TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Verkehrswissenschaften "Friedrich List" Institut für Automobiltechnik Dresden – IAD

Lehrstuhl Kraftfahrzeugtechnik

Entwicklung einer Längsdynamik-Gesamtfahrzeugsimulation am Beispiel eines "Formula Student" Rennfahrzeuges

Hans Blättermann

Born on: 1st December 1996 in Radebeul

Matriculation number: 4514066

Matriculation year: 2015

Interdisciplinary project work

Supervisor

Dipl.-Ing. Stefan Plaettner

Supervising professor

Prof. Dr.-Ing. Günther Prokop

Submitted on: 31st May 2020

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Verkehrswissenschaften „Friedrich List“ - Institut für Automobiltechnik Dresden - IAD

Lehrstuhl Kraftfahrzeugtechnik

Aufgabenstellung für eine Interdisziplinäre Projektarbeit

Bearbeiter: *Hans Blättermann*
Matrikel-Nr.: *4514066*
Studiengang: *Maschinenbau, Raumfahrtsystemtechnik*

Thema: **Entwicklung einer Längsdynamik-Gesamtfahrzeugsimulation am Beispiel eines “Formula Student” Rennfahrzeuges**

Konventionelle Simulationstools eignen sich für benutzerdefinierte Untersuchungszwecke hinsichtlich der Editierbarkeit und dem notwendigen Berechnungsaufwand nicht sehr gut. Deshalb soll im Rahmen dieser Arbeit eine dedizierte GFZ-Simulation entwickelt werden, um z.B. die Längsdynamik eines Formula Student Fahrzeuges genauer analysieren zu können. Die Simulationsumgebung soll dabei auf die, für die Längsdynamik, notwendigen Parameter reduziert und vollständig “Open Source” entwickelt sein. Zusätzlich bietet dies die Möglichkeit des Erprobens von spezielleren Lösungsverfahren, die unter anderem eine bessere Recheneffizienz z.B. durch Multithreading ermöglichen können.
Das Ziel der Simulation soll es sein, die Rundenzeit (“Laptime-Optimization”), sowie existierende Kräfte und Bewegungen am GFZ über den Verlauf einer Fahrt (z.B. entsprechend dem Acceleration Event) zu berechnen und darzustellen zu können. Dies soll dazu dienen Sensitivitätsanalysen durchzuführen, um dadurch Abhängigkeiten zwischen Parametern für anschließende Optimierungsaufgaben zu erkennen.

Folgende Schwerpunkte sind zu bearbeiten:

- Literaturrecherche zu Software-Architektur und Lösungsansätzen von Gesamtfahrzeug-Simulationsumgebungen
- Implementierung/Verifizierung der numerischen Lösungsmethode für Mehrkörpersimulation und der Modellsubsysteme (Fahrwerk, Antrieb)
- Analyse der Simulationsumgebung (Simulationsperformance und Stabilität)
- Absicherung anhand Vergleich mit Realfahrdaten eines Formula Student Fahrzeuges
- Wissenschaftliche Dokumentation der Ergebnisse

Betreuer: Stefan Plaettner (TU Dresden LKT)

ausgestellt am: 01.03.2020
einzureichen am: 31.05.2020

Prof. Dr.-Ing. Günther Prokop
Betreuender Hochschullehrer

Declaration of originality

I confirm that I have written the present work independently and have not used any other sources and aids than those indicated. I am submitting it for the first time as an examination paper. I am aware that an attempted fraud will be punished with the grade "insufficient" (5,0) and may lead to exclusion from further examinations in case of repetition.

Name:	Blättermann
First name:	Hans
Matriculation number:	4514066

Dresden, 30.05.2020 .....

Hans Blättermann

Abstract

Conventional simulation tools are often not suitable for specialized investigation purposes. Therefore, a full-vehicle longitudinal simulation for racecars is developed within the scope of this thesis. To create a future-oriented design, special emphasis is put on extensibility and adaptability.
To meet these requirements a *Multibody simulation* (MBS) is used as a vehicle model and for its numerical integration, the Störmer–Verlet method is implemented.
For vehicle sub-models in general a theoretical approach was chosen, but the tires are modeled by the experimental *Magic Formula Tire* model. To take care of a correct implementation multiple verifications based on analytical and measured data are performed for sub-parts of the model. In a final accuracy examination, the simulation is compared to competition data of the real world *Formula Student* racecar Lille.
The simulation is executed in an open-source software framework developed from the ground up, which includes a user-friendly graphical user interface. A software architecture study was conducted to optimize its modifiability and performance.

Kurzreferat

Herkömmliche Simulationswerkzeuge sind für spezialisierte Untersuchungszwecke oft nicht geeignet. Daher wird im Rahmen dieser Arbeit eine Längsdynamik-Gesamtfahrzeugsimulation für Rennfahrzeuge entwickelt. Um ein zukunftsorientiertes Design zu schaffen, wird besonderer Wert auf Erweiterbarkeit und Anpassungsfähigkeit gelegt.
Um diesen Anforderungen gerecht zu werden, wird als Fahrzeugmodell eine *Mehrkörpersystem* genutzt (MKS). Als numerische Integrationsmethode, wird das Störmer-Verlet Verfahren implementiert.
Für Fahrzeugteilmodelle wurde im Allgemeinen ein theoretischer Ansatz gewählt, die Reifen nutzen jedoch das experimentelle *Magic Formula Tire* Modell. Um eine korrekte Implementierung zu gewährleisten, werden mehrere Verifikationen auf der Basis analytischer und gemessener Daten durchgeführt. In einer abschließenden Evaluierung der Genauigkeit, wird die Simulation mit Wettkampf-Daten des realen *Formula Student* Rennwagens LillE verglichen.
Die Simulation wird in einem von Grund auf neu entwickelten Open-Source-Software-Framework ausgeführt, dieses beinhaltet auch eine benutzerfreundliche grafische Benutzeroberfläche. Zur Optimierung dessen Modifizierbarkeit und Effizienz wurde eine Software-Architekturstudie durchgeführt.

Contents

Appendix

List of Figures

List of Tables

List of Symbols

Symbol	Unit	Description
m	kg	Mass
$\vec{J}$	$kg \cdot m^2$	Moment of Inertia
$\vec{r}$	m	Position
x	m	Position along x-axis
y	m	Rotation along y-axis
z	m	Rotation along z-axis
γ	rad	Rotation around x-axis
Ω	rad	Rotation around y-axis
Ψ	rad	Rotation around z-axis
$\vec{v}$	$\frac{m}{s}$	Velocity
$\vec{a}$	$\frac{m}{s^2}$	Acceleration
$\vec{F}$	N	Force
$\vec{M}$	Nm	Torque
t	s	Time
p	Pa	Pressure
Θ		Error
η		Efficiency
d	m	Diameter
C_1		1. Integration constant
C_2		2. Integration constant
g	$\frac{m}{s^2}$	Acceleration due to gravity
κ		Slip ratio
$F_{MFT,x}$	N	MFT longitudinal force
$F_{MFT,y}$	N	MFT lateral force
$F_{MFT,z}$	N	MFT load
$M_{MFT,x}$	Nm	MFT overturning moment
$M_{MFT,y}$	Nm	MFT rolling resistance moment
$M_{MFT,z}$	Nm	MFT self aligning moment
X		MFT input variable
Y		MFT output variable
B		MFT stiffness factor
C		MFT shape factor
D	N	MFT peak values
U		MFT curvature factor
S_H		MFT horizontal shift

S_V	N	MFT vertical shift
$K_{x\kappa}$		MFT brake slip factor
κ_x		MFT adjusted slip
$F_{MFT,z0}$	N	MFT nominal (rated) load
$F'_{MFT,z0}$	N	MFT adapted nominal load
df_z	N	MFT normalized change in vertical load
μ_x		MFT lon. friction coefficient
r_e	m	MFT effective rolling radius
r_0	m	MFT unloaded tyre radius
$V_{MFT,0}$	$\frac{m}{s}$	MFT reference velocity
$V_{MFT,r}$	$\frac{m}{s}$	MFT forward speed of rolling
γ^*_{Wheel}	rad	MFT spin due to chamber angle
λ_{Fz0}		MFT SF nominal (rated) load
$\lambda_{\mu x}$		MFT SF peak friction coefficient X
λ_{Cx}		MFT SF shape factor X
λ_{Ex}		MFT SF curvature factor X
$\lambda_{Kx\kappa}$		MFT SF brake slip stiffness
λ_{Hx}		MFT SF horizontal shift X
λ_{Vx}		MFT SF vertical shift X
λ_{Mx}		MFT SF overtuning couple stiffness
λ_{My}		MFT SF rolling resistance moment
p_{Cx1}		MFT factor pCx1
p_{Dx1}		MFT factor pDx1
p_{Dx2}		MFT factor pDx2
p_{Dx3}		MFT factor pDx3
p_{Ex1}		MFT factor pEx1
p_{Ex2}		MFT factor pEx2
p_{Ex3}		MFT factor pEx3
p_{Ex4}		MFT factor pEx4
p_{Kx1}		MFT factor pKx1
p_{Kx2}		MFT factor pKx2
p_{Kx3}		MFT factor pKx3
p_{Hx1}		MFT factor pHx1
p_{Hx2}		MFT factor pHx2
p_{Vx1}		MFT factor pVx1
p_{Vx2}		MFT factor pVx2
q_{sy1}		MFT factor qsy1
q_{sy2}		MFT factor qsy2

q_{sx1}		MFT factor qsx1
q_{sx2}		MFT factor qsx2
q_{sx3}		MFT factor qsx3
γ_{Wheel}	rad	Wheel inclination angle
Ω_{Wheel}	rad	Wheel rolling angle
Ψ_{Wheel}	rad	Wheel turning angle
$\vec{s}$	m	Position vector with origin in the COG of the Mainbody
$\vec{l}$	m	Position vector with origin in the COG of the wheel
M_{Motor}	Nm	Motor torque
m_{Tyre}	kg	Mass of a tyre
m_{Rim}	kg	Mass of a rim
$m_{Suspension}$	kg	Mass of a suspension
$m_{Suspension,n}$	kg	Mass of suspension n
$m_{Assembly}$	kg	Mass of motor, wheel carrier and transmission
m_{Wheel}	kg	Mass of a wheel combined
J_{rot}	$kg \cdot m^2$	MOI of the rotating parts of the wheel in Y-axis
$J_{Rotor/Transmi}$	$kg \cdot m^2$	MOI of the motor rotor and transmission Y-axis
c_{Tyre}	$\frac{N}{m}$	Spring coefficient of the tyre
d_{Tyre}	$\frac{kg}{s}$	Damper coefficient of the tyre
p_{Tyre}	kPa	Tyre pressure
$m_{MainCombined}$	kg	Mass of a Mainbody, Driver and part of the suspension combined
$m_{Mainbody}$	kg	Mass of a Mainbody
m_{Driver}	kg	Mass of a Driver
n		Enumeration count variable
$\gamma_{Mainbody}$	rad	Mainbody roll angle
$\Omega_{Mainbody}$	rad	Mainbody pitch angle
$\Psi_{Mainbody}$	rad	Mainbody yaw angle
$h_{Mainbody}$	m	Vertical height of the Mainbody
e		Controller input
u		Controller output
κ_{eff}		Motor controller set slip ratio
P_{out}		PID controller proportional term
K_p		PID controller proportional factor
I_{out}		PID controller integral term
$e_{i,max}$		PID controller maximal integral value
$e_{i,min}$		PID controller minimal integral value
K_i		PID controller integral factor
D_{out}		PID controller derivative term

K_d		PID controller derivative factor
C_l		Downforce coefficient
C_d		Drag coefficient
$C_l\alpha$	$\frac{1}{rad}$	Downforce pitch coefficient
$C_d\alpha$	$\frac{1}{rad}$	Drag pitch coefficient
A	m^2	Aerodynamic area
ρ_{Air}	$\frac{kg}{m^3}$	Air density
z_{offset}	m	Z axis offset
$\vec{c}$	$\frac{N}{m}$	Stiffness vector
$\vec{d}$	$\frac{kg}{s}$	Damping vector
d_c	$\frac{kg}{s}$	Critical damping coefficient
c	$\frac{N}{m}$	Spring constant
t_{final}	s	Track time
$c_{RealSus,z}$	$\frac{N}{m}$	Real suspension Z stiffness constant
$d_{RealSus,z}$	$\frac{kg}{s}$	Real suspension Z damping constant
E	kJ	Energy
W	kJ	Work
E_{mech}	kJ	Mechanical energy
w_{Joint}	kJ	Joint Work
E_{Solid}	kJ	Solid energy
w_{total}	kJ	Total work
E_{total}	kJ	Total energy
$E_{Mainbody}$	kJ	Mechanical energy of the Mainbody
E_{Wheel}	kJ	Mechanical energy of a Wheel
W_{Motor}	kJ	Work done by a Motor
W_{Tyre}	kJ	Work done by a Tyre
$W_{Aerodynamic}$	kJ	Work done by the Aerodynamic
t_{Sim}	s	Simulation time

*

List of Indices

Definition of Indices

$\vec{V}_i$	Physical property vector $\vec{V}$ of component i
$\dot{\vec{V}}_i$	1. Derivative of time of physical property vector $\vec{V}$ of component i
$\ddot{\vec{V}}_i$	2. Derivative of time of physical property vector $\vec{V}$ of component i
$\vec{V}_{i,j}$	Value of j-direction of physical property vector $\vec{V}$ of component i
$\dot{\vec{V}}_{i,j}$	Value in j-direction of 1. derivative of time of physical property vector $\vec{V}$ of component i
$\ddot{\vec{V}}_{i,j}$	Value in j-direction 2. derivative of time of physical property vector $\vec{V}$ of component i
$\vec{V}_{i,j,k}$	Value of j-direction of physical property vector $\vec{V}$ of component i in position k
$\dot{\vec{V}}_{i,j,k}$	Value in j-direction of 1. derivative of time of physical property vector $\vec{V}$ of component i in position k
$\ddot{\vec{V}}_{i,j,k}$	Value in j-direction of 2. derivative of time of physical property vector $\vec{V}$ of component i in position k

List of Component Indices i

A	Generic component A
B	Generic component B
Spring	Generic spring
Mainbody	Component Mainbody
MainCombined	Combined component of Mainbody and Driver
Driver	Component Driver
Suspension	Component Suspension
Wheel	Component Wheel
Rim	Component Rim
Tyre	Component Tyre
Assembly	Combined component of motor, wheel carrier and transmission
Aero	Component Aerodynamic

List of Direction Indices j

x	x-Axis
y	y-Axis
z	z-Axis

List of Position Indices k

1	Front left position on the car
2	Rear left position on the car
3	Rear right position on the car
4	Front right position on the car

*

List of Abbreviations

PID Proportional–Integral–Derivative 51
MFT Magic-Formula-Tyre IX
SIL Software-in-the-loop VII
HIL Hardware-in-the-loop 5
MIL Model-in-the-loop 11
3D Three Dimensional VII
CAD Computer-Aided Design 4
MOI Moment of Inertia IX
FS Formula Student XI
TV Torque Vectoring 2
TCS Traction Control System 2
DOF Degrees of Freedom 5
MBS Multibody Simulation IX
FEM Finite Element method 5
DAS Driver-Assistance Systems 5
MFT Magic Formula Tyre IX
COG Center of Gravity 41
GUI Graphical User Interface VI
ODE Ordinary Differential Equation 15
MVC Model-Viewer-Controller 18
OOP Object-Oriented Programming XI
OO Object-Oriented 22
CPU Central Processing Unit 19
UML Unified Modeling Language X
Pub-Sub Publisher-Subscriber 21
QML Qt Modeling Language 22
ECS Entity Component System 22
ang. angular 24
lon. longitudinal IX
max maximum 38
min minimum 38

1 Introduction

1.1 General

This work will be about simulations, so let us start by defining simulations:
"A *simulation* is the imitation of the operation of a real-world process or system over time" [2] In other words, it is able to represent the evolved state from an initial state after a certain time. The change of the system over time is described by a simulation *model*. This is usually based on a set of assumptions and simplified approximations. However, this also means that validation is a key aspect in the development of a simulation tool. [2]

But why make all that effort?
Validated simulations have the advantage of being able to give answers to the "what if" of complex questions in real-world systems. That would otherwise require a lot of time, expensive prototyping, thousands of variant tests, or simply could not be answered. Simulations allow small groups or even individuals to study and experiment with these systems to gain deep insight and understanding of the problems in a relatively short amount of time. For technical problems, they are particularly useful in the design stage of a project, where they allow the study of a not yet built system or can predict the impact on performance when changes are introduced. For these reasons they are widely used and accepted tools in system analyses. [2] They are therefore also of great benefit in the design and tuning process of vehicles, which will be the general topic this thesis will be about. In particular, the development of a simulation tool for vehicles under racing conditions, as it involves a lot of advance development and therefore optimization (e.g. laptime optimization) is a key point.

1.1.1 Application Example

In order to consider real-world requirements in the development of this simulation tool, the FS Team of the University TU-Dresden "Elbflorace" [12] was selected as exemplary user/partner. FS is an international challenge in which student teams build a single-seated formula racecar to compete in several competitions. Since the annual design process is an important part of the competition, the team has a great interest in a suitable simulation tool that supports and underlines their decisions. The team takes part in two separate FS competitions the FS-Electric and FS-Driverless. Both vehicles have multiple software controllers that could benefit from SIL testing and optimization. [29]

Together with them, a catalog of requirements was drawn up, which will guide the development of this tool. In general:

- Laptime simulation, to make a rough estimate of the influence of parameters on driving performance (e.g. weight vs. C_l).
- To provide data for sensitivity analysis to support concept development and design.
- Test environment for the driverless driver model, especially for the path and controller software (SIL)
- Test environment for a preliminary design of a controller (e.g. (SIL) for Traction Control System (TCS) / Torque Vectoring (TV))

While not all of these requirements are covered in this thesis, the techniques and processes used for the simulation structure shall support their future integration. The detailed requirements will be further explained in the following sections.
The team provides the necessary data for validation purposes. The test vehicle is their 2018/2019 competition vehicle "LillE".

1.1.2 Sensitivity analysis

An important use case of simulations in the design and concept phase is the sensitivity analysis. This defines a collection of mathematical methods used to investigate the relationship between input parameters and the output of a mathematical model. [26] It helps to easily gain a deeper understanding of how vehicle parameters influence each other and the vehicle's performance. These parameters can be for example rim mass, aerodynamics down-force as well as selected tire, motor control system and many others. While it is usually possible to determent the influence of a single factor, it becomes quite difficult to evaluate the impact of a decision that changes two or more parameters simultaneously, like the introduction of an additional aerodynamic wing, which allows more downforce on the one hand, but also increases weight and drag on the other. To make the right decision about the shape and size of this imagined wing for a minimal laptime, the right balance between these input parameters must be found (which also depends on a given driving scenario). An answer to this question can be found by combining a suitable simulation with a sensitivity analysis.
For a simulation tool to be used for this method, it must allow frequent system changes, provide a good insight into the value and how they changes over time to provide the possibility to compare the different system states later.

1.1.3 Software-in-the-loop

Another use case of simulations in the tuning process of vehicles is the possibility to use them in a SIL set-up. This is a process in which software is integrated into a replication of the surrounding technical system in order to perform tests on it. For vehicles, an implementation could be the testing of driver assistance software in vehicle simulation. Currently, an increasing part of the

software is used in vehicles that have high requirements in terms of correctness, safety and reliability. This means that many tests have to be carried out under all possible situations in which this system may get into. Which without SIL, can only be started when the hardware is operational. This reduces the possibility of parallelization in the development process and thus the time that can be spent on each individual task. Another advantage of SIL is that tests are easily repeatable and comparable since the boundary conditions of the simulation can be precisely defined and known. [19]

1.1.4 Simulation Scope

In order for a simulation to fulfill the described tasks, its model must contain all important factors that are to be analyzed or influence the outcome of the by systems' self-perceived values. Each factor on its own must be considered on its possible approximations or simplifications. This must be done as otherwise, it would quickly inflate the tool beyond its practical limits in terms of development time and computational complexity without adding an appropriate level of precision or requiring additional information. [25]:
To abbreviate the integral parts of the model developed in this thesis, the specific objectives and scope of the tool must first be defined. In summary, it should be a simulation for racing cars to optimize lap time. It must contain all the general physical relations required for vehicle movement, appropriately approximated or simplified, in other words, it will be a complete vehicle model. The included parts will be defined later. For SIL purposes as well as driving optimization, it will contain a motor controller algorithm that can be exchanged or adjusted. Additionally, there should be room for the possibility to later integrate a driver model or an interface to an autonomous driving system (for SIL) together with its sensors needed.
Since the development of such a system is overall quite extensive and would in its completeness go beyond the scope of this work, especially since some kind of verification is required for each part included, it will concentrate exclusively on the longitudinal movement. A driving scenario covered by this simulation could be an acceleration race on a linear path. This decision has no influence on the general simulation process or the tool design. It introduces some minor simplifications in the physical model but reduces the verification process by a significant amount. An additional reason is that a complex driver model would otherwise be needed for steering, which again would increase the development and verification effort. Nevertheless, care should be taken to ensure that the possibility of lateral movement is maintained during the design, development and implementation process of this tool. So that it can be added later without major changes. Consequently, the simulation model is developed for a 3D space.

1.1.5 Simulation Model

In this section, the simulation model will be introduced in more detail. To analyze this type of system, either a subsystem or a full-vehicle simulation (complete vehicle model simulation)

is performed. While a multi-domain simulation has the disadvantage of maximum effort in terms of parameterization, validation and computation time, only it enables the integration and analysis of multiple subsystems and their mutual influence. [5]
A complete vehicle model is composed out of the following system elements [25]:

1. Chassis
2. Suspension
3. Wheels
4. Powertrain
5. Brakes
6. Steering
7. Vehicle control [22]

For the specific application case of a longitudinal dynamics simulation and the requirements (subsection 1.1.1) must be focused more strongly than others. The major parts in the simulation model are the **chassis** (includes its aerodynamic parts), the **suspension**, the **wheels** (rim and tyres), **vehicle control** (all motor controller). Meanwhile, **Steering** is not covered, since no lateral movement is allowed. The **powertrain** is reduced to its direct torque on the wheels. Since the requirements are about electric vehicles without shift transmission and the electric motor is not part of this thesis, the **brakes** are similarly reduced because their rare influence in acceleration scenarios.
A separate simulation sub-model will be determined for each of the included subsystems. To describe this kind of dynamic behavior of real processes there are generally two ways to do so [25]:

- The theoretical approach uses physical laws to abbreviate the mathematical model.
- The experimental approach is usually also based on mathematically structured models. For this approach, specific models are then parameterized by using input and output data from real-world experiments.

For this work, both options will be used. For each vehicle subsystem, an appropriate variant must be chosen. The decision will be based on the availability of needed parameters, accuracy and if it is fitting the requirements of the system. Typical sources for these parameters are:

- Computer-Aided Design (CAD) - Models [25]
- Direct measurement [25]
- Assumptions/Estimations (especially for values like friction coefficient) [25]

- Calculated from other data source (like simulations or characteristic maps) [25]
- Specification from manufacturer

Another important part of the simulation model is its complexity and thus its computational costs. Although there are no direct requirements for real-time simulation, a short or even faster then real-time computation time will be pursued. This enables fast generation of data required for laptime as well as controller optimization. And could also be of advantage for Hardware-in-the-loop (HIL) / SIL purposes. Real-time simulation means that the calculation of a timestep must not require more time than the timestep itself. Therefore, the simulation must be sufficiently fast to finish the calculation before the event itself is completed. [28] This problem is also strongly influenced by the computing hardware and is taken into account when designing the software architecture.

1.1.6 Types of Vehicle Models

For modeling the dynamic behavior of vehicles there is a range of approaches available. They differ from each other in terms of complexity, amount of Degrees of Freedom (DOF) and thus in their ability to take different physical effects into account. In Table 1.1 a list of common vehicle model types and their DOF is presented. The MBS model subsystems mentioned there is representative for modules such as powertrain, brake, steering, or mechatronic systems like Driver-Assistance Systems (DAS). [25]

Types	DOF
Single-track model linear	2
Single track model non linear	3
two-track model	4-30
complex MBS	> 20
MBS model subsystems	
Finite Element method (FEM) model	> 500
Hybrid model	> 500

Table 1.1: Model types of vehicles [25]

In this thesis, the focus is on MBS systems. They are well suited for a mechanical system consisting of rigid bodies connected through bearings and joints. Loads are mainly introduced by individual forces or torque in discrete points on the body. [24]
The Figure 1.1 shows typical elements of MBS together with an example from automotive engineering. In this model type, only rigid bodies (hereafter also called *Solids*) have mass and thus inertia. Bearings and joints are therefore massless and serve to transport forces between bodies. Actuators can be realized by forces or forced movement. The mathematical description of MBS system kinetics and kinematics is based on ordinary or differential-algebraic systems of equations. [25]

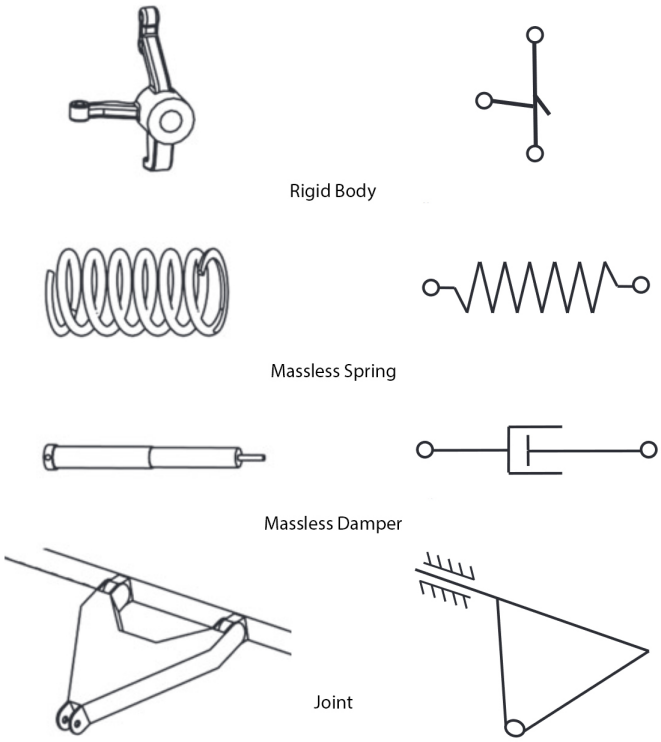


Figure 1.1: Typical elements of MBS [25]

1.1.7 Tyre Model

Wheels in general and tyres, in particular, are of decisive influence when it comes to describing the dynamics of vehicles. Apart from the aerodynamic forces, the tyre-road interaction is the only way to influence vehicle movement. Its main functions are [25]:

- Transmission of the normal wheel load and protecting against shocks
- Transmission of lateral forces
- Transmission of longitudinal forces

For the modeling of tyres, several types have been developed in the last century equivalent to vehicle models. These all serve different purposes and are therefore unequal in their accuracy and complexity. This is also evident when comparing the different types of approaches used for their development, which are presented in Figure 1.2. The approaches of the left-hand category are based on characteristics tables or mathematical formulae with data from full-scale tyre experiments. A well-known example is the MFT model. The more approaches are tending towards the right side, the more they are based on physical theory about the structure of tyres. An example of a complex physical model is a FEM tyre model. This would be less suitable for vehicle motion simulations and more for tyre performance associated with its design. [21]
In this thesis, the MFT model is used. Reasons for its use are its relatively high accuracy in the values considered by vehicle dynamics as well as its low computational effort and its low

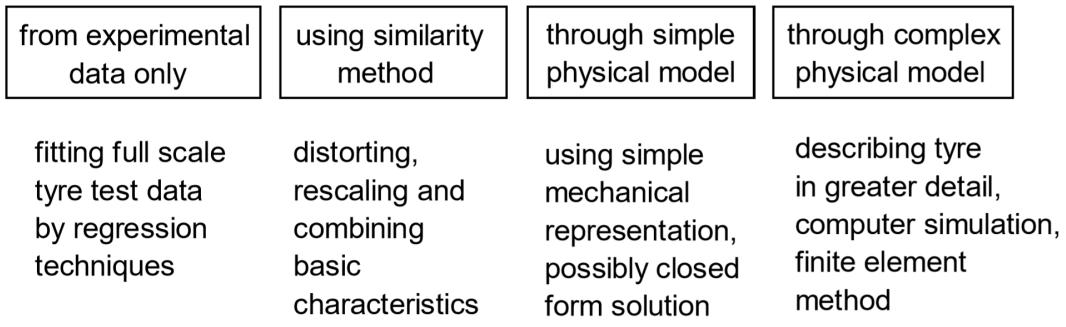


Figure 1.2: Approaches to develop a tyre model [21]

complexity, which allows a more simple integration into the software. Another point is the availability of its parameters since it is widely used in motorsports. [21]

1.1.8 Model Requirements

This section discusses the model requirements that correspond to the subsection 1.1.1. They also include assumptions that are used for the development and testing of the simulation and should be considered when evaluating the results.
The following aspects are addressing the environment model:

- Driving scenario according to *Acceleration* event from the FS-Germany rule set [23].
  - The track is a straight line of 75m length and 5m width.
  - Accelerate starts from standing start.
- The track is flat, therefore it has an inclination and bank angle of 0°.
- The friction coefficient is variable(to account for different track and weather conditions).
- Wind is not considered.

Some requirement set for the vehicle model:

- All solids are rigid.
- The masses, COGs and MOIs of solids have to be adjustable.
- Play of joints and bearings do not need to be taken into account.
- Reduced driver model that
  - keeps the steering angle at 0°
  - and gives full accelerator pedal input when the race is started.
- The driver as a physical object is rigid.
- Aerodynamics is modeled as a function of speed.

Additional assumptions are defined during the development of the subsystem models and documented in the corresponding sections.

1.1.9 Software Requirements

This section discusses the software requirements that should guarantee an effective workflow and fulfill special needs for the development of racing vehicles. These aspects are defined as guidelines for the software architecture of the developed tool and for additional software that is used for development/adjustment or the general work process of the simulation. The following three are to be respected:

- Open source
- Efficient
- User friendly

The first requirement is to have a complete insight into all simulation processes and be able to adjust or even change them. This also means that the ability to profoundly customize the model and all other parts of the software must be a feature that must be considered in the development and when selecting tools, frameworks and other third-party software. The reason behind this is that modern applications such as motorsport development are characterized by frequent changes in vehicle design and concept, which are too specialized and fluid for a general simulation solution and therefore require models that are tailored to their specific requirements and can be changed/extended together with them. Also, insight into the assumptions and approximations of simulation must be available when used to extremely optimize a design.
These criteria certainly overlap with the open-source concept. This license guarantees, among other things, that the source code of an application is available and modifications are allowed.[27] Therefore, the software used for this work is generally available under an open-source license. An exception is the use of MATLAB for evaluation purposes.
For the second requirement, although it does not have to be explicitly executed in real-time, it is still aimed to be designed to run efficiently and the selected tools should be suitable for computing in performance-critical tasks.
The last requirement is the availability of a user-friendly interface and documentation of the software. The latter shall be achieved with this thesis. An integrated solution is aimed at, therefore the use of additional packages or tools of third parties is not necessary

1.1.10 Methodology

In section 1.1 the problem was defined, the advantages of simulations where discussed, an introduction to simulation models specifically for vehicles and tyres were given and the general requirements for the model and the software usage were specified. The next section 1.2

will give an overview of already existing vehicle simulation tools and compare them with the requirements.

