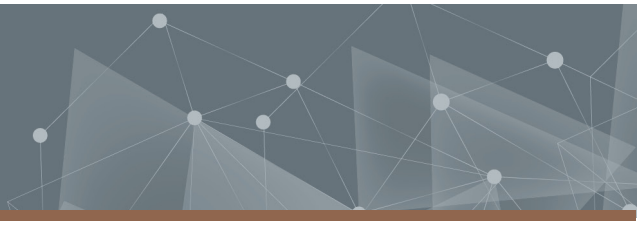
CHALMERS
UNIVERSITY OF TECHNOLOGY



Driving Scenario Generation Using Generative Adversarial Networks

Master's thesis in Systems, Control, and Mechatronics

Martin Håkansson
Joel Wall

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Driving Scenario Generation Using Generative Adversarial Networks

MARTIN HÅKANSSON
JOEL WALL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

Driving Scenario Generation Using Generative Adversarial Networks
MARTIN HÅKANSSON

JOEL WALL

Supervisor: Mohammad Hossein Moghaddam, Department of Electrical Engineering
Majid Khorsand Vakilzadeh, Zenseact
Ghazaleh Panahandeh, Zenseact
Examiner: Jiajia Chen, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical engineering Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Abstract

There is a huge interest today to move towards highly autonomous vehicles. To establish a framework that ensures robustness and reach the highly set qualifications set by authorities to allow autonomous vehicles on roads, scenario-based verification will be an important tool. In order to further diversify, augment and improve their already established framework, car manufacturers and software companies wish to investigate the use of generative machine learning methods to model new scenarios. In this thesis, a *Generative Adversarial Network*, modified to handle time series data, is used to generate trajectories of lane changes in a highway environment. The generated data manages to visually and to a great extent, statistically capture the properties of the real data. In order to quantify the performance of the proposed framework, the generated data is also compared to data generated using related machine learning approaches. The proposed framework however outperforms the bench-marking methods in almost every evaluation metric. Even though results are promising, a lot of work is still to be done in order to provide a robust tool chain that generates driver-like behaviour of great use within scenario-based verification.

Time-series, Generative adversarial network, Autonomous driving, Deep Learning, Neural Networks, Scenario-based verification

Acknowledgements

Firstly, we would like to thank Zenseact, and the supervisors Ghazaleh Panahandeh and Majid Khorsand Vakilzadeh who proposed the thesis scope and ensured that we stayed on track through out the process. We are also grateful for the help and feedback provided by Mohammad Hossein Moghaddam from the academic side of the thesis supervision from Chalmers. Lastly we want to thank all the employers at Zenseact that we have reached out to that have happily helped us with various challenges throughout the project.

Martin Håkansson, Gothenburg, June 2021
Joel Wall, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

The automotive industry is in the race of developing different levels of autonomous vehicles and a key component of reaching these levels of self-driving are machine learning algorithms. These algorithms are relevant to the decision making of vehicles but lately, studies have been made were machine learning algorithms have been used for verification purposes and even diversification of the data needed to train the decision making algorithms. In order for autonomous vehicles to finally be allowed on open roads, autonomous driving will first have to outperform human driving. For this reason, new ways of verification and testing to further improve decision making algorithms is of high relevance. This report aims to focus on the implementation of machine learning solutions to the verification part of autonomous drive.

1.1 Background

In the automotive industry today there is a huge interest in moving towards autonomous driving. Autonomous driving is today seen as an important step for the future of transportation. To be able to take this step, it is crucial to have an extremely safe and reliable solution. Verifying such a solution have been shown to require hundreds of millions kilometers test driving [1]. This is clearly not feasible due to the time and effort required. One way to combat this issue is using so called scenario-based verification. In scenario-based verification, a database of scenarios that the vehicle of study get exposed to in the field, is used to test autonomous driving functionality [2]. Due to the power of a large and up to date scenario database, there is a large interest in the automotive industry to keep developing tools to establish and expand such a database.

There are multiple ways of creating the scenarios needed for a database. One simple but time consuming alternative is to extract and label driving scenarios from raw sensor data [2]. This also has the advantage of giving full control throughout the process. Since the extraction of scenarios is based on explicit rules it is subject to missing unknown cases. A complementary approach to the explicit rule-based extraction of scenarios is a machine learning approach. A machine learning approach allows for generation of new data as well as giving an opportunity to generate diverse and potentially safety critical scenarios. To be able to utilize the machine learning approach it is crucial to find a method that allows for mimicking the variation in the collected data and with that creating realistic synthetic scenarios.

To accomplish this in the thesis work, we investigate the usage of generative deep machine learning tools for this purpose. Primarily different variations of *Vari-*

ational Auto-Encoders (VAE) and *Generative Adversarial networks* (GAN) have shown promise when it comes to generating data. For example GAN were introduced [3] for image generation and VAE [4] have also shown great promise within the field of image generation. However, both methods have later been developed and utilized for generating sequential sensor data.

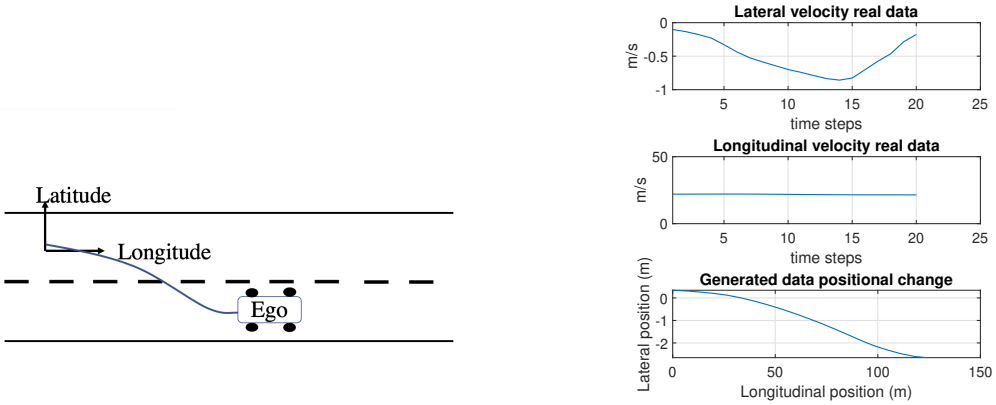
1.2 Previous Studies

The properties of sensor data differs from images. Most significantly sensor data are sequential and each time step depend on the prior time steps. Capturing temporal aspects has been one of the main topics in *Natural Language Processing* (NLP) and has been successfully managed using Recurrent Neural Networks and LSTM cells. To handle the temporal property of the scenarios a version of GAN that can handle time series is necessary. Some of the most successful studies of time-series generation seem to take inspiration from NLP and base their GAN network structure of LSTM-cells. A few options that has done this is *Auto-Encoder* GAN (AE-GAN) [5] and *Recurrent conditional* GAN (RC-GAN) [6]–[8]. Another state-of-the art version of GAN that has been used for time series generation is *DoppelGANger* (DG) [9] which also uses similar structure to original RC-GAN [6] but also introduces some training features that significantly improves the generated sequences. Some research has also been done within the field of driving scenario time series [8], [10], [11].

VAEs has also shown promise in capturing time-series properties [12]–[14]. These studies however focus on classification for time series. In the RC-GAN paper [6] a Recurrent VAE (RVAE) is used and trained to generate sine-wave data, these results are mentioned as poor by the authors and are not presented in the paper. A RVAE is used by Ding, Wang, and Zhao [15] with great result in generating trajectories of vehicle encounters.

1.3 Problem Description

The aim of this thesis project is to generate realistic scenarios of driving behaviour in the category of lane changes in a highway setting. A typical lane change carried out by an ego vehicle and its features are illustrated in Figures 1.1a and 1.1b. These scenarios should consist of time series with at least the observations of velocity in lateral and longitudinal direction. However, the generated data should from this be able to capture underlying properties such as acceleration in lateral and longitudinal direction, the positional changes and also the steering angle of the vehicle. All parameters are referenced to the lane that the ego-vehicle is driving in. The goal is to generate scenarios that represents the distribution of the real dataset, and by doing that, create data that can further increase the robustness of the testing and verification of autonomous driving.



(a) Illustration of a typical lane change scenario referenced from the centre of initial lane and the start of the scenario.

(b) Top is lateral velocity, middle is longitudinal velocity and bottom is the derived position.

Figure 1.1: Illustration of a highway lane change in highway environment with corresponding features of study that constitutes a lane change.

1.4 Limitations

Since the aim of the project is to create trajectories of the ego-vehicle, there will not be any mentioning of target vehicles. The aim of the project is also to do the generation for highway scenarios, which means that no data outside of highways will be used. Another important limitation is that no non machine learning approach will be used as a main method to generate driving scenarios. No other data than that supplied by Zenseact will be used to train the algorithms.

1.5 Thesis Outline and Contributions

This thesis is made up of a theory section where the fundamentals of the Generative Adversarial Network will be presented, from that further developments of GAN and further work more closely related to the problem description of this report will be presented. In the Method chapter, all implementation steps and contributions made in this report will be presented along with the extensive evaluation process of the GAN. In the result and discussion section, all findings and evaluations will be presented and discussed. In the conclusion section, the relevance and possible ramifications of this work will be presented.

2
Theory

In order to solve the objectives, an extensive literature study is needed to identify relevant methods for addressing the problem, but also for finding the state-of-the-art model that are used within these methodologies in order to achieve desired results. First, GANs will be explored. After that, other relevant machine learning methodologies will be explained in order to grasp the holistically of what makes a sequential generative model state-of-the-art. Specifically the RCGAN [6] will be presented as well as an improved variant of it, DG [9].

2.1 Generative Adversarial Networks

Generative Adversarial Networks is a ground breaking generative method in the area of deep learning. GANs were introduced by Goodfellow, Pouget-Abadie, Mirza, *et al.* [3] in 2014 and consists of two competing neural networks.

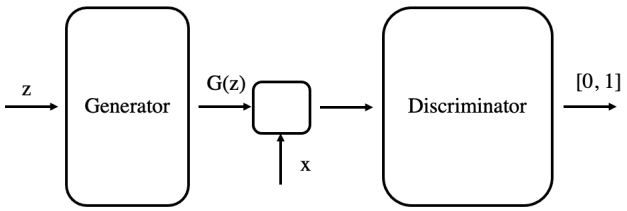


Figure 2.1: A schematic view of a GAN. The generator takes a noise vector z as input and maps this to $G(z)$. The discriminator then receives input that are either x (real data) or $G(z)$ (generated data). The discriminator then outputs a prediction if the input data is either fake (0) or real (1). These outputs are used to train the network.

One of the networks, the generator maps a random input noise vector, z to a data space. The goal of the generator is to map z as similar to an actual data, x as possible. It achieves this by the function $\mathbf{G}(z, \sigma_g)$ where $\mathbf{G}$ is a multi layer neural network and σ_g represents its parameters[3]. For the second network, the discriminators, role is to estimate the probability that the x actually came from the data instead of as an output of $\mathbf{G}$(z). This is done by the function $\mathbf{D}(x, \theta_d)$ where $\mathbf{D}$ is a multi layer neural network and θ_d represents its parameters [3]. Training the discriminators, the

goal is to assure that the discriminator makes the correct estimation on whether the input is the original data, x or the generated data $\mathbf{G}(z)$. The generator is trained to minimize equation (2.1) when fed with generated data [3]. I.e. the generator is trained to make the discriminator predict the generated data as real data.

$$\log(1 - D(G(z))) \tag{2.1}$$

Having two networks for this task is where the two networks compete in a minimax game described in equation (2.2) [3]. This kind of training is called adversarial training [16]

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.2}$$

The objective of the GAN is to pull the distribution of the generated data towards the distribution of the real data using the loss function. When the discriminator performs optimally the loss function in equation (2.2) could be derived to be equal to the Jensen-Shannon divergence that is described in equation (2.3) [3].

$$JS_{div}(P||Q) = \frac{1}{2}K(P||M) + \frac{1}{2}K(Q||M) \tag{2.3}$$

where $M = \frac{1}{2}(P + Q)$ and K is the KL divergence [17]. The KL divergence can be calculated in the following way.

$$K(P||Q) = \sum_i P(i) log\left(\frac{P(i)}{Q(i)}\right) \tag{2.4}$$

KL divergence is a maximum likelihood method and will go towards infinity if the supports for the two distributions does not overlap. When combining different KL divergences for the JS divergence the same case with no overlap of the distributions gives the logarithm of two. This is the cause of some challenges in adversarial training using the loss function suggested in (2.2).

2.1.1 Challenges with Adversarial Training

The main challenge occurring when using the loss function (2.2) is that it is possible to saturate the discriminator resulting in that the generator gets no meaningful feedback [18]. This could further be explained by the behaviour of the JS divergence described previously. When there is no significant overlap between two distributions the JS divergence will become constantly log two [19]. A constant result means that the gradient over the loss function will be zero and the network will not learn anything. This is what happens when the discriminator in a regular GAN is trained to optimality. Further it can be explained by looking at the probability distributions. When the discriminator improves and not the generator the joint support for real and fake data becomes smaller [19]. If this joint support becomes negligible and updates from the generator does not change that, then there is a discriminator win case [20].

