FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

FH Aachen

Fachbereich
Elektrotechnik und Informationstechnik

Masterarbeit

Der Titel der Arbeit ist zweizeilig

Vorname Nachname
Matr.-Nr.: 123456

Referent: **Prof. Dr-Ing. ...**

Korreferent: **Prof. Dr.-Ing. ...**

July 7, 2020

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Aachen, July 7, 2020

Danksagung

Danke.

Contents

Abstract Modern day railway system operations require automated train control mechanisms, e.g. European Train Control System ETCS, to maximize efficiency, which is often times limited by outdated infrastructure, as well as safety of operations. One way to achieve this is by lowering the required distance between two trains on the same track, which in turn demands a reliable method of predicting the braking distance at any given moment.

While determination of the necessary braking curves is feasible for a limited number of train formations, the large diversity of vehicles in freight operations poses an issue. One approach for a solution would be using Big Data, which would be able to process the required amounts of data to calculate reliable braking curves even for freight operations.

The problem here is there is simply not enough data available since freight trains usually don't have the sensory equipment needed. To circumvent that obstacle, this work proposes generation of artificial data via white box modeling to be then used in further big data operations.

Chapter 1

Introduction

Introduction This section describes the background and motivation of the research (Sect. 1.1), the problem to be addressed (Sect. 1.2) and the proposed solution (Sect. 1.3)

1.1 Background

In recent years, data science has become more and more prominent. Since large quantities of data have become omnipresent, big data processing is applicable in various fields of research and operations. This work focuses of the application of big data processing to railway system operations, more specifically the preconditions. Mainly, two areas show promise to benefit from big data usage, which is predictive maintenance as well as optimization of railway operations. As the name suggests, lots of data is required, think hundreds of terabyte for freight operations in Germany alone. As it stands though, unfortunately, this data source remains largely untapped today, owing to freight wagons typically lacking necessary sensory equipment. This is where this work comes into play.

1.1.1 Railway Vehicle Operations

Road and rail traffic are fundamentally different, mainly in two regards. First, the physical properties, which will be discussed in chapter 2, second, the actual modalities of operations, which will be topic of this section.

In road traffic, there is many different vehicles, all operating independently from one another. Although there is ongoing experimentation to utilize autonomous vehicles to recreate train-like lorry configurations, this is not the norm. In contrast to that, the track guiding of wheels offered by rails enables the formation of trains possibly kilometers long, reducing labor cost, infrastructure usage and energy consumption. This, of course, calls for special safety measures to be taken, since higher speeds and payloads result in long braking distances.

<TODO> Abstandsparadigma < / >

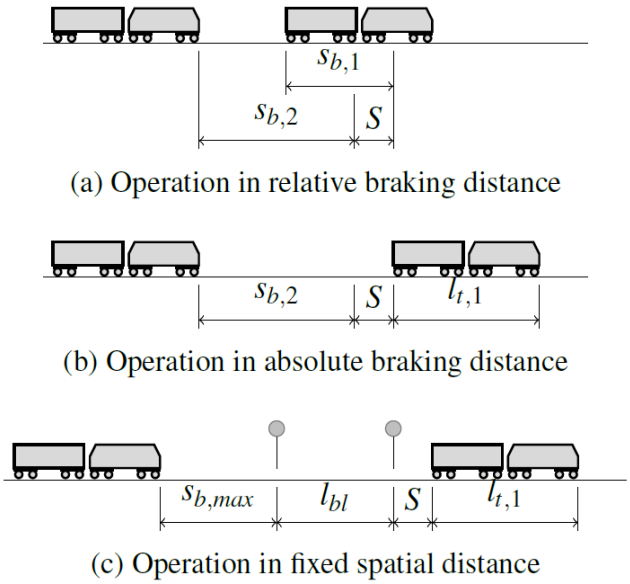


Figure 1.1: Spacing paradigms

The above figure illustrates the three main principles of spacing between two trains on the same track. $s_{b,i}$ denoted the braking distance of train i, $t_{l,i}$ the length of train i, l_{bl} the length of a track section, and S a safety margin. Using approach (c), the distance between trains A and B must at any time be at least A's maximum braking distance plus the length

1.1.2 Automatic Train Protection and Cab Signaling

Automatic train protection systems are designed to ensure safe operation in case of human or technical error or malpractice by the train operator. Generally, all trains operate on track sections which are free of other vehicles, reserved and locked. Think for example a section between two signals. These sections are also referred to as movement authority, and a train has to be capable of coming to a complete halt before the end of movement authority as to not violate a track section locked by another train. For this, the driver and or the onboard computer needs to be informed about the endpoint of the current movement authority as well as speed limits. The train protection system enforces application of brakes in case of violation of restrictions, for example if the train exceeds the speed limit or the train would otherwise be unable to stop in time before movement authority expires. Train protection systems may be categorized by means of information transfer. Spatially discrete acting systems use transmitters placed at strategic points along the track, for example signals, whereas continuous systems may transmit information at all times, either via track-sided wire loops or radio communication. These allow higher degrees of automation than discrete systems, as trains have access to the necessary real-time data.

Cab signaling relays all relevant information to the train operator so they may act in the most optimal way. The European train control system, short ETCS, encompasses both of these.

1.1.3 Braking Curves

For the ETCS to be able to supervise train velocity, it needs to determine the vehicle's braking capacity for any given moment in time, using a mathematical model of the braking dynamics and of the track characteristics. This prediction is called a braking curve, which describe the probable braking process for a set of input parameters. As a simple example, refer to the following figure **<TODO> Abbildung einfuegen < / >**. Since braking performance cannot be predicted with absolute certainty, an ϵ parameter needs to be factored in to account for random behavior. Therefore, there is a number of discrete braking curves for any set of parameters, forming a probability distribution. The parameters may be classified in four categories:

- Physical parameters, which is real time measurements from on-board equipment
- Fixed values, like driver reaction time
- Trackside data, like track gradient or signaling data
- Train data, mostly captured beforehand, relating to the rolling stock braking system itself

The fourth category, train data, is the key factor for this work, as will be explained below.

1.2 Problem

As mentioned, one category of input parameters for the calculation of braking curves is data directly associated with the rolling stock, and in particular to the braking system itself. While this does not necessarily present itself as an issue in passenger trains, freight wagons, in contrast, are usually not electrified and therefore do not currently posses the sensory equipment that would be necessary to obtain such data in an adequate quantity and quality, especially in regards to big data processing. Although it has been proposed to equip freight wagons accordingly **<TODO> ref zu wagon4.0 < / >**, it will be years before enough rolling stock has been retrofitted as to make it possible to obtain the desired data. Since braking curves vary according to the rolling stock, this lack of data makes calculation thereof impossible.

