, N , the following concepts need to be introduced to interpret and describe the traffic. The descriptions below consider the experiment (see Chapter 6) and the available data and, as such, may not be a generalization of all possible scenarios.

Routes:

A sequence of edges of connected edges $[E_{\alpha_0, \alpha_1}, E_{\alpha_1, \alpha_2}, \dots, E_{\alpha_{n-1}, \alpha_n}]$, $\alpha_i \in V : E_{\alpha_i, \alpha_{i+1}}$ connected to $E_{\alpha_{i+1}, \alpha_{i+2}}$, $i \in [0, \dots, n-2]$

Vehicles:

Each vehicle $V_i \in V$ has an associated model M_i , which dictates its behavior within the network. The physical properties of the vehicle and driving behavior are part of the model. When present in the network, a vehicle will be assigned a route that it will follow until it reaches the end of the last edge and is thereby removed.

Time:

In the context of the simulation, time $t_i \in [0, \dots, t_{end}]$ is measured in seconds, is discrete, and is increased with each step. Each step corresponds to a certain amount (or fraction) of seconds. In this work, a step is assumed to be 1 second.

Sensor:

Each sensor $S_i \in S$ is located at a point P in an edge $E_{\alpha,\beta}$ or in a lane E_{α,β_a} , and represents the sensor device used to obtain traffic measurements. An edge with a sensor can be called a sensor edge.

Measurements:

Each sensor S_i can collect multiple types of data in each step, which can be aggregated as the data of a given period. Given a type of data represented by X , $X_i(t_a)$ is the data collected by sensor S_i at time t_a , and $X_i(t_a, t_b)$ is the data collected by sensor S_i during the interval $[t_a, t_b]$. Instead of a sensor, a measurement can also be collected in an edge $X_{i,j}$ or in a lane of an edge X_{i,j_k} . In particular, the following measurements can be defined:

- Q is the count of vehicles that pass through a sensor at either a point in time or in an interval.
- F is the flow of vehicles and represents the interpolation of the respective Q such that F is measured in veh/h (Example, given $Q(i, j) = x$, $F(i, j) = \frac{60}{j-i} * x$).
- v is the average speed of vehicles that pass through a sensor at either a point in time or in an interval. If no vehicles have passed the sensors within that time it defaults to 0.

The previous measurements are the ones focused on the methodology, though any others can also be used. We consider the set of measurements X (or any measurement) as being the simulated data and X' (or any measurement) the real data.

Additional information

Any additional information I that can be correlated to traffic (for example, weather information, current events in the city, etc.).

5.1.3 Objective

Let $T \subseteq X$ be the target measurements, and $K \subseteq S$ be the known sensors for which real data is available. The objective is to minimize the difference between the target measurements of the real data and the simulated data at every point in time. In general:

$$\min \sum_{t_k=0}^{t_{end}} \sum_i T_i(t_k) - T'_i(t_k), S_i \in K \quad (5.1)$$

For this work, considering flow and average speed as the target measurements:

$$\min \sum_{t_k=0}^{t_{end}} \sum_i (F_i(t_k) - F'_i(t_k)) + (v_i(t_k) - v'_i(t_k)) \quad S_i \in K \quad (5.2)$$

Additionally, we intend to describe a strategy to attribute to each vehicle a model M_i .

The following information is known:

- The network N and its underlying graph G .

- $X_i(t_a, t_b), S_i \in S$, all simulated measurements of the sensors.
- $T'_i(t_a, t_b), S_i \in K$, real measurements of the known sensors.
- I , additional information.
- The position and state of every simulated vehicle V_i .

The following control can be exerted:

- Insertion of vehicles at any point P .
- Removal of vehicles at any point P .
- Choice of the route of any vehicle V_i .
- Definition of M_i for any vehicle V_i .

5.2 Materials

There are two essential data components when it comes to designing a simulation of this nature:

- Network data.
- Traffic data.

The network data must be sufficient to create a model of the network in the chosen software. It can be complemented with other data sources.

For this work, the main source of data was OpenStreetMap (OSM), whereas Google Maps was a complimentary one. The first one was used as the primary source from which the network was built in SUMO. The satellite view from Google Maps was used to fix any inconsistencies in the network and to better precise the position of the sensors.

As for the traffic data, the more sensors there are available, the more complete the information about the traffic is. The data must contain the location of the sensors, precise enough that a specific position in the edges of the network can be associated with the sensor (see Section 5.3.2), and at the very least the targets measurements, T' . Any additional measurements, X' , for example, the vehicles by type, their length, and width; or information (weather, pressure, road friction) can also be used. The methodology assumes that said sensors collected data in regular intervals. In this specific case, the periodicity was five minutes.

5.3 Methods

The methodology presented in this work is based on the methodology in [40], and also draws inspiration from [47]. Additionally, it tries to tackle some of their shortcomings while proposing a new Machine Learning approach for the calibration of the simulation.

The implementation was done in Python ¹ and it is the language used for example code. On a technical note, the code is purely illustrative and might not work as is. Keys of dictionaries are objects, and it is recommended the keys are instead the IDs of the objects they represent. For clarity and brevity of the algorithms described this was not demonstrated. Additionally, no specific technique is recommended for serialization and deserialization. Pickling ² might require that any properties or references to objects derived from sumolib (mainly the Edge objects) be deleted. Instead, the ids of said objects should be enough to reconstruct the information upon deserialization.

5.3.1 Network

In order to obtain the target network in SUMO there are three steps involved:

- Obtain an encompassing network that includes the target network;
- Obtain the target network;
- Calibrate the target network.

Obtain an encompassing network

OSM is used to obtain the encompassing network. The OSM platform allows one to export the selected area into .osm files. These are XML files which hold the information of the road network. A short example of a section of these files can be seen in [Listing A.1](#).

Due to limits on the amount of nodes that can be exported, multiple selections should be made and downloaded. There is no problem if overlaps occur. The files should be stored inside a single folder.

Next, the netconvert tool ³, which is part of the SUMO environment, is used to assemble a network that the software can read. A script handles the conversion by invoking the tool. It takes a folder (where the .osm files are stored), a configuration for the conversion, and the output file. The configuration used can be seen in [Listing A.2](#).

It should be noted that the original names and the street names are kept. This will ease the process of altering the network to meet our needs. Additionally, we are removing the isolated edges since these cannot be part of the network. Junctions are also joined as recommended by the official documentation when importing from OSM.

¹<https://www.python.org/>

²<https://docs.python.org/3/library/pickle.html>

³<https://sumo.dlr.de/docs/netconvert.html>

Obtain the target network

The initial OSM selected may include other roads that are not in the target, a problem that may vary in severity depending on the size and type of the network, the network, and the precision of the selection. As such, it could be assumed that in the majority of the cases, the target network is a subnetwork but not the network obtained in the previous step. It also means that extra space and computing power would be used hereon both for acquiring the rest of the network information (see Section 5.3.2) and during the simulation itself (see Section 5.3.3). To obtain just the target network, it is advised that we perform a filtering process that shall now be described, the result of which is a network with only the edges that are to be simulated. The netconvert tool and a library to interact with the network are recommended for filtering. In this case, the python sumolib⁴ was utilized as it provides an easy way to get information on the edges, vertexes, and how they are connected to each other.

Because we are working with the very concrete scenario of a ring highway, the strategy for the filtering was fine-tuned to this case. Alterations or modifications to the pipeline might be required according to the specificity of the target network and how much it deviates from this paradigm. The filtering's objective is to acquire all of the edges that shall be present in the final network, by recording their ids. First, the names of the relevant roads should be collected from OSM. In the scenario of highways, these are likely to be long-spanning stretches of road and, therefore, require little manual work, depending on the overall network size. Second, since the names were kept in the first use of the netconvert, using sumolib, we can save the ids of only the edges whose name belongs to the names of the selected roads.

This step can be applied iteratively until we are satisfied with the network. Get a list with the names of the edges, filter the network as described, visualize it using netedit⁵, and change the list of edges. Names of roads can be added or removed accordingly. A complementary list of ids can be used. This list allows adding edges for roads that happen to not have a name or to solve the problem of a road name, including edges that both should and should not be in the network.

Additionally, to increase the automation of this process and not rely on the manual name or ID selection, another selection criterion was implemented by using the type of edge that is stored by the .osm files and also kept in SUMO's network. Therefore, This step focuses on adding to the edges already selected in the previous step other edges adjacent to those that match some designated types. Selecting adjacent edges is a recursive process that is done a number of times equal to the specified depth. For example, for a depth of one, only the immediately adjacent edges shall be added to the selected edges. For a depth of two, the initial edges, the adjacent ones, and also the ones adjacent to the union of the first two sets are added. This procedure follows analogously for any depth or for an infinite depth (edges are added until no more different adjacent edges are found).

If multiple types of edges are selected with varying degrees of adjacency and if the selected edges are cumulative (the adjacent edges that are found are added to the selected edges after each

⁴<https://sumo.dlr.de/docs/Tools/Sumolib.html>

⁵<https://sumo.dlr.de/docs/Netedit/index.html>

operation), the order in which the operations are performed can affect the final result. If, for example, consider that edges of type `highway.primary` with depth 1 and type `highway.secondary` with depth 1 are chosen to be added in a network given by $E_{0,1}$ adjacent to $E_{1,2}$ (`highway.primary`), in turn adjacent to $E_{2,3}$ (`highway.secondary`) where $E_{0,1}$ is already selection. If first the adjacency for `highway.secondary` is computed and then `highway.primary`, the final edges will only be $\{E_{0,1}, E_{1,2}\}$. On the other hand, if first adjacency for `highway.primary` is computed and then `highway.secondary`, all edges are included. As can be seen, the order of operations matters. However, since the requirements of other studies might be different and updating the selected edges after each operation might be needed, might not be needed, or might just be needed for certain configurations, a single implementation of this algorithm is naive. Instead, an example implementation can be seen in Section 6.1 (see Listing 6.1) that can serve as the base for others.

Finally, there is the possibility that edges outside of the target network also got added (for example due to OSM having incorrect road names). To prevent these edges to be included in the final network, we consider that the network is made up of only one weakly-connected graph. Luckily, `netconvert` allows us to extract only the biggest connected component by setting the attribute `-keep-edges.components` to 1. If the used software does not offer this functionality, a custom implementation must be performed to further restrict the selected edges to those belonging to this graph.

Calibrate the target network

After these two steps, we should have correctly obtained the target network. However, despite the collaborative efforts of OSM aims to minimize inconsistencies with the real world, it is not guaranteed that the characteristics of the edges are correct. Some aspects to consider are therefore:

- Speed.
- Number of lanes.
- Geometry (the physical shape of the lane that will affect the behavior of the microscopic mode).
- Connections with other edges (define, at a vertex where two or more edges intersect, from which lane to which vehicles can move).
- Continuous lines.
- Width.
- Type of vehicle allowed.

While a simple edition of those properties might be sufficient, it might be necessary that edges are manually partitioned into two or more edges so that speeds, number of lanes, or continuous lines can be adjusted for each of the edges. It is recommended that these editions (but for changes

to the continuous lines) are performed using the graphical software netedit since the edition of the XML file might prove more difficult.

Though continuous lines are saved when converting the osm files, the version of netedit used (version 1.19.0) did not properly save this information when the `changeLeft` or `changeRight` properties of a lane were edited in the graphical application. It is unknown if this was fixed in later versions. Therefore, in order to edit the continuous lines of the network it is necessary to access the `.net.xml` file manually and introduce or modify the attributes `changeLeft` and `changeRight` of a given lane. This attribute defines the types of vehicles that can take the described lane change and, as such, the reserved vehicle type *authority* is recommended to be used. Therefore, if, for example, to the left of a lane there is a continuous line, in the XML element pertaining to that lane it should be added the text `changeLeft="authority"` and to the lane immediately left (that presents that same continuous line but to the right) `changeRight="authority"`. It is also possible to have changes of lane allowed just in one direction by adding the changing restriction just on one of the sides. Henceforth, the target network shall be known as just the network.

5.3.2 Network information

In order to run the simulation, SUMO requires the following files.

- Config file: The file with which SUMO is initialized takes all other files as input. Also defines other information such as the begin, end and duration of each simulation step.
- Net file: The network file developed in Section 5.3.1.
- Route files: The files with the routes each vehicle can take. Each route is defined by an id and a sequence of edge ids.
- Additional files: There are many additional files that can be used⁶. For this work, we focus on calibrators, rerouters, and induction loops.

However, the methodology requires other files and information that are essential for the extension of the functionalities of the simulator. These files can be created by the Python scripts that were developed. Overall, we shall explain and discuss the following files:

- SUMO edges' induction loops (additional file).
- SUMO edges' routes.
- SUMO edges' entry calibrators (additional file).
- Sensors information.
- SUMO sensors' induction loops (additional file).
- SUMO sensors' calibrators (additional file).

⁶<https://sumo.dlr.de/docs/Netedit/elementsAdditional.html>

- Sensor calibrators information.
- Routes information.
- Edges' routes information.
- Rerouters' routes information.
- SUMO vehicles routes (additional file).
- Reachable edges information.
- Independent systems network (network file).
- Independent systems equations.
- Additional information.
- SUMO configuration file (configuration file).

SUMO files are included in SUMO's simulation configuration file whereas the others are loaded by the simulation framework implementation. The `sumolib` library is used to interact and obtain the network's information ^{7 8}, and the `lxml` library ⁹ is used to create and change the XML files required by SUMO.

SUMO edges' induction loops

Each and every one of the lanes in the network shall have an induction loop at its mid-position. The mid position is chosen as it is assumed to give the best representation of the traffic on the lane, as either the start or the end of the lane may be influenced by traffic attempting to change directions (vehicles in the simulations trying to change lanes and edges due to a split or a merge). The purpose of these induction loops is to provide simulation data that the machine learning models can use in their predictions (see Section 5.3.3).

An induction loop element specification can be seen in Listing 4.1. `Sumolib` is used to access the network and iterate through every lane of every edge. For every lane, its id and mid-position are used, whereas the output file and the periodicity are the same for every induction loop. The name given to the output file was `counters_output.xml` but any other is appropriate as long as it is not shared with another SUMO additional output file. The periodicity is equal to the calibration period (see Section 5.3.3), which, for this work, was 300 seconds but could vary according to the available data (see Section 5.2).

Given a lane id *id*, a mid-position *pos*, a period *period* and considering `lxl` represents the value of variable *x*, the induction loop XML element can be seen in Listing 5.1.

⁷<https://pypi.org/project/sumolib/>

⁸<https://sumo.dlr.de/docs/Tools/Sumolib.html>

⁹<https://lxml.de/>

```

1 <inductionLoop id="e_|id|" lane="|id|" pos="|pos|" period="|period|"
2   file="counters_output.xml"/>

```

Listing 5.1: Definition of a SUMO edge's induction loop

A file is created with all of the edges' induction loops.

SUMO edges' routes

Every entry edge in the network has an associated route which is composed of just the respective edge. These routes are created so that any calibrator in the network can be correctly initialized by having its route set to the one corresponding to the edge where it is located. Even though there are a limited number of calibrators in fixed places, setting a route for every edge prevents having to calculate what those places are and makes this step almost trivial.

Just like with the SUMO edges' induction loops, every edge is iterated with `sumolib` to collect its id. Given the edge id *id* and considering $|x|$ represents the value of variable *x*, the route XML element can be seen in Listing 5.2.

```

1 <route id="r_|id|" edges="|id|"/>

```

Listing 5.2: Definition of a SUMO's edge route

A file is created with all of the edges' routes.

SUMO edges' entry calibrators

Every entry edge in the network has a single calibrator placed at position 0 of the edge. These entry calibrators will be responsible for inserting vehicles into the simulation (see Section 5.3.3).

A calibrator element specification can be seen in 4.4 Akin to SUMO edges' induction loops and routes, `sumolib` is also used to iterate, this time through every edge, and for each of those, its id and speed are used. The end represents the end time of the simulation and, if not known exactly, can be set to a large number.

Given an edge id *id*, a speed *speed*, and an expected end time *end*, considering $|x|$ represents the value of variable *x*, the calibrator XML element can be seen in Listing 5.3.

```

1 <calibrator id="ca_|id|" edge="|id|" pos="0.0">
2   <flow begin="0" end="|end|" speed="|speed|"/>
3 </calibrator>

```

Listing 5.3: Definition of a SUMO edge's entry calibrator

A file is created with all of the edges' calibrators.

Sensors information

Before the SUMO sensors' induction loops and calibrators files can be created, it is necessary to compute where the respective additional (induction loops and calibrators) will be placed in the network. It is assumed that for each sensor device, the corresponding geographical location (longitude and latitude) is known (see Section 5.2). It is also essential to know whether each sensor device is associated with just one or both of the traveling directions, as this affects which algorithm should be used to calculate the edge and position of the sensors in the network. Moreover, it also affects if further processing needs to be applied to the data before it can be used for the simulation. In this methodology, each sensor has a unique id. If no id is provided in the sensor device's data, an incremental id is appropriate. If, additionally, each sensor device covers two directions, two different sensors must be fabricated, and their ids can be trivially obtained by adding a suffix to the original id (see Section 6.2).

The algorithm to calculate the edge and position of a sensor when a sensor device covers just one traveling direction can be described by the following steps:

- Get the geographical coordinates of the sensor device
- Convert the geographical coordinates to SUMO's internal coordinate system (x,y) to obtain the location of the sensor
- Using SUMO, get the closest edge to the location of the sensor
- Using SUMO, get the closest position on the edge to the location of the sensor
- Store the edge and position

A possible implementation is described in Listing 5.4.

```

1 def calculate_sensor_position(network:Net, sensor: tuple[float, float]):
2     lon, lat = sensor
3     x, y = net.convertLonLat2XY(lon, lat)
4     start_radius = 1
5     closest_edge = None
6     while closest_edge is None:
7         possible_edges = net.getNeighboringEdges(x, y, start_radius)
8         if len(possible_edges) == 0:
9             start_radius *= 2 # It is possible that the coordinates are not precise
10             enough so increase the search radius
11             continue
12         possible_edges.sort(key=lambda x: x[1]) # Sort by increasing distance
13         closest_edge = possible_edges[0]
14         closest_position = closest_edge.getClosestLanePosDist((x, y))
15     return closest_edge, closest_position

```

Listing 5.4: Implementation of sensor position calculating algorithm

For the case where each point represents two sensors, due to the distance that separates both directions of traffic, it is possible that two adjacent edges in the same direction are chosen. Imagine the trivial scenario where the location of the sensor is exactly in the location of a vertex, and two edges are equidistant from it and in the same direction. To account for this, we consider that the adjacent edges of the first sensor to be found are not the edge where the second sensor is. The following steps describe this algorithm:

Get the geographical coordinates of the sensor device

Convert the geographical coordinates to SUMO's internal coordinate system (x,y) to obtain the location of the sensor

Using SUMO, get the closest edge to the location of the sensor

Using SUMO, get the closest position on the edge to the location of the sensor

Store the edge and position

Create a list of excluded edges made of the found edge and its adjacent edges

Get the next closest edge to the location of the sensor that is not in the excluded edges

The following code describes a possible implementation.

```

1 def calculate_sensor_position_bidirectional(network:Net, sensor: tuple[float, float
2     ]):
3     lon, lat = sensor
4     x, y = net.convertLonLat2XY(lon, lat)
5     start_radius = 1
6     first_edge = None
7     second_edge = None
8     excluded_edges = []
9     while first_edge is None or second_edge is None:
10         possible_edges = net.getNeighboringEdges(x, y, start_radius)
11         possible_edges.sort(key=lambda x: x[1]) # Sort by increasing distance
12
13         for possible_edge in possible_edges:
14             if possible_edge in excluded_edges:
15                 continue
16             if first_edge is None:
17                 first_edge = possible_edge
18                 adjacent_edges = first_edge.getIncoming() + first_edge.getOutgoing
19                 ()
20                 excluded_edges = adjacent_edges.append(first_edge)
21                 continue
22                 second_edge = possible_edge
23                 break

```

```

23 first_edge_position = first_edge.getClosestLanePosDist((x, y))
24 second_edge_position = second_edge.getClosestLanePosDist((x, y))
25
26 return (first_edge, first_edge_position), (second_edge, second_edge_position)

```

Listing 5.5: Implementation of sensor position calculating algorithm for bidirectional sensor devices

Note that there can still be specific situations where this algorithm does not work. For example, when the first object not belonging to the excluded edges is still in the same direction as the first sensor. That is the case when the gap between both directions is large enough for an edge that is not adjacent to the first sensor's edge to be closer to the sensor's location than the edge in the opposite direction.

In spite of that, it is recommended that after the induction loops have also been calculated (see below) to verify that they are in the right place by importing them in `netedit`. If not, changing the sensors' geographical coordinates until the sensors' positions align is recommended instead of changing the algorithm itself.

A file is saved in which each sensor id corresponds to the id and position of the edge.

SUMO sensors' additional

The information obtained about the sensors (see above) can be used to create the correspondent SUMO additional files: calibrators and induction loops.

SUMO sensors' induction loops

Each of the sensors in the network can have multiple corresponding induction loops, one for each lane. They are used both in training (see Section 5.3.4) and to obtain the measurements that can be used for the validation of the simulation.

An induction loop element specification can be seen in Listing 4.1. Using `sumolib` we can iterate each of the lanes of the corresponding edge of every sensor. The lane id and the sensor's position in the edge are used. If the induction loop is too close to the start of the edge, it will not count the traffic inserted by entry calibrator. The position is confirmed to be at least 20 units from the start of the edge, and if it is not, it is adjusted accordingly. The name given to the output file was `sensors_induction_loop_output.xml` but any other is appropriate as long as it is not shared with another SUMO additional output file. The periodicity is equal to the calibration period (see Section 5.3.3), which, for this work, was 300 seconds but could vary according to the available data (see Section 5.2).

Given a lane id *lane_id*, a sensor id *sensor_id*, a sensor position *pos*, a period *period* and considering $|x|$ represents the value of variable *x*, the induction loop XML element can be seen in Listing 5.6.

```

1 <inductionLoop id="e_|sensor_id|_|lane_id|" lane="|lane_id|" pos="|pos|"
  period="|period|"
2   file="sensors_output.xml"/>

```

Listing 5.6: Definition of a SUMO sensor's induction loop

Note that there will be lanes with both an edge induction loop and a sensor induction loop. For the purposes of running the simulation (see Section 5.3.3 the sensor induction loops are used.

A file is created with all of the sensors' induction loops.

SUMO sensors' calibrators

The sensors divide the network into multiple independent systems (see Section 5.3.2), each of them having a calibrated driving behavior. Driving behavior is determined by vehicle type, and therefore, the sensors' calibrators calibrate driving behavior by adjusting the type of the vehicles to the one corresponding to the independent system. Additionally, the sensors' calibrators can be used to ensure that, in spite of the performance of calibration of the simulation, the flow of vehicles passing on a sensor matches the measurements of the real data. This is useful because previous calibrators may result in a disparity in the flow that would be propagated to the independent system downstream of the calibrator.

Due to the function of calibrators within SUMO, the same calibrator cannot be used to both ensure the flow of vehicles and change their type to the wanted one. Therefore, for each of the sensors there shall be two calibrators: the flow calibrator and the vehicle type calibrator. Additionally, since the type of the vehicle is changed as soon as the vehicle reaches the edge of a vehicle type calibrator, its position within the edge can be 0. Furthermore, the position of the flow calibrator in the edge will not be the exact position calculated. A fixed value of 20 units will be added so that the calibrator is after the respective induction loop. This happens so that the data collected by the induction loop that is used during runtime is from traffic that has not been calibrated by the flow calibrator. Otherwise, since both the flow calibrator and the induction loop would overlap, it is possible that the readings could be biased.

Given an edge id *edge_id*, a sensor id *sensor_id*, a sensor position *pos*, a period *period* and considering $|x|$ represents the value of variable *x*, the calibrator XML elements can be seen in Listing 5.7.

```

1 <calibrator id="ca_|sensor_id|_|edge_id|" edge="|id|" pos="|pos|" output="
  sensors_calibrator_output.xml">
2   <flow begin="0" end="|end|" speed="|speed|"/>
3 </calibrator>
4 <calibrator id="vt_|sensor_id|_|edge_id|" edge="|id|" pos="0">
5   <flow begin="0" end="|end|" speed="|speed|"/>

```

```
6 </calibrator>
```

Listing 5.7: Definition of a SUMO sensor's calibrators

A file is created with all of the sensors' calibrators.

Sensor calibrators information

To be able to easily correlate the data of a sensor to the corresponding edge and calibrators, a *.csv* file is created in which each sensor has the sensor id, the flow calibrator id, and the vehicle type calibrator id.

Routes information

Whereas part of the simulations contemplated in the [chapter 3](#) consider traffic from the standpoint of origins and destinations of traffic, the methodology adopted by this work is similar to that in [40] and is instead focused on the traffic flows at an edge level. Not only does this allow to forego an analysis aimed at fabricating an OD matrix, but also facilitates the development of online and real-time methodologies that leverage continuous data reading from sensors. Therefore, the calibration of the simulation mostly relies on the rerouting of vehicles at every bifurcation in the road. The probability with which a vehicle gets rerouted to one or another edge is determined at run time as explained in [Section 5.3.3](#).

In SUMO, vehicles will follow along the given route until they reach its end, at which point they will be removed from the simulation. A vehicle can have a route assigned even if it is not on the first edge of the route but is instead located in another and follows along until the end of the route as normal. If, however, the current edge where the vehicle is located does not belong to the assigned route SUMO will issue an error. A vehicle's route can either be set through the ids of its edges, or a route id. For a route id to be valid, it must be included in one of the route files passed into the SUMO simulation configuration (see [Section 4.2](#)). This methodology uses route ids to identify the routes.

While SUMO includes a rerouter object ¹⁰ its usage was found not to be adequate since it does not allow calibration at run time. Instead, a rerouter object (implemented as *RerouterObject*) was implemented from scratch. Henceforth, rerouter refers to this rerouter object and not SUMO's native rerouter.

Each of the rerouters corresponds to one of the split edges in the network and is responsible to redirect vehicles to one of the two outgoing edges. During runtime, the rerouter will be assigned a probability such that the distribution of the vehicles it reroutes shall follow that same probability. Before discussing the runtime implementation, however, the required information shall be created.

Remember the definition of a path from [Section 5.1.1](#). Notice that, since every pair of subsequent edges is continuous, from a route point of view, a path is deterministic, meaning that a

¹⁰<https://sumo.dlr.de/docs/Simulation/Rerouter.html>

real driver that started in the first edge would necessarily arrive at its last edge. Two paths are connected if the last edge of the first path is connected to the first edge of the second path.

Let a route object (implemented as `RouteObject`) be an object composed of two connected paths, the common and the split paths, a route and its id, the split edge (the last edge of the common path) and its id, the end edge (the last edge of the second path) and its id; and the next route objects. The route is the concatenation of the common and split paths such that the last edge of the common path, the aforementioned split edge, has as an outgoing edge the first edge of the common path. There is a bijection between a route and a route object. Given the id of the first edge of the common and split paths, *common_first_edge_id* and *split_first_edge_id*, respectively, and considering $|x|$ represents the value of variable x , then the route id is $r_|common_first_edge_id|_|split_first_edge_id|$. Given the bijection between routes and route objects the next routes of a route can be derived from the respective route objects.

Entry route objects are route objects that do not have a common path or a split edge. The split path's first edge is an entry edge and the route itself is just the split path. Given the id of the entry edge *entry_id*, and considering $|x|$ represents the value of the variable x , then the route id is $r_entry_|entry_id|$.

In the context of the methodology, route shall, therefore, acquire a definition stronger than just a list of edges, being a list of edges that can be assigned to a vehicle in the simulation and follows the above definition.

To obtain all of the information about the routes the following processes must be executed:

- Compute every path;
- Initialize every route object;
- For each route object calculate the next route objects;
- Create SUMO vehicles routes;
- Aggregate the route information for the edges;
- Aggregate the route information for the rerouters.
- Save all the information to the respective files.

Since the calculation of the routes is so paramount to the rerouting of the vehicles (see Section 5.3.3), each step will be described in depth.

Compute paths

The code in [Listing 5.8](#) represents the algorithm to get every path.

```

1 def get_paths(net: Net, split_edges: set[Edge], entry_edges: set[Edge]):
2     all_paths = set()
3     # Get all of the edges that start a path

```

```

4     outgoing_edges = set()
5     for split_edge in split_edges:
6         outgoing_edges.update(split_edge.getOutgoing())
7     start_path_edges = outgoing_edges.union(entry_edges)
8
9     # Get the paths starting in those edges
10    for edge in start_path_edges:
11        path_edges = []
12        current_edge = edge
13        while True:
14            if is_split_edge(current_edge) or is_exit_edge(current_edge):
15                path_edges.append(current_edge)
16                all_paths.add(path_edges)
17                break
18            current_edge = current_edge.getOutgoing()[0]
19
20    return all_paths

```

Listing 5.8: Algorithm to calculate every path

Initialize route objects

The code in [Listing 5.9](#) initializes the route objects from the paths that were previously calculated. The entry route objects are also calculated.

```

1 def get_route_objects(all_paths: set[list[Edge]]):
2     all_route_objects = set()
3
4     for path in all_paths:
5
6         # Initialize a normal route object
7         next_edges = last_edge.getOutgoing()
8         for next_edge in next_edges:
9             for next_path in all_paths:
10                if next_edge == next_path[0]:
11                    # This is the path that comes after the one we are
12                    iterating
13
14                    common_path = path
15                    split_path = next_path
16                    route = common_path + split_path
17                    route_id = f"r_{path[0].getID()}_{next_path[0].getID()}"
18                    split_edge = common_path[-1]
19                    split_edge_id = split_edge.getID()
20                    end_edge = split_path[-1]
21                    end_edge_id = end_edge.getID()
22                    next_route_objects

```

```

21         route_object = RouteObject(common_path, split_path, route
    , route_id, split_edge, split_edge_id, end_edge, end_edge_id,
    next_route_objects)
22         all_route_objects.add(route_object)
23
24     # Initialize the entry route objects
25     if is_entry_edge(path[0]):
26         common_path = None
27         split_path = path
28         route = split_path
29         route_id = f"r_entry_{path[0].getID()}"
30         split_edge = None
31         split_edge_id = None
32         end_edge = split_path[-1]
33         end_edge_id = end_edge.getID()
34         next_route_objects = []
35         route_object = RouteObject(common_path, split_path, route,
    route_id, split_edge, split_edge_id, end_edge, end_edge_id,
    next_route_objects)
36         all_route_objects.add(route_object)
37
38     return all_route_objects

```

Listing 5.9: Algorithm to initialize every route object

Calculate next route objects

The next route objects of a route object is every route object such that their common path is the same as the reference's target path. As a consequence, the next route objects' split edge is the end edge of the analysed route object. The following code uses this definition to cross reference the route objects.

```

1 def get_next_route_objects(route_objects: set[RouteObject]):
2     for route_object in route_objects:
3         for next_route_object in route_objects:
4             if next_route_object.common_path == route_object.split_path:
5                 route_object.next_route_objects.append(next_route_object)

```

Listing 5.10: Algorithm to cross reference to calculate the next route objects

Additionally, we want to guarantee that the order of the next routes is consistent with the order of the outgoing edges of the end edge. More precisely, the list with the first edge of the split path of the next route objects is the same as the outgoing edges of the end edge of the route object. The code in [Listing 5.11](#) guarantees that.

```

1 def order_next_route_objects(route_objects: set[RouteObject]):

```

```

2     for route_object in route_objects:
3         if route_object.next_route_objects == []:
4             # Is a route object that ends in an exit edge. Skip
5             continue
6         next_route_first_split_path_edges = [next_route_object.split_path[0]
7         for next_route_object in route_object.next_route_objects]
8         outgoing_edges = route_object.end_edge.getOutgoing()
9         ordered_next_route_objects = []
10        for outgoing_edge in outgoing_edges:
11            idx_in_next_route_objects = next_route_first_split_path_edges.
12            index(outgoing_edge)
13            next_route_object = route_object.next_route_objects[
14            idx_in_next_route_objects]
15            ordered_next_route_objects.append(next_route_object)
16
17        route_object.next_route_objects = ordered_next_route_objects

```

Listing 5.11: Algorithm to ensure order of the next route objects

SUMO vehicles routes

During runtime, the rerouting of a vehicle (see Section 5.3.3) will imply assigning it a new route, using its id, as explained before. Therefore a file must be created with those routes so they are available to the simulation. There will be a route for each of the route objects.

A route element specification can be seen in Listing 5.2, for the case when the edge is composed of just one edge. Each SUMO vehicles route can be composed of multiple edges and will be written down as a space-separated list of edge ids that can be obtained with the following line `edges_str = " ".join([edge.getID() for edge in route_object.route])`. Given the route id *route_id*, the edges in the appropriate format *edges_str* and considering $|x|$ represents the value of variable *x*, the route XML element can be seen in Listing 5.12:

```

1 <route id="r_|route_id|" edges="|edges_str|"/>

```

Listing 5.12: Definition of a specific SUMO route

A file is created with all of the SUMO vehicles routes.

Aggregate rerouter information

The rerouting of a vehicle happens only when the vehicle is in the split path of its route before arriving at the last edge, which is either an exit edge (a trivial case since there are no worries about rerouting) or a split edge (see Section 5.3.3). Recall that when a vehicle is assigned a new route, the new route must contain the current edge of the vehicle. Since there are multiple split paths (and, by extension, multiple routes) ending in the same split edge, we must guarantee that the vehicle is rerouted to a compatible route, meaning that the

split path of the route must be the same as the common path of the newly assigned route. By definition, a vehicle following a route must, therefore, be assigned any of the route's next routes. Each rerouter is associated with a split edge and will keep track of every route and respective next routes that end in that split edge. The code in [Listing 5.13](#) gathers and centralizes this information.

```

1 def get_rerouter_information(split_edges: set[Edge], route_objects: set[
    RouteObject]:
2     rerouter_information: dict[Edge, dict[RouteObject, list[RouteObject]] =
        {}
3     for split_edge in split_edges:
4         split_edge_routes = {}
5         for route_object in route_objects:
6             if route_object.end_edge == split_edge:
7                 # This route object belongs to this rerouter
8                 split_edge_routes[route_object] = route_object.next_routes
9
10    rerouter_information[split_edge] = split_edge_routes

```

Listing 5.13: Algorithm to calculate rerouter routes and next routes

Aggregate edge information

To initialize a calibrator, we must assign it a route. We will want to assign our calibrators a route such that the vehicle will be rerouted as described above. Therefore, the edge of the calibrator must be in the split path of the route it is assigned. Therefore, besides mapping routes to edges, we are going to map edges to routes. Not only that, but this mapping will describe if the edge is in the common path or in the split path of said route. This can be done as described in [Listing 5.14](#).

```

1 class EdgesLookup:
2     def __init__(self, edge):
3         self.edge = []
4         self.in_common_path_of_routes = []
5         self.in_split_path_of_routes = []
6
7     def add_in_common(route_object: RouteObject):
8         self.in_common_path_of_routes.append(route_object)
9
10    def add_in_split(route_object: RouteObject):
11        self.in_split_path_of_routes.append(route_object)
12
13 def map_edges_to_routes(edges: set[Edge], route_objects: set[RouteObject]):
14     edges_mapping: dict[Edge, EdgesLookup] = {}
15     for edge in edges:
16         edge_look_up = EdgesLookup(edge)

```

```

17     for route_object in route_objects:
18         if edge in route_object.common_path:
19             edges_look_up.add_in_common(route_object)
20             continue
21         if edge in route_object.split_path:
22             edges_look_up.add_in_split(route_object)
23         edges_mapping[edge] = edge_look_up
24
25     return edges_mapping

```

Listing 5.14: Algorithm to map edges to routes where it is featured

Save information

The file with SUMO vehicles routes was already created. The remaining files, which were identified before, can finally be created from the other gathered data:

- Routes information: Save all of the RouteObject's.
- Edges' routes information: Save the aggregated edge information.
- Rerouters' routes information: Save the aggregated rerouter information.

Reachable edges information

The methodology described so far, particularly when it comes to the routes information (see Section 5.3.2, naively assumes that, as long as two edges are connected, a vehicle in the first edge can move to the second. This, however, is untrue since continuous lines exist on the roads that restrict traffic. If a vehicle comes to a split edge and is in the right-most lane of the road, there might be a continuous line that prevents it from going to the edge on the left. As such, it becomes necessary to identify which movements are possible so that routing can take that into account.

For example, in Figure 5.2, where traffic flows from the bottom to the top, vehicles coming from the entry edge 2 cannot make their way to edge 4. Similarly, vehicles on the right-most lane of edge 1 that do not change left must proceed forward to 5.

The reachable edges information will solve this problem. For each split edge, we will evaluate the reachability, that is, which of its outgoing edges are reachable, of the lanes of the previous edges. Edges are evaluated backward from the split edge stopping when arriving at a split edge. Going further back is unnecessary since the calculation of the reachable edges, if applicable, will fall under the previous split edge. Reachability will only be recorded if there are limitations to the edges that can be reached from an edge. Therefore, if a lane is not featured in the reachable edges information it means that no such concerns exist for its edge. For example, every split edge that has no continuous line does not affect reachability and there is no need to calculate the reachable edges of the previous lanes.

Although the continuous lines information is stored by the network (see Section 5.3.1), the version of sumolib used (the one installed with SUMO version 1.19.0) does not provide a way to

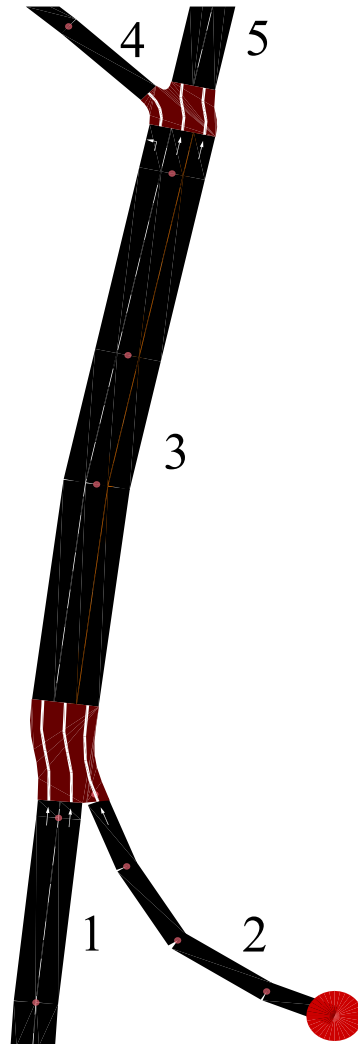


Figure 5.2: Example of a network with a continuous line on a split edge

access these. Because of that, the source code had to be altered to support the calculation of the reachable edges information. The relevant changes can be seen in [Listing A.4](#) and [Listing A.5](#) which allow the introduction of the methods *getChangeLeft* and *getChangeRight* which allow determining if a lane has a continuous lane to its left or its right, respectively.

First, we shall lay the groundwork to determine which lanes on the same edge can be reached from a lane, as is computed in [Listing 5.15](#).

```

1 def can_change_left(lane: Lane):
2     return l.getChangeLeft() != ["authority"]
3
4 def can_change_right(lane: Lane):
5     return l.getChangeRight() != ["authority"]
6
7 def edge_has_continuous_line(edge: Edge):
8     for lane in edge.getLanes():
9         if len(lane.getChangeLeft()) > 0 or len(lane.getChangeRight()) > 0:
10            return True
11    return False
12
13 def get_reachable_lanes_from_lane(lane: Lane):
14     edge = lane.getEdge()
15     number_lanes = len(edge.getLanes())
16     lane_index = lane.getIndex()
17     reachable_lanes = set()
18
19     change_left_index = lane_index
20     current_lane = lane
21
22     # Get reachable lanes to the left
23     while True:
24         current_lane = edge.getLane(changeLeftIndex)
25         if current_lane.canChangeLeft():
26             reachable_lanes.add(current_lane)
27
28         change_left_index += 1
29         if change_left_index >= number_lanes
30             break
31
32     # Get reachable lanes to the right
33     change_right_index = lane_index
34     while True:
35         current_lane = edge.getLane(changeRightIndex)
36         if current_lane.canChangeRight():
37             reachable_lanes.add(current_lane)
38         change_right_index -= 1
39         if change_right_index <= 0:
40             break

```

```

41
42     return reachable_lanes

```

Listing 5.15: Algorithm to determine which lanes are reachable from a lane within an edge

The lanes in the next edge can be trivially retrieved in SUMO using the *getOutgoingLanes* function. The calculation of the reachable edges information is demonstrated by the following algorithm:

```

1
2 def flatten(list_of_lists: list[list[object]]):
3     return [item for sublist in list_of_lists for item in sublist]
4
5 def get_reachable_edges(split_edges: set[Edge]):
6     reachable: dict[Lane, set[Edge]] = {}
7     for split_edge in split_edges:
8         if not edge_has_continuous_line(split_edge):
9             # There is no continuous line. No need to compute reachable edges
            information
10            continue
11            outgoing_edges = set(split_edge.getOutgoing())
12            previous_edges_to_calculate = [split_edge]
13
14            while len(previous_edges_to_calculate) > 0:
15                current_edge = previous_edges_to_calculate.pop(0) # Get one edge to
                calculate reachability
16
17                if is_split_edge(current_edge) and current_edge != split_edge:
18                    # We traced the previous edges back to the previous split edge. We
                    will not calculate this edge or further back
19                    continue
20                    all_edges_reachable = True
21                    for lane in current_edge.getLanes(): # Analyze reachability for every
                    lane in the edge
22                        reachable_next_edges = set()
23                        reachable_lanes = get_reachable_lanes_from_lane(lane)
24                        reachable_next_lanes = [lane.getOutgoingLanes() for lane in
                    reachable_lanes]
25                        reachable_next_lanes = set(flatten(reachable_next_lanes))
26
27                        for reachable_next_lane in reachable_next_lanes:
28                            if reachable_next_lane in reachable:
29                                # Reachability is transitive
                                    reachable_next_edges.update(reachable[reachable_next_lane])
30                            else:
31                                # lane was in the split edge and the reachable_next_lane
                                    belongs to the outgoing edges
32                                reachable_next_edges.add(reachable_next_lane.getEdge())
33

```

```

34         reachable[lane] = reachable_next_edges
35
36         if reachable_next_edges != outgoing_edges:
37             all_edges_reachable = False
38
39
40         if not all_edges_reachable:
41             # The current edge has restrictions on routes. Analyse the previous
42             edge and its lanes
43             previous_edges_to_calculate.extend(current_edge.getIncoming())

```

Listing 5.16: Algorithm to calculate reachable edges information

A file should be created with the *reachable* mapping.

For the example network (see [Figure 5.2](#)), [Table 5.2](#) represents the reachable mapping for the relevant edges. Edges 4 and 5 are not included in the data since they are after the split edge that guides the calculations: edge 3. Edge 1 is not included in the data since, from every lane, every possible outgoing edge after the split is reachable (since a vehicle would be allowed to change lanes within edge 1). Edge 2 is included in the data since there are limitations to the edges it can reach. Since vehicles can only proceed to lane 0 of edge 3, only edge 5 is reachable. Since edge 3 is a split edge with a continuous line, it is also included in the data, and the reachabilities of each of its lanes are featured.

Table 5.2: Reachable edges for network in [Figure 5.2](#)

Edge	Lane	Reachable edges
2	0	5
3	0	5
3	1	4, 5
3	2	4, 5

Independent systems

Let an independent system be defined as a sub-network in which every entry edge and exit edge is either an entry or exit edge in the original network or sensor edge and contains no other independent systems. In a more practical sense, think that the independent systems of a network are obtained by doing a "cut" at the sensor edges and extracting the resulting networks that, having no edges connecting them, are considered "independent". See [Section 6.2](#) for examples of this procedure and the resulting independent system. This division into independent systems provides an intuitive partitioning of the network, and each of the independent systems can undergo calibration. The calibration of an independent system includes both objectives (see [Section 5.1.3](#)): minimize the differences between the simulated and the real measurements and derive a specific

vehicle model for each independent system. There shall be a number of vehicle models equal to the number of independent systems.

Additionally, given that the system is delimited by sensors whose measurements are known, we can control the simulation in such a way that shortcomings in the calibration of the other independent systems do not affect this one. For example, given two independent systems where the exit edge of the first is the entry edge of the second (because that edge has a sensor), the calibration of the first, for some reason, is not as accurate. If too many vehicles travel through the shared sensor edge, even if the second system is well-calibrated, the greater number of vehicles will influence its simulated measurements and result in an error. To prevent this, calibrators can be put in place (see Section 5.3.2) to control the flow coming into the system to minimize these errors.

Not only that, but each independent system can be simulated individually, allowing a paradigm change where, instead of considering the whole network, we can just consider a smaller and less computationally expensive sub-network. Considering that different systems, due to their size or availability of data, get calibrated at different rates, this partition allows for a more flexible training and validation methodology. For that to be possible, all of the network information files computed so far must also be calculated just for the independent system. The methodology and the implementation do not analyze the caveats of the simulation of multiple independent systems at the same time, even though such a feature is plausible.

Independent systems networks To calculate the independent systems we are going to inspire ourselves in the "cutting" the sensor edges analogy. Starting on a given edge with a sensor, we can recursively traverse the graph (for example, with a depth-first search) using the connected edges and stopping additional traversal when another sensor edge is found ("cutting" at the sensor edge). The traversed edges will make up the independent system. Remember that the sensor edges themselves are going to be part of the independent system, and they can either belong to just 1 independent system if they were an entry or exit edge of the original network or 2 independent systems otherwise.

From an implementation point of view, it is easier to consider the vertices on both extremes of a sensor edge instead of the edge itself. When "cutting" a sensor edge, we will instead "cut" a vertex. The algorithm will continue until all vertices are explored, as is seen below:

```

1 def independent_system_edges(sensor_edges: set[Edge]):
2     sensor_from_nodes = [edge.getFromNode() for edge in sensor_edges]
3     sensor_to_nodes = [edge.getToNode() for edge in sensor_edges]
4     sensor_nodes = sensor_from_nodes + sensor_to_nodes
5
6     analysed_sensor_nodes = set()
7     independent_systems_edges: list[set[Edge]] = []
8
9     for node in sensor_nodes:
10         if node in analysed_sensor_nodes:

```

```

11         continue
12
13         system_nodes = dfs_stop_on_sensor_nodes(sensor_edges, sensor_nodes) #
14         Get all of the nodes in the system
15         system_sensor_nodes = system_nodes.intersection(sensor_nodes)
16         analysed_sensor_nodes.update(system_sensor_nodes) # Update the sensor
17         nodes that have been analyzed
18
19         system_edges = set()
20         for system_node in system_nodes:
21             system_edges.update(system_node.getIncoming())
22             system_edges.update(system_node.getOutgoing())
23
24         independent_systems_edges.append(system_edges)
25
26 def dfs_stop_on_sensor_nodes(current_node, sensor_edges, visited_nodes):
27     visited_nodes.add(current_node)
28
29     # Traverse the incoming edges
30     for incoming_edge in current_node.getIncoming():
31         if outgoing_edge not in sensor_edges:
32             incoming_node = incoming_edge.getFromNode()
33             if incoming_node not in visited_nodes:
34                 visited_nodes = dfs_stop_on_sensor_nodes(incoming_node,
35                 sensor_edges, visited_nodes)
36
37     # Traverse the outgoing edges
38     for outgoing_edge in n.getOutgoing():
39         if outgoing_edge not in sensor_edges:
40             outgoing_node = outgoing_edge.getToNode()
41             if outgoing_node not in visited_nodes:
42                 visited_nodes = dfs_stop_on_sensor_nodes(outgoing_node,
43                 sensor_edges, visited_nodes)
44
45     return visited_nodes

```

Listing 5.17: Calculatiog independent system edges

After the edges for each system have been computed, each network is extracted from the main network and stored. Once again, the netconvert tool is used by applying to the original network the *-keep-edges.explicit* argument with the list of edges which can be obtained with `" ".join([edge.getId() for edge in system_edges])`. An example of a network divided into systems is featured in Chapter 6 (see Appendix A.1).