2.1.2 Wasserstein Loss

To address the issue that occurs when using JS as a method to match distributions, a new metric is introduced in the *Wasserstein GAN* paper [19]. The authors propose a metric for comparing distributions which they call *Earth-Mover distance* or *Wasserstein distance*. Wasserstein distance differs from KL and JS since it manages to quantify the distance even when the compared distributions do not have a non-negligible intersection [19]. How the Wasserstein distance is computed is illustrated in equation (2.5). Incorporating Wasserstein distance for GANs allows us to always have a meaningful feedback to the generator, since the gradient does not become zero [19].

$$W\left(\mathbb{P}_r, \mathbb{P}_g\right) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2.5}$$

In the Wasserstein proposed GAN, the discriminator network is replaced with a critic which differs to the discriminator mainly in the fact that it does not output a probability-prediction of fake or real, as the network in Figure 2.1, but instead computes the Wasserstein distance between the two distributions as in equation (2.6) [19].

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}\left[f_w(x)\right] - \mathbb{E}_{z\sim p(z)}\left[f_w\left(g_\theta(z)\right)\right] \tag{2.6}$$

The loss function in (2.6) describes the Wasserstein distance between the generator and the critic. Having a critic instead of a discriminator has a lot positive consequences when it comes to the training of the network. The greatest benefit of using the critic is that it is not possible to saturate a critic, since it now is a distance instead of the probability of a sample being fake or not that the critic outputs. This property results in that the generator always gets meaningful feedback and that it is always beneficial to have a better performing critic. Since it is always better to have a better critic, there is no longer a need to balance the training. This simplifies tuning of the network significantly.

2.2 Recurrent Neural Networks

For dealing with sequential data, which time series are, a class of neural networks called *Recurrent Neural Networks* (RNN) [21] has shown promising results when applied to attempt to generalize time dependent properties [13], [8], [9], [7], [5]. A RNN takes the previous hidden state h_{t-1} and the current input x_t in to account. This process is illustrated in Figure 2.2. The update of the current state is then derived according to (2.7) where W and U are weight matrices and b is bias [21].

$$a(t) = b + Wh(t-1) + Ux(t) \tag{2.7}$$

The update is then calculated using a tanh function as in equation (2.8)[21].

$$h(t) = tanh(a(t)) \tag{2.8}$$

In order to then calculate the hidden-to-output of the RNN equation a further layer is derived with bias c and weights V [21].

$$o(t) = c + Vh(t) \tag{2.9}$$

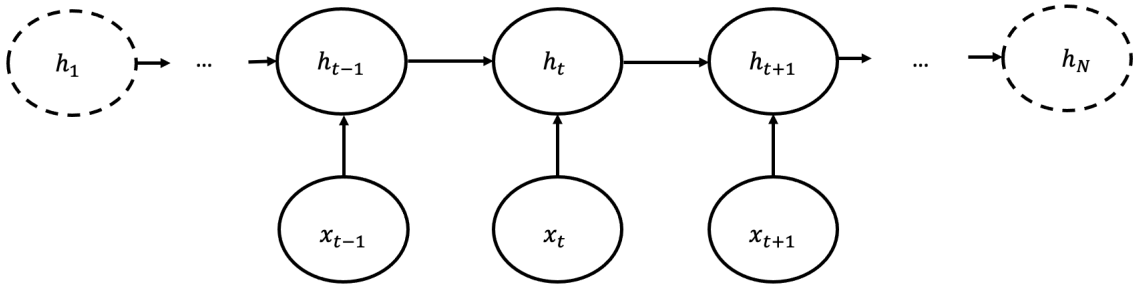


Figure 2.2: A schematic view of time steps in a RNN where each RNN cell x_i ouputs a corresponding time step h_t

on top of the final linear layer a softmax function is used. [21]

$$\hat{y} = softmax(o(t)) \tag{2.10}$$

RNNs uses recurrent back-propagation through time to update its weights in the network. In an RNN this is done by unrolling the time steps. If a time series consists of multiple steps, this will cause a computational issue since all time steps has the previous time steps as input. Calculating the gradients with this dependency of time steps will have an exponential effect, since the jacobian of the previous weights will be multiplied with each other as many times as there are time steps in the time series [21]. This will cause the gradient to explode, if the gradients are larger than absolute 1 or make the gradient vanish, if the gradients are smaller than absolute 1.

2.2.1 Long Short-term Memory

Long Short-term Memory (LSTM) [22] uses more layers in each time-step than the vanilla RNN and introduces a cell that controls what parts of the history that is relevant to store by weighting the previous history. This history cell $C(t)$ which consist of an input gate and a forget gate is then used to derive the hidden state $h(t)$ [21], deriving the hidden state is shown in equations (2.11)-(2.12).

$$C(t) = f(t)C(t-1) + i_t\tilde{C}_t \tag{2.11}$$

In this case f is the forget gate, i the input gate and $\tilde{C}$ is the new information [21],

$$h(t) = o(t)tanh(C(t)) \tag{2.12}$$

2.2.2 Gated Recurrent Units

Gated Recurrent Units (GRU) are similar to LSTMs cell but only consists of a single step update which should make them less computationally heavy. GRUs were introcuced in 2014 [23]. What makes GRUs different from LSTMs is mainly that GRUs has one gate that concurrently controls the decision and the forget task in

the state [21]. To derive the hidden state in a GRU, see equation (2.13) where u is the update gate and r is a reset gate used to control what information is kept [21].

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + \left(1 - u_i^{(t-1)}\right) \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)} \right) \quad (2.13)$$

2.3 State-of-the-art GANs for Time Series Generation

In the previous work section, different methods of generating time series was introduced. What is common for all these methods is that they all have recurrent components to its structures. Since this method have shown the most promise when it comes to handling temporal aspects. The two main network structures that are used are the RCGAN and the AE-GAN. In this section, the different methodologies of GANs will be explained more in detail.

2.3.1 Challenges in Time Series Generation

Most machine learning models requires a fixed input size, so also RNNs. This becomes an issue since time series differ in length in most applications. All solutions of this will in some sense impact the performance of the network. There is two main challenges with generating variable length time series. Firstly, the distribution of the lengths of the generated data should match the real data. Secondly, a generated synthetic scenario S_s, should ideally be ended at a value that is realistic for the time series generated. Existing models tends to handle variable length in different ways. A solution could simply be to either pad by interpolation or end-pad sequences. Interpolation removes the equidistant property of the data and would require another solution for the time dependency. By end padding scenarios by either zeros or the final value, the network gets additional information and some solution is necessary for when the scenario ends. Training the model on single batches could also be a solution, but would bring the negative consequence of being computationally heavy.

An important challenge with generating time series is to handle the temporal aspect of the data well. The main solution to this problem is using RNNs. The RNNs does though still struggle a little bit with capturing long term dependencies. Another solution that could improve the handling of long term dependencies is down sampling the data. This helps the network by reducing the need for long term dependencies.

In all forms of GANs it is a common problem with mode collapse. Mode collapse is when the network only generates samples that correspond to one part of the real distribution. It can also result in that the network is over fitted to noise in individual samples. One simple solution to this problem is using more data, but since data can be hard to come by it is not always an efficient solution. When studying state-of-the-art models it is relevant to take into consideration how they approach these three issues.

2.3.2 Auto-encoder GANs

One of the commonly used architectures for generating time series is a combination of the Auto Encoder [4] and the GAN [3], denoted as AE-GAN. The idea behind AE-GAN is based on the fact that GANs are difficult to train and when the complexity and dimensions of the data to reproduce is increased the output of the GAN can be expected to be of various results [24]. The AE-GAN approach to this problem is to use a encoder to decompose and learn the components of the data into a simpler product, a latent space representation. The GAN is then trained to generate latent space representations that the decoder part of the auto-encoder reconstructs to represent the original distribution. The idea is that the latent space representation is easier to generalize for a GAN than an entire time series. [24] uses this approach for text generation while [8] and [5] uses this approach for time series generation. The principle of the AE-GAN architecture is illustrated in Figure 2.3.
The AE-GAN with its latent space representation presents an opportunity to handle time series of various lengths. Demetriou, Alfsvåg, Rahrovani, *et al.* [8] trains a separate network to estimate the length of the sequence based on the features of the latent space representation. In this existing solutions however, no clear evidence is shown that solutions for the two challenges are successfully provided.

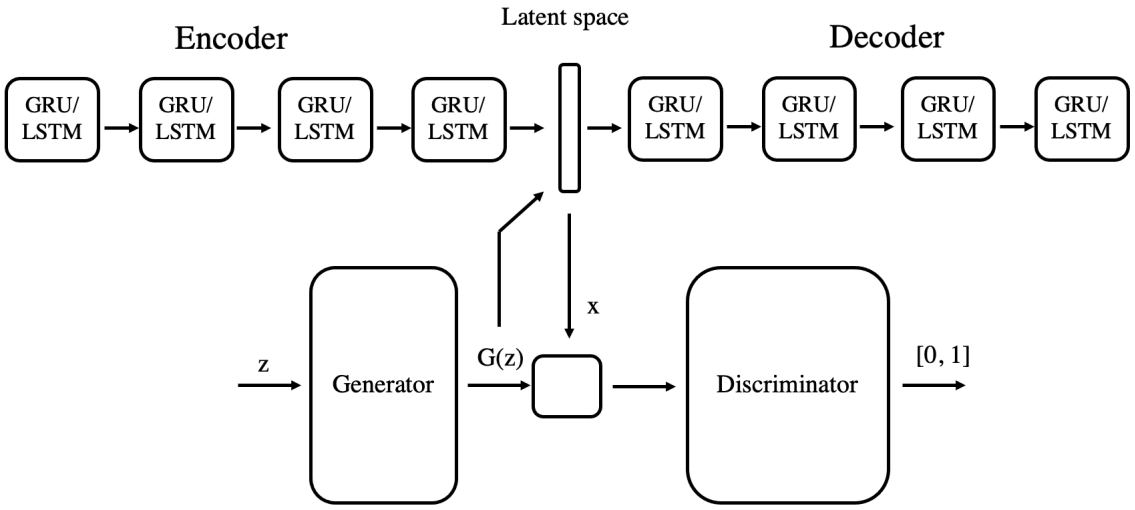


Figure 2.3: A schematic view of the auto-encoder GAN architecture. The encoder breaks down the properties of a time series into a latent space representation. The purpose of the GAN is to a generate an output with the properties of the latent space representation that the decoder can reconstruct into a new, synthetic time series.

2.3.3 Recurrent GANs

Recurrent- or Recurrent Conditional GANs has been one of the main methodologies for time series generation. It was first introduced by Esteban, Hyland, and Rätsch [6] in 2017 with the implementation area of generating medical time series data for

doctor training simulations. The architecture is similar to that of a regular GAN where the generator competes with the discriminator. The two networks however are substituted with RNNs. [6] also introduces a conditional element in which the time series can be conditioned on certain features or metadata aspects. The original RCGAN model is trained to minimize cross-entropy loss between its own prediction and the label of the time series of whether it is synthetic or real. The generator on the other hand is trained to minimize the negative cross-entropy loss between the discriminator predictions of the synthetic samples and the True labels, i.e. the generator is trained to trick the discriminator.
From this original work, many studies have attempted to reproduce and in many ways improve the performance of the Recurrent GAN and RCGAN. Among those are who utilize a Recurrent GAN or RCGAN methodology are [8], [7], [9] where their contributions are either data from different application areas, different training conditions, altered architecture structures or implementation of metadata handling (mainly [9]). Of these methods, [9] really stand out due to the fact that the authors present results that show evidence of outperforming many previous methods including RGAN, RCGAN [6] and Time-GAN.
When training a RCGAN, a solution to the variable length issues is to condition on lengths and by doing that, structuring the batches of training based on lengths of scenarios [8], [5].

2.3.4 DoppelGANger

One adaptation to the original RCGAN is a network called *DoppelGANger* [9]. This network uses the basic structure with a LSTM generator and a *Multi-layer Perceptron* as discriminator. It does though have some important differences that is presented below.

2.3.4.1 Batch Generation

To increase the ability of the network to learn long term dependencies the authors of DG suggests to generate a batch of time steps at each RNN pass-through [9]. The way that this is improving long term dependencies is by reducing the amount of passes through the RNN that is necessary to generate the entire time series. The idea is that with a lower amount of passes through the RNN the internal state will have more information from steps further away [9]. Generating a batch of samples at each RNN pass comes with a difficulty that is increasing rapidly with the length of this batch. This induces a trade off and the length of the batch generated at each RNN pass has to be tuned [9]. An example of how DG implemented the batch generation with S=2 i.e. two time steps are outputted from each cell is shown in Figure 2.4 [9].

2.3.4.2 Auxiliary Discriminator

A feature that is desirable in a method for time series generation is the ability to also generate matching metadata, this has been investigated for a few different methods [5], [9]. DG has its own solution to this issue which is adding a second discriminator

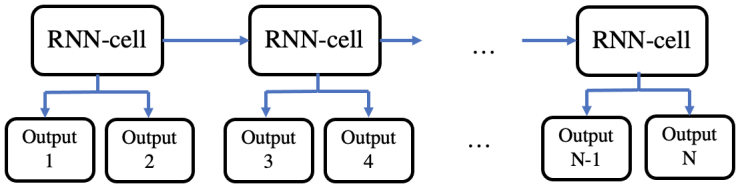


Figure 2.4: The batch generation principle with S=2, Each RNN cell outputs two timesteps ahead for a given time series length N.

that discriminates how well the generated metadata represents the real metadata [9]. The main discriminator then discriminates on generated features and metadata. This also means that the main discriminator evaluates whether the the metadata correlates well with the generated features.