1.3 Solution

Since the problem is lack of data from real life operations, this work proposes to circumvent this by generation of artificial, simulated data instead. The data set should replicate the actual distribution of braking behavior as close as possible. It is therefore necessary to first create a model encompassing the braking process of a freight train. This model will be discussed in depth in chapter 3. By using that model to run a large number of simulated braking processes, one may obtain data about the behavior of the braking system, which, albeit being artificial, should at least satisfy requirements for the calculation of rudimentary braking curves. A freely configurable simulation environment further allows for great flexibility in terms of input parameters, therefore enabling for covering a very large range of data, where computing power

and time are the only limiting factors. The process of data generation and simulation will be discussed further in chapter 4.

As real life operations would yield very high quantities of data, the simulation output must be stored in a data structure which is suitable for big data processing. This structure will also be discussed in chapter 4.

Chapter 2

Fundamentals of Railway Vehicle Engineering

Introduction This section deals with the engineering backgrounds of the work, especially in regards to railway vehicle engineering and railway physics. For the creation of a braking model, it is necessary to have at least a basic understanding of the engineering and physics behind rail traffic. The most fundamental component to understand is, of course, the actual braking process.

2.1 The train brake

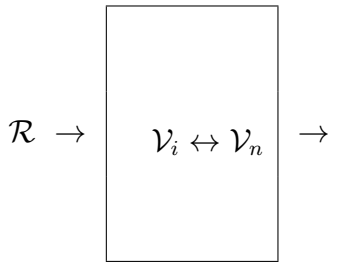
There is a number of different construction methods for brakes, but each must address two problems: The means of transmission of commands and the actual form of brake force generation. The focus here will lie on the pneumatic brake, since it is still the most prevalent in cargo vehicles. From a macro perspective, the pneumatic brake has three main components: The train driver's brake valve, the brake pipe and one or more brake cylinders for each wagon or locomotive. The driver's valve and the brake pipe are primarily the means of command transmission, while the brake cylinder's job is the generation of brake force.

The braking system is an indirect one, which means the brake pipe needs to be vented to apply the brakes. Therefore, the brake pipe is kept at a running pressure of 5 bar. One advantage of this approach is that brakes are automatically applied in case of train separation. The downside, however, is that command transmission is limited to the speed of sound. In order to generate that pressure, the locomotive contains a compressor and an air reservoir. Initially, the brake pipe as well as all auxiliary air reservoirs are brought to operating pressure. If the train operator wishes to brake, he has to actuate the driver's brake valve, which in turn vents the brake pipe, usually as far as 3.5 bar (full braking), sometimes even lower. As rule of thumb, the more pressure is vented, the more braking force is applied. Each wagon brake has, in essence, an auxiliary air reservoir, a distributor valve and a braking cylinder. When the brake pipe is vented, the air in the auxiliary reservoir has a higher pressure. This difference in pressure triggers a switch of the distributor valve, and the air from the reservoir may go into the brake cylinder. This increase in the cylinder generates the actual braking force. To summarize, the amount of generated braking pressure is directly proportional to the amount

of pressure vented.

2.2 Influences on braking performance

Since the aim of this work is to monitor braking performance, it becomes necessary to understand which factors influence the braking process, and how. The braking process is, in its core, a system. A system may be defined as a set of objects, which are interconnected by relations. It is enclosed by its environment, which may or may not affect the system itself. Below is an illustration, also called box model:



The box represents the actual system, $\mathcal{V}_i$ are the system variables, or objects, their relations represented by the $\leftrightarrow$. The $\mathcal{R}$ represents the system's environment and its influence on the system itself. Finally, the rightmost arrow $\rightarrow$ is the influence the system might have on the environment, though this is often disregarded as the focus lies on what happens inside the system. Creating a box model for the braking process makes for an easier identification of the influencing factors, and allows for their categorization into inherent, i.e. system variables, and external, i.e. system environment.

From a modeling perspective, it is sensible to differentiate as follows: Everything related to the train itself is part of the system, and everything else is part of the system's environment. As per definition, it is compromised of a set of variables, a set of relations, and a set of constants. Let us make a formal definition of the model for the braking process system:

$$\mathcal{M} = \{\{\mathcal{V}\}, \{\mathcal{R}\}, \{c\}\}$$

where $\{\mathcal{V}\}$ is the set of system variables, $\{\mathcal{R}\}$ is the set of relations (a) between elements of $\{\mathcal{V}\}$ and (b) between system and environment, and $\{c\}$ is the set of system constants. The system variables may be subsystems themselves. We shall take a closer look at $\{\mathcal{V}\}$ first.

2.2.1 Inherent factors, model variables

Looking at $\mathcal{M}$, which is the system describing the braking process, $\{\mathcal{V}\}$ consists of all factors inherent to the train which have an impact on the braking, as well as the relevant quantifiers. The main ones are:

- The train brake
- The wagon mass, denoted as m

- The train velocity, denoted as v
- The train composition, i.e. in which order the wagons are, for example the heaviest wagons being in front and the lighter ones at the back, denoted as *comp*
- The generated braking force, denoted as F_b
- The train's deceleration, denoted as a
- The train's braking distance, denoted as d

It is noteworthy that the train brake is another system in itself, which is perfectly fine of course. Alas, it is therefore necessary to define another model.

$$\mathcal{M}_B = \{\{\mathcal{V}_B\}, \{\mathcal{R}_B\}, \{c_B\}\}$$

Let $\mathcal{M}_B$ be the model describing the system train brake, with $\{\mathcal{V}_B\} = \{\mathcal{M}_{bv}, \mathcal{M}_{bp}, \mathcal{M}_{bc}\}$, where $\mathcal{M}_{bv}$ is the model of the brake valve, $\mathcal{M}_{bp}$ the model of the brake pipe and $\mathcal{M}_{bc}$ the model of the brake cylinder, which in turn shall be defined as follows:

$$\mathcal{M}_{bv} = \{\{\mathcal{V}_{bv}\}, \{\mathcal{R}_{bv}\}, \{c_{bv}\}\}$$

Let $\mathcal{M}_{bv}$ be the model describing the system brake valve, with $\mathcal{V}_{bv} = \{x\}$, $\mathcal{R}_{bv} = \emptyset$ and $c_{bv} = \emptyset$, where x is the state of the brake valve, i.e. its opening percentage, ranging from 0 (fully closed) to 1 (fully open, full braking).

$$\mathcal{M}_{bp} = \{\{\mathcal{V}_{bp}\}, \{\mathcal{R}_{bp}\}, \{c_{bp}\}\}$$

Let $\mathcal{M}_{bp}$ be the model describing the system brake pipe, with $\mathcal{V}_{bp} = \{l_{bp}, p_{bp}\}$, $\mathcal{R}_{bp} = \emptyset$ and $c_{bp} = \{v_{bp}\}$, where l_{bp} is the physical length of the brake pipe, p_{bp} is the pressure on the brake pipe and v_{bp} is the propagation velocity of the pipe's medium.

$$\mathcal{M}_{bc} = \{\{\mathcal{V}_{bc}\}, \{\mathcal{R}_{bc}\}, \{c_{bc}\}\}$$

Let $\mathcal{M}_{bc}$ be the model describing the system brake cylinder, with $\mathcal{V}_{bc} = \{p_{bc}\}$, $\mathcal{R}_{bc} = \emptyset$ and $c_{bc} = \{t_{bc}\}$, where p_{bc} is the cylinder's pressure and t_{bc} is the cylinder's fill time.

