TECHNISCHE UNIVERSITÄT MÜNCHEN

MASTER'S PROGRAM IN TRANSPORTATION SYSTEMS

Master's Thesis

# Analysis and Development of Car-Following models using xFCD

**Mohd Shareef Adil**

*Supervised by:*

**Dr. Tao Ma**

**Univ.-Prof. Dr. Constantinos Antoniou**

**External - M.Sc. Michael Harth (AUDI AG)**

**March 31st, 2020**

# Disclaimer

I confirm that this master's thesis is my own work and I have documented all sources and material used..

Munich, March 31st, 2020 Mohd Shareef Adil

# Acknowledgments

I would like to express my sincere appreciations to Dr. Tao Ma and Prof. Constantinos Antoniou for accepting the thesis topic and supervising it. Their support, guidance and valuable feedback throughout the work was crucial in the successful completion of the work.

I would like to thank M.Sc Michael Harth and Dr. Wolfram Remlinger of Audi AG, for hiring me into an amazing research project - SAVe. They have constantly supported and mentored me through out the time.

Finally all this would not been possible without the support of my family, my father, my mother, my brother and sister. I would like to specially thank my wife, Ilma Khan. She always have stood by me in the most lowest points and encouraged me to keep going.

# Abstract

Car following is one of the primary driving tasks observed in traffic flow and is thus an important part of traffic flow modelling, traffic operation and control. Conventional car following models like IDM, the Krauss model and the Wiedemann model are based on mathematical equations and are derived from the traffic flow theory. Although these models have been extensively used over the years until today, they however are currently in use but have limitations to incorporate any additional information. Technological development in data collection and validation techniques motivate the researchers to use real driving data to calibrate the existing models, understand the dynamics of human drivers and develop data-driven models. One such data collection method is Extended Floating Car Data (xFCD) which allows to capture the naturalistic driving behaviour of the driver. The xFCD data can be used to develop an alternative approach of data-driven car following models which can capture the real driving behaviour of drivers. The data-driven car-following models also offer more flexibility and allow to integrate additional information to models.

In this thesis, firstly, a state machine based methodology is developed to extract the car-following data from the raw xFCD . The car-following traces extracted in this study are focused on an urban environment. The extracted car-following traces are then used to analyse the three conventional car-following models, namely the Krauss model, IDM and the Wiedemann model. The analysis is done by calibrating the parameters of the mentioned car-following models using a real-coded genetic algorithm. The performance of each of the car-following model is evaluated on the basis of their ability to simulate the observed velocity, acceleration, gap and the trajectory of the following vehicle. Then an LSTM based car-following model is proposed in this study to capture the dependency of the driver's memory in the car-following behaviour. The LSTM model is trained with multiple lengths of the input sequence (driver's memory length) and results show that the driver's memory plays an important role in car-following behaviour. The results show that the LSTM based car-following model outperforms the three conventional car-following model in simulating the observed acceleration and velocity of the following vehicle. A permutation feature importance method is applied to find the importance of the different input features in LSTM based car-following model and results show that to simulate the observed acceleration, the most important features are: acceleration, velocity, relative velocity and distance gap.

**Keywords:** car-following model, genetic algorithm, state machine, LSTM, acceleration prediction, data-driven, Krauss model, IDM, Wiedemann model, permutation feature importance.

# Contents

# 1. Introduction

This chapter begins with the thesis background followed by the research objectives, questions and contribution. In the end, the thesis framework and the report structure are summarised.

## 1.1. Background and Motivation

Car-following models are an essential part of all traffic simulation software. They play a very vital role in traffic flow modelling, traffic operation and control. These models define the interaction of drivers in the traffic stream by defining the driving behaviour of the subject vehicle with respect to the preceding vehicle ahead in the same lane (Olstam & Tapani, 2004).

Car-following models have been in research from almost half a decade. The first car-following model was proposed by Pipes in 1953 (Pipes, 1953). Since then a number of car-following models have been developed based on different logical approaches like safe distance, collision avoidance, optimal velocity, psycho-physical etc. (Olstam & Tapani, 2004). The Krauss model (Krauss, 1998) and the Wiedemann model (Wiedemann & Reiter, 1992) are two such models in which the former is dependent on the safety distance and the later is based on the physcho-physical approach. These models are the default models of widely known traffic simulator software. The Krauss model is the default car-following model of SUMO traffic simulation software and the Wiedemann model is the default car-following model of PTV VISSIM software. These conventional car-following models are based on a set of mathematical equations with some tunable parameters. This dependency on the set of mathematical equations limit these models to incorporate any new additional information.

The recent development in the data collection techniques have contributed to record very detailed car-following data. Next Generation Simulation (NGSIM) (FHWA, n.d.) is one such dataset which is predominantly used in the studies related to car-following models (Hao et al., 2018), (Zhao et al., 2018), (D. Yang et al., 2019), (Zhou et al., 2017), (Mitra & Eric, 2018). One important data that is used in traffic engineering is Extended Floating Car Data (xFCD). The NGSIM dataset and the xFCD can be classified as trajectory data and floating car data. The trajectory data contains the time series of the longitudinal motion of each vehicle in an observed spatio-temporal area (Kesting, 2007). The trajectory data are extracted from the high frequency digital images or videos recorded from an elevated observer position, e.g. high rise building (NGSIM, (FHWA, n.d.)) or drone or helicopter (HighD (Krajewski et al., 2018)). Trajectory data contains the complete information of all the vehicles on a certain road section (Kesting, 2007). On the other hand, floating car data is recorded by the vehicle that floats with

the traffic and serves as measuring station (Kesting, 2007). Floating car data usually contains the position (GPS) and the speed information of the vehicle. The xFCD is an extended form of floating car data. The xFCD contains the information from the Controller Area Network (CAN) bus of the subject vehicle which precisely records the detailed information related to the subject vehicle plus the information about the surrounding vehicles. The vehicles collecting the xFCD are usually equipped with stereo vision sensors for detecting the local traffic around the sensor equipped vehicle (Chen, 2015). (Vinagre Díaz et al., 2012) used the xFCD to propose a novel approach to the level-of-service (LOS) calculations. (Astarita et al., 2020) used the xFCD to develop an adaptive traffic light signal. (Treiber & Kesting, 2013) used the xFCD to calibrate and validate the car-following model. (Treiber & Kesting, 2013) compared the xFCD data with the NGSIM dataset and found that the NGSIM dataset contains more noise and discontinuities.

Due to the availability of the high quality datasets, an approach to data-driven car-following models is extensively explored in recent times. The data-driven models provide more flexibility compared to conventional car-following models. They also allow to integrate additional information which is not possible for the conventional car-following models. Recent researches show that the machine learning and neural networks methods are extensively explored to develop the car-following models. (Zhu, Wang, & Wang, 2018) proposed a novel car-following model using Deep Reinforcement Learning (DRL) using the naturalistic driving data. (Zhou et al., 2017) developed a recurrent neural network (RNN) based microscopic car-following model to predict the traffic oscillations using NGSIM dataset. (Khodayari et al., 2012) developed a modified neural network model using NGSIM dataset.

Most of the studies done on the analysis and development of car-following models have used the trajectory data. Nowadays, intelligent vehicles are equipped with sophisticated sensors to record the xFCD. These collected xFCD data are raw and contain the naturalistic driving behaviour. Contrary to the trajectory data, the raw xFCD data is not location specific, as they are recorded by the fleet vehicle which can drive anywhere in the city. There are very few studies done using the raw xFCD to analyse and develop a car-following model. Since the raw xFCD captures the naturalistic driving behaviour of the driver, the data in the raw xFCD is not labelled for car-following scenarios as the raw xFCD contains the information of all the surrounding vehicles and not necessarily the leading vehicle (vehicle ahead in the same lane of the subject vehicle). Therefore, a methodology to extract the car-following traces out of raw xFCD needs to be defined. Conventional car-following models capture the traffic characteristics and driver's car-following behaviour well if correctly calibrated. Therefore the performance of conventional car-following models on the extracted naturalistic car-following traces need to be evaluated. Machine learning allow to develop car-following models with a certain set of input features. These features can be different to those of the conventional car-following models, e.g. the presence of the vehicle on the left and right lane of the subject vehicle. Also, the conventional car following models describe the relationship between a following and leading vehicle as the response and stimulus. The response of the driver in the

conventional car-following model is instantaneous without considering the historical traffic information. (G. Lee, 1966) introduced a memory function in the linear car-following theory that defines a way in which the following driver processes his information. The historical vehicle pair information between perceiving stimuli and predicting time is closely related to the driver's subsequent behaviour, but this information is missing in the conventional car-following models. The relationship between the historical traffic information and the car-following behaviour can be modelled using the recurrent neural networks, as they are capable to model the sequential data. This sequential data can be the short term historical information of the driver which can be termed as the driver's memory.

## 1.2. Research Objective and Questions

Based on the discussion above, the research objective of the study is to determine how the raw xFCD data can be used for the analysis and development of a car-following model. The main objective of the study is associated with the following research questions:

- What methodology should be developed for the extraction of car-following data extraction from the raw xFCD?
- How will the conventional car-following models perform on naturalistic driving data captured by xFCD?
- Is the driver memory relevant in driver's behaviour in car-following and what other features can be used as input in the data-driven car-following model?

These research questions lead to some secondary research questions:

- Which method to use for the analysis of conventional car-following models?
- Which data-driven technique to be used to develop the car-following model?

## 1.3. Contribution

This study has attempted to make the following contributions:

- Development of a methodology to extract out the car-following data from the raw xFCD.
- Analysis of the performance of conventional car-following models by calibrating them using xFCD.
- Data-driven development of a memory based car-following model.
- Analysis of the input feature's importance for the data-driven development of a car-following model.

## 1.4. Thesis Framework and Report Outline

The thesis frame work is shown in Figure 1.1.

In line with the defined objective and research questions and following the thesis framework, this report will be organized as follows. Firstly, a preliminary study (chapter 2) is conducted regarding the car-following models and neural networks. Afterwards, a detailed literature review on methods of car-following data extraction, analysis of car-following models and development of car-following models is done in chapter 3. Then, chapter 4 presents the data collection method and an overview of the raw xFCD data used in this study. Chapter 5 then presents the methodology adopted in this study. The first part of the methodology deals with the extraction of car-following data from the raw xFCD. The second part contains the methodology adopted to analyse the three conventional car-following models, namely IDM, the Krauss model and the Wiedemann model. The last part of the methodology presents the development of a new data-driven car-following model using the xFCD data. Thereafter, chapter 6 describes the analysis and processing of the extracted car-following traces from the xFCD. Then, the results (chapter 7) of the analysis and the development of new car-following model are presented. Finally, the results of the study are summarised and its limitations and future work is discussed in chapter 8.

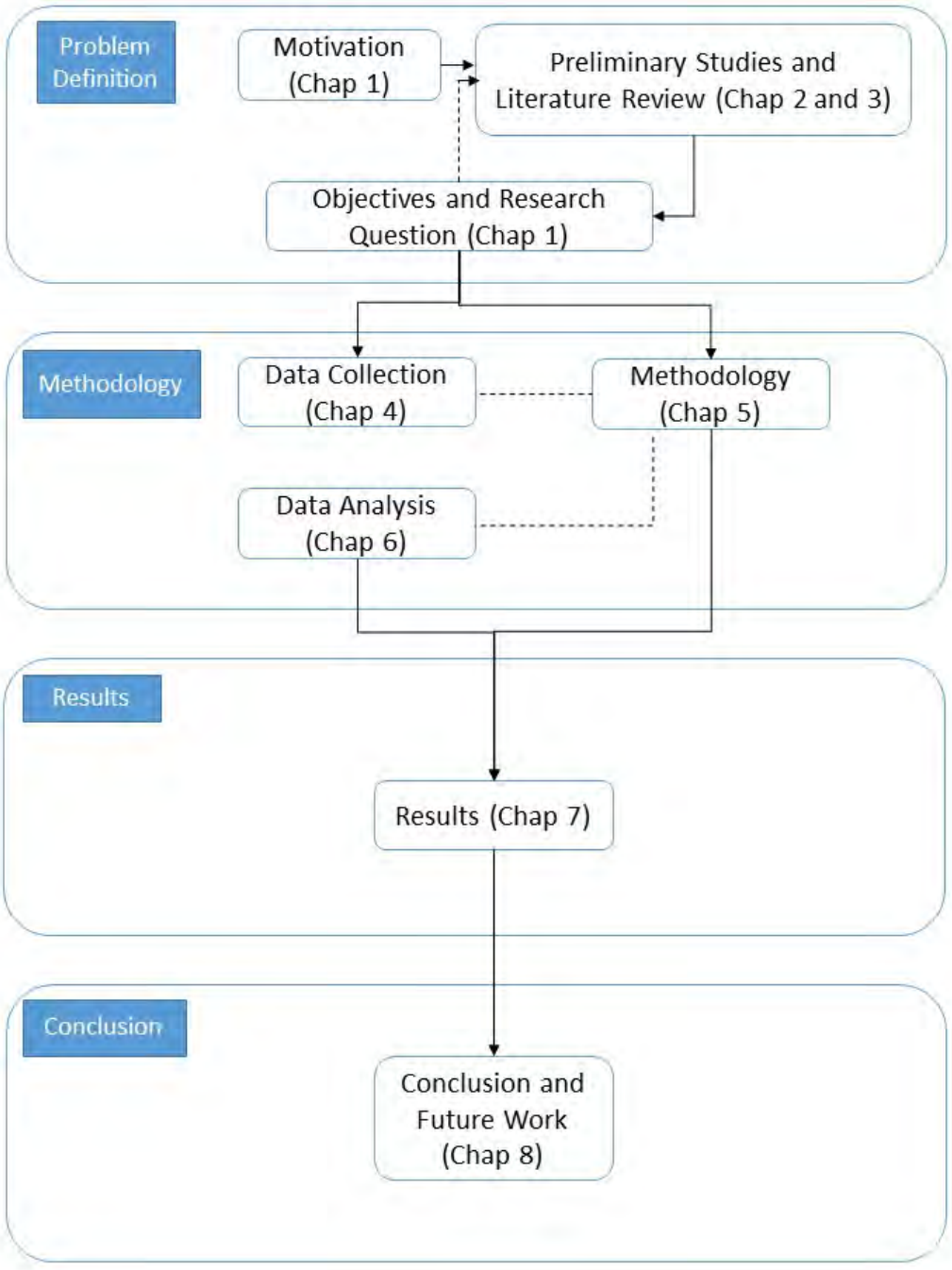



Figure 1.1.: Thesis Framework.

# 2. Preliminary Studies

This chapter provides an overview of the previous research in car following models and methods applied in this study such as genetic algorithm (GA) and neural networks (NN).

## 2.1. Car-Following Models

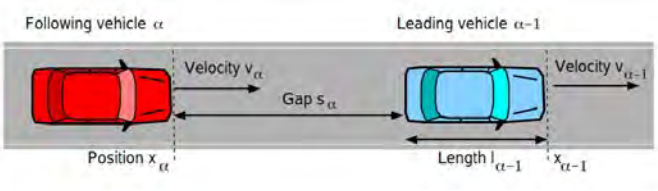



Figure 2.1.: A typical car-following situation (Kesting, 2007)

Car-following is a process which determines, how drivers follow each other in the traffic stream. Car-following theory describes the relationship between a following and the leading vehicle as a response of the following vehicle to the stimulus of leading vehicle. Thus, the following vehicle reacts to the stimulus of the leading vehicle with changes in acceleration or deceleration. A typical car-following situation is presented in Figure 2.1, where the following vehicle is shown in red color and the leading vehicle is shown in blue colour.

Car-following theory was initially proposed by Pipes in 1953 (Pipes, 1953), He considered only the relative velocity between the follower and the leader vehicle, without including the impact of the distance between them. Since then, a large number of car-following models have been developed, e.g., Gazis-Herman-Rothery (GHR) model (Gazis et al., 1961), the Optimal Velocity Model (OVM) (Bando et al., 1995), Gipps model (Gipps, 1980), Wiedemann model (Wiedemann & Reiter, 1992) etc. Technically the car-following models are classified into 5 categories:

- Stimulus based models
- Safety distance models
- Desired measure models
- Optimal velocity models
- Psycho-physical models

In stimulus based models, the acceleration of a following vehicle is determined by the driver's reaction to the speed and position differences of the leading vehicle (May, 1990). The General Motors models are well known representatives of stimulus-based models. They have been under development since 1950s, with one of their latest modifications proposed by Ozaki (Ozaki, 1993).

The safety-distance models are based on the idea that the driver of a following vehicle would adopt a speed and maintain a distance such that the driver can bring the vehicle to a safe stop if the leading vehicle applies sudden brakes. The Gipps model (Gipps, 1980) is such a model which is based on the safety-distance idea.

The desired measures models assume that the driver has a preferred situation represented by certain measures (e.g., following distance and following speed) and the driver continuously attempts to eliminate the difference between the preferred and the actual situation. The intelligent driver model (IDM) (Treiber & Kesting, 2013) is one of the most widely used model based on the desired measure theory.

The optimal velocity models assume that each following vehicle has an optimal safe velocity that depends on the relative distance between the following and the leading vehicle. The acceleration of the following vehicle can be determined according to the difference between the actual velocity and the optimal velocity (Saifuzzaman et al., 2015).

The psycho-physical models are based on the fact that the driver's behaviour varies depending on the present driving regime. These regimes can be free-flow, approaching the vehicle in front, following the vehicle in front, or braking. The boundary conditions defining the different regimes are usually expressed as a combination of relative speed and relative distance to the leading vehicle (Wiedemann & Reiter, 1992). Wiedemann model is one of the best-known psycho-physical model.

### 2.1.1. Krauss Model

The Krauss model (Krauss, 1998) was developed by Krauß in 1997. It is a microscopic, space-continuous car-following model which is based on safe speed. SUMO traffic simulation software uses this model as its default car-following model for microscopic simulations. The safe speed of krauss model is calculated as represented in Equation 2.1 (Krauss, 1998).

$$v_{\text{safe}} = v_{\text{l}}(t) + \frac{g(t) - v_{\text{l}} * t_{\text{r}}}{\frac{v_{\text{l}}(t)+v_{\text{f}}(t)}{2b} + t_{\text{r}}}\,, \tag{2.1}$$

where $v_l(t)$ represents the speed of the leading vehicle in meter per second (m/s) at time $t$, $v_f(t)$ represents the speed of the following vehicle in meter per second (m/s) at time $t$, $t_r$ is the reaction time of the driver in seconds (s), $b$ is the maximum deceleration of the vehicle in

meter per second square ($m/s^2$) and *g(t)* is the gap in meters (m) between the leading vehicle and the follower vehicle at time $t$ which is calculated as :

$$g(t) = x_l(t) - x_f(t) - l \ , \tag{2.2}$$

where $x_l(t)$ is the position of the leading vehicle at time $t$, $x_f(t)$ is the position of the following vehicle at time $t$ and $l$ is the length of the vehicle in meters (m)

The calculated $v_{safe}$ can be greater than the speed limit of road/street type or higher than the vehicle's capability. Therefore, the desired speed is calculated using Equation 2.3 which is minimum of the given three values: maximum speed of the street, the vehicle's maximum capable speed, and the $v_{safe}$ calculated by Equation 2.1.

$$v_{des}(t) = min[v_{max}, v(t) + a * \Delta t, v_{safe}(t)] \ , \tag{2.3}$$

The final velocity $(v(t + \Delta t))$ and the position $(x_f(t + \Delta t))$ at the next simulation step $(t + \Delta t)$ are updated using Equation 2.4 and Equation 2.5 respectively, where $\eta$ represents the random perturbation to allow deviation from the optimal driving and $\Delta t$ represents the step length of the simulation.

$$v(t + \Delta t) = max[0, v_{des}(t) - \eta] \ , \tag{2.4}$$

$$x_f(t + \Delta t) = x_f(t) + v(t + \Delta t) * \Delta t \tag{2.5}$$

### 2.1.2. Intelligent Driver Model

The Intelligent Driver Model (IDM) (Treiber & Kesting, 2013) is a time-continuous accident free model which produces realistic acceleration profiles and plausible behaviour in all single lane traffic situations. IDM can be defined mathematically as :

$$\dot{v} = a \left[ 1 - \left( \frac{v(t)}{v_0(t)} \right)^{\delta} - \left( \frac{s^*(v(t), \Delta v(t))}{s(t)} \right)^2 \right] \ , \tag{2.6}$$

where $\dot{v}$ represents the calculated acceleration of the following vehicle in meter per second squared ($m/s^2$) , $a$ represents the maximum acceleration of the following vehicle in meter per second squared ($m/s^2$), $v(t)$ represents the velocity of the following vehicle in meter per second (m/s) at time $t$, $\Delta v(t)$ represents the velocity difference of leading vehicle and the following vehicle at time $t$, $\delta$ is the acceleration exponent, $s(t)$ represents the distance in meters (m) between the leading and the following vehicle at time $t$ and $s^*(v(t), \Delta v(t))$ represents the desired distance at time $t$ which is calculated as:

$$s^*(v(t), \Delta v(t)) = s_0 + max \left( 0, v(t)T, \frac{v(t)\Delta v(t)}{2\sqrt{ab}} \right) \ , \tag{2.7}$$

where $T$ represents the time headway in seconds (s) and $b$ represents the maximum deceleration of the following vehicle in meter per second squared ($m/s^2$).

Equation 2.6 explains that the acceleration is divided into two parts. The first part defines the desired acceleration on a free road which compares the current speed $v$ to the desired speed $v_0$. The acceleration on a free road decreases from the initial acceleration $a$ to zero when approaching the desired speed $v_0$. The second part defines the braking deceleration induced by the front vehicle which compares the current distance $s$ to the desired distance $s*$. In case the actual gap $s(t)$ is approximately equal to $s^*$, then the braking deceleration essentially compensates the free acceleration part, so that the resulting acceleration is nearly zero (Treiber & Kesting, 2013). In addition, analysing Equation 2.7 gives an interpretation that $s^*$ corresponds to the gap when following a vehicle in steadily flowing traffic. $s^*$ increases dynamically when approaching a slower vehicle and decreases when the front vehicle is faster. IDM has in total 6 parameters which are listed below.

- the desired speed when driving on free road, $v_0$
- the desired safety time headway when following other vehicles, $T$
- the maximum acceleration of the vehicle , $a$
- comfortable braking deceleration, b
- minimum bumper to bumper distance to the leading vehicle, $s_0$
- acceleration exponent, $\delta$

To realise the simulation with $\Delta t$ as the simulation time step, Equation 2.8 is used to calculate the speed of the following vehicle at time step (t + $\Delta$t). Where, $v(t + \Delta t)$ is velocity of the vehicle in meters per second (m/s) at time step $(t + \Delta t)$, $v(t)$ is velocity of the vehicle in meters per second (m/s) at the previous simulation step, $\dot{v}$ is the calculated acceleration using Equation 2.6.

$$v(t + \Delta t) = v(t) + \dot{v} * \Delta t \ , \tag{2.8}$$

Equation 2.9 is used to calculate the new position of the following vehicle, where $x_f(t + \Delta t)$ is the position of the following vehicle at time step $(t + \Delta t)$, $x_f(t)$ is the position of the following vehicle at previous time step, $\dot{v}$ is the calculated acceleration using Equation 2.6.

$$x_f(t + \Delta t) = x_f(t) + v(t) * \Delta t + 0.5\dot{v} * \Delta t^2 \ , \tag{2.9}$$

Equation 2.10 is used to calculate the new gap between the leading and the following vehicle at time step $(t + \Delta t)$, where $s(t + \Delta t)$ is the new position of the following vehicle at time step $(t + \Delta t)$, `x1(t + Δt)` is the position of the leading vehicle at time step $(t + \Delta\ t)$, $x_f(t$

+ $\Delta t$) is the position of the following vehicle vehicle at time step ($t$ + $\Delta t$) calculated using Equation 2.9 and $L_l$ is the length of the leader vehicle in meter (m).

$$s(t + \Delta t) = x_l(t + \Delta t) - x_f(t + \Delta t) - L_l. \tag{2.10}$$

### 2.1.3. Wiedemann Model

The Wiedemann (Wiedemann & Reiter, 1992) car-following model is a typical psycho-physiological model which was originally formulated in 1974 by Rainer Wiedemann. It is the building block of the widely used microscopic traffic simulation tool, PTV VISSIM. The Wiedemann model describes the psycho-physiological aspects of the driving behaviour in terms of four discrete driving regimes: (i) free flow, (ii) approaching slower vehicles, (iii) car-following near the steady-state equilibrium and (iv) critical situation requiring stronger braking actions (Wiedemann & Reiter, 1992). Figure 2.2 shows a graphical representation of the Wiedemann car-following model. The different thresholds are shown with a certain shape that can only be amplified during the calibration procedure. Figure 2.2 is explained by analysing the trajectory of the following vehicle shown in black colour. Initially, the following vehicle approaches the leading vehicle (The gap between the leading and the following vehicle ($\Delta$x) decreasing due to higher speed of the following vehicle shown by a positive relative velocity $\Delta$v), then it enters the perception area after crossing the SDV threshold, where it has to reduce the speed. The follower vehicle then crosses another threshold (CLDV) where it reacts and reduces speed even further to enter an unconscious reaction car-following regime. The follower vehicle then continues the unconscious car-following regime as long as it remains bounded by the thresholds OPDV, SDX, and SDV.

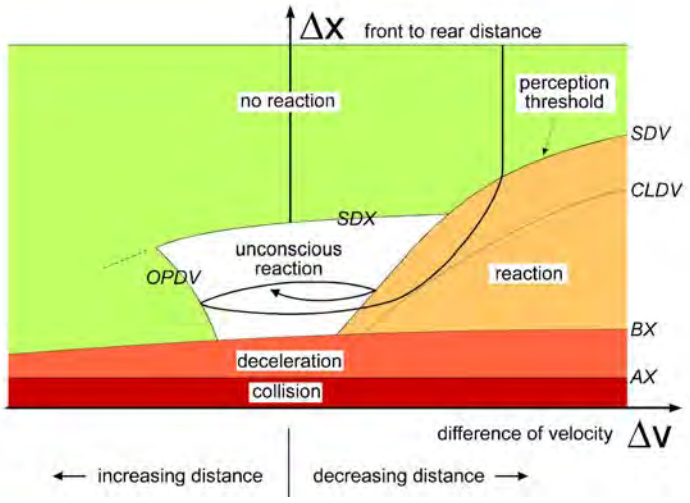



Figure 2.2.: Car-following logic of the Wiedemann model. (PTV Vissim, 2011)

According to (Wiedemann & Reiter, 1992), in each of the regimes a different acceleration function a(s,v,$\Delta$v) is applied which is a function of the gap between the leading and the

following vehicles, the speed of the following vehicle and the relative speed of the following vehicle. The boundaries between the regimes are given by a non-linear equation of the form f(s,v,Δv) = 0 defining curved areas in three-dimensional spaces (s,v,Δv). The (Wiedemann & Reiter, 1992) defines each of the thresholds as :

- AX: It is the desired distance between the front of the two successive vehicles in a standing queue.
- ABX: It is the desired minimum following distance which is a function of AX, a safety delta distance BX, and the speed of the vehicle.
- SDV : It is the action point where the driver consciously observes that he approaches the slower leading vehicle, SDV increases with increasing speed difference Δv.
- OPDV : It is the action point where the following vehicle driver notices that he is slower than the leading vehicle and starts to accelerate again.
- SDX : It is the perception threshold to model the maximum following distance which is about 1.5 - 2.5 times of ABX.

Complete flow chart to calculate the acceleration of the Wiedemann model is attached as Appendix A.

## 2.2. State Machine

A state machine (SM) is a mathematical abstraction model that represents a limited number of states and behaviour, such as actions and transition between those states. The basic building blocks of a state machine are states and transitions. A state is a situation of a system depending on previous inputs and causes a reaction on following inputs. One state is marked as the initial state, in which the execution of the machine starts. A state transition defines for which input a state is changed from one to another. Depending on the state machine type, states and/or transitions produce outputs.

State machines with finite number of states are called finite state machines (FSM) and they can be defined as a triple, T = (Q, I, $\phi$) (Gladyshev & PA, 2005), where

- I is the finite set of all possible events,
- Q is the finite set of all possible states and
- $\phi$ : I x Q $\rightarrow$ Q is a transition function that determines the next state for every possible combination of event and state.

Consider the simple state machine in Figure 2.3. It consists of two states, namely Off and On. On is the initial state here, that is activated when the state machine is executed. The arrows between the states denote the possible state transitions. They define for which

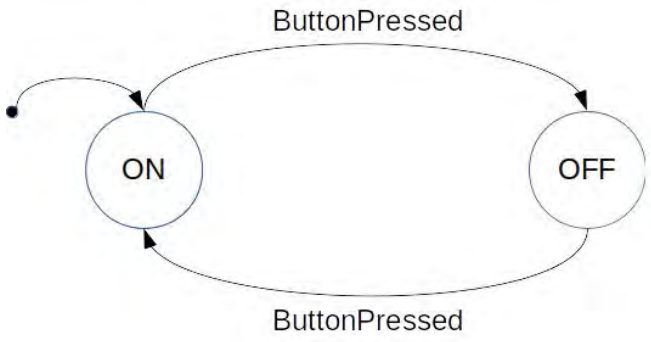



Figure 2.3.: A simple state machine

input a state change occurs. Here, the active state is changed from On to Off for the input, "ButtonPressed", and back again to On for the same input.

## 2.3. Genetic Algorithm

A genetic algorithm (GAs) is a search technique which is used to find the true or the approximate solutions of optimisation and search problems. A genetic algorithm is essentially a type of evolutionary algorithm (De Jong, 2012). GA is different to the traditional optimisation techniques as it works with coding of the parameters instead of the parameters themselves.

In GA, the decision variables are encoded into finite-length strings of alphabets of certain cardinality. These alphabets are called genes and the finite length strings, which are the candidate solutions of the optimisation problems, are called chromosomes (Sastry et al., 2005). GA unlike traditional methods relies on the population of the candidate solution and the size of the population is determined by the user. The population size is an essential part of GA as it affects the scalability and performance of the algorithm (Sastry et al., 2005). A smaller population size may lead to early convergence and does not yield the optimum solution. In order to find the best solution, GA uses the fitness value of each candidate solution. The fitness is calculated by defining a mathematical objective function related to the optimisation problem. Once the decision variables are encoded and the objective function for the fitness is defined, the GA evolves using the followings steps (Sastry et al., 2005):

- Initialization: The initial population of the candidate is generated across the search space defined by the user for each parameter.
- Evaluation: Once the population is initialised or the offspring population is created, the fitness value of each candidate solution is calculated.
- Selection: Selection is a process in which the concept of survival of the fittest is realised by imposing higher number of copies of the solutions having higher fitness. The main

idea of selection process is to select the better fitting solution compared to the worse ones. There are various types of selection processes in GA, e.g. roulette-wheel selection, stochastic universal selection, ranking selection and tournament selection.

- Crossover: Crossover is also called recombination. It is an important step of GA in which two or more parental solution are combined to create a new and potentially better solution. This new and better solution is called offspring in GA terminology. In GA, there are various ways to implement the crossover operation, some of them are listed below:
  - K-point crossover
  - Uniform crossover
  - Uniform order-based crossover
  - Order-based crossover
  - Partially matched crossover
  - Cycle crossover
- Mutation: Mutation is one of the important steps in GA. It essentially modifies a solution randomly. It improves the chances of the algorithm not to get stuck in the local solutions by providing diversity to the population.
- Replacement: Once the steps of selection, crossover and mutation are done, a new population called offspring population, replaces the original parental population. Elitist replacement, generation-wise replacement and steady state replacement methods are some of the widely used ones.
- The steps 2-6 are repeated in the evolution of the GA until the termination condition is met. The termination condition is determined by the user. It can be either a maximum/minimum fitness value or a maximum number of generations (iterations).

## 2.4. Neural Network

Deep learning is considered as reliable technique to solve computationally intensive and challenging problems. Today the base for deep learning is an Artificial Neural Network (ANN), which is a computing system inspired by the structure of the human brain.

The basic structure of an ANN is a network of small processing units called nodes which are arranged in layers and connected by weighted connections. The network is activated by providing inputs to the first layer and this activation is spread through out the network using the weighted connections (Kuri-morales, 2014). The first layer is called the input layer, followed by one or more hidden layers and finally an output layer. A typical structure of an ANN is shown in Figure 2.4.

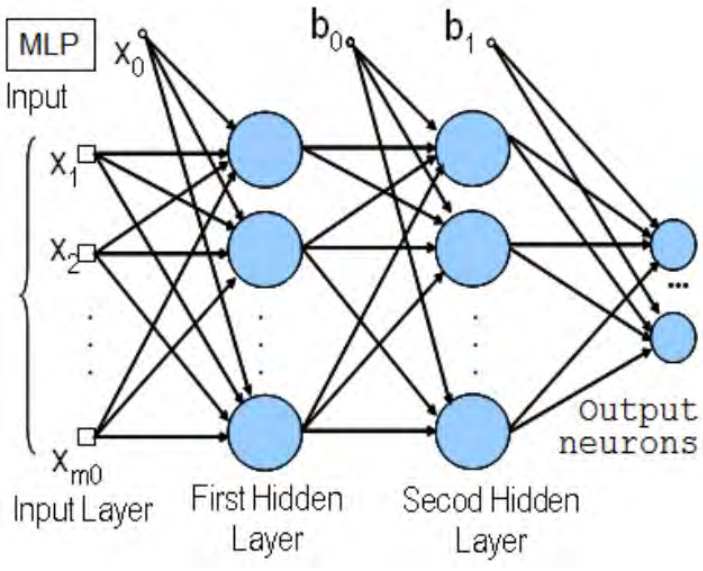



Figure 2.4.: A simple Artificial Neural Network (Kuri-morales, 2014)

Multiple variants of ANNs have been developed depending on different types of problems and datasets. A Convolutional Neural Network is such a class of ANNs, which is predominantly used when the dimensions of input data is too large, a typical example is computer vision applications. Another class called Recurrent Neural Network are used for solving the sequential data problems. An overview of these ANNs is given in next sub-sections.

### 2.4.1. Convolutional Neural Network (CNN)

A Convolutional Neural Network is realised by adding an additional convolution operator to a classic Neural Network. A convolution operator is a special kind of linear filter which maps the input vector or matrix to a smaller output based on the individual weights. The most important feature of a convolution operator is that, it allows the models to extract spatial and temporal features of the input data. This allows the model to reduce the dimensionality of the data without loosing the important features and emphasizing only on important features. This makes the training faster and enhances the performance of the neural network.