2.3.4.3 Auto-normalization Heuristic

DG uses a normalization method that is based on normalizing each time series on its own [9]. The idea behind this is that it reduces issues that can come from having time series in a wide range. The authors of DG suggests that it alleviates mode collapse which is a common issue when training GANs [9]. The issue that occurs when normalizing each time series by itself is that the information lost when normalizing needs to be kept somewhere. DG solves this issue by calculating metadata values using maximum and minimum of each time series according to equation (2.14).

$$meta_i = \frac{max_i \pm min_i}{2}, \tag{2.14}$$

By then training on this metadata and generating it together with the time series it can be used to re-normalize generated time series [9].

2.3.4.4 Generation Flag

DG approach to making sure that the distribution of the lengths of the synthetic scenarios matches the lengths of the real scenarios and that the scenarios are ended at a suitable time step [9]. The solution that is presented is a generation flag that is fed to the model along with the real data with the purpose of signalling when the the scenario is ending and the end padding starts of each sequence.

					End-padding	
Time step	1	2	3	4	5	6
Scenario data	0.2	0.4	0.3	0.1	0	0
	13	14	13	12	0	0
Generational flag	1	1	1	0	0	0
	0	0	0	1	0	0

Figure 2.5: Illustration of how the generation flag is embedded with each scenario data

For each time step the generator outputs a time step for a sequence, it also outputs a probability $[p_1, p_2] \in [0, 1]$ of how likely it is that the scenario is over where p_1 represents the probability that the scenario is should continue and p_2 represents the probability that the scenario should end. If $p_1 > p_2$ the scenario is ended and padded with zeros to a fixed length. The principle of how the generation flag is fed to the network along with the features is shown in Figure 2.5. Intuitively, this method should solve the issue of ending scenarios at the right time step.

3
Methods

In this section the implementation of the framework for generating driving scenarios is presented. The data used and how it is processed is also presented. Finally the methods used for evaluating the result is described.

3.1 Data and Processing

The data used for training the GAN models was filtered scenarios from a dataset of lane changes supplied by Zenseact. The two parameters of study were lateral and longitudinal velocity for the ego vehicle. These two parameters were derived from measurements of the ego vehicles position. The lateral and longitudinal velocities were derived as the gradients of the position with time frequency of 0.111s between each time step.

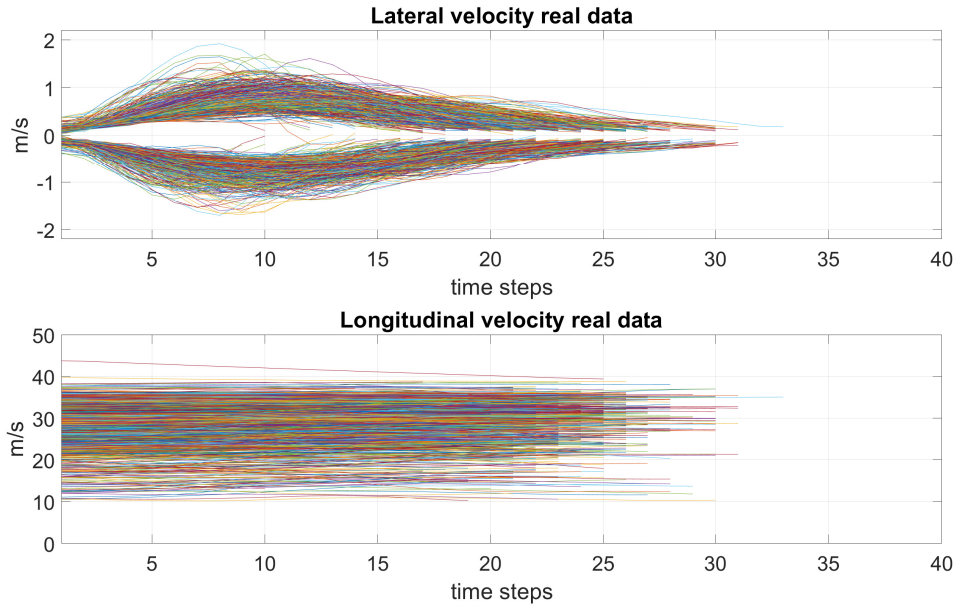


Figure 3.1: 1000 real scenarios that make the validation dataset. Top plot illustrates lateral velocity and bottom illustrates longitudinal velocity

From this, a set of rules was established to ensure that only lane changes that matches our definition are extracted from the data. Based on these rules, the definition of a scenario was set as a scenario that surpasses 10 m/s in longitudinal velocity.

At the same time the scenarios must have had a starting position less than 1.5 m from the initial lane center, it had to move 1.5-6 m laterally and could not end up closer than 1 m from the initial lane center. The scenario must also have had a starting lateral velocity lower than 0.4 m/s and end up with a lower lateral velocity than 0.25 m/s. A scenario can also have at most two local maximas or minimas in lateral velocity. I.e. if a scenario has more than two peaks before returning to close to zero, it is discarded. The dataset was ultimately consisting of 7301 scenarios. Of those, 1000 scenarios were separated in to a validation set which are shown as a group in Figure 3.1. These scenarios were not exposed to the model during training, but instead used to compare the generated scenarios. The reason for this is to compare the results to data that is not biased. The 1000 scenarios in the validation data is randomly extracted from the 7301 scenarios. This leaves 6301 scenarios that are used for training.

3.1.1 Normalization

Since the nature of the data in lateral and longitudinal velocity differs, two different approaches on how to handle the features were used. The auto-normalization heuristic suggested by DG was used for longitudinal velocity. This method suits data with a large range well and the values in longitudinal velocity is over a quite large range. Utilizing the DG auto-normalization heuristic, two normalization constants is generated for each time series. The network needs to be fed these two normalization constants as metadata, derived according to equation (2.14). Since the lateral velocity is close to normalized from the start, with almost zero mean and a suitable range, no normalization was used for lateral velocity.

3.1.2 Attribute Standardization

An issue that arises when training multiple dimensions of data is the difference in amplitude of the data. The attribute discriminator is trained to learn the starting lateral position, starting lateral velocity as well as the fake metadata of the longitudinal velocity. metadata values differs in order of magnitude, which is probable to have a negative effect on training. In order to solve this issue, a regular standardization was done for each attribute (a). Standardization is the subtraction of the mean and divided by the standard deviation of each attribute category seen in equation (3.1),

$$a^i = \frac{a^i - \mu^i}{\sigma^i} \tag{3.1}$$

where μ and σ is the mean and standard deviation of each attribute. This causes all attributes that are fed into the network to have a mean of zero and a standard deviation of one. The attributes will therefore be weighted more equally when the network is trained.

3.2 Network

The network structure was heavily influenced by the DG structure [9]. The Generator is a RNN constructed of LSTMs with three layers and 100 units. At the end of the generator, for longitudinal there is a tanh layer. The initialization of the first time step is set randomly and continuously updated during the training. The generator is trained to capture the features of the lateral velocity and a normalized version of the longitudinal velocity. Complementing the generator is a MLP attribute generator with four layers and 100 units. The attribute generator has two main purposes. Firstly, the attribute generator generates metadata. For this application the metadata is the initial position in the lateral direction. This is information that is not included in the generator input. Secondly, generating normalization attributes, later used to re-normalize the scenarios. This attribute generator architecture is similar to the one used in DG [9], however, it differs in the fact that DG uses a sigmoid or tanh layer as a final layer depending on if the input data is normalized to between minus one to one, or zero to one. These layers are removed in the model presented.

The network is trained using Wasserstein loss, therefore, the discriminators are exchanged with critics. The critic is a seven layer MLP with 100 units. This discriminates based on output from both the generator and the attribute generator I.e. it is fed real data, synthetic generated data, real attributes and synthetic generated attributes. To increase fidelity of the critic performance, an attribute critic is used to only discriminate the synthetic generated attributes and real attributes. This attribute critic is a MLP with 7 layers and 100 units. The total loss is then a sum of the loss from the two discriminators. This loss can be weighted using an additional tuneable hyper parameter determining the importance of the attribute critic.

3.2.1 Capturing Long-term Dependencies

A few different tools was used to try and capture the temporal aspects of the data as well as possible. The main challenge being to capture long-term dependencies. The most important addition was the network structure, using LSTM-cells like DG and RCGAN. Another thing was the downsampling of the time series. The original data had a sampling rate of 0.111s between time steps. From this the data is downsampled to a third resulting in a sampling rate of 0.333s between time steps. With fewer steps in the time series the networks task of capturing the entire time series becomes simpler. In DG it is suggested to generate multiple samples in each RNN pass which is a method also used. This however has a trade-off of each RNN pass becoming harder to generate. Combining the downsampling with the batch generation introduces the opportunity to substantially decrease the number of passes through the LSTMs needed for each scenario which ultimately results in an increased ability to handle long-term dependencies. After testing various values of S, it was found that S=2, was a value that was well suited for this type of data.

3.2.2 Generating Scenarios of Various Lengths

Of all the methods presented in section 2.3 regarding training a network with data of various lengths. The generation flag by DG stands out by solving the issue with generating sequences of various lengths and also shows empirical evidence of managing to capture the distribution of the lengths of the scenarios. The generation flag is implemented as Figure 2.5 and trained as an additional feature to lateral and longitudinal velocity in the generator. Since this feature had a discrete time-domain, a softmax layer is used as a final layer of the generator for this feature.

3.3 Evaluation

There is no standard method for evaluating the performance of GANs [25]. One of the most common methods used is to just visually inspect the generated data. Visualizing the data could be done in a large variety of ways. It is also of interest to be able to quantify the results. This has been done in many different ways and there is no perfect solution that fits every data type. What can be said though is that it is of great interest to be able to look at distributions of the data and not single samples. This since the goal of GANs is to capture the distribution of the data and not recreating single samples. One metric that measures the similarity between distributions is the *Jensen-Shannon distance* (JSD) which is the square root of the JS divergence from equation (2.3)[26]. It is also interesting to look at different properties of the data. First of all it is important to see that both the generated data and metadata has properties that matches the real data. To be able to do this properly the position is also derived using the velocities and the starting position from the metadata. The length of the scenarios is also studied to see that the network properly handles variable length.

3.3.1 Removing Temporal Aspect

The main issue with JSD for evaluating time series is that it can not measure the temporal aspect directly. To be able to use the JSD on the data the temporal aspect of the data was ignored, storing all data points for all time series in a large vector. It is then possible to calculate the JSD on this data as well as visualizing the data in histograms for each feature. Comparing total distributions is a way to study if the generated scenarios covers the entirety of the data, the tails of the data and if the models is fixed on generating a certain type of data more or less frequent.

3.3.2 Time Dependencies

Since lateral and longitudinal velocity both are time dependent component it is of high relevance that the generated samples manages to capture the entirety of these components. It is also important that the change from one time step to the next is accurately captured as well as longer time dependencies such as between multiple time steps. Studies within similar field have labeled this evaluation metric in multiple ways such as auto-correlation [9], first difference [7] or traffic rationality

analysis [15]. In this study, the difference is the same as acceleration and we wish to study the acceleration over different time-intervals. This is an important metric to cover since time dependencies in both long- and short-term is one of the main challenges of this study. In this study, we will attempt to evaluate the distribution of first, third and fifth difference. The first difference is the difference in velocity between time step n and n+1 where n is a time step in a time series of length N, $n \in [1,\ N-1]$. The third and fifth difference is derived in the same manner, with the exception of comparing n with n+3 and n with n+5 respectively.

3.3.3 Lane Change and Steering Behaviour

In order to ensure that the underlying properties such positional change and heading are kept, it is important to further study and ensure that the generated scenarios represented proper lane changes. One easy way of examining this was to derive the positional change that each generated scenario made and compare that with the real data. The position in each time step in lateral and longitudinal direction was derived according to equation (3.2). From section 3.1, a criteria for a lane change is a scenario that travels 1.5-6 m laterally and moves at least 1 m from the initial lane centre. A generated scenarios should therefore uphold this criteria.

$$p_i = p_{i-1} + v_i * t \tag{3.2}$$

Another important driving behaviour to study is the yaw of the vehicle also called heading. Heading can be examined by visually inspecting velocities, but a much better comparison can be done if the heading is calculated. The heading (α) in each time step was instead derived as seen in equation (3.3).

$$\alpha_i = atan2(v_i^{lat},\ v_i^{lon}) \tag{3.3}$$

The first difference, which represents yaw rate, is also derived to see that the temporal aspects are kept in the steering behavior of the lane change.

3.3.4 Nearest Neighbour

Some studies within the field of time series generation uses a comparison between individual samples of the generated data and real samples. For example Demetriou, Alfsvåg, Rahrovani, *et al.* uses *dynamic time warping* and *Hungarian algorithm* [8] to illustrate the performance of their model. These kinds of comparisons however could be deceiving. If the error between a generated sample and a real sample is very small it could be signs that the model is overfitted to only output copies of the real data. However, comparing individual samples could be relevant, in order to show that the data is not overfitted and that the generated samples actually manages to capture the data.
For this reason we use a comparison metric of finding nearest neighbours. First, the generated data is studied, for all samples of the generated data the *root mean-squared error* (RMSE) compared to each scenario in the training data is derived using equation (3.4). The generated scenarios that has the three most similar real scenarios determined by RMSE, are visually compared to these generated scenarios.