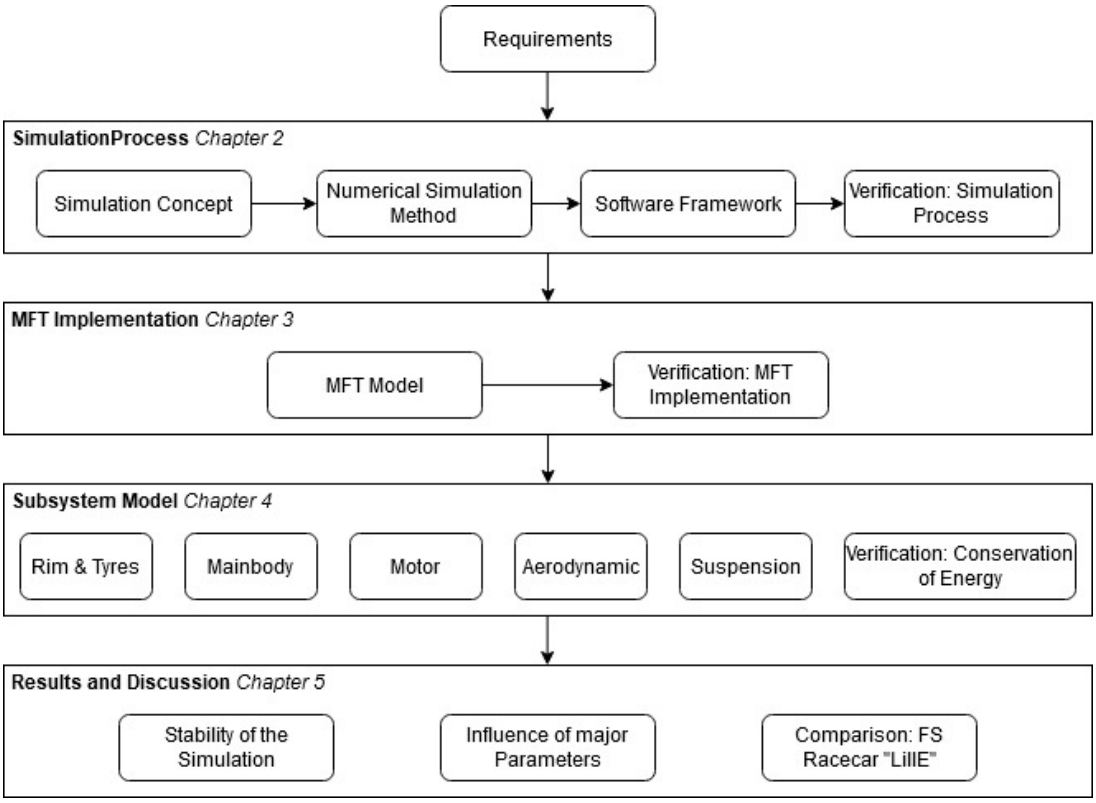


Figure 1.3: Development process of the simulation tool

In chapter 2, chapter 3 and chapter 4 the different modules of the simulation tool are implemented, shown in Figure 1.3. It starts with the introduction of the used simulation model concept and discussion of the numerical method used for solving it. This is followed by their integration and the development of the software framework. At the End of chapter 2 a verification of the general simulation process takes place. In chapter 3 the MFT model is explained in detail and implemented. For verification, the output of the model is compared with test data and data provided by the tyre manufacturer (Continental). The chapter 4 then deals with the description and development of the equations of motion for all solids and the subsystem. At the end of the chapter, there is again a verification part. It is about the conservation of energy within the simulation model.

The results of the simulation are then compared and discussed in chapter 5, with special emphasis on simulation stability and the influence of major parameters. A comparison to the real-world vehicle "LillE" of the FS team Elbflorace will also take place there.

1.2 State of Art

In this section, existing vehicle simulation tools are analyzed and compared in their functionality. The following are considered:

- CarMaker [6]
- wivw SILAB [10]
- AVSimulation [1]
- FSSIM [15]

Besides the first three commercial ones, also, the open source tool *FSSIM* developed by the FS team "Academic Motorsports Club Zurich" is compared. This is commonly used by Autonomous driving (AD) teams for driver model optimization.
Additionally, these simulations are compared to the simulation objective set for this thesis. The results are displayed in section 1.2. The information is obtained from their individual homepages [6][10][1][15].
In the property "Adjustability to FS scenario" the effort and possibility to use the tool in the specialized scenario of FS is estimated. The adaptability and insight into the simulation process also play a role here.

	CarMaker	wiww Silab	AVSimulation	FSSIM	Simulation objective
Open-Source	no	no	no	yes	yes
Vehicle model	unspecified AD/ADAS	complex physical model	unspecified AD/ADAS	Single-trackmodel	MBS AD/ADAS
Simulation application	Vehicle Dynamics Powertrain	Vehicle Dynamics	Vehicle Dynamics Human Factors Studies Driving Simulator	AD	Vehicle Dynamics
Simulation setup	MIL SIL Driving Simulator	MIL Driving Simulator	SIL HIL MIL	SIL	SIL MIL
Adjustability to FS scenario	+	0	+	+++	+++
Simulation performance	real-time	real-time	faster then real-time*	real-time	real-time

*Allows for massive simulation on High Performance Computer (HPC); Advanced driver-assistance Systems (ADAS); Model-in-the-loop (MIL)

Table 1.2: Comparison of existing vehicle simulations and the simulation objective

2 Simulation Process

This chapter focuses on the development of the simulation software framework, starting with the concept of a simulation model and the identification of a mathematical solving method. Thereafter a programming language, as well as a GUI toolkit, are selected and the general software architecture is developed. At this point, the importance and meaning of *multithreading* are explained as well.
Afterwards, the architecture is implemented and its important aspects are highlighted and discussed. Finally, the process is verified using a test scenario and its performance and accuracy are examined.

2.1 Multibody Simulation model

Since the MBS vehicle model is already introduced in section 1.1.6, the focus of this section is to go into more detail and explain its application for the existing simulation problem.
The main component of MBS are solids (rigid bodies), which means they are not deformable or flexible. Therefore their shape, size or structure are not relevant as long as the effect of their inertia is contained in the parameters of mass m and MOI $\vec{J}$. For this reason, solids in this work are treated as point particles with the two corresponding inertia parameters. As point particles in a reference system they also have a position (x ,x ,z), a velocity $\vec{v}$ and a rotation (section 2.4.3).
To abbreviate possible solids for this simulation application, all subsystems of the complete vehicle model (section 1.1.5) must be considered. Important aspects for the selection are that the solid must be single rigid objects or a group of objects which under simplifications can be considered as a single object. In addition, more solids increase the complexity and thus the computational effort. For these reasons as well as the requirements for the model (section 1.1.8), possible solids are a wheel, the mainbody (or chassis), and a quarter suspension. This means it could result in a 5-body or a 9-body MBS model (with or without four times a quarter suspension). This question is further discussed in section 2.4.3.
In MBS, the connection between two solids or a solid and the environment is modeled by joints, which introduce kinematic constraints or a force element, transmitting loads between them. An example of a joint is a one-dimensional rigid connection between two bodies. A force element can be a spring connection. The disadvantage of using joints that rigidly connect two bodies in their movement is that the independence of the bodies is lost. This is shown in the basic kinetic equation of such a system, like eq. (2.1).

$$\vec{a}_{AB,x} = \frac{\vec{F}_{A,x} + \vec{F}_{B,x}}{m_A + m_B} \tag{2.1}$$

Equation (2.2) shows that the two bodies behave like one for the connected direction.

$$\vec{a}_{AB,x} = \vec{a}_{A,x} = \vec{a}_{B,x} \tag{2.2}$$

Therefore, when the motion of the Solid A is calculated, all information about the loads $\vec{F}_B$ on Solid B must be known. This eliminates the separation of the individual solids and thus the possibility of information encapsulation, which can be of great benefit for software parallelization (further discussed in section 2.3.4). Especially for simplifying methods used for numerical solving (see section 2.2).
Meanwhile, force elements such as springs depend on the position (dampers depend on the velocity) of the connected solids, as shown in eq. (2.3).

$$\vec{a}_{A,x} = \frac{\vec{F}_{A,x} + \vec{F}_{Spring,x}(x_A, x_B)}{m_A} \tag{2.3}$$

This enables the preservation of information encapsulation for solids. In other words, every solid knows the loads acting on it and the position and speed of the solids surrounding it but not the loads of other bodies. For this reason, the joints in their definition described above and force elements are combined into a single connection type, hereafter referred to as *Joint*. These "new" *joints* enable the connection between two bodies only based on their relative position (springs) and speed (dampers) as well as the force-based connection to the environment. A classical rigid connection is therefore not possible. Similar effects might be achievable by spring and damper connections with extremely high coefficients. This problem is investigated in section 4.5.3. In this model, the applications of joints as solid to solid connection are the suspensions. Examples of joints as a connection of solids to the environment are aerodynamics and the connection of tyre to road. They are also used to model the connection of the motor with its software controller integrated to transmit the torque to the wheel.

In summary, the simulation model type used is an abbreviation of the MBS model. It consists of two different objects. The solid with its mass, MOI, velocity, position and orientation in a 3D space, and the joint for introducing loads on solids. These loads are based only on the information given by the solids and the environment. Therefore, no load depends on information from another joint.

2.2 Mathematical Methods

In this section, the methods for solving the equations of motion are discussed. In a MBS equations of motion can become quite complex because they describe the motion of several bodies over time and are influenced by each other and the environment. This can lead to differential-algebraic systems of equations. [25]

There are generally two ways to solve these systems, the analytical and the numerical. Analytical solutions have the advantage that they provide an exact solution. To implement them in a simulation tool, they must be solved in advance and the resulting formula must then be implemented in the simulation model. This has the advantage of less computational effort, but also the big disadvantage of a complex and time-consuming process of solving the system, which, depending on the complexity of the system, may not even be possible at all. Besides, every change to the overall system requires a complete or partial re-solving of the system. For a better explanation an example along Newton's second law of motion:

$$\vec{F}(t) = m\ddot{\vec{r}}(t) \tag{2.4}$$

Equation (2.5) is the analytical solution of the second-order linear Ordinary Differential Equation (ODE). It also shows the downside of the analytical method. The load function $\vec{F}(t)$ is still within two integrals, which contains every force affecting the body ($\vec{F}(t) = \sum \vec{F}(t)_{Joint}$). Therefore if an additional force is added or removed, the integrals must be solved again. This drastically lowers the ability to frequently introducing changes to the model.

$$\vec{r}(t) = C_2 + \frac{\int C_1\, m + \int \vec{F}\,(t)\; dt\, dt}{m} \tag{2.5}$$

The numerical solution of the equation of motion comes along with the idea of discretization of time. This also introduces a discretization error that depends on the time step duration Δt. A basic example, the Euler scheme, is shown in eq. (2.6). It is based on a shortened form of the Taylor extension. [16]

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{\vec{F}(t)}{2m}\Delta t^2 + \Theta(\Delta t^3) \tag{2.6}$$

Compared to the analytical solution, there is no differential operator influencing the load function of this solution. Therefore numerical solving allows much more flexibility in changing loads.

Since a lot of algorithms for integration of equations of motion are available, one has to be selected. General properties of good numerical algorithms are [13]:

- They are easy to implement, are efficient in their execution, and have low memory consumption.
- They fulfill the conservation of total energy, angular, and linear momentum.
- Their calculated trajectory should be reversible under time reversal.

The final decision was made for the Verlet Algorithm. The reasons for that are discussed in the next section.

2.2.1 Verlet Algorithm

The Verlet integration offers a lot of advantages compared to the simpler Euler method. First, it has greater stability and preserves the physical properties of a system. The algorithm is simple and needs only modest memory. Additionally, it offers time-reversibility and preservation of phase space. [16]
The original idea of the Verlet scheme is the sums of two third-degree Taylor expansion in two different directions of time in the position coordinate (shown in "Simulation Methods in Physics 1"[16]). Another way to obtain the Verlet integrator is through the second-order central finite difference for the acceleration, shown in eq. (2.7).

$$\vec{a}(t) = \frac{\frac{\vec{r}(t+\Delta t)-\vec{r}(t)}{\Delta t} - \frac{\vec{r}(t)-\vec{r}(t-\Delta t)}{\Delta t}}{\Delta t} = \frac{\vec{r}(t+\Delta t) - 2\vec{r}(t) + \vec{r}(t-\Delta t)}{\Delta t^2} + \Theta\left(\Delta t^4\right) \tag{2.7}$$

After rearrangement and inserting Newton's second law, this does result in the Verlet integration:

$$\vec{r}(t+\Delta t) = 2\vec{r}(t) - \vec{r}(t-\Delta t) + \frac{\vec{F}(t)}{m}\Delta t^2 + \Theta\left(\Delta t^4\right) \tag{2.8}$$

Meanwhile, this scheme does also have some downsides. It is not self-starting because it requires $\vec{r}(t+\Delta t)$, which is not available at $t = 0$. This problem is generally solved by using an initializing step like this[16]:

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{\vec{F}(t)}{2m}\Delta t^2 + \Theta(\Delta t^3) \tag{2.9}$$

Furthermore, the speeds must be calculated independently of the positions. While the positions have a higher order of accuracy of up to $\Theta(\Delta t^4)$, the calculation of the velocities is less accurate. They can be derived by [16]:

$$\vec{v}(t) = \frac{\vec{r}(t+\Delta t) - \vec{r}(t-\Delta t)}{2\Delta t} + \Theta(\Delta t^2) \tag{2.10}$$

Since the value of the acceleration is of interest for the evaluation of the simulation and therefore needs to be calculated, attention must be paid that the velocity is calculated for timestep $t + \Delta t$ and the acceleration at t. Therefore the acceleration vector is running one timestep behind the velocity.

As described in the previous section, every movement of a solid is independent from each other. Therefore every solid has its own motion equation depending on its position and loads, which needs to be known at t to calculate $\vec{r}(t+\Delta t)$. Loads/Joints depend on position as well as on the velocities of solids. Hence the velocity must be also known at t and the general approach of the

central difference (2.10) is not functional. To solve this problem, the Verlet scheme is altered and an forward difference is used for the velocities:

$$\vec{v}(t + \Delta t) = \frac{\vec{r}(t + \Delta t) - \vec{r}(t)}{\Delta t} + \Theta(\Delta t) \tag{2.11}$$

Since this again reduces the accuracy of the velocities, special attention must be paid to them when evaluating. Nevertheless, the higher error of the velocities should not matter too much since it is not directly used for the numerical integration and thus its error does not propagate.

2.3 Software Architecture

"The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both." [3] Its definition is the next important step in simulation tool development. It describes the framework in which the functionality is later implemented. The choices made while considering the software architecture determines to a large degree, the system's modifiability, extensibility, portability, performance, security, development efficiency and availability. Different types of software structures have individual advantages in these fields. There are three general types of software structures: [3]:

- Modules as structures, divide the system's functionality into multiple components.
- Dynamic structures, that focus on the mutual interaction between elements at runtime. For example, in a system built out of multiple services, the interaction, synchronization and infrastructure these services interact with, is meant by this kind of structure.
- Allocation structures, which are bound to the system's organizational, developmental, installation, and execution environment.

According to the requirements, important attributes for the simulation's architecture are modifiability, extensibility and performance. The following subsections introduce types that are used to develop the structure and explain how they are implemented. An overview of useful architectural structures is given in Figure A.1 [3].

2.3.1 Software Structure: Decomposition

Decomposition belongs to the modules of software structure and is often a starting point when designing the architecture. It is about dividing modules by functionality into smaller modules, which are again divided into other modules until they are small enough to be easily understood. These modules are related by a *is-a-submodule-of* relation. The decomposition of a project has a major influence on the system's modifiability. [3]

Decomposition structures also come along with abstraction[3], which is a fundamental concept in computer science. It is is about separating/encapsulation of information, with the goal to focus on details that are important to a task and hiding non-important information in separate places or removing them entirely. [7]
For the simulation tool, the decomposition structure was used as a coarse division into the major function of the program (described in subsection 2.3.2). These modules then use their individual software structure or design pattern. In practice, design patterns are solutions to common problems that are used rapidly. Their properties are well known and allow reuse. Compared to general structural parts, as mentioned in Figure A.1, design patterns are much more precise and restrictive and often combine several design decisions. Therefore, they are a very useful tool when it comes to creating software architecture and dealing with common problems. [3]

2.3.2 Model Viewer Controller

The three main modules, that initially divide the program are the *viewer*, the *model* and the *controller*. Their name and function correspond with the Model-Viewer-Controller (MVC) design pattern.
The MVC pattern is used to separate the GUI functionality and the application functionality, while still maintaining responsiveness for user input change in the underlying application data. Therefore it uses[3]:

- A *model*, in which the application data is contained.
- A *viewer* that displays the data to the user and allows user inputs.
- A *controller* which introduced changes to the model and manages input signals from a view.

While MVC introduces complexity due to the separation and therefore might not be the best solution for small projects with simple interfaces. It also has the big benefit of abstracting GUI from data and application functionality. This introduces a big advantage in maintainability and extensibility. [3]

2.3.3 Software Structure: Class

Classes belong to the module structure type as well. They are connected through a *inherits from or is an instance of* relationship. This means properties and functions from one class are passed to all its child classes. The class structure type is well suited to group modules with similar functions while abstracting general properties as well as functions to base-classes and parameterize/specialize them in child-classes. It highly increases the modifiability and extensibility of the total system.

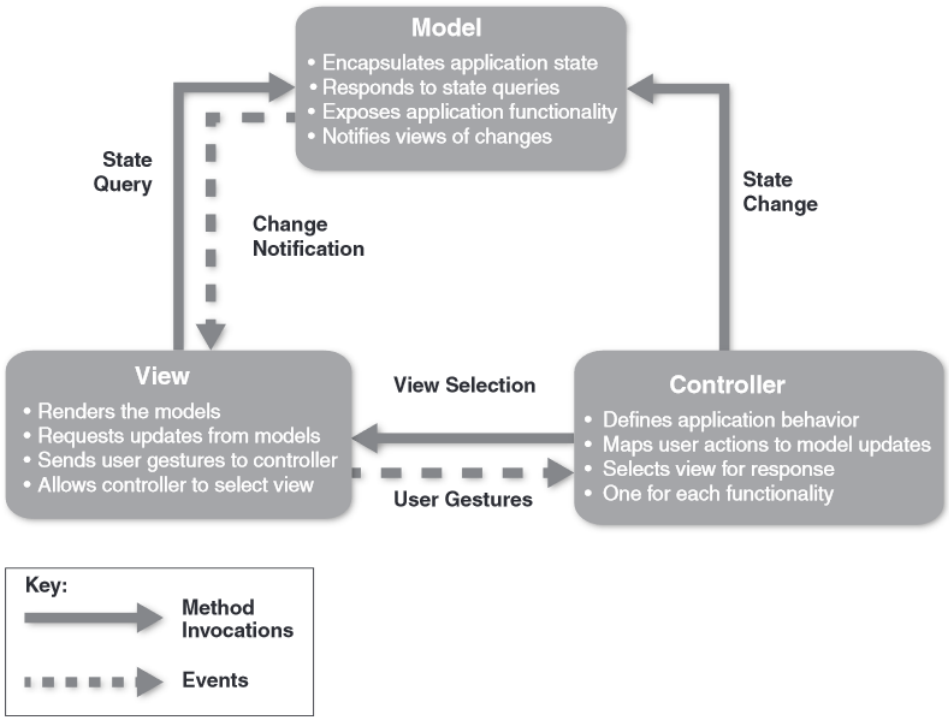


Figure 2.1: The model-view-controller pattern [3]

In general, this type of structure comes often along with OOP, where the classes describe an object type and instances of them can be created. These instances can contain individual states of data. [3]

For the simulation tool, classes are used as the backbone of architectural structures. They directly take over the structure after the split into the three main parts of MVC but also describe the tool in detail. One requirement of the programming language is, therefore, the support of the OOP style.

2.3.4 Software Structure: Concurrency

The third and last used software structure type is concurrency. It describes the ability of software to be executed in parallelism. In other words, multiple parts of the code are run at the same time. This can highly increase the performance. The improvements depend on how many Central Processing Unit (CPU) cores are provided to the application and how well the application algorithms scales. In optimum, the number of cores could be proportional to the performance. Nevertheless, parallelism does often introduce some inefficiency. [3]
First, not every algorithm can be parallelized. Many are sequential and have no parts that are mutually independent. Second, access to shared data must be treated carefully and must be well protected. Therefore, it must be prevented that two individual processes of execution (which are also called *Thread*) can access or manipulate one data at the same time, which can

lead to an undefined state of the program. This is also one reason, why concurrency must be planned in the architectural phase. Since connected modules and functions must be designed for *Multithreading*. And thirdly, many algorithms that support this cannot be entirely run in parallel execution and require phases in which *Threads* need to be synchronized. Therefore, more execution heavy *Threads* can bottleneck the entire system.

Since one numerical timestep for every solid can be calculated independently, the numerical Verlet algorithm together with a MBS, allow *Multithreading*. After one step the *Threads* are joined (which means it is awaited, that all have finished) and afterward the boundary values are saved and the next step can be calculated. For this process a *Multithreading* efficiency of $\eta_{Multithreading} = 50\%$ as well as an behaviour as in eq. (2.12) is estimated.

$$t_{Multithreading} = t_{Single} \cdot \eta_{Multithreading} \cdot (AmountCores - 1) \tag{2.12}$$

These estimates will be evaluated in the conclusion.

2.3.5 Controller

The controller module, as described by MVC, reacts to inputs from the viewer and updates the model. Its specific functions are:

- Implement Start, Stop, Pause and Reset functionality.
- Initialize parts of the model, dependent on the simulations scenario.
- Initiate and manage the simulation process.
- Distribute simulation process over multiple threads (concurrency).

Since the controller module is by far the smallest, no additional architecture besides concurrency is needed.

2.3.6 Model

The model provides data storage and data access control. Additionally, it includes the simulation model. The specifications for the model are:

- It provides storage for all the data, that should be stored, should be displayed, or manipulated by multiple instances.
- It's data access is thread-safe and accessible for the Controller and the Viewer.
- The model notifies about updated data.
- The simulation model is integrated in a way that is modifiable and extendable.

For the data part, design patterns were used again. The storage and access-type are defined by the *Shared-Data Pattern*. It describes one or more central shared-data stores that exchange persistent data with multiple modules. When data is used, an accessor reads information from the shared pool, processes it and writes the results back into the storage. This makes data persistent and supports access control. The disadvantage of this design is that it could represent a single point of failure. Also, it could lead to a performance bottleneck, since the calculation of all modules depends on the availability of its data. [3]
The second used design pattern is Publisher-Subscribe Pattern. It describes that multiple components can publish and subscribe to events. For example, it is used when one component provides information and other components need to be notified when this information changes to react accordingly. Therefore, according to the Publisher-Subscriber (Pub-Sub) pattern, infrastructure is created that allows the components to subscribe to events. If a publisher changes information or invokes another event, this infrastructure will distribute the event to all subscribers. The advantages of this system are again modifiability and extensibility because the components are unaware of each other's identity or even existence. Downsides are the predictability and latency of the event delivery. [3]

The data part of the model will combine both of these patterns. On one hand, all parts of the simulation algorithm and conducting components have access to the persistent data of the shared data. And on the other, parts like the viewer which need to be informed about changes in data are able to subscribe to them. In this application, the downsides of the Pub-Sub pattern also will not come into play since the view has no critical influence in the simulation process.
For the detailed architecture of the data part and the simulation model part the class software structure is used as well. It again supports the abstraction, modifiability and extensibility and is, accordingly, a good fit.

2.3.7 Viewer

The Viewer's duty is about displaying all the information going through the simulation in a user-friendly, clear and practical way. Additionally, is must enable the change of settings and parameters. Since the Viewer is not a major part of this work, its architecture must allow a fast and efficient development while still supporting the general requirements.
Therefore, a GUI framework is used. It provides general tools and presets to create user interfaces and accelerates the graphical development by a lot. Because these frameworks come with their own inbuilt architecture, no additional structure is defined upfront. Additional requirements that influence the choice of this toolkit are that it must match the chosen programming language for easy integration and that it must work well with huge amounts of data.

2.3.8 Programming Language Selection

Now that the software architecture has been defined, the next step is to select a suitable programming language. It must be well adapted to the simulation requirements and support the integration of the structure. Since the class structure approach is generally used, the language must support the Object-Oriented (OO) paradigm. It must also have excellent execution performance and should be widely used to provide easy access for changes and extensions. The following table shows a comparison of some languages:

Language	Performance	Popularity in %
Java	0	41.1
C++	++	23.5
C#	+	31.0
Python	- -	41.7
PHP	-	26.4
JavaScript (Node js)	0	67.8
Rust	++	3.2

Table 2.1: Comparison of some OOP languages

The performance comparison was taken from "Which programs are fastest?"[30]. There a comparable program was implemented in different programming languages. The results show that not only the chosen programming language determines the performance but also the correct implementation. However, since not too much time should be spent on optimizing the integration, a generally fast language is preferred. [30]
For the popularity the results of the 2019 developer survey from *stackoverflow*, which is one of the biggest software developer forums/communities, are used. [9]
The decision was made for C++ because of its outstanding performance. From the common OOP languages only Rust was comparable but has far worse popularity.

2.3.9 GUI Framework Selection

The decision for C++ as programming language greatly reduces the choices for GUI frameworks. There are only two major competitors Qt and GTK+, both provide an open-source license, cross-platform functionality and are easy to use.
The one advantage of Qt over GTK+ is its support for easy 3D rendering in its 2D interface with Qt3D. For this reason, the tools of QtQuick (2D) and Qt3D (3D) are used for the user interface development. Both are using an additional markup language called Qt Modeling Language (QML) and some JavaScript for GUI functionality. The software architecture of QtQuick is class-based and Qt3D uses the Entity Component System (ECS) pattern. This is about entities that contain one or more components describing their behavior. These components can be altered or exchanged at runtime and therefore the entity's actions are changed. This leads to greater flexibility. [11]

2.3.10 UML class diagramm

In the previous section, the entire software architecture used was described and explained. The programming languages are also specified. Thus, the prerequisites for the detailed implementation, which is the next step of development, have been created. This begins with the definition of all classes used, their attributes and functions. For this purpose, simplified UML class diagrams are used. They are shown in Figure A.3, Figure A.4, Figure A.5, Figure A.6. Since the Viewer uses a different programming language and the class structure is not fitting well, its class diagram Figure A.2 is further simplified and shows no attributes or functions. Afterwards, these classes are implemented into C++/QML/JavaScript code. Since complete documentation of the code would exceed the scope of this work, only software parts that are important aspects for the simulation process are discussed. This takes place in the next section 2.4. The complete project can be reviewed on the attached digital data storage or in the git repository [14].

2.4 Simulation Process

In this section, the detailed integration of the simulation process and its components are explained.

2.4.1 Data

The implemented *data* part of the model consists mainly of four classes (see Figure A.4):

The `class ValueBase` as base class implementing the functionality of the Pub-Sub pattern and the access control, protecting against parallel access to data by multiple *threads*. For the purpose of mutual exclusion it uses the *mutex* functionality. `ValueBase` are not meant to be directly instantiated, therefore it is an abstract class and its constructor is `protected`.

The `class Parameter` is an instantiable child of `ValueBase` and provides simple functionality to save a single parameter. It is meant to be used for values that keep constant over one simulation execution, like settings.

The counterpart is the `class Value` which is also a child of `ValueBase` but connects its values with a timestamp. Additionally, it saves every state and its corresponding timestamp over the course of one simulation execution. Therefore, it is used for values that change over time and whose rate change is of interest for later review. It also uses a lock-functionality, meaning that only an object that owns the key object is allowed to change its value. Finally, it also provides some well-optimized methods for providing data to the viewer.

The fourth major class of the model data section is the `class Data` it is a *singleton*, therefore only one instance of it can be created. It instantiates every Parameter and Value with their initial condition and gives every other class access to them. Consequently, it provides the functionality of the shared-data pattern.

2.4.2 GUI

This section gives a short overview of the Graphical User Interface, shown in fig. 2.2. As displayed, the coordinate system was selected according to the DIN 70000. Important additional information is that the GUI calculation and update is executed in an independent GUI-thread.

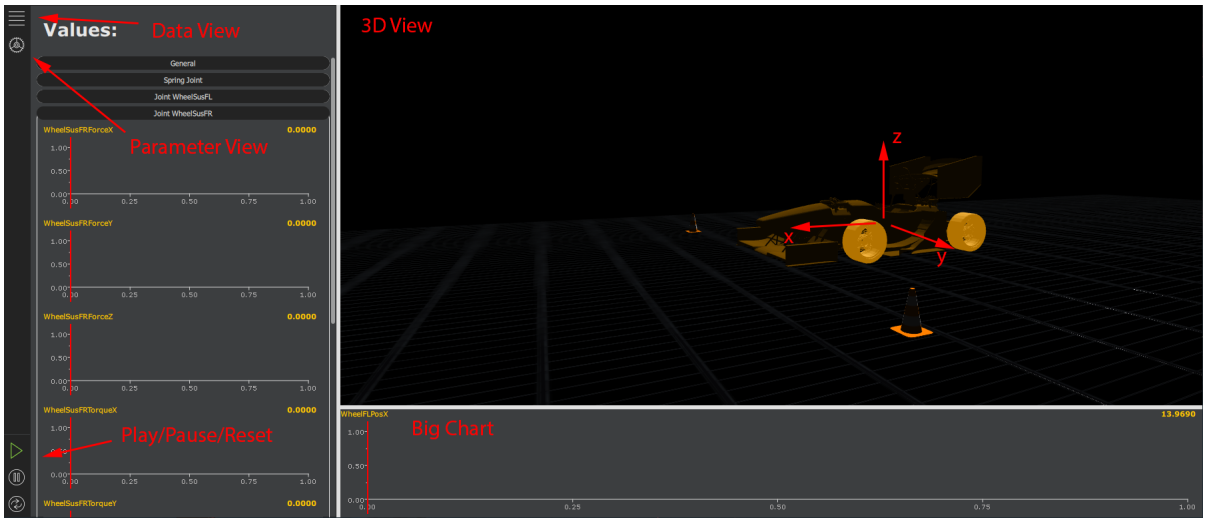


Figure 2.2: Overview Viewer

Appendix B.1 contains a short manual for the usage of the GUI.

2.4.3 Solid

The base `class Solid` corresponds to the eponymous object of MBS described in section 2.1. Its UML class and its child classes are shown in Figure A.5. To accomplish its functionality, the class provides the following attributes:

- `Value &posX, &posY, &posZ` representing their position vector $\vec{r}$
- `Value &veloX, &veloY, &veloZ` representing their velocity vector $\vec{v}$
- `Value &accelX, &accelY, &accelZ` representing their vector $\vec{a}$
- `Value &eulerX, &eulerY, &eulerZ` representing their rotation around their mass center γ, Ω, Ψ
- `Value &angularAccelX, &angularAccelY, &angularAccelZ` representing their angular (ang.) velocity $\dot{\gamma}, \dot{\Omega}, \dot{\Psi}$

- `Value &angularAccelX, &angularAccelY, &angularAccelZ` representing their ang. acceleration $\ddot{\gamma}, \ddot{\Omega}, \ddot{\Psi}$

Their three main simulation functions are:

```
virtual Solid* calculateAcceleration(double timestep) = 0;
```

This is a virtual method, needing to be implemented individually by the child classes of `Solid`. It is called to calculate acceleration $\vec{a}$ at t_n. In general, the `Solid` retrieves the reaction forces from its connected `Joint` classes and then uses the individual underlying formula to calculate the acceleration.

```
Solid* calculateNextPosVelo(double timestep)
```

This method integrates the numerical Verlet integration as described in section 2.2.1 and thus calculates $\vec{v}$ and $\vec{r}$ at t_{n+1} (with $t_{n+1} = t_n + \Delta t$).

```
Solid* updateValues(double timestep);
```

The two previous methods both save their results in internal data buffers and are not publishing their new information directly to the shared-data storage. This is necessary to keep data persistent in the same timestamp until all `Solid` and `Joint` classes have finished their calculation. Afterward, this method is called to push the new values into the storage.

