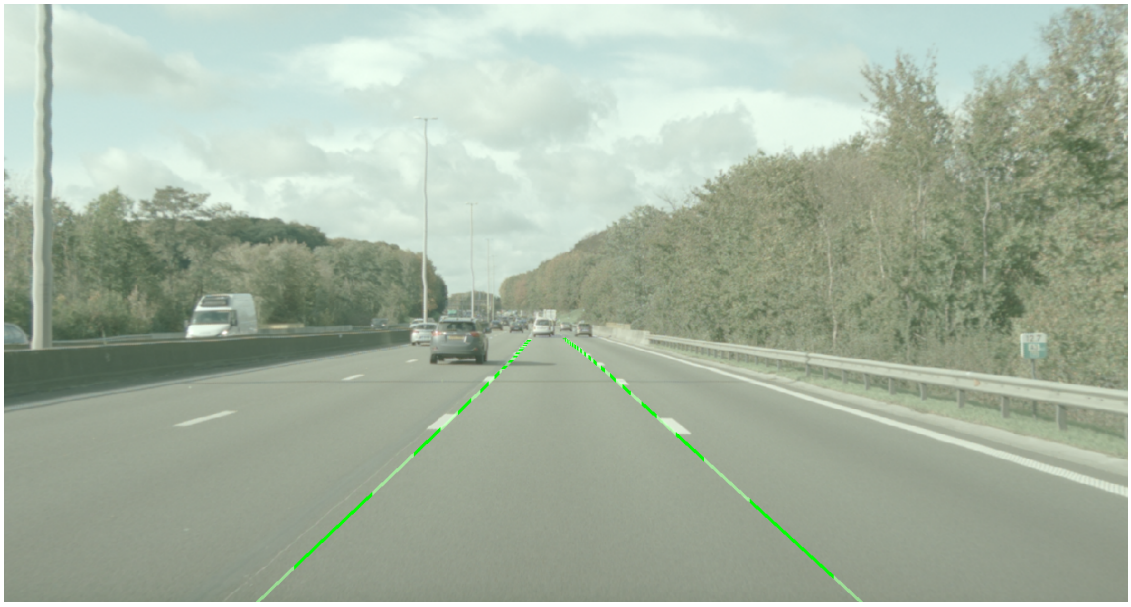
CHALMERS
UNIVERSITY OF TECHNOLOGY



Data Driven Lane Marker Filtering

A research on the capabilities of machine learning to replace hand-crafted heuristics

Master's thesis in Systems, Control & Mechatronics

Johan Karlsson

Carl Lindström

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Data Driven Lane Marker Filtering

A research on the capabilities of machine learning to replace hand-crafted heuristics

Johan Karlsson Carl Lindström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department Electrical Engineering
Division of Signal Processing
Chalmers University of Technology
Gothenburg, Sweden 2020

Data Driven Lane Marker Filtering
A research on the capabilities of machine learning to replace hand-crafted heuristics
Johan Karlsson, Carl Lindström

Supervisor: Huu Le, Department of Electrical Engineering
Jessica Andersson, Zenuity
Examiner: Christopher Zach, Department of Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Division of Signal Processing
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Resulting lane marker estimation from the reference system, which is described in section 2.6.

Typeset in LaTeX
Gothenburg, Sweden 2020

Data Driven Lane Marker Filtering
A research on the capabilities of machine learning to replace hand-crafted heuristics
Johan Karlsson, Carl Lindström
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Many functions within the field of autonomous driving and advanced driver-assistance systems require a solid estimation of the road ahead to operate properly, commonly provided via a Kalman filter. The filter utilizes information from different sources including perceived lane markers, under the assumption that these follow the traversed lane, with minor Gaussian distributed deviations. Thus, any non-Gaussian deviations or disturbances in the lane marker measurements must be rejected prior to the filtering. A feasible but tedious method is to continuously check for noise corrupted segments via hand-crafted heuristics, then truncate the lane markers such that these parts are not passed to the filter. This thesis proposes a machine learning solution for truncating the lane markers in the event of non-Gaussian disturbances, as a possible replacement to the hand-crafted heuristics. The thesis aims to investigate how well the machine learning solution performs compared to the heuristic system, in terms of resulting road geometry estimation, as well as finding the key characteristics for a suitable algorithm in this application. Models based on Support Vector Regression, Random Forest, Multilayer Perceptron and Long Short-Term Memory, are trained and evaluated, based partly on their ability to return predictions close to the corresponding annotations, but also on their resulting capability to aid the Kalman filter. The training data consist of 220 real-life driving hours, automatically annotated with adequate truncation spots, via a proposed method based on comparisons of lane markers and ground truth assessment. The Long Short-Term Memory excels in both performance measures, closely followed by the Multilayer Perceptron. The two types perform likewise, but the sequential understanding of the Long Short-Term Memory eventually lets it surpass the others. All algorithms, except Support Vector Regression, outperform the heuristic system, in terms of estimation quality on the used test set. The thesis eventually contributes with a method for replacing the hand-crafted heuristics with a machine learning model, with the potential to enhance the road geometry estimation, as well as increased generalization capability to new scenarios and other application areas.

Keywords: Machine Learning, Road Geometry Estimation, Autonomous Driving, Advance Driver-Assistance Systems, Neural Networks, Lane Marker Filtering, Auto-Annotation

Acknowledgements

First of all, we want to thank Christopher Zach and Huu Le for their academic advice and feedback during the writing of this report.
We would also like to thank Zenuity for the opportunity to finish our master studies at such an aspiring company. Furthermore, we want to especially thank the members of team Kalman for sharing their team spirit, as well as assisting us with our work. Last but not least, we would like to extend our greatest gratitude to Jessica Andersson. We could not have wished for a more engaged and helpful supervisor. Without her continuous guidance and support, the completion of this thesis would have been a lot more troublesome.

Johan Karlsson & Carl Lindström, Gothenburg, June 2020

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ACC Adaptive Cruise Control
AD Autonomous Driving
ADAS Advanced Driver-Assistance System
ANN Artificial Neural Networks
BPTT Backpropagation Through Time
CNN Convolutional Neural Network
FCA Forward Collision Avoidance
GTA Ground Truth Assessment
LKA Lane-Keeping Assistance
LSTM Long Short-Term Memory
MLP Multilayer Perceptron
MSE Mean Squared Error
RF Random Forest
RGF Road Geometry Fusion
RMSE Root Mean Squared Error
RNN Recurrent Neural Network
STD Standard Deviation
SVR Support Vector Regression

1
Introduction

This chapter will give an introduction to the fields of Autonomous Driving (AD) and Advanced Driver-Assistance System (ADAS), as well as Road Geometry Fusion (RGF). It also defines the purpose of the thesis, presents its contributions and the related work studied. The chapter is eventually closed with an analysis of the concerned ethical aspects and an outline for the remaining content of the report.

1.1 Background

The fields of AD and ADAS grew rapidly during the last decade and is continuing to do so. The incentives to the development of these systems are many, but potential safety improvements, enhanced driver convenience and increased sustainability are perhaps the most significant [4]. According to the National Highway Traffic Safety Administration, there were 37 461 fatal motor accidents on American roads in 2016. Around 94% of these were caused by human errors or mistakes, such as the driver being intoxicated, drowsy, distracted or inattentive [5]. With autonomous vehicles, these types of accidents can potentially be avoided, meaning that 35 213 lives could be saved every year in the U.S, based on the data from 2016.

ADAS can be seen as a subfield to AD, aiming not to replace the human driver, but providing warnings and assistance based on the current traffic situation. Examples of such systems are Lane-Keeping Assistance (LKA), Adaptive Cruise Control (ACC) and Forward Collision Avoidance (FCA) [6]. To ensure precision in both AD and ADAS, it is crucial that these systems are informed of the host vehicle's surroundings, as for instance the shape of the road ahead. Information about the road geometry can be obtained from several different sources, such as lane markers and the movement of preceding vehicles, which are in turn perceived by the different on-board sensors. Hence, an established method within the industry is to fuse the information from these sources using a Kalman filter, as described in [7]. Such a system, implemented in the context of road geometry estimation, will in this thesis be referred to as a RGF system. The Kalman filter creates an estimation of the road by fusing the sensor information with a model generated prediction, under the assumption that the measurement disturbances are Gaussian distributed [8]. In this application, this implies that the perceived lane markers and preceding vehicles are assumed to follow the lane geometry, with Gaussian distributed deviations. For the lane markers, this assumption holds for deviations caused by inaccuracies in their placement and the measurement noise in the perceptive sensors. Still, the lane markers may also deviate from the lane of interest when there is a highway exit in

front, as shown in figure 1.1. Such deviations and other misleading markers can not be considered Gaussian disturbances, which is why the Kalman filter fails to reject them.

Figure 1.1: Highway exit in Florida, where the right lane marker, if continuing straight ahead, is not following the lane of interest. Photo: [1], CC BY-SA 4.0

A feasible solution is to reject the non-Gaussian disturbances via handcrafted heuristics, prior to the measurement update in the Kalman filter. The heuristics contain logical gates, extracting information about the incoming lane markers by checking them against various conditions. Based on which logical conditions that are fulfilled and which gates that are passed, non-Gaussian disturbances can be detected and eliminated, before entering the Kalman filter. Although this approach works fairly well, the problem is that it requires a lot of engineering hours and the hand-coded gates must be continuously redirected for newly discovered disturbances and sensor updates.

However, the recent years' evolution of computing power and data accessibility allows for new types of solutions. Nowadays, data can be handled in greater volumes and at higher speeds, which has lead to the development of new tools, especially within the field of machine learning. As there typically exists a lot of valuable data in the domain of ADAS, in form of driver logs with estimated ground truth available, a possible replacement to the hand-coded gates could be a data-driven machine learning algorithm. Given a sufficient amount of data with descriptive features, the algorithm could potentially discover all relevant types of disturbances itself and learn how to reject them. Not only would this save engineering hours, but also cover up for disturbances not yet found or comprehended by the engineers. In the event of sensor updates, no time-consuming system recalibration would be required, but instead the machine learning algorithm would be retrained, which is an automatic process.

1.2 Purpose

This thesis aims to implement a machine learning algorithm, replacing the handcrafted heuristics rejecting non-Gaussian disturbances in the Road Geometry Fusion filter, within an existing reference system. Even though this type of disturbances occurs in the measurements of preceding vehicle movements as well, this thesis will solely focus on the filtering of lane marker measurements. The functional domain of the system is intended to be worldwide including both left- and right-hand traffic, on roads with visible lane markers, and during weather conditions where that visibility remains. Weather with acceptable visibility includes clear, cloudy and rainy conditions, for both daily and nightly driving.

The thesis ought to answer how well, in terms of the resulting road geometry estimation, a machine learning algorithm compared to handcrafted heuristics can be used to reject non-Gaussian disturbances in lane marker measurements. It further seeks to investigate what the key characteristics are for selecting a suitable machine learning algorithm in this application.

1.3 Related work

Machine learning algorithms have successfully replaced handcrafted heuristics before, for instance in [9], where a neural network was able to outperform heuristic algorithms with the objective of playing different board games. Although the application is different from the setting of this thesis, it shows that a machine learning algorithm has the potential to learn a specific task and eventually outperform handcrafted heuristics.

Previous work has also been done with machine learning algorithms in combination with Kalman filtering. Attempts to improve the performance of a Kalman filter with machine learning has for instance been completed successfully in [10]. In this project, the filter is made adaptive to noise level changes, by continuous adjustments of the filter parameters with a machine learning algorithm. The aim of this thesis is not to control the filter itself, but adjusting the measurements prior to their entry in the filter. Nevertheless, controlling the filter in this way could be an alternative solution and is proof of the potential in this type of system chain.

As mentioned in section 1.1, the fields of AD and ADAS have grown a lot lately and consequently a vast amount of research projects have been carried out within detection and classification of lane markers. A review of different techniques for road and lane marker detection is given in [11], which also suggests further use of machine learning algorithms. It adds to the collection of works indicating that machine learning has the potential to improve several AD and ADAS functions. An example of a fruitful lane detection system that employs machine learning techniques is [12]. In this project, a Convolutional Neural Network (CNN) is used together with a Long Short-Term Memory (LSTM) to estimate the road geometry from camera images. The LSTM algorithm is used in several similar applications, such as systems for predicting the movement of other vehicles [13][14]. Although the aims of these applications are different from this particular project, the environment of usage and

data at hand are similar.

An extensive performance comparison of different machine learning algorithms in different tasks has been done in [15]. The study shows that for regression tasks, Support Vector Machines, Neural Networks and Random Forest, are the most prominent of the tested algorithms.

Recently, there have also been advancements in the field of neural filtering. Neural filters are machine learning algorithms that are trained to interpret noisy measurements and serve as an alternative to more conventional filter methods. There are several examples of successful implementations of neural filters, for instance [16], where it has been applied to reject noise in electroencephalogram signals. The developed neural filter manages to reduce both additive and multiplicative noise, with good preservation of the signal characteristics. This project is an indication that a neural filter could be a profitable replacement to the current heuristics. It is also an inspiration for the methodology of developing a machine learning algorithm for noise rejection.

In contrast to previously mentioned projects, [17] presents a method for robust Kalman filtering, without involving machine learning. The proposed solution is based on weighting each measurement, such that detected outliers are assigned a lower weight and thereby less influence on the resulting estimation. The weights are modeled as gamma-distributed variables and estimated via an Expectation-maximization framework. As already discussed, this thesis aims to exploit a machine learning solution, but the idea with weighted measurements could very well be included in that. Another interesting project free of machine learning is [18], which presents an alternative way of robust Kalman filtering, based on avoiding the assumption of Gaussian noise only. The suggested solution is substantially different from the intended approach of this thesis, but it is a good example showing that machine learning and heuristics are not the only ways of improving the Kalman filter.

1.4 Contribution

Instead of rejecting the non-Gaussian disturbances in the lane marker measurements via hand-crafted heuristics, this thesis proposes a machine learning method, which is less tedious and can enhance the filter performance even further. The method adds to the list of possible combinations of machine learning and conventional filters, which can be well transferred to other fields and applications.

As an alternative to the existing end-to-end machine learning systems for road geometry estimation, this thesis suggests a modular solution. In contrast to the end-to-end systems, the modular solution does not require that the user has access to the complete product chain, from perceptive sensors to the road geometry estimation.

Lastly, the thesis contributes with a method that, based on collected ground truth geometry, can auto annotate the disturbance-free sections of the perceived lane markers. This allows for a supervised machine learning model to be trained on the resulting data, which in turn, if trained successfully, can be used to perform the task of rejecting disturbances. It is important to note that the quality of the proposed annotation method has a considerable impact on the machine learning

models' filtering performance.

1.5 Ethical Aspects

The development and implementation of the machine learning algorithm has no direct effect in terms of ethical or sustainable aspects. However, the algorithm is a contribution to the advancement of autonomous vehicles and active safety systems, which have or will have, a big impact on society.

As mentioned in section 1.1, 94% of the fatal motor accidents in the U.S during 2016, were caused by human errors. Since AD and ADAS systems can reduce or potentially eliminate these errors, they can increase traffic safety and furthermore supply the fulfillment of the third UN sustainability goal, regarding good health and well-being [19]. Of course, this demands rigorous testing, which is yet another technological area, subject to several ethical concerns.

Furthermore, while benefiting health through increased safety, autonomous vehicles could also improve the overall convenience of transportation. Not only would there be fewer cars on the roads, but they could also communicate and cooperate, achieving smooth traffic flow and reduced traffic congestion. As a result, transportation would be quicker and more convenient, allowing people to put time and focus into other matters and potentially relieving stress. In the long run, improved transportation capabilities could facilitate the control of urbanization, preventing decay of smaller settlements and rural areas.

In terms of sustainability, autonomous vehicles allow for a considerable change to transportation and infrastructure. Self-driving cars could facilitate shared ownerships and carpools, serving as a type of public transportation, in favor of increased efficiency and utilization of each vehicle [4]. Hence, autonomous vehicles can be considered a valuable contribution to the ninth UN sustainability goal, about building resilient infrastructure [19].

Unfortunately, not all consequences of bringing AD and ADAS to public vehicles are positive. ADAS systems partially operate via warnings, but may also involve interventions and control of the car, in order to avoid critical situations and accidents. The access to the car controls is a prerequisite for these systems to work, but can also be perceived as an inflict on human autonomy and freedom of choice. Who to declare responsible in case of an accident involving a car with this type of system is a question that remains to be answered. Similarly, there are no established strategies or priorities, for facing multiple threat situations. The trolley problem is often used as an example of such an ethical dilemma and is highly applicable to the development of critical decision making in autonomous vehicles.

Another important aspect is the general impact of increased automation in society. More and more daily tasks, such as driving to work, are being automated. This diminishes the significance of the human individual and potentially leads to increased unemployment, which can be psychologically harmful to society in several ways. Furthermore, people may doubt the functioning of automation, consequently making them feel unsafe in the presence of for instance autonomous vehicles. This matter is aggravated for machine learning systems, as the functionality of these is typically hard to validate.

1.6 Thesis Outline

The work carried out in this thesis is presented according to the following structure. Chapter 2 presents the background and theoretical concepts necessary for this thesis. Here concepts about classical machine learning and Artificial Neural Networks (ANN) are presented, along with an introduction to Kalman filtering and the notion of Road Geometry Fusion (RGF). Chapter 3 describes the methodology used in the thesis. The chapter starts with an introduction to the proposed conceptual system used for evaluation, and is followed by explanations of the methods applied for processing and automatic annotation of the dataset. The chapter closes with a brief description of the algorithms used in the thesis, along with the tools used for evaluating them. In chapter 4, the results from evaluating the chosen methods are presented. The chapter begins with results from evaluations on the annotated data and is followed by simulations in the proposed conceptual system. The chapter ends with results from evaluating the function used for annotating the dataset, as well as a brief presentation of the time analysis performed on the algorithms. The results are then analyzed and discussed in Chapter 5, together with a discussion of the chosen methods and suggested topics for future work. Finally, Chapter 6 concludes the findings in the thesis.

2
Theory

This chapter intends to give a brief but sufficient presentation of the fundamental theory that the method and discussion in this thesis are based on. The first section gives an introduction to machine learning and covers different common practices within the field. Following this, there are four sections that elaborate on the details of four different algorithms. The chapter is then closed with two sections describing general Kalman filtering and the Road Geometry Fusion (RGF) filter.

2.1 Introduction to Machine Learning Regression

Classic machine learning is usually divided into two subcategories, namely supervised and unsupervised learning. In supervised learning, a set of input vectors X_i and their corresponding outputs Y_i are provided by the user, and the algorithm models the mapping function $Y = f(X)$ which it can then use to predict the output from unseen inputs. For unsupervised learning, no output examples are provided and instead the algorithms are focused on information extraction from the available data [20]. The supervised learning category is further divided into classification and regression algorithms. Classification is for outputting distinct classes like "cat" or "dog", while regression is for outputting continuous values, like house prices [21][22]. This section is initiated with a brief description of a common problem within supervised learning and then proceeds to describe how the performance of a machine learning algorithm can be evaluated.

2.1.1 Overfitting and Underfitting

A common problem within machine learning is that the algorithms may adapt too well to the training data, thus losing their ability to generalize and make correct predictions from unseen data. It can be a consequence of having too much flexibility in the model or having unbalanced training data, meaning that the training only includes a subset of the cases or combinations of inputs that the algorithm will face. In this particular case, this could for instance happen if the algorithm would be trained solely on single-lane driving scenarios, making it deficient on highway driving where there are multiple lanes.

The problem of fitting the prediction model too well to the training data is called "overfitting", which different algorithms suffer from to different extents. Hence, the algorithms also have different ways of regularizing their prediction model and dealing with this problem. However, a general way to prevent overfitting is to extend

the training data set, such that its variance increases and forces the algorithm to generalize. Still, the amount of data at hand is often limited and this solution only works if the added data points are different from the existing ones, as only then would the algorithm have to generalize further [23].

The opposite problem of overfitting is called "underfitting", which occurs when the model has insufficient capacity to fit to the training data. Finding the right balance between overfitting and underfitting, which means matching the complexity of the model to that of the task, is one of the main challenges in machine learning. Of course, the model can also seem to fit the data poorly when there are not enough data points available. To conclude, more data is always beneficial, but does not guarantee an improvement if the model capacity is incorrect. The latter is a matter of tuning, which is done differently for different algorithms [24].

2.1.2 Handling the Available Data

The purpose of supervised machine learning is to eventually have a model that can output a prediction or an estimate, based on previously unseen data points. Thus, when evaluating its performance to do so, it is important not to test it on the same data it was given during the training process. This means that, before training, the available data must be divided into separate sets for training and testing. For successful training as well as a complete and fair test, it is important that both of the individual sets are representative of the full dataset. Since the training process benefits from increased amounts of data, it is desirable to maximize the size of the training set, without disregarding the sets' levels of representation. It is common to assign about 75% of the data for training and 25% for testing, but the more data that is available, the larger the proportion of the training set can be.