### 2.4.2. Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) (Afshine & Shervine, 2020) is a class of an artificial neural network architecture that is inspired by the cyclical connectivity of neurons in the brain, which uses iterative function loops to store information. This means, RNN introduces the concept of memory. In normal Neural Network architecture, it is assumed that all inputs and outputs of the network are independent of each other, whereas in RNN the output of the previous layer is used as input for the subsequent layers. Both inputs and outputs to RNN

are typically sequences. An input sequence can be denoted as ($x^{<1>)}$, $x^{<2>}$, $x^{<3>}$...$x^{<T>}$), where each element in an input sequence is a vector and a target sequence can be denoted as ($y^{<1>}$, $y^{<2>}$, $x^{<3>}$...$y^{<T>}$). There are multiple variants of RNN architecture depending on the output, e.g. One-to-One, Many-to-Many, Many-to-One, shifted Many-to-Many. Different architectures have different applications, e.g. Many-to-One is used for the time series prediction, Many-to-Many is used to solve sequence to sequence prediction problems and Shifted Many-to-Many can be used to solve the problems of language translations.

A simple One-to-One architecture is shown in Figure 2.5, where 'X' is the input at each time step and 'Y' is the output. 'A' is the computational unit that receive the input $X_{(t)}$ from the current timestamp and a hidden state $h_{(t-1)}$ from the previous timestamp to calculate the output $Y_{(t)}$ for the current timestamp and hidden state $h_{(t)}$ for the next timestamp.

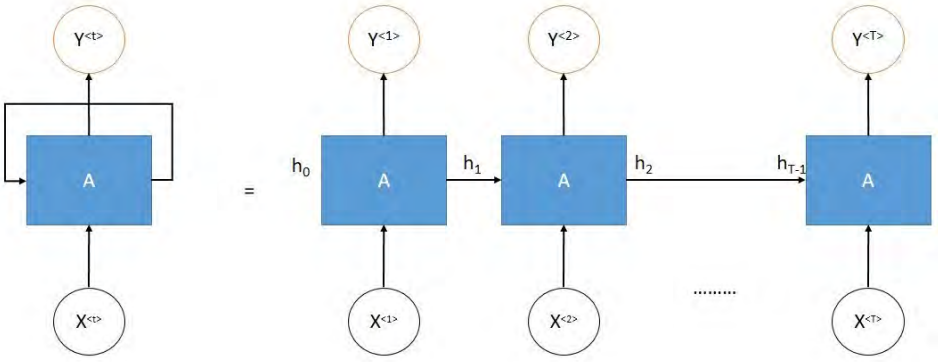



Figure 2.5.: A simplified One-to-One RNN model, adopted from (Afshine & Shervine, 2020)

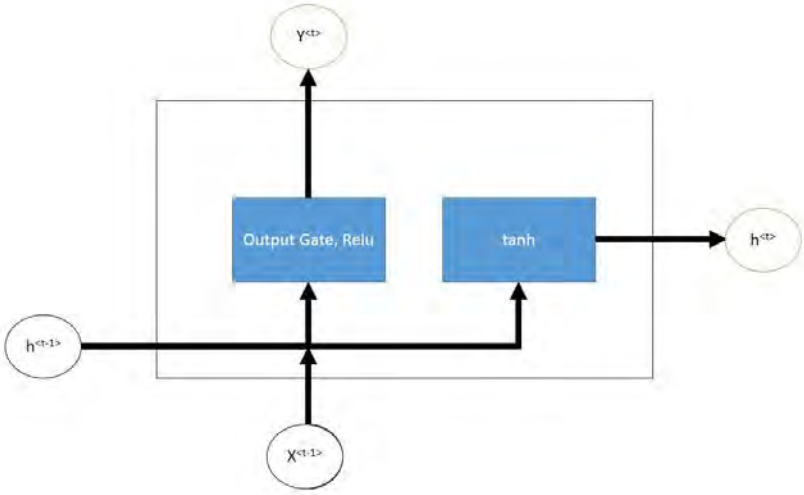



Figure 2.6.: Internal Structure of the computational unit A of a simple RNN, adopted from (Afshine & Shervine, 2020)

The structure of the computational unit is shown in Figure 2.6. There are two operations performed in the computational unit, one is to predict the current output Y and the other is to compute the hidden state for the next timestamp. During computation, each neuron shares two types of weights. Weight $w_{(1)}$ is associated with the current input and the other $w_{(2)}$ is associated with the hidden state of the previous timestamp. The formulas for the computational unit can be summarised by Equation 2.11 and Equation 2.12 .

$$Y_t = Relu(w_1 x_t + w_2 h_{t\text{-}1}) \,, \tag{2.11}$$

$$h_t = tanh(w_1 x_t + w_2 h_{t\text{-}1}) \,, \tag{2.12}$$

where the Relu function is defined as:

$$Relu(z) = max(0, z) \,, \tag{2.13}$$

and tanh, called hyperbolic tangent, is defined as:

$$tanh(z) = \frac{e^z - e^{\text{-}z}}{e^z + e^{\text{-}z}} = \frac{e^{2z} - 1}{e^{2z} + 1} \,. \tag{2.14}$$

RNNs have several properties that make them an attractive choice for sequence data prediction. They are flexible in their use of context information because they can learn what to store and what to ignore. They accept many different types and representations of data and they can recognise sequential patterns in the presence of sequential distortions. Depending on the structure and computation complexity of the computation unit 'A', RNNs have different variants and one such variant is Long Short Term Memory architecture of RNN. The following is described in the next subsection.

### 2.4.3. Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) network is a special kind of Recurrent Neural Networks which are capable of learning long term dependencies (Hochreiter & Schmidhuber, 1997). LSTMs also have a chain like structure similar to RNNs but with modifications in the computational unit. A typical LSTM cell is shown in Figure 2.7 which contains a memory cell. It controls the flow of information by using input, forget and output gate layers which discard the non-essential information and memorize only essential information. These operations in the LSTM cell can be represented by the following set of equations (Olah, 2019).

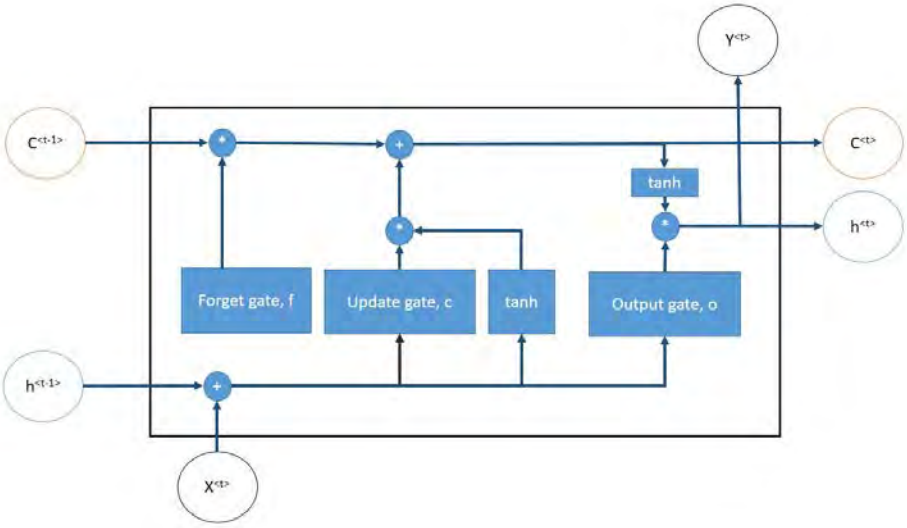



Figure 2.7.: The internal structure of the computational unit of an LSTM, adopted from (Olah, 2019)

$$forgetgate, f_t = \sigma(x_t U^f + h_{t\text{-}1} W^f) \,, \tag{2.15}$$

$$inputgate, i_t = \sigma(x_t U^i + h_{t\text{-}1} W^i) \,, \tag{2.16}$$

$$g = tanh(x_t U^g + h_{t\text{-}1} W^g) \,, \tag{2.17}$$

$$cellupdate, c_t = c_{t\text{-}1}.f + g.i \,, \tag{2.18}$$

$$outputgate, o = \sigma(x_t U^o + h_{t\text{-}1} W^o) \,, \tag{2.19}$$

$$h_t = tanh(c_t).o \,, \tag{2.20}$$

where, $x_t$ is the current input, $h_{t-1}$ is the last hidden state, U and W are the weighting matrices and i, f, o are the input, forget and output gates respectively.

The gates here are referred to sigmoid functions which can be defined Equation 2.21, It essentially maps the input to a scalar between 0 and 1.

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{2.21}$$

To understand the working of an LSTM computational unit, the first step is to define which information is not needed in the cell state and this decision is made by forget gate layer. It looks at the input $x_t$ and hidden state $h_{t-1}$ then by using sigmoid. It outputs a number between 0 and 1, where 0 means forget completely. The next step is to decide how much is the new information or the current input is allowed to pass for the calculations of the internal states. This operation has two parts, one is handled by an input gate which decides which value to be updated and the other is handled by tanh layers (Equation 2.17) that creates a vector of new candidates that could be added to the state. Then the cell update operation is done which updates the old cell state $c_{t-1}$ to the new cell update $c_t$. Finally, the output of the memory cell needs to be calculated which is the current output $y_t$ and the hidden state for the next layer $h_t$ which are calculated using the Equation 2.19 and Equation 2.20 respectively.

In summary, LSTM cell can learn to recognize a certain input with the help of an input gate, store this input in the long-term state, learn to preserve it for as long as necessary with the help of the forget gate and extract it on demand. This ability of LSTM cells makes them suitable for capturing the long-term patterns in time-series data.

# 3. Literature Review

In recent years, researchers have proposed various methods to extract the car-following scenarios, analyse the performance of existing car-following models and develop new car-following models. This chapter gives an overview of the different researches done to realise the above mentioned tasks. In Section 3.1 different techniques that researchers have used to extract the driving scenarios from the data are explained. Section 3.2 discusses the recent researches in the area of analysis and improvement of existing car-following models. Recent developments towards data-driven car-following models are explained in Section 3.3 and Section 3.4 summarises the literature.

## 3.1. Car-Following Scenario Extraction

During driving, a driver performs various maneuvers, e.g. changing lanes, turning left or right and following a leading vehicle. These maneuvers are broadly classified as driving scenarios. Figure 3.1 gives a classification of the normal scenarios a driver undergoes.

| *Scenario* | *Scenario Classes* | *Semantic description* |
|---|---|---|
| Free driving/ Vehicle following | Free driving | No predecessor, ego vehicle is following lane |
| | Vehicle following | Ego vehicle's intention is to keep the lane and is influenced by a predecessor vehicle |
| Lane change | Lane change right | Ego vehicle's intention is to change to a near lane |
| | Lane change left | |
| | No lane change | |
| Cut-In | Cut-In | passive, another traffic participant intention is to merge into the lane of the ego vehicle |
| | No Cut-In | |

Figure 3.1.: Scenarios and scenario classes in a typical driving (Roesener et al., 2016)

Car-following is one of the essential driving scenarios. Various studies in the past have been done to classify, predict and extract driving scenarios for the research purposes. The subsequent subsections discuss the rule based methods and machine learning methods used by the researchers for car-following extraction and classification.

### 3.1.1. Rule Based Methods

Rule based methods use logical rules to solve the problems, e.g. classification of driving regimes. They are simple and easy to implement. Knowing the structure of the data and the problem statement, one can define a set of rules to get the desired output of the modelled system. Researchers have used these methods to solve multiple types of problems and extraction of car-following scenarios is also one of them.

A study done on the car-following modelling was done by (Chong et al., 2013). They used the data from the Naturalistic Car Driving Study (NCDS) collected by Virginia Tech Transportation Institute. The NCDS data consists of 2 million vehicle kilometers and 43,000 hours of driving. The data was collected using 100 different drivers. Measured data consisted of speed, longitudinal and lateral acceleration, yaw angle, brake status, relative distance to the leader vehicle, relative speed to the leader vehicle and azimuth (the horizontal direction of the front vehicle). Car-following data was then extracted using the below listed conditions/rules:

- Radar target ID > 0.
- Radar range $\leqslant$ 120 m.
- 1.9 m < $\Delta x$ * Sin(Azimuth) < 1.9 m, $\Delta x$ is the relative distance of the following vehicle to the leading vehicle.
- Speed $\geqslant$ 20 km/h.
- Duration of car-following period > 30 seconds.

After the implementation of above rules to extract the car-following data, the processed data was manually verified using the video, which was recorded at the time of data collection.

Another study (Zhu, Wang, & Wang, 2018) was done to develop a human-like car-following model using Reinforcement Learning (RL) method. Data collected for the Shanghai Naturalistic Driving Study was used in the study. Five vehicles were used to collect the driving dataset with 60 different drivers across a total distance of 161,055 Kilometers. The collected dataset consisted of the position, longitude and lateral accelerations, steering and throttle angle of the instrumented vehicle, relative distance between the instrumented and the surrounding vehicle and speed of the instrumented and surrounding vehicle. The dataset was collected using the radar system and four synchronised cameras. The video data was also recorded. The four synchronised cameras recorded the videos of face and hand movements of the driver, front and rear view of the vehicle. To extract the car-following data, the listed rules were proposed.

- Radar identification number.
- Relative Distance of the instrumented vehicle to surrounding vehicle < 120 meters.
- Lateral distance < 2.5 meters.

- Duration of car-following period > 15 seconds.

The results of the extraction were then analysed with the video recorded during the data collection.

Another study done by (Hülnhagen et al., 2010) introduced fuzzy reasoning in recognition of only turn scenarios. The dataset considered for the study was collected using a driving simulator. (Gerdes, 2006) introduced a probabilistic Bayesian model. The study was done to identify the lane change scenario. The output of the system was the identified lane change with a confidence level.

### 3.1.2. Machine Learning based Methods

(Tango & Botta, 2009) classified different driving scenarios using a simulated dataset. The research was done in the Human Machine Interaction Laboratory in University of Italy to collect the dataset using the driving simulator. The driving simulator was used to collect the data for lane change and car-following scenarios. The car-following labelled scenario was then classified using Support-Vector Machine (SVM) and Neural Network (NN) models. SVM is a machine learning algorithm that is used in supervised classification problems. SVM classifier performed better than NN classifier with an accuracy of 97% for the car-following instances.

(Roesener et al., 2016) classified the driving scenarios in real driving data using time series classification of human-driving behaviour. The driving data was classified into 3 driving scenarios, namely lane change, free driving and cut-in using three different classification algorithms: Naive Bayes, Ada Boost and Medium tree algorithms respectively. First, the data was manually labelled each scenario and then this labelled dataset was used to train the above listed classification algorithms. Each separate algorithm gave satisfactory results for each of the scenarios.

Some recent studies focused on classification of scenarios like lane change and cut-in were proposed using machine learning algorithms. (Tang et al., 2018) proposed an adaptive fuzzy neural network to predict the steering angle and thus the lane change scenario. However, none of the study was done to extract the car-following scenario using unlabelled naturalistic driving dataset.

## 3.2. Analysis and Improvement of Existing Car-Following Models

Car-following models are researched for more than half a century. The performance of these models is very essential, not only for realistic traffic simulations but also for the research in the field of autonomous driving. The initial proposed car-following models were discussed in

chapter 2, apart from them many researchers have invested their crucial time and energy in analysing, calibrating and improving those models and those researches are discussed in this section.

(Mitra & Eric, 2018) calibrated the traditional car-following models; the IDM, Wiedemann model and Krauss model. They used the Next Generation Simulation (NGSIM) (FHWA, n.d.) dataset to analyse the performance of different car-following models. A Genetic algorithm was used for the research objective but information regarding the mutation, crossover and selection types of genetic algorithm was not provided. The result showed that amongst the three driving models, the calibrated IDM model fitted best to the real world trajectories while the Krauss model fitted worst.

One analysis study of car-following models using trajectory data was done by (Zaky et al., 2016). Author developed the methodology to classify different driving regimes during car-following and calibrated the IDM for each driving regime. Markov-regime switching model was used to classify driving regimes. Markov-regime switching is a non-linear time series model (Kuan, 2002). To realise the Markov-regime switching model, velocity of the following vehicle was considered as the time series variable and acceleration, distance headway and the velocity difference were considered as exploration variables. The mathematical representation is shown in Equation 3.1.

$$v_{\mathrm{t+1}} = \phi_{1,\,\mathrm{s_t}} * a_t + \phi_{2,\,\mathrm{s_t}} * dv_t + \phi_{3,\,\mathrm{s_t}} * h_{\mathrm{j,t}} + \epsilon_t \ , \tag{3.1}$$

where $v_{t+1}$ is the velocity of the following vehicle, $a_t$ is the acceleration of the vehicle, $h_{j,t}$ is the gap, $\phi$ represents the coefficients and $\epsilon$ is the error term.

They used the Robert Bosch GmbH research group dataset. The dataset had stop-and-go speed profiles of the following vehicle on a single lane road. The dataset was explicitly recorded for the car-following behaviour and hence the dataset had traces with 250, 400 and 300 seconds of car-following behaviour. The results showed that the IDM model calibrated for different regimes fitted better to the trajectories in comparison to the model fitted for the whole trajectory. The approach has one drawback that, changing parameters for each time step in running simulation is a cumbersome task.

Another similar study was done by (Ma & Andréasson, 2007). The author implemented a fuzzy clustering algorithm to classify the regimes during the car-following behaviour. The dataset was collected with an instrumented vehicle from Volvo. Two video cameras with lidars were installed to record the vehicles at the rear and front of the instrumented vehicle. The data was collected explicitly to record the car-following behaviour with a set of five different drivers. The Volvo ERS software was used to synchronise the video and the car-following and lane changing situations. The data was recorded at a sample rate of 50 Hz. However, none of the existing car-following models was analysed with these regimes and hence the study did not confirm the effect of these classified regimes in improvement of the car-following models.

(Y. Zhang et al., 2017) presented an improved version of Full Velocity Difference (FVD) model by incorporating the driver's characteristics and the leading vehicle's acceleration. The data used in the study was Next Generation Simulation (NGSIM) (FHWA, n.d.). The driver's characteristics were determined by clustering the recorded data using k-means clustering algorithm. The speed of the leading vehicle and the space between the leader and the follower vehicle were used as input features for the clustering algorithm. The proposed model was then analysed using numerical simulations. The result concluded that the introduction of the leading vehicle acceleration can contribute to avoid the negative velocity which possibly appears in FVD. The performance of the proposed model compared to the real world data was not investigated in the study.

A recent study on the improvement of the existing car-following model was done by (Zhao et al., 2018). The Full Velocity model (FVD) was extended by considering the acceleration of the leading vehicle in the equation of the existing FVD model as presented in Equation 3.2.

$$\begin{aligned} a_n(t) &= \kappa\left(V(\Delta x_n(t)) - v_n(t)\right) + \lambda \Delta v_n(t) \\ &+\mu_1 H(-\Delta v_n(t))(\Delta x_n(t) - \mu_2) + \mu_3 H(\Delta v_n(t)) a_{\text{n-1}}(t)\ , \end{aligned} \tag{3.2}$$

where, $\lambda$ and $\kappa$ are the parameters of the model, $\Delta x_t$ represents the gap between the following and the leading vehicle, $v_n(t)$ is the velocity of the following vehicle at time $t$, $\Delta v_n(t)$ is the velocity difference between the leading and the following vehicle at time $t$, $a_{n-1}(t)$ is the acceleration/deceleration of the leading vehicle at time t, $\mu_1$ is the sensitivity parameter of the distance headway when the vehicle gathers at an intersection, $\mu_2$ is the safe driving distance headway, $\mu_3$ is the sensitivity parameter of the acceleration/deceleration of the leading vehicle.

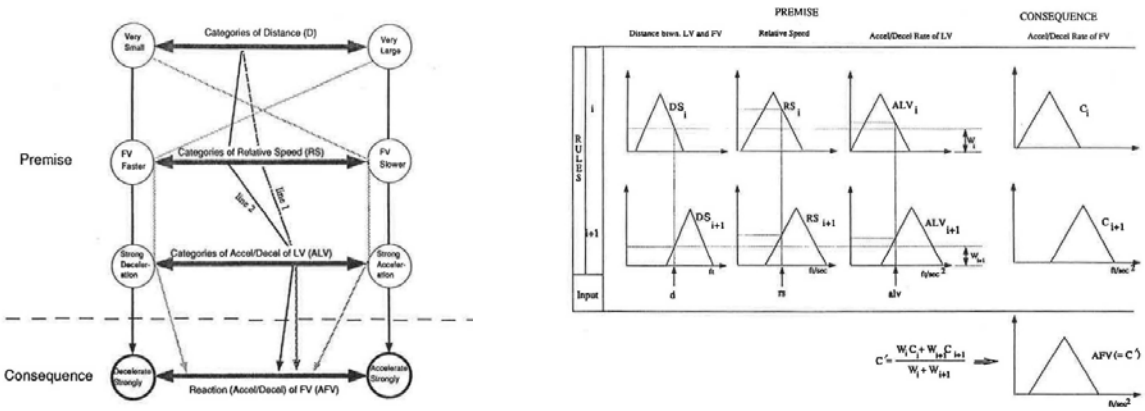
The research objective was to improve the performance of the model on intersections. The performance of the existing FVD and the proposed model were analysed during the deceleration and acceleration profiles of vehicles at an intersection/junction approach. The NGSIM (FHWA, n.d.) data was used for the analysis where the trajectories near to the intersection were considered. The main finding of the results showed that the proposed model could distinguish the vehicles that gather and dissipate near to the intersection depending on the velocities of the leader and the following vehicles. The results also showed that the improved model output fits better to the observed data than the existing car-following model. The model also had shortcomings when the author tried to fit the model to the long car-following trajectories. The author claimed that the additional parameters which are coefficient of $a_{n-1}(t)$ and the $\mu_2$ might be the reason for the irregularities in the final fitness of the model. The parameter $\mu_2$ defines the average safe driving distance headway and since different drivers have different characteristics, the parameter $\mu_2$ needs to be dynamically calibrated as per the driver's characteristics which again is a tedious task during the simulation runs. The model also showed the problem of advance acceleration profiles during the dissipation of vehicle at intersection.

## 3.3. Data-Driven Development of Car-Following Models

Each of the car-following models presented in the previous section either consists of one or more equations to predict the driver's behaviour. Due to their dependency on fewer equations, those models are easier to present and understand. However, it is difficult to capture human driver behaviour with these fixed sets of equations because of the broad scope and variance of human driving. Recent development in the data collection techniques, such as Extended Floating Car Data (xFCD) allows researchers to use real world data to analyse and model the car-following behaviour in a more advanced way. Several data-driven researches have been done using machine learning techniques like fuzzy logics, Neural Networks, Reinforcement Learning etc. which are discussed in this section.

Driver understanding and behaviour is qualitative and sometimes the actions are based on a set of rules. For example, during the deceleration step, if the driver is close to the vehicle ahead and with its current speed if the driver gets further close (closing). Then, the driver may decide and act according to the experience, logic and judgement instead of exact relative speed and spacing between the vehicles (Aghabayk et al., 2015). In these kinds of situations the "close" and "closing" is basically a fuzzy value and the deceleration is a fuzzy decision making. To consider the fuzzy perception and driver's decision, several fuzzy logic based models were developed in the field of car-following behaviour modelling.

(Kikuchi & Chakroborty, 1992) proposed a fuzzy logic based car-following model which is consisted of two modules, a fuzzy inference system (shown in Figure 3.2a) and the system that executes the inference system (shown in Figure 3.2b). In the inference system they defined the two rules 'Premise' and 'Consequences'.




(a) Formation of rules

(b) Execution of fuzzy inference system

Figure 3.2.: The fuzzy logic based car-following model (Kikuchi & Chakroborty, 1992)

For the premise rule, they used the three input variables, namely the relative distance $\Delta x$ , the relative velocity $\Delta v$ and the acceleration $a$ of the leading vehicle and grouped each of

the first two input variables into six natural-language based categories while the acceleration was grouped into twelve (six for acceleration and six for deceleration). Each of these grouped categories comprises the fuzzy set. The consequence was dependent on premise rule. It was the acceleration or the deceleration rate of the following vehicle which was expressed in fuzzy quantity. Each fuzzy quantity was represented by a natural language term such as "VERY STRONG DECLARATION". The results showed that the reaction of the following vehicle was similar to the leading vehicle's action but there was no successful implementation due to the lack of calibration of the model itself. Further, many other car-following models based on fuzzy logic rules were developed. (Won et al., 2006) used the fuzzy approximate reasoning to find the driver's sensitivity with the traffic situation. The driver based conditions were driver's gender, age, experience, vehicle specific information like year of manufacture, engine volume etc. However, amongst all the proposed models using fuzzy logic, the most common problem was to determine the fuzzy rules as used by humans. If the rules are not set appropriately, the model will predict inappropriate results.

A method called Rough set theory which is closely related to Fuzzy logic was also applied in car-following behaviour modeling. Rough set theory is a mathematical tool for dealing with vague, imprecise, inconsistent and uncertain knowledge (Q. Zhang et al., 2016). (Hao et al., 2018) developed a data-driven car-following model based on rough set theory. The main idea of the research was to extract out the optimal decision rules from the raw data using the rough set theory and determine the follower's behaviour in the next time step. NGSIM (FHWA, n.d.) 180-1 data was used in the research. The model used the status of the follower and the leader at time $t$ plus the traffic situation as an input for the decision system to predict the velocity of the follower at time $t+1$ using the optimum decision rule set proposed by the author. The study showed that the data-driven car-following model performed better than the conventional full velocity difference model (FVDM). Due to the dependency of the proposed model on existing decision rule set, the model had a drawback of poor performance under an unrealistic decision rule set.

(Papathanasopoulou & Antoniou, 2015) developed a non-parametric data-driven car-following model based on a locally weighted regression (loess) model. The methodology consisted of two parts, namely training and application. In the training part the recorded data was used to fit the macroscopic traffic models and later in the application part the fitted traffic models were used to predict the speed. They considered velocity of the following vehicle, velocity of the the leading vehicle and the distance between the two vehicles as an input feature. The output response of the model was the estimated velocity of the following vehicle at the next time step. The biggest advantage of the proposed method was that it does not require a specific function to fit a model simultaneously to all the data points. The model was compared to the Gipps (Gipps, 1980) model as reference and the results showed that machine learning models can outperform traditional car-following models. Although the proposed model does not considered the impact of other input features on the car-following behaviour which gives a research gap to work in those aspects.

Another technique called Artificial Neural Networks (ANN) has been extensively applied in the last decade in the field of transportation. (Panwai & Dia, 2007) developed a neural agent car-following model. The model was based on desired distance headway. They used the speed of the leading and following vehicle and the distance headway to model the car-following behaviour using the ANN. AIMSUN traffic simulator was then used to evaluate the performance of the proposed model. They compared the fuzzy logic based model and the Gipps model as reference to evaluate the performance. The results showed a good fit of the simulated result to the observed data but the model failed to replicate the car-following behaviour during the stop and start phase of the vehicle. A similar study was done by (S. Lee et al., 2019), where they developed an integrated deep learning and stochastic car-following model. A multi-lane stochastic optimal velocity model (SOVM) model was integrated to a Convolutional Neural Network (CNN). SOVM was used to model the car-following behaviour and CNN to predict the lane change.

(Khodayari et al., 2012) developed a modified neural network model. They proposed a new idea to calculate the instantaneous reaction time and use it along with the relative distance, the relative velocity and the velocity of the following vehicle as an input to predict the acceleration of the following vehicle. Figure 3.3 shows the basic architecture of the Neural Network used in the study. The model was tested with fixed reaction times of 0.1 seconds, 0.2 seconds and 0.6 seconds and results showed that the model with instantaneous reaction time performed better as compared to the fixed reaction times. The model was neither compared to any existing car-following models nor was integrated to any of the traffic simulation software.

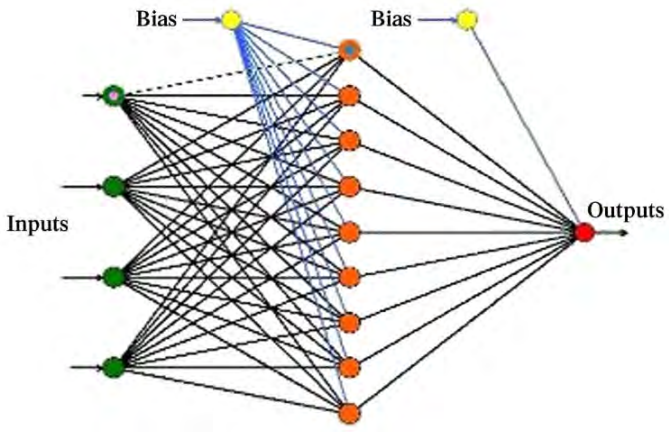



Figure 3.3.: Neural Network structure for a car-following model (Khodayari et al., 2012)

Apart from these, multiple studies in the past used different forms and architectures of neural networks. (Jia et al., 2003) applied the 'back propagation' algorithm to develop a car-following model using data collected by a technique named 'five-wheel system'. The model predicted the acceleration or deceleration values in response to the distance headway

and the velocity difference of the leading and the following vehicle. The model lacked the generalisation capability due to the dataset used by the researcher.

Machine learning models can capture more complex relations as compared to fixed mathematical equation based models, but some researchers argue that machine learning models can fail when it comes to test safety aspects. So, a model combining machine learning and a kinematic model was developed by (D. Yang et al., 2019). Their main research aspect was to overcome the machine learning based model's output in uncommon situations like an accident during the automated driving. They combined two machine learning techniques namely, the Back-Propagation neural network and the random forest with the Gipps Model (Gipps, 1980) (Kinematic Model). Next Generation Simulation (NGSIM) (FHWA, n.d.) dataset was used for calibration and modelling. For the testing, a ring-test was performed by creating a pseudo ring having the perimeter of the same length of a vehicle platoon extracted from the NGSIM dataset at a particular time instant. On the proposed ring, the congestion area, stable time and crash rates were calculated. The model showed improved safety results as compared to the Gipps model.

Reinforcement Learning is one of the most actively researched domains in the present situation. It has been used widely in artificial intelligence research. (Zhu, Wang, & Wang, 2018) proposed a novel car-following model using Deep Reinforcement Learning (DRL). They used the Shanghai Naturalistic Driving data which was predominantly collected on freeways and highways. This data was fed into a simulation environment where the Reinforcement Learning agent learned from trial and error based on a reward function. The reward function signals how much the agent deviates from the recorded data. The conceptual methodology of the model is shown in Figure 3.4.

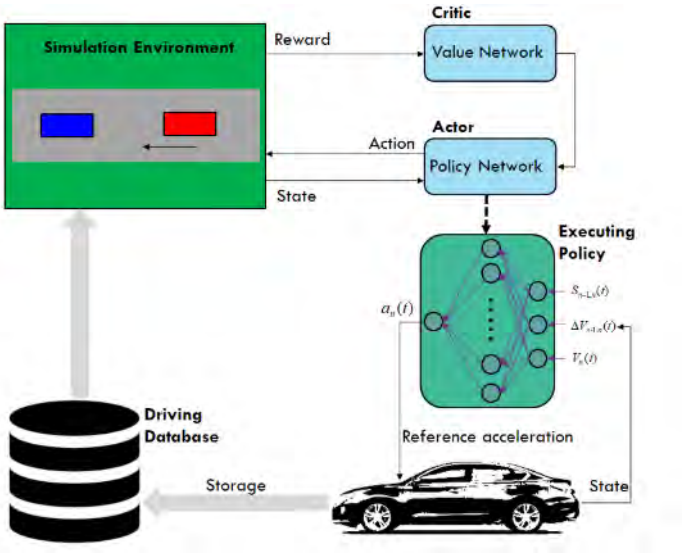



Figure 3.4.: Conceptual Diagram of the Deep Reinforcement learning based Car-Following Model (Zhu, Wang, & Wang, 2018)

They proposed Deep Deterministic Policy Gradient to develop the Reinforcement Learning algorithm, where the critic network approximates the action-value function and the actor network represents the agent's current policy for the Q-function (part of Reinforcement learning) which is used to map the state of the environment to an action of an agent. The agent here refers to the driver.

Through this learning policy they developed a car-following model that can predict the acceleration of the following vehicle given the speed of the following vehicle, velocity difference and gap between the two interacting vehicles. They compared the proposed model with the Intelligent Driver Model (IDM) and the Deep neural network based model. The numerical simulation results showed better performance as compared to the two considered models on highway car-following scenarios. The integration of the proposed models was not investigated with any traffic simulator.

Recent studies are using Recurrent Neural Networks (RNNs) to capture the car-following behaviour. (Zhou et al., 2017) developed an RNN based microscopic car-following model to predict the traffic oscillations. The RNN model developed predicted the gap between the vehicles. Author found that the existing models were not able to be calibrated for the traffic oscillations. Author's RNN model outperformed the conventional calibrated IDM model. The data used in the research was the NGSIM dataset.

## 3.4. Research Gap and Conclusion

### Car-Following Scenario Extraction

Table 3.1 summarises the literature on the extraction of car-following scenarios from the different datasets. It is clear that most of the studies used the datasets which are recorded for the driving behaviour studies. The rule based methods are used for the unlabelled datasets where as the machine learning based methods are used for the labelled datasets. There are only few studies done on the raw xFCD. Therefore, there is a need to develop a methodology to extract the car-following scenarios from raw xFCDs.

### Analysis and Development of car-following models

Table 3.2 and Table 3.3 summarise the studies related to the analysis and data-driven development of car-following models respectively. It is clear from the studies that most of the studies used the NGSIM trajectory data while there are very few studies which are focused on the raw xFCD. Apart from this, most of the studies used the binary-coded genetic algorithm for the calibration of car-following models, whereas the calibration task of car-following models is a continuous real value optimisation problem. There is a need to study the performance of the convolutional car-following models by calibrating them using the real-coded algorithms on the car-following traces extracted from the raw xFCDs, as the xFCDs can have unbiased naturalistic driving behaviour since the drivers are unaware of the data recordings and its purpose. Also, the studies show that the machine learning is an actively researched field in the development of data-driven car-following models. The machine learning models are much more flexible than the mathematical equations based conventional car-following models. So, the extracted car-following traces can be used to model a new data-driven car-following model. The researches show that the additional information like the presence of the vehicle on the left and right lane of the following vehicle is not explored much in the given studies. Also, since the driving task can be considered as time series modelling problem as suggested by (Zaky et al., 2016). The impact of the the time series sequence length on the driver's behaviour is also not explored much. This sequence length can be defined as the historical information that the driver processes to take a decision for the next time step. Machine learning models like RNNs/LSTMs allow to explore the sequential data, so there is a need to explore the impact of the historical information on the performance of the data-driven car-following model.

