Master's degree in Advanced Automotive Engineering

Racing Car Design

Development of a Toolchain for parametrisation of a simplified double track model for laptime simulation starting from a multibody model currently in use at the DiL simulator

Supervisor:

Prof. Ing. Andrea Toso

Company Supervisor:

Ing. Jacopo Gentile

Author:

Petra Ivezić

190141

Academic Year 2025/2026

I want to thank my mentor and colleagues in Dallara Automobili for this opportunity and the shared knowledge during this internship, as well as my professor for the mentorship.

I want to thank my friends back home, and all the new friends that I met during these 2 years who I couldn't have done this without.

Mostly I want to thank my family, for the unconditional support in everything I do.

Petra Ivezić

TABLE OF CONTENTS

LIST OF FIGUERS

SUMMARY

The objective of this thesis is to develop a toolchain for the parameterisation of a simplified double track vehicle model for laptime simulation. Since the simplified model is implemented in Python, the toolchain is primarily developed in Python, while utilising multibody vehicle models in Dymola to extract the vehicle parameters required to run the simulation.

The work is divided over five main areas: static vehicle analysis, acceleration and braking, tyres, suspension and aerodynamics. For each of these areas, a script is developed to extract and process the necessary data.

The static vehicle analysis defines the vehicles mass and geometric characteristics. The acceleration and braking module computes the torque available at the wheels and determines the maximum braking torque capability. The tyre module focuses on the optimisation, fitting the simple Pacejka formula, currently implemented in the simplified model, to the 5.2. Pacejka tyre model. In the aerodynamics module, the existing multi-inputs variable linear interpolation of aerodynamic maps is replaced with a neural network approach. Lastly, the suspension module introduces the suspension nonlinearities into the model by defining the motion ratio of elastic elements using the principle of virtual work.

The developed toolchain represents a framework for bridging multibody and simplified simulation models.

Due to confidentiality reasons, the final code won't be shown, and the graphs will be omitted of numerical values.

Keywords: toolchain, vehicle dynamics, Dymola, Python, Dallara

1. Introduction

1.1. Dallara Automobili

This thesis was developed as a part of curricular internship done in the Vehicle Dynamics and Project Development department at Dallara Automobili.

Dallara Automobili is an engineering company founded by Giampaolo Dallara in 1972, with its headquarters located in Varano de' Melegari, Italy. The main activities are split over 3 business units Racing, Automotive and Aerospace. Over the years, the company has earned a winning reputation in motorsport, emphasising its expertise in aerodynamics , vehicle dynamics and lightweight materials design.

Within the racing business unit, Dallara is currently a sole supplier for several top-level international championships. These include the IndyCar Series, Indy NXT, FIA Formula 2 Championship, FIA Formula 3 Championship, and Super Formula. Dallara is also partnered with Hass F1 since 2016. Furthermore, the company is actively involved in the World Endurance Championship partnering with Cadillac, BMW, Ferrari, and soon McLaren.

Figure 1. Dallara headquarters in Varano de' Melegari

Vehicle dynamics activities at Dallara include simulation, indoor testing, and outdoor testing. The Vehicle Dynamics and Project Development department focuses on the development and application of in-house models and tools used to predict vehicle behaviour and support design decisions. The models carry out so called “offline” simulations with virtual driver, or “real-time” simulations at the driving simulator with “driver in the loop”. DiL simulator presents an integral part of the department. It allows full vehicle simulations to be executed with real-time response, combining detailed subsystems models such as powertrain, control systems, and tyres, together with highly accurate track representations. DiL simulator has the ability to reproduce both the physical behaviour of the car and the driving sensation allowing drivers to provide meaningful feedback, making it a fundamental tool for vehicle development and performance optimization.

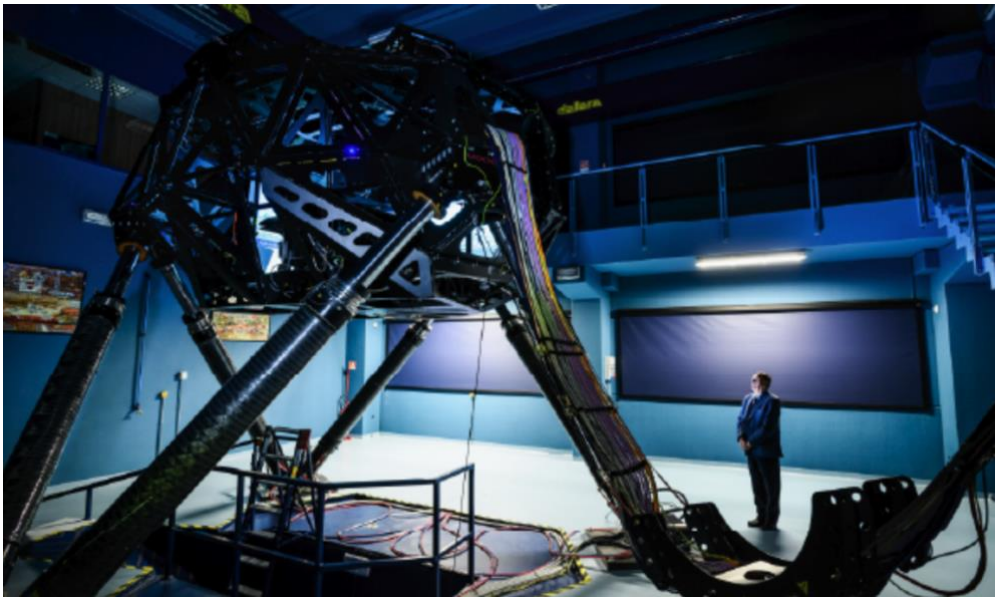

Figure 2. DiL Simulator at Dallara in Varano de' Melegari

1.2. Motivation and objectives of the work

The main task of Vehicle Dynamics department is to simulate vehicle performance and behavior with a high level of accuracy, providing essential input for the overall design process and performance optimisation. As already mentioned, in the department there are very complex multibody models that represent the complete vehicle, with all of it subsystems, that run offline or in real time within the DiL simulator. Currently, the department is developing a reduced order model based on simplified vehicle representation, a double-track model, which needs to allow significantly faster computation times. These types of models are particularly useful in the early phases of a project, when not all vehicle characteristics are fully defined. Despite their reduced complexity, these models must retain as much relevant information as possible, which is achieved by utilizing the DiL multibody models within the model parametrization process.

The objective of this work is to develop a toolchain for parametrisation of a simplified double-track model, intended for laptime simulation, starting from the multibody models currently in use at the DiL simulator. The approach is based on utilising available detailed vehicle models and extracting relevant information either directly from their parameter files or through dedicated simulation experiments.

As the goal of the simpler model is to run simulations faster, also the parametrisation process must be efficient and largely automated in populating the model. Since the smiplified model is developed in Python, the core of the toolchain is also developed in Python. In addition to Python scripts used to extract and process data, Modelica (Dymola) scripts are used to interface with the multibody models and generate the required simulation outputs. The final outcome of this process is a set of parameters compatible with the new simplified simulation model, ensuring consistency between the two simulation environments.

1.3. Tools used

The Toolchain is primary developed in Python, which serves as the central platform of the toolchain, enabling efficient data handling and automation of the parametrisation process. The required data is extracted either directly from the vehicle parameters file or by setting up decicated simulation experiments in Dymola. These experiments can either be run in Dymola using Modelica scripts, or by exporting them as a Functional Mockup Unit (FMU) to then run the simulation directly from Python. Both approaches are used where appropriate and will be explained in more detail in the later sections outlining the toolchain workflow. At this stage, only a general overview of the tools used is provided.

1.3.1. Python

Python is an open-source programming language widely used in engineering because of its scientific and numeric computing libraries. It enables data processing and numerical analysis making it suitable for developing toolchains such as the one presented in this work. Additionally, what is usefull for this work, specific libraries enable the execution of FMUs directly within Python, allowing seamless integration between Dymola-based models and the Python environment. Examples of such libraries include *FMPy* and *PyFMI*, which provide interfaces for loading and simulating FMUs, as well as extracting results for further processing.

1.3.2. Dymola

Dymola, which stands for dynamic modeling labartory, is a commerical modeling and simulation enviroment for anlaysing complex physical systems. Dymola is based on the Modelica language. Modelica is an open source, object-oriented, equation based modeling language. Modelica is a very powerful language since it allows users to build models from preexisitng classes that can also be taken from third-party companies. Another advantage is the hirarchy approach to parameters definition. Complex models can be made of many classes and subclasses but the simulation only considers the parameters that are assined at the top level. In this way the users don’t have to enter each subclasse in order to modify the model.

Dymola interface has five main sections.

- Graphics - in this section the user can see the subsystems as blocks, that make up the model, and the signal flows, and can edit the parameters in a graphical way.
- Text - in the text section users can access the code (equations) behinde the graphic model of the selected class.
- Documentation - contains descriptions, explanations, and additional information about filing the selected classes.
- Simulation - here the user can translate, simulate the model, analyze simulation results and see the animation. This section also includes a command window in which a user can communicate with the model, change parameters or run a batch of simulations.
- Tools - this section includes additional options for model management, analysis, debugging, and export functionalities.

1.3.3. Functional Mock-up Unit (FMU)

Functional Mock-up Interface (FMI) is an open standard for exchanging dynamical simulation models between different tools in a standardized format. Functional Mockup Unit (FMU) is a file that contains a simulation model, that adheres to the FMI standard and can be executed in external tools. When exporting the model, we can choose between two different kinds of FMUs that are supported by the FMI standard:

1. **Model Exchange (ME**) - ME FMUs provide the model equations but to simulate the system the importing tool needs to connect the FMU to a numerical solver.

2. **Co-Simulation (CS) -** CS FMUs include the model and its own numerical solver, allowing it to run independently while exchanging data with the importing tool.

2. Toolchain structure and workflow

This chapter describes the structure and workflow of the developed toolchain, outlines the different moduels in the parametrization process and the overall data processing logic. The main task of the toolchain is to automaticaly populate the simplified model for laptime simulation. The parametrization of the model is split over 5 modules:

1. static vehicle analysis,
2. acceleration and braking,
3. tyres,
4. suspension and
5. aerodynamics.

In the static vehicle anaysis the goal is to extract the fundamental vehicle properties required for the model definition. The acceleration and braking module focus on the longitudinal behaviour of the vehicle, providing the parameters required to represent traction and braking performance. The tyre module is responsible for generating the set of parameters defining the Magic Formula tyre model implemented in the simplified model. In the suspension parametrization the goal is to develop a method that allows to keep some complexities from the multibody model in the simpler reduced model. Lastly, in the aerodynamics analysis the goal was to replace the existing multi-inputs variable linear interpolation of aerodynamic maps with a more suitable approach for the simple model.

For each module a specific Python script is developed to extract and process the necessary data. Depending on the module, the starting point for the parametrisation can be parameters files, FMUs, that are ran in Python, or running experiments directly in Dymola and then processing simulation results in Python. Each script produces a specific set of outputs such as characteristic curves, maps, or quantities saved into a format suitable for the reduced order model. This enables an efficient transfer of vehicle information from complex to the simplified model while preserving key physical characteristics.

The scripts related to static vehicle analysis, acceleration and braking and tyre parametrization are integrated within a central Python script (referred to as "*Play*"), that enables the automated execution of the complete workflow. This central script significantly reduces the running time by allowing the parallel execution of multiple FMU simulations. Since both the static and acceleration analysis rely on FMU simulations, this parallelization is essential to accelerate the parameterization process. For the *Play* script to run correctly, all input files must be placed in the same working directory as the script. Once the execution is completed, the script generates a single Excel file containing the parameters from all the modules, ready to be used in the simplified simulation model.

The suspension parametrization relies on batch simulations performed directly in Dymola through Modelica scripts. The generated result files are later processed in Python, for this reason, this module is treated separately from the automated workflow. The aerodynamic module, which will be described in detail in the following sections, can be used within the simplified model but is for now kept as a standalone tool, allowing it to be executed independently of the main workflow.

A schematic representation of the overall toolchain and information flow is shown in figure 3. On the left side differnet input sources are highlighted, including parameters file and used FMUs. Parameters file is an XML file containing main vehicle characteristics such as mass, geometrical properties, engine map, aero map. FMUs, as already explained, represent the exports of dedicated expreiments of dymola models needed to extract information for each of the moduls. These are then processed through dedicated Python scripts. The diagram also distinguishes between modules integrated within the Play script and those implemented as standalone tools. All modules ultimately contribute to the generation of a set of parameters used to populate the simplified simulation model. These parameters are saved in an Excel file, but more information about the output of each module will be given in the following chapters.