This comparison is done to ensure that the samples with the smallest error does not coincide with their three nearest neighbours in the real dataset.

$$RMSE = \sqrt{mean(\hat{X} - X)^2} \tag{3.4}$$

In order to study whether or not the GAN model manages to capture all types of scenarios in the real data, the three nearest neighbours to all real samples in the generated data are found. In order to draw conclusions on whether or not some category of samples are generated, the samples with the highest RMSE are studied visually. This method will further on be called farthest neighbour.

3.3.5 Benchmarking

In order to put the outcome of this project in to context, a set of benchmarking model was used. Two alternative generative methods were trained with the same training data as the proposed model of this study. These two models were TimeGAN [5] and the RVAE presented in the RCGAN paper [6]. The reason that these two models were utilized for benchmarking is that TimeGAN previously have shown great performance for generating timeseries, the AE-GAN structure was also used for generating cut-ins in [8]. The RVAE model on the other hand represents a simpler network structure than the presented framework of this project and it was therefore of high relevance to examine if this simpler model can achieve satisfactory results on the data. Both models was used as presented in their respective code repository with a few alterations:

- The data was scaled using standardization. This was done by removing the mean of the entire dataset and dividing with the standard deviation of the entire dataset. The generated data was then re-standardized.
- A generation flag similar to the DG method [9] was added as a third feature to the training were ones represented that the scenarios were active and zeros represented that the scenario had ended.
- For the RVAE model, a sigmoid layer were used for the generational flag feature. When it outputted a probability of less than 0.9 the scenarios were considered ended.

4

Results and Discussion

The results from the proposed model framework is presented in this section, along with the results of the benchmarking models. In order to evaluate the generative models, a total of 1000 generated scenarios are compared with the 1000 scenarios from the validation dataset. These generated scenarios are then compared to the real data and evaluated to see how well the model handles the challenges from section 2.1.1. The results and the implications of them are also discussed in this section.

4.1 Overview of Results

The generated lateral and longitudinal velocity is illustrated in Figure 4.1. The grouped generated scenarios visually matches the real data quite well. Compared to the validation data in Figure 3.1 the generated scenarios have a few more scenarios with lower amplitudes of lateral velocity, and the generated ones does not fully capture the outliers that have an amplitude above 1.5 m/s in lateral velocity. This could be due to the fact that scenarios with high amplitudes, i.e. aggressive steering behavior are not well represented in the training data.

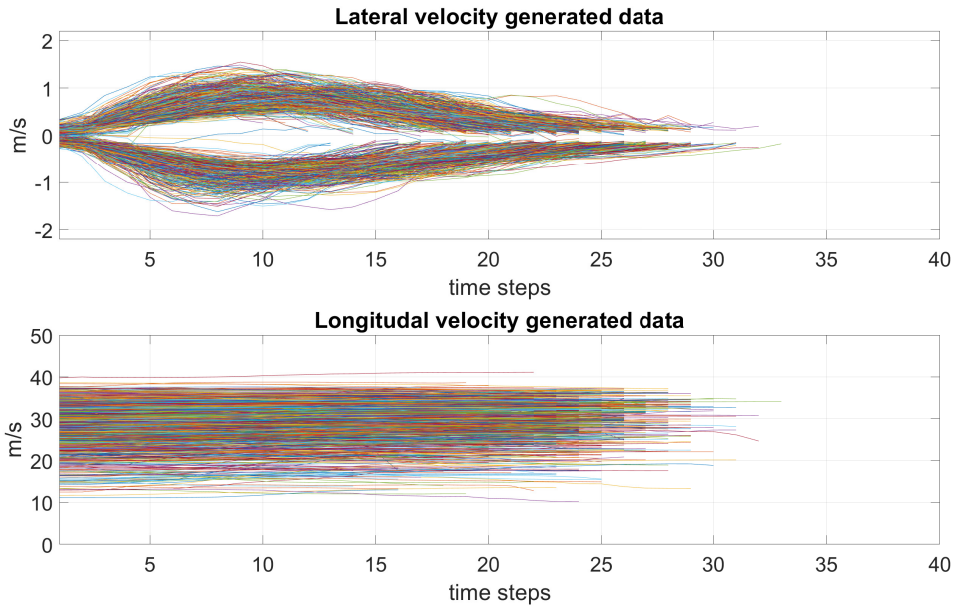
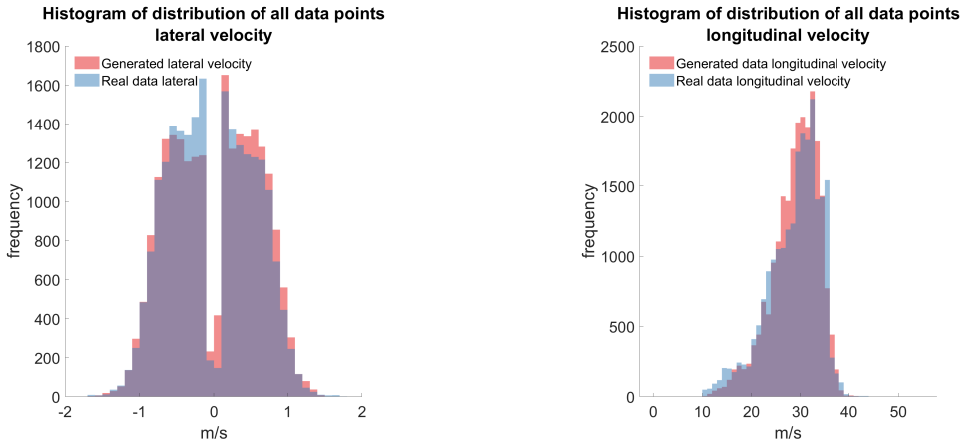


Figure 4.1: 1000 generated scenarios of lane changes. Top plot illustrates lateral velocity and bottom illustrates longitudinal velocity

To ensure that the total distribution is captured, disregarding the time dependencies, the distribution of all generated time steps in lateral and longitudinal velocity is studied. The overall distribution of the data also match quite well. With the exception that the model generate some scenarios with lateral velocities very close to zero, which is shown in Figure 4.2. In Figure 4.2a, it can be seen that the model generates more scenarios with positive maneuver while the validation data generates more scenarios with a negative maneuver From Figure 4.1. Some generated scenarios have a quite low amplitude and is crossing zero more frequently than in the validation data in Figure 3.1. This could cause the behaviour we see in Figure 4.2a where the generated model has more data point in the bin around zero.



(a) Distribution of lateral velocity

(b) Distribution of longitudinal velocity

Figure 4.2: Distribution of the values of velocities in all time steps in a 1000 generated scenarios compared with the validation scenarios.

To see how well the network manages to capture the length of the scenarios it is easiest to just look at a histogram of the scenario lengths. This does though not tell how reasonable the ending of the scenarios are. This is instead necessary to evaluate visually looking at Figure 4.1, where it can be seen that the scenarios ends at reasonable points. The histogram of the generated lengths of the generated data compared to that of the validation data is presented below.

Looking at the distributions in Figure 4.3 it is possible to see that the distributions matches well. Though it is possible to see a slight bias towards generating longer scenarios.

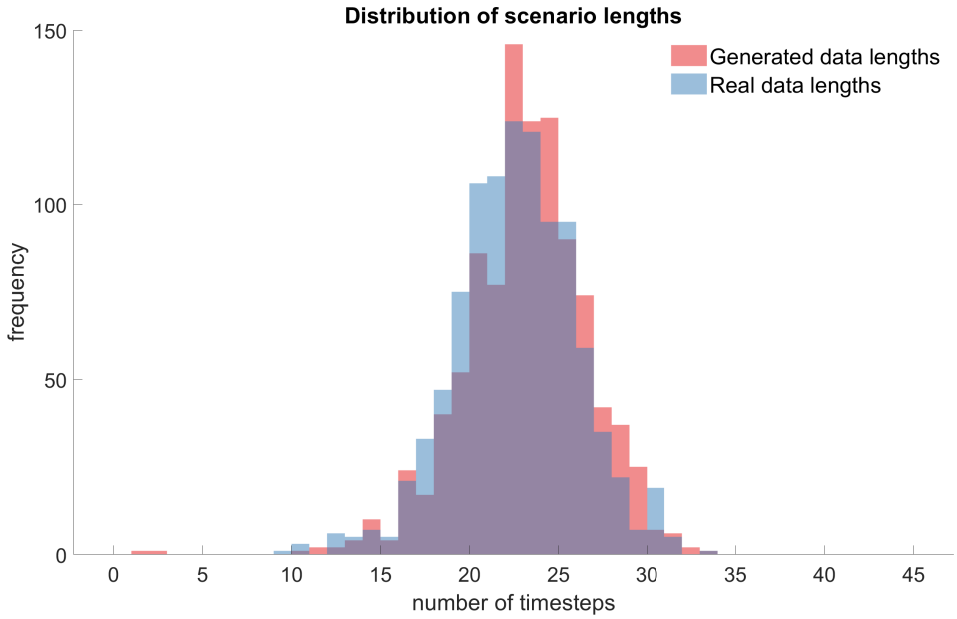


Figure 4.3: Distributions of scenario lengths for generated and validation data

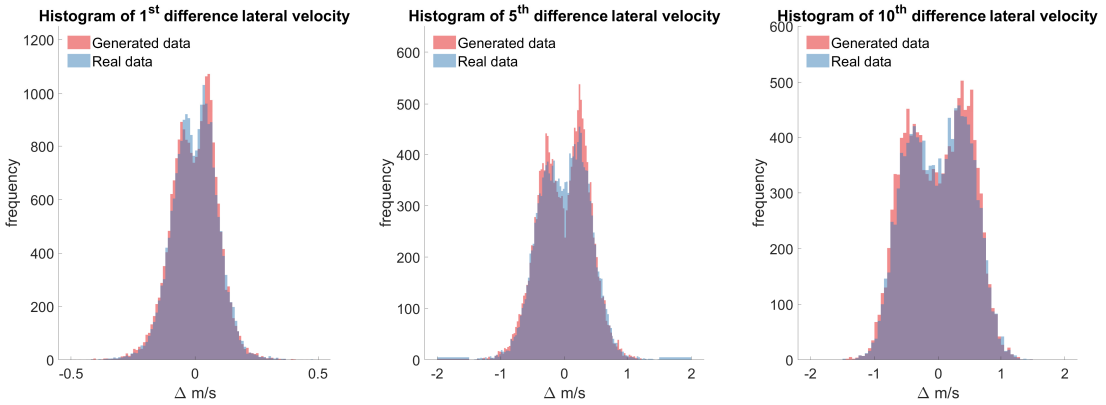
Table 4.1: JSD between total distributions of all time steps in 1000 generated scenarios compared to the validation data

Data	JSD lat. vel	JSD lon. vel.	JSD scenario lengths
Training	0.0359	0.0776	0.0990
Model	0.0664	0.1090	0.1224
RVAE	0.1985	0.1452	N/A
TimeGAN	0.2074	0.2426	N/A

The JSD distance between the generated distributions in Figure 4.2 is computed and compared to the benchmarking methods in Table 4.1. The model outperforms both the benchmarking methods when comparing the distributions in the JSD metric. Since the benchmarking methods of RVAE and TimeGAN does not handle a variation of scenario length, a custom generation flag is implemented, therefore comparing the scenario lengths is not of high relevance. It can also be seen that there is a non-negligible distance between the training and the validation data. This is relevant to take into account when evaluating the performance. For example, the model is quite far away from the validation set on its distribution of scenario lengths compared to the other metrics. This is most likely due to the fact that this part of the training data does not perfectly represent the validation data. This statement is also supported by the JSD for scenario length presented in Table 4.1.

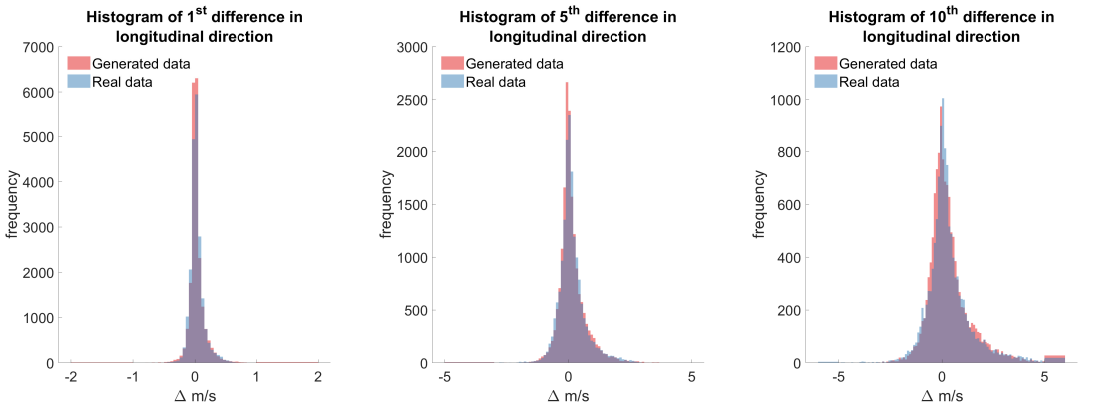
4.2 Temporal Aspects

To be able to compare how well the model and the benchmarking methods captures long-term dependencies, the difference between time steps within each sequence is evaluated. More specifically, the distribution of the difference between each time step, each fifth time step and each tenth time step is studied. Each distribution is compared using the JSD to the distribution of the validation data. Figure 4.4 and 4.5 illustrates how the distribution of all differences is captured by the model compared to the validation set for lateral and longitudinal velocity respectively. It can be seen that it captures the overall distribution and the peaks of the distributions to a large extent.