RNN is an actively used machine learning technique to explore sequential data. Since, the driving task is considered in this thesis as a sequential task, RNNs capability to model the sequential data provides the motivation to explore its application in car-following modelling. However, since LSTM can learn long-term dependency in the dataset, the LSTM seems to be a better choice than RNN.

| *Objective* | *Method* | *Technique* | *Data used* | *Labelled Dataset* | *raw xFCD used* | *Source* |
|---|---|---|---|---|---|---|
| Car-following extraction | Rule Based | State Machine | real driving | ✗ | ✗ | (Chong et al., 2013) |
| Car-following extraction | Rule Based | State Machine | real driving | ✗ | ✗ | (Zhu, Wang, & Wang, 2018) |
| Driving scenarios classification | Machine learning | SVM [a], NN [b] | simulated | ✓ | ✗ | (Tango & Botta, 2009) |
| Driving scenarios classification | Machine learning | NB [c], AB [d] | real driving | ✓ | ✗ | (Roesener et al., 2016) |

Table 3.1.: Aggregated studies on car-following scenario extraction

[a] Support Vector Machine
[b] Neual Network
[c] Naive Bayes
[d] Ada Boost

| *Objective* | *Method* | *Technique/Algorithm* | *Dataset type* | *Models* | *raw xFCD used* | *Source* |
|---|---|---|---|---|---|---|
| Performance Analysis | Calibration | GA [a] | trajectory | IDM [b], W[c], K[d] | ✗ | (Mitra & Eric, 2018) |
| Performance Analysis | driving regime calibration | GA, MRS[e] | real driving | IDM | ✗ | (Zaky et al., 2016) |
| Performance analysis | Model Improvement | change in model equation | trajectory | FVD [f] | ✗ | (Y. Zhang et al., 2017) |
| Performance analysis | Model Improvement | change in model equation | trajectory | FVD | ✗ | (Zhao et al., 2018) |
| Performance analysis | Calibration | GA | real Driving, trajectory | IDM | ✓ | (Treiber & Kesting, 2013) |
| Performance analysis | Calibration | GA | real Driving | IDM, W, G[g] | ✗ | (Zhu, Wang, & Wang, 2018) |

Table 3.2.: Aggregated studies on analysis and improvement of the car-following models

[a] Genetic Algorithm
[b] Intelligent Driver Model
[c] Wiedemann model
[d] Krauss Model
[e] Markov Regime Switching
[f] Full Velocity Difference model
[g] Gipps Model

| *Objective* | *Method* | *Technique/Algorithm* | *modelled param* | *Data* | *Data Type* | *raw xFCD used* | *Source* |
|---|---|---|---|---|---|---|---|
| Data-driven development | ML [a] | Fuzzy logic | acceleration | | | ✗ | (Kikuchi & Chakroborty, 1992) |
| Data-driven development | ML | Rough set theory | velocity, optimum decision rule | NGSIM | trajectory | ✗ | (Q. Zhang et al., 2016) |
| Data-driven development | Regression | loess [b] | velocity | real driving | real driving | ✗ | (Papathanasopoulou & Antoniou, 2015) |
| Data-driven development | ML | ANN [c] | velocity | | real driving | ✗ | (Panwai & Dia, 2007) |
| Data-driven development | ML | CNN [d] | velocity | NGSIM | trajectory | ✗ | (S. Lee et al., 2019) |
| Data-driven development | ML | ANN | acceleration, instantaneous $t_r$ [e] | NGSIM | trajectory | ✗ | (Khodayari et al., 2012) |
| Data-driven development | ML | BP [f], RF[g] | acceleration | NGSIM | trajectory | ✗ | (D. Yang et al., 2019) |
| Data-driven development | ML | RL[h] | RL learning policy, acceleration | own test vehicle | real driving | ✗ | (D. Yang et al., 2019) |
| Data-driven development | ML | RNN [i] | gap | NGSIM | trajectory | ✗ | (Zhou et al., 2017) |

Table 3.3.: Aggregated studies on the data-driven development of car-following models

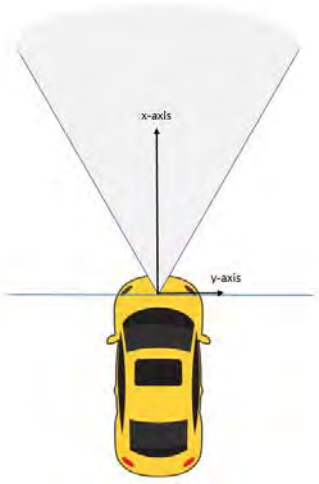
[a] Machine Learning
[b] locally weighted regression
[c] Artificial Neural Network
[d] Convolutional Neural Network
[e] Reaction time
[f] Back Propogation
[g] Random Forest
[h] Reinforcement Learning
[i] Recurrent Neural Network

# 4. Data Collection

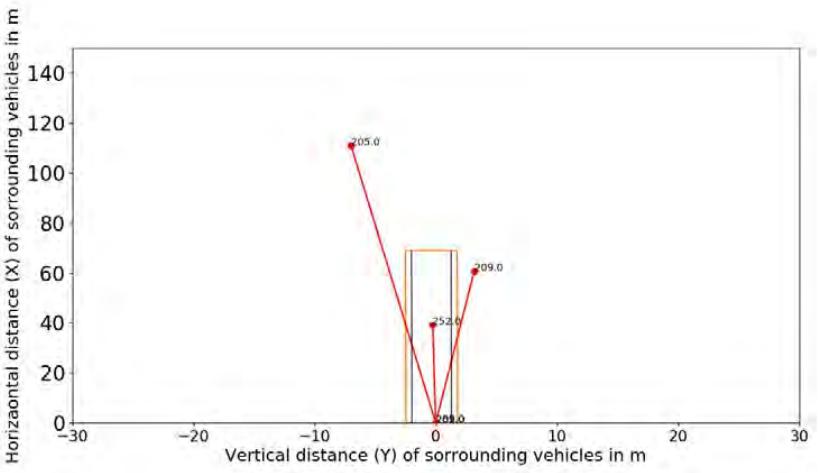
This study uses the Raw Extended Floating Car Data (xFCD) collected by Audi AG using their endurance test vehicles. The SAVe project team from Audi provided the dataset explicitly for this study.

## 4.1. Data Collection Overview

The data is collected and distributed by Audi AG. The number of cars used to collect the dataset is not known. The vehicles used in the data collection were the test vehicle from the endurance testing of Audi. The endurance testing is a part of vehicle testing which could be summarised as a sub-part of the vehicle development process. The data is not collected specifically for the purpose of car-following or any other scenario study. So, the sensors in these vehicles are not designed for any specific task and thus are normal sensors. Each of the vehicles is equipped with a camera and radar in the front part of the vehicle. So, they are capable of detecting the other vehicles in front of the test vehicle. Figure 4.1a shows the field of view of the sensors of the test vehicles and Figure 4.1b shows the visualisation of the data using the visualisation tool developed for this thesis. The reference for the dataset is the vehicle's own coordinate system as shown in Figure 4.2. Usually, the origin is located in the middle of the rear axle, The x-axis points towards the forward direction of the vehicle and z-axis points upwards. Apart from the information about surrounding vehicles, the data of the test vehicle is also recorded and it contains the information of speed, lateral and longitudinal accelerations, heading, position, day and time etc. The drivers of these vehicles are not aware of the purpose of the data collection. Therefore, the collected dataset captures the naturalistic driving behaviour of different drivers.




(a) Field of view of sensors with sensor coordinate system




(b) Visualisation of data

Figure 4.1.: Field of view of sensors in the test vehicle

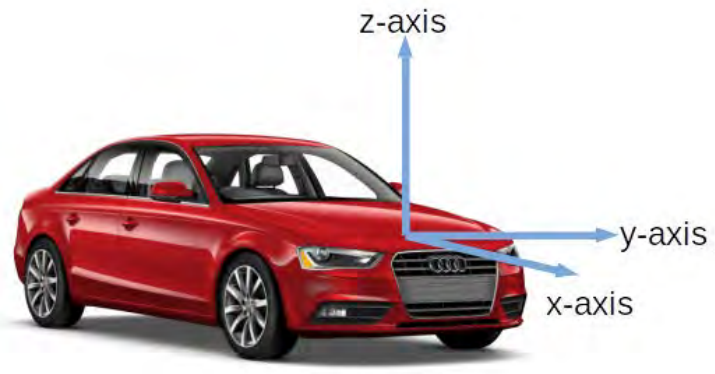



Figure 4.2.: Example vehicle coordinate system

In this study the data from the urban environment is used. Since Audi headquarter is located in Ingolstadt (Germany), a lot of vehicle development and testing process take place here. Therefore, the xFCDs from the Ingolstadt city were filtered first for the further study and analysis. The bounding box for the filtering process is shown in Figure 4.3. A total of 40 hours of driving data was first received for the study which later filtered to 23 hours of the driving data from the city of Ingolstadt.


Figure 4.3.: Study Area Bounding Box (blue) and an example vehicle trace (red)

## 4.2. Meta Details of the Raw xFCD

Each vehicle delivers one file, called trace, mostly generated during an 8-hours shift. Each trace (dataset) contains information regarding the test vehicle, environment, e.g. lanes and surrounding vehicles, e.g. speed, location in Cartesian coordinate system etc. The dataset in the Audi vehicles is collected using the FlexRay Protocol. FlexRay is an automotive network communication protocol developed by FlexRay Consortium to given on-board automotive computing. The Audi xFCD dataset consists of two FlexRay buses namely "FRIA" and "FRIB". The FRIA consist of the data of the test vehicle, for example, speed, location in GPS coordinates, timestamp, lateral and longitudinal acceleration, heading, etc. FRIA also contains the information about the surrounding vehicles and the information of the environment if the information is extracted from the images captured by the camera using the image processing. If the information about the surrounding vehicles and the environment is calculated using the sensor data fusion of the image and the radar signals, then the information is stored in the FRIB bus. There are more than 2000 signals recorded at each timestamp. Different signals in the FlexRay bus system may have different sampling rates. This means that the signals are recorded with different frequencies. So, there are chances that some signals can be empty at every timestamp. The vehicles contain the Protocol Data Units (PDU). These PDUs schedule the sending frequency of the signals, e.g. signals like speed and acceleration are transmitted more frequently than the other signals, like the location signal of the vehicle. Besides this, since the sensors are not designed to capture data for any specific task, they have some instabilities and some signal channels do not contain any information. Therefore a manual check of the signals is necessary.

Table 4.1 is a snippet of the original data (the actual signal names are simplified for better understanding), where, T is the timestamp in milliseconds, Vel is the velocity of the test vehicle in kilometers per hour (km/h), Lat is the geo-coordinate latitude of the test vehicle, $Acc_x$ is the longitudinal acceleration of the test vehicle in meter per second squared ($m/s^2$), a street category is the street/road type and $obj_{1x}$ is the x-coordinate of the surrounding vehicle. The Table 4.1 shows that the sampling frequency of the velocity signal is 40Hz, whereas the sample frequency of acceleration is 20Hz. Apart from this, it can be noticed that the street category signal is empty, which indicates that all signals do not have valid information. Hence, the selection of the important signals and their processing is required. An overview of different signals selected for the further study related to test vehicle and environment are compiled in Table 4.2 and Table 4.3 respectively. Information related to the surrounding vehicles is described in the next section.

Table 4.1.: A snippet of the raw xFCD dataset.

| T (ms) | Vel (km/h) | $Acc_x$(m/s$^2$) | Lat | street category | $obj_{1x}(m)$ | ..... |
|---|---|---|---|---|---|---|
| 0 | 21.42 | -0.009 | 49.194073 | null | 10.11 | ..... |
| 25 | 21.42 | null | null | null | 11.2 | ..... |
| 50 | 21.41 | 0.0599 | null | null | 13.5 | ..... |
| 75 | 21.43 | null | null | null | 13.9 | ..... |
| 100 | 21.435 | 0.044 | null | null | 15.8 | ..... |
| 125 | 21.445 | null | null | null | 16.2 | ..... |
| 150 | 21.44 | -0.0049 | null | null | 16.5 | ..... |
| 175 | 21.435 | null | null | null | 16.7 | ..... |
| 200 | 21.455 | 0.0399 | null | null | 15.9 | ..... |
| 225 | 21.459 | null | 49.194073 | null | 15.4 | ..... |

Table 4.2.: Signal overview of the test vehicle.

| ***Description*** | ***Unit*** |
|---|---|
| The recorded time | [ms] |
| latitude position | [geo] |
| longitide position | [geo] |
| speed | [km/h] |
| Longitudinal acceleration | [m/s$^2$] |
| Lateral acceleration | [m/s$^2$] |
| Heading of the vehicle in the driving direction reference to north | [rad] |
| Date | [-] |
| Time | [-] |
| Street category | [-] |
| No. of lanes | [-] |

Table 4.3.: Signal overview of the environment.

| *Description* | *Unit* |
|---|---|
| X position of the start of left lane marking w.r.t ego vehicle | [m] |
| position of the start of the left lane w.r.t test vehicle | [m] |
| Yaw angle of the left lane marking | [rad] |
| Curvature of the left lane marking | [-] |
| X position of the end of left lane marking | [m] |
| X position of the start of right lane marking w.r.t ego vehicle | [m] |
| Y position of the start of the right lane w.r.t test vehicle | [m] |
| Yaw angle of the right lane marking | [rad] |
| Curvature of the right lane marking | [-] |
| X position of the end of right lane marking | [m] |

### 4.2.1. Information about the surrounding vehicles

In xFCD dataset of Audi, the information of the surrounding vehicle is stored as objects. As mentioned earlier, the data set has two FlexRay buses and depending on the process of information extraction about the surrounding vehicles they could be stored in either of the two. The data about the objects is stored in the memory slot which has a capacity of 10. It means that at a single timestamp the data can give a maximum of 10 objects (surrounding vehicles) information. These slots during the data recording can be filled in parallel. Objects have two most important signals. One is their unique ID and other is the history signal. Objects are always identified with their unique IDs and the history provides the information if the object with given ID is registered for the first time or has been tracked for a long time. So, to track each surrounding vehicle the history information with the ID of the object is very essential. This information is of great importance as the object's information is not fixed in a particular memory slot. Rather, it can change during the entire tracking time. An example of how these memory slots are filled and how the information of different object rotates inside the memory slot is shown in Figure 4.4. The figure shows an example of a rotation of three objects with ids 1, 6 and 224. As it can be seen that the object can jump from one slot to a different slot on every timestamp. Therefore, the history signal and the understanding of memory slot organisation in the dataset play a very vital role in tracking different objects to extract car-following data from the traces. Table 4.4 compiles the different signals/features about the surrounding vehicles.

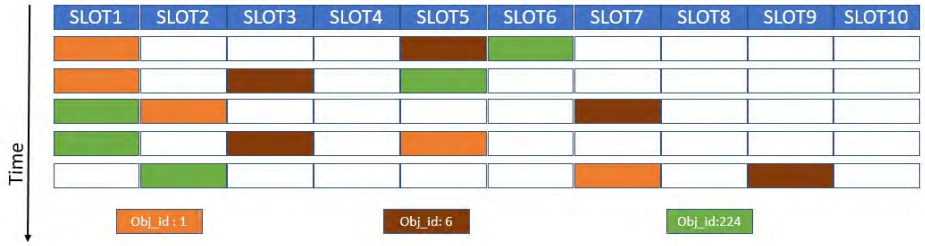



Figure 4.4.: Rotation of object (surrounding vehicle) information in the memory slots

Table 4.4.: Signal overview of the surrounding vehicles (object in the description refers to the surrounding vehicle)

| ***Signal*** <br> ***Unit*** | ***Description*** |
|---|---|
| Unique ID of the object | [-] |
| Y position of the object w.r.t test vehicle | [m] |
| X position of the object w.r.t test vehicle | [m] |
| Orientation of the object w.r.t to test vehicle | [rad] |
| History signal of the object | [-] |
| Object class, for example car or truck etc | [-] |
| Longitudinal speed of the object | [Km/h] |
| Lateral speed of the object | [Km/h] |
| Longitudinal acceleration of the object | [$m/s^2$] |
| Lateral acceleration of the object | [$m/s^2$] |
| Width of the object | [m] |

## 4.3. Comparison Study with Other Datasets

As observed in the literature review chapter, most of the researchers have used the Next Generation Simulation (NGSIM) (FHWA, n.d.) dataset for the data-driven development of car-following models. A comparison of the NGSIM dataset with the Audi xFCD dataset is given in Table 4.5. The Audi xFCD data set is the floating car data while the NGSIM dataset is the trajectory data. The Audi xFCD dataset is larger in number of hours of recording. Audi xFCD dataset is also not location dependent, as the data is recorded from the car. Since, the drivers are unaware about the recording purpose, the dataset captures naturalistic driving behaviour of different drivers in different traffic condition in the entire city. Contrarily in the NGSIM dataset, the data is captured using a set of cameras over a fixed stretch of freeway section. The number of lanes in the Audi xFCD datasets is not fixed as the car could drive in any region of the city and thus have road sections with 1 lane, 2 lanes and 3 lanes. Since the NGSIM dataset is recorded from the fixed section of road the lane number does not vary that much. There are various studies using the NGSIM dataset (Hao et al., 2018) (Zhao et al.,

2018) (D. Yang et al., 2019) (Zhou et al., 2017) (Mitra & Eric, 2018). But with the raw xFCD dataset, there are very few studies related to analysis or development of car-following model.

Table 4.5.: Comparison of NSGIM and Audi xFCD dataset.

| ***Features*** | ***NGSIM*** | ***Audi xFCD*** |
|---|---|---|
| Total duration | 1.5 hours | > 20 hours |
| Number of Locations | 2 | Location independent |
| Lanes per direction | 5-6 | 1-3 |
| Length of Road section | 0.5 - 1.0 Km | Driving data over the whole city, no fixed length of the road |
| Recording Frame Frequency | 10 Hz | 25 Hz |

## 4.4. Conclusion

The Audi xFCD has a large number of recordings of vehicles over different times and days of the week. This captures naturalistic driving behaviour in different parts of the city as well as on highways (freeways and highway dataset is not included in the study), which is significant to capture the realistic driving behaviour in car-following situations over all the city areas. Apart from this, very few studies use the raw xFCD dataset for the data-driven development of car-following models.

# 5. Methodology

This chapter presents the methodology to achieve the objectives of this study. The methodology consists of three modules, namely extraction of car-following scenarios from xFCD data, analysis of existing car-following models using xFCD data and finally the third is the development of a data-driven car-following model using xFCD data. A complete overview of the methodology is given in Figure 5.1.

## 5.1. Software and Tools

Python programming language is predominantly used in this study. Python is widely used in the scientific community because of its extensive collection of libraries for data processing, visualisation and computation. Apart from this, python's machine learning framework is quite mature and this allows it to be used widely in the scientific community. The list of the python libraries used in this study are given below:

- Visualisation and plotting : Matplotlib and Seaborn
- Computation: Numpy
- Data Handling and processing: Pandas
- Genetic Algorithm: Deap
- Deep Learning: Scikit learn and Keras (Tensorflow backend) with GPU extension.

The hardware used in this study is a Dell Workstation with i7 processor, 16 GB RAM and NVIDEA GeForce GTX 980 graphical processor unit (GPU).

## 5.2. Car-Following Data Extraction from xFCD

The methodology flowchart is shown in Figure 5.2. The discussions on the steps in the flowchart is explained in the following subsections.

### 5.2.1. Goal

The goal is to develop a methodology to extract the car-following traces from the raw xFCD. Previous researches have shown that machine learning techniques are good for classifying events like lane-change, cut-in etc. Car-following, on the other hand, is a complex scenario which comes under the scenario of free-following. In this study, a state machine technique is used to extract the car-following traces from the raw Extended Floating Car Data (xFCD).

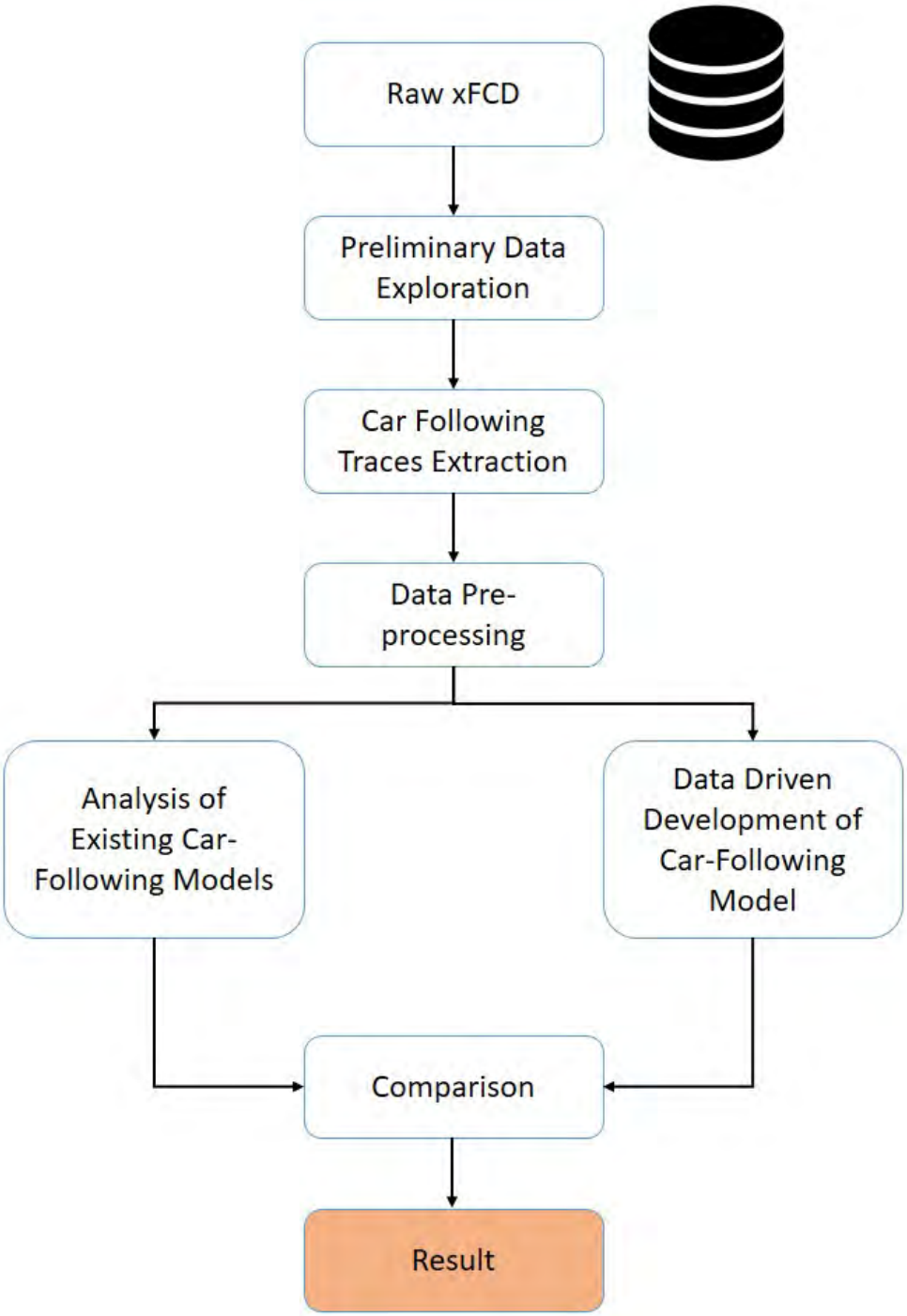



Figure 5.1.: Thesis Methodology.

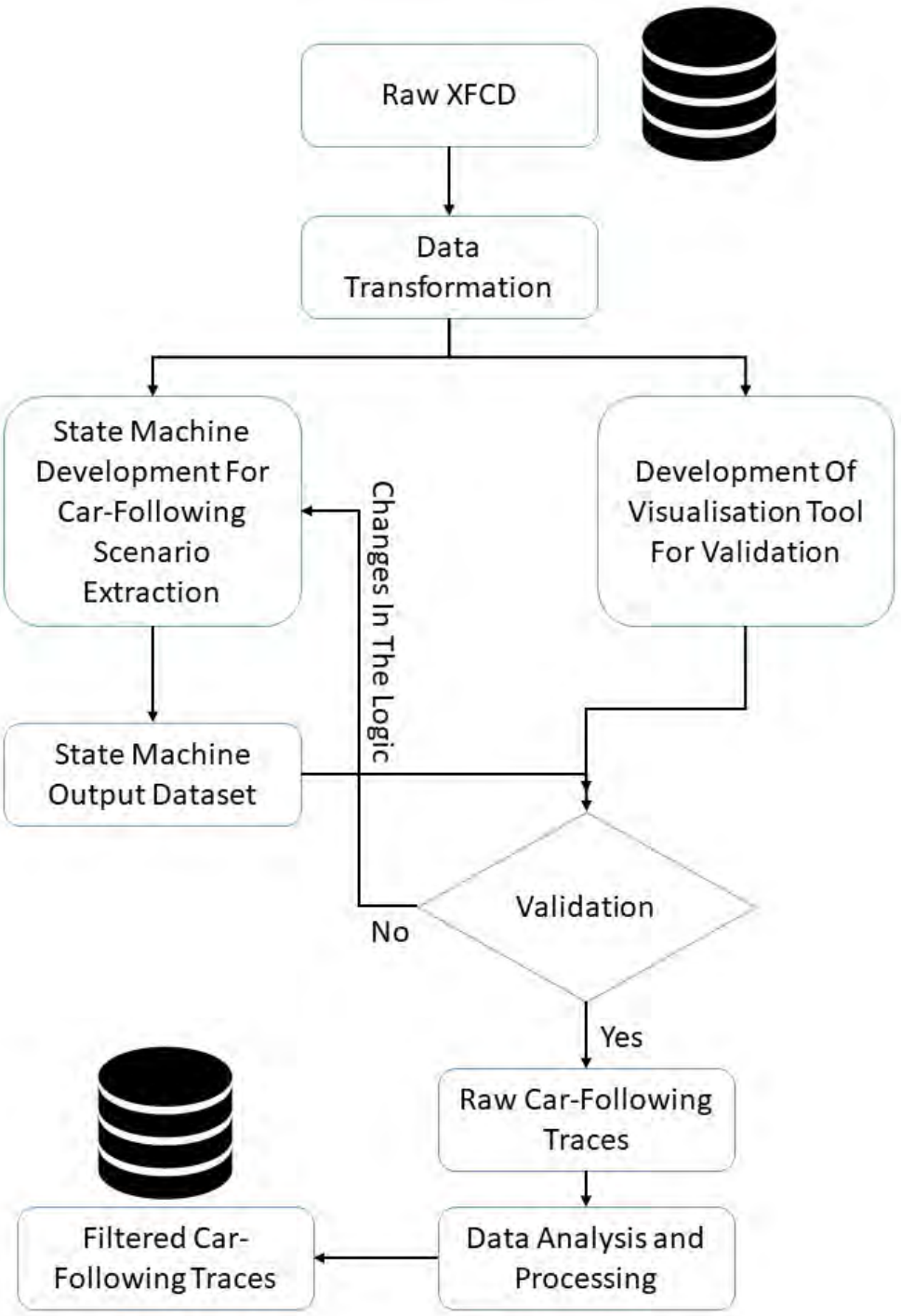



Figure 5.2.: Flow chart for car-following traces extraction

### 5.2.2. Preliminary Data Exploration

It is important to have an overview of the available dataset for the study. This step includes the initial data exploration. It is necessary to get an overview of the different signals (features/attributes) available in the raw Extended Floating Car Data (xFCD). Features/attributes or signals here means information like acceleration, velocity, lane information, surrounding vehicles related information etc. The different signals related to the test vehicle, environment and surrounding vehicles are analysed. As already explained in Data Collection chapter that many of the signal values are empty and not all the signals are important for this study. So, the raw xFCD data is transformed to a new definition which is explained in the next sub-section.

#### Data Transformation

The raw xFCD dataset is transformed to a new dataset. In this step, the raw xFCD data is divided into two separate csv files. One contains the information of the test vehicle and the environment and it is named "ego.csv", The other contains the information of the surrounding vehicles and it is named "surrounding_vehicle.csv". A new and simple naming convention of the signals is also adopted to transform the dataset, few examples of which are given in the Table 5.1.

Table 5.1.: Examples of the new naming convention of signals

| New Signal Name | Description | Units |
|---|---|---|
| ego_speed | Speed of the test vehicle | m/s |
| ego_accel_ x | Longitudinal acceleration of the test vehicle | $m/s^2$ |
| lat | latitude position of test vehicle | geo location |
| pos_x | X position of the object 1 | m |
| ... | ... | ... |

The flow-chart of the data transformation is shown in the Figure 5.3. The flow-chart executes for all the traces available in the raw xFCD data.

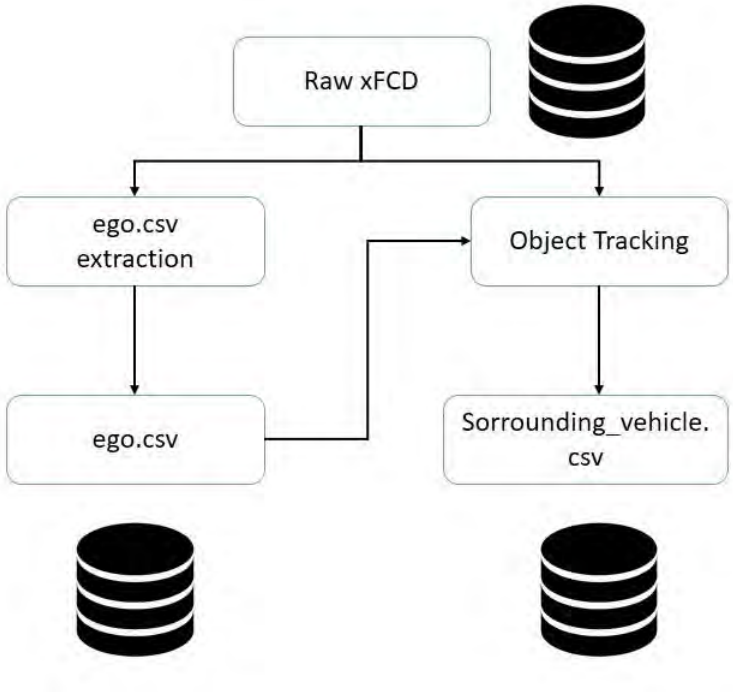



Figure 5.3.: Data transformation flow chart

**"ego.csv" Extraction** : The ego vehicle here refers to the test/following vehicles. A python script is developed to transform the dataset. Each trace of the raw xFCD data is passed to a python script where the attributes related to the test vehicle and environment are extracted. Each row of the raw xFCD data is considered as frame. For each frame, the signals listed in the Table 4.2 and Table 4.3 are separated and stored. The velocity signal is also transformed from km/h to m/s. The timestamp signal of the raw xFCD data is replaced by the frame number in the new dataset definition. A snippet of the ego.csv is shown in the Table 5.2.

Table 5.2.: A snippet of "ego.csv" file

| *ego_speed(m/s)* | *ego_accel_x(m/s$^2$)* | *lat* | *t_headway* | .... |
|---|---|---|---|---|
| 12.89 | 0.83 | 48.806232 | 2.46 | ... |
| 12.95 | 0.72 | 48.806232 | 2.461 | ... |
| 13.004 | 1.01 | 48.806232 | 2.457 | ... |
| 13.055 | 1.23 | 48.806232 | 2.452 | ... |
| 13.125 | 1.38 | 48.806232 | 2.445 | ... |
| ... | ... | ... | ... | ... |

**"surrounding_vehicle.csv" Extraction**: To extract the surrounding vehicles, a need for a

concrete methodology was realised. Since this study is focused on the car-following scenarios, the first and the foremost thing to construct the database for surrounding vehicles is to track them. The logic for storing the information of different surrounding vehicles information was already explained in subsection 4.2.1. To have any car-following vehicle scenario, it is decided that the surrounding vehicle must be tracked for a minimum duration of 10 seconds and only the information of these surrounding vehicles is stored. This helps in reducing the size of the database and contains only important data. The algorithm developed for object tracking is explained below.

**Object Tracking Algorithm**: The object tracking algorithm extracts data for each object that is either detected using the image processing technique or the sensor data fusion. The object here refers to the surrounding vehicles. The most important signals for the tracking algorithm are object ID and object history. With these signals in hand for each object, the following steps are executed to track the objects:

- Step 1: If the history signal is 0, save the associate object ID and the current slot (subsection 4.2.1 explains the slot and the ID). Then create one data object in which the information can be saved and fill it with the current information.
- Step 2: Iterate in the current slot until the object ID does not equal current ID and always save the information required for the recognized object.
- Step 3: In the next step, check whether current ID is found on another slot, if the current ID is not found in other slots then the algorithm terminates and returns the information collected about the recognized object.
- Step 4: If there is another slot in which the current ID is shifted and if the history signal is 1, then set the current slot to the slot in which the object is now located. then go to step 2 and repeat until the termination criteria as described in step 3 is met.

When evaluating the entire trace, iterate through it as well as through all available slots. This ultimately results in the measured value of all the objects detected during the entire journey, which is then stored in the surrounding_vehicle.csv file. Then the question arises that what information is important for the car-following study. Basically, all the attributes that can be detected by the sensors and thus all the information listed in the Table 4.4 is extracted and saved in surrounding_vehicle.csv file with the criteria of minimum tracking duration of 10 seconds which is determined by the number of frames the object was tracked. Since each frame corresponds to 25 ms so a minimum of 250 continuous frames is necessary for the object to be stored in the surrounding_vehicle.csv file. A snippet of the surrounding_vehicle.csv file is given in Table 5.3

Table 5.3.: A snippet of "sorrounding.csv" file

| *obj_id* | *speed_x(m/s)* | *accel_x(m/s²)* | *pos_x* | ... |
|---|---|---|---|---|
| 2 | 14.571 | 0.75 | 25.95 | ... |
| 2 | 14.559 | -0.56 | 26.00 | ... |
| 2 | 14.53 | -0.78 | 26.04 | ... |
| 2 | 14.50 | -0.92 | 26.09 | ... |
| 2 | 14.46 | -0.98 | 26.13 | ... |
| ... | ... | ... | ... | ... |

### 5.2.3. Development of the Visualisation Tool

As the video is not recorded while collecting the raw xFCD data, the need of a visualisation tool was realised to assess the quality of the data of the surrounding vehicles and also to validate the extracted car-following traces.

To develop the visualisation tool, the newly constructed database containing "ego.csv" and "surrounding_vehicle.csv" are considered. For each frame from ego.csv, the information of the test vehicle and the environment is plotted. To visualise the lane markings in the plot, the trajectory of the lane markings needs to be computed. Equation 5.1 is then used to calculate the trajectory using three signals: *ego_lane_left_distance_y*, *ego_lane_left_curvature* and *ego_lane_left_end_x*. The equation is given for the left lane marking and the same equation is used for the right lane markings.

$$Y = ego\_lane\_left\_distance\_y + ego\_lane\_left\_curvature * \frac{(x-2)^2}{2}, \quad for, x \subseteq [0, ego\_lane\_left\_end\_x] \tag{5.1}$$

The visualisation tool plots the information frame by frame. This information is important to plot the objects (surrounding vehicles). For each frame, the "surrounding_vehicle.csv" file is checked and if there is any object tracked for the current frame, the particular object is plotted in the visualiser with its x and y positions. Figure 5.4 shows the visualisation tool example, the unit of x-axis and y-axis in the tool is in meter (m), and the centre of the ego vehicle is located at the origin (0,0).