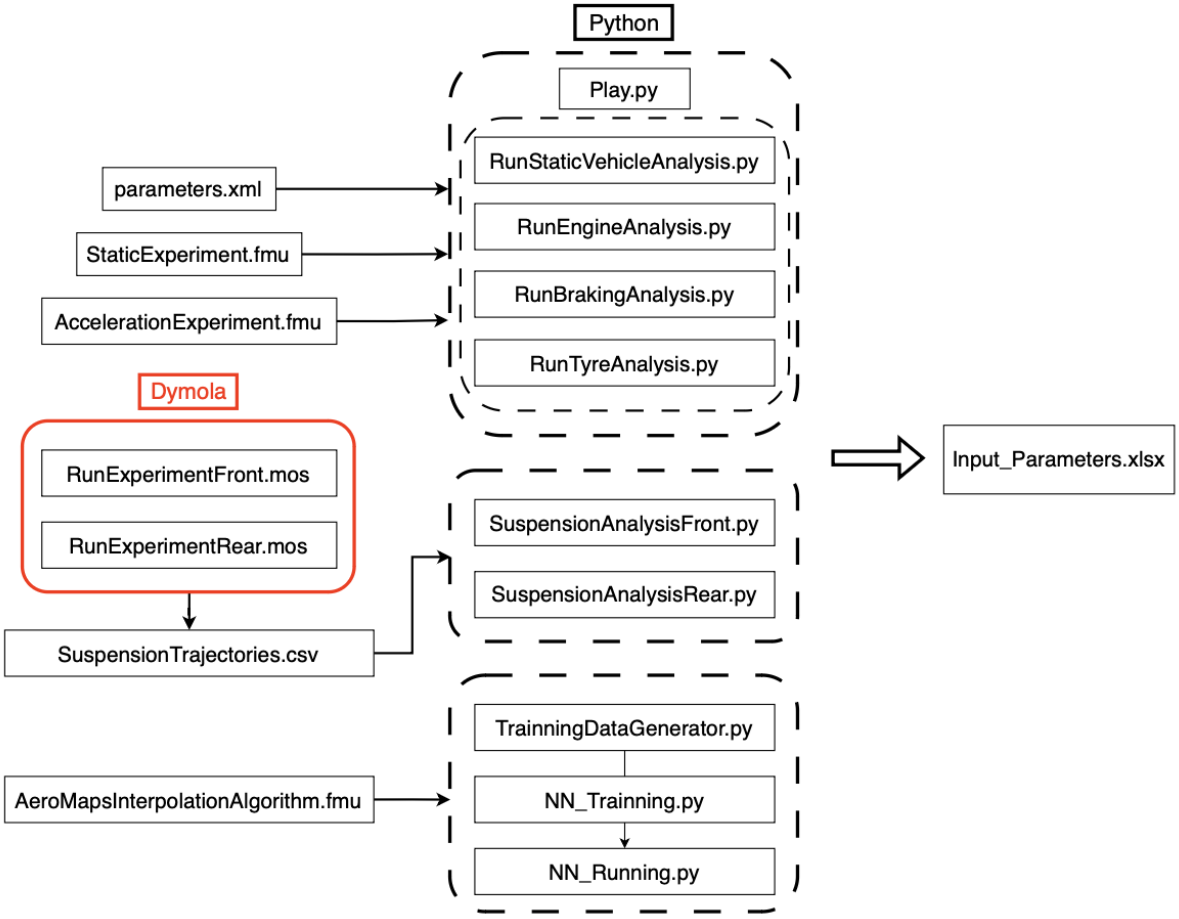

Figure 3. Toolchain workflow

The following chapters describe in detail the methodology, implementation, and outputs of each module introduced in this overview. Due to confidentiality constraints, the details of the developed scripts are not reported in this thesis. For the same reason, numerical values in figures and plots are omitted. The focus is therefore placed on the methodology and workflow rather than on specific data or code implementation.

3. Static Analysis

The most obvious thing to start with is the basic information about the vehicle that is needed to run the simulation. This includes general mass and geometric properties of the vehicle, which define its overall layout and mass distribution. Typical parameters in this group are the total vehicle mass, driver and ballast contributions, inertia, and key geometrical dimensions such as the wheelbase, track widths, and centre of gravity position.

The simplified model requires these parameters as direct inputs to define the baseline configuration of the car. They are used to define how the mass is distributed within the vehicle and to set the geometric reference for all subsequent calculations. Defining these parameters correctly makes it possible to represent the vehicle behaviour and solve the governing equations of motion. For this reason, these parameters form the foundation of the entire model, and their correct extraction and definition is a crucial first step in the overall toolchain.

The next step is to define the parameters that are needed as the input of the simple model. These parameters represent the output of the static vehicle analysis and are listed below.

- Unsprung Vehicle mass
- Sprung Vehicle mass
- Roll inertia
- Pitch inertia
- Yaw inertia
- Wheelbase
- Front track
- Rear track
- CoG position
- CoG height
- Front ride height
- Rear ride height
- Tyre rolling radius
- Tyre loaded radius

To access this data, two approaches were considered: extracting the information directly from the vehicle parameters file or by setting up an experiment using the Dymola vehicle model. The first option, based on reading the XML parameters file and processing them with a Python script, is less time-consuming and provides many of the required parameters. These files are also used by the Dymola models and contain a large number of characteristic vehicle properties, such as mass, inertia, and tyre rolling radius. However, parameters such as CoG height, ride heights and tyre compression need to be evaluated for the actual run condition of the car setup, so a dynamic simulation of the model is necessary to evaluate them. This approach is also more general, in the case of different vehicles with different ballasts layouts, that would otherwise require different analytical calculation for each case. Furthermore, parameters related to the vehicle geometry are not explicitly defined in the parameters file but are instead embedded within the multibody model geometry. To obtain these values, it is necessary to run a static experiment in Dymola.

To ensure consistency and maintain a clear data flow, it was decided to adopt a single approach and extract all required parameters from the Dymola model through a static simulation experiment. The static simulation experiment consists of simulating the vehicle in a stationary condition, resting on its wheels for a short period of time. Under these conditions, the model reaches equilibrium, and all relevant vehicle parameters can be accessed through the simulation outputs.

Some of the required parameters are directly available as outputs of the simulation, while others require additional processing. For example, quantities such as total mass and moments of inertia are straightforward, and their values can be obtained directly. However, geometric properties such as the wheelbase, front and rear track widths are not explicitly available and must be derived from the available outputs.

To determine these geometric parameters, the positions of the wheels and the center of gravity position are extracted from the simulation outputs. These values are then post-processed in a Python script, where simple calculations are performed to obtain the desired geometrical properties.

Once the required output parameters are identified, they must be defined at the top level of the experiment in Dymola. This is achieved by declaring them as Real outputs in the "*Text*" part of the Dymola model. During the FMU export process, it is then possible to either export all variables or selectively include only the defined outputs.

The final step is to export the model as a Functional Mock-up Unit (FMU) in the "*Simulation*" tab, using the co-simulation standard, which allows the model to be executed and accessed externally within the Python-based toolchain. The exported FMU serves as the input to the *RunStaticVehicleAnalysis* Python script. This script loads the FMU and, using the FMpy library, runs the simulation.

Once the simulation is completed, the predefined output variables can be accessed using the functions provided by the library. These outputs are then read and stored as variables within the Python environment. While some parameters are directly available, additional processing is required for the geometric properties of the vehicle. Wheelbase and track widths are computed from the wheel and CoG positions obtained from the simulation. These calculations take into account the definition of the coordinate system and the location of its origin in the vehicle model.

Once all the parameters are defined, they are saved together with their units into an Excel file, which serves as a structured and accessible input for the simplified simulation model.

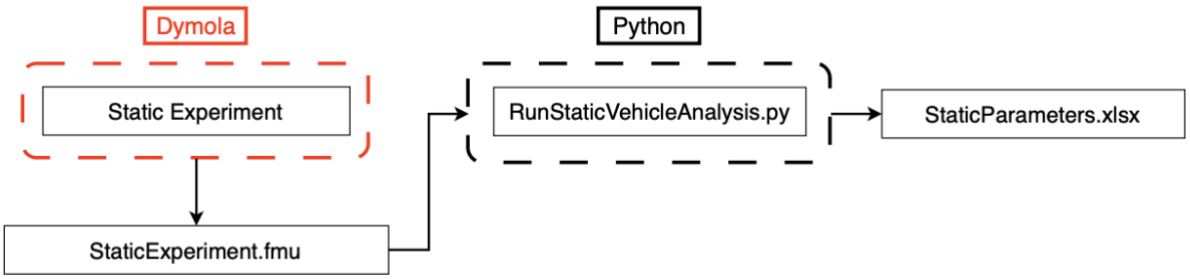

Figure 4. Static Analysis Workflow

The overall workflow of the static vehicle analysis module is illustrated in Figure 4. Starting from the Dymola static vehicle experiment, we define the needed outputs on the top level of the experiment. This experiment is then exported as an FMU. This FMU represents an input into the Python script. Python script runs the FMU and gives acces to all the outputs we defined. Based on these results, additional calculations are performed to obtain the required parameters in the form needed by the simplified model. Finally, all parameters are organised and saved into an Excel file. This file represents the final output of the module.

4. Acceleration and braking analysis

Next module deals with the longitudinal capabilities of the model, providing the parameters required to represent traction and braking of the vehicle. The work was split into acceleration and braking analysis. These were implemented as separate scripts, as different approaches were adopted for each case. Therefore, the two analyses will be presented and discussed independently in the following sections.

4.1. Acceleration analysis

In the simplified model, traction is defined through the torque applied at the wheels. Consequently, the objective of this module is to generate the wheel torque curve as a function of wheel rotational speed. Two approaches were implemented, and both are described below.

4.1.1. Engine Analysis

The first approach is based on the analytical processing of engine maps. It is based on the engine characteristic data available in the parameters file, which are processed inside the Python script and combined with drivetrain information to generate the required wheel torque curve.

Starting from the parameters file we can extract the engine map. Engine maps are a 2D look up tables that provide the available engine torque as a function of engine rotational speed and the throttle position. For this analysis the maximum engine torque is considered, so the engine torque curve for throttle position equal to one is extracted. Once we have the engine torque curve, we need to transform this to the torque at the wheel. This is done by considering the drivetrain gear ratios using the formula shown below. Gear ratios for all the available gears are also parameters that can be found in the vehicle parameters file.

$$\tau_w = \tau_e \cdot i_g \cdot i_d$$

Where:

τ_w – torque at the wheel,

τ_e – torque at the engine,

i_g – ratio of the gear,

i_d – ratio of the final drive.

Using this relation, we can get a torque curve at the wheel for each of the gears. This is shown on the graph below.

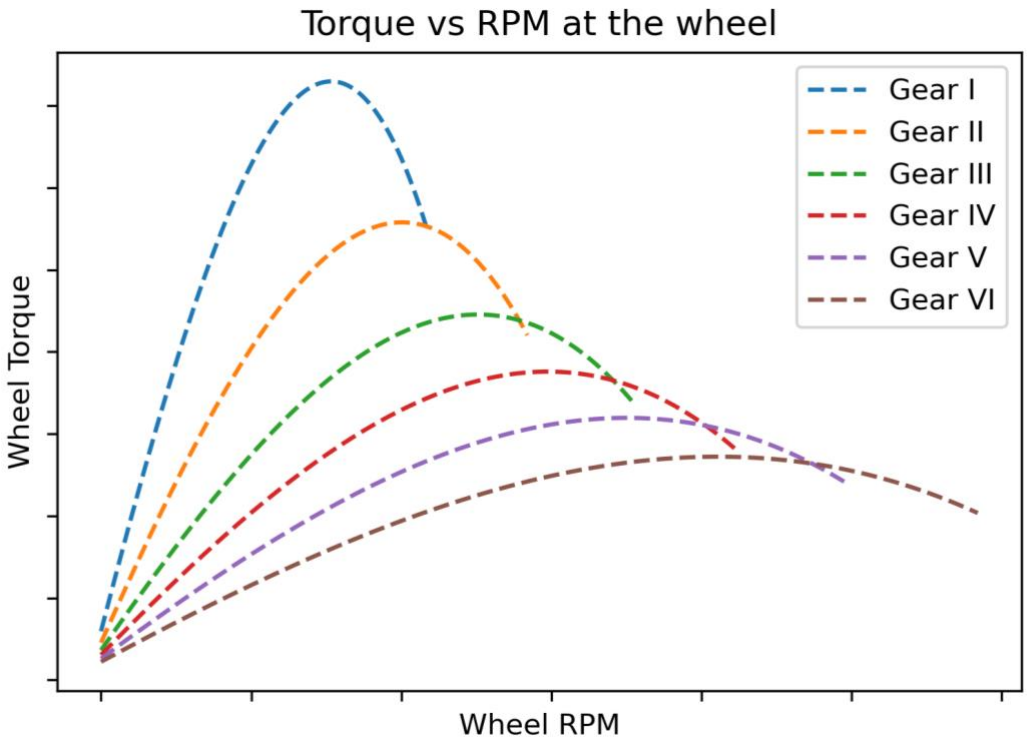

Figure 5. Torque at the wheel for every gear

The goal is to obtain a single, representative wheel torque curve. Since the simplified model doesn't include a driver model or a shifting strategy, shifting is assumed to be idea and torque delivered to the wheels is always maximum. The adopted strategy follows the torque curve of each gear until it intersects with that of the subsequent gear, at which point an upshift is performed. In this way, the operating point always corresponds to the maximum available wheel torque across the entire speed range.

To implement this approach, it is necessary to identify the intersection points between the wheel torque curves of consecutive gears. For each gear we have a torque curve as a function of wheel speed so for two consecutive gears we have:

$$f_i = \tau_{w,i}(\omega) \text{ and } f_{i+1} = \tau_{w,i+1}(\omega)$$

where *i* is the number of the gear.