This wraps up the definition of $\mathcal{V}_B$. We may now get to the definition of $\mathcal{R}_B$, which is the relations (a) between the elements of $\mathcal{V}_B$ and (b) between $\mathcal{M}_B$ and its environment. Let us begin with (a):

Convention For readability purposes, $\propto$ shall from here on denote direct proportionality, and $\sim$ shall denote inverse proportionality.

$\mathcal{M}_{bv}$ relates to $\mathcal{M}_{bp}$ in the sense that the state of the brake valve has a direct influence on the pressure on the brake pipe. More specifically, $x \in \mathcal{V}_{bv}$ is inversely proportional to $p_{bp} \in \mathcal{V}_{bp}$. The higher the value of x, i.e. the opening percentage of the brake valve, the lower the value of p_{bp}, i.e. the pressure on the brake pipe. We can therefore denote $x \sim p_{bp}$, and subsequently define a relation $R_{bv,bp} : \mathcal{M}_{bv} \sim \mathcal{M}_{bp}$.

$\mathcal{M}_{bp}$ relates to $\mathcal{M}_{bc}$ in the sense that the pressure on the brake pipe has a direct influence on the brake cylinder's pressure. More specifically, $p_{bp} \in \mathcal{V}_{bp}$ is inversely proportional to $p_{bc} \in \mathcal{V}_{bc}$. The lower the value of p_{bp}, i.e. the pressure on the brake pipe, the higher the value p_{bc}, i.e. the pressure in the brake cylinder. We can therefore denote $p_{bp} \sim p_{bc}$, and subsequently define a relation $R_{bp,bc} : \mathcal{M}_{bp} \sim \mathcal{M}_{bc}$.

For simplicity, we can look at $\mathcal{M}_B$ as being isolated from its environment, i.e. the environment does not impact $\mathcal{M}_B$, and vice versa. Therefore, (b) is void. Consequently, we can define $\mathcal{R}_B$ as follows:

$$\mathcal{R}_B = \{R_{bv,bp}, R_{bp,bc}\}$$

This finally leaves us with c_B. Since call constants are specific to the respective sub-models $\in \mathcal{V}_B$, $\mathcal{M}_B$ has no constants of its own, therefore $c_B = \emptyset$. This completes the definition of $\mathcal{M}_B$, and we can properly define the set of model variables of the model braking process:

$$\{\mathcal{V}\} = \{\mathcal{M}_B, m, v, comp, F_b, a, d\}$$

<TODO> Wie genau Bremsmodell definieren? Notizen: Bremse besteht aus, wie oben definiert:

- ~~**Brake valve - opening percentage** x~~
- ~~**Brake pipe - length** l_{bp}~~**, propagation velocity** ~~v_{bp}~~**, pressure** ~~p_{bp}~~
- ~~**Brake cylinder - fill time** t_{bc}~~**, pressure** ~~p_{bc}~~

Relationen sind:

1. ~~$x \sim p_{bp}$~~**, intern**
2. ~~$p_{bp} \sim p_{bc}$~~**, intern**
3. $p_{bc} \sim F_b$**, ausgangs-**
4. **Wie zeitliche Auswirkungen darstellen?**

< / >

2.2.2 Relations

Looking at our model $\mathcal{M}$, we have now defined its set of variables, $\mathcal{V}$. Next, it is necessary to define its set of relations, $\mathcal{R}$, which may be distinguished into internal and external relations.

Relations between system variables

Obviously, the train brake $\mathcal{M}_B$ relates to F_B as it is the actor responsible for generating the braking force. More specifically, $p_{bc} \in \mathcal{V}_{bc}$, which is the pressure in the brake cylinder, is directly proportionate to F_b, meaning the higher the pressure, the more force is applied. We can therefore denote $p_{bc} \propto F_b$, and subsequently define a relation $R_{\mathcal{M}_B,F_b} : \mathcal{M}_B \propto F_b$.

<TODO> Wie interagieren die Systemvariablen untereinander? Relationen sind:

- $\mathcal{M}_B \propto F_b$
- $m \propto F_b$
- $v \propto d$
- $F_B \propto a$
- $a \propto d$

< / >

Relations between environment and system

<TODO>

- **Wheel Rail Friction**
- **Track gradient**

< / >

2.3 What to monitor

<TODO> Gehört das zu Modellvariablen??

- **Brake force over time**
- **Brake pressure over time**
- **Deceleration**
- **Velocity**
- **Braking distance**

< / >

Chapter 3

Modeling of Train Operations

Introduction As has been noted in 1.3, it is necessary to model the braking process of freight trains. All modeling work has been performed with Matlab Simulink.

3.1 Initial Model

The initial model to be expanded upon describes a single braking process. It's sole input, apart from some constants, is pressure over time, meaning a distinct value ranging between 5 and 3.5 bar for every timestamp. For visualization, please refer to B. Let's take a look at the whole model first.