Independent systems equations More often than not, the available data will contain sensors that lead to incomplete network coverage, which means that not every edge nor continuous segment will have a sensor. Therefore, it becomes necessary to somehow infer the traffic in

a continuous segment without coverage. For that, we will adopt a methodology similar to that in [40] to devise a system of equations that allows us not only to obtain the maximum and minimum but also to determine a possible flow of the continuous segment.

Considering that entry edges are the only edges where vehicles can be introduced into the network and exit edges are the only edges where vehicles can be removed from the network (ignoring the existence of flow sensor calibrators as seen in Section 5.3.2), given a split edge $E_{1,2}$, all its traffic must go to either the outgoing edge on the left $E_{2,3}$ or the outgoing edge on the right $E_{2,4}$. Alternatively, we could say that the number of vehicles that will pass in both of the outgoing edges totals the number of vehicles that will pass in the split edge. Mathematically, this is expressed through Equation 5.3 for split edges. For split edges, the thought process is analogous and is represented in Equation 5.4.

$$Q_{1,2} = Q_{2,3} + Q_{2,4} \quad (5.3)$$

$$Q_{1,3} + Q_{2,3} = Q_{3,4} \quad (5.4)$$

Additionally, two continuous edges are assumed to have the same flow, meaning that a continuous segment is treated as a single unit for flow calculations. The variables of the equations will all be associated with a continuous segment.

Note that, given the wave-like dynamics of traffic, under circumstances of different flow, the propagation of said flow puts into question the deterministic nature of this relation. For example, given two continuous edges under constant flow, if the first edge suddenly has a reduction in flow, the associated reduction in flow of the second edge will be delayed. This situation can be even more problematic when talking of splits or merges. Consider the following scenario represented by Equation 5.5 where we take into account not only counts of vehicles but the count of vehicles in a given period of time.

$$\begin{aligned} Q_{1,2} &= Q_{2,3} + Q_{2,4} \\ Q_{1,2}(t_1) &= 5; Q_{2,3}(t_1) = 3; Q_{2,4}(t_1) = Q_{1,2}(t_1) - Q_{2,3}(t_1) = 5 - 3 = 2 \\ Q_{1,2}(t_2) &= 0; Q_{2,3}(t_2) = 3; Q_{2,4}(t_2) = Q_{1,2}(t_2) - Q_{2,3}(t_2) = 0 - 3 = -3 \end{aligned} \quad (5.5)$$

We have, therefore, wrongly concluded that the flow of an edge is negative, which is impossible. For this work, we do not consider this minutia and resign ourselves to the simplified scenario where flows accurately reflect the number of vehicles where the whole is the sum of the parts. Since the solution of the equations is determined periodically and not constantly, it is possible that these situations do occur or occur with lower frequency.

Following this methodology, a system of equations shall be computed for each independent system (see above) expressing the flows in its continuous segments. An equation shall be created for every split or merge vertex that expresses the relationship between the flows

of the incoming and outgoing continuous segments. Each continuous segment in the independent system corresponds to a variable that represents its flow. The system of equations is then simplified using sympy's ¹¹ *linsolve*. Additionally, the sensors in the independent system can provide the measurements of the flow corresponding to some of the system's variables. Therefore, we will have a system of n equations, with some known variables (from the sensors) and some unknown variables. Amongst the unknown variables, some of them will be expressed in function of the others, the free variables. If there are no free variables, then the system has a single solution.

If so, and every variable is non-negative, the solution can be used to calculate and configure the independent system by setting the flow of the entry calibrators (see Section 5.3.2, which may include the sensor flow calibrators (see Section 5.3.2), and the probability of the rerouters (see Section 5.3.2).

Still, it is likely that not a single solution exists, but a vast multi-dimensional solution space. Multiple algorithms can be used to obtain one of the points in this space to be used as the one solution. In our case, the scipy library is used ¹², particularly the linear programming optimization function ¹³ which uses either the dual revised simplex method or an interior point method. Further restrictions are put in place, mainly in the form of inequalities that dictate the flow of each continuous segment must be non-negative, as stated before, but also not bigger than a fixed value that was found to be greater than every recorded flow in the available data. The specific point that satisfies the system of equations and inequalities is the one that minimizes a linear objective function. The coefficients of this function are part of the calibration that can be performed and for which this work proposes a methodology. In [40] and [47] other methods are used to calculate these coefficients.

For each of the systems, the system of equations, the free and the known variables, and the correspondence between the continuous segments and the variables are saved.

SUMO configuration file

After all of the needed files have been created, the configuration file is responsible for aggregating all of the SUMO files so that the simulation can be initialized. Multiple configuration files should be created: one for the whole network and one for each of the independent systems in case those will be simulated instead.

Given the network file *network*, the edges' induction loops *edges_induction_loops*, the edges' route file *edges_routes*, the edges' entry calibrators *edges_entry_calibrators*, the sensors' induction loops *sensors_induction_loops*, the sensors' calibrators *sensors_calibrators*, and the SUMO vehicles routes *vehicles_routes*, the end time *end*, one step per second, and considering $|x|$ represents the value of variable x , the configuration XML can be seen in Listing 5.18.

¹¹<https://www.sympy.org/en/index.html>

¹²<https://scipy.org>

¹³<https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html>

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.xsd">
3   <input>
4     <net-file value="|network|" />
5     <route-files value="|edges_routes|,|vehicle_routes|" />
6     <additional-files value="|edges_induction_loops|,|edges_entry_calibrators
   |,|sensors_induction_loops|,|sensors_calibrators|" />
7   </input>
8
9   <time>
10    <begin value="0" />
11    <end value="|end|" />
12    <step-length value="1" />
13  </time>
14
15 </configuration>

```

Listing 5.18: SUMO configuration file

5.3.3 Simulation

There are multiple ways to run a SUMO simulation and to dictate the traffic and its behavior. For example, it is possible to predetermine all of the routes that the vehicles will take at a given point in time. For this work, however, since online data is going to influence the traffic, we opt for a different approach based on [40] where vehicles are introduced and rerouted in the network according to the newly received data. With these data we aim to adjust the traffic in the network, particularly so that the simulated measurements in the sensor edges match the real measurements. Additionally, we also shall configure and readjust the vehicle models.

To control the simulation in real time, the traci API (see Section 4.5) is used. Though powerful, there are some restrictions in its functionalities that will be described as they become relevant. We shall start by describing the overall architecture of the architecture, before explaining how the simulation is initialized, and then do a deep dive into each relevant component of the architecture.

Architecture

A diagram of the architecture of the simulation framework can be seen in Figure 5.3.

The Simulation Runner is the main component that is responsible for the whole simulation. Despite being a separate process, the SUMO Simulation is included inside of it as it more accurately describes the interactions. The SUMO simulation has five relevant classes: the induction loops, the calibrators, the simulation variables, the vehicles, and the vehicle types. Additionally, either one or the multiple independent systems (see Section 5.3.2) exist within the Simulation Runner and interact with the SUMO simulation using traci. The Data Source represents either the historical data or the sensor devices present on the roads and makes online data available to

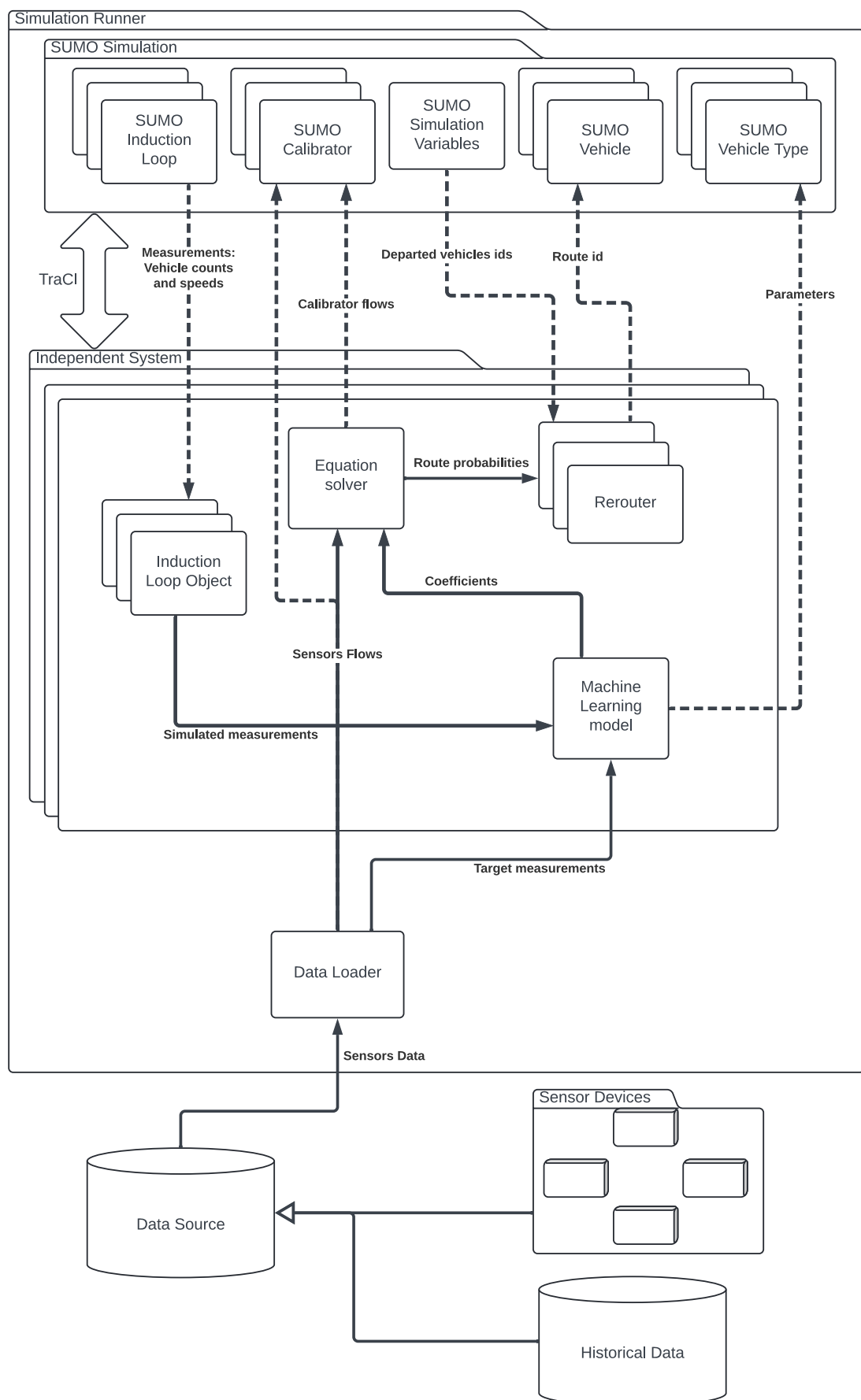


Figure 5.3: Architecture of the simulation framework

the Simulation Runner. The Data Loader is responsible for acquiring and processing this data for every independent system.

The independent system collects information from the SUMO simulation at every step and fetches new data from the data loader at every calibration step. The Induction Loop objects are needed to aggregate the SUMO induction loop's data over a period of time. The Rerouter will adjust the routes of the vehicles at every step. The Machine Learning model and the Equation solver are responsible for calculating the values for the calibration. A calibration happens when the interactions with the SUMO calibrators and the SUMO vehicle types take place. This is done in every partial calibration step. The difference between a partial calibration step and a calibration step should be made clear. In terms of the interaction with the SUMO simulation, a calibration is always the same. The distinction lies in the type of data used. For a calibration step, the real data that is used is fetched from the Data Loader, whereas a partial calibration step uses the most recent real data that is available fetched, effectively reusing the data. A calibration step is also a partial calibration step, and the partial calibration period, the number of steps between each partial calibration step, is a divisor of the calibration period, the number of steps between each calibration period. The calibration period should align with the periodicity of the real data. Given a real data periodicity of 300 seconds and with a step per second, the calibration period will be 300 steps, and the partial calibration step could be, for example, 100 steps. $t = 0$ is both a partial calibration step and a calibration step and data is newly fetched from the Data Loader. $t = 100$ and $t = 200$ are also partial calibration steps and the real data used is the same as was used in $t = 0$. Again, $t = 300$ is both a partial calibration step and a calibration step with more data being fetched from the Data Loader.

The introduction of this partial calibration step has two objectives. First, each partial calibration step is a data point that the Machine Learning models can train on as the comparison between the simulated data and the real data can happen. Second, we wish to study if the performance of the simulation increases by performing the calibration more often (every partial calibration step instead of every calibration step). It is possible that no performance increase is observed. Particularly when it comes to the measured simulated flows of vehicles, the changes in the rerouting probabilities and the flow of vehicles entering the system are not immediately noticeable at the sensors and the changes still need to propagate in the network.

Initialization

The initialization begins with starting SUMO by using traci and the configuration file (see Section 5.3.2). Additionally, SUMO can be started in graphical or console mode. The graphical mode allows for visualization of the simulation in a separate window and is appropriate if there is an interest in seeing the behavior of the simulation and fixing any problems with the implementation.

Next, in order to access the information from the induction loops and calibrators that were created (see Section 5.3.2), we must subscribe to them. To subscribe to an object, its id must be used, and the wanted variables must be enumerated. In the case of the induction loops, we

want the measurements equivalent to the real data measurements. In this work, since we are using the flow and speed of vehicles, we subscribe to the *LAST_STEP_VEHICLE_NUMBER* and *LAST_STEP_MEAN_SPEED*. The subscription of the calibrators is for statistical purposes only and we subscribe to the number of vehicles that passed through it *VAR_PASSED*, the number of vehicles inserted *VAR_INSERTED*, and the number of vehicles removed *VAR_REMOVED*. Furthermore, we subscribe to SUMO simulation itself to get the ids of the vehicles currently traveling in it *VAR_DEPARTED_VEHICLES_IDS*, which is needed for its rerouting. Like the calibrators, for statistical purposes, the number of departed vehicles *VAR_DEPARTED_VEHICLES_NUMBER*, the number of teleported vehicles *VAR_TELEPORT_ENDING_VEHICLES_NUMBER*, and the number of vehicles that arrived are also subscribed *VAR_ARRIVED_VEHICLES_NUMBER*. Notice that traci allows either subscription or a *get* method to access its variables. However, it was found that the subscription was faster so it was the adopted approach.

Lastly, multiple vehicle types are derived from the default vehicle type and introduced in the simulation to be used for the calibration of the vehicle model.

The sensors' calibrators are also initialized. If the sensor edge is an entry edge to the independent system, the respective sensor vehicle type calibrator will be set with the vehicle type of that independent system (see Section 5.3.2). Vehicle type calibrators work passively and, once configured, will not need to be changed. Therefore, this is the only measure that needs to be taken so that the vehicle type of a vehicle corresponds to the vehicle type of the independent system it is currently in for the whole simulation. Given the id of the calibrator *calibrator_id*, the end time of the simulation *end_time*, the vehicle type of this independent system *type*, and considering $|x|$ represents the value of variable x , the function will be called as seen in Listing 5.19:

```
1 traci.calibrator.setFlow("|calibrator_id|", 0, |end_time|, -1, -1, "|type|", "", "
   free", "max")
```

Listing 5.19: Function call to set flow of the sensor vehicle type calibrator

The sensor flow calibrator can also be initialized. The most important about these is setting the route id. Since this vehicle can insert vehicles, those vehicles will be assigned the route of the calibrator. Any route in which the sensor edge belongs to the split path is a viable route. Recalling the item Aggregate edge information in Section 5.3.2, the route id can be calculated as follows:

```
1 edges_mapping: dict[Edge, EdgesLookup] = {}
2
3 def calculate_sensor_edge_route_id(sensor_edge: Edge):
4     edges_lookup = edges_mapping[sensor_edge]
5     split_routes = edges_lookup.in_split_path_of_routes
6     possible_route: RouteObject = split_routes[0]
7     possible_route_id = possible_route.route_id
8     return possible_route_id
```

Listing 5.20: Calculate sensor flow calibrator route id

Data Loader

The Data Loader is responsible for fetching the sensors data in an online manner. This can either be done through historical data or real time data coming from the sensors devices. Every calibration step, the Simulation Runner will task the Data Loader with acquiring that data.

The implementation achieved uses historical data and ,as such, it was possible to preemptively sanitize and filter the data. The data is obtained as a pandas Dataframe ¹⁴ where to each measurement corresponds a column, and to each sensor a row.

In other works, if real data is used, probably through an API, it might be necessary to filter only data corresponding to the sensors in the simulation, which can be obtained by referring to the sensors information (see Section 5.3.2). Due to the preprocessing of the real historical data, the data was already curated and no validation mechanism was implemented, which might become necessary for this type of implementation.

The usage of each of the measurements will be described by the components who use them, below.

Induction Loop Objects

Through traci it is possible to obtain the flow and speed of the vehicles that passed a SUMO induction loop (see Section 5.3.3). However, that data is regarding a single step. Since calibration happens every partial calibration step, these data need to be aggregated to that time period. Therefore, the Induction Loop Object is introduced. In each step, the Induction Loop Object registers the data from the SUMO induction loop. The aggregated flow of a SUMO induction loop is done by counting the number of vehicles that have passed through it during the partial calibration period p :

$$Q_{i,j_k}(t_a, t_a + p) = \sum_{n=0}^p Q_{i,j_k}(t_a + n) \quad (5.6)$$

The count of vehicles can then be normalized to the period of one hour:

$$F_{i,j_k}(t_a, t_a + p) = Q_{i,j_k}(t_a, t_a + p) * 3600 / p \quad (5.7)$$

The aggregated mean speed will be the mean speed of the vehicles that passed through it. Since multiple vehicles can pass through the induction loop in one step, the Induction Loop Object will instead accumulate the product of the mean speed of the last step multiplied by the number of vehicles. When the data is fetched, the value obtained through the aforementioned accumulation will be divided by the total number of vehicles, which is also known:

¹⁴<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>

$$v_{i,j_k}(t_a, t_a + p) = \frac{\sum_{n=0}^p Q_{i,j_k}(t_a + n) * v_{i,j_k}(t_a + n)}{Q_{i,j_k}(t_a, t_a + p)} \quad (5.8)$$

Also, SUMO induction loops are placed on a single edge. The Induction Loop Object operates on an edge basis, thus being associated with as many SUMO induction loops as the correspondent edge has lanes. The calculation of the flow for the Induction Loop Object is trivially the sum of the flows of each lane.

$$F_{i,j}(t_a, t_a + p) = \sum_{k=0}^{lanes} F_{i,j_k}(t_a, t_a + p) \quad (5.9)$$

The mean speed must be weighed in a similar manner as before by obtaining the sum of the product of the mean speed and the number of vehicles in each lane for the time period and only afterward dividing by the total number of vehicles.

$$v_{i,j}(t_a, t_a + p) = \frac{\sum_{k=0}^{lanes} Q_{i,j_k}(t_a, t_a + p) * v_{i,j_k}(t_a, t_a + p)}{Q_{i,j}(t_a, t_a + p)} \quad (5.10)$$

All of the data kept by the Induction Loop Object is reset every partial calibration step after the data has been collected.

Rerouter

Each Rerouter is an object that is associated with a certain split edge and a probability. It is responsible for the rerouting of the vehicles, that is, setting the route of each individual vehicle while respecting the given probability. The probability is just the probability of a vehicle choosing the left outgoing edge of the split edge in comparison to the right outgoing edge. This probability is determined with each calibration from the flows calculated by the Equation solver (see Section 5.3.3). Given the flow of the left edge f_1 and the flow of the right edge f_2 , the probability is simply $\frac{f_1}{f_1 + f_2}$.

Though abstracted in the architecture (see Section 5.3.3), the Simulation Runner is the one to determine which vehicles shall be rerouted. For each of the vehicles in the simulation, represented by each of the depart vehicle ids obtained from the SUMO Simulation Variables, it is decided if the SUMO vehicle shall be rerouted. The high-level approach is the following:

```

1 class SimulationRunner:
2     routes: dict[str, RouteObject]
3     rerouters: dict[str, Rerouter]
4     def run():
5         while True:
6             traci.simulationStep() # Step the simulation
7             results: dict[str, dict[str, object]] = traci.vehicle.
            getAllSubscriptionResults()
8             for key, value in results.items():

```

```

9         vehicle_id = key
10        vehicle_info = value
11        reroute_vehicle(vehicle_id, vehicle_info)
12
13    def reroute_vehicle(vehicle_id, vehicle_info):
14        lane_id = vehicle_info[traci.constants.VAR_LANE_ID]
15        road_id = vehicle_info[traci.constants.VAR_ROAD_ID]
16        cur_route_id = vehicle_info[traci.constants.VAR_ROUTE_ID]
17
18        if is_to_be_rerouted(cur_route_id, lane_id, road_id):
19            cur_route = routes[cur_route_id]
20            split_edge = cur_route.split_edge
21            rerouter = rerouters[split_edge]
22            new_route = rerouter.chose_next_route(cur_route, lane_id, road_id)
23            traci.setRouteID(vehicle_id, new_route_id)

```

Listing 5.21: Running loop with vehicle rerouting

The function *is_to_be_rerouted* leaves room for different implementations. For example, the decision could be made based on the number of edges until the split edge, the distance that is left up until the split edge, and the speed of the vehicle. Holistic choices based on the amount of traffic in the the adjacent edges are also a possibility. In this work, we choose an approach that attempts to allow the vehicle to reroute to any of the split edge's outgoing edges as late as possible. Remember that this is not always possible (see Section 5.3.2). Due to that, the rerouting procedure must be adapted in such a way that vehicles are not rerouted to a route they cannot follow, and that the probability of the rerouter is respected.

To do that, a second implementation of the Rerouter, which we shall name *AdjustableRerouter* was developed. Instead of a probabilistic approach, randomizing the route of the vehicle, the Adjustable Rerouter can behave deterministically by keeping track of the vehicles that have passed in the left and right outgoing edges of its split edge and choose, if given the freedom to do so, the route that moves current ratio closer to the assigned probability. Below there is a possible implementation:

```

1
2 class Rerouter:
3     def __init__(self, probability, routel, route2):
4         self.probability = probability
5         self.routel: RouteObject = routel
6         self.route2: RouteObject = route2
7
8     def choose_random_route():
9         r = random.random()
10        if r < probability:
11            return self.routel
12        return self.route2

```

```

13
14 class RerouterAdjustable(Rerouter):
15
16     def __init__(self, probability, routel, route2):
17         super().__init__(probability, routel, route2)
18         self.number_vehicles_to_the_left = 0
19         self.total_vehicles = 0
20
21     def choose_route(self) -> str:
22         if self.total_vehicles == 0:
23             return self.choose_random_route()
24
25         p = self.number_vehicles_left / self.total_vehicles
26         dif = p - self.probability
27         if dif < 0:
28             return self.choose_mandatory_route(self.routel)
29         elif dif > 0:
30             return self.choose_mandatory_route(self.route2)
31         else:
32             # Reset the counts for the compensation
33             self.number_vehicles_to_the_left = 0
34             self.total_vehicles = 0
35
36             return self.choose_random_route()
37
38     def choose_random_route():
39         r = random()
40         if r < self.probability:
41             return self.routel
42         return self.route2
43     def choose_mandatory_route(self, chosen_route):
44         if chosen_route == self.routel:
45             self.number_vehicles_to_the_left += 1
46             self.total_vehicles += 1
47
48         return chosen_route

```

Listing 5.22: Adjustable Rerouter class

This implementation will make the ratio between the routes of the vehicles approach the intended probability. Consider, however, that the counts of the vehicles reset with each calibration (meaning the Rerouter gets updated with new flows from the Equation solver), and no knowledge of any previous disparities in the ratio is kept. As such, if calibrated too frequently, the adjustable rerouter might not be able to compensate for vehicles with no freedom of choice in the route chosen. The performance is also dependent on the incoming traffic. If all the vehicles that are to be rerouted must only choose one of the routes, there is no compensation that can be made.

Important to mention is that this alternative implementation is used only for the cases when

the split edge of the original Rerouter has restrictions on reachability. This can be done during the initialization of the simulation and using the reachable edges information (see Section 5.3.2). If any of the lanes of the split edge is in the reachable edges information then the AdjustableRerouter implementation must be used.

The algorithm to determine if both routes can be chosen or not can be seen in Listing 5.23.

```

1 reachable: dict[Lane, set[Edge]] # Reachable edges information
2 net: Net
3 class RerouterAdjustable(Rerouter):
4     choose_next_route(self, cur_route: RouteObject, lane_id, road_id):
5         lane = net.getLane(lane_id)
6         reachable_edges_from_lane = reachable[lane]
7         if len(reachable_edges_from_lane) == 2:
8             # All next routes are reachable
9             return self.choose_route()
10        only_reachable_edge = reachable_edges_from_lane[0]
11        if only_reachable_edge == self.route1.route[0]:
12            # Only the left route is reachable
13            return self.chose_mandatory_route(self.route1)
14        else:
15            # Only the right route is reachable
16            return self.chose_mandatory_route(self.route2)

```

Listing 5.23: Adjustable Rerouter’s algorithm to choose the next route

Even though it was ignored in the above implementation, recall what was stated in Section 5.3.2, particularly in the Aggregate rerouter information item. The new route chosen must be compatible with the previous route of the vehicle. Therefore, choosing the new route must take into account the previous route the vehicle was already traveling. The necessary changes can be seen in Listing A.6

Independent systems

The simulation can contain one or multiple independent systems, and the calibration happens on an independent system-by-independent system basis. Each independent system has its own Machine Learning model (and respective vehicle type), Equation solver (and the respective system of equations), and Rerouters. When calibration happens (at every partial calibration step), the flows of the SUMO calibrators and the probabilities of the Rerouters (obtained by the Equation Solver), and the parameters of the SUMO Vehicle types (obtained by the Machine Learning Model) will be calibrated.

The simulation can contain one or multiple independent systems. To each independent system, there is a correspondent system of equations (see Section 5.3.2) and vehicle type. Therefore, calibration happens per independent system by setting the rerouters probability obtained from the

solution of the system of equations and changing the properties of the vehicle type according to the vehicle model.

Machine Learning model

The Machine Learning model is a significant contribution of this work. More information about the architecture and overall rationale is explained in Section 5.3.4. It is responsible for predicting the coefficients that will be fed to the Equation Solver and the parameters that will be used to change the SUMO Vehicle Type, which represents the vehicle model and affects the driving behavior.

The part of the output of the model associated with the SUMO vehicle type parameters is not used directly. Instead, it works as the multipliers that will be applied to the default parameters. This works as a sort of normalization of the output which is intended to prevent absurd situations due to the order of magnitude of different parameters. For example, if the max speed for the vehicle type is, by default, 120km/h, a value of 0.5 would mean the new max speed is a reasonable 60km/h instead of 0.5km/h, which would be the result of no normalization being applied.

The SUMO Vehicle type parameters ¹⁵ that are subject to calibration and their respective default values can be seen in Table 5.3.

¹⁵https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html

Table 5.3: Default values for used parameters of vehicle type

Variable	Value	Units
Acceleration	2.6	m/s^2
Deceleration	4.5	m/s^2
Apparent deceleration	4.5	m/s^2
Emergency deceleration	9.0	m/s^2
Height	1.5	m
Width	1.8	m
Length	5.0	m
Maximum speed	55.55	m/s
Maximum lateral speed	1.0	m/s
Minimum gap	2.5	m
Minimum lateral gap	0.6	m
Imperfection	0.5	
Scale	1.0	
Speed factor	1.0	
Tau	1.0	

The inputs for the Machine Learning model are the following:

Simulated measurements

The simulated measurements obtained from the Induction Loop Objects (see Section 5.3.3) for each of the edges of the independent system. Note that sensor edges have both the edges' induction loop and the sensors' induction loops (see Section 5.3.2), which differ on where they are positioned within the lane. For those edges, the measurements of the sensors' induction loops are used. This decision is based on the fact that the calculation of the error is done based on the difference between the measurements of the sensors' induction loops and, therefore, their data more relevant.

Target measurements

The measurements are obtained from the real-time data that the simulation is trying to approach. Has the same shape as the simulated measurements. Obtained from the real time data.

Number of lanes

The number of lanes of each of the edges in the independent system.

Additional information

Any additional information (see Section 5.1.2) that is available and can be processed in such a way to be used by a model. In the case of the experiment that took place (see Chapter 6) the data available were the traffic intensities by edge (see Section 6.5). Since each edge has a unique value, these data are analogous to the number of lanes in how it is passed as input (a vector of the traffic intensity for each edge).

Partial calibration step progress

A single fraction from 0 to 1 introduced because of the distinction between the partial calibration step and the calibration step. For example, given a partial calibration period of 200 steps, and a partial calibration step of 50 stepsm, at $t = 0$, $t = 50$, $t = 100$, $t = 150$ and $t = 200$ the partial calibration step progress would be 0, 0.25, 0.5, 0.75 and 1, respectively.

The implementation of the simulation allows for the customization of the inputs and the outputs. That means that specific outputs or inputs can be used or not. Each of SUMO's vehicle type parameters and the vector of coefficients can be included or not as the result of the prediction. Similarly, each individual measurement of the simulated measurements or of the target measurements, the number of lanes, the additional information and the partial calibration step progressed can be included or not as part of the input.

The Machine Learning Model is an optional part of the architecture. Without a model, the coefficients will default to all 0's and the SUMO Vehicle Types will not be calibrated, meaning that SUMO's default vehicle model will be used.

Equation solver

The Equation solver is responsible for outputting the flows of the calibrators and adjusting the rerouter probabilities. It receives the real flow measurements from the Data Loader (see Section 5.3.3) and a set of coefficients from the Machine Learning Model. If absent, the coefficients default to 0. The Equation solver has a system of linear equations and knows which of the variables are known and which ones are free (meaning that they do not depend on any others). The real flow measurements dictate the values of the known variables. There is a coefficient for each of the free variables. As described in Section 5.3.2, the solution is the point in the solution space (if existent) that minimizes the objective function, which is dependent on the coefficients. Given the free variables x_i and the coefficients c_i the objective function can be represented by Equation 5.11:

$$f(X) = \prod_i x_i * c_i \quad x_i \in X \quad (5.11)$$

After the flows of each of the continuous segments in the independent system have been calculated, they can be used to calibrate the simulation. The calibrator flows are assigned to the respective SUMO calibrators. Continuous segments with an entry edge will have the flow of the respective entry calibrator set, resulting in the insertion of vehicles into the simulation. The route of each entry calibrator is one of the entry routes that has been calculated in Section 5.3.2. Given

the edge of the calibrator *edge_id*, the end time of the simulation *end_time*, the calculated flow *flow*, the vehicle type of this independent system *type*, and considering $|x|$ represents the value of variable x , the function will be called as seen in [Listing 5.24](#).

```
1 traci.calibrator.setFlow("ca_|edge_id|", 0, |end_time|, |flow|, -1, "|type|", "
   r_entry_|edge_id|", "free", "max")
```

Listing 5.24: Function call to set flow of the SUMO edge's entry calibrator

We shall also take the opportunity to talk about how the Sensor Flow's (see [Figure 5.3](#)) can be used to directly set the flows of the SUMO's calibrators. As mentioned in [Section 5.3.2](#), the sensor's flow calibrators can be used to ensure that the independent system is calibrated in spite of the other systems' calibrators. Expanding on this, the only edges that are influenced by the rest of the simulation are the sensor edges, which are also entry edges to the independent system. Therefore, if we so choose (we might want to test the performance of the simulation without inserting or removing vehicles in edges other than entry or exit edges of the whole network itself), we can also calibrate the sensor flow calibrators. Given the id of the calibrator *calibrator_id*, the end time of the simulation *end_time*, the sensor flow obtained from the Data Loader *flow*, the vehicle type of this independent system *type*, the route of this sensor flow calibrator *route* (see [Section 5.3.3](#)), and considering $|x|$ represents the value of variable x , the function will be called as seen in [Listing 5.25](#).

```
1 traci.calibrator.setFlow("|calibrator_id|", 0, |end_time|, |flow|, -1, "|type|", "|
   route|", "free", "max")
```

Listing 5.25: Function call to set flow of the sensor flow calibrator

The route probabilities are also assigned to each of the Rerouters. For each continuous segment with a split edge, the respective Rerouter will have its probability set. This is done by updating the rerouter with the flows of each of the outgoing edges of that split edge. Internally, the Rerouter will take both of the flows and derive the probability of each of the edges being chosen by a vehicle.

Training

Training of the model happens after each partial calibration period. First, the simulated measurements are collected in the same fashion as they were collected to do the prediction of the parameters. The target measurements are the same as the ones used to make the calibration.

Data collection

The simulation automatically collects some data during its execution. SUMO induction loops are required to have a file to which the information is automatically saved. SUMO calibrators can have an output file defined, but the outputted information will be organized per flow interval. The

interval for the flow of the calibrators is from the start of the simulation to the end of the simulation. Therefore, each calibrator only outputs the data of the whole simulation. To save the more informative data of the calibration for each partial calibration step, the `SimulationRunner` will instead take over saving the information of the calibrator. The count and hourly rate of vehicles passing the calibrator, inserted by the calibrator, and removed by the calibrator, are saved. Analogously, the count and hourly rate of vehicles that departed, teleported, and arrived for the whole simulation are also saved. An implementation similar to the Induction Loop Objects, but that we shall not analyse since it does not influence the calibration, is used for keeping and saving the data corresponding to the calibrators and the simulation. These data can be used to deepen the analysis of the simulation.

5.3.4 Modeling

In Section 5.3.3 we abstracted the Machine Learning model used. Remember the model predicts the coefficients and the multipliers for the vehicle type parameters, which we will jointly refer as predictions, for a given independent system. In the end, the performance of the model is assessed by comparing the simulated measurements with the target measurements for said system. First we shall explain how we arrived at the methodology for the training so that the choice of architecture is clear.

The training of the machine learning model is heavily conditioned by the context of this work. First, the shapes of the predictions and the true values available are different. That is to say that the sizes of their vectors are different. This is because the model is but an intermediary step, with the final output, the simulated measurements, being obtained by the simulation. Its counterpart, what would be regarded as the true values, are the target metrics. Additionally, we assume that training will happen concurrently with each prediction, which means that it will only take place with each partial calibration step (see Section 5.3.3). This poses a problem: the training will be slow. For each iteration of the model, multiple steps of the simulation must have happened first. That said, three alternatives were considered for the training:

- Reinforcement Learning.
- Genetic Algorithms.
- Supervised Learning.

Reinforcement Learning is a good representation of the paradigm in the sense that the model is inserted within an environment (the simulation) and its prediction will reflect on its final state (by calibration). It offers the advantage that the training is independent from the shapes of the prediction and of the output. However, each prediction is viewed independently from previous predictions.

Genetic Algorithms' biggest flaw is that they would require multiple simulations running at the same time. Due to the heavy computational load expected this approach was disregarded.

Finally, Supervised Learning has the aforementioned flaw of a mismatch of the predicted values and the true values.

The approach chosen tries to solve the flaw of the Supervised Learning method by forcing the shapes of the outputs to be the same. Since the target measurements, the true values, have a fixed shape, it will be the predicted values that we will manipulate.

If we wanted a model to simply predict the simulated measurements without using SUMO, a Supervised Learning approach would be trivial. The training would happen by comparing the simulated measurements with the target measurements. Our approach seeks to do that by having the model work as a sort of autoencoder with an input layer receiving the data for the prediction (see Section 5.3.3), the latent space has the shape of the wanted predictions (see Section 5.3.3), and the output shape has the shape of the measurements (which is the same for the simulated and the target measurements).

Since the output of the prediction by the autoencoder has 0 correlation to the simulated measurements, in order to perform the training, we must instead fabricate the correlation between the predicted and the true values from the simulated measurements and the true measurements. This is done by computing the true values. The true values will result from summing the difference between the simulated and the target measurements to the predicted values. Therefore, for each one of the values, the difference between the predicted and the true value is the same as the difference between a given simulated measurement and target measurement. The possible differences in the order of magnitude from the predicted values to the true values in comparison with the simulated measurements to the target measurements are not accounted for, and it is possible to hinder the training of the model. After determining the predicted values and the true values, any supervised learning training routine can be employed through gradient descent. For the experiment (see Chapter 6), the Adam optimizer was used with a mean squared error loss.

Additionally, we can conclude that the encoder of the autoencoder has no real purpose. Therefore, it can be reduced to a single untrainable layer where all of the weights are 1's, and all of the biases are 0's.

Therefore, we replicate the difference between the simulated and the target measurements in the predicted and the true values. More concretely, first, the differences between the simulated measurements and the target measurements are computed, which are the values that really indicate the performance of the network. Next, we add these differences to the autoencoder predictions (as mentioned before, the shape of the output is that of the simulated measurements) to obtain the pretend true values. Finally, calculate the loss between the predicted values and the pretend true values. Training can then be performed as normal.

One of the possible advantages of this Supervised Learning-like training is that a methodology for the fine-tuning or adaptation of existing Neural Networks for model-less approaches (see Section 2.1.2) appears plausible, even with an architecture that is not that of an autoencoder.

The model is, therefore a dense autoencoder where the input and the latent space shapes are defined according to the configurations of the model (see Section 5.3.3, and the output shape is invariable and determined from the number of values of the measurements (which is equal for

the simulated and targets measurements). The decoder is a single untrainable layer, with every weight being set to 1, and every bias being set to 0, to connect the latent space to the output. The architecture of the encoder can be freely chosen as long as the shapes of its input and output (the shapes of the input and latent space of the autoencoder) are respected. The prediction of the model is not the values of the output layer but, instead, the values of the latent layer. In Figure 5.4 there is an illustration of the architecture of the described machine learning model, where $J_{o,l}$ is an all-ones matrix.

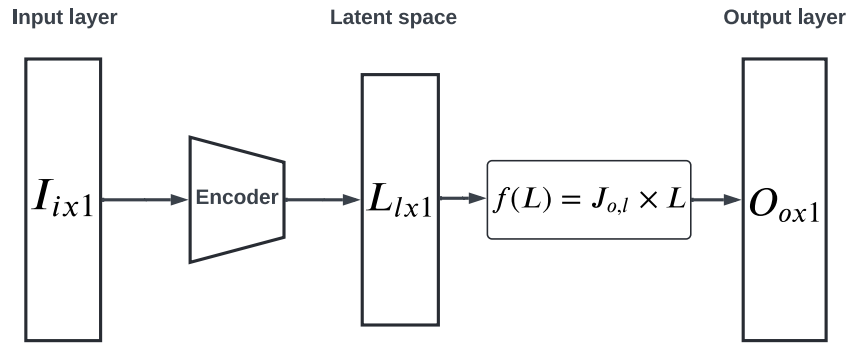


Figure 5.4: Architecture of the machine learning model

The performance of the model training method is to be analyzed for the experiment (see Chapter 6).

5.4 Assessment

As described in Section 5.1.3, the objective of the simulation and of the methodology is to minimize the difference between the simulated measurements and the target measurements. Indeed, it is by comparing these that the performance of the calibration will be assessed. As stated previously, the specific measurements considered are the flow and speed of the vehicles. Overall, the mean absolute error (MAE) and mean absolute percentage error (MAPE) can be used to interpret the simulation results as they provide a simple yet insightful perspective. The differences in traffic between peak and off-peak hours should be considered when comparing the two metrics. Additionally, the mean squared error (MSE) and relative squared error (RSE) can be used to confer more importance to larger disparities in the measurements. This should be done mostly when different simulations obtain similar performances regarding the mean absolute error or mean absolute percentage error, and the focus changes to fine-tuning the simulation.

Furthermore, the assessment should be made in multiple equivalent intervals, be it an interval of a few hours in multiple different days or an interval of certain days of the week in multiple different weeks. As such, there are three independent variables: time within the interval, sample (or the interval chosen), and sensor. This means that the performance can, for example, be compared between different samples, during the interval, or per sensor.

Note that the current methodology is evaluated by comparing the measurements taken in an interval by the sensors and the same measurements obtained in the simulated for the period immediately after. This effectively means that our simulation suffers a delay corresponding to the calibration period. For example, the measurements acquired from 16:55 to 17:00 will only take effect in the calibration of the simulation by 17:00 and until 17:05 (considering a calibration period of five minutes). It would be possible, however, to use the present machine learning architecture to do the calibration as a prediction of the next interval. This approach has the downside that the system of linear equations whose solution space is deterministic would no longer be representative of reality. Approaches would have to be devised to insert the uncertainty and prediction aspect into the solving of the system of equations.

Lastly, it is important to take notice of the time performance of the simulation. Put more simply, the simulation must run, at least, as fast as the real world, and one step in the simulation must take less time than the amount of time a step corresponds to. Since the simulation is devised to work based on real-time data, reducing the computation time infinitely does not bring any benefits since the simulation should not proceed until new data is received. Therefore, though it can be insightful to record the time taken for the simulation with different methods or models, knowing that the simulation ran in less time than the period of time simulated is enough.

Depending on the scale of the machine learning models (see Section 5.3.4), training might slow down the simulation. Even though the training of the models can be done offline and, therefore, it isn't necessary to conform to any time requirements, the faster the simulation, the faster the training will be performed.

5.5 Summary

In this section, we described the problem and specified all of the work needed to prepare the simulation, to run it, and to calibrate it while also exploring a Machine Learning approach. The objective is, therefore, to approximate the state of the real system by minimizing the difference between the measurements that result from the simulation and the real measurements from either historical data or real-time data. The preparation of the simulation is extensive. First, the network needs to be created. Next, using the network, we can create both the SUMO files that are necessary for the SUMO simulation and other data files that are required for our implementation of a simulation on top of SUMO. Additionally, multiple independent systems are created in order to increase the modularity of the simulation.

Our implementation has several working parts that use the online data to calibrate the simulation with a given periodicity. This is done both by targeting the flows of the vehicles (the rate at which they are inserted or possibly removed and the routes that they take) and their driving behavior (by changing the vehicle types). The calibration of the driving behavior is done exclusively with a machine learning model reliant on neural networks that take data from the simulation and real measurements. The training and architecture of this model are also explained. Finally, the

methodology for assessment of the performance shall be done by comparing the target and the simulated measurements with the chosen metrics.

Chapter 6

Experimental setup and results

To validate the methodology, and given the data available in the context of the present work, the "Via de Cintura Interna", or VCI, the inner ring highway in Porto, Portugal was chosen ¹. The road network includes the totality of the circuit around the city, the on and off-ramps, and interchanges with other highways.

6.1 Network

Following the methodology present in this work (see Section 5.3.1, multiple .osm files were used to encapsulate the road network of interest. The SUMO network obtained through netconvert included 13831 nodes, 23 types of edges, and 26067 edges.

Next, we apply filtering to obtain just the relevant roads in the VCI inner ring. First we apply a filtering by the names of the edges:

- no do areinho.
- via de cintura internal.
- vci.
- no de coimbroes.
- ic23.
- via cintura interna.

With this step alone, we get a network with 204 nodes, 2 types of edges, and 195 edges. Afterward, we also use the additional strategy described where the type and their order of adjacency are used. The depths per type of edge can be seen in Table 6.1.

As discussed in Section 5.3.1, the specific implementation of this step influences which edges are added to the base network.

¹ https://pt.wikipedia.org/wiki/Via_de_Cintura_Interna

Table 6.1: Depth for each edge type for filtering of the experiment network

Type	Depth
highway.motorway	-1
highway.motorlink	1
highway.primary	2
highway.primary_link	1
highway.secondary	2
highway.secondary_link	1

In this case, we treat every type with the same depth together. As can be seen by the [Table 6.1](#), edges of type highway.motorway with a depth of -1. For these edges, adjacency shall be recursively calculated until no more edges are found. Afterwards, we shall, for each depth, calculate the adjacent edges according. The new edges are only added at the end so that different depths do not influence the processing of the others. The pseudocode of the specific implementation can be seen in [Listing 6.1](#).