The intersection point is where these two have the same value of torque. To determine this point, a function is defined as the difference between the two curves. At the intersection, this difference is equal to zero. Considering the absolute value of this difference, allows us to find the intersection point by finding the minimum of this function.

$$g = |f_i - f_{i+1}|$$

The wheel speed at which the minimum occurs corresponds to the intersection point between the two torque curves. This procedure is repeated for all consecutive gears, defining the shift points that ensure maximum torque is always delivered at the wheels.

A graphical representation of this procedure is shown below where the wheel torque curves of two consecutive gears are plotted. The intersection point between the curves defines the optimal shift point.

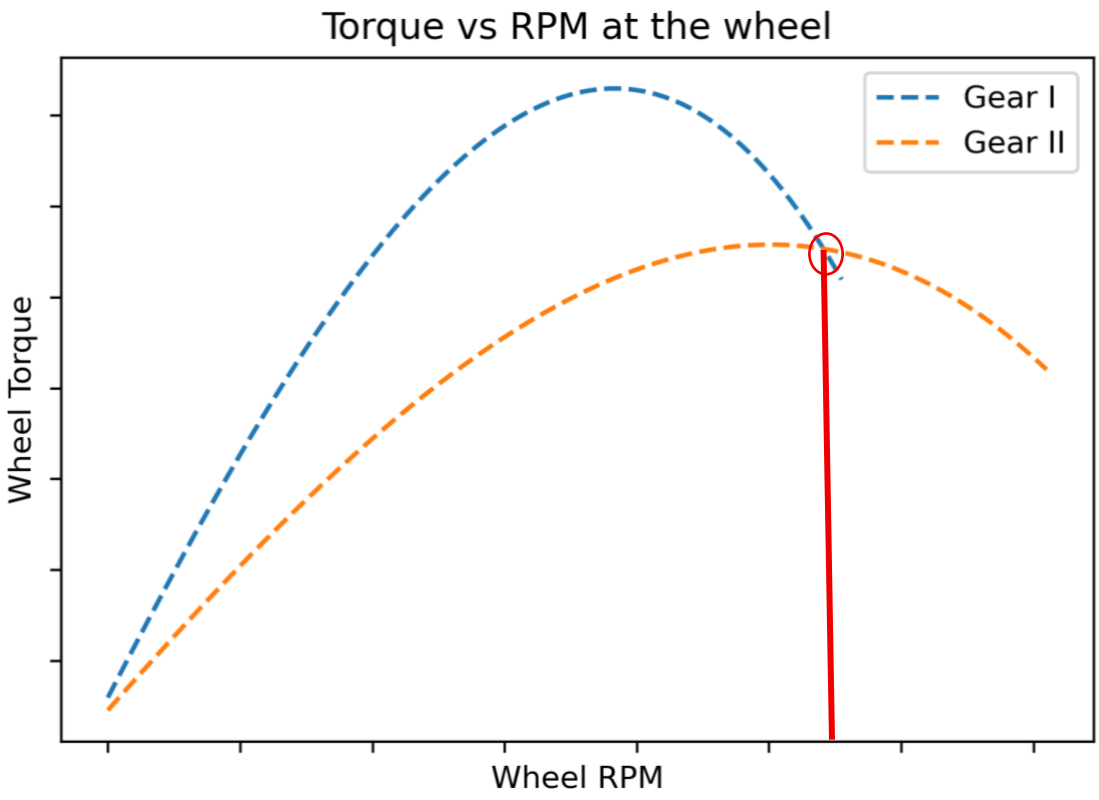

Figure 6. Defining the gear shift point – 1

Once the shifting points are determined, a single final wheel torque curve can be constructed. This is achieved by combining the torque curves of all gears using a masking approach implemented in Python. Since we know the working range for each gear, from the previous calculations, within this range, the torque values of that gear are selected, while values outside the range are excluded. In this way, each portion of the final curve corresponds to the gear that provides the highest available torque. By applying this selection process across all gears, a continuous curve of maximum wheel torque is obtained as a function of wheel speed. The resulting curve represents the maximum torque that can be delivered to the wheels and it is shown on the graph below.

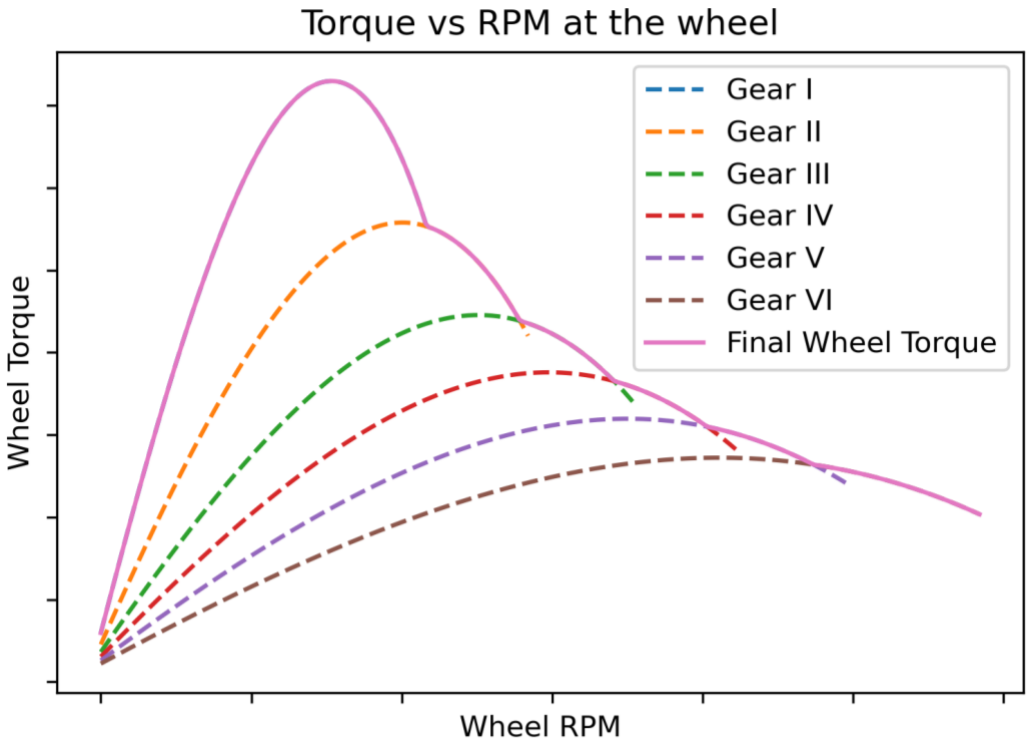

Figure 7. Final wheel torque curve

This final curve is then used as a direct input to the simplified model.

The overall workflow is shown in the figure below. Starting from the parameters file the exctracted data is processed in a Python script "*RunEngineAnlaysis.py*". In the end the final torque as a function of wheel speed is exported in an excel file.

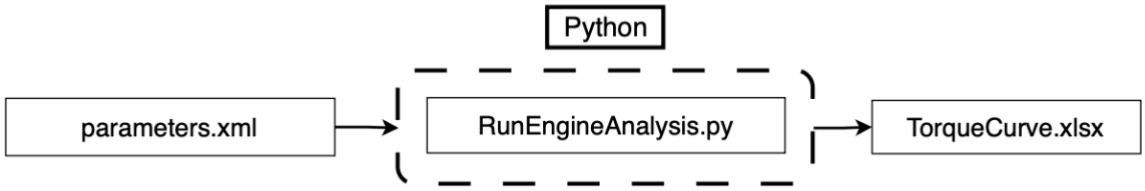

Figure 8. Acceleration Analysis Workflow - Method 1

This approach was initially adopted because the overall objective of the toolchain is to minimise the time required to populate the model. Using the data available in the parameters file and processing it in Python, the execution time is reduced to only a few seconds, while the approaches that rely on FMU simulations require significantly longer runtimes. This method also provides more predictable and easily repeatable results since there is no numerical solver involved. However, this approach is over simplified, and it is acceptable only for vehicles with very simple power units, that only have an internal combustion engine. In modern racing applications, power units are often significantly more complex, frequently incorporating hybrid systems, advanced shifting strategies as well as varying engine maps. Therefore a second approach was developed, based on Dymola models, which is described in the following section.

4.1.2. Acceleration Experiment

As previously mentioned, power units in modern race car can be very complex that's why a second approach was developed. This approach is based on a straight-line acceleration experiment set up in Dymola, which allows the wheel torque to be obtained directly from simulation outputs.

The straight-line acceleration experiment consists of a vehicle starting from a predefined speed and accelerating until it reaches the maximum speed. The required outputs for the Python script must first be defined at the top level of the experiment, after which the model can be exported as an FMU.

The exported FMU serves as an input into the Python script "*RunEngineAnlaysis.py*". After running the FMU, the required outputs are accessed within the Python environment. Unlike the analytical approach described previously, the resulting torque curve is not directly usable and requires additional processing.

The main limitation of this approach is that it relies on numerical simulation, so the output data can be affected by numerical noise, solver settings, and transient effects. This means that the raw torque signal is often not smooth and contains irregularities. For example,

we will have peaks in the torque curve each time there is an upshift. This requires some post-processing to construct a clean wheel torque curve. Because of this the Python script includes additional steps to filter the torque curve. For example, signals such as clutch engagement are used to recognize the shifting moments, allowing the associated torque peaks to be detected and removed.

This represents the main limitation of the method, as the filtering process may require adjustments for each vehicle. The script was developed to be as robust as possible and was tested on multiple vehicles available during the development process. Nevertheless, this approach is necessary as it is the only way to accurately capture the full behaviour of the complex powertrain.

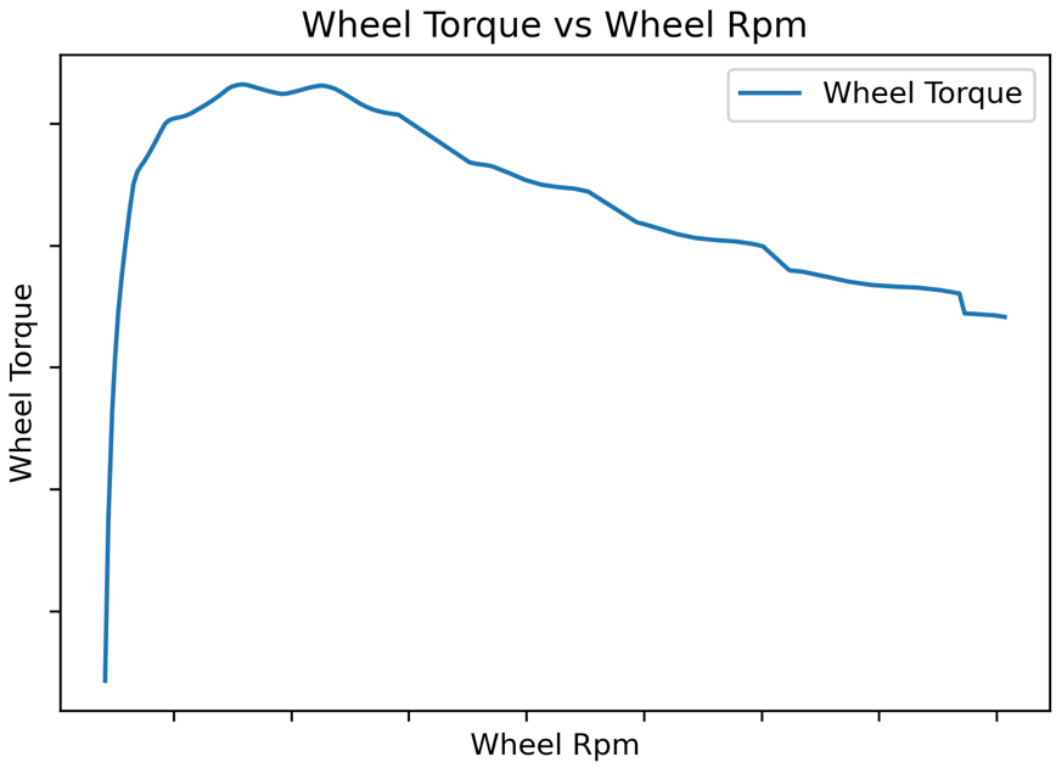

Figure 9. Wheel torque curve – experiment

The final wheel torque curve is shown in Figure 9.

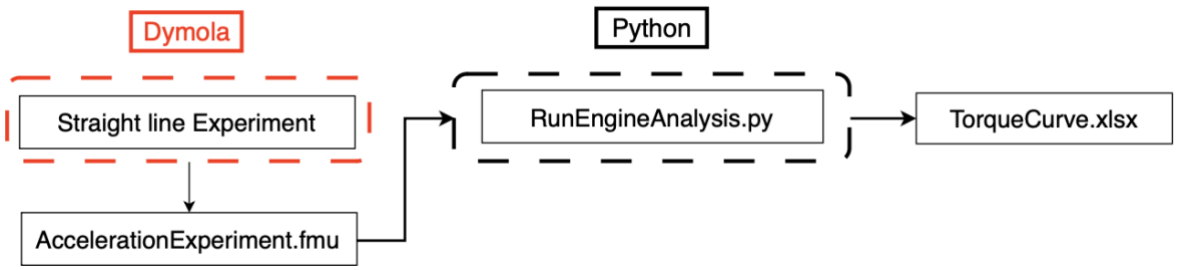

Figure 10. Acceleration Analysis Workflow - Method 2

The overall workflow is shown in the figure above. Starting from the Dymola straigh line acceleration vehicle experiment that is exported as a FMU. This FMU represents an input into the "*RunEngineAnlaysis.py*" Python script. The output is the final wheel torque as a function of wheel speed in an excel file.