(a) 1^{st} difference lateral velocity **(b)** 5^{th} difference lateral velocity **(c)** 10^{th} difference lateral velocity

Figure 4.4: Distribution of the error between each time step, each fifth time step, and each tenth time step in lateral velocity,



(a) 1^{st} difference longitudinal velocity **(b)** 5^{th} difference longitudinal velocity **(c)** 10^{th} difference longitudinal velocity

Figure 4.5: Distribution of the error between each time step, each fifth time step and each tenth time step in longitudinal velocity

The samples in the tail of the distribution shown in Figure 4.4b says that there are

a few, more aggressively turning samples in the validation data. Previously it was shown that the highest amplitude scenarios in the validation data is not captured by the network. These high amplitude scenarios is most likely the same scenarios that is the most aggressive. In a larger perspective it is a bit worrying that the network does not manage to capture the most aggressive scenarios, since it might very well be these scenarios that causes safety issues.

Table 4.2: JSD between distributions of time differences in lateral velocity

Data	**JSD** 1^{st} **difference**	**JSD** 5^{th} **difference**	**JSD** 10^{th} **difference**
Training	0.0409	0.0350	0.0372
Model	0.0458	0.0482	0.0430
RVAE	0.0937	0.0898	0.0956
TimeGAN	0.1638	0.1274	0.1051

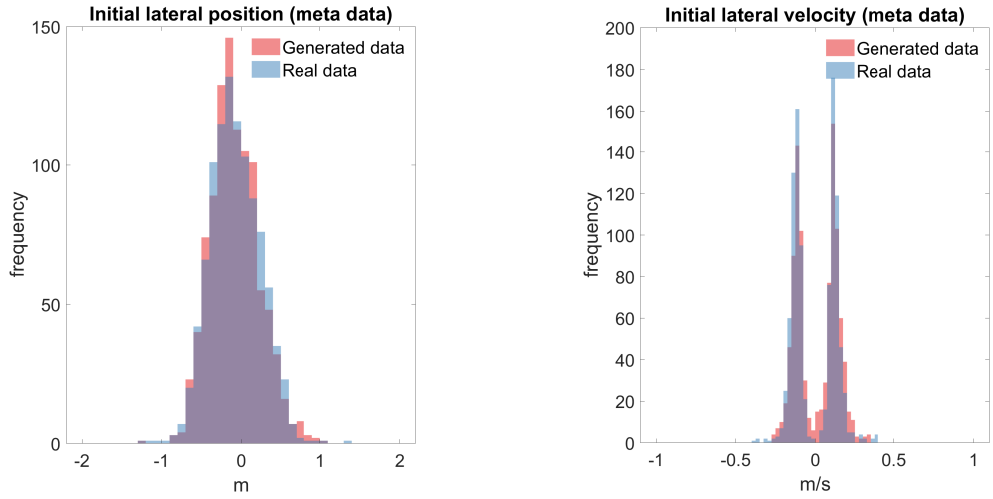
The performance on long-term dependencies by the model is compared to the benchmarking methods in Tables 4.2 and 4.3. In this metric, the model outperforms the benchmarking methods substantially. The performance in each difference is more similar to the training data than the benchmarking methods. This indicates that the models ability to capture temporal aspects is good.

Table 4.3: JSD between distributions of time differences in longitudinal velocity

Data	**JSD** 1^{st} **difference**	**JSD** 5^{th} **difference**	**JSD** 10^{th} **difference**
Training	0.0580	0.0666	0.0728
Model	0.1072	0.0723	0.0854
RVAE	0.1686	0.2329	0.2600
TimeGAN	0.3341	0.2926	0.2358

4.3 Metadata

Determining how well the model handles metadata was done by first comparing the distributions of generated metadata to that of the validation set. Histograms of generated and validation metadata along with the JSD is presented in the following Figure.



(a) Distribution of initial lateral position. JSD=0.0724

(b) Distribution of initial lateral velocity. JSD=0.1524

Figure 4.6: Distribution of the generated metadata compared.

1It is not enough to only show that the metadata on its own matches the validation data, the features generated by conditioning on the metadata must correspond to the metadata. A visual comparison for the initial lateral position can be done by looking at the positional trajectories. These can be seen in Figure 4.7. To visually see how well the initial lateral velocity performs it is easier to look at Figure 4.1. For both generated metadata it can be seen that it does not only fit the distribution of the validation data, but does also correspond nicely to the features. Though there is a few scenarios in the middle of the lateral velocity plot in Figure 4.1, these scenarios are not at all represented in the validation data. These scenarios can also be seen in the middle of the positional plot in Figure 4.7. It is not possible to compare the metadata generation with that of the benchmark models since they do not have any metadata generation, however for lateral velocity, the initial step is included as a feature and can therefore still be compared with.

Table 4.4: JSD initial lateral position and velocity of training data, generated data from model and benchmarking methods with regards to validation data

Data	JSD initial lateral position	JSD initial lateral velocity
Training	0.0622	0.0839
Model	0.0724	0.1524
RVAE	N/A	0.2492
TimeGAN	N/A	0.8209

4.4 Positional Changes

To evaluate how well the scenarios are capturing the properties of the real data it is relevant to study the total positional changes of each generated scenario. The behaviour that is sought to be captured is a lane change. In order to determine how well that is captured, the generated positional change is derived and compared to the positional changes of the scenarios in the validation dataset. The positional changes is show in Figure 4.7.

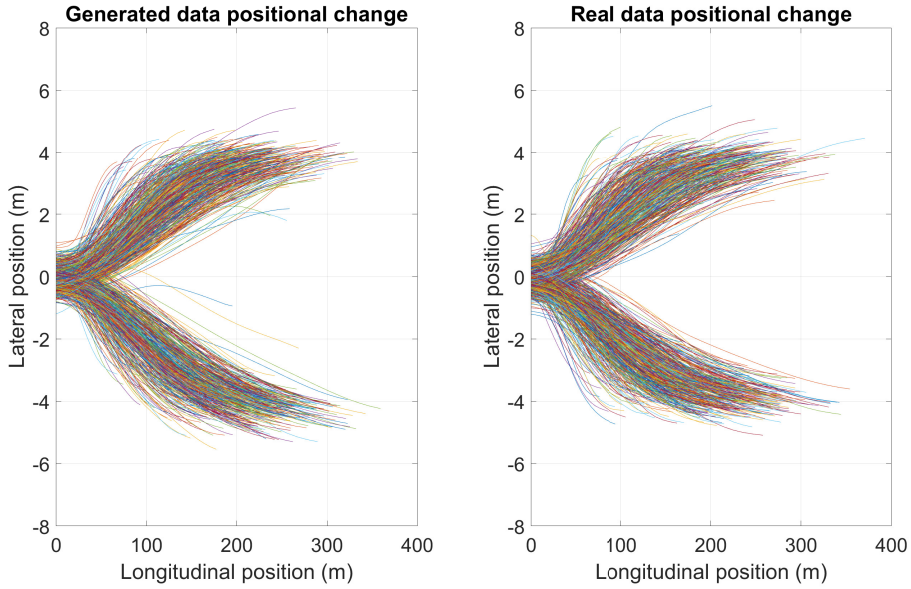
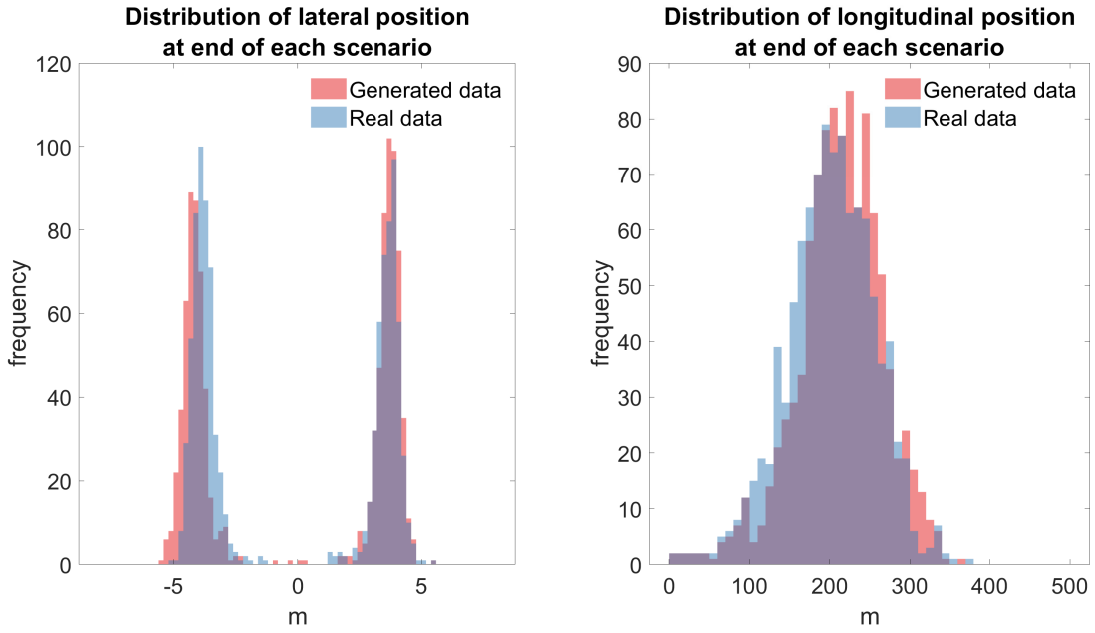


Figure 4.7: 1000 generated scenarios of lane changes of the total positional change. Left plot is the generated scenarios and the right plot is the real scenarios

The grouped generated scenarios matches the validation data well with the exception of a few generated outliers that end up in the region of [-2.5, 2.5]. To be able to determine how well the lane change is executed, it is relevant to study the distribution of were the trajectory end up, both laterally and longitudinally, when the scenarios is over. Figure 4.8 illustrates this distribution. In the lateral direction, in

Figure 4.8a, there is a slight bias for lane changes which move negatively referenced from the lane. In the longitudinal direction, in Figure 4.8b, there is also a slight bias where the generated scenarios is centered at a slightly higher mean end position. This bias, in both lateral and longitudinal end position could be due to the fact that the length of the generated scenarios have a mean that is slightly higher than the real scenarios in the validation dataset. Longer scenarios could have the affect of resulting in the vehicle travelling further.



(a) Distribution of lateral position at scenario ending

(b) Distribution of longitudinal velocity at scenario ending

Figure 4.8: Distributions of the position where scenarios end. To the left is for generated lateral end position compared to the corresponding data in the validation set. To the right is the generated longitudinal end position compared to the corresponding data in the validation set

The JSD of all methods is shown in Table 4.5. In this metric, The model performs similar to the RVAE in end longitudinal position and TimeGAN in the end lateral position. However, when visually studying the positional change (see appendix A), it can be seen that both the benchmarking methods generates scenarios that are smoothed and very similar to each other which is a typical trait of Auto-Encoder models. Visually, the generated scenarios by the model matches the validation scenarios better.

Table 4.5: JSD between of positional distribution of the final time step for the training data and each method with regards to validation data.

Data	JSD lateral end positions	JSD Longitudinal end position
Training	0.0420	0.0679
Model	0.1811	0.1318
RVAE	0.3372	0.1519
TimeGAN	0.1811	0.2016

4.5 Steering Behavior

In order to evaluate the driving behaviour, the steering in each lane change is also examined. Figure 4.9 illustrates the derived heading of the 1000 generated scenarios, along with the derived heading from the validation dataset. Visually they match each other quite well, apart from the fact that the real data has some outliers with aggressive steering. This can also be seen in Figure 4.10a where the real data has some tails that the generated data do not cover.