Developing a machine learning model usually consists of many iterations of training and testing, to find suitable parameters for the application. To make sure that the development is not biased towards the test set, it is common practice to further divide the training data into a training set and a validation set. The latter is used instead of the test set during development, such that when the model is properly tuned, it can be evaluated on the test set [20].

2.1.3 Evaluation Metrics

The choice of metrics used to evaluate different machine learning algorithms is of great importance as it influences how different characteristics in the model are valued, and ultimately lies as a foundation for the choice of algorithm. For classification problems, the most common evaluation metric is accuracy, i.e the ratio of correct predictions out of total predictions made. However, for regression problems, this is not a possible metric as there are no distinct classes. Instead, regression metrics are constructed to tell how much the prediction deviates from the true value. Commonly used regression metrics are Root Mean Squared Error (RMSE) and R^2 score.

2.1.3.1 R^2 Score

The R^2 score, also known as coefficient of determination, is a regression metric used to evaluate the performance of a model compared to a constant baseline. It is defined as the proportion of variance in the dependent variable explained from the independent variables, making it a suitable measure of successful predictions for regression models [25]. The R^2 score is given by,

$$R^2 = 1 - \frac{\sum\limits_i (y_i - f_i)^2}{\sum\limits_i (y_i - \bar{y})^2} \tag{2.1}$$

where y_i is the actual value, commonly known as target, and f_i is the predicted value. $\bar{y}$ is the mean of the targets, and is given by,

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i. \tag{2.2}$$

The score ranges from $-\infty$ to 1, where 1 is the best possible score.

2.1.3.2 Root Mean Squared Error

The Root Mean Squared Error (RMSE) is a widely used evaluation metric for regression models which measures, as the name suggests, the square root of the averaged squared prediction error. The RMSE is given by,

$$RMSE(\mathbf{f}, \mathbf{y}) = \sqrt{\frac{\sum\limits_{i=1}^{n} (f_i - y_i)^2}{n}}, \tag{2.3}$$

where y_i is the actual value, also known as target, and f_i is the predicted value. Since the errors are squared before taking the average, the RMSE gives a high penalty on large errors, making it useful for evaluating regression models were large errors are undesirable.

2.2 Random Forests

Decision Trees is a simple and intuitive, yet powerful machine learning algorithm, which can be used for both regression and classification. Figure 2.1 shows a simple decision tree, made to estimate the width of a lane, based on a few simple features. Every node has a related condition and if the condition is fulfilled, the algorithm continues along its right branch to the next node and condition, from the top and down. The splitting condition at each node is automatically chosen such that the variance of the resulting data subsets is minimized [22].

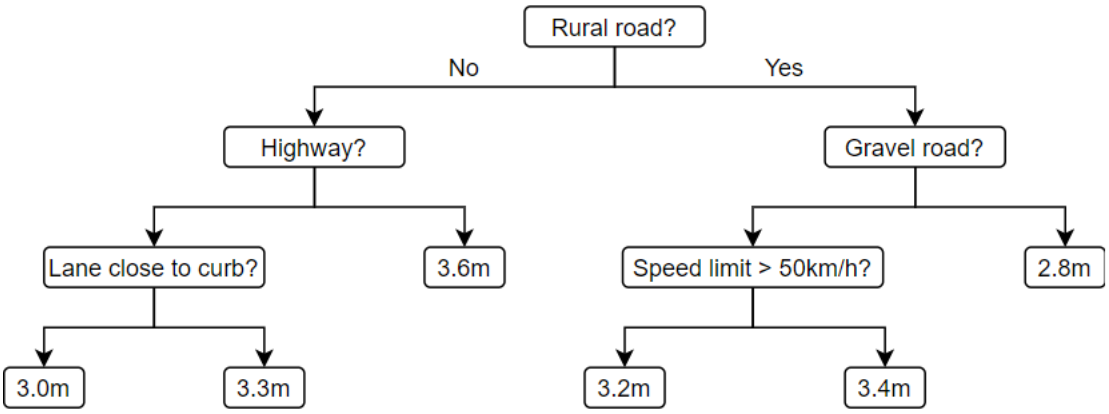


Figure 2.1: A simple decision tree estimating the width of the traversed lane. If the condition at a node is fulfilled, the algorithm continues along its right branch.

As previously mentioned, overfitting is a common problem within machine learning and decision trees are particularly sensitive to this. In a worst-case scenario, the tree can develop one ending node (leaf node), for every training example. This sensitivity has led to the development of Random Forest (RF), which is an extension to Decision Trees with the aim to prevent overfitting. It is an ensemble, or a ”forest”, containing several different decision trees. Every tree is based on a random subset of the input features and sometimes, it is solely fitted to a random subset of the training samples as well. This makes the trees different from one another and reduces their mutual risk of overfitting. The output of the forest is computed as an aggregate of every individual tree in the forest, which also limits the influence of any potentially overfitted trees [26].

Preventing overfit of a RF algorithm is mainly about controlling the growth of branches. The number of branches can either be restricted from the start (known as pre-pruning) or reduced after training (known as post-pruning). There are a few hyperparameters to control the pruning and the main two are the number of trees to use, along with their maximum depth. It is also possible to control the number of leaves and the minimum amount of training examples that must fall under each of them. Another effective way of limiting the growth of the trees is to set a minimum decrease in variance that a node must yield in order to grow [20].

2.3 Support Vector Regression

Support Vector Regression (SVR) is a machine learning algorithm that is based on fitting a hyperplane H to the data points x_i. The hyperplane can then be used to predict values from other sets of inputs. Finding the best weights w and bias b for $H(x) = wx + b$ is a matter of solving the optimization problem

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ & \textit{subject to} \\ & y_i - w \cdot x_i - b \leq \epsilon + \xi_i \\ & w \cdot x_i + b - y_i \leq \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{2.4}$$

where y_i is the corresponding label to the sample x_i, C and ϵ are tuning parameters and ξ is an optional variable for softening the constraints. The problem of overfitting is partially prevented by only adjusting the prediction model to errors above the threshold ϵ, which is user-controlled. To regularize the algorithm even harder, it is also possible to introduce soft constraints and tolerate deviations larger than ϵ to an extent controlled via the hyperparameter $C > 0$. This parameter is inversely proportional to the regularization strength, meaning that a small C allows for more violations of ϵ [27].

Even though the hyperplane is a linear function, the algorithm can model complex non-linear relations between input and output, thanks to the use of kernels. As shown in [27], the problem in (2.4) can be reformulated as a dual problem, via the so-called "Support Vector Expansion". In this formulation, the weights in w are described as dot products between the training samples x_i. What the kernel $K(x_i, x_j)$ does, is that it computes the dot products in an alternative vector space, to which the mapping can be non-linear. The idea is that, in contrast to the original feature space, it is possible to fit a hyperplane to the points in the alternative space. Different kernels compute the dot product in different ways and which kernel to choose depends on the application. However, some of the most commonly used are

- Linear kernel
$$K(x_i, x_j) = x_i^T \cdot x_j \tag{2.5}$$
- Polynomial kernel of degree d
$$K(x_i, x_j) = (\alpha x_i^T x_j + c)^d \tag{2.6}$$
- Sigmoid kernel
$$K(x_i, x_j) = \tanh(\beta x_i^T x_j + c) \tag{2.7}$$
- Gaussian Radial Basis Function (RBF)
$$K(x_i, x_j) = \exp -\frac{||x_i - x_j||^2}{\alpha^2} \tag{2.8}$$

where x_i, x_j are two different samples in X and α, β, c are tuning parameters of the kernels. The first one is simple and fast but limited to simpler problems. The

second is more advanced and for instance widely used within image processing. Lastly, the third and fourth are general-purpose kernels, which are useful when the data knowledge is limited [28][29].

Except for the choice of kernel and its settings, ϵ and C are the only hyperparameters that require tuning in SVR, which makes it relatively fast to get started with.

2.4 Artificial Neural Networks

Artificial Neural Networkss (ANNs) are biologically inspired computational systems that can learn to perform complex tasks by adapting several simpler computational nodes, often referred to as neurons, to observed data. ANNs make up the foundation of deep learning, and as been a heavily researched field in recent years, resulting in several advances in the fields of image processing and speech recognition. For a more comprehensive read about deep learning and the fundamentals in ANNs, the reader is referred to the book "Deep Learning" by Ian Goodfellow, Yoshua Bengio and Aaron Courville [24]. This chapter will introduce the ANN-related topics and concepts used in this thesis.

2.4.1 Multilayer Perceptron

The Multilayer Perceptron (MLP) belongs to the class of feedforward Artificial Neural Networkss (ANNs) and consists of three or more layers of fully connected neurons [30]. The connection between neurons in adjacent layers is defined by a set of weights, a bias and an activation function. For a hidden layer in the network, the output from the hidden neurons is computed as

$$\mathbf{h} = g(\mathbf{W}^T\mathbf{x} + \mathbf{c}) \tag{2.9}$$

where $\mathbf{W}$ is a matrix with the weights for the neurons in the layer, $\mathbf{x}$ is the output from the previous layer, $\mathbf{c}$ is the vector of biases and g is the activation function for this layer. The output from the hidden layer is then fed as input to the next layer, creating a feedforward chain all the way to the output layer. The weights and biases are then adjusted based on the output error, with the use of an optimization algorithm. The number of layers in the network is usually referred to as the depth of the model, which defines the length of the chain from input to output [24]. A deeper network usually increases the flexibility of the model, allowing it to approximate more complex functions, but with the cost of more parameters to train and thus an increased computational time, as well as a higher risk for overfitting.

2.4.2 Activation functions

Activation functions are used in artificial neural networks to define the output of a node, given some input. They act as a final gate in the neuron and are usually designed to introduce non-linear transformations between the layers, to allow the network to learn more complex patterns from the training data.

2.4.2.1 Rectified Linear Activation Function

The Rectified Linear (ReL) activation function is one of the most commonly used activation function in neural networks, and has been proven to accelerate training time and improve the performance in several different applications [31][32][33][34]. It is mathematically given by,

$$h^{(i)} = max(w^{(i)T}x, 0) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0 & \text{else} \end{cases}, \tag{2.10}$$

where w^i is the weights of layer i and x is the input from the previous layer. It is activated for all output above 0, and otherwise saturate the output to exactly 0 [33]. The ReL function solves the problem of vanishing gradients in the activated paths of the deep network, as the partial derivative of an activated ReL unit is 1. A potential disadvantage of ReL units comes from the fact that the gradient is 0 for in-active units during optimization. The weights of some units might thus never be adjusted by the gradient-based optimization algorithm, as units that never activates initially will not get their weights updated. Further, this could lead to slower training. To solve this problem, a version called Leaky Rectified Linear (LReL) units can be used. The leaky rectifier is instead given by,

$$h^{(i)} = max(w^{(i)T}x, 0.01w^{(i)T}x) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0.01w^{(i)T}x & \text{else} \end{cases}. \tag{2.11}$$

The LReL function is very similar to the original ReL function but allows a small, positive gradient for in-active units.

2.4.2.2 Sigmoid Function

Another common activation function in artificial neural networks is the sigmoid function, given mathematically by

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{2.12}$$

where z is the result of propagating the input through the layer. The result of the sigmoid is then fed as input to the next layer, or as the network output when used in the final output layer. It is specifically popular in classification problems as it restricts the output to exist between 0 and 1, making it suitable for predicting probabilities. Further, it is nonlinear, monotonic and continuously differentiable, which are desirable properties for an activation function.

2.4.2.3 Hyperbolic Tangent Function

The hyperbolic tangent function, also known as tanh, is like the sigmoid function a S-shaped function. It is defined as

$$\tanh z = \frac{\sinh z}{\cosh z} = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{2.13}$$

where in neural networks z is the output from the layer propagation. The result of tanh(z) is mapped between -1 and 1 and is fed as input to the next layer, or used as final output of the network.

2.4.3 Loss function

When training a neural network, the weights and biases, also known as learnable parameters, are updated based on some defined loss function that measures the difference between the predicted values and the targets. In regression problems, the loss is commonly defined as the Mean Squared Error (MSE), which is given by

$$MSE(\mathbf{f}, \mathbf{y}) = \frac{\sum_{i=1}^{n} (f_i - y_i)^2}{n} \tag{2.14}$$

where y_i is the target and f_i is the predicted value. The MSE is a widely used loss function due to being simple, continuous and differentiable. It also has the key characteristic of being especially sensitive to large errors, making a model trained with MSE as loss function biased to reduce larger errors at the cost of many smaller ones. This makes it a suitable loss function for this thesis, as automotive systems are heavily safety-critical where large errors could risk the harm of human-beings.

2.4.4 Back-Propagation

During the training of a feedforward neural network, different input samples are individually propagated through the network, which is a process called "forward propagation". The output is then compared to the corresponding ground truth of the training sample, which via the chosen loss function forms a scalar cost $J(\theta)$ as a function of the trainable parameters θ. With "back-propagation", this cost or loss is then processed backward through the network to compute the gradient $\nabla J(\theta)$ with respect to θ [35]. Knowing the gradient of $J(\theta)$, it can be optimized using for instance gradient descent. Briefly explained, gradient descent stepwise adjusts θ in the negative direction of the gradient, until $\nabla J(\theta) \approx 0$ and an optimum has been reached [24]. The next section will elaborate on the optimization of the loss function and different methods for doing so.

2.4.5 Optimization

The goal during neural network training is to update the learnable parameters, θ, of the network, such that the cost, $J(\theta)$, is significantly reduced. The algorithms used for this differ from traditional optimization algorithms in many ways, partly due to the fact that neural networks, and machine learning models in general, act indirectly. In most settings, the goal is some performance measure P, from which a different cost function $J(\theta)$ is reduced, with the assumption that minimizing $J(\theta)$ will improve the performance P. Preferably, the goal of the machine learning model should be to minimize the expected loss over the whole data distribution p_{data}, contrary to just the training set. This measure is known as the risk, and is thus given as

$$J^*(\theta) = \mathbb{E}_{\mathbf{x},y\sim p_{data}} L(f(\mathbf{x};\theta), y). \tag{2.15}$$

However, the whole distribution is not known in machine learning problems, as that would be a pure optimization problem. The true distribution is instead replaced

by the empirical distribution from the training samples, $\hat{p}(\mathbf{x}, y)$. This is known as minimizing the empirical risk and is given as

$$\mathbb{E}_{\mathbf{x},y\sim\hat{p}(\mathbf{x},y)}[L(f(\mathbf{x};\theta),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\mathbf{x}^{(i)};\theta),y^{(i)}), \tag{2.16}$$

where m is the number of samples in the training set. The risk is then optimized indirectly, by optimizing the empirical risk and hoping that the true risk decrease as a result. As can be seen from this, it is of great importance that the training set $\hat{p}(\mathbf{x}, y)$ is an accurate distribution of the whole data-generating distribution p_{data}. Further, it is also important to have a sufficiently large training set, in relation to the complexity of the model to train. Models with higher capacity, such as deep networks, are prone to overfitting, especially for smaller sets of training data. The model can simply memorize the training samples and thus reduce the cost in equation (2.16) significantly, but will generalize poorly to new unseen data. One way of preventing this is to increase the training set, forcing the model to generalize to more data and variance. In practice, the model is usually validated on subsets of the dataset that are not available during the training of the algorithm, to identify trends of overfitting.

One of the more recent and popular optimization algorithms used in deep learning is the *Adam* optimization algorithm, proposed by Diederik Kingma and Jimmy Ba in 2015 [36]. It is an extension to the classical stochastic gradient descent and utilizes separately adapting learning rates for each network weight. It has shown to work very well in practice, and outperform other well-known adaptive learning methods [36][37]. It is a common choice for deep learning and is the optimization method used for training of all neural networks in this thesis as well.

2.4.6 Mini-Batch Gradient Descent

Gradient descent based optimization algorithms are commonly used for updating the weights when training a neural network, and are often integrated and ready-to-use in most popular machine learning libraries such as Scikit-learn and Keras. However, when using these optimizers there is the choice of batch size. The batch size refers to the number of training examples used in one iteration and can be altered to change the learning behavior of the algorithm. Using a batch size of one, i.e one training sample per iteration, is often referred to as stochastic gradient descent. This variant is perhaps the easiest to understand and implement and can, depending on the problem at hand, result in faster learning due to the high frequency of model updates. There is also an added upside of getting an immediate response to the model performance and its improvement rate. The noisy update process can help in avoiding premature convergence, i.e getting stuck in local minima, but can also make it hard for the algorithm to find a stable minimum to settle on. The high update frequency is also computationally expensive on larger datasets as it requires m model updates per complete cycle through the training data, where m is the number of samples in the dataset. On the other side of the spectrum, there is the method of using all available training samples for each model update, referred to as batch gradient descent. It is more computationally efficient due to fewer updates and

often results in a steadier convergence, due to a more stable gradient of the error. However, this method has a risk of premature convergence, due to the more stable error gradient. There is also an added complexity of accumulating the prediction errors over the whole batch of samples, and is commonly implemented such that the whole dataset is needed to fit in memory. This, together with large model updates, makes it hard to use batch gradient descent for large datasets. The middle ground between these two methods is to split the dataset into smaller batches to update the model on, and is called mini-batch gradient descent. It is a trade-off between the more robust stochastic gradient descent and the more efficient batch gradient descent, that has proven to work well even for large datasets in the field of deep learning. The method has the benefit of utilizing a more efficient update process than the stochastic version, but has a higher update frequency and thus a more robust convergence than the batch gradient descent. It also avoids the problem of having to fit the entire training set in memory. However, the mini-batch gradient descent adds the extra work of having to find a suitable mini-batch size. This can be seen as an additional hyperparameter to the algorithm and can require a lot of tuning before finding a suitable size. The parameter works as a slider between the stochastic version and the full batch version where, in general, a smaller value will result in a faster converging learning process with more noise, while a larger value will have a slower convergence but with a more accurate and stable error gradient. A good starting value is usually considered to be a mini-batch size of 32, which is also the default value in the popular machine learning library Keras.

The batch size is typically chosen as a power of two, for instance, 32, 64, or 128. This is done since both CPU and GPU computations can benefit from this in terms of runtime, due to the functioning of their storage capacities [24]. As seen in several well-known architectures, for instance VGG-16 and DenseNet-121, it is also common to choose the number of neurons in a MLP this way.

2.4.7 Long Short-Term Memory

It is not unusual to have problems where the data is in the form of time series, meaning that the order of the data holds a lot of information. This is the case when the state at a given time step is dependent on previous states. For example, when predicting price changes of a stock, it is useful to know its price history, since it usually follows a trend. Similarly, to determine whether the curvature of a perceived lane marker at a given time step is reasonable or not, it may be convenient to know the curvature from previous time steps.

Fortunately, there are special types of ANNs created to profit from sequential information, namely Recurrent Neural Networks (RNNs). The simplest form of a RNN is a network where parts of the output are continuously cycled back to its input. This functionality is achieved through the architecture seen in figure 2.2, which showcases two different illustrations of the same network. In the leftmost illustration, the network is fed a sequence of inputs, but since the time steps are actually processed one by one, the RNN can be unfolded into a sequential view, shown to the right. The cycling or storage of information is done using a hidden state h_t, which is a function of the previous hidden state h_{t-1} and the input at time step t, x_t. The hidden state is computed as

$$h_t = g_h(Ux_t + Vh_{t-1} + b_h) \tag{2.17}$$

where U, V are weights, b_h is a bias and g_h is an activation function. Similarly, the output o_t is computed as

$$o_t = g_o(Wh_t + b_o) \tag{2.18}$$

where g_o, W, b_o are specific for the output layer. Depending on the application, the user can choose to extract either one or multiple time steps of the output sequence o.

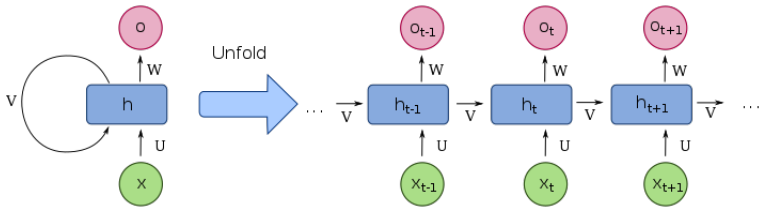


Figure 2.2: The unfolding of a RNN, with weights U, V, W and hidden state h. Furthermore, x is a sequence of inputs and o is the corresponding output sequence. Figure: [2] CC BY-SA 4.0

In contrast to regular feedforward networks, the output of a RNN is dependent on multiple inputs and thus multiple propagations through the network. Hence, the corresponding prediction error of the output is an accumulation of several input propagations. Furthermore, these propagations are mutually dependent, since the input at a given time step contains the output from the former step. This is why RNNs are trained using a special version of backpropagation, called Backpropagation

Through Time (BPTT), which can be studied in detail in [38]. Just like regular backpropagation, the idea in BPTT is to calculate the error gradient. For one sequence, the gradient is equal to the sum of the gradients corresponding to each time step, i.e

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W} \tag{2.19}$$

Due to the dependency between input and output, each of these individual gradients has to be computed with respect to the error of previous time steps. For longer sequences, this yields lengthy chain gradients that can lead to a problem known as "vanishing or exploding gradients", which in turn prevents the RNN of learning long time dependencies [39].