5 vs. 9 Solid System

There is still an open question about the total amount of solids in the MBS system. The decision has to be made between two possible configurations, a 5-solid configuration, including the wheels and the main body, and the 9-solid version, which additionally includes a separate body for wheel carrier/suspension. A brief comparison of these two configurations is shown in the table below:

	9-Solids	5-Solids
Accuracy*	0	0
Complexity	-	+
Fit class structure**	+	-
Performance***	0	+
	0	1

Table 2.2: Variant comparison: 9/5-Solid model

*The accuracy influence is difficult to estimate as on one hand, a 9-Solids version could better integrate complex behavior but on the other, more solid connections are introduced possibly leading to lower accuracy if the same level in detail, compared to a 5-Solid system, is used.

***Fit to class structure*, describes how well the physical model fits the software structure. The 5-Solid variant has the disadvantage of combining rotating objects (Tyre/Wheel/some Motor

components) and from the rotation decoupled parts (wheel carrier/suspension). This would need some workaround to fit.

*** The 5-Solids version provides a minor performance advantage, since less bodies need to be calculated.
For this work, the 5-Solid version is used. Decisive is mainly the lower complexity and thus the lower development and verification effort. For a generally more detailed approach, the 9-Solid version would certainly be advantageous.

Quaternions vs Euler Angles vs. Rotation Matrix

Quaternions, Euler angles and rotation matrices are all systems of orientation of solid bodies in coordinate systems. For the decision a variant comparison is used again:

	Quaternions	Euler angles	Rotation Matrix
Calculation efficiency	+ +	+	0
Complexity	- -	+	-
Human readable	-	+	+
Gimbal lock	+	-	+
	0	2	1

Table 2.3: Variant comparison: Orientation of bodies in coordinate system

Quaternions are based on an extension of the complex number system for four-dimension. A rotation using quaternion is, therefore, based on four parameters. Quaternions are widely used in computer graphics because of their calculation efficiency. Meanwhile, there complex mathematical nature makes it hard to get an intuition of the described rotation solely by considering its parameters.[18].
Here too, the less complex solution is preferred, the Euler angles are used. Especially since an angle Ψ is not used for pure longitudinal movement and therefore problems such as gimbal lock occur far less frequently
In later experiments, a combination was tested in which the inner state was represented by quaternions and the conversion to Euler angles was made for output, comparison and some calculation reasons. With more investigation, this system could give better results than Euler angles alone. Because of time reasons, this solution was not pursued further. It was discovered that the quaternions do not contain information about the total amount of full rotations around an axis, which is problematic. This is shown in fig. 2.3, showing the simulation output of a forward rolling wheel when quaternions are used internally and converted back to Euler angles.

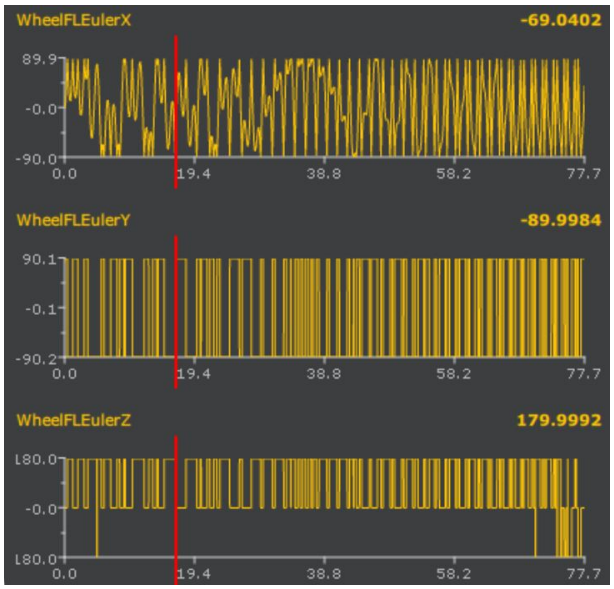


Figure 2.3: Quaternion conversion to Euler angles

2.4.4 Joint

The `class Joint` is like the `class Solid` the pendant to the eponymous Joint in the MBS system and tries to implement its functionality into a class. The `Joint` is also a basic class, and therefore the concrete realization of its physical relation, how the inner force is to be calculated, takes place in its child classes. This is done by overwriting the abstract `Joint` function:

```
virtual void calculate(double timestep) = 0;
```

As shown in Figure A.6 there are three major child classes:

- The `class AeroJoint` introduces aerodynamic drag and lift to the *mainbody*.
- The `class MotorJointBase` represents a combination of motor and motor controller and creates a torque on a wheel.
- The `class SusJointBase` connects the main body to the wheels and integrates the function of the suspension.

The explanation of their implementation is explained in chapter 4;

2.4.5 Simulation Controller

The `class SimController` (UML shown in Figure A.3), finally connects the individual components to the combined simulation procedure, which is schematically shown in fig. 2.4.
This procedure is done by the *main-thread* after the initialization of all components is finished. At the beginning of each loop it first checks if any new events about START, PAUSE and RESET are received. If so, the `SimController` changes its internal simulation status accordingly. If it still represents an active simulation state, then the simulation procedure of a timestep begins.

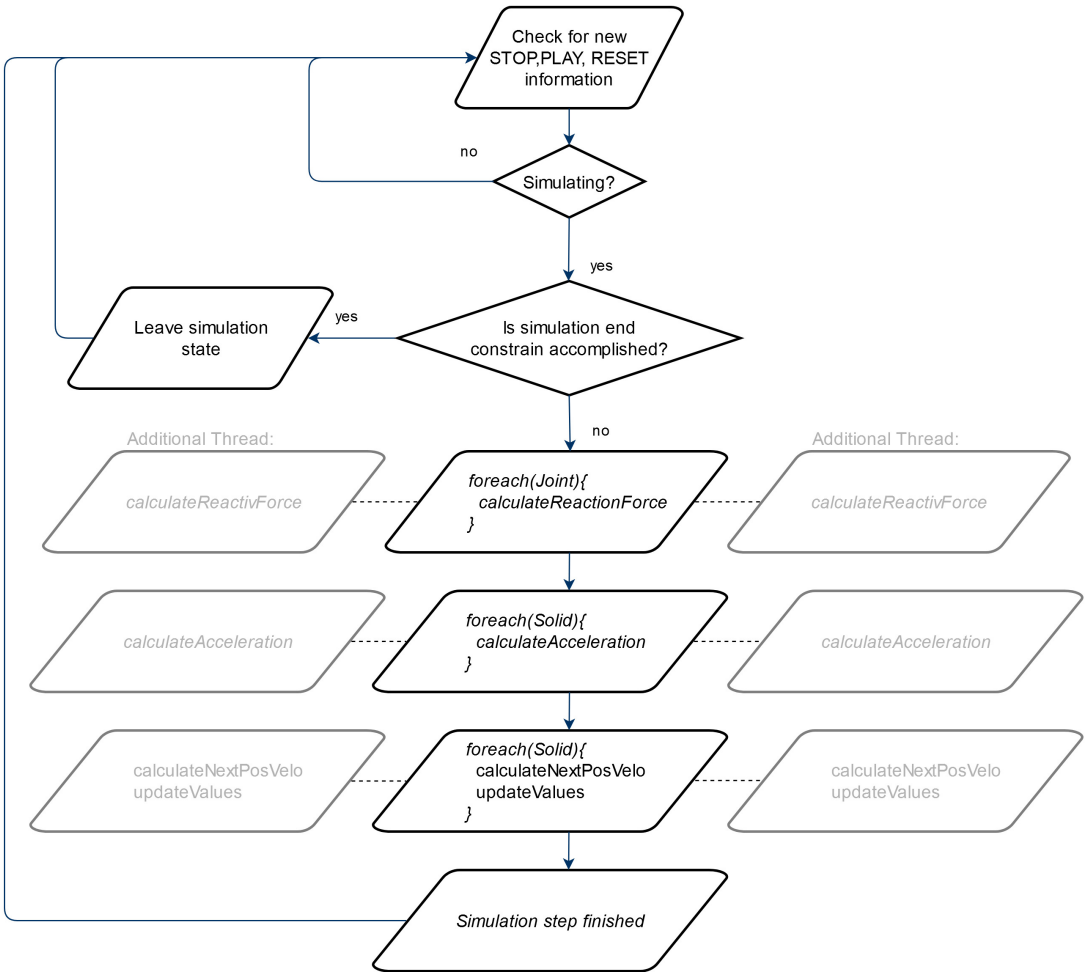


Figure 2.4: Overview: Schematic SimController

At the start, it checks if the simulation end constraint may already be fulfilled, which for an acceleration race according to the FS Germany rule-set is a total traveled straight distance of 75m. If this constraint is met, the simulation is finished and the simulation state is turned off. If not, then the calculation of the internal forces for every `Joint` is carried out, followed by the calculation of acceleration for each `Solid` and finally the calculation of their new position and velocity together with an information update to the `Data`. These three simulation steps can be distributed over multiple *threads* to increase performance.
At this point, the calculation of the timestep is finished and the loop is started again.

2.4.6 Verification: Rolling Wheel

This verification is used to check if the implementation is correct and to evaluate the performance of the Verlet integration in terms of accuracy. Therefore a scenario is used, which can easily be solved analytically. It contains a single rigid wheel rolling over a flat surface with a no-slip condition. It is accelerated by a motor torque M_{Motor} against its own inertia. It is considered how long it takes to travel 75m from standstill. The setup is displayed in fig. 2.5.
The Wheel attributes for this test are:

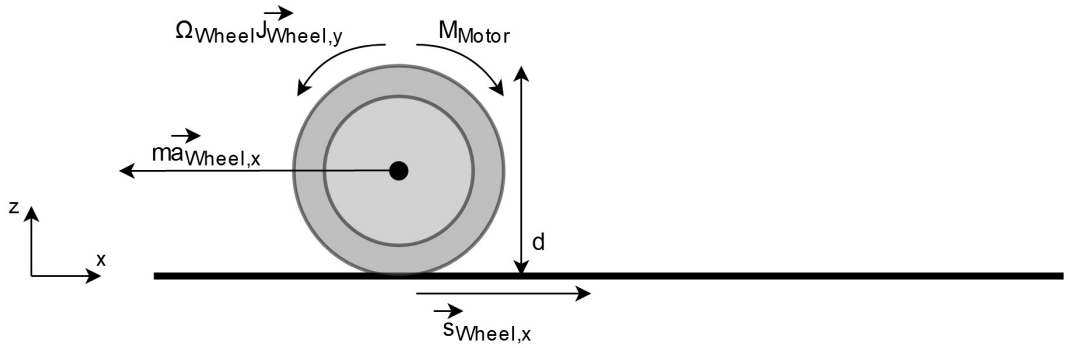


Figure 2.5: Overview: Rolling Wheel

- $d = 0.3m$
- $\vec{J}_{Wheel,x} = 5.0\frac{kg}{m^2}$
- $m = 2kg$

The equilibrium of forces around the support point in direction of the M_{Motor}:

$$0 = M_{Motor} - \frac{m \cdot \vec{a}_{Wheel,x} \cdot d}{2} - \vec{J}_{Wheel,y}\ddot{\Omega} \quad \text{with} \quad \ddot{\Omega} = \frac{2 \cdot \vec{a}_{Wheel,x}}{d} \tag{2.13}$$

For the analytical solution an equation for $\vec{v}_{Wheel,x}$ and t are derived from eq. (2.13).

$$\vec{v}_{Wheel,x} = \frac{M_{Motor}}{m \cdot \frac{d}{2} + 2\frac{\vec{J}_{Wheel,y}}{d}} t \tag{2.14}$$

$$t = \sqrt{\frac{\vec{r}_{Wheel,x}(m \cdot d + 4\frac{\vec{J}_{Wheel,y}}{d})}{M_{Motor}}} \tag{2.15}$$

To calculate Ω and $\dot{\Omega}$ the following additional equations are used:

$$\Omega = \frac{\vec{r}_{Wheel,x}}{\pi * d} \cdot 360° \quad \text{and} \quad \dot{\Omega} = \frac{\vec{v}_{Wheel,x}}{\pi * d} \cdot 360° \tag{2.16}$$

The results of the calculation and simulation and the errors between them are displayed in table 2.4. A total of four different setups 1-4 were tested. They differ from each other in their values for M_{Motor}. They were chosen so that several orders of magnitude of the total simulation time can be compared. Since the size of the timestep $\delta t = 0.01s$ is kept constant over the tests, different total simulation times, therefore, result in a different relative time resolution. The total investigated time span ranges from 1s to 100s, thus 100 to 10000 timesteps per simulation are represented. This is also reflected in the calculation time which increases proportionally with the number of timesteps, with the exception of *Attempt 1*. An explanation could be, that there is another performance bottleneck or offset for this low amount of timesteps. Or it might just be an irregularity in the underlying operating system. The test are all done with a CPU frequency of 4.0 Ghz.

$\Delta t = 0.01s$	Attempt 1	Attempt 2	Attempt 3	Attempt 4	Attempt A
M_{Motor} in Nm	5045.0	50.45	30.0	0.5045	5105.54
Total calculation time in ms	5	32	41	307	7
Analytical position in m °/s	75.0	75.0	75.0	75.0	75.0
Simulated position in m °/s	75.75	75.075	75.084	75.0075	75.141
Position error in %	1.0	0.1	0.112	0.01	0.188
Analytical time in s	1.0	10.0	12.97	100.0	0.994
Simulated time in s	1.0	10.0	12.97	100.0	0.99
Time error in %	0.0	0.0	0.0	0.0	-0.4
Analytical velocity in $\frac{m}{s}$	150.0	15.0	11.567	1.5	151.8
Simulated velocity in $\frac{m}{s}$	150.0	15.0	11.569	1.5	150.282
Velocity error in %	0.0	0.0	-0.017	0.0	-1.0
Analytical angle in °	28647.9	28647.9	28647.9	28647.9	28647.9
Simulated angle in °	28931.4	28676.5	28676.9	28650.8	28701.7
Angle error in %	0.99	0.1	0.1	0.01	0.188
Analytical ang. velocity in °/s	57295.8	5729.6	4418.3	573.0	57983.3
Simulated ang. velocity in °/s	57295.8	5729.6	4418.6	573.0	57403.5
ang. velocity error in %	0.0	0.0	0.0	0.0	-1.0

Table 2.4: Result of single wheel verification

In the results, all attempts show that they overshoot 75m and therefore have a relatively large error in position and angle. At the same time, velocity and ang. velocity error are ≈ 0.0. This contradicts the theoretical assumptions form section 2.2.1, where the velocity should produce a higher error compared to the positions. It is assumed, that this results from an unfortunate combination of analytical total simulation time and timestep length. Since most of the position error results from the time discretization, which in turn leads to an overshoot of the track length of $\vec{r}_{Rad,x} = 75m$, it seems that the position is the reason for the error and the speed is highly accurate. This conclusion is also faulty because the velocity is calculated out of the position.
For this reason, an additional *Attempt A* is conducted. The M_{Motor} was selected similar to *attempt 1* to archive a relatively large error. But it was selected, so that the position overshoots far less, which should better represent the error of the position and the distribution between position and velocity expected from section 2.2.1. This attempt fulfills the anticipations, with a final position of $\vec{r}_{Wheel,x} = 75.141m$ closer to the analytical result, the error of the velocities is suddenly far higher than the one of the positions.
In terms of the accuracy of simulated time, all simulations show a satisfying result. Whereby a similar problem as for the velocity might have occurred for *Attempt 1-4*. Nevertheless, *Attempt A* shows a satisfying accuracy when considering, that it uses only around 100 timesteps. *Attempt 4* shows, that for this kind of "model" more than 10 000 timesteps result in a neglectable error.

3 MFT Implementation

This chapter will cover a more in-depth explanation of the MFT model including its limits and boundaries. Afterward, the complete equations set for pure longitudinal slip, its code integration and the used parameter data are discussed. And ending on two verification attempts of the implementation.

3.1 Overview

MFT is a semi-empirical tire model, which has become an industry standard for tire models in vehicle handling simulation [4]. It was developed in cooperation by TU-Delft with Volvo and is published in H. B. Pacejka's Book "Tire and Vehicle Dynamics" [21]. Since then it was extended and modified to add functionality for example in "AN IMPROVED MAGIC FORMULA/SWIFT TYRE MODEL THAT CAN HANDLE INFLATION PRESSURE CHANGES" [4], which leads to multiple versions of the MFT. Here, the version 5.2 will be used.
Its underlying mathematical formula is based on a modified and parameterized sine function[21]:

$$Y = D\sin[C\arctan\{B(X+S_H) - U(B(X+S_H) - \arctan B(X+S_H))\}] + S_V \tag{3.1}$$

Here X is the input variable which for pure-longitudinal slip is represented by the Slip ratio κ. Y is the output variable representing the longitudinal force $F_{MFT,x}$, lateral force $F_{MFT,y}$ or the self aligning moment $M_{MFT,z}$. [21] As a function of the slip, this formula is able to generate characteristics that have a good match to measured results for longitudinal and lateral force. [21]
Additionally to a formula for $F_{MFT,x}$, $F_{MFT,y}$ and $M_{MFT,z}$, the Magic Formula model also provides equations for overturning moment $M_{MFT,x}$ and the rolling resistance $M_{MFT,y}$. A overview over all 139 inputs, parameters and outputs of a complete integrated Formula 6.1 is shown in fig. 3.1. Since this work focuses on longitudinal acceleration only, just the following parts of the MFT are integrated:

- Longitudinal force $F_{MFT,x}$
- Overturning moment $M_{MFT,x}$
- Rolling resistance $M_{MFT,y}$

The lateral force and the self aligning moment are not used because no lateral movement is allowed and therefore no lateral slip is present. Also, the self-aligning moment is neglected because no tire rotation around the Z-axis is allowed.

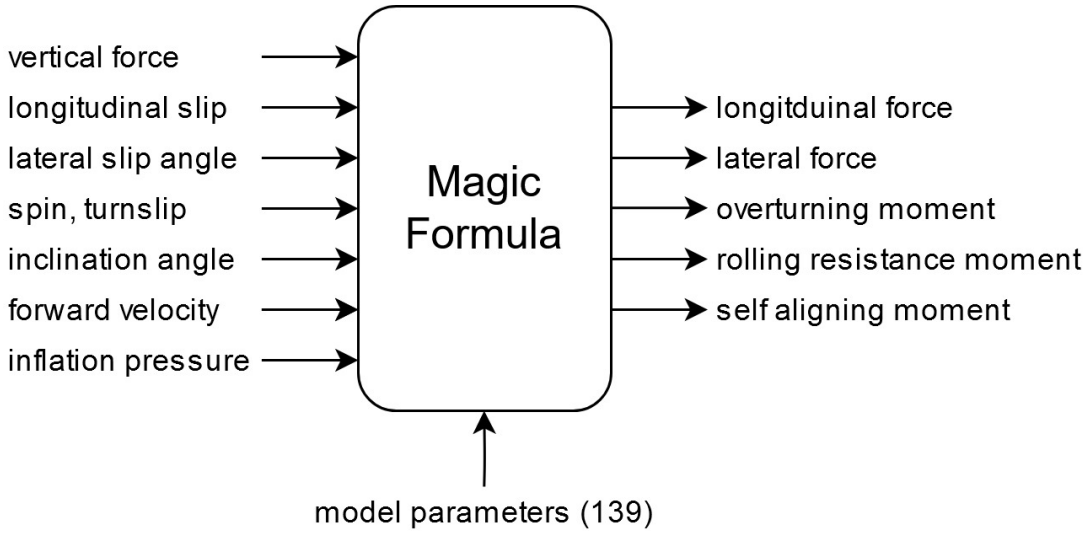


Figure 3.1: Inputs, parameters and outputs of the Magic Formula 6.1 [4]

To integrate MFT the TU Dresden FS team "Elbflorace" was provided information from Continental, containing all necessary parameters needed as well as some additional data about the correct usage of this data. It is based on their "C19" slick tire, which is specially developed for FS, mounted on a 7 x 13” rim. All used parameters are displayed in section B.2 and a fit to MFT version 5.2. They correspond to a tire pressure of 80kPa [8]

3.2 Limits and Boundaries

Like other models, MFT also has its limits and boundaries until which it will produces valid results.
In general, the current version of MFT (6.2) applies to large slip angles of more than 30°, longitudinal slip of +/- 100%, large load variations and high chamber angles. Additional effects like nonlinear tyre relaxation behavior depend on the timestep to produce correct results (here a timestep of $10^{-5}s$ is recommended). [20]

Nevertheless, the C19 data from Continental introduces far lower limits. They are displayed in table 3.1. Here especially the lon. slip range of only +25% might be a problem since racecars try to take full advantage of their tires when accelerating and, therefore, slip ratios higher then +25% can occur for a short amount of time.

Parameter	min.Value	max.Value
Normal load range	230N	1600N
lon. slip range	-25%	+25%
Slip angle range	-12°	+12°
Inclination angle range	-6°	+6°

Table 3.1: MFT model limitations for Continental C19 parameters [8]

3.3 Equations and Implementation

In this section the three MFT formulas for calculating $F_{MFT,x}$, $M_{MFT,x}$ and $M_{MFT,y}$ are implemented. For each there is a static function provided, including all the necessary equations to calculate the MFT property. These functions are stored in a C++ header file under `Simulation\solids\Moduls\MFTModul.h` and can be directly used by the `class WheelBase`. All important forces/moments of the MFT implementation are displayed in Figure 3.2. As one can see, the resulting forces and moments all take effect at the connection point between the tire and the surface.

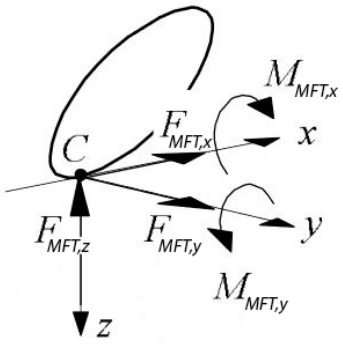


Figure 3.2: Forces/Moments of a wheel used for MFT [4]

```
static void MFT_FX_PureLongSlip (double Fz, double k, double IA,
double* Fx, double *ux)
```

The first function `MFT_FX_PureLongSlip` is used to calculate the force $F_{MFT,x}$ if only lon. slip is present, which in this simulation application is permanently the case. The following input parameter are use used:

- $F_{MFT,z}$ Vertical load between surface and tyre in N
- κ Slip rate
- γ_{Wheel} Inclination angle of the wheel in °

As Output it provides:

- $F_{MFT,x}$ Longitudinal force in N
- μ_x lon. friction coefficient

The calculation formula begins with a conversion of the inclination angle from degree to radiant and a check if $F_{MFT,z} >= 0$. Afterwards the MFT eqs. (3.2) to (3.13) are used [21].

$$F'_{MFT,z0} = F_{MFT,z0} \tag{3.2}$$

$$df_z = \frac{F_{MFT,z} - F'_{MFT,z0}}{F'_{MFT,z0} + 0.000001} \tag{3.3}$$

$$C = p_{Cx1} \cdot \lambda_{Cx} \tag{3.4}$$

$$\mu_x = (p_{Dx1} + p_{Dx2} df_z) \cdot \lambda_{\mu x} \cdot (1 - p_{Dx3} \cdot \gamma^2_{Wheel}) \tag{3.5}$$

$$D = \mu_x \cdot F_{MFT,z} \tag{3.6}$$

$$U = (p_{Ex1} + p_{Ex2} df_z + p_{Ex3} df_z^2) \cdot 1 - p_{Ex4} \operatorname{sgn}(\kappa) \cdot \lambda_{Ex} \tag{3.7}$$

$$K_{x\kappa} = F_{MFT,z} \cdot (p_{Kx1} + p_{Kx2} df_z) \cdot \exp(p_{Kx3} df_z) \cdot \lambda_{Kx\kappa} \tag{3.8}$$

$$B = \frac{K_{x\kappa}}{CD + 0.000001} \tag{3.9}$$

$$S_H = (p_{Hx1} + p_{Hx2} df_z) \cdot \lambda_{Hx} \tag{3.10}$$

$$S_V = F_{MFT,z} \cdot (p_{Vx1} + p_{Vx2} df_z) \cdot \lambda_{Vx} \tag{3.11}$$

$$\kappa_x = \kappa + S_H \tag{3.12}$$

$$F_{MFT,x} = D \sin[C \arctan\{B\kappa_x - U(B\kappa_x - \arctan(B\kappa_x))\}] + S_V \tag{3.13}$$

To avoid singularity, a small quantity was added to eq. (3.3) and eq. (3.9), as described by Pacejka [21]. An additional change to the original equation was made to eq. (3.5) as described in MFT version 6.1 [4] to include the effect of an inclination angle. Therefore it now resembles the lateral force of version 5.2.

Because the tire parameters are often done under laboratory conditions, scaling factors are included in the MFT equations. They allow the user to manipulate the tire characteristics afterward for a better fit to the vehicle testing environment. An example is the peak friction coefficient scaling factor $\lambda_{\mu x}$. It allows to correct the maximum of the coefficient to fit the real world (like temperature, surface material). Continental also provided some advice on how to scale the friction coefficient with temperature Figure A.7. [20] The Scaling Factor (SF) are described more detailed in "Tire and Vehicle Dynamics" by Pacejka [21].

List of Scaling factors used in `MFT_FX_PureLongSlip`:

- $\lambda_{Cx} = 1$ SF shape factor
- $\lambda_{\mu x} = 1$ SF peak friction coefficient
- $\lambda_{Ex} = 1$ SF curvature factor
- $\lambda_{Kx\kappa} = 1$ SF brake slip stiffness
- $\lambda_{Hx} = 0$ SF horizontal shift
- $\lambda_{Vx} = 0$ SF vertical shift

List of Parameters:

- $F_{z0} = 800N$ Nominal vertical load
- $R_0 = 0.235m$ Tire unloaded radius
- All other see appendix B.2

`static double MFT_MY_RollingResistance (double Fz, double Vr, double Fx)`
The second function `MFT_MY_RollingResistance` is used to calculate the moment $M_{MFT,y}$. The following input parameters are use used:

- $F_{MFT,z}$ Vertical load between surface and tyre in N
- $V_{MFT,r}$ forward seed of rolling m/s
- $F_{MFT,x}$ Longitudinal force in N

As Output it provides the torque of rolling resistance $M_{MFT,y}$ in Nm. Its calculation formula is as follows:

$$V_{MFT,0} = \sqrt{g \cdot r_0} \tag{3.14}$$

$$F_{MFT,z0} = \lambda_{Fz0} \cdot F_{MFT,z} \tag{3.15}$$

$$M_{MFT,y} = -F_{MFT,z} r_0 \cdot \left\{ q_{sy1} \ \arctan \frac{V_{MFT,r}}{V_{MFT,0}} + q_{sy2} \frac{F_{MFT,x}}{F_{MFT,z0}} \right\} \cdot \lambda_{My} \tag{3.16}$$

List of Parameters:

- $g = 9.81 \frac{m}{s^2}$ Nominal vertical load
- $r_0 = 0.235m$ Tire unloaded radius
- All other see appendix B.2

List of Scaling factors:

- $\lambda_{Fz0} = 1$ SF nominal load
- $\lambda_{My} = 1$ SF overtuning couple stiffness Y

`static double MFT_MX_OverturningCouple (double Fz, double IA, double Fy)`
The third function `MFT_MX_OverturningCouple` is used to calculate the moment $M_{MFT,x}$. The following input parameter are use used:

- $F_{MFT,z}$ Vertical load between surface and tyre in N
- γ_{Wheel} Inclination angle of the wheel in °

- $F_{MFT,x}$ lateral force in N

As Output it provides the overturning couple torque $M_{MFT,x}$ in Nm. Its calculation begins with a conversion of the inclination angle from degree to radiant, afterward the following calculation is executed:

$$\gamma^*_{Wheel} = \sin \gamma_{Wheel} \tag{3.17}$$

$$F'_{MFT,z0} = \lambda_{Fz0} \cdot M_{MFT,x} \tag{3.18}$$

$$M_{MFT,x} = M_{MFT,x} r_0 \cdot \left(q_{sx1} - q_{sx2} \gamma^*_{Wheel} + q_{sx3} \frac{F_{MFT,x}}{F'_{MFT,z0}} \right) \cdot \lambda_{Mx} \tag{3.19}$$

List of Parameters:

- $r_0 = 0.235m$ Tire unloaded radius
- All other see appendix B.2

List of Scaling factors:

- $\lambda_{Fz0} = 1$ SF nominal load
- $\lambda_{Mx} = 1$ SF overtuning couple stiffness X

3.4 Verification: Comparison to Continental Data

Because the implementation of this amount of equation into the software can easily result in errors, the first MFT verification is about the correct implementation. Therefore, the output of the MFT functions is compared to data provided by Continental [8]. The chart is shown in fig. 3.3. It plots the longitudinal force over the slip ratio at 80 kPa. Its data is not directly based on measured data from real-world testing, instead, the output of their Pacejka MFT version 5.2 is shown. Since Continental and the developed model are based on the same parameters and, in general, on the same equation set, it should be a well-suited comparison to check if the implementation is correct.
For this test, multiple points from fig. 3.3 are selected. Afterwards the developed MFT function for lon. force are used to recalculate these points, the results are compared in table 3.2.
All test points T1-T8 have an error of $\leq \pm 2\%$ and only T3, T4, T7 have an error $\geq \pm 1\%$. These small errors are probably derived from imprecise readout. Therefore these results in their scope do not display any error on the implementation side.

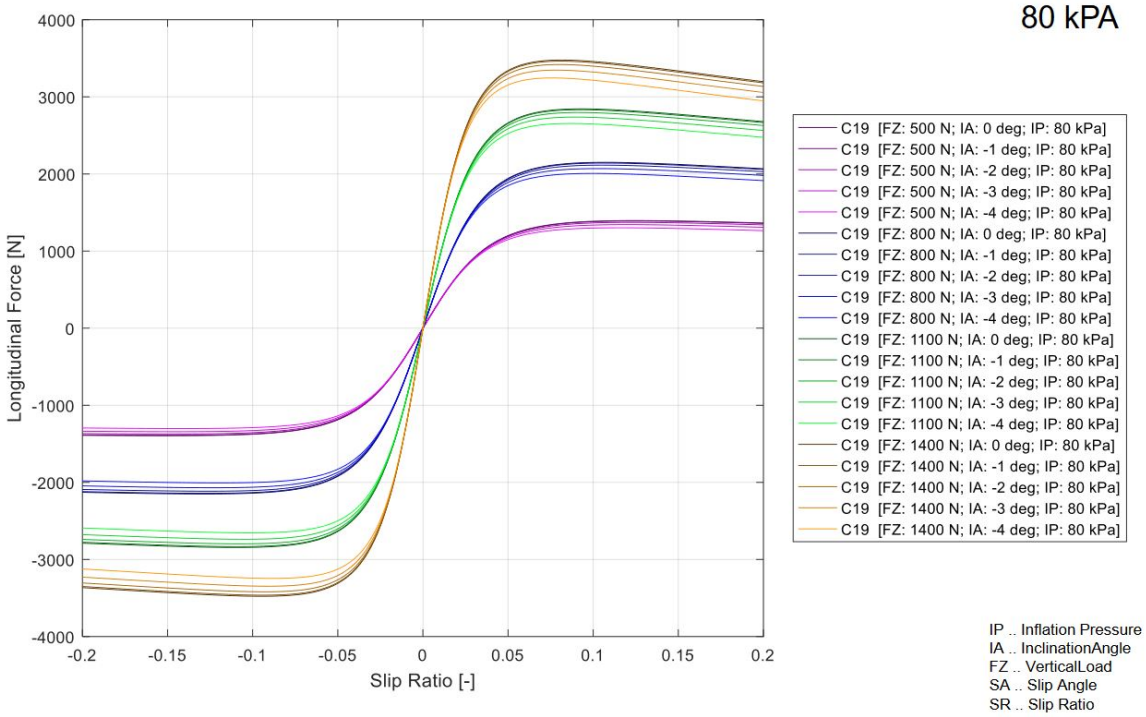


Figure 3.3: Plot of the longitudinal force over the slip ratio at 80 kPa by Continental[8]

	$F_{MFT,z}$ [N]	$F_{MFT,x}$ (Chart) [N]	$F_{MFT,x}$ [N]	Error [%]	κ	γ_{Wheel} [°]
T1	1400	3360	3384.73	0.71	0.05	0.0
T2	1400	-2520	-2498.78	-0.84	-0.025	0.0
T3	500	1400	1407.18	0.51	0.1	0.0
T4	500	-1400	-1424.43	1.745	-0.15	0.0
T5	800	2080	2067.93	-0.58	0.2	0.0
T6	800	1920	1914.15	-0.30	0.2	4.0
T7	800	-2160	-2128.88	-1.44	-0.2	0.0
T8	800	-1950	-1981.13	1.60	-0.2	4.0

Table 3.2: Verification: Results comparison to test data

3.5 Verification: Comparison to Test Data

The second verification is about comparing the results of the implemented model to measured data from tire tests (Formula Society of Automotive Engineer (FSAE) Tire Test Consortium (TTC) [17]). This data was provided by "Elbflorace".