4.2. Braking Analysis

The only input describing the braking capability of the simple model is the maximum braking torque. This value can easily be calculated with the information available in the vehicle parameters file. The necessary data is listed below.

- Maximum braking pressure
- Brake bias
- Piston surface
- Effective radius
- Friction coefficient

Once we have these parameters, we can calculate the braking torque at the front and the rear wheels using the formula below.

$$M_{B,F} = 2 \cdot \frac{BB}{100} \cdot p_{max} \cdot A_{piston,f} \cdot R_{e,f} \cdot \mu_f$$

$$M_{B,R} = 2 \cdot \frac{(1 - BB)}{100} \cdot p_{max} \cdot A_{piston,r} \cdot R_{e,r} \cdot \mu_r$$

Where:

$M_{B,F}$ – maximum braking torque at the front (both wheels),

$M_{B,R}$ – maximum braking torque at the rear (both wheels),

BB – brake bias,

p_{max} – maximum braking pressure,

A_{piston} – overall piston surface,

R_e – effective radius and

μ – friction coefficient.

Maximum braking torque is obtained by just summing these two contributions. The values of maximum pressure and brake bias are hard coded in the Python script as they represent an input coming from the driver in the DiL simulator so they are not present in the parameters file.

The described process is done in the "*RunBrakingAnalysis.py*" script, and the workflow is shown in the figure below.

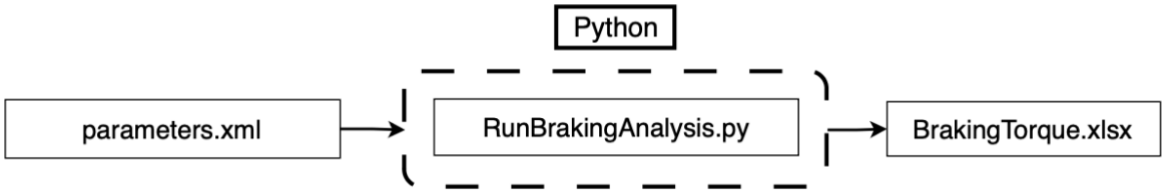

Figure 11. Braking Analysis Workflow

4.3. Conclusion

In summary, the objective of this module was to provide a reliable and efficient method for generating the wheel torque curve required by the simplified model. Two approaches were developed: an analytical method based on engine maps and drivetrain parameters, and a simulation based method using a straight-line acceleration experiment in Dymola. Additionaly a separate script was developed for the calculation of the maximum braking torque available. If the simulation based approach is adopted for acceleration analysis, future improvements could focus on further unifying the analysis procedures. One possible direction is to extend the acceleration experiment to include a braking phase, allowing both acceleration and braking characteristics to be extracted from a single simulation. This approach could be further investigated keeping in mind that inside the Dymola model there are also electronic systems like ABS that could affect the max braking torque in the simulation.

5. Tyre Analysis

The goal of this chapter is to define and parametrize the tyre model used in the simplified vehicle model and implement it inside the Python environment.

In the Dymola models, tyre behaviour is described using the Pacejka Magic Formula. The objective of this module is therefore:

1. to construct a Pacejka 5.2. tyre model in Python using the same parameters used by the Dymola models and
2. define a simplified Magic Formula by fitting a reduced set of parameters to the tyre force characteristics generated by the 5.2 model.

In the reduced order model so far, the implemented tyre model is based on the simple Pacejka formula. This is because these simplified models are often used in the early phases of the project when we want the model to have less dependencies, so using the simplified tyre model is suitable.

5.1. Theory background

Before explaining the developed script in Python, a theoretical background of the implemented tyre model is introduced, together with an overview of the optimization process used to obtain the tyre model parameters.

5.1.1. Simple Pacejka Magic Formula

Magic Formula is an empirical tyre model that describes the nonlinear relationship between the longitudinal and lateral tyre slip and the forces and moments between the tyre and the road. It is widely used to represent both longitudinal and lateral tyre behaviour under steady-state operating conditions. Three main cases can be distinguished:

- pure lateral slip (cornering),
- pure longitudinal slip or
- combined slip.

Under pure slip conditions, the longitudinal force F_x as a function of longitudinal slip κ, and the lateral force F_y as a function of slip angle α, show a similar characteristic shape. This behaviour is described by the Magic Formula:

$$Y(x) = f(F_z, x) = F_z D \sin(C \operatorname{atan}((Bx - E(Bx - \operatorname{atan}(Bx)))))$$

where:

$Y(x)$ – either F_x or F_y.

Inputs into the magic formula are:

- normal wheel load (F_z),
- inclination angle (γ),
- longitudinal slip (κ) or
- lateral slip (α).

The output of the model is therefore the generated tyre force depending on the considered case.

The shape of the tyre characteristic curve is defined by four main parameters of the Magic Formula:

- B (stiffness factor) – stretches the curve,
- C (shape factor) - determines the part used of the sine, influencing the shape of the curve,
- D (peak factor) – determines the peak of the characteristic and
- E (curvature factor) – modifies the characteristic shape around the peak.

The combined effect of these parameters defines the characteristic shape of the tyre force curve, as shown in Figure 12.

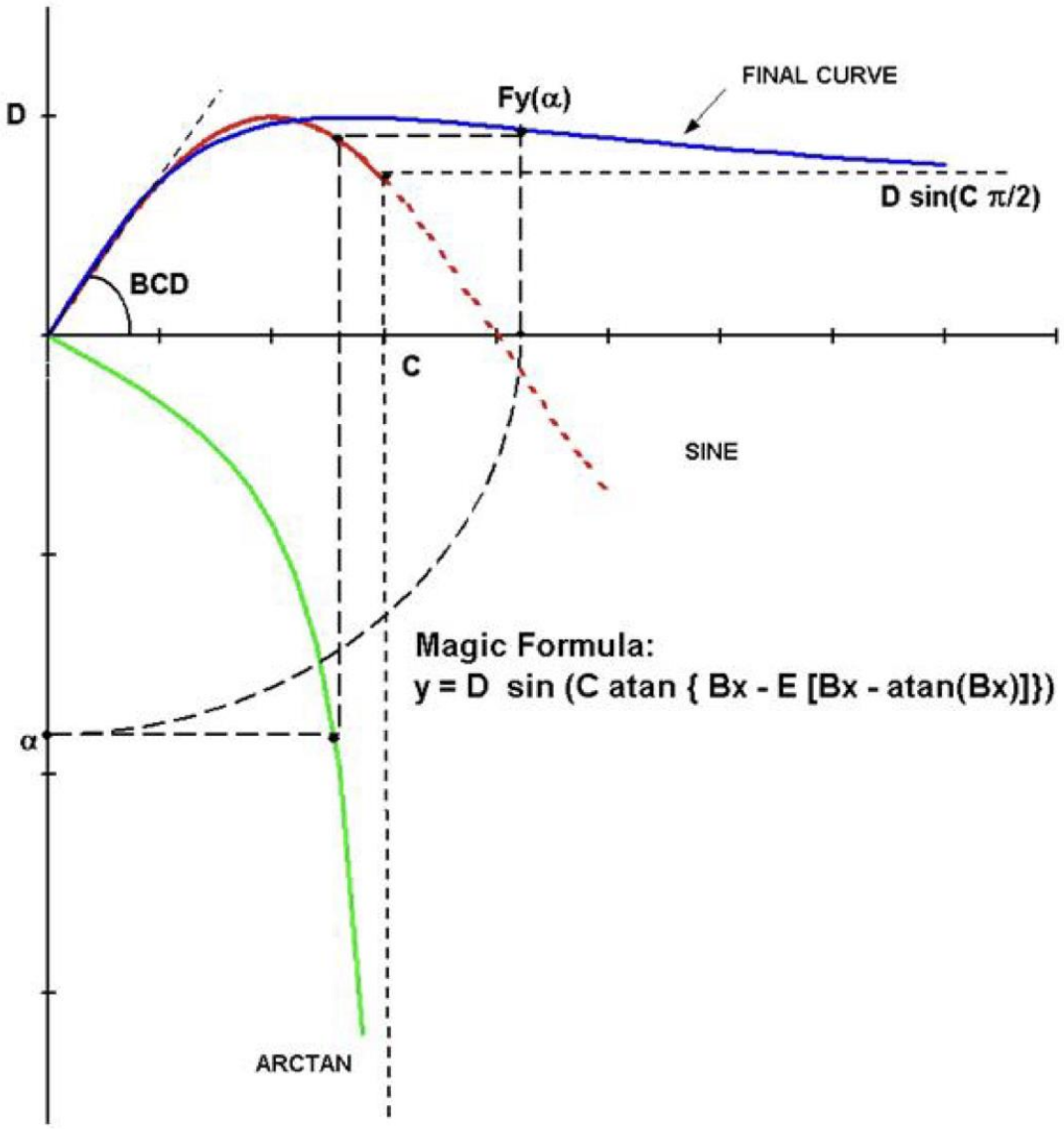

Figure 12. Magic Formula

The figure shows that the Magic Formula captures the typical tyre behaviour: an initial linear region where the force increases proportionally with slip, followed by a nonlinear region leading to a peak force, and finally a saturation region where the force slightly decreases.

As it was already mentioned the simplified Magic Formula parameters are obtained through an optimization process, in which a reduced set of parameters is fitted to the tyre force characteristics generated by the Pacejka 5.2 model. The Pacejka 5.2 model implemented in this work follows the formulation presented in the literature [3]. Due to the large number of equations and parameters, the full mathematical description is not reported here but can be found in the referenced source.

5.1.2. Optimization theory

Once we have defined the 5.2. Pacejka model we want to determine the four parameters, B, C, D and E of the simple Pacejka Magic Formula. This is achieved through a nonlinear optimization process, in which the simplified tyre model is fitted to the reference force curves generated by the Pacejka 5.2 model.

Nonlinear programing optimization is a field of mathematical optimization where the objective function or any of the constraints are nonlinear. The goal of the optimization is to minimize the difference between the forces predicted by the simplified Magic Formula and those obtained from the Pacejka 5.2 model. This difference is quantified through a cost function, defined as the sum of squared differences between the two models. The use of the sum of squared errors ensures that larger deviations between the models are penalized more. From a geometric point of view, this minimization corresponds to finding the lowest point of a parabola shaped cost function, where the optimal solution represents the best fit between the two models.

$$J = \sum (F_{5.2.} - F_{sp})^2$$

Minimizing this cost function allows finding the set of parameters that best reproduce the reference tyre behaviour.

5.2. Tyre analysis workflow

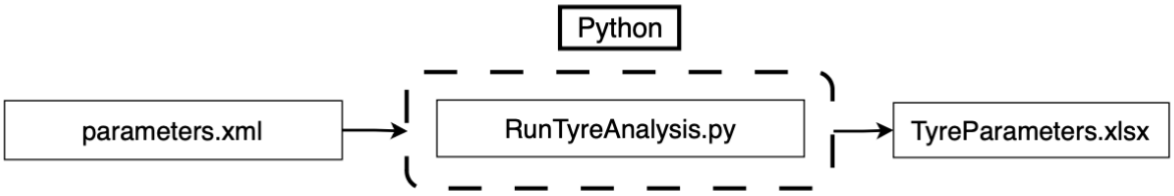

Figure 13. Tyre analysis workflow

The Tyre analysis script starts by building the 5.2. Pacejka tyre model within the Python environment. This model is parameterized using a set of coefficients that describe the tyre characteristics as functions of operating conditions. The parameters required for this parametrization are the same ones used in the tyre model implemented in the Dymola environment and can therefore be directly accessed from the parameters file, ensuring consistency between the DiL and simplified model.

Another input into the Magic Formula is the vertical load, F_z. In the current version of the script, the vertical load is hard coded, and it is calculated from the mass of the vehicle. This is adopted because the script is designed to also operate as a standalone tool, without direct coupling to a full vehicle model.

Following the formulation shown in [3] the Pacejka 5.2. model provides expressions for:

- pure longitudinal slip behaviour $F_x(\kappa)$
- pure lateral slip behaviour $F_y(\alpha)$ and
- combined slip conditions.

Under pure slip conditions, the longitudinal and lateral forces are computed independently as functions of slip. In combined slip conditions, the lateral force Fy will decrease due to longitudinal slip or the opposite, the longitudinal force Fx will decrease due to lateral slip. The forces and moments in combined slip conditions are based on the pure slip

characteristics multiplied by the weighing functions. If the coefficients needed for the weighing functions are not available then the computation is based on the friction ellipse to estimate the combined slip forces and moments. In our case those coefficients are available in the vehicel parameters file so the first approach is used in building the model for combined slip.

Once the tyre model is fully defined, including all required coefficents, vertical load F_z and inclination angle γ, the tyre characteristics can be generated by performing a sweep over the predefine range of slip values.

Two separate sweeps are carried out:

- a sweep over the longitudinal slip κ to obtain the longitudinal force characteristic $F_x(\kappa)$ and
- a sweep over the slip angle α to obtain the lateral force characteristic $F_y(\alpha)$.

The obtained curves are the tyre characteristics under pure slip conditions that are used as the reference data for the optimization process, in which the parameters of the simplified Magic Formula are identified.