However, Long Short-Term Memory (LSTM) is a more complex version of the recurrent unit, which is able to deal with this problem [40]. The idea is based on the addition of the cell state c_t, which is somewhat similar to the hidden state h_t. The difference is that the cell state is not continuously updated in the same way. Instead, the cell state content is handled by three different gates, usually called "forget gate", "input gate" and "output gate". The gates are based on sigmoid activation functions, which outputs corresponding vectors F_t, I_t, O_t, containing values between one and zero that determines how much of each component to let through the gate. Figure 2.3 shows an overview of the LSTM architecture, with the gates marked as σ.

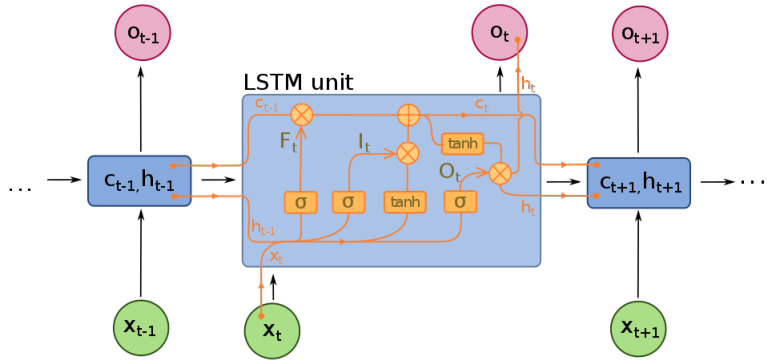


Figure 2.3: Unfolded view of the LSTM unit architecture. Figure: [3] CC BY-SA 4.0

The vectors F_t, I_t, O_t are computed as

$$\begin{aligned} F_t &= \sigma(W_f \cdot [h_{t-1} \quad x_t] + b_f) \\ I_t &= \sigma(W_i \cdot [h_{t-1} \quad x_t] + b_i) \\ O_t &= \sigma(W_o \cdot [h_{t-1} \quad x_t] + b_o) \end{aligned} \tag{2.20}$$

where W_f, W_i, W_o are the weights and b_f, b_i, b_o are the biases corresponding to the different gates. F_t is responsible for what parts of the cell state to keep and what parts to forget. I_t controls which parts of the new information that should be added to the cell state. Together, F_t and I_t control the cell state content as

$$C_t = F_t \cdot C_{t-1} + I_t \cdot \tanh\left(W_c \cdot [h_{t-1} \quad x_t] + b_c\right) \tag{2.21}$$

where W_c and b_c are the weights and bias corresponding to the cell state. Lastly, O_t controls what to include in the hidden state h_t as

$$h_t = O_t \cdot \tanh(C_t) \tag{2.22}$$

which is also the output of the LSTM unit. The LSTM is trained using BPTT just like the RNN, but thanks to the gated memory (cell state), vanishing or exploding gradients are avoided [41].

The complexity of the LSTM unit leads to a large number of parameters for bigger and deeper architectures. The number of parameters p in the LSTM-layer grows with the cell state size γ according to

$$p = 4(n \cdot \gamma + \gamma^2 + \gamma) \tag{2.23}$$

where n is the number of input features. Hence, to add depth and increased capacity to a LSTM based model, it is common to combine it with regular MLP layers as done in [42] and [13]. The LSTM is then used as a tool to transform the temporal information into features that can be interpreted and used by the MLP, in order to perform the regression or classification task at hand [43]. To conclude, tuning a LSTM model is mainly about choosing the appropriate size of the cell state c_t and how much of the sequential data history to include in the input. In case the LSTM is combined with a MLP, the performance is of course heavily dependent on the MLP tuning as well.

2.5 Kalman Filtering

With the machine learning theory reviewed, this section focuses on the fundamentals of the Kalman filter. As briefly described in section 1.1, the Kalman filter is a Bayesian fusion algorithm, that fuses a model generated prediction with observed measurements, to form an estimate $\hat{S}_t$ of the actual system state vector S_t. Depending on the quality of the predictive system model and the noise level in the measurements, it is possible to tune the filter such that one of these two sources is more influential on the resulting estimation than the other. This is done via the process- and measurement noise, Q and R, where the most influence is given to the source with the lowest noise level. The key assumption in Kalman filtering is that both of these noises are Gaussian, which is why its ability to reject non-Gaussian disturbances is limited.

The prediction generation and the fuse with observed measurements are performed in two different steps, commonly called the prediction step and the update step. In the prediction step, $\hat{S}_{t|t-1}$ is computed based on the previous estimate $\hat{S}_{t-1|t-1}$, via the dynamical model

$$\hat{S}_{t|t-1} = D(S_{t-1|t-1}, u_{t-1}) = A\hat{S}_{t-1|t-1} + Bu_{t-1} \tag{2.24}$$

where A, B are parameters of the model and u_{t-1} is the system input at time $t-1$. The corresponding error covariance, i.e. the uncertainty of the prediction, is estimated as

$$P_{t|t-1} = AP_{t-1|t-1}A^T + Q \tag{2.25}$$

where $P_{t-1|t-1}$ is the estimated error covariance of $S_{t-1|t-1}$.

In the update step, the prediction $\hat{S}_{t|t-1}$ is updated or corrected, based on the observed measurements at time t. This process is based on the measurement model $M(S_t)$, which maps the state vector S_t to corresponding measurements z_t, as

$$z_t = M(S_t) = HS_t + R \tag{2.26}$$

where H is a parameter of the measurement model. The update step is initiated by computing the Kalman gain K_t as

$$K_t = P_{t|t-1}H^T(HP_{t|t-1}H^t + R)^{-1}, \tag{2.27}$$

which determines to what degree the prediction shall be corrected based on the observed measurement at time t. The estimation, or the update of the prediction, is then computed as

$$\hat{S}_{t|t} = \hat{S}_{t|t-1} + K_t(z_t - H\hat{S}_{t|t-1}). \tag{2.28}$$

The error covariance of the estimation is also updated as

$$P_{t|t} = P_{t|t-1} - K_tHP_{t|t-1} \tag{2.29}$$

before the entire process is repeated, starting with the prediction step. How to initiate the algorithm with a starting estimation and error covariance, depends on whether the initial system state is known or not. More about this matter and how to deal with non-linear models is described in [8].

2.6 Road Geometry Fusion

The idea of road geometry estimation is to estimate the shape of the road, using observations from different sensors together with a mathematical model of the road. The sensors used for observations usually include cameras, radars and lidars and the observations are commonly processed through a perception algorithm to deliver more useful information such as the location of lane markers and surrounding objects. The concept of fusing such observations with a mathematical model of the road is presented in [44], where a Bayesian fusion framework is used, and in [45] where an extended Kalman filter is used. The notion of Road Geometry Fusion (RGF) will in this thesis refer to the conceptual system shown in figure 2.4, where observations are fused together with a mathematical road model in a Kalman filter. The reference system used in this thesis is based on a RGF system of this type, where the decision logic can be implemented using either hand-crafted heuristics or a machine learning algorithm.

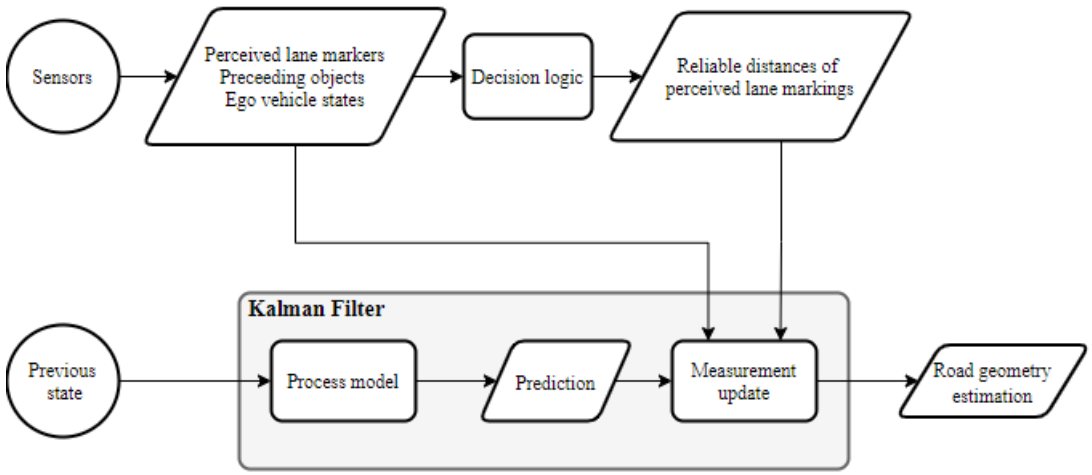


Figure 2.4: Conceptual overview of the RGF system used in this thesis. The decision logic box is the hand-crafted heuristics rejecting non-Gaussian disturbances.

3
Methods

This chapter begins with an overview of the concept and how the machine learning algorithms will be used in the RGF reference system. Followed by this are descriptions of the different methods used to train and develop the machine learning models, as well as evaluating their performance. This includes information about the dataset used in the thesis and the proposed method for annotating the data.

3.1 Concept Overview

As explained in section 1.1, there is a need to reject non-Gaussian disturbances in the Road Geometry Fusion (RGF) system, since the Kalman filter is incapable of this. One way to do this is to use hand-crafted heuristics, implemented as "decision logic" in the RGF reference system, presented in section 2.6. The heuristics analyze the incoming lane marker measurements and the findings are then used to process the perceived lane markers, before passing them to the measurement update of the Kalman filter. A simplified piece of such logic for detecting and handling the exit in figure 1.1, could be

Algorithm 1: Example of simplified heuristics

Result: Boolean indicating if an exit is upcoming on the right side
deltaHeading = leftMarkerHeading - rightMarkerHeading;
if *laneWidth60mAhead > laneWidth50mAhead AND deltaHeading < 0* **then**
    upcomingExitOnRightSide = true;
    followLeftMarkerOnly = true;
else
    upcomingExitOnRightSide = false;
    followLeftMarkerOnly = false;
end

This code utilizes information about the width of the lane and the heading of the lane markers. If the width increases and the right lane marker deviates from the left one, towards the right side, it is likely that there is an exit coming up. This is the underlying logic in this simplified example, but of course, it is not robust enough to work in a real-world application. The logic used in real-world applications are far more advanced and intelligent, and can handle far more complicated scenarios with high precision. However, such systems have the drawback of requiring a lot of work, both to develop in the first place but also to maintain and continuously improve.

The idea of the new conceptual RGF system is to truncate the lane markers in

the case of larger non-Gaussian disturbances. The distances at which to truncate are referred to as the "reliable distances" of the left and right marker, which are estimated using a machine learning algorithm that replaces the heuristics. After truncation, the remaining parts are to be free of major non-Gaussian disturbances, such that they can be passed to the Kalman filter, without violating its Gaussian assumption. Any remaining deviations should be small enough to be adequately approximated as Gaussian. An overview of the conceptual system, with the machine learning subsystem marked in red, is shown in figure 3.1.

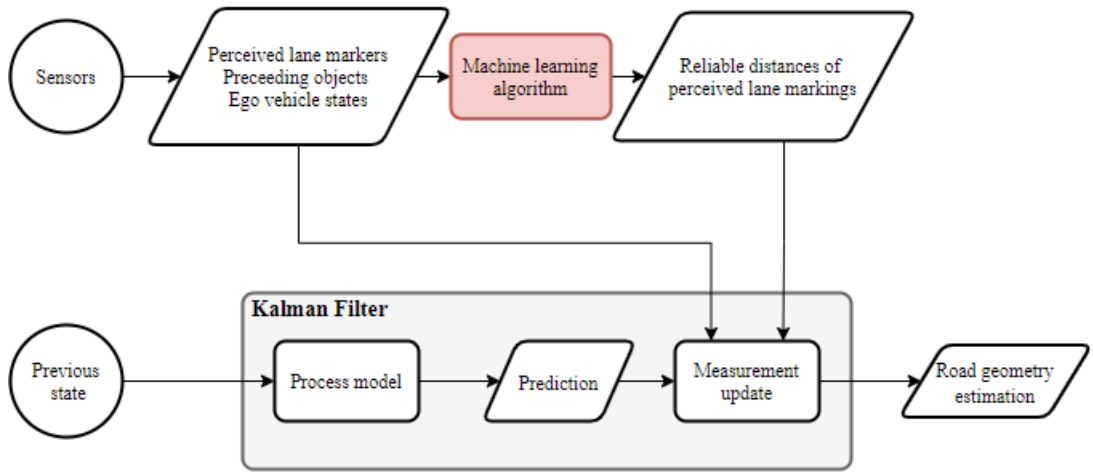


Figure 3.1: An overview of the conceptual RGF system, with the machine learning subsystem marked in red.

The first step in developing and implementing this system was to extract a dataset from provided driver logs. Furthermore, corresponding labels had to be created to enable supervised learning for the machine learning algorithms. Different algorithms were then trained and tested, of which the best was implemented and improved for a final concept. The final conceptual system was then evaluated against a reference system with hand-coded heuristics for the lane marker logic. The reference system used with an existing heuristic non-Gaussian disturbance rejection is robust enough to handle complex driving scenarios with high precision, making it a good measure to evaluate the conceptual system against. All of these steps will now be covered in detail throughout the rest of this chapter.

3.2 Dataset

The dataset used in this thesis consists of real-life data from 220 driving hours, gathered from multiple different vehicles in different operational domains, with ground truth of the coordinates for the ego vehicle's lane boundaries. The data contains features that are common in most widely used datasets in the autonomous driving domain, such as object detections, lane markers and road segmentation [46][47].

The dataset also includes signals describing the host vehicle's pose, velocity and acceleration, and was filtered to only contain drives with a mean velocity above 16 meters per second in order to remove scenarios from low-speed areas, such as suburbs or city driving, where these functions are not intended to be used. To fit the scope of the thesis, as stated in section 1.2, some drives were also removed based on weather or road types. For instance, scenarios with snow or sleet on the road were removed as they were considered to be outside of the operational domain of the proposed concept in this thesis. Scenarios classified as residential areas, living streets or roundabouts were also removed, with the same motivation. The dataset includes scenarios with clear, cloudy and rainy weather as well as both nightly and daily drives. The dataset was also sampled based on weather and time to obtain a more balanced representation of the different conditions.

3.2.1 Downsampling

The data collected from real-life driving contains multiple samples per second, which results in a lot of similar samples, as the state of the car and the road does not change that frequently. Training machine learning models on a lot of similar samples is unlikely to improve the performance of the model very much, but still has the associated cost of extra training time for those samples. If similar samples are removed, that reduces the number of samples needed to train on, thus increasing the computational efficiency. The dataset was therefore downsampled to 4 Hz, resulting in one sample every 0.25 seconds. The benefit in computational efficiency from this allowed data from more driving segments to be added, increasing the variance covered by the dataset. The machine learning models can then be trained on a more diverse dataset, in the same amount of training time as the original sampling, which will most likely lead to models that can generalize better to new unseen data.

3.2.2 Training and test split

The dataset was split into two parts, a training set and a validation set. The training set consists of 80% of the data and was used for fitting the models. The validation set consists of the remaining 20% and was used to evaluate the models on unseen data during the training phase. In addition to this, a test set consisting of 9.5 driving hours was used for the final evaluation of the models. This set was never used during the iterative phase of trying different models. Both splits were made on sequence level, such that frames of the same sequence did not appear in more than one of the sets.

3.2.3 Standardization

Standardization of datasets is common practice in machine learning, as many models may perform considerably worse if the data has individual features that are far from Gaussian distributed. Some learning algorithms even assumes that all features in the dataset have zero mean and same order of variance, such as the RBF kernel of Support Vector Machines. The features in the training set were thus scaled to have zero mean and unit variance. The exact same transformation was then performed

on the features in the test set, such that they were preprocessed without any prior knowledge of the feature distribution.

3.3 Features

The features used in this project come from information provided by the onboard sensors of the host vehicle. This section presents how the features were further engineered and selected for optimizing the capability of the machine learning algorithms.

3.3.1 Feature Engineering

One of the most important fields of information that the sensors provide is the lane marker information of the ”ego lane”, which is the lane that the host vehicle is currently following. The lane marker information contains the mathematical models

$$y_{left} = f_{left}(x) \quad and \quad y_{right} = f_{right}(x), \tag{3.1}$$

which describe the left and right marker, in the local Cartesian coordinate system of the host vehicle. The coordinate system is defined in figure 3.2, which also illustrates the mathematical representations of the perceived lane markers.

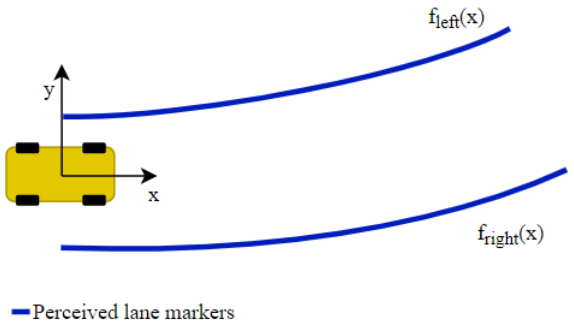


Figure 3.2: Definition of the host vehicle coordinate system, along with an illustration of the mathematical lane marker representations.

To make the lane marker information disposable to the machine learning algorithms, $y_{left} = f_{left}(x)$ and $y_{right} = f_{right}(x)$ were sampled for 17 different values of x, which are specified in table 3.1. The sample points were chosen to achieve a good resolution close to the vehicle, while still including distances up to 150 meters. A lower resolution was chosen for the distances further away to reduce the total number of sample points, and thus the resulting number of features. For clarification, this yielded two features for each value of x, one for the left side and one for the right.

From the sampled lane markers, the lane width L_w corresponding to the values of x was calculated as

$$L_w(x) = f_{left}(x) - f_{right}(x). \tag{3.2}$$

Table 3.1: The distances (values of x) in front of the host vehicle, at which the mathematical representations of the lane markers are sampled.

Sample Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Distance[m]	5	10	15	20	25	30	35	40	45	50	60	70	80	90	100	125	150

Lastly, the heading ϕ between each sample of each lane marker, was calculated as

$$\begin{aligned} \phi_{left,i} &= \arctan\left(\frac{y_{left,i} - y_{left,i+1}}{x_{i+1} - x_i}\right) \quad for\ i = 1, 2, .., 16 \\ \phi_{right,i} &= \arctan\left(\frac{y_{right,i} - y_{right,i+1}}{x_{i+1} - x_i}\right) \quad for\ i = 1, 2, .., 16. \end{aligned} \tag{3.3}$$

These were then used to calculate the difference in heading between the left and right lane marker $\Delta_{\phi,i}$, as

$$\Delta_{\phi,i} = \phi_{left,i} - \phi_{right,i}. \tag{3.4}$$

These additional extractions of the lane markers were included with the other pieces of information available in a set of feature candidates. How the final content of the dataset was chosen is accounted for in the next subsection.

3.3.2 Feature Selection

As mentioned, one source of information was the lane marker data of the ego lane, along with the additional extractions discussed in the previous subsection. Similar information fields were also available for road edges and adjacent lanes. Beyond these three fields of information, the ego vehicle states, such as velocity and acceleration, were also available for usage.

To evaluate which parts of information the algorithms could benefit from, a RF regressor was trained and tested with different subsets of the available features. It is important to make clear that a different model might give a different result, as the algorithms respond differently to changes in the feature space. However, the process of evaluating different features on all the algorithms would be far too time-consuming. Anyhow, the RF regressor was chosen as it is a relatively fast algorithm to train, thus allowing for an effective iteration of different features. The regressor consisted of 100 estimators or trees, where the minimum number of samples required for each split and leaf was set to 10. To obtain a baseline for comparison, a simple set was first created, only including the ego vehicle states and the most basic ego lane information. Relative to the results from this set, the yielded decrease in RMSE was computed for the different feature sets. The resulting RMSE decrease is presented together with a short description of the corresponding feature set in table 3.2.