```

1 def filter_edges_by_type(initial_edges: set[Edge], edges_types_depth: dict[str, int]
2     ) -> set[Edge]:
3     keys_with_infinite_depth = {key for key, value in edge_type_depth.items() if
4         value == -1}
5
6     edges = get_adjacent_edges(edges, edges, keys_with_infinite_depth, -1, 0)
7
8     extended_edges_by_depth = set()
9     depths = set(edge_type_depth.values())
10    for depth in depths:
11        keys_with_depth = {key for key, value in edge_type_depth.items() if value
12            == depth}
13        this_depth_extended_edges = rec_get_adjacent_same_type_edges(edges, edges,
14            keys_with_depth, depth, 0)
15        extended_edges_by_depth = extended_edges_by_depth.union(
16            this_depth_extended_edges)
17
18    # Only add the extended edges in the end so each depth does not influence the
19    others
20    edges = edges.union(extended_edges_by_depth)
21    return edges
22
23 def get_adjacent_edges(all_edges: set[Edge], to_get_adjacent_of: set[Edge],
24     edge_types: set[str], max_depth: int, current_depth: int) -> set[Edge]:
25     if len(to_get_adjacent_of) == 0:
26         return all_edges
27     if current_depth == max_depth:

```

```

21     return all_edges
22     new_adjacent_edges = set()
23     for edge in to_get_adjacent_of:
24         adjacent_edges = get_adjacent(edge)
25         this_edge_adjacent_edges_of_type = {edge for edge in adjacent_edges if edge
26         ._type in edge_types}
27         this_edge_new_edges = this_edge_adjacent_edges_of_type.difference(all_edges
28         )
29         new_adjacent_edges = new_adjacent_edges.union(this_edge_new_edges)
30
31     all_edges = all_edges.union(new_adjacent_edges)
32     adjacent_same_type_edges = get_adjacent_edges(all_edges, new_adjacent_edges,
33     edge_types, max_depth, current_depth+1)
34     return adjacent_same_type_edges
35
36 def get_adjacent(edge: Edge):
37     return set(edge.getOutgoing()).union(set(edge.getIncoming()))

```

Listing 6.1: Algorithm for computing adjacent edges according to type and debth

Table 6.2: Number of lanes per edge for experiment network

Number of lanes	How many edges
1	95
2	66
3	88
4	59
5	9

By keeping only the largest connected component, we obtain the final network, which presents 308 nodes, 2 types of nodes, and 317 edges.

The number of lanes per edge can be seen in [Table 6.2](#). There are 54 split edges, 54 merge edges, 46 entry edges, and 46 exit edges.

The final network can be seen in [Figure A.1](#).

6.2 Network information

After the final network is assembled, the developed implementation can be used to quickly obtain all of the other information to run the simulation.

The sensor information provided in the context of this experiment encompassed not only the target area but also some other surrounding highways. Therefore, it was first needed to curate which sensors would be included in the simulation. To do this, The coordinates were inserted into Google Maps to confirm if the position of the sensors was within the roads being studied. It is important to know that each of the sensors covers both directions of traffic traveling on the highway. In the traffic data, an additional column called direction has a value of either "C" or "D" depending on whether the traffic is flowing in an increasing (from the Portuguese "crescente") or decreasing (from the Portuguese "decrecente") road kilometer, respectively. Knowing that two sensors are needed, the sensors' ids will be *lidl_C* or *lidl_D*. To account for these changes, all of the relevant data is preprocessed to include a new column with this id.

When the sensors information is calculated, the prefixes *A* and *B* are given, as it is impossible to automatically know to which direction an edge corresponds. Because of that, a manual effort had to be employed to obtain the right id of each of the sensors. To do this, the direction of a road was obtained in Google Maps. Afterward, by visualizing the edges of the sensors in netedit, it was possible to determine if the edge was an increase edge, and the respective sensor should have suffix *C*, or a decreasing edge, and the respective sensor should have suffix *D*. For the small number of sensors, this was not too time-consuming. For a much bigger network it is possible that it could become a bigger task. However, given that it just needs to be done one time, no automation was devised.

Using the methodology described, 24 systems were found (see [Appendix A.1](#)). The number of continuous segments and edges per independent system can be seen in [Table 6.3](#).

Table 6.3: Network details of independent systems

System number	Number of continuous segments	Number of edges
1	44	67
2	32	57
3	11	19
4	13	18
5	13	18
6	11	16
7	9	15
8	13	15
9	13	14
10	7	11
11	5	10
12	7	9
13	7	9
14	7	9
15	5	8
16	5	7
17	5	7
18	5	6
19	5	6
20	3	5
21	3	5
22	5	5
23	3	4
24	3	3

It is important to note the differences in size between the different systems resulting from the positioning of the sensors throughout the network. Particularly, system 1 is a highway interchange where sensors are only present in the intersecting highways and, therefore, do not further partition the interchange itself. On the other hand, systems with 3 edges mean that there is an exit or a merge between two sensors (2 of the edges are those of the sensors). In practice, this means that the solution for the respective equation system will be deterministic and no objection function coefficients must be calibrated. In such systems, the model will only calibrate the vehicle type parameters.

6.3 Dataset

The data provided relates to the years 2014 and 2015 and includes 26 bidirectional sensors (each sensor covers both directions) with readings every 5 minutes. Only 15 of those were within the target network and were used for data. The data was preprocessed so that only times, when all of the sensors had readings, were kept, and a column with a unique ID corresponding to one of the directions of a sensor.

In this experiment, we will focus mainly on the speed and flow of traffic. During the analysis, the distinction between the average speed of the vehicles and the average speed of the interval will be made. Since the readings are periodic, the average speed of the intervals does not account for the flow during the interval. For intervals where no vehicles were recorded the speed is considered as 0. Other metrics such as length of vehicles, occupancy of sensors, the class of vehicles ², axle class, number of lanes, and the ratio of "light" vehicles is also recorded.

A total of 60468 readings for each of the sensors exists. Additionally, the count of readings per time of the day can be seen in [Figure 6.1](#). The number of readings is much less in periods of low traffic. These readings do not reflect the number of cars, but merely the periods when all of the 15 selected road sensors were active. It can be hypothesized that maintenance on the sensors is performed during these times.

²<https://eportugal.gov.pt/en/cidadaos-europeus-viajar-viver-e-fazer-negocios-em-portugal/veiculos-e-conduzir-em-portugal/regras-de-transito-e-mobilidade-em-portugal>

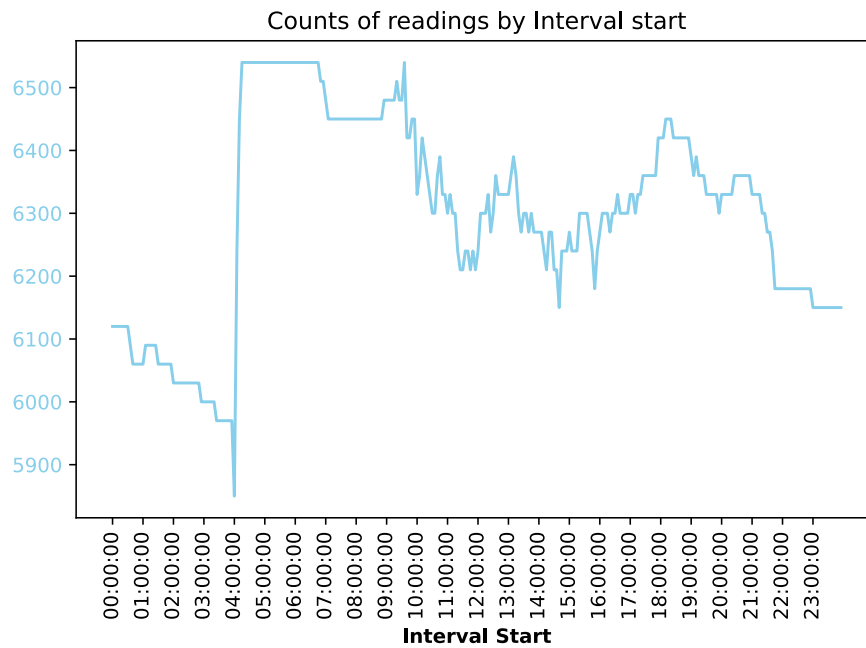


Figure 6.1: Count of all samples by interval start

An analysis of the flow and speed was also performed. Overall, the average hourly flow is 2045 vehicles per hour, with an average interval speed of 81.5 km/h and an average vehicle speed of 77 km/h. We interpret the average interval speed as the arithmetic average of speed values collected by the sensors, whereas the average vehicle speed weights those values with the number of vehicles. Taking the average of the flows by sensor id, we obtain [Figure 6.2](#). Additionally, the average of the speeds of the vehicles and of the intervals are compared in [Figure 6.3](#). Slight differences can also be seen in both of the speed averages.

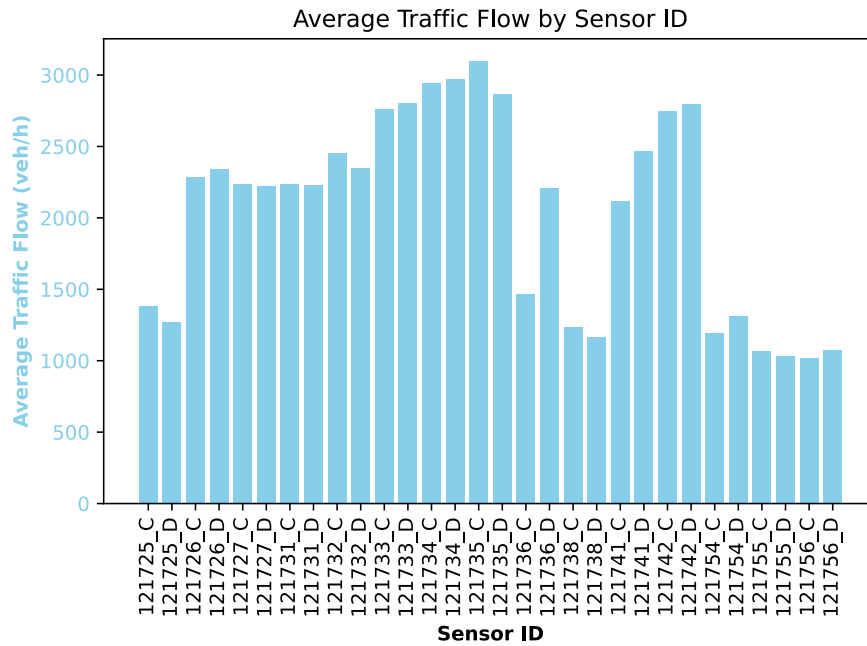


Figure 6.2: Traffic flow by sensor id

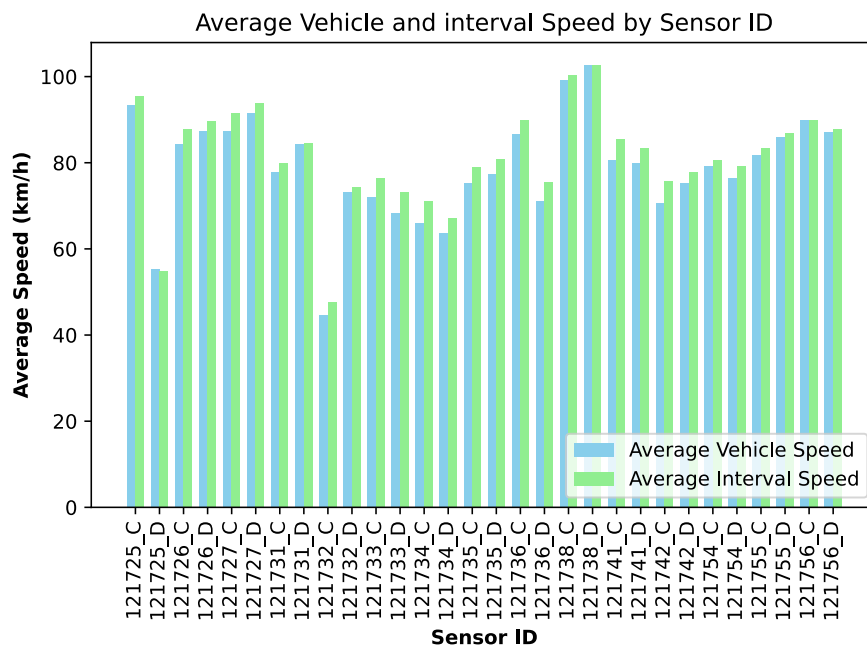


Figure 6.3: Comparison between average speeds

To gain some further insight, an analysis of the distribution of both flows and interval speeds can be visualized in [Figure 6.5](#). Notice how, for most sensors, there appears to be a lot of time periods when traffic flow is either intense or mild. On the other hand, the interval average speed

of some of the sensors is either very low or very high, with a lack of intermediary values. The negative values that are shown are most likely a caveat of the representation.

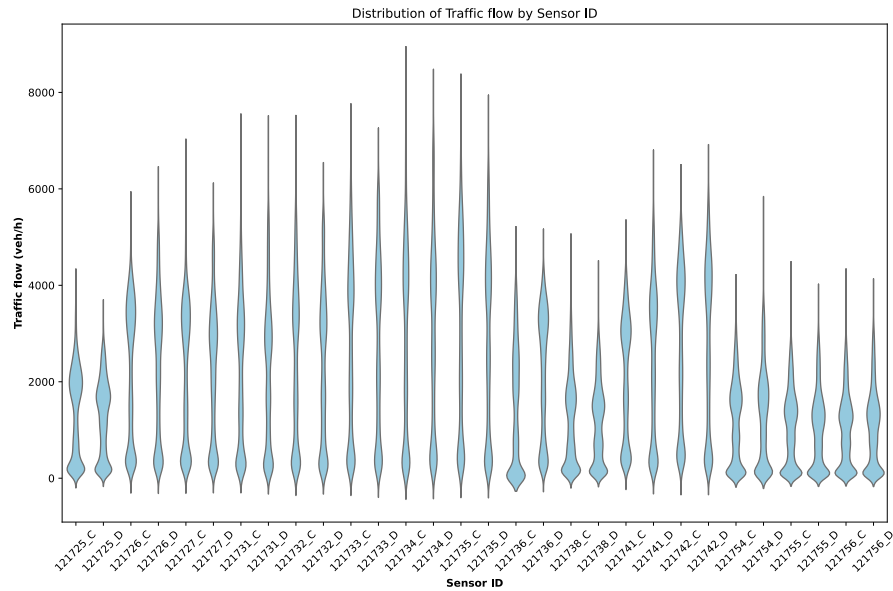


Figure 6.4: Distribution of flows by sensor id

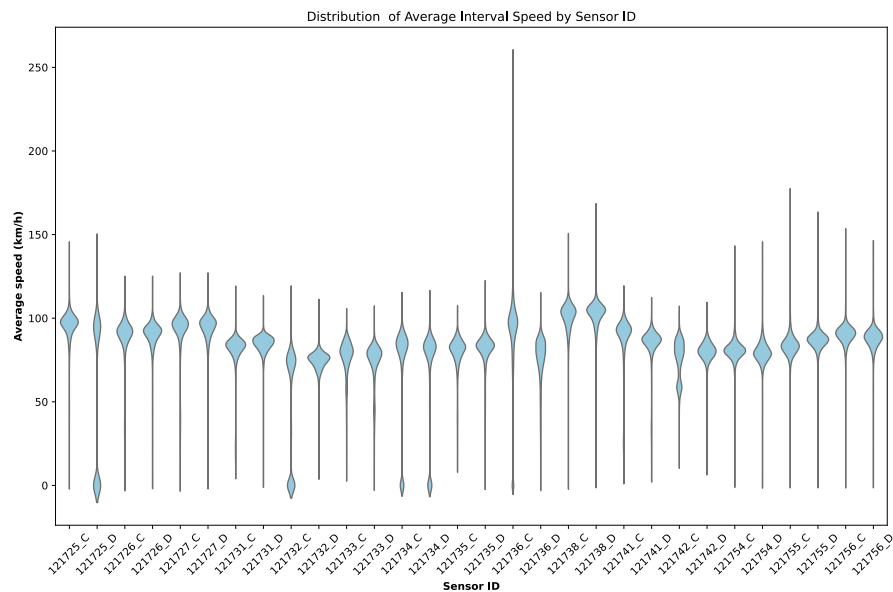


Figure 6.5: Distribution of speed of intervals by sensor id

Another comparison can be made by correlating the speed and traffic flow across time (Figure 6.6, and Figure 6.7). There is generally a decrease in speed with an increase in traffic, and vice-versa. Also, for this application, the interval average speed and the vehicle average speed are slightly different but highly correlated, as is clear in Figure 6.8.

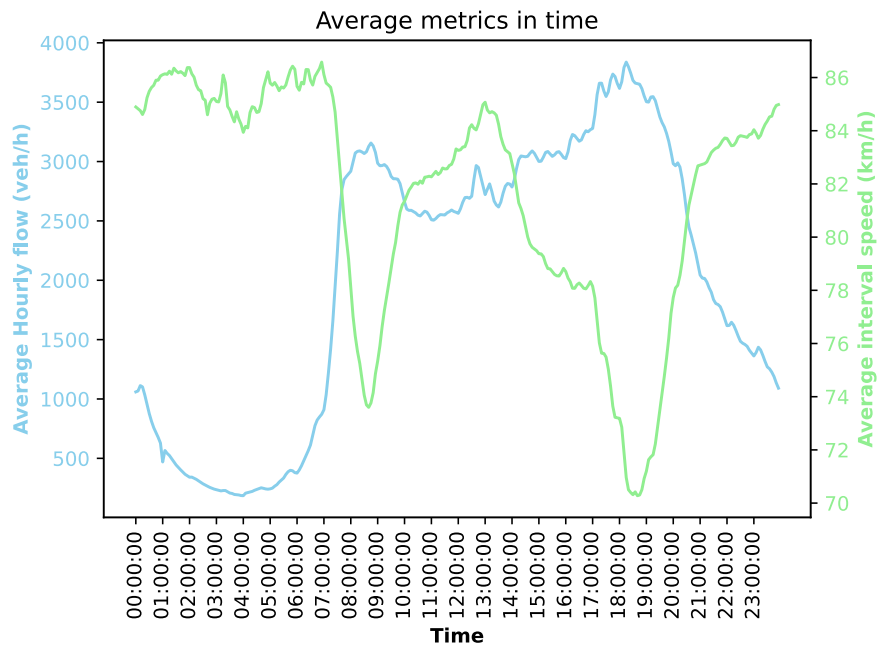


Figure 6.6: Average flow and interval speed in time

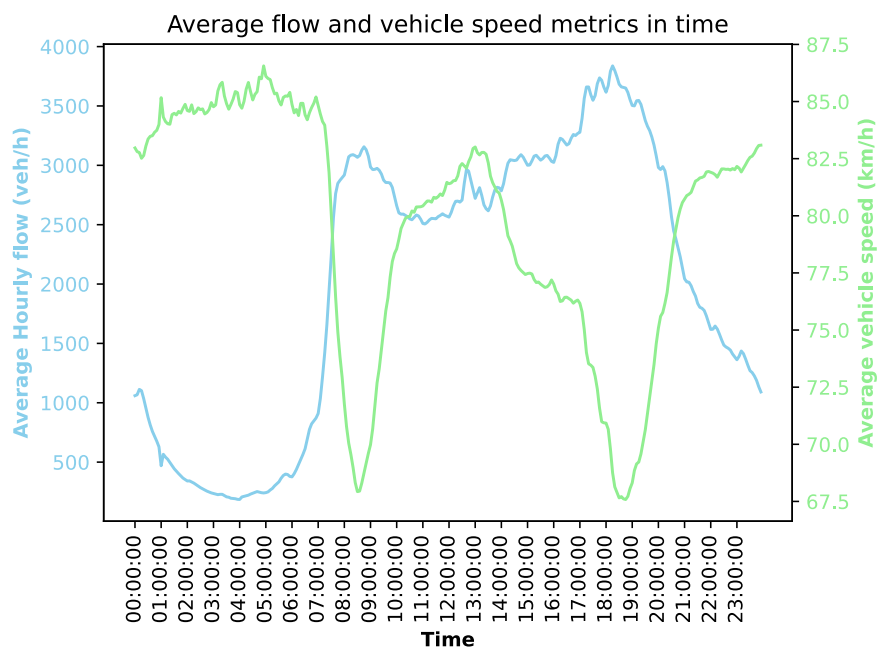


Figure 6.7: Average flow and vehicle speed in time

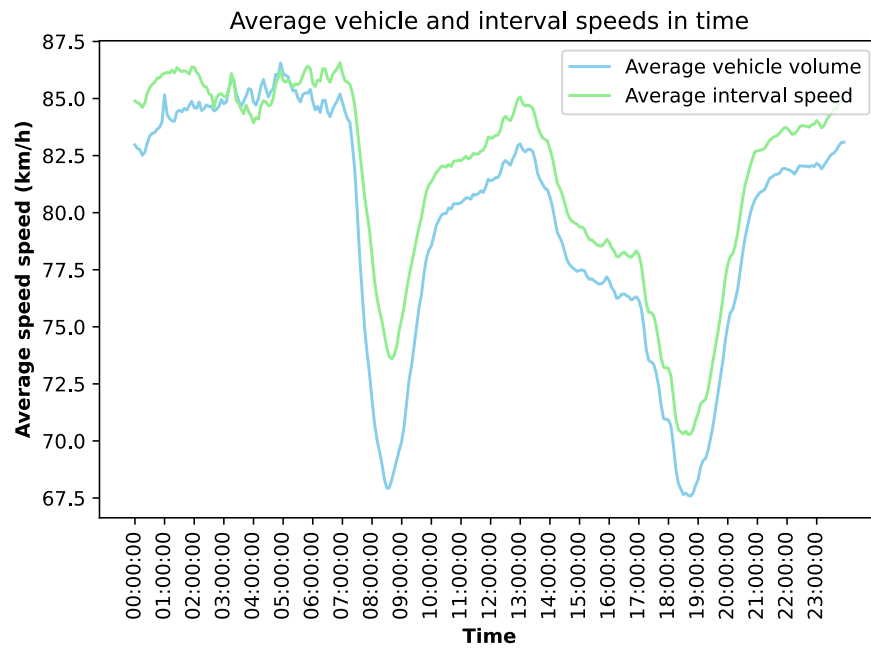


Figure 6.8: Average interval and vehicle speed in time

To run the experiment for a 24-hour period would imply fewer samples (a contiguous period for which there is no missing sensor data) would be available to train the machine learning model (see Section 6.6), even if, overall, more data points (a reading from all of the sensors) would be used. Additionally, each sample would take longer to run and contain a wider variety of conditions that a smaller model might have difficulties learning from. Furthermore, the additional information had to be collected manually (see Section 6.5) and a longer period would signify a significant increase in the manual work required. Therefore, the period between 17:00 and 19:00 was chosen as the focus of this study, corresponding to the evening rush hour, with the greatest amount of flow. Different samples were collected from the data, where all of the relevant sensors had continuous readings during the whole interval. 52 samples were collected, of which 41 were dedicated to training and 11 for validation.

The same analysis can be done to these intervals as was done to the whole dataset. For example, the comparison of the average vehicle and interval speeds during the target period (see Figure 6.9) gives us a segment of the original graph (see Figure 6.8).

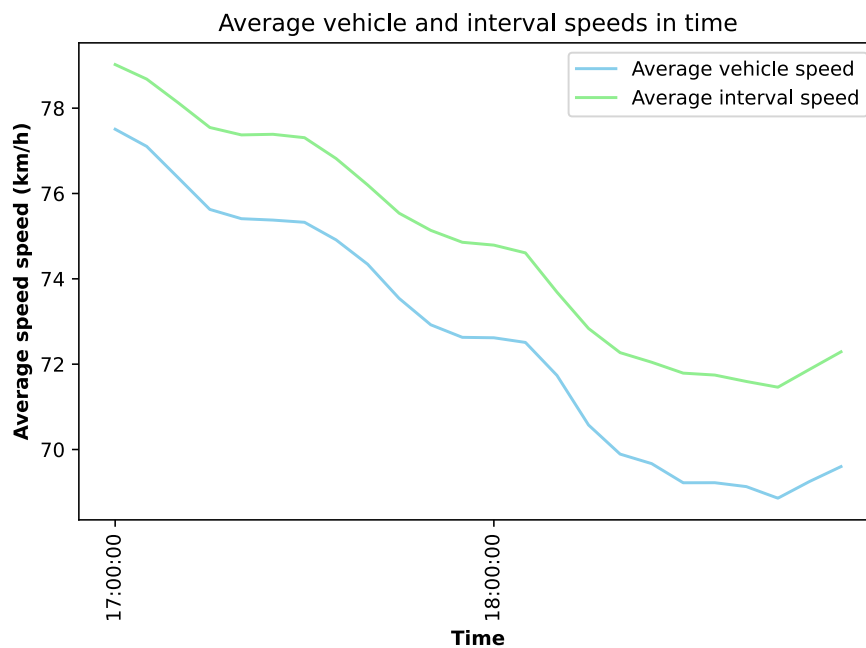


Figure 6.9: Average interval and vehicle speed in time for 17:00-19:00 period

Additionally, we can also perform an analysis in relation to the specific sample, that is to say, the day of the data. For example, Figure 6.10 gives us a comparison between the average vehicle and interval speeds for every given day featured.

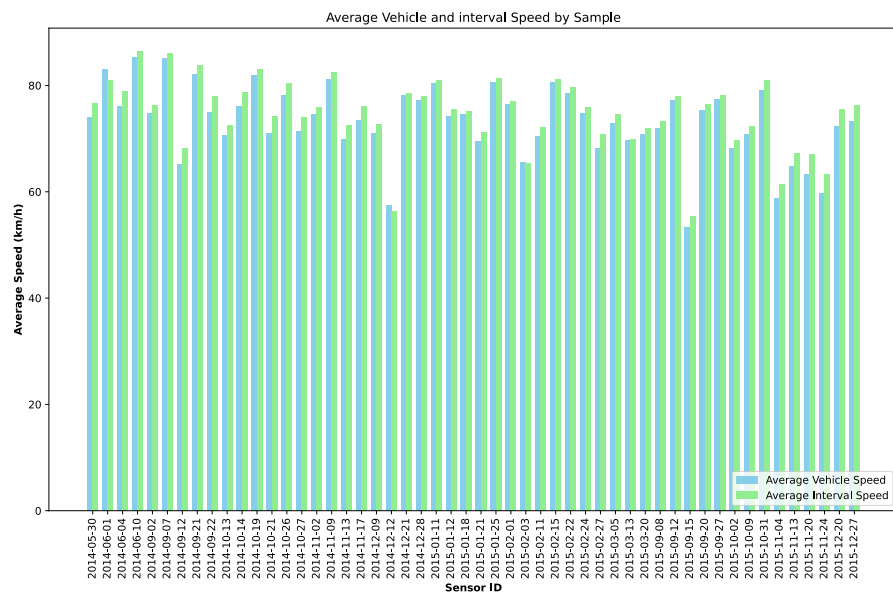


Figure 6.10: Average interval and vehicle speed in time for 17:00-19:00 period

6.4 Results baseline

Before proceeding with the training and the testing of the models, we shall establish a baseline for the performance of the methodology by running the simulation with no model and relying solely on the calibration of the flows and the rerouters (see Section 5.3.3). We perform the validation just for system 1 (see Figure A.2). Just one system was chosen due to how computationally expensive the training and validation of the different models would be for the whole system. System 1, being the biggest one obtained is therefore a good candidate to assess the methodology. The results can be seen in Figure 6.11, Figure 6.12, and Figure 6.13). Note that since multiple samples and multiple sensors are included for each point time, the values represented and respective metrics consider the average of all points of data for that time. For other graphics where the independent variable is different an analogous calculation was performed.

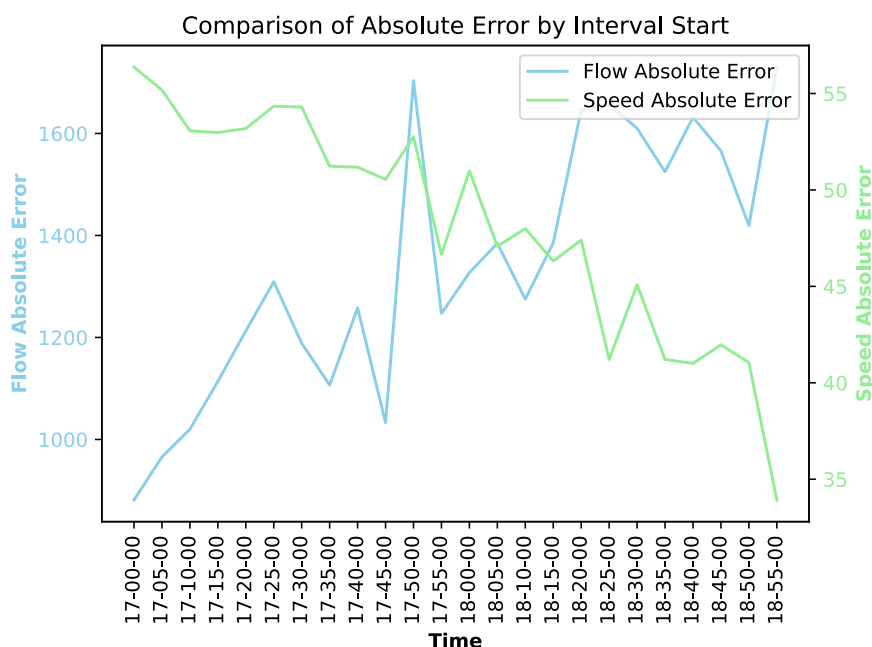


Figure 6.11: Absolute error for validation with no model in time

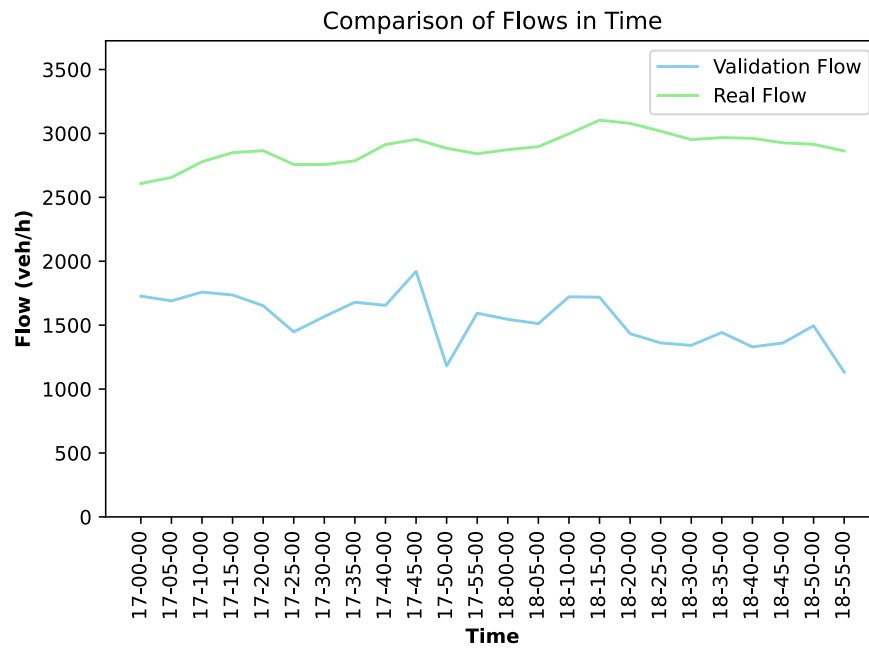


Figure 6.12: Comparison of real volume with simulated volume for validation with no model in time

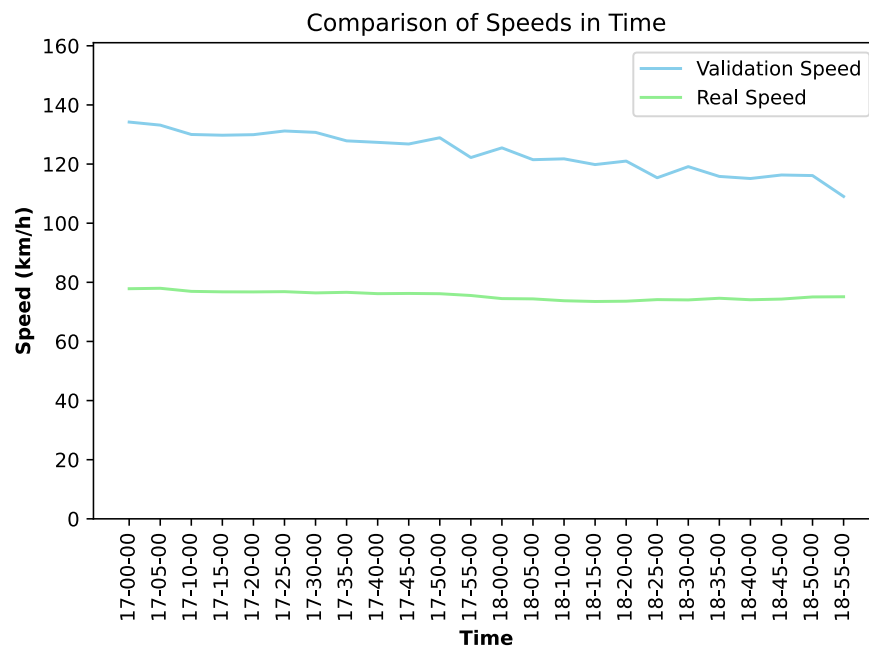


Figure 6.13: Comparison of real speed with simulated speed for validation with no model in time

The results indicate that the flow of the vehicles is smaller than expected, while the speeds are higher than the real data. In fact, the speeds go over the speed limit on the real road. However, it appears that the detected speeds are coincident with the max speed of each edge in the network.

Table 6.4: Training configurations of the machine learning models

Model number	Input			Output	
	Partial calibration step	Partial calibration step progress	Intensities	Coefficients	Vehicle Type Parameters
1	30	No	No	Yes	All
2	30	No	No	False	Speed
3	30	No	No	False	Speed; Speed factor
4	30	No	No	Yes	None
5	30	Yes	No	Yes	None
6	30	No	Yes	Yes	None
7	30	No	No	Yes	Scale

Given that the simulation with no model has no way to control the speed according to the real data, a disparity in this measurement is to be expected. Other relevant graphics are in Appendix A.3.1.

6.5 Additional information

As mentioned in Section 5.1.3, additional information can be leveraged to calibrate the simulation. This methodology proposes that information to be used in the predictions of the Machine Learning models (see Section 5.3.3). For this experiment, the traffic qualitative speeds gathered from Google Maps³, which classify color-code the speeds ranging from fast to slow in 4 levels, were associated with each edge in the network and served as input to the Machine Learning model. Each of the levels was associated with the values 0.25, 0.5, 0.75, and 1, from the fastest to slowest traffic, which we shall refer to as the intensities.

These data were collected just for one point in time since no automated procedure to retrieve the data from Google Maps was found. The intensities were collected for Friday at 18:30, within the selected period of 17:00 to 19:00, since a higher variety in intensities was visually observed. Ideally, the traffic intensities for every point in time during the simulation period should be collected.

6.6 Training

Since our methodology uses a Machine Learning model (see Section 5.3.3) it is necessary to train said model. A series of experiments were run with different configurations of the model to see if they made any significant difference in the performance of the simulation. In order to decrease the computational load, only the independent system 1 was used for training (see Figure A.2). The configurations for each of the models trained can be seen in Table 6.4. For more information on what each of the configurations do, refer to Section 5.3.3.

The architecture of the encoder is a dense neural network with 6 layers. The number of neurons in each layer can either be specified or determined automatically such that the number of layers follows a geometric series starting on the number of inputs and ending on the size of the latent space. For example, given an input size of 218 and a latent space size of 30, the dimensions of the layers would be 156, 112, 80, 58, 41, 30.

³google.com/maps

6.7 Results and discussion

Taking Table 6.4 as a reference, we try to tackle the disparity in flows with models 4, 5, 6, and 7 and the disparity in speeds with models 2 and 3. Model 1 tries to tackle both at the same time since its output is both the coefficients and the vehicle type parameters.

Models 2 and 3 were able to successfully reduce the difference between the real speed and the validation speed, even though model 2 performed better in that regard (see Figure 6.14 and Figure 6.15). As expected, none of these models influences the volume of the vehicles significantly.

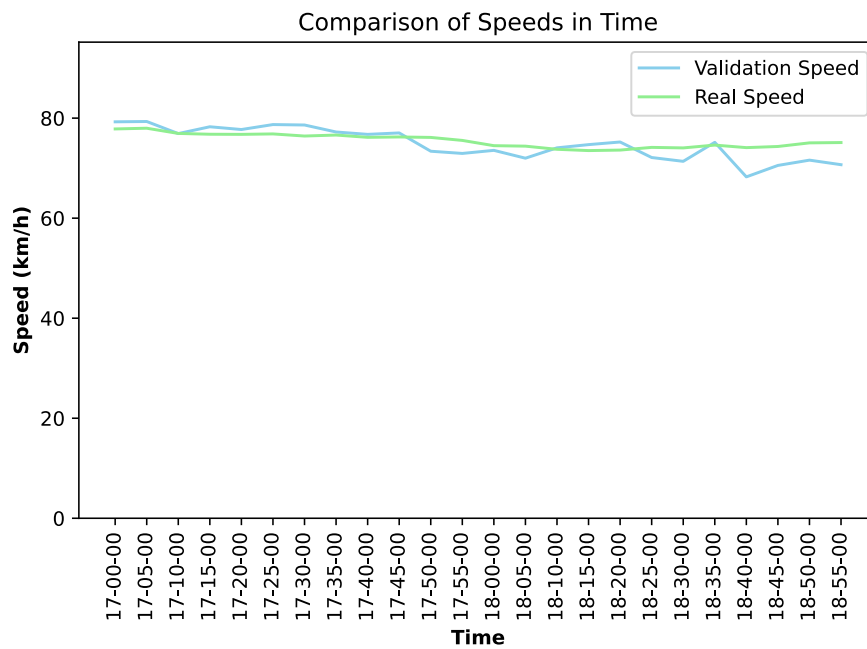


Figure 6.14: Comparison of real speed with simulated speed for validation for model 2 in time

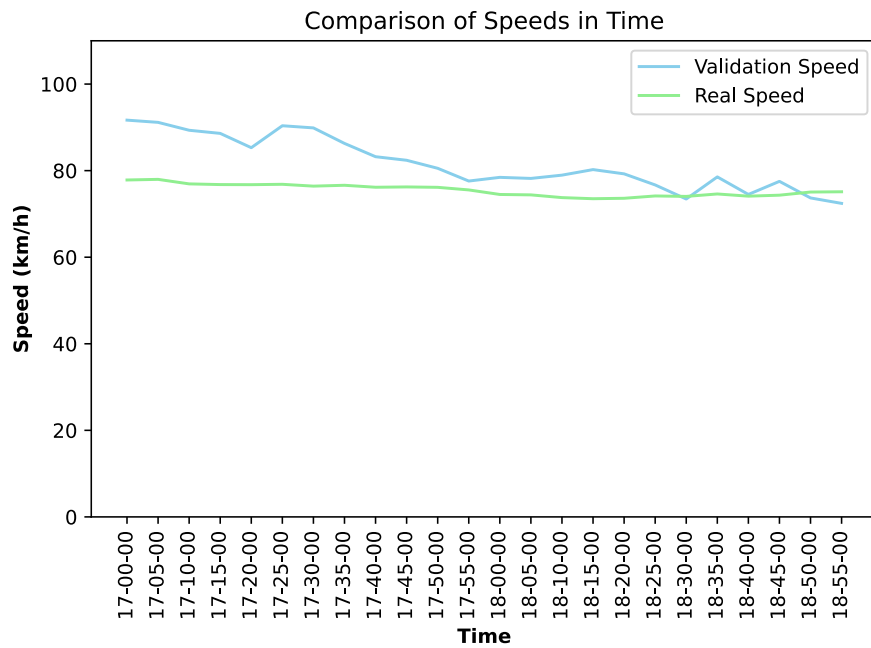


Figure 6.15: Comparison of real speed with simulated speed for validation for model 3 in time

Models 4, 5, 6, and 7 have a minimal effect in the flow. The four models, however, appear to decrease the absolute percentage error, as can be seen by the fact that the minimum and maximum flow errors in Figure 6.16, Figure 6.17, Figure 6.18, Figure 6.19 are smaller than those in Figure A.42. As expected, none of these models influence the speed of the vehicles significantly.

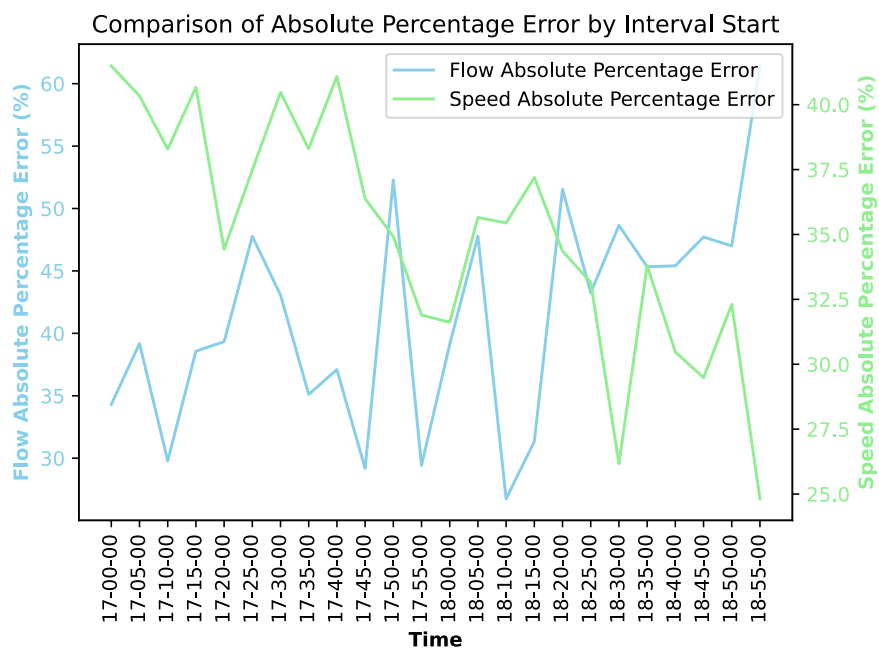


Figure 6.16: Absolute percentage error for validation for model 4 in time

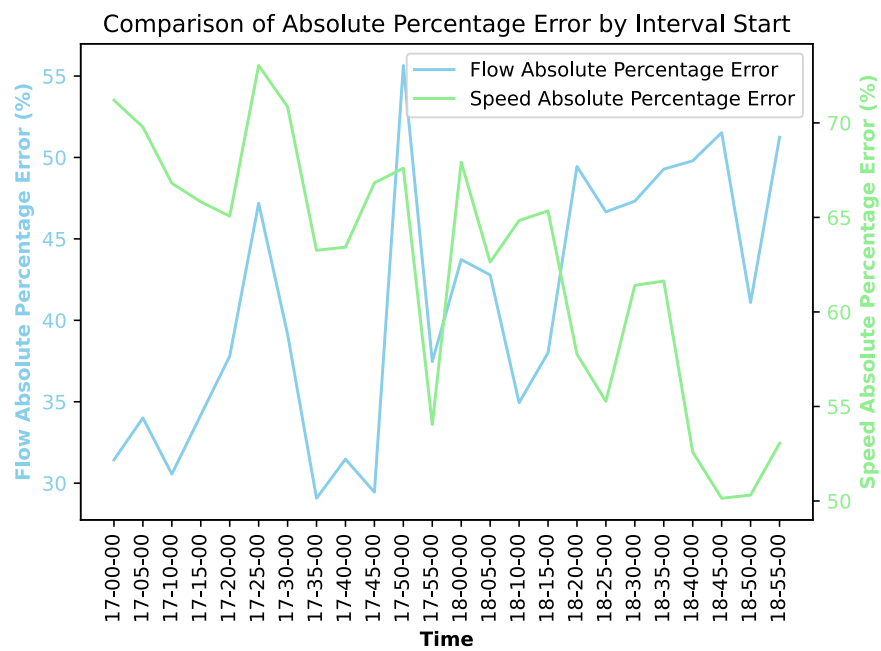


Figure 6.17: Absolute percentage error for validation for model 6 in time

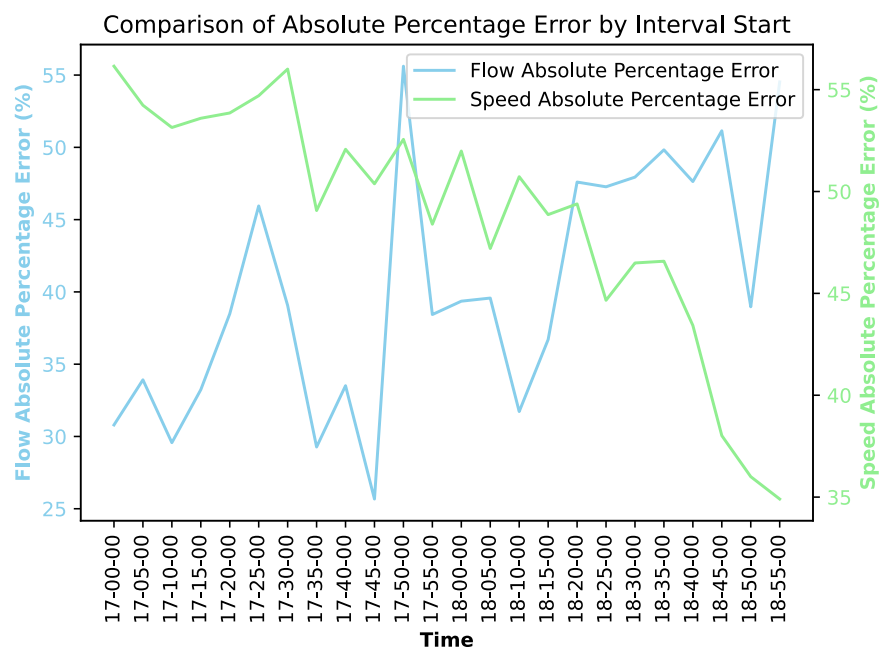


Figure 6.18: Absolute percentage error for validation for model 5 in time

Table 6.5: All metrics for the validated models. Averaging metrics of every data point

Model	Flow			Speed		
	MAE	MAPE (%)	MSE	MAE	MAPE (%)	MSE
NO	1541	53.1	3.54e6	59.4	55.4	6857
1	1869	65.7	4.95e6	54.6	57.2	4917
2	1700	58.9	4.22e6	42.2	40.6	3380
3	1793	62.2	4.65e6	48.6	50.4	4173
4	1696	59.9	4.27e6	63.5	60.4	7106
5	1576	55.7	3.73e6	61.5	57.6	7049
6	1516	53.1	3.49e6	59.3	55.1	6903
7	1598	55.3	3.69e6	58.5	54.3	6865

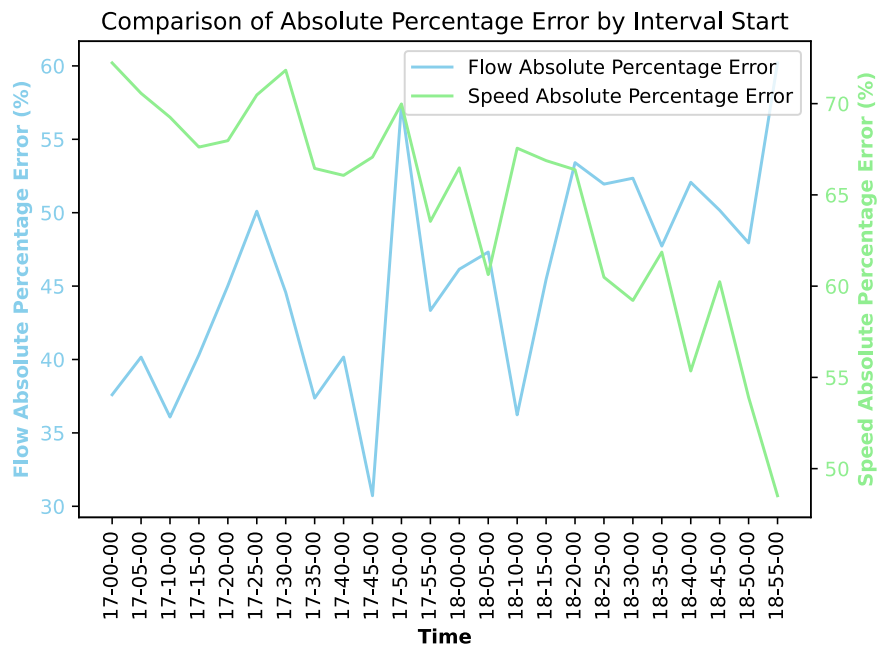


Figure 6.19: Absolute percentage error for validation for model 7 in time

All of the validation graphics for these models can be seen in Appendix A.3.

In Table 6.5 and Table 6.6, the overall metrics for all of the validated models are shown. The results are quite different and are due to how the metrics were calculated. For the first table, the metrics are computed as the average of every data point. For the second table, first the data was averaged so that only one data point remained for each point in time. Afterward, the metrics were computed and averaged in the end. Even though that explains why the verified disparities are possible, even disregarding the existence of the machine learning models, it is unknown what causes this contrast. Despite the worse results, Table 6.5 is analyzed since the dual averaging appears to introduce a bias that does not faithfully represent the performance of the simulation.

The MAE and MAPE are the most relevant and important information can be extracted from them. For example, in a more objective way, we confirm the good performance of models 2 and

Table 6.6: All metrics for the validated models. Averaging metrics by interval start

Model	Flow			Speed		
	MAE	MAPE (%)	MSE	MAE	MAPE (%)	MSE
NO	1342	46.4	1.86e6	48.2	63.8	2356
1	1470	50.9	2.23e6	11.4	15.1	142.2
2	1502	52.0	2.21e6	1.91	2.55	5.566
3	1403	48.6	2.03e6	6.50	8.54	61.49
4	1191	41.27	1.48e6	26.46	35.0	713.7
5	1163	40.2	1.42e6	36.9	48.9	1386
6	1185	41.0	1.46e6	47.2	62.5	2263
7	1317	45.6	1.79e6	48.8	64.6	2407

3 in regard to the simulation of the speed. Notice that the difference is much less significant than what the previously analyzed graphics indicated. This is most likely due to the previous graphs taking the average of the measurements for a given point in time, whereas the metrics in the table are acquired by averaging the metrics for each individual data point. Nevertheless, model 3 obtained a 5 percentual points lower MAPE and model 2 a 14.8 percentual points lower MAPE, demonstrating their impact in the calibration of the simulation. Contrary to what was previously observed, the other models do not improve the performance of the simulated flow, in fact, most of them worsen it. Model 6 shows how the MSE can be used to compare models for which the MAE or MAPE are similar. Since the flow MSE for model 6 is slightly lower than with no model, we can conclude that its simulated flows tend to be closer to the real values. Similarly, it presents a lower speed MAPE but higher speed MSE than the validation with no model, indicating the opposite.

Even though no specific durations were recorded, every simulation, including those in which training occurred, met time performance requirements, taking less time than the 2-hour period that was simulated.

Albeit simple and very focused, this analysis demonstrates that the introduction of a model was able to increase the similarity of the simulation to the real world, at least when it comes to the speeds. It is plausible that different models, with different parameters or even trained for longer, could achieve similar results for traffic flows and better results for the speeds.

6.8 Summary

The case study of the VCI inner ring in Porto uses the methodology previously established in Chapter 5 to create, calibrate, and run a simulation. An analysis of the network is performed, as well as an analysis of the data that was used to run the simulation.

For the calibration, several alternatives are tested. First, no model is used to set a baseline that has a significant error in the two measures chosen: flow and speeds. Afterward, several models were introduced to study how the errors on each of the measures would change. Positive results were found for the calibration of speeds with a relatively small and simple machine learning model.

Chapter 7

Conclusions

7.1 Objectives

In this work, a methodology for the development and calibration of a road network Digital Twin, particularly in an uninterrupted flow setting, is achieved. Throughout this work, the research questions previously formulated were studied, and we can now answer them in a more concrete way:

1. How to create a simulation?

Through the use of SUMO, a microsimulator, and network data obtained through OSM it is possible to obtain a representation of the network. Furthermore, the position of any known sensors is also required. Afterward, additional information must be computed as preparation for the simulation framework, such as the calculation of the systems of equations and the independent systems (see Section 5.3.2).

2. How to run an online simulation?

The running of the simulation is done through the simulation framework described in Section 5.3.3, which makes use of an API to communicate between SUMO and the Simulation-Runner, which controls the SUMO vehicles, vehicle types, and calibrators.

3. How to calibrate the simulation?

The calibration of the simulation happens on two fronts. First, there is the calibration of the flows, using a manual rerouting of each vehicle by assigning it a new route (see Section 5.3.3). Secondly, the driving behavior is calibrated by adjusting the parameters of the vehicle types (see Section 5.3.3). For the calibration, online data is required, containing measurements for the edges where the sensors are positioned. The data can either be a stream of historical data or real-time data.

4. How to validate the simulation?

The validation of the simulation is done through the comparison of the simulated measurements with the real measurements. Even though other metrics can be used, it was found that

the mean absolute error and the mean absolute percentage error were sufficient to visually and absolutely compare the results with each other.

7.2 Contributions and results

First, this work contributes to the field of Digital Twins by demonstrating and crediting its use to the scenario of road networks. Furthermore, it also expands upon the current work on the simulation of road networks. The developed methodology offers a novel paradigm that attempts to tackle the heavy computational load required for this type of simulation, through the automatic division of the network into several independent systems (see Section 5.3.3). Additionally, it proposes the use of data collected every day by the existing sensor infrastructure through a Machine Learning approach. Although, model-less macroscopic simulators have already taken advantage of the big amount of data available, the use of this data in an automated manner by a microsimulator is a new direction that is worthy of consideration.

Even though the results obtained in the case study (see Chapter 6) could be improved when it comes to the calibration of the flows, it was confirmed that the application of Machine Learning models can be used to decrease the error of the real data measurements.

7.3 Future work

The continuation of this work could see improvements made to the current implementation. First, it can be said that the rerouting algorithm is too complex. An alternative solution, instead of manually assigning each vehicle a route id determined by the Rerouter (see Section 5.3.3), would assign a route distribution¹ as done in the implementation of [47]. This approach was not tested and it is unknown whether or not it would allow for the same customizability of rerouting while allowing a simpler implementation. In any case, different criteria for the triggering of the rerouting of vehicles using the current route id assignment could also be explored (see Section 5.3.3). Additionally, running multiple independent systems within the same simulation was not implemented, and making the necessary changes to make it possible would increase the flexibility of the implementation to test more diverse scenarios. Most importantly, more intensive training and different architectures are key to obtaining machine learning models with better performance both for the flows and speeds measured.

There are multiple areas that future efforts could explore. First, other measurements beyond the flow and speeds of vehicles can be used. This would allow for a more complete knowledge and description of the traffic. As such, it would bring the methodology closer to a Digital Twin since the simulation would be closer to reflecting the state of the real system. Secondly, the additional information (see Section 5.1.2) that was mentioned but not thoroughly described or integrated into the methodology (only in Section 6.5 a specific case was explained), could also be studied. This study could evaluate the benefits of including said data as inputs for the Machine Learning model

¹https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html#route_distributions

or other parts of the simulation. Furthermore, the current approach assumes fixed and infallible sensors. A methodology that is flexible in dealing with the introduction or removal of sensors would be a positive aspect. Finally, much work is needed to truly assess to what extent a Machine Learning model applied to such a road network simulation can increase its performance in larger-scale scenarios and span larger time intervals.

References

- [1] Karl-Heinz Kastner, Robert Keber, Petru Pau, and Martin Samal. Real-time traffic conditions with SUMO for ITS austria west. In Michael Behrisch, Daniel Krajzewicz, and Melanie Weber, editors, *Simulation of Urban Mobility: First International Conference, SUMO 2013, Berlin, Germany, May 15-17, 2013. Revised Selected Papers*, volume 8594 of *Lecture notes in computer science*, pages 146–159, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [2] Stefan Krauß. Microscopic modeling of traffic flow: investigation of collision free vehicle dynamics. apr 1998.
- [3] Joao Soares, Cristina Lobo, Zita Vale, and P. B. de Moura Oliveira. Realistic traffic scenarios using a census methodology: Vila real case study. In *2014 IEEE PES General Meeting |Conference & Exposition*, pages 1–5. IEEE, jul 2014.
- [4] Haiyang Yu, Zhihai Wu, Shuqin Wang, Yunpeng Wang, and Xiaolei Ma. Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks. *Sensors (Basel, Switzerland)*, 17(7), jun 2017.
- [5] Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. *Sensors (Basel, Switzerland)*, 17(4), apr 2017.
- [6] Lingxiao Zhou, Shuaichao Zhang, Jingru Yu, and Xiqun Chen. Spatial-temporal deep tensor neural networks for large-scale urban network speed prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3718–3729, sep 2020.
- [7] Eclipse SUMO - simulation of urban MObility. Consulted on 02/02/2024.
- [8] Traffic simulation software IPTV vissim IPTV group. Consulted on 02/02/2024.
- [9] Y. Suda, Y. Takahashi, M. Kuwahara, S. Tanaka, K. Ikeuchi, M. Kagesawa, T. Shraishi, M. Onuki, K. Honda, and M. Kano. Development of universal driving simulator with interactive traffic environment. In *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pages 743–746. IEEE, jun 2005.
- [10] Kok Mun Ng and Mamun Ibne Reaz. Development and validation of the LWR-IM traffic simulator. In *2017 7th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 40–44. IEEE, nov 2017.
- [11] Sho Tokuda, Ryo Kanamori, and Takayuki Ito. Development of traffic simulator based on stochastic cell transmission model for urban network. In Hoa Khanh Dam, Jeremy Pitt, Yang Xu, Guido Governatori, and Takayuki Ito, editors, *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, volume 8861 of *Lecture notes in computer science*, pages 150–165. Springer International Publishing, Cham, 2014.

- [12] T. Ishikawa. Development of a road traffic simulator. *IEEE Transactions on Vehicular Technology*, 47(3):1066–1071, 1998.
- [13] RE Quandt. The construction of travel demand models with incomplete data. pages 115–121. ageconsearch.umn.edu, 1966.
- [14] Tom Domencich. *Urban Travel Demand: A Behavioral Analysis*. North-Holland Publishing Co., 1975.
- [15] F. Russo and G. Chilà. Dynamic approaches to demand model in evacuation conditions. In A. Pratelli and C. A. Brebbia, editors, *Urban Transport XVI*, volume 1 of *WIT transactions on the built environment*, pages 303–312, Southampton, UK, apr 2010. WIT Press.
- [16] Yongmei Zhao, Hongmei Zhang, Li An, and Quan Liu. Improving the approaches of traffic demand forecasting in the big data era. *Cities*, 82:19–26, jul 2018.
- [17] Vishwanath Bulusu, Emin Burak Onat, Raja Sengupta, Pavan Yedavalli, and Jane Macfarlane. A traffic demand analysis method for urban air mobility. *IEEE Transactions on Intelligent Transportation Systems*, 22(9):6039–6047, sep 2021.
- [18] Michael Grieves. Digital twin: Manufacturing excellence through virtual factory replication, 2014.
- [19] Mike Shafto, Michael Conroy, Rich Doyle, Ed Glaessgen, LeMoigne Jacqueline, and Lui Wang. (modeling, simulation, information technology and processing roadmap, 2010.
- [20] IBM. What is a digital twin?, 2022. Consulted on 01/02/2024.
- [21] Stefan Braun, Manuela Dalibor, Nico Jansen, Matthias Jarke, István Koren, Christoph Quix, Bernhard Rumpe, Manuel Wimmer, and Andreas Wortmann. Engineering digital twins and digital shadows as key enablers for industry 4.0. In Birgit Vogel-Heuser and Manuel Wimmer, editors, *Digital Transformation: Core Technologies and Emerging Topics from a Computer Science Perspective*, pages 3–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2023.
- [22] Eric VanDerHorn and Sankaran Mahadevan. Digital twin: Generalization, characterization and implementation. *Decision support systems*, 145:113524, jun 2021.
- [23] Andrzej Kochan, Piotr Folęga, Gintautas Bureika, and Remigijus Skirkus. MULTIGRAPH IS: part 2. VERIFICATION OF THE DIGITAL TWIN OF THE EUROPEAN TRAIN CONTROL SYSTEM APPLICATION. *Theoretical Physics*, 18(3):5–14, sep 2023.
- [24] Anders Sjöberg, Magnus Önnheim, Otto Frost, Constantin Cronrath, Emil Gustavsson, Bengt Lennartson, and Mats Jirstrand. Online geometry assurance in individualized production by feedback control and model calibration of digital twins. *Journal of Manufacturing Systems*, 66:71–81, feb 2023.
- [25] N. Vijayalakshmi, Y. Mohana Roopa, B. Ashreetha, J. V. N. Ramesh, E. Gurumohan Rao, and P. N. Sundararajan. Development of digital twin with external data resources in manufacturing with complex algorithms. *SN Computer Science*, 4(5):611, aug 2023.
- [26] Lenos Hadjidemetriou, Nearchos Stylianidis, Demetris Englezos, P. Papadopoulos, Demetrios Eliades, Stelios Timotheou, Marios M. Polycarpou, and Christos Panayiotou. A digital twin architecture for real-time and offline high granularity analysis in smart buildings. *Sustainable Cities and Society*, 98:104795, nov 2023.

- [27] Laura Donato, Chiara Galletti, and Alessandro Parente. Self-updating digital twin of a hydrogen-powered furnace using data assimilation. *Applied thermal engineering*, 236:121431, jan 2024.
- [28] Olga Romanova and Dmitry Sirotin. Digital twins in russian metallurgy: prerequisites and limitations of use. In Vikas Kumar, Jiewu Leng, Victoria Akberdina, and Evgeny Kuzmin, editors, *Digital transformation in industry: digital twins and new business models*, volume 54 of *Lecture notes in information systems and organisation*, pages 57–69. Springer International Publishing, Cham, 2022.
- [29] Jie Lu, Xiangning Tian, Chenxin Feng, Chaobo Zhang, Yang Zhao, Yiwen Zhang, and Zihao Wang. Clustering compression-based computation-efficient calibration method for digital twin modeling of HVAC system. *Building Simulation*, 16(6):997–1012, jun 2023.
- [30] Christian Vering, Sebastian Borges, Daniel Coakley, Hannah Kruetzfeldt, Philipp Mehrfeld, and Dirk Müller. Digital twin design with on-line calibration for HVAC systems in buildings. In *Proceedings of Building Simulation 2021: 17th Conference of IBPSA*, volume 17 of *Building simulation conference proceedings*. KU Leuven, sep 2021.
- [31] Houde Song, Meiqi Song, and Xiaojing Liu. Online autonomous calibration of digital twins using machine learning with application to nuclear power plants. *Applied Energy*, 326:119995, nov 2022.
- [32] Eduardo R. Conde López, Miguel Á. Toledo, and Eduardo Saleté Casino. The centroid method for the calibration of a sectorized digital twin of an arch dam. *Water*, 14(19):3051, sep 2022.
- [33] Bo Wang, Chi Zhang, Min Zhang, Changhe Liu, Zilong Xie, and Hong Zhang. Digital twin analysis for driving risks based on virtual physical simulation technology. *IEEE Journal of Radio Frequency Identification*, 6:938–942, 2022.
- [34] Changhe Liu, Chi Zhang, Bo Wang, Zhongze Tang, and Zilong Xie. Digital twin of highway entrances and exits: A traffic risk identification method. *IEEE Journal of Radio Frequency Identification*, 6:934–937, 2022.
- [35] Javier Argota Sánchez-Vaquero. Getting real: The challenge of building and validating a large-scale digital twin of barcelona’s traffic with empirical data. *ISPRS international journal of geo-information*, 11(1):24, dec 2021.
- [36] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018.
- [37] Samad M. E. Sepasgozar. Differentiating digital twin from digital shadow: Elucidating a paradigm shift to expedite a smart, sustainable built environment. *Buildings*, 11(4):151, apr 2021.
- [38] Thomas Bergs, Sascha Gierlings, Thomas Auerbach, Andreas Klink, Daniel Schraknepper, and Thorsten Augspurger. The concept of digital twin and digital shadow in manufacturing. *Procedia CIRP*, 101:81–84, 2021.
- [39] Asma Ladj, Zhiqiang Wang, Oussama Meski, Farouk Belkadi, Mathieu Ritou, and Catherine Da Cunha. A knowledge-based digital shadow for machining industry in a digital twin perspective. *Journal of Manufacturing Systems*, 58:168–179, jan 2021.

- [40] Krešimir Kušić, Rene Schumann, and Edouard Ivanjko. Building a motorway digital twin in SUMO: Real-time simulation of continuous data stream from traffic counters. In *2022 International Symposium ELMAR*, pages 71–76. IEEE, sep 2022.
- [41] Daniel Rodriguez-Rey, Marc Guevara, Ma Paz Linares, Josep Casanovas, Juan Salmerón, Albert Soret, Oriol Jorba, Carles Tena, and Carlos Pérez García-Pando. A coupled macroscopic traffic and pollutant emission modelling system for barcelona. *Transportation Research Part D: Transport and Environment*, 92:102725, mar 2021.
- [42] Joerg Schweizer, Cristian Poliziani, Federico Rupi, Davide Morgano, and Mattia Magi. Building a large-scale micro-simulation transport scenario using big data. *ISPRS international journal of geo-information*, 10(3):165, mar 2021.
- [43] Yunjie Zhao and Adel W. Sadek. Large-scale agent-based traffic micro-simulation: Experiences with model refinement, calibration, validation and application. *Procedia Computer Science*, 10:815–820, 2012.
- [44] Zhanhao Ji, Guojiang Shen, Juntao Wang, Mario Collotta, Zhi Liu, and Xiangjie Kong. Multi-vehicle trajectory tracking towards digital twin intersections for internet of vehicles. *Electronics*, 12(2):275, jan 2023.
- [45] Irena Ištoka Otković, Tomaž Tollazzi, Matjaž Šraml, and Damir Varevac. Calibration of the microsimulation traffic model using different neural network applications. *Future Transportation*, 3(1):150–168, feb 2023.
- [46] Dwayne Henclewood, Wonho Suh, Michael Rodgers, Michael Hunter, and Richard Fujimoto. A case for real-time calibration of data-driven microscopic traffic simulation tools. In *Proceedings Title: Proceedings of the 2012 Winter Simulation Conference (WSC)*, pages 1–12. IEEE, dec 2012.
- [47] Paulo Ribeiro. A modelling methodology towards automated generation of road network digital twins.
- [48] Oussama El Marai, Tarik Taleb, and JaeSeung Song. Roads infrastructure digital twin: A step toward smarter cities realization. *IEEE network*, 35(2):136–143, mar 2021.
- [49] Mozghan Pourmoradnasseri, Kaveh Khoshkhah, and Amnir Hadachi. Leveraging IoT data stream for near-real-time calibration of city-scale microscopic traffic simulation. *IET Smart Cities*, 5(4):269–290, dec 2023.
- [50] Haizheng Zhang, Ravi Seshadri, A. Arun Prakash, Constantinos Antoniou, Francisco C. Pereira, and Moshe Ben-Akiva. Improving the accuracy and efficiency of online calibration for simulation-based dynamic traffic assignment. *Transportation Research Part C: Emerging Technologies*, 128:103195, jul 2021.
- [51] Tingting Fan, Ivan Wang-Hei Ho, and Edward Chung. Digital twin-assisted lane-changing and variable speed limit control for weaving segments. In *2022 IEEE 2nd International Conference on Digital Twins and Parallel Intelligence (DTPI)*, pages 1–5. IEEE, oct 2022.
- [52] Qian Liu, Xiaoqing Qi, Shanshan Liu, Xiuju Cheng, Xin Ke, and Feng Wang. Application of lightweight digital twin system in intelligent transportation. *IEEE Journal of Radio Frequency Identification*, 6:729–732, 2022.

- [53] Feng Jiang, Ling Ma, Tim Broyd, Weiya Chen, and Hanbin Luo. Building digital twins of existing highways using map data based on engineering expertise. *Automation in Construction*, 134:104081, feb 2022.
- [54] Kevin Curran, John Crumlish, and Gavin Fisher. Openstreetmap. *International Journal of Interactive Communication Systems and Technologies*, 2(1):69–78, jan 2012.
- [55] Google maps. Consulted on 05/02/2024.
- [56] HERE technologies |the world’s #1 location platform. Consulted on 05/02/2024.
- [57] Digimap. Consulted on 05/02/2024.
- [58] Graphenintegrations-plattform GIP: Der multimodale, digitale verkehrsgraph für ganz Österreich. Consulted on 05/02/2024.
- [59] Sathish A. P. Kumar, R. Madhumathi, Pethuru Raj Chelliah, Lei Tao, and Shangguang Wang. A novel digital twin-centric approach for driver intention prediction and traffic congestion avoidance. *Journal of Reliable Intelligent Environments*, 4(4):199–209, dec 2018.

Appendix A

Appendix