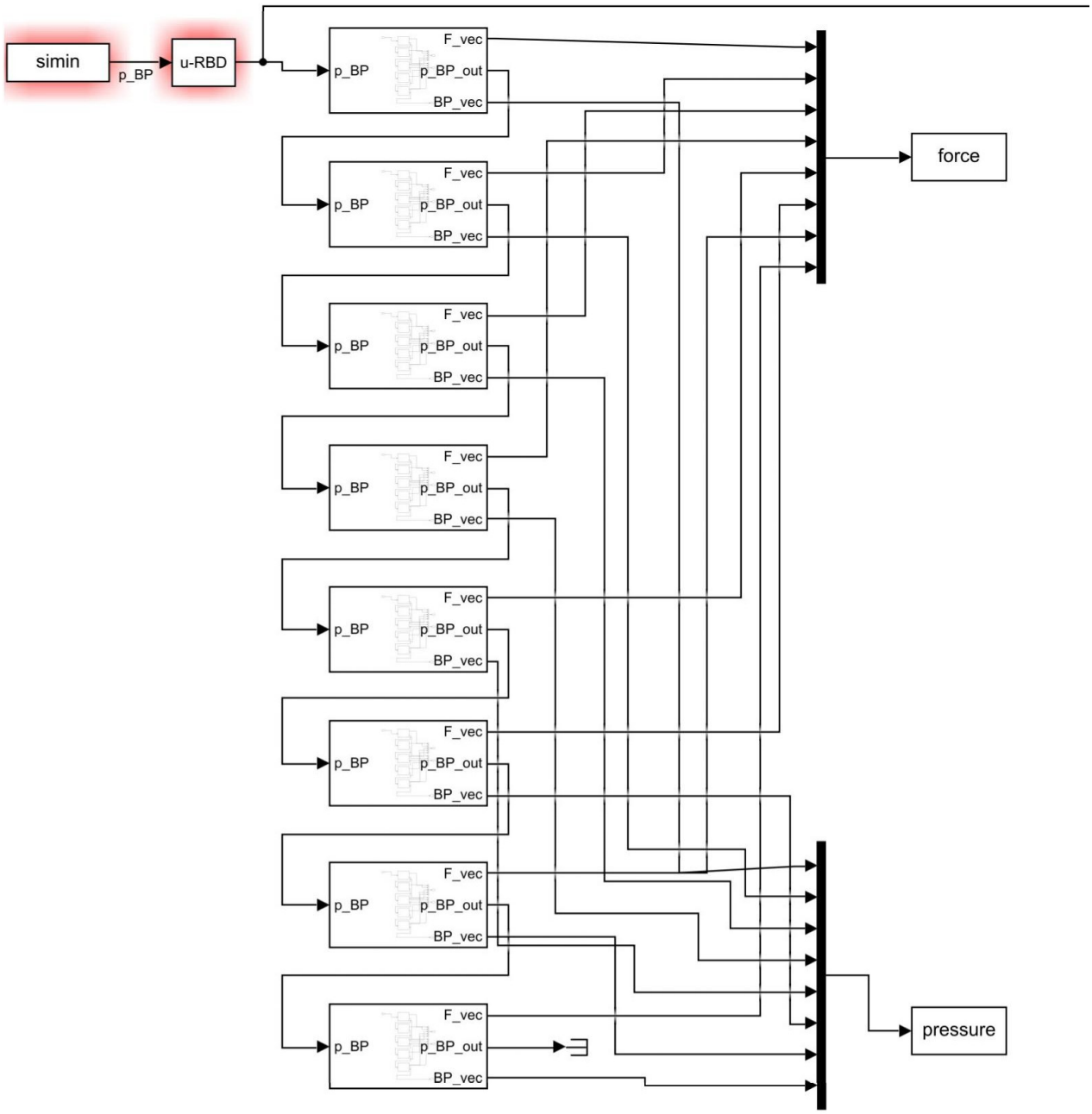


Figure 3.1: Initial Model

Here we see a model of a freight train of fixed length, consisting of 40 wagons, which are, for better readability, further condensed to subsystems of five wagons each, so there are eight of these subsystems. They are interconnected via braking pipe, which is also the sole input to each system. Outputs are braking pressure and braking force. We will take a look at the actual wagon model next.

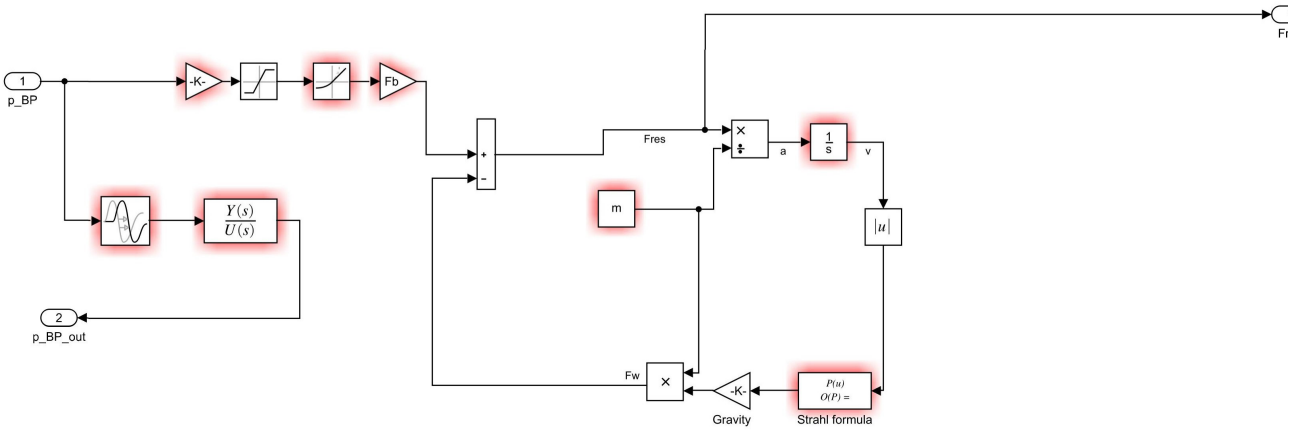


Figure 3.2: Initial Model - Wagon

Above is the initial wagon model. NOTE: All 40 wagon models are identical here. This will be addressed in section 3.2. It consists of three main components.

In the upper left corner is the input, which is the current pressure in the braking pipe. In the lower left corner, the propagation delay of the braking pipe is calculated. This is done by **<TODO> < / >**. Top center describes the calculation of the actual braking force, which is achieved by **<TODO> < / >**. Finally, the **<TODO> Formulieren: Fahrzeugwiderstand < / >**.

3.2 Model Expansion

This initial model is however not of sufficient detail. Where it merely describes one single braking process, we need to simulate a whole ride, with alternating phases of braking and accelerating. For that purpose, the simulation input has to be adjusted accordingly. Where previously it was only one braking process, using braking pressure as input was the obvious choice, whereas now the idea is to use a kind of track profile, which shall describe the maximum allowed velocity over time, of a notional track. For visualization, please refer to B. The simulation then only needs to brake or accelerate depending on train velocity versus maximum velocity at the current time.

Accordingly, the first expansion step is to create a mechanism to control the train so to speak. For this purpose, the system simply checks for each timestamp whether the current velocity of the train is greater than the maximum allowed velocity at the current time, according to simulation input. If this is the case, a braking pressure is applied to the pipe, scaling with the difference between v_{max} and v_{real}, v_{dif}. This means the higher v_{dif} is, the more braking pressure gets applied. This more or less covers the braking part of the system.