These results indicate that the information from road edges and adjacent lanes is hardly usable since the resulting decrease in RMSE is significantly smaller than for the features related to the ego lane. Thus, these sources of information are not included in the feature space used in this thesis. Instead, the final feature space consists of ego vehicle states and ego lane information, including sampled lane markers, lane width and heading difference. In total, these add up to $n = 85$ features.

Table 3.2: The mean improvement in RMSE of the left and right side, for different additional features, compared to the basic set 1.

Set	Description	RMSE decrease[%]
1	Ego vehicle states and basic ego lane information	0 (reference)
2	Set 1 + basic info from road edges and adjacent lanes	0.56
3	Set 1 + sampled ego lane	8.43
4	Set 1 + sampled lane width	8.58
5	Set 1 + sampled ego lane and lane width	10.37
6	Set 1 + additional ego lane information	3.06
7	Set 1 + sampled road edge	0.97
8	Set 1 + sampled heading difference	8.84
9	Set 1 + Adjacent lanes sampled	0.46

3.4 Automatic Annotation

As elaborated on in section 2.1, the supervised regression algorithms require mapping examples from input to output. That means that the training data inputs must be labeled with the correct or desired output. In this particular case, it means that each set of input information must have two corresponding labels, one for the reliable length of the left lane marker and another one for the right. With m training examples containing n features, the resulting data set obtains the form shown in figure 3.3. Note that this is just a clarifying illustration and does not represent the actual distribution of features within different fields of information.

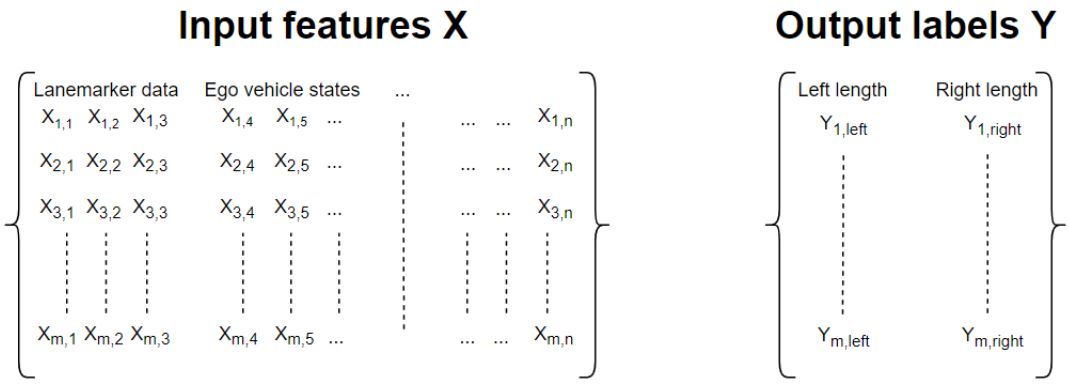


Figure 3.3: The form of the training data set. Note that this illustrative description does not represent the actual distribution of features within different fields of information in the dataset.

As previously mentioned, the annotations should mark the reliable distance of the incoming lane markers from the camera, i.e. how far along the host axis x that they are free from any larger non-Gaussian disturbances and are following the lane of interest. However, these did not exist in the dataset at hand and nor are there any established methods for creating such labels. Hence, a method for determining the reliable distances of the lane markers is proposed in the upcoming subsection.

3.4.1 Determining the reliable distance of lane markers

From logged drivings, the reliable lane marker distance can be determined for each time step by comparing the lane markers reported by the sensors, to the collected Ground Truth Assessment (GTA). The GTA contains the true coordinates for the lane markers, which can be used to calculate the Euclidean error e of the perceived markers. The distance where e exceeds a certain threshold $T(x)$, is then the distance up to which the lane marker is free from major disturbances and can be trusted. Since the immediate control of the host is mainly dependent on its proximity, the demand on precisely perceived lane markers is increased closer to the vehicle. Hence, it is suitable to let the threshold $T(x)$ increase linearly with x, as

$$T(x) = ax + b. \tag{3.5}$$

What values of a, b that are appropriate depends on the desired precision of the perceived lane markers, which in turn is dependent on the application. For some ADAS applications it is necessary to assign preceding vehicles to the correct lane, which requires that the estimation error of the lane never exceeds half the lane width L_w. Since the lane geometry is estimated up to 200m ahead, it is therefore suitable to enforce $T(200) < \frac{L_w}{2}$ [44]. Furthermore, a fitting value for the minimum threshold $T(0)$, is 0.3m according to [48]. From [49] it is reasonable to assume that $L_w > 2.6$m, meaning that both conditions can be fulfilled by letting $a = 0.005$ and $b = 0.3$. The comparison between perceived lane markers and GTA is shown in figure 3.4 together with their assessed errors, but note that this is only a clarifying illustration that does not represent the true precision of the perceived lane markers in any way.

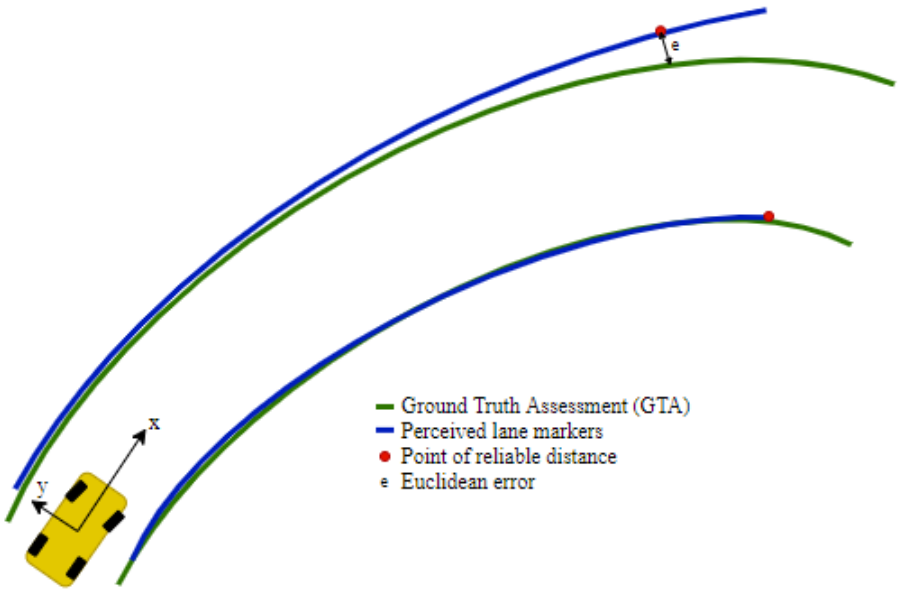


Figure 3.4: Illustration of how the perceived lane markers are compared to the GTA in order to find their reliable distance, based on the Euclidean error e. Note that this illustration does not represent the true precision of the perceived lane markers in any way.

3.5 Algorithms

To find the most suitable algorithm for replacing the heuristic subsystem, as explained in section 3.1, different candidates have been evaluated and compared. The algorithms that have been tested are Support Vector Regression (SVR), Random Forest (RF), Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM). These were chosen based on the thorough comparison in [15], as well as what has been used in similar work and research fields.

LSTM has for instance proven itself successful in [13] and [14], which are ADAS related regression tasks. With the sequential nature of the data, it was also appropriate to include a recurrent algorithm. The comparison in [15] does not include LSTM, but instead it highlights the high performance of SVR, RF and MLP, specifically on regression tasks. SVR is a powerful algorithm and has been fruitful in time series analysis before, for instance in [50]. It is also quickly applicable which made it a good algorithm to start with. Regarding RF, it has been successfully used in [51] and [52]. Although these implementations concern decision making and not regression, they are examples of RF performing well within the field of ADAS, which made it an interesting candidate for this implementation. The last algorithm included in the comparison was MLP, which is a general and flexible algorithm that is used in the core of many deep learning architectures [24]. In recent years, it has been implemented in many different applications with good results, including neural filtering [16]. Even though MLP and LSTM can be considered similar in some aspects, these four algorithms are otherwise different in terms of both architecture and functionality. The broad spectrum of algorithms increases the overall coverage of this thesis, which is another motivation to the choice of these particular candidates.

All algorithmic models have been created and trained in Python, mainly because of the available open-source libraries for machine learning and artificial intelligence. The LSTM models were implemented using Tensorflow and Keras, which are popular deep learning tools [53]. The rest of the models were implemented using the API from Scikit-learn, which is a commonly used Python library for scientific machine learning [54]. All four algorithms were explored for different sets of hyperparameters, feature spaces and sizes of the training set, before being compared and evaluated in the system explained in section 3.1. For clarification, the LSTM implementations process time-series sequences, where each time step contains all model inputs. Furthermore, this means that the complete lane marker measurements, from the host car to the last sample, are included in each step.

3.6 Learning Curves

Although a large dataset is beneficial for the final training of the algorithms, it is not feasible to use for exploring different parameters and architectures due to the increase in training time that comes from having more data. Instead, only a subset of the whole dataset was used to be able to efficiently iterate the development process. To find a good trade-off between the size of the training set and the training time, learning curves were used. Learning curves are a great tool for analyzing how algorithms respond to increased amounts of data, as well as show the trade-off between bias and variance. The learning curves were generated by training each algorithm for different training set sizes, with three split cross-validation for each size. The scores and training times were then computed as the mean from the splits at each training size, with RMSE used as score function. The learning curves are presented as semi-log plots in figures 3.5-3.7, where the x-axis is logarithmic and the y-axis is linear.

The learning curves for SVR were generated with $C = 1$ and $\epsilon = 0.1$, for training sizes of 1, 10, 10^2, 10^3, 10^4 and 10^5 samples, and can be seen plotted in figure 3.5. From the plots, it can be observed that the validation and training curves start to converge already for 10^3 samples, and continues to decrease when adding more data. However, as can be seen from the increase in training time, the SVR does not scale very well for training sets larger than a couple of 10^4 samples.

For the RF algorithm, learning curves were generated with 100 trees and only $\sqrt{n}$ features available for each tree, where n is the total amount of features. The learning curves were generated for 1, 10, 10^2, 10^3, 10^4, 10^5 and 10^6 samples, and can be seen plotted in figure 3.6. The validation and training errors decrease steadily for increased training data, although with a large gap between the curves. The algorithm suffers from high variance and low bias, which is likely caused by overfitting the training data. This is in turn a result of the trees' abilities to split branches and create leaves for each training sample. There are several parameters that can be tuned in RF to reduce the variance. For instance, the size of the random subset of features available for each tree can be lowered, with the trade-off of higher bias. Adding more data would most likely also help the model generalize better, although with the cost of increased training time.

The learning curves for the MLP were generated with three hidden layers of sizes (16, 16, 8), with ReLU activation between every layer. The model was trained for 200 epochs, with a batch size of 32, using Adam as solver with an initial learning rate of 0.001. The learning curves were generated for 1, 10, 10^2, 10^3, 10^4, 10^5 and 10^6 samples and can be seen in figure 3.7. The training and validation errors start to converge for larger training sets and results in a model with very low variance. This specific model would most likely not benefit from more training instances, but could instead be improved by increasing the complexity, via an introduction of more layers or more neurons in the existing layers. As for the RF model, the training time increases a lot going from 10^5 to 10^6 training samples.

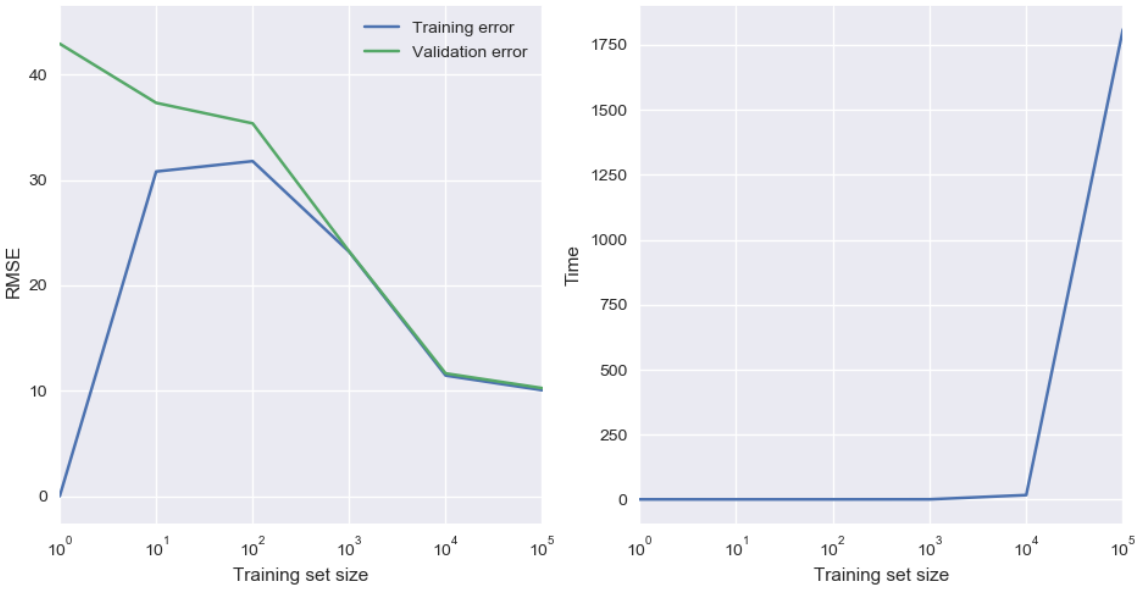


Figure 3.5: Learning curves for a SVR algorithm trained using $C = 1$ and $\epsilon = 0.1$, for training sizes of 1, 10, 10^2, 10^3, 10^4, and 10^5 samples. The left plot shows the training and validation RMSE versus the training size while the right plot shows the corresponding training time.

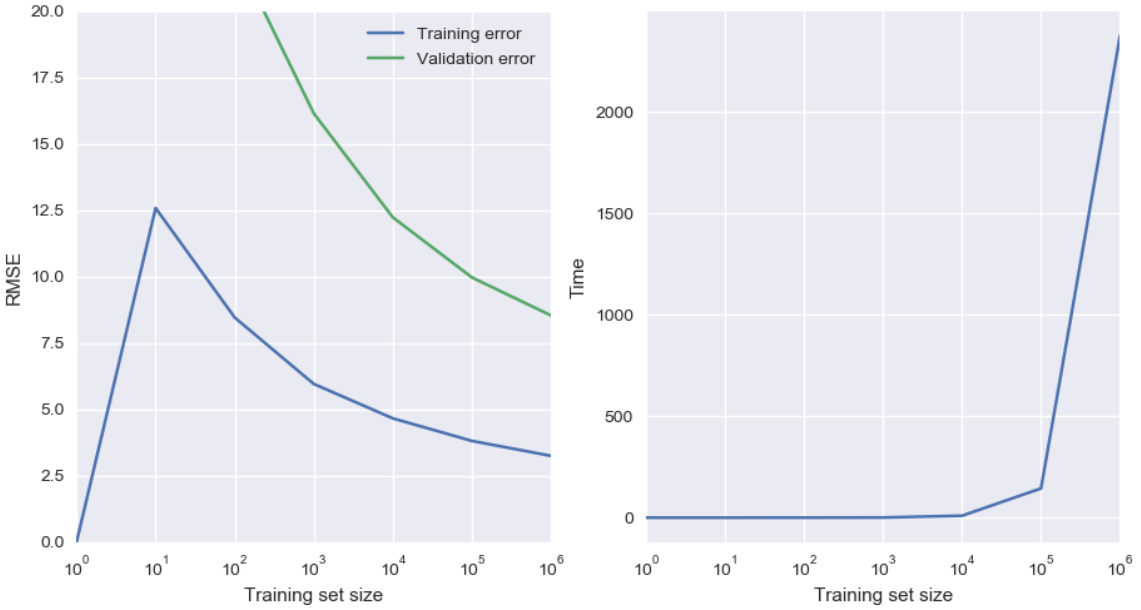


Figure 3.6: Learning curves for a RF algorithm trained using 100 trees and $\sqrt{n}$ features available for each tree, where n is the total number of features. The evaluated training set sizes are 1, 10, 10^2, 10^3, 10^4, 10^5, and 10^6 samples. The left plot shows the training and validation RMSE versus the training size, while the right plot shows the corresponding training time.

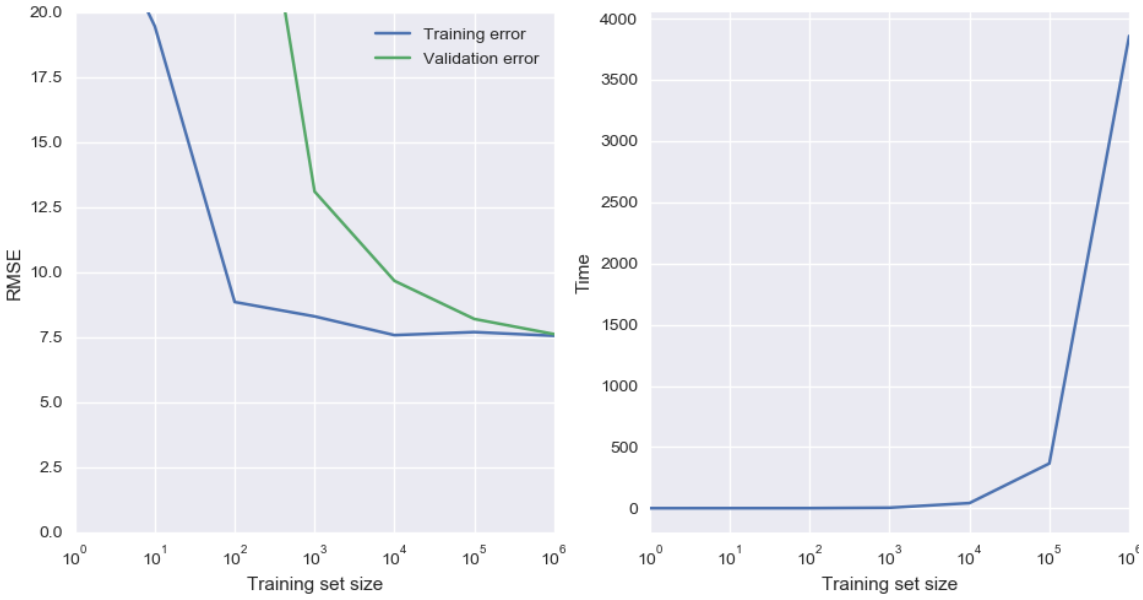


Figure 3.7: Learning curves for a MLP algorithm trained for 200 epochs using three hidden layers with 16, 16 and 8 neurons respectively, activated using ReLU. The model was trained with Adam as optimizer, for training sizes of 1, 10, 10^2, 10^3, 10^4, 10^5, and 10^6 samples. The left plot shows the training and validation RMSE versus the training size while the right plot shows the corresponding training time.

Comparing the learning curves from the three different algorithms it is obvious that the SVR is not very suitable for this application. Although it converges fast for a small set of data, it does not fit to the larger datasets as well as the other algorithms and scales badly in terms of training time. Many of the kernel methods used for Support Vector Machines are composed as quadratic programming problems, having a time complexity of $\mathcal{O}(m^3)$ where m is the number of training samples, and are thus infeasible for larger datasets. There exist a couple of techniques for kernel approximations such as the Nyström method proposed in [55] or greedy approximations [56], but they have other drawbacks and may still be infeasible for the large amount of data available in this thesis. The SVR was therefore deemed to not be suitable for this application. No learning curves are included for LSTM since it is used in combination with several fully connected layers and thus inherit the response to increased data amounts from the MLP.

3.7 Parameter Search

The result of the learning processes for the different machine learning algorithms is highly dependent on the chosen hyperparameters. Choosing these parameters is a crucial step in training an algorithm, but can often be a very tedious and time-consuming task. Several methods have been developed for this purpose, such as grid search over a set of parameters or Bayesian optimization algorithms that builds a probabilistic model over the mapping from hyperparameters to the objective value. However, constructing such methods for multiple types of algorithms, using different libraries for implementation, is nonetheless very time consuming and is no guarantee

for optimal parameters in the end. Instead, some simpler parameter searches were first performed to identify suitable ranges for the parameters and better understand their effect on the model.

3.8 Evaluating the Models

To find the most suitable combination of algorithm and setup, several versions of each model were trained and evaluated, with different architectures and sets of parameters. New versions were created iteratively, based on the performance of previous ones and parameter search results. Since the models were trained on an annotated dataset, the fairest performance measure is how close their regressions are to these annotations on unseen data. Even though they are intended to enhance the Road Geometry Fusion (RGF), this is not the task that they have been optimized to solve. Thus, the different versions of each model were first compared in terms of RMSE of their regressions on the validation set.

Based on the validation set results, one version of each algorithm was chosen for implementation and testing in the RGF reference system, as explained in section 3.1. The RGF returns an estimation of the road ahead of the host vehicle, of which the quality is in retrospect determined by two different main aspects. Those are the lateral error of the estimation and its availability, which is the length of the estimation. For optimal performance of the functions that utilize the road geometry estimation, it should have high availability while maintaining a low lateral error, but this is a trade-off. Keeping a low lateral error is easily done by only giving short and conservative estimations, but consequently, the availability is then decreased and vice versa.