```
1 <node id="25507582" visible="true" version="13" changeset="110521287" timestamp="
  2021-08-31T15:44:24Z" user="Ivan_" uid="1773299" lat="41.1513862" lon="
  -8.5797087">
2 <tag k="exit_to" v="A43"/>
3 <tag k="highway" v="motorway_junction"/>
4 <tag k="name" v="Freixo / Entre-os-Rios"/>
5 <tag k="ref" v="5"/>
6 </node>
7 <way id="497150665" visible="true" version="5" changeset="142980952" timestamp="
  2023-10-22T18:39:13Z" user="waldyrrious" uid="5177">
8 <nd ref="25507580"/>
9 <nd ref="4685582574"/>
10 <nd ref="4685582575"/>
11 <nd ref="25507582"/>
12 <tag k="highway" v="motorway"/>
13 <tag k="int_ref" v="E 01"/>
14 <tag k="lanes" v="5"/>
15 <tag k="maxspeed" v="80"/>
16 <tag k="name" v="Via de Cintura Interna"/>
17 <tag k="oneway" v="yes"/>
18 <tag k="ref" v="IP 1"/>
19 <tag k="surface" v="asphalt"/>
20 <tag k="turn:lanes" v="through|through|through|through|slight_right"/>
21 </way>
```

Listing A.1: Example sections of an .osm file

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
  noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/netconvertConfiguration.xsd">
4
```

```

5   <input>
6       <osm-files value="map_1.osm,map_2.osm,map_3.osm,map_4.osm,map_5.osm,map_6.
osm,map_7.osm,map_8.osm"/>
7   </input>
8
9   <output>
10      <output.street-names value="true"/>
11      <output.original-names value="true"/>
12  </output>
13
14  <processing>
15      <geometry.remove value="true"/>
16  </processing>
17
18  <edge_removal>
19      <remove-edges.isolated value="true"/>
20  </edge_removal>
21
22  <junctions>
23      <junctions.join value="false"/>
24  </junctions>
25
26  <random_number>
27      <random value="false"/>
28      <seed value="42"/>
29  </random_number>
30
31 </configuration>

```

Listing A.2: Configuration file for the conversion of the .osm files to a SUMO network file

```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
noNamespaceSchemaLocation=" http://sumo.sf.net/xsd/sumoConfiguration.xsd">
3   <input>
4       <net-file value="filter_by_edge.net.xml"/>
5       <route-files value="filter_by_edge_continuous_joint_routes_multiple.rou.xml,
filter_by_edge.rou.xml"/>
6       <additional-files value="filter_by_edge_all_counters_training.add.xml,
filter_by_edge_entry_calibrators_edges_training.add.xml,
filter_by_edge_sensors_calibrators_training.add.xml,
filter_by_edge_sensors_counters_training.add.xml"/>
7   </input>
8   <time>
9       <begin value="0"/>
10      <end value="3600.0"/>
11      <step-length value="1"/>
12  </time>

```

```

13 <report>
14   <verbose value="true"/>
15   <duration-log.statistics value="true"/>
16   <no-step-log value="false"/>
17 </report>
18 </configuration>

```

Listing A.3: Sumo configuration file for running a simulation

```

1 # Original code
2
3 class Net:
4     def addLane(self, edge, speed, length, width, allow=None, disallow=None):
5         return lane.Lane(edge, speed, length, width, allow, disallow)
6
7 class NetReader(handler.ContentHandler):
8     def startElement(self, name, attrs):
9         if name == 'location':
10             pass # [...]
11         if name == 'edge':
12             pass # [...]
13         if name == 'lane' and self._currentEdge is not None:
14             self._currentLane = self._net.addLane(
15                 self._currentEdge,
16                 float(attrs['speed']),
17                 float(attrs['length']),
18                 float(attrs.get('width', 3.2)),
19                 attrs.get('allow'),
20                 attrs.get('disallow')
21             self._currentLane.setShape(convertShape(attrs.get('shape', '')))
22             pass # [...]
23
24 # Edited code
25
26
27 class Net:
28     def addLane(self, edge, speed, length, width, allow=None, disallow=None,
29                 changeLeft=None, changeRight=None):
30         return lane.Lane(edge, speed, length, width, allow, disallow, changeLeft,
31                           changeRight)
32
33 class NetReader(handler.ContentHandler):
34     def startElement(self, name, attrs):
35         if name == 'location':
36             pass # [...]
37         if name == 'edge':
38             pass # [...]

```

```

38     if name == 'lane' and self._currentEdge is not None:
39         self._currentLane = self._net.addLane(
40             self._currentEdge,
41             float(attrs['speed']),
42             float(attrs['length']),
43             float(attrs.get('width', 3.2)),
44             attrs.get('allow'),
45             attrs.get('disallow'),
46             attrs.get('changeLeft'), # ADDED
47             attrs.get('changeRight')) # ADDED
48         self._currentLane.setShape(convertShape(attrs.get('shape', '')))

```

Listing A.4: Changes to sumolib/net/__init__.py

```

1
2 # Original code
3
4 Class Lane:
5
6     def __init__(self, edge, speed, length, width, allow, disallow):
7         self._edge = edge
8         self._speed = speed
9         self._length = length
10        self._width = width
11        self._shape = None
12        self._shape3D = None
13        self._shapeWithJunctions = None
14        self._shapeWithJunctions3D = None
15        self._outgoing = []
16        self._params = {}
17        self._allowed = get_allowed(allow, disallow)
18        self._neigh = None
19        self._selected = False
20        edge.addLane(self)
21
22 # Edited code
23
24 Class Lane:
25
26     def __init__(self, edge, speed, length, width, allow, disallow, changeLeft,
27 changeRight):
28         self._edge = edge
29         self._speed = speed
30         self._length = length
31         self._width = width
32         self._shape = None
33         self._shape3D = None
34         self._shapeWithJunctions = None

```

```

34     self._shapeWithJunctions3D = None
35     self._outgoing = []
36     self._params = {}
37     self._allowed = get_allowed(allow, disallow)
38     self._changeLeft = get_vehicle_classes(changeLeft)
39     self._changeRight = get_vehicle_classes(changeRight)
40     self._neigh = None
41     self._selected = False
42     edge.addLane(self)
43
44     def get_vehicle_classes(classes_string: str):
45         if classes_string is None:
46             return set()
47         classes_set = set(classes_string.split())
48         return classes_set
49
50     def getChangeLeft(self) -> set[str] | None:
51         """None if no restriction. Otherwise set of vehicle classes that can change
52         to the left lane"""
53         return self._changeLeft
54
55     def setChangeLeft(self, changeLeft):
56         self._changeLeft = get_vehicle_classes(changeLeft)
57
58     def getChangeRight(self) -> set[str] | None:
59         """None if no restriction. Otherwise set of vehicle classes that can change
60         to the right lane"""
61         return self._changeRight
62
63     def setChangeRight(self, changeRight):
64         self._changeRight = get_vehicle_classes(changeRight)

```

Listing A.5: Changes to sumolib/net/lane.py

```

1 rerouter_information: dict[Edge, dict[RouteObject, list[RouteObject]]]
2 route_objects[str, RouteObject]
3
4 class Rerouter:
5     def __init__(self, probability, rerouter_information):
6         self.probability = probability
7         self.rerouter_information: dict[RouteObject, list[RouteObject]] =
            rerouter_information
8
9     def choose_random_route(prev_route: RouteObject):
10         next_possible_route_objects = self.rerouter_information[prev_route]
11         r = random.random()
12         if r < self.probability:
13             return next_possible_route_objects[0]

```

```

14         return next_possible_route_objects[1]
15 class RerouterAdjustable(Rerouter):
16
17     def __init__(self, probability, rerouter_information):
18         super().__init__(probability, route1, route2)
19         self.number_vehicles_to_the_left = 0
20         self.total_vehicles = 0
21
22     def choose_route(self, prev_route: RouteObject) -> str:
23         if self.total_vehicles == 0:
24             return self.choose_random_route(prev_route)
25
26         p = self.number_vehicles_left / self.total_vehicles
27         dif = p - self.probability
28         if dif < 0:
29             return self.choose_mandatory_route(prev_route, 0)
30         elif dif > 0:
31             return self.choose_mandatory_route(prev_route, 1)
32         else:
33             # Reset the counts for the compensation
34             self.number_vehicles_to_the_left = 0
35             self.total_vehicles = 0
36
37             return self.choose_random_route()
38
39     def choose_random_route():
40         r = random()
41         if r < self.probability:
42             return self.route1
43         return self.route2
44     def choose_mandatory_route(self, prev_route: RouteObject, chosen_route_index):
45         if chosen_route_ == 0:
46             self.number_vehicles_to_the_left +=1
47             self.total_vehicles += 1
48
49         next_possible_route_objects = self.rerouter_information[prev_route]
50         chosen_route = next_possible_route_objects[chosen_route_index]
51
52         return chosen_route
53
54     def choose_next_route(self, cur_route: RouteObject, lane_id, road_id):
55         lane = net.getLane(lane_id)
56         reachable_edges_from_lane = reachable[lane]
57         if len(reachable_edges_from_lane) == 2:
58             # All next routes are reachable
59             return self.choose_route()
60         only_reachable_edge = reachable_edges_from_lane[0]
61         if only_reachable_edge == self.route1.route[0]:
62             # Only the left route is reachable

```

```
63         return self.chose_mandatory_route(cur_route, 0)
64     else:
65         # Only the right route is reachable
66         return self.chose_mandatory_route(cur_route, 1)
```

Listing A.6: Rerouter and AdjustableRerouter considering route compatibility

A.1 VCI independent systems

In the following networks, the blue edges represent the sensor edges.

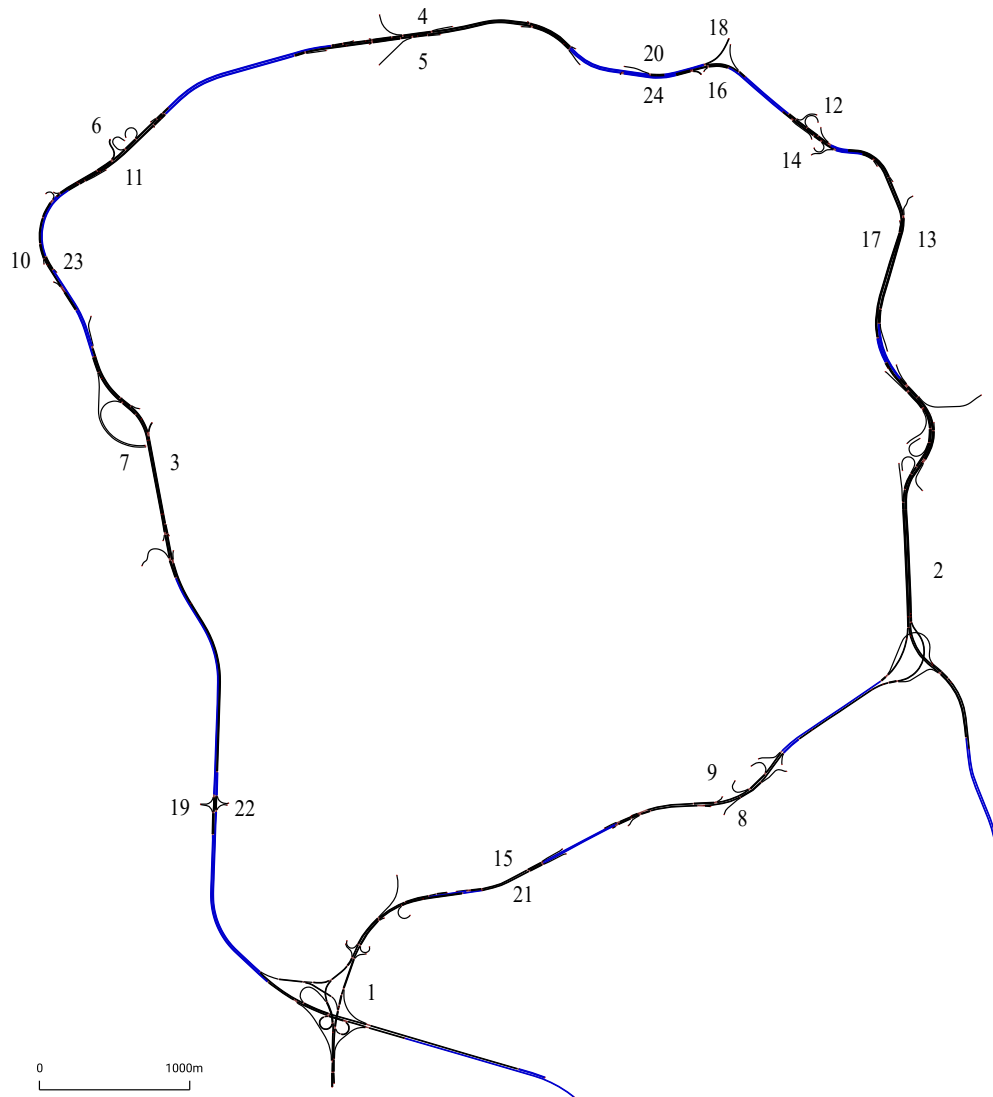


Figure A.1: VCI's independent systems numbered from 1 to 24

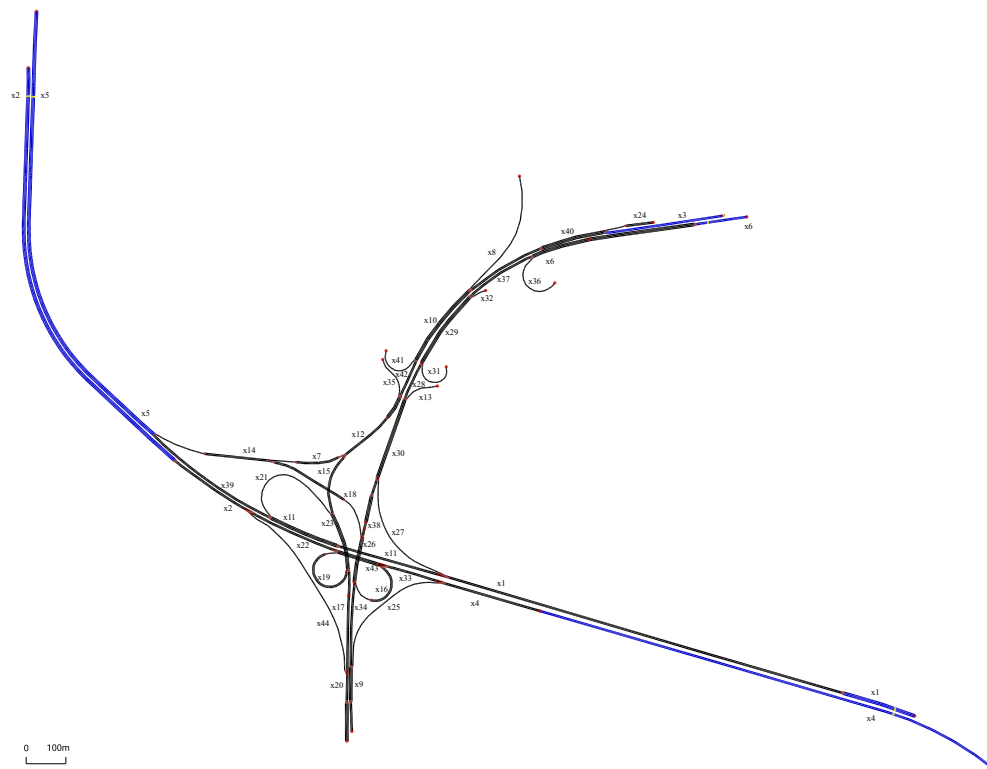


Figure A.2: Independent system 1

$$\begin{aligned}
x_{43} &= x_{22} + x_{19} \\
x_{43} &= x_{33} + x_{16} \\
x_{40} &= x_{24} + x_3 \\
x_{30} &= x_{38} + x_{27} \\
x_{30} &= x_{28} + x_{13} \\
x_{29} &= x_{31} + x_{28} \\
x_{29} &= x_{37} + x_{32} \\
x_{26} &= x_{34} + x_{16} \\
x_{26} &= x_{38} + x_{18} \\
x_{23} &= x_{21} + x_{15} \\
x_{23} &= x_{19} + x_{17} \\
x_{20} &= x_{44} + x_{17} \\
x_{14} &= x_{18} + x_7 \\
x_{12} &= x_{42} + x_{35} \\
x_{12} &= x_{15} + x_7 \\
x_{11} &= x_{39} + x_{21} \\
x_{10} &= x_{40} + x_8 \\
x_{10} &= x_{42} + x_{41} \\
x_9 &= x_{34} + x_{25} \\
x_6 &= x_{37} + x_{36} \\
x_5 &= x_{39} + x_{14} \\
x_4 &= x_{33} + x_{25} \\
x_2 &= x_{44} + x_{22} \\
x_1 &= x_{27} + x_{11}
\end{aligned}$$

(A.1)

Equation A.1: Plain equations for independent system 1

$$\begin{aligned}
x_1 &= x_1 \\
x_2 &= x_2 \\
x_3 &= x_3 \\
x_4 &= x_4 \\
x_5 &= x_5 \\
x_6 &= x_6 \\
x_7 &= x_7 \\
x_8 &= x_8 \\
x_9 &= x_9 \\
x_{10} &= x_{10} \\
x_{11} &= x_{11} \\
x_{12} &= x_{12} \\
x_{13} &= x_{13} \\
x_{14} &= x_{14} \\
x_{15} &= x_{12} - x_7 \\
x_{16} &= x_{16} \\
x_{17} &= x_{17} \\
x_{18} &= x_{14} - x_7 \\
x_{19} &= x_{11} + x_{12} + x_{14} - x_5 - x_7 - x_{17} \\
x_{20} &= x_{20} \\
x_{21} &= x_{11} + x_{14} - x_5 \\
x_{22} &= x_2 + x_{17} - x_{20} \\
x_{23} &= x_{11} + x_{12} + x_{14} - x_5 - x_7 \\
x_{24} &= x_{10} - x_3 - x_8 \\
x_{25} &= x_4 + x_5 + x_7 + x_{16} + x_{20} - x_2 - x_{11} - x_{12} - x_{14} \\
x_{26} &= x_2 + x_9 + x_{11} + x_{12} + x_{14} - x_4 - x_5 - x_7 - x_{20}
\end{aligned}
\tag{A.2}$$

Equation A.2: Simplified equations for independent system 1 (part 1)

$$\begin{aligned}
x_{27} &= x_1 - x_{11} \\
x_{28} &= x_1 + x_2 + x_9 + x_{12} - x_4 - x_5 - x_{13} - x_{20} \\
x_{29} &= x_{29} \\
x_{30} &= x_1 + x_2 + x_9 + x_{12} - x_4 - x_5 - x_{20} \\
x_{31} &= x_4 + x_5 + x_{13} + x_{20} + x_{29} - x_1 - x_2 - x_9 - x_{12} \\
x_{32} &= x_{32} \\
x_{33} &= x_2 + x_{11} + x_{12} + x_{14} - x_5 - x_7 - x_{16} - x_{20} \\
x_{34} &= x_2 + x_9 + x_{11} + x_{12} + x_{14} - x_4 - x_5 - x_7 - x_{16} - x_{20} \\
x_{35} &= x_{35} \\
x_{36} &= x_6 + x_{32} - x_{29} \\
x_{37} &= x_{29} - x_{32} \\
x_{38} &= x_2 + x_9 + x_{11} + x_{12} - x_4 - x_5 - x_{20} \\
x_{39} &= x_5 - x_{14} \\
x_{40} &= x_{10} - x_8 \\
x_{41} &= x_{10} + x_{35} - x_{12} \\
x_{42} &= x_{12} - x_{35} \\
x_{43} &= x_2 + x_{11} + x_{12} + x_{14} - x_5 - x_7 - x_{20} \\
x_{44} &= x_{20} - x_{17}
\end{aligned}$$

(A.3)

Equation A.3: Simplified equations for independent system 1 (part 2)

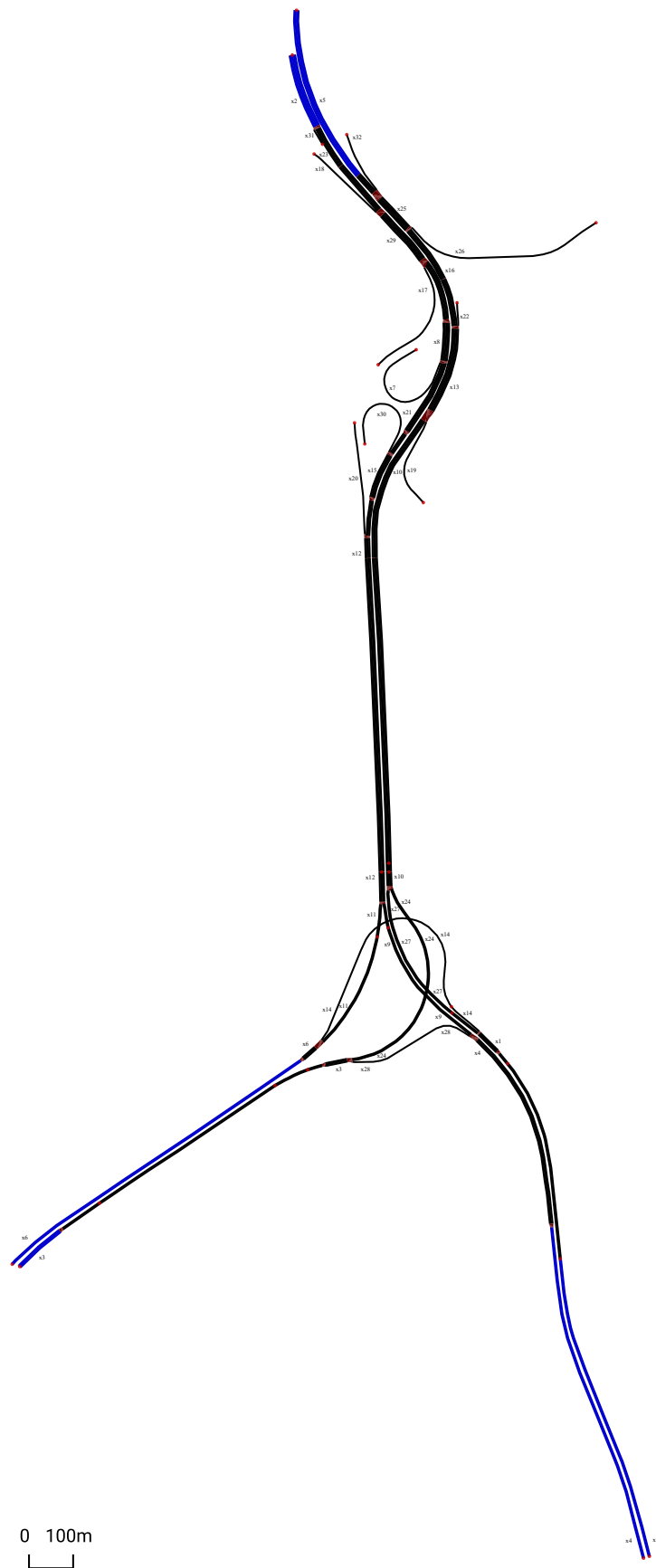


Figure A.3: Independent system 2

$$\begin{aligned}
x_{29} &= x_{23} + x_{18} \\
x_{29} &= x_{17} + x_8 \\
x_{25} &= x_{26} + x_{16} \\
x_{25} &= x_{32} + x_5 \\
x_{15} &= x_{30} + x_{21} \\
x_{13} &= x_{19} + x_{10} \\
x_{13} &= x_{22} + x_{16} \\
x_{12} &= x_{20} + x_{15} \\
x_{12} &= x_{11} + x_9 \\
x_{10} &= x_{27} + x_{24} \\
x_8 &= x_{21} + x_7 \\
x_6 &= x_{14} + x_{11} \\
x_4 &= x_{28} + x_9 \\
x_3 &= x_{28} + x_{24} \\
x_2 &= x_{31} + x_{23} \\
x_1 &= x_{27} + x_{14}
\end{aligned}$$

(A.4)

Equation A.4: Plain equations for independent system 2

$$\begin{aligned}
x_1 &= x_1 \\
x_2 &= x_2 \\
x_3 &= x_3 \\
x_4 &= x_4 \\
x_5 &= x_5 \\
x_6 &= x_6 \\
x_7 &= x_7 \\
x_8 &= x_8 \\
x_9 &= x_9 \\
x_{10} &= x_{10} \\
x_{11} &= x_4 + x_6 + x_{10} - x_1 - x_3 - x_9 \\
x_{12} &= x_4 + x_6 + x_{10} - x_1 - x_3 \\
x_{13} &= x_{13} \\
x_{14} &= x_1 + x_3 + x_9 - x_4 - x_{10} \\
x_{15} &= x_{15} \\
x_{16} &= x_{16} \\
x_{17} &= x_{17} \\
x_{18} &= x_{18} \\
x_{19} &= x_{13} - x_{10} \\
x_{20} &= x_4 + x_6 + x_{10} - x_1 - x_3 - x_{15} \\
x_{21} &= x_8 - x_7 \\
x_{22} &= x_{13} - x_{16} \\
x_{23} &= x_8 + x_{17} - x_{18} \\
x_{24} &= x_3 + x_9 - x_4 \\
x_{25} &= x_{25} \\
x_{26} &= x_{25} - x_{16} \\
x_{27} &= x_4 + x_{10} - x_3 - x_9 \\
x_{28} &= x_4 - x_9 \\
x_{29} &= x_8 + x_{17} \\
x_{30} &= x_7 + x_{15} - x_8 \\
x_{31} &= x_2 + x_{18} - x_8 - x_{17} \\
x_{32} &= x_{25} - x_5
\end{aligned}$$

(A.5)

Equation A.5: Simplified equations for independent system 2

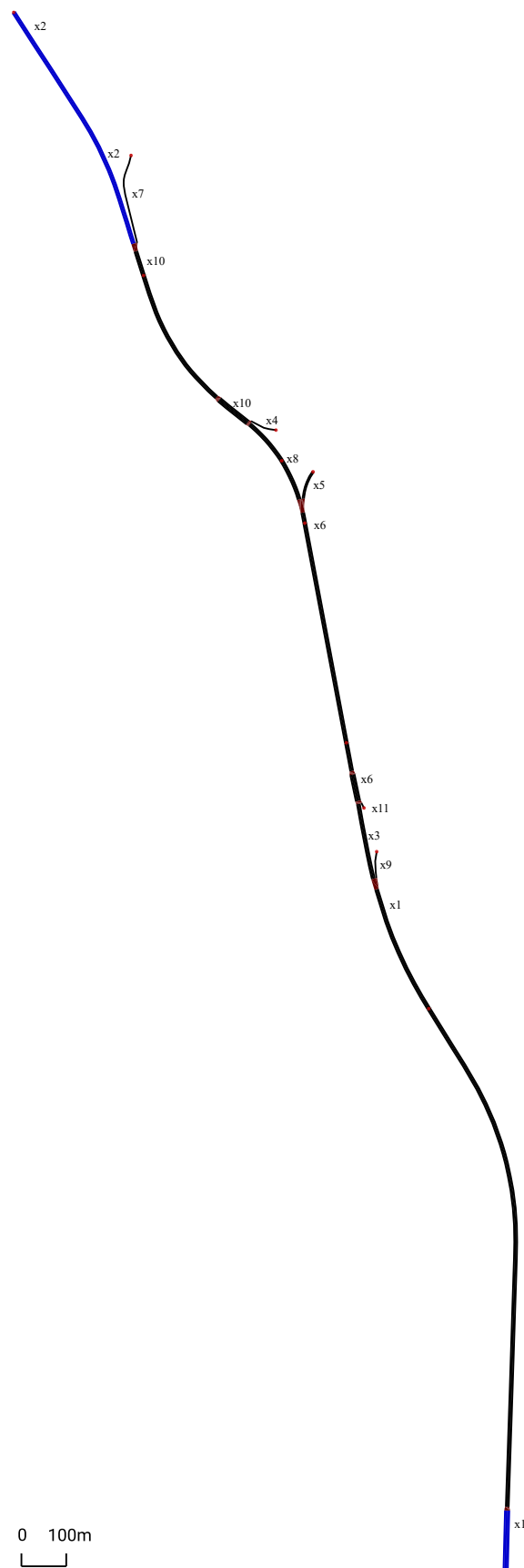


Figure A.4: Independent system 3

$$x_{10} = x_8 + x_4$$

$$x_{10} = x_7 + x_2$$

$$x_6 = x_{11} + x_3$$

$$x_6 = x_8 + x_5$$

$$x_1 = x_9 + x_3$$

(A.6)

Equation A.6: Plain equations for independent system 3

$$x_1 = x_1$$

$$x_2 = x_2$$

$$x_3 = x_3$$

$$x_4 = x_4$$

$$x_5 = x_5$$

$$x_6 = x_6$$

$$x_7 = x_4 + x_6 - x_2 - x_5$$

$$x_8 = x_6 - x_5$$

$$x_9 = x_1 - x_3$$

$$x_{10} = x_4 + x_6 - x_5$$

$$x_{11} = x_6 - x_3$$

(A.7)

Equation A.7: Simplified equations for independent system 3

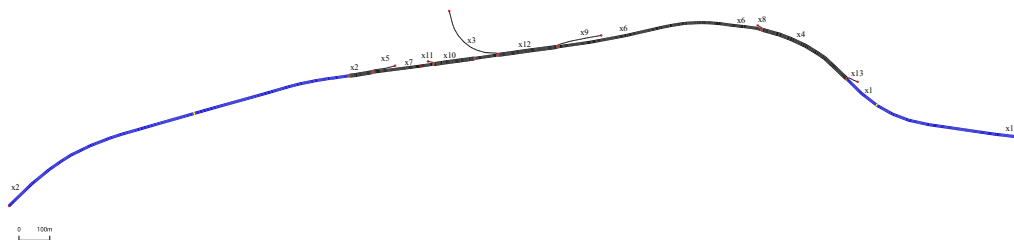


Figure A.5: Independent system 4

$$x_{12} = x_9 + x_6$$

$$x_{12} = x_{10} + x_3$$

$$x_{10} = x_{11} + x_7$$

$$x_4 = x_{13} + x_1$$

$$x_4 = x_8 + x_6$$

$$x_2 = x_7 + x_5$$

(A.8)