The next step in the script is to define the simple Pacejka model, which is then used to formulate the cost function of the optimization problem. The optimization process is performed separately for both front and rear, for pure longitudinal and pure lateral slip conditions.

The optimization problem is defined using the CasADi library in Python. CasADi is a symbolic open source tool for nonlinear optimization and algorithmic differentiation. The optimizer varies the parameters B, C, D, and E in order to minimize the cost function.

In the end a separate set of simplified Magic Formula parameters is obtained for each tyre characteristic considered and saved in an Excel file, providing the reduced order model with a description of tyre behaviour.

The graphs below show the tyre characteristic obtained with the Pacejka 5.2. model and the fitted simple Pacejka model.

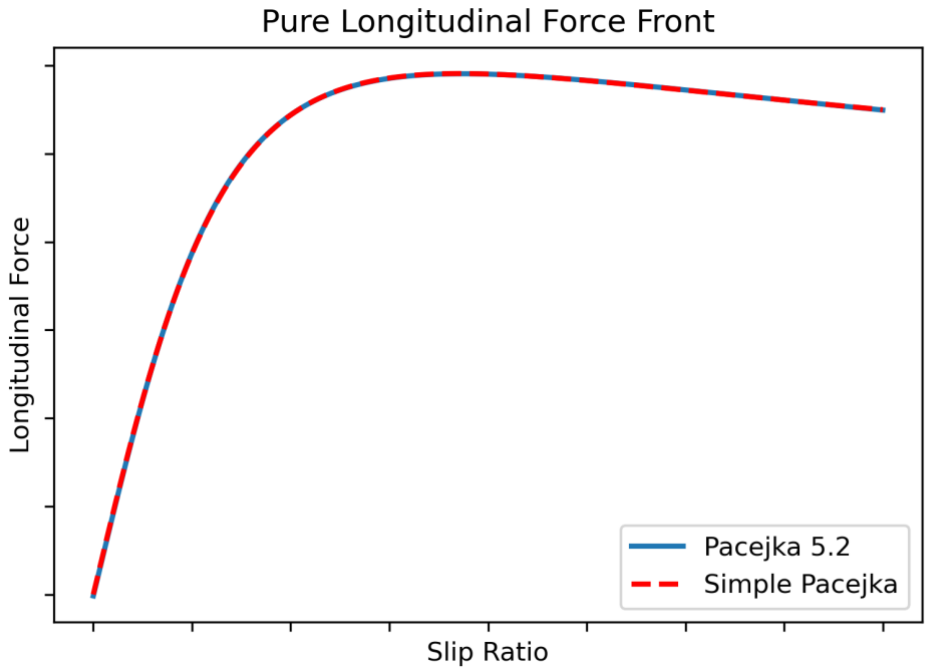

Figure 14. Longitudinal tyre force front

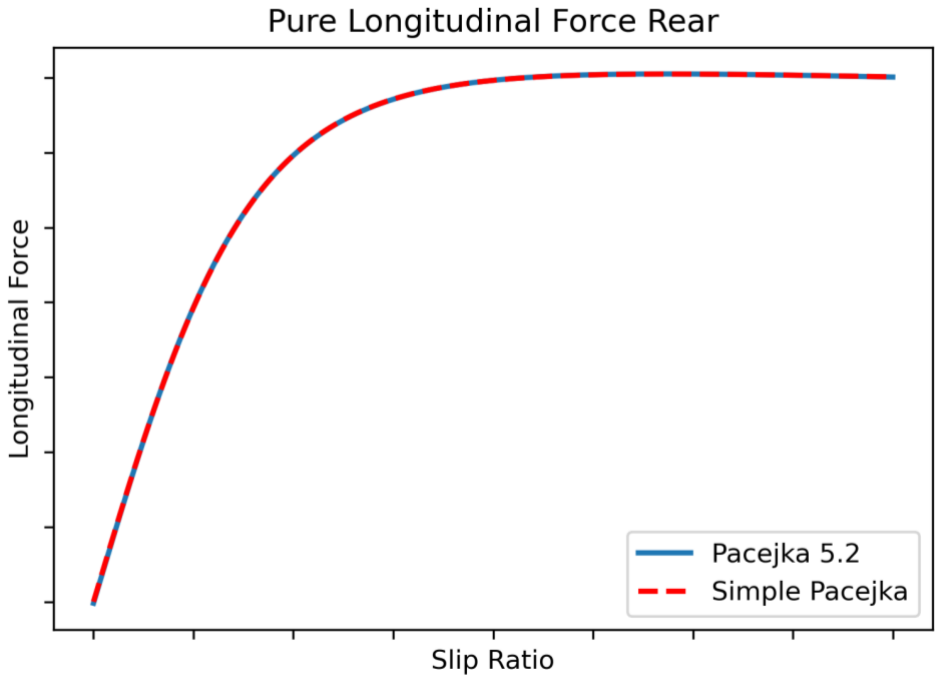

Figure 15. Longitudinal tyre force rear

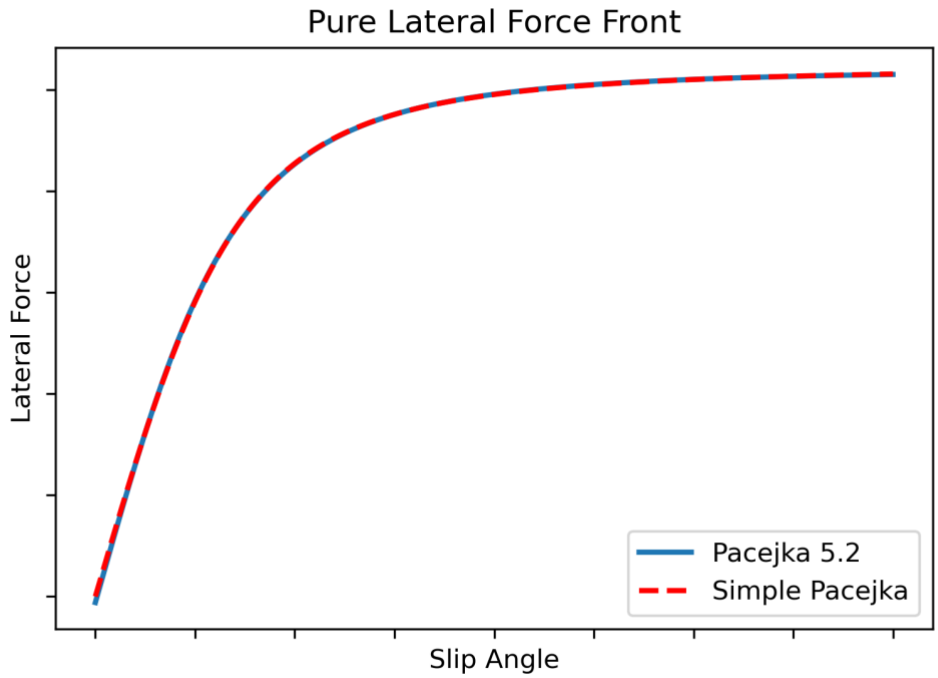

Figure 16. Lateral tyre force front

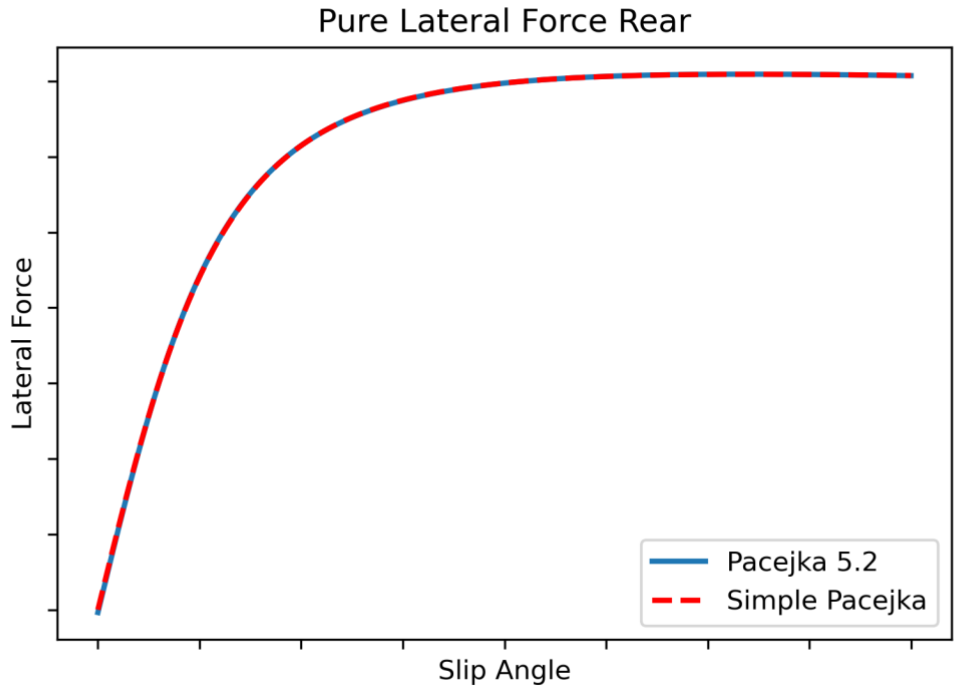

Figure 17. Lateral tyre force rear

6. Suspension Analysis

In the suspension analysis the goal is to find a method to introduce the nonlinear behaviour of the suspension system in the simplified model. The idea, consistent with the overall toolchain approach, is to use the Dymola multibody models and the existing suspension experiments to transfer a level of complexity in the simpler model. This is achieved by extracting the motion ratios (MR) of the elastic elements using the principle of virtual work.

6.1. Definition of MR applying the virtual work principle

Starting by defining the motion ratio (MR). The motion ratio describes the relationship between the wheel displacement and the deformation of a given elastic element in the suspension system. This relationship is described by the formula below:

$$MR = \frac{\mathrm{d}\delta_{wheel}}{\mathrm{d}\delta_{elemnt}}$$

Motion ratio is typically nonlinear and varies as a function of wheel position, for the front suspension elements it can also be a function of the steering angle. This is also the reason why it is convenient to use the multibody models to extract this relationship and get the realistic behaviour of suspension.

To compute the motion ratio the principle of virtual work is applied. Principle of virtual work states that if a system is in equilibrium, the total virtual work done by all external forces during an infinitesimal displacement (virtual displacement) is zero. This principle can be generalized to include the rotations, stating that the virtual work of the applied forces and applied moments is zero for all virtual displacements of the system from static equilibrium. This principle is applied to define the formulas used to calculate the motion ratios of all elastic elements for both the front and rear suspension of the vehicle under examination, as described in the following section.

Corner elements

Starting from the corner elements. Let's first consider the corner spring and damper, which ideally have a displacement along their axis. We can write the virtual work balance applied at the wheel by considering an infinitesimal virtual displacement at the wheel and the corresponding deformation of an elastic element. Here the relation is given for a corner spring.

$$F_s \cdot \delta_s = F_w \cdot \delta_w$$

Where:

F_s – force at the spring,

F_w – force at the wheel,

δ_s – virtual displacement of the spring and

δ_w – virtual displacement of the wheel.

We know that the force at the spring can be written as:

$$F_s = K \cdot s$$

where:

K – stiffness and

s – spring displacement.

Introducing that into the virtual work balance we can obtain the following expression:

$$F_w = K \cdot s \cdot \frac{\delta_s}{\delta_w} = K \cdot s \cdot \frac{1}{MR}.$$

From this we can calculate the MR from the Dymola models with the following expression:

$$MR = \frac{der(z_w)}{der(s)}$$

where:

z_w – vertical wheel travel.

Next, we can do the same for the corner torsion spring. In this case the virtual displacement of the elastic element is the angular displacement. And the expression can be written as follows:

$$\tau \cdot \delta_\theta = F_w \cdot \delta_w$$

Where:

τ – moment at the torsion spring and

δ_θ – virtual angular displacement of the torsion spring.

We know that the moment at the torsion spring can be written as:

$$\tau = K \cdot \Delta_\theta$$

where:

Δ_θ – torsion spring angular displacement.

Combining these formulas, we get that the MR is obtained from the following expression:

$$MR = \frac{der(z_w)}{der(\theta)}$$

where:

θ – rocker angle.

The MR of corner elements can be extracted from a one wheel bump experiment. Meaning we move one wheel from full rebound to full bump at zero steer. The calculated MR is a function of just the vertical wheel travel.

Heave elements

Moving on to heave elements. This includes the heave spring and damper. The heave elements are linked elements, they couple both side of the vehicle, so they depend on the travel of both wheels. Therefore, the virtual work principle applied at the wheel was defined as:

$$F_s \cdot \delta_s = 2 \cdot F_w \cdot \delta_{w,h}$$

where:

$\delta_{w,h}$ - virtual displacement of the average wheel travel.

Following the same procedure as before we can obtain the expression for the MR of heave elements.

$$MR = \frac{2 \cdot der(z_{w,h})}{der(s)}$$

where:

$z_{w,h}$ – average vertical wheel travel.

MR of heave elements can be obtained from a parallel wheel travel experiment, in which both wheels move from full rebound to full bump. In this case, the MR is not only a function of average vertical wheel travel but can also depend on the steering angle if the pushrod is connected to the upright, since its position changes with steering. In this case the MR is a function of both average vertical wheel travel and steering angle, and the experiment must be carried out over the range of both variables.