The model however also needs a component for acceleration. To simplify things, the logic here is that if the train is not braking, it is accelerating, which actually works out pretty well. To accelerate, a traction force is applied, which also scales with v_{dif}, so the higher v_{dif}, the higher the applied traction force.

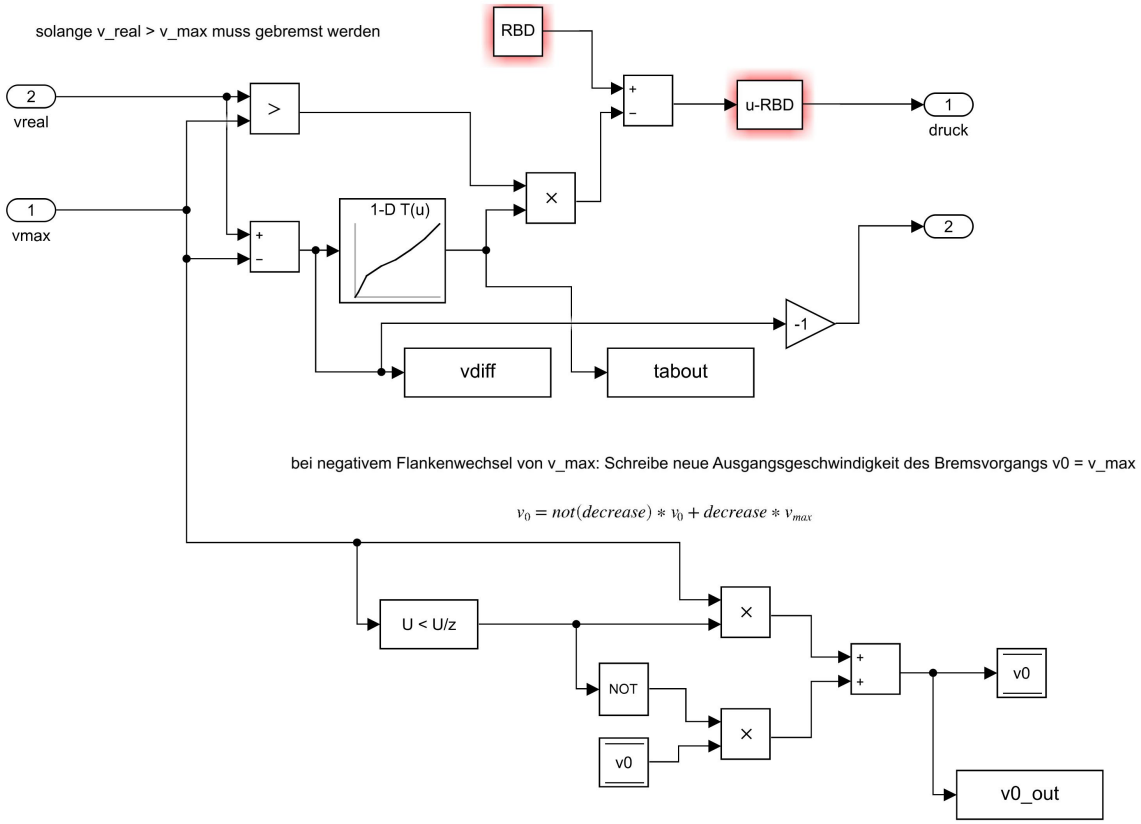


Figure 3.3: Expanded Model - Pressure Calculation

Depicted above is the system which determines braking pressure to apply. It calculates v_{dif} by subtracting v_{max} from v_{real}, which is then fed into a one-dimensional lookup table. The table is a sampled representation of a function with fixed breakpoints, mapping one function value to each breakpoint, like so

$$H(n) = \begin{cases} 0.1 & \text{if } n = 1 \\ 0.7 & \text{if } n = 15 \\ 0.8 & \text{if } n = 20 \\ .. \end{cases} \tag{3.1}$$

where n is the breakpoints of v_{dif}. Since the pressure should only be applied if v_{real} is greater than v_{max}, the ultimate result follows the logic of the following equation

$$P(n,t) = H(n) * (v_{real}(t) > v_{max}(t)) \tag{3.2}$$

where $v_{real}(t)$ is train velocity over time, $v_{max}(t)$ is maximum velocity over time, and $v_{real}(t) > v_{max}(t)$ is either 1 or 0.

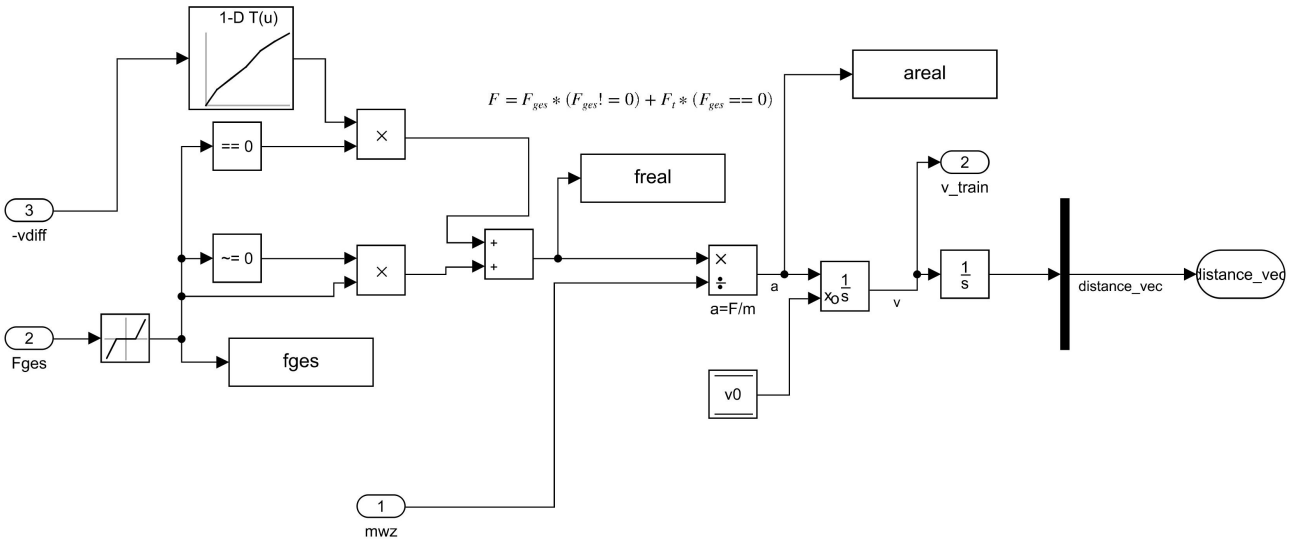


Figure 3.4: Expanded Model - Traction Force Calculation

The above system determines the traction force to apply. As has been discussed earlier, this works like a simple bang-bang controller. The design is very similar to the braking pressure system: v_{dif} is again fed into a one-dimensional lookup table, which outputs different values for traction force accordingly. The higher v_{dif} is, the higher the traction force to apply. It is then added to the current braking force, however only either traction or braking force is at any given time positive while the other is zero, which is achieved by the equations

$$f(n,t) = H(n) * (F_B(t) == 0) \tag{3.3}$$

where n is v_{dif}, $H(n)$ is the lookup table function (see equation 3.1), $F_B(t)$ is the braking force over time, and

$$g(t) = F_B(t) * (F_B(t) \neq 0) \tag{3.4}$$

where $F_B(t)$ is the braking force over time, so we have

$$F(n,t) = f(n,t) + g(t) \tag{3.5}$$

where F is the actual force over time, either braking or traction.