Equation A.8: Plain equations for independent system 4

$$x_1 = x_1$$

$$x_2 = x_2$$

$$x_3 = x_3$$

$$x_4 = x_4$$

$$x_5 = x_5$$

$$x_6 = x_6$$

$$x_7 = x_2 - x_5$$

$$x_8 = x_4 - x_6$$

$$x_9 = x_9$$

$$x_{10} = x_6 + x_9 - x_3$$

$$x_{11} = x_5 + x_6 + x_9 - x_2 - x_3$$

$$x_{12} = x_6 + x_9$$

$$x_{13} = x_4 - x_1$$

(A.9)

Equation A.9: Simplified equations for independent system 4



Figure A.6: Independent system 5

$$x_{13} = x_5 + x_4$$

$$x_{13} = x_9 + x_7$$

$$x_6 = x_9 + x_8$$

$$x_6 = x_3 + x_2$$

$$x_4 = x_{11} + x_{10}$$

$$x_1 = x_{12} + x_{11}$$

(A.10)

Equation A.10: Plain equations for independent system 5

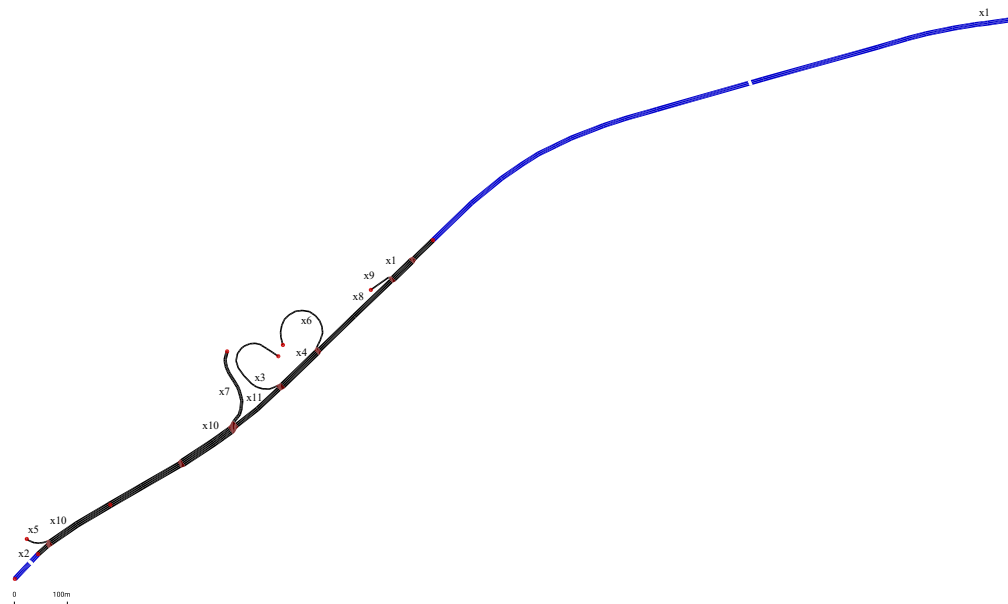


Figure A.7: Independent system 6

$$\begin{aligned}
x_1 &= x_1 \\
x_2 &= x_2 \\
x_3 &= x_3 \\
x_4 &= x_4 \\
x_5 &= x_5 \\
x_6 &= x_2 + x_3 \\
x_7 &= x_7 \\
x_8 &= x_2 + x_3 + x_7 - x_4 - x_5 \\
x_9 &= x_4 + x_5 - x_7 \\
x_{10} &= x_{10} \\
x_{11} &= x_4 - x_{10} \\
x_{12} &= x_1 + x_{10} - x_4 \\
x_{13} &= x_4 + x_5
\end{aligned}$$

(A.11)

Equation A.11: Simplified equations for independent system 5

$$\begin{aligned}
x_{10} &= x_{11} + x_7 \\
x_{10} &= x_5 + x_2 \\
x_4 &= x_8 + x_6 \\
x_4 &= x_{11} + x_3 \\
x_1 &= x_9 + x_8
\end{aligned}$$

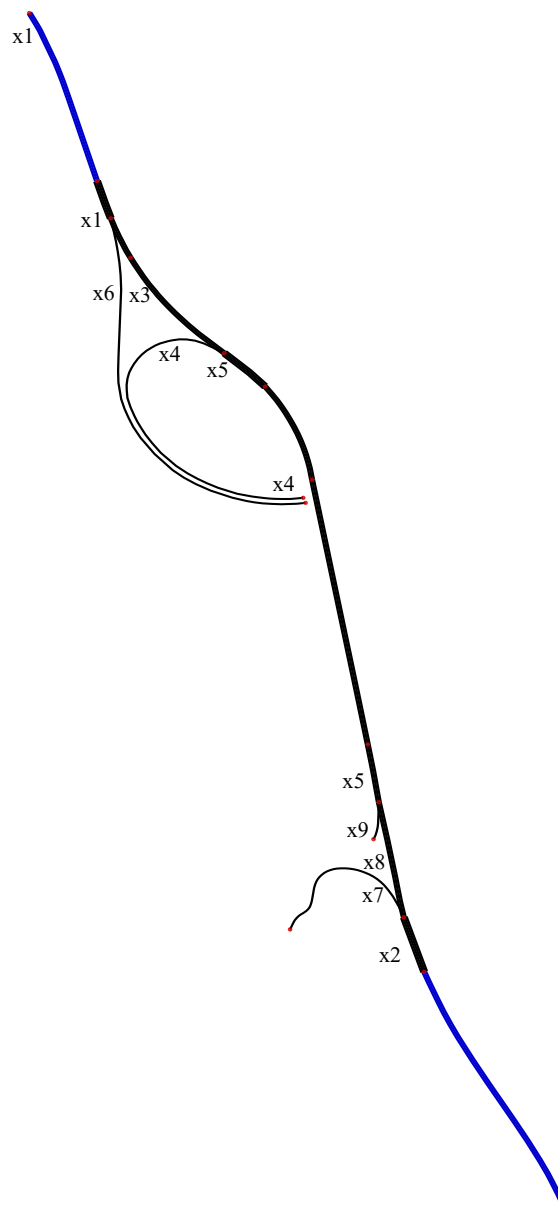
(A.12)

Equation A.12: Plain equations for independent system 6

$$\begin{aligned}
 x_1 &= x_1 \\
 x_2 &= x_2 \\
 x_3 &= x_3 \\
 x_4 &= x_4 \\
 x_5 &= x_5 \\
 x_6 &= x_6 \\
 x_7 &= x_2 + x_3 + x_5 - x_4 \\
 x_8 &= x_4 - x_6 \\
 x_9 &= x_1 + x_6 - x_4 \\
 x_{10} &= x_2 + x_5 \\
 x_{11} &= x_4 - x_3
 \end{aligned}$$

(A.13)

Equation A.13: Simplified equations for independent system 6



$$x_5 = x_4 + x_3$$

$$x_5 = x_9 + x_8$$

$$x_2 = x_8 + x_7$$

$$x_1 = x_6 + x_3$$

(A.14)

Equation A.14: Plain equations for independent system 7

$$x_1 = x_1$$

$$x_2 = x_2$$

$$x_3 = x_3$$

$$x_4 = x_4$$

$$x_5 = x_3 + x_4$$

$$x_6 = x_1 - x_3$$

$$x_7 = x_7$$

$$x_8 = x_2 - x_7$$

$$x_9 = x_3 + x_4 + x_7 - x_2$$

(A.15)

Equation A.15: Simplified equations for independent system 7

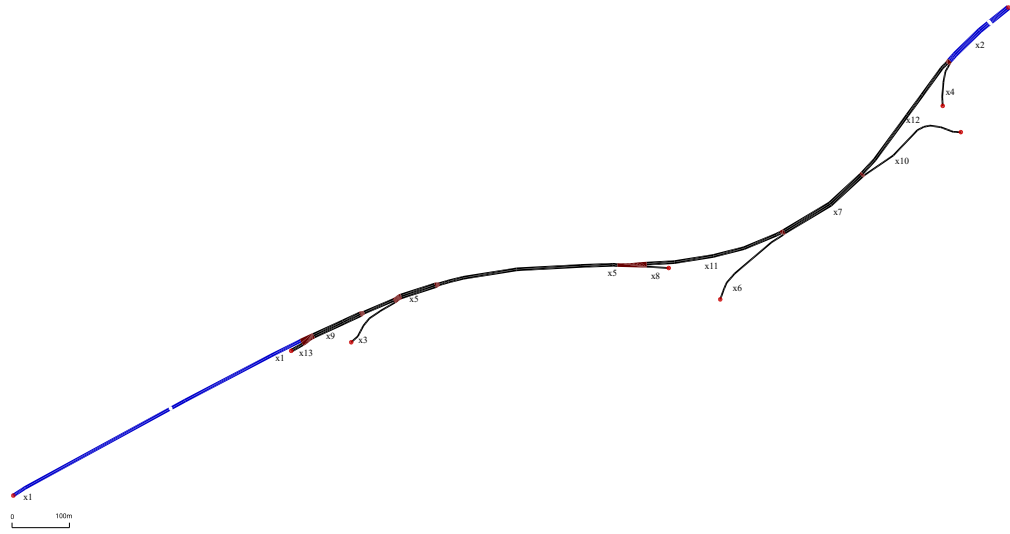


Figure A.9: Independent system 8

$$x_9 = x_{13} + x_1$$

$$x_7 = x_{11} + x_6$$

$$x_7 = x_{12} + x_{10}$$

$$x_5 = x_9 + x_3$$

$$x_5 = x_{11} + x_8$$

$$x_2 = x_{12} + x_4$$

(A.16)

Equation A.16: Plain equations for independent system 8

$$\begin{aligned}
 x_1 &= x_1 \\
 x_2 &= x_2 \\
 x_3 &= x_3 \\
 x_4 &= x_4 \\
 x_5 &= x_5 \\
 x_6 &= x_6 \\
 x_7 &= x_7 \\
 x_8 &= x_5 + x_6 - x_7 \\
 x_9 &= x_5 - x_3 \\
 x_{10} &= x_4 + x_7 - x_2 \\
 x_{11} &= x_7 - x_6 \\
 x_{12} &= x_2 - x_4 \\
 x_{13} &= x_5 - x_1 - x_3
 \end{aligned}
 \tag{A.17}$$

Equation A.17: Simplified equations for independent system 8

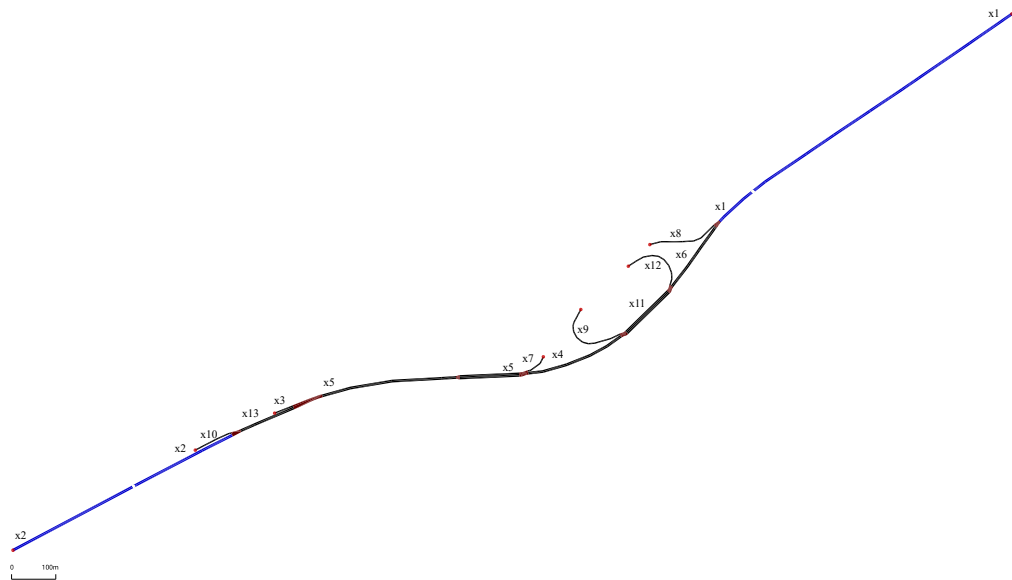


Figure A.10: Independent system 9

$$x_{13} = x_{10} + x_2$$

$$x_{11} = x_{12} + x_6$$

$$x_{11} = x_9 + x_4$$

$$x_5 = x_7 + x_4$$

$$x_5 = x_{13} + x_3$$

$$x_1 = x_8 + x_6$$

(A.18)

Equation A.18: Plain equations for independent system 9

$$x_1 = x_1$$

$$x_2 = x_2$$

$$x_3 = x_3$$

$$x_4 = x_4$$

$$x_5 = x_5$$

$$x_6 = x_6$$

$$x_7 = x_5 - x_4$$

$$x_8 = x_1 - x_6$$

$$x_9 = x_9$$

$$x_{10} = x_5 - x_2 - x_3$$

$$x_{11} = x_4 + x_9$$

$$x_{12} = x_4 + x_9 - x_6$$

$$x_{13} = x_5 - x_3$$

(A.19)

Equation A.19: Simplified equations for independent system 9

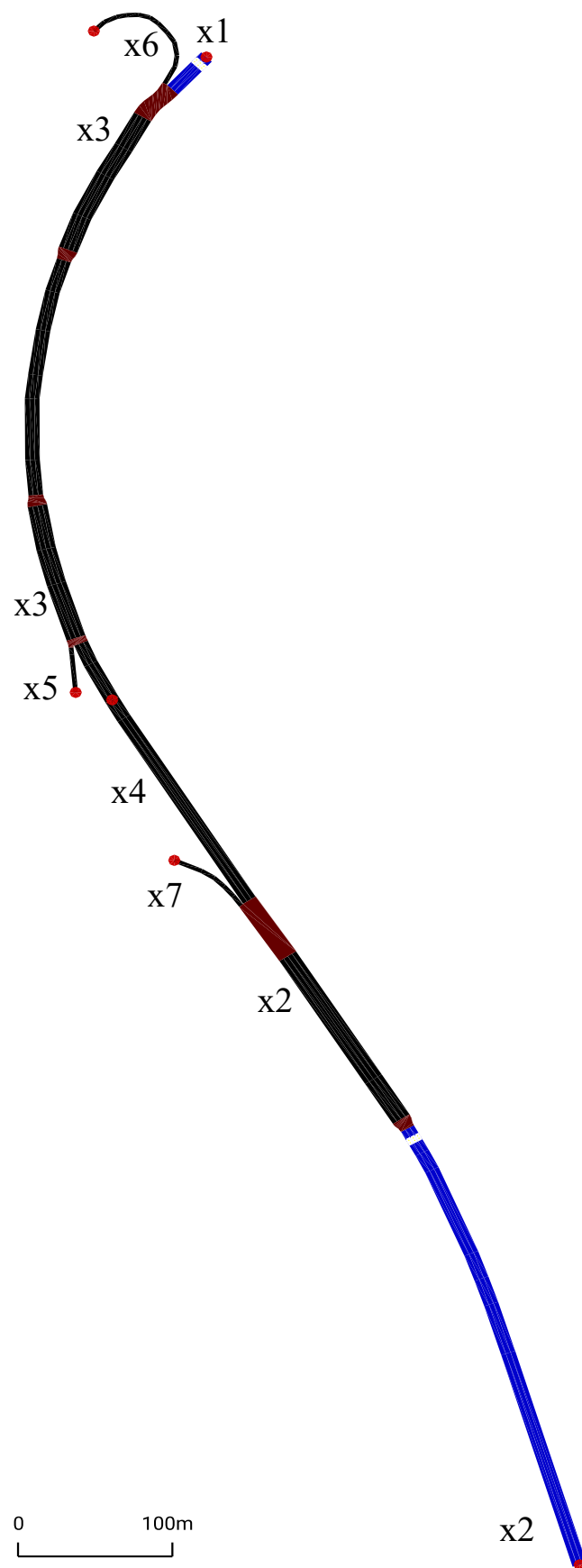


Figure A.11: Independent system 10

$$\begin{aligned}x_3 &= x_6 + x_1 \\x_3 &= x_5 + x_4 \\x_2 &= x_7 + x_4\end{aligned}\tag{A.20}$$

Equation A.20: Plain equations for independent system 10

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_3 \\x_4 &= x_4 \\x_5 &= x_3 - x_4 \\x_6 &= x_3 - x_1 \\x_7 &= x_2 - x_4\end{aligned}\tag{A.21}$$

Equation A.21: Simplified equations for independent system 10

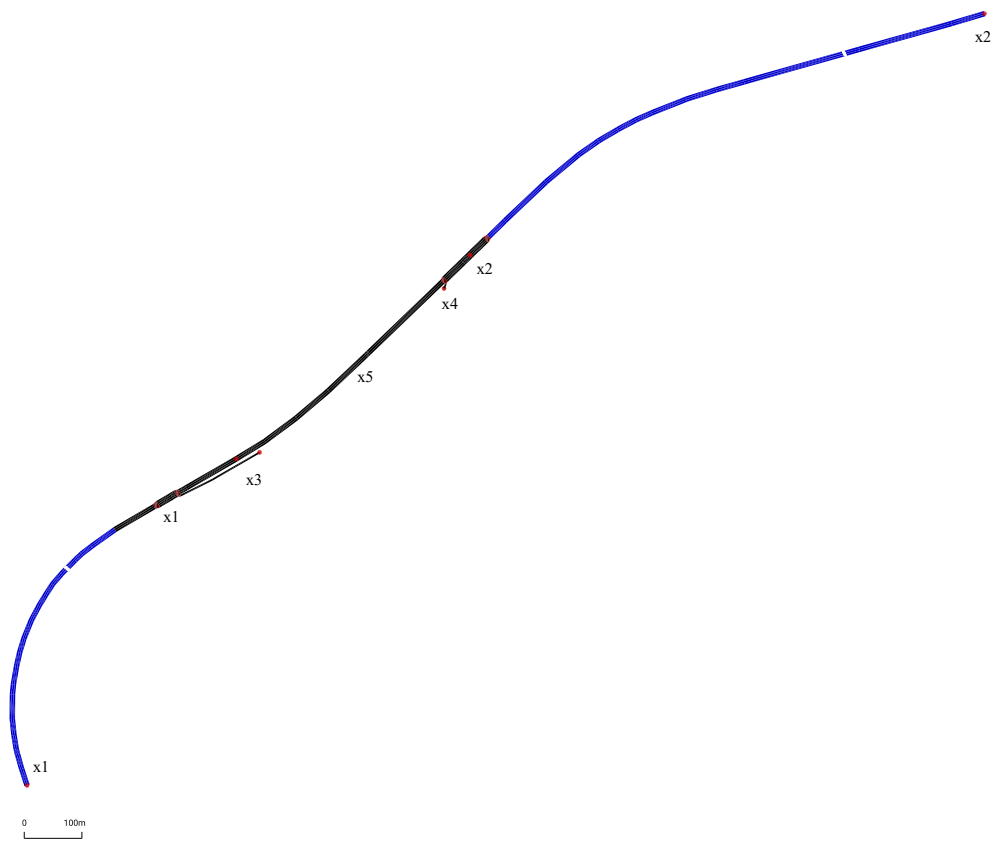


Figure A.12: Independent system 11

$$x_2 = x_5 + x_4$$

$$x_1 = x_5 + x_3$$

(A.22)

Equation A.22: Plain equations for independent system 11

$$\begin{aligned}
 x_1 &= x_1 \\
 x_2 &= x_2 \\
 x_3 &= x_3 \\
 x_4 &= x_2 + x_3 - x_1 \\
 x_5 &= x_1 - x_3
 \end{aligned}
 \tag{A.23}$$

Equation A.23: Simplified equations for independent system 11

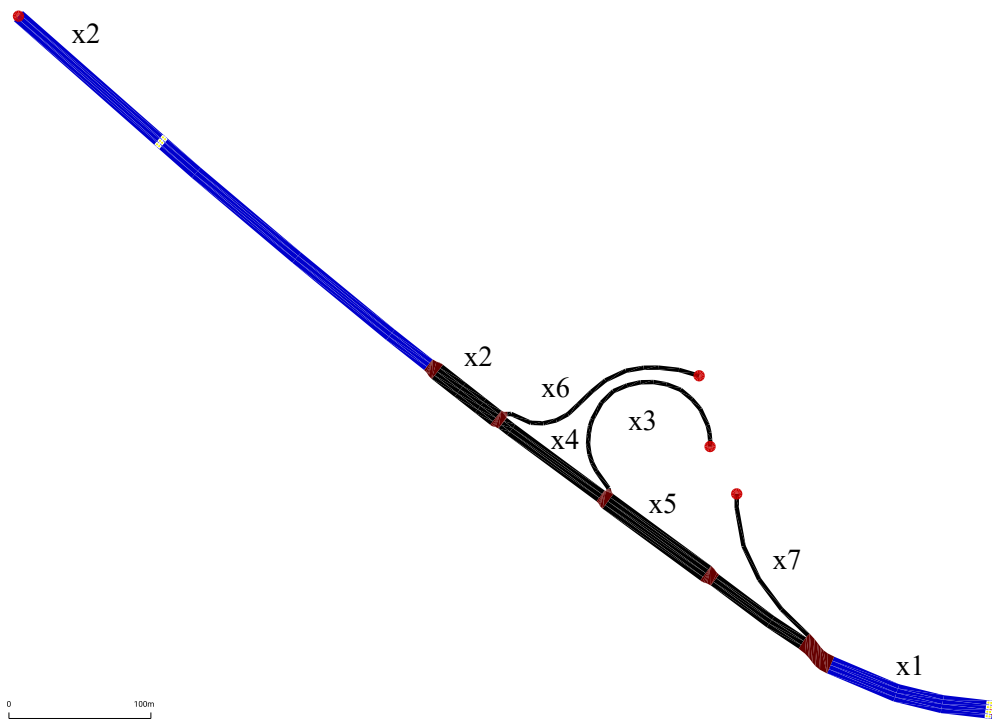


Figure A.13: Independent system 12

$$\begin{aligned}
 x_5 &= x_4 + x_3 \\
 x_2 &= x_6 + x_4 \\
 x_1 &= x_7 + x_5
 \end{aligned}
 \tag{A.24}$$

Equation A.24: Plain equations for independent system 12

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_3 \\x_4 &= x_4 \\x_5 &= x_3 + x_4 \\x_6 &= x_2 - x_4 \\x_7 &= x_1 - x_3 - x_4\end{aligned}\tag{A.25}$$

Equation A.25: Simplified equations for independent system 12

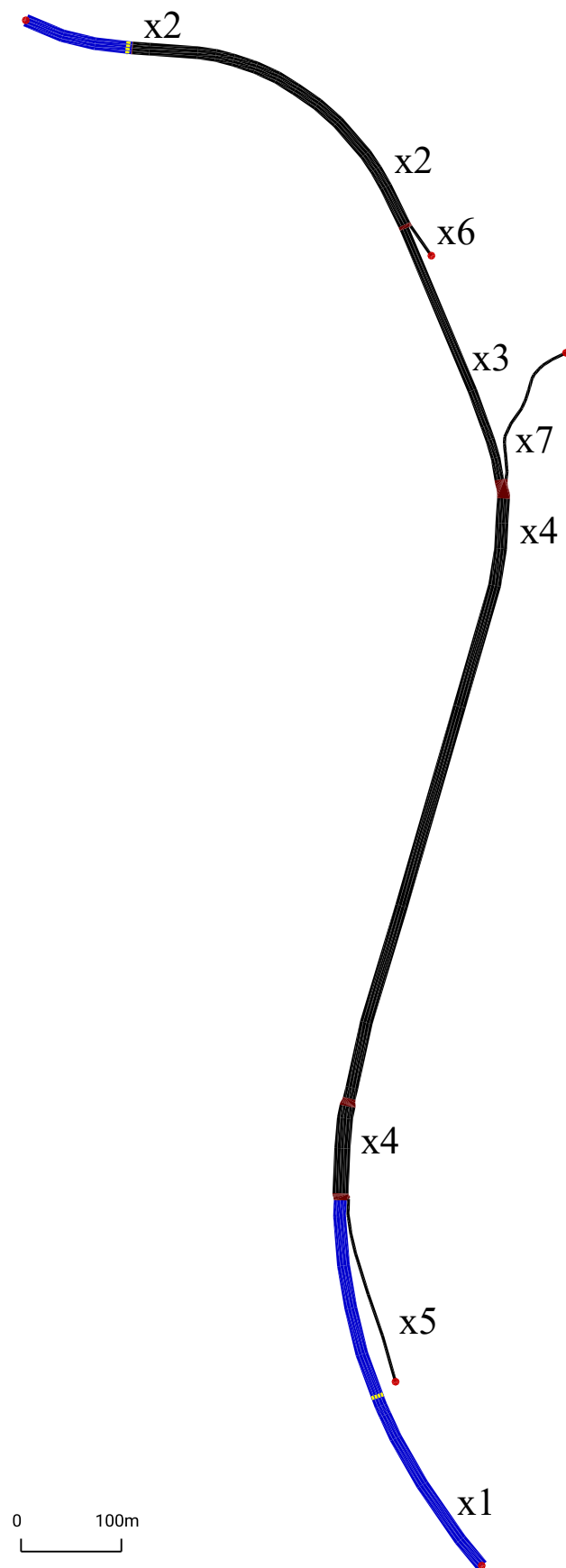


Figure A.14: Independent system 13

$$x_4 = x_5 + x_1$$

$$x_4 = x_7 + x_3$$

$$x_2 = x_6 + x_3$$

(A.26)

Equation A.26: Plain equations for independent system 13

$$x_1 = x_1$$

$$x_2 = x_2$$

$$x_3 = x_3$$

$$x_4 = x_4$$

$$x_5 = x_4 - x_1$$

$$x_6 = x_2 - x_3$$

$$x_7 = x_4 - x_3$$

(A.27)

Equation A.27: Simplified equations for independent system 13

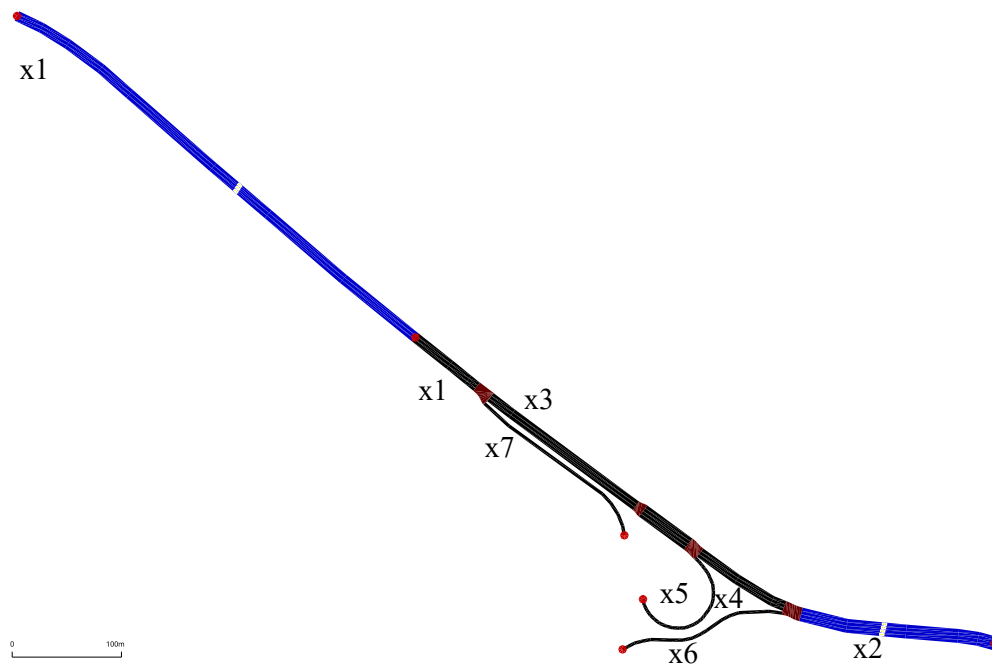


Figure A.15: Independent system 14

$$x_3 = x_6 + x_5$$

$$x_2 = x_6 + x_4$$

$$x_1 = x_7 + x_3$$

(A.28)

Equation A.28: Plain equations for independent system 14

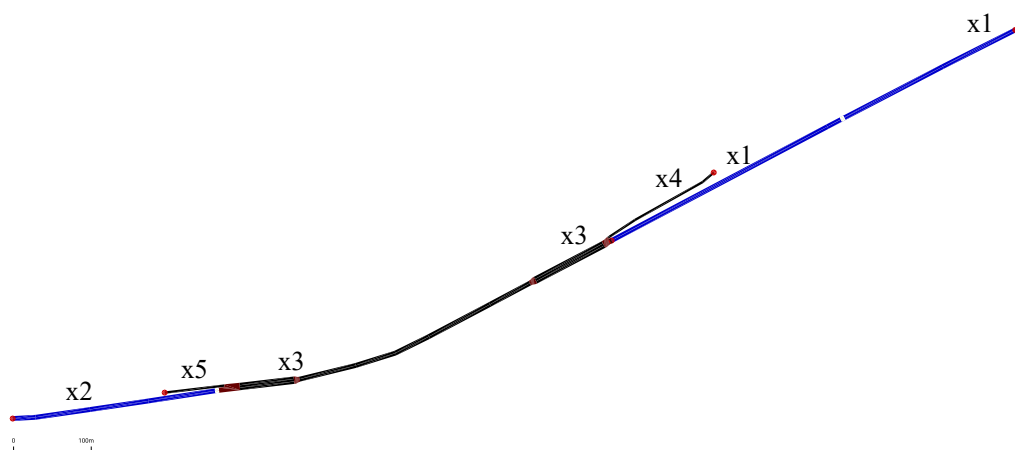


Figure A.16: Independent system 15

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_3 \\x_4 &= x_4 \\x_5 &= x_3 + x_4 - x_2 \\x_6 &= x_2 - x_4 \\x_7 &= x_1 - x_3\end{aligned}$$

(A.29)

Equation A.29: Simplified equations for independent system 14

$$\begin{aligned}x_3 &= x_4 + x_1 \\x_3 &= x_5 + x_2\end{aligned}$$

(A.30)

Equation A.30: Plain equations for independent system 15

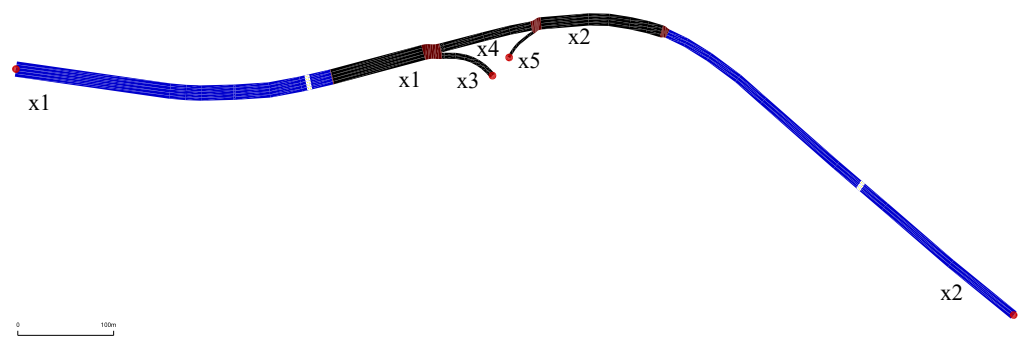


Figure A.17: Independent system 16

$$\begin{aligned}
x_1 &= x_1 \\
x_2 &= x_2 \\
x_3 &= x_3 \\
x_4 &= x_3 - x_1 \\
x_5 &= x_3 - x_2
\end{aligned}$$

(A.31)

Equation A.31: Simplified equations for independent system 15

$$\begin{aligned}
x_2 &= x_5 + x_4 \\
x_1 &= x_4 + x_3
\end{aligned}$$

(A.32)

Equation A.32: Plain equations for independent system 16

$$\begin{aligned}
x_1 &= x_1 \\
x_2 &= x_2 \\
x_3 &= x_3 \\
x_4 &= x_1 - x_3 \\
x_5 &= x_2 + x_3 - x_1
\end{aligned}$$

(A.33)

Equation A.33: Simplified equations for independent system 16

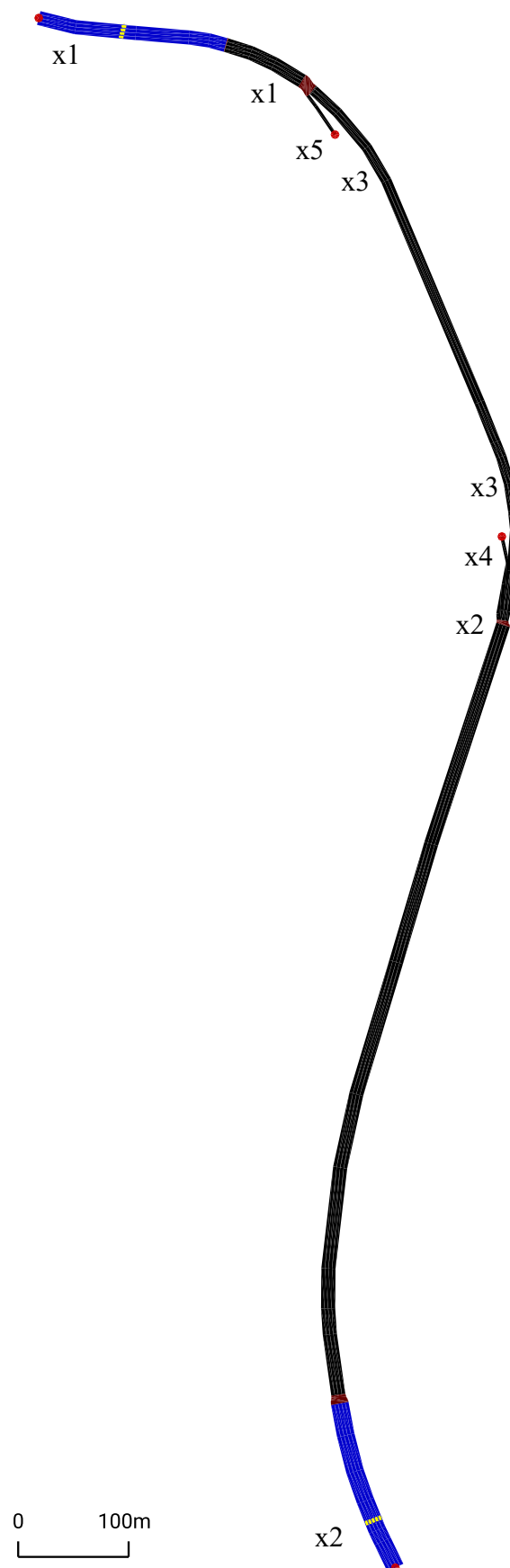


Figure A.18: Independent system 17

$$x_2 = x_4 + x_3$$

$$x_1 = x_5 + x_3$$

(A.34)

Equation A.34: Plain equations for independent system 17

$$x_1 = x_1$$

$$x_2 = x_2$$

$$x_3 = x_3$$

$$x_4 = x_2 - x_3$$

$$x_5 = x_1 - x_3$$

(A.35)

Equation A.35: Simplified equations for independent system 17

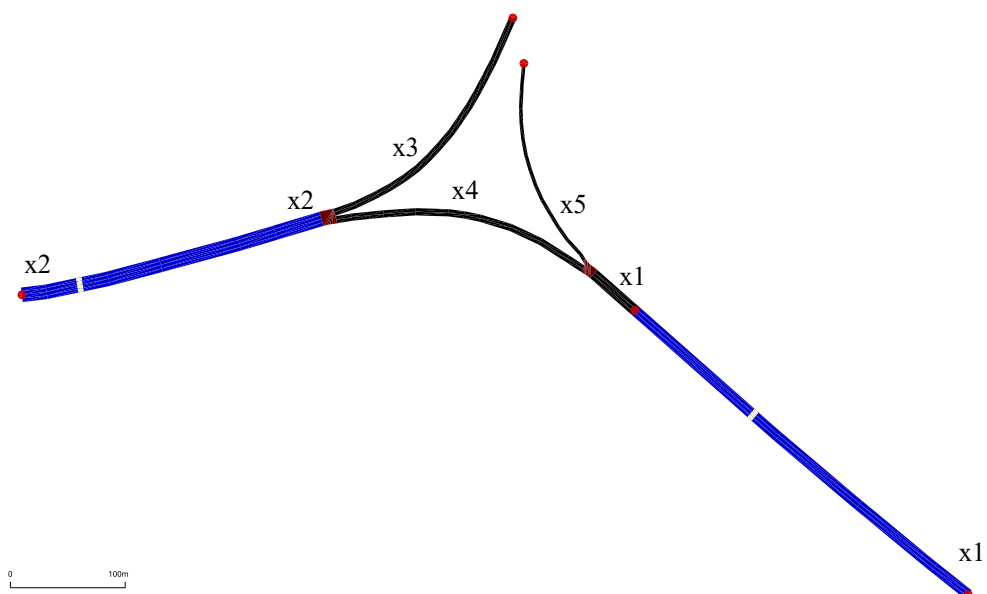
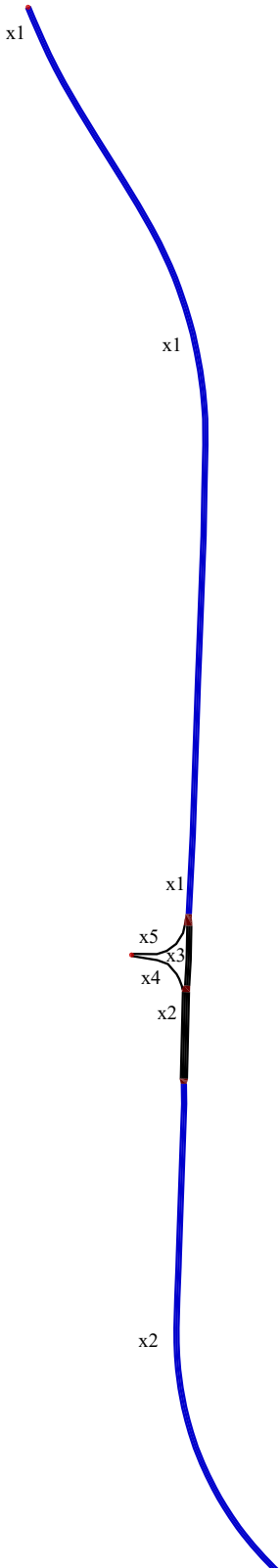


Figure A.19: Independent system 18

$$\begin{aligned}x_2 &= x_4 + x_3 \\x_1 &= x_5 + x_4\end{aligned}$$

(A.36)

Equation A.36: Plain equations for independent system 18



$$\begin{aligned}x_2 &= x_4 + x_3 \\x_1 &= x_5 + x_4\end{aligned}\tag{A.38}$$

Equation A.38: Plain equations for independent system 19

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_3 \\x_4 &= x_2 - x_3 \\x_5 &= x_1 + x_3 - x_2\end{aligned}\tag{A.39}$$

Equation A.39: Simplified equations for independent system 19

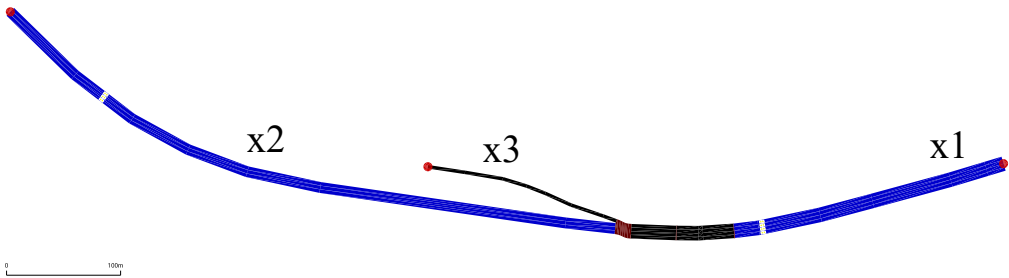


Figure A.21: Independent system 20

$$x_1 = x_3 + x_2\tag{A.40}$$

Equation A.40: Plain equations for independent system 20

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_1 - x_2\end{aligned}\tag{A.41}$$

Equation A.41: Simplified equations for independent system 20

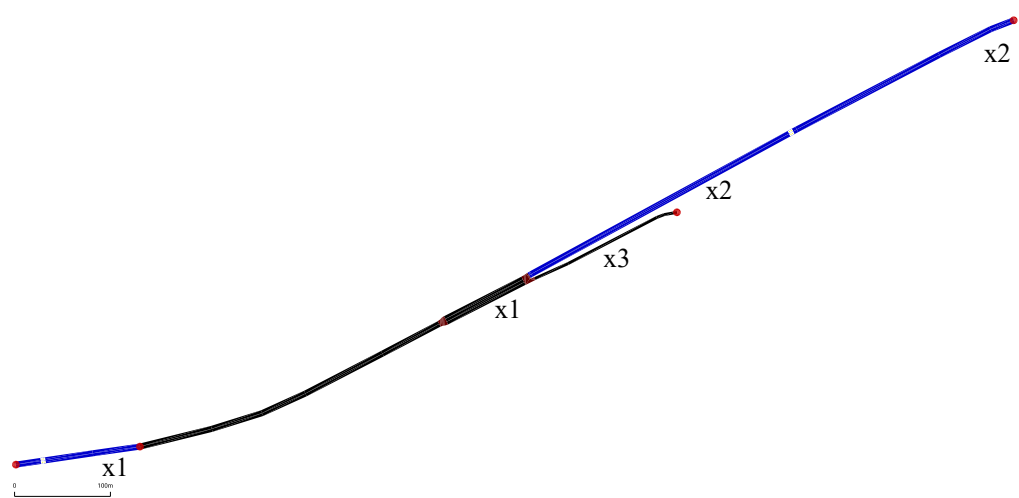


Figure A.22: Independent system 21

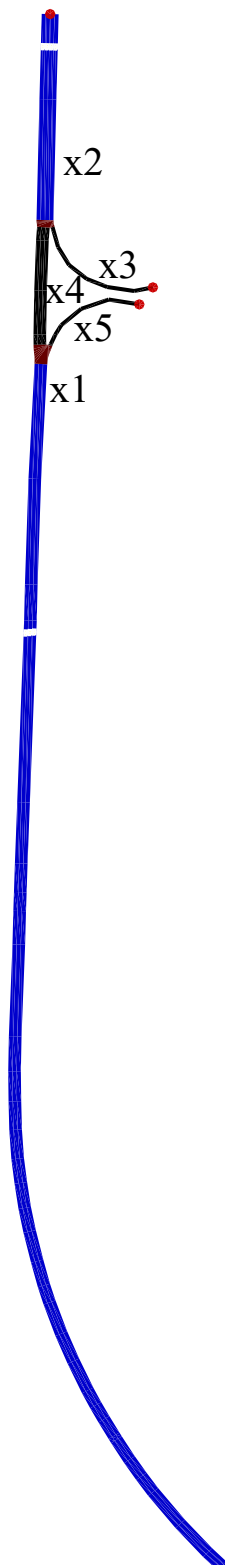
$$x_1 = x_3 + x_2\tag{A.42}$$

Equation A.42: Plain equations for independent system 21

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_1 - x_2\end{aligned}$$

(A.43)

Equation A.43: Simplified equations for independent system 21



$$\begin{aligned}x_2 &= x_4 + x_3 \\x_1 &= x_5 + x_4\end{aligned}$$

(A.44)

Equation A.44: Plain equations for independent system 22

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_3 \\x_4 &= x_2 - x_3 \\x_5 &= x_1 + x_3 - x_2\end{aligned}$$

(A.45)

Equation A.45: Simplified equations for independent system 22

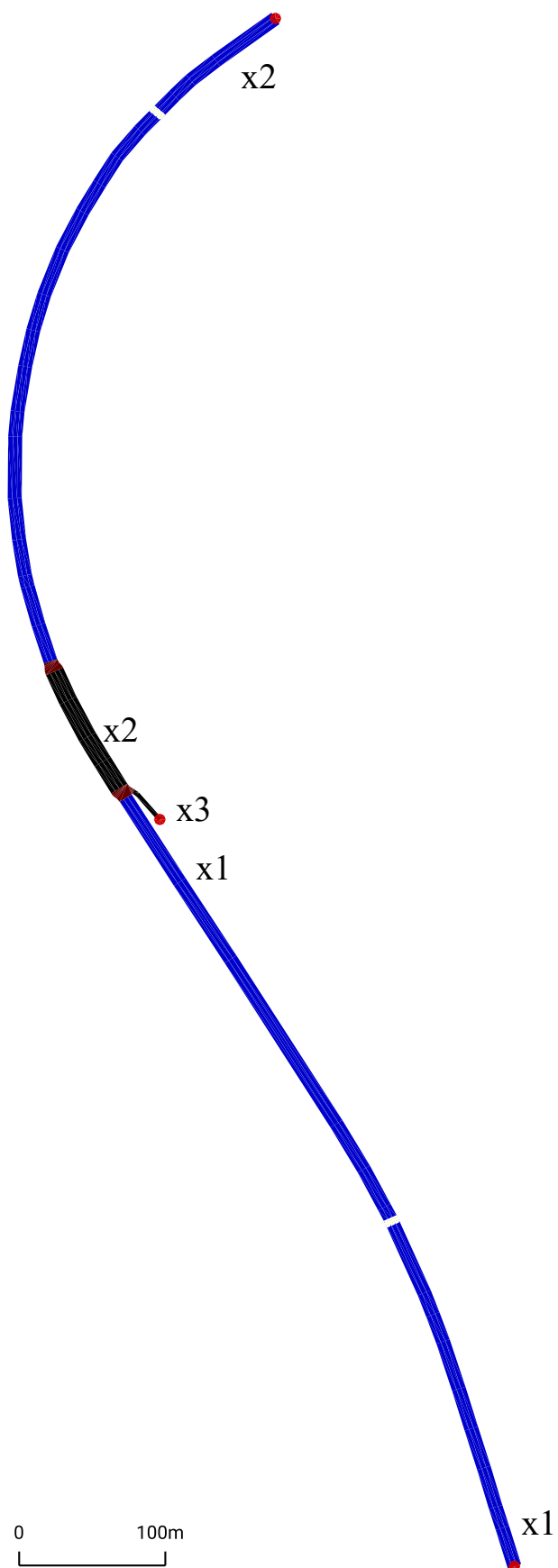


Figure A.24: Independent system 23

$$x_2 = x_3 + x_1 \tag{A.46}$$

Equation A.46: Plain equations for independent system 23

$$\begin{aligned} x_1 &= x_1 \\ x_2 &= x_2 \\ x_3 &= x_2 - x_1 \end{aligned} \tag{A.47}$$

Equation A.47: Simplified equations for independent system 23

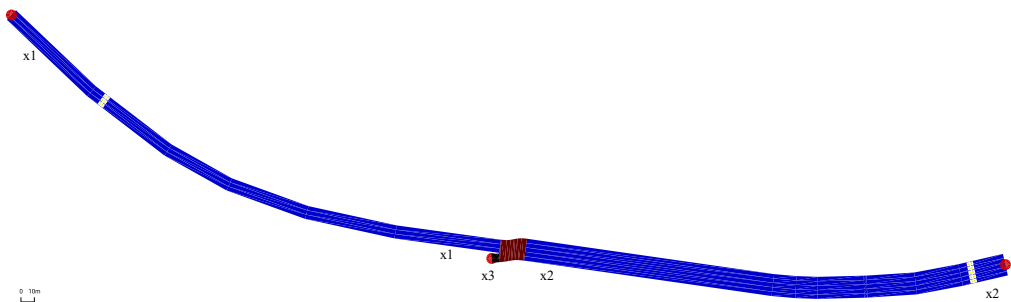


Figure A.25: Independent system 24

$$x_2 = x_3 + x_1 \tag{A.48}$$

Equation A.48: Plain equations for independent system 24

$$\begin{aligned}x_1 &= x_1 \\x_2 &= x_2 \\x_3 &= x_2 - x_1\end{aligned}\tag{A.49}$$