Anti-Roll Bar

Lastly, the same procedure is applied for the anti-roll bar. The anti-roll bar is an element that links both sides of the vehicle, so its behaviour depends on the travel of both wheels. However, if that wheel travel is equal the anti-roll bar is not active, so it depends on the difference in wheel travel. The virtual work principle is defined in terms of the suspension roll angle and the vehicle roll moment. Therefore, the virtual work principle applied is defined as:

$$\tau \cdot \delta_{\theta,arb} = T \cdot \delta_{\theta,car}$$

where:

$\delta_{\theta,arb}$ – virtual angular displacement of the anti-roll bar,

T – roll moment of the vehicle and

$\delta_{\theta,car}$ – virtual suspension roll.

Substituting inside that expression the formulas for the moment at the anti-roll bar and the vehicle roll moment we get:

$$K \cdot \Delta_{\theta,arb} \cdot \delta_{\theta,arb} = \Delta F_z \cdot t \cdot \delta_{\theta,car}$$

where:

$\Delta_{\theta,arb}$ – anti-roll bar angular displacement,

ΔF_z – vertical load transfer and

t – track width.

Following the same procedure as before we can obtain the expression for the MR of the anti-roll bar.

$$MR = \frac{t \cdot der(\theta_{car})}{der(\theta_{arb})}$$

where:

θ_{car} – suspension roll angle and

θ_{arb} – angle of the anti-roll bar.

The roll angle of the suspension is defined in Dymola using the center of the wheel position in respect to the chassis position that is fixed.

$$\theta_{car} = \frac{z_L - z_R}{y_L - y_R}$$

where:

z_L, z_R – positions of the left and right wheels, along the z-axis, measured with respect to the chassis fixed frame and

y_L, y_R – positions of the left and right wheels, along the y-axis, measured with respect to the chassis fixed frame.

The motion ratio of the anti-roll bar can be obtained from an opposite wheel travel experiment, in which the two wheels move in opposite directions from full rebound to full bump. In this case, the MR depends on the difference in vertical wheel travel. Additionally, it can depend on the steering angle if the pushrod is connected to the upright, and on the average wheel travel, since the position of the anti-roll bar varies with the overall heave level. Therefore, the MR is a function of wheel travel difference, average wheel travel, and steering angle, and the experiment must be carried out over the range of all these variables.

It should be noted, the motion ratio depends on the steering angle only in the front suspension.

6.2. MR definition in Dymola

As previously stated, the MRs will be taken as outputs from the Dymola suspension experiments. In the previous section, the formulas for MR calculation were defined together with the required parameters. It is important to note that, within the Dymola environment, the obtained motion ratios are computed using time derivatives of the relevant quantities. Taking the corner spring as an example, the motion ratio is computed as follows:

$$MR = \frac{\frac{d}{dt}(z_w)}{\frac{d}{dt}(s)}$$

Next step is to define the needed experiments in Dymola and define those MR expressions at the top experiment level.

Starting with the front suspension, an experiment is defined to account for the dependency of the motion ratios on the steering angle. The experiment is set up by moving one wheel from full rebound to full bump using a smooth ramp input, while the steering angle and the position of the opposite wheel are kept constant during each simulation. The steering angle and the opposite wheel position are treated as discrete inputs, taking values over a predefined range with a predefined step. This process is implemented in Dymola through a Modelica script. The script initializes the model, performs the batch simulations using for loops over the discrete variables, and stores the motion ratios, previously defined at the experiment level, together with the corresponding steering angle and wheel positions in a CSV file. These variables are required since the motion ratios are functions of both steering angle and vertical wheel travel.

For the rear suspension the same principle is applied, but now there is no dependency on the steering angle. Therefore, the experiment is setup by moving one wheel from full rebound to full bump using a smooth ramp input, while the position of the opposite wheel is kept constant during each simulation. This constant value is an input in each run, defined over a predefined range (full rebound to full bump) with a predefined step. This is again performed

through a batch of simulations using a Modelica script, with all relevant outputs stored for further processing.

This experiment can also be used for the corner elements. In the data processing in Python, we can select only the MR values for one discreet wheel position and one steering angle position. In this way the MR is only a function of the one wheel vertical travel.

For the heave elements, due to MR nonlinearities when there is roll, this approach led to significant data scatter. To deal with this, two approaches can be adopted. The first is to use the same experiment, but during data processing in Python, select only the data points corresponding to parallel wheel travel. Or an alternative approach is to define a separate experiment in Dymola with parallel wheel travel to extract the heave MRs.

6.3. Motion Ratio Fitting

Once we have the CSV files with all the MRs and the corresponding input variables, vertical wheel travel and steering angle, we can do the fitting of the MR. This is done in a dedicated Python script. For each fit, the coefficients of the polynomial or surface are computed and stored as outputs in an Excel file.

Starting with the corner elements, when the MR depends on a single variable, vertical wheel travel of the respective wheel, the data is approximated with a simple third order polynomial as shown below. This allows the MR to be expressed as a smooth function of wheel displacement, ensuring easy implementation in the simplified model.

$$MR(z) = a_0 + a_1 \cdot z + a_2 \cdot z^2 + a_3 \cdot z^3$$

Where:

z – vertical wheel travel and

a_0, a_1, a_2, a_3 - polynomial coefficients obtained from the fitting process.

For more complex elements, such as the heave element in the front suspension, where the MR depends on the two variables, average wheel travel and steering angle, a third order polynomial surface approach is used, as shown below.

$$MR(z_{w,h}, \delta) = a + b \cdot z_{w,h} + c \cdot \delta + d_1 \cdot {z_{w,h}}^2 + d_2 \cdot z_{w,h} \cdot \delta + d_3 \cdot \delta^2 + e_1 \cdot {z_{w,h}}^3 + e_2 \cdot {z_{w,h}}^2 \cdot \delta + e_3 \cdot z_{w,h} \cdot \delta^2 + e_4 \cdot \delta^3$$

Where:

$z_{w,h}$ – average vertical wheel travel and

δ – steering angle.

Lastly for elements such as the front anti-roll bar, where the MR depends on three variables, difference between the wheel travel, the average wheel travel and steering angle, a second order multi-dimensional polynomial surface approach is used, as shown below.

$$MR(z_{w,ab}, z_{w,h}, \delta) = a + b_1 \cdot z_{w,arb} + b_2 \cdot z_{w,h} + b_3 \cdot \delta + c_{11} \cdot {z_{w,arb}}^2 + c_{12} \cdot z_{w,arb} \cdot z_{w,h} + c_{13} \cdot z_{w,arb} \cdot \delta + c_{22} \cdot {z_{w,h}}^2 + c_{23} \cdot z_{w,h} \cdot \delta + c_{33} \cdot \delta^2$$

Where:

$z_{w,arb}$ – difference between wheel travel.

The following figures show the motion ratio data points together with the corresponding polynomial and surface fits for the different suspension elements.