The RGF testing was done by simulating usage of the reference system with the implemented machine learning algorithms, on the test set of driving logs described in subsection 3.2.2. To achieve a baseline for comparison, the reference system was also simulated with heuristics for the rejection of non-Gaussian disturbances. The availability was measured at each time step and then summarized as the mean value. Similarly, the lateral error was measured continuously at the distances presented in table 3.3, but then summarized as the standard deviation of the error at each distance. For the measurement distances longer than the road geometry estimation at a given time step, the errors were set to zero. Lastly, for samples with missing or invalid ground truth assessment, the calculations of the lateral error and availability were simply skipped.

Table 3.3: The distances (values of x) in front of the host vehicle, at which the lateral error is measured during evaluation of the RGF performance.

Measure index	1	2	3	4	5	6	7	8	9
Measure distance [m]	0	10	20	30	40	50	100	150	The estimation length

4
Results

This chapter presents the results of the parameter searches, as well as performance evaluations of the machine learning models and the proposed function for automatic annotation. As mentioned in section 3.6, the SVR training complexity was found to scale badly to increased training set size, which made it unsuitable to use in this thesis. Hence, there are no presented results for SVR.

4.1 Evaluation of the Parameter Search

Several parameter searches were performed on the algorithms to find suitable ranges and get a better understanding of the effect of tuning different settings. Each search only contains changes in one parameter, to isolate the effect from the changes applied. All searches were performed on a small subset of the data, around 10^5 samples, with three split cross-validation, and were evaluated both on the training and the validation data with RMSE as the scoring function. Recorded scores and fit times were then averaged over the cross-validation splits.

4.1.1 Random Forest

The RF algorithm was evaluated for different ranges of maximum depth, maximum number of features available for each tree, minimum number of samples per leaf, minimum samples needed for split, and number of trees in the ensemble. The result of each search is presented as plots in figures 4.1-4.5.

The parameter search over the maximum depth allowed was done for parameter values of 1, 10, 10^2, 10^3 and 10^4. As seen in figure 4.1, the results from the search shows that a RF with deeper trees in the ensemble, manages to fit much better to the data, resulting in a lower RMSE. As observed, the fit time increases with the depth in the trees. At a maximum depth of 100, the trees seem to be able to expand such that all leaves are pure, indicated by the plateau in training score and fit time. However, more data will likely require a larger maximum depth to reach the same point.

The search over the maximum number of features allowed was done for parameter values of 0.1, 0.4, 0.7, and 1.0, where the value represents the portion of the total number of features available. The results are illustrated in figure 4.2, and show that the RMSE decreases for an increasing parameter value up until 0.4. A higher portion of features allowed seem to increase the variance in the algorithm, resulting in a higher RMSE on the validation set. The fit time increases linearly with the

percentage of features allowed.

The parameter search over the minimum number of samples per leaf was performed with values of 1, 10, 10^2 and 10^3. The results can be seen in figure 4.3, which indicates that a higher parameter value decreases the variance in the algorithm, but with the cost of greatly increased bias. The search over the minimum number of samples needed to split a branch was done with parameter values of 2, $2 \cdot 10^1$, $2 \cdot 10^2$, and $2 \cdot 10^3$. As observed in figure 4.4, the results from this search shows the same behavior as the parameter for the minimum number of samples per leaf, where the parameter value seems to be a trade-off between bias and variance. The fit time decreases with an increase in the parameter value for both parameters.

Lastly, the search over the number of trees used in the ensemble was performed using 1, 10, 10^2 and 10^3 trees. The results are presented in figure 4.5, and show that increasing the number of trees in the ensemble greatly enhances the algorithm's ability to fit the data, resulting in a lower RMSE both for the training and validation set, up until around 100 trees. A larger ensemble does not seem to give any further improvements to the score but increases the training time.

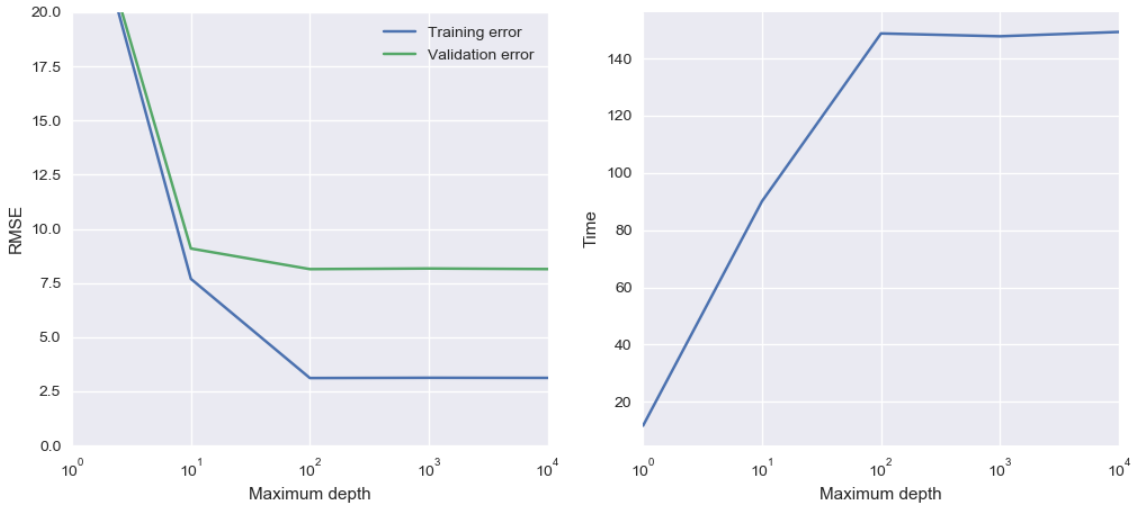


Figure 4.1: Plot of a RF algorithm trained using different values for the maximum depth allowed, with a training size of 10^5 samples. The left plot shows the training and validation RMSE versus the parameter value, while the right plot shows the corresponding training time.

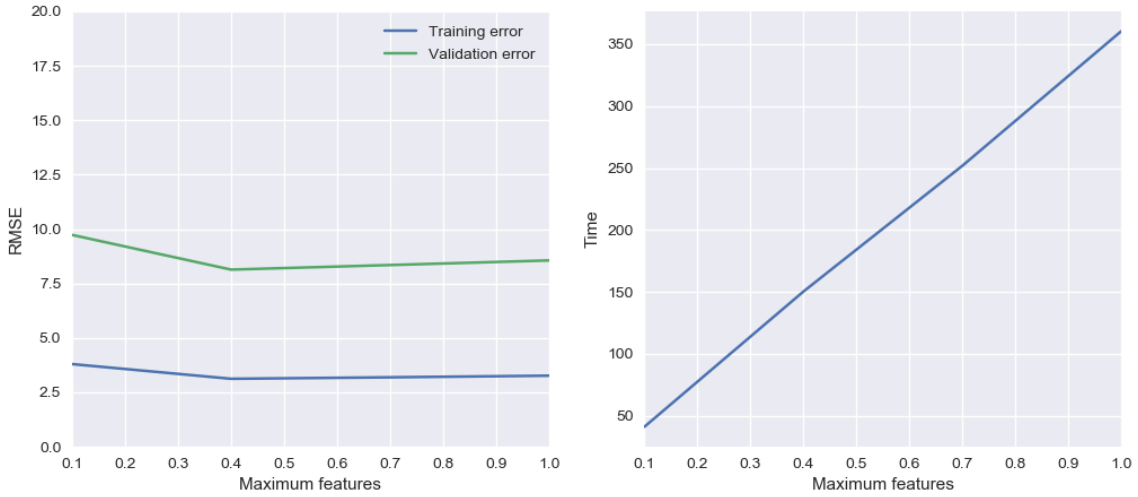


Figure 4.2: Plot of a RF algorithm trained using different values for the maximum number of features allowed for each tree, with a training size of 10^5 samples. The left plot show the training and validation RMSE versus the parameter value while the right plot shows the corresponding training time.

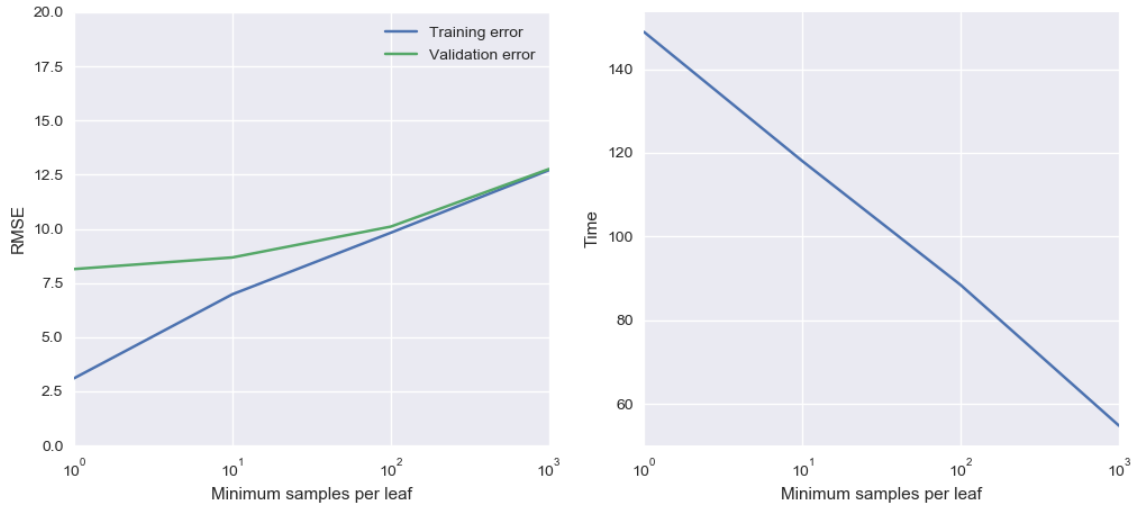


Figure 4.3: Plot of a RF algorithm trained using different values for the minimum number of samples per leaf, with a training size of 10^5 samples. The left plot shows the training and validation RMSE versus the parameter value while, the right plot shows the corresponding training time.

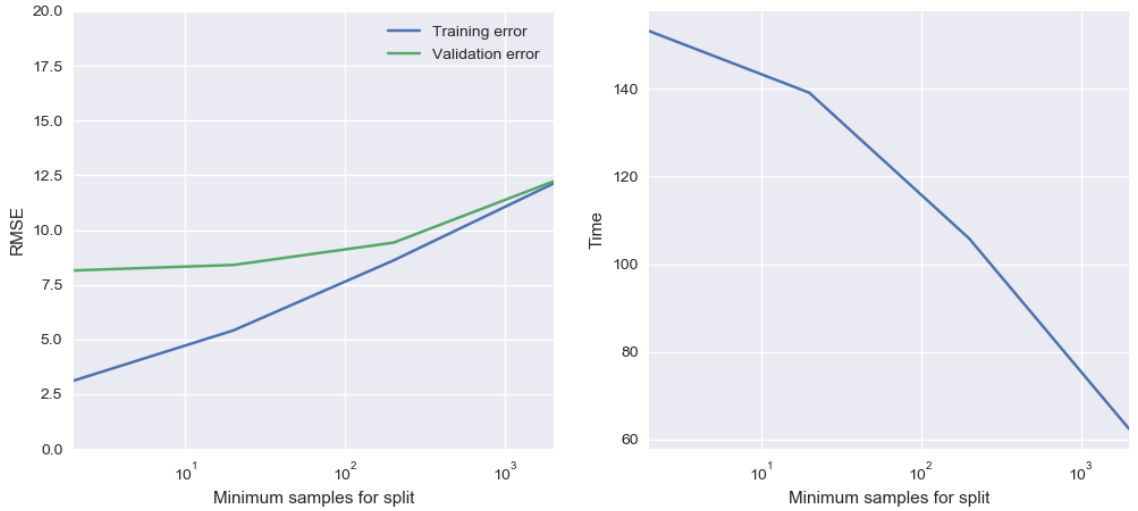


Figure 4.4: Plot of a RF algorithm trained using different values for the minimum number of samples needed to split a branch, with a training size of 10^5 samples. The left plot shows the training and validation RMSE versus the parameter value, while the right plot shows the corresponding training time.

Figure 4.5: Plot of a RF algorithm trained using different values for the number of trees in the ensemble, with a training size of 10^5 samples. The left plot shows the training and validation RMSE versus the parameter value, while the right plot shows the corresponding training time.

4.1.2 Multilayer Perceptron

The MLP model was evaluated for a different number of neurons, as well as several different batch sizes. The parameter searches were made with Adam as the optimizer, with an initial learning rate of 0.001, and ReLU as activation between the hidden layers. The different models were all trained for 10 epochs.

The result of the evaluations for the different number of neurons can be seen in 4.6, where a model with three hidden layers between the input and output layer was trained with 10, 40, 70 and 100 neurons, in each of the hidden layers. The results show that an increased depth in the network, i.e more neurons, increases the model's ability to fit the data, thus resulting in a lower RMSE for both the training and validation set. However, the increased complexity also seems to increase the variance in the model, resulting in an increased gap between the training and validation curves for a higher number of neurons. As observed, the training time increases with the added depth in the network.

The MLP model was also trained and evaluated for batch sizes of 1, 10, 10^2, 10^3 and 10^4. The results from the search over batch sizes can be seen in figure 4.7, and show that a larger batch size can decrease the computation time per iteration a lot. However, as stated in section 2.4.6, a higher batch size can also lead to a slower learning process which is exactly what can be observed here as well. For larger batch sizes than 10, the RMSE starts to increase again, indicating that the model has not converged as fast as for the versions with smaller batch sizes.

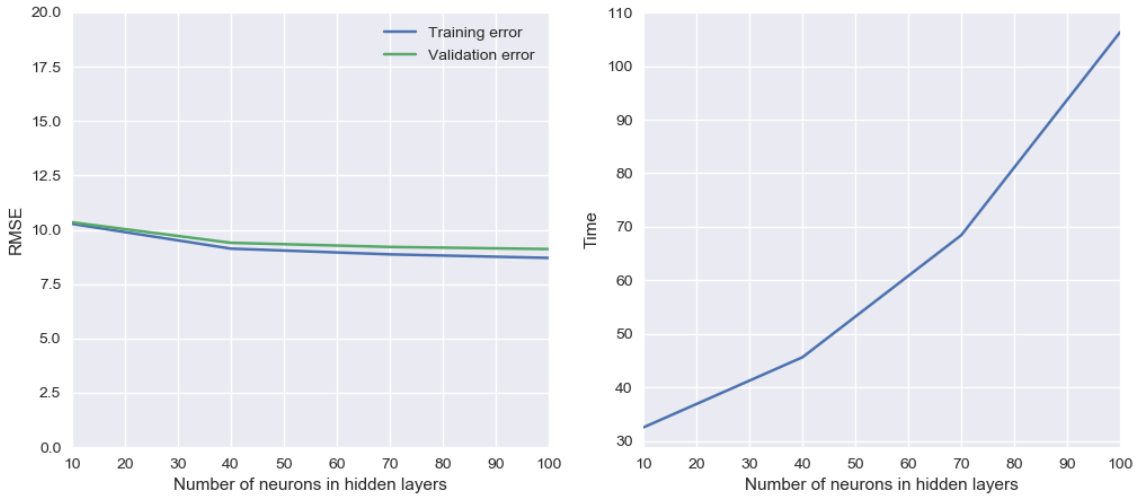


Figure 4.6: Plot of a MLP model trained using different values for the number of neurons in three hidden layers, with a training size of 10^5 samples. The left plot shows the training and validation RMSE versus the parameter value while, the right plot shows the corresponding training time.

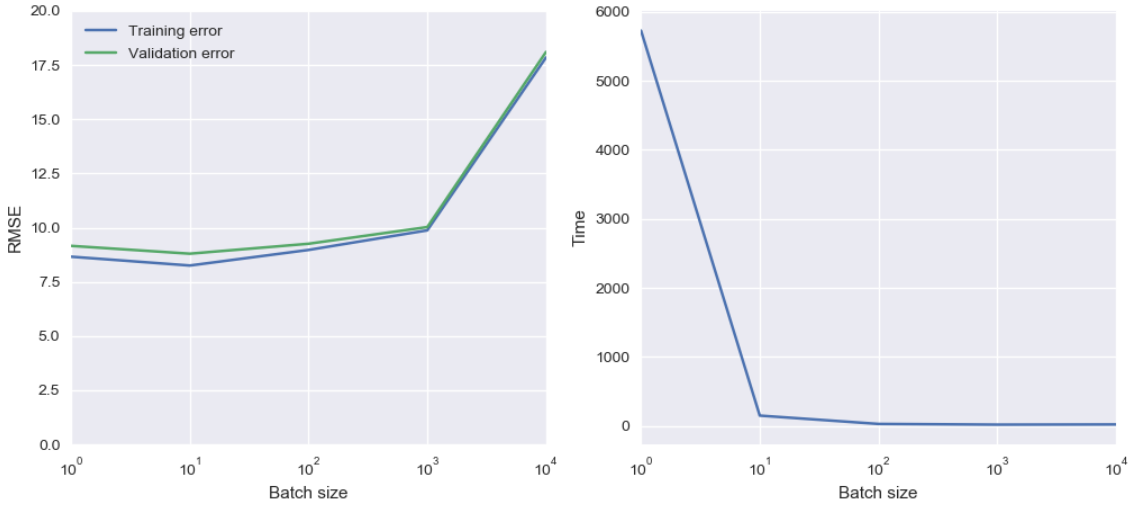


Figure 4.7: Plot of a MLP model trained using different values for the batch size, with a training size of 10^5 samples. The left plot shows the training and validation RMSE versus the parameter value, while the right plot shows the corresponding training time.

4.1.3 Long-Short Term Memory

The LSTM model was trained with a range of different numbers of LSTM-neurons, or cell state sizes. It was also trained using different batch sizes, but always using the same 10^5 sequences, of 5 time steps each. All the model versions were trained for 10 epochs and the LSTM-layer was always combined with three fully connected layers, consisting of 100, 50, and 10 neurons, before the final output layer. The fully connected layers all had ReLU activation functions, while the LSTM neurons were activated using tanh . The model was trained with Adam as the optimizer and MSE as the loss function.

The results from the search over different number of LSTM-neurons are presented in figure 4.8. The model was trained with 1, 25, 50, and n number of LSTM-neurons, where n is the total number of features in the dataset. A higher number of neurons seems to increase the model's ability to fit the data, resulting in a lower RMSE on the training set. However, the increased complexity seems to also increase the model variance, resulting in a larger gap between the RMSE-curves of the training and validation set. Further, it can also be concluded that the fit time increases with the depth in the LSTM-layer.

The LSTM model was, as the MLP, also trained for batch sizes of 1, 10, 10^2, 10^3, and 10^4. The results can be seen in figure 4.9, and follows the same general shape as the results from the MLP. Increasing the batch size from 1 to 10 seems to greatly reduce the training time without suffering from the downsides of a slower learning process. Further increasing the batch size to 100 samples per batch seems to have the benefit of additional reduction of the training time, with a slightly negative trend on the training error. An increasing batch size after that point does not reduce the training time much further, but harms the learning process, resulting in a higher RMSE for both the training and validation sets.

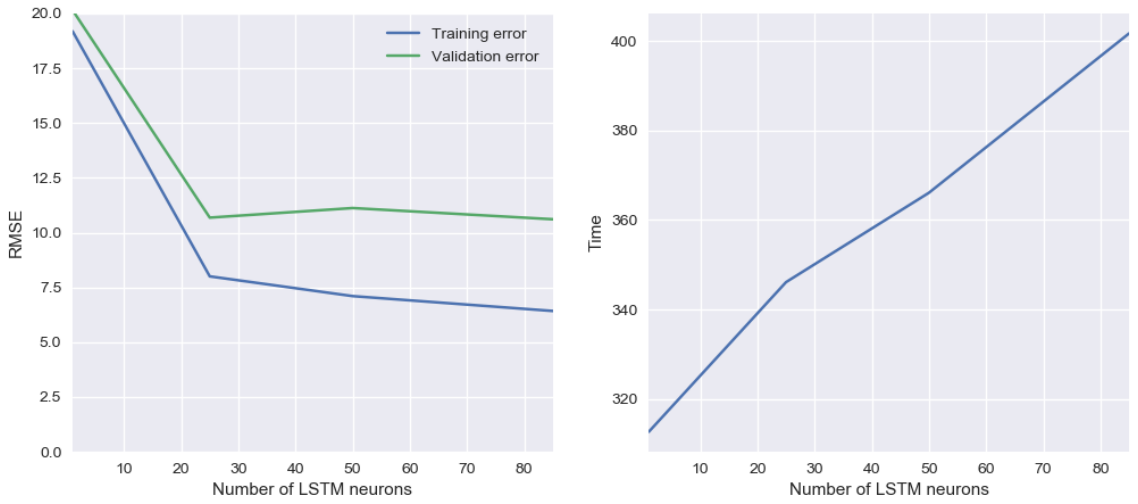


Figure 4.8: Plot of a LSTM model trained using different values for the number of LSTM-neurons, or length of the cell state. The model consisted of a single LSTM-layer connected to a MLP, and was trained using 10^5 sequences of 5 time steps each. The left plot shows the training and validation RMSE versus the parameter value, while the right plot shows the corresponding training time.

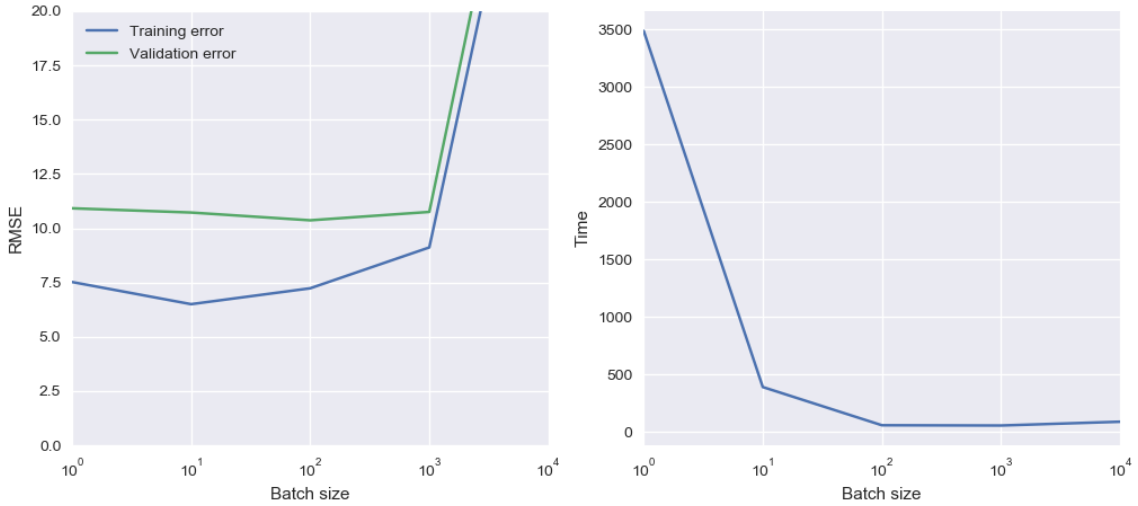


Figure 4.9: Plot of a LSTM model trained using different values for the batch size, with a training size of 10^5 samples. The left plot shows the training and validation RMSE versus the parameter value, while the right plot shows the corresponding training time.

4.2 Regression Results on Labels

An iterative process of training and validating models was performed for RF, MLP and LSTM, to compare the algorithms against one another, as well as to find the best candidate of every algorithm for further experiments. A variety of models were trained and evaluated against the annotations, using RMSE as the evaluation metric.

4.2.1 Implementation details

The full dataset was used, consisting of approximately 2.4 million data points, where 80 percent was used for fitting the models and the remaining 20 percent for validation. The RF models were all trained without any limit on the maximum depth allowed, and with 100 trees in the ensemble. Both the MLP models and the LSTM models were trained with ReLU as the activation function for all hidden layers, and a final output layer of 2 neurons without any activation. Adam was used as the optimizer for both algorithms, with an initial learning rate of 0.001. The activation in the LSTM-layers was set to tanh , and the LSTM models were trained with a 5 state memory. The default number of epochs in the training was set to 25 for the MLP and LSTM models. All models were trained with two outputs, corresponding to the reliable distance of the left respectively right lane marker.

4.2.2 Evaluation results

The models were evaluated on their ability to predict the reliable distance for both the left and right lane markers, using RMSE as the evaluation metric. The results are presented in table 4.1, where each model is presented together with its algorithm type, configuration and RMSE. The difference in RMSE between different models is quite small, where the best model overall has an average score of 7.53, and the worst model has an average of 7.89. The RF algorithm seems to have the smallest spread between different versions but is also the worst-performing algorithm of the three in terms of average RMSE. The best RF model is the third version, which has an average RMSE of 7.82. The MLP algorithm performed slightly better, where the worst performing version matches the best RF version in terms of average RMSE. The learning process for most of the MLP models stagnated around 20 to 25 epochs and would not benefit from further training. A few models, however, like the one shown in figure 4.10, showed a trend of continued learning and was therefore trained for more epochs. The best MLP version resulted in an average RMSE of 7.68. The LSTM models have the biggest spread between versions, where the best LSTM version also is the best model overall with an average RMSE of 7.53. The LSTM models showed no indications of increased learning after 25 epochs.

Table 4.1: Results from the evaluation of different model versions on the annotated data. The models are evaluated using RMSE as scoring function, where a smaller score is better. The best version of each algorithm is marked in bold.

Model		Configuration	RMSE		Avg.
			Left	Right	
RF	1	Max. features = 0.2, Min. samples for split = 5, Min. samples per leaf = 5	7.88	7.90	7.89
	2	Max. features = 0.4, Min. samples for split = 2, Min. samples per leaf = 1	7.83	7.84	7.84
	3	Max. features = 0.4, Min. samples for split = 5, Min. samples per leaf = 5	**7.81**	**7.82**	**7.82**
	4	Max. features = 0.4, Min. samples for split = 10, Min. samples per leaf = 10	7.85	7.86	7.86
	5	Max. features = 0.6, Min. samples for split = 5, Min. samples per leaf = 5	7.86	7.85	7.86
MLP	1	Hidden layers = (64, 64, 64, 32, 16, 8), Batch size = 32	7.77	7.73	7.75
	2	Hidden layers = (64, 64, 64, 32, 16, 8), Batch size = 64	7.75	7.65	7.70
	3	Hidden layers = (64, 64, 64, 32, 16, 8), Batch size = 64, Epochs = 35	**7.67**	**7.70**	**7.68**
	4	Hidden layers = (64, 64, 64, 32, 16, 8), Batch size = 64, Epochs = 50	7.90	7.74	7.82
	5	Hidden layers = (64, 64, 64, 32, 16, 8), Batch size = 256, Epochs = 50	7.82	7.67	7.75
	6	Hidden layers = (64, 64, 64, 32, 16, 8), Batch size = 128	7.84	7.70	7.77
	7	Hidden layers = (128, 128, 128, 64, 32, 16, 8), Batch size = 128	7.79	7.80	7.80
	8	Hidden layers = (256, 256, 128, 128, 64, 32, 16, 8), Batch size = 64	7.85	7.76	7.81
	9	Hidden layers = (512, 256, 128, 64, 32, 16, 8), Batch size = 32	7.73	7.73	7.73
LSTM	1	LSTM layers = (32), FC layers = (64, 64, 32, 16, 8), Batch size = 64	**7.50**	**7.55**	**7.53**
	2	LSTM layers = (64), FC layers = (64, 64, 32, 16, 8), Batch size = 64	7.68	7.63	7.66
	3	LSTM layers = (64), FC layers = (64, 64, 32, 16, 8), Batch size = 64, 10 state memory	7.85	7.61	7.73
	4	LSTM layers = (85), FC layers = (64, 64, 64, 32, 16, 8), Batch size = 64	7.81	7.75	7.78
	5	LSTM layers = (85), FC layers = (256, 128, 64, 32), Batch size = 32	7.71	7.74	7.73
	6	LSTM layers = (128), FC layers = (256, 128, 64, 32), Batch size = 32	8.05	7.71	7.88
	7	LSTM layers = (85), FC layers = (128, 256, 128, 64, 32), Batch size = 32	7.77	7.76	7.77
	8	LSTM layers = (85, 128), FC layers = (128, 256, 128, 64, 32), Batch size = 32	8.02	7.73	7.88
	9	LSTM layers = (85), FC layers = (512, 256, 128, 64, 32, 16, 8), Batch size = 32	7.87	7.85	7.86
	10	LSTM layers = (85), FC layers = (256, 128, 64, 32), Batch size = 32, 10 state memory	7.83	7.55	7.69
	11	LSTM layers = (85), FC layers = (256, 128, 64, 32), Batch size = 64	7.74	7.79	7.77

In addition to the evaluation results in table 4.1, the distribution of deviations from the annotations was also plotted for the best versions of each algorithm. Estimation errors were computed as

$$\text{Estimation error} = \text{Prediction} - \text{Annotation}, \tag{4.1}$$

such that a negative error means that the model predicted too short and a positive error indicates that the prediction was too long. The error distributions are presented as histograms in figures 4.11-4.13, where outliers larger than 40 meters or smaller than -40 meters are not shown. Still, the errors shown in the histograms

make up for more than 99% of the data. The histograms have been normalized such that the integral sums to 1. The distributions for the best RF model are shown in figure 4.11. The errors are distributed around zero, with a bias towards negative errors. The histograms for the MLP and LSTM models are shown in figure 4.12 and 4.13 respectively. As for the RF model, the errors are distributed around zero for both models. The MLP shows a small shift towards negative values, while the LSTM model is more evenly distributed.

Figure 4.10: A representative example of a typical learning progression, showing the R^2 score during training of MLP version 2. Considering the small gap between the training and validation score, there is barely any presence of overfitting. In this case, the curves indicated that the model could benefit from a longer learning duration, which is why this model was retrained with higher number of epochs.

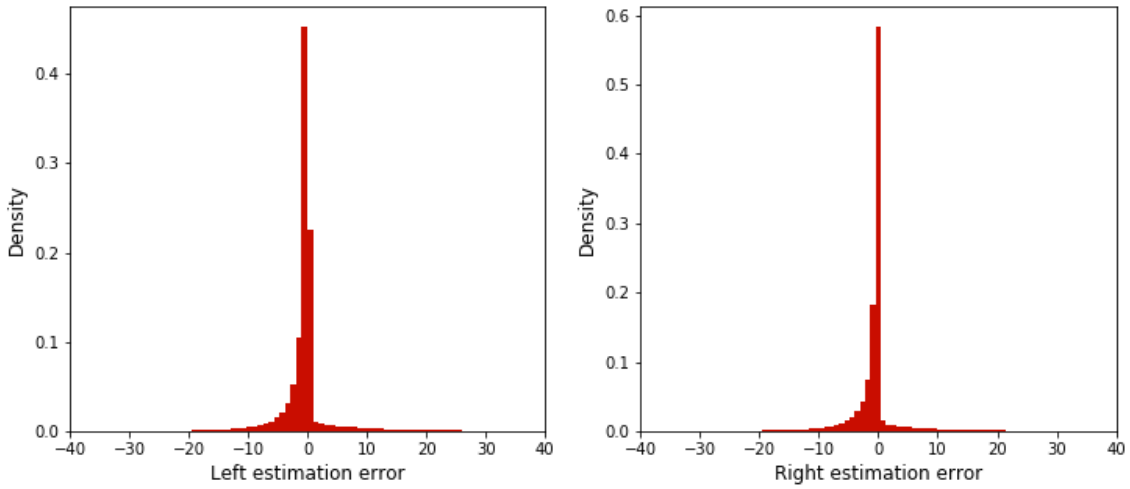


Figure 4.11: Distribution of the estimation errors for the best RF version, where outliers outside $[-40, 40]$ are not shown. The outliers not shown in the histograms make up for less than 1% of the errors.

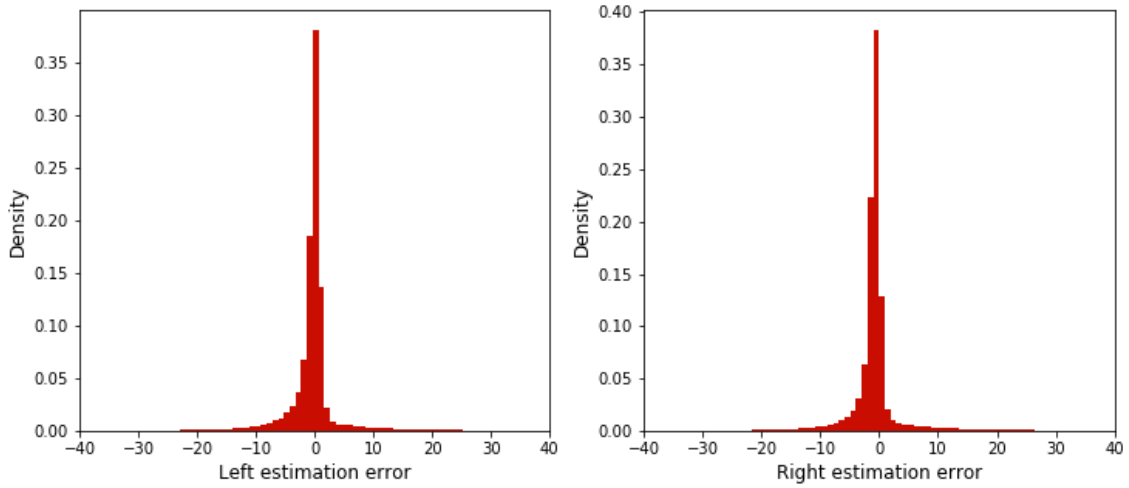


Figure 4.12: Distribution of the estimation errors for the best MLP version, where outliers outside $[-40, 40]$ are not shown. The outliers not shown in the histograms make up for less than 1% of the errors.

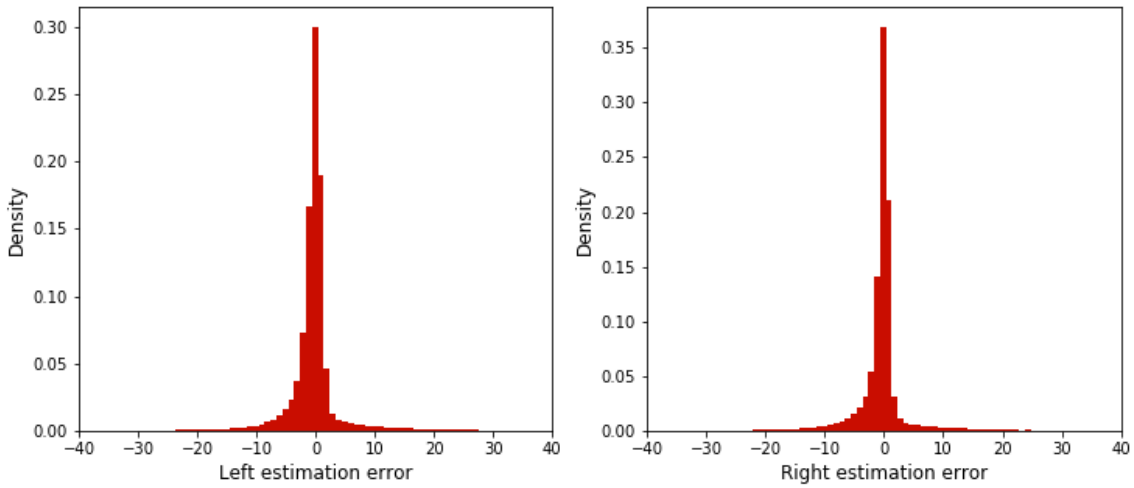


Figure 4.13: Distribution of the estimation errors for the best LSTM version, where outliers outside $[-40, 40]$ are not shown. The outliers not shown in the histograms make up for less than 1% of the errors.

4.3 RGF Results

The best performing model of each type, in terms of accuracy on the annotated data, was implemented in the RGF reference system, described in section 2.6, and simulated on the test set described in subsection 3.2.2. The highest performing LSTM model is version 1, which has a LSTM-layer with cell state length 32, followed by 5 fully connected layers with 64, 64, 32, 16 and 8 neurons respectively. It has a state memory of 5 time steps, meaning that it processes sequences of 5 steps, which corresponds to 1.5s of data with a sampling frequency of 4Hz. Furthermore, it was trained for 25 epochs using a batch size of 64 samples. The best MLP model is rather similar to the LSTM and has 6 fully connected layers, with 64, 64, 64, 32, 16 and 8 neurons respectively. Its training was done with 35 epochs and a batch size of 64 samples. Lastly, the best version of RF consists of 100 trees, where each tree has access to 40% of the available features. The minimum amount of samples under each branch split and leaf was set to 5 during its training.

As elaborated on in section 3.8, these models were implemented in the RGF reference system, which was then used in driving simulations, consisting of 9.5 hours of driving. The RGF performance was measured in the form of the lateral error and availability of the resulting road geometry estimation. The standard deviations of the lateral error at the different measurement distances are presented in table 4.2, as percentages of the corresponding results of the reference system with heuristics implemented. This gives a relative comparison between the machine learning models and the heuristic rejection method, implemented in the same RGF reference system. A percentage lower than 100 indicates a lower lateral error compared to the heuristic-based system, and vice versa. A percentage of 100 means that the model achieves the same lateral error. The resulting availability is presented in the same manner in table 4.3, where a higher percentage indicates a higher availability. From studying the two tables, it is possible to see that the LSTM excels in terms of lateral error on most distances, but has slightly lower availability than the heuristic-based system. The MLP maintains a low lateral error up to 100m but has the lowest availability. The RF model yields the largest lateral errors but is still performing better than the heuristic-based system on many distances. Regarding availability, the RF model achieves the best result, being the only machine learning model that on average generates better availability than the heuristic baseline.

Table 4.2: The lateral error standard deviations of the resulting road geometry estimation, with each algorithm type implemented in the RGF system. The Standard Deviation (STD) of the lateral error at each measuring distance, is presented as a percentage of the corresponding results using the heuristic-based system. The lowest percentage, which corresponds to the lowest lateral error, is written in bold for each distance.

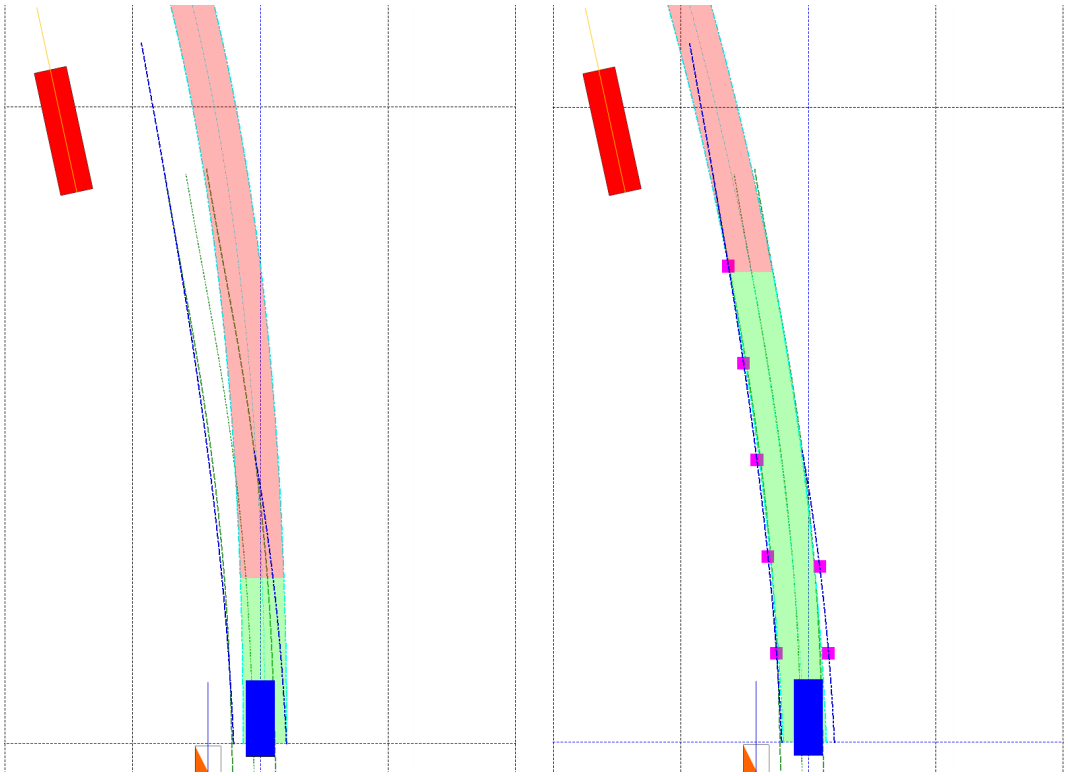
STD at distance[m]	**RF**[%]	**MLP**[%]	**LSTM**[%]
0	96.1	95.4	**94.2**
10	96.1	95.7	**94.8**
20	96.2	96.4	**95.2**
30	97.5	**96.1**	96.5
40	97.3	**95.5**	96.7
50	100.2	96.0	**95.9**
100	97.9	100.9	**88.9**
150	**106.2**	107.3	114.3
Reliable dist.	96.4	96.0	**94.5**

Table 4.3: The availability of the resulting road geometry estimation, with each algorithm type implemented in the RGF system. The availability is presented as percentages of the corresponding results using the heuristic-based system. The highest percentage, which corresponds to the highest availability, is written in bold for each distance.

Availability	**RF**[%]	**MLP**[%]	**LSTM**[%]
Average	**100.5**	96.6	98.0
Minimum	**102.3**	101.9	99.4
Maximum	**99.9**	96.0	99.5

The performance of the models implemented in the RGF system was further evaluated for a few selected examples of real-life driving scenarios, compared to the heuristic-based system. One of these examples is shown in figure 4.14, where the host vehicle is approaching a road fork. The host vehicle is represented as a blue box, and objects are illustrated as red boxes. The dashed blue lines represent the perceived lane markings and the purple squares represent the lane measurements fed to the Kalman filter. The dashed green lines are the GTA, showing the true ego lane to follow. The green and red areas constitute the resulting RGF estimation, with green representing an area of higher certainty. As seen in the left plot of the figure, the heuristic-based system has started to track the departing lane and chooses to discard the lane measurements at this point, resulting in a shorter certainty area of the lane estimation and a small lateral error from the GTA. The MLP, shown in the right plot of figure 4.14, manages to track the ego lane with a longer certainty area and a smaller lateral error in this particular situation. Another example of a fork scenario is shown in figure 4.15 and figure 4.16, where two succeeding time steps are shown. The ego vehicle in this example is again approaching a road fork, where the two preceding vehicles are following the departing lane. The first time

step is shown in figure 4.15, where the heuristic-based system chooses to sample the perceived lane markers from the right side and discards the measurements on the left, resulting in an estimation that follows the departing lane. The MLP, shown in the right plot of figure 4.15, keeps the measurements on the left and makes a more conservative decision on the lane markings for the right side. As a result of this, the MLP ends up with an estimation closer to the GTA in the next time step, as shown in figure 4.16.



(a) **(b)**

Figure 4.14: Example of the RGF system for a driving situation with a road fork, shown for a single time step. Plot (a) shows the heuristic-based system and plot (b) shows the MLP-based system. The host vehicle is represented as a blue box and objects in front of the vehicle are shown as red boxes. The dashed blue lines are the perceived lane markings and the purple squares represent the measurements passed to the Kalman filter. The dashed green lines are the GTA, showing the true ego lane to follow. The red and green areas constitute the resulting RGF estimation, representing areas of different certainty.

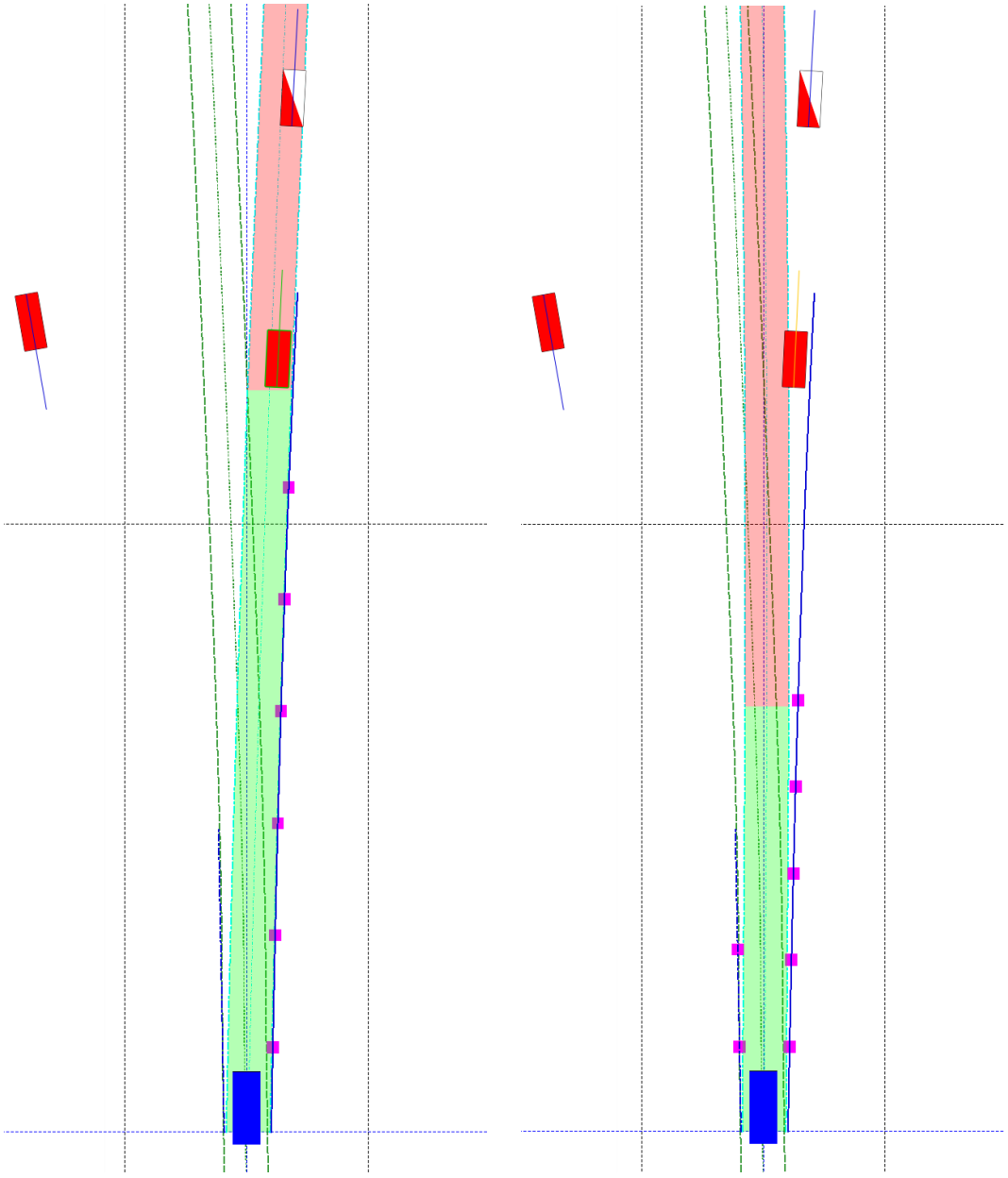


Figure 4.15: Example of the RGF system for a driving situation with a road fork, shown for a single time step. Plot (a) shows the heuristic-based system and plot (b) shows the MLP-based system. The host vehicle is represented as a blue box and objects in front of the vehicle are shown as red boxes. The dashed blue lines are the perceived lane markings and the purple squares represent the measurements passed to the Kalman filter. The dashed green lines are the GTA, showing the true ego lane to follow. The red and green areas constitute the resulting RGF estimation, representing areas of different certainty.

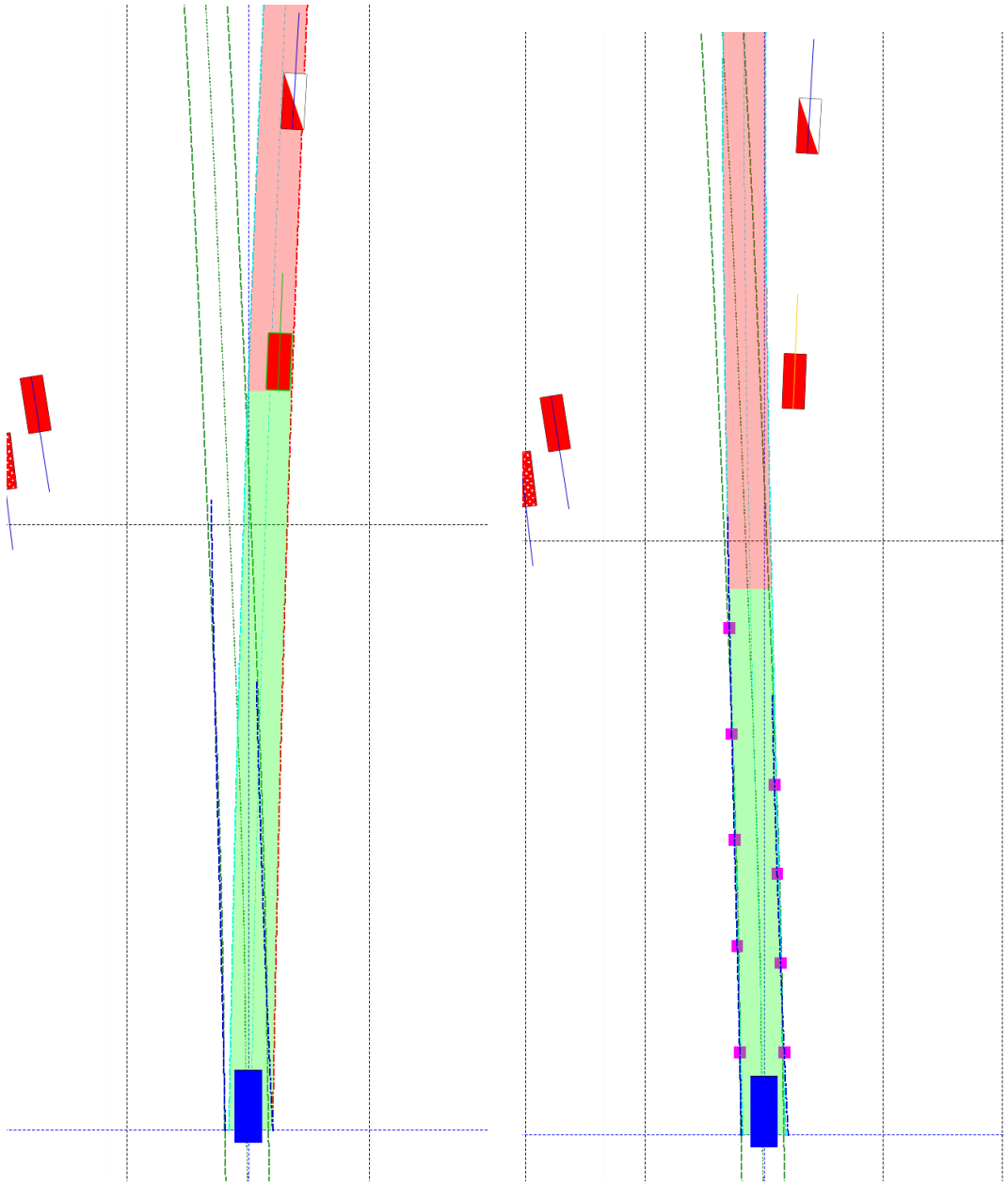


Figure 4.16: Example of the RGF system for a driving situation with a road fork, shown for a single time step. Plot (a) shows the heuristic-based system and plot (b) shows the MLP-based system. The host vehicle is represented as a blue box and objects in front of the vehicle are shown as red boxes. The dashed blue lines are the perceived lane markings and the purple squares represent the measurements passed to the Kalman filter. The dashed green lines are the GTA, showing the true ego lane to follow. The red and green areas constitute the resulting RGF estimation, representing areas of different certainty.

4.4 Evaluation of Auto Annotation Function

To verify the quality of the auto annotation function and its resulting output labels, it was implemented and tested in the RGF reference system, in the same way as the algorithms. However, due to issues with gaps in the GTA of some of the test files, the auto annotation function could only be evaluated on a subset of the RGF test set. In contrast to the annotation function, the algorithms do not require complete ground truth to function, since it is solely used to compute their lateral estimation errors in retrospect.

The remaining files with complete GTA constitutes about 5% of the entire RGF test set. The auto annotation algorithm was tested on these, with two different sets of parameters for the threshold function $T(x)$, to show its affect on the resulting output. The first set of parameters tested are the ones motivated in subsection 3.4.1, namely $T(x) = 0.3 + 0.005x$. The second setting was $T(x) = 0.4 + 0.0045x$, to showcase the affect of a less conservative threshold, allowing larger deviations closer to the host vehicle. The results are presented similar to the model evaluations in section 4.3, where the results are shown as percentages of the heuristic-based system results. The Standard Deviation (STD) of the lateral errors of the RGF for both settings of $T(x)$ are presented in table 4.4, and the resulting availability is presented in table 4.5.

Table 4.4: The auto annotation function implemented in the RGF, tested on the 5% of the data with complete GTA, for two different sets of parameters for the threshold $T(x)$. The best result on each distance is marked in bold.

STD at distance[m]	T(x) = 0.3 + 0.005x [%]	T(x) = 0.4 + 0.0045x [%]
0	**70,7**	72,8
10	80,3	**79,4**
20	**70,0**	85,4
30	**75,0**	91,9
40	**79,2**	96,0
50	**85,5**	101,0
100	**87,5**	90,5
150	131,2	**115,0**
Reliable dist.	**48,3**	52,1

Table 4.5: The resulting availability for the RGF, with the auto annotation function implemented, for two different settings for $T(x)$. The results are given as percentages of the corresponding results for the heuristic-based system, with the best results marked in bold.

Availability	T(x) = 0.3 + 0.005x [%]	T(x) = 0.4 + 0.0045x [%]
Average	63,5	**69,9**
Minimum	56,3	**61,0**
Maximum	68,5	**73,8**

As seen from these two tables, the more conservative setting of $T(x)$ yields a lower estimation error at most distances but also decreases the availability.

4.5 Model Prediction Time

The prediction time of each algorithm was evaluated for the best performing models from table 4.1. The evaluation was done by recording the time of each model performing 1000 predictions of individual samples and then dividing by the total number of samples. See section 4.2.1 for the model implementation details. The results from the prediction time evaluation are presented in table 4.6, and show that the MLP model was by far the most efficient with a prediction time of 0.26 milliseconds. The LSTM model was more than three times slower with a prediction time of 0.88 milliseconds. The RF model was the slowest of the models with a prediction time of 103.50 milliseconds, close to 400 times slower than the MLP model.

Table 4.6: Table showing the results from the prediction time evaluation. The time is averaged over 1000 predictions.

Algorithm	**Configuration**	**Prediction Time [ms]**
RF	Max. features = 0.4, Min. samples for split = 5, Min. samples per leaf = 5	103.50
MLP	Hidden layers = (64, 64, 64, 32, 16, 8), Batch size = 64, Epochs = 35	0.26
LSTM	LSTM layers = (64), FC layers = (64, 64, 32, 16, 8), Batch size = 64	0.88

5
Discussion

In this chapter, an analysis of the obtained results is given along with a discussion of the underlying methods. The chapter is closed with a section of recommendations and suggestions for any future work within the area.

5.1 Results Analysis

The analysis in this section is mainly based on the obtained regression accuracies and the results of the RGF implementations. The results will be compared between themselves, but also connected to the findings of the parameter searches.

5.1.1 Accuracy on Annotations

From table 4.1 it can be concluded that the most prominent algorithm, in terms of regression accuracy on the annotated data, is the Long Short-Term Memory. Nevertheless, the LSTM is closely followed by the Multilayer Perceptron and Random Forest. The MLP yields an RMSE about 2% higher than that of the LSTM. Furthermore, RF generates an RMSE about 2% higher than that of the MLP. The relatively consistent performance between the algorithms indicates that the current accuracy is close to what is obtainable with the available data and extracted features. Tuning of parameters and architectures may increase the performance a little bit further, but larger enhancements are more likely achieved via improvements of the feature space, not the algorithms themselves.

Despite small differences, the results and parameter searches show that different choices of parameters and architectures affect the performance in different ways. Starting with Random Forest, it seems that it is adequate to give each tree access to at most 40% of the total feature space. This can be seen from the leftmost plot in figure 4.2 as well as the results in table 4.1, where the best versions of RF utilizes a feature space limit of 40%. Regarding the minimum amount of samples for each split and leaf in the trees, the plots in figures 4.4 and 4.3 displays an expected trend for increased sample limit, namely that the overfitting decreases. This is seen from the fact that the training and validation error approach one another as the minimum number of samples under each split and leaf is increased. Unfortunately, the validation error increases even though the effect of overfitting is decreased. This is reflected in the regression results, where the best performance was achieved for low limits on the required number of training samples under each split and leaf. All of the RF models consisted of 100 trees, which is motivated by the plots in figure

4.5, where it is evident that a higher number of trees yields a very low improvement relative the increased computation time.

From figure 4.6 it can be observed that, for the MLP, the performance improvement slows down for layers with more than 40 neurons. This trend is confirmed by the results in table 4.1, where the best performing models are the ones with a maximum amount of 64 neurons per hidden layer. Nevertheless, version 9 of the MLP performs rather well with 512 neurons in its biggest layer, but it is unnecessarily big since smaller models are at least as good. The computational complexity is likely to be more fruitful if spent on increased depth instead of large layers. Training wise, 64 appears to be a good batch size, which can be seen by comparing otherwise similar versions of the MLP in table 4.1. Still, the left plot in figure 4.7 suggests that a batch size closer to 10 would be better. However, the parameter search was only carried out on a smaller subset of the data, which is why its findings may deviate from those of the regression results. What both sources of results agree on, is that a batch size larger than 100 worsens the performance. This is probably caused by premature convergence, which occurs when the batch size is too big, as described in subsection 2.4.6. It can also be a consequence of the decreased learning rate that an increased batch size contributes to, such that the training requires more epochs, but the training process becomes too slow for more epochs than 50. Regarding the number of epochs, an interesting observation can be made from table 4.1 and the results of MLP versions 2, 3, and 4. These are essentially the same model, as they have the same configuration, but have been trained for a different number of epochs. The number of epochs was increased for version 2, as its learning progression, presented in figure 4.10, showed potential to advance even further for an increased training duration. From table 4.1, it is apparent that 35 epochs were an appropriate training duration, while 50 epochs were too much, since the model started overfitting at that point. An increased number of epochs were not tried for models that did not exhibit any signs of potential improvement from more training.

A lot of the reasoning around the MLP can be carried over to the discussion of the LSTM results, since the LSTM models used in this thesis are hybrids, containing several fully connected layers. A proof of their correlation is the fact that the best performing LSTM and MLP share all layers except one, which is the first, where the hidden layer of the MLP was replaced with a LSTM-layer. With the idea being that the LSTM-layer could extract sequential information for all features and pass it to the MLP part, it appeared intuitive not to change the dimension of the input space before the fully connected layers, which was achieved by letting the LSTM cell state have the same size as the feature space. To clarify, the cell state was given length n such that the LSTM-layer output also had size n, which means that the input size was preserved. However, as revealed by the plot in figure 4.8, the rate of improvement is quite low for an increased number of LSTM-neurons or cell state size. This is probably why the models with cell state lengths 64 and 32 managed to outperform those with length n. The gained capacity of a larger cell state was simply too low, compared to the fact that it required over 50% more parameters to be trained, as can be shown using (2.23).

The reasoning around the batch size for the MLP applies to the LSTM as well, but a new aspect that comes with the LSTM is the length of the input sequences. In

this case, the models seem to have a hard time to exploit the sequential information, since increasing the sequence length does not yield any particular improvements, as seen in table 4.1. This can be a consequence of the data being too noisy or simply not including enough sequential information. It can also be that the difference between a memory or sequence of 5 or 10 time steps is too small to show any advancements. Perhaps the model could benefit from longer sequences, but that would increase the training time, since every estimation would require more computations, making it infeasible to try in this thesis.

Looking at the histograms in figures 4.11-4.13, it can be observed that all of the distributions are slightly skewed to the left, resulting in more negative than positive errors. This shows that the models are conservative, as they tend to make shorter rather than longer predictions of the reliable distance, compared to the corresponding label. This is a desirable behavior since the lateral error of the prediction grows with the prediction length and a conservative model will thereby result in smaller lateral errors. The data represented in the histograms also has a few larger outliers, although less than 1%. The effect of the negative outliers is that the model will simply output a very short reliable distance, which in the worst case will result in shorter lane estimation. The effect of positive outliers could have greater consequences, since the lateral error at the reliable distance might be larger than the accepted threshold, resulting in an inaccurate lane estimation. However, some of the outliers might be a result of the auto annotation function truncating the lane markers at a very short distance due to a temporary deviation close to the car, when in reality the lane markers are within the threshold for a much greater distance ahead.

5.1.2 RGF Performance

The RGF results, presented in section 4.3, reflect the regression accuracies displayed in subsection 4.2.2. The LSTM excels in terms of regression accuracy and consequently also yields the best RGF performance, while maintaining good availability.

Since the models are trained on data annotated by the created annotation function, they perform likewise when implemented in the RGF reference system. Consequently, a model with higher regression accuracy will perform more similar to the auto annotation function. This explains why all three models yield poor lateral errors at 150m, with the LSTM being the worst. Nevertheless, all three models seem to maintain better availability than the auto annotation function. This can appear contradicting to the plots in figures 4.11 - 4.13, which depict that the algorithms on average truncate the lane markers earlier than the corresponding annotations suggest. However, the road geometry estimation is not solely dependent on the perceived lane markers, meaning that neither is its availability. Depending on the situation, the lane marker truncation affects the resulting estimation and availability differently. Furthermore, the comparison between the results of the annotation function and the models is not adequate, since the annotation function was only tested on 5% of the actual test data.

Regarding the MLP, its RGF performance level lies somewhere in between the LSTM and RF model. It manages to decrease the lateral error up to 100m in front of the host vehicle, being the best at 30 to 40m but has worse availability than the

other two algorithms. The RF achieves the best availability but decreases the lateral errors the least. It has the best result on 150m, which is likely because it has the lowest regression accuracy and deviates more from the annotation function.

The MLP and LSTM showcase rather similar performance, which can be expected since they share a majority of their structures with one another. What separates the LSTM from the MLP, is the ability to process time series and extract sequential information. This advantage is likely what increases the capacity of the LSTM beyond that of the MLP.

Both the RF, MLP and LSTM manage to outperform the heuristic-based system in terms of lateral errors. A part of the lateral improvement can be explained from the slight decrease in average availability for the MLP and LSTM, since measurements close to the vehicles typically deviate less from the ground truth. Yet, this does not explain the entire improvement which can be seen in the RF results, where the lateral error is improved with maintained availability. A lot of the improvements are probably due to the increased capacity in the machine learning models. In contrast to the hand-crafted heuristics, the machine learning models do not require the problems to first be interpreted by humans, which can be a limiting factor in complex scenarios. Furthermore, the models have been generalized over a vast amount of data, which would be infeasible to manually analyze and derive heuristics from. However, the hand-crafted heuristic approach allows for backtracking of any errors made by the algorithm, while the machine learning models acts more like a "black box" which is harder to troubleshoot. In addition, the hand-crafted heuristics are easier to adapt to handling specific scenarios that rarely occur in the dataset. Thus, in the event of using a machine learning model, some level of heuristics may be required to ensure robustness for such scenarios.

5.1.3 Prediction Times

The applicability of an algorithm in the actual RGF system is not solely determined by the resulting precision and availability, but also at what frequency it can function, which makes the prediction times in table 4.6 an interesting study. Although the current implementation had no focus on being as efficient as possible, the results still show some important characteristics in the algorithms. The MLP and LSTM models were both sufficiently fast, being able to predict over a thousand samples per second in the current implementation, despite having deep and complex networks enough to fit very well to the data. Thus, the trade-off in complexity versus prediction time does not seem to be a concern for these models in the current setting. The RF however, was more than 100 times slower than the other two algorithms, and is not considered efficient enough in the current implementation. Another implementation, and perhaps a harder limit on the depth of the trees, would be necessary to consider using it in the RGF system.

5.2 Discussion of Chosen Methods

Some of the problems and tasks encountered in this thesis could have been approached in several ways. This section aims at discussing the methodical choices made for four significant parts of the method. Those parts are the conceptual system, the auto-annotation function, the dataset along with the selection of features, and the used evaluation metrics.

5.2.1 The Conceptual System

Although the conceptual system presented in section 3.1 has been proven functional through the results, there could potentially be other system solutions working as well or even better. For instance there are machine learning end-to-end solutions for lane marker tracking, as briefly discussed in section 1.4. Unfortunately, it has been hard to find comparable results for other solutions, but given the results in related work, it is fair to assume that an end-to-end solution has the potential to achieve good results in this application as well. An end-to-end solution could be a more effective way to go, as other parts like the Kalman filter could be dispensed, without adding any extra steps in the development of the machine learning model. However, for an end-to-end solution to be applicable, a larger part of the system chain must be accessible, like cameras and other sensors. Furthermore, a larger machine learning system complicates troubleshooting and the task complexity, which in turn requires a model with more capacity and longer training processes.

The machine learning subsystem in this thesis utilizes a multivariate model, which means that a single model is used to estimate multiple variables, which in this case are the reliable distances of the left and right lane marker. This could instead have been done using multiple models, one for the left lane marker and another one for the right. On one hand, the left and right lane marker may be subject to different types of disturbances, allowing two separate models to yield better results, as no weights and parameters would have to be shared. On the other hand, it could increase the risk of overfitting and making it hard to use the same system for both left- and right-hand side driving. Anyhow, provided that the desired model capacity would be the same, having two separate models would double the required time for training. The time aspect together with the desire to have a system that can take on both left- and right-hand side driving, are the reasons why the proposed solution was limited to a multivariate model.

Despite the machine learning models' success in enhancing the RGF system, that is not the task they have been optimized for. Instead, they have been trained to create regressions as similar as possible to the reliable distances produced by the auto-annotation function, to minimize the RMSE between the two. This indirect way of training the models to improve the RGF system was chosen to make the solution as independent and modular as possible. With this approach, the Kalman filter can be altered and tuned, without requiring retraining of the machine learning subsystem.

5.2.2 Auto-Annotation Function

As described in section 3.4, there were no established methods available for annotating the perceived lane marker data with their reliable distances. This demanded the development of such a method, which became a part of the work in this thesis. Using collected ground truth assessment to compute the lateral error of the perceived lane markers appeared to be a good starting point, but how to compare them was not as obvious. However, as explained in subsection 3.4.1, the RGF applications allow for bigger deviations for estimations further ahead of the host vehicle. This means that the error threshold T, which is ultimately used to annotate where each lane marker must be truncated, should increase with the estimation distance x. Without any knowledge about what type of function that would be appropriate, $T(x)$ was made linear to keep it as simple as possible.

From the results presented in table 4.4, it can be concluded that the annotation function works rather well since the RGF improves on all distances except one when utilizing the produced annotations. Nevertheless, the RGF error at 150m is increased, which means that the annotations produced at that distance are imperfect. Furthermore, $T(x)$ must be somewhat inadequate at that distance. This, together with the fact that a lot of availability is lost, as seen in table 4.5, makes it reasonable to believe that there are other sets of parameters or function types that could have worked better. Anyhow, as depicted in section 4.3, the models trained on the dataset annotated with this function showed promising results when implemented in the RGF system. Of course, the RGF results have the potential to improve even further, with proper tuning of the threshold $T(x)$.

Another benefit with the possibility to adjust $T(x)$, is that it is easy to change the balance of the trade-off between precision and availability in the road geometry estimation. This is showcased in section 4.4, where a less conservative version of $T(x)$ is introduced, in favor of better availability.

To conclude, the suggested method for auto-annotation fulfills its purpose in the current state, but there is room for improvement of $T(x)$, which can further enhance the functioning of the machine learning models in the RGF system.

5.2.3 Dataset and Feature Selection

As discussed in section 5.1.1, the results from the evaluation on annotated data indicate that further tuning of parameters and architectures is not likely going to increase the accuracy of the models that much. Any substantial improvements to the model accuracy are instead more likely to come from improving the feature space that the models operate on. The features used in this thesis were selected based on the method described in subsection 3.3.2, where different sets of features were tested using a Random Forest algorithm. Although this simple approach works fairly well to efficiently distinguish the more significant features in the full feature space, it has a few drawbacks. The most prominent issue with this method is that a different algorithm might give a different result, as they react differently to changes in the feature space. Hence, it is possible to make the argument that the feature selection in this thesis is made with a slight bias towards the Random Forest algorithm, as that was the only algorithm taken into consideration in this process. Ideally, all

of the algorithms would weigh in on the feature selection to make for an as fair comparison as possible. However, this would be a far too time-consuming activity for the scope of this thesis. Another drawback of this method is that it relies on a simplified assumption of the causality of different features when in reality relevant features in one set might be redundant in another and vice versa. Other wrapper approaches for selecting features could be to do a more exhaustive search of the feature space or to use a search algorithm such as particle swarm optimization.

In addition to the feature selection, improvements could also be found in new representations of the features. The most significant feature for this task is, without doubt, the information regarding the "ego lane". This information was sampled at 17 different distances, as described in section 3.3.1, but could benefit from another resolution or even another representation. Another type of representation could be to provide the actual coefficients of the polynomials, $f_{left}(x)$ and $f_{right}(x)$, that are used to describe the lane markers.

Finally, increasing the size of the dataset is also most likely going to increase the model accuracy, as more data would allow the models to learn from more scenarios and consequently generalize better to new unseen data.

5.2.4 Evaluation Metrics

Due to the layout of the host vehicle coordinate system, the sign of the resulting estimation error differs between left and right turns. Thus, the mean of the lateral error is heavily affected by the number of different turns in the test set. This is why the STD of the lateral error has been used as its evaluation metric, rather than the RMSE. In contrast to the RMSE, the STD displays the variation of the lateral error but not the bias, which leaves it unaffected by the distribution of left and right turns. Consequently, the STD is also unaffected by any other bias present in the lateral error. This means that constant errors, caused by inaccuracies of the RGF system, could exist in the resulting estimation without being conveyed by the STD results. With this said, it is unlikely that such a bias would be caused by the developed machine learning system, as the models have shown level performance for the estimation of the left and right reliable distances. Still, further investigation of the lateral error and an eventual bias is needed in order to validate the final RGF system.

5.3 Future Work

The previous section described different alternatives to parts of the used methodology, of which some have the potential to improve the resulting machine learning system even further. One of the more significant parts was the feature selection, which can be done more comprehensively in the future, using other tools and methods. This thesis could then serve as a basis for choosing a single algorithm to proceed with, which would simplify the feature selection process, as it is somewhat dependent on the model type. Another important matter regarding the features is the feature engineering, which is how the information from lane markers and other

sources is conveyed to the machine learning models. Some features could likely be disposed and combined differently, to be more beneficial for the algorithms.

Another possible performance improvement is to include the Kalman filter in the training loop. Then, instead of minimizing the difference between the regressions and created annotations, the error between the resulting estimation and GTA could be minimized directly. The feature space could be the same and so could the way of truncating the lane markers before they are passed to the filter. The difference would be that the models are directly trained on the task that they are eventually intended to solve. One way of doing this could be via reinforcement learning, although it would be challenging, considering the size and complexity of the RGF system.

As stated in section 1.2, the non-Gaussian disturbances occur in the movement measurements of preceding vehicles as well, for instance in the form of lane changes. The trajectory of a preceding vehicle at a lane change deviates from the ego lane, similar to the lane markers at a road fork. Preceding vehicles at a road fork poses the same problem, assuming that they follow a different lane to that of the host vehicle. An interesting study for the future would be to apply a similar machine learning system to the measurements of preceding vehicles. Perhaps, it is also possible to combine the filtering of both measurement types in one machine learning system, such that the model can base its output on information from both sources.

The possibility of having separate models for the left and right lane marker has already been discussed, but an extension of this idea is to also develop separate models for different driving scenarios. It could be as simple as having two different models for left and right-hand side driving, or more complex, such that there are several models for different road types. Although this solution might be less flexible, the task complexity could be significantly reduced for each model. The models could then generalize less and specialize more, improving the performance within their respective areas of usage.

6
Conclusion

This thesis investigates how well a machine learning algorithm, compared to hand-crafted heuristics, can discover and reject non-Gaussian disturbances in perceived lane markers. Four different algorithms are included in the study, chosen based on comparative studies and methodology in related work. Several models of each algorithm type are trained on a dataset, based on information about the traversed lane and internal vehicle states, collected from 220 hours of driving. Furthermore, the thesis proposes a method for using ground truth assessment, to annotate the data with the reliable distance of the perceived lane markers at each time step. From training, the models learn to output estimations of these distances, which are then used to truncate the lane markers before they are passed to the Kalman filter.

One of the candidate algorithms is SVR, but it is quickly discarded from the study, as it scales badly to the vast amount of data used to train the models. The remaining algorithms, RF, MLP and LSTM, are evaluated based on their ability to enhance the RGF reference system and return estimations similar to the corresponding annotations.

The resulting road geometry estimation from utilizing each algorithm is compared to that of the heuristic-based system, which utilizes hand-crafted heuristics to truncate the lane markers. All three algorithms, RF, MLP, and LSTM, manage to fit the training data and improve the road geometry estimation on the test set, beyond the performance of the heuristic-based system, by successfully rejecting non-Gaussian disturbances in the lane marker measurements. The characteristics of a successful machine learning algorithm in this setting include a large enough complexity and capacity to be able to generalize to the wide range of possible driving scenarios, an attribute that the SVR fails to fulfill. Further, the ability to remember sequential data is proven to a be desirable characteristic in this environment.

Random Forest proves to be simple and quickly applicable, yielding high estimation availability, but not quite reaching the same precision as the MLP and LSTM. Furthermore, it has an unacceptably high prediction time in its current implementation, making it inappropriate to use in a real-time system. In comparison, the MLP increases the estimation precision even further while losing a bit of availability. It has the lowest prediction time of all models, making it a better choice for actual implementation. However, the LSTM manages to advance from the MLP in terms of both precision and availability of the resulting estimation, while maintaining an acceptable prediction time. Nevertheless, the MLP and LSTM performs likewise, which is to expect since they have similar architecture. What separates the LSTM from the MLP, is the ability to process time series and extract sequential information, which in the end is an advantage that lets it outperform the MLP.

Based on its performance, in terms of both regression accuracy and RGF improvement, the LSTM is the suggested choice for replacing the hand-crafted heuristic disturbance rejection. Further advancements could be obtained by tuning the architecture and parameters but are more likely to be achieved through improvements of the feature space and extension of the dataset. It is also possible to deploy multiple models, for different areas of usage, such that the task complexity is reduced for each model.

This thesis eventually contributes with a conceptual machine learning system, that can improve the road geometry estimation, while also decreasing the number of hours spent on maintenance and development work. It also contributes with a method, featuring an adjustable balance of the trade-off between precision and availability, for automatically annotating perceived lane markers with their reliable distances. The method is validated through the success of the well-performing algorithms but can be further improved by finding a better function for the error threshold.

Bibliography

[1] Michael Rivera. Florida I95 SB Exit 311, 2016.

[2] F Deloche. Recurrent neural network unfold, 2017.

[3] F Deloche. A diagram for a one-unit Long Short-Term Memory (LSTM), 2017.

[4] Todd Litman. Autonomous Vehicle Implementation Predictions: Implications for Transport Planning. Technical report, Victoria Transport Policy Institute, Victoria, 2020.

[5] National Highway Traffic Safety Administration. 2016 Fatal Motor Vehicle Crashes: Overview. *Traffic Safety Facts: Research Note (DOT HS 812456)*, 1(October):1–9, 2017.

[6] Meng Lu, Kees Wevers, and Rob Van Der Heijden. Technical feasibility of Advanced Driver Assistance Systems (ADAS) for road traffic safety. *Transportation Planning and Technology*, 28(3):167–187, 2005.

[7] Christian Lundquist and Thomas B Schön. Joint ego-motion and road geometry estimation. *Information Fusion*, 12(4):253–263, 2011.

[8] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. *In Practice*, 7(1):1–16, 2006.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, dec 2018.

[10] Israr Ullah, Muhammad Fayaz, and Do Hyeun Kim. Improving accuracy of the Kalman filter algorithm in dynamic conditions using ANN-based learning module. *Symmetry*, 11(1), 2019.

[11] Aharon Bar Hillel, Ronen Lerner, Dan Levi, and Guy Raz. Recent progress in road and lane detection: A survey. *Machine Vision and Applications*, 25(3):727–745, 2014.

[12] Qin Zou, Hanwen Jiang, Qiyu Dai, Yuanhao Yue, Long Chen, and Qian Wang. Robust lane detection from continuous driving scenes using deep neural networks. *IEEE Transactions on Vehicular Technology*, 69(1):41–54, 2020.

[13] Byeoung Do Kim, Chang Mook Kang, Jaekyum Kim, Seung Hi Lee, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-March, pages 399–404. Institute of Electrical and Electronics Engineers Inc., mar 2018.

[14] Dong Fan Xie, Zhe Zhe Fang, Bin Jia, and Zhengbing He. A data-driven lane-changing model based on deep learning. *Transportation Research Part C: Emerging Technologies*, 106:41–60, sep 2019.

[15] David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing*, 55(1-2):169–186, 2003.

[16] Yongjian Chen, Masatake Akutagawa, Masato Katayama, Qinyu Zhang, and Yohsuke Kinouchi. Neural network based EEG denoising. In *Proceedings of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS'08 - "Personalized Healthcare through Technology"*, pages 262–265, 2008.

[17] Jo Anne Ting, Evangelos Theodorou, and Stefan Schaal. A Kalman filter for robust outlier detection. *IEEE International Conference on Intelligent Robots and Systems*, pages 1514–1519, 2007.

[18] Gabriel Agamennoni, Juan I. Nieto, and Eduardo M. Nebot. An outlier-robust Kalman filter. *Proceedings - IEEE International Conference on Robotics and Automation*, 1(May):1551–1558, 2011.

[19] UN. Transforming Our World: the 2030 Agenda for Sustainable Development United Nations United Nations Transforming Our World: the 2030 Agenda for Sustainable Development. A/RES/70/1. Technical Report October, United Nations, 2015.

[20] Max Bramer. *Principles of Data Mining*. Springer-Verlag London, Portsmouth, 2 edition, 2007.

[21] Stephen Marsland. *Machine Learning - An Algorithmic Perspective*. CRC Press Taylor & Francis Group, Boca Raton, 2 edition, 2015.

[22] Wei Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, jan 2011.

[23] Douglas M. Hawkins. The Problem of Overfitting, jan 2004.

[24] Ian Goodfellow and Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[25] N. J.D. Nagelkerke. A note on a general definition of the coefficient of determination, 1991.

[26] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, jan 2001.

[27] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[28] Rimah Amami, Dorra Ben Ayed, and Noureddine Ellouze. Practical Selection of SVM Supervised Parameters with Different Feature Representations for Vowel Recognition. *International Journal of Digital Content Technology and its Applications(JDCTA)*, 7(9), 2013.

[29] Mariette Awad and Rahul Khana. *Efficeient Learning Machines - Theories, Concepts and Applications for Engineers and System Designers*, volume 1. Apress, Berkeley, 2015.

[30] Richard P Lippmann. An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, 4(2):4–22, 1987.

[31] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *ICASSP,*

IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, pages 8609–8613, 2013.

[32] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve Restricted Boltzmann machines. In *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, pages 807–814, 2010.

[33] Andrew L Maas, Awni Y.Hunnun, and Andrew Y.Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *JMLR WC&P*, 28, 2013.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[35] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure*, volume 1. MIT Press, Massachusetts, 9 edition, 1987.

[36] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[37] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, 2016.

[38] Paul J. Werbos. Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[39] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. In John F. Kolen and Stefan C. Kremer, editors, *A Field Guide to Dynamical Recurrent Networks*. IEEE Press, New York, 1 edition, 2001.

[40] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[41] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[42] Yuelei Xiao and Yang Yin. Hybrid LSTM neural network for short-term traffic flow prediction. *Information (Switzerland)*, 10(3), 2019.

[43] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak. Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 2015-Augus, pages 4580–4584, 2015.

[44] Lars Hammarstrand, Maryam Fatemi, Ángel F García-Fernández, and Lennart Svensson. Long-range road geometry estimation using moving vehicles and road-side observations. *IEEE Transactions on Intelligent Transportation Systems*, 17(8):2144–2158, 2016.

[45] Christian Lundquist and Thomas B Schön. Road geometry estimation and vehicle tracking using a single track model. Technical report, Division of Automatic Control, Linköping University, Linköping, 2008.

[46] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon

Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. *arXiv*, 2019.

[47] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. *arXiv*, 2018.

[48] Yaniv Alon, Andras Ferencz, and Amnon Shashua. Off-road path following using region classification and geometric projection constraints. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 689–696, 2006.

[49] M. Slop & F.C.M. Wegman (Eds.) H.G.J.C.M. Ruyters. Safetyeffects of road design standards. Technical report, SWOV Institute tor Road Safety Research, The Netherlands, 1994.

[50] K. R. Müller, A. J. Smoła, G. Rätsch, B. Schölkopf, J Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1327, pages 999–1004, 1997.

[51] Huikun Bi, Tianlu Mao, Zhaoqi Wang, and Zhigang Deng. A Data-driven Model for Lane-changing in Traffic Simulation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, pages 1–11, 2016.

[52] Yi Hou, Praveen Edara, and Carlos Sun. Situation assessment and decision making for lane change assistance using ensemble learning methods. *Expert Systems with Applications*, 42(8):3875–3882, may 2015.

[53] François Chollet and Others. Keras. \url{https://keras.io}, 2015.

[54] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[55] Christopher K.I. Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, 2001.

[56] Alex J. Smola and Bernhard Schölkopf. Sparse Greedy Matrix Approximation for Machine Learning. *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.