Equation A.49: Simplified equations for independent system 24

A.2 Data analysis

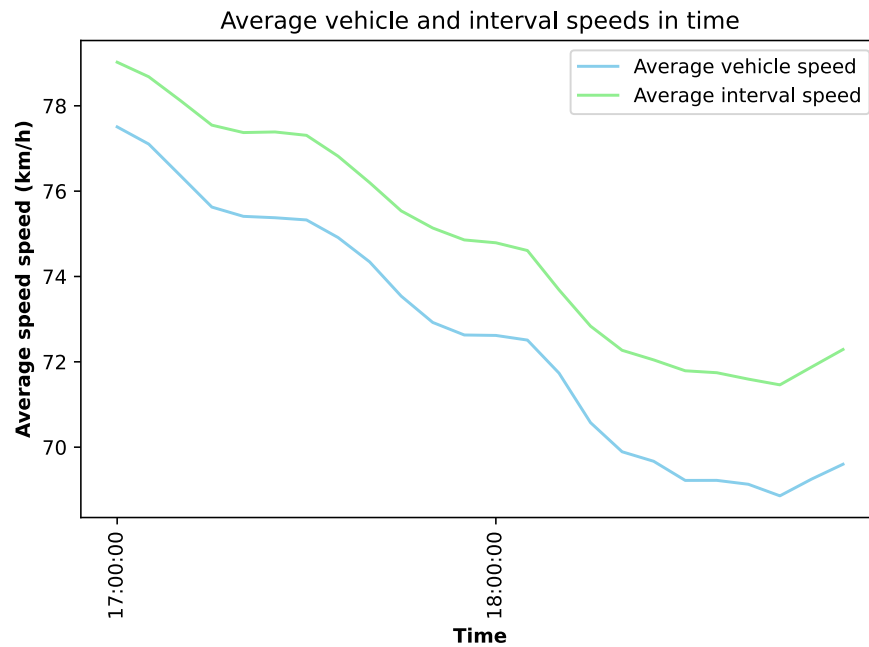


Figure A.26: Comparison between vehicle and interval average speeds in time for 17:00-19:00 period

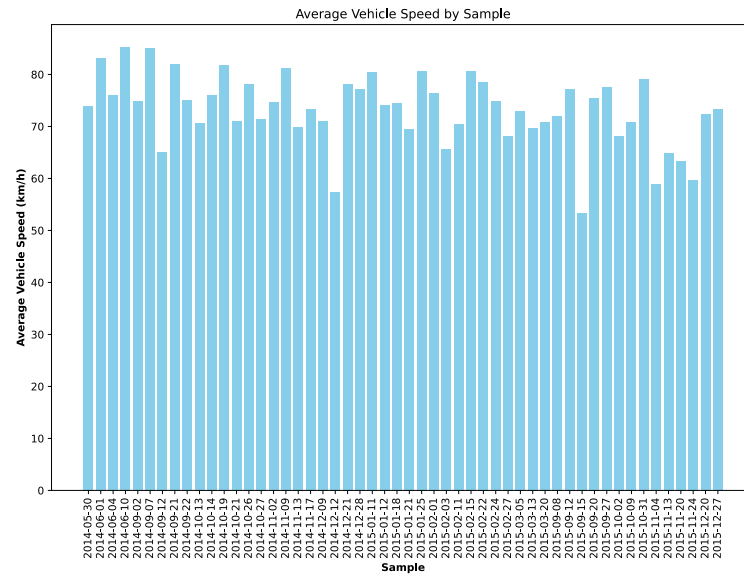


Figure A.27: Vehicle average speed by sample for 17:00-19:00 period

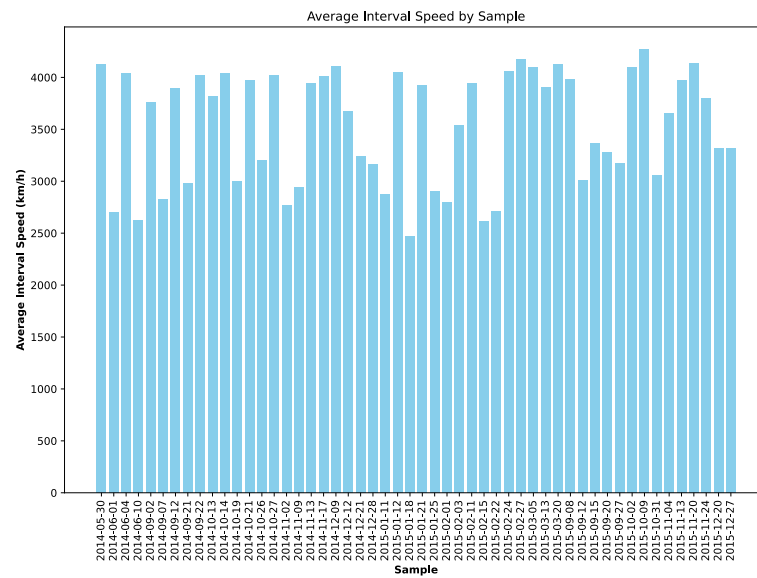


Figure A.28: Interval average speed by sample for 17:00-19:00 period

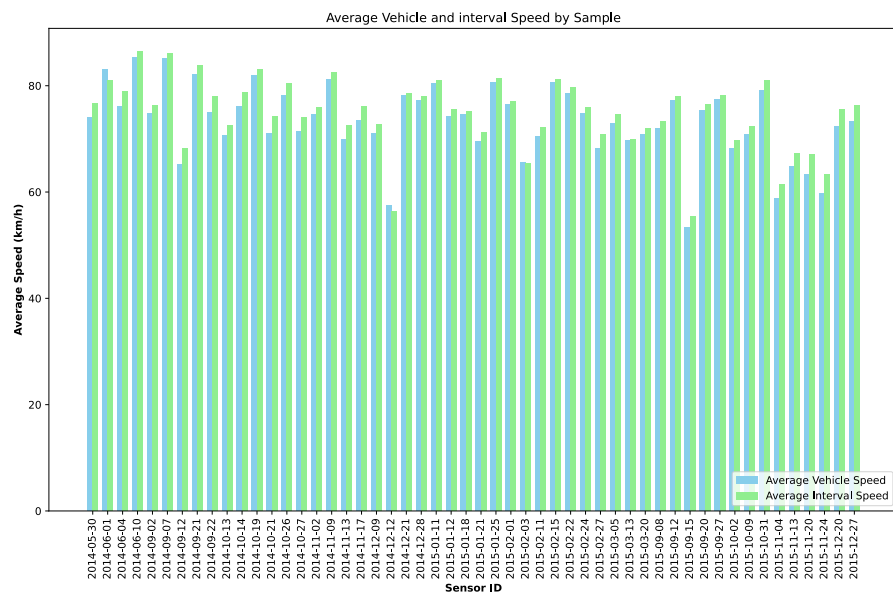


Figure A.29: Comparison between vehicle and interval average speeds by sensor for 17:00-19:00 period

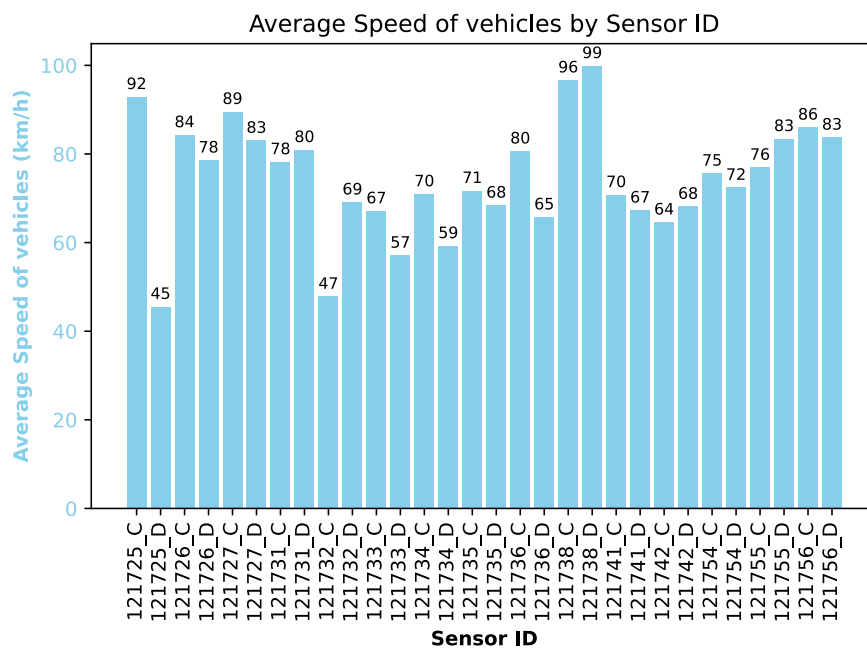


Figure A.30: Vehicle average speed by sensor for 17:00-19:00 period

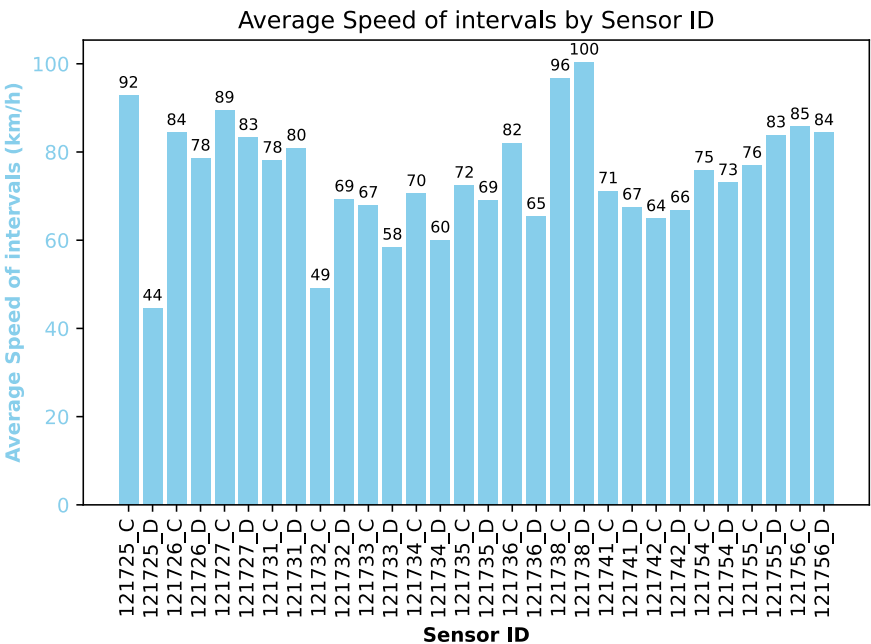


Figure A.31: Interval average speed by sensor for 17:00-19:00 period

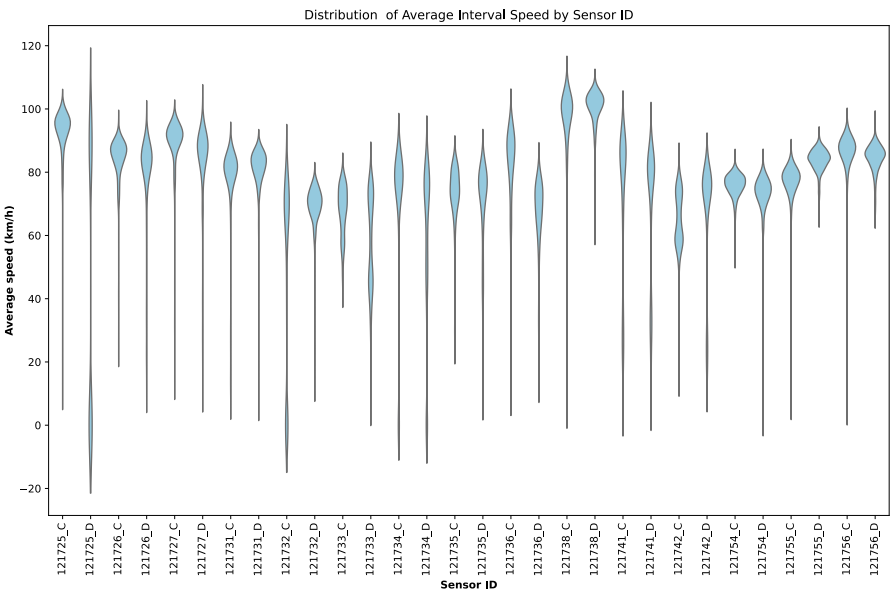


Figure A.32: Interval speed distribution by sensor for 17:00-19:00 period

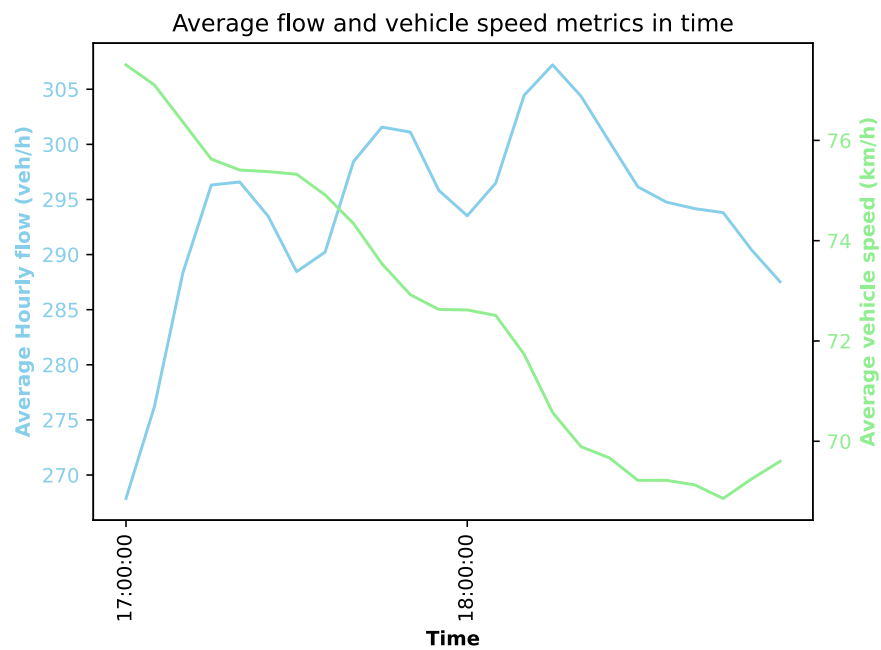


Figure A.33: Traffic flow and average vehicle speed in time for 17:00-19:00 period

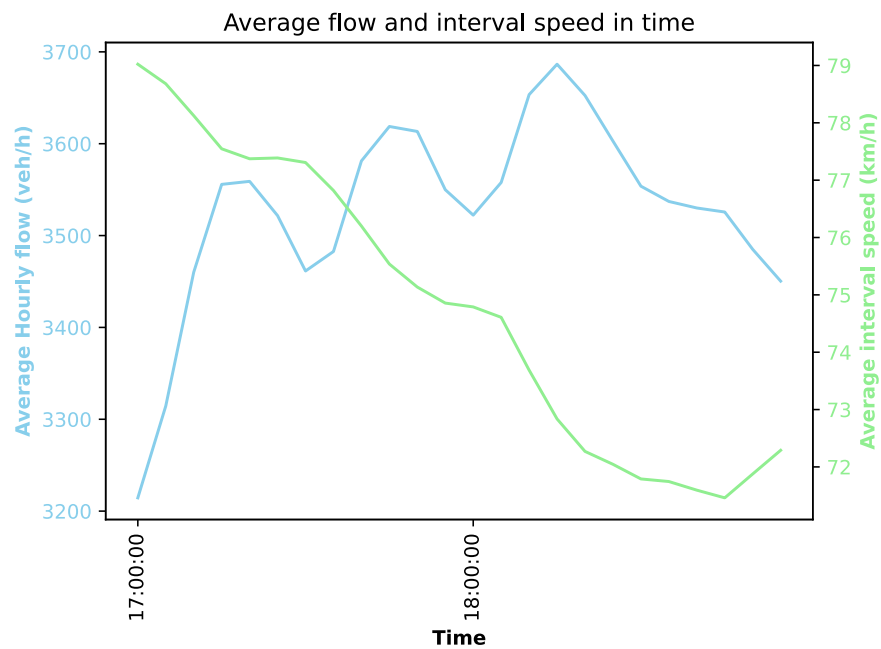


Figure A.34: Traffic flow and average interval speed in time for 17:00-19:00 period

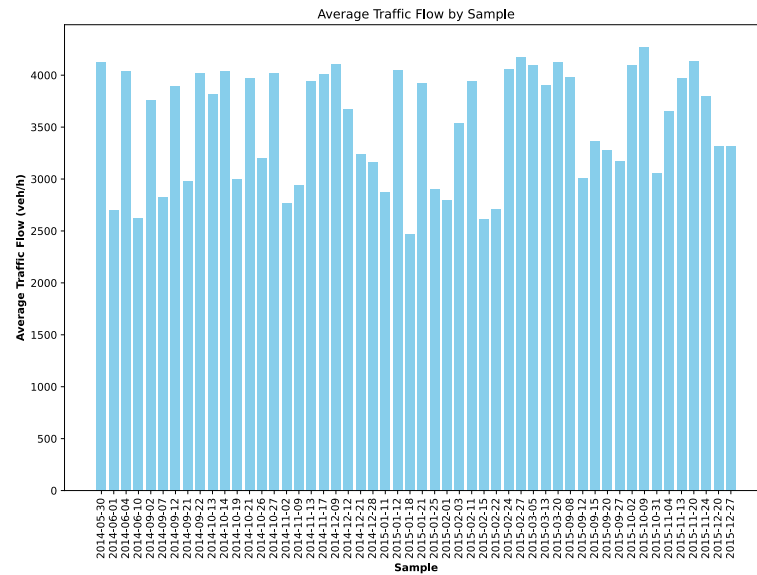


Figure A.35: Traffic flow by sample for 17:00-19:00 period

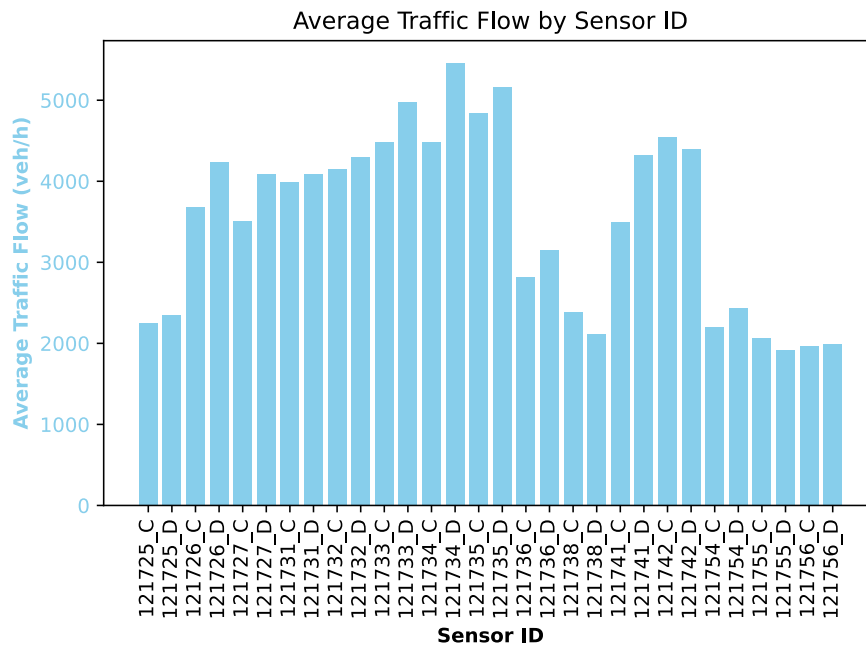


Figure A.36: Traffic flow by sensor for 17:00-19:00 period

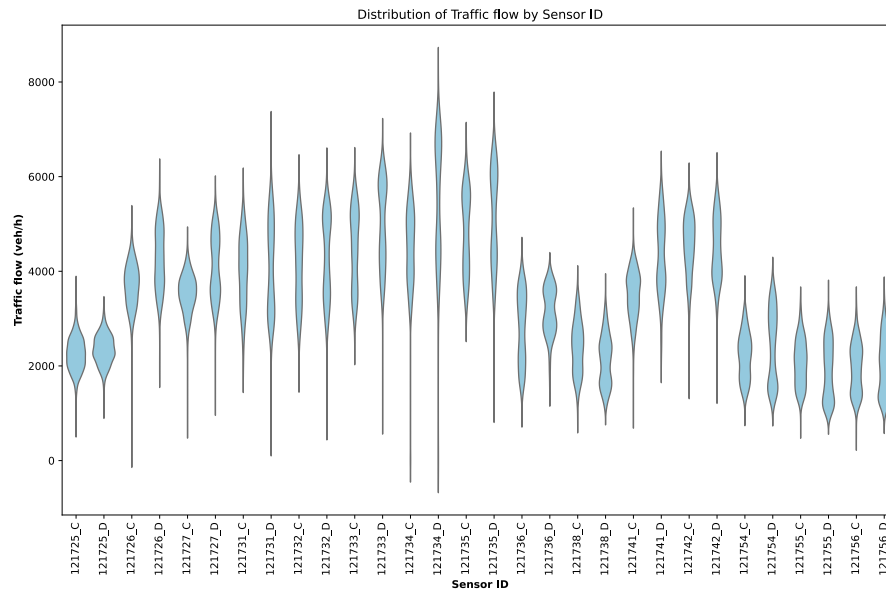


Figure A.37: Traffic flow distribution by sensor for 17:00-19:00 period

A.3 Experiment Validation

A.3.1 Validation of the simulation with no model for system 1

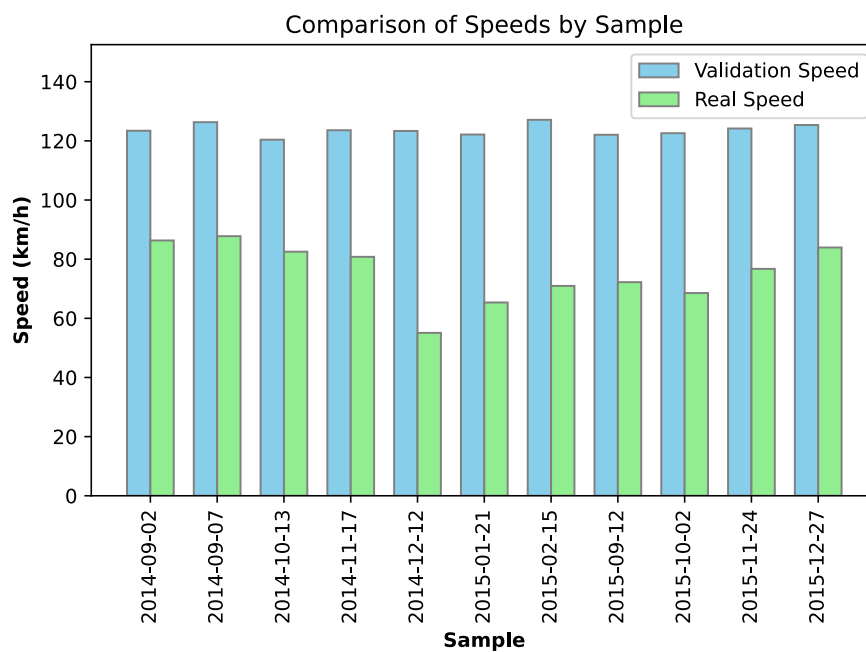


Figure A.38: Comparison of real speed with simulated speed for validation with no model per sample

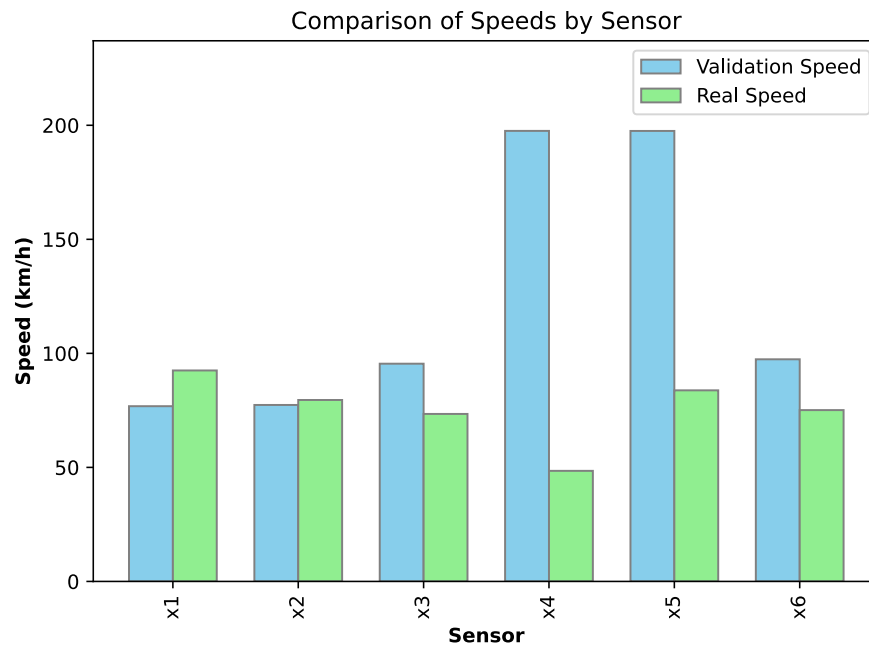


Figure A.39: Comparison of real speed with simulated speed for validation with no model by sensor

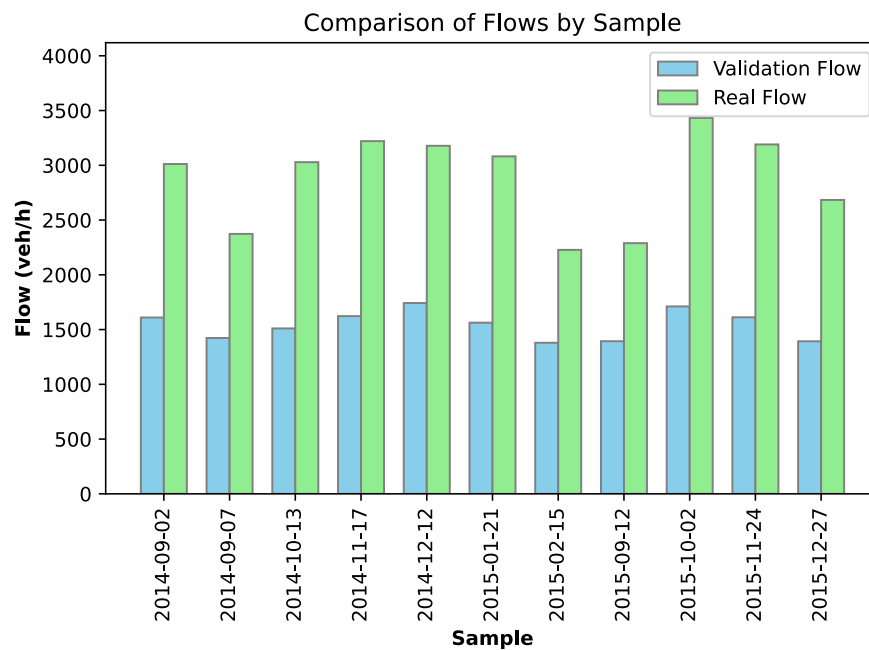


Figure A.40: Comparison of real volume with simulated volume for validation with no model per sample

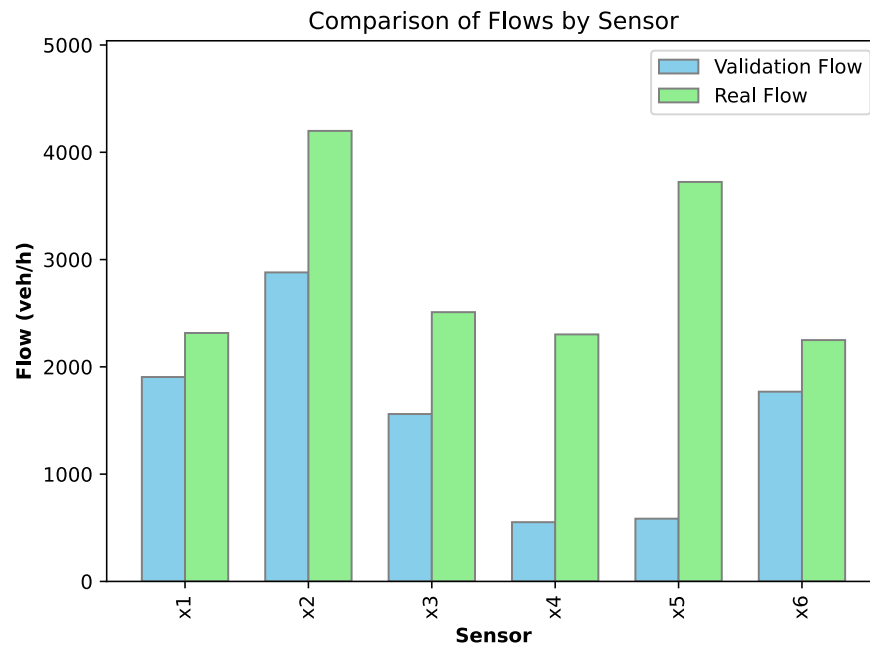


Figure A.41: Comparison of real volume with simulated volume for validation with no model by sensor

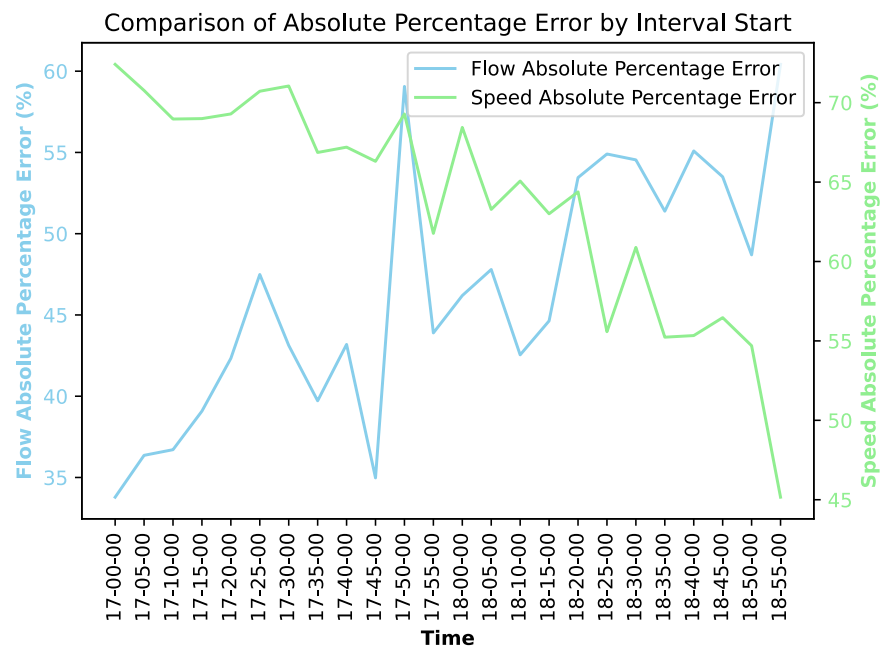


Figure A.42: Absolute percentage error for validation with no model in time

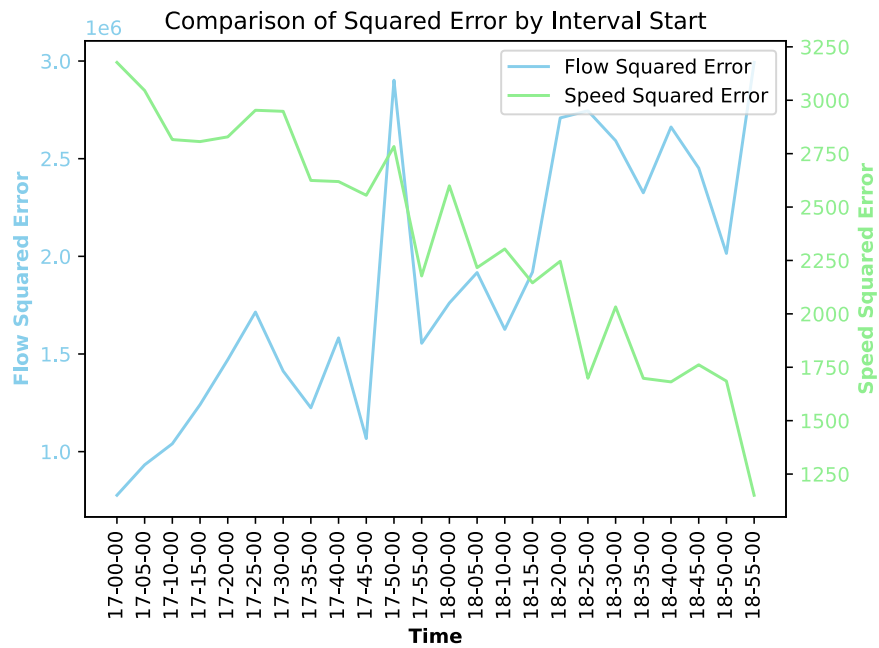


Figure A.43: Mean squared error for validation with no model in time

A.3.2 Validation of the simulation for model 1 for system 1

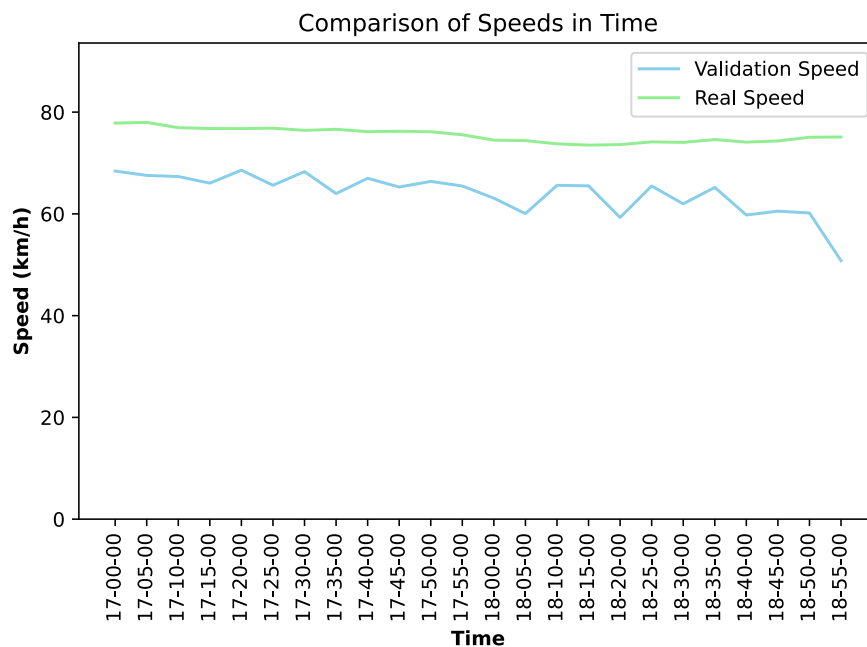


Figure A.44: Comparison of real speed with simulated speed for validation for model 1 in time

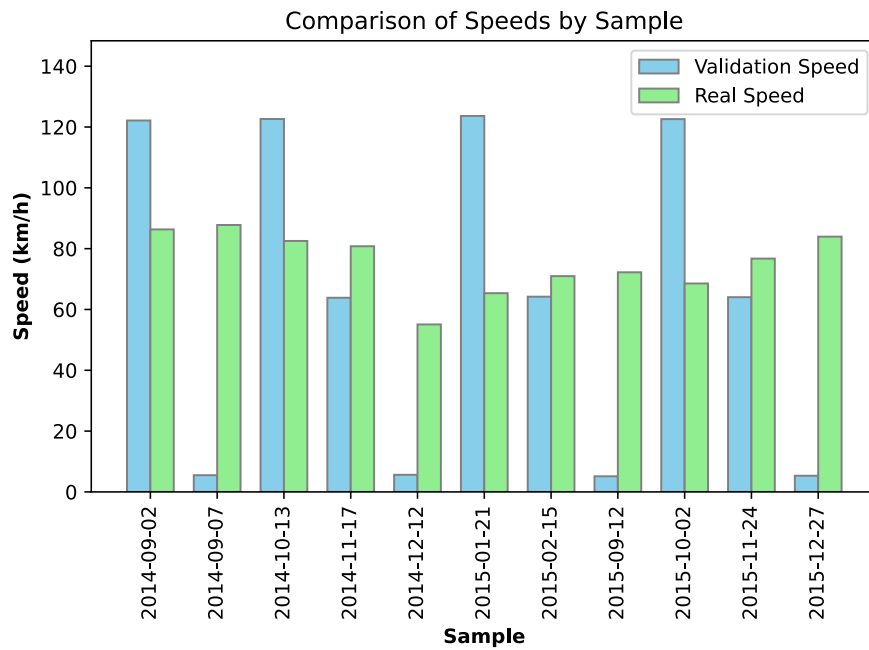


Figure A.45: Comparison of real speed with simulated speed for validation for model 1 per sample

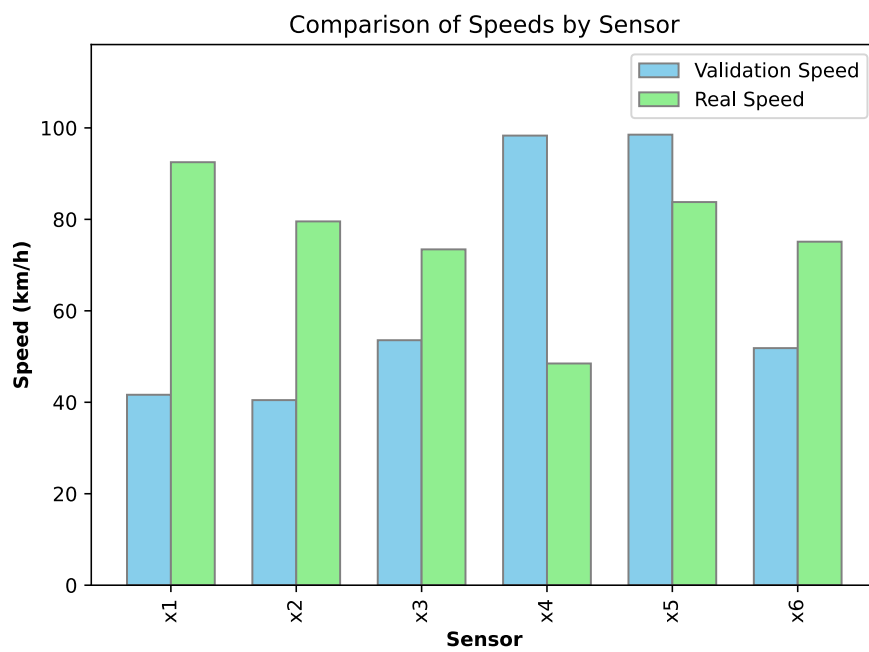


Figure A.46: Comparison of real speed with simulated speed for validation for model 1 by sensor

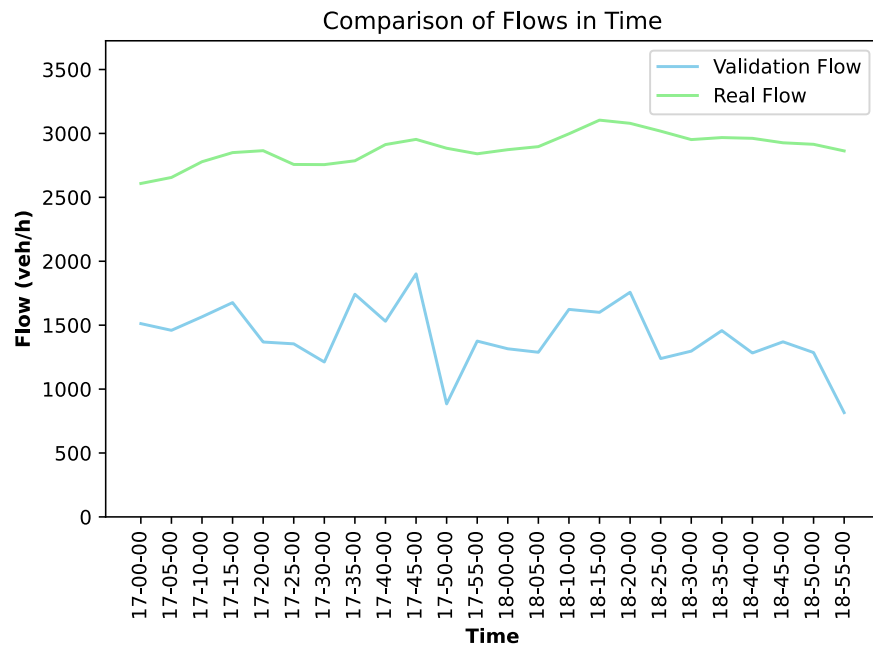


Figure A.47: Comparison of real volume with simulated volume for validation for model 1 in time

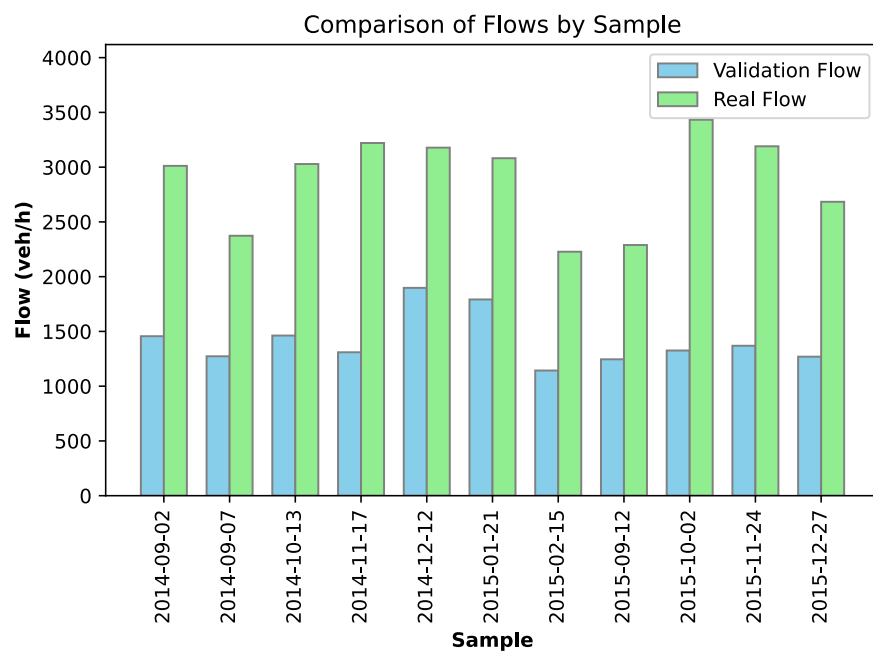


Figure A.48: Comparison of real volume with simulated volume for validation for model 1 per sample

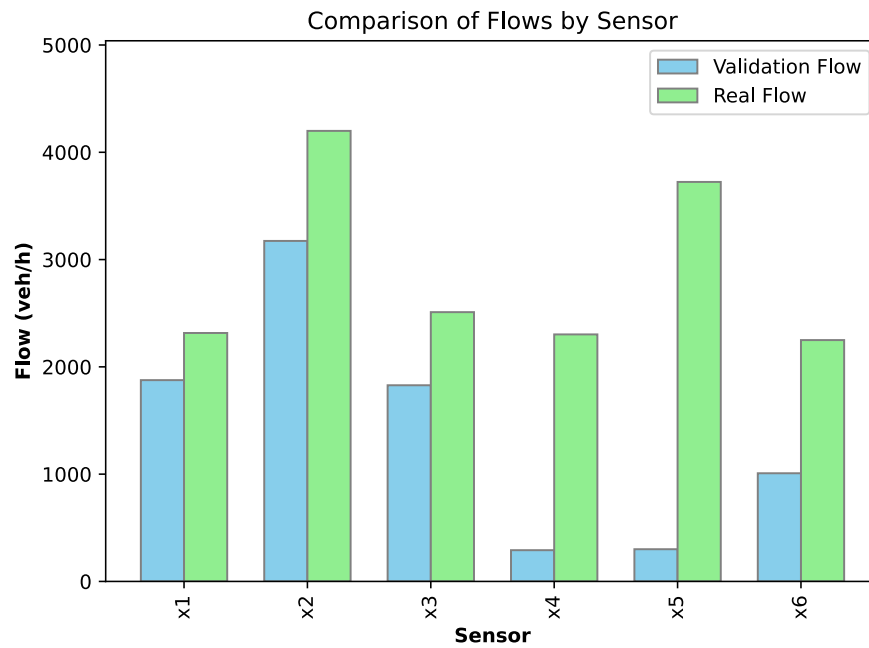


Figure A.49: Comparison of real volume with simulated volume for validation for model 1 by sensor

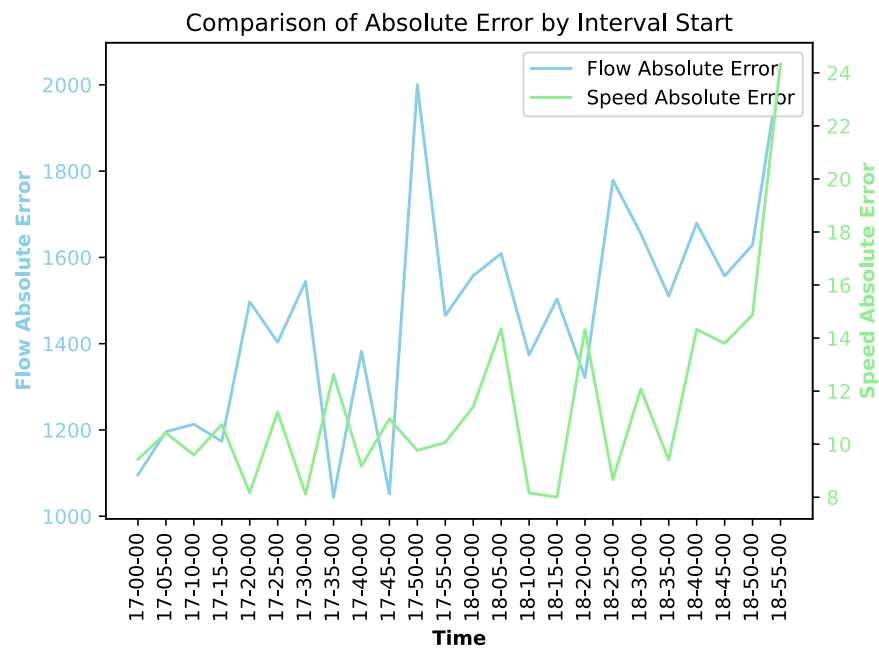


Figure A.50: Absolute error for validation for model 1 in time

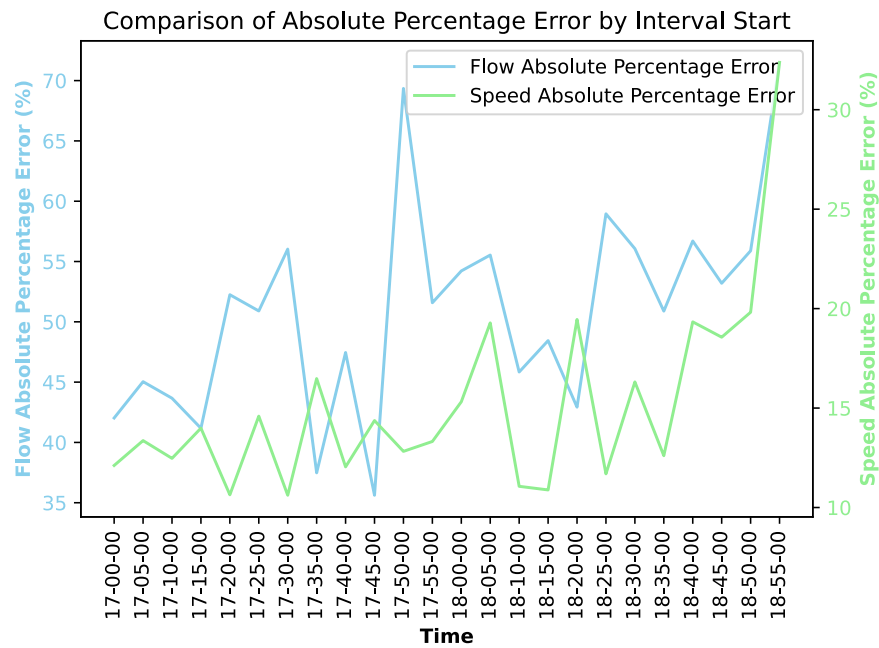


Figure A.51: Absolute percentage error for validation for model 1 in time

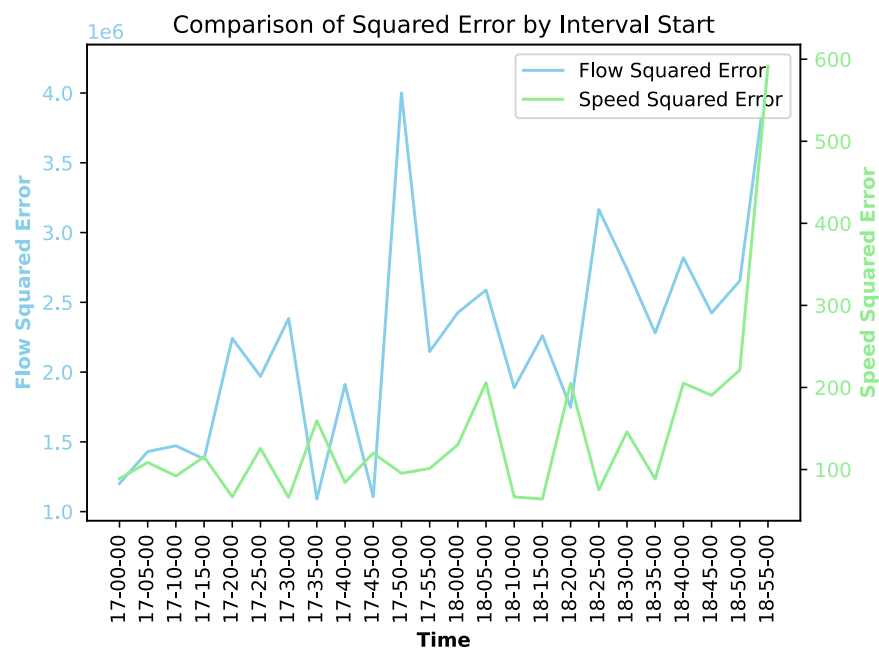


Figure A.52: Mean squared error for validation for model 1 in time

A.3.3 Validation of the simulation for model 2 for system 1

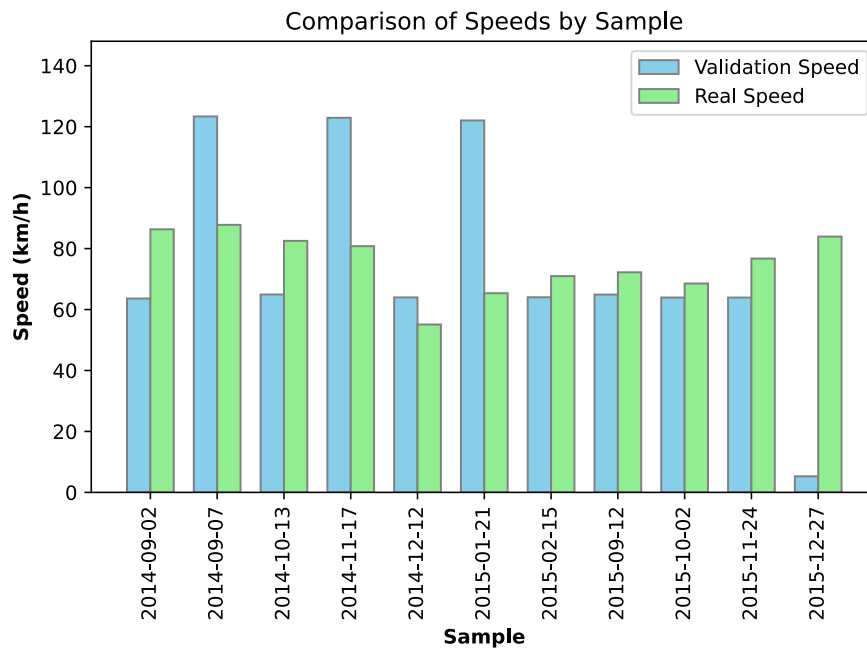


Figure A.53: Comparison of real speed with simulated speed for validation for model 2 per sample

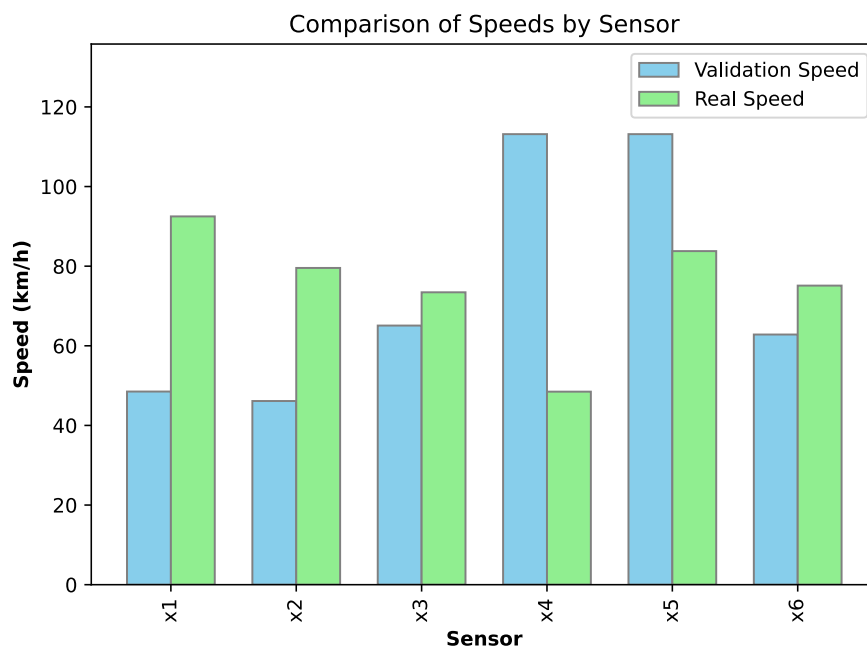


Figure A.54: Comparison of real speed with simulated speed for validation for model 2 by sensor

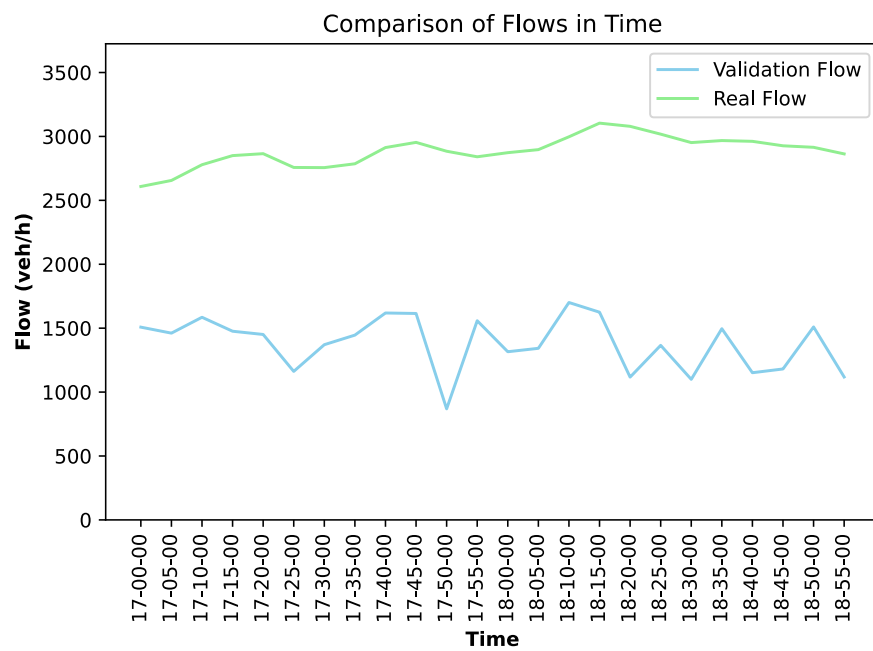


Figure A.55: Comparison of real volume with simulated volume for validation for model 2 in time

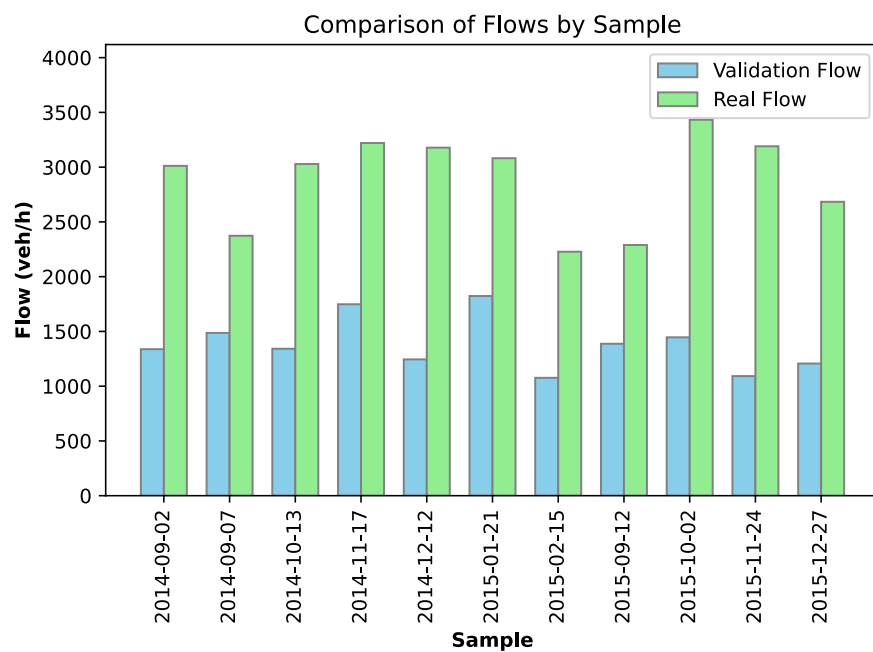


Figure A.56: Comparison of real volume with simulated volume for validation for model 2 per sample

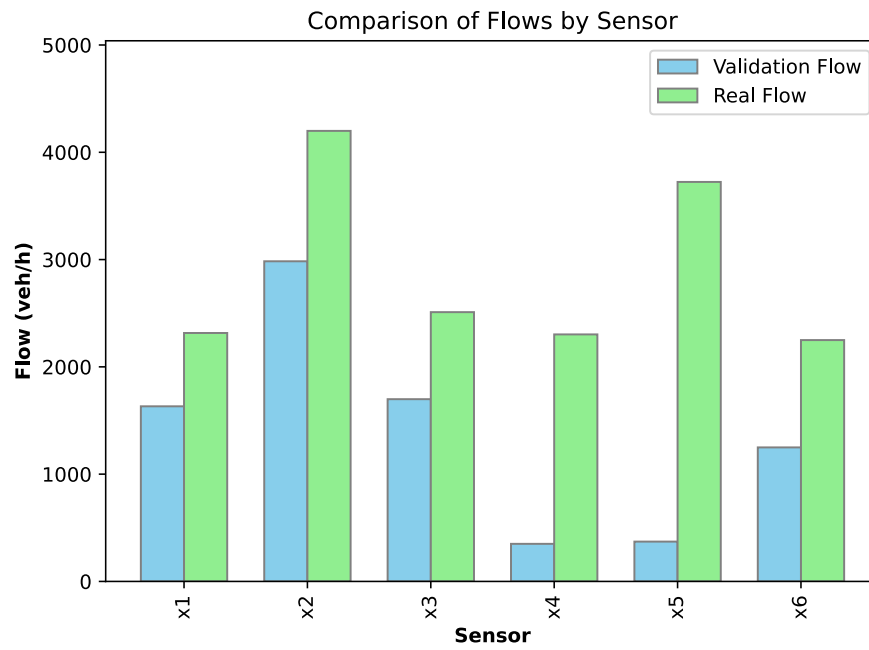


Figure A.57: Comparison of real volume with simulated volume for validation for model 2 by sensor

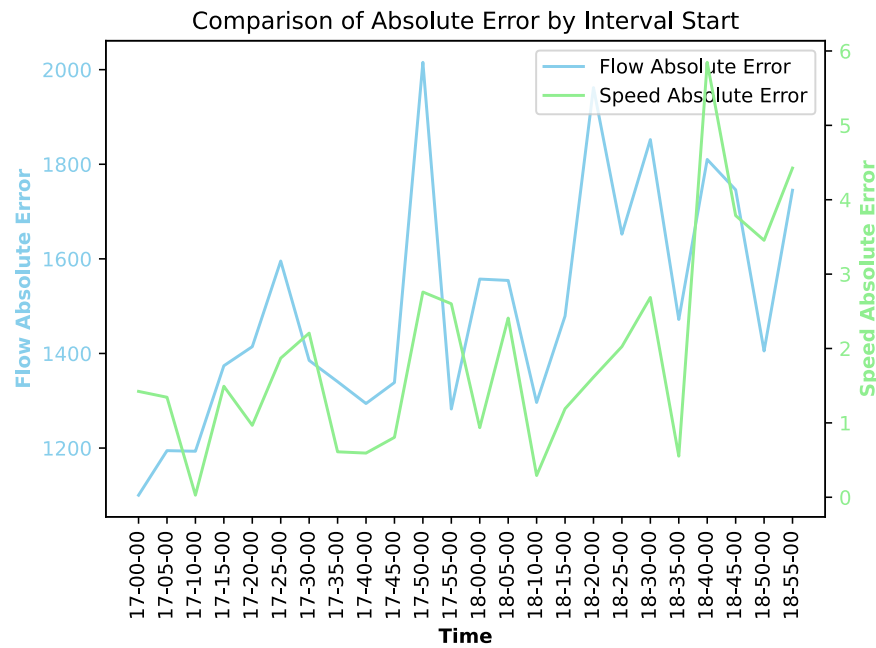


Figure A.58: Absolute error for validation for model 2 in time

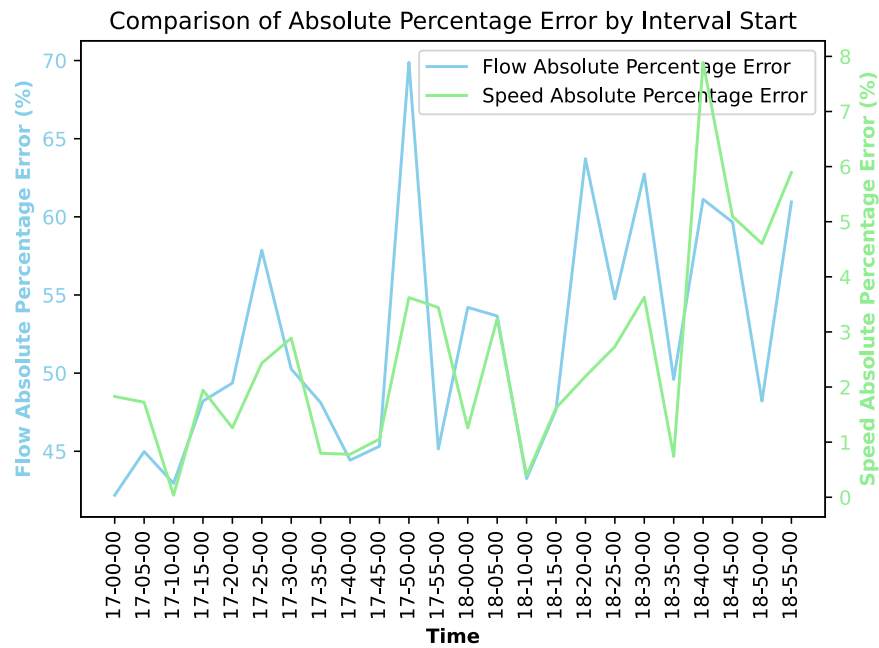


Figure A.59: Absolute percentage error for validation for model 2 in time

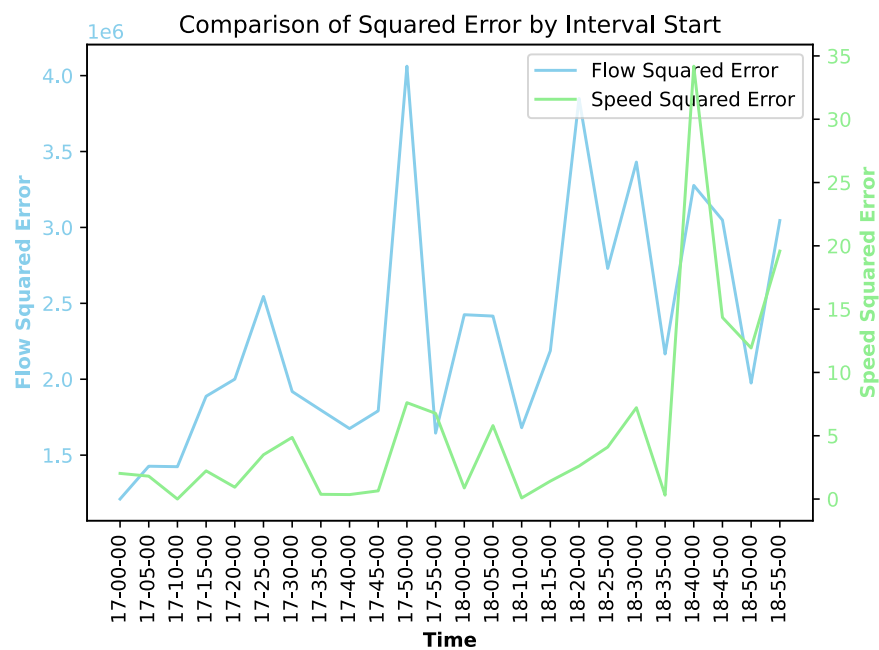


Figure A.60: Mean squared error for validation for model 2 in time

A.3.4 Validation of the simulation for model 3 for system 1

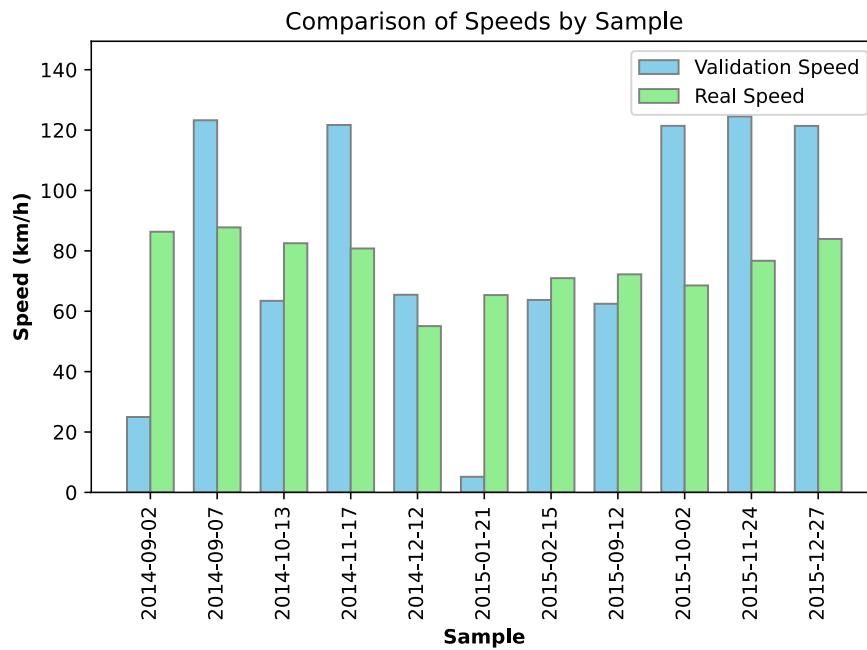


Figure A.61: Comparison of real speed with simulated speed for validation for model 3 per sample

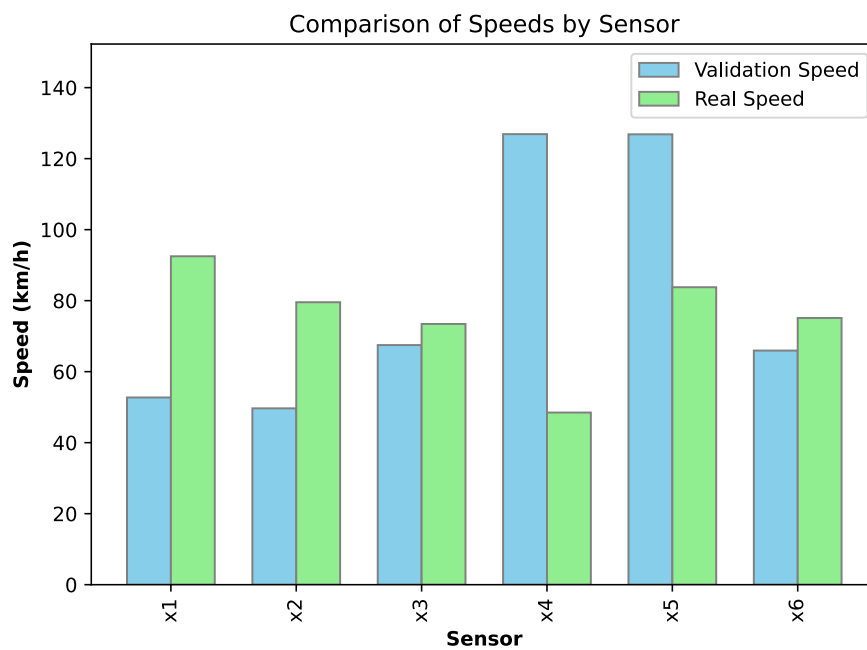


Figure A.62: Comparison of real speed with simulated speed for validation for model 3 by sensor

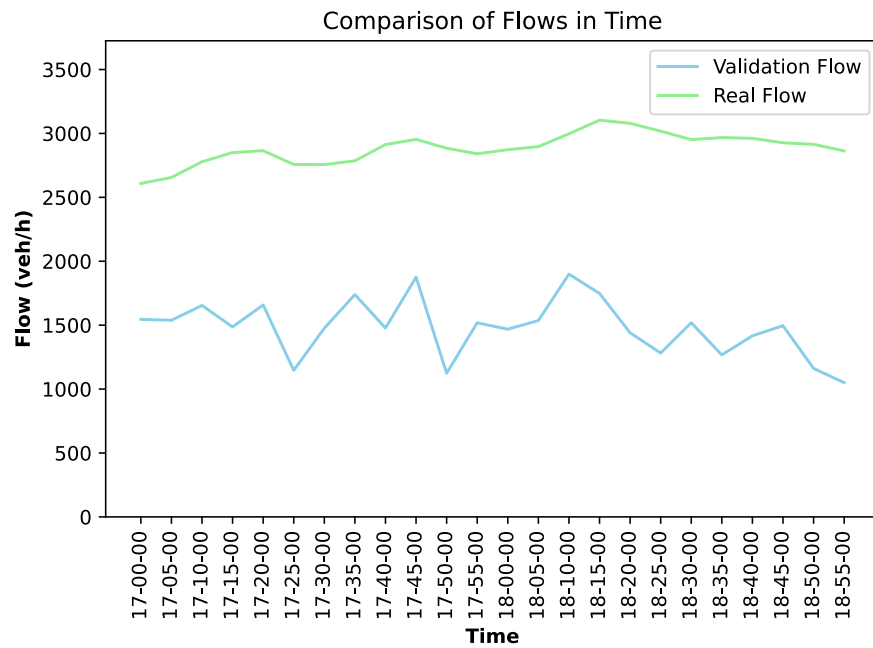


Figure A.63: Comparison of real volume with simulated volume for validation for model 3 in time

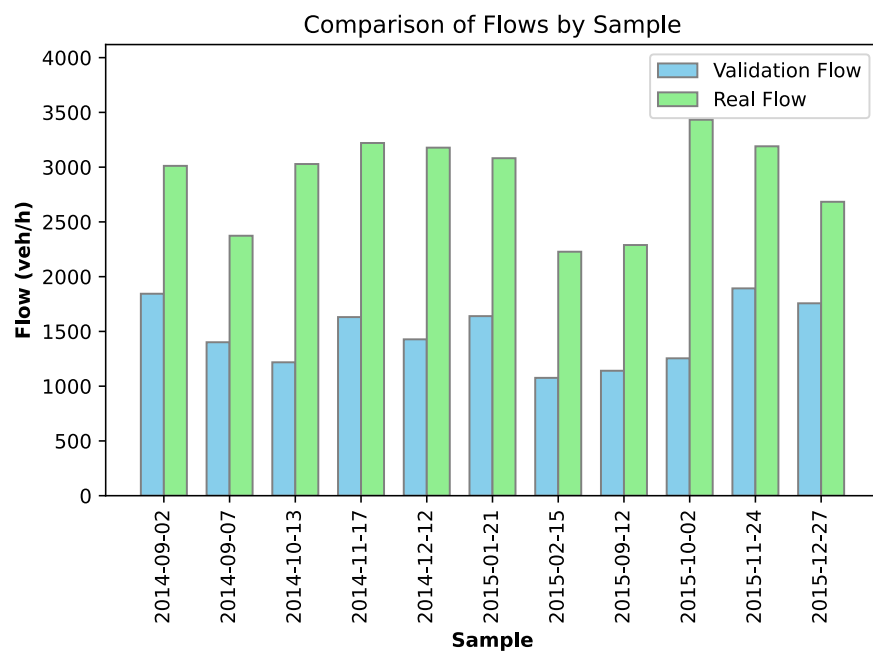


Figure A.64: Comparison of real volume with simulated volume for validation for model 3 per sample

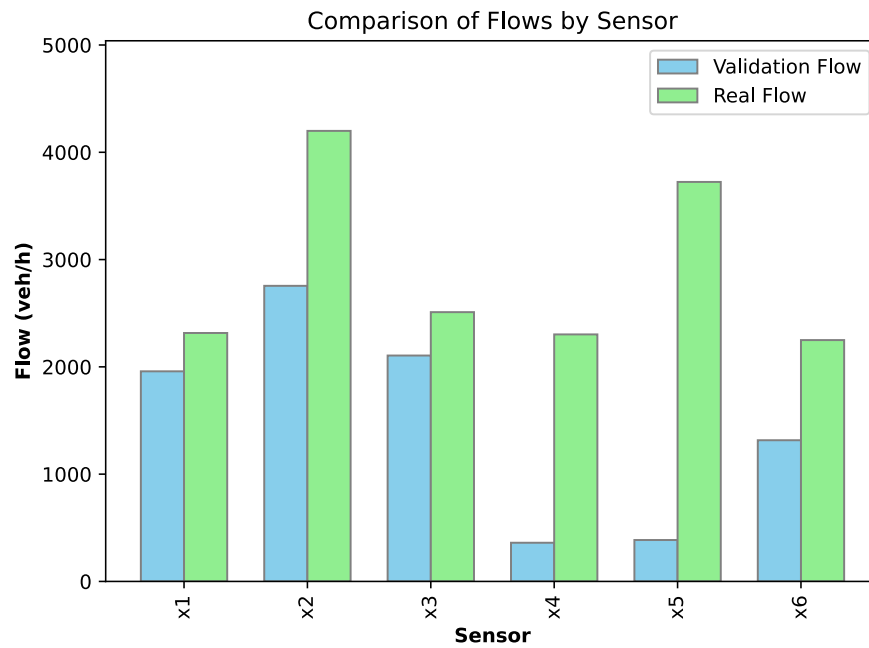


Figure A.65: Comparison of real volume with simulated volume for validation for model 3 by sensor

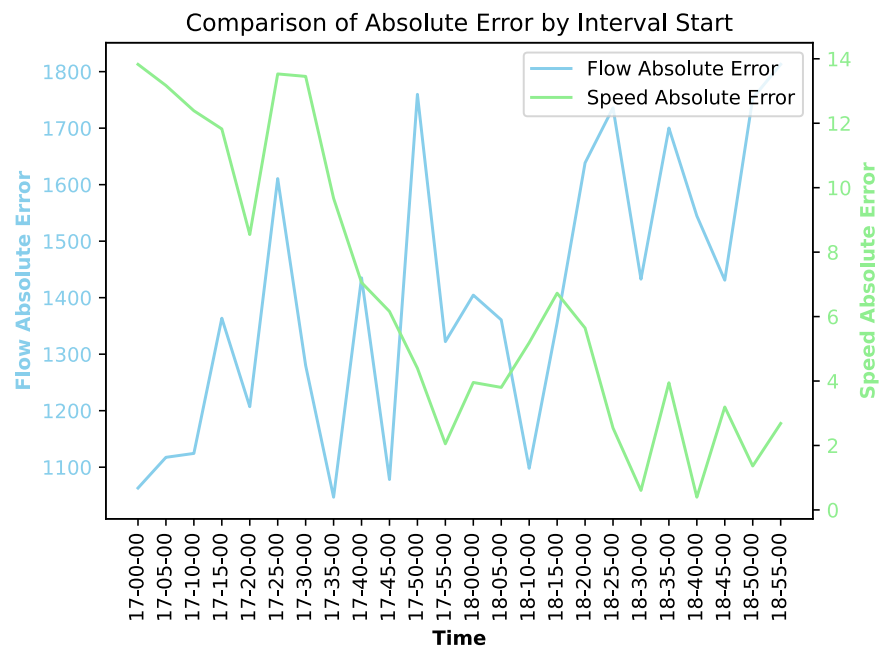


Figure A.66: Absolute error for validation for model 3 in time

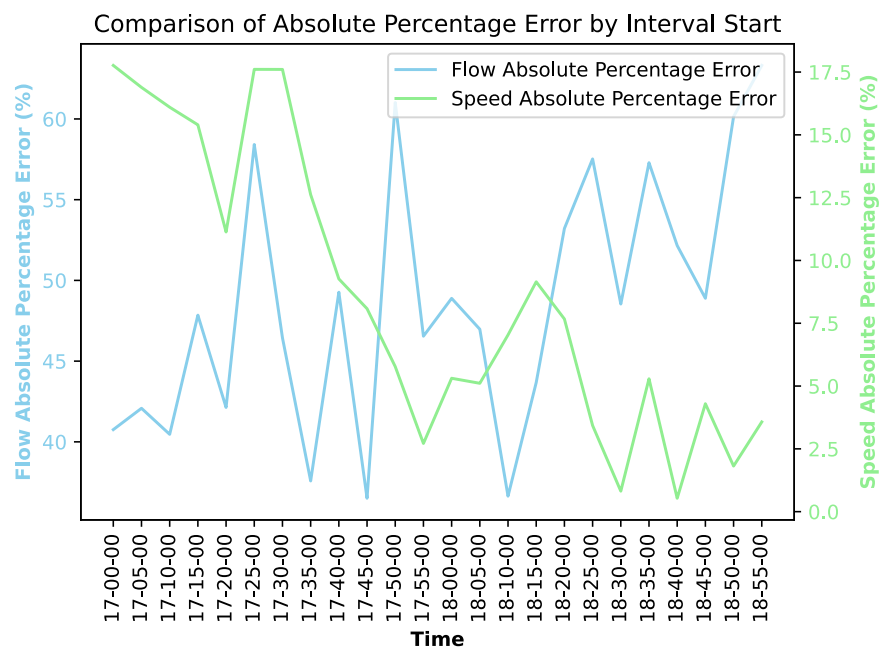


Figure A.67: Absolute percentage error for validation for model 3 in time

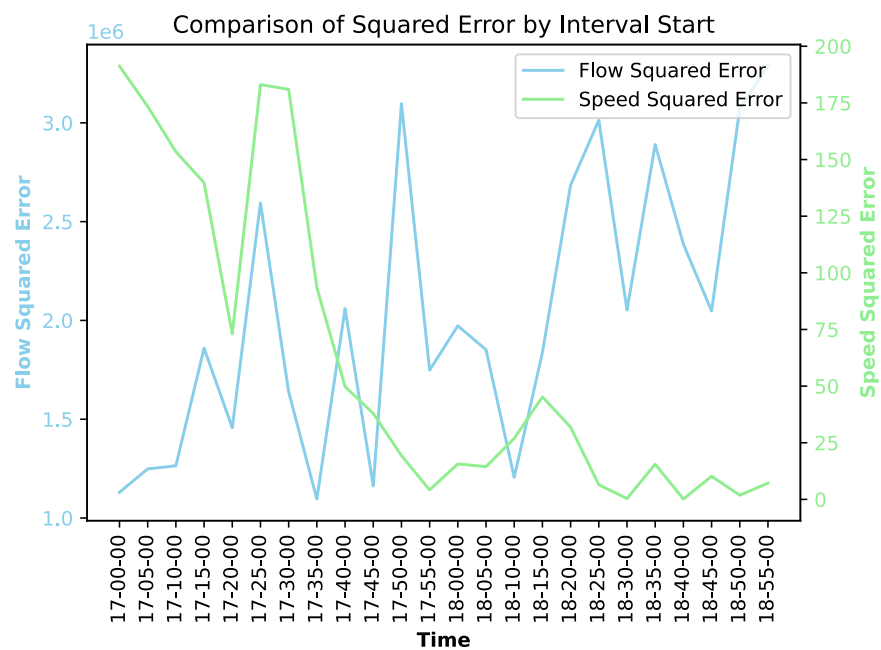


Figure A.68: Mean squared error for validation for model 3 in time

A.3.5 Validation of the simulation for model 4 for system 1

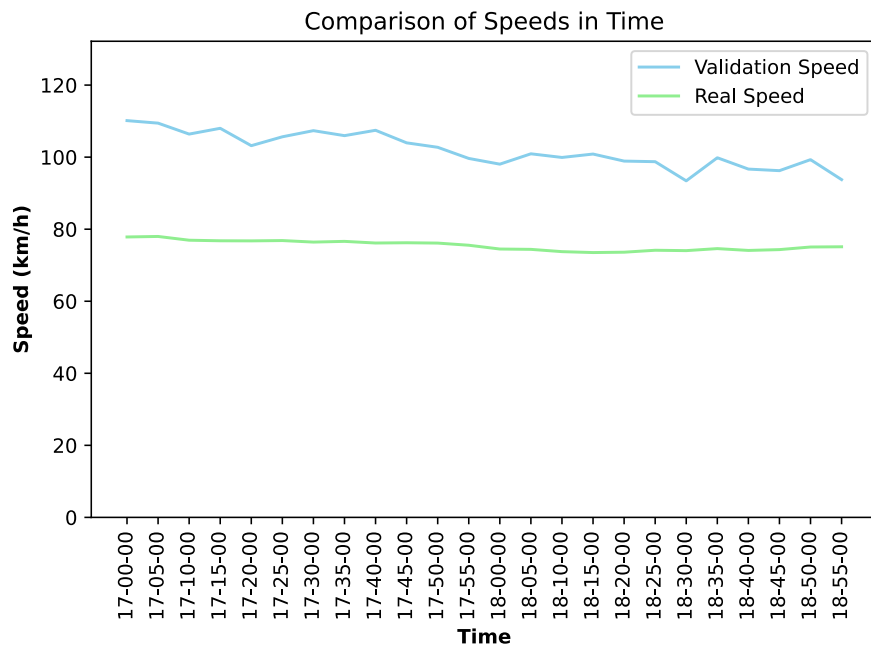


Figure A.69: Comparison of real speed with simulated speed for validation for model 4 in time

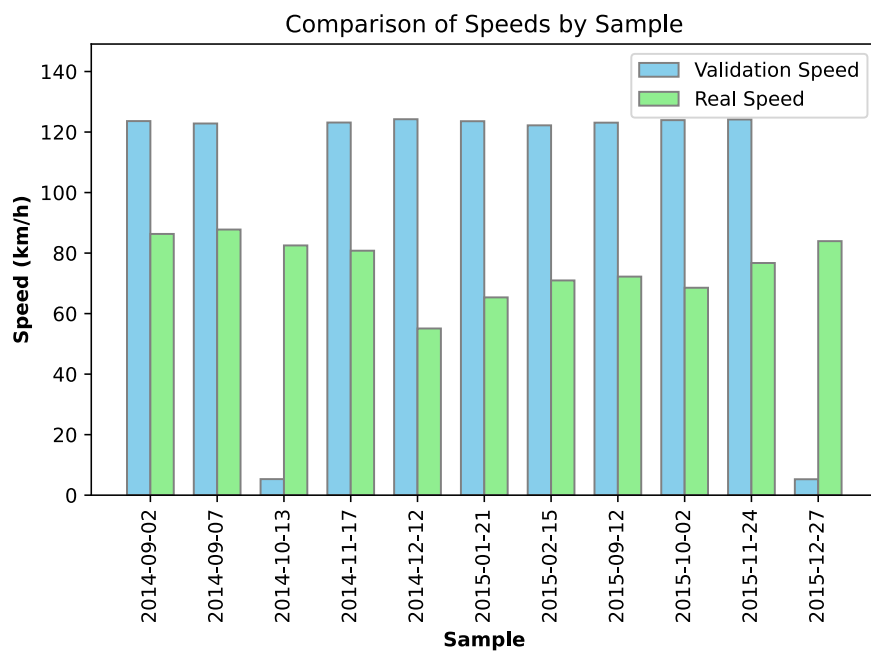


Figure A.70: Comparison of real speed with simulated speed for validation for model 4 per sample

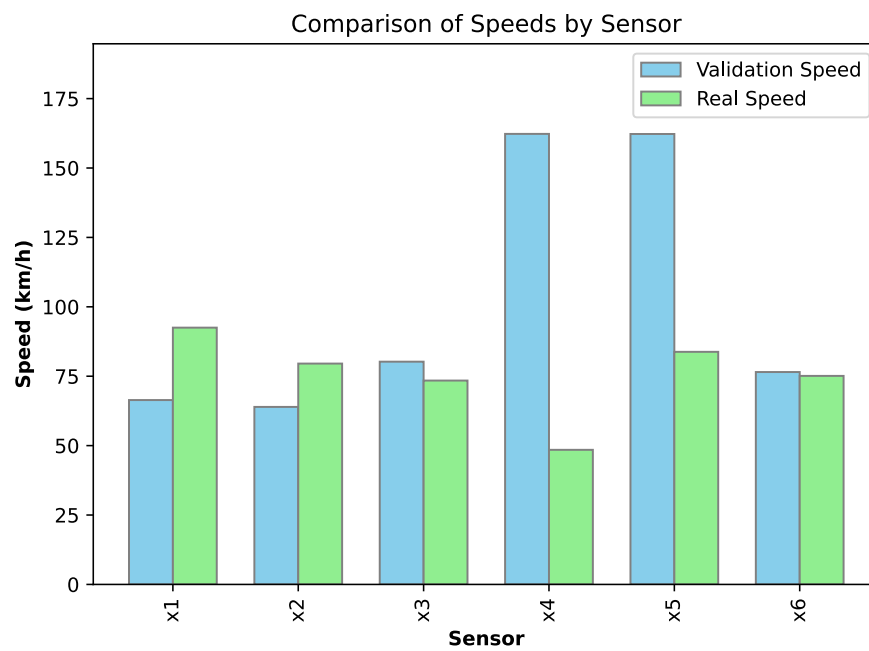


Figure A.71: Comparison of real speed with simulated speed for validation for model 4 by sensor

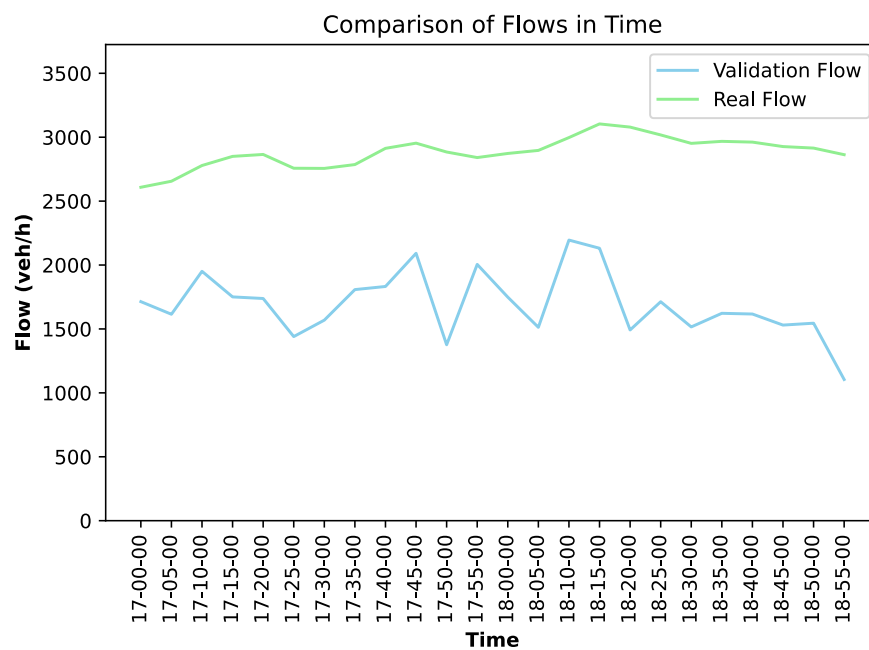


Figure A.72: Comparison of real volume with simulated volume for validation for model 4 in time

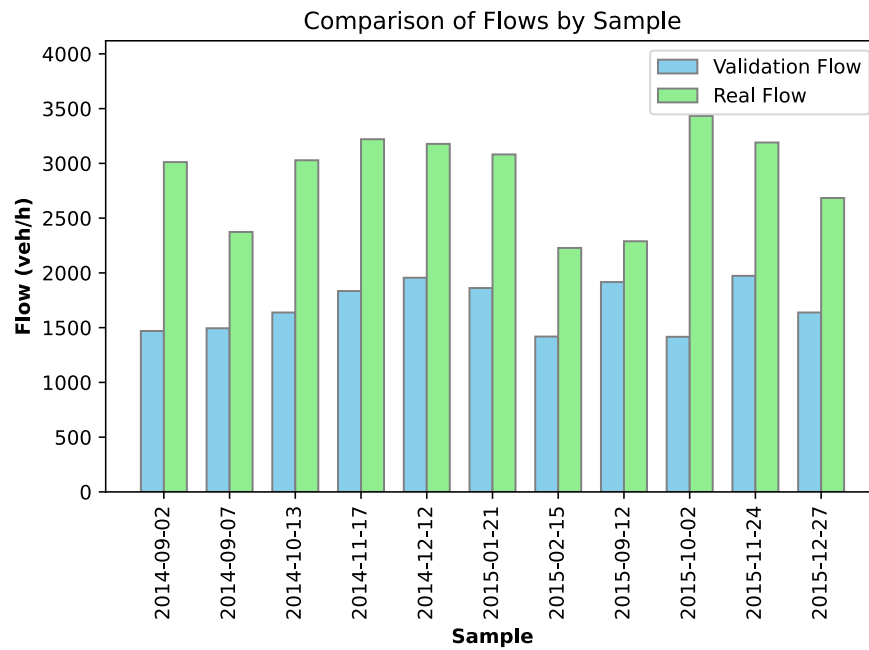


Figure A.73: Comparison of real volume with simulated volume for validation for model 4 per sample

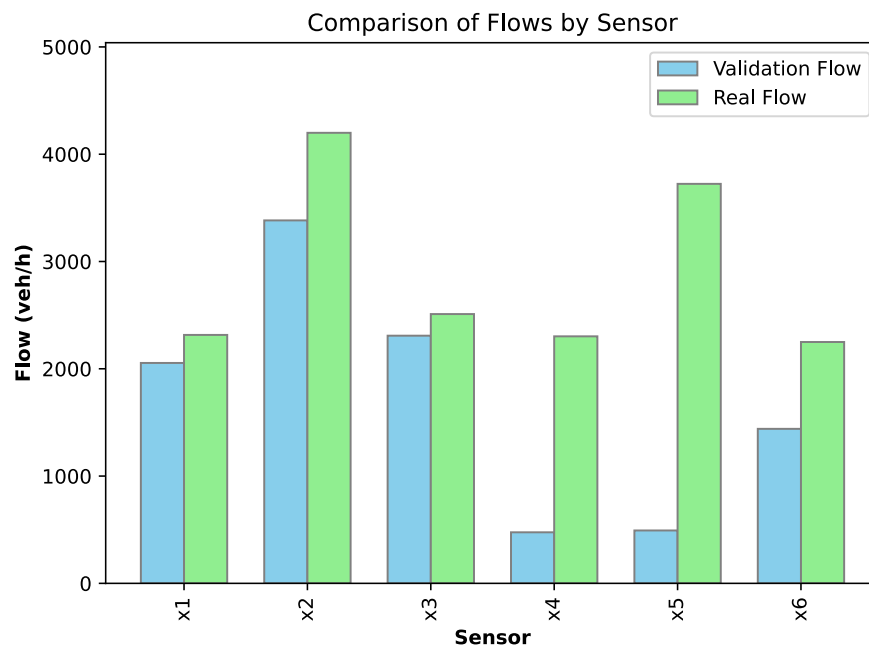


Figure A.74: Comparison of real volume with simulated volume for validation for model 4 by sensor