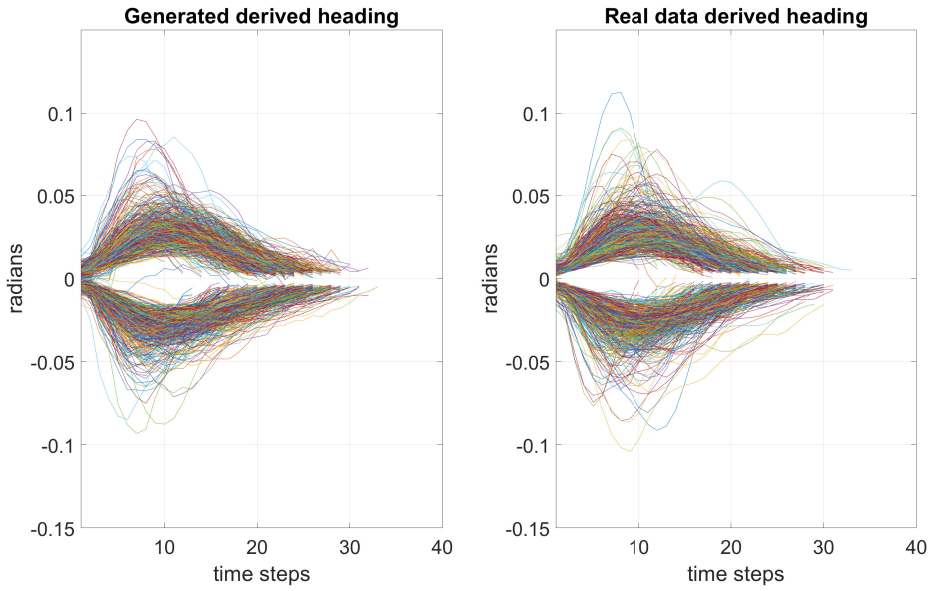


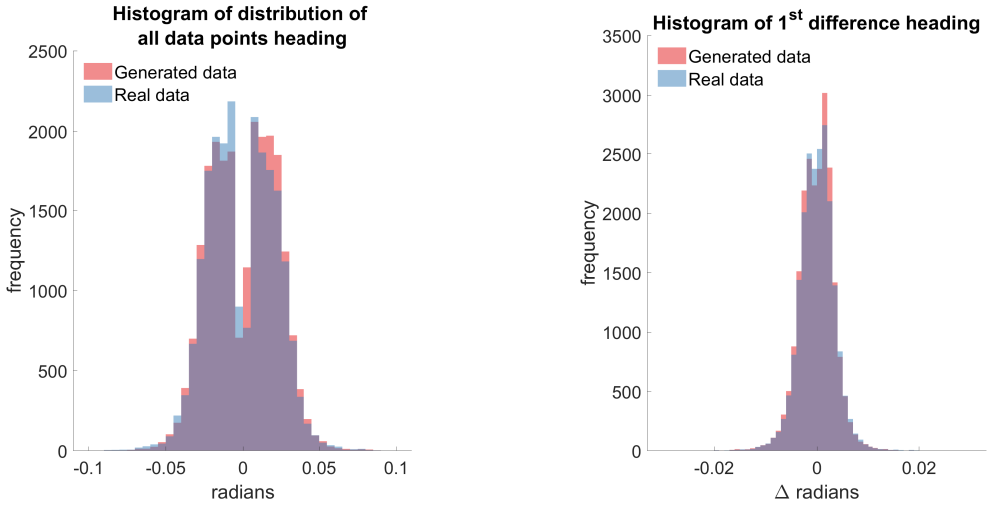
Figure 4.9: Steering behaviour of 1000 grouped scenarios. Left plot is the derived heading from the generated data and the right plot is the derived heading from the real scenarios

In Figure 4.10b, the first difference is also included to show that the temporal aspect between time steps is also well matched compared to the real data. Overall, the model captures the steering behavior of the distribution. Table 4.6 shows the JSD between the training data, the proposed framework and the benchmarking methods.

Table 4.6: JSD of total distribution and 1^{st} difference of heading between generated and validation data

Data	JSD total distribution heading	JSD 1^{st} difference heading
Training	0.0420	0.0501
Model	0.0649	0.0509
RVAE	0.1833	0.0863
TimeGAN	0.2200	0.1618

The model manages to capture the distribution of heading nearly as well as the training data does, which confirms that the model manages to capture the heading very well.



(a) Distribution of all the entire distribution of the heading

(b) Distribution of the 1^{st} difference in heading

Figure 4.10: Total distribution and 1^{st} error of the derived generated heading compared to the derived heading from the validation dataset.

4.6 Evaluation of Overfitting

Up to this point, comparisons have only been made between the generated data and the validation data. As mentioned in section 3.1 the validation data is 1000 scenarios randomly extracted from the total dataset. These scenarios do resemble the training data in many ways. For this reason, it is relevant to compare the generated data with the training set to ensure that the generated scenarios are not only a set of identical reproductions. This is evaluated using the metric nearest neighbour from section 3.3.4. For each generated sample, the three nearest neighbours, i.e. the three with the smallest RMSE from equation (3.4) are studied. Of all these scenarios, the

three that are closest to their nearest neighbours are presented in Figure 4.11. The Figures show that the generated samples is close to its neighbour but no time series are identical since they have some characteristics that sets them apart. This show that the model is not heavily overfitted to the data.

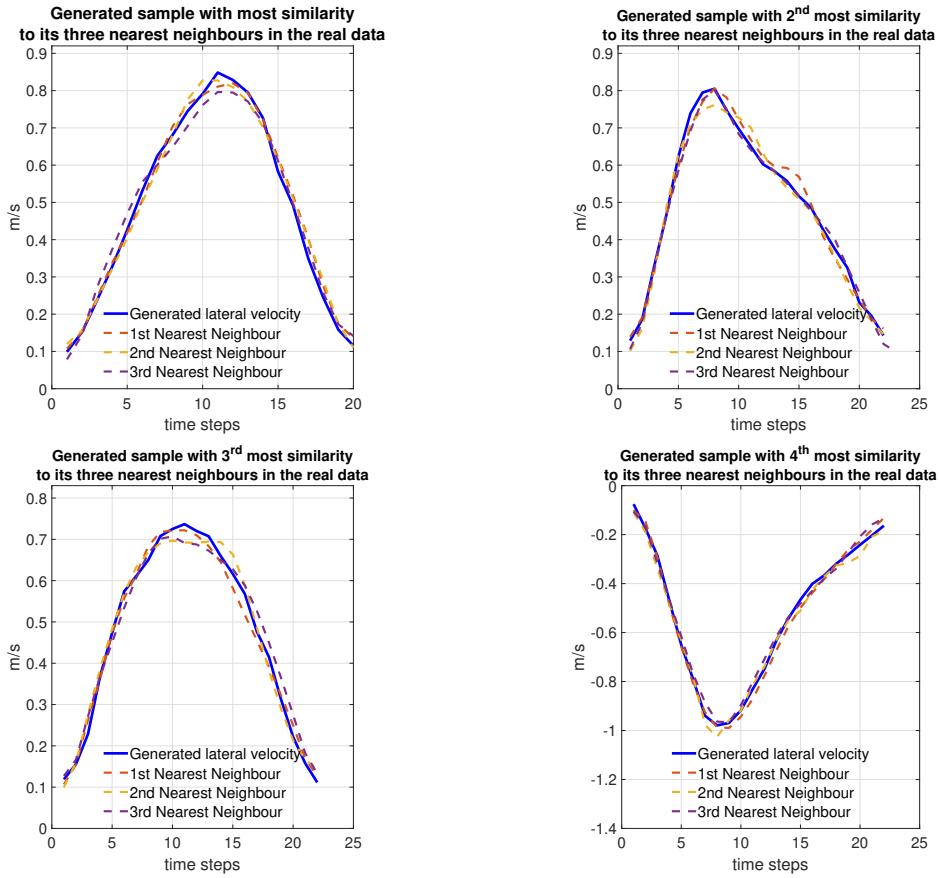


Figure 4.11: Nearest neighbour evaluation of the four generated scenarios with the smallest RMSE error to its three nearest neighbour in the real dataset.

4.7 Capturing the Entirety of the Distribution

When comparing real scenarios with the generated ones, it is hard to pinpoint exactly where the model is insufficient. Some outliers or tails that were visually pinpointed were mentioned in section 4.1. To computationally find the outliers that are not covered by the model, we use farthest neighbour evaluation which was explained in section 3.3.4. In the subfigures of Figure 4.12, a set of scenarios in the training set with the neighbours in the generated data with the highest RMSE are visualized. What they have in common is that the generated scenarios can not capture the real scenarios with amplitudes higher than 1.5-2 m/s. That the network does not capture the most aggressively turning scenarios have previously been discussed and the complications of it.

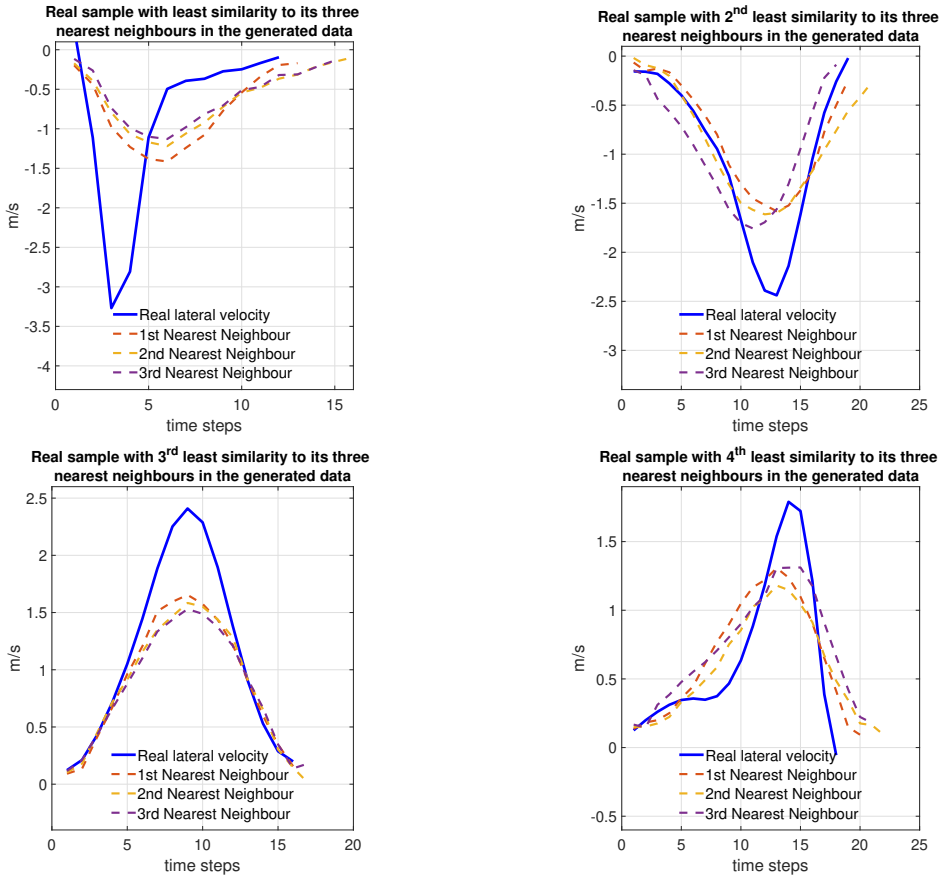


Figure 4.12: Farthest neighbour evaluation of the four real scenarios with the largest RMSE error to its three nearest neighbour in the real dataset

5
Conclusion

The objective of the thesis was to present a framework that could generate realistic scenarios of highway lane changes. The generated lane changes does visually and to a great extent statistically represent that of real lane changes. Results shows that even the generated scenarios with the most similarities does not purely mimic the real scenarios but instead it has similar characteristics which show that the created framework solves the task. The framework however, seems to not be able to capture the scenarios with the most aggressive lateral steering behaviour. The framework outperforms the benchmarking methods in almost every evaluation metric. However, comparing to real data, the model seem to be slightly off in a few metrics.
With regards to challenges faced in previous work, both in time series generation in general and driving scenario generation in particular, the proposed framework handles the challenges successfully. The model framework can handle scenarios of various lengths which is a key factor for driving scenarios generation. The model can also handle metadata aspects with great success. This property was not utilized to its full potential in this work but could potentially be of great use in the field of driving scenario generation. Temporal aspects is also a challenge that the framework captured successfully. The data however, was down sampled and only contained isolated single scenarios. In order to draw full conclusions regarding temporal aspects, exposing the framework for longer scenarios would be suited.

5.1 Future Work

The proposed framework has shown great results with the proposed tuning, however parameter tuning has been done by extensive manual grid search. In order to reach better results, Bayesian Optimization could be utilized to ensure convergence of the model.
Lane changes are only quite simple scenarios, and then it is also possible to have combinations and sequential scenarios. That the metadata generation works well is promising for future works with different types of scenarios. The reason that the metadata is useful here is that it is possible to have scenario types as meta data. For example discrete time aspects such as road conditions, weather, driver or rate of traffic could be used to further categorize the data.
An important challenge that comes with sequential scenarios is the increased length. If the networks performance drops due to the increased difficulty, it could become necessary to utilize more advanced network structures. A structure that has shown a lot of promise in language processing for this is attention based networks [27]. Going

from RNN based GANs to attention based could help a lot if long term dependencies becomes a problem.

References

[1] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" In *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, 2016, pp. 1–16. [Online]. Available: http://www.jstor.org/stable/10.7249/j.ctt1btc0xw.1.

[2] A. Pütz, A. Zlocki, J. Bock, and L. Eckstein, "System validation of highly automated vehicles with a database of relevant traffic scenarios," 2017.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Departement d'informatique et de recherche opérationnelle Université de Montréal*, 2014. [Online]. Available: https://arxiv.org/abs/1406.2661.

[4] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2014. arXiv: 1312.6114 [stat.ML].

[5] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019, pp. 5508–5518. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

[6] C. Esteban, S. L. Hyland, and G. Rätsch, *Real-valued (medical) time series generation with recurrent conditional gans*, 2017. arXiv: 1706.02633 [stat.ML].

[7] E. Listo Zec, H. Arnelid, and N. Mohammadiha, "Recurrent conditional gans-for time series sensor modelling," Jan. 2021.

[8] A. Demetriou, H. Alfsvåg, S. Rahrovani, and M. H. Chehreghani, *A deep learning framework for generation and analysis of driving scenario trajectories*, 2020. arXiv: 2007.14524 [cs.CV].

[9] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar, *Using gans for sharing networked time series data: Challenges, initial promise, and open questions*, 2020. arXiv: 1909.13403 [cs.LG].

[10] F. S. Hoseini, S. Rahrovani, and M. H. Chehreghani, *A generic framework for clustering vehicle motion trajectories*, 2020. arXiv: 2009.12443 [cs.LG].

[11] W. Ding, B. Chen, B. Li, K. J. Eun, and D. Zhao, *Multimodal safety-critical scenarios generation for decision-making algorithms evaluation*, 2020. arXiv: 2009.08311 [cs.LG].