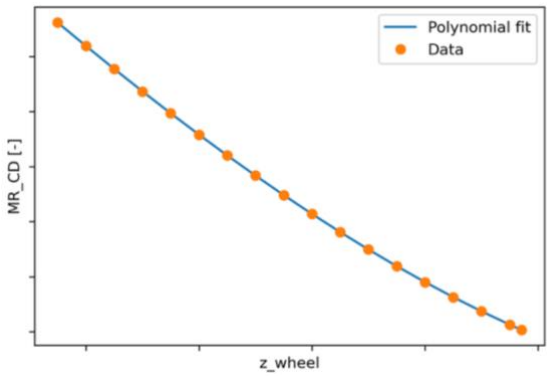

Figure 18. Front corner damper MR

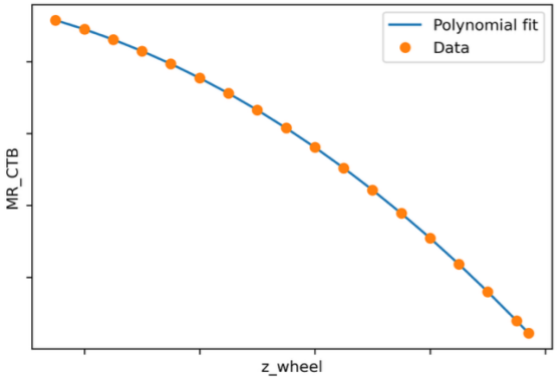

Figure 19. Front corner torsion bar MR

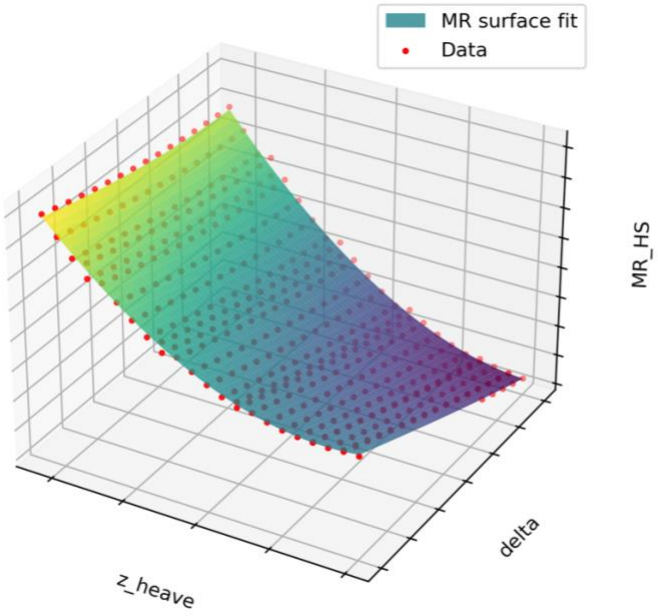

Figure 20. Front heave spring MR

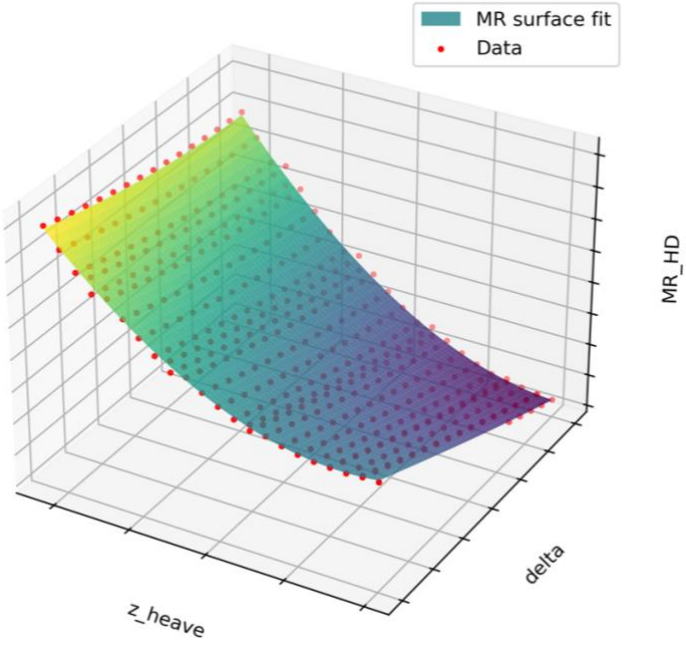

Figure 21. Front heave damper MR

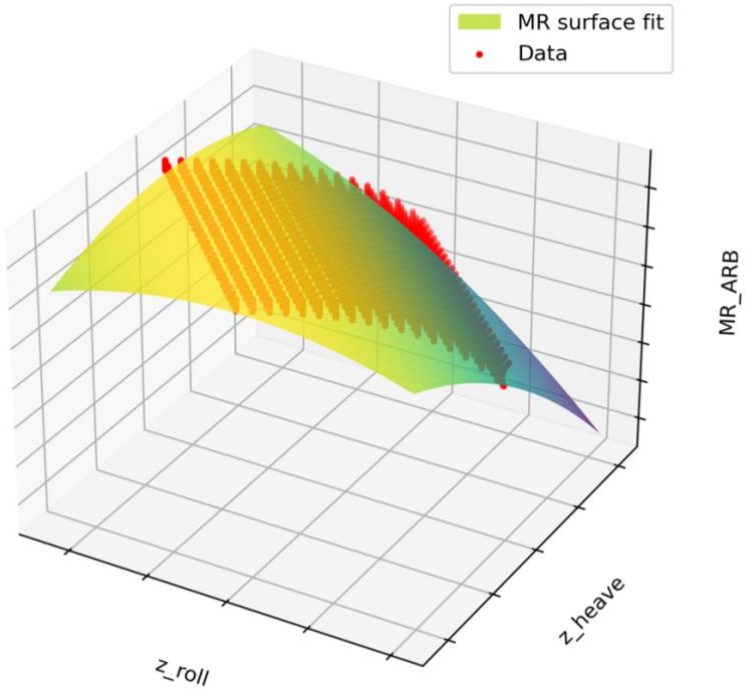

Figure 22. Front anti-roll bar MR

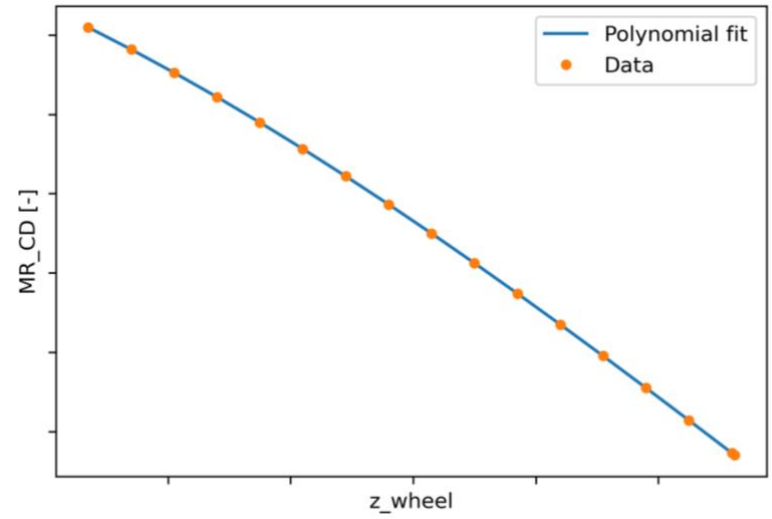

Figure 23. Rear corner damper MR

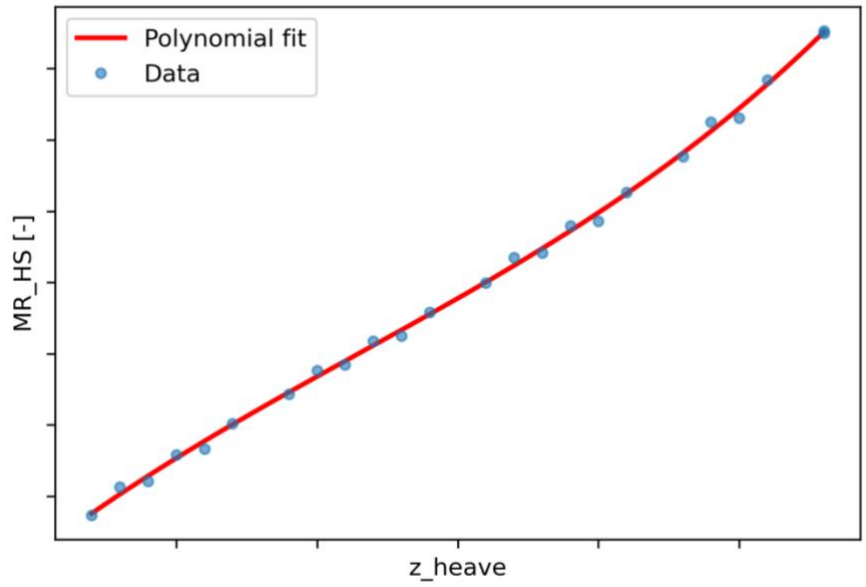

Figure 24. Rear heave spring MR

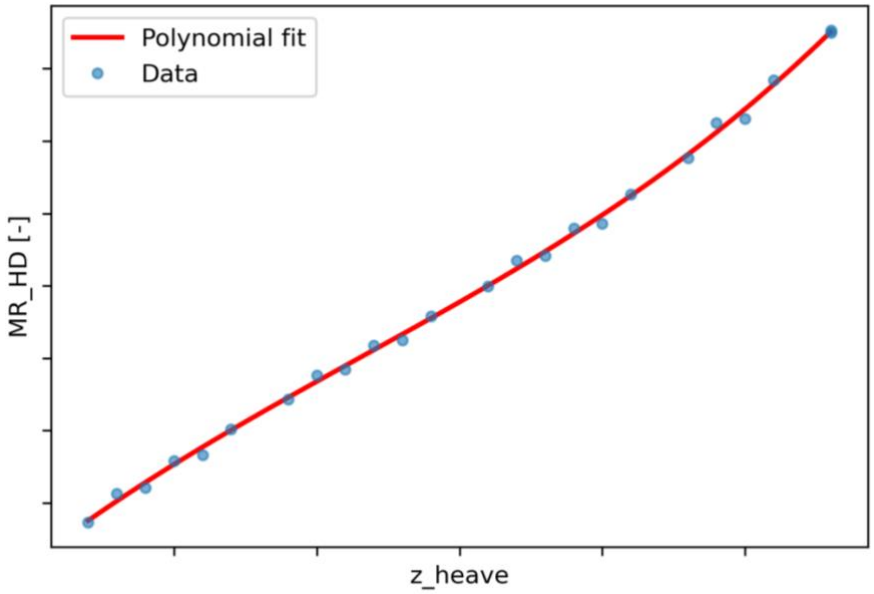

Figure 25. Rear heave damper MR

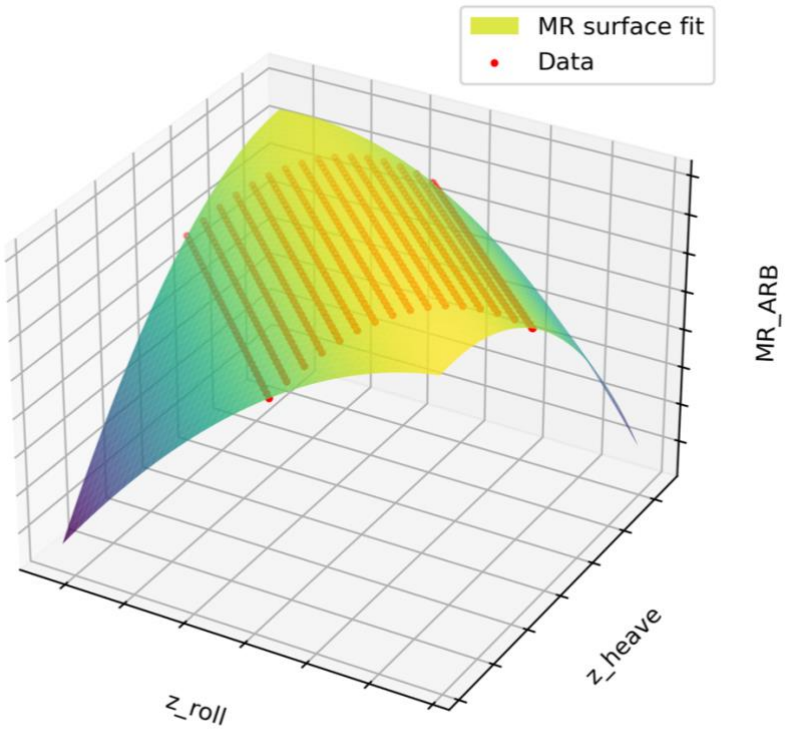

Figure 26. Rear anti-roll bar MR

6.4. Conclusion

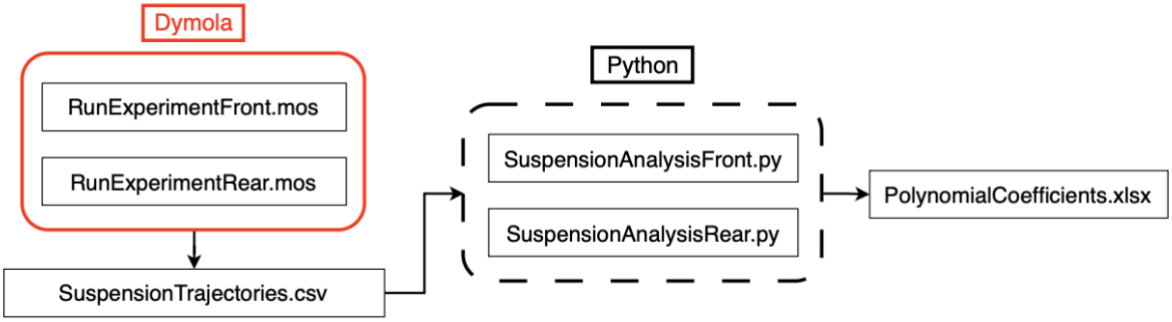

Figure 27. Suspension analysis workflow

The suspension analysis workflow within the toolchain starts by defining the MR expressions in Dymola experiments and running the batch simulations using the Modelica scripts. The resulting MRs for all the suspension trajectories are then used as an input to the Python post processing script, where they are fitted with the polynomial as functions of their respective variables. The output of the Python script is an excel file with all the polynomial coefficients saved.

One remark regarding the possible future implementation. If the suspension analysis would be integrated as a part of the central script of the toolchain, "*Play*", then the Dymola experiment could be exported as an FMU and run directly from Python, accessing the outputs and doing the post processing all in the Python environment.

7. Aerodynamic analysis

Last module that was covered in this work is the aerodynamics. Aerodynamics plays a fundamental role in the performance of race and high-performance vehicles, so it must be represented in even the simplified simulation models. The aerodynamic behaviour is described with aero maps, and an interpolation algorithm is used to evaluate the map through the simulation. The goal of this work was to replace the existing multi-inputs variable linear interpolation of aerodynamic maps with a neural network approach. The current interpolation algorithm is computationally inefficient, even for complex models, so the aim is not necessarily to achieve identical results, but to develop a model that provides an accurate representation of aerodynamic behavior while improving computational efficiency.

Since this module is centered around neural network training, it has been developed as a standalone tool. Before describing the implementation, a brief theoretical background on the neural network approach is provided.

7.1. Theory background

Machine learning is the process of finding a model from the data. Once we have found the model using the so-called training data the model can be applied to actual field data. This general idea of machine learning is shown in the figure below. The vertical flow represents the learning process, and the horizontal flow represents the using of the obtained model.

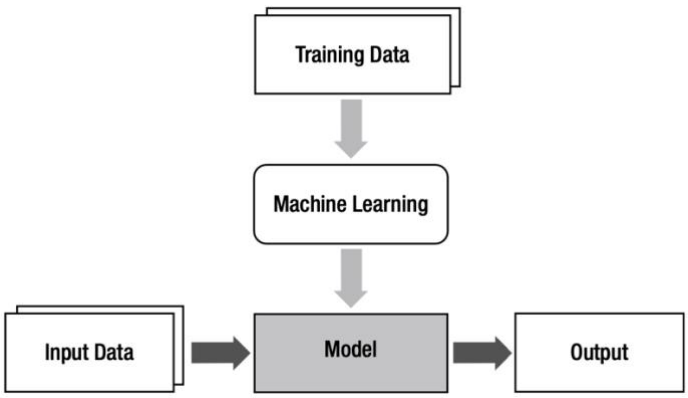

Figure 28. Machine learning workflow

An important concept in the model training is the generalization or making the performance of the network consistent regardless of the data set. One of the main problems of generalization is the overfitting of the model, which means that the model performs good on the training data set but not on other data sets. To determine whether the trained model is overfitted or not we use validation. Validation is a process that reserves a part of the training data and uses it to monitor the performance. To do so, the validation set is not used for the training process but later to compare the performance of the model to the expected outputs. Here we use supervised learning, meaning that our training data set consists of the input and the correct output that the model is supposed to produce.

One of the widely used models is the neural network. The neural network is built of nodes that receive inputs, that are weighted, and give an output. The information of the trained neural network is saved in the form of weights and biases. At each node the weighted sum is calculated so the input with the greater weight has a bigger effect. The equation of the weighted sum in the matrices form is written below:

$$v = wx + b$$

where x are the inputs, w are the weights and b are the biases. The node enters the weighted sum into the activation function that describes the node behaviour and gives one output. This output is then an input into the next layer of nodes in the neural network if there are any. The overall supervised learning of the neural network is done with the following steps:

1. initialize the weights,
2. take the input from the training data into the network, get the output and calculate the error of the network output with respect to the correct output from the training data,
3. adjust the weights and repeat the process.

The process of adjusting the weights based on the calculated error is called the learning rule. The applied learning rule is the so-called delta rule, and it can be expressed as:

$$w_{ij} \longleftarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = \alpha \delta_i x_j$$

$$\delta_i = \varphi'(v_i) \cdot e_i$$

where:

α – learning rate,

x_j – output from the input node,

e_i – error of the output nod and

$\varphi'(v_i)$ – the derivative of the activation function of the output node.

The problem is that we only know the correct output, and subsequently the error, of the last layer of nodes. To apply the delta rule to the neural network with hidden layers we need to calculate the error of the hidden layers. The error of the hidden layer is defined as the weighted sum of the delta functions of the first layer on the right. Once we have the error of the hidden layer, we can calculate the delta function of that layer and adjust the weights.

The neural network training is essentially an optimization problem where the objective is to minimize the cost function. The delta rule is derived from this, where the cost function is the sum of the squared differences and the minimum is found by applying the gradient descent method.

7.2. Aerodynamic analysis workflow

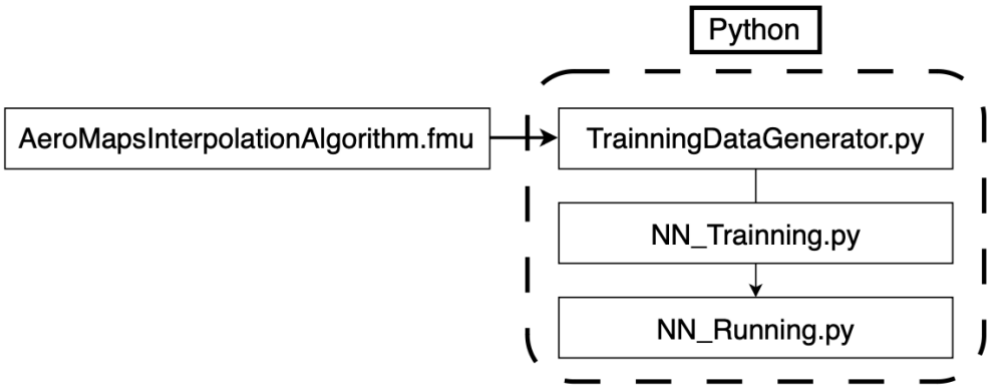

Figure 29. Aerodynamics analysis workflow

The overall workflow of the aerodynamic analysis starts within the Dymola environment, where an experiment based on the aerodynamic interpolation algorithm is defined to generate training data. This model is then exported as an FMU which serves as an input into the training data generator Python script. Within the Python environment, the workflow is divided into three main steps. First, the training dataset is generated by running the FMU over a predefined range of input parameters. Next, a neural network is trained using this dataset, producing the weights and biases. Finally, the trained network is used to predict aerodynamic outputs for given inputs.

7.2.1. Training Data generator

The first step is to generate the training data for the neural network. This training data is created by setting up a standalone experiment of the aerodynamics interpolation algorithm in Dymola. The algorithm takes five input variables, interpolates the aero maps and produces six coefficients. This experiment is exported as an FMU and represents an input into the "*TrainningDataGenerator.py*" script.

In the Python script, predefined ranges are assigned to the five input variables: front and rear ride heights, vehicle yaw, roll angle, and steering angle. An input table is then constructed containing all combinations of these parameters. Each combination represents a

single evaluation of the aerodynamic model and produces a corresponding set of six outputs: C_x(drag coefficient), $C_{z,f}$(downforce coefficient of the front axle), $C_{z,r}$(downforce coefficient of the rear axle), $C_{y,f}$(lateral coefficient front), $C_{y,r}$(lateral coefficient rear) and C_{mx}(rolling moment coefficient). The complete set of the input-output pairs forms the training dataset used for the supervised learning of the neural network model.

7.2.2. Training the neural network

The following step is the neural network training to get the weights and biases. The code used for the neural network training was available in the department, therefore, due to confidentiality, no detail information about the neural network training will be given.

What is presented is the sensitivity analysis that was done at the end of the training process. In the sensitivity analysis the goal is to evaluate the number of nodes of the neural network layers against the validation accuracy and the time spent training the network. This is best shown graphically in the figure below.

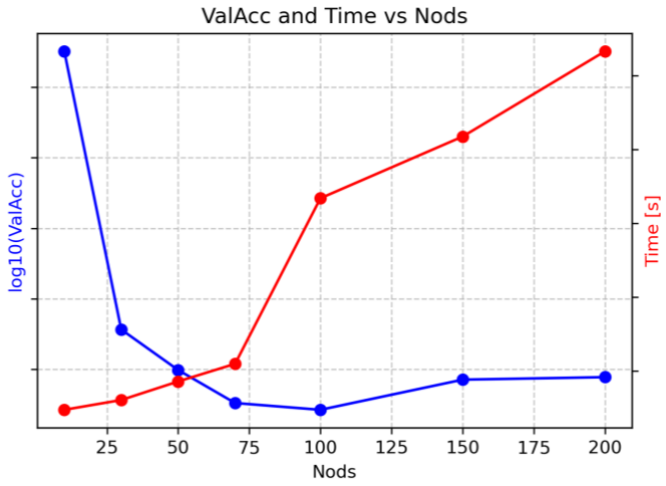

Figure 30. Neural network sensitivity analysis

From the graph we can say that using the neural network with 75 to 100 nodes makes most sense. Using 100 nodes will have a significant jump in the time necessary for the training but the accuracy will be the best achievable.

7.2.3. Running the neural network and validation of the results

Once the neural network has been trained, it is used to predict aerodynamic outputs for a given set of inputs. As this module is implemented as a standalone tool this is done in the third script "*NN_Running.py*". The first step is to build the neural network equations with the weights and biases obtained from the network training. The neural network is a two layer network with 100 nodes in both layers, and the used activation function of nodes is the hyperbolic tangent ($\tanh$). The neural network can be expressed in the following form for both layers:

$$v = wx + b$$

$$y = \varphi(v)$$

where:

y – output from the node and

$\varphi(v)$ – the activation function.

Once the network is defined, it can be evaluated for any set of input parameters. The input parameters are obtained from the DiL logged data of a lap around Silverstone circuit. From the logged data we extract the time traces of the 5 inputs of the neural network as well as the outputs for the validation. We are particulary interested in C_x, $C_{z,f}$ and $C_{z,r}$.

The final thing to do is to evaluate the neural network results against the interpolation algorithm currently in use. This is done in a similar way to the training data generation process, with the difference being that the inputs used to run the FMU of the aerodynamic algorithm are now the time histories of the five input variables. The comparison of results from the neural network, the interpolation algorithm and DiL logged data for the same inputs are shown below.

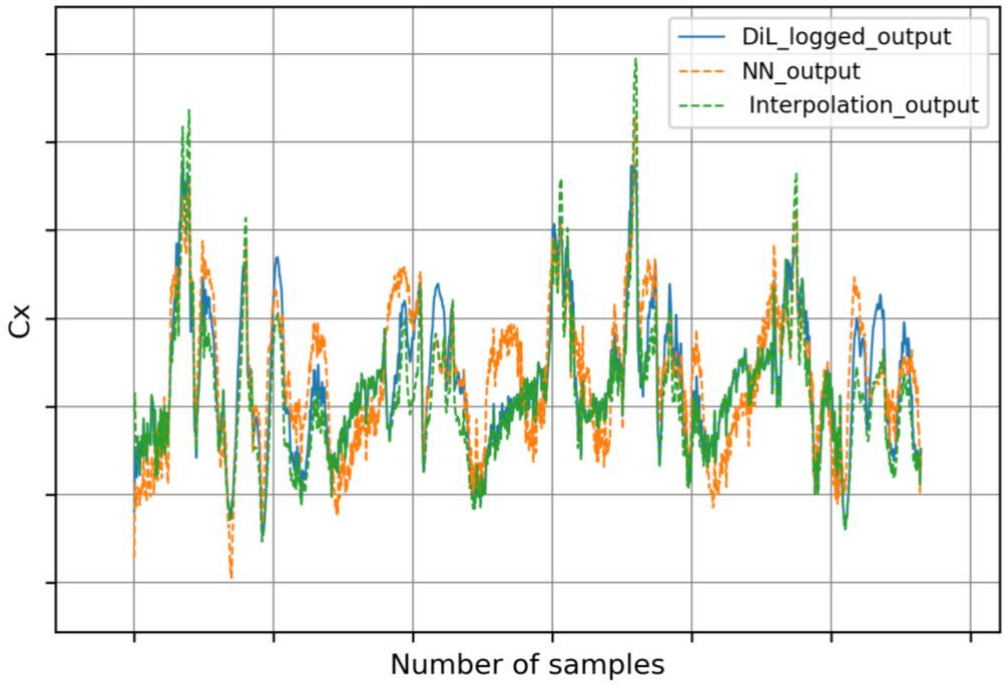

Figure 31. Comparison of neural network, interpolation algorithm and DiL logged data: Cx

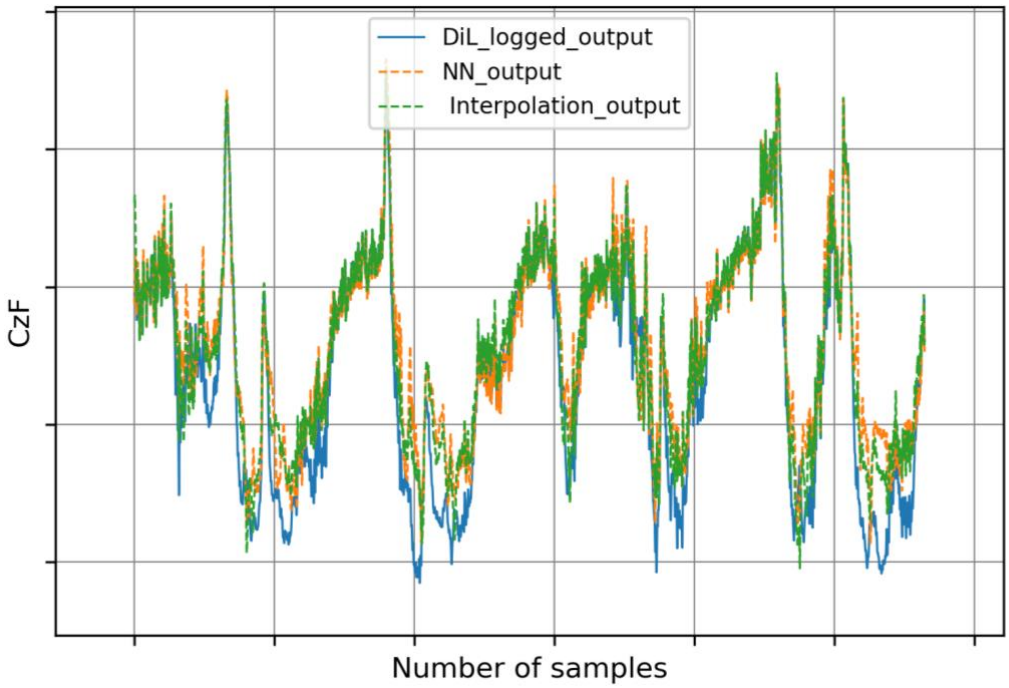

Figure 32. Comparison of neural network, interpolation algorithm and DiL logged data: Czf

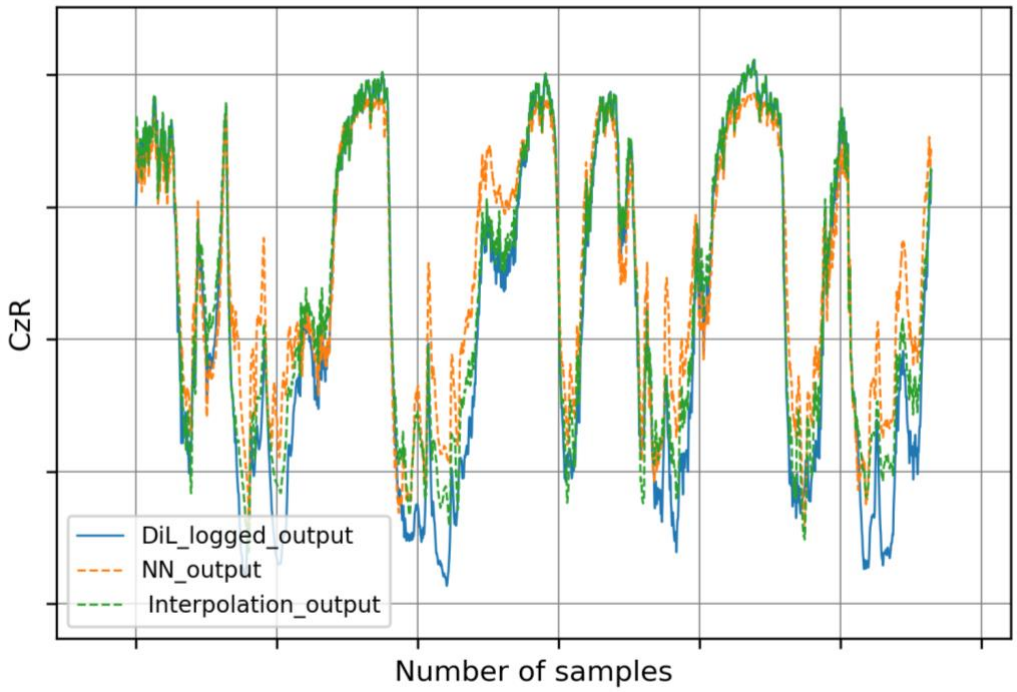

Figure 33. Comparison of neural network, interpolation algorithm and DiL logged data: Czr

8. Conclusion

The goal of this work was to develop a toolchain for parametrisation of a simplified double-track model for laptime simulation starting from the multibody model currently in use at the DiL simulator. It is essential to retain as much relevant physical information as possible within the simplified model, which is achieved by extracting parameters from detailed multibody models, either directly from the parameters file they use or through dedicated simulation experiments.

The developed toolchain is primarily implemented in Python and is structured around five main modules: static vehicle analysis, acceleration and braking, tyre modelling, aerodynamics, and suspension. For each module, dedicated scripts are created to process the required data and generate the parameters for the simplified model. The necessary inputs are obtained either directly from the vehicle data or by running experiments in Dymola, with the models exported as FMUs and executed through Python.

The static vehicle analysis module defines the mass and geometric properties of the vehicle. The acceleration and braking module, computes the available wheel torque, and evaluates the braking capabilities. The tyre module focuses on fitting a simplified Pacejka formulation to the reference Pacejka 5.2 model through an optimisation procedure. The suspension module captures the nonlinear behaviour by defining motion ratios of the elastic elements applying the principle of virtual work. Finally, in the aerodynamics module, the multi-input interpolation is replaced with a neural network model to obtain aerodynamic coefficients.

Overall, the developed toolchain represents a framework for bridging multibody and simplified simulation models.

LITERATURE

[1] Josef Bertucco, Development and Dynamic Analysis of a Multibody Model for the FSAE MG14.19 in Dymola

[2] https://modelon.com/blog/functional-mock-up-interface-fmi/, 20.03.2026.

[3] https://help-be.hexagonmi.com/bundle/Adams_2021.0.2_Adams_Tire_User_Guide/raw/resource/en_us/Adams_2021.0.2_Adams_Tire_User_Guide.pdf, 22.03.2026.

[4] MATLAB Deep Learning, With Machine Learning, Neural Networks and Artificial Intelligence, Phil Kim