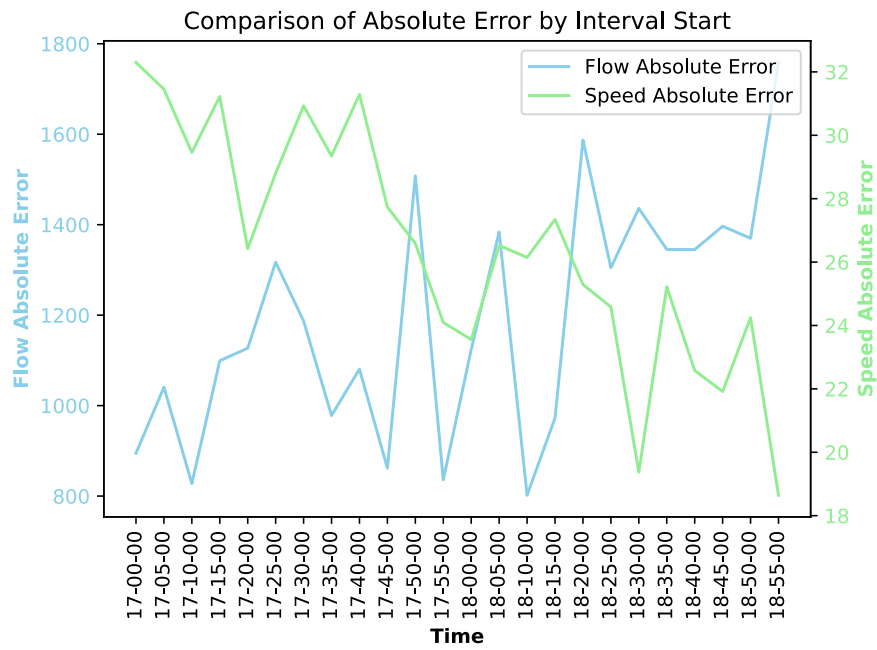


Figure A.75: Absolute error for validation for model 4 in time

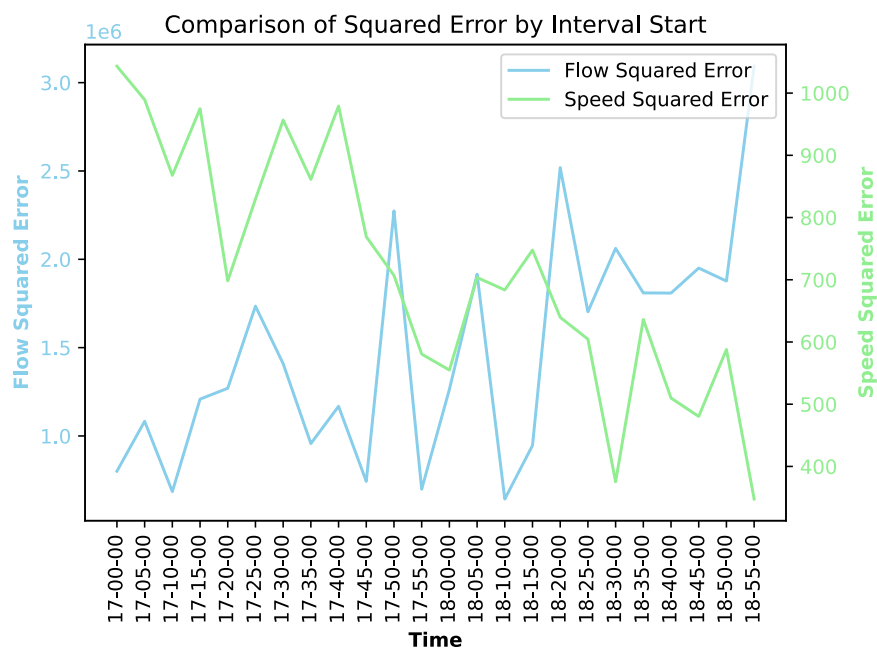


Figure A.76: Mean squared error for validation for model 4 in time

A.3.6 Validation of the simulation for model 5 for system 1

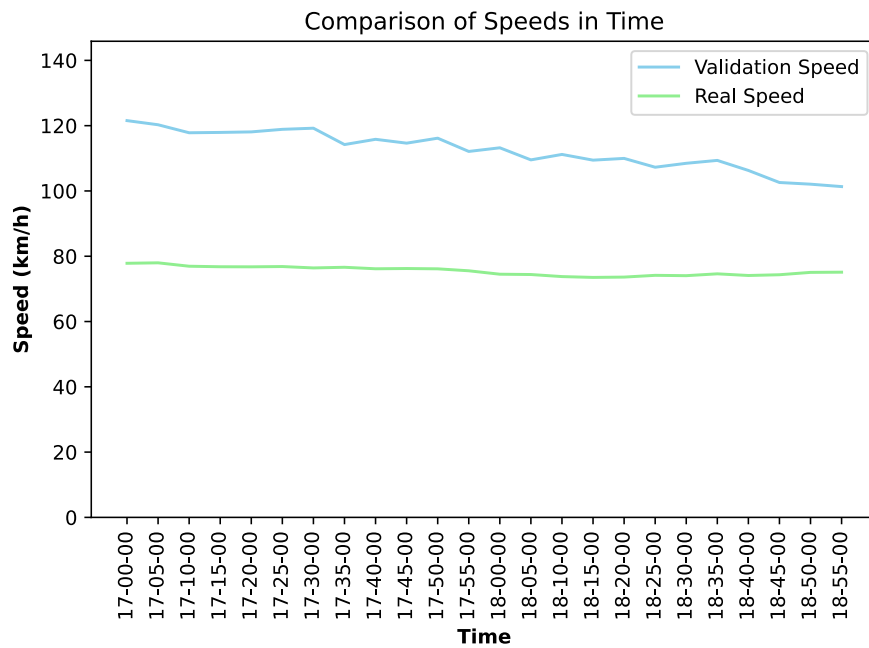


Figure A.77: Comparison of real speed with simulated speed for validation for model 5 in time

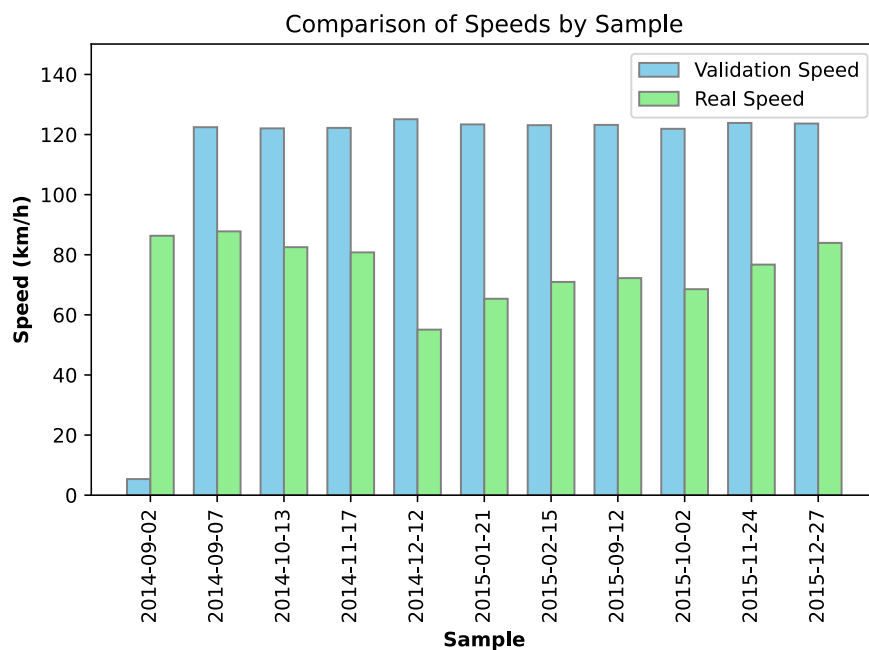


Figure A.78: Comparison of real speed with simulated speed for validation for model 5 per sample

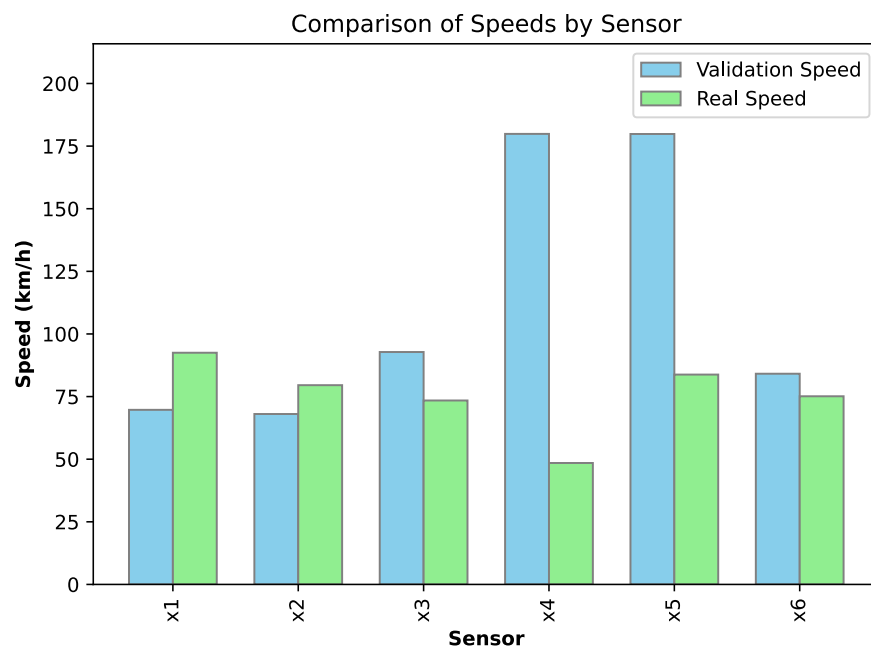


Figure A.79: Comparison of real speed with simulated speed for validation for model 5 by sensor

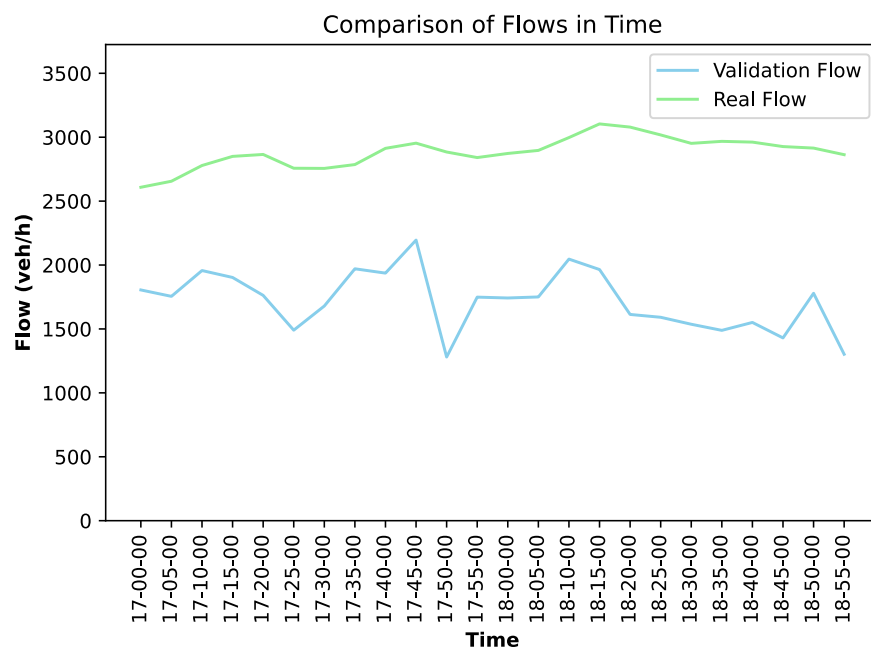


Figure A.80: Comparison of real volume with simulated volume for validation for model 5 in time

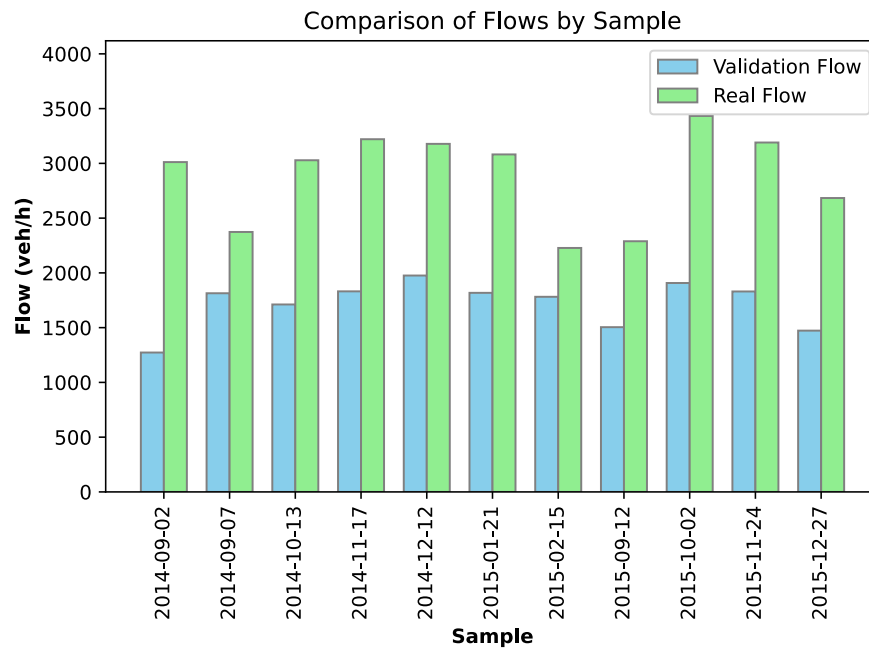


Figure A.81: Comparison of real volume with simulated volume for validation for model 5 per sample

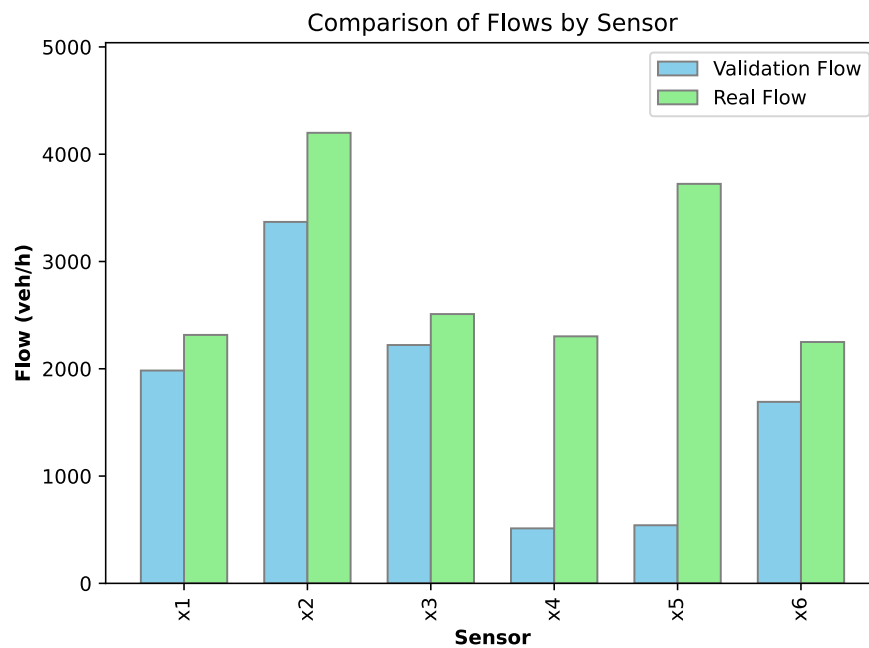


Figure A.82: Comparison of real volume with simulated volume for validation for model 5 by sensor

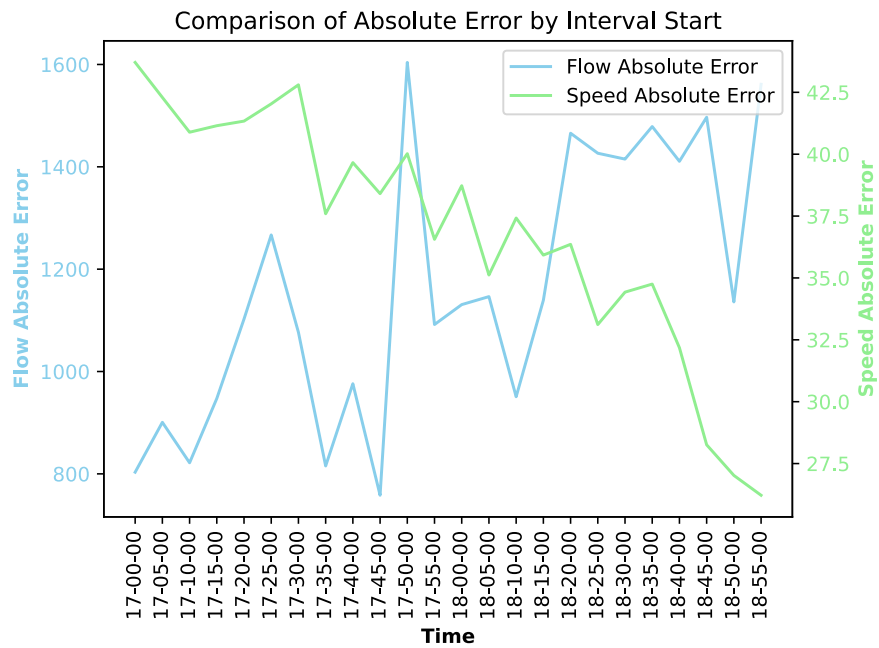


Figure A.83: Absolute error for validation for model 5 in time

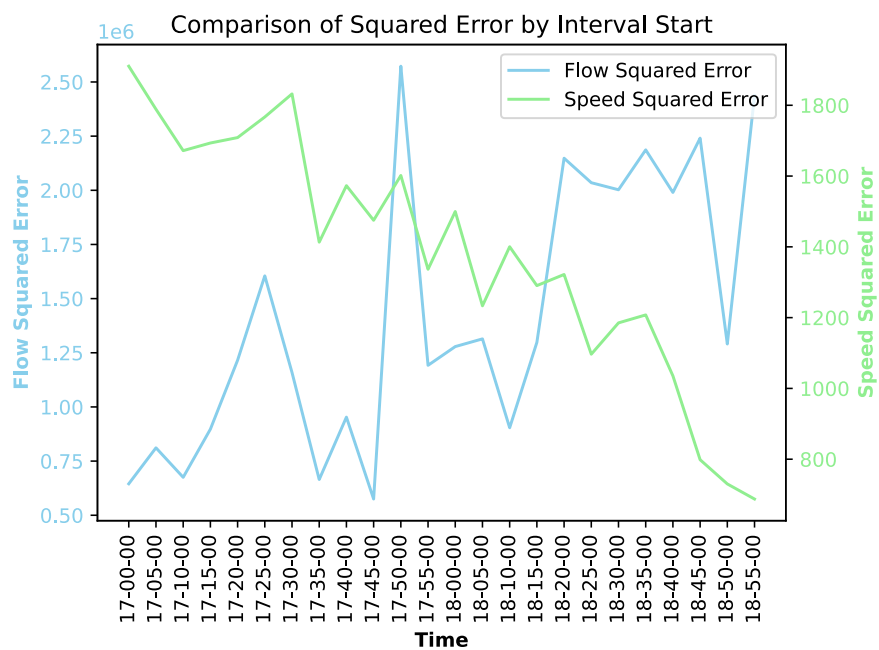


Figure A.84: Mean squared error for validation for model 5 in time

A.3.7 Validation of the simulation for model 6 for system 1

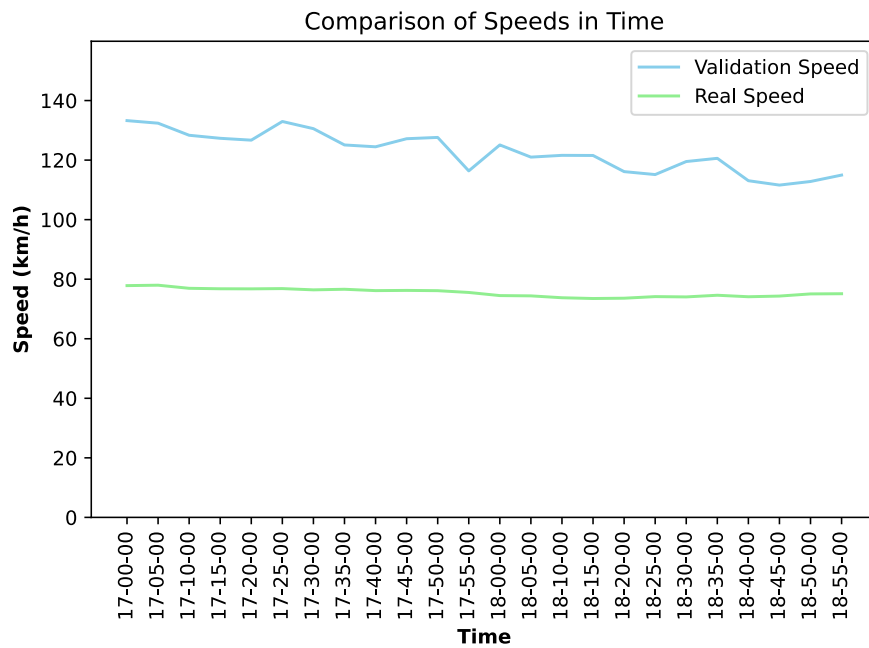


Figure A.85: Comparison of real speed with simulated speed for validation for model 6 in time

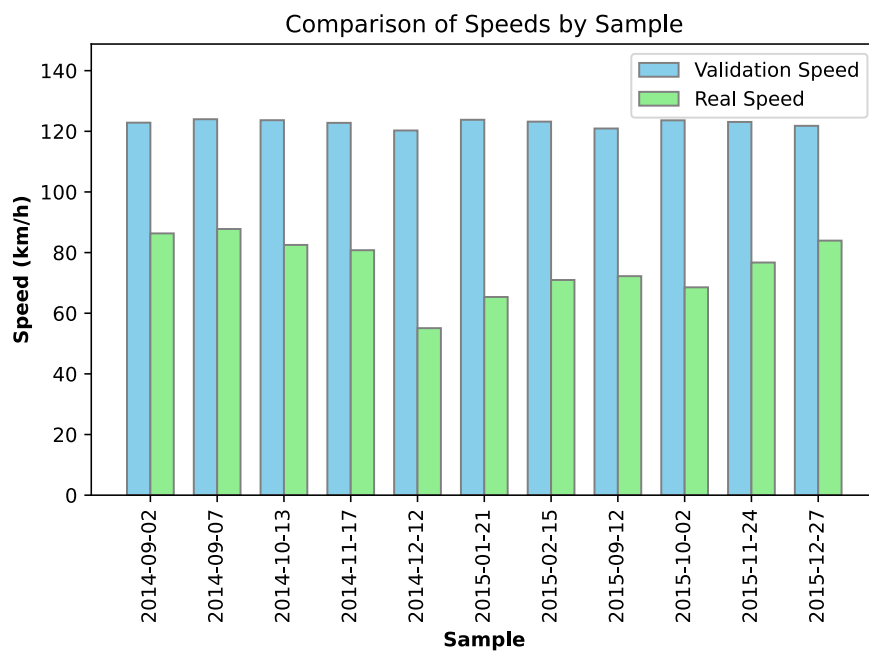


Figure A.86: Comparison of real speed with simulated speed for validation for model 6 per sample

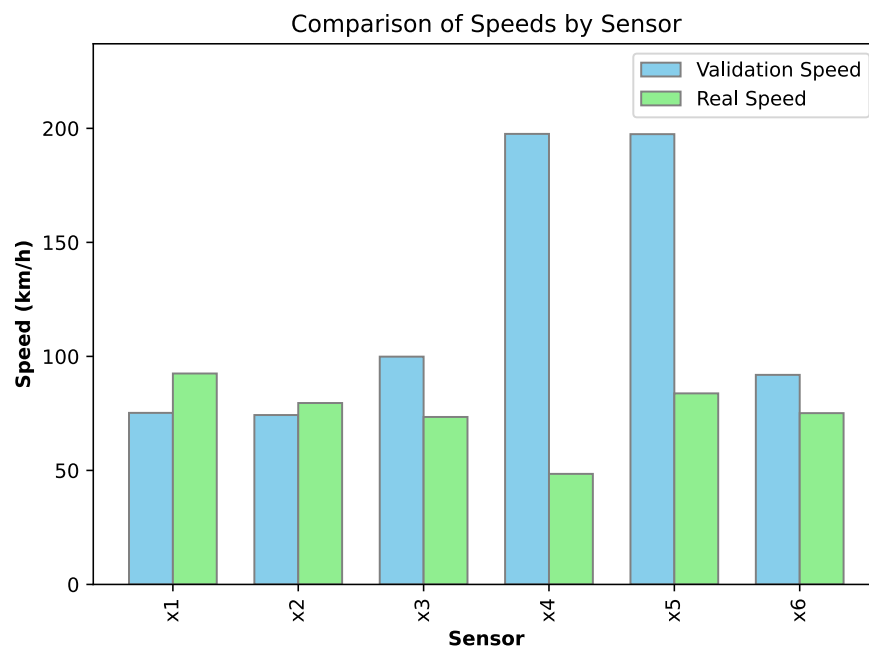


Figure A.87: Comparison of real speed with simulated speed for validation for model 6 by sensor

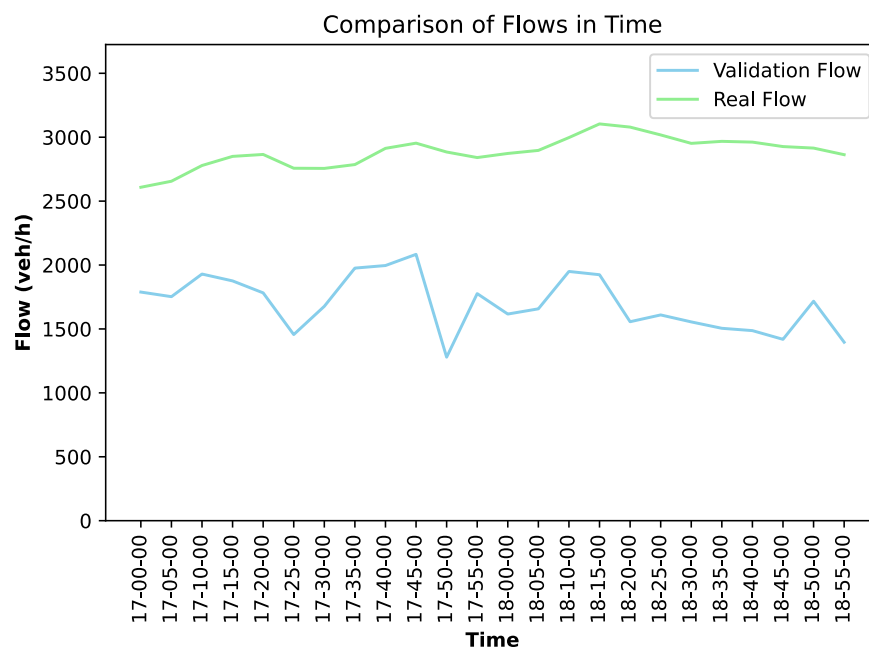


Figure A.88: Comparison of real volume with simulated volume for validation for model 6 in time

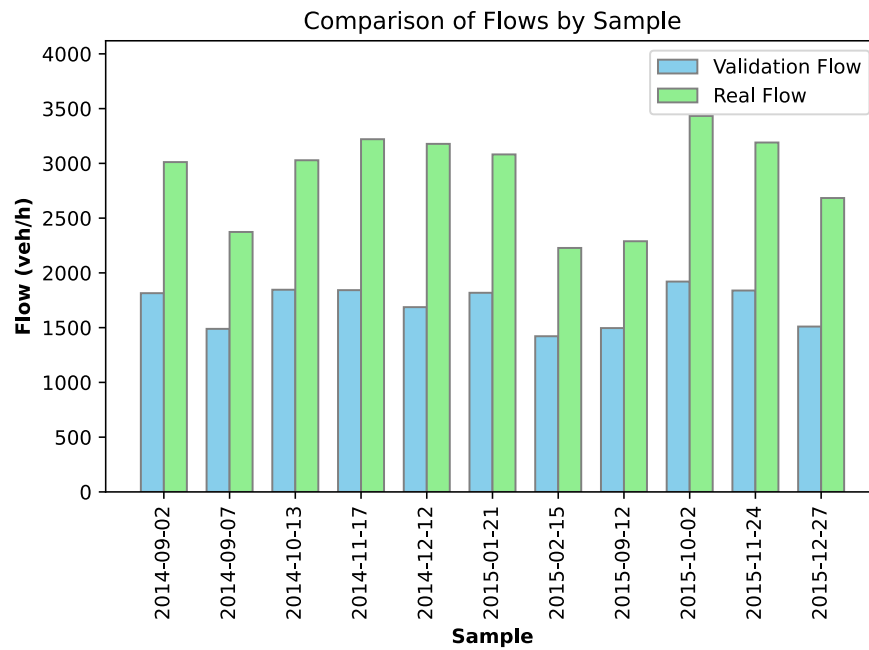


Figure A.89: Comparison of real volume with simulated volume for validation for model 6 per sample

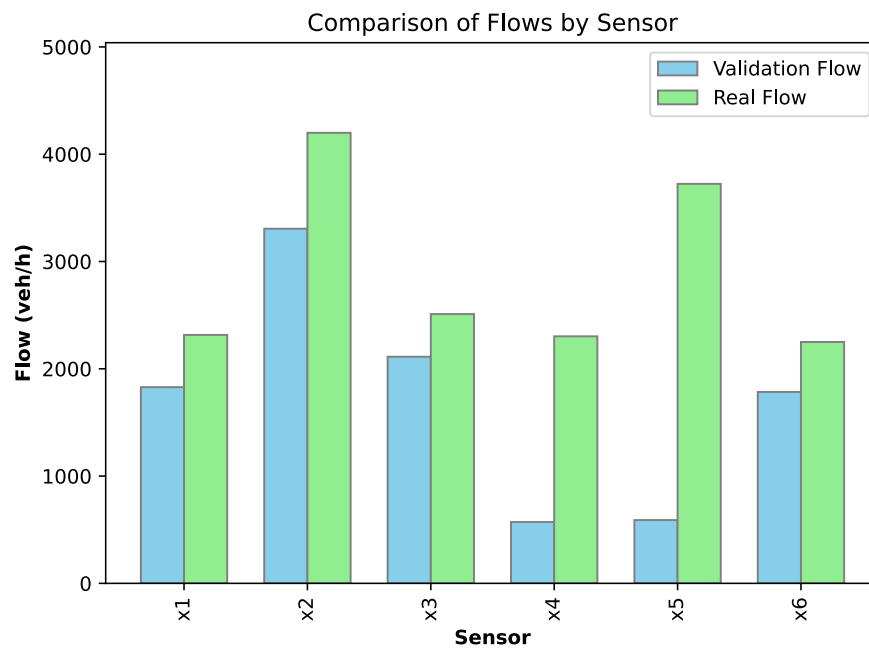


Figure A.90: Comparison of real volume with simulated volume for validation for model 6 by sensor

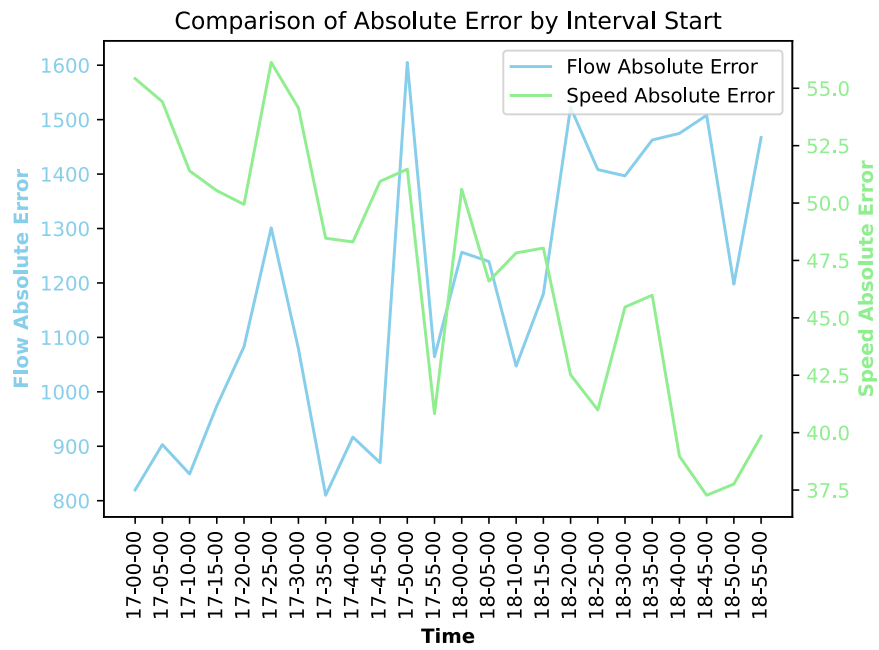


Figure A.91: Absolute error for validation for model 6 in time

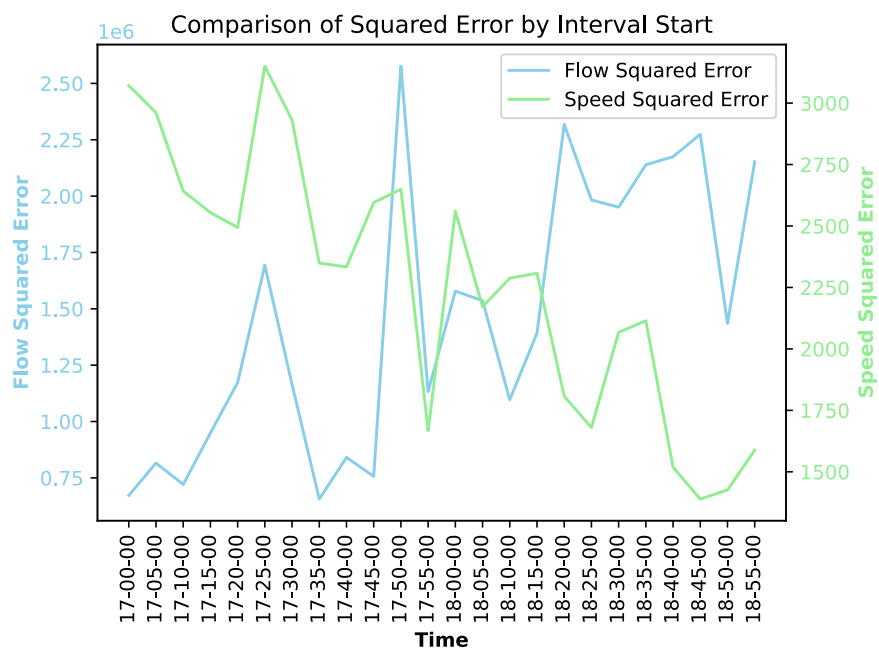


Figure A.92: Mean squared error for validation for model 6 in time

A.3.8 Validation of the simulation for model 7 for system 1

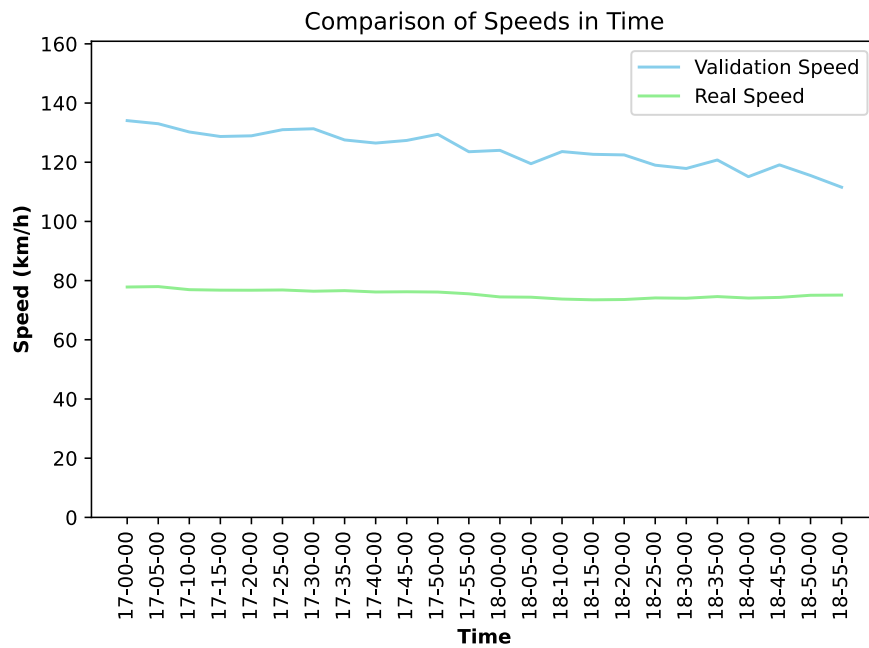


Figure A.93: Comparison of real speed with simulated speed for validation for model 7 in time

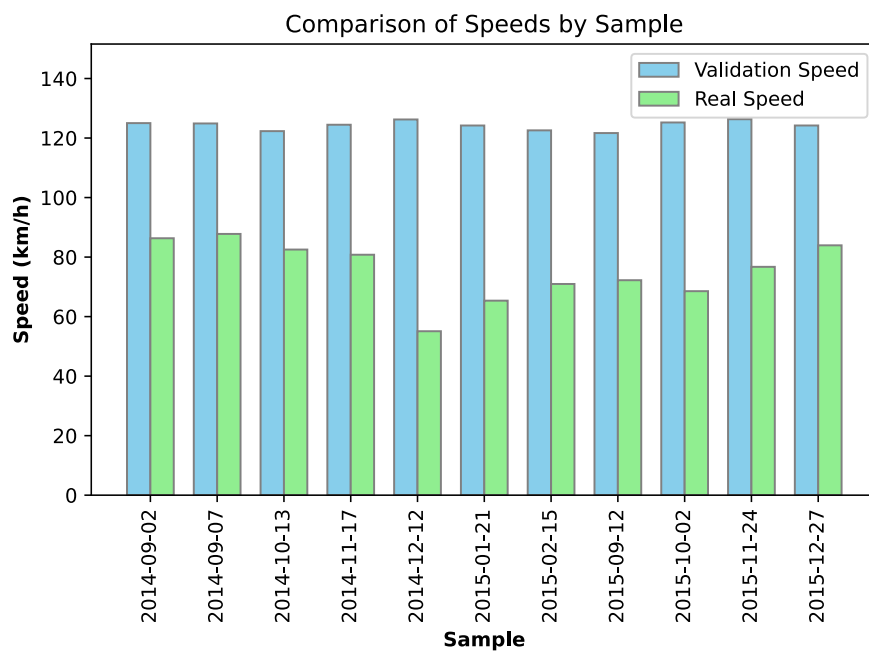


Figure A.94: Comparison of real speed with simulated speed for validation for model 7 per sample

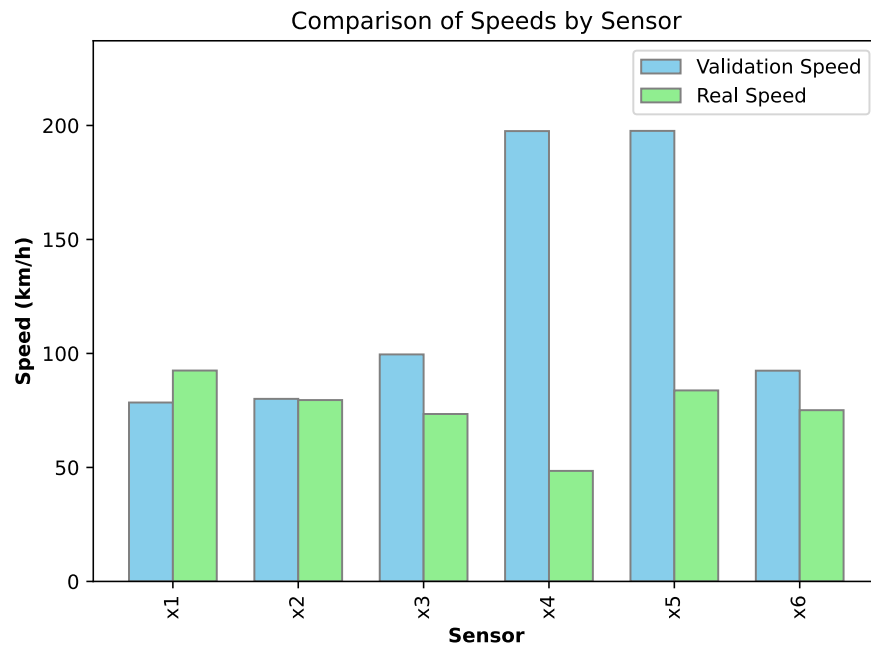


Figure A.95: Comparison of real speed with simulated speed for validation for model 7 by sensor

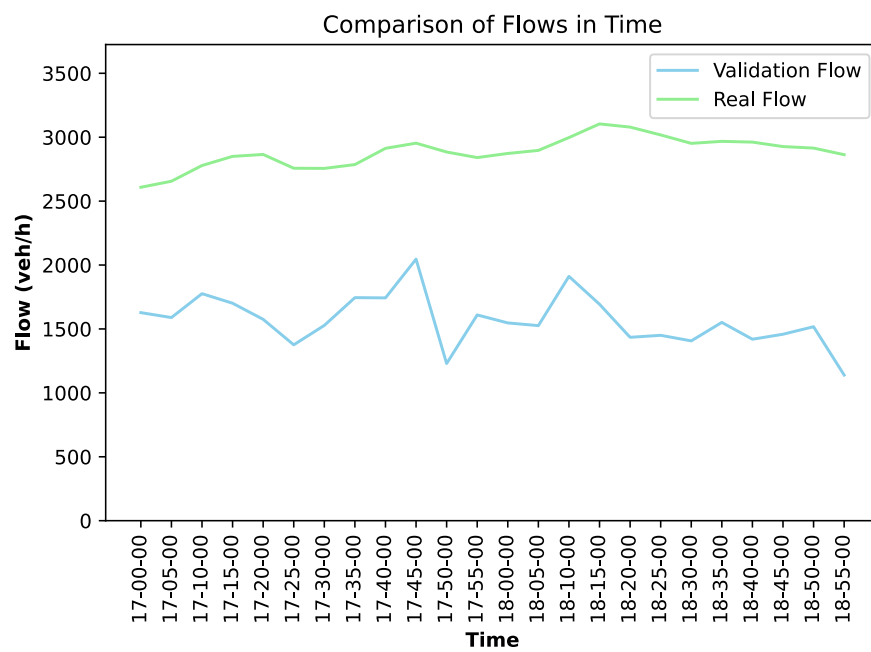


Figure A.96: Comparison of real volume with simulated volume for validation for model 7 in time

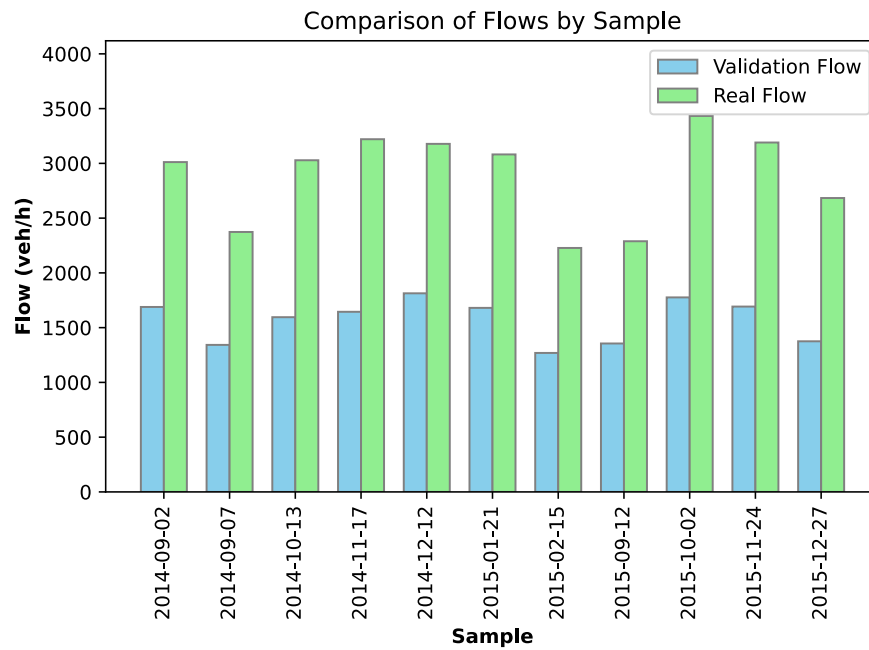


Figure A.97: Comparison of real volume with simulated volume for validation for model 7 per sample

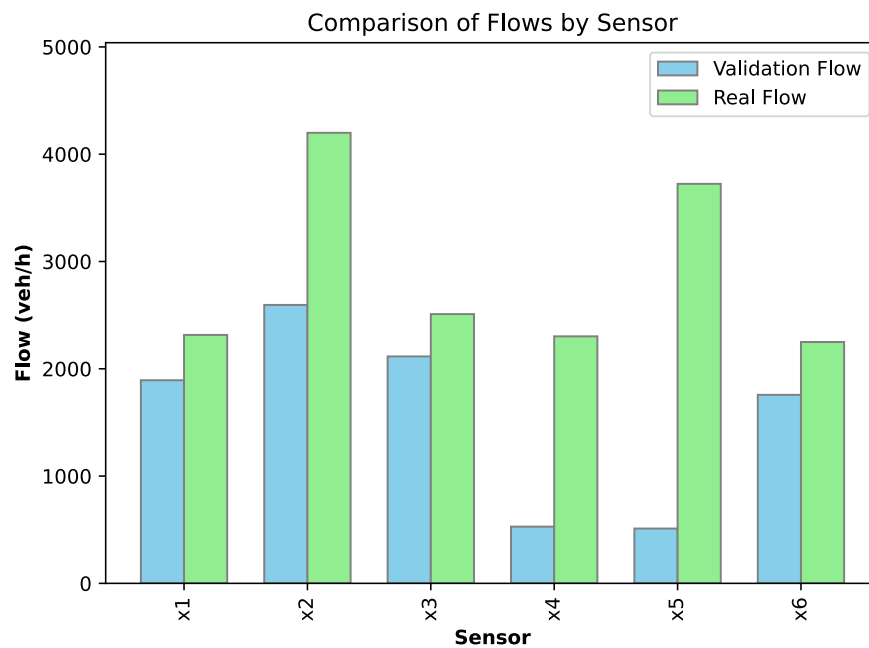


Figure A.98: Comparison of real volume with simulated volume for validation for model 7 by sensor

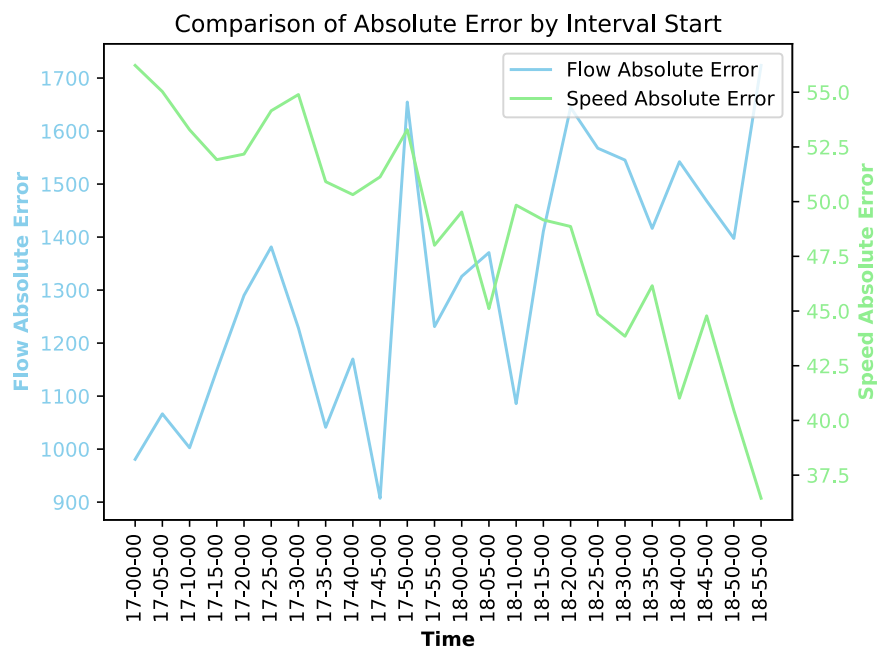


Figure A.99: Absolute error for validation for model 7 in time

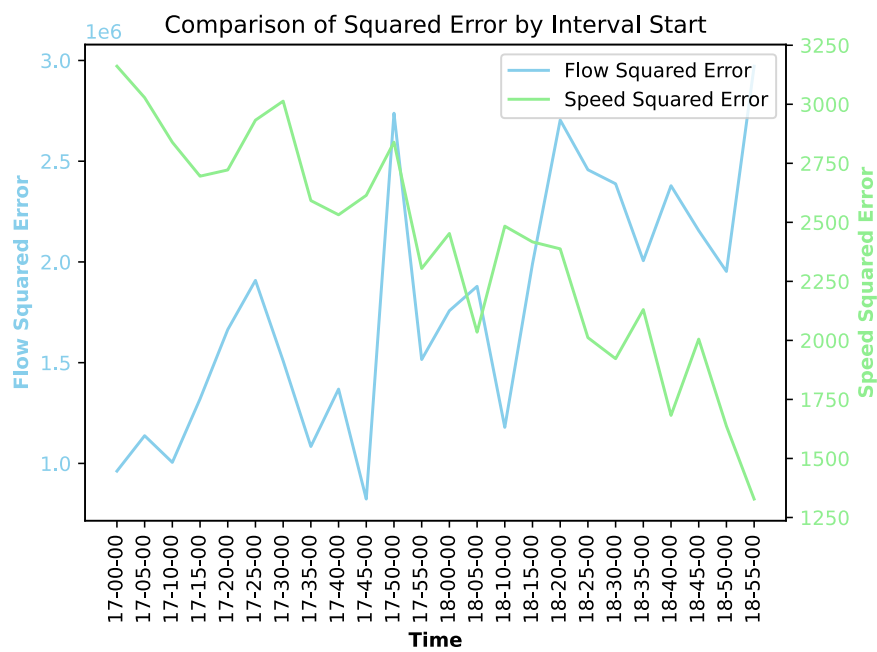


Figure A.100: Mean squared error for validation for model 7 in time