F is then used to calculate acceleration. According to Newton's Second Law,

$$F = m * a \tag{3.6}$$

Accordingly, acceleration is

$$a = F(n,t)/m \tag{3.7}$$

where m is the accumulated mass of all wagons and $F(n,t)$ relates to equation 3.5. The acceleration is then used to calculate the velocity by integrating a in relation to v_0, which is the initial velocity of the current braking or acceleration process **<TODO> überprüfen..** < / >. Integration of v in turn allows calculation of the traveled distance.

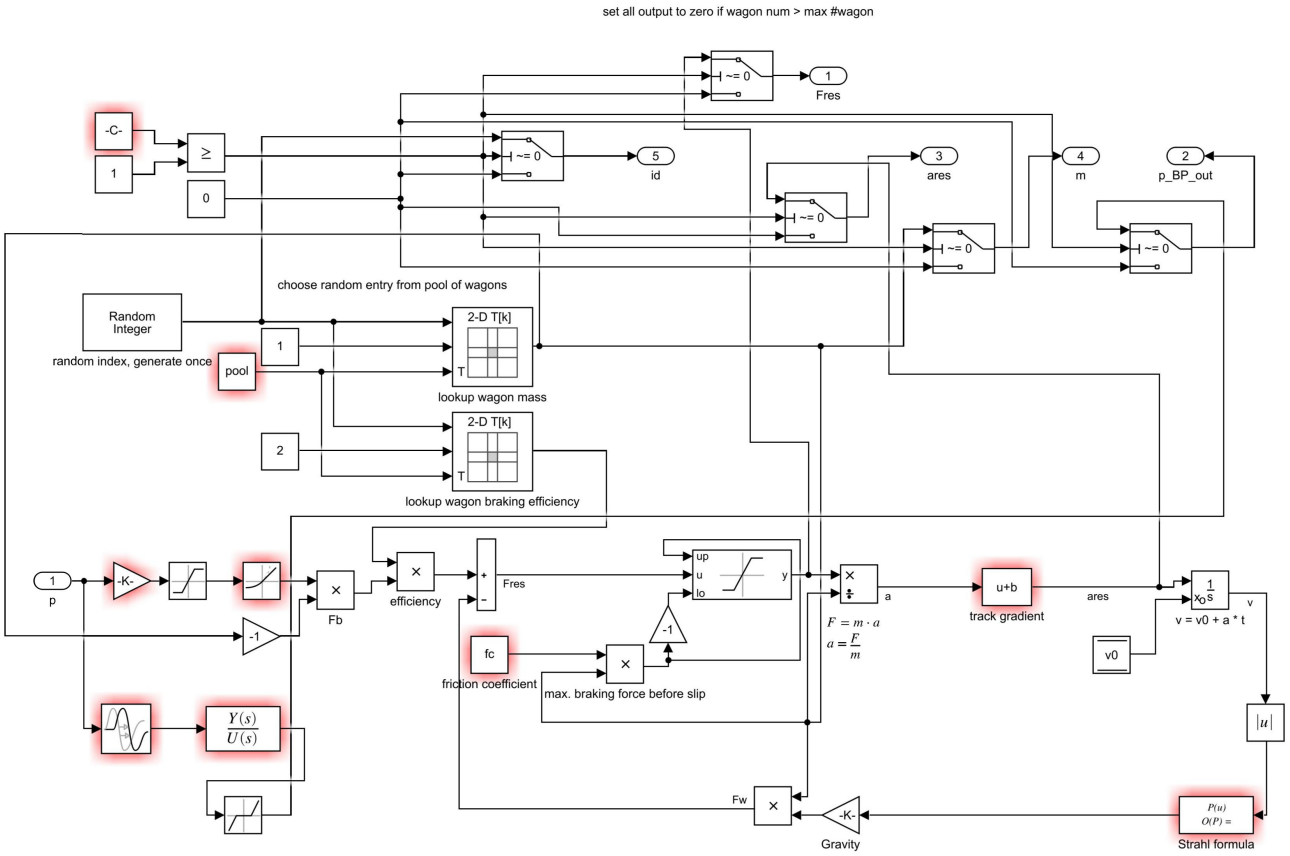


Figure 3.5: Expanded Model - Wagon

The last subsystem is the actual wagon. We will take a look at the largely unchanged elements first. The sole input is still the braking pressure on the brake pipe. Simulation of the propagation delay has also remained the same as before.

One new addition is a pool of different wagons. Whereas before all 40 were distinguishable only by their position, they are now assigned with different parameters. To that end, a pool of 500 wagons has been randomly generated via python script, where each wagon has a unique ID, as well as randomly generated mass and braking efficiency. In actual simulation, up to

40 of these 500 are, currently by generation of random indices, selected and their properties used accordingly. It would also be possible to determine the wagon ids to be used beforehand, instead of choosing randomly.

Another requirement was to make the number of wagons variable. In the initial model, the modeled train had a fixed number of 40 wagons, therefore also 40 wagon subsystems. Unfortunately, simulink offers no way to disable certain subsystems dynamically, but only by manually turning them off via model explorer, which would be unfeasible for such a large number of simulations. To circumvent this issue, output gets disabled for all unwanted wagons. For a simulation of a train of 20 wagons, the first 20 remain untouched, while the latter 20 produce no output and therefore also have no impact on the overall simulation. The turning off is achieved by simple switches; each wagon subsystem has a unique index from one to forty. If the index is greater than the specified number of wagons, all switches are turned to output zero.

3.3 Further Expansion

Chapter 4

Data Generation

Introduction With the finished model, it is now possible to generate the actual data, via simulation. For this purpose, a matlab script initializes all relevant model parameters and feeds the simulation input, in this case the previously discussed track profile, into the model.