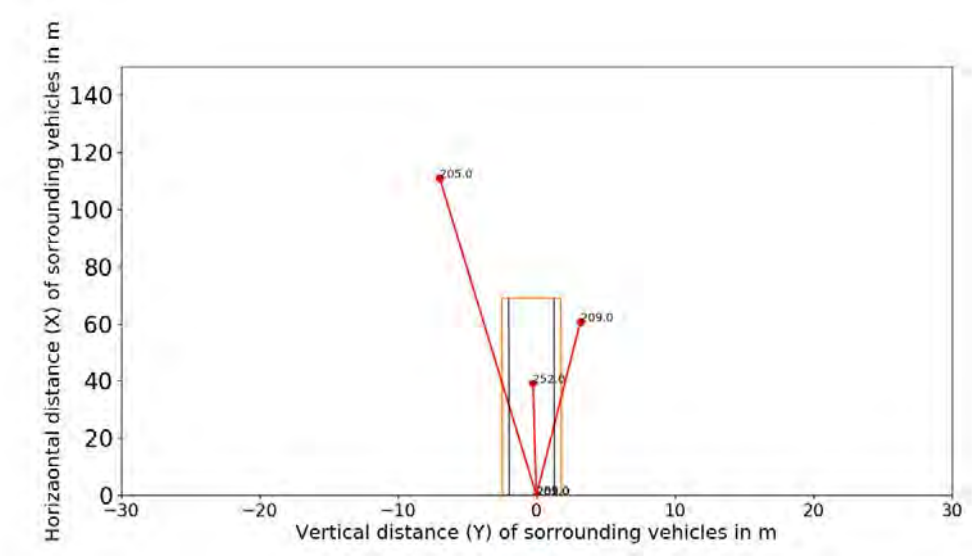



Figure 5.4.: Visualisation of the raw xFCD

During the visualisation of raw data, several problems were encountered. Capacity of the sensor hardware, instabilities in the algorithms, noise interference or different sensor steps may lead to these problems in the recorded dataset. The main problem encountered during the visualisation is of the surrounding vehicles. The position signals especially has lot of instabilities. The jump between the x and y location for two consecutive frame was unrealistic and irregular. This problem may lead to severe problem in car-following trace extraction. Apart from this, the other problem is related to the missing values, like the latitude and longitude signals of the test vehicle have the sample rate of 5 Hz and on the other hand the sample rate of other signal is 25Hz. So, the position signals of the test vehicles were missing in between.

### 5.2.4. Data Pre-Processing

As described in the previous section, two major problems were found during data visualisation. Hence, mathematical methods are applied to clean the dataset. In this section, the two mathematical processes used for data pre-processing are discussed: Smoothing and Interpolation.

#### Smoothing

The location signal of surrounding vehicle, i.e. the x and y position data of the surrounding vehicle have high level of noise. This noise can be considered as white noise because it is mostly caused by disturbances of the environment, instabilities of hardware and algorithms which extract this data either from images or from the sensor fused data. A technique called smoothing is applied to recover the reliable data for the surrounding vehicles. A 1D-Gaussian

filter with adjustable window length is applied to prevent the shift after filtering. This helps in effective noise reduction. The formula of the Gaussian filter (Steger, 1996) is:

$$G(x) = (1/\sqrt{2\pi}\sigma)e^{-x^2/2\sigma^2}\,, \tag{5.2}$$

where $\mu$ is the mean and $\sigma$ is the standard deviation.

For each frame, x and y location data of the surrounding vehicle is assigned a new filtered value which is a function of the original value of that frame and the surrounding frames. The Gaussian kernel allows the frame at the centre of the filter window to have higher weight than those at the periphery. Boundary effects at the beginning and end of the output signal are minimised by introducing a reflected window length.

**Interpolation**

To mitigate the problem of intermediate missing values in the dataset, especially for the location of the following vehicle. Linear interpolation technique is used. It fills new values for the frames where the signal value is missing using the linear polynomials. Equation 5.3 represents the formula of linear interpolation, which interpolate the value $y_1$ of point $x_1$ located between the known points $(x_0, y_0)$ and $(x_2, y_2)$

$$y_1 = y_0 + \frac{(y_2 - y_0)}{(x_2 - x_0)} * (x_1 - x_0)\,, \tag{5.3}$$

### 5.2.5. Development of State Machine

The state machine for the extraction of car-following traces is developed. The flowchart of the state machine is given in the Figure 5.5. The explanation of the flow-chart is summarised in the following step.

- Step 1: Load the database containing the "ego.csv" and the "surrounding_vehicle.csv".
- Step 2: Group the "surrounding_vehicle.csv" by the object IDs using the python pandas function called "groupby". This function essentially divides the whole data into smaller data-frames of each object corresponding to the unique object IDs.
- Step 3: For each object data-frame, extract the initial frame number which corresponds the frame number when it was recognised first time by the test vehicle. Apart from this, also extract the last frame number which corresponds to the last frame of its tracking.
- Step 4: Separate out the information from the "ego.csv" files lying between the initial frame and the last frame obtained in step 3.
- Step 5: Set the initial state from the first frame of the object data-frame as "Not Following".

- Step 6: For each frame, create a bounding box using the lane information signals. The trajectory of the left lane marking and the right lane marking is calculated using Equation 5.1 and with the calculated trajectory a polygon is created which is used as bounding box. Note, the length of the bounding box is considered as 60 meters and 100 meters. The 60 meter is considered when the vehicle is driving in the urban environment and 100 meters is considered when the vehicle is driving in the highways within the study area.
- Step 7: Check whether the position data of object, i.e. X and Y are lying inside the bounding box for each frame.
- Step 8: If "Yes", set the state of the particular frame as "Car-Following", and go to step 6 and 7 again to check for the next frame until all the frames of the particular object are iterated through.
- Step 9: If "No", set the state of the particular frame as "Not Following". Then go to step 6 and 7 and check for the next frame. Also, if the state has already been triggered once from "Not Following" to "Car-Following", then count the number of continuous frames for which the algorithm says "Not Following". If the continuous counting of the frame exceeds the number 25, which corresponds to 1 second of data, then do nothing, else, if the continuous number is less than 25, set the state of all these frames as "Car-Following".
- Step 10: Go to step 2 for other object IDs in the data set, if all the data-frames of different objects are processed. Go to step 1 to load a new database.

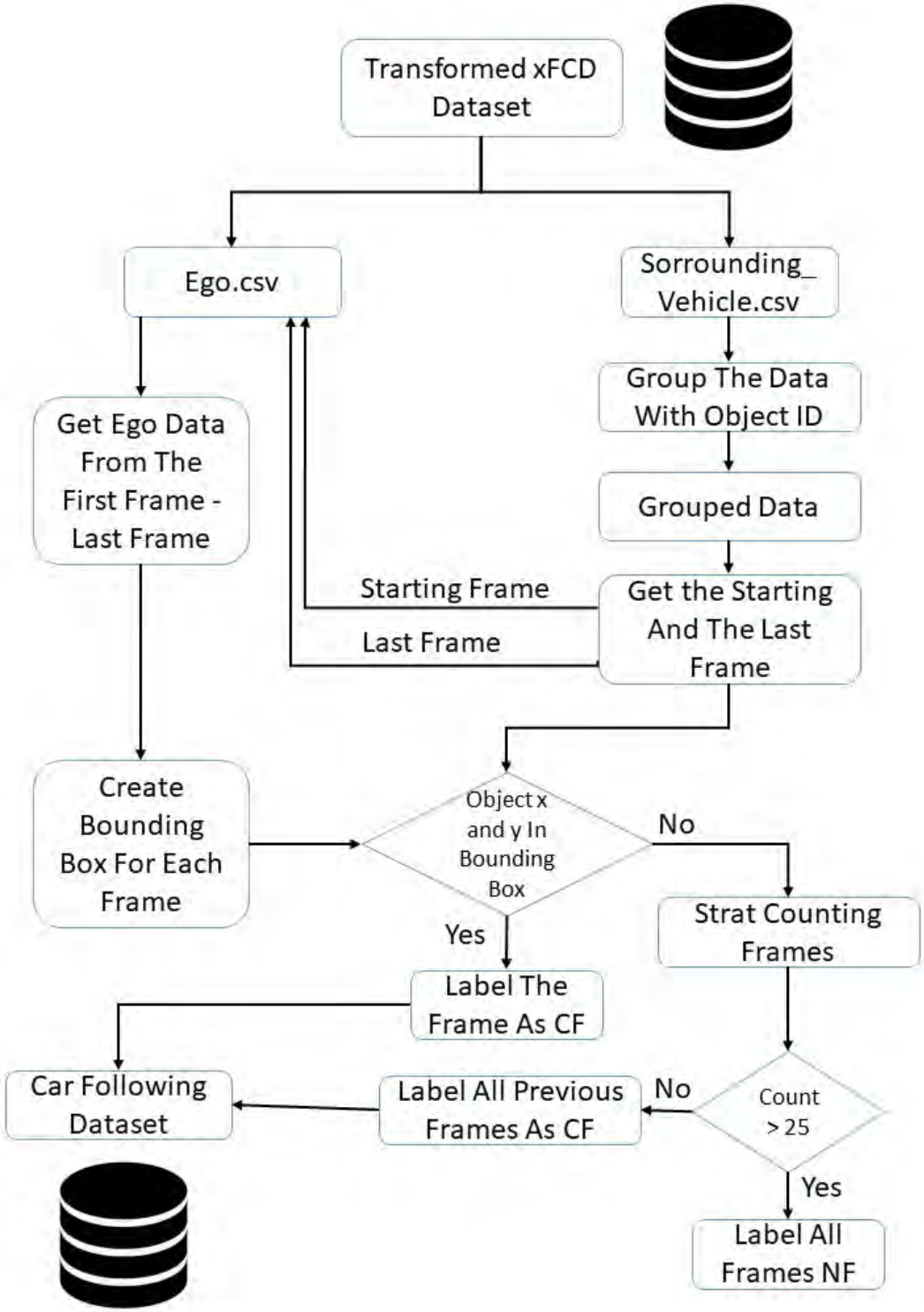



Figure 5.5.: Flow chart of the state machine development to extract the car-following traces from raw xFCD data.

### 5.2.6. Data Analysis and Processing

This step analyses and processes the different signals extracted in the car-following trajectories. Apart from the analysis, data post-processing is also done. The velocity signal and the acceleration signal of the test vehicle are analysed and in case of any anomalies, the corresponding signals are smoothed using a Gaussian filter (explained in section 5.2.4). Similarly, the acceleration signal and velocity signal of the leading vehicle are analysed and smoothed and in last if found necessary the gap between the vehicles, which is the x-distance of the leading vehicle is filtered and smoothed.

## 5.3. Analysis of the Existing Car-Following Models Using xFCD

The method flowchart is shown in Figure 5.6. The subsequent subsections explains each part for the flowchart.

### 5.3.1. Goal

The goal here is to evaluate the performance of the existing car-following models on the extracted car-following traces. The selected exisiting models in this study are, the Krauss model, the Wiedemann model and the IDM, because the Krauss model is the default model of SUMO traffic simulation software and the Wiedemann model is the default car-following model of PTV VISSIM traffic simulator and the IDM model performed best on calibration as per (Mitra & Eric, 2018).

### 5.3.2. SUMO Network Development for the Different Car-Following Trajectories

The extracted car-following trajectories are from all the regions of Ingolstadt city. In order to run the simulation for these trajectories, an algorithm to create a SUMO network for each trajectory is developed.

The Figure 5.7 shows the flow-chart used to develop the SUMO network and run the simulation for each trace using the developed SUMO network. A python script is developed to create the SUMO network and run the SUMO simulations autonomously. The python script is run with two arguments, first is the address of car-following trace and second is the name of the car-following model. Then, this argument information is passed to another developed python class to create the SUMO network. This python class firstly extracts the geo-locations of the following and the leading vehicle from the car-following trace. Then the first geo-position of the following vehicle is considered as the starting node position of the network and the last geo-position of the leading vehicle is considered as the last node position for the network. In this way the node file is defined for the creation of the SUMO network. Then using all the geo-locations of following and the leading vehicle, the shape of the edge connecting the starting and the ending node is computed. This gives the edge file needed to create the SUMO network. In last, the node file and the edge file are used to create the SUMO

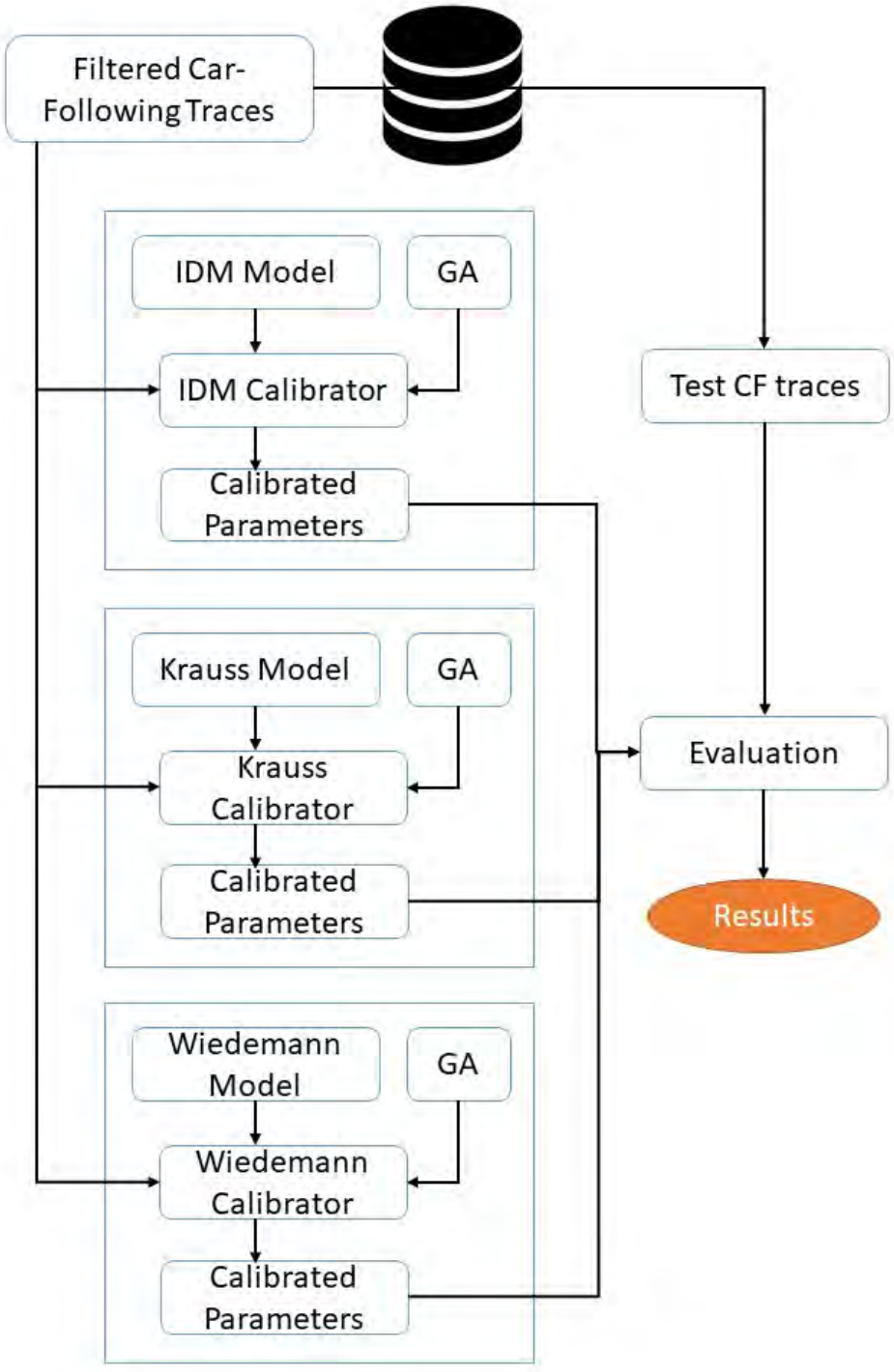



Figure 5.6.: Flow chart of the methodology of conventional car-following models analysis.

network using the SUMO inbuilt program called "netconvert". Then, the developed network is used to create the route file for the leading and the following vehicle. Finally, the SUMO configuration file is generated to run the simulation using the created network file generated and the route file. To run the simulations for the car-following traces, another python script is developed. This python script runs the SUMO simulation with "TraCI". TraCI stands for Traffic Control Interface. TraCI allows the user to interact with the SUMO simulation using an external controller. The external controller here is the python script. In the beginning of the simulation both the leading and the following vehicle start with zero velocities. The python script initially gets the velocity of the leading vehicle from the simulation for each simulation step and compares it with the leading vehicle velocity from the first frame of the car-following trace. Once the velocity becomes equal, the python script takes the control over the simulation. The python script sets the location of the following and leading vehicle at the start of the network with the gap taken from the first frame of the car-following trace. The simulation then gets the velocity of the leading vehicle from the python script using TraCI interface and the simulation returns the velocity of the following vehicle to the python script for each simulation step. The obtained velocity of the following vehicle is then saved and in this way, the behaviour of the following vehicle is recorded.

The approach when tested worked fine. This approach of simulation has one problem. When analysing all the traces of the car-following, this approach takes a lot of time to simulate. Apart from this, the later part of the analysis is the calibration of parameters. The calibration is an iterative task and calibrating each parameter of each model will take a lot of time with this approach. So, instead of using this approach a numerical simulation of each model is considered. To realise the a numerical simulation, each of the car-following models, namely the Krauss model, the Wiedemann model and the IDM are developed using python language.

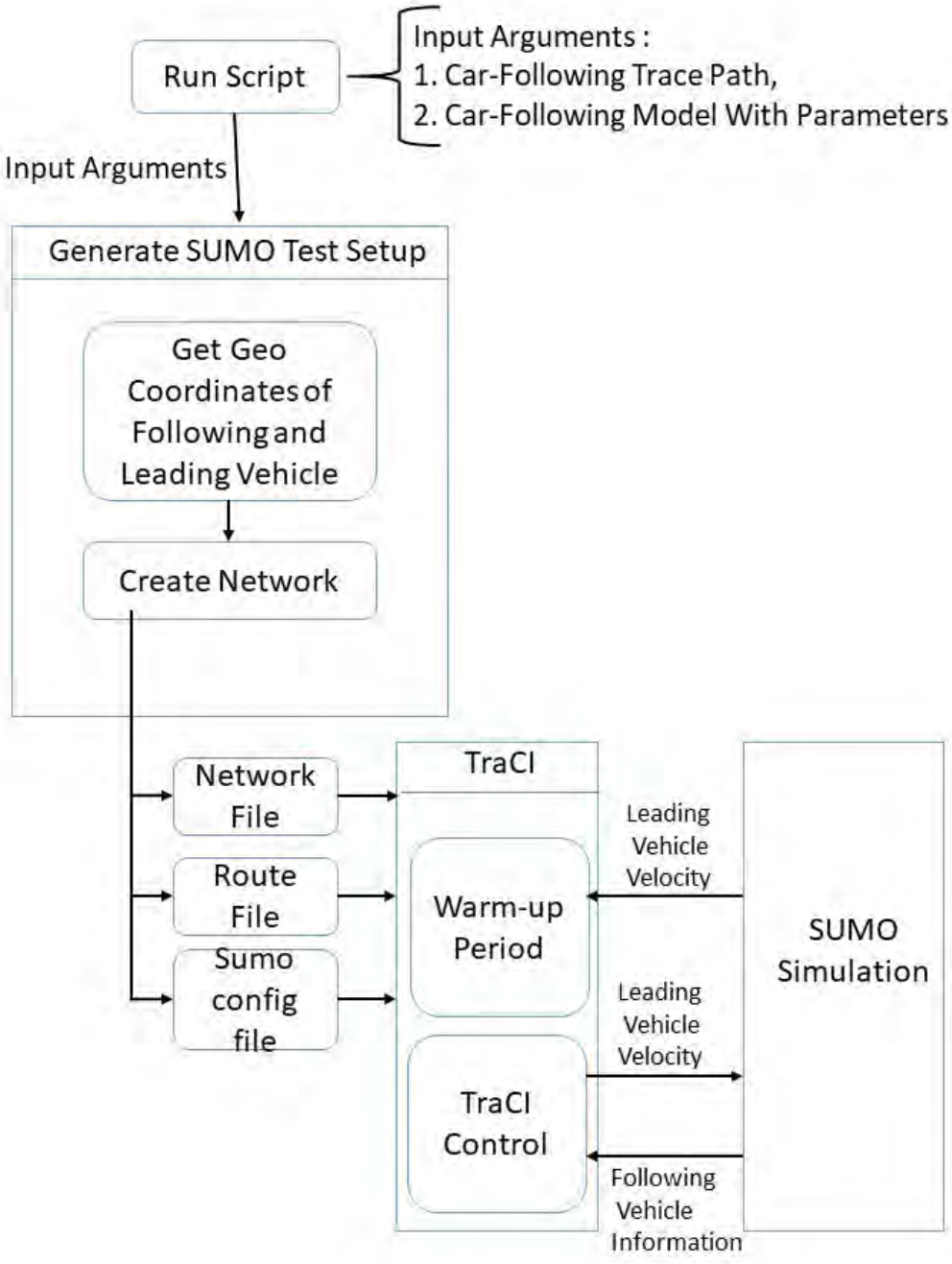



Figure 5.7.: Flow chart for the development and simulation of SUMO network from the car following trace.

### 5.3.3. Development of the Car-Following Models in Python

To run the numerical simulations of the three car-following models, python script for each model is developed. This saves a lot of time to run the simulation for each trace. The SUMO simulation takes more time than the numerical simulation using the python script for each model for each trace. Apart from this, in SUMO simulation, to simulate each trace, the warm-up time of the simulation also needs to be considered. Contrarily this is not needed in the numerical simulation. For the development of IDM and the Krauss model, the equations explained in the chapter 2 for each of the model are used. To develop the Wiedemann model script, the flow chart in the Appendix A is used.

### 5.3.4. Calibration of Car-Following Models

All car-following models have certain set of parameters, these parameters play an essential role in the performance of the model. Every model has default values for its parameter set. These default values when used in the simulation, the model shows the generic behaviour, but to analyse the performance of car-following models on the real world data, calibration of the car-following models is important. It means for every car-following trace, an optimised set of parameters for each car-following model needs to be calculated. Typical calibration technique includes manual methods, derivative based optimisation algorithms, metaheuristics approaches, and other techniques (Lidbe et al., 2017).

Manual methods are based on trial and error. Judgemental values of each parameter and feasible combination of multiple parameters are taken during each trial and the corresponding results are analysed for the error between the simulated output and the observed values. The derivative based, also called gradient based algorithms use the information of gradients. They are efficient as local search algorithms, but have disadvantages of being trapped in a local optimum if the optimisation problem is not convex (X. S. Yang, 2011). The objective function must be sufficiently smooth, i.e. its first and second derivative must exist to apply the gradient based optimisation algorithms. Opposite of derivative based algorithms are derivative free algorithms. In case of discontinuities in the objective function, they perform better than their counterpart. One drawback of these algorithms is that these algorithms are deterministic. They usually have disadvantages in dealing with highly non-linear global optimisation problems (X. S. Yang, 2011). Metaheuristic algorithms are one of the most powerful optimisation techniques. They are often inspired by the nature and are suitable for non-linear global optimisation problems. Many metaheuristic algorithms implement some kind of stochastic optimization, so that the final solution found is dependent on the randomly generated values. Metaheuristic algorithms search the solution over a large set of feasible solutions and thus, can often result good solutions with less computational effort than optimization algorithms or iterative methods. A Genetic algorithm is one of the most widely used metaheuristic algorithm in car-following model calibration (Mitra & Eric, 2018), (Zaky et al., 2016), (Lidbe et al., 2017). This offers the reliability of using it in this study.

Basics of the genetic algorithm were explained in section 2.3. A genetic algorithm can be classified as binary-coded genetic algorithms and real-coded genetic algorithms. Binary-coded genetic algorithms have shown good results when used in car-following models calibration (Zaky et al., 2016), (Lidbe et al., 2017). Binary coded genetic algorithms have discrete search space. The performance of the algorithm largely depends on the coding used to represent the problem variables and on the crossover operation which is very essential to create the children strings (children solutions) from the parent strings. To solve optimisation problems with continuous search space, binary-coded genetic algorithms discretize the search space by encoding the problem variables in binary strings. These binary strings are of finite lengths. Coding the real valued variables in finite length strings causes a number of difficulties: fixed mapping of problem variables, inability to achieve arbitrary precision in the obtained solution, inherent cliff problem because of binary coding (Deb & Bhushan Agrawal, 1995). Since, the parameters of the car-following models can take continuous values, real-coded genetic algorithm is adopted in this study. The approach to define the population, mutation, crossover and selection operation used in this study are explained below.

**Population Generation**

The IDM, the Wiedemann model and the Krauss model have a different set of parameters. These parameters can take the continuous values between the upper and the lower bound defined for each parameter. To realise the real-coded genetic algorithm, the upper bound and the lower bound of each model's parameters are defined. The method used to define the initial population of parameters for each model is defined below.

**Intelligent Driver Model:** To calibrate the IDM, maximum acceleration $a$ in ($m/s^2$), maximum comfortable deceleration $b$ in ($m/s^2$), gap at standstill $s_0$ in (m), time headway $T$ in (s) and acceleration exponent $\delta$ are chosen for the optimisation. The upper and lower bounds of the parameters are selected from the (Mitra & Eric, 2018) which are shown in Table 5.4.

Table 5.4.: Upper bound and lower bound values of IDM parameters

| *Parameters* | *Lower Bound* | *Upper Bound* |
|---|---|---|
| $T$ (s) | 0.7 | 3 |
| $a$ ($m/s^2$) | 0.1 | 5 |
| $b$ ($m/s^2$) | 0.1 | 5 |
| $s_0$ (m) | 0.5 | 3 |
| ($\delta$) | 3 | 5 |

To generate the population of the solutions, an individual needs to be defined first. The individual solution for the IDM is a set of the given five parameters which can be realised as [$a$, $b$, $s_0$, $T$, $\delta$]. To create this individual, $a$, $b$, $s_0$, $T$ and $\delta$ need to be defined. Since,

real-coded genetic algorithm is used in this study. Value for each parameters is chosen from a uniform distribution of the values between the upper and the lower bound. The probability distribution of the uniform distribution is defined as:

$$p(x) = \begin{cases} 1/(b-a), & \text{if } \forall x \in [a,b) \\ 0 & otherwise \end{cases} \tag{5.4}$$

where, $b$ is the upper bound and $a$ is the lower bound.

After the creation of an individual, the population of the solutions is created by an iterative process and the number of iterations are defined by the population size. The size of the population is a user defined variable. The population size of 100 is used in this study. So, 100 individual solutions are created using the above defined approach of individual definition to create an initial population of solutions.

**Wiedemann Model:** All 10 parameters of the Wiedemann model, namely standstill gap CC0 (m), time headway CC1 (s), following variation CC2 (m), threshold entering following CC3 (s), negative following threshold CC4 (m/s), positive following threshold CC5 (m/s), speed dependency of oscillations CC6 (rad/s), oscillation acceleration CC7 ($m/s^2$), standstill acceleration CC8 ($m/s^2$) and acceleration at 80 km/h CC9 ($m/s^2$) are calibrated in this study. The process of creation of an individual and the creation of initial population is same as explained for IDM population creation. The upper bound and the lower bound values of the parameters are shown in Table 5.5.

Table 5.5.: Upper bound and lower bound values of the Wiedemann model parameters

| ***Parameters*** | ***Lower Bound*** | ***Upper Bound*** |
|---|---|---|
| *CC0* (m) | 0 | 5 |
| *CC1* (s) | 0.5 | 3 |
| *CC2* (m) | 0 | 25 |
| *CC3* (s) | -20 | 0 |
| *CC4* (m/s) | -1.5 | 0 |
| *CC5* (m/s) | 0 | 5 |
| *CC6* (rad/s) | 0 | 20 |
| *CC7* ($m/s^2$) | 0 | 1 |
| *CC8* ($m/s^2$) | 0 | 8 |
| *CC9* ($m/s^2$) | 0 | 8 |

**Krauss Model:** The parameters maximum acceleration $a$ in $m/s^2$, maximum comfortable deceleration $b$ in $m/s^2$ and reaction time $\tau$ of the driver in s are chosen for the calibration of the Krauss model. The process of creation of an individual and the creation of initial

population is same as explained for IDM population creation. The upper bound and the lower bound values of the parameters are adopted from (Mitra & Eric, 2018) which are shown in Table 5.6.

Table 5.6.: Upper bound and lower bound values of Krauss model parameters

| ***Parameters*** | ***Lower Bound*** | ***Upper Bound*** |
|---|---|---|
| $a$ (m/s$^2$) | 0.01 | 5 |
| $b$ (m/s$^2$) | 0.01 | 5 |
| $\tau$ (s) | 0.2 | 3 |

**Fitness Function Definition**

Fitness function is an essential part of a genetic algorithm. The fitness function plays an important role in the process of selection of an individual solution for the next generation of solutions in a genetic algorithm. To calibrate a car-following model, the fitness function acts as an objective function of the optimisation problem which either can be minimized or maximized by an optimisation algorithm. In this study the Root Mean Square Percentage Error (RMSPE) is used to define the fitness function. The RMSPE is defined as:

$$RMSPE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{S}_i - S_i^o)^2}{\sum_{i=1}^{n}(S_i^o)^2}}\ , \tag{5.5}$$

where, $\hat{S}_i$ is the simulated value, $S_i^o$ is the observed value and $n$ represents the number of observations.

The RMSPE between the simulated and the observed velocity of the following vehicle and the RMSPE between the simulated and the observed gap between the leading and the following vehicle are used. Fifty percent weight to both of the RMSPEs is given and the genetic algorithm is then designed to minimize these error. The equation used for the defined fitness function is given below.

$$Fitness = 0.5 * RMSPE_{\text{v-follow}} + 0.5 * RMSPE_{\text{gap}}\ , \tag{5.6}$$

where $RMSPE_{v\text{-}follow}$ is the RMSPE between the simulated and observed velocity of the following vehicle and $RMSPE_{gap}$ is the RMSPE between the simulated and observed gap between the leading and following vehicle.

**Selection**

Selection is a step in a genetic algorithm where the individual solutions are chosen from the population for the next steps of the genetic algorithm. Selection operation improves the

quality of the next generation population by giving higher probability to individuals of higher quality from previous population. The quality of an individual is measured by a fitness function. The selection operator can be represented mathematically as:

$$s : P_{\text{old}} -> P_{\text{new}}, \tag{5.7}$$

where the $P_{old}$ represents the population before selection and $P_{new}$ represents the population after selection.

In this study the tournament selection algorithm with elitism is used. In tournament selection algorithm, the size of the tournament (t) is defined first, then in an iterative process for $n$ times, where $n$ is the size of the population, (t) individuals are selected from the previous population $P_{old}$ and the best individual amongst them is copied to the next generation and hence creates a new population $P_{new}$. But this process does not guarantee the survival of the best individual through the selection process. So, an elitism strategy is adopted which guarantees the preservation of the fittest individual of the population. With elitism, the best individuals of the old population are copied directly to the new generation.

**Crossover**

The crossover operator is the main search operator in the working of a genetic algorithm (Deb & Bhushan Agrawal, 1995). The purpose of the crossover operator is to combine the good portion of parent solutions to create better children solutions for the next generation of the population. For binary-coded genetic algorithms, there are single point crossover and multiple point crossover. In single point crossover, a random cross-site along the length of the string of the parents solution is chosen and the bits on one side of the cross site of one parent are swapped with the other side of the cross site of another parent (Deb & Bhushan Agrawal, 1995) to produce the children solutions. In multiple point crossover, the cross-site are more than one.

In this study, a real-coded genetic algorithm is used. So, instead of single point or multiple point binary coded crossover operators, the simulated binary crossover operator is used given by (Deb & Bhushan Agrawal, 1995). The simulated binary crossover operator uses a probability distribution, where a large probability is assigned to a point close to the parent solution and small probability to a point away from the parent solution. The probability distribution is controlled by the user defined parameter called crowding degree $\eta$ of crossover. Simulated binary crossover is implemented parameter wise, that is, for parameter A in range ([lowerbound, upperbound]), two parent values of that parameter $Parent_1$ and $Parent_2$ are recombined to create the two children solutions $\text{child}_1$ and $\text{child}_2$ as follows (Deb, 2012):

$$child_1 = \frac{Parent_1 + Parent_2}{2} - \beta_L \frac{Parent_2 - Parent_1}{2}, \tag{5.8}$$

$$child_2 = \frac{Parent_1 + Parent_2}{2} + \beta_R \frac{Parent_2 - Parent_1}{2}, \tag{5.9}$$

where the parameters $\beta_{\mathrm{L}}$ and $\beta_{\mathrm{R}}$ depend on the random numbers $u_L$ and $u_R$ both in the range [0,1], and are defined as:

$$\beta_L = \begin{cases} (2u_L(1-\alpha_L))^{1/(1+\eta)}, & \text{for } 0 \leqslant u_L \leqslant 0.5/(1-\alpha_L)\ , \\ \frac{1}{2(1-u_L(1-\alpha_L))^{1/(1+\eta)}}, & \text{for } 0.5/(1-\alpha_L) < u_L < 1 \end{cases} \tag{5.10}$$

$$\beta_R = \begin{cases} (2u_R(1-\alpha_R))^{1/(1+\eta)}, & \text{for } 0 \leqslant u_R \leqslant 0.5/(1-\alpha_R)\ , \\ \frac{1}{2(1-u_R(1-\alpha_R))^{1/(1+\eta)}}, & \text{for } 0.5/(1-\alpha_R) < u_R < 1 \end{cases} \tag{5.11}$$

where, $\alpha_L$ and $\alpha_R$ are defined as,

$$\alpha_L = \frac{0.5}{(1+2(Parent_1 - lowerbound)/(Parent_2 - Parent_1))^{(\eta+1)}}\ , \tag{5.12}$$

$$\alpha_L = \frac{0.5}{(1+2(upperbound - Parent_1)/(Parent_2 - Parent_1))^{(\eta+1)}}\ , \tag{5.13}$$

The above equations ensure that the $child_1$ and $child_2$ does not lie outside the parameter's bounded values ([lowerbound, upperbound]).

**Mutation**

Mutation operation in a genetic algorithm primarily maintains the diversity in the population (Deb & ayan Deb, 2014). Mutation operates on one individual of the evolving population at a time and modifies it independent to the other members. In this study, the polynomial mutation operator suggested by (Dobnikar et al., 1999) is used. The polynomial function similar to simulated binary crossover has a user defined parameter called spread factor $\eta$.