These tests include, amongst other scenarios, sweeps over the slip ratio with constant normal loads and inclination angles. All the data was measured with a predecessor model (Continental C16 205/510R13) of the modeled (Continental C19 205/470R13) tire but similar characteristics are expected.

From these data, two sweeps were selected, one with an inclination angle $\approx 0°$ and another with $\approx 4°$. A diagram is available for both, which shows the longitudinal force over the slip ratio (fig. 3.4 and fig. 3.5. They contain the measured points, a smoothed approximation and the

output of the implemented MFT model. The displayed points are additionally shown in two tables for comparison (table 3.3 and table 3.4).
For the first sweep with an inclination angle of $\approx 0°$, the following additional parameters were taken from the measured data:

- Average $p = 83.0602kPa$
- $\gamma_{Wheel} = -0.0295°$
- minimum (min) $F_{MFT,z} = 965.84N$
- maximum (max) $F_{MFT,z} = 1144.6N$
- Average $F_{MFT,z} = 1053.2N$

The γ_{Wheel} and the average $F_{MFT,z}$ were used as input for the MFT model.

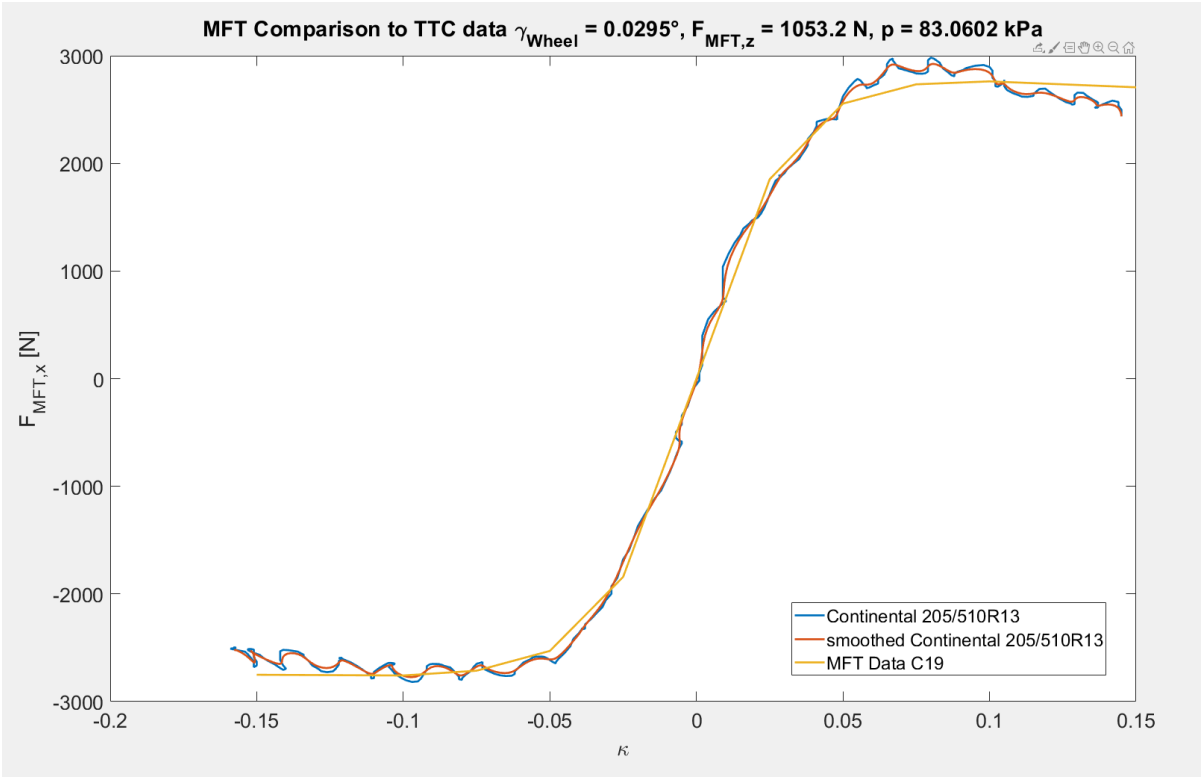


Figure 3.4: Verification: Chart for comparison of TTC data to MFT output with 0° inclination

The first sweep, in general seems to be a good fit (fig. 3.4 & table 3.3). But in detail the asymmetry with the increase at $\kappa \approx 0.075$ is not followed exactly by the model graph. Additionally, the decrease of the calculated longitudinal force above $\kappa > |0.1|$ is far less steep than the one in the measured data. The second inequality of the two graphs can probably be traced back to the different tires' types. Especially, since the models output fits perfectly to data from Continental from the previous attempt. In general, this result further increases the confidence in the implemented model.

$F_{MFT,x}$ TTC [N]	$F_{MFT,x}$ Model [N]	κ
/	3384.73	0.15
2804	2761.91	0.1
2882	2734.56	0.075
2570	2555.45	0.05
1690	1852.92	0.025
45.28	0	0.2
-1674	-1842.04	-0.025
-2590	-2529.33	-0.05
-2699	-2713.01	-0.075
-2732	-2757.5	-0.1
-2569	-2750.85	-0.15

Table 3.3: Verification: Results comparison of TTC data to MFT output with 0° inclination

For the second sweep with an inclination angle of $\approx 4°$ new parameters are taken from the measured data:

- Average $p = 82.9189kPa$
- $\gamma_{Wheel} = 3.9547°$
- min $F_{MFT,z} = 1416.1N$
- max $F_{MFT,z} = 1632.5N$
- Average $F_{MFT,z} = 1529.9N$

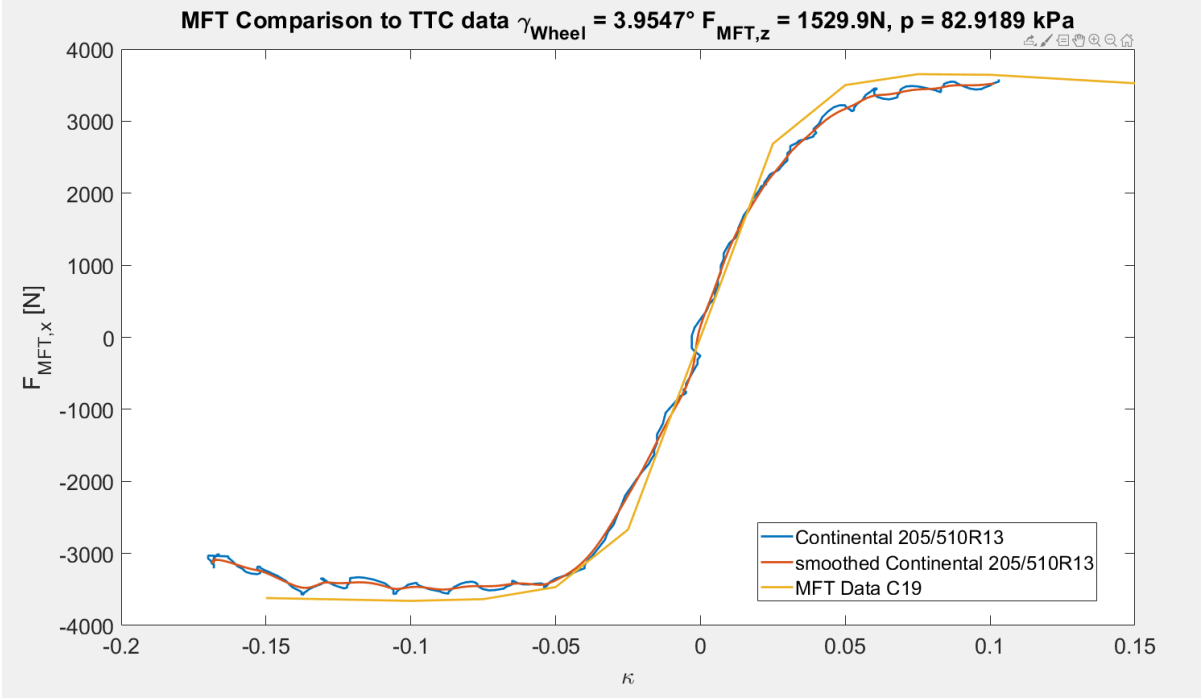


Figure 3.5: Verification: Chart for comparison of TTC data to MFT output with 4° inclination

$F_{MFT,x}$ TTC [N]	$F_{MFT,x}$ Model [N]	κ
/	3526.72	0.15
3522	3644.46	0.1
3441	3652.79	0.075
3178	3501.41	0.05
2278	2687.32	0.025
259.5	0.0	0.2
-2194	-2668.67	-0.025
-3372	-3466.82	-0.05
-3455	-3634.74	-0.075
-3263	-3619.385	-0.15

Table 3.4: Verification: Results comparison of TTC data to MFT output with 4° inclination

The second sweep shown in fig. 3.5 with its data in table 3.4, shows a far worse fit compared to the first one. The model results in a generally more longitudinal force than what was measured. This again may be a result in further development of the tire by Continental, so the C19 can transfer higher forces with greater inclination angle than the older C16. Compared to the first sweep, the $F_{MFT,x}$ of the second are higher because of its 50% higher tire load. Even if the predictions of the model and the measured data do not match perfectly, the final result of this verification seems plausible, as the error between the two could be due to the tire development. It is therefore not a proof of the accuracy of the model, but a way to gain confidence in the plausibility of the model, and the results have supported the model.

4 Subsystem Models

In this chapter, the full vehicle simulation is finally assembled. Therefore remaining modules Wheel, Mainbody, motor controller and aerodynamics are described in detail and implemented. Also, multiple verification attempts are contained to support the model integration.

4.1 Solid: Wheel

4.1.1 Overview

This section describes the *Solid* Wheel. It contains the model for the tire (uses the MFT modul), the rim, the wheel carrier, the motor in terms of its inertia influence and parts of the suspension. Here, the suspension is referred to as the connection between the mainbody and one Wheel. Therefore the full vehicle contains four suspensions. The Wheel model is described by a point particle with the system of motion equations based on the laws of kinetics. Therefore, an equation is provided for all its three directions of movement and its three rotations. Since the Wheel consists of multiple physical object, its inertia properties have to be defined first:

$$m_{Wheel} = m_{Tyre} + m_{Rim} + m_{Assembly} + \frac{1}{2} m_{Suspension} \tag{4.1}$$

$$\vec{J}_{Wheel} = \vec{J}_{Tyre} + \vec{J}_{Rim} + m_{Assembly}\vec{l}^{2}_{Assembly} + \frac{1}{2} m_{Suspension}(\vec{s}_{Wheel} - \vec{s}_{Suspension})^2 \tag{4.2}$$

$$J_{rot} = \vec{J}_{Tyre,y} + \vec{J}_{Rim,y} + J_{Rotor/Transmi} \tag{4.3}$$

The Equation (4.1) combines all the masses of the contained sub-bodies. Thereby $m_{Assembly}$ describes the sum of the motor, the wheel carrier and the transmission. Only half of the suspension's mass counts toward the Wheel, the other half is contained in the Mainbody. In Equation (4.2) the Moment of Inertia of the Wheel is calculated according to the parallel axis theorem. The variable $\vec{s}_{Suspension}$ describes the distance between the Center of Gravity (COG) of the Mainbody and the COG of the suspension. Meanwhile, $\vec{l}_{Assembly}$ refers to the distance of the COG of the Wheel to the combined one of the assembly. It should be considered, that $\vec{J}_{Wheel,y}$ is not used for the motion equation. The reason is that, as described in section 2.4.3, the *Solid* Wheel contains rotating bodies like:

- Tyre,
- Rim,
- and Rotor of the Motor.

Additionally, it contains non-rotating parts like:

- Motor
- Wheel carrier
- Parts of the suspension

Therefore, adjustments had to be made to the model, which led to the decision to fix the non-rotating parts in their rotation around the Y-axis. Otherwise, the *Solid* Wheel would contain two different rotation states for its sub-parts. This would introduce a second degree of freedom into the equilibrium of moments around the Y-axis. And that, in turn, would introduce a lot of complexity and would require additional simulation steps. For this reason, an additional MOI (J_{rot}) property is added. It is described by eq. (4.3) and does not include the inertia of non-rotating parts.

As a further simplification, the small-angle approximation was introduced. Since the γ_{Wheel} is expected to be $\leq 10°$, it does not influence any levers for the torque calculation and does not introduce any new ones. To accommodate for purely longitudinal movement $\ddot{\vec{r}}_y = 0; \dot{\vec{r}}_y = 0$ and $\ddot{\Psi}_{Wheel} = 0; \dot{\Psi}_{Wheel} = 0$ for the *Solid* Wheel is set. An additional functionality that the Wheel must provide is the spring and damping effect of the tire. Therefore the following equation relationship is used:

$$F_{MFT,z} = c_{Tyre}(r_0 - r_e) - d_{Tyre}\dot{\vec{r}}_{Wheel,z} \tag{4.4}$$

For the two parameters, the following values are derived form the Continental C19 data appendix B.2, $c_{Tyre} = 98820\frac{N}{m}$ and $d_{Tyre} = 3429\frac{Ns}{m}$. The equation is used to calculate the resulting force from the tire to the surface $F_{MFT,z}$. The deformation of the tires influences the length of the moment levers.
For later connection of the Wheel to the Mainbody, the force vector $\vec{F}_{Suspension}$ and torque vector $\vec{M}_{Suspension}$ are used. They are described in detail in section 4.5.

4.1.2 Equations

In this section, the motion equations of the wheel are described. For better visualization fig. 4.1, fig. 4.2 and fig. 4.3 give an overview over the system.
Equilibrium of forces along x-axis:

$$0 = F_{MFT,x}(F_{MFT,z}, \kappa, \gamma_{Wheel}) - \vec{F}_{Suspension,x} - \frac{M_{MFT,y}(F_{MFT,z}, \dot{\vec{r}}_{Wheel,x}, F_{MFT,x})}{r_e} - \ddot{\vec{r}}_{Wheel,x} m_{Wheel} \tag{4.5}$$

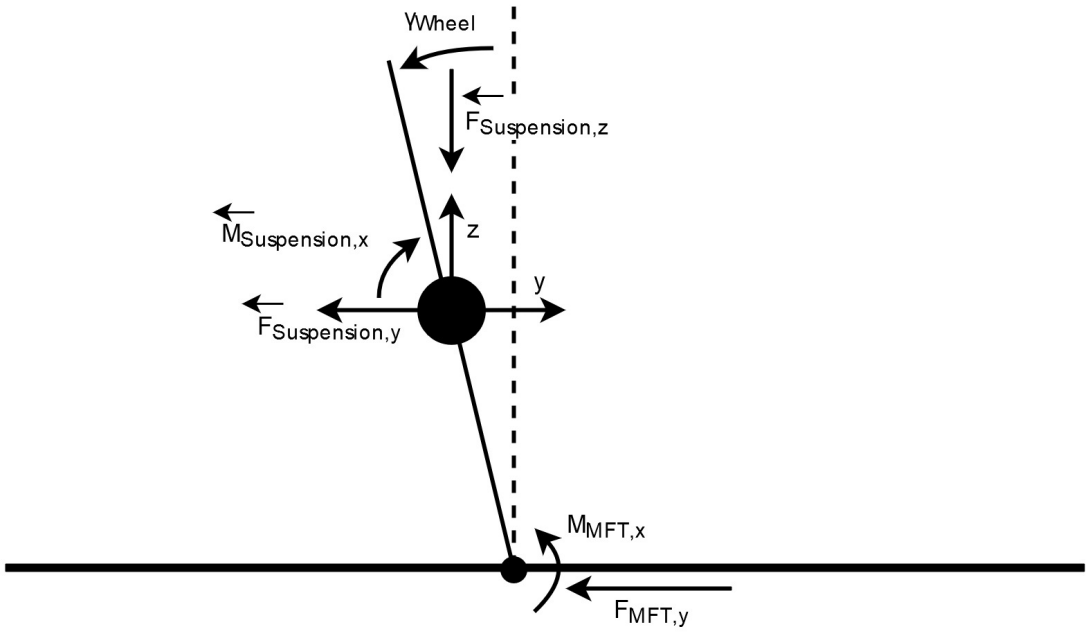


Figure 4.1: Overview forces and moments of the Solid Wheel X-plane

Equilibrium of forces along y-axis:

$$0 = -F_{MFT,y} - \vec{F}_{Suspension,y} - \frac{M_{MFT,x}(F_{MFT,z}, \gamma_{Wheel}, F_{MFT,y})}{r_e} \tag{4.6}$$

Equilibrium of forces along z-axis:

$$0 = F_{MFT,z} - \vec{F}_{Suspension,z} - (g + \ddot{\vec{r}}_{Wheel,z}) m_{Wheel} \tag{4.7}$$

Equilibrium of moments around the x-axis:

$$0 = -F_{MFT,y} r_e - \vec{M}_{Suspension,x} - \vec{J}_{Wheel,x} \ddot{\gamma_{Wheel}} \tag{4.8}$$

Equilibrium of moments around the y-axis:

$$0 = M_{Motor} - F_{MFT,x}(F_{MFT,z}, \kappa, \gamma_{Wheel}) r_e - J_{rot} \ddot{\Omega_{Wheel}} \tag{4.9}$$

Equation (4.9) is used for the rotating part of the wheel. Therefore J_{rot} has replaced $\vec{J}_{Wheel,y}$. Because between the rotating parts of the wheel and the Mainbody are its fixed parts, the Ω_{Wheel} of the wheel is independent from $\vec{M}_{Suspension,x}$. To still transmit the recoil of the motor to the Mainbody, the following equation is used:

$$\vec{M}_{Suspension,y} = M_{Motor} \tag{4.10}$$

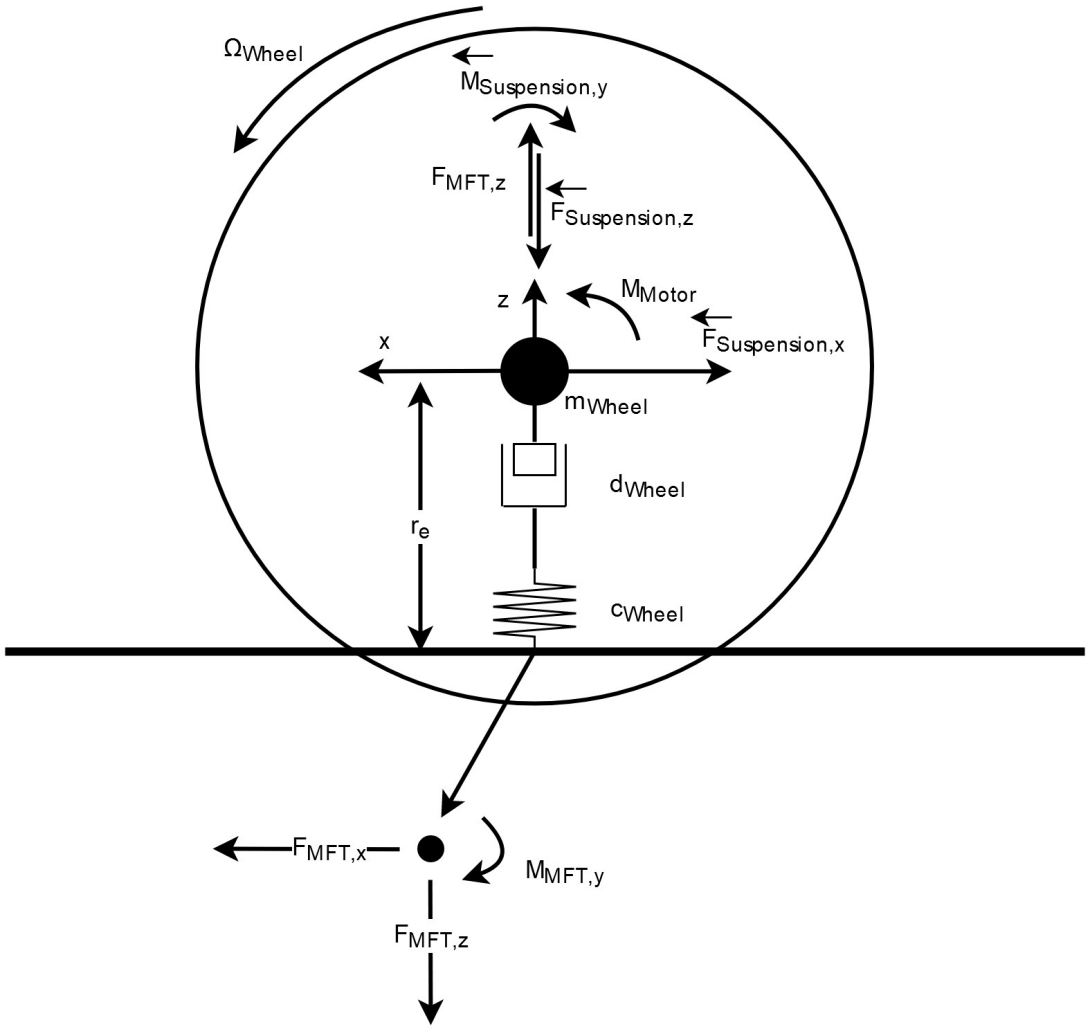


Figure 4.2: Overview forces and moments of the Solid Wheel Y-plane

It is independent from eq. (4.9). As a downside, this solution introduces some additional energy into the system. But since the movement of the Mainbody is quite limited this effect should be minor.
Equilibrium of moments around the z-axis:

$$0 = \vec{M}_{Suspension,z} \tag{4.11}$$

The Wheel monitors the basic values of its base class *Solid* as well as $F_{MFT,x}$, $F_{MFT,z}$, $M_{MFT,x}$, $M_{MFT,y}$, κ and μ_x .

4.1.3 Comparison: Solid Wheel with analytic calculation

To get an overview of the results of the *Solid* Wheel, a similar approach as in section 2.4.6 is taken, where a single wheel is accelerated against its own inertia by a M_{Motor} on a distance of 75m. For comparison, an analytic result is also calculated for each scenario. There, the same

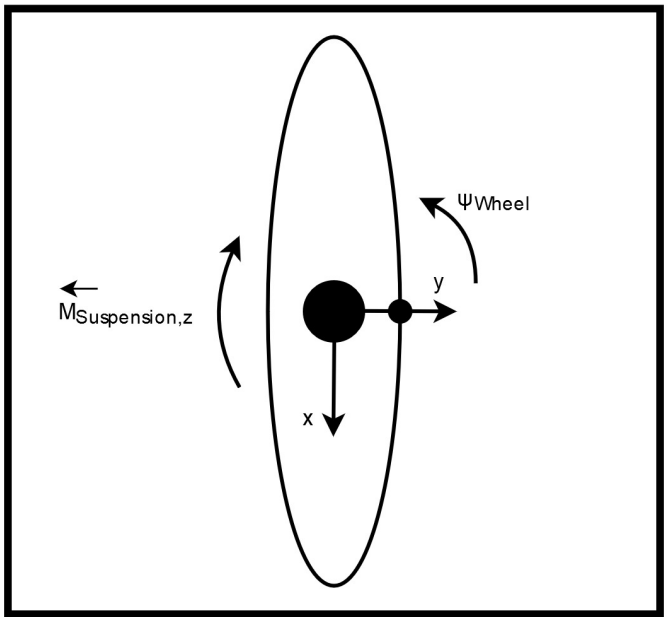


Figure 4.3: Overview forces and moments of the Solid Wheel Z-plane

wheel was accelerated, but with a no slip-condition, therefore, MFT does not influence the analytic approach. For calculation, the same equations as in section 2.4.6 were used.
The parameter of the Wheel are:

- $m_{Wheel} = 70kg$
- $J_{rot} = 5kg \cdot m^2$
- $\vec{J}_{Wheel} = \vec{0}$
- $\gamma_{Wheel} = 0°$
- $F_{MFT,z} = 10\frac{m}{s^2} \cdot m_{Wheel}$
- $\vec{F}_{Suspension} = \vec{0}$
- $\vec{M}_{Suspension} = \vec{0}$
- $r_0 = 0.235m$

The Wheel mass is chosen to be far higher than realistic to have a better intuition for the results. It should emulate a part of the total vehicle's weight.
The comparison between the analytic and the simulated result is limited because of different physical properties. The following factors can be reasons for deviation of the analytic solution:

- It does not contain rolling resistance
- The analytic solution contains the no-slip condition, therefore relatively less energy is put into the rotation motion compared to the simulation model.

At $\vec{r}_{Wheel,x} = 75m$	analytical	$\Delta t = 5 \cdot 10^{-5}s$	$\Delta t = 10^{-4}s$	$\Delta t = 0.001s$
Time [s]	2.38	3.21	3.21	3.25
Speed [$\frac{m}{s}$]	63.06	47.30	47.29	46.77
ang. speed [°/s]	15373.8	26695.7	26711.1	27355.3
Angle [°]	18285.9	42833.3	42875.5	44616.7
Calculation time [ms]	/	4916	2584	229
$M_{MFT,y}$ [Nm]	/	34.17	34.16	33.64
μ_x	/	2.727	2.727	2.727
Maximum κ	0.0	11.7	23.0	234
Average κ	0.0	1.246	1.247	1.327
$F_{MFT,x}$ [N]	/	≈ 1200	≈ 1200	≈ 1200

Table 4.1: Comparison results: Solid Wheel with analytic calculation with motor torque 1000 Nm

At $\vec{r}_{Wheel,x} = 75m$	analytical	$\Delta t = 5 \cdot 10^{-5}s$	$\Delta t = 10^{-4}s$	$\Delta t = 0.001s$
Time [s]	16.82	19.94	19.94	19.94
Speed [$\frac{m}{s}$]	8.92	7.05	7.05	7.05
ang. speed [°/s]	2174.1	1775.1	1775.1	1775.1
Angle [°]	18285.9	18845.1	18845.1	18847.6
Calculation time [ms]	/	45038	18010	1571
Maximum $M_{MFT,y}$ [Nm]	/	8.15	8.15	8.15
μ_x	/	2.727	2.727	2.727
Maximum κ	0.0	135.0	40.6	102.3
Average κ	0.0	0.0011	0.0011	0.0011
$F_{MFT,x}$ [N]	/	57.6	57.6	57.7

Table 4.2: Comparison results: Solid Wheel with analytic calculation with motor torque 20 Nm

Under these conditions, it is estimated, that the analytic calculation should result in lower track times and higher final velocities. In a shorter simulation, and with a relatively greater driving force, the lower times are mainly attributable to less energy put into the rotation compared to the simulation model. Whereby in results with lower motor torque the advantage should mainly come through the absence of rolling resistance. Additionally, for this kind of scenario, the slip κ of the simulated model should be ≈ 0 and so the total driven angle should be quite similar. To display both of these scenarios, the two considered torques are chosen to be quite extreme, the first with M_{Motor} = 1000 Nm and the second with M_{Motor} = 20. Their results are displayed in table 4.1 and table 4.2. For insight into the behavior of different time resolution, each of the scenarios uses three different timestep sizes $\Delta t = 5 \cdot 10^{-5}s$, $10^{-4}s$ and 0.001s.

Conclusion: The results of the two scenarios show exactly the estimated behavior for high and low driving force. In terms of accuracy compared to time resolution, only the lowest resolution of $\Delta t = 0.001s$ with the higher torque setup, shows a small error compared to the converging results of the two higher resolutions. Therefore it seems as the $\Delta t = 10^{-4}s$ would be sufficient enough to provide for MFT simulation. But a problem occurred while simulating, which is not displayed in the results. It was observed, that especially in the start of simulations the slip ratio

constantly switches from > 1 to < -1 and back to > 1 again with every timestep. In further testing, this effect does especially occur, when the $\vec{J}_{Wheel}$ is relatively low. An explanation for this effect is that at low velocity the motor torque accelerates the wheels rotation $\dot{\Omega}_{Wheel}$ in one timestep so much, that it results in a very high κ often > 1, which in turn creates a high longitudinal force $F_{MFT,x}$, that slows the wheel down. This could also result in negative velocities in the same scenario. In the next simulation step, the torque accelerates the rotation of the wheel, again creating a high slip ratio for the next step. In higher velocities, this does not occur because the relatively small change of velocity in one step compared to the driven velocity does not allow for such drastic influences in the slip ratio. A reason for this effect is the steep increase of longitudinal force with κ of the tires, comparable to a strong spring connections between rotation speed and forward speed of the tire. This effect can be reduced by decreasing the timestep. In this comparison the visibility of the effect increased with reduced time resolution and in the low torque set-up as, here, the vehicle was longer in the critical low velocity area. Even though neither it does disappear with the lowest timestep, its duration is reduced to $< 100ms$. Additionally, while the data does not "look" correctly because the energy is still conserved it has no major effect on the track time. Actually, this effect can never fully disappear even with an infinitely low timestep. Because in and after the very first simulation step a singularity is present. Here the lon. velocity $\vec{s}_{Wheel} = 0$, while the rotation speed is not. Accordingly, the slip ratio becomes infinite. To address this problem, measures were taken. First κ was capped in its maximum and minimum ($-1 \leq \kappa \leq 1$. Additionally if $\kappa > 0.2$ being the maximum for correct usage of the MFT implementation, it is divided by 100 for the calculation of $F_{MFT,x}$. Even though this additionally changes the physical correctness and introduces an error at high slip ratios, it should be guaranteed, that going out of boundaries should **never** result in a superior track time. This could otherwise end up in confusing results for motor controller optimization and sensitivity analysis. In addition, a well implemented motor control system should maintain the slip ratio in such a way that it maximizes the longitudinal force and thus lies within its limits. To resolve the singularity error, a small speed entity was added for the slip calculation:

$$\kappa = \frac{\dot{\Omega}_{Wheel} \cdot r_e - \dot{\vec{s}}_{Wheel,x}}{|\dot{\vec{s}}_{Wheel,x}| + 0.0001 m/s} \tag{4.12}$$

One last thing the comparison results show is that the real-time calculation aspect highly depends on the time resolution. While $\Delta t = 0.001s$ provides it, $\Delta t = 5 \cdot 10^{-5} s$ does not. The simulation were done with a 3.5 Ghz single-core speed.

4.2 Solid: Mainbody

4.2.1 Overview

The model for the Mainbody is, like the wheel, also defined by a point particle with a system of motion equations based on the laws of kinetics. It contains three directions of motion and three directions of rotation as well. Similarly, too, its inertia properties are connected the mass of the driver and a half of every suspension (the other half is contained in the according wheel). These combined inertia properties are calculated as follows:

$$m_{MainCombined} = m_{Mainbody} + m_{Driver} + \sum_n \frac{1}{2} m_{Suspension,n} \tag{4.13}$$

$$\vec{J}_{MainCombined} = \vec{J}_{Mainbody} + m_{Driver}\vec{s}^2_{Driver} + \sum_n \frac{1}{2} m_{Suspension,n}\vec{s}^2_{Suspension,n} \tag{4.14}$$

$$\vec{s}_{MainCombined} = \frac{m_{Driver}}{m_{MainCombined}}\vec{s}_{Driver} + \sum_n \frac{\frac{1}{2}m_{Suspension,n}}{m_{MainCombined}}\vec{s}_{Suspension,n} \tag{4.15}$$

Equation (4.15) marks an important decision which was made for the Solid Mainbody. Contrary to the Wheel the point for the equilibrium of moments was not chosen around the combined COG but instead around the one of the Mainbody. The reason behind this decision was to have a relatively fixed reference point for all the relations throughout the vehicle. Where the combined might change its position from race to (especially due to different driver weight and positions) the position of the Mainbody's COG does not change that much. Therefore no conversion of all the different position vectors $\vec{s}$ referred to the Mainbody is needed, when drivers or suspensions weight/positions changes. The Mainbody contains the vehicle's chassis and all inner parts, therefore the unsprung mass (this includes aerodynamic devices). The variable n is used to refer to an object like a suspension or Wheel and its corresponding forces at a defined position. It starts with n = 1 referring to the front left position and increases in the direction of $\Psi_{Mainbody}$ around the vehicle. Therefore n = 4 is corresponding to the front right. Two major simplifications are introduced into the Mainbody model. The first one is its non-deformability and the second one is again using the small-angle approximation for all levers for each direction of rotation.

4.2.2 Equations

Equilibrium of forces along x-axis:

$$0 = \sum_n \vec{F}_{Suspension,x,n} - \vec{F}_{Aero,x} - m_{MainCombined}\ddot{\vec{r}}_{Mainbody,x} \tag{4.16}$$

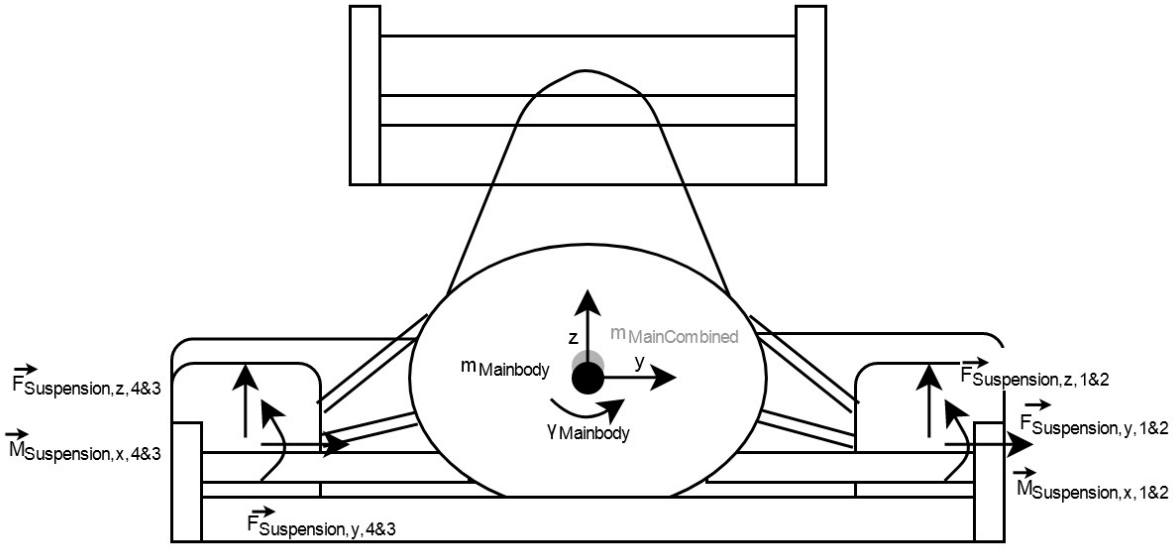


Figure 4.4: Overview forces and moments of the Solid Mainbody X-plane

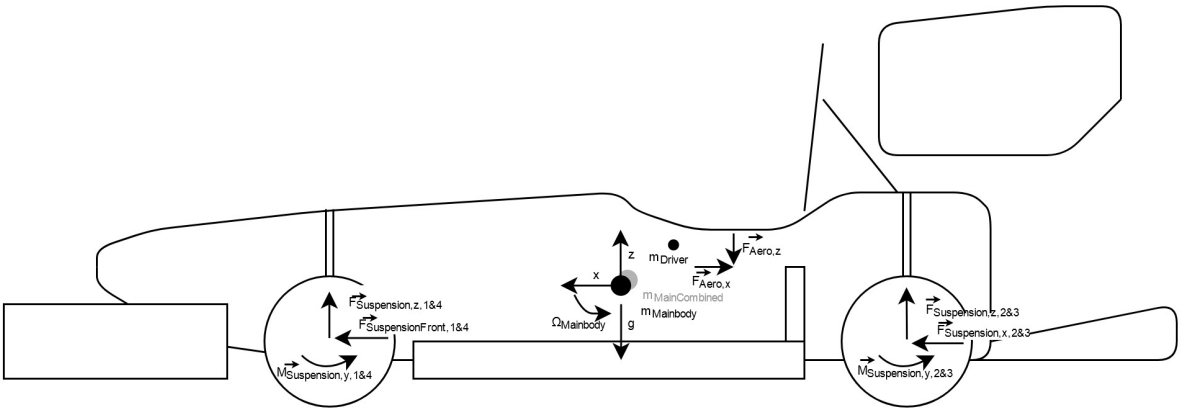


Figure 4.5: Overview forces and moments of the Solid Mainbody Y-plane

Equilibrium of forces along y-axis:

$$0 = \sum_n \vec{F}_{Suspension,y,n} - m_{MainCombined}\ddot{\vec{r}}_{Mainbody,y} \tag{4.17}$$

Equilibrium of forces along z-axis:

$$0 = \sum_n \vec{F}_{Suspension,z,n} - \vec{F}_{Aero,z} - m_{MainCombined}g - m_{MainCombined}\ddot{\vec{r}}_{Mainbody,z} \tag{4.18}$$

Equilibrium of moments around the x-axis:

$$\begin{aligned} 0 = &\sum_n \vec{M}_{Suspension,x,n} + \sum_n \vec{F}_{Suspension,z,n}\vec{s}_{Wheel,y,n} \\ &- \sum_n \vec{F}_{Suspension,y,n}\vec{s}_{Wheel,z,n} - \gamma_{M}\ddot{}_{inbody}\vec{J}_{MainCombined,x} \end{aligned} \tag{4.19}$$

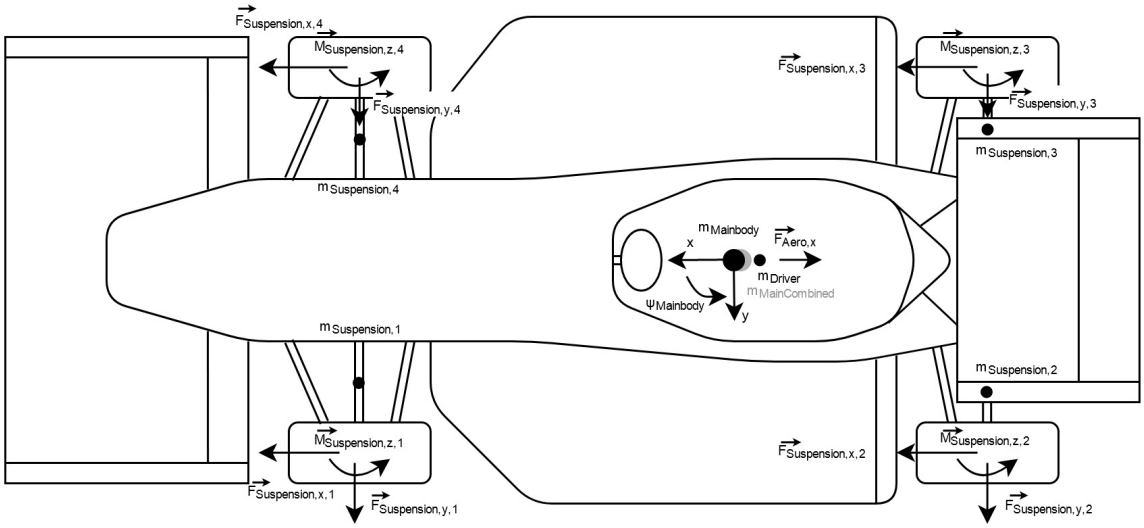


Figure 4.6: Overview forces and moments of the Solid Mainbody Z-plane

Equilibrium of moments around the y-axis:

$$\begin{aligned} 0 = &\sum_n \vec{M}_{Suspension,y,n} - \sum_n \vec{F}_{Suspension,x,n}\vec{r}_{Mainbody,z} - \sum_n \vec{F}_{Suspension,z,n}\vec{s}_{Wheel,x,n} \\ &+\vec{F}_{Aero,z}\vec{s}_{Aero,x} + m_{MainCombined}g\vec{s}_{m_{MainCombined},x} - \ddot{\Omega}_{Mainbody}\vec{J}_{MainCombined,y} \end{aligned} \tag{4.20}$$

Equilibrium of moments around the z-axis:

$$\begin{aligned} 0 = &\sum_n \vec{M}_{Suspension,z,n} + \sum_n \vec{F}_{Suspension,y,n}\vec{s}_{Wheel,x,n} \\ &- \sum_n \vec{F}_{Suspension,x,n}\vec{s}_{Wheel,y,n} - \ddot{\Psi}_{Mainbody}\vec{J}_{MainCombined,z} \end{aligned} \tag{4.21}$$

4.3 Joint: Motor & Controller

4.3.1 Overview

As previously mentioned, the motor and its controller are combined and modeled as a `Joint`. Thereby, there is no major emphasis put on a detailed modelling of the engines, but it should be sufficient enough for a functional model of an electric motor. They are characterized by instant response time and have maximum power and torque output. Each is directly connected to a `Solid Wheel Joint` and can be enabled and disabled.

The motor controller is used to control the torque/power output. This is essential even for acceleration races since the maximum lon. force output needs to be achieved. And since tires have a maximum at a certain slip ratios, it is the motor controller's duty to control the torque of the motor, so that the slip ratio stays around the position of maximal lon. force. An additional

reason why it is highly important for the simulation, because if $\kappa \geq 0.2$, the tyre model exceeds its boundaries. The controller was implemented using a common Proportional–Integral–Derivative (PID) controller, which is easy to use and can archive good results if well-tuned. As input e the error of the slip rate κ is used to a set target slip rate κ_{eff} and its output is corresponds to the motor torque:

$$e(t) = \kappa(t) - \kappa_{eff} \tag{4.22}$$

4.3.2 PID Motor Controller

A PID controller is based on the following base equation:

$$u(t) = P_{out}(t) + I_{out}(t) + D_{out}(t) \tag{4.23}$$

Here P_{out} corresponds to the proportional term:

$$P_{out}(t) = K_p e(t) \tag{4.24}$$

The integral term is defined as followe:

$$I_{out}(t) = K_i \int_0^t e(t)dt \tag{4.25}$$

And D_{out} describes the derivative term:

$$D_{out}(t) = K_d \frac{de(t)}{dt} \tag{4.26}$$

Since the PID controller is used in a numerical environment, for the integral and the derivative term time discretization is used. This results in the following two equations:

$$I_{out}(t) = K_i \sum_n e(n\Delta t) \tag{4.27}$$

$$D_{out}(t) = K_d \frac{e(t) - e(t - \Delta t)}{\Delta t} \tag{4.28}$$

Here n is the total amount of timesteps. A PID controller is based on there factors K_p , K_i and K_d . These factors must be chosen correctly for the controller to work well in a given scenario. For this, multiple methods are available, like theoretic considerations or even algorithms that automatically adjust them. In this thesis, these parameters are determined empirically since results can be obtained quickly and not too much additional complexity should be introduced.

The code implementation looks as follows:

```
e.newValue(vals.MotorC_keff.val() - wheel.k.val() , timestamp + timestep,
    &lock);
e_i += e.val();
if(e_i > vals.motorC_eimax.val()) e_i = vals.motorC_eimax.val();
if(e_i < vals.motorC_eimin.val()) e_i = vals.motorC_eimin.val();
double torque = e.val() * vals.motorC_Kp.val()
                + e_i * vals.motorC_Ki.val()
                + (e.val() - e.getLastValue()) / (timestep) * vals.
                   motorC_Kd.val();
bool reachedLimit = false;

if(torque > torqueMotorMax.val()){
    torque = torqueMotorMax.val();
    reachedLimit = true;
}else if(torque < 0){
    torque = 0;
    reachedLimit = true;
}
double power = torque * wheel.angularVeloY.val() * M_PI / 180.0;

if(power > powerMotorMax.val()) {
    power = powerMotorMax.val();
    torque = power / (wheel.angularVeloY.val() * M_PI / 180.0);
    reachedLimit = true;
}
if(reachedLimit) e_i -= e.val();
```

The integral term has an additional max $e_{i,max}$ and min $e_{i,min}$ limit, to provide more stability to the controller. The torque and power limit of the motor are shown in the code as well. Since the model is not fully integrated yet, the empirical search for the PID parameters will take place in the next verification part section 4.5.4.

4.4 Joint: Aerodynamic

4.4.1 Overview & Equations

The model of the aerodynamic effects is kept simple. It only consists of two equations for the base forces drag $\vec{F}_{Aero,x}$ and downforce $\vec{F}_{Aero,x}$(the inverse of lift):

$$\vec{F}_{Aero,x} = \frac{\rho_{Air}}{2} \cdot A \cdot \dot{\vec{r}}^2_{Mainbody,x} \cdot (C_d + C_d\alpha * \Omega_{Mainbody}) \tag{4.29}$$

$$\vec{F}_{Aero,z} = \frac{\rho_{Air}}{2} \cdot A \cdot \dot{\vec{r}}^2_{Mainbody,x} \cdot (C_l + C_l\alpha * \Omega_{Mainbody}) \tag{4.30}$$

The two coefficients C_l and C_d can be obtained through separate Computational Fluid Dynamics (CFD) simulations or wind tunnel tests. A represents the aerodynamic cross-sectional area and is often defined when analyzing the aerodynamic coefficients. In terms of simplifications, the following effects are amongst others not integrated:

- The change of the center of pressure with changing velocity.
- The aerodynamic influence of the spinning wheel.

4.5 Joint: Suspension

4.5.1 Overview

The task of the `Joint: Suspension` is to connect the `Solid: Wheel` to the `Solid: Mainbody`. It needs to transfers a force for each direction of movement and a torque for each direction of rotation. As described in section 2.1 its calculation is based on spring and damping effects, resulting in the following base equation for the forces:

$$\vec{F}_{Suspension} = \Delta\vec{r}_{Suspension} \cdot \vec{c}_{Suspension} + (\dot{\vec{r}}_{Wheel} - \dot{\vec{r}}_{Mainbody}) \cdot \vec{d}_{Suspension} \tag{4.31}$$

To obtain the current force of the spring, its deflection $\Delta\vec{r}_{Suspension}$, depending on the position and rotation of both bodies, must be known. For a 3D objects, this can result in complex relations. In order to keep the focus on the important parts and not to introduce too much complexity, some of the dependencies of the deflection to the body's rotation were simplified. Especially those having low or no influence in simulation with only longitudinal movement. The following simplification are introduced:

- The yaw angle of the Mainbody $\Psi_{Mainbody}$ and the Wheels is set constant at 0°.
- Because there is no change in the yaw angle, not $\vec{M}_{Suspension,z}$ is transferred.
- Since the `Solid: Wheel`'s angle around the Y-Axis is describing the rotation of its rotating parts (tire, rim and motor parts) and thus the non-rotating parts (wheel carrier, motor, suspension parts) are locked in their position, the spring and damper connection cannot be applied. For this reason $\vec{M}_{Suspension,y}$ only transfers the recoil of the motor.
- The Tyres are considered to be set at an inclination angle and position along the Y-axis, so no rolling moment or lateral force is transferred by the suspension.
- Since no later force is transferred, there is no change in track width due to the suspension.

After all these simplifications only two main suspension force directions remain, the longitudinal force $\vec{F}_{Suspension,x}$ and the vertical load $\vec{F}_{Suspension,z}$. The first should model a rigid connection, and the latter, the general damping and springing task. The analysis of the implementation of the rigid connection is done in section 4.5.3.

During the implementation of the suspension it was noticed, that because of the vertical spring there was no direct control over the driving height of the vehicle. Additionally, when setting up the 3D positions of each solid it would never result in a static equilibrium inside the spring connections. This ended up in a strange effect at the start of each simulation. For example, if the vertical initial positions of the Mainbody was set too high, then it would bounce up and down for the first seconds. For this reason, the simulation starts at a negative time like -2s in which the springs have time to settle and no motor torque is put on the wheels. At a timestamp of 0s the simulation continues as before. To account for the change in height of the main body, a z offset z_{offset} is calculated:

```
offsetZ = -(carBody.massComb * Data::instance().g.val()) / ( 4 * Data
    ::instance().wheelSusRealc.val())
```

4.5.2 Equations

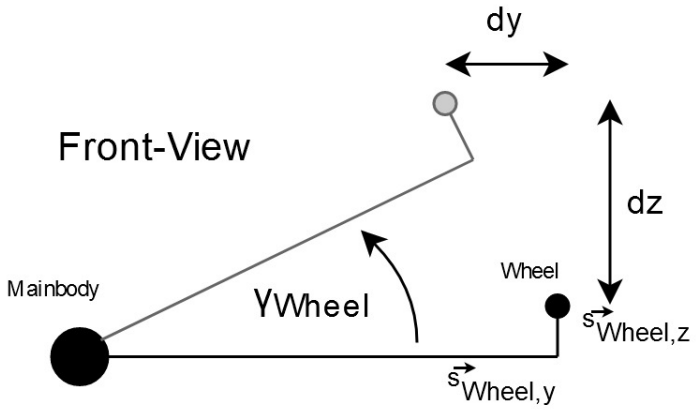


Figure 4.7: Overview forces and moments of the Joint Suspension X-plane

General force transfer:

$$\vec{F}_{Suspension} = \Delta\vec{r}_{Suspension} \cdot \vec{c}_{Suspension} + (\dot{\vec{r}}_{Wheel} - \dot{\vec{r}}_{Mainbody}) \cdot \vec{d}_{Suspension} \tag{4.32}$$

Displacement x-axis:

$$\begin{aligned}\Delta\vec{r}_{Suspension,x} = \vec{r}_{Wheel,x} - \vec{r}_{Mainbody,x} - \vec{s}_{Wheel,x} - \mathrm{sgn}(\vec{s}_{Wheel,x}) \cdot (\sqrt{\vec{s}^2_{Wheel,z} + \vec{s}^2_{Wheel,x}} \\ \cdot \cos\left(\Omega_{Mainbody} - \arcsin\left(\frac{\vec{s}_{Wheel,z}}{\sqrt{\vec{s}^2_{Wheel,z} + \vec{s}^2_{Wheel,x}}}\right)\right) - \mathrm{sgn}(\vec{s}_{Wheel,x}) \cdot \vec{s}_{Wheel,x}))\end{aligned} \tag{4.33}$$

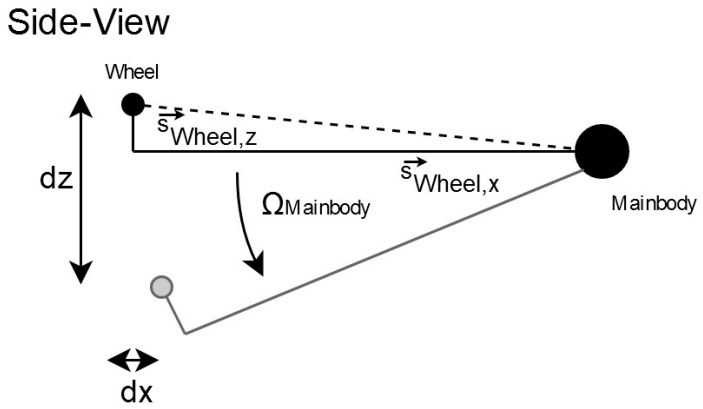


Figure 4.8: Overview forces and moments of the Joint Suspension Y-plane

Displacement y-axis:

$$\Delta\vec{r}_{Suspension,y} = 0 \tag{4.34}$$

Displacement z-axis:

$$\begin{aligned}\Delta\vec{r}_{Suspension,z} = \vec{r}_{Wheel,z} - \vec{r}_{Mainbody,z} - (\vec{s}_{Wheel,z} + z_{offset}) \\ + \operatorname{sgn}(\vec{s}_{Wheel,x}) \cdot (\sqrt{\vec{s}^2_{Wheel,z} + \vec{s}^2_{Wheel,x}} \\ \cdot \sin\left(\Omega_{Mainbody} - \operatorname{sgn}(\vec{s}_{Wheel,z}) \cdot \arcsin\left(\frac{\vec{s}_{Wheel,z}}{\sqrt{\vec{s}^2_{Wheel,z} + \vec{s}^2_{Wheel,x}}}\right)\right) + \vec{s}_{Wheel,z})) \\ - \operatorname{sgn}(\vec{s}_{Wheel,y}) \cdot (\sqrt{\vec{s}^2_{Wheel,y} + \vec{s}^2_{Wheel,z}} \\ \cdot \sin\left(\gamma_{Mainbody} + \operatorname{sgn}(\vec{s}_{Wheel,z}) \cdot \arcsin\left(\frac{\vec{s}_{Wheel,z}}{\sqrt{\vec{s}^2_{Wheel,z} + \vec{s}^2_{Wheel,y}}}\right)\right) - \vec{s}_{Wheel,z})\end{aligned} \tag{4.35}$$

Torque transfer x-axis:

$$\vec{M}_{Suspension,x} = 0 \tag{4.36}$$

Torque transfer y-axis:

$$\vec{M}_{Suspension,y} = -M_{Motor} \tag{4.37}$$

Torque transfer z-axis:

$$\vec{M}_{Suspension,z} = 0 \tag{4.38}$$

4.5.3 Solving rigid Connections

Every Solid is calculated independently. Therefore it must move independently. But at the same time there should be rigid connections between them. This creates a problem, for which a solution should be found in this section. The approach as mentioned is about spring damping connections with extremely high coefficients. If these coefficients are too low, there will be a

lot of unwanted displacement between them. But if they are too high for a certain time step, the simulation becomes unstable. The reason for this is, that a spring reaction in numerical calculation needs a deflection in a first timestep and reacts with a counter force only in the second step. So, if the timestep in relation to the force or the inertia of a moving body is too high, the deflection of the spring is exceedingly big. This, in turn, can result in an inverse deflection of the spring that is larger then the initial one and increasing the amplitude with time. This ultimately results in an unstable system.

To find the correct parameters an empirical search takes place. For its evaluation a new value `WheelFL\DisplacementWheelToBody` is implemented. It displays the distance between the target and the current position ($\Delta\vec{r}_{Suspension,x}$) relative to the Mainbody in mm.
For the test the following major parameters are used:

- $m_{Wheel} = 3kg$
- $J_{rot} = 0.27kg \cdot m^2$
- $\vec{J}_{Wheel} = \vec{0}$
- $\gamma_{Wheel} = 0°$
- $m_{MainCombined} = 300kg$ (with $\vec{r}_{m_{MainCombined}} = \vec{0}$)
- Static $M_{Motor} = 50Nm$ (no PID or motor limits)
- Tyre C19
- Without `Joint: Aerodynamic`
- All distances approximately selected to fit a regular FS

Since the only the small effect of displacement between Wheel and Mainbody is considered, only the major parameters are of interest.

Requirements:
A maximal displacement $\Delta\vec{r}_{Suspension,x} < 1mm$ is allowed at a timestep of $\Delta t = 0.00005s$. In reality, this displacement could also result of some minor backlash. Furthermore, it should be possible to simulate with 0.001s timestep without reaching instability if faster simulations are intended.
For evaluation purposes, the output of the software is shown. It displays the development of the value `WheelFL\DisplacementWheelToBody` for one track run. Hereby the x-axis of the graph shows the time in s and the y-axis the displacement in mm.

Solution “Spring” connection:

In the first attempts only the spring relationship is considered to find the stiffness of the rigid connection, as shown in eq. (4.39).

$$\vec{F}_{Suspension,x} = (\vec{r}_{Wheel,x} - \vec{r}_{Wheel,x} - \vec{s}_{Wheel,x}) \cdot \vec{c}_{Suspension,x} \tag{4.39}$$

The disadvantage of using a spring without a damper is that it oscillates constantly. For this reason, it is to be expected that the result will show a long term oscillation until the energy is consumed by some damping effects, such as a tire damper or a vertical suspension damper.
Attempt 1.1: $\vec{c}_{Suspension,x} = 10^8 N/m$ with $\Delta t = 0.001s$ unstably
Attempt 1.2: $\vec{c}_{Suspension,x} = 10^7 N/m$ with $\Delta t = 0.001s$ $t_{final} = 8.45s$

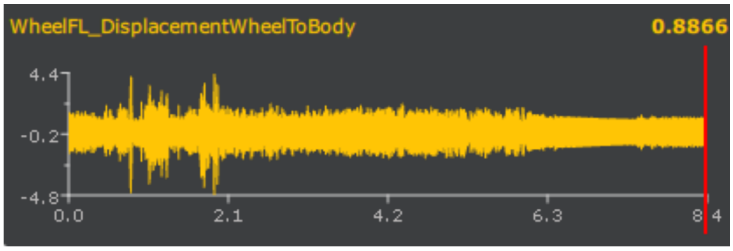


Figure 4.9: Software output: Solving rigid connections Attempt 1.2

Attempt 1.3: $\vec{c}_{Suspension,x} = 10^6 N/m$ with $\Delta t = 0.001s$ $t_{final} = 8.47s$

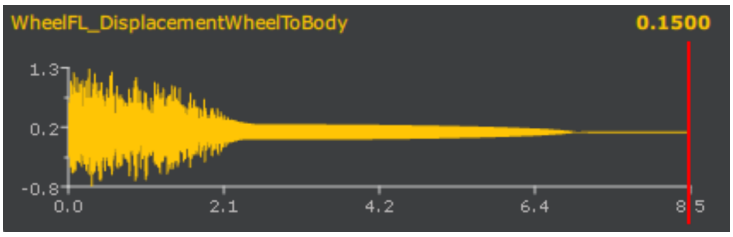


Figure 4.10: Software output: Solving rigid connections Attempt 1.3

Attempt 1.4: $\vec{c}_{Suspension,x} = 10^5 N/m$ with $\Delta t = 0.001s$ $t_{final} = 8.47s$

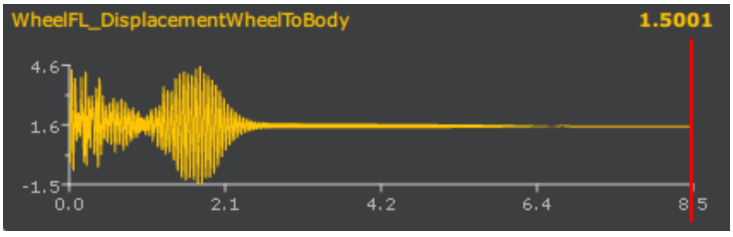


Figure 4.11: Software output: Solving rigid connections Attempt 1.4

The attempt 1.3 shows that $\vec{c}_{Suspension,x} = 10^5 N/m$ nearly fulfill the condition and seems to be at an optimum.
Attempt 1.5: $\vec{c}_{Suspension,x} = 10^5 N/m$ with $\Delta t = 0.00005s$ $t_{final} = 8.47s$

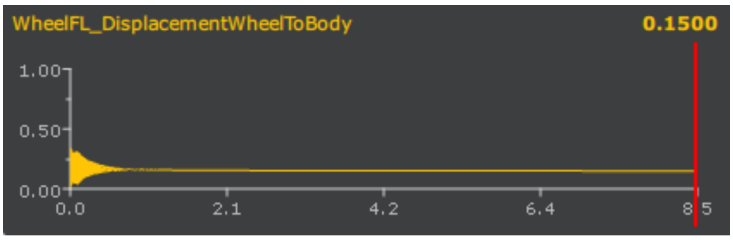


Figure 4.12: Software output: Solving rigid connections Attempt 1.5

The attempt 1.5 displays that $\vec{c}_{Suspension,x} = 10^5 N/m$ fulfills the requirements, therefore is used from now on.

Solution "Spring-Damping" connection:

Now a damper is added to reduce the oscillation.

$$\vec{F}_{Suspension,x} = (\vec{r}_{Wheel,x} - \vec{r}_{Wheel,x} - \vec{s}_{Wheel,x}) \cdot \vec{c}_{Suspension,x} + (\dot{\vec{r}}_{Wheel,x} - \dot{\vec{r}}_{Mainbody,x}) \cdot \vec{d}_{Suspension,x} \tag{4.40}$$

The critical damping coefficient should obtain the best results:

$$d_c = 2\sqrt{cm} \tag{4.41}$$

As mass parameter the mass of the Solid Wheel is used since the mass of the Mainbody is two orders higher and thus should be of less influence for the high frequency oscillation. $m = m_{Wheel} = 3kg$

Attempt 2.1: $\vec{c}_{Suspension,x} = c = 10^5 N/m$; $\vec{d}_{Suspension,x} = d_c = 3464.1 Ns/m$; $\Delta t = 0.00005s$
$t_{final} = 8.47s$

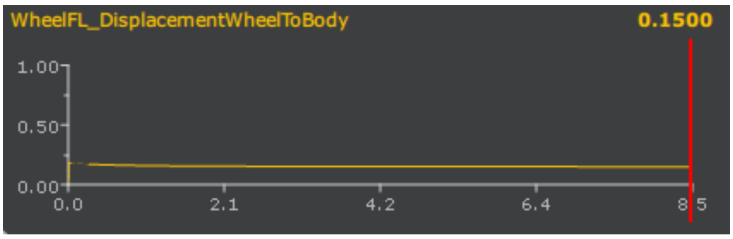


Figure 4.13: Software output: Solving rigid connections Attempt 2.1

Attempt 2.2: $\vec{c}_{Suspension,x} = c = 10^5 N/m$; $\vec{d}_{Suspension,x} = d_c = 3464.1 Ns/m$; $\Delta t = 0.001s$
$t_{final} = 8.48s$
The attempts 2.1 & 2.2 show that rigid connections with the defined requirements can be achievable using spring and damper effects. In attempt 2.2 the integration of a damper also introduces a small additional error of 0.01s, probably resulting from oscillation and consumed

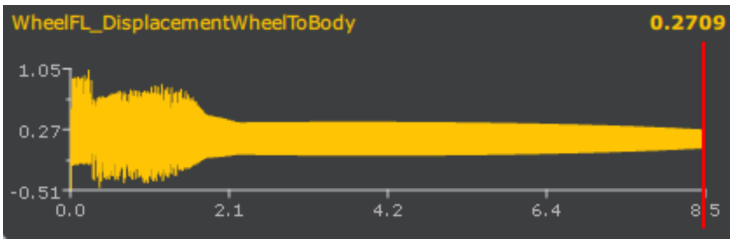


Figure 4.14: Software output: Solving rigid connections Attempt 2.2

energy by dampers. Nevertheless, this error is in an acceptable margin. The critical damping coefficient is dynamically implemented, so that if the mass changes, the damping coefficient adjust correctly, as long as $m_{Wheel} \ll m_{MainCombined}$.
Hereafter, $\vec{d}_{Suspension}$ and $\vec{c}_{Suspension}$ are defined as follows:

$$\vec{d}_{Suspension} = \begin{bmatrix} 10^5 N/m \\ 0N/m \\ c_{RealSus,z} \end{bmatrix} \tag{4.42}$$

$$\vec{c}_{Suspension} = \begin{bmatrix} 2\sqrt{\vec{c}Suspension, xm_{Wheel}} \\ 0Ns/m \\ d_{RealSus,z} \end{bmatrix} \tag{4.43}$$

In this chapter the parameters of the vertical suspension movement are chosen similarly to a rigid solid ($c_{RealSus,z} = 100000\frac{N}{m}$ and $d_{RealSus,z} = 1000\frac{kg}{s}$). This way the system is oscillating far less and therefore evaluation is simpler. More realistic values are implemented in section 5.4.

4.5.4 Verification: Conservation of Energy

The verification is about the energy conservation of the simulation. Therefore, the work introduced into the system by the motors is compared with the mechanical energy contained in the solid at the end of the simulation. The work introduced by the aerodynamic resistance and the tyre resistance is subtracted. Then the difference between the work and the energy is considered.

$$\Delta E = E_{total} - w_{total} \tag{4.44}$$

With $E_{total} = \sum E$ and $w_{total} = \sum W$. Although this verification is also not a measure of the accuracy of the simulation, it can indicate whether an error was made in the implementation of the physical models. For example, if a large amount of additional energy is introduced into the system or energy is consumed. For this purpose, the joints and solids are extended by an energy or work calculation with each time step.

The Solids preserved mechanical energy, which is calculated as follows:

$$E_{Solid} = E_{mech} = \frac{1}{2}\dot{\vec{r}}^2 \cdot \vec{m} + \frac{1}{2} \begin{bmatrix} \dot{\gamma} \\ \dot{\Omega} \\ \dot{\Psi} \end{bmatrix} \cdot \vec{J} \tag{4.45}$$

The work done by the Joints is numerically calculated for each timestep and then summed up for the entire simulation process.

$$w_{Joint}(t) = \vec{F}(t) \cdot \dot{\vec{r}} \cdot \Delta t + \vec{M}(t) \cdot \begin{bmatrix} \dot{\gamma} \\ \dot{\Omega} \\ \dot{\Psi} \end{bmatrix} \cdot \Delta t \tag{4.46}$$

In total the following effects are considered in the calculation of the work:

- Aerodynamic drag and downforce
- Tyre rolling resistance
- Motor torque

Not included are the effects of the damping of the tyres and the suspension. Therefore an energy difference $< 0kJ$ is expected.
From here on, the parameter's of the real world racing vehicle LillE from the FS Team Elbflorace are used. They are provided by the team and are a mixture of theoretical data from CAD, measurements and estimates. The complete total list is provided in table C.1. To accomplish the estimated friction coefficient of $\mu_x \approx 1.3 - 1.5$ a scaling factor $\lambda_{\mu x} = 0.5$ was chosen. The comparison to this vehicle is further proceeded in section 5.4.

Since this is the first simulation after all submodels have been implemented, the parameters of the motor controller are tuned before verification. To avoid overloading the work, the process of determining these parameters is not described in detail. Basically, first the K_p , then the K_i , and finally the K_d were fitted through several runs with the goal of minimizing the t_{final} at a time step of $\Delta t = 0.00005s$. Hereafter the following factors are used:

- $K_p = 1000$
- $K_i = 100$
- $e_{i,max} = 1000$
- $e_{i,min} = -1000$
- $K_d = 0.2$

During these runs it was found that the integration of the motor recoil into the suspension connection had a large negative influence on the energy difference. So from here on $\vec{M}_{suspension,y} = 0$ is used.
The output of the energy contained in the solids and the work done by the joints is then recorded.

- $E_{Mainbody} = 118.37kJ$
- $E_{Wheel} \approx 12.5kJ$*
- $W_{Motor} \approx 55kJ$*
- $W_{Tyre} \approx -8kJ$*
- $W_{Aerodynamic} = -11.00kJ$
- $E_{total} = 168.56kJ$
- $w_{total} = 177.30kJ$
- $\Delta E = 8.74kJ$

*These values have a slight difference between their front and rear parts. Additionally, the course of the energy in the simulation output is shown in fig. 4.15.

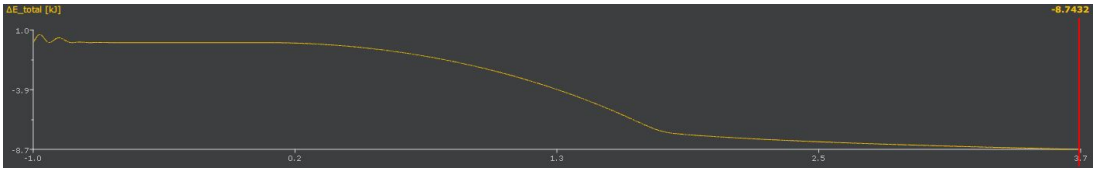


Figure 4.15: Software output: Energy difference

The results indicate a loss of about $\approx 5\%$ of the added energy. Figure 4.15 also shows that a large percentage of the difference was generated in the first half of the simulation. This part is where the main acceleration of the system takes place. After that it is limited by the max maximum allowed power output of the motors. Reasons for this difference could be, as written, the effect of the dampers, which is not considered in the working calculation. Nevertheless, this difference may indicate a small error in the models or their implementation and should be investigated in further work.

5 Results and Discussion

In this chapter, the simulation undergoes multiple verification attempts. Each one should consider a different aspect/property of the developed tool. These tests should provide the information to later compare it to the initially set requirements.
For a better exposition and to compare multiple simulation runs, an additional .csv export function was added. After each simulation, all the data contained in the `Values` is printed to the file `efrSIM_output.csv` which is placed in the root directory of the program. To reduce the data size and calculation effort the time resolution on these exports is reduced.

5.1 Simulation Performance

Simulation performance is the total time required for the calculation of one run relative to the time resolution. It does highly depend on the timestep Δt, the efficiency of the implementation and the provided computer hardware. In general, a high simulation performance is favored, since it allows to obtain information from a simulation in a shorter time.
A major attribute is the online simulation ability. This means that the simulation's performance is high enough to provide the its data in real-time and faster. Even though the requirements do not require online ability, its possibility would still be a benefit of the simulation tool in SIL applications. Therefore it is examined if real-time calculation is possible and if under which conditions.
In this section, the benefit and efficiency of the multi-threading implementation are analyzed, too, and compared to the single-core setup.
These test were all conducted with an intel I7-9750H CPU at a frequency of 4GHz. Since the c++ compiler might also influence the code performance, the MinGW 64 bit QT 5.13.2 compiler is used. A coefficient of friction $\lambda_{\mu x} = 1$ is used in this test.
In the first round, the standard setup with 1 (+1) thread is used. The second thread is the GUI thread, which updates the interface information. In this configuration a sweep from $\Delta t = 0.00005s$ up to $0.005s$ is conducted. For these runs the track time t_{final} and the required time for its calculation t_{Sim} are considered. Since the t_{Sim} fluctuates with each run, five runs for each timestep are captured and the average (avg.) time is compared. The results are displayed in table 5.1.
*These runs use a $K_d = 0.05$ because the motor controller, with parameters as defined in section 4.5.4, starts oscillating at these timestep lengths.
**Fulfills online conditions.
The results of the performance analysis with a single calculation thread indicate that a decrease in time resolution to roughly one-fifth results in half the calculation time t_{Sim} . At a timestep of about $t = 0.00015s$ the simulation allows for only operation. However, the simulation accuracy

	Full GUI $\Delta t = 0.00005$	Full GUI $\Delta t = 0.0001$	Full GUI $\Delta t = 0.00015$	Full GUI $\Delta t = 0.00015$*	Full GUI $\Delta t = 0.0005$*
Thread count	1 (+1)	1 (+1)	1 (+1)	1 (+1)	1 (+1)
t_{final} [s]	3.1323	3.1453	3.8142	3.181	3.376
1. t_{Sim} [s]	9814	4644	3621	3089	967
2. t_{Sim} [s]	9350	4630	3619	3110	964
3. t_{Sim} [s]	9333	4650	3600	3093	989
4. t_{Sim} [s]	9413	4682	3590	3096	980
5. t_{Sim} [s]	9363	4623	3581	3091	993
avg. t_{Sim} [s]	9454.6	4645.8	3602.2**	3095.8**	978.6**

Table 5.1: Verification: Performance single calculation thread

declines with the decrease of the time resolution. At timesteps > 0.00015s the configuration of the motor controller's parameter starts oscillating, resulting in an additional error in the t_{final} . By tweaking these parameters the error of the highest timestep compared to the lowest can be reduced to 7.8% with a calculation time advantage of $\approx 1000\%$

In the second examination, the GUI's functionality is strongly reduced. With this test it should be evaluated what impact the interface has on the simulation performance. Major functions, which could have an influence:

- The 3D rendering, of which the update rate is fixed to the frame rate.
- Parameter presentation which updates in real-time through Pub-Sub pattern.
- Update of the graphs, which is generally capped to 1 Hz during simulation and is conducted in the GUI thread.
- Presentation of the values with a 60 Hz update rate.

	No GUI $\Delta t = 0.00005$	No GUI $\Delta t = 0.0001$	No GUI $\Delta t = 0.0005$*
Thread count	1 (+1)	1 (+1)	1 (+1)
t_{final} [s]	3.1323	3.181	3.376
1. t_{Sim} [s]	8133	4087	867
2. t_{Sim} [s]	8132	4083	839
3. t_{Sim} [s]	8154	4059	841
4. t_{Sim} [s]	8099	4064	838
5. t_{Sim} [s]	8090	4060	850
avg. t_{Sim} [s]	8121.6	4070.6	847.0
Performance advantage [%]	16.4	12.4	14.7

Table 5.2: Verification: Performance no GUI

In this test all these functions are turned off, besides the update of the parameters, which is reduced to an update rate of 40 Hz. The results are shown in table 5.2. The performance is additionally compared to the previous full GUI single core variant.
The results indicate, that even though measures are taken against a major efficiency impact of the GUI, a performance reduction of $\approx 14.5\%$ is still present. Optimisation could take place here in future work.

In the third performance analysis, the effect of multi-threading is evaluated. Therefore, the Joints and Solids are evenly distributed over the available threads. The setup and results are displayed in table 5.3.

	Full GUI $\Delta t = 0.00005$	Full GUI $\Delta t = 0.0005$*	Full GUI $\Delta t = 0.00005$	Full GUI $\Delta t = 0.0005$*
Thread count	2 (+1)	2 (+1)	3 (+1)	3 (+1)
Thread 0 Solids & Joints	Suspension Wheel (4) Mainbody	Suspension Wheel (4) Mainbody	Suspension (1,2,3) Mainbody	Suspension (1,2,3) Mainbody
Thread 1 Solids & Joints	Motor Aerodynamic Wheel (1,2,3)	Motor Aerodynamic Wheel (1,2,3)	Suspension (4) Motor(1,2) Wheel (1,2)	Suspension (4) Motor(1,2) Wheel (1,2)
Thread 2 Solids & Joints	/	/	Aerodynamic Motor(3,4) Wheel (3,4)	Aerodynamic Motor(3,4) Wheel (3,4)
1. t_{Sim} [s]	8605	805	8655	858
2. t_{Sim} [s]	8964	805	8668	943
3. t_{Sim} [s]	8140	812	8706	982
4. t_{Sim} [s]	8139	917	8700	907
5. t_{Sim} [s]	8150	841	8550	879
avg. t_{Sim} [s]	8399.8	836.	8655.8	913.8
Performance advantage*** [%]	11.2	14.6	8.4	6.6

Table 5.3: Verification: Performance multi-threading

In section 2.3.4 a multi-threading efficiency of $\eta_{Multithreading} = 50\%$ with the eq. (2.12) was estimated. With a second calculating thread, a multi-threading efficiency of 12.9% is present and with a third thread this drops to only 3.75%. These results are far behind the expectations. With a third thread the calculation time compared to the two-thread setup even increases. This result indicates, that the software architecture contains a major bottleneck negating the advantage of additional calculation threads more or less completely. Possible reasons could be:

- Shared data implementation has a major overhead or blocks parallelism.
- Implementation of additional thread does not allow performance advantage.

In further work, these issues should be evaluated, since multi-threading could still offer the biggest performance advantage.

5.2 Stability

In this subsection, in general, the boundaries which can be simulated are discussed. In general, all submodules include their own boundaries, simplifications and conditions, like a maximum slip rate of ($\kappa > 0.2$, that the tire model's parameters correspond to a tire pressure of 80 kPa, or that the Wheel's mass should not exceed the Mainbody's one. Since these are all explained and defined in their corresponding sections, they will not be repeated in this section. The focus is more on general boundaries, resulting from the mutual influence of multiple systems. One parameter that especially influences the simulation stability is the timestep size. It has a major influence in the following areas:

- In the stability of all spring connections. Their effect in resulting in unstable systems with increasing deflection is described in section 4.5.3. Spring connections are used in the suspension and the vertical tire movement. Additionally, the tire's slip to longitudinal force translation behaves like a spring.
- In the effectiveness of the motor controller. This effect is described in section 5.1, where the PID parameters need to be adjusted to the timestep length.
- Also dampers might result in strange effects.

Since the stability of the many systems of the vehicle depend on the time resolution, its total influence ranges from oscillating values, to an additional error in track time to as far as a flying, disappearing or jumping vehicle ultimately leading to a crashing simulation. To analyze these effects, again, a timestep sweep from $\Delta t = 0.00005s$ is conducted until the simulation does not longer finish.
Besides the t_{final} , the simulation tool's output of the slip ratio of the rear tire is captured, too. This value was chosen because it displays all areas in which the timestep has a critical influence. When the wheel oscillates along the x-axis, the slip ratio is directly influenced and in the case of vertical vibration, the MFT's load changes indirectly influence the motion of the wheel as well. Also, an unstable motor controller directly influences the slip ratio.
The results are shown in table 5.4. Here, again, a peak friction scaling factor of $\lambda_{\mu x} = 0.5$ was used.
As can be seen, the error in track time increases with a decreasing timestep. With further reduced time resolution, the time error increases faster. Additionally, the oscillating part at the start of the acceleration grows in length. With timestamps > 0.002 s simulation is not possible anymore.
Besides the obvious error in simulation time, the oscillation also might be a problem when the simulation is used in a SIL application. Hence, it must also be considered during the selection of the timestep.

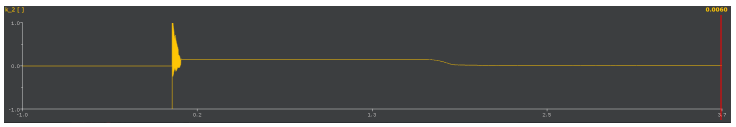
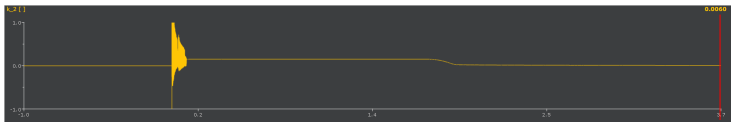
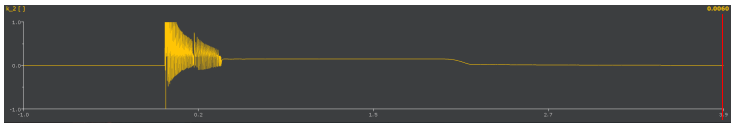
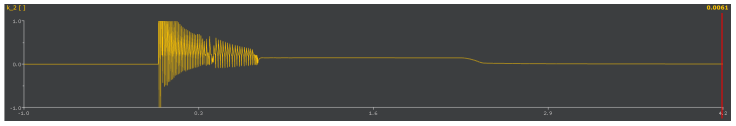
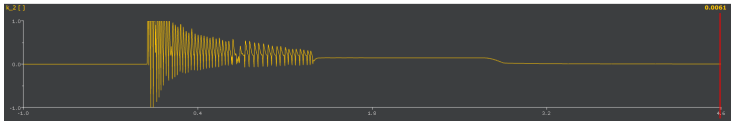
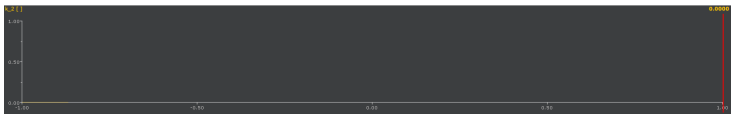
$\Delta t[s]$	t_{final} [s]	Simulation output: Slip ratio rear
0.00005	3.677	
0.0001	3.713	
0.0005	3.950	
0.001	4.191	
0.002	4.622	
>0.002	/	

Table 5.4: Verification: Stability

5.3 Influence of major Parameters

In this section the simulation should be verified by comparing different vehicle setups. So, several evaluations are carried out, each with only small changes in a parameter that has a major impact on the changed performance of the vehicle. The parameters were chosen so that the result can be easily estimated and the general trend between both outputs evaluated.

5.3.1 Major Parameter: Weight

In the first attempt, the weight of the driver is increased and it's position set backwards. The changes and the corresponding simulation results are displayed in table 5.5.
In the following enumeration, the estimated effects on the vehicles behavior are named and the corresponding changes in the results are indicated.

- Higher vehicle mass reduces the possible acceleration and, therefore, increases the track time. In the results a maximum acceleration in x-direction declines by around $1.166 \frac{m}{s^2}$ and additional $0.1151s$ are required to complete the track.
- Since all other inertia effects are equal, a relatively higher weight of the Mainbody, leads to in relative increase of its total mechanical energy compared to that of the complete vehicle. The Mainbody's energy changes from 70.2% to 73.2% of the total energy.

	Driver setup 1	Driver setup 2
m_{Driver} [kg]	53.0	80
$\vec{s}_{Driver,x}$ [m]	0.05	-1
t_{final} [s]	3.6768	3.7919
$m_{MainCombined}$ [kg]	167.99	194.99
Front m_{Wheel} [kg]	13.515	13.515
Rear m_{Wheel} [kg]	12.99	12.99
Total m [kg]	221.02	248.0
max $\ddot{\vec{r}}_{Mainbody,x}$ $[\frac{m}{s^2}]$	13.997	12.831
Front $F_{MFT,z}$ [N]	924.175	704.878
Rear $F_{MFT,z}$ [N]	1023.030	1312.852
$E_{Mainbody}$ [kJ]	118.37	127.543
Front E_{Wheel} [kJ]	12.72	11.77
Rear E_{Wheel} [kJ]	12.37	11.56

Table 5.5: Comparison: Results of different driver setups

- Because the COG of the drivers mass is pushed backwards, the tyre load ratio between rear and front changes from 0.903:1 to 0.537:1.
- Since there is more tire load overall and general acceleration is slower, the transition from being limited by the tires to being limited by maximum power output occurs later. This is shown in fig. 5.1

The initial spike of slip ratio is due the error of MFT during the initial timesteps. Because of the higher mass, the second setup needs more time to leave the critically low speeds and has a bigger spike.

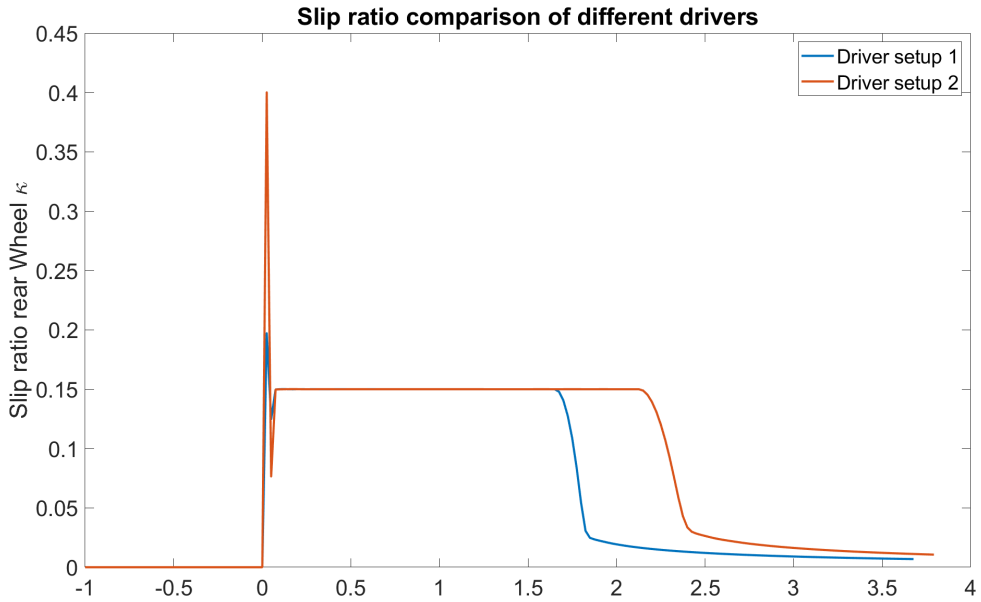


Figure 5.1: Comparison: Slip rate graph of different driver setup

5.3.2 Major Parameter: Max power

The second comparison alters the maximum's total power output of the motors from 80kW to 100kW (results in table 5.6).

	Power setup 1	Power setup 2
max power rear motor [kW]	80	100
t_{final} [s]	3.6768	3.6162
Front W_{Motor} [kJ]	55.40	54.191
Rear W_{Motor} [kJ]	54.49	69.758

Table 5.6: Comparison: Results of different power setups

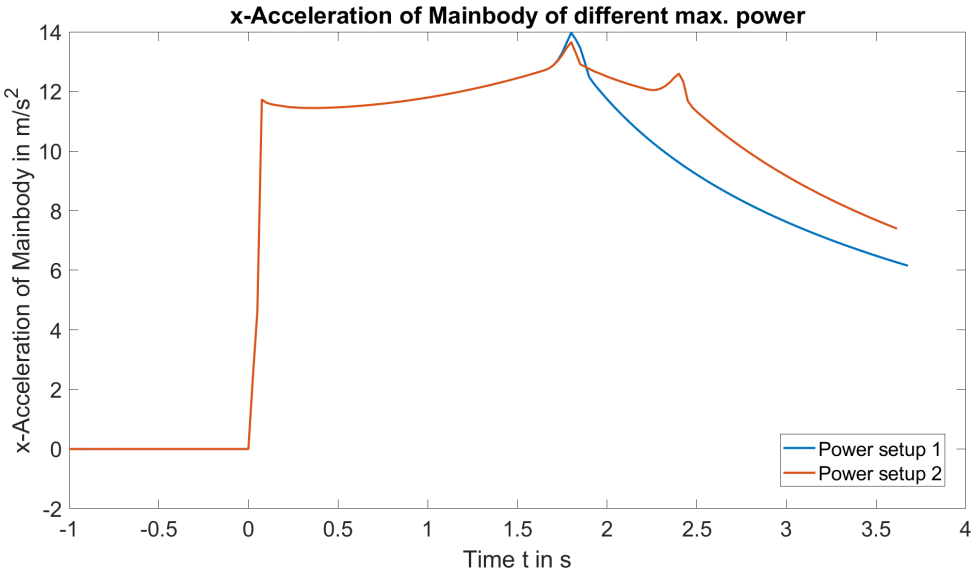


Figure 5.2: Comparison: lon. acceleration of different power setups

The following list specifies the estimated effects on vehicle behaviour and indicates the corresponding changes in the results.

- Additional power increases the longitudinal acceleration in the power capped section. However, there should be no effect during the torque/tire capped part. This is displayed in fig. 5.2 after around $t = 2s$.
- The generally higher acceleration, due to more power, decreases the total track time by 0.06s
- The initial slip rate should not be effected but it can be maintained in faster velocities (Shown in fig. 5.3).

5.3.3 Major Parameter: Wheel MOI

In the third evaluation the $\vec{J}_{Tyre,y}$ is increased.

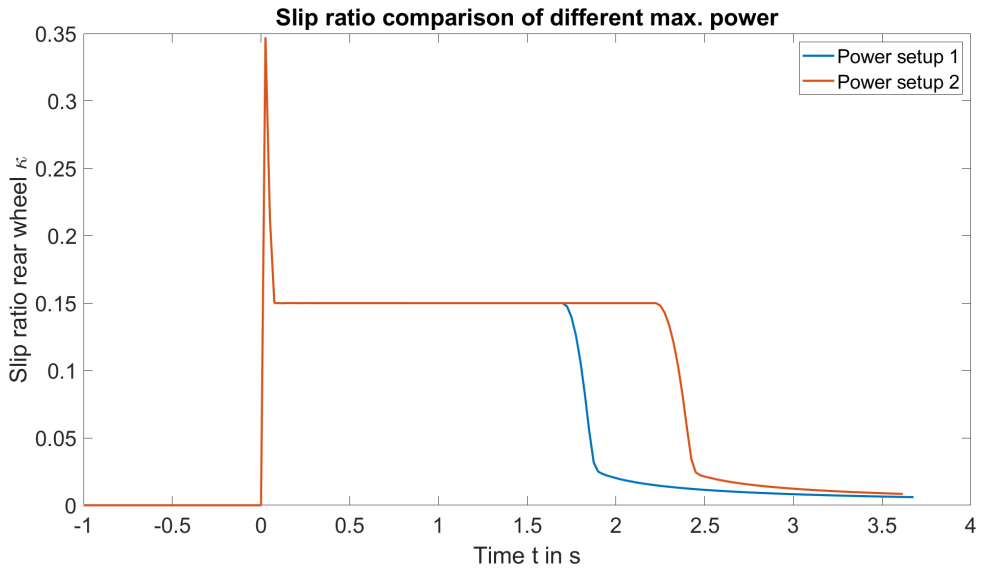


Figure 5.3: Comparison: Slip rate graph of different power setups

	MOI setup 1	MOI setup 2
Front $\vec{J}_{Tyre,y}$ $[(kg \cdot m^2)]$	0.189	0.5
Rear $\vec{J}_{Tyre,y}$ $[(kg \cdot m^2)]$	0.189	0.5
t_{final} [s]	3.6768	3.6874
Front E_{Wheel} [kJ]	12.72	16.35
Rear E_{Wheel} [kJ]	12.37	16.05

Table 5.7: Comparison: Results of different MOI setups

Increased rotational inertia requires more energy to be accelerated. This is indicated by $\approx 30\%$ of mechanical energy being contained in the Wheels at the end of the track. During the first half of the acceleration, additional torque is required to keep the tyres at a constant slip rate. Since the maximum motor moment is not reached, no time is lost during this part. However, in the second power limited phase, the higher MOI has a slight disadvantage as the maximum energy flux of the motors is reached. Both effects are displayed in fig. 5.4.

5.3.4 Major Parameter: Aerodynamic

In the last comparison, two different aerodynamic setups are compared. There, changes and results are displayed in table 5.8
The following enumeration explains, again, the different estimated amendments and indicates them in the simulation results:

- The initial phase, which is limited by the tyre, benefits from the increased load through the higher downforce. Therefore, the tyre can generate more longitudinal force. This leads to a higher acceleration and faster translation to the power limited section. This effect is displayed in fig. 5.5 from until $\approx 0.8s$. The lon. acceleration in the second aerodynamic setup is increased by $0.813\frac{m}{s^2}$.

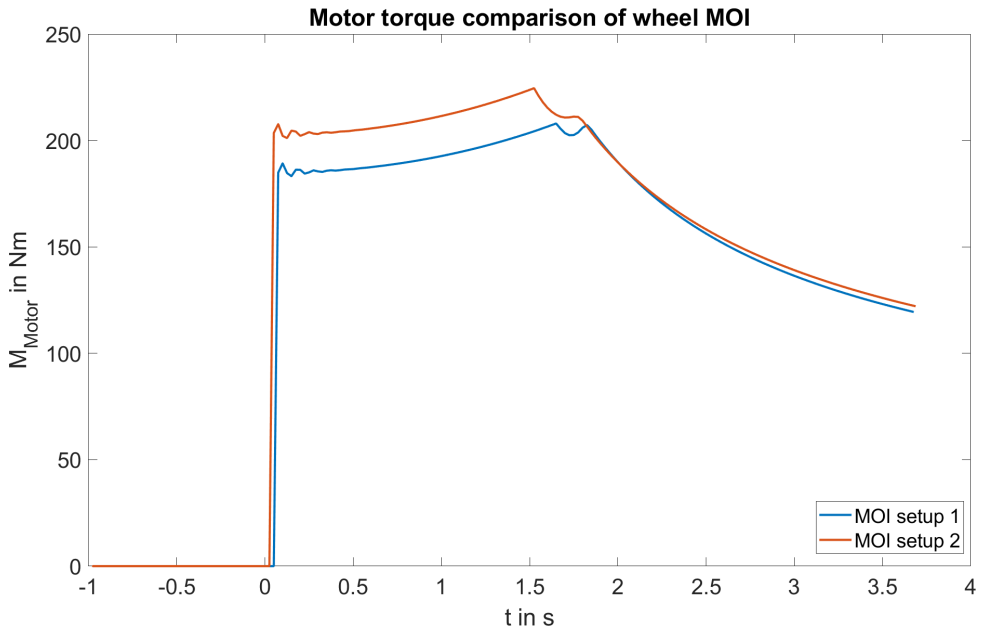


Figure 5.4: Comparison: Motor torque graph of different MOI setups

	Aerodynamic setup 1	Aerodynamic setup 2
C_l []	2	3
C_d []	0.3	0.5
t_{final} [s]	3.6768	3.6756
max $\ddot{\vec{r}}_{Mainbody,x}$ [$\frac{m}{s^2}$]	13.997	14.81
final $\vec{s}_{Mainbody,z}$ [m]	0.2763	0.2725
final $\Omega_{Mainbody}$ [°]	-0.0743	-0.1205

Table 5.8: Comparison: Results of different aerodynamic setups

- In the power limited part, the acceleration is reduced, due to additional drag. This aerodynamic forces has a quadratic relationship to the fluid velocity and therefore increases over-proportionally. This leads to a compensation of its initial advantage and ends up in a similar track time.
- The downforce has the additional effect of reducing the height of the Mainbody above the ground by $\approx 4mm$ and tilting it backwards by additional $0.05°$. (The vertical stiffness still corresponds to a rigid connection $c_{RealSus,z} = 100000$)

5.4 Comparison: to Formula Student Race-car

In this overall last verification, the simulation is compared to a real-world vehicle. The test vehicle is, as mentioned multiple time throughout this work, the FS Team Elbfloraces's 2019 racing vehicle LillE. This examination finally allows to evaluate the accuracy of the models.
The vehicle is a cutting edge development in terms of lightweight construction, equipped with a four-wheel drive electric propulsion and contains over 100 sensors for evaluation and its

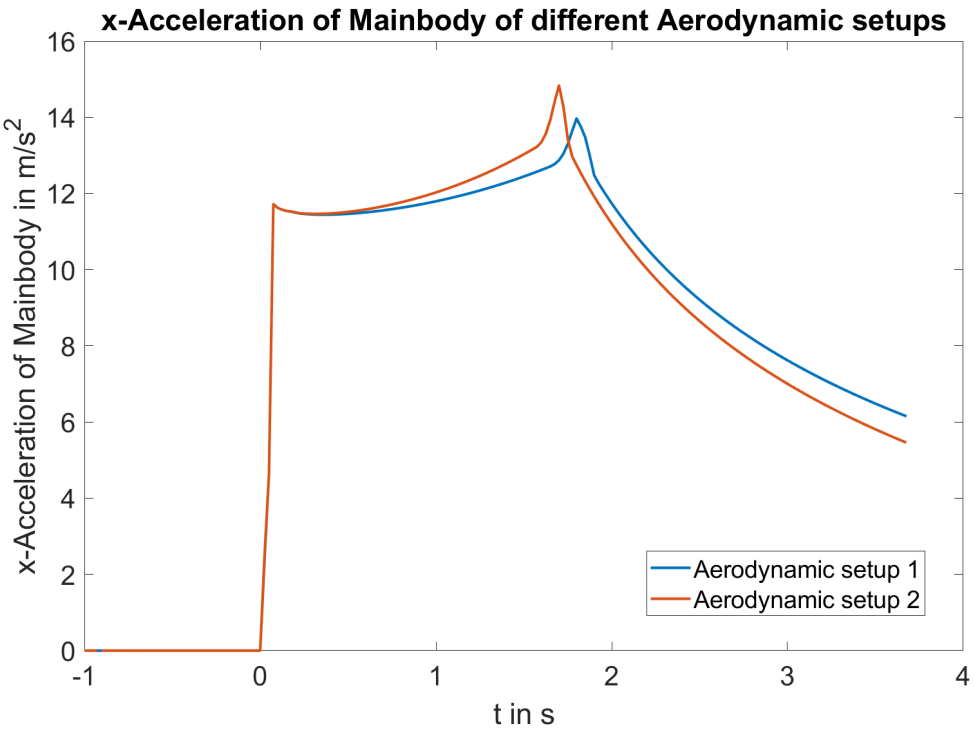


Figure 5.5: Comparison: lon. acceleration graph of different aerodynamic setups

controllers. Therefore, it is especially optimized for acceleration races. Further information of the team and its vehicles can be found on their website [12].
Since the vehicle is driven by a central unit, its sensor concept is designed to obtain data with high sample rates and distribute the same with extremely low latency through a Controller Area Network (CAN). This data is then received by the central control unit, where it can be exported for analysis. For the evaluation of the simulation, the team has provided the complete sensor data set for its FS acceleration race in Spain from summer 2019 on the Circuit de Barcelona-Catalunya. Additionally, they provided their full vehicle setup table C.1.

To adapt the simulation optimally to the vehicle and the racing conditions, the following settings are made additionally to the already configured simulation setup:

- The peak friction coefficient is one of the most important parameters of the tyre model but is simultaneously hard to obtain. For example, it depends on variable parameters like the temperature of the track and the tyres, the track's material and humidity. There was also no measurement and just an estimation by the team (1.3-1.4). For these reasons the guidelines for the tyre model of Continental is used fig. A.7. Because the acceleration event took place after sunset and the tyres were not heated the exemplary scaling factor for low tread temperature $\lambda_{\mu x} = 0.6$ is used. This resulted in a front tyre friction coefficient of 1.6 at $F_{MFT,z} = 800N$ and a rear one of 1.56 with $F_{MFT,z} = 1150N$.
- For the electric powertrain, an efficiency of around 90% is estimated. Accordingly, the maximum power output of the motors is reduced to $17000kW$ front and $19000kW$ rear.

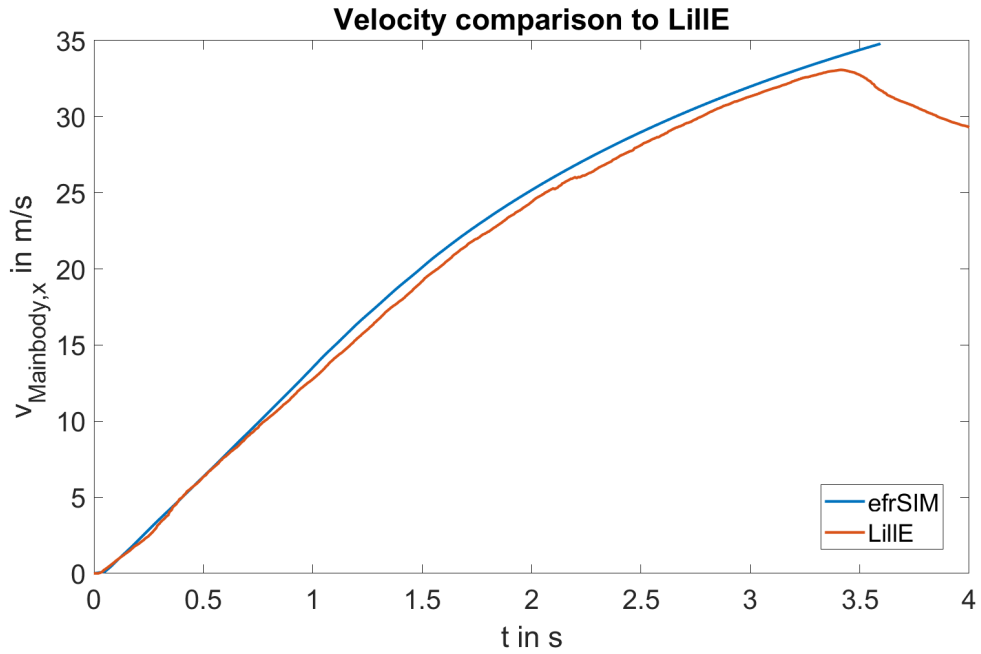


Figure 5.6: Comparison to LillE: Velocity

- Because the team has not provided parameters for the suspension's vertical stiffeners and damping, they were estimated $c_{RealSus,z} = 17000\frac{N}{M}$ and $d_{RealSus,z} = 100\frac{kg}{s}$.

To compare the simulation's output to the LillE's, major values are selected and a graph is created for each:

- Velocity of the Mainbody, see fig. 5.6
- Front slip rate, seefig. 5.8
- Rear slip rate, see fig. 5.9
- Front Motor torque, see fig. 5.10
- Rear Motor torque, see fig. 5.11
- Total motor power, see fig. 5.12
- Front motor power, see fig. 5.13
- Rear motor power, see fig. 5.14

In general, the simulation predicted a track time of $t_{final} = 3.59$ while LillE run took $3.42s$.
The velocity comparison shows that the simulation in general predicts a higher velocity than what is indicated by the data set. This contradicts with the generally lower track time of LillE. Since both vehicles travelled the same distance, the velocity of the real car must be higher if it finishes the track faster. A reason for this might be, that the velocity data of LillE is obtained through a combination of a Global Positioning System (GPS) and a Kalman filter. The result

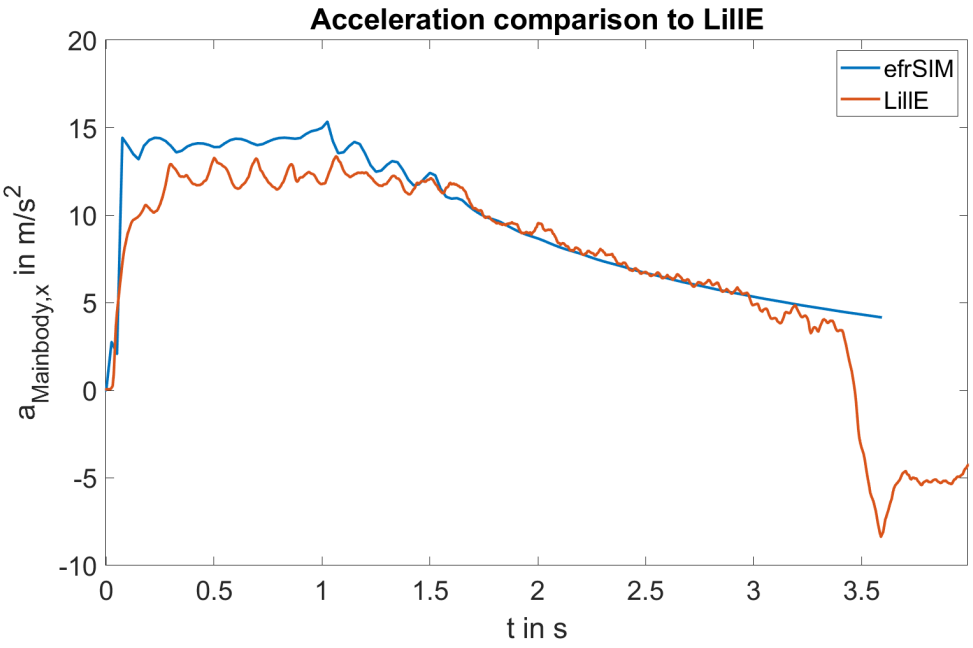


Figure 5.7: Comparison to LillE: lon. Acceleration

shows, that there might be a tuning potential for better velocity accuracy of the Kalman filter.

The acceleration comparison shows that the simulation might over predict the initial acceleration. However, since the simulation, similarly to the velocity, predicts a higher acceleration overall but still requires more time to finish the track, there might be, again, some inaccuracy to this recorded value. The value is generated through an accelerometer, which might reach its maximal acceleration boundary during the initial acceleration or just has lower accuracy in this phase. After around 1.5s when the acceleration starts being capped by the maximum power constraint, the acceleration of vehicles lowers and becomes more smooth. In this part, the predictions have a good fit to the measured data.

Since the slip rate does highly depend on the motor controller implementation, its frequency and sensor data, it is expected that the simulated ratio is far more smooth, because the simulation environment allows for perfect controller conditions. This effect is visible in both graphs. LillE's rear slip rate might also indicate, that motor reaches its maximum torque and cannot sustain the slip.

The motor torque comparison displays that in the torque/tyre limiting phase of the acceleration LillE's wheel load distribution puts more load on the rear wheels compared to the simulation. This results in an additional offset of around $75Nm$ in rear motor torque and equivalent less in the front. Therefore, LillE reaches, in this phase, the maximum motor torque and cannot sustain the slip. This effect does not appear in the simulation. The oscillation of around $150Nm$ in the rear motor results out of the oscillation of the Mainbody's rotation around its y-axis. This leads to a swinging tyre load which in turn requires different torque to sustain its slip rate. One

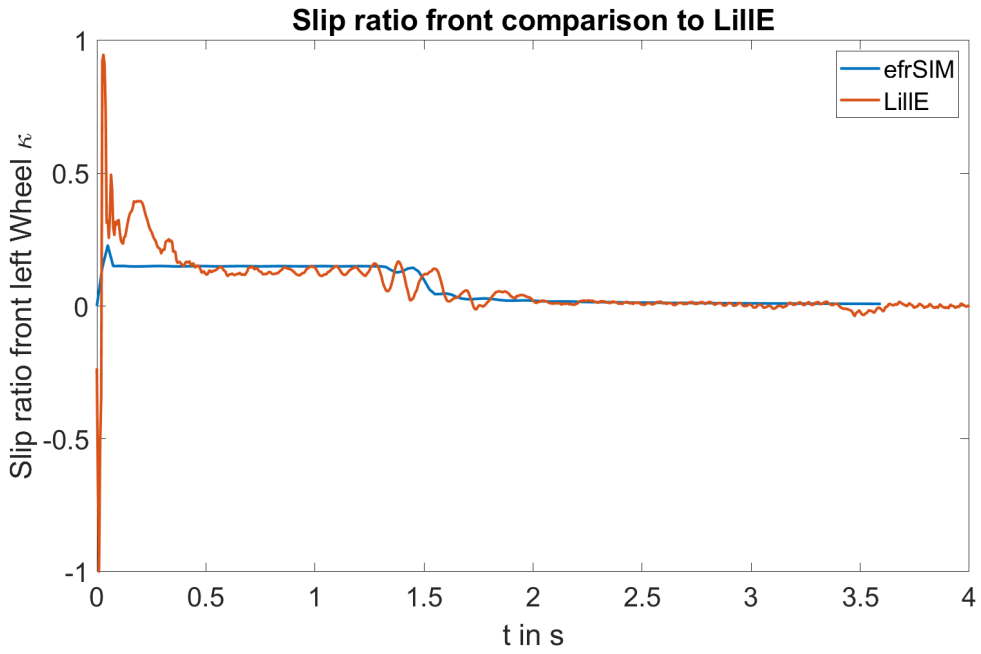


Figure 5.8: Comparison to LillE: Front slip rate

reason for the lower amplitude of the simulations oscillation is the low peak friction coefficient $\lambda_{\mu x} = 0.6$. Because the load sensitivity decreases with lower coefficient of friction [8]. This comparison generally shows:

- That the tyre model does not fit perfectly to the real condition. Reasons might be a too low friction coefficient but also the tyre pressure. The tyre model was created with a tyre pressure of $80kPa$ and during the acceleration event a pressure of $40 - 45kPa$ was used. Figure A.8 and fig. A.9 show an exemplary simulation with higher peak friction coefficient.
- The suspension model or the provided construction data of LillE does not fit well and cannot predict the wheel load distribution accurate.
- To mirror the acceleration run of LillE adjustments to the motor controller might be considered. A perfect controller might not be the best fit.

It must be noted, that the front motors of LillE turned of during the last 0.25s.
The comparison, the total power output of the motors indicate that the estimated efficiency of $\approx 90\%$ fits relatively well. In general, the simulation predicts a higher power output but requires more time to accelerate. A reason for this might be because the CAD inertia parameters used in the simulation vary from the real ones. On the other hand, the power values are obtained through revolutions per second of the motor and its torque, therefore, no efficiency is considered. The separate power output of the front and the rear motor show a similar difference to the motor torque, which, again, might result from mismatching load distribution. There is, also, the difference that in the simulation the total max power is separated for each motor, while LillE's motor controller can distribute the maximum power between each motor as needed.

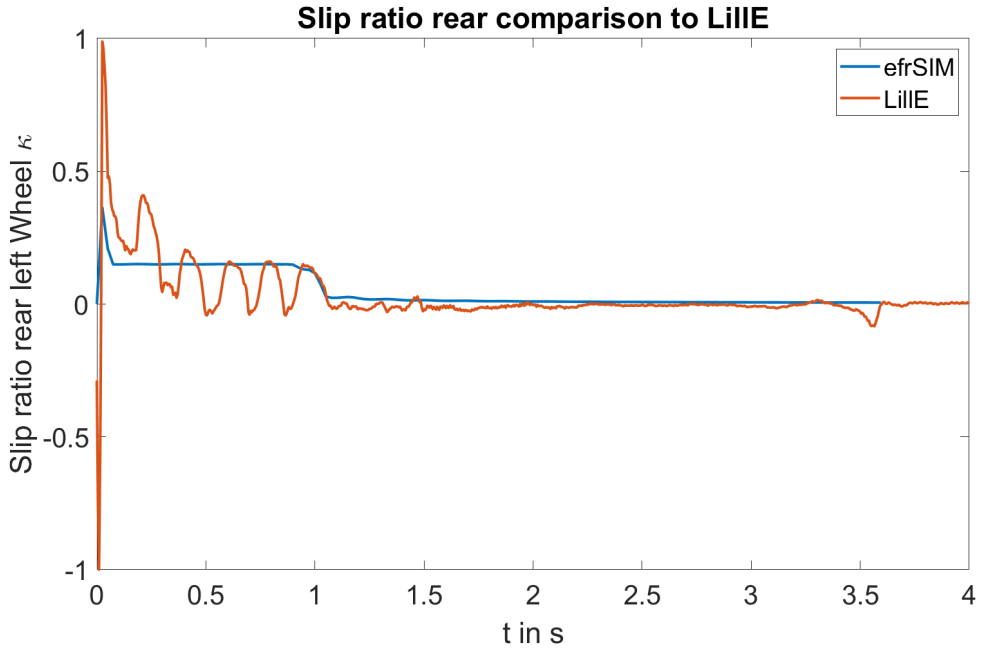


Figure 5.9: Comparison to LillE: Rear slip rate

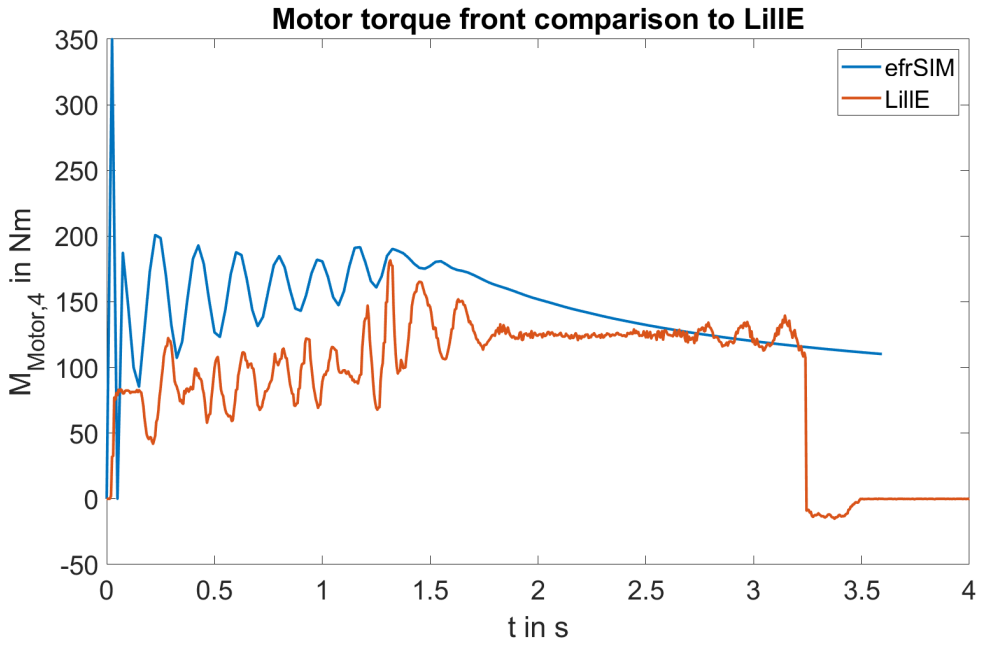


Figure 5.10: Comparison to LillE: Front Motor torque

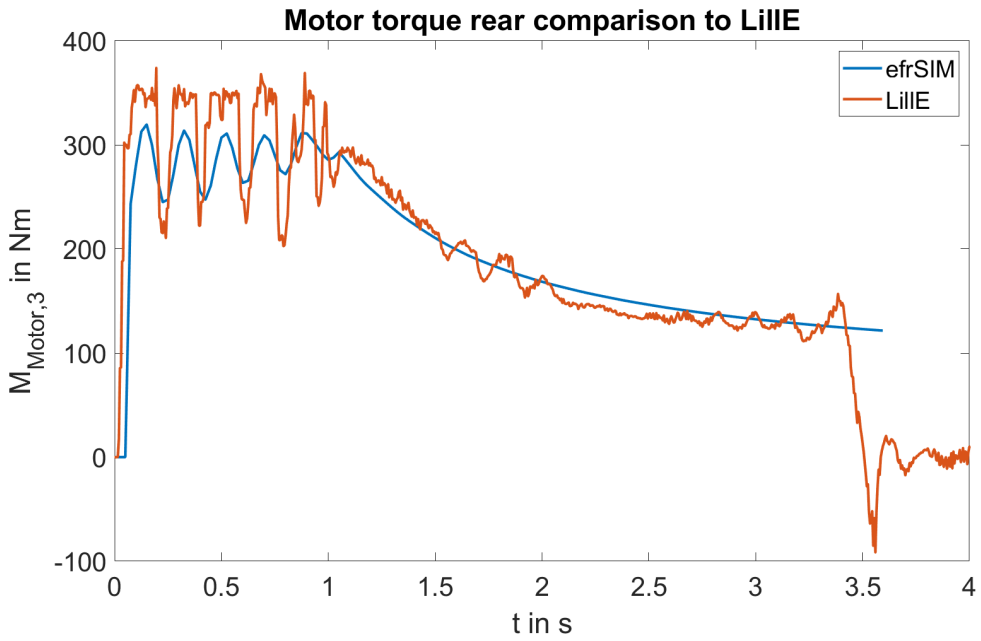


Figure 5.11: Comparison to LillE: Rear Motor torque

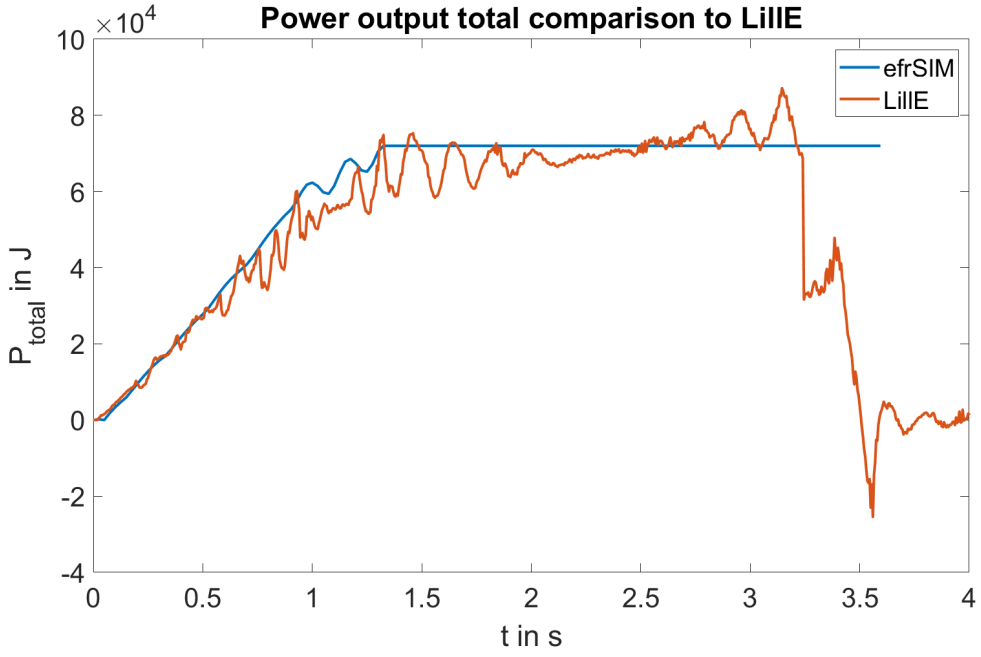


Figure 5.12: Comparison to LillE: Total motor power

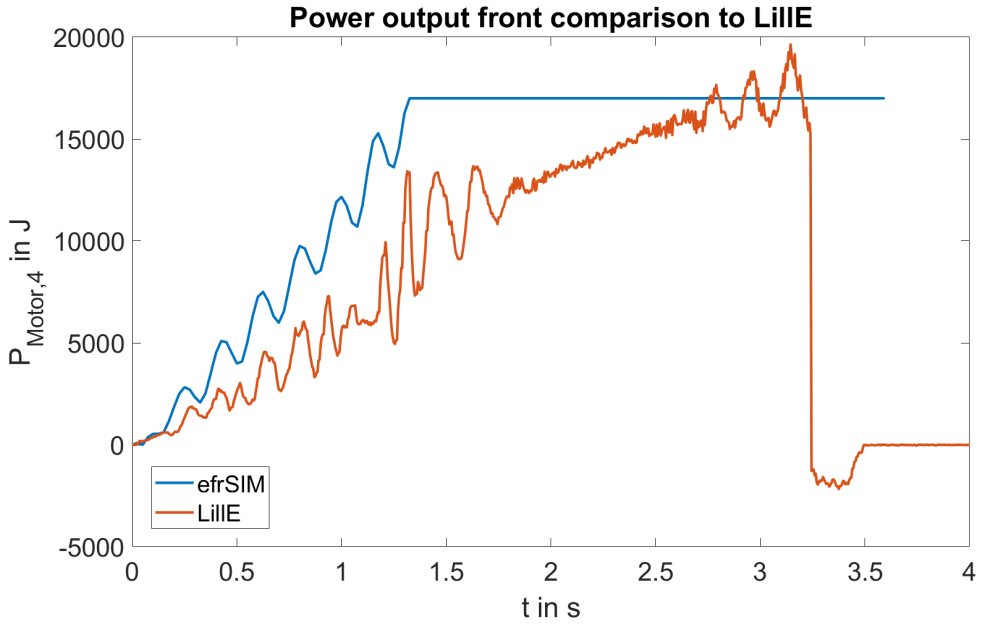


Figure 5.13: Comparison to LillE: Front motor power

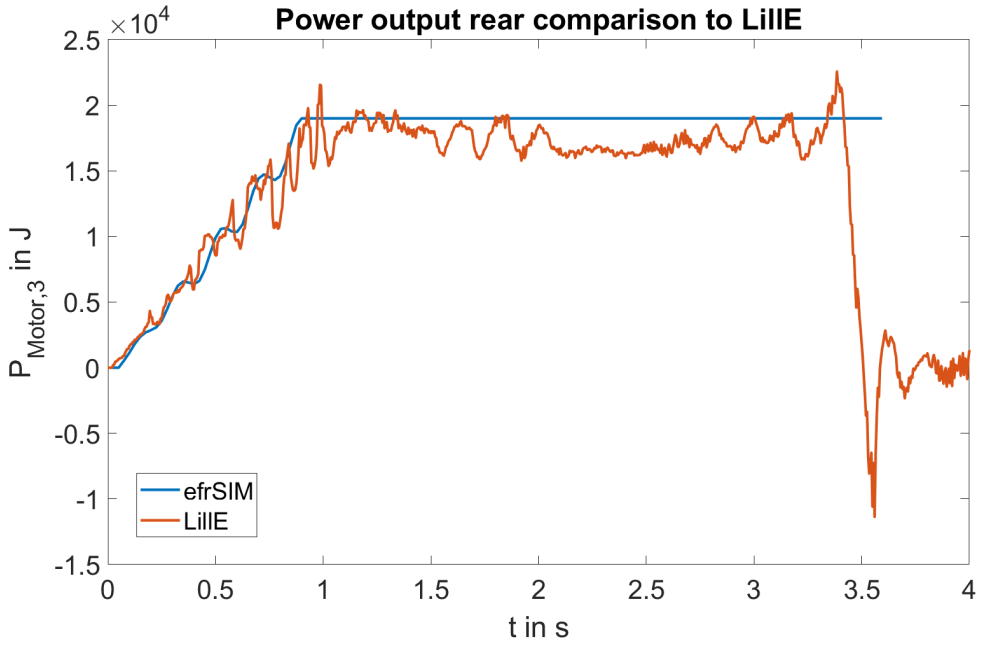


Figure 5.14: Comparison to LillE: Rear motor power

6 Conclusion

The objective of this was the development of a longitudinal simulation adapted to special vehicles according to the FS.
As mathematical basis, a combination of MBS and the numerical Verlet-algorithm was chosen. The Joint/Solid system derived from it has proven to be a good approach for a complex topic like vehicle simulation. Its modality allows to integrate modules separately and, therefore, facilitated the implementation and verification procedure. The associated separation of solving differential equations and the force calculation of each sub-model simplifies modification and comprehensibility. As it has turned out in multiple verification attempts, the discretization error that comes along with the Verlet-algorithm has a negligible error as long as the timestep stays in the microsecond range. Generally, all verifications to analytic and test data (besides the comparison to LillE) delivered satisfying results, the errors were always within an expected range resulting from the test settings. The "Rolling Wheel" verification section 2.4.6 even had an error below 0.01%.
The decision to reduce connections between solids down to spring and damper effects, enabled, on one hand, the separated calculation of the Solids' movement. But also allowed to simulate rigid connections trough high stiffness and a critical damping coefficient. These connections were able to reduce the relative movement between two solids to below $1mm$ this could in many applications even be attributed to the play in bearings. A downside of this decision is the high requirement to time resolution. Longer timesteps can result in unwanted oscillation or even an abort of the simulation. Also around 5% of energy put into the total system disappears as discovered in section 4.5.4. It is expected that this energy dissipates in the suspension or tyre damping effect, but this is not proved by the verification.

On of the goals of this simulation is to analyze the dynamic behaviour of racing vehicles and how it is influenced by its parameter. As examined in section 5.3, the concept and implementing of the simulation allows to easily adjust all of the vehicles parameter and therefore allow to investigate a wide range of vehicle setups. As long as the timestep is sufficiently small, the stability of these different settings is not endangered.
The comparison to the race vehicle Lille, show that even small effects like the oscillation of its pitch rotation and its influence on the motor torque can be displayed.
However, this verification also indicates, that the sub-model suspension is generally not the best fit in predicting load distribution correctly. Also the parameterization of the MFT model does only fit moderately. The implementation of the motor controller as a PID controller allows to exactly set up a slip rate, which might be not the best fit when mirroring the behavior of a real vehicle, because its result might be too smooth and accurate. But in general, the simulation tool offers a good possibility to study the vehicle behavior. For example it was presumed by the

Elbflorace team that the oscillation of the motor torque in the initial acceleration phase resulted only due to the misbehavior of the motor controller, but the simulation might indicate its origin in the oscillation of the Mainbody's pitch rotation.

In term of software, it was possibly to develop an open-source software framework, which creates a user-friendly and highly modifiable environment for the implementation of the simulation. Especially the initial design considerations based on software architectural concepts and the use of QML allowed an efficient development and implementation. With the usage of Qt a GUI, allowing for rapid examination of the simulated results, a relative small performance loss of around 12% was achieved. While real-time simulation above a timestep of $\Delta t = 0.00015s$ is possible, the expected performance improvement through multi-threading was not achieved. It is assumed, that a bottleneck in the shared data implementation is the reason for the only minor performance increase of 14.6% with an additional calculation thread. The developed software concept should allow an expansion for the integration of complex controllers like a vehicle driver model, to perform SIL applications.

7 Future Work

This last chapter is intended to show opportunities for proceeding with this work.

7.1 Full 3D movement & Sub-model refinement:

Since the developed simulation tool is developed with expansion and modification in mind, it is also designed to allow the integrate lateral physics. This addition would allow to analyze vehicles in far more general drive scenarios, like on a complex track. In terms of Formula Student, one could start with the relatively simple Skidpad drive scenario and later extend to Autocross. For the 3D the the following models must be extended/modified:

- The MFT tyre model must be extended with lateral slip and lateral force, additionally to some minor tweaks in its motion equations.
- The Aerodynamic is to be altered to consider a complete velocity vector.
- The Suspension transfer functions need to consider lateral movement and rotation around the yaw axis.
- A track model and driver model must be developed

In general, no modification to the MBS and Verlet simulation framework is required for 3D movement. It is, however, advisable to rethink the the Euler orientation system as effects like Gimbal lock need to be considered. An additional recommended modification to improve the model's accuracy is an update MFT model to version 6.2. The newer version contains inflation pressure changes and wider longitudinal slip ratio ranges.

7.2 Driver model SIL integration & optimization

Since the development of a driver model is a very extensive topic, this simulation tool has the opportunity to, instead, containing its own driver for complex track shapes, implementing a universal interface for external ones.
As such a connector the Robot Operating System (ROS) could be used. Therefore, the simulation can host nodes that publish data generated by virtual sensors inside the simulation, as well as virtual actuator nodes that obtain data from external controllers and are able manipulate the simulation during its processing. ROS has the advantage that it is often used in prototype development like FS vehicles. Accordingly, some driver models native use it as well. However, it does also support a wide range of programming languages and thus would allow a possible integration of many systems.
For controller-like driver models, a SIL optimization in a physic simulation creates completely different design possibilities. For example, it could allow the usage of learning-based algorithms like neural networks. These kinds of algorithms have huge advantages in their performance in complex systems. But generally have the downside of requiring exhaustive data sets for their learning and validation phase or the possibility of controlling a system in a scenario over and over again while only making small changes. These requirements can almost always not be met for systems like vehicles. This is where simulation comes into play, if the models are sufficient enough a learning-based algorithm can be trained virtually and then implemented in a real-world application. This is especially effective when the performance of the simulation is much higher than real-time and thus the training's phase of the algorithm can be massively shortened.
For this reason, it is strongly recommended to optimize the performance of the simulation tool when used for SIL applications, for example by improving the efficiency of multi-threading or integrating variable timestep lengths during simulation.

Bibliography

[1] *AVSIMULATION Homepage*. English. AVSIMULATION. URL: https://www.avsimulation.com/ (visited on 05/04/2020).

[2] J. Banks et al. *Discrete-Event System Simulation*. English. 5th ed. Prentice Hall, 2010. ISBN: 0136062121.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. English. 3rd ed. Pearson Education Inc, 2012. ISBN: 9780321815736.

[4] I.J.M. Besselink, A.J.C. Schmeitz, and H.B. Pacejka. *AN IMPROVED MAGIC FORMULA/SWIFT TYRE MODEL THAT CAN HANDLE INFLATION PRESSURE CHANGES*. English. Department of Mechanical Engineering, Eindhoven University of Technology. URL: http://www.mate.tue.nl/mate/pdfs/11281.pdf (visited on 05/07/2020).

[5] Dipl.-Ing. M. Bös. *Subsystem- and full-vehicle-simulation of mobile machines using SimulationX*. English. Liebherr-Werk-Bischofshofen GmbH, 2012. URL: https://www.fast.kit.edu/download/DownloadsMobima/ITI_Symposium_2012_Boes.pdf (visited on 04/06/2020).

[6] *CarMaker: Virtual testing of automobiles and light-duty vehicles*. English. IPG-automotive. URL: https://ipg-automotive.com/products-services/simulation-software/carmaker/ (visited on 05/04/2020).

[7] T. Colburn and G. Shute. "Abstraction in Computer Science". English. In: (2007). DOI: 10.1007/s11023-007-9061-7.

[8] *Continental Formula Student Tire: Competition Tire 2019 (C19) – Documentation*. English. Continental Reifen Deutschland GmbH, Apr. 24, 2020.

[9] *Developer Survey Results 2019*. English. stackoverflow. URL: https://insights.stackoverflow.com/survey/2019#key-results (visited on 05/04/2020).

[10] *Driving Simulation and SILAB*. English. wivw. URL: https://wivw.de/en/silab (visited on 05/04/2020).

[11] *ECS Backend*. English. Qt. URL: https://doc.qt.io/qt-5/qt3d-overview.html#ecs-backend (visited on 05/04/2020).

[12] *Elbflorace Hompage*. English. URL: https://www.elbflorace.de/en/ (visited on 04/09/2020).

[13] D. Frenkel and B. Smit. *Understanding molecular simulation - From algorithms toapplications*. English. Academic Press, 2001.

[14] *Git efrSIM*. English. URL: https://github.com/Olgidos/efrSIM (visited on 05/24/2020).

[15] *Git FSSIM*. English. URL: https://github.com/AMZ-Driverless/fssim (visited on 05/24/2020).

[16] Prof. Dr. C Holm. *Simulation Methods in Physics 1*. English. University of Stuttgart, 2012.

[17] E. M. Kasprzak. *FSAE Tire Test Consortium – ROUND 6*. English. Formula SAE Tire Test Consortium (FSAE TTC), Calspan Tire Research Facility (TIRF), 2015.

[18] K. Kunze and H. Schaeben. *The Bingham distribution of quaternions and its spherical radon transform in texture analysis*. English. Springer-Verlag, 2004. DOI: 10.1023/B:MATG.0000048799.56445.59.

[19] Dr. O. Maibaum et al. *Abschlussbericht des Projekts SiLEST*. German. DLR e.V., TU Berlin, Fraunhofer FIRST, webdynamix GmbH, Ingenieurgesellschaft Auto und Verkehr GmbH. 2007. URL: https://elib.dlr.de/54340/1/BMBF01ISC12A.pdf (visited on 03/30/2020).

[20] *MF-Tyre/MF-Swift 6.2 Help Manual*. English. TNO, 2013. URL: https://functionbay.com/documentation/onlinehelp/Documents/Tire/MFTyre-MFSwift_Help.pdf (visited on 05/10/2020).

[21] H. Pacejka. *Tire and Vehicle Dynamics*. English. 2nd ed. Butterworth-Heinemann, 2005. ISBN: 9780750669184.

[22] V. Patil. "Generic and complete vehicle dynamic models foropen-source platforms". English. 2017. URL: https://repository.tudelft.nl/islandora/object/uuid:aae755a2-a9f4-4866-9af8-714c31458f7d/datastream/OBJ/download (visited on 04/06/2020).

[23] *Rules and important documents*. English. Formula Student Germany. URL: https://www.formulastudent.de/fsg/rules/ (visited on 04/14/2020).

[24] W. Schiehlen and P. Eberhard. *Technische Dynamik*. German. 2nd ed. B. G. Teubner, 2004. ISBN: 9783519123651.

[25] D. Schramm, M. Hiller, and R. Bardini. *Modellbildung und Simulation der Dynamik von Kraftfahrzeugen*. German. 3rd ed. Springer-Verlag Berlin Heidelberg, 2018. ISBN: 9783662544808.

[26] Alison S. Tomlin Tamás Turányi. *Analysis of Kinetic Reaction Mechanisms*. English. Springer Berlin Heidelberg, 2014. ISBN: 9783662445617.

[27] *The Open Source Definition*. English. Open Source Initiatice. URL: https://opensource.org/osd (visited on 04/14/2020).

[28] T. Tischen. "Auswahl eines geeigneten Werkzeuges für den Hardware-in-the-Loop Einsatz". German. 2012.

[29] *What is the Formula Student Germany competition?* English. Formula Student Germany. URL: https://www.formulastudent.de/about/concept/ (visited on 04/09/2020).

[30] *Which programs are fastest?* English. benchmarksgame-team. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fastest.html (visited on 05/04/2020).

A Additional Figures

	Software Structure	Element Types	Relations	Useful For	Quality Attributes Affected
Module Structures	Decomposition	Module	Is a submodule of	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control	Modifiability
	Uses	Module	Uses (i.e., requires the correct presence of)	Engineering subsets, engineering extensions	"Subsetability," extensibility
	Layers	Layer	Requires the correct presence of, uses the services of, provides abstraction to	Incremental development, implementing systems on top of "virtual machines"	Portability
	Class	Class, object	Is an instance of, shares access methods of	In object-oriented design systems, factoring out commonality; planning extensions of functionality	Modifiability, extensibility
	Data model	Data entity	{one, many}-to-{one, many}, generalizes, specializes	Engineering global data structures for consistency and performance	Modifiability, performance
C&C Structures	Service	Service, ESB, registry, others	Runs concurrently with, may run concurrently with, excludes, precedes, etc.	Scheduling analysis, performance analysis	Interoperability, modifiability
	Concurrency	Processes, threads	Can run in parallel	Identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed	Performance, availability
Allocation Structures	Deployment	Components, hardware elements	Allocated to, migrates to	Performance, availability, security analysis	Performance, security, availability
	Implementation	Modules, file structure	Stored in	Configuration control, integration, test activities	Development efficiency
	Work assignment	Modules, organizational units	Assigned to	Project management, best use of expertise and available resources, management of commonality	Development efficiency

Figure A.1: Useful Architectural Structures [3]

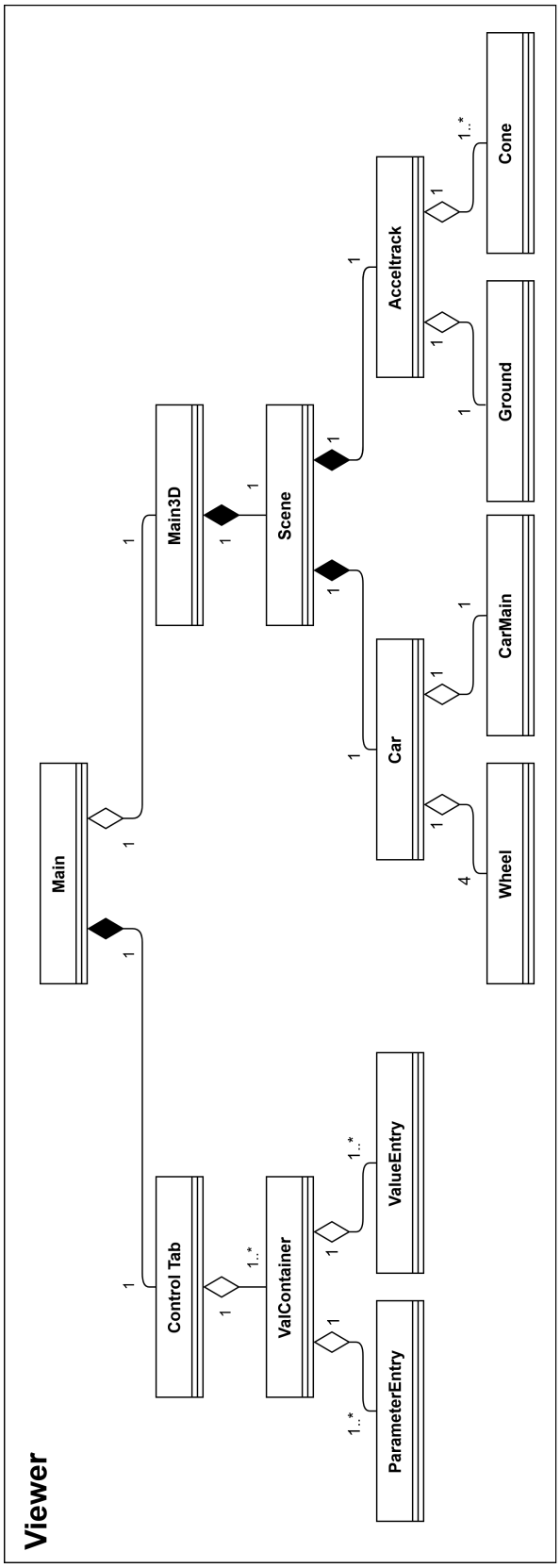


Figure A.2: UML class diagram Viewer version 1.0

Figure A.3: UML class diagram Controller

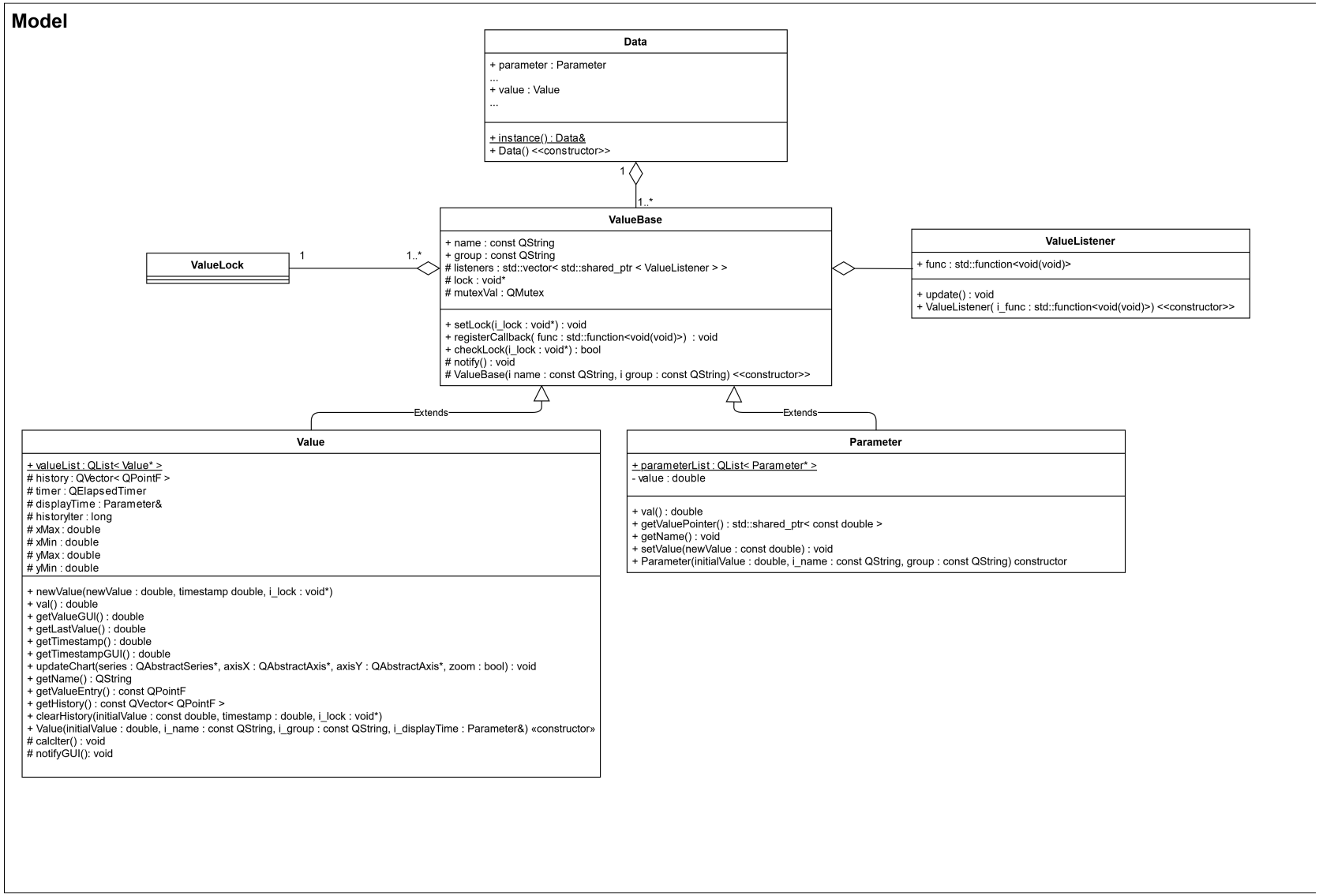


Figure A.4: UML class diagram Model Data

Figure A.5: UML class diagram Model Solid

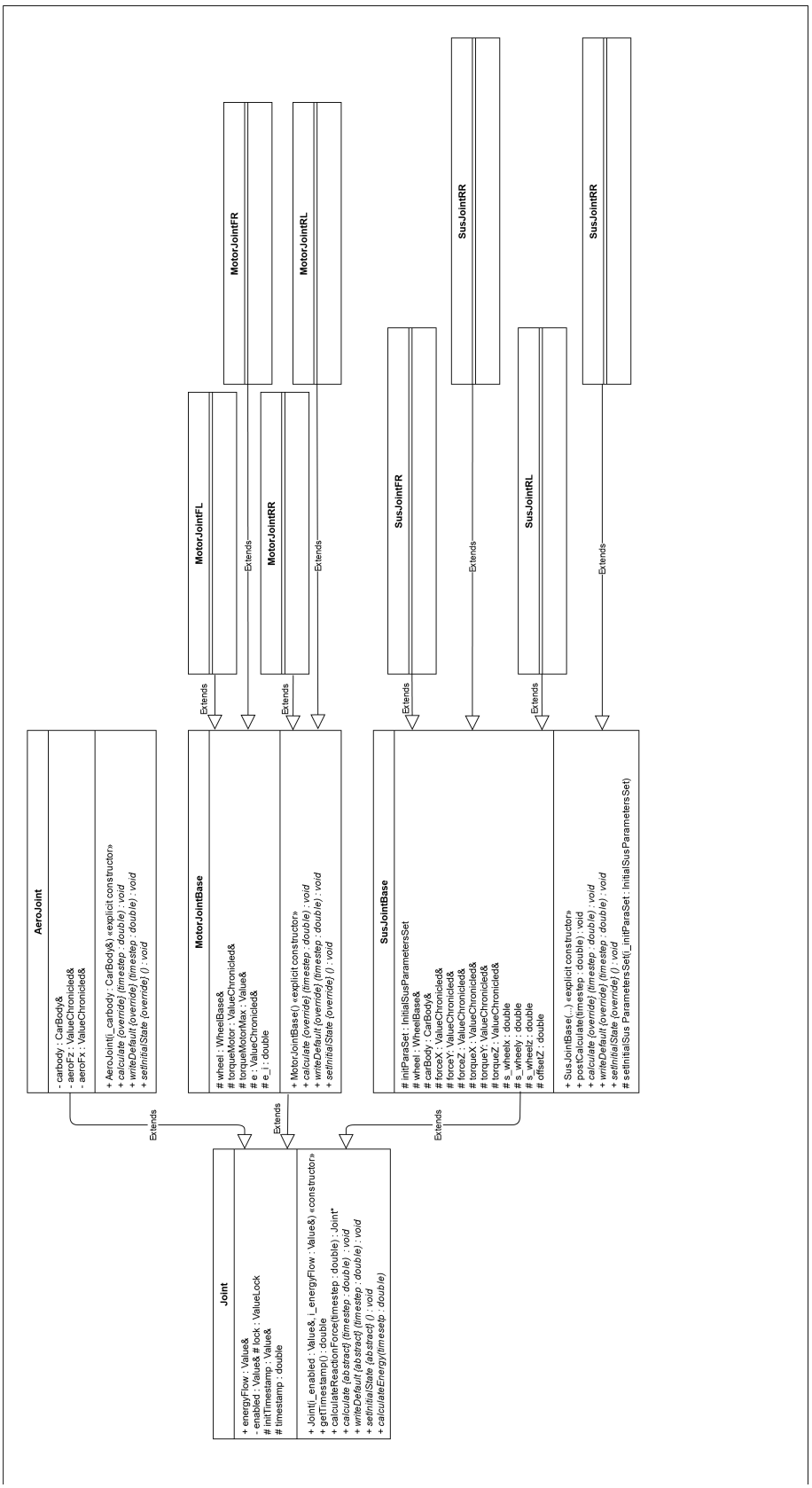


Figure A.6: UML class diagram Model Joint

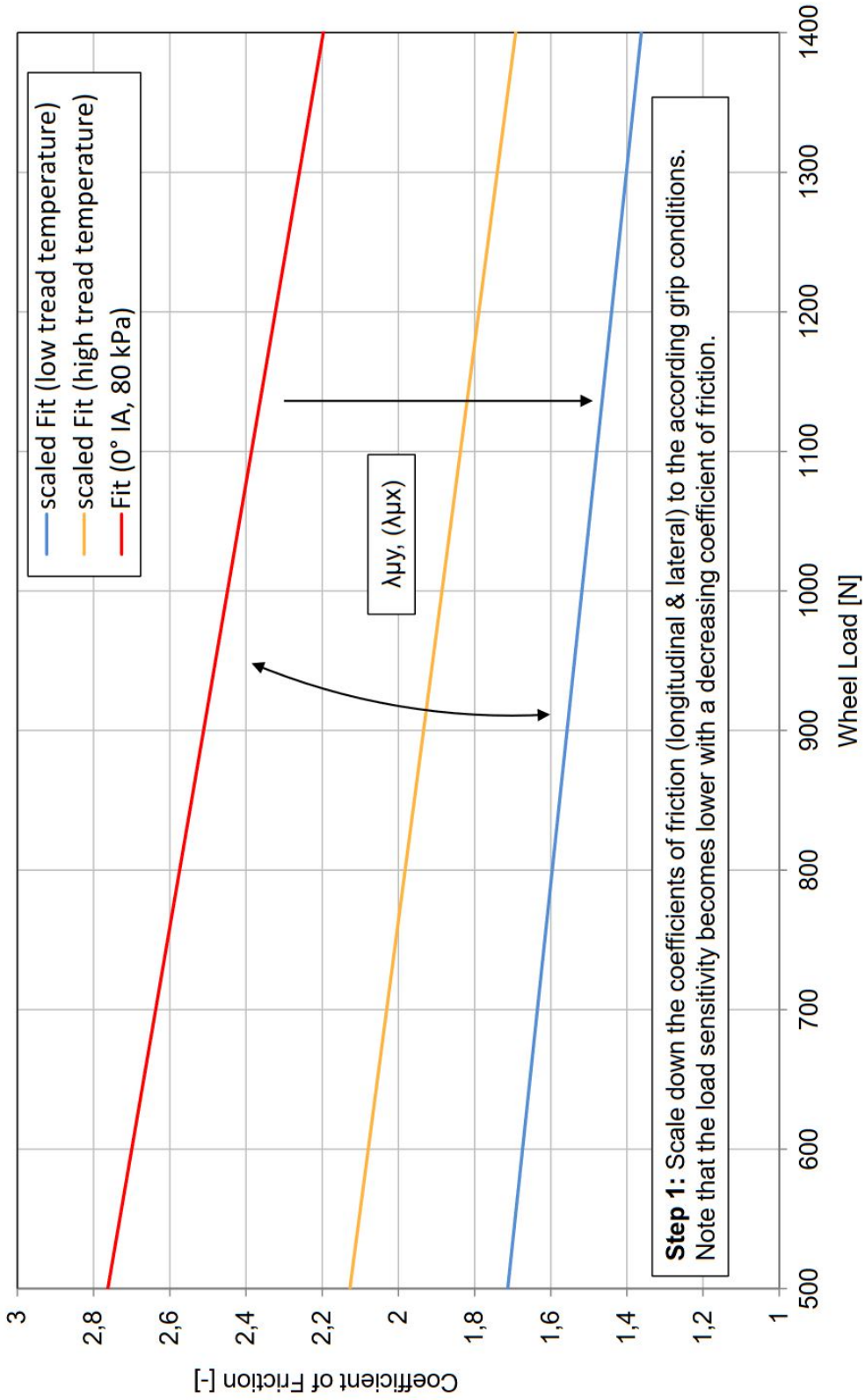


Figure A.7: Scaling advise for the MFT friction coefficient by Continental [8]

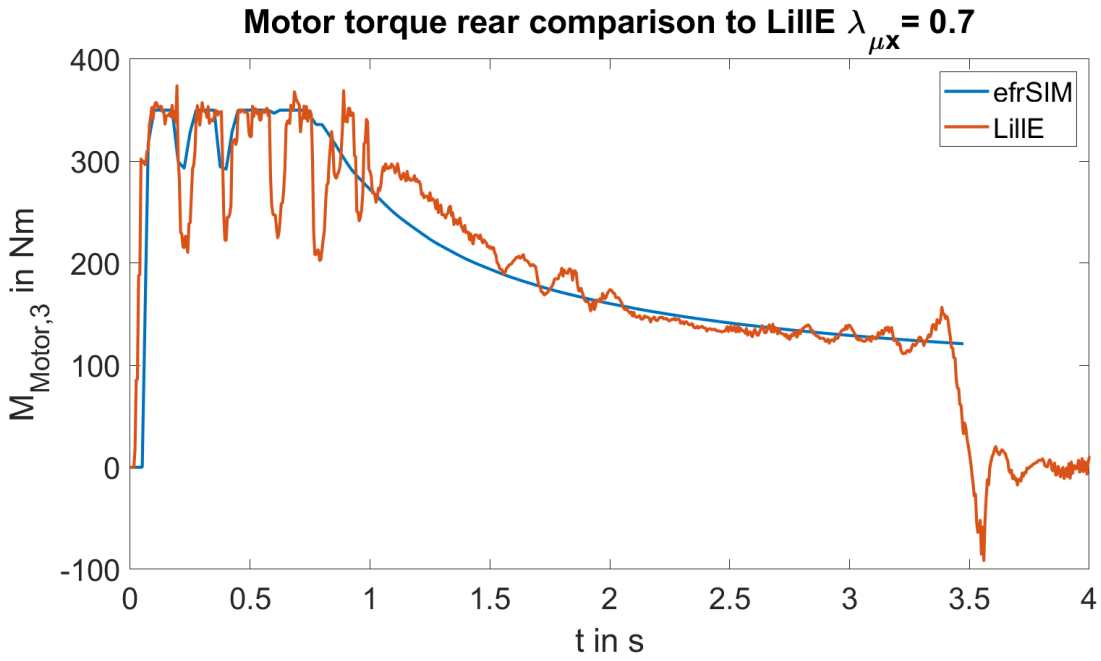


Figure A.8: Comparison to LillE: Rear motor torque with updated friction coefficient

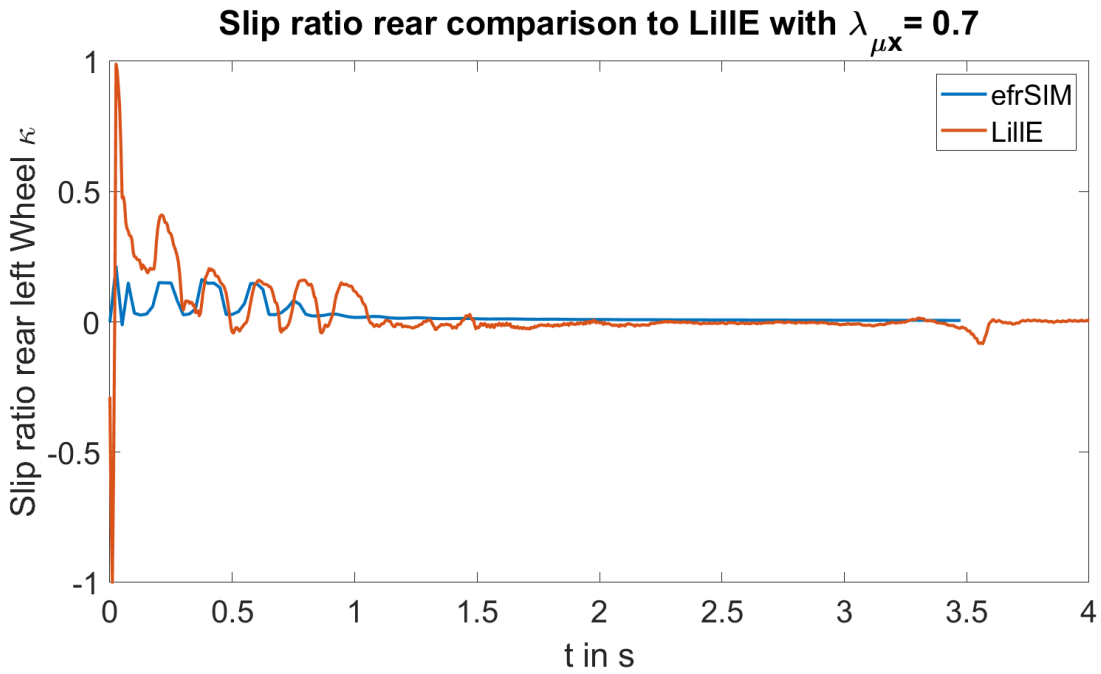


Figure A.9: Comparison to LillE: Rear slip ratio with updated friction coefficient

B Additional Information

B.1 GUI Manual

3D View movement:
Page up: forward
Page down: backward
Arrow keys: left, right, up, down
Left click + Mouse movement: rotate view

STOP, START, RESET Buttons are on the lower left side.
As soon as simulation data is available, the red marker in the *Big Chart* can be used to scroll through the simulated time and specific moments can be displayed. A double click on one of the left-sided charts will display the course of its value in the *Big Chart*.

Data View: Every `Value` available in the `Data` class is displayed here. They are all sorted into folders depending on their information group. While simulating, the folders should be closed, since drawing charts uses performance.

Parameter View: Every `Parameter` available in the `Data` class is displayed here. They can be changed with clicking on their values (keeping the mouse inside the number area) and confirm with ENTER. There are also `Parameter` for chart zoom and replay speed available.

The standard setting correspond the setup of the racecar LillE at the Formula Student Spain 2019 event.

The project can be downloaded on its GitHub page (https://github.com/Olgidos/efrSIM). There also an download link for a compiled version is provided.

B.2 C19 Continental Tyre Parameters

Extracted MFT tyre parameters from the Continental data that is provided to the "Elbflorace" TU Dresden Team:
"C19_CONTINENTAL_FORMULASTUDENT_205_470_R13_80kPa.tir"

- Tire Width: 0.200 m
- Tire Unloaded Radius r_0 : 0.235 m
- Tire Aspect Ratio: 0.34
- Rim Width: 7 in
- Rim Radius: 0.1651 m
- Inflation Pressure: 80 kPa
- Vertical stiffness c_{Tyre} : 9.882e+004 N/m
- Vertical damping d_{Tyre} : 3.429e+003 Ns/m
- $F_{MFT,z0} = 800N$
- $p_{Cx1} = 1.786$
- $p_{Dx1} = 2.688$
- $p_{Dx2} = -0.272$
- $p_{Dx3} = 13.700$
- $p_{Ex1} = 0.871$
- $p_{Ex2} = -0.038$
- $p_{Ex3} = 0.000$
- $p_{Ex4} = 0.071$
- $p_{Kx1} = 81.250$
- $p_{Kx2} = -20.250$
- $p_{Kx3} = 0.500$
- $p_{Hx1} = 0.000$
- $p_{Hx2} = 0.000$

- $p_{Vx1} = 0.000$
- $p_{Vx2} = 0.000$
- $q_{sx1} = 0.000$
- $q_{sx2} = 1.385$
- $q_{sx3} = -0.009965$
- $q_{sy1} = -0.0309$
- $q_{sy2} = -0.0921$
- $\lambda_{Cx} = 1$
- $\lambda_{\mu x} = 1$
- $\lambda_{Ex} = 1$
- $\lambda_{Kx\kappa} = 1$
- $\lambda_{Hx} = 0$
- $\lambda_{Vx} = 0$
- $\lambda_{Fz0} = 1$
- $\lambda_{My} = 1$
- $\lambda_{Mx} = 1$

C Additional Tables

Parameter	Unit	General	X-axis	Y-axis	Z-axis
m_{Rim}	[kg]	1.45	/	/	/
m_{Tyre}	[kg]	3.95	/	/	/
$m_{Suspension,1}$	[kg]	1.83	/	/	/
$m_{Suspension,2}$	[kg]	1.78	/	/	/
$m_{Suspension,3}$	[kg]	1.78	/	/	/
$m_{Suspension,4}$	[kg]	1.83	/	/	/
$m_{Assembly,1}$	[kg]	7.2	/	/	/
$m_{Assembly,2}$	[kg]	6.7	/	/	/
$m_{Assembly,3}$	[kg]	6.7	/	/	/
$m_{Assembly,4}$	[kg]	7.2	/	/	/
$m_{Mainbody}$	[kg]	111.38	/	/	/
m_{Driver}	[kg]	53.0	/	/	/
$\vec{J}_{Tyre}$	$[kg \cdot m^2]$	/	0.116	0.189	0.116
$\vec{J}_{Rim}$	$[kg \cdot m^2]$	/	0.022	0.035	0.022
$\vec{J}_{Mainbody}$	$[kg \cdot m^2]$	/	32.487	41.1	68.289
$J_{Rotor/Transmi}$	$[kg \cdot m^2]$	/	0.001	0.004	0.001
$\vec{r}_{Driver}$	[m]	/	0.05	0	0.012
$\vec{r}_{Wheel,1}$	[m]	/	0.799	0.6	0.05
$\vec{r}_{Wheel,2}$	[m]	/	-0.763	0.575	0.05
$\vec{r}_{Wheel,3}$	[m]	/	-0.763	-0.575	0.05
$\vec{r}_{Wheel,4}$	[m]	/	0.799	-0.600	0.05
$\vec{r}_{Suspension,1}$	[m]	/	0.799	0.130	0.160
$\vec{r}_{Suspension,2}$	[m]	/	-0.763	0.130	0.160
$\vec{r}_{Suspension,3}$	[m]	/	-0.763	-0.130	0.160
$\vec{r}_{Suspension,4}$	[m]	/	0.799	-0.130	0.160
$\vec{l}_{Assembly,1}$	[m]	/	0.0	-0.02	0.02
$\vec{l}_{Assembly,2}$	[m]	/	0.0	-0.02	0.02
$\vec{l}_{Assembly,3}$	[m]	/	0.0	0.02	0.02
$\vec{l}_{Assembly,4}$	[m]	/	0.0	0.02	0.02
$h_{Mainbody}$	[m]	0.285	/	/	/
C_l	[ ]	2.0	/	/	/
C_d	[ ]	1.0	/	/	/
$C_l\alpha$	[1/°]	0.6	/	/	/
$C_d\alpha$	[1/°]	0.8	/	/	/
A	$[m^2]$	1	/	/	/
ρ_{Air}	$[\frac{kg}{m^3}]$	1.225	/	/	/
g	$[\frac{m}{s^2}]$	9.81	/	/	/
max Power	[W]	80000	/	/	/
max Torque	[Nm]	350	/	/	/
κ_{eff}	[]	0.15	/	/	/
μ_x	[]	$\approx 1.3 - 1.4$	/	/	/
p_{Tyre}	[kPa]	$\approx 40 - 45$	/	/	/

Table C.1: Parameter set of FS racecar LillE

D CD Data

D.1 Digital Version of the Thesis

D.2 Software

D.2.1 Source Code

D.2.2 Compiled Simulation

D.3 UML Diagrams

D.4 Data Set LillE FS Spain acceleration Race

D.5 Additional Documents