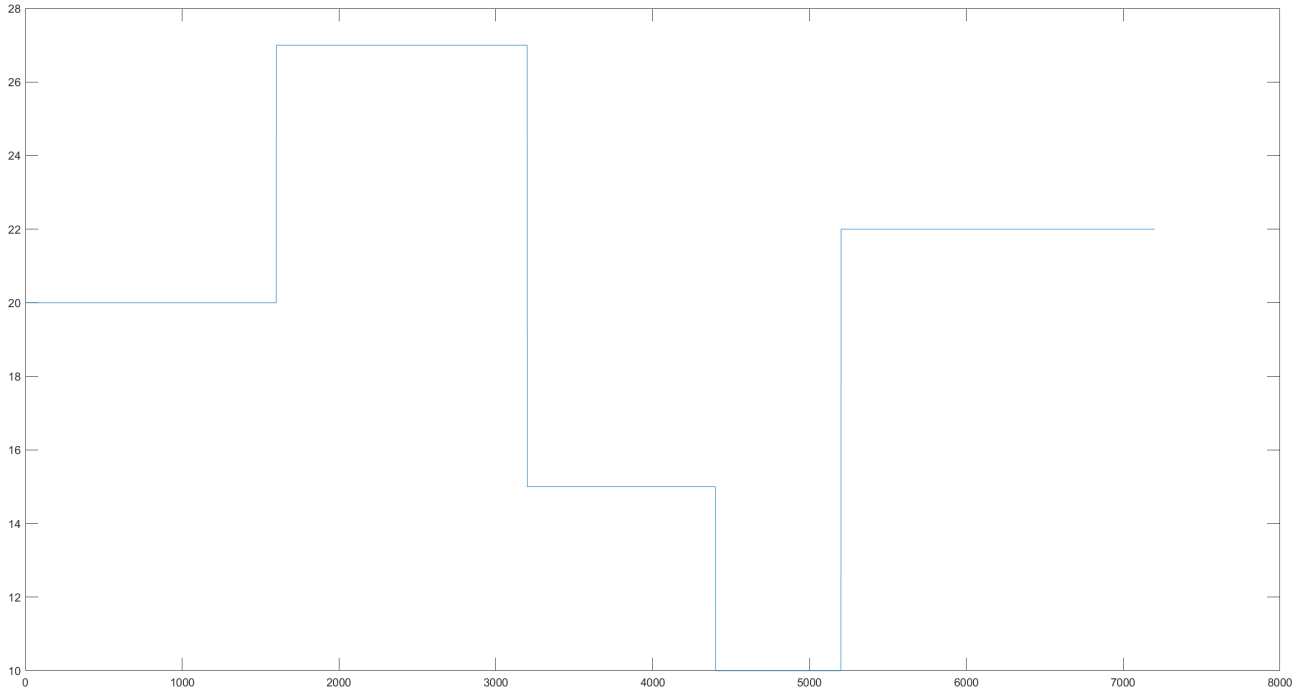


Figure 4.1: Simulation Input

The track profile, as depicted above, basically describes at which time which maximum velocity is allowed, and thus the train either brakes or accelerates to match that value at all times. What we can gather from this is that the simulation is time-based. Here we have 7200 data points as in input, which is 3600 seconds times two for better precision.

4.1 Matlab Code

We need to do some general configuration first, like clearing the workspace off all output which might still be present from previous simulations.

```
%% Initialise
clear all
clc
warning ('off','all');
numcores = 10; % configure number of cores to use in parallel processing
```

We then need to initialise all constants which values remain the same for all simulations. Please refer to the code snippet below for descriptions:

```
%% Constants

RBD = 5; % regular operations pressure (bar)
VBD = 3.5; % full breaking pressure (bar)

l = 18; % wagon length (meters)
c = 250; % propagation velocity (km/h)

tf = 4; % brake cylinder filling time
tl = .1;

p0 = 0; % initial pressure
Pres = 0/1000*[5.7/771 0 1.6]; % strahl formula for m/s velocity

alpha = 0.9;

BPnum = [0.3 alpha];
BPden = [1 alpha];

%% Simulation input
tmax = 3600; % simulation time (seconds)
nmax = tmax * 2; % number of data points
t = linspace(0, tmax, nmax); % simulation time input
u = [20*ones(800*2,1); 27*ones(800*2,1); 15*ones(600*2,1); 10*ones(400*2,1);
   22*ones(900*2,1); 0*ones(100*2,1)]; % track profile
simin.time = t; % set simulation time input
simin.signals.values = u; % set simulation signal input

trackgradient = 0; % pitch angle
```

After having initialised all necessary constants, we now get to configuring variable simulation input.

```
pool = readmatrix('pool.csv'); % read wagon pool
mkdir output; % make output directory

wagons = 1:1:40; % create array for number of wagons to loop over
friction = 0.05:0.01:0.78; % create array for friction coefficient to loop over
tracforce = 200000:1000:400000; % create array for traction force to loop over

track = 1; % counter to index input array for one batch of simulations
allruns = 1; % counter to keep track of total number of simulations completed

for i = length(tracforce):-1:1   % all possible combinations of traction force,
   for j = length(friction):-1:1  % friction coefficient and
      for k = length(wagons):-1:1 % number of wagons
         % save all previously initialised constans and sim input into
         % one array in
         in(track) = Simulink.SimulationInput('Simulation_v2');
         in(track) = in(track).setVariable('num_wagons',wagons(k));
         in(track) = in(track).setVariable('fc',friction(j));
         in(track) = in(track).setVariable('Ft',tracforce(i));

         in(track) = in(track).setVariable('alpha',alpha);
         in(track) = in(track).setVariable('BPden',BPden);
         in(track) = in(track).setVariable('BPnum',BPnum);
         in(track) = in(track).setVariable('c',c);
         in(track) = in(track).setVariable('l',l);
         in(track) = in(track).setVariable('nmax',nmax);
         in(track) = in(track).setVariable('p0',p0);
         in(track) = in(track).setVariable('pool',pool);
         in(track) = in(track).setVariable('Pres',Pres);
         in(track) = in(track).setVariable('RBD',RBD);
         in(track) = in(track).setVariable('simin',simin);
         in(track) = in(track).setVariable('t',t);
         in(track) = in(track).setVariable('tf',tf);
         in(track) = in(track).setVariable('tl',tl);
         in(track) = in(track).setVariable('tmax',tmax);
         in(track) = in(track).setVariable('trackgradient',trackgradient);
         in(track) = in(track).setVariable('u',u);
         in(track) = in(track).setVariable('VBD',VBD);

         % also save the respective values for traction force and
         % friction coefficient into arrays
         Ft(track) = tracforce(i);
         fc(track) = friction(j);

         track = track + 1;
      end
```

```
    end

    fprintf('Starting␣pool␣for␣%d␣simulation␣runs.\n',length(in));
    parpool(numcores); % create a pool of #numcores workers for parallel processing
    out = parsim(in,'ShowProgress','on');
    % start parallel simulations for all
    % entries of in, save output to out

    matrix = []; % create an empty matrix

    parfor l = 1:1:length(out)
    % parallel loop over out, which is an array that
    % holds all simulation outputs

       tmp = Write(l + allruns,out(l).get('velocity'),out(l).get('force')
          ,out(l).get('pressure'),out(l).get('distance'),
          out(l).get('acceleration_neg'),out(l).get('ids'),t,u,
          trackgradient,Ft(l),fc(l)); % for each entry, pack all output into one
              single matrix
       matrix = [matrix;tmp]; % append to matrix
    end

    allruns = allruns + length(out); % update total number of simulations

    writematrix(matrix,'output/output.tsv','FileType','text','WriteMode','append',
       'Delimiter','tab');
    % write matrix, which holds all output matrices of the
    % current batch, to a tsv file

    track = 1; % reset batch index
    delete(gcp('nocreate')); % delete parallel pool
    fprintf('Run␣%d␣(of␣%d␣total)␣complete.\n',length(tracforce)-i,length(tracforce))
        ;
end
```

As our aim is to generate large quantities of data, it becomes feasible to use use parallel processing for our simulations. For this purpose, we need to create an array to hold input for all simulation runs. There are three variables which change for each run, they are traction force of the locomotive, wheel/rail friction coefficient and number of wagons, respectively. A nested loop is used to create all possible combinations of these variables, and each single combination gets stored, as a new entry, in the array which holds all input.

Variable ranges are from one to 40 for number of wagons, .05 to .78 for friction coefficient and 200000 to 400000 for traction force, with step sizes one, .01 and 1000 respectively. We therefore have a total number of $40 * 74 * 200 = 592000$ combinations. The obvious approach here would be to generate the full batch of 592000 input configurations, but performance and

hardware limitations proved this to be unfeasible. Instead, we use rather small batche sizes of 2960 for parallel simulation.

Writing of the output is also done in smaller batches. The best approach to minimize the number of file accessions is of course writing everything into one big matrix, thus storing in RAM, and then writing the matrix in one go. Unfortunately, growing an array by assignment or concatenation can be expensive. For large arrays, MATLAB must allocate a new block of memory and copy the older array contents to the new array as it makes each assignment. The other extreme would of course be to write every single output line directly, but the large number of file accessions required makes this even more unfeasible, so a middle ground is needed, and a batch size of 3000 works relatively well, although this could surely be optimized with a bit of runtime analysis.

The function below compresses all output of one single simulation into one matrix. Raw simulation output consists of a few time-series objects, like braking force and braking pressure, as well as some metadata. These objects are fed into the write function, which does as many loops as there are rows in the time-series objects. It then packs all rows from each time-series into one big row, adds the meta-data and appends the large row to the matrix, which gets returned at the end. Refer to section 4.2 for a more detailed look at the structure of the output matrix.

```
function ret = Write(id,velocity,force,pressure,distance,acceleration,wagon_ids,t,u,
    grad,ft,fc)
   numrows = get(force,'Length');
   wagon_ids = double(wagon_ids);
   time = force.Time;
   matrix = [];

   for i = 1:1:numrows
      f = getdatasamples(force,i);
      p = getdatasamples(pressure,i);
      v = getdatasamples(velocity,i);
      a = getdatasamples(acceleration,i);
      d = getdatasamples(distance,i);

      row = [id,time(i),f,p,v,a,d,wagon_ids,grad,ft,fc];  % TODO: add simulation
          input aka track profile to matrix
      %  print wagon ids as list delimited by colon (,)
      %  performance?
      matrix = [matrix;row];
   end
   fprintf('Created␣output␣matrix␣for␣simid␣%d\n',id);
   % writematrix(matrix,'output/output.tsv','FileType','text','WriteMode','append','
       Delimiter','tab');
   % ret = 1;
   % fprintf('Create output matrix for Simulation ID %d\n', id);
   ret = matrix;
```

```
end
```

```
import csv
import random

with open('pool.csv', 'w', newline='') as pool:
   fieldnames = ['Wagon_ID', 'Mass', 'Braking_eff']
   writer = csv.DictWriter(pool, fieldnames=fieldnames)

   writer.writeheader()
   for i in range(0,500): # create 500 randomized wagons
      mass = random.randrange(12000, 90000, 100) # create random wagon mass between
          12 t and 90 t with step size 100
      braking_eff = round(random.uniform(0.75, 0.95),2) # create random braking
          efficiency as uniform distribution between 75 % and 95 %
      writer.writerow({'Wagon_ID': i, 'Mass': mass, 'Braking_eff': braking_eff})
```

The above python script is used to create a randomized pool of 500 wagons to be used in simulation. Each wagon has a unique id, a randomized mass, ranging between 12000 and 90000 kilograms, and a braking efficiency, which is a uniform distribution between .75 and .95.

4.2 Data Structure

As has been denoted previously, generated data should be suitable for big data processing. The first approach was to create a new directory for each iteration, and also save the different parameters to different files. Since this is very inefficient in terms of number of file accessions and therefore negatively impacts performance, as well as being unsuitable for transformations to big data file systems like Apache Hadoop or Apache Hive, it has been proposed to write all output into one single file. Let's take a look at the structure first.

simID	Timestamp	F_{wagon0}	...	P_{wagon0}	...	v_{wagon0}	...	a_{wagon0}	...	Distance	Wagons	Tra

Some output objects are time-series, namely braking force, braking pressure, acceleration, distance and velocity. Force, pressure and acceleration, which get measured for every wagon, look like this

Timestamp	$Wagon_1$	$Wagon_2$	...	$Wagon_{40}$
0	0	0	0	0
10	x_1	x_2	...	x_{40}
...	...	...	...	...
3600	y_1	y_2	...	y_{40}

whereas distance and velocity only have two columns and thus look like this

Timestamp	Value
0	0
..	..
3600	x

Since they all posses the same number of lines, which is the number of timestamps at which measurements were taken, all columns from all these objects may be compressed into a single table. All columns containing meta-information, for example simulation id, which theoretically only needs to be printed once, get filled with the same value for all lines, and since the number of wagons may vary from one to 40, all possibly empty columns get filled with zeros, so that we do not get any columns containing no value.

4.3 Analysis of generated Data

Chapter 5

Performance Analysis

Chapter 6

Conclusion

List of Figures

List of Tables

Appendix A

Quellcode

1. Source 1
2. Source 2

Appendix B

Data visualization

<TODO> Visualisierungen einfügen < / >

Initial model - simulation input Expanded model - simulation input