A polynomial probability distribution is defined which is used to perturb a solution in a parent solution vicinity. The probability distribution in both left and right of a parameter value is adjusted so that no value outside the specified range [lowerbound, upperbound] is created by the mutation operator. For a given parent solution $A \in$ *[lowerbound, upperbound]*, the mutated solution $\mathrm{A_{mutated}}$ is created for a random number $u$ in range [0,1] as (Deb & ayan Deb, 2014):

$$A_{\mathrm{mutated}} = \begin{cases} A + \delta_L(A - lowerbound), & \text{for } u \leqslant 0.5 \\ A + \delta_R(upperbound - A), & \text{for } u > 0.5\ , \end{cases} \tag{5.14}$$

where the parameters ($\delta_{\mathrm{L}}$,$\delta_{\mathrm{R}}$) are calculated as:

$$\delta_L = 2u^{1/(1+\eta)} - 1\ , \text{for}\ \ u \leqslant 0.5, \tag{5.15}$$

$$\delta_R = 1 - (2(1-u))^{1/(1+\eta)}\ , \text{for}\ \ u > 0.5. \tag{5.16}$$

In this study, the parameter $\eta$ is chosen 10.

### 5.3.5. Evaluation

To evaluate the calibrated models, models with the optimum parameters are simulated on 10 different car-following trajectories. The performance of each model is evaluated by calculating the model's accuracy in replicating four different profiles of the following vehicle, namely:

- Acceleration
- Velocity
- Gap between the following and leading vehicle
- Trajectory of the following vehicle

The accuracy is evaluated using the Root Mean Percentage Square Error (RMSPE), as defined in Equation 5.7.

## 5.4. Data Driven Development of Car-Following Model

In this section, the approach to select the machine learning model to develop the data-driven car-following model is discussed, along with the dataset preparation for the selected model. The methodology flowchart is shown in figure Figure 5.8

### 5.4.1. Goal

The goal is to develop a data-driven car-following model, using xFCD. The method to develop the model is based on the objectives of this study, that is to find the driver's memory impact in the car-following behaviour. Apart from this, the impact of the features other than the velocity difference and the gap between the leading and following vehicles.

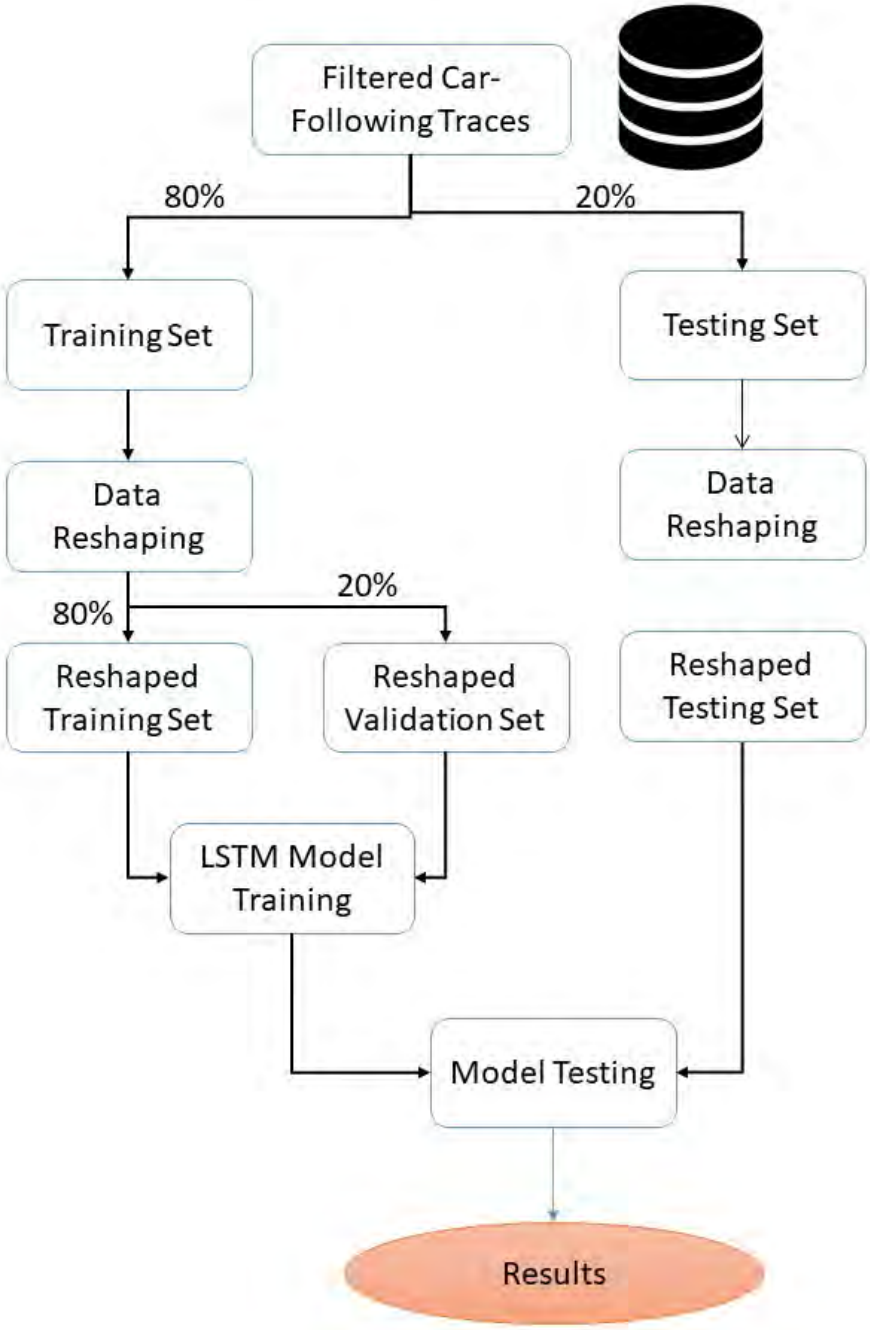



Figure 5.8.: Flow chart for Data-Driven Car-Following Development

### 5.4.2. Model Development

The car-following behaviour of a driver can be considered as a combination of time series of acceleration, velocity, gap between leading and following vehicle etc. So the problem of estimation of car-following parameters, like velocity or acceleration of the following vehicle can be modelled as time series prediction problem. Since the predicted parameter, like acceleration or velocity of the following vehicle, can take continuous real values, the car-following model is formulated as time series prediction problem, in which the past values of the time series along with explanatory time series (gap, time headway etc.) past values are used to predict the next timestamp value of car-following parameter.

The RNN or LSTM are suitable for the sequence prediction problem. Further, LSTM is capable of learning long-term dependencies. Therefore, LSTM is used as the primary model to realise the car-following model in this study. The main parameters of the LSTM neural network are as follows:

- Number of LSTM layers
- Number of memory cells in each LSTM layer
- Learning rate
- Optimizer
- Cost function

The basic architecture of the neural network used in this study is shown in Figure 5.9. The first layer is the convolutional layer, followed by the deep architecture of LSTM layers. The final output layer is a dense layer with one output which is the predicted parameter of car-following model. To achieve the best combination of the number of layers and the number of LSTM cells in each layer, multiple combinations are tested.

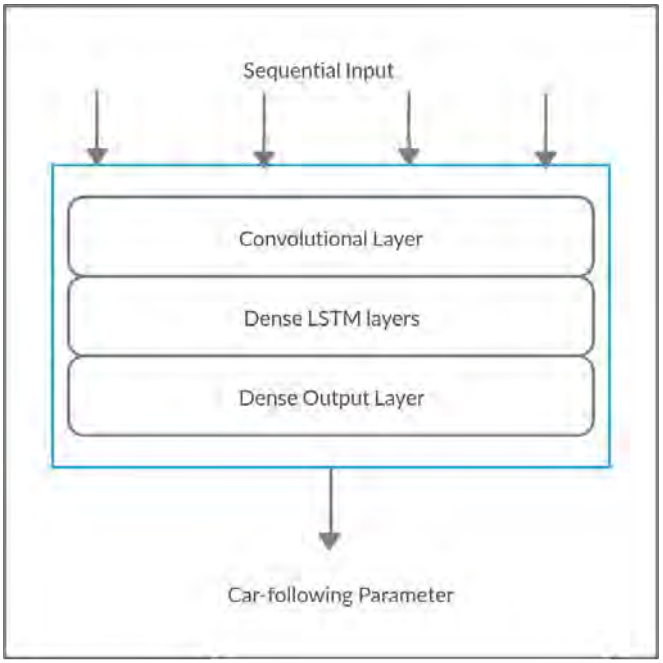



Figure 5.9.: Basic architecture of LSTM neural network used in this study

Learning rate of the neural network determines the step-size while changing the weights of the model parameters based on the value of cost function. A large learning rate might not lead to the discovery of local minima, whereas a small learning rate might lead to a very slow convergence. In this study, the learning rate of the model is optimized by using the ADAM (Kingma & Ba, 2015) optimiser. The ADAM optimiser is an adaptive learning rate optimisation algorithm. Its name is derived from adaptive moment estimation. ADAM uses estimations of first and second moments of a gradient to adapt the learning rate for each weight of the neural network. Where $N^{th}$ moment of a random variable is defined as the expected value of that variable to the power of n (Kingma & Ba, 2015).

A cost function measures how good the neural network trained with respect to the expected output. It can be defined as:

$$C(W, b) = (\hat{Y} - Y)^2 \,, \tag{5.17}$$

where $\hat{Y}$ denoted the predicted value and the Y represents the observed value. Since the predicted value in this study is a continuous real value, the cost function in this case is called the mean squared error (MSE), which helps in learning the weights of the model through back-propagation.

### 5.4.3. Data Preparation

The LSTM model learns the sequential data. The input data to the LSTM neural network is reshaped in a way that the neural network understands. Before the reshaping of the data, the data needs to be divided into a test and a training set. The training set is the data on which the model learns and the testing data set is the unseen data to the model on which the performance of the model is evaluated. There are 1453 extracted car-following traces. These 1453 traces are divided into training and testing traces with the ratio of 80-20. That is, eighty percent of the traces are used to build the training dataset and the remaining twenty percent of these traces are used to test the model performance.

The traces of the training dataset are then concatenated and further divided into training and the validation set in the ratio of 80-20. Before reshaping the dataset, the training dataset is standardized. Standardizing of the dataset involves the re-scaling of the distribution of different feature values of the dataset, so that the mean of the feature value is zero and the standard deviation is 1. Feature here refers to the signals/attributes that the data have, like acceleration, velocity etc. Standardization assumes that the values of the features fit a Gaussian distribution. The formula used to standardize the dataset is:

$$Y = (X - \mu_X)/\sigma_X \,, \tag{5.18}$$

where Y is the standardize feature, X is the feature input to be standardize and $\mu_X$ is the mean of the X and $\sigma_X$ is the standard deviation of X.

The data is standardized feature wise, because different features have different value ranges. Once the dataset is standardized, the dataset looks like a table, where we have the target column and the other columns are the input features which are also called the independent variables. As already mentioned, an LSTM learns the sequences. So, to train LSTM models the input data need to be reshaped into a three dimensional tensor. An n-dimensional tensor is technically an n-dimensional matrice. This three dimensional input tensor has a shape which can be represented as : [dataset size, window length, feature dimensions], The first dimension corresponds to the size of the dataset, here it is the number of training data points. The second dimension is the window length, which corresponds to the sequence length. The third dimension is the feature dimension, which is the number of input features.

### 5.4.4. Evaluation

The performance of the trained model is then evaluated on the test data set traces. The model performance is evaluated by evaluating the errors in replicating the four profiles of the following vehicle.

- Acceleration
- Velocity

- Gap
- Trajectory

The errors are calculated using the Root Mean Square Percentage Error (RMSPE) as defined in Equation 5.7.

## 5.5. Comparative Study

The best LSTM neural network based car-following model is then compared to the three conventional car-following models, namely the Krauss, the Wiedemann and the IDM model based on the evaluation strategies explained in subsection 5.4.4.

## 5.6. Conclusion

The methodology presented here provides the general methods and formulation to extract car-following traces from the raw xFCD data, analysis of the three conventional car-following models (IDM, Krauss model and Wiedemann model) and the data-driven development of a car-following model.

# 6. Data Analysis and Processing

This chapter provides preliminary explanatory data analysis of the extracted car-following traces. The results of the post-processing of the data applied in the study are shown and discussed.

### 6.0.1. Data Analysis

The explanatory data analysis of the extracted car-following traces is done. This helps in gaining the understanding of the data and discovering the patterns. The data for the following vehicle and the leader vehicle is analysed. The summary statistics of the following vehicle and the leading vehicle data is shown in Table 6.1 and Table 6.2 respectively. The extracted car-following dataset contains 1453 car-following traces. Since the dataset is recorded at 25 Hz frequency, one second of data for the vehicle generates 25 data points. The total data points as shown in the statistics table are 1188516. The longitudinal velocity of the following vehicle varies from 2.77 m/s to 34.6 m/s with a mean close to 14.8 m/s. The velocity of the leading vehicle varies between 10.01 m/s and 36.6 m/s with a mean value close to 14.5 m/s. The x and y position of the leading vehicle defines its position with respect to the following vehicle. The x position varies between 3.265 m and 100.03 m with a mean value of 38.9 m. The y position of the leading vehicle varies between -0.32 m and 2.5 m with a mean value of -0.06. The negative value of y position corresponds to the left side of the following vehicle whereas the positive value corresponds to the right side of the following vehicle. The statistics of the lateral acceleration, longitudinal acceleration, time headway and the relative velocity can also be seen in these tables.

The plots for a sample car-following trace from the extracted car-following traces is plotted for the analysis and the data quality check. Figure 6.1 shows the longitudinal velocity, longitudinal acceleration, trajectory of following and leading vehicle plot with the relative distance vs. relative velocity plot of the following vehicle. The elliptical shape plot of the relative velocity vs. gap between the vehicles shows the interaction of the leading and the following vehicle. The plot show the changes in the response of the driver's speed of following vehicle in relation to the gap between the leading and the following vehicle. Figure 6.2 shows the initial 150 plotted data points of velocity, acceleration and trajectory of the following and the leading vehicle for a closer look at the relationship between these signals. The plots shows that the acceleration profile of the following vehicle is very noisy. Also, if the acceleration profile of the leading vehicle is observed, it shows that the behaviour is very digital in nature, The changes in the values are step changes and not smooth. The same observation is also valid for the velocity profile of the leading vehicle. Further, Figure 6.3 figure shows the

comparison of the calculated acceleration from the following vehicle velocity vs. the observed acceleration in the car-following trace. The noisy calculated acceleration profile in Figure 6.3 leads to a conclusion that the observed velocity of the following vehicle is also noisy. Only one trace is shown here to avoid repetition. To de-noise this data, data processing is required which is explained in the next section.

Table 6.1.: Description of the features of following vehicle

| | $v_x$ (m/s) | $a_x$ (m/s$^2$) | $a_y$ (m/s$^2$) | v-diff | t-headway(s) |
|---|---|---|---|---|---|
| **count** | 1188516 | 1188516 | 1188516 | 1188516 | 1188516 |
| **mean** | 14.8 | -0.094 | 0.05 | -0.338 | 2.97 |
| **std** | 6.209 | 0.65 | 0.46 | 2.19 | 5.08 |
| **min** | 2.77 | -9.38 | -6.48 | -0.203 | 0.17 |
| **25%** | 1.07 | -0.25 | -0.17 | -1.03 | 1.78 |
| **50%** | 1.44 | -0.085 | 0.08 | -0.05 | 2.37 |
| **75%** | 1.93 | 0.175 | 0.265 | 0.77 | 3.43 |
| **max** | 34.6 | 5.19 | 4.85 | 16.7 | 12.3 |

$v_x$, $a_x$: longitudinal velocity and acceleration; $a_y$: latitudinal acceleration; v-diff: relative velocity of the following vehicle w.r.t leading vehicle; t-headway: time headway of the following vehicle; 25%, 50% and 75% represents the first, seconds and third quartiles respectively.

Table 6.2.: Description of the features of leading vehicle

| | x (m) | y (m) | $v_x$ (m/s) | $v_y$ (m/s) | $a_x$ (m/s$^2$) |
|---|---|---|---|---|---|
| **count** | 1188516 | 1188516 | 1188516 | 1188516 | 1188516 |
| **mean** | 38.9 | -0.06 | 14.5 | -0.12 | -0.034 |
| **std** | 23.02 | 1.9 | 6.53 | 3.56 | 0.533 |
| **min** | 3.265 | -0.32 | 10.01 | -4 | -7.5 |
| **25%** | 22.4 | -0.04 | 10.3 | -0.9 | -0.125 |
| **50%** | 32.8 | -0.002 | 14.6 | 0 | 0 |
| **75%** | 48.9 | 0.42 | 19.1 | 0.675 | 0.125 |
| **max** | 100.03 | 2.5 | 36.6 | 3.8 | 5.625 |

x, $v_x$, $a_x$: position, velocity and acceleration respectively; y, $v_y$: latitudinal position and velocity.

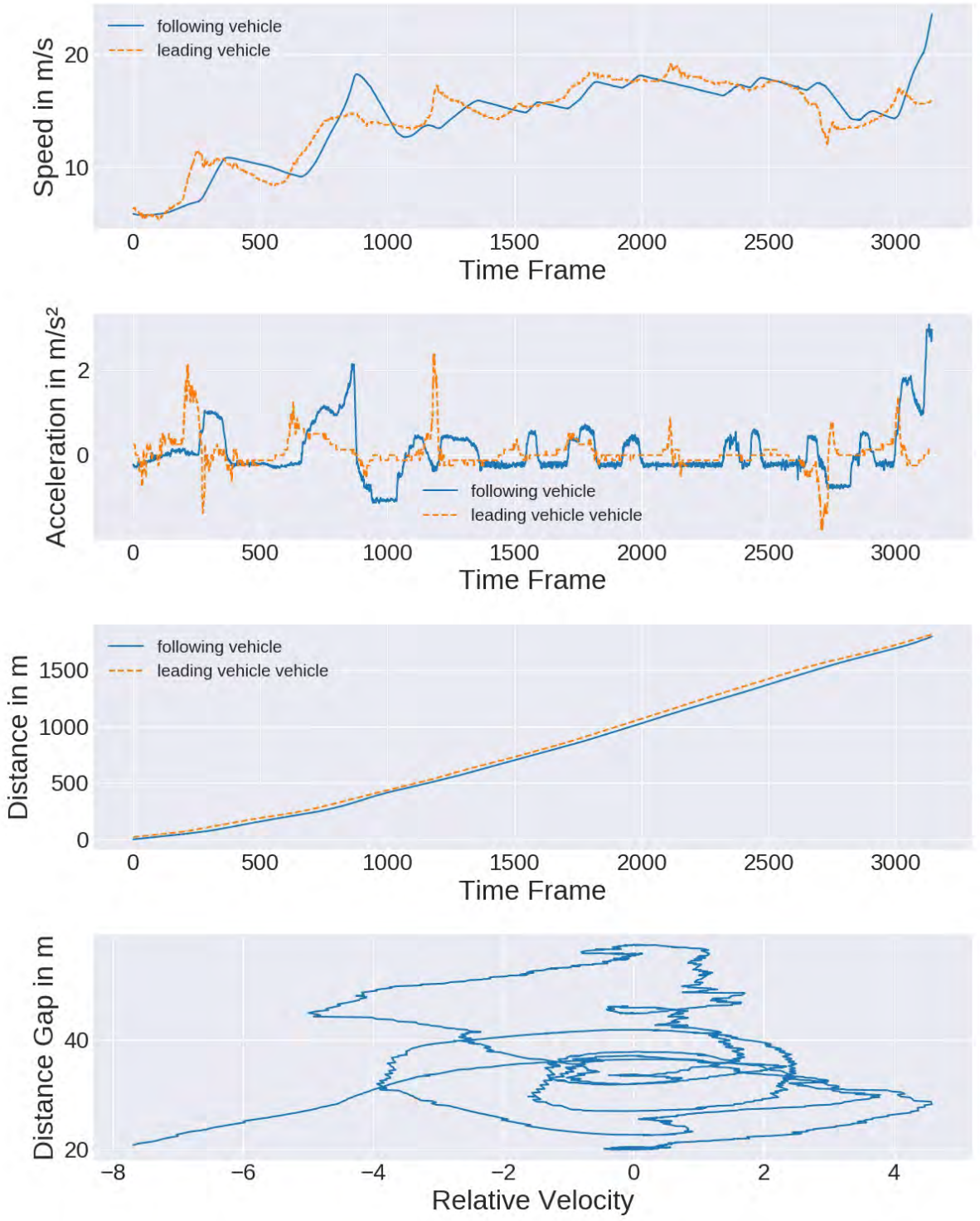



Figure 6.1.: Velocity, acceleration, trajectory and speed-drift plots of a sample car-following trace

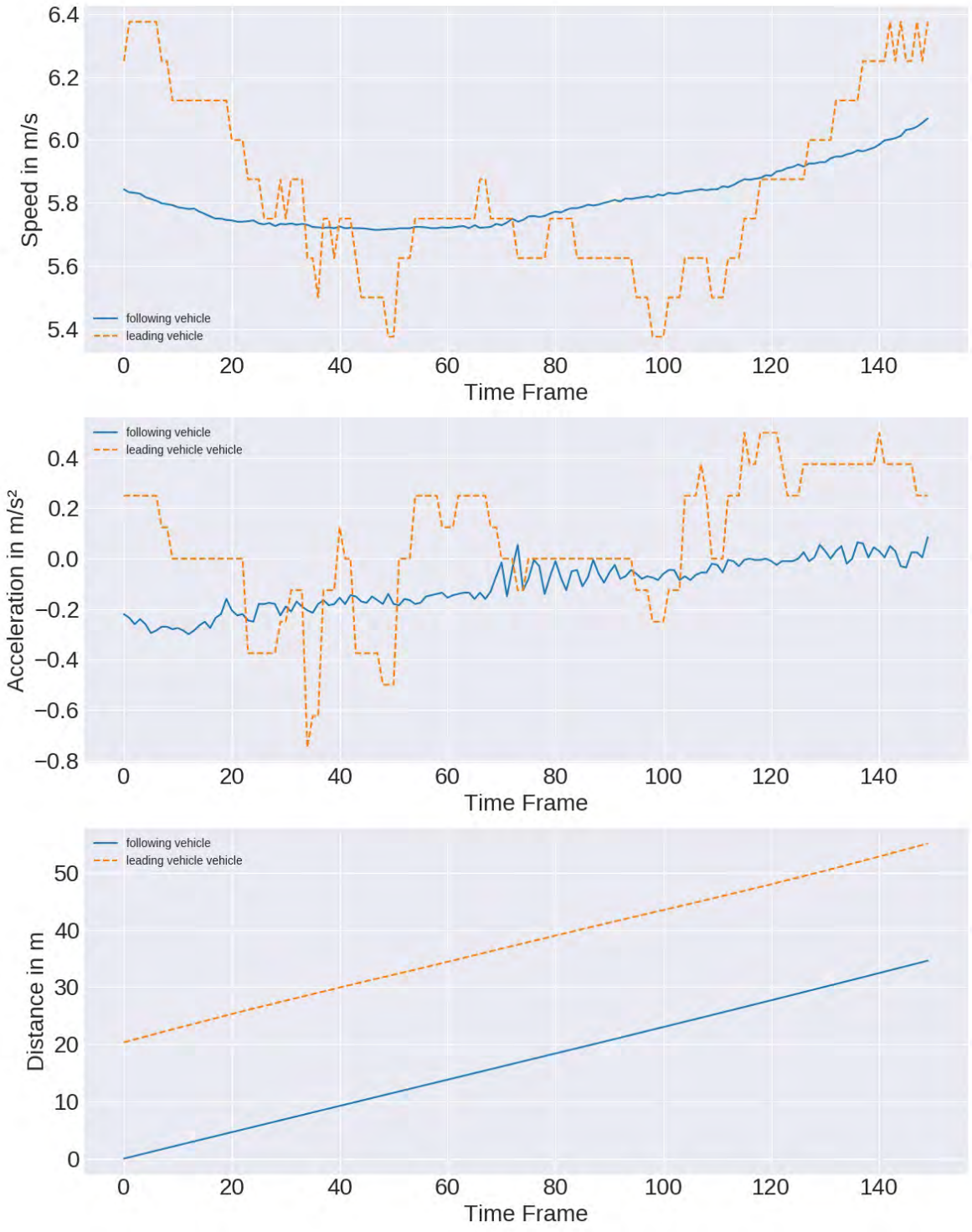



Figure 6.2.: Velocity, acceleration and the trajectory plots of following and the leading vehicle of a sample car-following trace

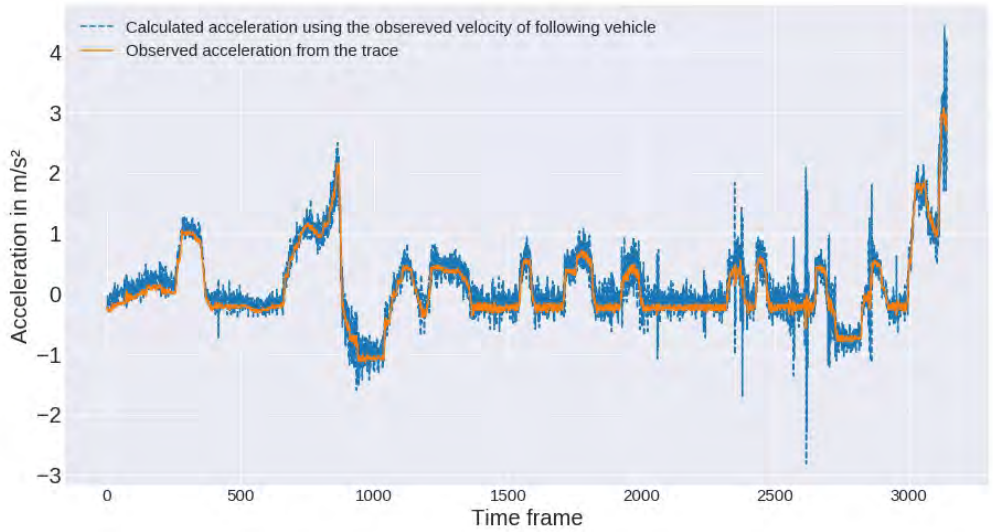



Figure 6.3.: Observed acceleration vs. the calculated acceleration from the observed velocity of the following vehicle

### 6.0.2. Data Post Processing

During the data analysis, it is observed that the velocity and the acceleration values from the dataset are very noisy. The processing of this data is very important. These errors may lead to bad results in the analysis and development of car-following models.

The data collection procedure explains that the data collected for the surrounding vehicles is either using the image processing technique or data fusion technique. These techniques to collect the data may lead to an error in the velocity, acceleration and position data of the leading vehicle. Apart from this, the measurement noise can lead to an error in the data collected for the following vehicle. To remove this noise, data processing using a Gaussian filter is done. A Gaussian filter is applied on the data of longitudinal velocity of the following and the leading vehicle. Then using this filtered values of the velocity, the acceleration values for the following and the leading vehicles are calculated. The gap between the following vehicle and the leading vehicle is then smoothed using the smooth acceleration and velocity of the leading and the following vehicle.

The plots of the smoothed acceleration of the leading and the following vehicle of the sample car-following trace are shown in Figure 6.4 and Figure 6.5. The plots are shown only for one trace to avoid the repetition.

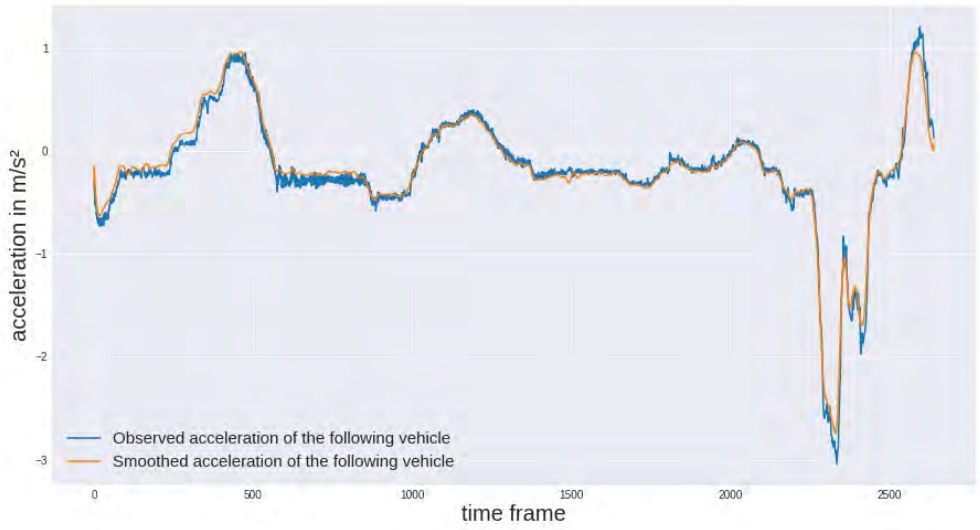



Figure 6.4.: Smoothed acceleration of the following vehicle


Figure 6.5.: Smoothed acceleration of the leading vehicle

# 7. Results

This chapter summaries the results and finding of this study.

## 7.1. Analysis of Car-Following Models

The results of the analysis of calibrated car-following models is presented in this section. The results are shown in two parts. The first part presents the finding of the calibration of each of the three car-following models, namely the IDM, the Krauss model and the Wiedemann model. The second part shows the performance of the calibrated car-following models on the 10 traces chosen out of total 1453 car-following traces.

### 7.1.1. Calibration Results

The three mentioned car-following models are calibrated with a real-coded genetic algorithm. The parameters selected for the genetic algorithm are shown in Table 7.1.

Table 7.1.: Parameters of genetic algorithm used to calibrate the car-following models

| *Genetic Algorithm Parameter* | *Value* |
|---|---|
| Population Size | 100 |
| Generation Size | 100 |
| Crossover Spread Factor ($\eta$) | 10 |
| Mutation Spread Factor ($\eta$) | 10 |
| Crossover Probability | 0.9 |
| Mutation Probability | 0.5 |

Figure 7.1 show the plot of the inverse of the average fitness of the population and the inverse of fitness of the best individual of each generation while calibrating the IDM. The plots with a population size of 30, 50 and 100 are shown. The results converge much faster with a population size of 100 as compared with the population size of 30 and 50. Also, the best fitness is achieved with the population size of 100 and thus it is used for calibration in this study for all the three car-following models.

Table 7.2, Table 7.3 and Table 7.4 shows the summary of the results of calibrated parameters of IDM, the Krauss model and the Wiedemann model respectively.

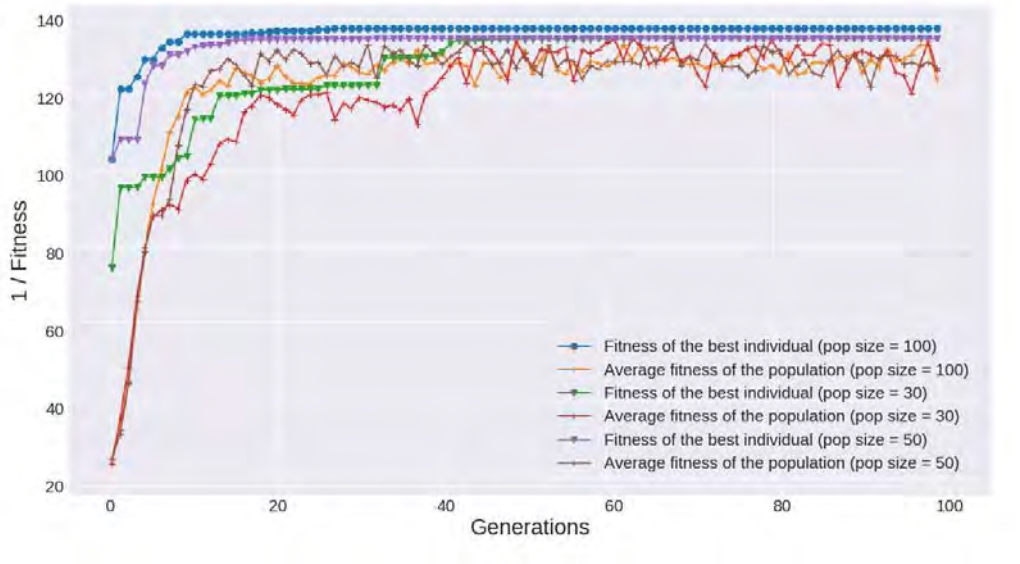



Figure 7.1.: Fitness curve while calibrating IDM using genetic algorithm

Table 7.2.: Summary of the calibrated parameters of IDM

| ***Parameters*** | ***Mean Value*** | ***Std*** |
|---|---|---|
| $T$ | 1.9041 | 0.78 |
| $a$ (m/s$^2$) | 2.064 | 1.74 |
| $b$ (m/s$^2$) | 2.423 | 1.86 |
| $s_0$ (m) | 5.10 | 2.76 |
| ($\delta$) | 3.919 | 0.87 |
| *Error* | 0.0606 | 0.0770 |

Table 7.3.: Summary of the calibrated parameters of the Krauss model

| ***Parameters*** | ***Mean*** | ***Std*** |
|---|---|---|
| $a$ (m/s$^2$) | 1.13 | 1.56 |
| $b$ (m/s$^2$) | 1.24 | 1.64 |
| $\tau$ (s) | 1.880 | 0.93 |
| *Error* | 0.06 | 0.10 |

Table 7.4.: Summary of the calibrated parameters of the Wiedemann model

| ***Parameters*** | ***Mean*** | ***Std*** |
|---|---|---|
| *CC0* (m) | 3.86 | 1.56 |
| *CC1* (s) | 2.36 | 0.82 |
| *CC2* (m) | 17.14 | 8.55 |
| *CC3* (s) | -4.37 | 5.61 |
| *CC4* (m/s) | -0.54 | 0.50 |
| *CC5* (m/s) | 2.36 | 1.48 |
| *CC6* (rad/s) | 6.78 | 6.62 |
| *CC7* ($m/s^2$) | 5.17 | 3.07 |
| *CC8* ($m/s^2$) | 2.53 | 2.64 |
| *CC9* ($m/s^2$) | 2.83 | 2.41 |
| *Error* | 0.09 | 0.13 |

Figure 7.2 shows the distributions of the optimal parameters of the IDM car-following model namely, maximum acceleration $a$ in $m/s^2$, maximum comfortable deceleration $b$ in $m/s^2$, gap at standstill $s_{(0)}$ in m, time headway $T$ in s and acceleration exponent $\delta$. The histogram plots give the idea of the frequency of values of each parameter for all the 1453 traces while the cumulative distribution function (CDF) gives an overview of the statistical summary of the parameter values. The graph shows that the about eighty percent of the traces have $a$ value less than 4 $m/s^2$ but it also shows that around thirty percent of the traces have values either close to 0 $m/s^2$ or close to 5 $m/s^2$. The values close to 0 $m/s^2$ can be considered of the traces with fewer changes in the acceleration during driving and the values close to 5 $m/s^2$ can be considered of the traces in which vehicle starts from zero velocity and accelerates to gain the desired velocity. The plot of maximum comfortable deceleration also has the same profile with a difference that nearly twenty percent of the traces have a value near to 5 $m/s^2$. Around 30 percent of the traces have values less than 3 $m/s^2$. The distribution of standstill gap $s_{(0)}$ gives some interesting results. Around ten percent of the traces have the value near to zero which can be considered of those traces in which the vehicle either follows too closely or the driver characteristics can be considered as aggressive. Also, eighty percent of the traces have values more than 2 m and nearly twenty percent are those traces in which the value of $s_{(0)}$ is near to 8 m describing the larger percentage of careful drivers. The time headway $T$ have a mixed distribution and for nearly twenty percent of the traces have a value near to 3 s. The distribution of acceleration exponent $\delta$ does not seem to vary much and lies either close to 3 or near to 5 for almost sixty percent of the traces. These distributions of the parameters can be used directly when allocating different driver's profiles in the simulation and the values can be chosen using this distribution.

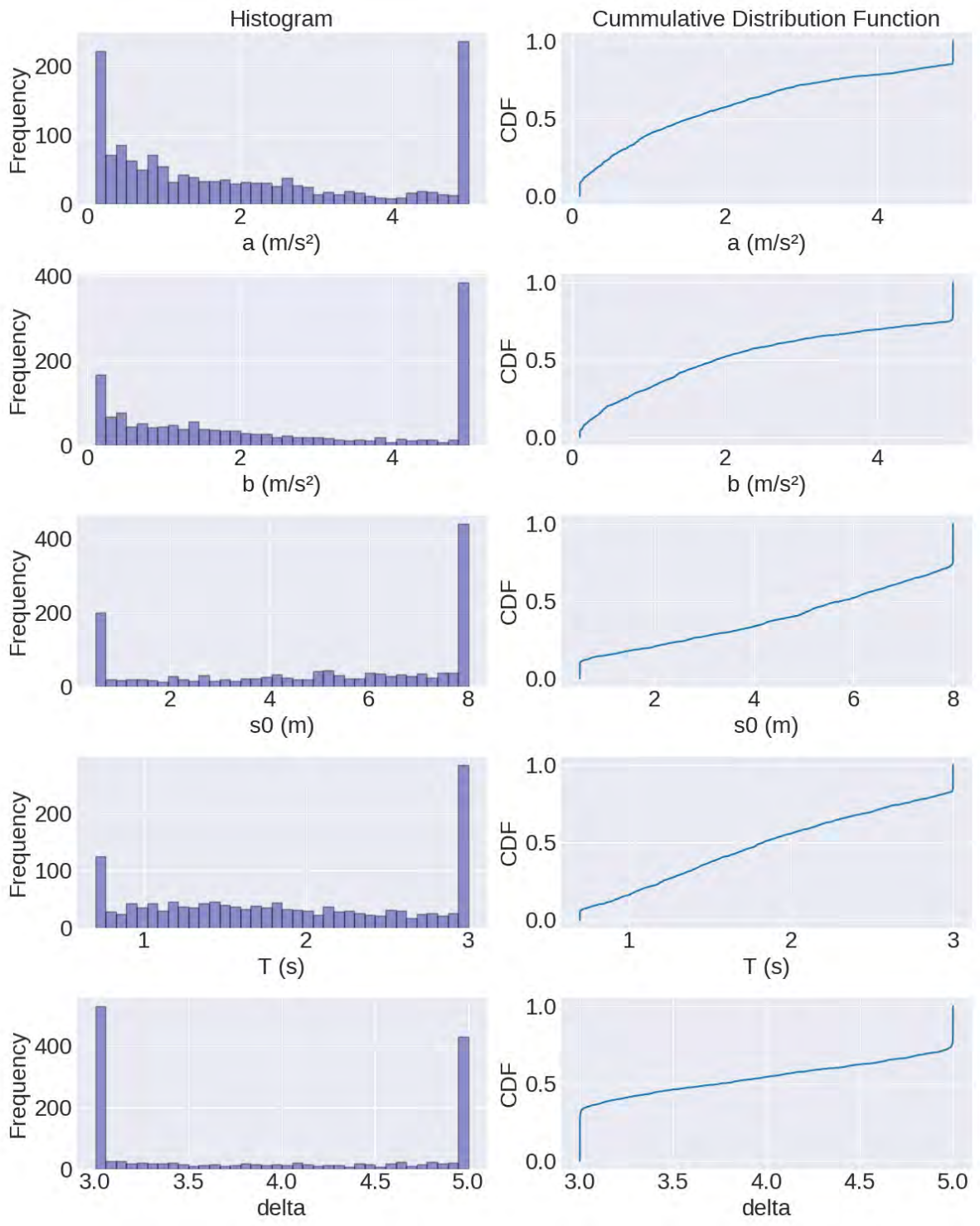



Figure 7.2.: The distribution of the optimal parameters of IDM

The Figure 7.3 shows the distribution of the optimal parameters of the Krauss model, namely maximum acceleration $a$ in $m/s^2$, maximum comfortable deceleration $b$ in $m/s^2$ and reaction time $\tau$ of the driver in s. The results show that $a$ and $b$ although having the same meaning to that of maximum acceleration and maximum comfortable deceleration parameters of IDM but have different distributions. This might be because these values of the optimal parameters are the outcome of the optimisation algorithm. The combination of values of the parameters that minimize the objective function and the value of the certain parameters is affected by the value of the other parameter within the model, for example, reaction time in the Krauss model. The results show that the more than sixty percent of the car-following traces have reaction time of more than 4 s. This reaction time $\tau$ is defined as the driver's attempt to maintain the minimum gap of $\tau$ seconds between the following and the leading vehicle. The value of more than 4 s for sixty percent of the traces explains that most of the drivers are careful drivers.

Figure 7.4 and Figure 7.5 show the distribution of 10 parameters of the Wiedemann model, namely standstill gap CC0 (m), time headway CC1 (s), following variation CC2 (m), threshold entering following CC3 (s), negative following threshold CC4 (m/s), positive following threshold CC5 (m/s), speed dependency of oscillations CC6 (rad/s), oscillation acceleration CC7 ($m/s^2$), standstill acceleration CC8 ($m/s^2$) and acceleration at 80 km/h CC9 ($m/s^2$). The distribution shows that most of the parameters do not show much variation except, the parameter CC5 and CC7. The distribution of parameter CC9 shows half of the values near to zero as the traces are from an urban environment and there are very few traces in which the vehicle speed is near to 80 km/h.

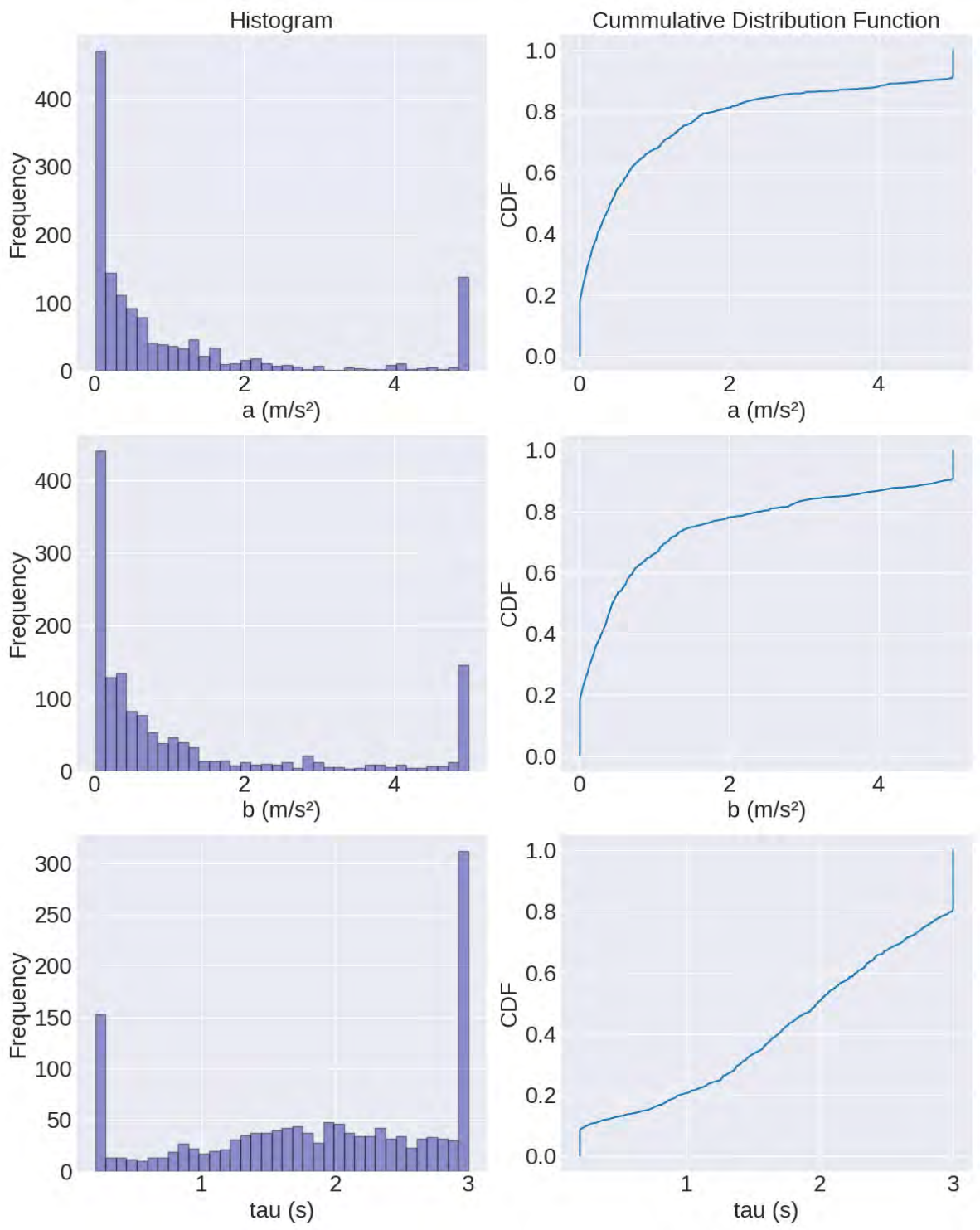



Figure 7.3.: The distribution of the optimal parameters of the Krauss model

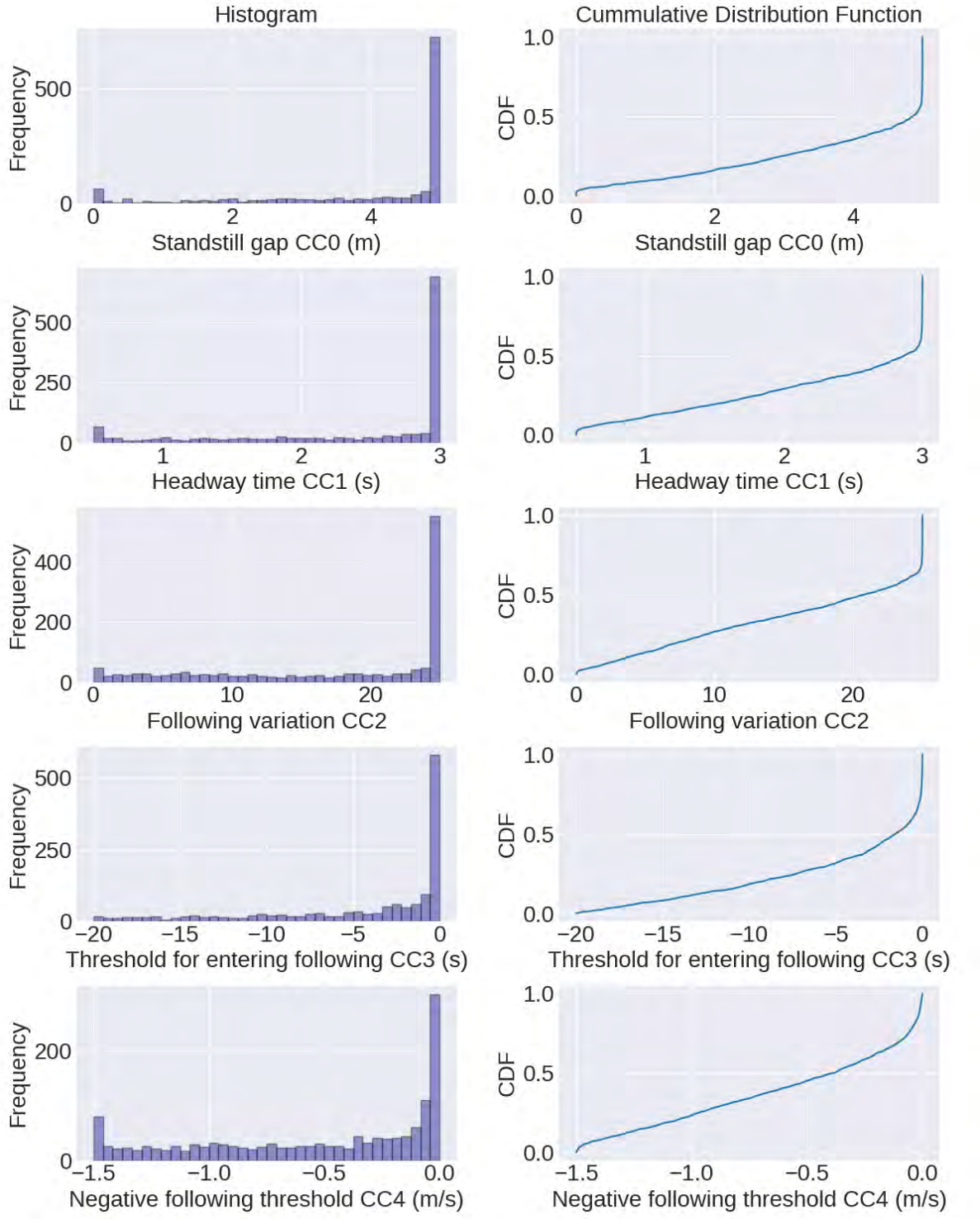



Figure 7.4.: The distribution of the optimal parameters of the Wiedemann model (CC0 - CC4)

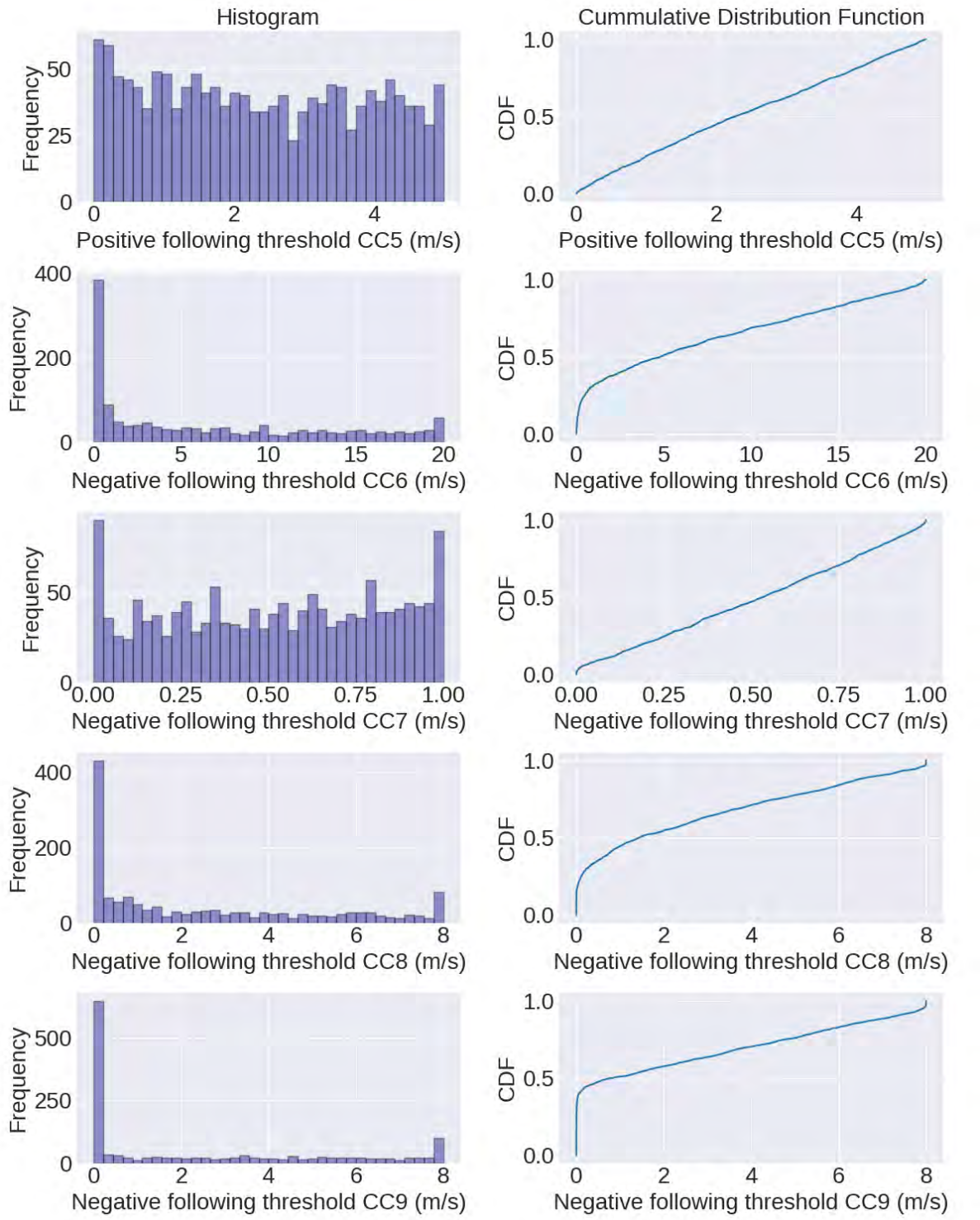



Figure 7.5.: The distribution of the optimal parameters of the Wiedemann model (CC5 - CC9)

### 7.1.2. Evaluation of Performance of Calibrated Models

The performance of the three models with the optimal parameters for 10 chosen car-following traces is evaluated. The 10 car-following traces are chosen from the test car-following traces which are divided while creating the test dataset and training dataset for the training of the LSTM model as explained in subsection 5.4.3. The details of the 10 car-following traces is given in Table 7.5

Table 7.5.: Details of the selected car-following traces for evaluation.

| *Traces* | *No. of data points* | *Mean - Speed* | *Speed - Std* |
|---|---|---|---|
| CF_338345 | 1863 | 25.40 | 3.14 |
| CF_613037 | 1419 | 20.40 | 1.216 |
| CF_140528 | 1428 | 11.55 | 3.30 |
| CF_881846 | 1972 | 11.51 | 3.47 |
| CF_324484 | 5987 | 17.9 | 4.29 |
| CF_744052 | 4549 | 21.14 | 1.8 |
| CF_275974 | 826 | 13.40 | 0.8 |
| CF_507728 | 3989 | 17.18 | 1.78 |
| CF_785485 | 2788 | 10.89 | 3.89 |
| CF_014816 | 1075 | 12.44 | 3.91 |

The Figure 7.6 shows the comparison of fitness (RMSPE as described by Equation 5.6) while calibrating the different models on these traces. The figure shows that the Wiedemann model's calibration error is worst amongst all the three models while the Krauss and IDM have comparable results. For the five traces, namely CF_338345,CF_140528, CF_275974, CF_785485 and CF_014816, the Krauss model error is lowest as compared to the other three models while the IDM has the lowest error for the traces rest five traces.

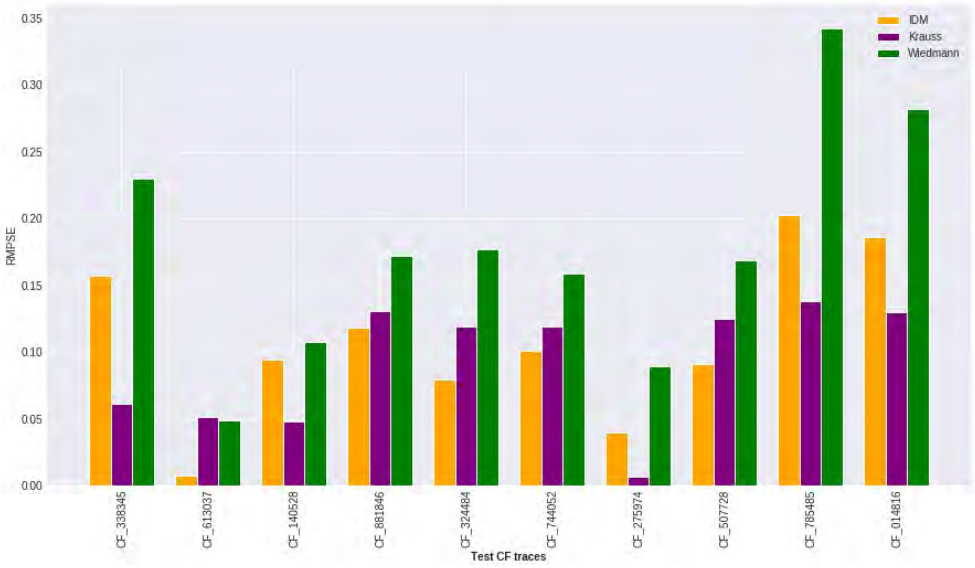



Figure 7.6.: RMSPE comparison of the genetic algorithm calibration for test car-following traces.

The evaluation of the calibrated models is then done on the basis of their ability to replicate the velocity, acceleration, gap between the following vehicle and the leading vehicle and the trajectory of the following vehicle. Figure 7.7 shows the comparison of the RMSPE between the simulated velocity and the observed velocity of the following vehicle for the three calibrated models. The figure explains that the Wiedemann model performs worst in replicating the velocity of the following vehicle for the selected car-following trajectories while the IDM and Krauss model performance is comparative. IDM model perform best for the trace CF_613037.

Figure 7.8, Figure 7.9 and Figure 7.10 show the RMSPE comparison of the acceleration of the following vehicle, gap between vehicles and trajectory of the following vehicle predicted by the three calibrated models. The results of the acceleration profiles shows that the IDM performs best amongst all three models, while the Krauss model performs worst. This is because the Wiedemann and the IDM predict the acceleration as their output while the Krauss predicts the velocity as the model output. The simulated gap between the following and the leading vehicle shows that the IDM and Krauss model performs comparable while the performance of the Wiedemann model is worst among all. The same observation holds true for the predicted trajectory of the following vehicle.

car-following trajectories CF_613037 and CF_785485 are further analysed because the IDM performs best for trace CF_613037 and as per the Figure 7.6 Wiedemann has the worst fitness value. But, Figure 7.7, Figure 7.8, Figure 7.9 and Figure 7.10 show that the Wiedemann

performance is comparable to the other two models. Apart from this, the simulated/predicted velocity RMSPE is also highest for this trace for all the three models.

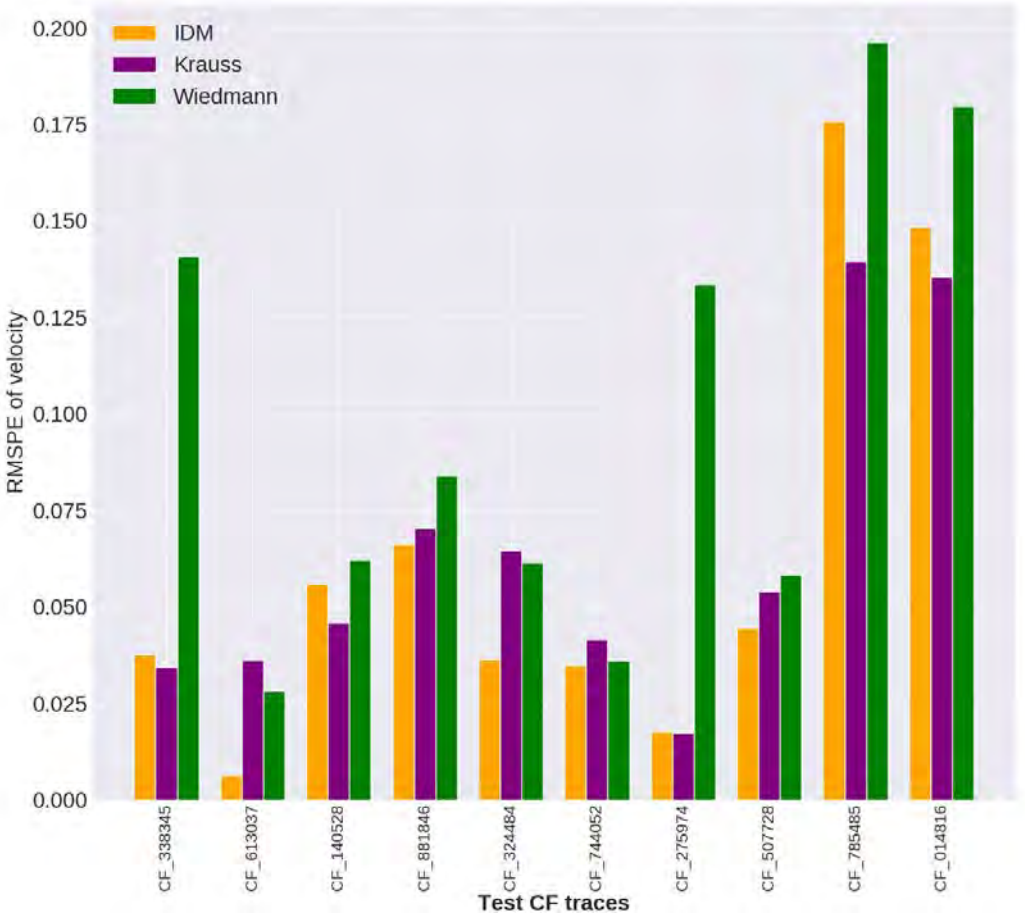



Figure 7.7.: RMSPE comparison of the velocity predicted by calibrated models.

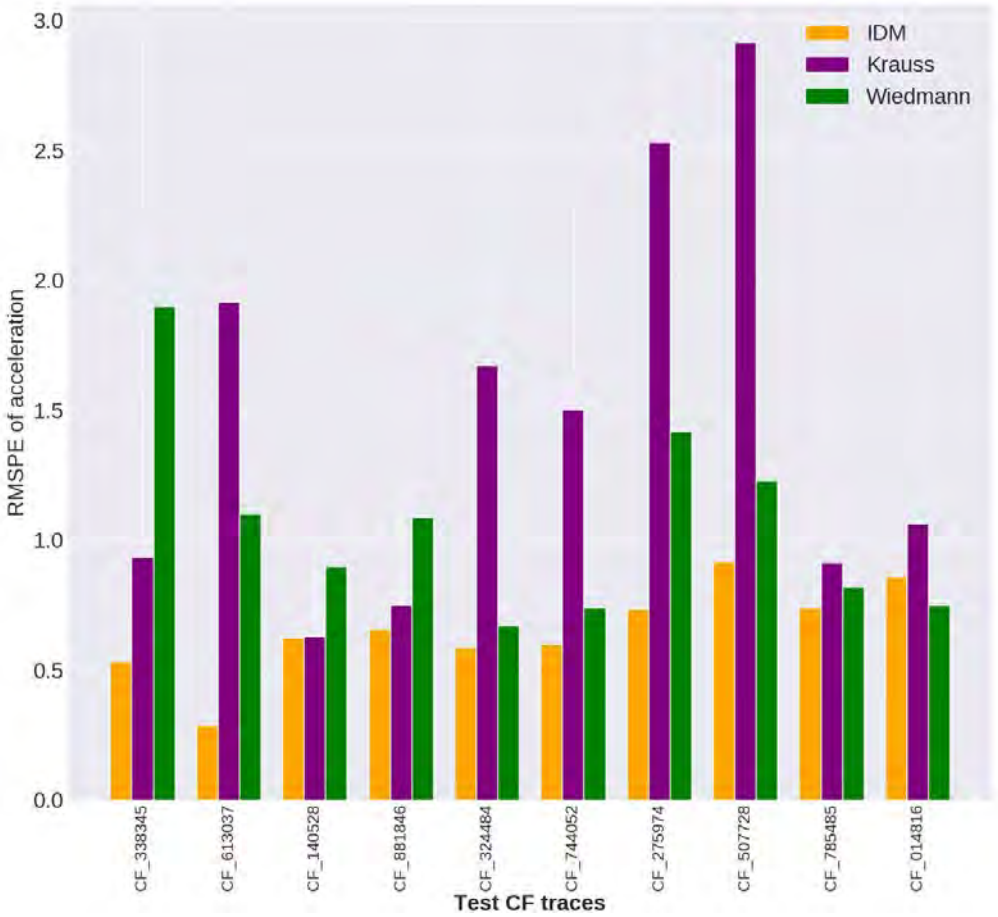



Figure 7.8.: RMSPE comparison of the acceleration predicted by calibrated models.

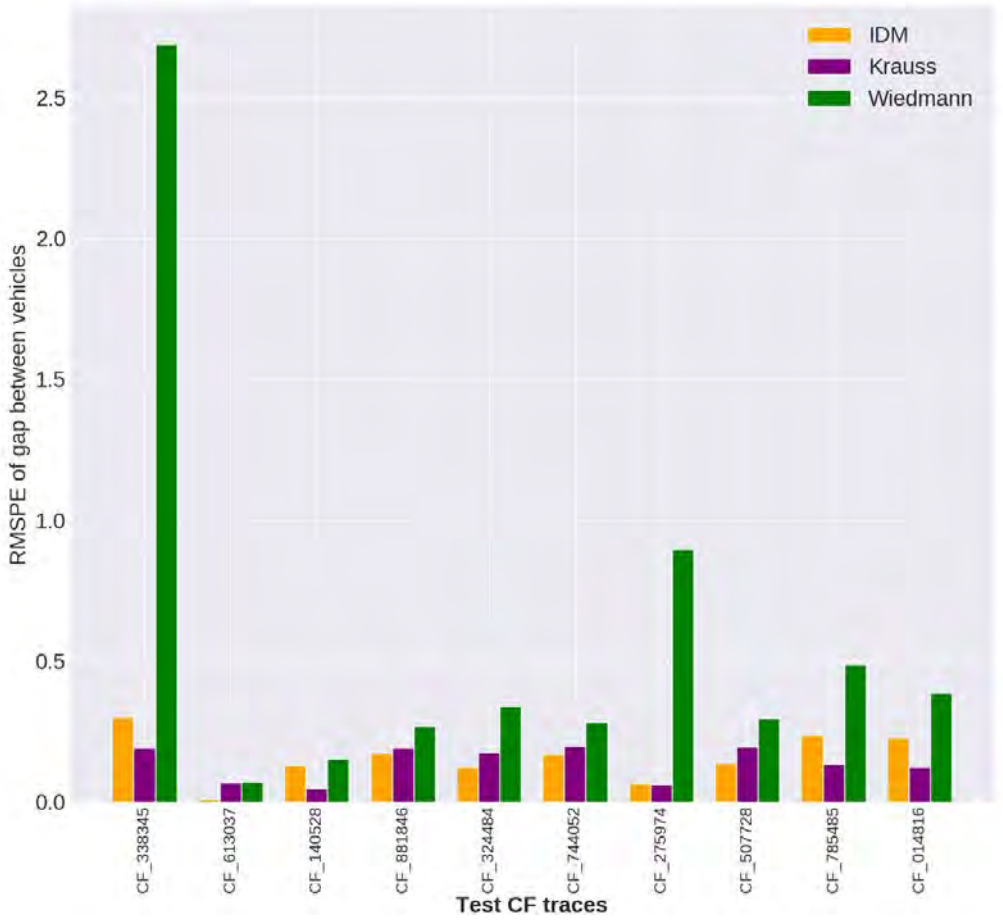



Figure 7.9.: RMSPE comparison of the gap between the following vehicle and the leading vehicle predicted by calibrated models.

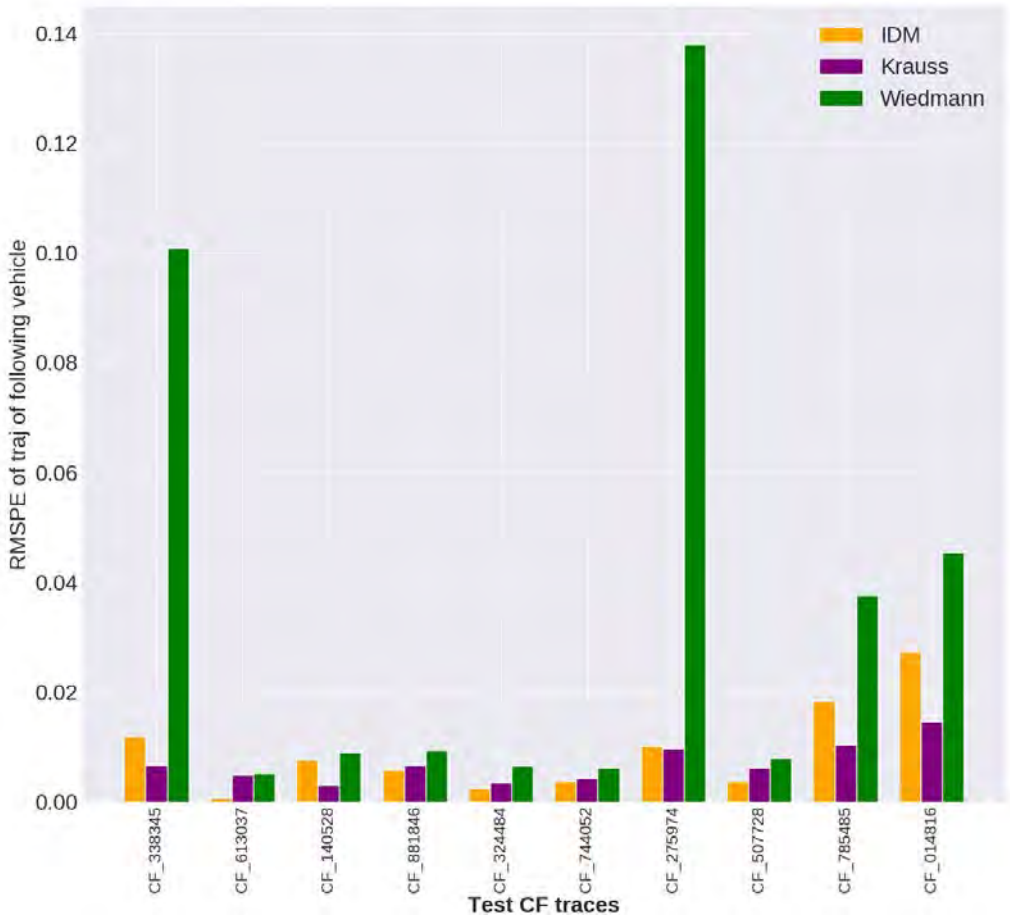



Figure 7.10.: RMSPE comparison of the trajectory of the following vehicle predicted by calibrated models.

To analyse the behaviour of car-following models on the selected car-following traces CF_613037 and CF_785485, the simulated results of the models are plotted against the observed values. Figure 7.11 and Figure 7.12 show the plots for the simulated velocity of the following vehicle for the three car-following model against the observed velocity of the following vehicle for the traces CF_613037 and CF_785485 respectively. The plots show that the velocities simulated by the Krauss model do not seems to be smoothed. This is because the Krauss model output is directly the velocity which is the reason for the occurring fluctuations. The Wiedemann models also shows the fluctuation because of the way the Wiedemann model works. The Wiedemann model has defined thresholds and that is why the output of the Wiedemann can have sharp changes. The IDM model velocity profile, on the other hand, shows a smooth profile. One important observation in Figure 7.11 and Figure 7.12 is that the trace CF_785485 has oscillations in the velocity profile which neither of the model is capable

to simulate.

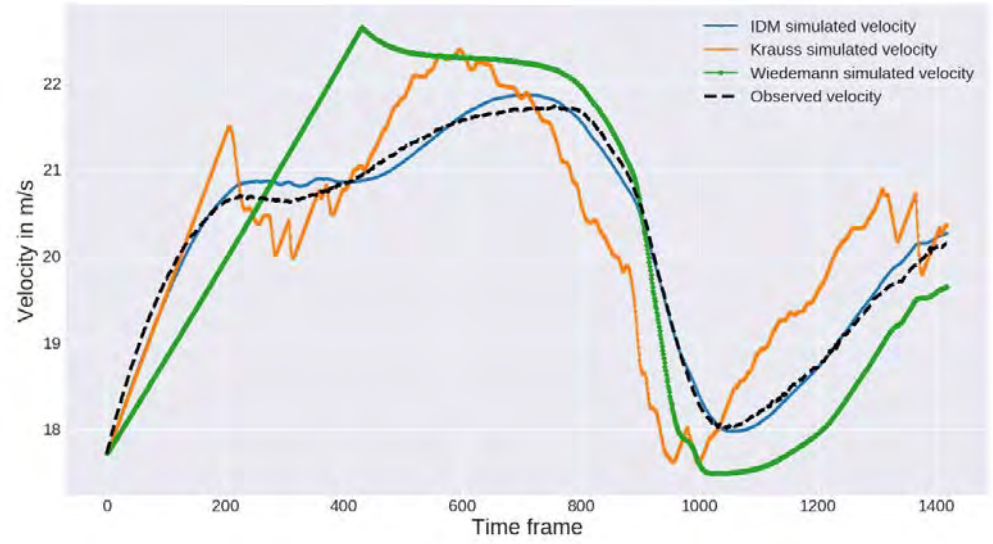



Figure 7.11.: Simulated velocities of the three car-following models against the observed velocity for trace CF_613037

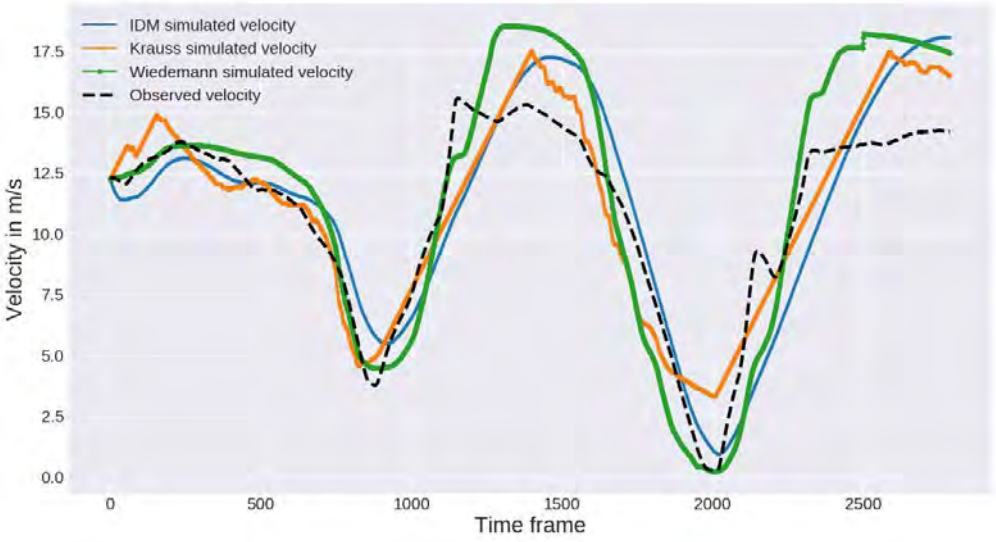



Figure 7.12.: Simulated velocities of the three car-following models against the observed velocity for trace CF_785485

Figure 7.13 and Figure 7.14 show the plots for the simulated acceleration for the three car-following model against the observed acceleration for the traces CF_613037 and CF_785485 respectively. Since the IDM and the Wiedemann model predict directly the acceleration, the profiles of these models seem to fit better to the measured values than those predicted by the Krauss model. The Krauss model has very sharp changes in the predicted acceleration profiles which is because of the fluctuations in the velocity predicted by the Krauss model. Apart from this, the Figure 7.14 shows that the Wiedemann predicts a much closer acceleration profile in case of oscillations. On the other hand, the IDM model is not able to model these sharp changes in the acceleration. Figure 7.14 also shows one sharp change in the predicted acceleration near to the time frame 2500. This can be because of the different thresholds in the Wiedemann model.

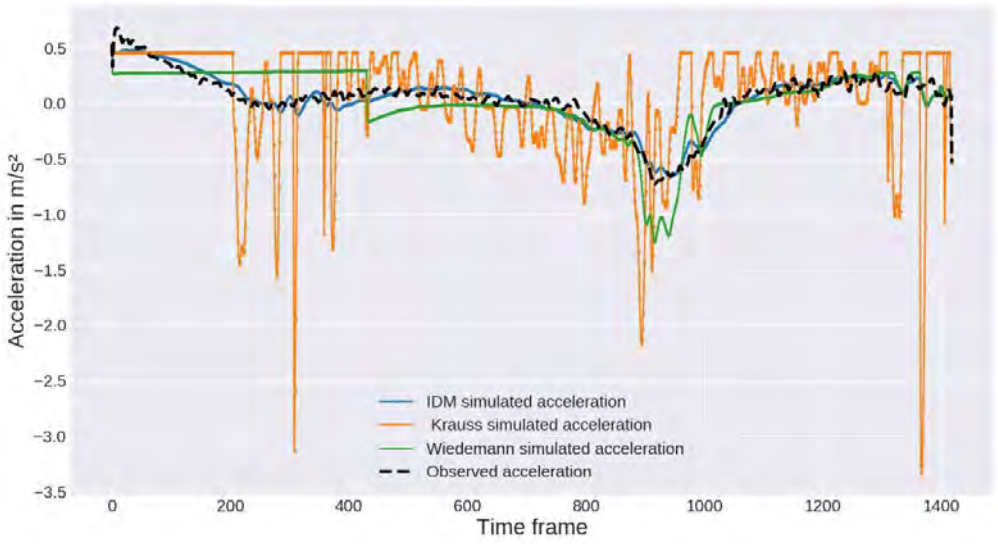



Figure 7.13.: Simulated acceleration of the three car-following models against the observed acceleration for trace CF_613037

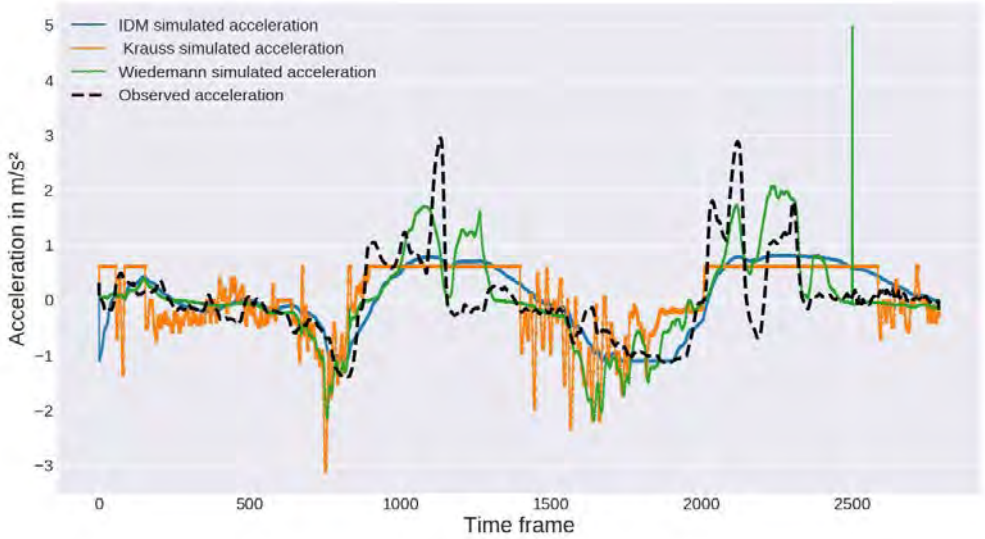



Figure 7.14.: Simulated acceleration of the three car-following models against the observed acceleration for trace CF_785485

Figure 7.15 and Figure 7.16 show the plots for the simulated gap between the following and the leading vehicle for the three car-following models against the observed gap for the traces CF_613037 and CF_785485 respectively. Figure 7.15 shows that the IDM model performs best among all the three models, while the performance of the Wiedemann model is worst in replicating the gap profile. Figure 7.15 on the other hand shows that the Krauss model and the IDM have comparable performance in replicating the gap. The Krauss model output is understandable as, during simulation the Krauss model always tries to maintain the safe distance to the front vehicle and predicts the velocity in correspondence to this gap. If the gap is too large, the Krauss model velocity does not show much variation as can be seen in Figure 7.11 and Figure 7.15. The velocity increases util the time frame 400 and once the gap reaches 50 m, the Krauss model velocity response shows the variation.

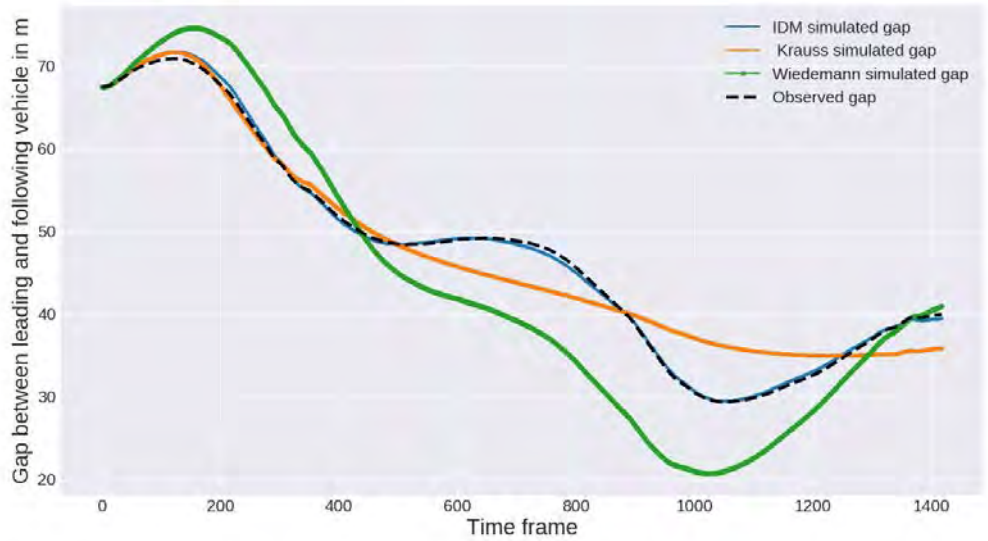



Figure 7.15.: Simulated gap between following and leading vehicle of the three car-following models against the observed gap for trace CF_613037

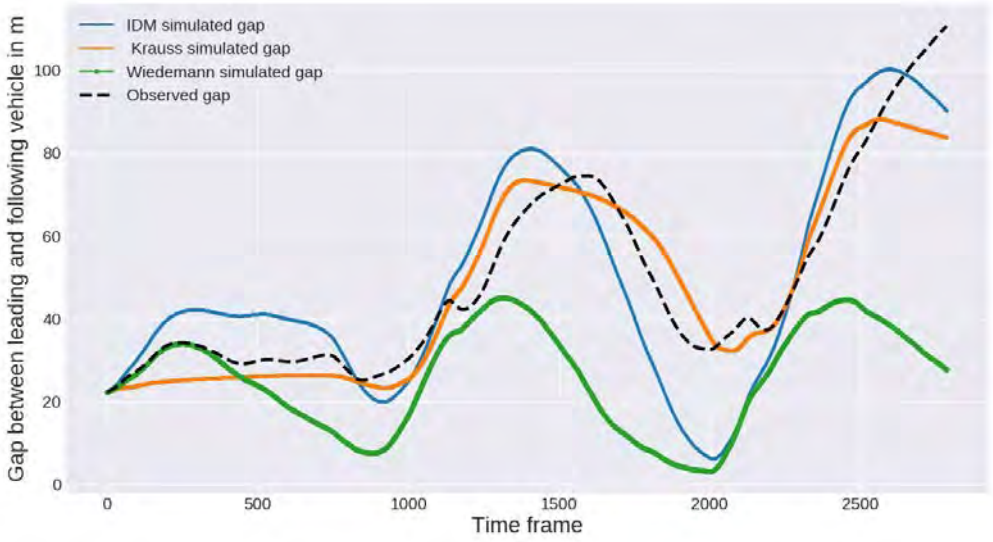



Figure 7.16.: Simulated gap between following and leading vehicle of the three car-following models against the observed gap for trace CF_785485

Figure 7.17 and Figure 7.18 show the plots for the simulated trajectory of the following vehicle for the three car-following models against the observed trajectory for the traces CF_613037 and CF_785485 respectively. The Figure 7.17 shows that all the three models performs comparable and good, because there are no oscillations in the velocity and the acceleration profiles in trace CF_613037. While for the trace CF_785485, the oscillations can be observed in both the velocity and the acceleration profile and hence the result for the trajectory also does not seems to be as exact as that of trace CF_613037. This again confirms that none of these three models are good in simulating the naturalistic driving data if the data have oscillations or sudden changes in it.

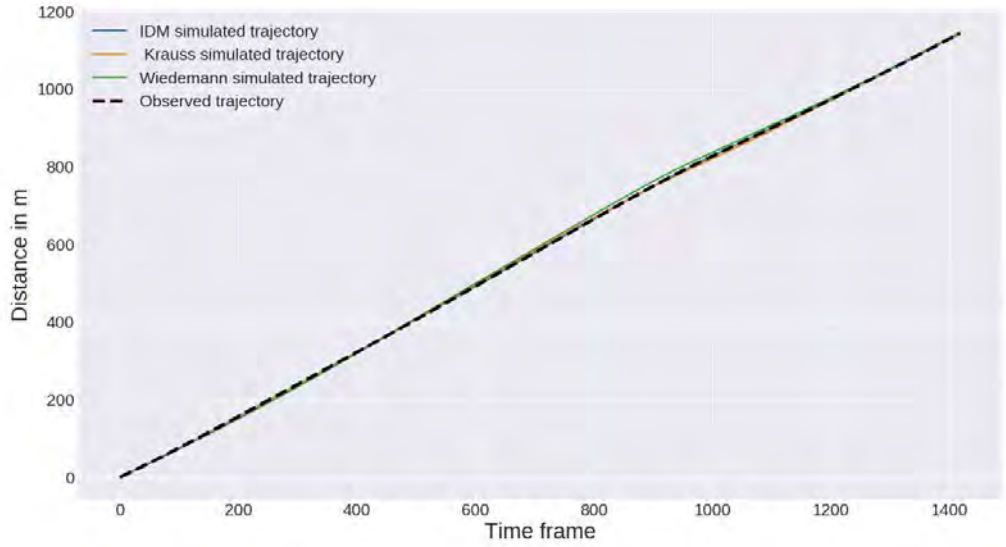



Figure 7.17.: Simulated trajectory of the three car-following models against the observed trajectory for trace CF_613037

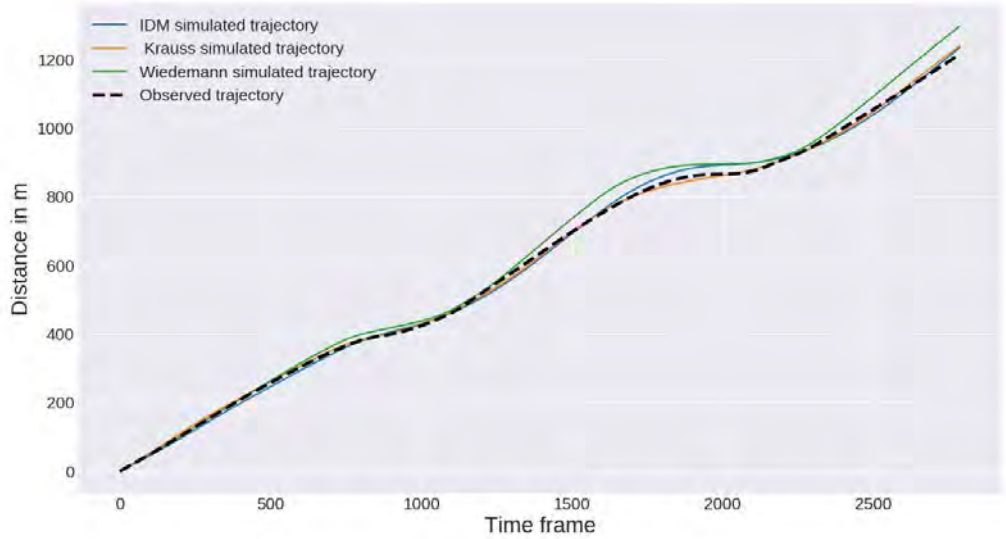



Figure 7.18.: Simulated trajectory of the three car-following models against the observed trajectory for trace CF_785485

## 7.2. Data Driven Car-Following Model

The finding of the analysis of the three car-following models shows that the three models are not suitable to simulate the xFCD data. The RMSPE error for the acceleration and velocity profiles is considerable high. Apart from this, the models are not able reproduce in case of sharp changes or oscillations in the acceleration or the velocity profiles in the observed data. One more finding is that the Krauss model output, which is velocity, is very fluctuating. These types of fluctuations are never seen in the real world driving data.

To develop the data-driven model, the LSTM neural network is used which is a type of Recurrent Neural Networks. LSTM is chosen because of its ability to learn the long term dependencies in the data. The results of the data-driven car-following model are summarised on the basis of the followings:

- Input features and model structure.
- Sequence length / window length.
- Evaluation of the best model on test car-following traces.
- Feature importance.

### 7.2.1. Input Features and Model Structure

To model the LSTM based car-following model, seven input features are considered in this study. The input features are as follows:

- Longitudinal acceleration of the following vehicle.
- Longitudinal velocity of the following vehicle.
- Relative velocity of the following vehicle w.r.t. the leading vehicle.
- The gap between the following and the leading vehicle.
- Time headway of the following vehicle.
- Presence of the vehicle on the left lane of the following vehicle.
- Presence of the vehicle on the right lane of the following vehicle.

The output of the LSTM based car-following model is considered as the acceleration of the following vehicle for the next time step.

Once the input and the output features are finalised, the best configuration of the LSTM neural network model is identified by its performance on the validation dataset and the training time. The adopted model consists of the first layer as convolutional layer. The layer has a filter size of 8 with a kernel size of 2. Apart from this, the model consists of two hidden LSTM layers with the first containing 256 LSTM cells and the second containing 128 LSTM cells. The batch normalisation and dropout layers are also inserted after each LSTM layer. Dropout deactivates a percentage of neurons of each layer randomly on per epoch. Therefore, during the training of the neural network these neurons are not used on that specific epoch. This avoids over-fitting of the model on training dataset. Batch normalization adds a little noise to the network and hence adds some regularisation. The layers are stacked on top of each other to enable the model to learn a higher level of temporal dependencies. The second LSTM layer after the dropout and batch normalisation layer is connected to a dense layer with 1 neuron. This dense layer is the output layer with a linear activation function.

Note: The filter size and the kernel size of the convolutional layer is changed for the dataset with 40 ms of sequence length. The kernel size used for the dataset with 40 ms sequence length is 1 with a filter size of 4.

### 7.2.2. Sequence Length and Input Features

To train the model, four different sequence lengths are chosen. The idea here is to find the importance of historical data in the car-following behaviour of the driver. The chosen four sequence lengths are:

- 0.04 s : 1 data point

- 1 s : 25 data points
- 2 s : 50 data points
- 3 s : 75 data points

The results of the training loss and the validation loss of these sequence lengths is shown in Figure 7.19.

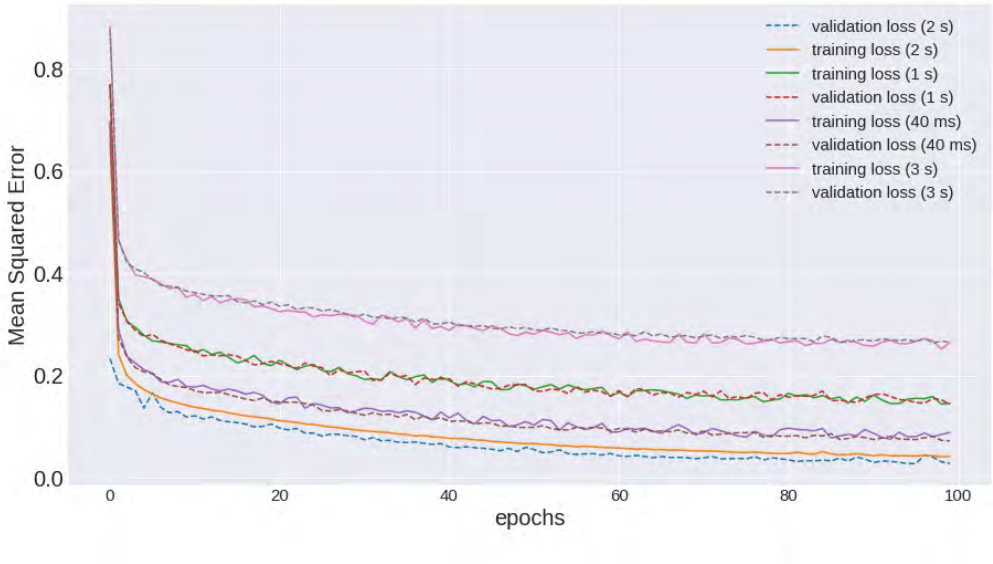



Figure 7.19.: Loss during training and validation of LSTM neural network for different input sequence lengths

The curve shows that the model structure and the data input is fine, as there is no overfitting of any model. Apart from this, the loss curve also shows that the LSTM model learns best with the input sequence length of 2 s. Further analysis of the four different models with different sequence length is also done on the 10 selected car-following traces which is explained in the next sub-section.

### 7.2.3. Evaluation of Models

The evaluation of the LSTM based car-following model is done on the 10 selected car-following traces as mentioned in subsection 7.1.2. Figure 7.20 shows the comparison of the RMSPE between the predicted and the observed velocity of the following vehicle from the ten test car-following traces (Table 7.5).

It can be observed that the model with the input sequence length of 2 s performs better than the other three models. The error profile of the model with an input sequence length of

3 s shows very bad results which is shown in a black bar in the Figure 7.20. This means that either the dependencies are too long to be learned by the LSTM model or the driver behaviour does not depend on very long historical data, such as 3 s. Further, if we compare the error profiles of models with an input sequence length of 1 s and 40 ms with the error profile of the model with an input sequence of 2 s, it can be observed that the performance of the former two models is worst than the later one. This implies that there is a dependency between the driver's past driving manoeuvres and the current driving state. The Figure 7.21, Figure 7.22 and Figure 7.23 show the comparison of RMSPE profiles of the predicted acceleration of the following vehicle, the gap between the vehicles and trajectory of the following vehicle respectively. These figures also show that the model with an input sequence of 2 s is performing more realistic than the other three models. This explains that, in a car-following scenario, the driver's behaviour is dependent on the historical data inputs and the optimum value is 2 s for the dataset used in this study.

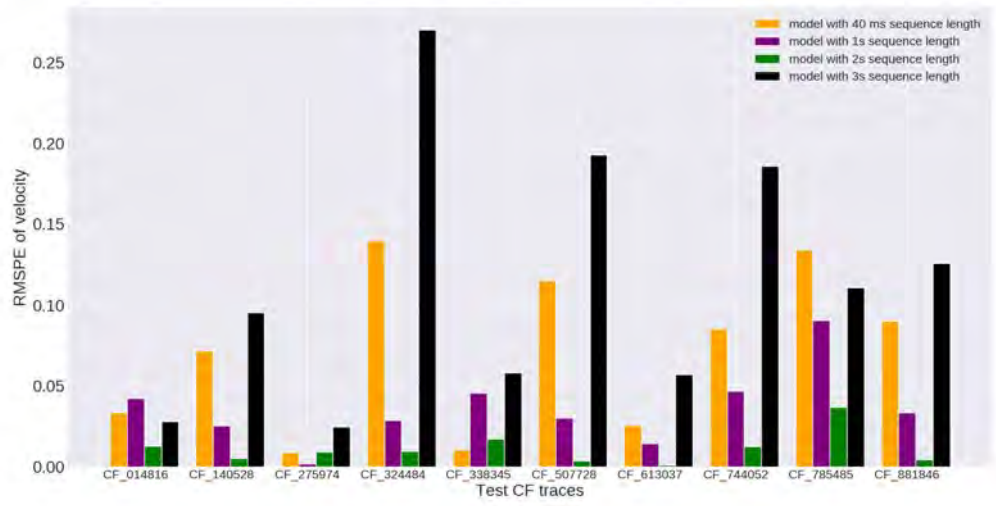



Figure 7.20.: RMSPE comparison of the velocity predicted by LSTM based car-following model.

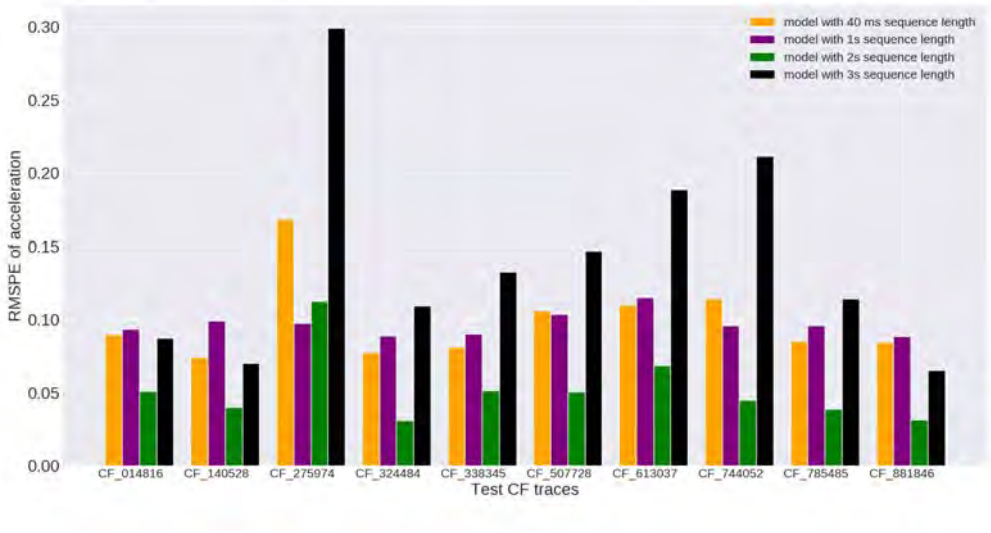



Figure 7.21.: RMSPE comparison of the acceleration predicted by LSTM based car-following model.

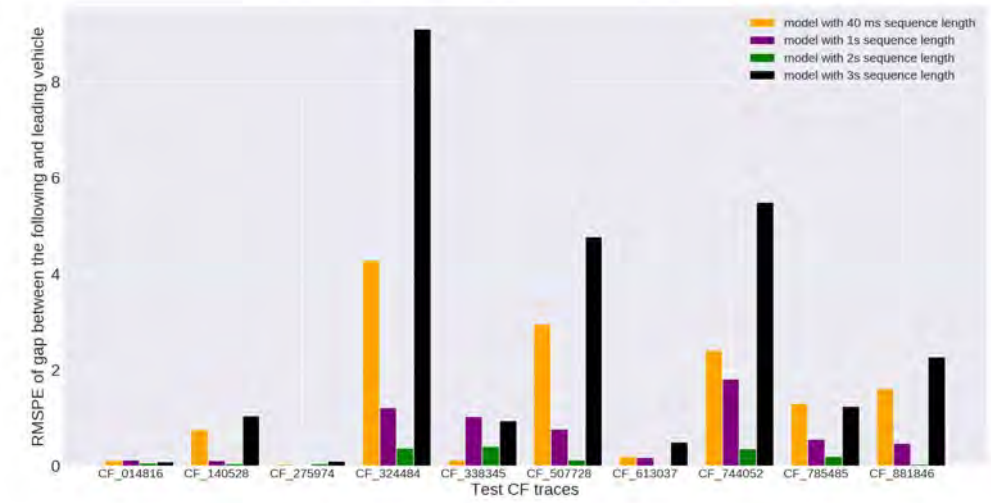



Figure 7.22.: RMSPE comparison of gap between the following vehicle and the leading vehicle predicted by LSTM based car-following model.

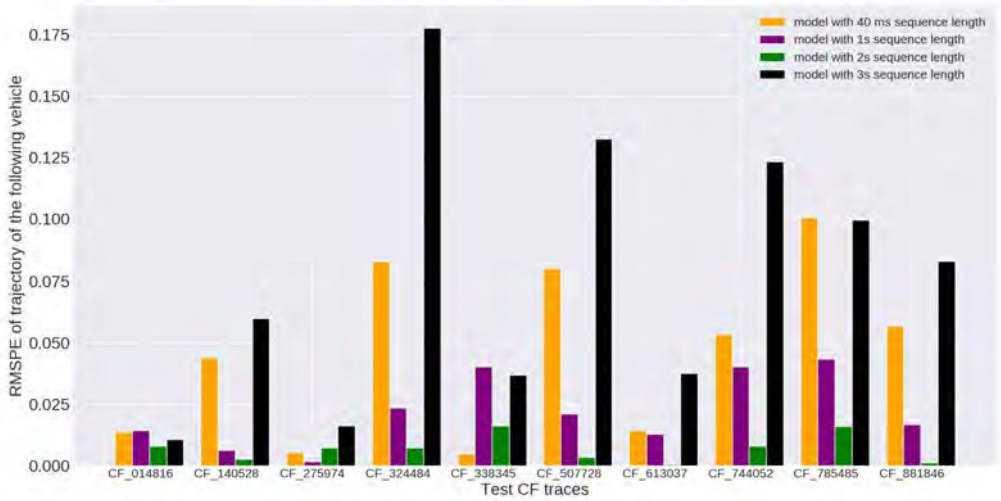



Figure 7.23.: RMSPE comparison of trajectory of the following vehicle predicted by LSTM based car-following model.

### 7.2.4. Feature Importance

Machine learning methods allow to find the importance of each feature in predicting their output. A permutation feature importance is implemented in this study. This Permutation feature importance measures the percentage increase in the prediction error, if any of the input features is shuffled randomly. This allows to find the relationship between the input feature and the outcome of the machine learning method.

In this study, seven input features are considered to predict the acceleration of the following vehicle using the LSTM neural network. To implement the permutation feature importance, the validation dataset is considered. Seven different datasets are created, each dataset with one of the feature randomly shuffled. The model with input sequence length of 2 s is then used to find the loss on validation dataset and all the seven datasets with shuffled inputs. The percentage increase in the loss is then calculated using the formula:

$$Importance = \frac{(loss\_with\_shuffled\_dataset - loss\_with\_validation\_datastet)}{loss\_with\_validation\_datastet}, \tag{7.1}$$

The results of the permutation importance are shown in Table 7.6. The results show that the historical data of acceleration is the most important feature to predict the acceleration of the following vehicle in the next time step. Shuffling the input acceleration values of the input sequence leads to 80% increase in the model's prediction loss. The second most important feature is the input velocity of the following vehicle while the third and the fourth important features are the relative gap between the vehicle and the relative velocity of the following

vehicle with respect to the leading vehicle. The time headway shows very less importance and just contributes to 0.5 percent increase in the loss. The input features about the presence of a vehicle on the left and the right lane of the following vehicle does not seem to be important. An explanation can be given on this aspect when analysing the presence of this data in the input car-following traces. Table 7.7 shows the comparison of the number of data points in which the vehicle is present on either the left or right lane of the following vehicle in the training dataset. The vehicle is present on the left lane for only 0.64 percent of total data points while on the right lane it is present for only 1.78 percent. This low percentage leads to the imbalance in the training dataset and can be considered as one of the reasons for the low importance of this feature.

Table 7.6.: Feature importance of input features of LSTM based car-following model

| *Input Feature* | *Importance* |
|---|---|
| Acceleration of following vehicle | 80% |
| Velocity of following vehicle | 27% |
| Gap between vehicles | 20% |
| Time headway | 0.5% |
| Relative velocity | 10% |
| Vehicle present on left | 0.01% |
| Vehicle present on right | 0% |

Table 7.7.: Counts of the data points where a a vehicle is present on the left lane or the right lane of following vehicle

| *Presence* | *Counts* | *Percentage* |
|---|---|---|
| Vehicle present on left lane | 7660 | 0.64 |
| Vehicle absent on left lane | 1180856 | 99.36 |
| Vehicle present on right lane | 21089 | 1.78 |
| Vehicle absent on right lane | 1167427 | 98.22 |

## 7.3. Comparative Study

The Comparative study of the best LSTM based car-following model, i.e. model with an input sequence of 2 s is done with the calibrated IDM, the Wiedemann model and the Krauss model.

Table 7.8 shows the comparison in the RMSPE of the predicted velocity. The LSTM based car-following model outperforms every other model. Similarly, Table 7.9 shows the RMSPE comparison of the predicted acceleration and the LSTM model have the least RMSPE than

all the traces. Table 7.10 and Table 7.11 show the RMSPE comparison of the predicted gap between the vehicles and predicted trajectory of the following vehicle respectively. The performance of the LSTM based car-following model is comparable for most of the traces except trace CF_338345, CF_324484 and CF_744052 where the error of LTSM is not the lowest but slightly higher than those of Krauss model. The RMPSE of the predicted trajectory is the lowest for all the traces of the LSTM based car-following model.

Table 7.8.: RMSPE comparison of the predicted velocity of the following vehicle

| *Car-Following Trace* | *RMSPE-IDM* | *RMSPE-Krauss* | *RMSPE-Wiedemann* | *RMSPE-LSTM (2s)* |
|---|---|---|---|---|
| CF_338345 | 0.037 | 0.034 | 0.140 | 0.009 |
| CF_613037 | 0.006 | 0.03 | 0.02 | 0.001 |
| CF_140528 | 0.05 | 0.04 | 0.06 | 0.005 |
| CF_881846 | 0.06 | 0.07 | 0.08 | 0.004 |
| CF_324484 | 0.036 | 0.064 | 0.061 | 0.009 |
| CF_744052 | 0.034 | 0.04 | 0.035 | 0.012 |
| CF_275974 | 0.017 | 0.017 | 0.13 | 0.009 |
| CF_507728 | 0.04 | 0.058 | 0.053 | 0.003 |
| CF_785485 | 0.17 | 0.13 | 0.19 | 0.036 |
| CF_014816 | 0.14 | 0.13 | 0.17 | 0.01 |

Table 7.9.: RMSPE comparison of the predicted acceleration of the following vehicle

| *Car-Following Trace* | *RMSPE-IDM* | *RMSPE-Krauss* | *RMSPE-Wiedemann* | *RMSPE-LSTM (2s)* |
|---|---|---|---|---|
| CF_338345 | 0.53 | 0.93 | 1.9 | 0.05 |
| CF_613037 | 0.28 | 1.9 | 1.1 | 0.06 |
| CF_140528 | 0.62 | 0.63 | 0.9 | 0.04 |
| CF_881846 | 0.65 | 0.74 | 1.09 | 0.031 |
| CF_324484 | 0.58 | 1.67 | 0.67 | 0.031 |
| CF_744052 | 0.60 | 1.50 | 0.74 | 0.04 |
| CF_275974 | 0.73 | 2.5 | 1.42 | 0.011 |
| CF_507728 | 0.91 | 2.91 | 1.23 | 0.05 |
| CF_785485 | 0.74 | 0.91 | 0.82 | 0.038 |
| CF_014816 | 0.85 | 1.06 | 0.75 | 0.05 |

Table 7.10.: RMSPE comparison of the predicted gap between the following and the leading vehicle

| *Car-Following Trace* | *RMSPE-IDM* | *RMSPE-Krauss* | *RMSPE-Wiedemann* | *RMSPE-LSTM (2s)* |
|---|---|---|---|---|
| CF_338345 | 0.30 | 0.19 | 2.69 | 0.415 |
| CF_613037 | 0.008 | 0.06 | 0.07 | 0.006 |
| CF_140528 | 0.12 | 0.04 | 0.15 | 0.04 |
| CF_881846 | 0.17 | 0.19 | 0.26 | 0.033 |
| CF_324484 | 0.12 | 0.17 | 0.33 | 0.36 |
| CF_744052 | 0.17 | 0.19 | 0.28 | 0.36 |
| CF_275974 | 0.06 | 0.06 | 0.89 | 0.04 |
| CF_507728 | 0.13 | 0.13 | 1.29 | 0.12 |
| CF_785485 | 0.23 | 0.13 | 0.48 | 0.13 |
| CF_014816 | 0.22 | 0.12 | 0.38 | 0.06 |

Table 7.11.: RMSPE comparison of the predicted trajectory of the following vehicle

| *Car-Following Trace* | *RMSPE-IDM* | *RMSPE-Krauss* | *RMSPE-Wiedemann* | *RMSPE-LSTM (2s)* |
|---|---|---|---|---|
| CF_338345 | 0.01 | 0.006 | 0.10 | 0.01 |
| CF_613037 | 0.0006 | 0.004 | 0.005 | 0.0004 |
| CF_140528 | 0.007 | 0.003 | 0.008 | 0.002 |
| CF_881846 | 0.005 | 0.006 | 0.009 | 0.001 |
| CF_324484 | 0.002 | 0.003 | 0.006 | 0.007 |
| CF_744052 | 0.003 | 0.004 | 0.006 | 0.008 |
| CF_275974 | 0.01 | 0.009 | 0.13 | 007 |
| CF_507728 | 0.003 | 0.006 | 0.007 | 0.003 |
| CF_785485 | 0.01 | 0.01 | 0.03 | 0.01 |
| CF_014816 | 0.02 | 0.14 | 0.04 | 0.007 |

The simulated results of all the four car-following models, namely LSTM, IDM, Krauss and Wiedemann, are also visualised for the traces CF_613037 and CF_78548 as shown in Figure 7.24, Figure 7.25,Figure 7.26, Figure 7.27, Figure 7.28, Figure 7.29.

The Figure 7.24 and Figure 7.25 show the comparison of the simulated velocity of the following vehicle using all the four models for the traces CF_613037 and CF_78548 respectively. As can be seen from the Figure 7.24 that the LSTM based car-following model with an input sequence of 2 seconds outperforms all the other three car-following models for both traces. Also, the LSTM based model is able to model the oscillations in the velocity profile which is present in trace CF_78548. Figure 7.26 and Figure 7.27 show the comparison of the simulated

acceleration of all four models. The LSTM based car-following model simulates smooth acceleration profiles. Since the LSTM based car-following model predicts the acceleration, the predicted acceleration profile is very close to the observed acceleration profile.

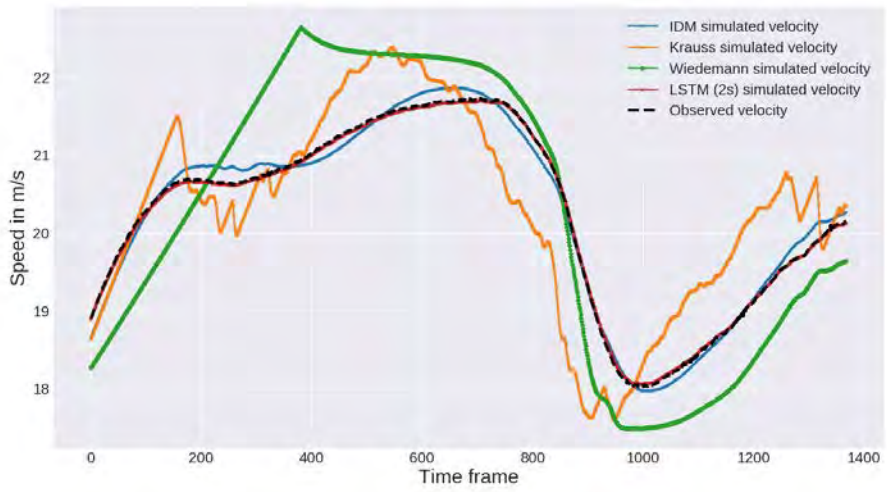



Figure 7.24.: Simulated velocity of the four car-following models against the observed acceleration for trace CF_613037

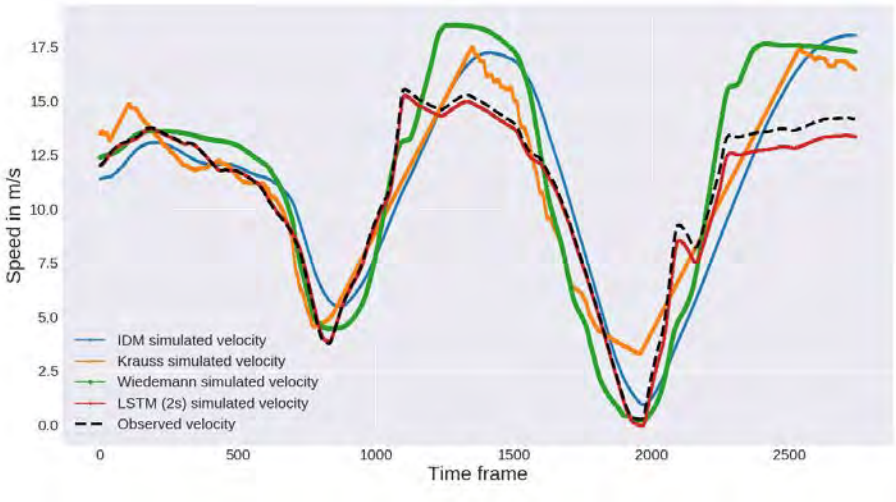



Figure 7.25.: Simulated velocity of the four car-following models against the observed acceleration for trace CF_785485

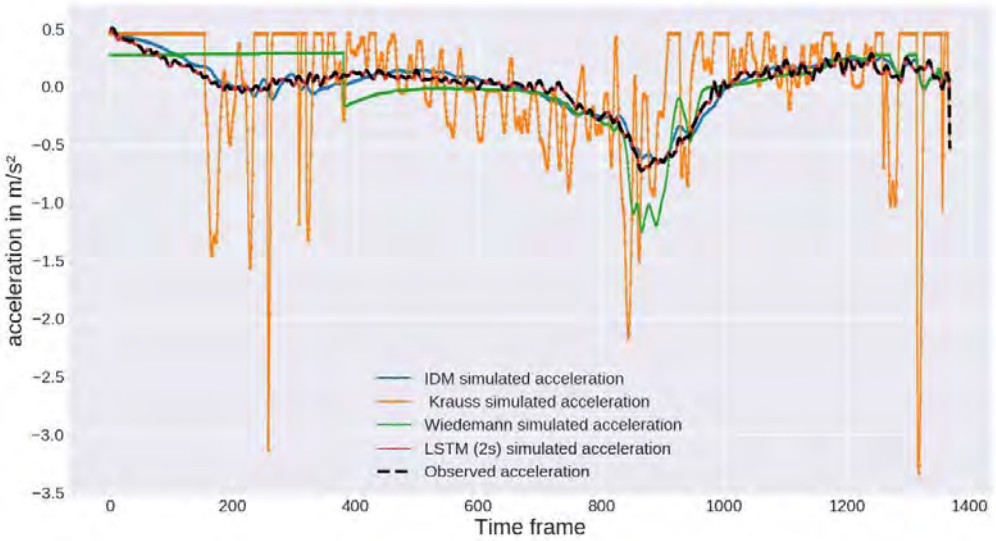



Figure 7.26.: Simulated acceleration of the four car-following models against the observed acceleration for trace CF_613037

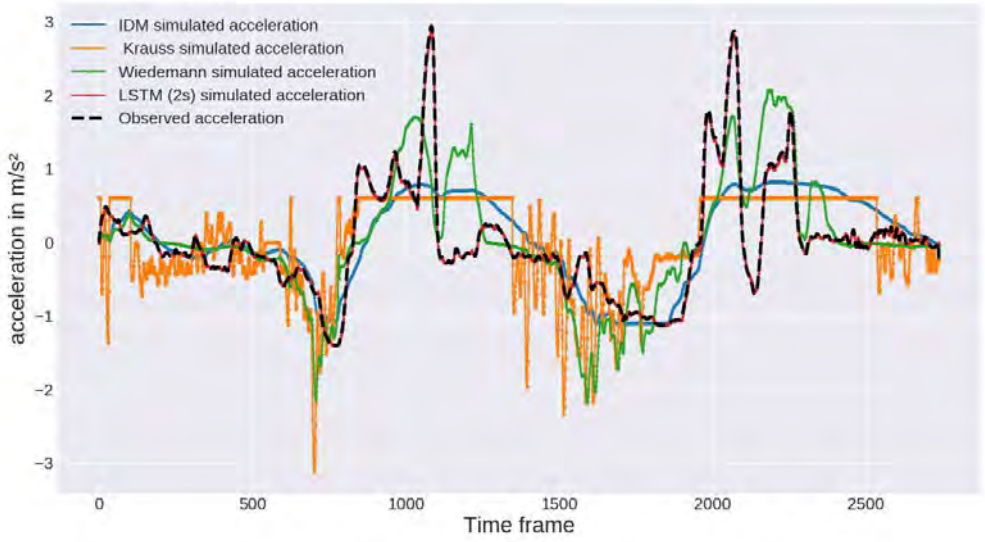



Figure 7.27.: Simulated acceleration of the four car-following models against the observed acceleration for trace CF_785485

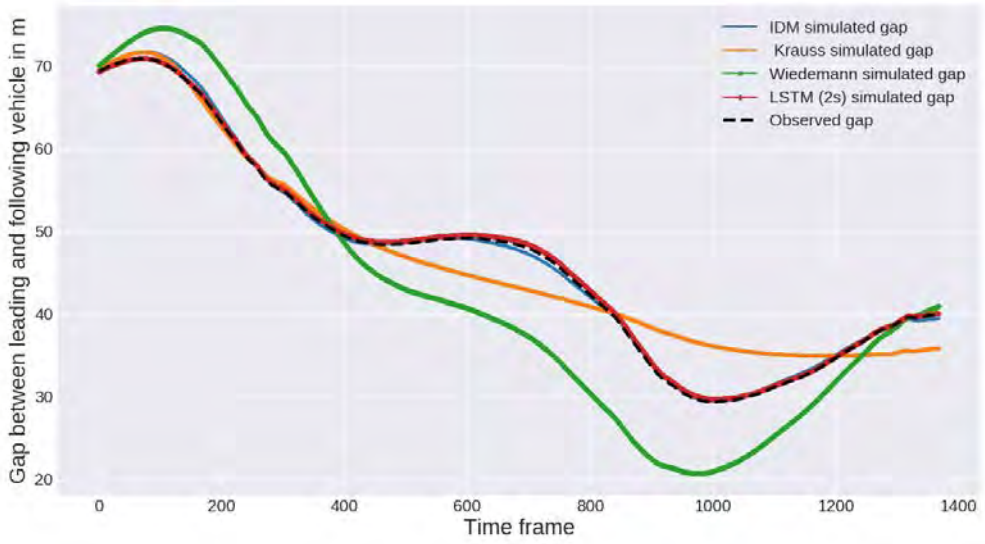



Figure 7.28.: Simulated gap between following and leading vehicle of the four car-following models against the observed gap for trace CF_613037

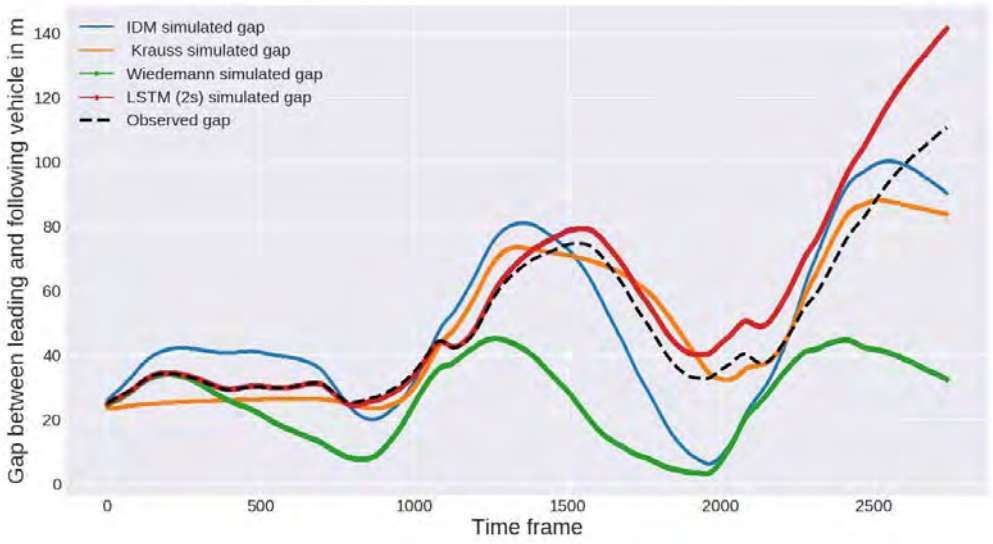



Figure 7.29.: Simulated gap between following and leading vehicle of the four car-following models against the observed gap for trace CF_785485

## 7.4. Main Findings

Firstly, the calibrated conventional car-following models, namely the Krauss model, IDM and the Wiedemann model, are not able to perfectly simulate the observed car-following traces extracted from the raw xFCD data. The conventional car-following models show problems when simulating the oscillations in the observed car-following data, meaning by the conventional car-following model are not fit well to the observed data if the data have sharp changes in the acceleration or the velocity profiles. Secondly, the LSTM based car-following model outperforms all the conventional car-following models. The comparison of the LSTM based car-following model with 40 ms, 1 s, 2 s and 3 s input sequence length shows that the driver's behaviour in car-following situation is dependent on the historical manoeuvres/traffic information. The optimal value of the input sequence length to model car-following behaviour using the LSTM neural network is 2 s for the dataset used in this study. The presence of a vehicle on the left lane or the right lane of the following vehicle does not play any role as per the data used in this study whereas the most important features found are acceleration, velocity of the following vehicle and the relative velocity.

# 8. Conclusion

Extended Floating Car Data (xFCD) is a data collection technique which also captures the naturalistic driver behaviour. This data can be used to model the data-driven car-following models close to the realistic natural behaviour of different drivers. However, the way the data is stored in the raw xFCD data is not straight forward. The methodology presented in this study to extract the car following models works well. The analysis presented in chapter 6 on a sample trace shows that the methodology successfully extracts out the car-following traces out of raw xFCD.

The analysis of the three calibrated conventional car-following models summarises that the Wiedemann model performs worst in simulating the naturalistic driving behaviour. The reason can be the large number of parameters of Wiedemann model that have to be calibrated. Apart from this the Krauss model and the IDM model show a comparable performance. For some traces, the performance of the IDM is better than the other two while for other the performance of the Krauss model is best. The real-coded genetic algorithm used in this study seems to successfully calibrate the three car-following models. As, despite the errors in replicating the velocity, acceleration, gap and trajectory, the plots of the simulated results show that the models are capable to follow the trends. The models are hard to calibrate if the traces contain sharp changes in the acceleration or velocity profile of the following vehicle.

The LSTM based car-following model shows a significant increase in performance when compared to the conventional car-following models. The input sequence of 40 ms, 1 s, 2 s and 3 s, when used to train the LSTM based car-following model, shows that the driver behaviour is actually dependent on the historical data, here it means the memory. The LSTM model with an input sequence of 2 s produces best results in this study. Apart from this, the feature importance shows that the acceleration of the following vehicle, velocity of the following vehicle, the relative velocity and the gap between the following the leading vehicle are the most important features, whereas the time headway does not seem to be that important. The results also show that the presence of the vehicle on the left and right side of the following vehicle are not significant. This is because of the lack of data available for these two cases during the car-following scenario in the data used in this study. The performance of the LSTM model in predicting the acceleration is best amongst all the other models, whereas for a few traces, the Krauss model performance is comparable in simulating the gap between the vehicles.

### 8.0.1. Limitations and Future Work

The presented research is not without its own limitations. The data used to train the LSTM based car-following model is explicitly from the Audi fleet. Since the performance of the data-driven car-following model is highly dependent on the kind of the dataset they are trained on. The model needs validation with different kinds of datasets. Also, the LSTM based car-following model is not integrated into any of the traffic simulation software. The model also needs analysis and validation of its performance when used in the traffic simulation software.

Car-following is the sub-scenario of free flow driving. In this study, only the traces of the car-following are considered during the training of the LSTM based car-following model. So, for future work, the LSTM based car-following model needs to be developed for the free flow driving conditions to be successfully integrated into any traffic simulation software.

The evaluation of the performance of the LSTM based car-following model to replicate the traffic flow theory is also not done in this study which can be considered as the future work in this direction.

# List of Figures

# List of Tables

# Bibliography


Afshine, A., & Shervine, A. (2020). *Recurrent neural networks cheatsheet*. Retrieved December 30, 2019, from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Aghabayk, K., Sarvi, M., & Young, W. (2015). A State-of-the-Art Review of Car-Following Models with Particular Considerations of Heavy Vehicles. *Transport Reviews*, *35*(1), 82–105. https://doi.org/10.1080/01441647.2014.997323

Astarita, V., Giofré, V. P., Festa, D. C., Guido, G., & Vitale, A. (2020). Floating car data adaptive traffic signals: A description of the first real-time experiment with "connected" vehicles. *Electronics (Switzerland)*, *9*(1). https://doi.org/10.3390/electronics9010114

Bando, M., Hasebe, K., Nakayama, A., Shibata, A., & Sugiyama, Y. (1995). Dynamical model of traffic congestion and numerical simulation. *Phys. Rev. E*, *51*, 1035–1042. https://doi.org/10.1103/PhysRevE.51.1035

Chen, W. (2015). *Vehicular communications and networks: Architectures, protocols, operation and deployment*. Woodhead Publishing.

Chong, L., Abbas, M. M., Medina Flintsch, A., & Higgs, B. (2013). A rule-based neural network approach to model driver naturalistic behavior in traffic. *Transportation Research Part C: Emerging Technologies*, *32*, 207–223. https://doi.org/10.1016/j.trc.2012.09.011

De Jong, K. (2012). *Evolutionary computation: A unified approach*. https://doi.org/10.1145/2330784.2330914

Deb, K. (2012). *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd.

Deb, K., & ayan Deb. (2014). Analysing mutation schemes for real-parameter genetic algorithms. *International Journal of Artificial Intelligence and Soft Computing*, *4*(1), 1. https://doi.org/10.1504/ijaisc.2014.059280

Deb, K., & Bhushan Agrawal, R. (1995). Simulated Binary Crossover for Continuous Search Space, *9*, 115–148. https://doi.org/10.1.1.26.8485Cached

Dobnikar, A., Steele, N. C., Pearson, D. W., Albrecht, R. F., Deb, K., & Agrawal, S. (1999). A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms. *Artificial Neural Nets and Genetic Algorithms*, 235–243. https://doi.org/10.1007/978-3-7091-6384-9_40

FHWA. (n.d.). Ngsim–next generation simulation. http://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm.

Gazis, D. C., Herman, R., & Rothery, R. W. (1961). Nonlinear Follow-the-Leader Models of Traffic Flow. *Operations Research*, *9*(4), 545–567. https://doi.org/10.1287/opre.9.4.545

Gerdes, A. (2006). Automatic maneuver recognition in the automobile: the fusion of uncertain sensor values using bayesian models. *Proceedings of the 3rd International Workshop on Intelligent Transportation (WIT 2006)*, 129–133. http://elib.dlr.de/22833/01/AU%7B%5C_%7DFS%7B%5C_%7DManeuverRecognition%7B%5C_%7DGerdes%7B%5C_%7D060316.pdf

Gipps, P. G. (1980). Gipps_ABehaviouralCarFollowingModel.

Gladyshev, P., & PA, A. (2005). Finite State Machine Analysis of a Blackmail Investigation. *International Journal of Digital Evidence*, *4*(1), 1–13. http://www.formalforensics.org/publications/Finite%20State%20Machine%20Analysis%20of%20a%20Blackmail%20Investigation.pdf

Hao, S., Yang, L., & Shi, Y. (2018). Data-driven car-following model based on rough set theory. *IET Intelligent Transport Systems*, *12*(1), 49–57. https://doi.org/10.1049/iet-its.2017.0006

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hülnhagen, T., Dengler, I., Tamke, A., Dang, T., & Breuel, G. (2010). Maneuver recognition using probabilistic finite-state machines and fuzzy logic. *IEEE Intelligent Vehicles Symposium, Proceedings*, 65–70. https://doi.org/10.1109/IVS.2010.5548066

Jia, H., Juan, Z., & Ni, A. (2003). Develop a car-following model using data collected by "five-wheel system
. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, *1*, 346–351. https://doi.org/10.1109/ITSC.2003.1251975

Kesting, A. (2007). Microscopic modeling of human and automated driving: Towards traffic-adaptive cruise control. *Physical Review E*, (February), 218. http://www.qucosa.de/urnnbn/urn:nbn:de:bsz:14-ds-1204804167720-57734%7B%5C%%7D5Cnhttp://www.qucosa.de/fileadmin/data/qucosa/documents/842/1204804167720-5773.pdf

Khodayari, A., Ghaffari, A., Kazemi, R., & Braunstingl, R. (2012). A modified car-following model based on a neural network model of the human driver effects. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, *42*(6), 1440–1449. https://doi.org/10.1109/TSMCA.2012.2192262

Kikuchi, S., & Chakroborty, P. (1992). Car-following model based on fuzzy inference system.

Kingma, D. P., & Ba, J. L. (2015). ADAM: A method for stochastic optimization, arXiv arXiv:1412.6980v9, 1–15.

Krajewski, R., Bock, J., Kloeker, L., & Eckstein, L. (2018). The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems, In *2018 21st international conference on intelligent transportation systems (itsc)*. https://doi.org/10.1109/ITSC.2018.8569552

Krauss, S. (1998). Microscopic modeling of traffic flow: investigation of collision free vehicle dynamics. *Forschungsbericht - Deutsche Forschungsanstalt fuer Luft - und Raumfahrt e.V.*, (98-8).

Kuan, C.-m. (2002). Lecture on Markov Regime Switching Models.

Kuri-morales, A. F. (2014). LNAI 8857 - The Best Neural Network Architecture, (1), 72–84.

Lee, G. (1966). A Generalization of Linear Car-Following Theory. *Operations Research*, *14*(4), 595–606. https://doi.org/10.1287/opre.14.4.595

Lee, S., Ngoduy, D., & Keyvan-Ekbatani, M. (2019). Integrated deep learning and stochastic car-following model for traffic dynamics on multi-lane freeways. *Transportation Research Part C: Emerging Technologies*, *106*(July), 360–377. https://doi.org/10.1016/j.trc.2019.07.023

Lidbe, A. D., Hainen, A. M., & Jones, S. L. (2017). Comparative study of simulated annealing, tabu search, and the genetic algorithm for calibration of the microsimulation model. *Simulation*, *93*(1), 21–33. https://doi.org/10.1177/0037549716683028

Ma, X., & Andréasson, I. (2007). Behavior measurement, analysis, and regime classification in car following. *IEEE Transactions on Intelligent Transportation Systems*, *8*(1), 144–155. https://doi.org/10.1109/TITS.2006.883111

May, A. D. (1990). *Traffic flow fundamentals*.

Mitra, P., & Eric, B. (2018). Calibration and evaluation of car following models using real-world driving data. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, *2018-March*, 1–6. https://doi.org/10.1109/ITSC.2017.8317836

Olah, C. (2019). *Understanding lstm networks*. Retrieved December 10, 2019, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Olstam, J. J., & Tapani, A. (2004). Comparison of car-following models for simulation. *Transportation Research Record*, (1678), 116–127. https://doi.org/10.3141/1678-15

Ozaki, H. (1993). Reaction and anticipation in the car-following behavior.

Panwai, S., & Dia, H. (2007). Neural agent car-following models. *IEEE Transactions on Intelligent Transportation Systems*, *8*(1), 60–70. https://doi.org/10.1109/TITS.2006.884616

Papathanasopoulou, V., & Antoniou, C. (2015). Towards data-driven car-following models. *Transportation Research Part C: Emerging Technologies*, *55*, 496–509. https://doi.org/10.1016/j.trc.2015.02.016

Pipes, L. A. (1953). An operational analysis of traffic dynamics. *Journal of Applied Physics*, *24*(3), 274–281. https://doi.org/10.1063/1.1721265

PTV Vissim. (2011). VISSIM 5.30-05 User Manual, 130–132. https://www.et.byu.edu/%7B~%7Dmsaito/CE662MS/Labs/VISSIM%7B%5C_%7D530%7B%5C_%7De.pdf

Roesener, C., Fahrenkrog, F., Uhlig, A., & Eckstein, L. (2016). A scenario-based assessment approach for automated driving by using time series classification of human-driving behaviour. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 1360–1365. https://doi.org/10.1109/ITSC.2016.7795734

Saifuzzaman, M., Zheng, Z., Mazharul Haque, M., & Washington, S. (2015). Revisiting the Task-Capability Interface model for incorporating human factors into car-following models. *Transportation Research Part B: Methodological*, *82*, 1–19. https://doi.org/10.1016/j.trb.2015.09.011

Sastry, K., Goldberg, D., & Kendall, G. (2005). Chapter 4 Genetic Algorithms. *Search Methodologies*, 97–125.

Steger, C. (1996). Extracting curvilinear structures: A differential geometric approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1064*, 630–641. https://doi.org/10.1007/bfb0015573

Tang, J., Liu, F., Zhang, W., Ke, R., & Zou, Y. (2018). Lane-changes prediction based on adaptive fuzzy neural network. *Expert Systems with Applications*, *91*, 452–463. https://doi.org/10.1016/j.eswa.2017.09.025

Tango, F., & Botta, M. (2009). ML techniques for the classification of car-following maneuver. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5883 LNAI*(1050), 395–404. https://doi.org/10.1007/978-3-642-10291-2_40

Treiber, M., & Kesting, A. (2013). *Traffic Flow Dynamics*. https://doi.org/10.1007/978-3-642-32460-4

Vinagre Díaz, J. J., Fernández Llorca, D., Rodríguez González, A. B., Quintero Mínguez, R., Llamazares Llamazares, Á., & Sotelo, M. Á. (2012). Extended floating car data system: Experimental results and application for a hybrid route level of service. *IEEE Transactions on Intelligent Transportation Systems*, *13*(1), 25–35. https://doi.org/10.1109/TITS.2011.2178834

Wiedemann, R., & Reiter, U. (1992). Microscopic traffic simulation: The simulation system mission, background and actual state.

Won, J., Lee, S., Lee, S., & Kim, T. H. (2006). Establishment of Car Following Theory Based on Fuzzy-Based Sensitivity Parameters (T.-J. Cham, J. Cai, C. Dorai, D. Rajan, T.-S. Chua, & L.-T. Chia, Eds.), 613–619.

Yang, D., Zhu, L., Liu, Y., Wu, D., & Ran, B. (2019). A Novel Car-Following Control Model Combining Machine Learning and Kinematics Models for Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *20*(6), 1991–2000. https://doi.org/10.1109/TITS.2018.2854827

Yang, X. S. (2011). Optimization algorithms. *Studies in Computational Intelligence*, *356*, 13–31. https://doi.org/10.1007/978-3-642-20859-1_2

Zaky, A. B., Gomaa, W., & Khamis, M. A. (2016). Car following Markov regime classification and calibration. *Proceedings - 2015 IEEE 14th International Conference on Machine Learning and Applications, ICMLA 2015*, 1013–1018. https://doi.org/10.1109/ICMLA.2015.126

Zhang, Q., Xie, Q., & Wang, G. (2016). A survey on rough set theory and its applications. *CAAI Transactions on Intelligence Technology*, *1*(4), 323–333. https://doi.org/https://doi.org/10.1016/j.trit.2016.11.001

Zhang, Y., Ni, P., Li, M., Liu, H., & Yin, B. (2017). A new car-following model considering driving characteristics and preceding vehicle's acceleration. *Journal of Advanced Transportation*, *2017*. https://doi.org/10.1155/2017/2437539

Zhao, H., He, R., & Ma, C. (2018). An Extended Car-Following Model at Signalised Intersections. *Journal of Advanced Transportation*, *2018*. https://doi.org/10.1155/2018/5427507

Zhou, M., Qu, X., & Li, X. (2017). A recurrent neural network based microscopic car following model to predict traffic oscillation. *Transportation Research Part C: Emerging Technologies*, *84*, 245–264. https://doi.org/10.1016/j.trc.2017.08.027

Zhu, M., Wang, X., & Wang, Y. (2018). Human-like autonomous car-following model with deep reinforcement learning. *Transportation Research Part C: Emerging Technologies*, *97*, 348–368. https://doi.org/10.1016/j.trc.2018.10.024

Zhu, M., Wang, X., Tarko, A., & Fang, S. (2018). *Modeling car-following behavior on urban expressways in Shanghai: A naturalistic driving study* (Vol. 93). https://doi.org/10.1016/j.trc.2018.06.009

# A. Appendix

## A.1. The Wiedemann Model Acceleration Calculations

Figure A.1 shows the flow chart of the calculation process of the acceleration of the Wiedemann Model.

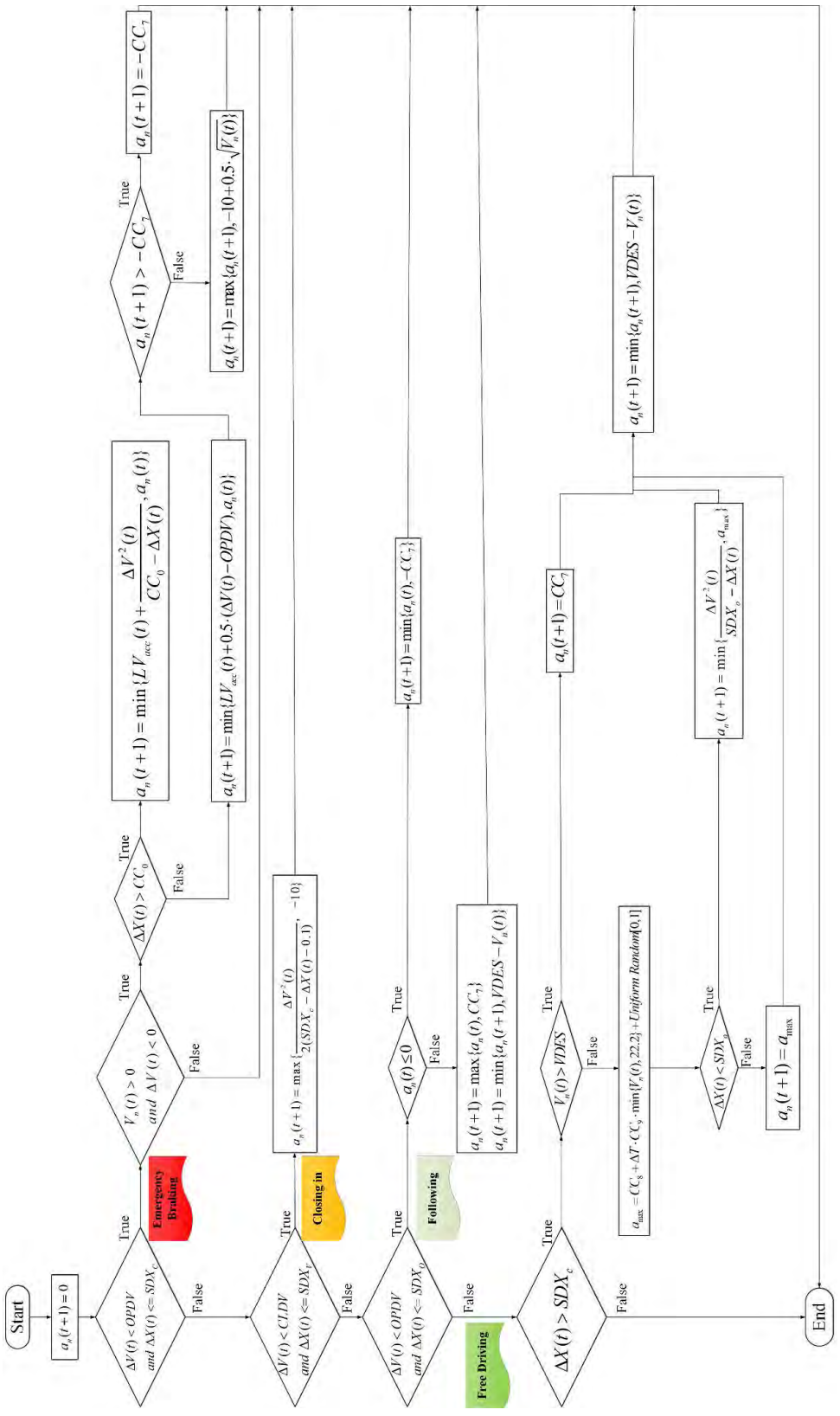



Figure A.1.: The flow chart of the calculation process of the Wiedemann model (Zhu, Wang, Tarko, et al., 2018)