[12] P. Malhotra, V. TV, L. Vig, P. Agarwal, and G. Shroff, *Timenet: Pre-trained deep recurrent neural network for time series classification*, 2017. arXiv: 1706.08838 [cs.LG].

[13] W.-H. Lee, J. Ortiz, B. Ko, and R. Lee, *Time series segmentation through automatic feature learning*, 2018. arXiv: 1801.05394 [cs.LG].

[14] J. Chen, Z. Wu, and J. Zhang, "Driver identification based on hidden feature extraction by using adaptive nonnegativity-constrained autoencoder," *Applied Soft Computing*, vol. 74, pp. 1–9, 2019, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2018.09.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494618305477.

[15] W. Ding, W. Wang, and D. Zhao, "A multi-vehicle trajectories generator to simulate vehicle-to-vehicle encountering scenarios," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 4255–4261. DOI: 10.1109/ICRA.2019.8793776.

[16] A. Kurakin, I. Goodfellow, and S. Bengio, *Adversarial machine learning at scale*, 2017. arXiv: 1611.01236 [cs.CV].

[17] M. Menéndez, J. Pardo, L. Pardo, and M. Pardo, "The jensen-shannon divergence," *Journal of the Franklin Institute*, vol. 334, no. 2, pp. 307–318, 1997, ISSN: 0016-0032. DOI: https://doi.org/10.1016/S0016-0032(96)00063-4. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0016003296000634.

[18] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, *Unrolled generative adversarial networks*, 2017. arXiv: 1611.02163 [cs.LG].

[19] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, 2017. arXiv: 1701.07875 [stat.ML].

[20] B. Zhu, J. Jiao, and D. Tse, "Deconstructing generative adversarial networks," *IEEE Transactions on Information Theory*, vol. 66, no. 11, pp. 7155–7179, 2020. DOI: 10.1109/TIT.2020.2983698.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[23] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: 1406.1078 [cs.CL].

[24] D. Donahue and A. Rumshisky, *Adversarial text generation without reinforcement learning*, 2019. arXiv: 1810.06640 [cs.CL].

[25] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, *Improved techniques for training gans*, 2016. arXiv: 1606.03498 [cs.LG].

[26] M. Menéndez, J. Pardo, L. Pardo, and M. Pardo, "The jensen-shannon divergence," *Journal of the Franklin Institute*, vol. 334, no. 2, pp. 307–318, 1997, ISSN: 0016-0032. DOI: https://doi.org/10.1016/S0016-0032(96)00063-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0016003296000634.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2017. arXiv: 1706.03762 [cs.CL].

A

Appendix 1

A.1 TimeGAN

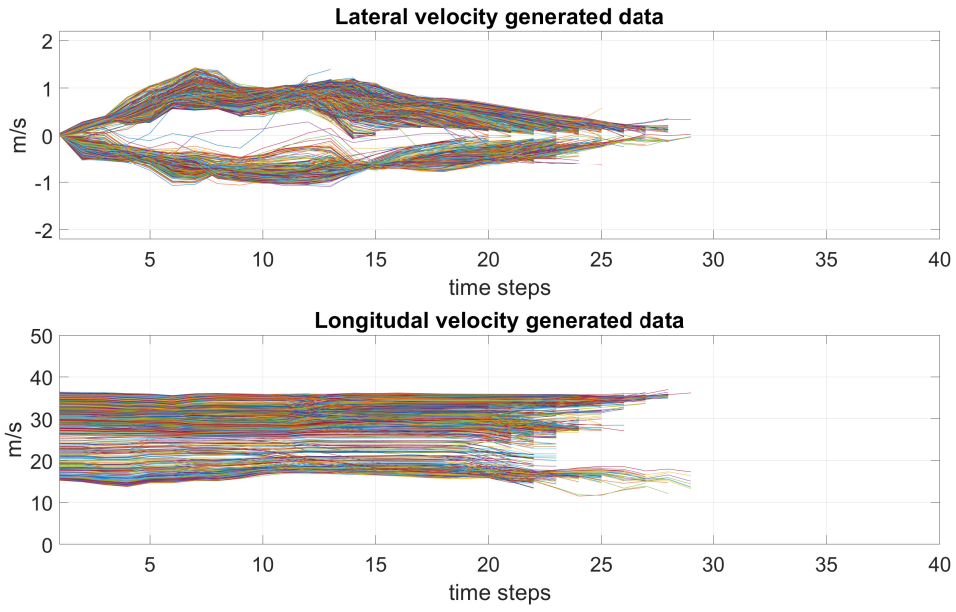
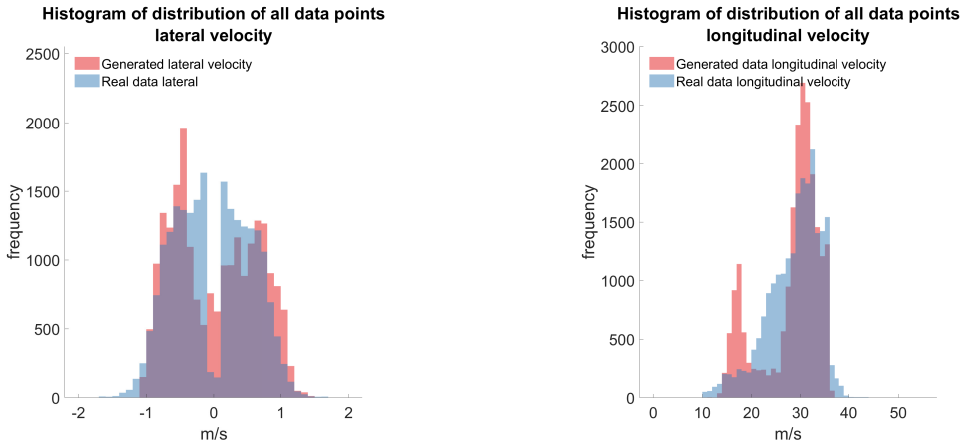
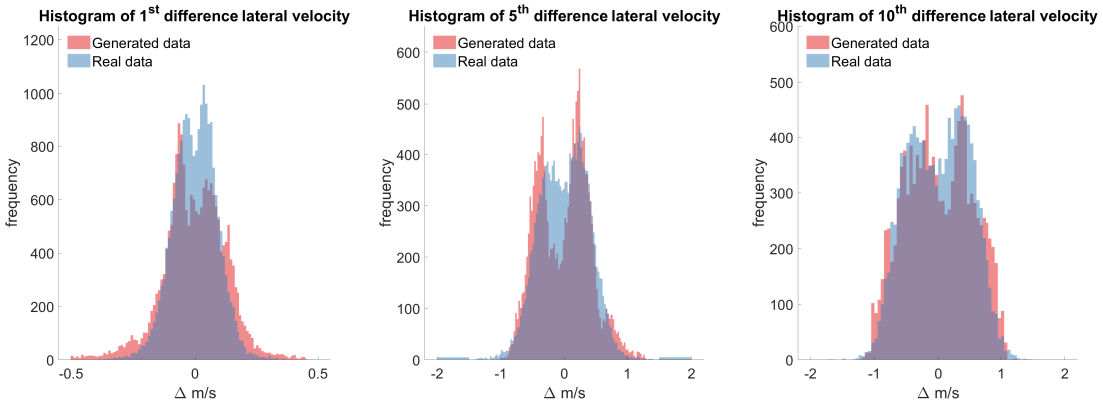


Figure A.1: 1000 generated scenarios of lane changes. Top plot illustrates lateral velocity and bottom illustrates longitudinal velocity



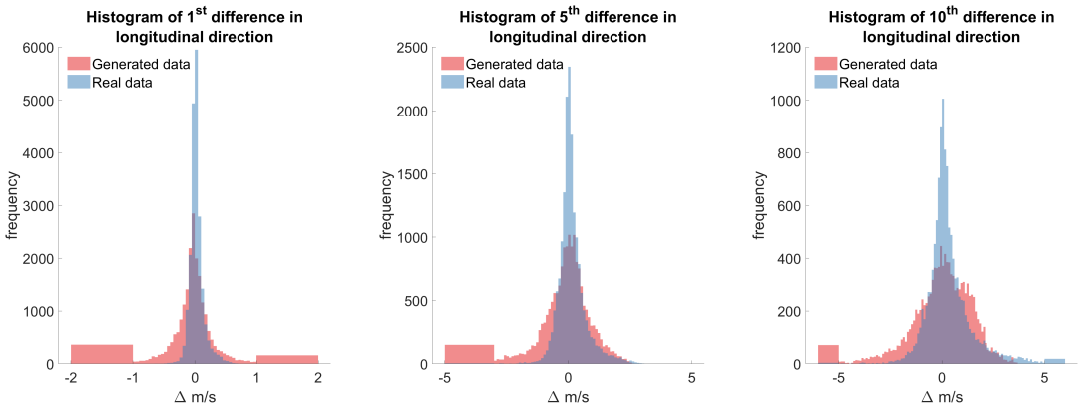
(a) Distribution of lateral velocity **(b)** Distribution of longitudinal velocity

Figure A.2: Distribution of the values of velocities in all time steps in a 1000 generated scenarios compared with the validation scenarios.



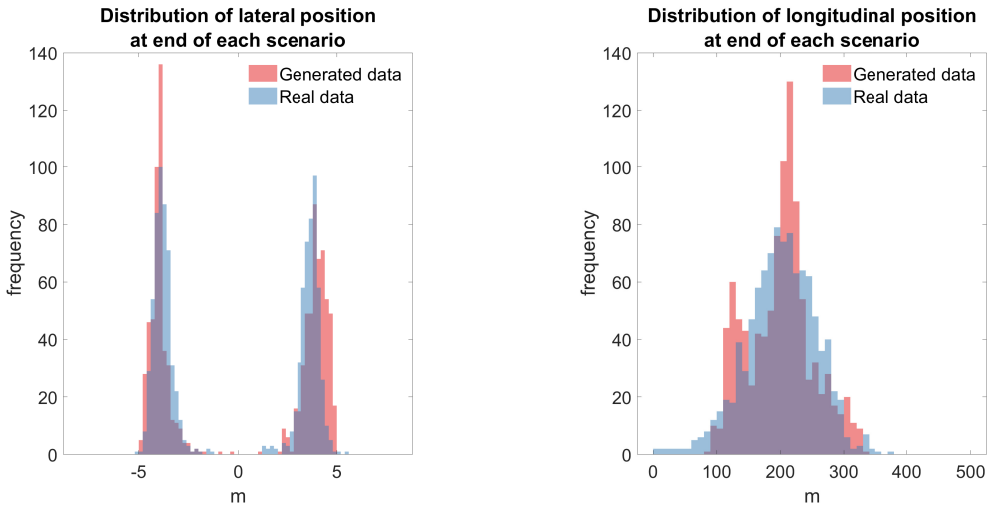
(a) 1^{st} difference lateral velocity **(b)** 5^{th} difference lateral velocity **(c)** 10^{th} difference lateral velocity

Figure A.3: Distribution of the error between each time step, each fifth time step, and each tenth time step in lateral velocity.



(a) 1^{st} difference longitudinal velocity **(b)** 5^{th} difference longitudinal velocity **(c)** 10^{th} difference longitudinal velocity

Figure A.4: Distribution of the error between each time step, each fifth time step and each tenth time step in longitudinal velocity



(a) Distribution of lateral position at scenario ending **(b)** Distribution of longitudinal velocity at scenario ending

Figure A.5: Distributions of the position where scenarios end. To the left is for generated lateral end position compared to the corresponding data in the validation set. To the right is the generated longitudinal end position compared to the corresponding data in the validation set

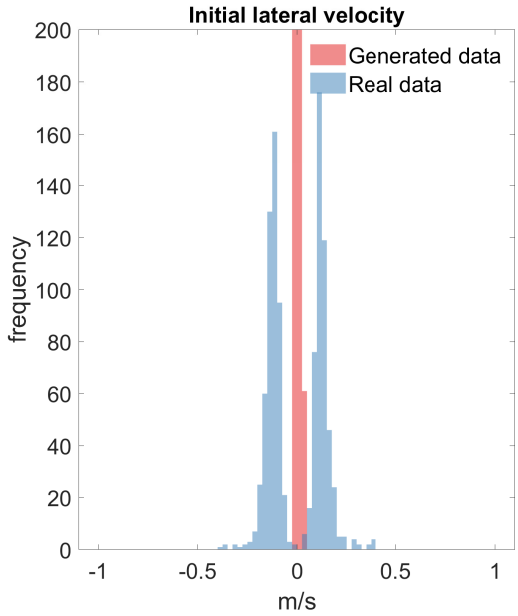


Figure A.6: Lateral velocity at time step 1

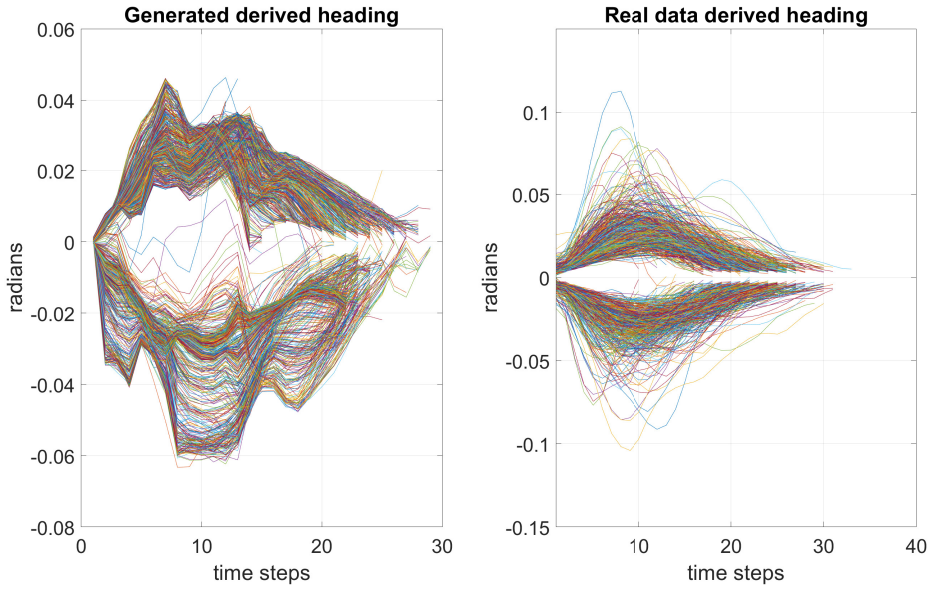


Figure A.7: Steering behaviour of 1000 grouped scenarios. Left plot is the derived heading from the generated data and the right plot is the derived heading from the real scenarios

A.2 Recurrent Variational Auto-Encoder

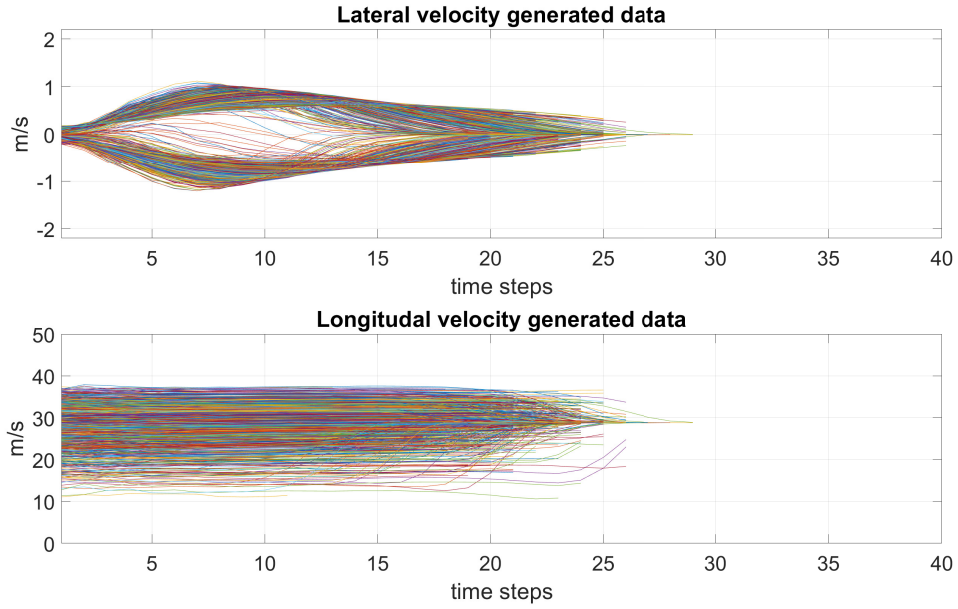
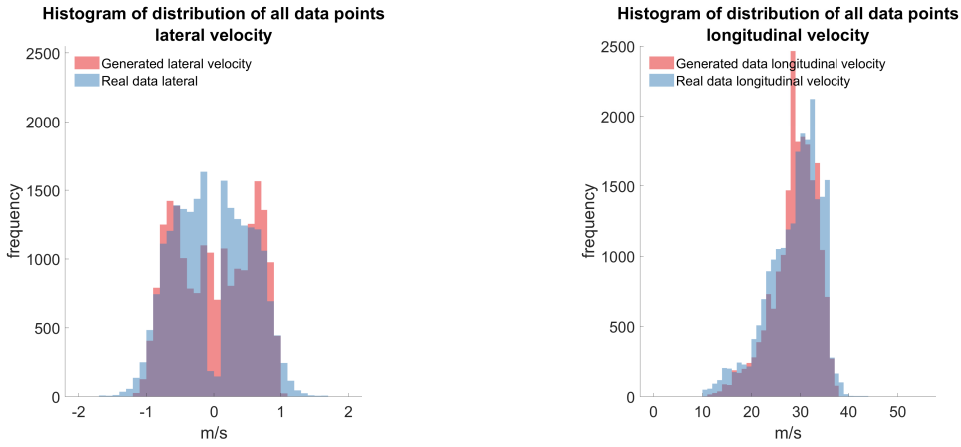


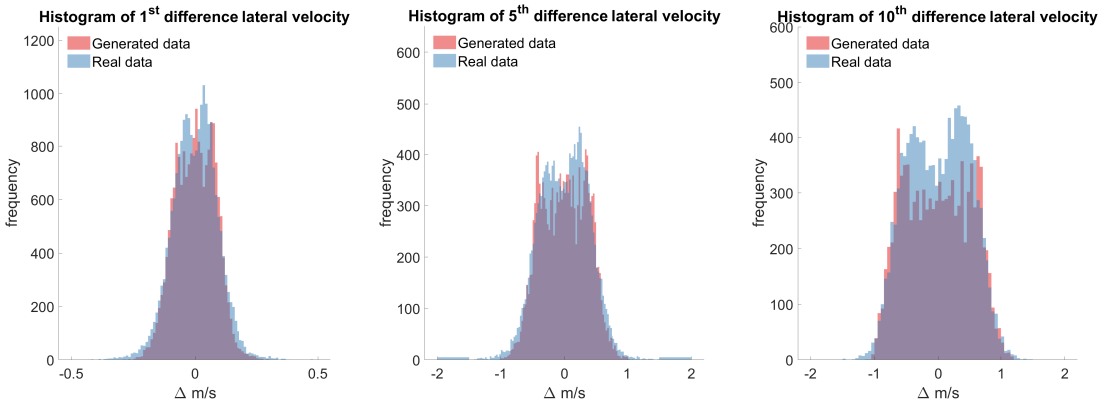
Figure A.8: 1000 generated scenarios of lane changes. Top plot illustrates lateral velocity and bottom illustrates longitudinal velocity



(a) Distribution of lateral velocity

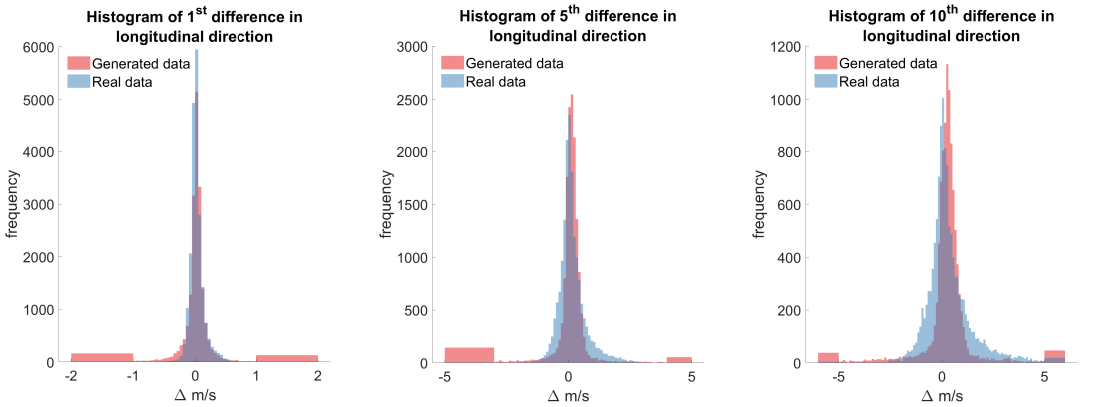
(b) Distribution of longitudinal velocity

Figure A.9: Distribution of the values of velocities in all time steps in a 1000 generated scenarios compared with the validation scenarios.



(a) 1^{st} difference lateral velocity **(b)** 5^{th} difference lateral velocity **(c)** 10^{th} difference lateral velocity

Figure A.10: Distribution of the error between each time step, each fifth time step, and each tenth time step in lateral velocity,



(a) 1^{st} difference longitudinal velocity **(b)** 5^{th} difference longitudinal velocity **(c)** 10^{th} difference longitudinal velocity

Figure A.11: Distribution of the error between each time step, each fifth time step and each tenth time step in longitudinal velocity

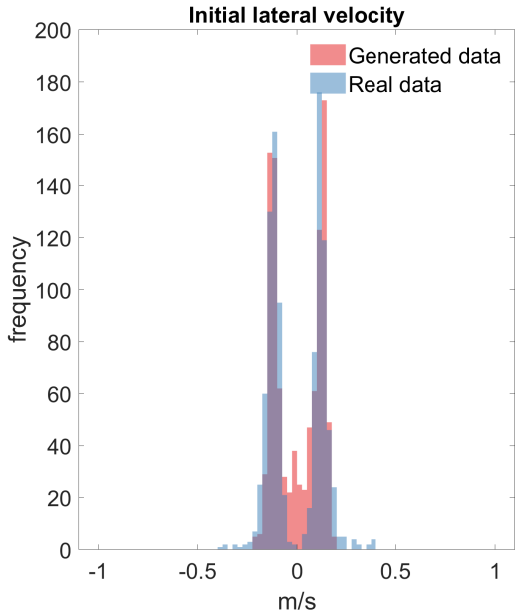
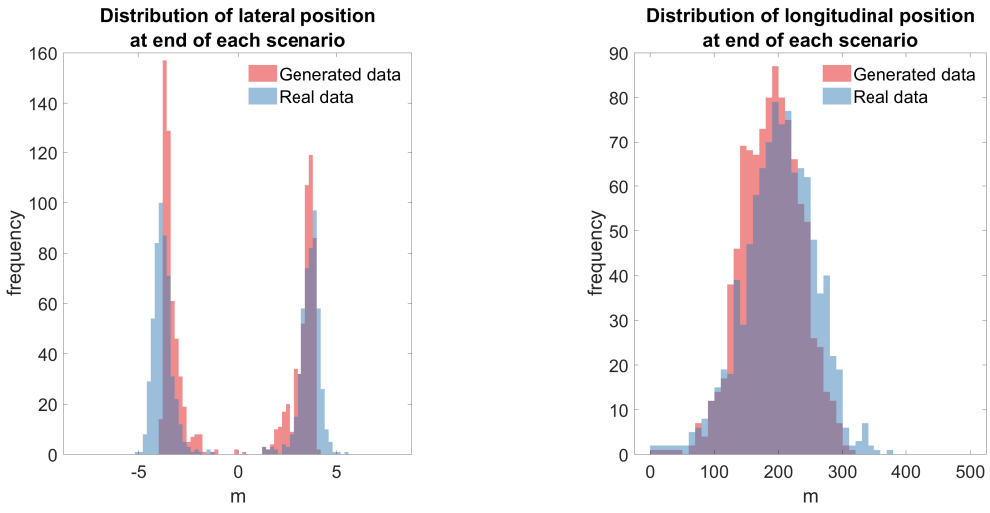


Figure A.12: Lateral velocity at time step 1



(a) Distribution of lateral position at scenario ending

(b) Distribution of longitudinal velocity at scenario ending

Figure A.13: Distributions of the position where scenarios end. To the left is for generated lateral end position compared to the corresponding data in the validation set. To the right is the generated longitudinal end position compared to the corresponding data in the validation set

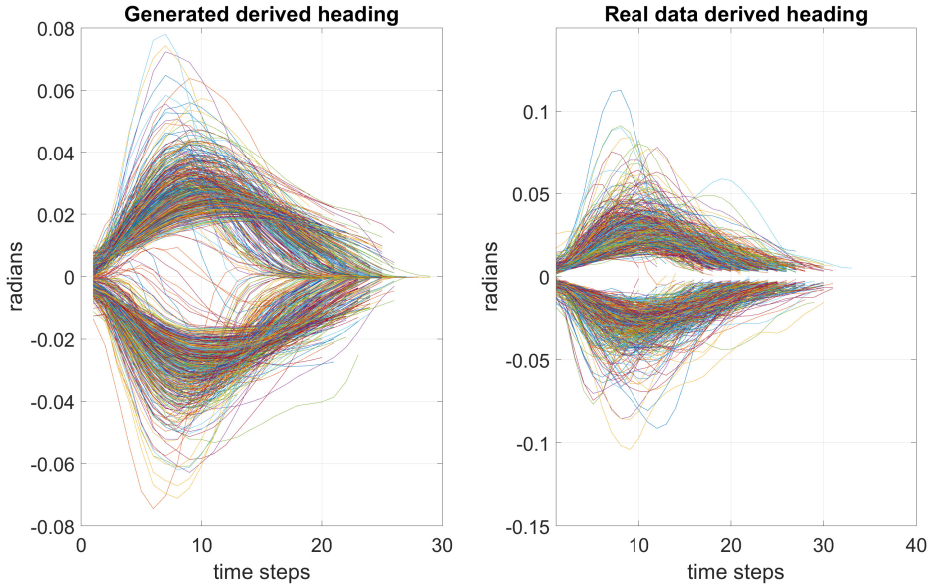


Figure A.14: Steering behaviour of 1000 grouped scenarios. Left plot is the derived heading from the generated data and the right plot is the derived heading from the real scenarios

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY