CHALMERS
UNIVERSITY OF TECHNOLOGY

Data-Driven Model Optimization for Battery Electric Vehicles

A development of digital model component optimization methods using logged sensor data

Master's thesis in Engineering Mathematics and Computational Science

HANNES BJÖRCK
ERIK DAHLSTEDT

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Data-Driven Model Optimization for Battery Electric Vehicles

A development of digital model component optimization methods using logged sensor data

HANNES BJÖRCK
ERIK DAHLSTEDT

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2025

Data-Driven Model Optimization for Battery Electric Vehicles
A development of digital model component optimization methods using logged sensor data
HANNES BJÖRCK
ERIK DAHLSTEDT

Supervisors: Abdolreza Taheri, Karl Svantesson, and Noel Hall (Volvo Group Trucks Technology)
Examiner: Johan Jonasson, Department of Mathematical Sciences

Master's Thesis 2025
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Futuristic digital truck from Open Studio/stock.adobe.com

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Abstract

Digital models of real world trucks are beneficial for analysis of truck components, especially in regards to degradation and anomaly detection. This thesis' objective is to develop methods for optimizing model performance. This was done by using real-world data from logged driving sessions as input to the modeled components, and comparing its output to the corresponding logged signals. One method that was developed optimizes the accuracy of lookup tables. The method consisted of the gradient descend method ADAM combined with Gaussian kernel smoothing. It was developed and tested on one lookup table and its corresponding signal, showing an MAE reduction to 34-38% of the original MAE. The updated values in the table had a generally good shape but lack of data in the extremes led to some inconsistencies. The other method focused on using linear regression to fit a simulated signal to its corresponding logged signal, and preferably overestimating the logged signal. This was done by using a weighted penalty function where the weight was decided by a custom-built algorithm. The linear regression resulted in small changes in MAE, but the maximum error decreased for all vehicles while the fraction of the time the simulated signal overestimated the logged signal increased across all vehicles.

Keywords: machine learning, linear regression, model optimization, digital model, lookup table optimization.

Acknowledgments

We would like to express our gratitude to our wonderful supervisors, Abdolreza Taheri, Karl Svantesson, and Noel Hall at Volvo Group Trucks Technology, for their invaluable guidance and support throughout our journey. Also, to the rest of the team at Simulation Dragons, thank you for your help and camaraderie. Your insights and encouragement have made a significant difference in our work. A special thank you also goes to our examiner, Johan Jonasson, for his continuous support and guidance. Thank you all for being such an integral part of our success!

Hannes Björck & Erik Dahlstedt, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BEV	Battery electric vehicle
EM	Electric machine
EMD	Electric motor drive
MAE	Mean absolute error
MDS	Motor drive system
MD	Mean difference
ME	Maximum error
TAR	Time above ratio

Contents

List of Figures

List of Tables

1
Introduction

Electric trucks have become more and more popular all over the world in recent years, with the demand expected to increase in the future [1]. Consequently, people all over the world are, and increasingly will be, indirectly dependent on electric trucks and their maintenance. In order for people to keep living their lives without disturbances from the supply chain it is imperative that the trucks that deliver their food to the grocery stores, their clothes to the clothing stores and their medicine to the pharmacies do not fault in the middle of operation. This project's ultimate aim is to help prevent this from happening by advancing a digital model of Volvo battery electric trucks that can be used to detect abnormal behavior or degradation in the trucks that are in operation.

1.1 Background

The functionality and competitiveness of battery electric vehicles (BEVs) depend on furthering the driving range and intelligently identifying potential component issues. This can be done by analysis of components such as the Electric Machine (EM), Energy Storage System (ESS), and supporting software systems. Maximizing their efficiency and minimizing their problems will be a key element in the development of BEVs.

These tasks can be achieved by examining and working with the large amount of sensor data already gathered from BEVs in operation. The vastness of this data, due to the large amount of sensors and the high-frequency nature of the data collection, makes the analysis difficult. There is a need for development of efficient data management to handle the vast amount of information as well as high performance and large-scale computing of this data. There is also a need for specialized algorithms to extract knowledge from the continuous stream of information.

Volvo Group Trucks Technology have developed a virtual model that simulates a driving session by a truck. A virtual truck operating accurately, that can be compared to trucks in service, is beneficial for performance analysis. Utilizing the abundance of sensor data available for each individual truck, the model can be tuned so that all trucks have their own digital twin, a model specifically tuned for each truck. With a tailor-made model for each truck, the analysis can go from more general to product specific. A deviation in the expected behavior between the truck and its virtual twin signals a potential issue in the BEV. Then the faulty or degrading components can be noticed and exchanged for the truck leading to better service.

As electric trucks operate over time, the degradation of various components leads to changes in their performance characteristics. This degradation can affect things such as the efficiency and reliability of the vehicle. In order to have a digital twin that accurately represents each truck's current characteristics, it is essential to develop digital twins that evolve and change alongside the physical vehicle. These digital models must be capable of continuous adaptation, reflecting changes in the truck's state due to wear and tear. Creating a system that can continuously utilize and adapt to a continuous stream of data will give a more accurate digital twin. This will enable the digital twin to provide a comprehensive and dynamic representation of the vehicle, promote proactive maintenance and optimization strategies.

1.2 Aim and Objectives

Analyze the current digital model with real data and evaluate its performance across different vehicles and components. Develop a workflow for optimization of digital model components of Volvo BEVs. The specific objectives are:

- Data extraction, transformation, and signal processing at large-scale.
- Analyze the model in its current form with real logged data.
- Develop methods for optimization of model parameters to better fit real data.
- Given data from a fleet of vehicles analyze the distribution of parameters within the fleet of trucks.

1.3 Scope and Limitations

- We will only look at the parameters of a handful of components which we assume are the most influential parameters and every unmodeled effect is assumed to be a disturbance.
- The simulation model comprises simplified models for the components and does not take all the details into account.

1.4 Ethical and Sustainability Aspects

As this project will be carried out by working on computers, there is no way to avoid the ecological issue of energy usage. Furthermore, there may be some heavy computations over the course of the project which require a lot of electricity. Given that this project's aim is to improve BEVs, it hopefully saves more energy in the future than it requires to be completed. All use of trucks' data will be performed without personal identification of drivers and in full compliance with applicable regulations.

2

Theory

2.1 Simulink Model

2.1.1 Model Subcomponent

Simulink is a software that allows the user to model and simulate complex systems [2]. The truck dynamics are integrated altogether in a Simulink model, allowing simulation of various driving cycles as accurately as possible. This enables the analysis of how variations of conditions such as ambient temperature, rolling resistance, driver behavior, and the truck's thermal dynamics, affects the vehicle. Driving sessions are simulated using this model so that data that might not be readily available can be created digitally. The model consists of numerous components, most of which have several levels of subcomponents, resulting in an intricate architecture. This paper focuses on developing methods for component optimization, and as such one of these components was used for development.

2.2 Model Evaluation Metrics

The metrics that were used to evaluate the simulated data against the real data were the mean absolute error (MAE), maximum error (ME), mean difference (MD) and time above ratio (TAR). These metrics were computed according to the following formulas:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|, \tag{2.1}$$

$$ME = \max_{i\in[1,n]}\{|\hat{y}_i - y_i|\}, \tag{2.2}$$

$$MD = \frac{1}{n}\sum_{i=1}^{n} \hat{y}_i - y_i, \tag{2.3}$$

$$TAR = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(\hat{y}_i > y_i), \tag{2.4}$$

where n is the length of the signal of interest, $\hat{y}_i$ is the i:th instance of some simulation output signal, y_i is the i:th instance of the corresponding logged signal and $\mathbb{1}$ is the indicator function.

2.3 Optimization

2.3.1 Multidimensional Lookup Table

A lookup table is a data structure with precomputed values that replaces a complex function or experimentally produced data to speed up calculations. The table is a mapping of input values to output values, corresponding to the function calculated at specific, finite, input values. Given the input values $(x_i, y_j, z_k), i = 1, 2, ..., N, j = 1, 2, ...M, k = 1, 2, ..., O$, the table stores the output value from the function or experiment in a matrix as $T(i, j, k) = g(x_i, y_j, z_k)$. When attempting to calculate a value of the function using the lookup table, the input is used to find where in the table the corresponding function value is found [3].

The input used for finding the value will rarely be a precomputed point, but somewhere in between two points. In order to be able to fetch a value from a point between the matrices indices, the table uses index and fraction, these are calculated from inputs and breakpoints. Breakpoints are the specific values for the dimensions with which the function was calculated. Given an input from the first dimension, x, and a vector of breakpoints, $B = [b_1, b_2, ..., b_{n-1}, b_n]$, which corresponds to input values of the precomputations. We firstly calculate

$$\begin{aligned} i &= \underset{j}{\operatorname{argmax}} \quad b_j < x \\ f &= \frac{x - b_i}{b_{i+1} - b_i}, \end{aligned} \tag{2.5}$$

which are the index and fraction. The fraction carries the information of where the value is in the interval between the indexed breakpoint and the next breakpoint. These are then used to locate where in the lookup table the output value is to be extracted from with interpolation.

In this project, the lookup tables are of three dimensions and use the linear point-slope interpolation method to calculate the intermediate values [4]. Therefore, any value not on exact break points will be in the interior of a cube, where the eight corners are the precomputed values in the table. Using the three pairs of indices and fractions $(i_1, f_1), (i_2, f_2), (i_3, f_3)$ calculated from input (x, y, z), and the lookup tabel's matrix of precomputed values M,

$$\begin{aligned} c_{000} &= M(i_1, i_2, i_3,) \\ c_{100} &= M(i_1 + 1, i_2, i_3) \\ &\vdots \\ c_{111} &= M(i_1 + 1, i_2 + 1, i_3 + 1) \end{aligned}$$

are the values of these corner points. With information of the values in the lattice, the intermediate point in the middle can be calculated from the corners. First, a linear interpolation along the x-axis is performed as

$$
\begin{aligned}
c_{00} &= c_{000} \cdot (1 - f_1) + c_{100} \cdot f_1 \\
c_{01} &= c_{001} \cdot (1 - f_1) + c_{101} \cdot f_1 \\
c_{10} &= c_{010} \cdot (1 - f_1) + c_{110} \cdot f_1 \\
c_{11} &= c_{011} \cdot (1 - f_1) + c_{111} \cdot f_1.
\end{aligned} \tag{2.6}
$$

This transforms it to a two dimensional interpolation, the next step,

$$
\begin{aligned}
c_0 &= c_{00} \cdot (1 - f_2) + c_{10} \cdot f_2 \\
c_1 &= c_{01} \cdot (1 - f_2) + c_{11} \cdot f_2,
\end{aligned} \tag{2.7}
$$

interpolates along the y-axis. The last step is then to do it again for the z-axis which gives

$$
C = c_0 \cdot (1 - f_3) + c_1 \cdot f_3. \tag{2.8}
$$

The final value is then the predicted function or experiment value in the intermediate point. With fixed indices and fractions, this interpolation can be written as a function solely dependant on the corner points:

$$
\begin{aligned}
I(c) = {} & [(c_{000} \cdot (1 - f_1) + c_{100} \cdot f_1) \cdot (1 - f_2) \\
& + (c_{010} \cdot (1 - f_1) + c_{110} \cdot f_1) \cdot f_2] \cdot (1 - f_3) \\
& + [(c_{001} \cdot (1 - f_1) + c_{101} \cdot f_1) \cdot (1 - f_2) \\
& + (c_{011} \cdot (1 - f_1) + c_{111} \cdot f_1) \cdot f_2)] \cdot f_3.
\end{aligned} \tag{2.9}
$$

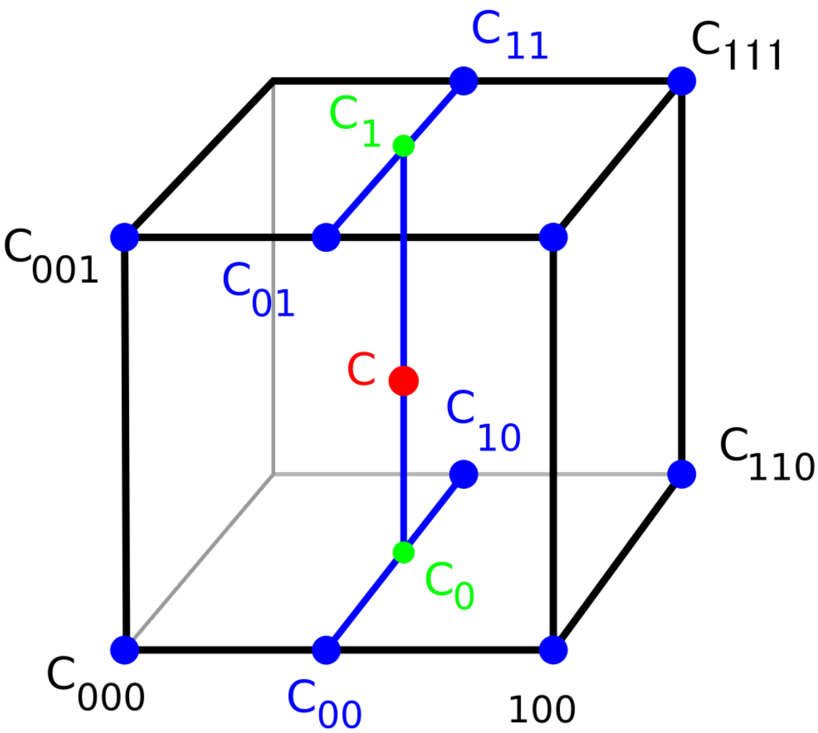

Figure 2.1: 3D interpolation illustration, By Marmelad - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4355112.

2.3.1.1 Objective Function

Optimizing the lookup table involves adjusting the precomputed values in the matrix to ensure that the output aligns more closely with the desired target values. These precomputed values are the parameters that should be tuned during the optimization process. For a given time point, $t \in 0, 1, ..., T$, the indices and fractions are fixed. The output at this time point can therefore be expressed as the function $I_t(c_t)$ of the eight values ($c_t = (c_{t,000}, c_{t,100}, \dots, c_{t,111})$). The desired target is represented by the logged data L_t. The optimization problem can be formulated as a minimization problem, where the objective is to minimize the cost function that is defined as the squared difference between the output $I_t(c_t)$ and the target L_t. This can be mathematically expressed as:

$$\min_{c} \sum_{t=0}^{T} (L_t - I_t(c_t))^2. \tag{2.10}$$

2.3.1.2 Gradient of Interpolation

To perform gradient based optimization of the minimization problem (Equation 2.10), the gradient must be calculated. The total gradient for a route is the sum of the gradient across all time steps. For a single time point, the contribution to the gradient of the cost function with respect to one of the parameters is given by:

$$\begin{aligned} \frac{\partial (L - I(c))^2}{\partial c_{000}} &= -2(L - I(c)) \frac{\partial I(c)}{\partial c_{000}} = 2\epsilon \frac{\partial I(c)}{\partial c_{000}} \\ &= 2\epsilon(1 - f_1)(1 - f_2)(1 - f_3) \end{aligned} \tag{2.11}$$

which is the partial derivative of Equation 2.9 times the ϵ, which represents the error between the output and the target. Similar derivation can be performed for the other partial derivatives to get the specific time step's gradient. To obtain the total gradient for the time series, the gradients from each time step are summed, noting that the corners, $c_{000}, c_{001} \dots, c_{111}$ correspond to specific matrix indices at a given time step.

2.3.1.3 Adam (Adaptive Moment Estimation)

Gradient-based minimization of functions is a core concept in optimization for machine learning. The main idea of the method is based on calculating the direction of descent for a cost function. This is done by finding the gradient which represents the path of ascent. The cost function is minimized by iteratively updating the parameters according to the negative of the direction.

In machine learning, learning rates are an important hyperparameter to make learning functional and efficient. Having a dynamic update rule that changes the learning rate in accordance to the models response can lead to better performance. Adaptive Moment Estimation (Adam) is a popular algorithm for complex machine learning, it combines the strengths from AdaGrad and RMSProp [6].

It starts with initializing a step size α, two empty momentum matrices m_0 and v_0, along with their respective exponential decay rates $\beta_1, \beta_2 \in [0, 1)$ and an initial parameter vector θ_0. Then for each timestep t, the objective function $f(\theta)$ gradient is calculated

$$g_t \leftarrow \nabla_\theta f(\theta_{t-1}). \tag{2.12}$$

$$\begin{aligned} m_t &\leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t && (2.13) \\ v_t &\leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. && (2.14) \\ & && (2.15) \end{aligned}$$

$$\begin{aligned} \hat{m}_t &\leftarrow \frac{m_t}{1 - \beta_1^t} && (2.16) \\ \hat{v}_t &\leftarrow \frac{v_t}{1 - \beta_2^t} && (2.17) \end{aligned}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\hat{v}_t + \epsilon}, \tag{2.18}$$

2.3.1.4 Isotropic Kernel

An isotropic kernel is a mapping in space, that is invariant to translations in the input space and solely dependent on the size of the difference between the inputs [7]. Given the inputs x, x_0, and a function, f, the kernel is a measurement of the difference,

$$K(x, x_0) = K(|| \ x - x_0 \ ||) = f(|| \ x - x_0 \ ||). \tag{2.19}$$

2.3.1.5 Smoothing Kernel

A kernel smoother is a statistical technique that can be used to even out the data. By smoothing the values with weighted averages of neighboring data points, a flattening occurs. Assume the kernel is normalized such that $\int K(t)dt = 1$, and with the isotropic property as above, one can define it as $K(t, s) = f(\|t - s\|) = K(t - s)$ as it only depends on the difference [8].

Let $X(t)$ be a set of datapoints and K a kernel as above, then

$$Y(t) = K * X(t) = \int K(t - s)X(s)ds \tag{2.20}$$

is the kernel smoothing of X. The data points in Y is a superposition of all the points in the data set X.

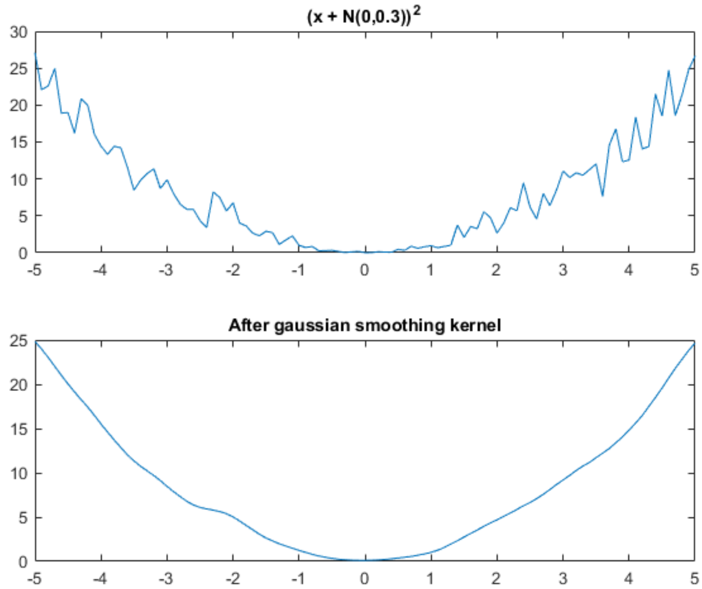

Figure 2.2: Example of Gaussian smoothing.

2.3.1.6 Gaussian Kernel

The Gaussian kernel is an isotropic kernel and defined as

$$K_\sigma(X_0, X) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(X_0 - X)^2}{2\sigma^2}\right). \tag{2.21}$$

It is a commonly used distance measurement kernel and is also known as the radial basis function kernel. This kernel is a Gaussian curve around X_0, so when used in kernel smoothing the points further away from X_0 contribute less [9].

2.3.1.7 Filter Size

Specifying a filter size for kernel smoothing limits the kernel to smooth the data using only a specific amount of neighbors. Given a size ϵ, the smoothing becomes

$$Y(t) = K * X(t) = \int_{|t-s|<\epsilon} K(t-s)X(s)ds. \tag{2.22}$$

The smoothing is therefore restricted to only neighboring points close enough, this is also known as window size [10].

2.3.1.8 Multidimensional Discrete Kernel

Applying these kernel methods to a lookup table requires turning the continuous nature of integrals into discrete sums, and the one dimensionality to three dimensionality. Define a three dimensional discrete dataset $D(x, y, z)$, a kernel filter size

$F = [s_1, s_2, s_3]$, corresponding to the number of neighbors for the smoothing kernel to average over in each dimension, and the standard deviation in each dimension, $\sigma = [\sigma_x, \sigma_y, \sigma_z]$. Additionally the Kernels are normalized over each respective dimension [9].

Then a data point (x, y, z) spreads out to a neighboring point (x_i, y_i, z_i) as

$$\begin{aligned} Y(x_i, y_i, z_i) = & K_{\sigma_x}((x, y, z), (x_i, y, z))K_{\sigma_y}((x, y, z), (x, y_i, z)) \\ & K_{\sigma_z}((x, y, z), (x, y, z_i))D(x, y, z), \\ & \mid x_i - x \mid < s_1, \mid y_i - y \mid < s_2, \mid z_i - z \mid < s_3, \end{aligned} \tag{2.23}$$

which for ease of notation can be rewritten to

$$F((x, y, z), (x_i, y_i, z_i)) * D(x, y, z), \tag{2.24}$$

the fraction multiplied by the data point. Now each point after smoothing is a super positioning of all the points within the filter size [11]. The total value for a new datapoint after smoothing can therefore be written as

$$Y(x, y, z) = \sum_{x_i = x - s_1}^{x + s_1} \sum_{y_i = y - s_2}^{y + s_2} \sum_{z_i = z - s_3}^{z + s_3} F((x, y, z), (x_i, y_i, z_i)) * D(x_i, y_i, z_i). \tag{2.25}$$

2.3.1.9 Moving Average

Comparing two time series can be difficult, as even a slight shift at certain points in the data can misalign them, leading to significant discrepancies. This problem requires a smoothing. Then a moving average can be used [5]. For a time series $x_1, x_2, ..., x_3$ and a window size k the moving average is

$$\bar{x}_i = \frac{\sum_{i-k}^{i+k} x_i}{2k + 1}. \tag{2.26}$$

This negates the problem of a few data points not being aligned perfectly.

2.3.2 Linear Regression

The relationship between an independent variable x and an dependent variable y is modeled using a linear function:

$$\hat{y} = ax + b + \epsilon, \tag{2.27}$$

where a and b, the slope and intercept, are the parameters to be estimated and ϵ is the random error which, in this paper, is set to zero [12]. The objective is to find the parameter values which minimize the prediction error

$$e = y - \hat{y} = y - (ax + b) \tag{2.28}$$

To introduce asymmetry in the penalty for over- and underestimation, the following weighted square error penalty function is used:

$$l = \begin{cases} e^2 \cdot w & \text{if } e > 0 \\ e^2 \cdot (1 - w) & \text{if } e \leq 0, \end{cases} \tag{2.29}$$

where $w \in (0, 1)$ is a weight parameter. For a data set of n samples $\{x_i, y_i\}_{i=1}^{n}$, the total penalty is computed as the sum of all individual penalties:

$$L(a, b) = \sum_{i=1}^{n} l_i. \tag{2.30}$$

When the weight parameter w is greater than 0.5, a positive error ($\hat{y} < y$) contributes more to the total penalty L than negative errors ($\hat{y} > y$) of the same magnitude. As a result, when L is minimized, the estimated parameters a^* and b^* will be biased towards overestimation in order to reduce the more heavily penalized underestimation.

2.3.3 L-BFGS-B Minimization

The L-BFGS-B Algorithm minimizes a nonlinear function

$$\min f(x) \tag{2.31}$$

subject to

$$l \leq x \leq u \tag{2.32}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ whose gradient g is available and l and u are the variables' lower and upper bounds for each component $x^i,\ i = 1, \ldots, n$. The L-BFGS-B algorithm is a quasi-Newton algorithm, meaning that it uses an approximation of the Hessian instead of the exact Hessian, like traditional Newton algorithms do. Quasi-Newton algorithms iteratively approximate the inverse of the Hessian matrix using an initial estimate and past computations [13][14]. Since computing and inverting the Hessian can be computationally expensive, quasi-Newton algorithms accelerate the process of obtaining the optimum by utilizing an approximation instead. Below follows a description of each step in the algorithm. This algorithm is based on the algorithm presented in [14] with a few alterations that are described in [15]. It is initialized with an initial guess x_0, an integer m that defines the maximum number of limited memory corrections that can be stored, and the initial limited memory Hessian approximation B_0 which is set to be the identity of dimensions $n \times n$. Beginning at $k \coloneqq 0$, the steps described below are completed each iteration.

2.3.3.1 Termination Criteria

There are two termination criteria that are checked at each iteration k. The first criterion terminates the algorithm if the objective function at the current iterate x_k, $f_k = f(x_k)$, has not changed enough from the previous iteration f_{k-1} and is formulated as:

$$\frac{f_{k-1} - f_k}{\max(|f_k|, |f_{k-1}|, 1)} \leq \textbf{factor} \cdot \textbf{epsmch}, \tag{2.33}$$

where **factor** is controlled by the user and **epsmch** is the machine precision. The second termination criterion is satisfied if the maximum change from moving in the direction of the negative gradient $g_k = g(x_k)$, within the bounds, is less than a user-defined value **pgtol**. This is formulated as

$$||P(x_k - g_k, l, u) - x_k||_\infty \leq \textbf{pgtol}, \tag{2.34}$$

where $P(x, l, u)$ is defined component-wise as

$$P(x, l, u)^i = \begin{cases} l^i & \text{if } x^i < l^i \\ x^i & \text{if } x^i \in [l^i, u^i] \\ u^i & \text{if } x^i > u^i, \end{cases} \tag{2.35}$$

where x^i is the i:th component of x.

2.3.3.2 Cauchy Point Calculation

The Cauchy point $\hat{x}$ serves as an initial approximation for computing the next iterate x_{k+1}. It is defined as the first local minimizer of the quadratic model of f at x_k:

$$m_k(x) = f_k + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k (x - x_k). \tag{2.36}$$

For the remainder of this section, subscript is no longer used to denote the current outer iteration and is instead used to denote the inner iterates during the search for the Cauchy point. The current iterate x_k is instead denoted as $\tilde{x}$. To find the Cauchy point $\hat{x}$, the breakpoints t^i, $i = 1, \ldots, n$, which determine when each component encounters a bound are computed:

$$t^i = \begin{cases} (\tilde{x}^i - l^i)/g^i & \text{if } g^i > 0 \\ (\tilde{x}^i - u^i)/g^i & \text{if } g^i < 0 \\ \infty & \text{otherwise.} \end{cases} \tag{2.37}$$

They are then sorted in increasing order to produce the ordered set $\{t_j : t_j \leq t_{j+1}, j = 1, \ldots, n\}$. For clarification, t^i denotes the i:th component's breakpoint and t_j is the j:th smallest breakpoint. Next, the algorithm proceeds to evaluate the function

$$x(t) = P(\tilde{x} - tg, l, u) \tag{2.38}$$

at the breakpoints. Using Equation 2.35 and Equation 2.37, Equation 2.38 can be defined component-wise as

$$x(t)^i = \begin{cases} \tilde{x}^i - tg^i & \text{if } t \leq t^i \\ \tilde{x}^i - t^i g^i & \text{otherwise.} \end{cases} \tag{2.39}$$

If $t > t^i$, the value $\tilde{x}^i - tg^i$ lands outside the allowed interval $[l^i, u^i]$. This is because t^i is defined as the maximum value t' for which the component $\tilde{x}^i - t'g^i$ remains within the bounds. In that case, t is replaced by t^i to ensure feasibility. The algorithm then examines one interval $[t_{j-1}, t_j]$ at a time, where the path is defined as

$$x(t) = x(t_{j-1}) + \Delta t d_{j-1}. \tag{2.40}$$

Here, $\Delta t = t - t_{j-1}$ represents the change in the parameter t within the current interval and d_{j-1} is the direction vector, defined component-wise as

$$d^i_{j-1} = \begin{cases} -g^i & \text{if } t_{j-1} < t^i \\ 0 & \text{otherwise.} \end{cases} \tag{2.41}$$

The direction vector ensures that components that reached a bound in the first term of Equation 2.40 are not affected by the second term, and that the components that did not reach a bound move along the steepest descent direction.

For the interval $[t_{j-1}, t_j]$ and using $z(t_{j-1}) = x(t_{j-1}) - \tilde{x}$, the quadratic model can be rewritten as:

$$\begin{aligned}
m(x(t)) =& f + g^T(x(t) - \tilde{x}) + \frac{1}{2}(x(t) - \tilde{x})^T B(x(t) - \tilde{x}) \\
=& f + g^T(x(t_{j-1}) + \Delta t d_{j-1} - \tilde{x}) \\
&+ \frac{1}{2}(x(t_{j-1}) + \Delta t d_{j-1} - \tilde{x})^T B(x(t_{j-1}) + \Delta t d_{j-1} - \tilde{x}) \\
=& f + g^T(z(t_{j-1}) + \Delta t d_{j-1}) \\
&+ \frac{1}{2}(z(t_{j-1}) + \Delta t d_{j-1})^T B(z(t_{j-1}) + \Delta t d_{j-1}) \\
=& f + g^T z(t_{j-1}) + \frac{1}{2}z(t_{j-1})^T B z(t_{j-1}) \\
&+ \Delta t(g^T d_{j-1} + \frac{1}{2}z(t_{j-1})^T B d_{j-1} + \frac{1}{2}d^T_{j-1} B z(t_{j-1})) \\
&+ \frac{1}{2}\Delta t^2 d^T_{j-1} B d_{j-1} \\
=& f + g^T z(t_{j-1}) + \frac{1}{2}z(t_{j-1})^T B z(t_{j-1}) \\
&+ \Delta t(g^T d_{j-1} + d^T_{j-1} B z(t_{j-1})) \\
&+ \frac{1}{2}\Delta t^2 d^T_{j-1} B d_{j-1}
\end{aligned} \tag{2.42}$$

Defining

$$\begin{aligned}
f_{j-1} &= f + g^T z(t_{j-1}) + \frac{1}{2}(z(t_{j-1}))^T B z(t_{j-1}), \\
f'_{j-1} &= g^T d_{j-1} + d^T_{j-1} B z(t_{j-1}), \\
f''_{j-1} &= d^T_{j-1} B d_{j-1},
\end{aligned} \tag{2.43}$$

the quadratic model can now be expressed as a function of Δt:

$$\hat{m}(\Delta t) = h + h'\Delta t + \frac{1}{2}h''\Delta t^2. \tag{2.44}$$

Since B is always positive definite [14], the critical value of $\hat{m}(\Delta t)$ is therefore a minimum. The critical point is obtained by differentiating $\hat{m}(\Delta t)$, equating the result to zero and solving for Δt. After completing this calculation, the solution $\Delta t^* = -\frac{h'}{h''}$ is obtained. If $t_{j-1} + \Delta t$ lies in the interval $[t_{j-1}, t_j)$, the Cauchy point is at $\hat{x} = x(t_{j-1} + \Delta t^*)$. If it does not, the sign of f'_{j-1} is examined:

- If $f'_{j-1} \geq 0$, the Cauchy point is at $\hat{x} = x(t_{j-1})$.
- If $f'_{j-1} < 0$, the Cauchy point is at $x(t_j)$ or a point further away. f_{j-1}, f'_{j-1} and f''_{j-1} are then updated according to the next interval $[t_j, t_{j+1}]$ and the calculations are repeated.

2.3.3.3 Subspace Minimization

When the Cauchy point $\hat{x}$ has been obtained, the active set $\mathcal{A}(\hat{x})$ is defined as the set of the variables whose value are either at l^i or u^i at $\hat{x}$, and $\mathcal{F}$ denotes the index set of variables that are not at their bounds at the Cauchy point and are therefore free to move. The variables in $\mathcal{A}(\hat{x})$ are held fixed, and the quadratic model is minimized over the subspace of the free variables. This is formulated as

$$\min \{m_k(x) : x^i = \hat{x}^i, \ \forall i \in \mathcal{A}(\hat{x})\} \tag{2.45}$$

$$\text{subject to} \quad l^i \leq x^i \leq u^i, \ \forall i \notin \mathcal{A}(\hat{x}). \tag{2.46}$$

Let p be the number of free variables at $\hat{x}$ and define Z_k to be a $n \times p$ matrix that spans the subspace of the free variables at the Cauchy point, its columns being unit vectors. The vector $v \in \mathbb{R}^p$ represents the optimal step, within the subspace of the free variables, from the Cauchy point $\hat{x}$ that minimizes the quadratic model m_k. It is computed by solving a reduced linear system involving the inverse of the reduced Hessian $\hat{B}_k = Z_k^T B_k Z_k$, which is is obtained using the Sherman-Morrison-Woodbury formula. This vector can land the free variables outside of the feasible region, i.e. $\hat{x}^i + (Z_k v)^i \notin [l^i, u^i]$ for some $i \in \mathcal{F}$. A feasible solution can then be obtained by computing

$$v^* = \alpha^* v, \tag{2.47}$$

where

$$\alpha^* = \max\{\alpha : \alpha \leq 1, \ l^i - \hat{x}^i \leq \alpha v^i \leq u^i - \hat{x}^i, \ i \in \mathcal{F}\}. \tag{2.48}$$

The approximate solution to Equation 2.45 is then calculated:

$$\bar{x}^i_{k+1} = \begin{cases} \hat{x}^i & \text{if } \ i \notin \mathcal{F} \\ \hat{x}^i + (Z_k v^*)^i & \text{if } i \in \mathcal{F}. \end{cases} \tag{2.49}$$

.

2.3.3.4 Computation of the New Iterate

To obtain the next iterate, a line search is performed along $d_k = \bar{x}_{k+1} - x_k$. The goal is to find a step size λ_k such that the next iterate

$$x_{k+1} = x_k + d_k \lambda_k \tag{2.50}$$

satisfies the condition

$$f(x_{k+1}) \leq f(x_k) + \alpha \lambda_k g_k^T d_k \tag{2.51}$$

and, if possible, satisfies

$$|g_{k+1}^T d_k| \leq \beta |g_k^T d_k|, \tag{2.52}$$

where $\alpha = 10^{-4}$ and $\beta = 0.9$ are typical values. These conditions ensure that the objective function value and gradient decrease sufficiently between the current iterate x_k and the new iterate x_{k+1}.

2.3.3.5 Updating the Limited Memory Hessian Approximation

After x_{k+1} is obtained, the gradient $g_{k+1} = \nabla f(x_{k+1})$ is computed along with $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. If the condition

$$\frac{y_k^T s_k}{-g_k^T s_k} > \mathbf{epsmch}, \tag{2.53}$$

is satisfied, s_k and y_k are added to the correction matrices $Y_k = [y_{k-m}, \ldots, y_{k-1}]$ and $S_k = [s_{k-m}, \ldots, s_{k-1}]$, respectively. Furthermore, if there are already m columns stored in S_k and Y_k, the oldest column from each matrix is removed so that only the m most recent pairs are stored. The limited memory Hessian approximation matrix is then updated using the limited memory BFGS formula:

$$B_{k+1} = \theta I - W_k M_k W_k^T, \tag{2.54}$$

where

$$\theta = \frac{y_k^T y_k}{y_k^T s_k}, \tag{2.55}$$

$$W_k = \begin{bmatrix} Y_k & \theta S_k \end{bmatrix}, \tag{2.56}$$

$$M_k = \begin{bmatrix} -D_k & L_k^T \\ L_k & \theta S_k^T S_k \end{bmatrix}^{-1}, \tag{2.57}$$

$$(L_k)_{i,j} = \begin{cases} s_{k-m-1+i}^T y_{k-m-1+i} & \text{if } i > j \\ 0 & \text{otherwise,} \end{cases} \tag{2.58}$$

$$D_k = diag\left[s_{k-m}^T y_{k-m}, \ldots, s_{k-1}^T y_{k-1}\right]. \tag{2.59}$$

The algorithm then moves on to the next iteration and starts over again, beginning with checking the termination criteria defined in Section 2.3.3.1.

2.3.3.6 Algorithm Summary

To initialize the algorithm, an initial guess x_0 is selected along with an integer m that defines the maximum number of limited memory corrections that can be stored. The initial limited memory Hessian approximation B_0 is defined to be the identity matrix of dimensions $n \times n$. Start at $k := 0$ and iterate through the following steps:

1. Terminate the algorithm if either of the conditions described in Equation 2.33 and Equation 2.34 is satisfied.

2. Find the first local minimizer, the Cauchy point $\hat{x}$, of the quadratic model

$$m_k(x) = f(x_k) + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k(x - x_k), \tag{2.60}$$

along the path that is the steepest descent direction projected onto the feasible region, meaning the region where $x \in [l, u]$.

3. Minimize $m_k(x)$ using a direct primal method, focusing only on the free variables (the coordinates not at their bounds, as determined by the Cauchy point). The coordinates that are at their bounds, also determined by the Cauchy point, are held fixed. The minimization starts at the Cauchy point $\hat{x}$, which is a feasible point, and is carried out in the subspace of the free variables. This minimization results in the finding of an approximate solution $\bar{x}_{k+1}$.

4. When $\bar{x}_{k+1}$ has been found, a line search along $d_k = \bar{x}_{k+1} - x_k$, considering the variable bounds, is performed. This is done to enforce the condition described in Equation 2.51 and, if possible, the condition defined in 2.52. The next iterate $x_{k+1} = x_k + \lambda_k d_k$ is then computed.

5. Compute a new limited memory Hessian approximation B_{k+1} using the L-BFGS formula.

6. Set $k \coloneqq k + 1$ and start over from Step 1.

3

Methods

3.1 Isolating the Component

The full Simulink model consists of multiple smaller models communicating with each other: the environment component, the driver component and the truck component. The driver component "controls" the truck component in a situation defined by the environment component. Combined, these components emulate a truck operator driving specific truck in a particular environment as closely as possible. In this project, a subcomponent of the truck component was isolated and analyzed. This was done for two main reasons, the first being that the information concerning the environment and driver is complex as well as difficult to model and compare. Two driving sessions by the same driver in the same truck and route, done a few hours apart can have different environmental conditions such as rolling resistance or temperature. Furthermore, drivers can never operate the vehicle exactly the same way twice which introduces more uncertainties. By isolating this component and modeling it with logged data as input and comparing its output to the corresponding logged output data, the true modeling performance can be analyzed without exterior effects. Secondly, simulating the full process of operating a vehicle within a specific environment is computationally expensive. There are a multitude of components that perform calculations and communicate with each other, making the simulation both time consuming and resource intensive. By simulating just one component, the process can be simplified and the required computational resources can be reduced. Similarly, isolating part of the model reduces the number of input and output data that has to be processed, decreasing the computer's workload.

Both of these problems were solved by isolating a component, and modeling it with input signals from logged data. The components that constitute the full model were available as individual Simulink models, so a new Simulink model was created with two copies of the chosen component to represent the actual vehicle configuration. By creating this dual system, the component copies can be evaluated individually and simulated simultaneously (from hereon, *the model* will refer to this twin setup).

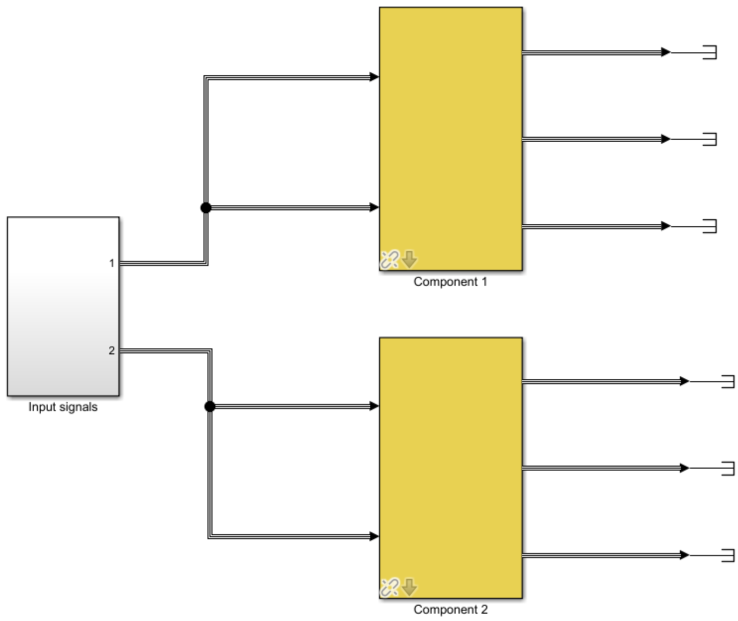

Figure 3.1: The twin setup with inputs to the model on the left and the two component copies on the right.

3.2 Data Processing

The input signals that are used by the the model were gathered from real-world driving events, where data is collected using a multitude of different sensors. Sensor data is gathered and processed from multiple trucks and stored in a database as driving session logs. In order to evaluate the model's performance for a single log, the specific input signals used by the model were extracted from the log, the model was simulated with the input signals, and the output signals were collected. The model output was then evaluated against the corresponding signals from the log. This section illustrates in detail each stage the data from one log goes through in the pipeline from database to model evaluation. The vehicles whose data is used in this project have multiple instances of this component and therefore separate sensor data was collected for each component. Each stage described in this section processes the data from the two components in parallel.

3.2.1 Filtering and Transforming the Data

The first step after fetching the logs from the database was to filter the logs that had faulty data. Data that had missing signals were filtered as well as data which had constant signal values, indicating faulty sensors. The sensor data from the driving sessions was not congruent with the format the model requires, so the signals were transformed so they could be used as inputs in the following ways:

- The signals were converted from the logged units to the model's required input units.

- Signals were converted from reference and percentage values to SI units.
- Vehicle-specific signal formats were transformed to a standardized format.
- The recorded time data was transformed from date-time format to float values.
- Any value exceeding a specified signal threshold was clipped to that threshold.

3.2.2 Splitting the Data

After the filter and transformation stages, the data was split into two halves. The first half consisted of the signals that the model requires as input. This data was converted into a `mat` file which contained a Matlab structure array with a field for each input signal. Each field was a $n \times 2$ double, where n was the number of time steps. The first column contained the time data and the second column contained the data for the given input signal. The second half of the split data contained the signals which corresponded to the output signals of the model. This half of the data was converted into a `parquet` file for its efficiency with large-scale data.

3.3 Simulation

The model received the input signals it requires by loading the `mat` file containing the previously mentioned signals into Matlab, and then using the `createInputDataset` function in tandem with the `setExternalInput` function. The former generates a dataset object and the latter directs that dataset object to a model. This process handled most of the inputs, but some of the model's inputs did not have corresponding signals in the logged data. The values of these input signals were decided upon after discussion with model experts.

3.4 Model Evaluation

After simulation, the output signals were converted into a `parquet` file so they could be compared to the corresponding signals from the log. By quantifying the similarity between these signals for multiple logs, the model's performance could be evaluated for each truck and component. For most of the signals, a minimized error is desired. In these cases, mean absolute error was used as the similarity metric. A drawback of using the mean absolute error is that it is difficult to know what value indicates good performance; a mean absolute error of one is good if the signal values are in the order of hundreds, but it indicates poor model performance if the signal values are fractions. In this project, the signals were not compared to each other, they were instead compared to themselves before and after the model optimization. The value of the error is therefore not as interesting as the degree to which it is changed after optimization, making MAE a suitable metric.

It is beneficial for some signals if the model slightly overestimates the different output signals, so that the digital twin degrades faster than its real-world equivalent and the at-risk parts can be fixed or replaced before they fail. For these signals, the mean absolute error is not optimal as it does not take into account if the logged or simulated signal has higher values. The metric TAR, introduced in Section 2.2, was therefore used as a model evaluation metric because measures how often a simulated signal exceeds the value of the corresponding logged signal.

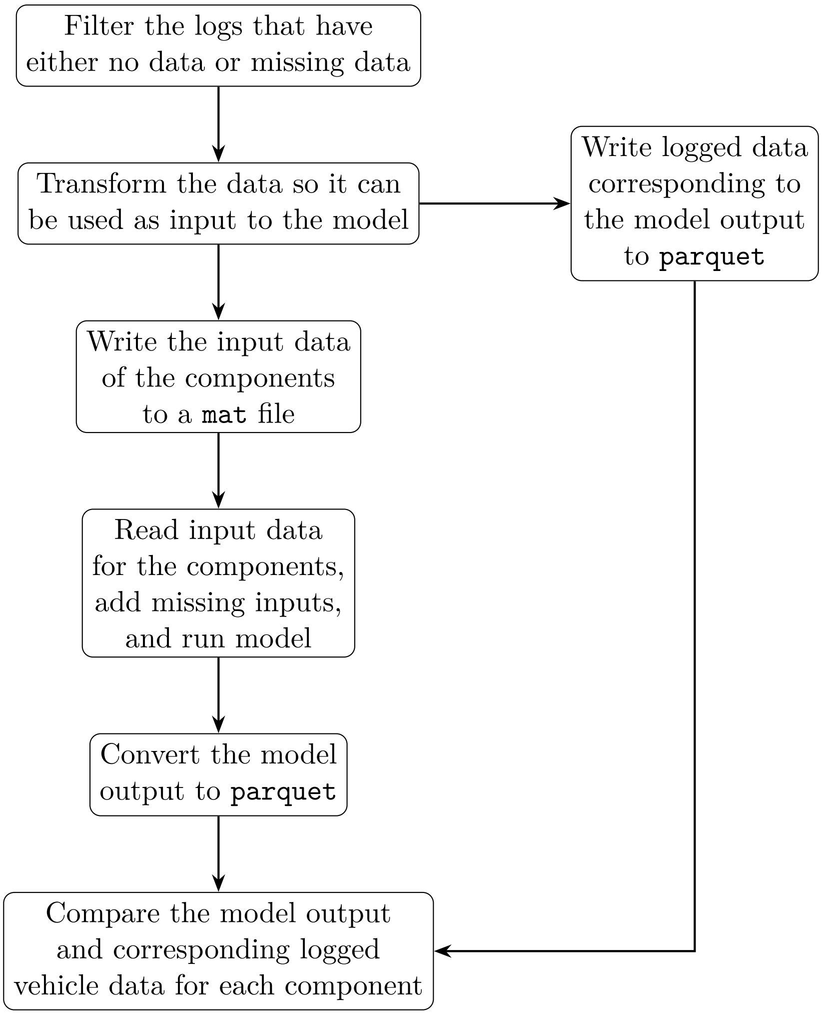

Figure 3.2: Flowchart of how the data is handled from the filtering stage to model evaluation.

3.5 Parameter Optimization

3.5.1 Linear Regression on Output Signal

Linear regression was employed to transform a model output signal to better align with the corresponding signal from the logged data. The linear regression was conducted independently on the two components, allowing each component to have its own scaling factor (slope) and constant offset (intercept) parameters for transforming the output signals.

3.5.1.1 Data Processing

The logged signal and corresponding simulated signal from each available log was collected. The model computes output values at a higher frequency than the sensors collect data. The signals were aligned by discarding the values from the simulated data that did not have a corresponding value in the logged data and by truncating the longer signal to the length of the shorter signal. The first 50 data points were then removed from each signal to avoid potential abnormal behavior in the beginning of the signals. When each logged signal had been processed, they were collected and stitched together into one long signal, and the processed simulated signals were collected and stitched together into another combined signal.

3.5.1.2 Minimization Method Selection

The Python library SciPy, which was used for optimizing the linear regression, has multiple minimization methods available [16]. To select the most appropriate method for this application a comparison test was conducted. The process described in this section was applied to all vehicles, but for simplicity, the section will only describe the process for a generic vehicle. The combined simulated data was divided into a train set, $\mathbf{x}_{\text{train}}$, and a test set, $\mathbf{x}_{\text{test}}$, with an 80-20 split. The combined log data was likewise divided into a train set and test set, $\mathbf{y}_{\text{train}}$ and $\mathbf{y}_{\text{test}}$. The SciPy library's function `minimize` was then used one minimization method at a time with $\mathbf{x}_{\text{train}}$ as the independent variable, $\mathbf{y}_{\text{train}}$ as the dependent variable, the initial guess $\{a = 1,\ b = 0\}$ and the penalty function described in Equation 2.30 with $w = 0.5$. If the method allowed for it, the following parameters were also applied: an enforced bound on a in the range $[0.95, 1.05]$, the Jacobian of the penalty function and the Hessian of penalty function.

The optimal parameters a^* and b^* returned by `minimize`, were then used to transform $\mathbf{x}_{\text{test}}$:

$$\hat{\mathbf{y}}_{\text{test}} = a^* \mathbf{x}_{\text{test}} + b^*. \tag{3.1}$$

The mean absolute error and the maximum error between $\hat{\mathbf{y}}_{\text{test}}$ and $\mathbf{y}_{\text{test}}$, the mean of $\hat{\mathbf{y}}_{\text{test}} - \mathbf{y}_{\text{test}}$ and the computation duration for each minimization method were then computed for comparison between the different methods. A scaling factor close to one was desirable, therefore the optimal factor a^* for each method was also recorded for comparison between the methods.

The methods were first filtered by keeping the methods whose optimal scaling factor was close enough to one, which was defined as the interval $[0.95, 1.05]$. The remaining methods were then compared using the previously mentioned metrics to select the most appropriate method. Most of the methods performed similarly across mean absolute error, mean difference and max error, so L-BFGS-B was selected due to its low computation time.

3.5.1.3 Finding the Optimal Weight

To find the optimal weight w^* for the penalty function defined in Equation 2.30, the combined simulated data was first split into a training set ($\mathbf{x}_{\text{train}}$), a validation set ($\mathbf{x}_{\text{val}}$) and test set ($\mathbf{x}_{\text{test}}$) with 60-20-20 splits. The combined real data was likewise divided into a training set ($\mathbf{y}_{\text{train}}$), a validation set ($\mathbf{y}_{\text{val}}$) and test set ($\mathbf{y}_{\text{test}}$). Linear regression was performed using the L-BFGS-B algorithm to minimize the penalty function, as detailed in Section 2.3.2. This process was repeated for 50 different weight values, $\{w^j : j = 1, \ldots, 50\}$, ranging from 0.5 to 0.99 with an interval of 0.01, where $\mathbf{x}_{\text{train}}$ comprised the independent variables and $\mathbf{y}_{\text{train}}$ comprised the dependent variables. For each w^j's optimal parameters a^j and b^j, the validation set of the signal was transformed:

$$\hat{\mathbf{y}}^j_{\text{val}} = a^j \mathbf{x}_{\text{val}} + b^j. \tag{3.2}$$

The mean absolute error between $\mathbf{y}_{\text{val}}$ and $\hat{\mathbf{y}}^j_{\text{val}}$ as well as the mean of $\hat{\mathbf{y}}^j_{\text{val}} - \mathbf{y}_{\text{val}}$ were then calculated for each w^j. Next, an algorithm was employed to find the weight value which best balances overestimation (relatively high mean) and good performance (low MAE). In Algorithm 1, "mae_list" is a list of the MAE values that were computed for each w^j, ordered by the corresponding w-values in ascending order; "mean_list" is a list of all mean values, ordered in the same way; "weight_list" is a sorted list of the w-values; "original_mae" is the MAE value before the transformation of $\mathbf{x}_{\text{val}}$, i.e. the MAE between $\mathbf{x}_{\text{val}}$ and $\mathbf{y}_{\text{val}}$; "original_mean" is the mean value of $\mathbf{x}_{\text{val}} - \mathbf{y}_{\text{val}}$; $\alpha \in (0, 1)$ is a parameter.

In this case, a slight overestimation of the real data is preferred so the model makes conservative predictions. The ideal outcome would therefore be to increase the mean difference, indicating higher values in the simulated data, while decreasing (or maintaining) the MAE. The algorithm therefore penalizes w-values whose MAE increased or whose MD decreased compared to before regression. The parameter α biases the score calculation towards lower MAE or higher MD. The most appropriate value was selected by comparing the outcomes of different α-values, and selecting the value which maximized the mean difference while making sure that the new mean absolute error was not greater than the mean absolute error before optimization. The value that was eventually chosen was $\alpha = 0.8$ because it resulted in a weight parameter selection that resulted in the desired outcome describe above, see Figure 3.3 for an example of how α affects the optimal weight selection of Algorithm 1.

When α had been selected, the corresponding optimal weight w^*'s optimal scaling factor a^* and optimal constant offset b^* were used to transform the simulated test data:

$$\hat{\mathbf{y}}_{\text{test}} = a^* \mathbf{x}_{\text{test}} + b^*. \tag{3.3}$$

Algorithm 1 Optimal Weight Algorithm

```
Input: mae_list, mean_list, weight_list, original_mae, original_mean, α
Output: optimal_weight
scores ← []
n← length(weight_list)
for i ← 1 to n do
    mae_diff ← mae_list[i]-original_mae
    mean_diff ← mean_list[i]-original_mean
    if mae_diff > 0 then
        mae_penalty ← mae_diff · 10
    else
        mae_penalty ← mae_diff
    end if
    if mean_diff < 0 then
        mean_penalty ← mean_diff · 10
    else
        mean_penalty ← mean_diff
    end if
    score ← α · mae_penalty − (1 − α) · mean_penalty
    Append score to scores
end for
optimal_idx ←argmin(scores)
optimal_weight ← weight_list[optimal_idx]
```

This data was then compared to the real test data $\mathbf{y}_{\text{test}}$.

3.5.1.4 Creation of Aggregated Dataset

In addition to the real trucks that were compared, an artificial truck dataset was constructed by combining the different splits from the real trucks. Its training set, validation set and test set comprised the combined corresponding data sets from all of the real trucks. This aggregated truck was evaluated the same way as the other trucks to measure the performance across all the trucks.

3.5.2 Optimization of Lookup Table

Optimization of the lookup tables means correcting the value it outputs according to what the real logged data is. For each instance in the time series where indices and fractions are provided to produce a value, that value should be as close as possible to the real data. The inputs to the table are three signals derived from the simulated truck. Since every instance's output is based on eight values from the lookup table, each one of them has to get its own correction based on the errors. This is achieved by a gradient-based algorithm, and in order to keep the table smooth, a Gaussian smoothing and final correction.

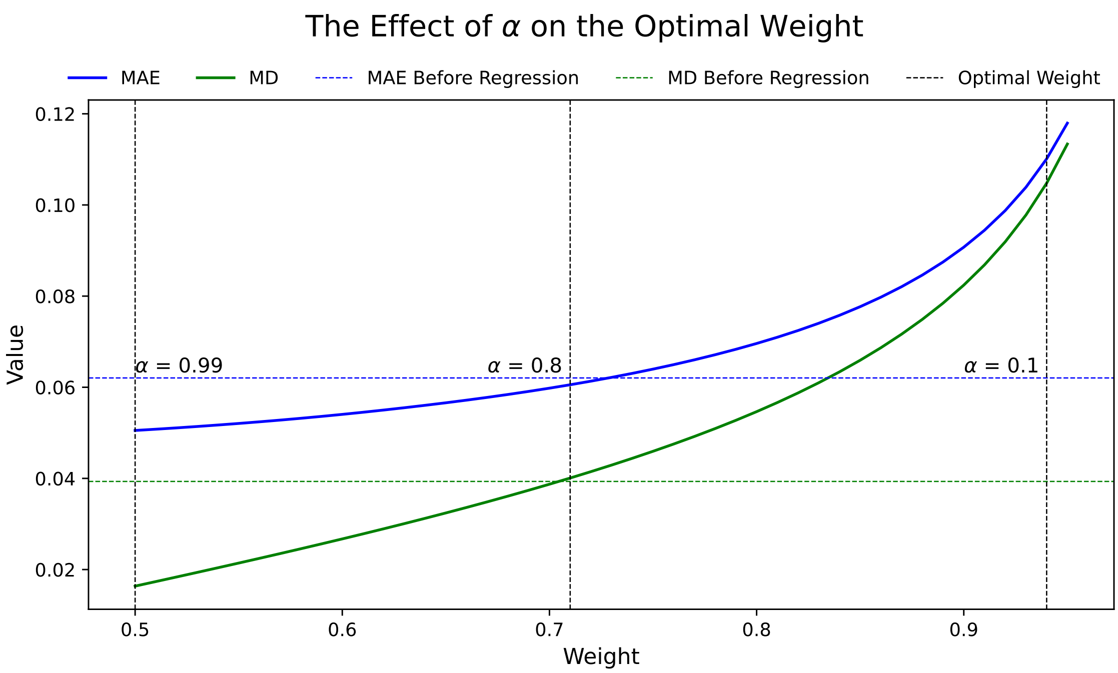

Figure 3.3: Figure showing how different values on the α parameter affects the choice of the optimal weight w that is used in the penalty function defined in Equation 2.30. The data in the figure is from the validation set of Vehicle 8 and the last five w-values have been removed to make the figure more interpretable.

3.5.2.1 Additional Data Collection

In order for an algorithm to optimize the table it is essential to know the source of each value within the table. Therefore a new output was recorded from the model. For each time step, the indices and fractions, along with the corresponding output were recorded. With this information and the logged data, all the information about the error, how and from where it was created could be found. The optimization could therefore take place outside of the simulation model, where only the lookup table was manually calculated instead of running the entire simulation.

3.5.2.2 Overview

In order to get an overview of the algorithm to understand where each step fits in, the outline will be presented. For each route and each timestep, the interpolated value was calculated from the indices and fractions. The difference between this interpolated value and the logged value was then computed. A portion of the gradient was attributed to the corresponding indices, and the gradient was smoothed using a Gaussian kernel. The lookup table was updated according to the Adam update rule, followed by a minor correction. The data for each truck were divided into training, validation, and test sets, at 70-20-10 splits.

Algorithm 2 Lookup Table Optimization Algorithm

Input: routes with fractions and indices, lookuptable, logged data, hyperparameters
Output: Optimized lookuptable
for iteration **do**
  for route **do**
    outputs ← interpolation(indices, fractions)
    errors ← outputs - logged data
    route gradient ← gradient(error, indices, fractions)
    total gradient ← total gradient + Gaussian kernel(route gradient)
  end for
  lookup table ← lookup table - Adam update rule(total gradient)
  lookup table ← correction(lookup table)
end for

3.5.2.3 Gradient

For a single route of data, the output values of the table were initially calculated as the trilinear interpolation explained in section 2.3.1. Due to the high frequency of the data and the nature of the data collection process, the real data and simulated data will not always line up perfectly. Therefore, firstly the absolute largest mistimings were removed as they were seen as outliers that would heavily skew the gradient, so instances where the error was greater than a threshold was removed. For the rest a simple moving mean of the interpolated value and logged data was used.

This counteracts the potential major errors from a slight mistiming between them, with the gradient descent and Gaussian smoothing the errors still gets attributed to the correct areas where the inaccuracies are.

The error for each instance was then calculated as the difference between the simulated and logged data. With the error at each instance along with the fractions and indices of the matrix, the total gradient for each route was calculated. For stability the gradient for each route was divided by the length of the route.

3.5.2.4 Kernel smoothing

For a lookup table, some areas will be used more frequently than others. If a point constantly returns a value that is faulty, it is reasonable to assume that the surrounding values may also be similarly inaccurate and require correction. Due to the discrete nature of the lookup table, the optimization may only reach certain indices of the matrix. In order for the optimization to generalize to new routes that uses other parts of the lookup table the correction needs to propagate to surrounding indices. This is done with kernel smoothing. When the gradient for a route is computed a Gaussian kernel smoothing is applied to the calculated gradients. This helps to distribute the correction not just to the specific index that was identified as faulty, but also to its neighboring indices, thereby ensuring that the adjustments are more comprehensive.

3.5.2.5 Updating

After aggregating the gradients from all routes, the lookup table was updated with the Adam algorithm. This update rule was chosen for multiple reasons. It is suitable for high-dimensional and sparse gradients, the gradient for a small table with 20 indices for each dimension would effectively be updating $20^3 = 8000$ parameters, out of which many will rarely be traversed. Adam also requires less hyperparameter tuning, which is advantageous for our purposes. The chosen values were $\beta_1 = 0.9, \beta_2 = 0.999$, as these were recommended defaults by the algorithm's creators.

3.5.2.6 Correction

Taking the physical laws into account, the lookup table's output has to increase with an increase in dimension two. To make sure the table still follows this, a correction of the table was made. By analyzing the difference between adjacent points in the second dimension, it should correlate with the change in amplitude. This fault can happen if a certain point in the data is not reached by the updating, while points close to it are. Therefore if such a fault was detected between two neighbors after updating, half of the difference was added to one point, while half was subtracted from the other.

3.5.2.7 Early Stopping

Early stopping was used to avoid overfitting. After each iteration, the lookup table was used with the validation set. The total absolute error for the set was calculated, and the minimum over the whole training is recorded. In the event that the validation error is worse than this minimum in three iterations in a row, the training is stopped and the lookup table matrix that produced the minimal validation error is used as the final product. An additional stopping criteria is when the algorithm corrects too much, which would indicate that the updating is trying to fit to unreasonable data and constantly need correction. Therefore in the case where the correction is a substantial fraction of the size of the update the algorithm breaks. Since the corrections should be minor changes, if the correction is more than a few percent of the gradient update it should stop. This was applied as the mean of the correction divided by the mean of the update, with the limit set at five percent.

3.5.2.8 Hyperparameters

As Adam has an effective adaptive learning rate for each parameter that is being optimized, the learning rate was set to 0.05, which is slightly higher than the default. This was because of the large dataset and the initially substantial corrections needed. The Gaussian filters that spread the the correction to neighbors needs tuning. The filter size was set to be $F = [5, 15, 3]$ which corresponds to around one fourth of the size for the first two dimensions. These were chosen in order to have the capacity to affect neighboring points but not so extended that they change the whole table. The tuning was further done for the standard deviation for each dimension $\sigma = [\sigma_x, \sigma_y \sigma_z]$. A kernel with a high standard deviation is preferred when a stronger smoothing effect is desired, as it influences neighboring points more significantly. A kernel with a low standard deviation is suitable when a more localized smoothing effect is preferred. The objective function is the error after training the whole model, therefore no gradient can be calculated and the evaluation itself is computationally expensive. Pattern search was used for this hyperparameter optimization for these reasons. When performing the pattern search the data was downsampled in order to lessen the computations needed without completely removing any routes that might contain specific data.

4

Results

4.1 Lookup Table Optimization

This section will cover the optimization method of the lookup table. Firstly, the training phase will be presented, then the effective improvement, and finally how that improvement is reflected in changes in the table. As the table has a three dimensional input and a one dimensional output, illustrating all the changes is difficult. Therefore the figures focus on one page of the lookup table; a 2D slice of the matrix obtained by fixing the third dimension to a set index. The remaining two dimensions are displayed as the different plots and the x-axis of these plots respectively, with the output value of the table on the y-axis.

4.1.1 Hyperparameters

The hyperparameter optimization for the standard deviation of the filter size was performed on vehicle 1. The pattern search algorithm found the combination $\sigma = [0.2344, 1.3438, 0.2250]$ to be optimal for this data set.

Figure 4.1 shows the progression of the hyperparameter optimization process, where the cost function decreases quickly in the initial steps, and stabilizes after a few iterations. After 16 iterations the pattern search algorithm reached the termination criterion. With these hyperparameters, the optimization of the lookup table for Vehicle 1 was performed. For reference, a filter with the shape $[1, 1, 1]$ gave a relative function value of 1.1587.

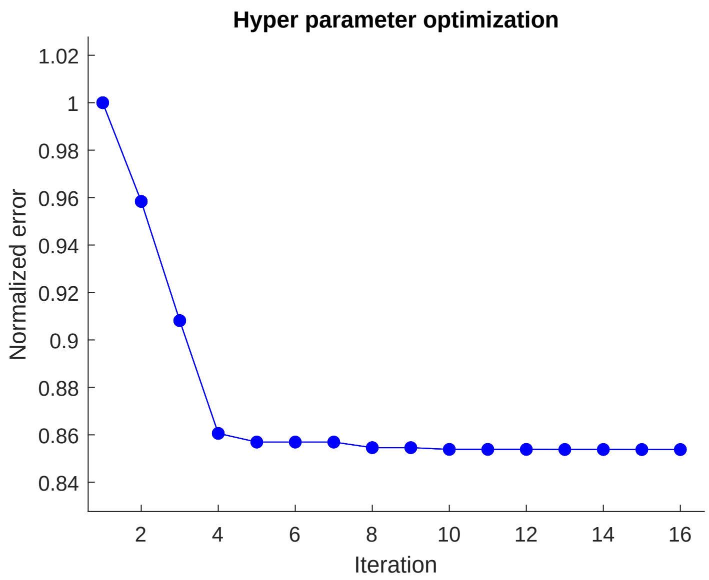

Figure 4.1: Hyperparameter optimization error over iterations, performed on Vehicle 1 Component 1.

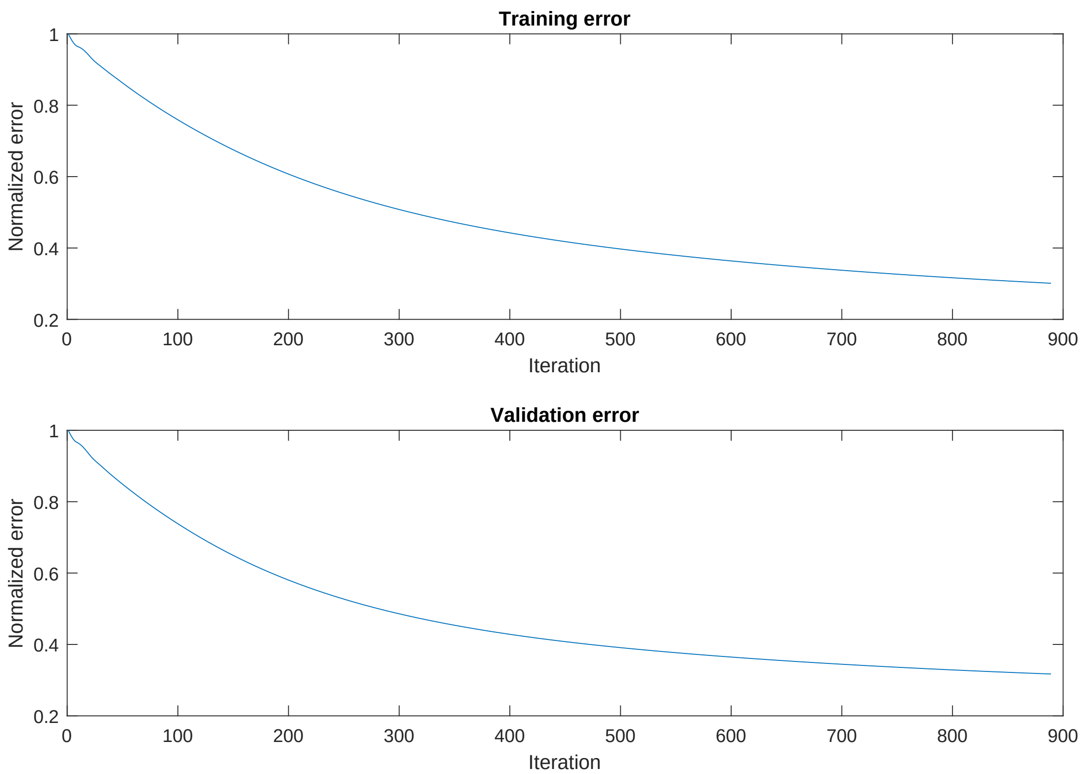

Figure 4.2: Training and validation error during training on Vehicle 1 Component 1.

Figure 4.2 illustrates the error progression for both the training and validation data sets during the model training phase. Initially both data sets experience a significant improvement in error, particularly during the first 400 iterations, indicating a training of the lookup table that corrects the table. As it progresses the error decrease stagnates, with a slightly steeper curve continuing for the training error.

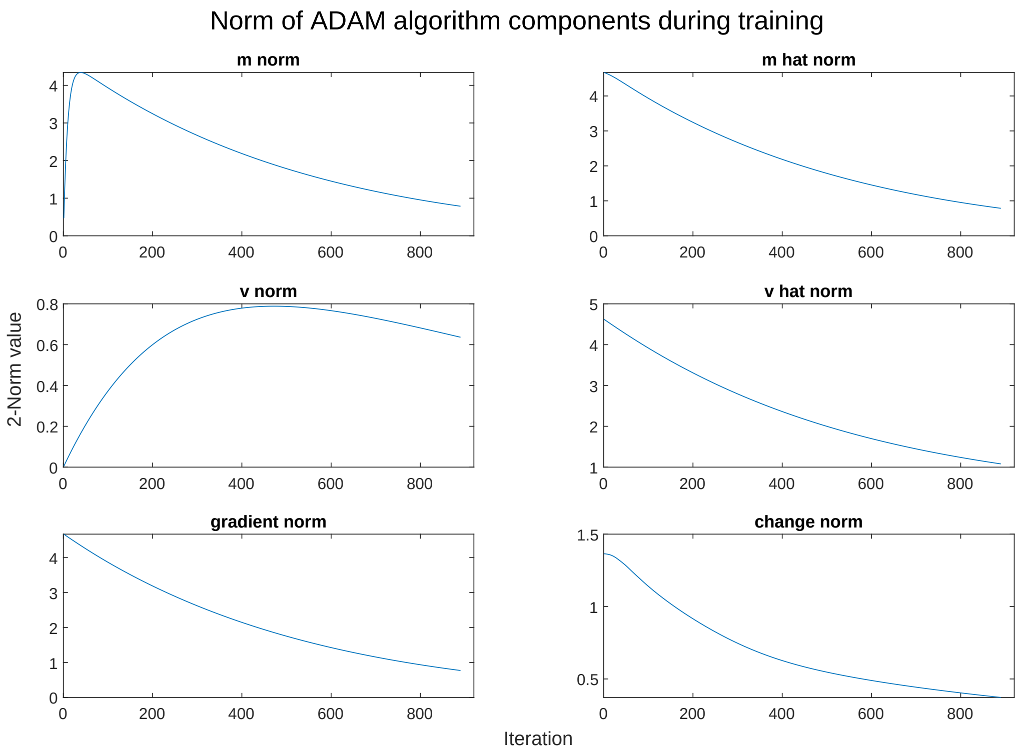

Figure 4.3: Norms of Adam components during training for one page of the table for Vehicle 1 Component 1.

In Figure 4.3 it is illustrated how the norm of a page in the components within the Adam optimization algorithm changes. At the start, both m and v are small, and the bias correction adjusts them to prevent any bias towards zero. The magnitude of the bias corrected norms exhibits a notable decrease, reflecting the trend observed in the error curves. They are reduced to approximately 15% of their initial value, indicating a substantial decrease in the size of the updates to the model parameters. The norm of the change between iterations show that as the training proceeds, the updates decrease in size, indicating that the model is adapting to the data.

Table 4.1: Summary of MAE of test set for vehicles and components.

Data set	MAE before	MAE after	Improvement
Vehicle 1 Component 1	0.8587	0.3245	0.3779
Vehicle 1 Component 2	1.0000	0.3604	0.3604
Vehicle 2 Component 1	0.8978	0.3115	0.3470
Vehicle 2 Component 2	0.7728	0.2895	0.3746

Table 4.1 provides a summary of the Mean Absolute Error for the test sets associated with different vehicles and their respective components, both before and after the optimization process. Vehicle 1 Component 1 exhibits an MAE of 0.8587, which is compared to Component 2 that shows a slightly higher MAE. This suggests that Component 1 is more accurately predicted compared to Component 2, indicating potential areas for further optimization in the latter. The reverse relation can be seen between the components in Vehicle 2. As seen in the change of MAE, the largest improvement is observed for the initially worse components.

4.1.1.1 Refined Tables

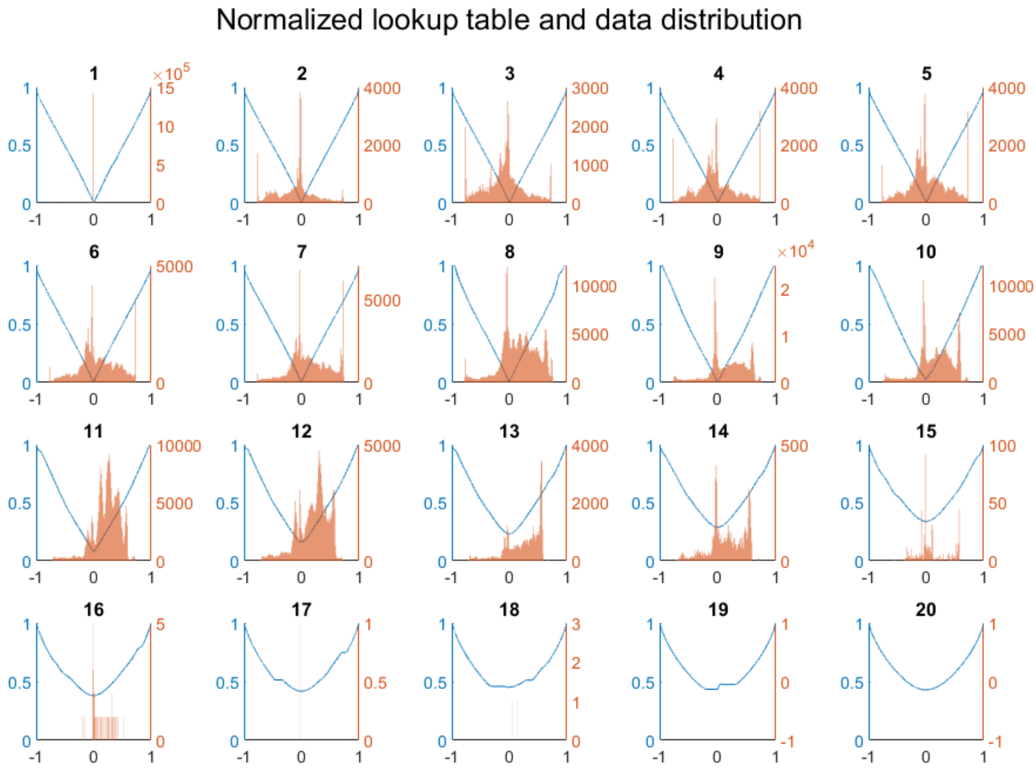

Figure 4.4: Optimized table for Vehicle 1 Component 1 and the distribution of the training data.

Figure 4.4 presents a detailed view of one page of the lookup table following the optimization algorithm, along with the distribution of training data. The optimized table retains a shape largely similar to the original, although changes are notable. Certain areas along the sides exhibit rougher characteristics, showing localized adjustments made during optimization. The training data is centered around the middle of the second dimension, but with an inclination to be skewed towards the right. The data is similarly centered in the middle of the first dimension. This is clearly seen in the later plots, specifically plot 15 and 16, where plot 15 shows hundreds of data points, plot 16 tens of data points and only a few data points are seen in the last plots. The shape of the last plots are dissimilar from the others, as notable plateau formations can be seen.

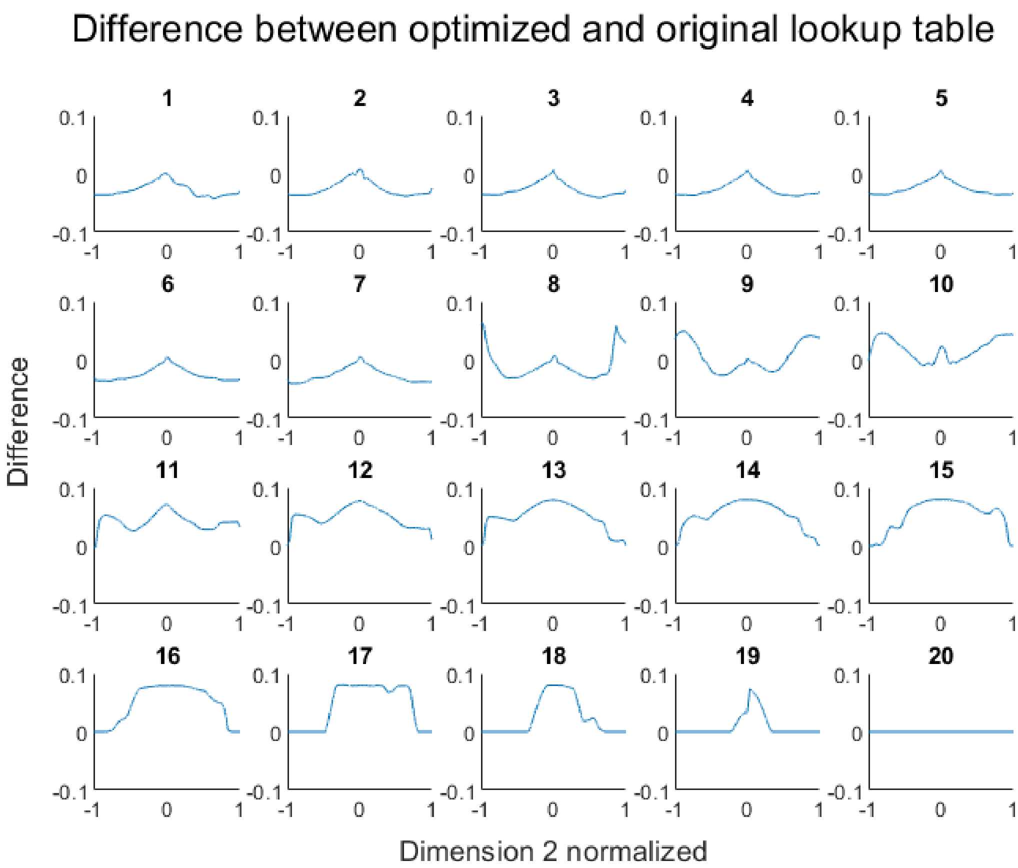

Figure 4.5: Difference in the lookup table of Vehicle 1 Component 1 before and after optimization.

The difference, presented in Figure 4.5, follows a pattern and a smooth change over the first dimension. The first seven plots have a homogeneous shape, where the values around the middle change less compared to those further out, where the original table overestimated what the output should be. From plot 7 to 8 there is a clear difference. The middle of the plots are similar but about halfway out in both directions of the x-axis, the updated table shows that the original table underestimated the new values. Then, moving through dimension one, the shape is somewhat continuously changed from an "W" to an upside-down "U" as the values in the middle are subsequently larger in the optimized table.

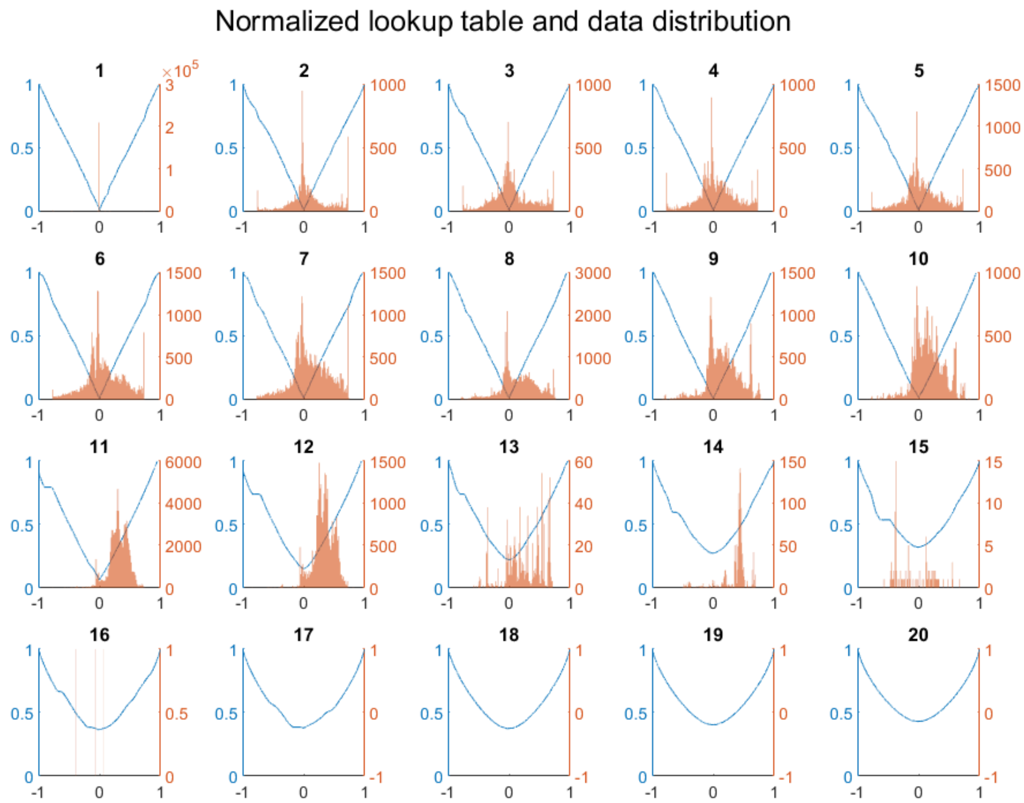

Figure 4.6: Optimized table for Vehicle 2 Component 1 and the distribution of the training data.

The plots in Figure 4.6 show the same optimization for Vehicle 2. The optimized table looks generally similar to the one in Figure 4.4. A notable difference is the amount of data and the shape of the table for the values on the left in dimension two. There, the same plateaus can be observed again.

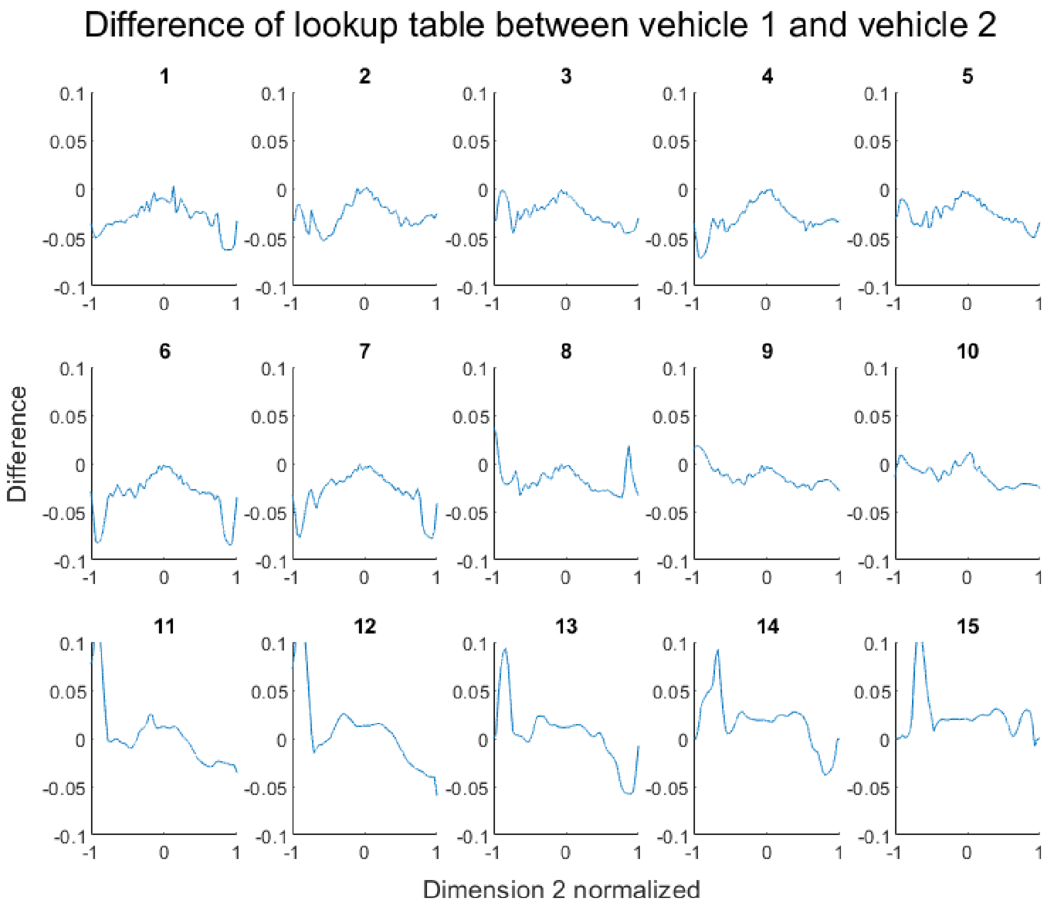

Figure 4.7: Component 1 lookup table values difference between vehicles.

Figure 4.7 shows the difference in the optimized lookup table values of Component 1 for the two vehicles. For the first 10 values in dimension one, Vehicle 1's optimized table shows lower values overall. They are more similar in the middle but with a larger difference as the second dimension is progressed outwards. For the latter plots, plateaus can be observed where the is less data. Closer to the center values of dimension two, Vehicle 1 shows altogether higher values.

Figure 4.8 shows a section of a time series where the lookup table's output can be compared to the logged data. The optimized table shows to be closer to the logged data than the original table.

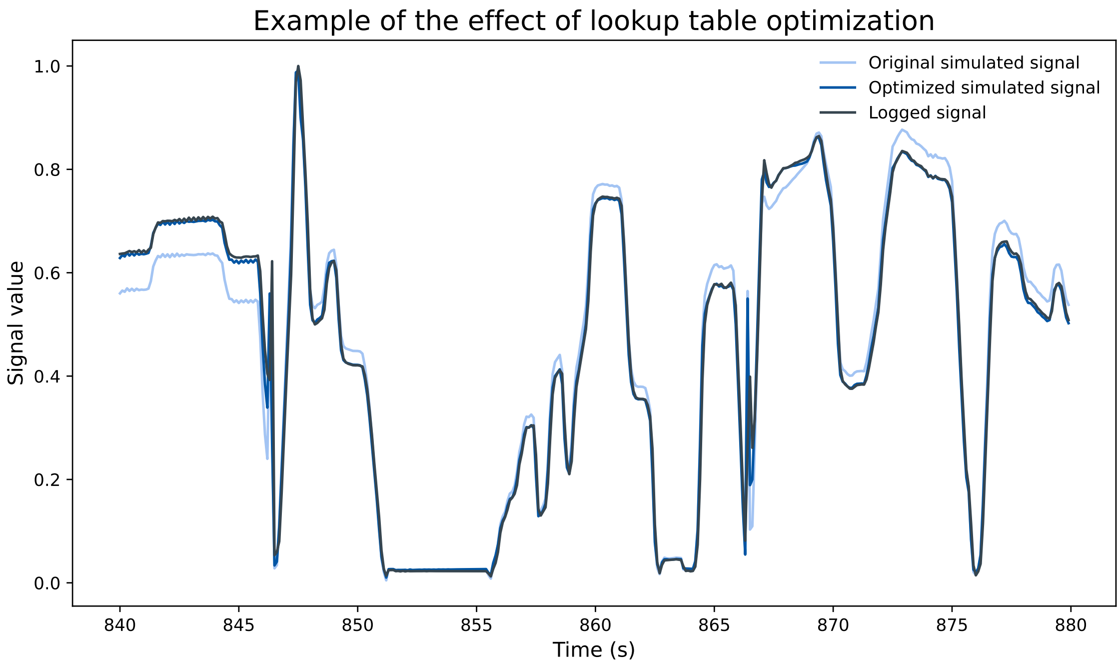

Figure 4.8: Example of logged data compared to simulated data before and after optimization.

4.2 Linear Regression Results

The vehicle labeled *All Vehicles* in the graphs and tables presented in this section represents to the aggregated data set of all the evaluated trucks, as described in Section 3.5.1.4. Figure 4.9 shows an example of how the linear regression method can better align the simulated signal to the real signal with some desired overestimation. The optimized signal slightly overestimates the log signal in the first half of the time series, but during the latter half where there are more oscillations, the alignment is more exact.

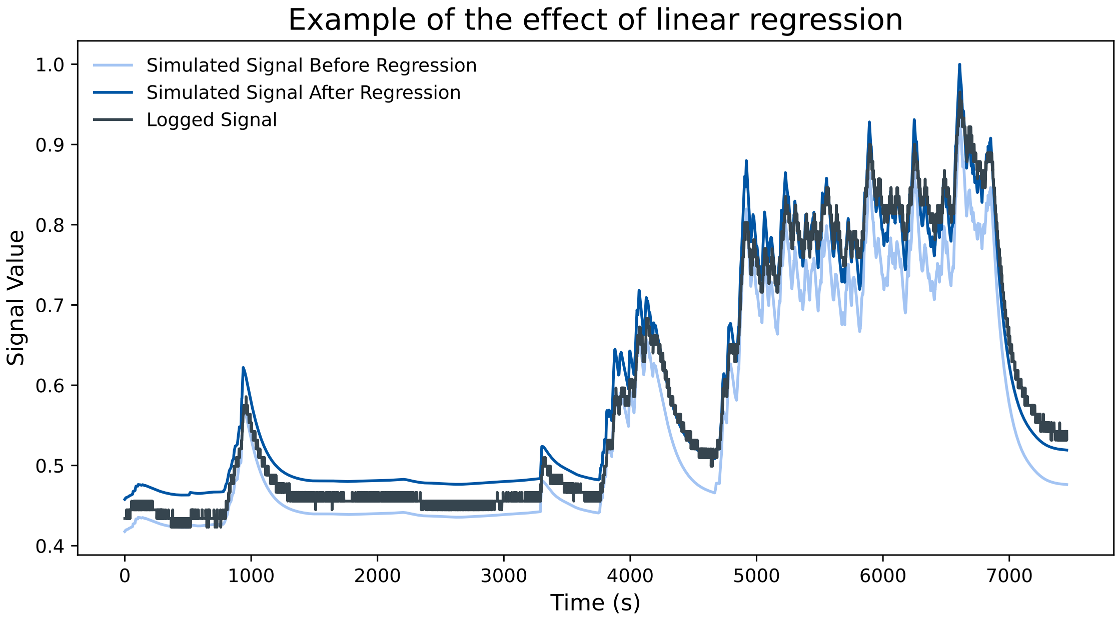

Figure 4.9: An example from a log recorded by Vehicle 1 of how the simulated data changes after the linear regression, compared to the signal from the log.

Tables 4.2 and 4.3, exhibit consistent results. Vehicle 1 is the only vehicle with a positive scaling factor, all other vehicles have a scaling factor of -5%. Vehicles 1, 2 and 6 display a notably lower optimal weight than the other vehicles, while Vehicles 3 and 4 have the highest weights. The only feature that is not shared between the components is the offset of Vehicle 1, Component 1, which is considerably higher than all other offset values.

Table 4.2: The optimal weight and corresponding parameter values, scaling factor and constant offset, for each vehicle (Component 1).

Vehicle	Optimal Weight	Scaling Factor	Offset
Vehicle 1	0.50	+5%	0.043
Vehicle 2	0.50	-5%	0.062
Vehicle 3	0.75	-5%	0.063
Vehicle 4	0.85	-5%	0.054
Vehicle 5	0.68	-5%	0.051
Vehicle 6	0.50	-5%	0.123
Vehicle 7	0.69	-5%	0.041
Vehicle 8	0.71	-5%	0.048
Vehicle 9	0.66	-5%	0.065
All Vehicles	0.66	-5%	0.053

Table 4.3: The optimal weight and corresponding parameter values, scaling factor and constant offset, for each vehicle (Component 2).

Vehicle	Optimal Weight	Scaling Factor	Offset
Vehicle 1	0.50	+2.7%	0.069
Vehicle 2	0.50	-5%	0.071
Vehicle 3	0.91	-5%	0.069
Vehicle 4	0.91	-5%	0.064
Vehicle 5	0.88	-5%	0.049
Vehicle 6	0.55	-5%	0.058
Vehicle 7	0.88	-5%	0.050
Vehicle 8	0.84	-5%	0.051
Vehicle 9	0.89	-5%	0.056
All Vehicles	0.83	-5%	0.058

Figure 4.10 illustrates the impact of the linear regression on the simulated signal of Component 1 across all vehicles. Comparing the box plots before (left) and after (right) the transformation, it is clear that the vehicles which markedly underestimated the logged data (Vehicle 1 and Vehicle 6) show improved alignment with the real data after the regression. While the adjustment is most noticeable in these trucks, an increase in the value of the simulated data is observed in most of the other vehicles as well. Vehicles 3 and 4 that already tended to overestimate the real data do not change noticeably after the transformation. The figure also shows that there are many outliers, i.e. there are many instances where the real signal and corresponding simulated signal diverge considerably. Even though the right group of box plots show that values of the simulated signals have increased, there are still a vast amount of simulated values that underestimate the real values.

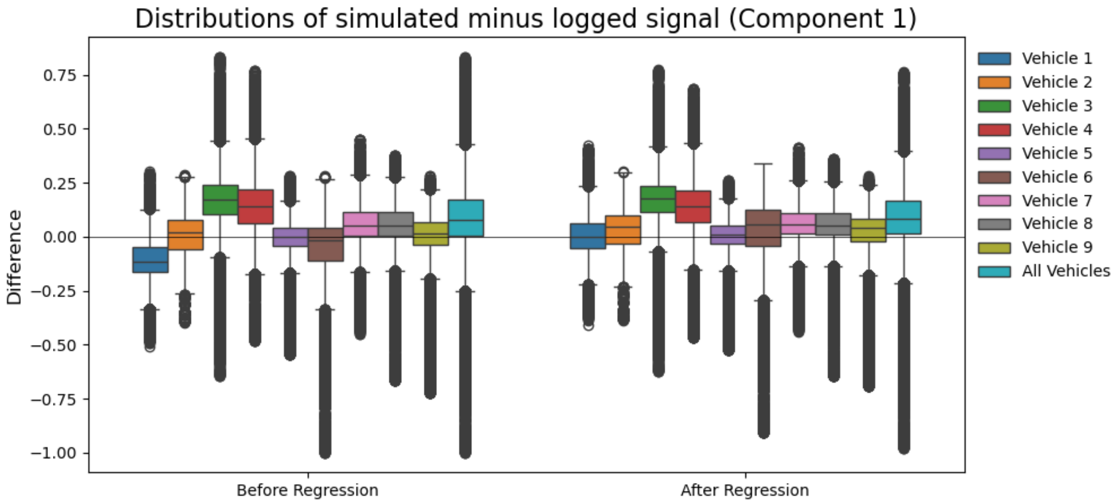

Figure 4.10: Distribution of the differences between the simulated data and the logged data for each vehicle before and after linear regression (Component 1).

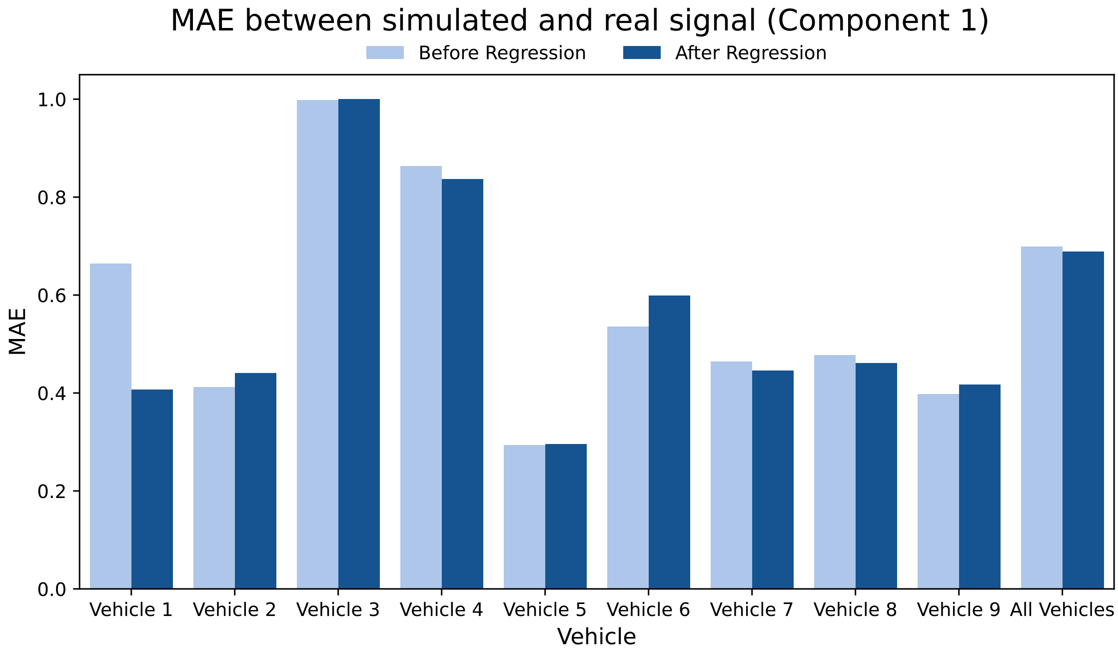

Figure 4.11: Comparison of the mean absolute error between the simulated signal and the logged signal before and after linear regression (Component 1).

Analyzing Component 1's shift in mean absolute error, Figure 4.11 shows that the MAE changes only slightly, either increasing or decreasing marginally for most of the trucks. The most significant changes are observed in Vehicle 1, whose MAE decreases substantially, and Vehicle 6 whose MAE instead increases.

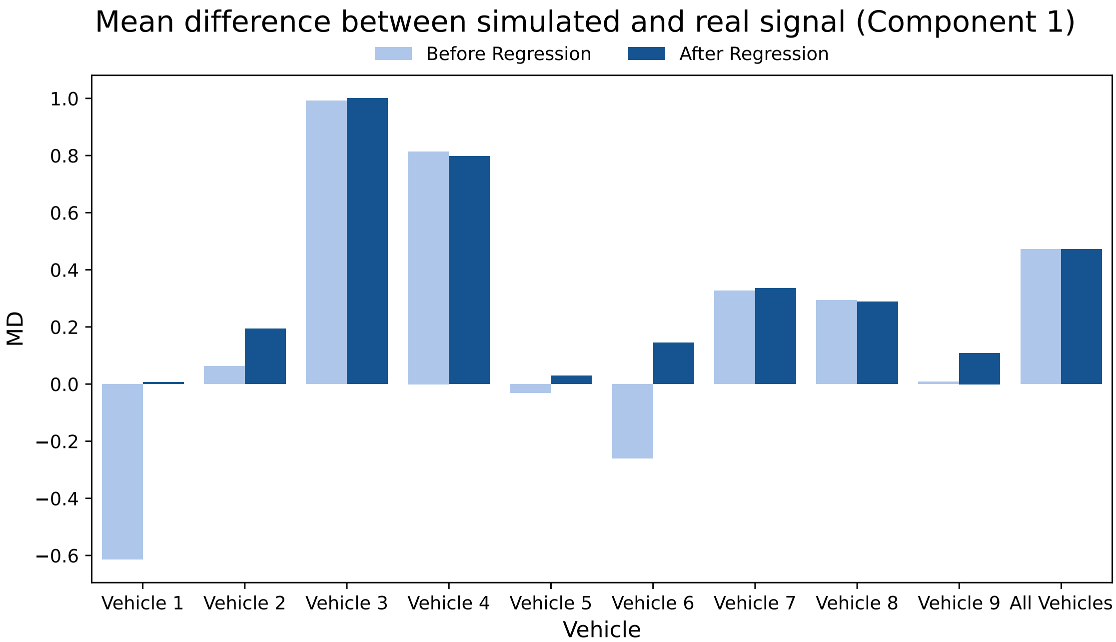

Figure 4.12: Comparison of the mean difference between the simulated signal and the logged signal before and after linear regression (Component 1).

The mean difference (MD) between the simulated data and logged data, shown in Figure 4.12, demonstrates clearer improvements. Vehicles 1, 2, 6 and 9 show notable improvements in this metric, with the other vehicles showing varying results again.

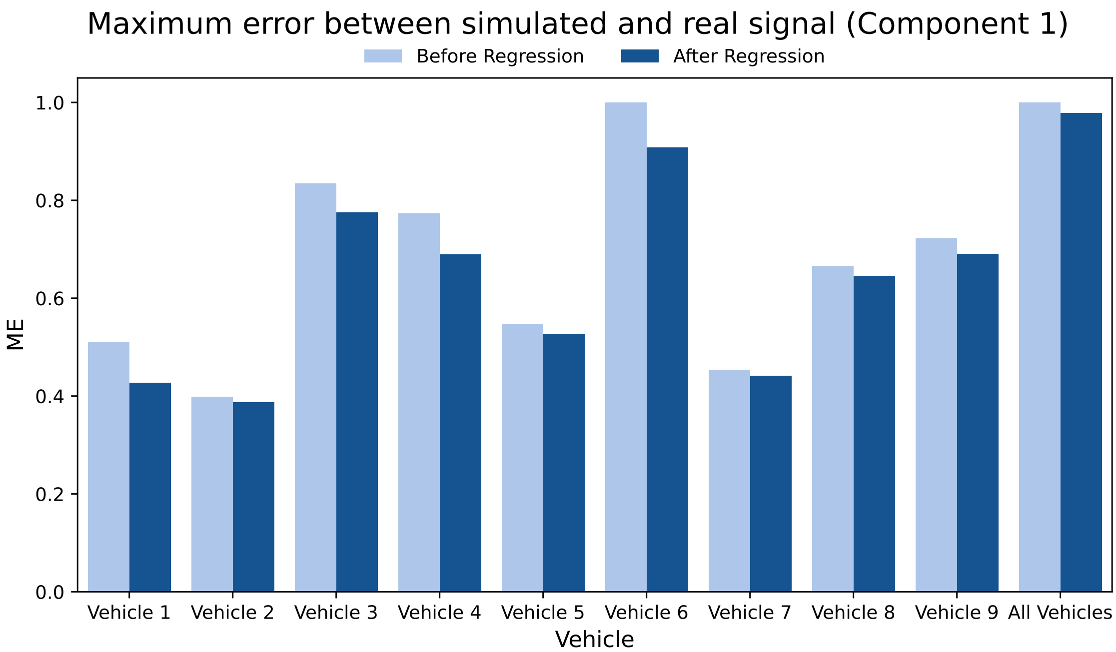

Figure 4.13: Comparison of the maximum error between the simulated signal and the logged signal before and after linear regression (Component 1).

Figure 4.13 shows that the maximum error decreased across the board, with Vehicles 1, 3, 4 and 6 showing significant improvements.

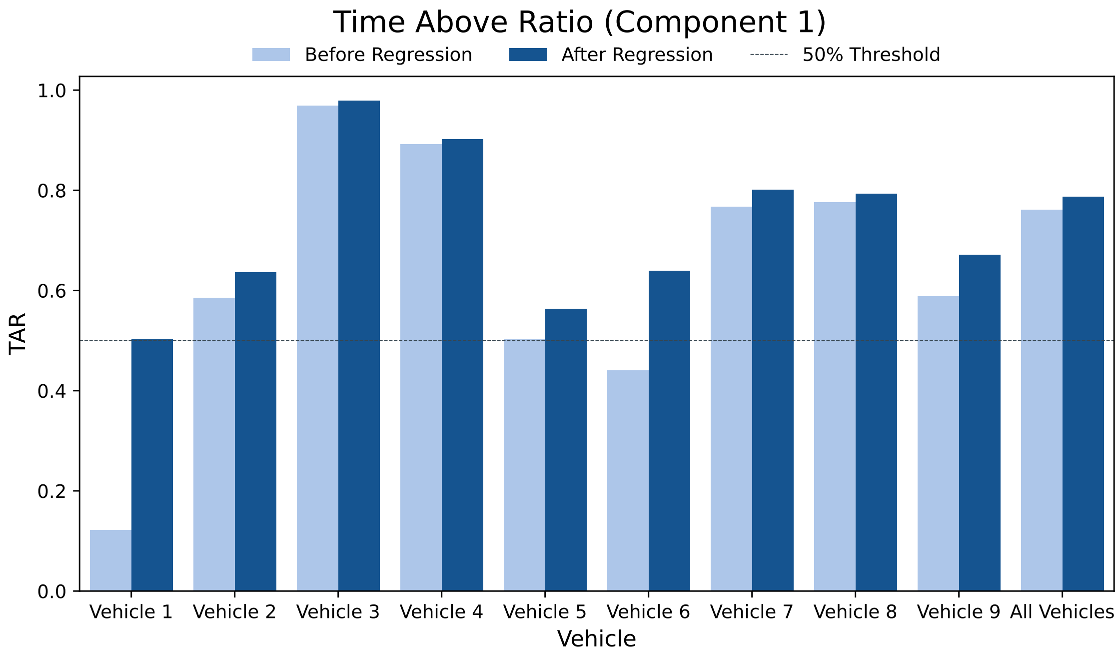

Figure 4.14: Comparison of TAR before and after linear regression (Component 1).

Figure 4.14 displays the time above ratio (TAR) metric (defined in Equation 2.4). This metric increases for all vehicles with the performance of Vehicle 1 improving significantly again, with Vehicles 6 and 9 also noticeably increasing their TAR.

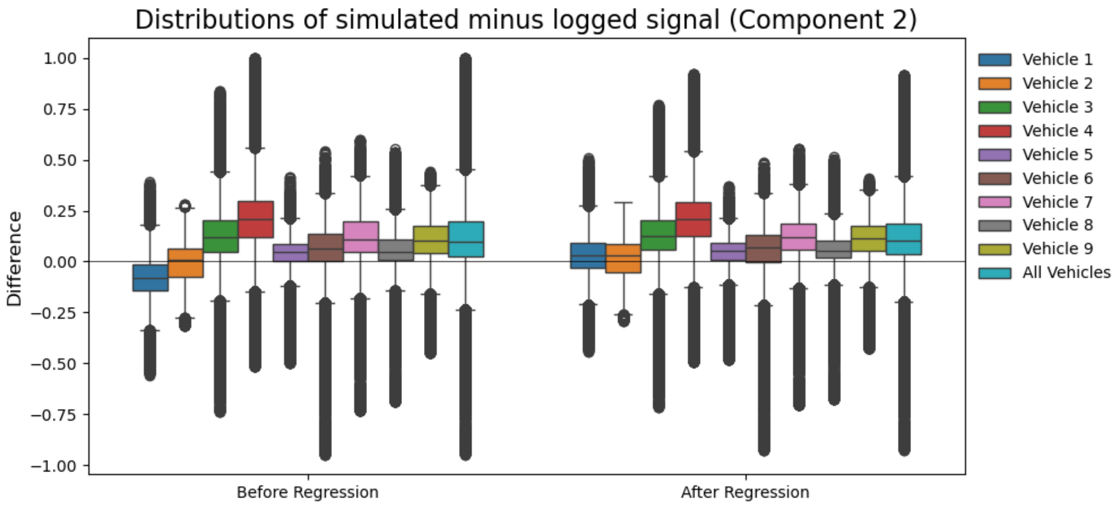

Figure 4.15: Distribution of the differences between the simulated signal and the logged signal for each vehicle before and after linear regression (Component 2).

The distributions for Component 2 across the vehicle, shows in Figure 4.15, displays similar characteristics to the equivalent plot of Component 1, Figure 4.10. A notable difference is that the magnitude of the underestimations is lower, and the differences between the box plots before and after regression are also lower.

The performance metrics for Component 2 (Figures 4.16, 4.17, 4.18 and 4.19) show similar results as those from Component 1. The MAE performance is inconsistent across the vehicles, MD shows little to no improvement for most trucks, and ME and TAR show an overarching improvement. Vehicle 1 is again the only vehicle which shows notable improvement in all metrics.

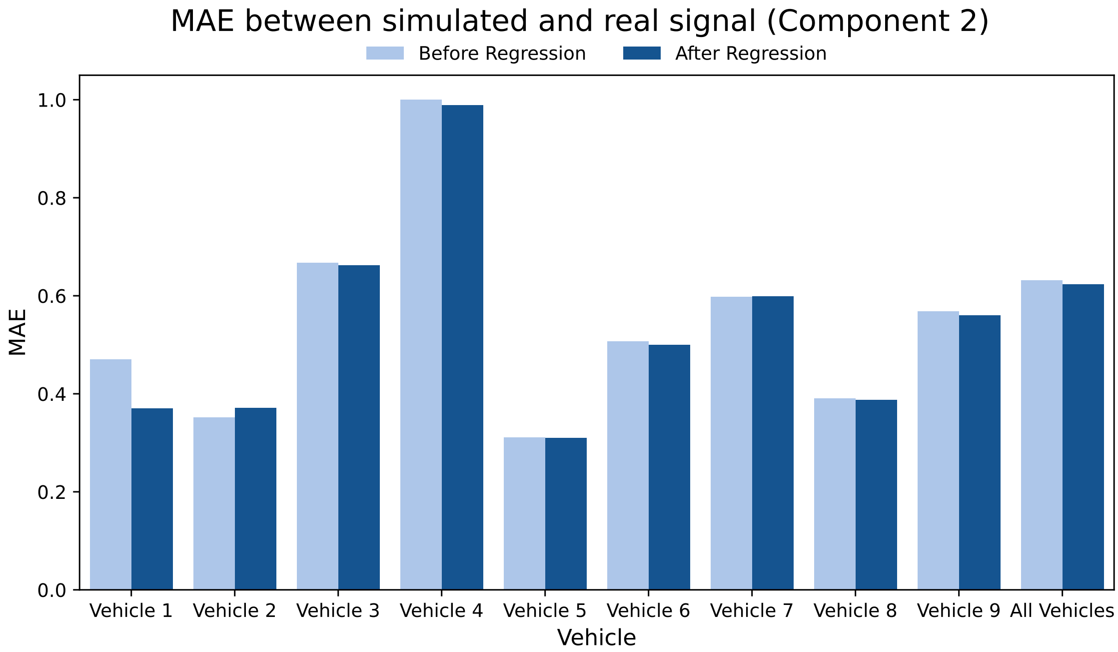

Figure 4.16: Comparison of the mean absolute error between the simulated signal and the logged signal before and after linear regression (Component 2).

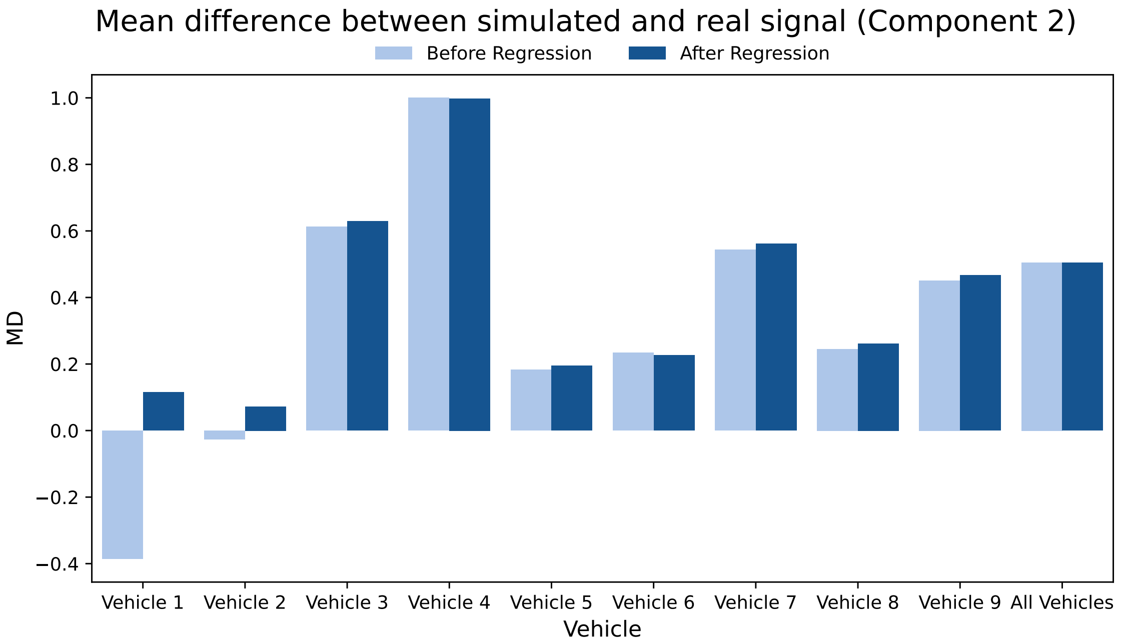

Figure 4.17: Comparison of the mean difference between the simulated signal and the logged signal before and after linear regression (Component 2).

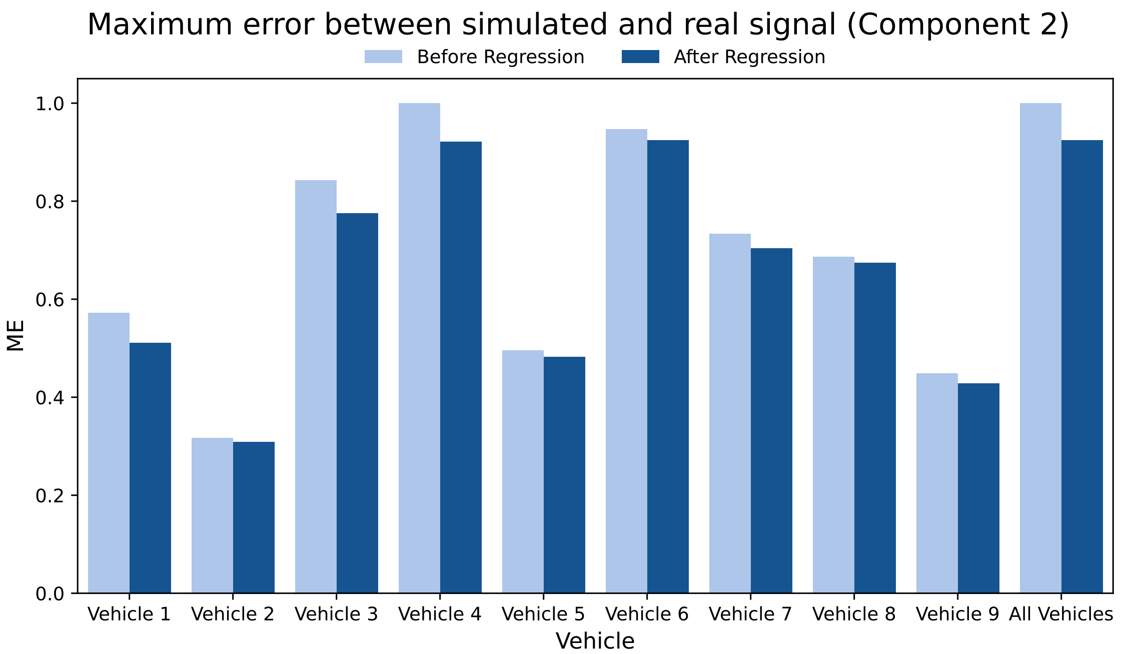

Figure 4.18: Comparison of the maximum error between the simulated signal and the logged signal before and after linear regression (Component 2).

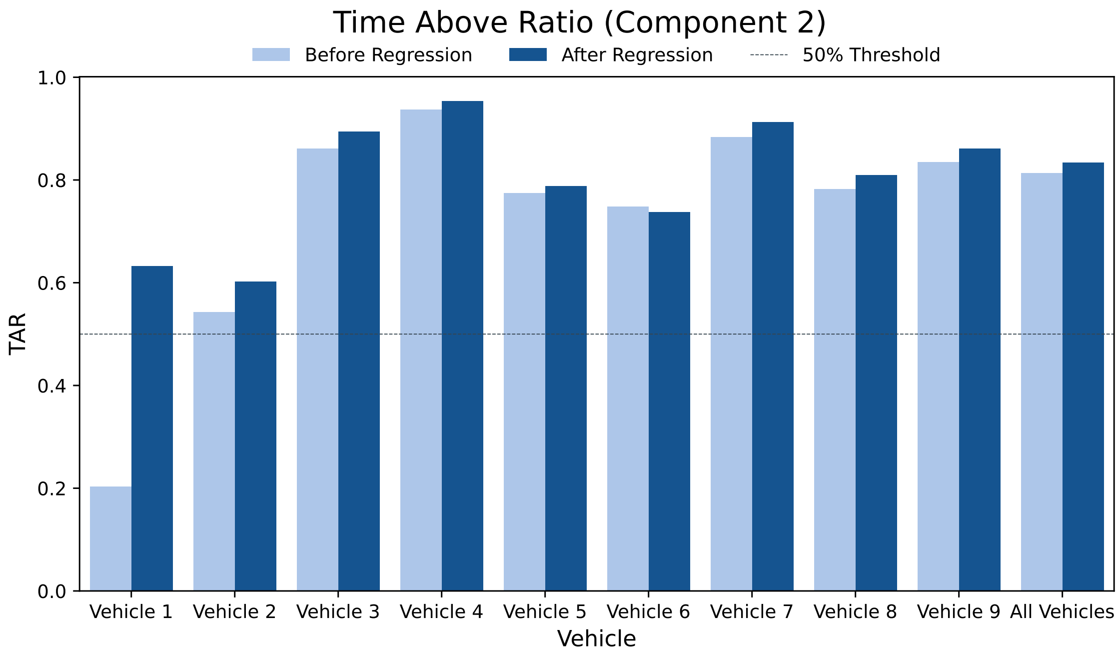

Figure 4.19: Comparison of TAR before and after linear regression (Component 2).

5
Discussion

5.1 Lookup Table

The hyperparameters that the pattern search optimization identified to be optimal were $[0.2344, 1.3438, 0.2250]$. The values in the first and third dimensions are situated toward the low end, which significantly influences the behavior of the Gaussian filter. With a very small standard deviation, the curve that averages the neighboring points becomes increasingly sharp, resulting in minimal smoothing of the data. When there is lower correlation between neighboring indices within a dimension, the potential of generalization is decreased. In contrast, dimensions with more uniform changes are likely to yield higher standard deviation parameters, as they allow for smoother transitions between indices. This continuity enhances the likelihood that corrections made in one area can effectively generalize to neighboring points. As seen in Figure 4.1 there seems to be little room for more improvement, and a minima was reached. However, it is important to note that the pattern search optimization method does not guarantee convergence to a global minimum.

Around the center of the data in dimension two, the output of the table has a smooth and consistent form and appears promising. Following the curves outwards there are some oscillations, which can be beneficial. These small fluctuations may reflect the characteristics of a specific component in a particular vehicle, indicating that the model is capturing relevant details. There should be a balance between keeping the shape and structure of the original table for generalization and fitting to the actual component that the table is adapting to. The method seems to be fulfilling this balance. However, if the fluctuations arise from a lack of information in that area, they are more likely to point to that specific problem, i.e. there is insufficient data.

The change from the original table to the optimized table, as illustrated in Figure 4.5 shows generally good adaptation. The change is mainly continuous which is good. The large difference between plot 7 and 8 is surprising, where the outer values in dimension two drastically increase. The same change can be seen for the values in the middle of plots 10 and 11. As seen in Figure 4.4, plot 10 is the first instance where the lowest value of the table's first dimension is greater than zero. This increase appears to have been consistently underestimated and the method corrected this trend consistently throughout the dimension.

The lack of data in the outer regions of the dimension caused some problems with the optimization. Studying plot 15 in Figure 4.4, the data is many orders of magnitude more sparse than for the prior plots and the table is still updated consistently in that region. The problem arises when the data is so scarce that the algorithm only sees a singular digit number amount of data, such as in plots 17-19. In this case, a few singular indices in the table gets updated, when the surrounding indices then do not get updated similarly enough, the correction steps in. If the singular points have a significant error relative to the logged data, they will continue to undergo frequent and substantial updates. The correction part of the algorithm which is made for small corrections then propagates this relatively large change out to neighboring indices. This is what creates the plateaus that are visible in the later plots. The severity of the plateaus are proportional to the severity of the error and the lack of data.

The idea of optimizing singular components and tuning them to data from a particular physical counterpart shows its effectiveness in Figure 4.7. The values for the lower half of dimension one are consistently lower for Vehicle 1 compared to Vehicle 2. This demonstrates the method's ability to specifically tune the table according to specific characteristics of different physical counterparts of the same component. The outer edge of the latter plots again demonstrates the problem with data in the outer corners of the dimensions. Studying Figure 4.6 the plateaus are persistent along the low values of dimension two in plots 11-15 due to the non-existent data.

Analyzing the effect of the optimization on the simulation in Figure 4.8 shows a great improvement. The time series of the simulated signal after optimization is laying on top of the logged to a point that it is difficult to differentiate them. There are sections with large discrepancies between the original simulation and logged time series where the optimization manages to correct, as well as smaller error. This shows that the method works for both large and small errors. Abrupt jumps in the time series signal also show improvement, but they are more difficult to correct. Around the time steps at 846 and 867 in the figure, this phenomenon can be noted. The optimized table and logged data diverge slightly during the rapid changes, but the error is still approximately halved.

Generalization of the algorithm to other tables and different data should be possible due to its adaptive nature. With the Adam algorithm's ability to steer the updating in the case of large or small errors and noisy or sparse data, different data signals and table with other characteristics would still be trained effectively. Additionally the parameters of the Gaussian kernel can be tuned according to the data, if it required more precise and local corrections for accuracy lower standard deviations can be set. If more sweeping general corrections are needed, perhaps due to less data overall, larger values can be set. The algorithm can also be scaled up and down in number of dimensions. For an n-dimensional table, an n-dimensional linear interpolation and Gaussian kernel can be used with similar theory and ideas behind the algorithm.

To improve the algorithm in the case of sparse data but with large errors which creates these plateaus, as exemplified in plot 19 of Figure 4.4, an improved update algorithm and correction method could be used. An update rule where singular data points get less influence over the corrections would lead to less seemingly odd behaviors. This goes hand in hand with a correction method that would not spread out and affect the tables values like this.

5.2 Linear Regression

5.2.1 Evaluation Metrics

In Tables 4.2 and 4.3, all except one of the scale factor values are at the bounds of the interval they were constricted to. This seems to indicate that without this restriction, the factor would assume values outside of $[0.95, 1.05]$ to minimize the penalty function defined in Equation 2.30. The scaling factor parameters were bounded to be close to one so that the model's performance could be improved by only adding the constant offset value to the simulated signal and ignoring the value of the factor. If linear regression is to be used to enhance model performance, the obtained optimal scaling values being at the bounds seems to suggest that the scaling should not be bounded when transforming the original simulated signal.

Vehicles 1, 2 and 6 have considerably lower weight values (w) than the other vehicles, i.e. the penalty function is not biased towards overestimation for these vehicles. This is surprising since those vehicles' simulated data underestimated the logged data, so a higher w-values would result in higher simulated values after regression. An optimal weight close to 0.5 most likely increased the MD significantly while not affecting the MAE much. Higher w-values probably did not increase the MD-value enough to offset the higher MAE (see an example of how both MAE and MD increase with the weight w in Figure 3.3). Conversely, Vehicles 3 and 4 have the highest overestimation prior to the transformation, and they also have the highest w-values. Low weight parameters would decrease the MD to better align the simulated data to the logged data, but Algorithm 1 penalizes weights which result in a decrease of MD. Therefore, the optimal weight most likely needed to be high to match the overestimation before the linear regression.

Figures 4.11 and 4.16 show that the change in MAE before and after linear regression is not consistent across the vehicles. The Optimal Weight Algorithm was employed to select the best weight by penalizing the weight values which increased the MAE. Furthermore, the algorithm favored higher mean difference values which is achieved with a higher value on the weight parameter. Because a higher weight value also increases the MAE, the algorithm likely selected a weight with a MAE very close to the original MAE so that the mean difference increased while the MAE value remained relatively stationary. Due to small deviations between the validation data and the test data, this led to the MAE either slightly increasing or decreasing when the performance was evaluated on the vehicles' test data.

The mean difference results (presented in Figures 4.12 and 4.17) are a bit more consistent. Only the MD of Vehicle 4, 6 and 8 decrease slightly across both components, and they already overestimated the real data. Whether or not an increase in mean difference is an improvement in performance is not obvious. Vehicles 3, 4 and 7 significantly overestimated the logged data before transformation, so further overestimation might not be beneficial. The vehicles that previously underestimated the real data show considerable improvement, which might compensate for the slight increase in overestimation of Vehicles 3, 4 and 7.

The outcome of the maximum error evaluation is even more consistent, with every vehicle showing improvement, as seen in Figures 4.13 and 4.18. The maximum error contributes the most to the penalty function (Equation 2.30), so it is not unexpected for the maximum error to decrease across all trucks as it will be the primary "target" of the scaling factor and constant offset. Again, the change in TAR presented in Figure 4.14 and Figure 4.19 also shows an overarching consistency in increased performance, with Vehicle 6's Component 2 data being the exception.

Vehicle 1 is the only vehicle which, across both components, demonstrates considerable improvement in all four metrics that were evaluated. Referring to Figures 4.10 and 4.15, it is clear that Vehicle 1 underestimated the real data more often than the other vehicles. It is noteworthy that the performance of Vehicle 6 for Component 1 improved considerably for MD, ME, and TAR, while its MAE increased the most of all vehicles. A possible explanation is that the linear regression overcompensated, which the high constant offset value supports, (see Table 4.2). A similarly consistent improvement as Vehicle 1 could probably have been achieved with a lower offset value. The other vehicles' performances before and after the transformation is more inconsistent and the changes are less notable. This indicates that linear regression to transform the simulated signals is most effective on vehicles whose simulated data largely underestimates the real data. This is reasonable because simulated data which is already (mostly) well-aligned to the logged data is not going to show consistent improvement, the changes are going to be more dependent on what data the training-validation-test splits contains.

The artificial truck *All Vehicles* shows negligible differences in MAE and MD, but the decrease in ME and increase in TAR is notable. This pattern is consistent across both components. The data that comprises *All Vehicles* is simply the combined data of the real trucks that were evaluated, so the data of *All Vehicles* will be unbalanced if the trucks do not have the same amount of available logs. The corresponding optimal scaling factor and constant offset of *All Vehicles* will therefore be biased towards improving the performance of the vehicles with the most available logs. This means that the optimal parameters most likely will not perform as well on vehicles that are underrepresented in *All Vehicles*. To make the result more balanced, an equal amount of logs could be taken from each vehicle to create *All Vehicles*, but the risk is that the data is not representative of all available data if too few logs can be used. Furthermore, if some vehicles have more available logs because they are used more frequently, then biasing *All Vehicles* to better align with those vehicles would be beneficial.

5.2.2 Optimal Weight Algorithm

The value of the α parameter in Algorithm 1 was selected by comparing what MAE and MD values were selected for different α-values, as seen in Figure 3.3. The value $\alpha = 0.8$ was selected because it looked to be the value which best balanced relatively low MAE and relatively high MD across the vehicles. The selection was arbitrary which introduces human bias, but it also makes the algorithm more flexible for changing the ultimate objective of the linear regression. The Optimal Weight Algorithm considered the MAE and MD after linear regression in comparison to the MAE and MD before the linear regression. For a vehicle like Vehicle 3 whose simulated data already overestimated the log data (Figure 4.12), this method resulted in the MAE increasing after linear regression, as seen in Figure 4.11.

An alternative algorithm could use a set threshold for MAE and MD respectively that is the same for all vehicles. This would probably result in the MD decreasing for vehicles with previously high MD and vice versa for vehicles with previously low MD, while decreasing the MAE for both. A consideration to take into account is that an alternative method like this would likely also decrease the TAR for Vehicle 3 and other vehicles with similar data profiles as it would decrease the simulated values to lower the MAE. This highlights another limitation of Algorithm 1, which is that it only takes two metrics into account. A more complex algorithm could take more metrics into account to make sure that the performance does not depreciate too much in any given metric. A possible drawback of this method would be that the increased number of factors in such an algorithm would make it more difficult to decide what the thresholds should be, and with too many metrics it might be impossible to find a weight that performs sufficiently well across all of them.

6

Conclusion

In this thesis, a workflow was established to process driving session logs and utilize the data as input for an isolated model. First, driving session logs were systematically processed to extract the relevant signals necessary for model input. Subsequently, algorithms for lookup table optimization and linear regression were developed to optimize a specific set of parameters for each individual truck. This optimization process resulted in the creation of optimization methods that can be applied to further the construction of digital twin models.

In the optimization of the lookup table, the training and validation errors displayed a clear reduction, particularly in the initial iterations, indicating effective learning and adaptation of the lookup table to the data. This was further iterated by the MAE results, as they show substantial improvements across all vehicle components, with the most significant enhancements observed in components that initially exhibited higher errors. The examination of the optimized lookup tables revealed that while the overall structure remained similar to the original tables, localized adjustments were made to raise the accuracy. There were some problematic parts of the method, as the presence of plateaus in the optimized tables indicates areas where data was sparse and the optimization troublesome. The improvements in the time series comparisons between the optimized outputs and logged data further validate the effectiveness of the optimization method. The algorithm's adaptive nature allows for effective tuning to specific characteristics of different vehicle components which should lend itself useful for optimizing other lookup tables with different qualities.

The linear regression method significantly improved simulated data which severely underestimated the corresponding real data prior to optimization. Vehicles which did not underestimate to the same degree before the linear regression showed small changes in MAE as well as in MD, while ME and TAR decreased and increased respectively. The linear regression method therefore showed positive results in terms of lessening the severity of the underestimation, with the magnitude of the improvement being proportional to the vehicle's underestimation. The α-parameter in Algorithm 1 introduces subjectivity to the process, but it also allows for more flexibility depending on the end goal of the linear regression.

Bibliography

[1] IEA (2024), Global EV Outlook 2024, IEA, Paris https://www.iea.org/reports/global-ev-outlook-2024/trends-in-heavy-electric-vehicles, Licence: CC BY 4.0

[2] Simulink - Simulation and Model-Based Design. Accessed 28 May 2025. https://se.mathworks.com/products/simulink.html.

[3] The MathWorks Inc. (2022). Statistics and Machine Learning Toolbox Documentation, Natick, Massachusetts: The MathWorks Inc. https://se.mathworks.com/help/simulink/slref/ndlookuptable.html

[4] Y. Bai and D. Wang, "On the Comparison of Trilinear, Cubic Spline, and Fuzzy Interpolation Methods in the High-Accuracy Measurements," in IEEE Transactions on Fuzzy Systems, vol. 18, no. 5, pp. 1016-1022, Oct. 2010, doi: 10.1109/TFUZZ.2010.2064170.

[5] R. J. Hyndman, "Moving Averages," in International Encyclopedia of Statistical Science, M. Lovric Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 866-869.

[6] D. P. Kingma and J. L. Ba, “Adam: A Method for Stochastic Optimization”, The International Conference on Learning Representations (ICLR), 2015.

[7] Bishop, Christopher M. Pattern recognition and machine learning. New York: Springer, 2006.

[8] Chung, Moo K. "Gaussian kernel smoothing." arXiv:2007.09539v4 2021.

[9] M. Wang, S. Zheng, X. Li and X. Qin, "A new image denoising method based on Gaussian filter," 2014 International Conference on Information Science, Electronics and Electrical Engineering, Sapporo, Japan, 2014, pp. 163-167.

[10] Hastie, Trevor, Tibshirani, Robert and Friedman, Jerome. The Elements of Statistical Learning. New York: Springer, 2001.

[11] The MathWorks Inc. (2022). Statistics and Machine Learning Toolbox Documentation, Natick, Massachusetts: The MathWorks Inc. https://se.mathworks.com/help/images/ref/imgaussfilt3.html

[12] Xin, Yan, and Su Xiaogang. Linear Regression Analysis: Theory And Computing, World Scientific Publishing Company, 2009. ProQuest Ebook Central,

https://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=477274.

[13] G. Kinkino. "Quasi-Newton line search algorithm for solving unconstrained non-linear least square optimization problem," in African Journal of Mathematics and Computer Science Research, vol. 13, no. 1, pp. 11-21, 2021. doi: 10.5897/A-JMCSR2020.0813

[14] R. H. Byrd, L. Peihuang, J. Nocedal and C. Zhu. "A Limited Memory Algorithm for Bound Constrained Optimization," in SIAM Journal of Scientific Computing, vol. 16, no. 5, 1995. doi: 10.1137/0916069.

[15] C. Zhu, R. H. Bird, L. Peihuang and J. Nocedal. "Algorithm 778: L-BFGS-B," in ACM Transactions on Mathematical Software, vol. 23, no. 4, pp. 550-560, 1997. doi: 10.1145/279232.279236.

[16] minimize - SciPy v1.15.3 Manual. Accessed 25 May 2025. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY