

FH Vorarlberg
Vorarlberg University of Applied Sciences

Object-Oriented Modeling of Wheels
and Tires in Dymola/Modelica

Master's in Mechatronics

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering, MSc

Dornbirn, 2009

Supervisors

Dipl.-Ing. Dr. Reinhard Schneider - FH Vorarlberg

Prof. Dr. François E. Cellier - ETH Zürich

Dirk Zimmer MSc - ETH Zürich

Submitted by

Markus Andres BSc

Abstract

The thesis presents a free Modelica library for modeling wheels and tires. The contained models are intended to be used in vehicle simulations where computational performance is a major concern. Semi-empirical single contact point models are well suited for this kind of applications and are therefore applied in the presented library. These models enhance physical aspects by means of empirical equations that represent measurement results covering e.g. friction and slip characteristics. All featured tire models are split into seven objects, enabling a convenient customization of all relevant properties, with the necessary communication structure tailored to the special needs. Therefore the focus of this research effort concerns itself less with modeling new tire properties, but more with an improved organization of existing knowledge.

The *Wheels and Tires* library provides a tool to quickly build custom tire models. This is realized by a modular and expandable design system utilizing well established models. In addition, a set of ready-made models is provided to get a quick insight into the used modeling structure and to enable a direct application in vehicle models.

Kurzfassung

In der vorliegenden Arbeit wird eine frei verfügbare Modelica Library zur Modellierung von Rädern und Reifen vorgestellt. Die damit erstellten Modelle sind zur Verwendung in Simulationen von Fahrzeugen vorgesehen, was Rechenaufwand zu einem wichtigen Aspekt macht. Semiempirische Ein-Kontakt-Punkt Modelle sind gut für diese Art von Applikationen geeignet und finden daher in der Library Verwendung. Diese Modelle ergänzen physikalische Aspekte durch empirisch gefundene Gleichungen, welche Messergebnisse wie z.B. Reib- und Schlupfeigenschaften widerspiegeln. Alle enthaltenen Reifenmodelle sind in sieben Objekte aufgespalten, was eine bequeme Anpassung aller relevanten Eigenschaften ermöglicht. Die dazu nötige Kommunikationsstruktur wurde an die speziellen Bedürfnisse angepasst. Der Fokus der Arbeit liegt daher weniger auf der Modellierung neuer Reifeneigenschaften, als in einer Verbesserung der Organisation bestehenden Wissens.

Die *Wheels and Tires* Library stellt ein Werkzeug zur Verfügung, um Reifenmodelle möglichst einfach erstellen zu können. Ermöglicht wird dies durch eine modulare und erweiterbare Struktur, welche auf etablierten Modellen basiert. Zusätzlich werden eine Reihe von vorgefertigten Modellen zur Verfügung gestellt, um einen schnellen Einblick in die Struktur der Modelle, sowie eine direkte Anwendung als Teil von Fahrzeugmodellen zu ermöglichen.

Acknowledgments

Firstly I would like to thank Prof. Dr. François E. Cellier for the possibility to join his “Mathematical Modeling of Physical Systems” class, providing the topic for the master thesis and the advice during the last months. I would also like to thank Dirk Zimmer very much for the much appreciated time he spent discussing and reviewing parts of the library.

Furthermore I would like to express my gratitude to the Mechatronics administration team surrounding Dr. Johannes Steinschaden for their flexibility and great support. Many thanks as well to Reinhard Schneider for the most welcome mentoring during the thesis.

Thomas - a big thank you for the many times I needed friendship, motivation and technical discussion and most important for making all this possible! Thank you for being such a good colleague. I hope I have helped you a little bit as well.

Moreover, I would like to thank the other guys from the master’s program (MEM07). I really enjoyed the great time with all of you. Special regards to Mathias Schneller and Hannes Wachter for the extraordinary friendship we gained over the past years.

Finally I really want to thank my whole family for their reliable backing whenever possible and everything else I cannot even mention here.

Thanks!

Contents

1	Introduction	1
1.1	Structure of this Document	1
1.2	Motivation for Object-Orientation in Tire Modeling and the Semi-Em- pirical Single Contact Point Model	2
1.3	Mathematical Modeling of Physical Systems	3
1.3.1	Differential and Algebraic Equations	3
1.3.2	Bond Graphs	3
1.3.3	Object-Oriented Modeling	5
2	Basic Considerations regarding Tires	7
2.1	A Closer Look at Tires	7
2.2	State of the Art Tire Models	11
2.2.1	TMeasy by G. Rill	11
2.2.2	Magic Formula by Hans B. Pacejka	11
2.2.3	Object-Oriented Real-Time Model by D. Zimmer and M. Otter .	12
2.3	Definition of Vectors and Coordinate Systems	12
3	Object-Oriented Tire Modeling	17
3.1	Decomposition into Objects	17
3.2	Structure of the Tire and Libaray	20
3.2.1	Tire Model Structure	20
3.2.2	Library Structure	22
3.3	Communication Structure	23
3.4	Rim Class	27
3.5	Friction	27
3.5.1	The Friction Base Class	30
3.5.2	Slip Properties	33
3.5.3	The Rolling Resistance	42
3.5.4	Bore Torque	44
3.5.5	Camber Force	48
3.5.6	Influence of the Load on the Friction Coefficient	50
3.5.7	Overturning Torque	52
3.5.8	Self Aligning Torque	53

3.6	Geometry	56
3.6.1	Geometry Base	57
3.6.2	Flat Disc Model	58
3.6.3	Circular Tire	59
3.6.4	Belted Tire	61
3.6.5	Finding the Contact Point	64
3.7	Contact Physics	65
3.7.1	Contact Physics Base	66
3.7.2	Contact Bond No Dynamics	66
3.7.3	Contact Bond With Dynamics	68
3.8	Center to Contact Point	68
3.8.1	Center to Contact Point Base	70
3.8.2	Center to Contact Point Bond	70
3.9	Vertical Dynamics	72
3.9.1	No Elevation	72
3.9.2	Kelvin Elevation	72
3.9.3	Gehmann Elevation	75
3.9.4	Elasto Elevation	75
3.10	Belt Dynamics	77
3.10.1	Belt Dynamics Base	77
3.10.2	Rigid Belt	78
3.10.3	Torsion Belt	78
3.10.4	Longitudinal Lateral Belt	78
3.10.5	Quad Order Two Belt	78
3.11	Utilities	84
3.11.1	Additional Bond Graphs	84
3.11.2	Additional Multi Bond Graphs	87
3.11.3	Icons	93
4	Provided Tire Models	95
4.1	Predefined Models	95
4.2	Parameter Aggregation	97
4.3	Initialization	97
4.3.1	Initialization of Non-Slipping Tires	98
4.3.2	Initialization of Slipping Tires	98
4.4	Defining New Tires	99
5	Environment	101
5.1	Surface Base	101
5.2	Surface Class	102
5.2.1	The used Method	102
5.2.2	Changing the Surface Class	108
5.2.3	Implementation Details	109

6	Simulation Results	111
6.1	The Test Bench	111
6.2	Provided Examples	114
7	Conclusions	119
8	Further Work	121
	Bibliography	126
A	Appendix	i
A.1	Library Structure	i
A.2	Complete Code for the Surface Class	xi
A.2.1	WheelsAndTires.Environment.SurfaceBase	xi
A.2.2	WheelsAndTires.Environment.Surface	xi
A.2.3	WheelsAndTires.Visualization.SurfaceVisualization	xix
A.3	Paper submitted to Modelica Conference 2009	xxii
A.4	Paper submitted to “Tag der Mechatronik 2009”	xxxiii

List of Figures

1.1	A directed harpoon (bond) to model energy flow.	4
2.1	The basic elements of a tire.	8
2.2	Tire brush model under different driving conditions.	9
2.3	Function resulting from the magic formula.	12
2.4	The unitary vectors of the tire without a lean angle.	13
2.5	The unitary vectors of the tire with a lean angle φ of 10°	14
2.6	An enlarged view of the contact point with vectors for longitudinal speed v_{Long} , rotational speed ω_{Pitch} and longitudinal slip velocity $v_{SlipLong}$	15
2.7	Standardized names of angles and angular velocities.	15
3.1	Composition of wheels and properties of rims and tires.	18
3.2	Properties from Figure 3.1 shown in relation to the corresponding Tire Component classes which are closely related with the semi-empirical contact point model.	19
3.3	Final decomposition of the tire properties.	20
3.4	Model of the ideal tire showing the seven used objects and the communication structure on the top level.	21
3.5	The library's top level packages.	22
3.6	The elements contained in the <i>Tire Bus</i>	24
3.7	The elements contained in the <i>Contact Point Connector</i>	25
3.8	<i>Geometry Base</i> model showing the communication on the second level of modeling.	26
3.9	Model of an ideal rim containing mass and inertia tensors.	28
3.10	Model of an ideal frictional behavior.	29
3.11	Friction Base class defining in- and outputs as well as some of the used variables.	31
3.12	The frictional coefficient μ depending on the sliding velocity v_{Slip} showing the two descriptive points.	34
3.13	The dependence of force generation in the longitudinal direction on the sliding velocity.	35
3.14	The dependence of force generation on the overall sliding velocity with two different cases shown.	37

3.15	The friction coefficient in dependence on the overall sliding velocity and the velocity due to pitch angle rate.	40
3.16	The influence of the parameter <code>tol</code> on the transition between the two frictional models.	41
3.17	Constant roll resistance torque depending on the pitch angle velocity.	43
3.18	Speed dependent roll resistance torque depending on the pitch angle velocity.	44
3.19	The bore torque depending on the yaw rate at different pitch angle velocities in the linear model.	46
3.20	The bore torque depending on the yaw rate at different pitch angle velocities in the model from [ZO09].	47
3.21	View of a wheel with lean angle and highlighted contact area including a sketch of the maximal camber deflection.	48
3.22	The lateral velocity due to the camber angle.	51
3.23	The load influence factor and longitudinal friction force in dependence on the normal load f_N	53
3.24	The overturning torque in dependence on the lean angle φ	54
3.25	The dynamic trail depending on different physical quantities.	55
3.26	The pneumatic trail by [ZO09] depending on different variables with the <i>Combined Friction</i> model for the slip properties.	56
3.27	A tire with an ideally stiff belt penetrating the surface.	57
3.28	Geometric properties of the ideal tire.	58
3.29	Geometric properties of the circular tire.	60
3.30	Geometric properties of a tire with side walls and tread area.	62
3.31	The model applying forces and torques to the contact point and measuring its velocities without any dynamics.	67
3.32	Model that enables the contact point to shift in longitudinal and lateral direction.	69
3.33	Bond graphic model for the translation from the center of the rim/belt to the contact point.	71
3.34	Model that makes the tire stick to the ground.	73
3.35	Model of the vertical dynamics with possible elevation and a parallel spring damper elements realizing vertical dynamics.	74
3.36	Model of the vertical dynamics with possible elevation and a special spring damper combination to overcome the sticking effect.	76
3.37	Model of an ideal belt without any dynamics.	79
3.38	Model of a belt able to rotate around its spinning axis.	80
3.39	Model of a belt able to translate in longitudinal and lateral direction.	81
3.40	Model of a belt defined by four spring damper systems and eight translational elements.	82
3.41	Alternative model of an angle source with one less equation compared to the original one from the <i>Bond Library</i> and a non ideal derivative block.	86
3.42	Bond graphic model of a serial spring damper system.	88

3.43	Model enabling a force free lift from the ground (with sticking effect).	89
3.44	The icon of the <i>Boolean</i> signal.	90
3.45	Model of a <i>Modulated Actuated Prismatic</i> element.	92
3.46	Icons used in the library.	94
4.1	Predefined tires available in the <i>Wheels and Tires</i> Library.	96
5.1	The foundation for the interpolation used to compute the elevation y and normal vector in a square.	102
5.2	Coordinates for multiple rectangles assembled to form a more complex surface.	105
5.3	The parameter window for customizing the <i>Surface's</i> properties.	106
5.4	The result of the surface configuration shown in Figure 5.3 without any further interpolation for the visualization without a PNG as texture.	107
5.5	The result of the surface configuration shown in Figure 5.3 without a PNG as texture and five interpolation points between the vertices.	107
6.1	Model that applies a driving torque making the tire slip.	112
6.2	Tire elevating from the ground while driving a curve. Result of <i>Elevation Advanced</i> .	113
6.3	Tire's overturning torque <i>Test Bench</i> result.	113
6.4	A tire rolling up an uneven surface with an initial lean angle (resulted from <i>Uneven Surface Rotating Tire</i>).	114
6.5	Tire dropping on an uneven surface (resulted from <i>Uneven Surface Dropping Tire</i>).	115
6.6	Two similar bicycles with non-slipping tires differing in their geometric properties.	115
6.7	Two similar bicycles with slipping tires differing in their frictional properties.	116
6.8	A bicycle with non-ideal tires accelerated by a torque on the rear tire, with the front tire lifting from the ground.	116
6.9	A motorcycle jumping over a gap.	116
6.10	Result of the <i>Four Wheeled Vehicle</i> example after 22.5s (vehicle is drifting)	117

1 Introduction

The term “wheel” may be used in a narrow sense to describe the part connecting the tire to the rim or in a wide sense to contain the whole of the rotating elements including the tire [Dix96]. The latter one is used in this work.

Modelica libraries, packages and models cannot have spaces in their names. As a result, the beginning of a new word in a name, made up of multiple words, is marked with a capital letter in Dymola. For the documentation, spaces are added for the sake of readability and abbreviations are written in full. To be able to identify classes, packages and libraries easily, these are referred to in *italic letters*, e.g. the “NoRollRes” model is referred to as *No Rolling Resistance*. Additionally references to elements in figure are written in *italic font*. Names of elements contained in models like variable names, and functions as well as sub-models and classes are referred to in a **typewriter** font.

1.1 Structure of this Document

The first chapter covers the most general parts of the thesis including the motivation for this work in Section 1.2 and a very general introduction into the used tools and techniques in Section 1.3. Readers familiar with the used techniques and tools can skip Section 1.3. The following Chapter 2 is a very compact presentation of general tire properties that can be skipped by tire specialists. The third and main chapter of this work covers the development of the tire model’s structure, followed by an introduction of the communication elements used and all the classes the object-oriented tire can be composed of. Therefore, it is a rather extensive chapter that can either be used as reference, when an effect is not totally clear or it can be read completely to gain a deeper understanding of the library. It is followed by Chapter 4 introducing the ready-made tire models provided in the library. Chapter 5 then depicts how the surface class covering even and uneven surfaces are implemented. The following chapter tries to give an insight into the kind of simulations which can be carried out and how tests are realized. Therefore, it is probably the best chapter to get a general impression of the library’s capabilities and is therefore highly interesting for somebody totally new to the library. It is recommended to be read first if the applicability of the library is not

ensured. The last two Chapters 7 and 8 present possible enhancement to the library and some concluding thoughts respectively.

To get a compact overview of the whole work, the reader is referred to a paper contained in Appendix A.3. This ten page paper introduces all important properties of the tire model and is well suited to provide a quick insight in the work. An even shorter introduction is provided in Appendix A.4.

1.2 Motivation for Object-Orientation in Tire Modeling and the Semi-Empirical Single Contact Point Model

During the last decades a fairly large number of tire models of varying levels of complexity suiting strongly differing fields of application have been developed. These range from simple non-slipping tires to very complex FEA (finite element analysis) models for performance prediction [BSGR08].

The library developed is intended to be used in simulations that cover entire vehicles, therefore computational effort is an important issue. Hence, the selection of the appropriate level of detail for the used models is essential for the overall simulation performance. Semi-empirical single contact point models provide a very good trade-off between accuracy and computational effort. Such models are based on physical considerations, like those emerging from multibody dynamics. These physical aspects get enhanced with empirical formulas representing measurement results that cover e.g. friction and slip characteristics. Two of these semi-empirical models are commonly accepted and widely used. These are “TMeasy” by G. Rill [Ril07] and the “magic-formula” model by H. B. Pacejka [Pac06]. However, both are often implemented in a flat and mainly unstructured fashion, which makes them difficult to understand and maintain. Customizing these models for particular situations or expanding them in order to cover new aspects of tires can be hard and is often error-prone.

A paper by D. Zimmer and M. Otter [ZO09] builds on the previously mentioned models and demonstrates how models of varying levels of complexity can be integrated within the object-oriented framework of Modelica. However, the object orientation in these models limits itself primarily to their external interfaces. The models themselves continue to be mostly flat. For instance the most complex tire model created, defines approximately 200 equations [ZO09] and is a good example showing the difficulties that arise from the common flat structure.

Another example for a quite flat structure can be found in the freely available but outdated Vehicle Dynamics library [And03]. There, a wheel base model gets extended with friction models of [Ril07] and [Pac06] but not much further effort was spent regarding object orientation.

In [AJ02] a tire model is modularized in hub, belt and road elements. A further enhancement is made in [BA03] by redesigning the model's structure as well as enabling uneven road surfaces and losing contact to the ground due to enhanced vertical dynamics. This modularization is well defined, but still the different aspects of friction are summarized in the *Tyre-Road* class and cannot be customized easily. Moreover the libraries presented in [ZO09], [AJ02] and [BA03] are not freely available.

The newly developed library takes the object-orientation even further than in [BA03]. Therefore the focus of this research effort concerns itself less with modeling new tire properties, but more with an improved organization of existing knowledge. This will enable future modelers to conveniently customize the models to their own purposes.

1.3 Mathematical Modeling of Physical Systems

The content of this section can also be found in [Sch09] because the same basic requirements and prerequisites are valid for this work as well. It was created by the two authors in cooperation.

The purpose of *modeling* is to describe a physical system by a mathematical model composed of differential and algebraic equations (DAE's). To this end, the behavior of such a system is described as accurately as possible in order to compute and predict the system's behavior. As soon as the physical system is described by an appropriate mathematical model, a simulation can be carried out. In the following sections, the steps necessary to simulate a physical system are introduced. This is done with reference to three different modeling techniques.

1.3.1 Differential and Algebraic Equations

In general, a physical system can be described by DAE's. The most basic approach is to find the system of DAE's manually by setting up the equations of the elements that the system is composed of and combining them in an appropriate manner. As soon as a proper description of the system is found, the DAE's have to be transformed into a state-space representation to be solved. The state space representation is a description of the system by means of ordinary differential equations (ODE's) i.e. a system of 2^{nd} order can be described with 2 ordinary differential equations.

1.3.2 Bond Graphs

Another approach to model physical systems are bond graphs. They are based on energy flow through systems with the basic laws of energy conservation applied. These state that energy can only be affected by three mechanisms: It can be *stored*, *transported*

or *converted*. Energy flow is the derivation of energy by time which is also referred to as power. In every physical system, power is the product of an effort and a flow variable, e.g. for electrical systems the effort is the voltage, the flow is the current.

The energy flow is represented in a graphical manner by directed harpoons as shown in Figure 1.1 and is mathematically described by the relation $P = e \cdot f$. A bond is thus the

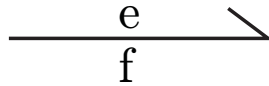


Figure 1.1: A directed harpoon (bond) to model energy flow.

first element representing the mechanism “transport” of the energy conservation laws. The bonds connect passive (resistive, capacitive and inductive) and active (sources) components via 0-junctions and 1-junctions which are used to state either the same effort or the same flow (e.g. in an electrical circuit, a 0-junctions represents Kirchhoff’s current law whereas a 1-junction represents Kirchhoff’s voltage law). Basically, resistive components are used to convert energy, e.g. the energy flow through a resistive element generates heat. However, in electrical and mechanical systems more often than not such elements are treated as dissipative elements since the thermodynamical aspects are of less interest. Capacitive and inductive elements store energy. The former is a so-called flow storing element whereas the latter stores effort. The energy itself is supplied by flow and effort sources.

As the elements are very basic, they can be used when modeling several different domains including mechanics (1D translational and rotational), electrics, thermal, hydraulics etc. For a more detailed description regarding bond graphs, the reader is referred to [Cel91] or [McB05].

1.3.2.1 Multi Bond Graphs

Multi Bond Graphs are a vector extension of regular bond graphs. A freely selectable number of regular bonds can be combined to form a Multi Bond. This was done in [Zim06] developing a Modelica/Dymola *Multi Bond Library*¹ for the modeling of 2D and 3D mechanics. This eases the modeling of mechanical systems, which can get cumbersome with regular bond graphs due to difficulties when handling positional information and holonomic constraints.

¹The exact name of the library is “MulitBondLib”.

1.3.3 Object-Oriented Modeling

The intention behind object-oriented modeling is basically the same as in object-oriented programming. The modeler tries to describe parts of a physical system with classes that behave as the modeled part of the real system. The reusability of these classes should be as high as possible so e.g. an electric machine should be described by the same model when acting as a generator and when acting as a motor. This makes models based on equations (a-causal) rather than assignments (causal) a necessity. The modeling environment has to be able to handle the resulting sets of equations by a symbolic preprocessing.

1.3.3.1 Modelica

Modelica is an object-oriented open-source modeling language providing the language definition [Ass07] as well as a standard library [Ass09] for modeling in different physical domains. Different commercial simulation environments like *SimulationX*, *MathModelica* and *Dymola* as well as some free tools like *OpenModelica* use the language as a base. A very detailed introduction to Modelica can be found in a book, published by Peter Fritzson [Fri04].

1.3.3.2 Dymola

Dymola from Dynasim is a very advanced Modelica environment capable of performing all necessary symbolic transformations required for convenient modeling, able to handle very big systems (> 100,000 equations). It features a graphical editor for model creation as well as a text based view. Interfaces to MATLAB and Simulink exist in order to integrate models in existing simulation environments.

For this work Dymola 6.1 that utilizes Modelica 2.2.1 was used.

2 Basic Considerations regarding Tires

The following section is intended to give a broad overview of tires and how they can be modeled. Readers, who are familiar with tire and their modeling, can skip Sections 2.1 and 2.2, but should consider reading Section 2.3 as basic definitions used throughout the following chapters are introduced there.

2.1 A Closer Look at Tires

In motion dynamics of vehicles, the forces exerted by the tire-road contact are of major importance. This section is intended to provide basic knowledge about tire properties, buildup and force generation. For more detailed information, the reader is referred to [Ril07] for a good compact German presentation of tire modeling, to [Lei09] for a very good and general German introduction, to [Dix96] for a rather praxis-oriented introduction or to [Pac06] for probably the most detailed description of tire modeling available, the latter ones both in English.

The first historically known wheels had totally different duties compared to today's wheels and tires. They were attached to vehicles which were dragged by a translational force. Hence they "just" had to carry the vertical load and overcome their own rolling resistance. Later on, first braking systems were introduced widening the field of requirements of tires without a major impact on their construction. When tires got driven by torque, a redesign became necessary in order to transmit great forces to the ground or road. Various alternative ideas have been investigated, with the rubber-carcass pneumatic tire resulting to be superior to all competitors.

The modern tire is a complex construction resulting from clashing requirements. Tires basically have to carry the vertical load, transmit forces to accelerate (and slow down) the vehicle and generate cornering forces to guide the vehicle through curves securely. This has to be fulfilled under a large variety of environmental conditions with a long life time ensured. The rolling resistance has to be as small as possible, with damping and acoustic properties suiting modern demands. As one can imagine, there is no optimal solution to this problem and this leads to a large variety of different tires for varying demands, especially when peak performance is required as under racing conditions.

A quite basic design example for a tire is depicted in Figure 2.1. For today's passenger

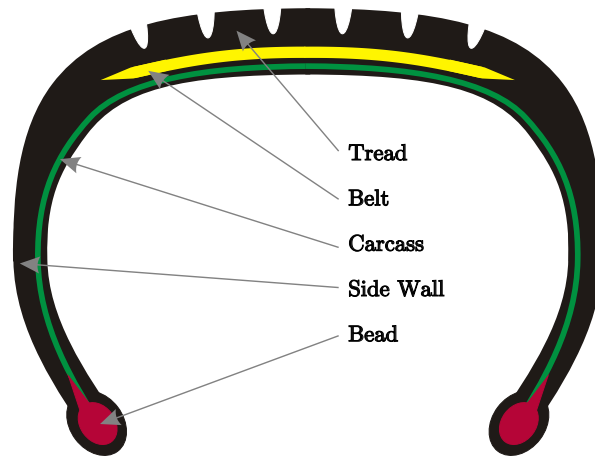


Figure 2.1: The basic elements of a tire.

cars steel-belt tires are used exclusively, which differ in construction only marginally, when treated from such a basic point of view. On the inside of the tire a coating (not depicted in Figure 2.1) inherits the function of the tube, preventing the over-pressurized air to leak to the outside. The *Bead* that is usually built of steel wire with synthetic rubber components, ensuring a tight fit of the tire on the rim, allows a reliable operation under difficult conditions e.g. when driving over a curbstone. The rubber elements building the sidewall strongly affect the vertical dynamics of the tire and are important when it comes to handling precision and stability. The *carcass* is the element absorbing the tension from the inflation pressure. Therefore, it has to be protected from damage, which is ensured by the *side wall*. The *tread* is responsible for the force generation by establishing a reliable contact to the road and is therefore a very central element of the tire. Its composition is a major factor when it comes to the frictional properties of the tire. The *tread* is reinforced by the steel *belt* that enhances mileage and reduces the rolling resistance. Overall a mixture of more than 20 rubber composites form the tire, which makes them quite difficult to describe as well as enabling the engineer to adjust the tire properties to several different needs.

For the explanation of the mechanisms which are responsible for the generation of force in the tread shuffle of the tire, the brush model is quite common [Ril07], [Pac06], [Lei09]. Looking at Figure 2.2, one can see the shuffle described by multiple spring like elements. These elements generate a force when getting deflected. The force of these elements is assumed to be proportional to the deflection. Under the conditions depicted in Figure 2.2(a), the tread elements are not getting deflected and no force is being generated. This is the case when the tire is driven with a torque that exactly makes it overcome the rolling resistance. If the vehicle accelerates, the tires has to generate a

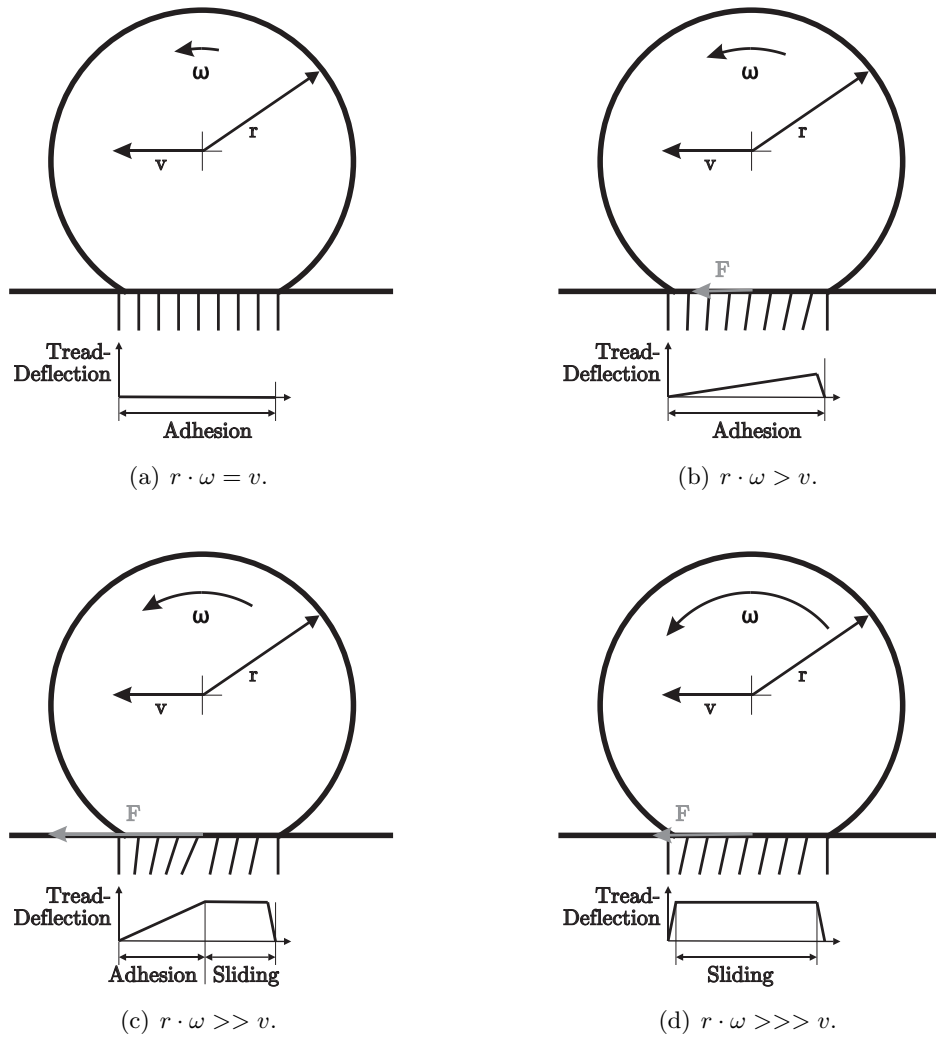


Figure 2.2: Tire brush model under different driving conditions.

force F which results in the rotational velocity rising as shown in Figure 2.2(b). Now $r \cdot \omega > v$ which results in deflecting elements of the tread shuffles. The elements at the right side generate the bigger part of the overall force due to the greater deflection and the (assumed) linear characteristics. If the driving torque rises further, the maximum possible deflection that is determined by the frictional conditions is reached and the elements partially start to slide reducing their deflection and generated force. This is depicted in Figure 2.2(c). An even further increase of the driving torque results in very quick sliding of the elements as shown in Figure 2.2(d). Still, there is a short period of adhesion at the very beginning of the tread shuffle that can be neglected. Due to the frictional properties, the transmitted force is lower than in Figure 2.2(c). The result of this behavior and its influence on the modeling is explained in Section 3.5 on Page 27 ff where the models computing the frictional forces are explained.

The difference between the two velocities $\omega \cdot r$ and v is called the slip velocity from which the slip can be calculated as shown in Section 3.5.1 on Page 30. One could split this slip or slip velocity into a “virtual” slip that is caused by the deformation of the still adhering components and the “real” slip that is caused by the sliding of the tread elements over the ground. [MW03] discusses this in more detail.

Taking a closer look at the effects causing the friction, one can find two major reasons for the generation of frictional forces: adhesion and hysteresis [Lei09], [Dix96]. Measurements [MW03] show that for dry surfaces, the adhesion is the main reason for frictional forces. The postulate for the adhesion to fully apply is a direct contact between the tire’s rubber and the road. If this contact is pretended by a water layer, the adhesion component decreases making the hysteresis component more important. The hysteresis applies in combination with form closure that is caused by micro-roughness of the tire and the road. In contrast to the classical Coulomb friction, there is a dependence on the contact area (the tread shuffle). For the adhesion component, the frictional coefficient rises with decreasing surface pressure (force per area). Therefore, a big tread shuffle is of advantage for the adhesion component of the friction, making wider tires the better alternative on dry surfaces. On wet surfaces the situation inverts because higher surface pressure enlarges the expulsion of water and the effect of the hysteresis directly. So, for tires that are used in dry as well as wet conditions, a compromise has to be found. Another factor that is very important, is the distribution of pressure in the tread shuffle. To make the situation even more difficult, both of the effects show a dependence on sliding velocity and temperature.

To enable the tire to transmit big forces, the tread area has to have a high damping coefficient to enhance both adhesion and hysteresis. This is the only part of the tire that is designed to have a big damping constant because in all other parts this just causes rising losses. To get an overview of the tire losses, [MW03] states that about 50% of the losses emerge from the tread area, 20% arise in the belt, 10% from the carcass and another 10% from the side walls.

2.2 State of the Art Tire Models

The following sections quickly introduce the very basics of the tire models used in the development of the *Wheels and Tires* library.

2.2.1 TMeasy by G. Rill

G. Rill's *TMeasy* or the *Easy to Use* tire model is intended to provide a quite accurate but still simple model [Ril07]. The main emphasis is to make it work with little information (parameters) of the tire. Based on relatively simple geometric assumptions and piecewise polynomial equations, slip models are provided to compute

- frictional forces,
- force due to camber angle,
- nonlinear influence of the normal load,
- self aligning torque,
- bore torque,
- overturning torque and
- rolling resistance.

TMeasy is implemented in most parts of the frictional class that is presented in Section 3.5 on Page 27.

2.2.2 Magic Formula by Hans B. Pacejka

The probably most popular tire model of today is introduced in [Pac06]. Rather than using piecewise polynomial equations it is mainly based on a single equation that is called the magic formula which has the following form.

$$y = D \sin[C \arctan\{Bx - E(Bx - \arctan Bx)\}] \quad (2.1)$$

With B being the *stiffness factor*, C the *shape factor*, D the *peak value*, E the *curvature factor*, x the input, and y being the output of the function. It results in a curve like the one depicted in Figure 2.3, with some of the basic relations shown. These relations are quite straightforward making the formula that popular. For the other relations not depicted in Figure 2.3 the reader is referred to [Pac06].

Equation (2.1) can be used to model the following effects.

- Longitudinal force

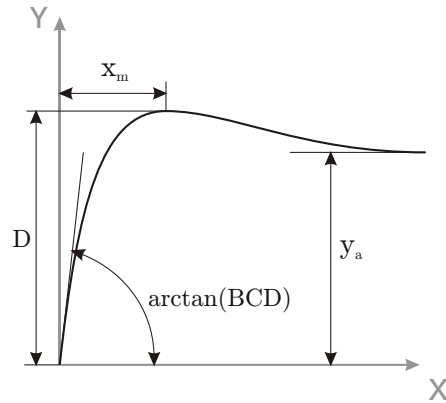


Figure 2.3: Function resulting from the magic formula.

- Lateral force
- Aligning torque

The semi-empirical tire model in Chapter 4 of [Pac06] additionally introduces equations for the friction property's dependence on normal load, overturning couple and rolling resistance and therefore defines a model of about the same complexity as *TMeasy*. In the subsequent chapters of [Pac06], tire properties are discussed in extensive detail covering non-steady-state properties, the shimmy phenomenon, transient behavior, short wavelength models as well as road unevenness.

Unfortunately, the *Magic Formula* model was not implemented in the current version of the *Wheels and Tires* library due to time constraints.

2.2.3 Object-Oriented Real-Time Model by D. Zimmer and M. Otter

The object-oriented real-time model presented in [ZO09] is mainly based on *TMeasy* by G. Rill with some adjustments made to suit the Modelica environment and to enhance its properties. A seven level model with increasing complexity is introduced enabling the modeler to choose the level of complexity suiting the current application. As it is developed for the Modelica environment equations found in [ZO09] are used as a base for the model presented in this work rather than the *Magic Formula* model.

2.3 Definition of Vectors and Coordinate Systems

For the modeling of wheels, unit vectors describing the orientation of the tire are essential. This is true for the wheel's (rim's) center point as well as the contact point.

The center's properties are important to find the contact point and to describe translational movement. For the contact point, it is important to have longitudinal, lateral and normal vector forming a coordinate system that is used when forces and torques are applied on the contact point.

Figure 2.4 shows the unitary vectors of the contact point (e_{Long} , e_N and e_{Lat}) and the

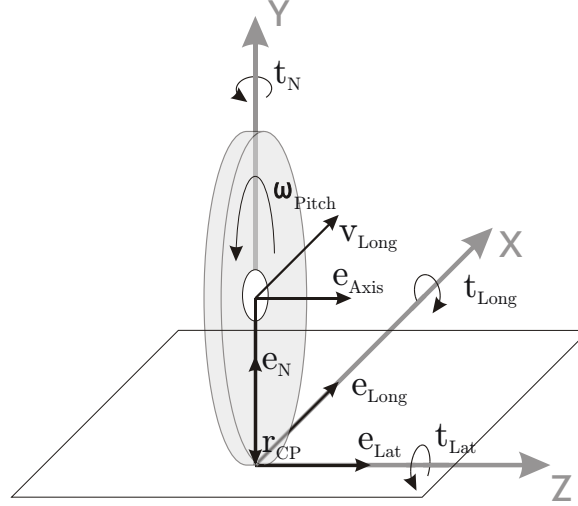


Figure 2.4: The unitary vectors of the tire without a lean angle.

rotational as well as the (longitudinal) translational velocity (ω_{Pitch} and v_{Long}). Forces and translational velocities always point in the direction of the corresponding unit vectors, whereas the corresponding torques act about them. The vector r_{CP} points from the center of the rim to the contact point and is a fundamentally descriptive element of the tire.

In Figure 2.5 the same wheel is shown again with a lean angle φ of 10° to show the changes when tilting the wheel. Additionally the vector e_{Plane} is added to the figure that was not shown in Figure 2.4 to enhance clarity.

For forces generated by the tire, slip and sliding velocity are major factors. How slip is computed from the sliding velocity is shown in detail in Section 3.5.1 on Page 30. Figure 2.6 depicts the sliding velocity in longitudinal direction $v_{SlipLong} = (\omega \times r_{CP}) \cdot e_{Long} + v_{Long}$. There is a lateral sliding velocity as well $v_{SlipLat} = v_{Lat}$ that was not depicted in the figure for the sake of clarity. The overall slip velocity is then computed from the two longitudinal and lateral components by Pythagoras' rule as they are orthogonal. The computation of the unit vectors is presented in Section 3.6.1 on Page 57.

Figure 2.7 depicts the standardized names of angles as well as angular velocities used in this work.

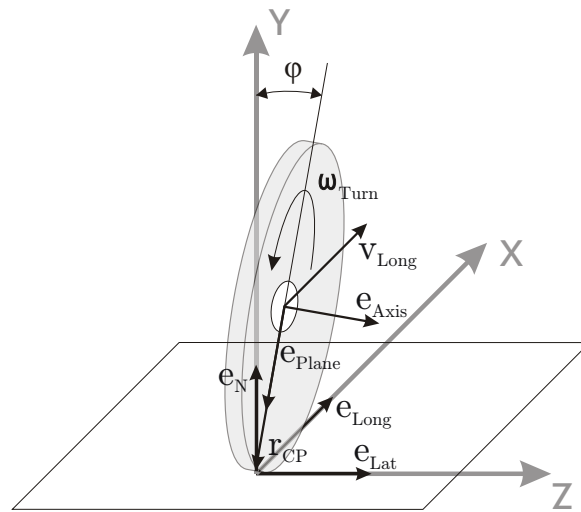


Figure 2.5: The unitary vectors of the tire with a lean angle φ of 10° .

The coordinate system was chosen this way in order to suit the simulation environment Dymola. DIN (Deutsche Industrie Norm) defines the coordinates differently as well as the SOE (Society of Automotive Engineers), whereas these differ from each other also.

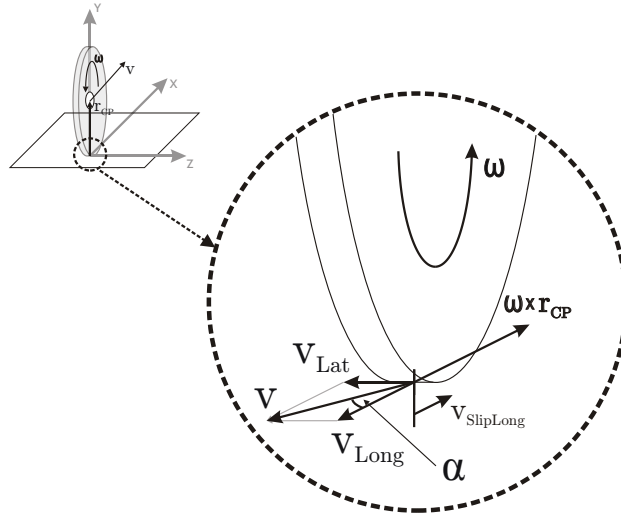


Figure 2.6: An enlarged view of the contact point with vectors for longitudinal speed v_{Long} , rotational speed ω_{Pitch} and longitudinal slip velocity $v_{SlipLong}$.

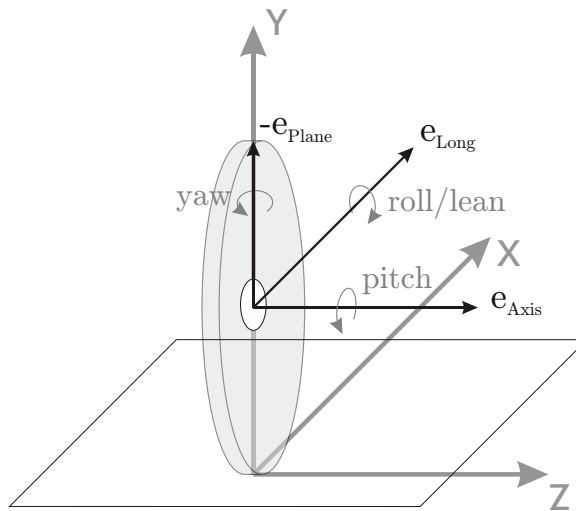


Figure 2.7: Standardized names of angles and angular velocities.

3 Object-Oriented Tire Modeling

In this section the separation of the different tire properties into objects will be described. First the splitting into the used objects is justified. Afterwards a basic outline shows the structure of the tire models and the library. This is followed by a description of the communication structure which is used to share information among the objects. The remaining sections explain the single classes forming the overall tire model.

3.1 Decomposition into Objects

The motivation for a decomposition of a tire model into objects was explained in Section 1.2 on Page 2. Therefore this section focuses on demonstrating the considerations that lead to the actual structure of the model.

Thinking about wheels, the first division of the model is quite obvious, as there are two physical components: the rim and the tire. The rim does not need to be split up into further objects, as the important properties for the overall wheel are very limited in a semi-empirical tire model. This characteristic is shown in Figure 3.1, which also lists the most important properties of tires regarding modeling, split up into different groups. The model of the rim is quite simple just modeling its mass and inertia tensor. It can be found in more detail in Section 3.4 on Page 27 ff.

The tire does require a much closer look concerning its properties, as they are of much bigger importance for the overall behavior of the wheel than the ones of the rim. Modeling every single of the properties shown in Figure 3.1 by a class of its own is infeasible task, suggesting a certain grouping of properties. Due to the semi-empirical single contact point model certain constraints are fixed. There has to be a contact point as part of the model, on which forces and torques act. The model realizing the contact point is named *Contact Physics* and explained in detail in Section 3.7 on Page 65 ff.

One of the most challenging tasks of a tire model is to calculate the forces the tire excites in different driving situations. There are plenty of different models trying to describe the relation resulting in the forces which act on the contact point. Hence, a decision was made to create a class that gathers the different effects which are responsible for the generation of forces and torques acting on the contact point. Due to its origin is it

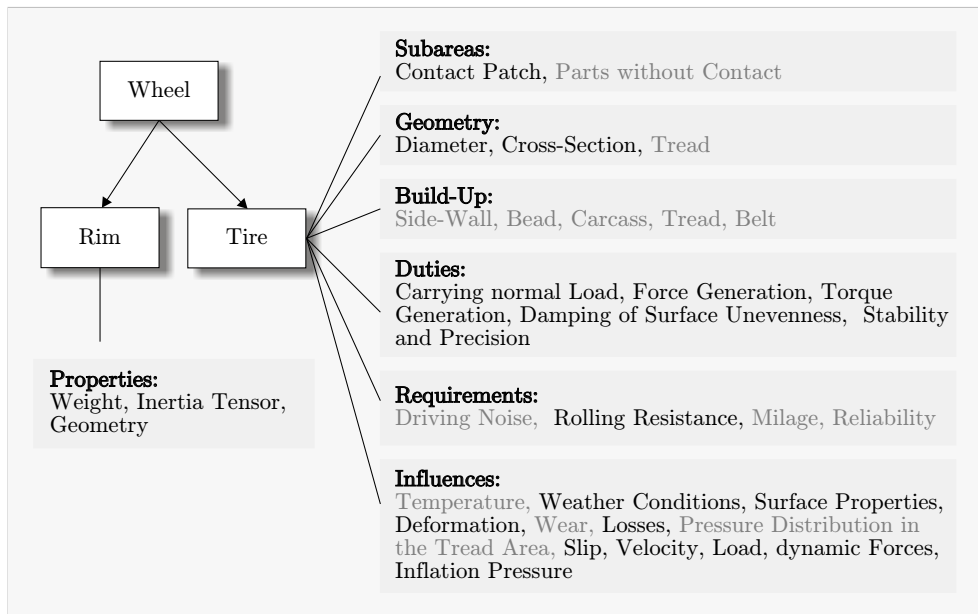


Figure 3.1: Composition of wheels and properties of rims and tires. Properties depicted in gray are neglected in the tire model.

is called the *Friction* class. As one can imagine this is a rather complex class covering multiple effects. Hence, a further breakdown into different sub-classes is made, for the model to be well manageable with a structure which is simple to adapt and expand. How this is realized is demonstrated in detail in Section 3.5 on Page 27 ff.

Figure 3.2 depicts the status of modeling, including the restriction by the single contact point model and the reasonable introduction of the *Friction* class. One can easily identify that absolutely basic properties of a tire are not yet part of the model. The probably first to catch the reader’s eye are the *Tire Diameter* and the *Cross-Section*. These properties basically define the geometry of the tire, which shall be changeable conveniently in the final tire model, allowing basically different geometric properties of the tire. Therefore a *Geometry* class is introduced, defining the positional relation between the tire hub and the contact point and some other properties which are further discussed in Section 3.6 on Page 56 ff.

The next very basic duty the tire has to fulfill, which is not yet modeled, is the *Carrying of normal Load*. Depending on the tire model different techniques are utilized to realize this functionality. These range from the holonomic constraints defining ideal tires to more complex models including a certain *Damping of Surface Unevenness* by the introduction of more complex models. Due to the changing requirements to these aspects, the effects were gathered in a class named *Vertical Dynamics*. It basically determines how the tire reacts to normal load and if the ideally stiff *Geometry* is able

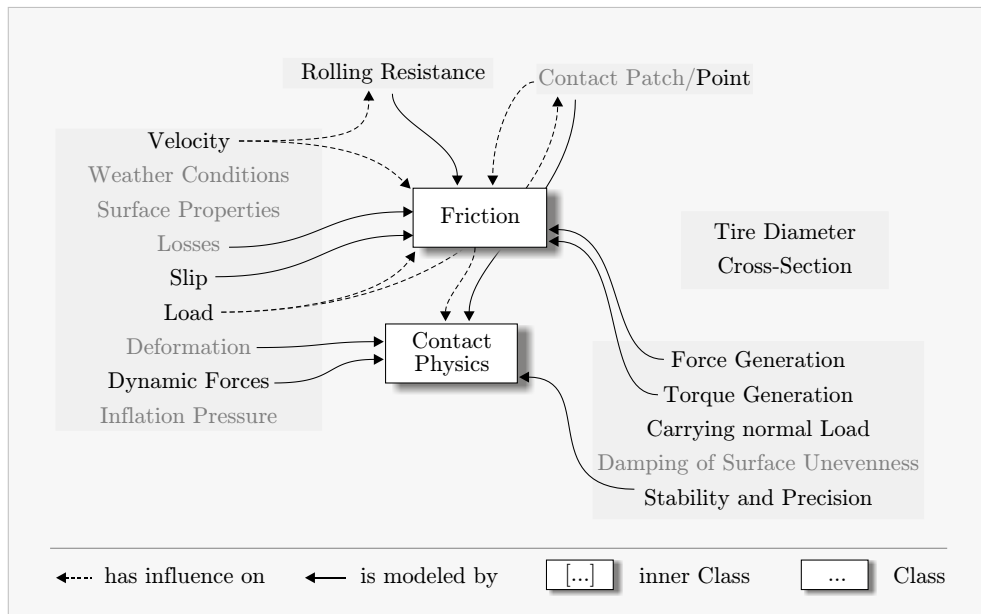


Figure 3.2: Properties from Figure 3.1 shown in relation to the corresponding Tire Component classes which are closely related with the semi-empirical contact point model. Properties depicted in gray are simplified in the model.

to penetrate into the ground¹ and lift from it. More details can be found in Section 3.9 on Page 72 ff.

Until now all possible tire models would be totally rigid. To enable a certain deformation of the tire, the *Belt Dynamics* class is introduced. It allows a flexibility of the still rigid tire, defined by the *Geometry* class, related to the tire's hub. The part of the tire that would best suite the ideal geometry is the belt. Hence, this class is called *Belt Dynamics*. A description can be found in Section 3.10 on Page 77 ff.

The tire is now modularized into six classes including the *Rim* class which is not depicted in Figure 3.3, as it was shown in Figure 3.1. Section 3.2.1 explains the implemented version of this classes from the modelers point of view. It also explains why the final model of the tire contains seven classes, introducing a *Center to Contact Point* class. Section 3.8 on Page 68 ff describes *Center to Contact Point* class in detail.

All the classes mentioned above are calculating variables that are at least partially used by other objects for further computations. Quite a few of these variables are passed to other objects, therefore considerable effort was put into the development of a communication structure enabling the user to understand the connection between the

¹This is necessary to model a very simple variant of a contact patch although the tire, defined by the *Geometry*, on its own is ideally stiff.

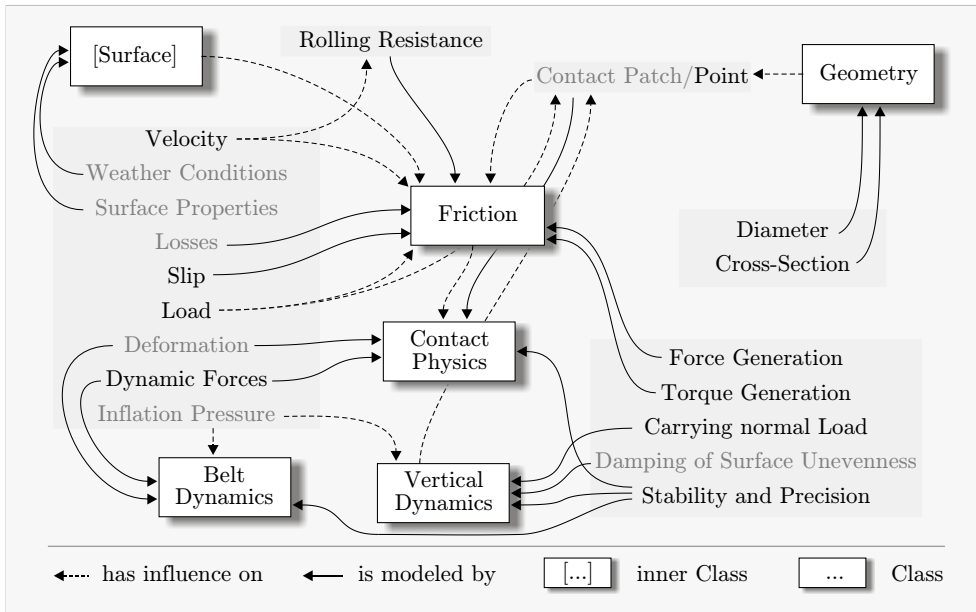


Figure 3.3: Final decomposition of the tire properties. Properties depicted in gray are simplified in the model.

objects just from looking at the graphical view of the models. The *Communication Structure* is explained in Section 3.3.

Still there is one class found in Figure 3.3 that has not yet been introduced. This is justified as it is an *inner* model and is not part of the actual tire model. It realizes uneven surfaces and enables a position-dependent friction coefficient. This model is described in detail in Chapter 5 on Page 101 ff.

3.2 Structure of the Tire and Library

The following sections are intended to give an broad overview of the implemented version of the tire model (Section 3.2.1) and the Modelica library (Section 3.2.2).

3.2.1 Tire Model Structure

The connection to the superordinate vehicle model is established by the three-dimensional frame named *Tire Hub* depicted in Figure 3.4². The *Tire Hub*, a standard element

²Taking a closer look at the *Center to CP* class, one can recognize rectangles surrounding the frames of the class. These are depicted by Dymola 6.1 because they are replaceable elements of the *Multi Bond Lib*.

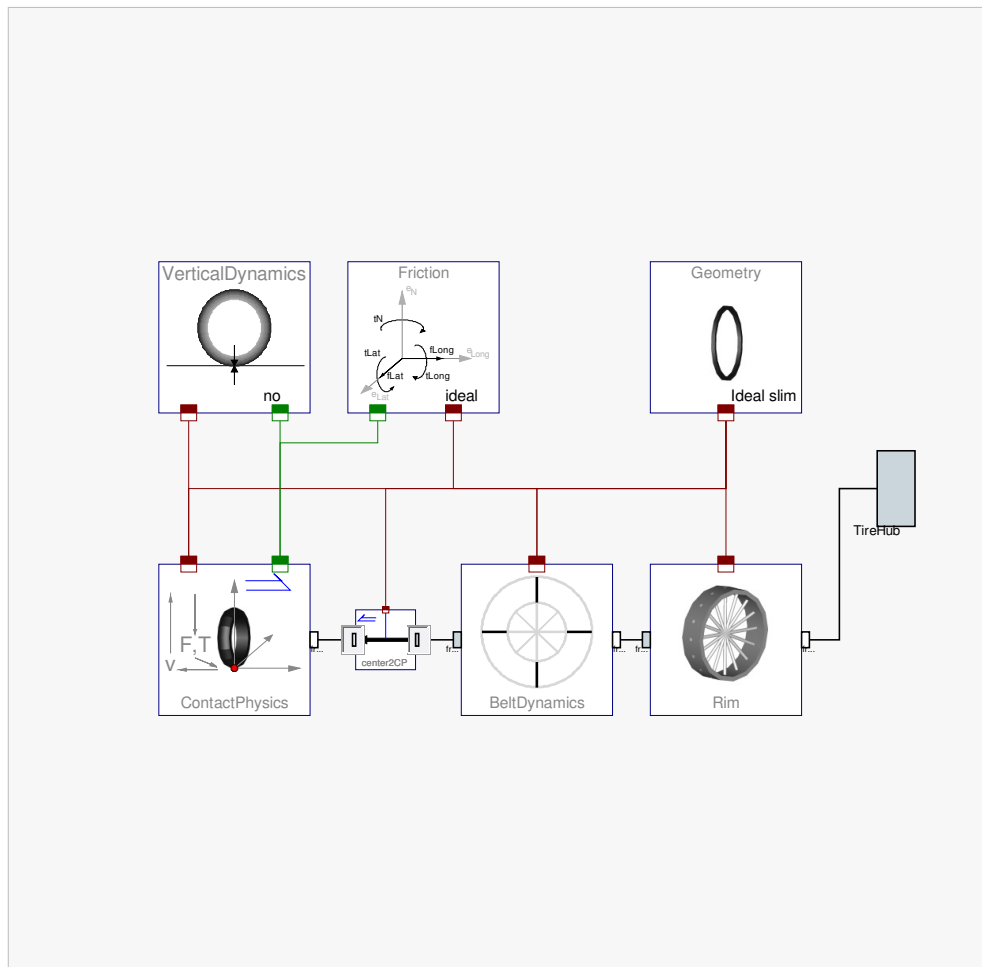


Figure 3.4: Model of the ideal tire showing the seven used objects and the communication structure on the top level.

of the *MultiBondLib* [Zim06], is rigidly connected to the model of the *Rim*. The rim's frame connected to the *Tire Hub* therefore models the center-point of the rim. The “output” of the *Rim* again models the center-point of the *Rim*, and is connected to the *Belt Dynamics* model. In this model the relative movement between the rigid belt and the rim can be described. The “output” of the *Belt Dynamics* is again positioned at the center-point of the belt. Therefore an element is needed to realize the translation from this point to the contact point. As this cannot be modeled by a standard element, the *Center To Contact Point* model has been created. It connects the *Contact Physics* model that applies forces and torques to the contact point. This lower part of the model represents the mechanically connected part of the model. The upper three classes are not directly connected by a mechanical connector, although the *Contact*

Point Connector implies kind of a mechanical connection. The *Vertical Dynamics* class determines if and how the tire is able to lift from the ground and how it responds to normal load. The *Friction* class determines the longitudinal and lateral forces as well as the torques that act on the contact point. Finally, the *Geometry* class determines the unitary vectors shown in Figure 2.4 and the position of the contact point depending on the actual geometry, position and orientation of the tire.

In order to be able to identify different tires easily, the visualization of the single parts is done in the corresponding objects. So e.g. the rim is visualized in the *Rim Class*, the tire's geometry is visualized in the *Geometry Class* and contact forces are visualized in the *Contact Physics* class. Therefore, changing the *Rim* class leads to a different visualization.

3.2.2 Library Structure

This section is intended to give a shallow overview of the library's structure. The top level packages of the library are depicted in Figure 3.5. The *Tire Components* package

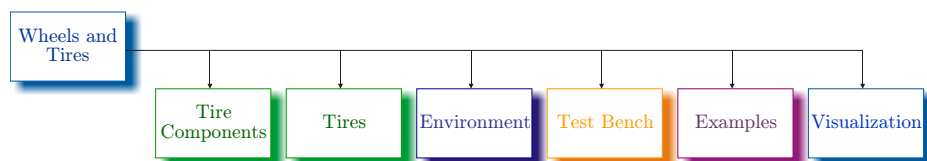


Figure 3.5: The library's top level packages.

covers the classes the *Tires* are built of. The *Tire Components* are described in detail in Sections 3.4 to 3.10. Both *Tires* and *Tire Components* are colored in green, in the library as well as in Figure 3.5, as they have the same basis. The *Environment* package contains the *Surface Base* as well as a exemplary implementation for even and uneven surfaces. It is described in Chapter 5 precisely. To check basic functionality of *Elevation* and *Friction* classes as well as the *Surface*, the *Test Bench* package has been created. The *Examples* package demonstrates how tire models can be used as part of a vehicle model. *Examples* and *Test Bench* are described in Chapter 6. Finally the *Visualization* package gathers a few models used to enable the animation of all necessary parts of the library.

A detailed overview of all models contained in the library can be found in Appendix A.1, showing the used icons and models names.

3.3 Communication Structure

The tire's objects compute a bunch of different variables of which some are used in different objects as well. Therefore, the objects have to be able to share variables among each other. Different types of communication structures have been implemented with varying quality of results, either having too many customized connectors or an overwhelming amount of connections on the top level.

For this reason a decision was made to create a bus connector called *TireBus* that gathers the variables used in most of the objects. For a better understanding of the model structure a further splitting ensures that not too many variables are directly on the top level of the bus. The division was made into *Unit Vectors*, *Contact Properties* and *Sensor Values* as shown in Figure 3.6. For each of these sets of variables, separate inputs and outputs have been created. This makes it possible to show in which objects variables are computed or just used. To conveniently connect these inputs and outputs (containing multiple variables) to the *Tire Bus* a record for each of these set of variables is created in the *Tire Bus*. This enables the user to connect the records directly instead of connecting each single variable, although this would be possible as well³. The icons used and an exact listing of which variables are contained in the different connectors can be found in Figure 3.6. As usual in Modelica the filled icons depict inputs, whereas the icons without a fill color depict outputs.

The second communication structure is the connector that contains the velocities as well as the forces and torques which act on the contact point. It is named *Contact Point Connector*. This connector is implemented in a a-causal manner due to the requirements of this connection. Hence, no inputs and output are defined here, because depending on the used classes there are either forces or velocities set and therefore the causality can change depending on the model. As well there is no second level that further splits the contained variables as there are just nine variables contained in the connector. Figure 3.7 shows the contained variables and the icons of the connector.

An exemplary top level view of the communication structure is shown in Figure 3.4 depicting the two different connectors. The very few different connectors make it straightforward for the first-time user to understand the structure of the tire model. Looking at the objects directly reveals a view shown in Figure 3.8. The user can easily identify that unit vectors are calculated in this class because UV is an output and that sensor values (SV) are used as an input. In addition the contact properties (CP) are calculated in the geometry class. The rhombuses placed in the shadowed area above the inputs

³This fact is useful in some cases, because e.g. in the advanced elevation models (see Figure 3.35 on Page 74) two inputs (CP and the input of the position element) would have to be connected directly (the Real `penetrationDepth` is ignored here), which causes Dymola to respond with an error message. The user can overcome this by connecting the input of the position element directly to the *Tire Bus*.

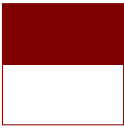
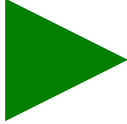
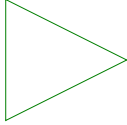
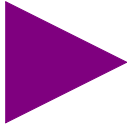
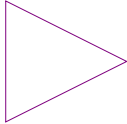
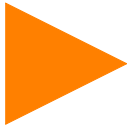
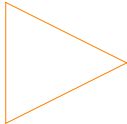
Top Level Icon	Second Level Icon	Contained Variables
	Unit Vector Signals (UV)	
	 	RealSignal eLong[3]; RealSignal eLat[3]; RealSignal eN[3]; RealSignal ePlane[3]; RealSignal eAxis[3];
	Contact Property Signals (CP)	
	 	RealSignal xCP[3]; RealSignal rCP[3]; RealSignal lCR; RealSignal bCR; RealSignal penetrationDepth; BooleanSignal Contact[3];
	Sensor Value Signals (SV)	
	 	RealSignal eAxis[3]; RealSignal RBelt[3,3]; RealSignal wBelt[3]; RealSignal xBelt[3];

Figure 3.6: The elements contained in the *Tire Bus*.


Top Level Icon	Contained Variables
	<pre> RealSignal vLong; RealSignal vLat; RealSignal vN; RealSignal fLong; RealSignal fLat; RealSignal fN; RealSignal tLong; RealSignal tLat; RealSignal tN; </pre>

Figure 3.7: The elements contained in the *Contact Point Connector*.

and outputs define the variables that are used in this model. Utilizing the real connectors in this way, one does not have to declare variables in the code separately and the user can identify which variables are really used in the model. This is important due to the fact that not every variable from the *Tire Bus* has to be used in a model connected to the bus. Moreover for a graphically built model, connecting them is very convenient.

The communication structure and graphical declaration of variables is done with protected items because they would be visible in the icon at the top level otherwise. As well, this way the user can decide which variables should be shown in the simulation window directly. Moreover every variable contained in the *Tire Bus* can be shown in the simulation tab because the *Tire Bus* is a public element in every class of the tire.

One thing that has to be mentioned about the communication structure is that lots of vectors are used inside of different connectors. Connecting them in the sub-classes is a little inconvenient, as Dymola 6.1 does not recognize that the variables used in the bus are vectors. Therefore, connecting a `Real[3]` to a vector in the bus has to be done in two steps. Therefore, the connection is done graphically first with the typical Dymola window appearing when connecting vectored variables. In the window, one has to choose a single item from the vector of the `Real[3]`. That will make Dymola accept the connection. Then switching to the code view reveals some kind of equation like

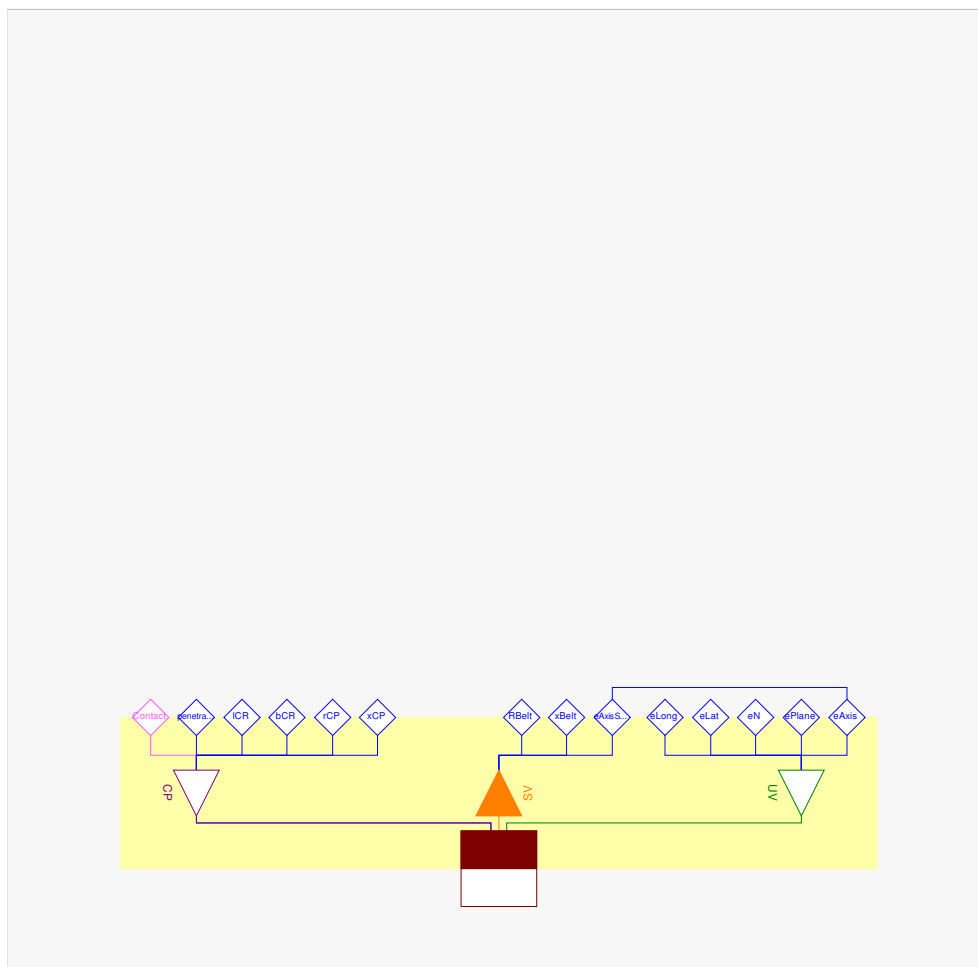


Figure 3.8: *Geometry Base* model showing the communication on the second level of modeling.

```
connect(eLong[1], UV.eLong)
```

This equation then has to be manually changed into

```
connect(eLong[:], UV.eLong[:])
```

to make the connection work. The reason for this behavior seems to be that Dymola does not recognize the variable in the bus to be a vector. This could be caused by a wrong definition in the records which are the base for the connectors, but during this work no better solution could be found, so this workaround has to be applied.

3.4 Rim Class

The *Rim Class* is a rather simple model, as the rim can be modeled ideally just consisting of a body that has a mass and an inertia tensor (see figure 3.9). The second thing the model does is measure the unit vector `eAxis` directly from `frame_a`. This unit vector is then used in the *geometry class* to calculate the other unit vectors used in the model. In order to describe the rim (as well as the belt dynamics presented in Section 3.10) a generally applicable approach was considered to be the usage of the center-points as “in- and outputs” to the model.

The second important thing the class does is visualizing the rim. This way it is possible to create different rims that feature different looks. No more than one rim is realized in the current version of the library but it could be done easily.

3.5 Friction

The following chapter is intended to explain the function of the friction class and its subcomponents. For the sake of object-oriented modeling, the different modeled effects are put into subclasses (see Figure 3.10 as an example) that are of different complexity. All of the featured classes which model a certain type of effect are extended from the corresponding base class, that ensures the necessary output to be calculated. This as well ensures that all models stay exchangeable not depending on the implementation of the modeled effect.

The simplest model is always a class which just sets the corresponding output to a constant value. These are useful if the overall model should not consider this effect. If a tire shall not have a bore torque because it is unnecessary in that application. Additionally, there are one or more models which apply different effects in various levels of detail. The modeled effects are the following.

- Slip Properties
- Roll Resistance
- Bore Torque
- Load Influence
- Camber Force
- Overturning Torque
- Self Aligning Torque

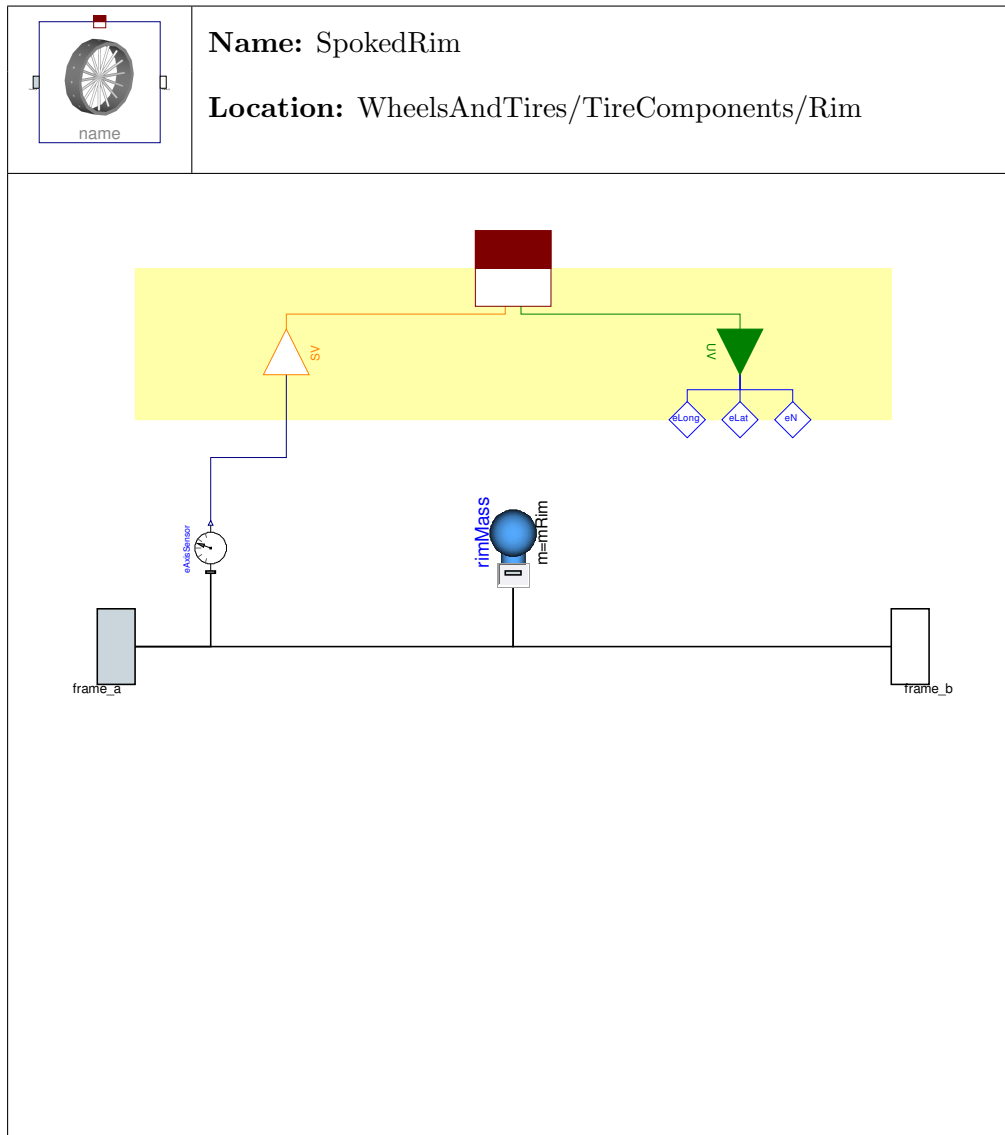


Figure 3.9: Model of an ideal rim containing mass and inertia tensors.

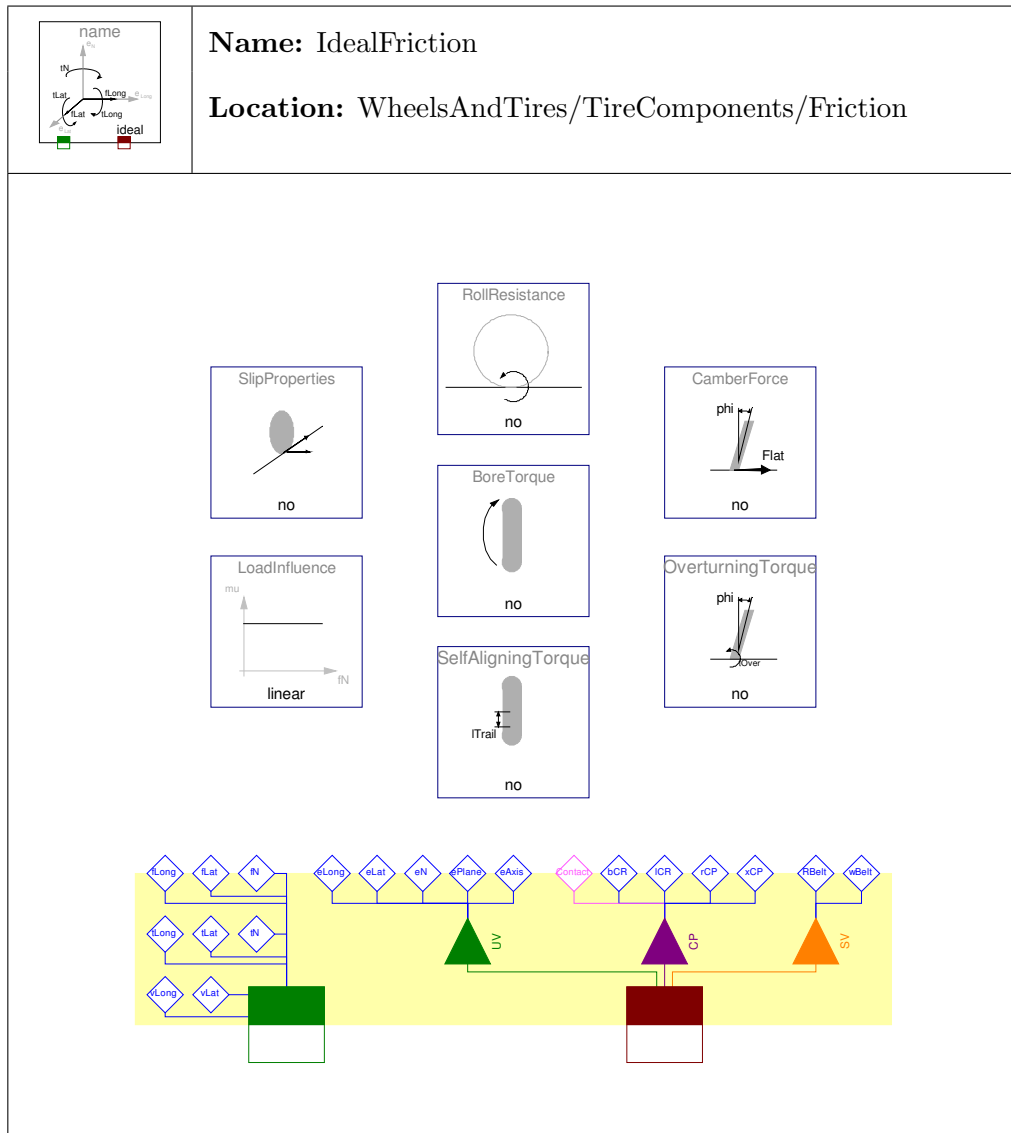


Figure 3.10: Model of an ideal frictional behavior.

All of these models are combined in a base model that is called *Friction Base*. Common variables are computed here and it provides the `inner/outer` framework for the communication among the sub-models. The following sections will describe the different classes in more detail.

3.5.1 The Friction Base Class

The base class of the friction is shown in Figure 3.11. Here the user can see that the class uses variables from the *Tire Bus* (`UV`, `SV` and `CP` are inputs) to calculate frictional forces and torques. To be able to calculate either forces or set velocities, it is important that the *contact point connector* on the left is a-causal.

The friction base class computes many of the variables needed in the sub-classes which are added to determine the frictional behavior of the tire. The rest of this section shows how these are calculated exactly and what type of limitations they imply.

The slip velocities are computed by the following equations. See Figure 2.6 on Page 15 for a partial graphical representation.

$$v_{SlipLong} = v_{RotLong} + v_{Long} \quad (3.1)$$

$$v_{SlipLat} = v_{Lat} \quad (3.2)$$

$$v_{Slip} = \sqrt{v_{SlipLong}^2 + v_{SlipLat}^2} \quad (3.3)$$

The translational velocities used in the equations above are determined by the following equations. Whereas the index of the speeds `Rot` stands for velocity due to angular velocities.

$$v_{Rot} = \omega_{Belt} \times r_{CP} \quad (3.4)$$

$$v_{RotLong} = v_{Rot} \cdot e_{Long} \quad (3.5)$$

$$v_{RotLat} = v_{Rot} \cdot e_{Lat} \quad (3.6)$$

$$v_{RotPlane} = v_{Rot} \cdot e_{Plane} \quad (3.7)$$

Whereas ω_{Belt} was used instead of ω_{Rim} due to the fact that models presented later can result in a ω_{Belt} different from ω_{Rim} .

From these quantities the slip values can be calculated⁴. The equations are formulated in a conditional way due to the fact that for wheels that do not turn ($v_{RotLong} = 0$ or

⁴There are several different definitions of slip. The one used by G. Rill in [Ril07] was used in this implementation in a slightly modified version.

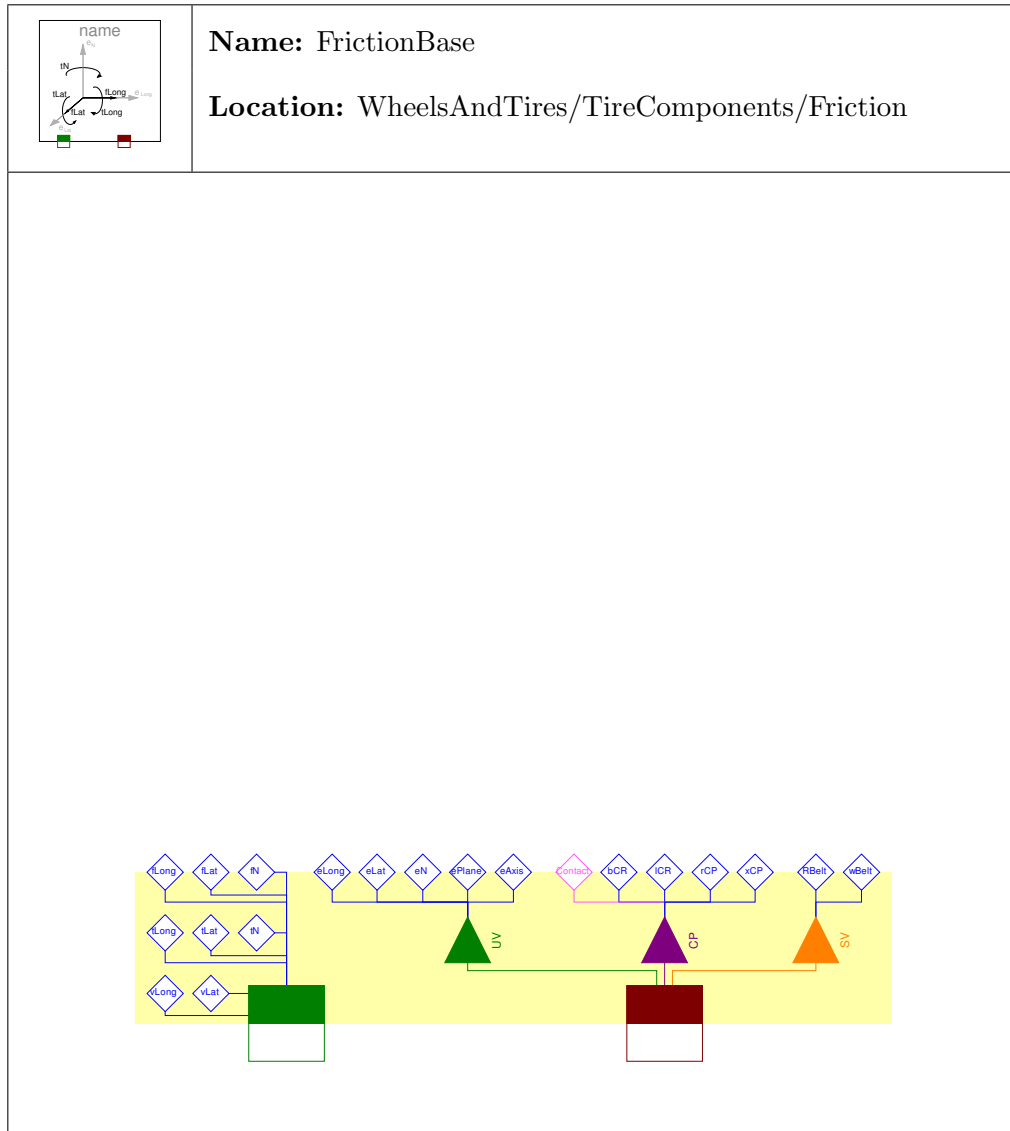


Figure 3.11: Friction Base class defining in- and outputs as well as some of the used variables.

$\omega_{Rot} = 0$), the slip becomes infinite. Therefore, the following equations are used.

$$s_{Long} = \begin{cases} v_{SlipLong}/|v_{RotLong}| & \text{if } |v_{RotLong}| > |v_{RotMin}| \\ v_{SlipLong}/|v_{RotMin}| & \text{else} \end{cases} \quad (3.8)$$

$$s_{Lat} = \begin{cases} -v_{SlipLat}/|v_{RotLong}| & \text{if } |v_{RotLong}| > |v_{RotMin}| \\ -v_{SlipLat}/|v_{RotMin}| & \text{else} \end{cases} \quad (3.9)$$

$$s = \sqrt{s_{Long}^2 + s_{Lat}^2} \quad (3.10)$$

Whereas v_{RotMin} is a parameter value with a default value of $10^{-6}ms^{-1}$. The influence of this limitation can be seen in Figure 3.15(b) on Page 40.

In [Ril07], a different approach is used to get rid of the division by 0. Instead of the border introduced in Equations 3.8 and (3.9), v_{RotMin} a term small compared to the pitch angle velocity, is added to the denominator resulting in the equation

$$s = \frac{v}{R \cdot |\Omega| + v_{Num}} \quad (3.11)$$

with v being the sliding velocity, R the tires' radius and $|\Omega|$ the absolute rotational velocity. $v_{Num} \ll R|\Omega|$ should be valid throughout the simulation. However, the limitation in the implemented way is more suitable for the Modelica environment.

Section 3.5.2.5 introduces the variables v_{Ad} and v_{Sl} that are directly connected to s_{Ad} and s_{Sl} in a similar way as shown in Equations (3.8) and (3.9). As they are introduced due to the friction model described in this section, they are explained there.

Another value that is computed in the `frictionBase` class is the slip angle. It is determined in the following way:

$$\alpha = \begin{cases} \arctan(s_{Lat}) & \text{if } |v_{RotLong}| > |v_{RotMin}| \\ \frac{\pi}{2} & \text{else if } v_{Lat} < 0 \\ -\frac{\pi}{2} & \text{else} \end{cases} \quad (3.12)$$

Finally, the effects which arise due to the sub-models are considered. This is done by either adding output values to the originally calculated ones (for the sliding velocity due to camber angle) or by setting the output to the calculated value (for the calculated torques due to overturning, bore effects, self aligning and roll resistance). The relevant

equations for this are:

$$v_{SlipLat} = v_{Lat} + v_{LatCamber} \quad (3.13)$$

$$t_{Long} = t_{Overturn} \quad (3.14)$$

$$t_N = t_{Bore} + t_{Align} \quad (3.15)$$

$$t_{Lat} = t_{Roll} \quad (3.16)$$

For a graphical representation of the torques and the slip angle, see Figure 2.4 on Page 13.

Note that Equation (3.13) replaces Equation (3.2) in the final model. This adds a force that arises due to lateral displacement of parts traveling through the contact area. The effect is described more precisely in Section 3.5.5 on Page 48.

3.5.2 Slip Properties

This section is intended to describe models relating slip or sliding velocity to the forces acting on the contact point. There are basically three different approaches to calculate contact forces. One is to set the sliding velocity to 0, the second to use slip as a base, the last is to use sliding velocities. All have advantages over the other which will be explained in the following sections. One approach is shown that combines the advantages of the latter both.

3.5.2.1 No Slip

The simplest way to model a tire's slip behavior is to assume an ideal tire having no slip at all. This is done in the *No Slip* class by defining the following equations.

$$v_{SlipLong} = 0 \quad (3.17)$$

$$v_{SlipLat} = 0 \quad (3.18)$$

This introduces two holonomic constraints that result in the two contact point forces **fLat** and **fLong**. This is a quite similar situation as with the *No Elevation* class described in Section 3.9.1 on Page 72 which results in **fN**.

One property of the model is that it does not take into account the variable **Contact** that is a boolean variable defining if the tire has contact to the ground (**Contact = true**) or not. So in the event that a model, which allows the tire to lift from the ground, is chosen in the *Vertical Dynamics* class and the tire really does lift from the ground, it can still transmit forces although it has no contact to the ground. That property is not a big issue, because the combination of a model with the ability to lift from the ground

but without slip is one that would usually not be simulated, as usually the slip is one of the first things that is modeled. Additionally every model that imposes slip and sliding velocity depends on the normal force. If there is no contact, the normal force will be zero and therefore no longitudinal or lateral force can be generated, independent of the contact variable.

3.5.2.2 Sliding Dry Friction Longitudinal

This model utilizes the properties of the *No Slip* model for the lateral force generation but allowing slip in the longitudinal direction. Therefore, the same that is valid for the *No Slip* class is true for the lateral direction of the *Sliding Dry Friction Longitudinal* class. Lateral forces can be transmitted if there is no contact to the ground.

The generation of the longitudinal force is based on Coulomb's law of dry friction. Therefore, the equation for the equation used to calculate the longitudinal force is quite simple:

$$f_{Long} = -f_N \cdot \mu(v_{Slip}) \quad (3.19)$$

What makes that model meaningful for the modeling of tires is that the friction coefficient μ depends on the (overall) sliding velocity. Otherwise that model would not represent a tire accurately. The function that is used to calculate the friction coefficient depending on the sliding velocity is shown in [Ril07] and [ZO09]. It is a curve determined by a continuous interpolation between characteristic points $v_{Adhesion}/\mu_{Max}$ and v_{Slide}/μ_{Min} shown in Figure 3.12. One more parameter in the used interpolation

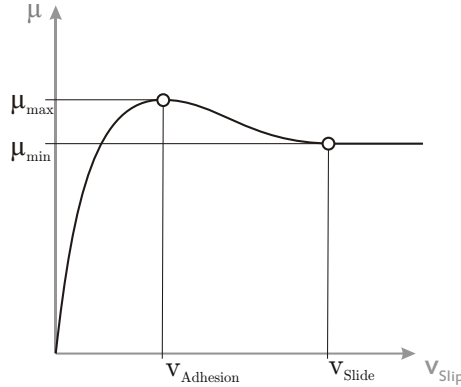


Figure 3.12: The frictional coefficient μ depending on the sliding velocity v_{Slip} showing the two descriptive points.

is the slope of the curve in the origin. The slope $d\mu/dv_{Slip}$ is a final parameter and is

set to $2/v_{Adhesion}$ as this one is usually not that important and difficult to determine for the average user.

The used equations for this piecewise polynomial interpolation are the following.

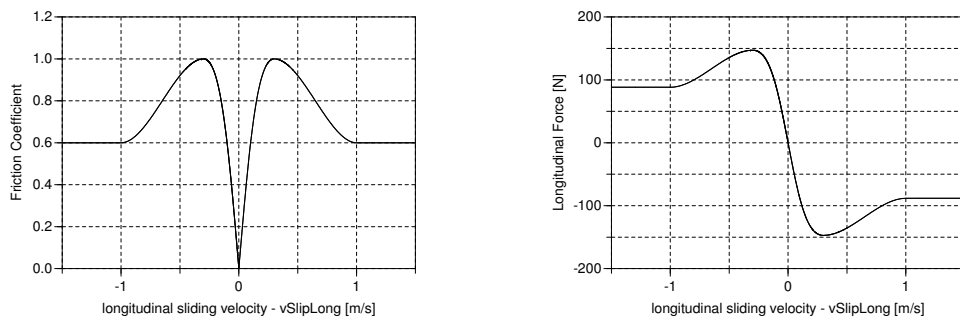
$$\mu(v_{Slip}) = \begin{cases} \frac{v_{Slip} \cdot \dot{\mu}_{vSlip}(0)}{1 + \sigma \left(\frac{v_{Adhesion}}{\mu_{max}} \dot{\mu}_{vSlip}(0) - 2 + \sigma \right)} & \text{if } v_{Slip} \leq v_{Adhesion} \\ \mu_{max} - (\mu_{max} - \mu_{min})\tau^2(3 - 2\sigma) & \text{if } v_{Adhesion} < v_{Slip} < v_{Slide} \\ \mu_{min} & \text{else} \end{cases} \quad (3.20)$$

with

$$\sigma = \frac{v_{Slip}}{v_{Adhesion}} \quad (3.21)$$

$$\tau = \frac{v_{Slip} - v_{Adhesion}}{v_{Slide} - v_{Adhesion}} \quad (3.22)$$

The result of this interpolation can be seen in Figure 3.13(a), where one can also clearly



(a) The friction coefficient in dependence on the longitudinal slip velocity.

(b) The longitudinal force in dependence on the longitudinal slip velocity.

Figure 3.13: The dependence of force generation in the longitudinal direction on the sliding velocity.

recognize the two characteristic points of the curve. These are the parameters of the

model which were set to

$$v_{Adhesion} = 0.3 \frac{m}{s} \quad (3.23)$$

$$v_{Slide} = 1 \frac{m}{s} \quad (3.24)$$

$$\mu_{maxN} = 1 \quad (3.25)$$

$$\mu_{minN} = 0.6 \quad (3.26)$$

for this simulation. The “ N ” in the indexes indicates that these are nominal values because the friction coefficient can change in dependence of the normal load. This fact is explained more closely in Section 3.5.6 on Page 50. The simulation also results in a longitudinal force shown in Figure 3.13(b) that is the output of Equation (3.19) with a normal force that results from 15kg weight.

Regarding the implementation of Equation (3.20), it has to be mentioned that it was done in a way more suitable for the computation. The function `get_mu_vSlip` (that computes the interpolation) does not directly return the friction coefficient μ but rather the quotient μ/v_{Slip} . The reason for that can be found in the way the force is actually computed. The way it is implemented differs a little from the mentioned Equation (3.19), due to the fact that the direction the force acts in has to be considered. Therefore, and for similarity with the other classes based on the sliding velocity, the following equation has been used.

$$f_{Long} = \begin{cases} -f_N \cdot \left(\frac{\mu}{v_{Slip}} \right) v_{SlipLong} & \text{if } \text{Contact} \\ 0 & \text{else} \end{cases} \quad (3.27)$$

In this case v_{Slip} and $v_{SlipLong}$ are always the same values because there is no slip in lateral direction, it does not matter which one is taken to multiply with. Of greater importance is that in the implementation the fraction μ/v_{Slip} is one variable of type `Real`⁵ which is named `mu_vSlip`. Therefore the Equation (3.20) was manually divided by v_{Slip} directly returning the term (μ/v_{Slip}) . This moves the division from Equation (3.27) to (3.20), which overcomes the problem with the division by zero as the term v_{Slip} cancels out in the case of $v_{Slip} \leq v_{Adhesion}$. For the other terms, the resulting division by v_{Slip} does not matter as v_{Slip} cannot be 0 in these cases. This way, Equation (3.27) cannot result in a division by 0 and becomes generally usable.

It is important to notice, that the `if` statement accounting for the `Contact` variable that would not be necessary in general, because the normal force f_N is 0 when there is

⁵The reason for using type `Real` without any unit is that the function gets called by models based on sliding velocities and slip as well. Therefore, it would have to return different units depending on the type of class calling it. This would be possible to realize with an additional input to the function but better usability is considered to be more important.

no contact resulting in f_{Long} and f_{Lat} to obtain 0. Still, it has been implemented this way to ensure that the `Contact` variable can have influence on the simulation result not depending on the normal force.

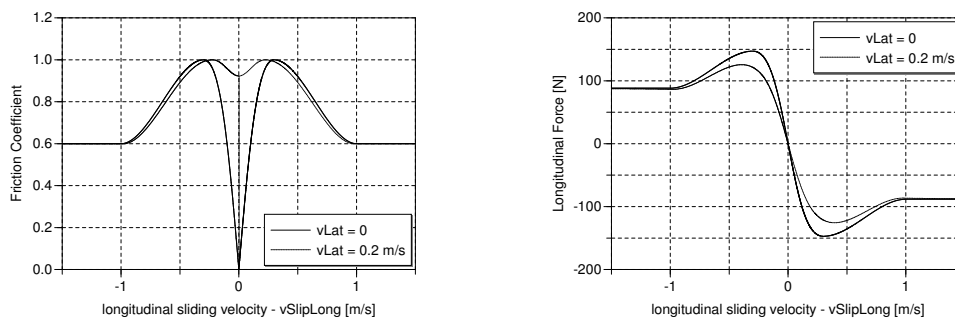
As the friction coefficient is not directly calculated because the variable `mu_vSlip` has been used, the equation

$$\mu = \left(\frac{\mu}{v_{Slip}} \right) v_{Slip} \quad (3.28)$$

calculates the friction coefficient to display it in the simulation results.

3.5.2.3 Sliding Dry Friction

The main difference between the *Sliding Dry Friction* and the foregoing *Sliding Dry Friction Long* is that for the former, lateral sliding velocity is possible as well. The characteristics for the friction are the same in principle (see Figure 3.13), but one has



(a) The friction coefficient in dependence of the overall sliding velocity, with and without lateral slip of 0.2ms^{-1} .

(b) The longitudinal force in dependence of the overall sliding velocity, with and without lateral slip of 0.2ms^{-1} .

Figure 3.14: The dependence of force generation on the overall sliding velocity with two different cases shown.

to keep in mind that a lateral slip velocity adds up to the total sliding velocity in a quadratic sense as shown in Equation (3.3) on Page 30. Therefore, a lateral slip has an effect on overall coefficient in the longitudinal direction as well.

This effect is shown in Figure 3.14, where a simulation of two different cases was carried out. The one is the same as shown in Figure 3.13 on Page 35. The second one adds a lateral sliding velocity of 0.2ms^{-1} which results in the corresponding curves. In 3.14(b), one can see that the force available for the longitudinal force is reduced by the lateral sliding velocity. In 3.14(a) the friction coefficient is shown. It does not reach 0 in the

case of $v_{Lat} = 0.2ms^{-1}$ due to the fact that the overall slip velocity cannot get $0ms^{-1}$ with $v_{Lat} \neq 0ms^{-1}$ during the simulation. The “jump” of the friction coefficient at $v_{SlipLong} = 0ms^{-1}$ has its reasons in the simulation. The lateral sliding velocity is realized by a ramp function, hence the friction coefficient raises with the lateral sliding velocity from 0 to a value above 0.9 at $v_{SlipLong} = 0ms^{-1}$.

The implementation is a consequential extension to the one in *Sliding Dry Friction Long*. The equations used are:

$$f_{Long} = \begin{cases} -f_N \left(\frac{\mu}{v_{Slip}} \right) v_{SlipLong} & \text{if Contact} \\ 0 & \text{else} \end{cases} \quad (3.29)$$

$$f_{Lat} = \begin{cases} -f_N \left(\frac{\mu}{v_{Slip}} \right) v_{SlipLat} & \text{if Contact} \\ 0 & \text{else} \end{cases} \quad (3.30)$$

3.5.2.4 Rill Friction

The major difference of the *Rill Friction* to the foregoing classes is that it depends on the slip rather than the sliding velocity. The results of this difference are explained in Section 3.5.2.5. Another difference is that friction properties for longitudinal and lateral friction can separately be set by parameters. These are used during simulation to determine the frictional properties in every slipping situation. The equations are

$$s_{Adhesion} = \sqrt{(\cos(\alpha) \cdot s_{LongAdhesion})^2 + (\sin(\alpha) \cdot s_{LatAdhesion})^2} \quad (3.31)$$

$$s_{Slide} = \sqrt{(\cos(\alpha) \cdot s_{LongAdhesion})^2 + (\sin(\alpha) \cdot s_{LatAdhesion})^2} \quad (3.32)$$

$$\mu_{MaxN} = \sqrt{(\cos(\alpha) \cdot \mu_{LongMaxN})^2 + (\sin(\alpha) \cdot \mu_{LatMaxN})^2} \quad (3.33)$$

$$\mu_{MinN} = \sqrt{(\cos(\alpha) \cdot \mu_{LongMinN})^2 + (\sin(\alpha) \cdot \mu_{LatMinN})^2} \quad (3.34)$$

$$\frac{\mu}{s_N} = \sqrt{\left(\cos(\alpha) \cdot \frac{\mu}{s_{LongN}} \right)^2 + \left(\sin(\alpha) \cdot \frac{\mu}{s_{LatN}} \right)^2} \quad (3.35)$$

taken from [Ril07].

The interpolation for the friction coefficient is the same one that was used in the foregoing classes. It does not matter whether it is used based on sliding velocities or the slip. So again the actual value of μ has to be computed from the term (μ/s) by

multiplying with s . For the actual forces, the equations

$$f_{Long} = \begin{cases} -f_N \left(\frac{\mu}{s}\right) s_{Long} & \text{if Contact} \\ 0 & \text{else} \end{cases} \quad (3.36)$$

$$f_{Lat} = \begin{cases} -f_N \left(\frac{\mu}{s}\right) s_{Lat} & \text{if Contact} \\ 0 & \text{else} \end{cases} \quad (3.37)$$

are used, which take into account the overall slip.

3.5.2.5 Combined Friction

The *Combined Friction* class is named this way because it combines the advantages that arise from the usage of the sliding velocity and the ones that arise from the slip as a base for the frictional forces. The method was published in a paper by the DLR [ZO09]. As it can be seen easily from Figure 3.15 on Page 40 there are essential differences between the models that base on the sliding velocity which were named “dry sliding” models and the one based on the slip. The former does not change its behavior with the pitch angle velocity and is therefore very stable but valid only for a small velocity range. The model used by Rill in [Ril07] is better suited for higher rolling velocities but gets stiff at low pitch angle velocities. The stiffness was regarded to by the limitation of the slip (see Equation (3.10)) with the result visible in Figure 3.15(b).

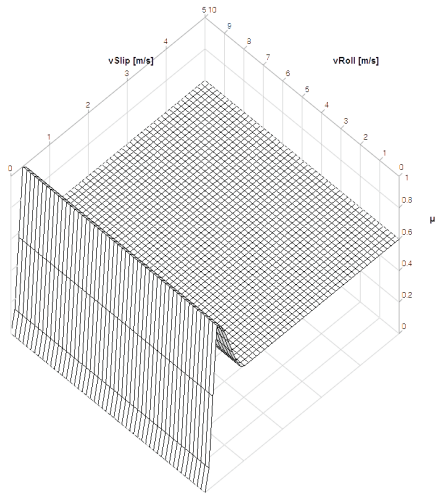
In [ZO09], a possibility is shown to combine the advantages of these two models based on the function `softMaxTol`. This combines the stability of the first (Figure 3.15(a)) approach with the accuracy of the second approach (Figure 3.15(b)). The result of this smooth transition between the two models is shown in Figure 3.15(c). The influence of the parameter `tol` on the transition between the two models is depicted in Figure 3.16. Values that are much bigger or smaller than the used ones do not provide useful results. The `softMaxTol` function is implemented in the following way

$$\text{softMaxTol}(a, b) = \log \left(e^{a/tol} + e^{b/tol} \right) \cdot tol \quad (3.38)$$

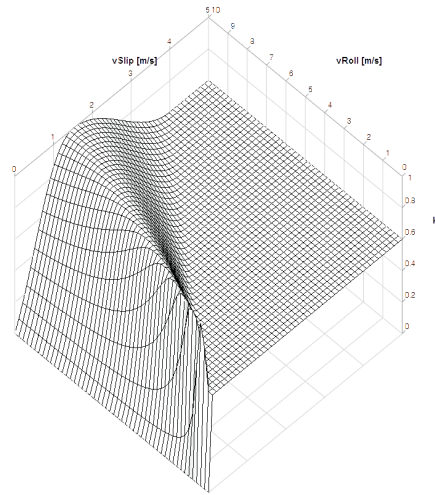
with `tol` = 0.1 by default.

It is silently assumed that μ_{max} and μ_{min} are the same for both models, which is arguable because the actual process of rolling should not influence these attributes drastically. Also the values for `vAd` and `vSl` can just rise with increasing pitch rotation velocity [ZO09].

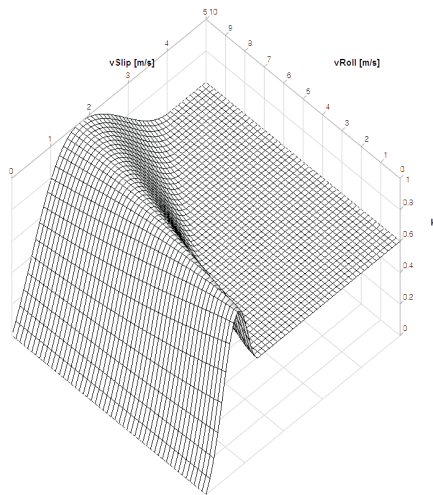
The resulting velocities at which adhesion and full sliding occur now depend on the actual driving situation, therefore they cannot be described by a simple parameter



(a) Based on the sliding velocity.



(b) Based on the slip.



(c) Based on a combination of both.

Figure 3.15: The friction coefficient in dependence on the overall sliding velocity and the velocity due to pitch angle rate.

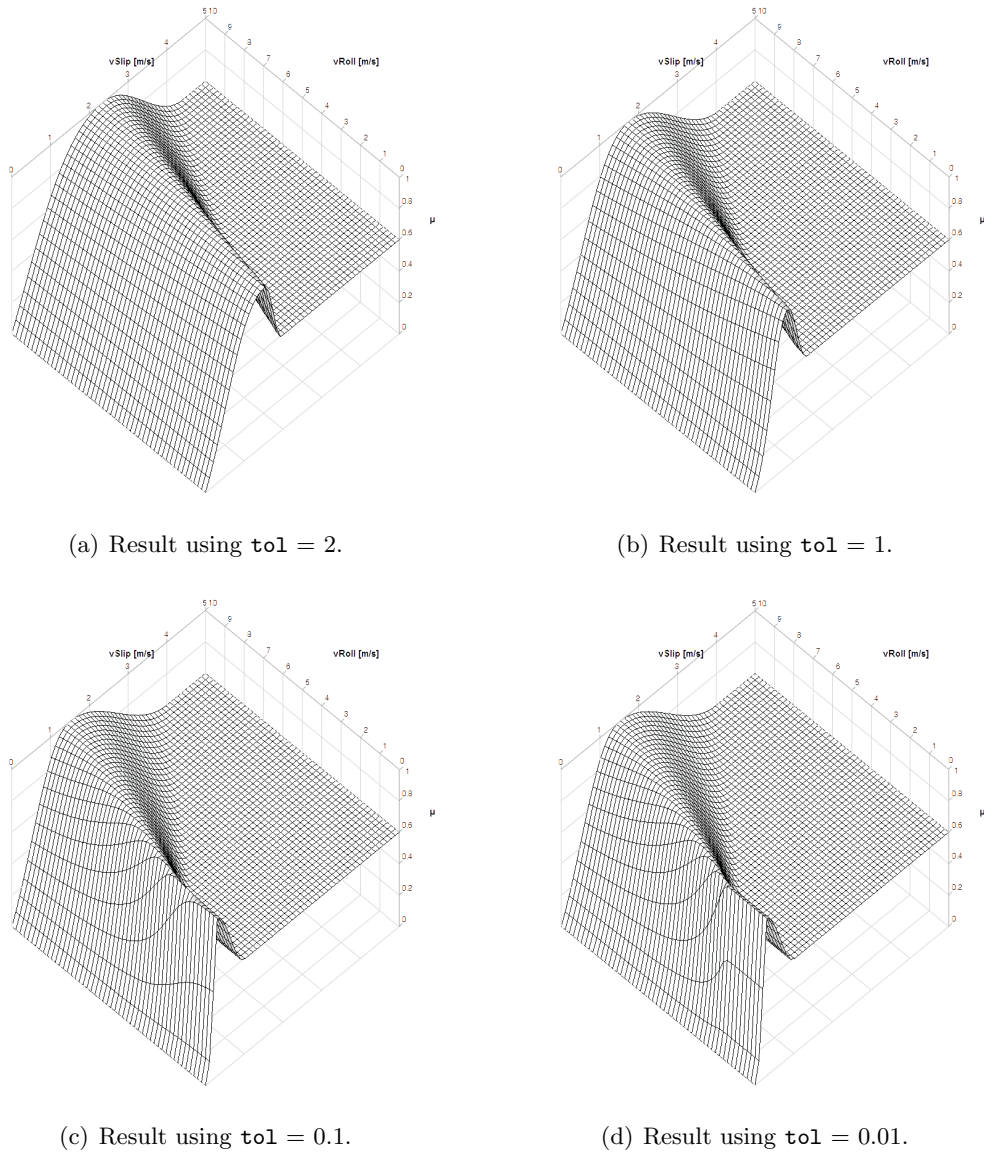


Figure 3.16: The influence of the parameter tol on the transition between the two frictional models.

anymore. Therefore, two new variables `vAd` and `vSl` are introduced. These describe the velocity of adhesion (`vAd`) and the one of full sliding (`vSl`). These two variables are necessary for the calculation of the actual friction coefficient by the `get_mu_vSlip` function and for other models like e.g. the bore torque. Generally, throughout all models these variables can be used for comparison whether slipping occurs or not. Therefore, all the foregoing models have set these variables to the corresponding values which are parameters there. The *Combined Friction* model is the only one to have changing values for `vAd` and `vSl`. For all the other models, these values do not change during the simulation time⁶.

The *Rill Friction* is based on the slip and future models may as well need slip values of adhesion (`sAd`) and full sliding (`sSl`) as well. Therefore the `frictionBase` class features two equations that compute `sAd` and `sSl` from `vAd` and `vSl` in a similar fashion to Equation (3.8) and (3.9) on Page 32.

With this as a base, it is possible for models to get information whether the tire is sticking or sliding by comparing with the presented values. This is possible with every type of friction model independent of the type of friction model used.

3.5.3 The Rolling Resistance

The rolling resistance arises due to an asymmetric pressure distribution in the tread shuffle and a number of other effects that shall not be mentioned here. For a more detailed description, the reader is referred to [MW03]. The result of these effects is that the normal force f_N does not act in the middle of the contact area but a certain distance in front of it. This positional shift in longitudinal direction and the normal force result in a torque that tries to slow down the movement of the tire. For reasons of generality, the measured positional shift x_R is usually divided by the tires radius r resulting in a friction coefficient f_R .

$$f_R = \frac{x_R}{r} \quad (3.39)$$

It is important to notice that no force has to be transmitted via the contact point in order to make the rolling resistance slow down the tire. Therefore, it is possible to have a decreasing tire velocity for a freely rolling wheel on a surface with a friction coefficient of $\mu = 0$.

⁶Models based on slip have constant `sAd` and `sSl`, models based on the sliding velocities have constant `vAd` and `vSl`, with the respectively other one changing depending on the pitch angle velocity .

3.5.3.1 Roll Resistance Base and No Rolling Resistance

The partial base class ensures t_{Roll} to be an output of every model with *No Roll Resistance* setting it to 0 in the event that the rolling resistance should not be part of the simulation.

3.5.3.2 Constant Rolling Resistance

The constant rolling resistance is specified by one parameter f_{R0} that is the relative friction coefficient defined in Equation (3.39). The index 0 indicates that it is not velocity dependent. In Figure 3.17, the result of the model is shown. The torque does

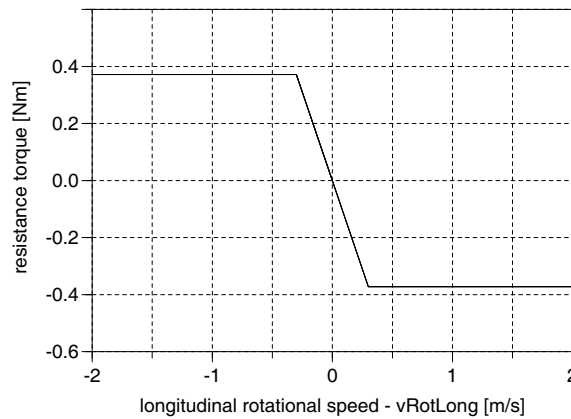


Figure 3.17: Constant roll resistance torque depending on the pitch angle velocity.

not rise with growing velocity, but a limitation has been added if the velocity gets lower than the velocity of adhesion v_{Ad} to avoid step like change at the zero crossing. The used equation is

$$t_{Roll} = \begin{cases} -f_N \cdot f_{R0} \cdot r_{Wheel} \cdot \text{sign}(v_{RotLong}) & \text{if } |v_{RotLong}| > v_{Ad} \text{ AND Contact} \\ -f_N \cdot f_{R0} \cdot r_{Wheel} \cdot \frac{v_{RotLong}}{v_{Ad}} & \text{if Contact} \\ 0 & \text{else} \end{cases} \quad (3.40)$$

with the result shown in Figure 3.17.

3.5.3.3 Speed Depending Rolling Resistance

This model improves the accuracy of *Constant Rolling Resistance* by the two additional parameters f_{R1} and f_{R4} whereas the number indicates the order of term it is used in. In this model, f_R is dependent on the speed in the following way.

$$f_R = f_{R0} + f_{R1} \cdot \frac{|v_{RotLong}|}{100/3.6} + f_{R4} \cdot \left(\frac{|v_{RotLong}|}{100/3.6} \right)^4 \quad (3.41)$$

Whereas the denominator 100/3.6 indicates that the velocity v is divided by 100km/h in [MW03].

The speed dependent factor f_R computed by equation (3.41) is then used instead of f_{R0} in Equation (3.40). This type of model for the rolling resistance as well as the default parameters are shown in [MW03]. Figure 3.18 shows the rising rolling resistance in

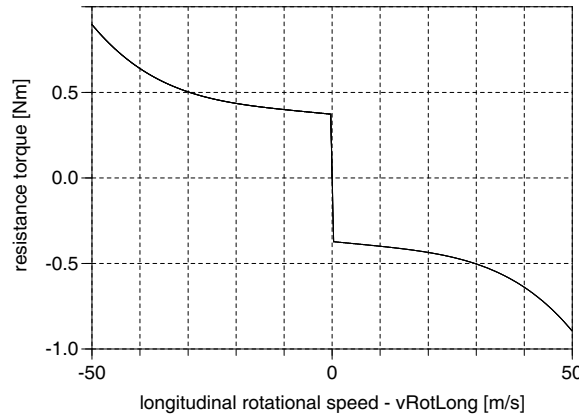


Figure 3.18: Speed dependent roll resistance torque depending on the pitch angle velocity.

dependence of the velocity. The area close to the zero crossing applies the same linear dependences below the velocity of full adhesion as shown in Section 3.5.3.2.

There is no quadratic term in the equation because the quadratic term of the frictional force acting on a vehicle is usually dominated by the aerodynamic forces, not by the tire.

3.5.4 Bore Torque

Bore torque appears when the tire is rotated around its vertical axis at a yaw rate $\neq 0$. In the contact area, parts are sticking to the ground and therefore a kind of

spring has to be stretched to get the tire spinning until the parts of the tread shuffle start sliding. The bore torque is the physical quantity describing its characteristic. It always counteracts this rotating movement. For a more detailed description, the reader is referred to [Ril07]

3.5.4.1 Bore Torque Base and No Bore Torque

The *Bore Torque Base* is a partial class ensuring, that every class extending it has `tBore` (the bore torque) as an output variable. *No Bore Torque* is a simple class defining the output `tBore` to be zero. So this class is to be used if there is no bore torque shall be modeled.

3.5.4.2 Linear Bore Torque

The linear bore torque model is a modified version of the one is shown in [ZO09]. It has been modified to better suit friction models that do not use `vAd` and `vSl` as a base. The equations that were used are the following.

$$t_{Bore} = \begin{cases} -f_N \cdot \text{loadInfluence} \cdot \mu_{Turn} \frac{1}{16} l_{CR}^2 + b_{CR}^2 \cdot \frac{\omega_{Plane}}{v_{RotLong}} \\ \quad \text{if } |\omega_{Plane}| \frac{1}{4} \sqrt{l_{CR}^2 + b_{CR}^2} < |v_{RotLong}| \text{ AND Contact} \\ -f_N \cdot \text{loadInfluence} \cdot \mu_{Turn} \frac{1}{4} \sqrt{l_{CR}^2 + b_{CR}^2} \cdot \text{sign}(\omega_{Plane}) \\ \quad \text{else if Contact} \\ 0 \text{ else} \end{cases} \quad (3.42)$$

The linearly rising torque is calculated in the `if` part of Equation (3.42). The `else` part takes care of the limitation to the maximum torque with a changing border. This limitation is realized by dividing the actual equation for the depending torque by the condition that of the `if` clause, what results in the equation that is used in the `else if` section. Finally the `else` section sets the bore torque to 0 if there is no contact to the ground. The factor `loadInfluence` enables the model to have a friction coefficient that is dependent on the normal load. The factor is explained in more detail in Section 3.5.6 on Page 50.

The result of this model can be seen in Figure 3.19. The bore torque strongly depends on the pitch rotation velocity of the tire which is true according to experience. But for every driving situation the torque is limited to the maximum value that depends on the size of the contact area, which in turn depends on the normal load via the *Vertical Dynamics* determined in the elevation class. From Figure 3.19 one can see

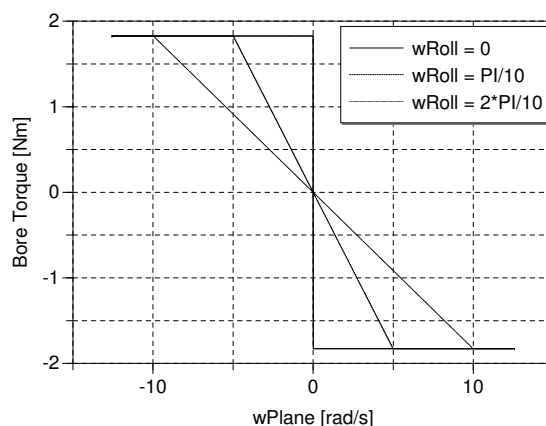


Figure 3.19: The bore torque depending on the yaw rate at different pitch angle velocities in the linear model.

that the situation of the tread elements sticking to the ground is not modeled assuming an immediate sliding of the tread elements.

The only parameter of the model μ_{Turn} (`muTurn`) basically determines the amplitude of the maximum value for the bore torque. The slope of the bore torque depending on ω_{Plane} and the contact area is a result of the model and cannot be influenced directly.

3.5.4.3 Combined Bore Torque

The *Combined Bore Torque* model is basically the same as the foregoing model. The main difference is that for the `if` condition, not the pitch rotation speed is used, but the velocity of adherence. This is a valid assumption if this velocity of adhering changes with the pitch rotation velocity, what is true for the *Combined Friction*. Therefore, this model should just be used in combination with the combined model for friction, indicated by the names of the models.

The used equations are the following which are very similar to the ones shown in

Equation 3.19.

$$t_{Bore} = \begin{cases} -f_N \cdot \text{loadInfluence} \cdot \mu_{Turn} \frac{1}{16} \sqrt{l_{CR}^2 + b_{CR}^2} \cdot \frac{\omega_{Plane}}{v_{Ad}} & \text{if} \\ \quad |\omega_{Plane}| \frac{1}{4} \sqrt{l_{CR}^2 + b_{CR}^2} < |v_{Ad}| \text{ AND Contact} \\ -f_N \cdot \text{loadInfluence} \cdot \mu_{Turn} \frac{1}{4} \sqrt{l_{CR}^2 + b_{CR}^2} \cdot \text{sign}(\omega_{Plane}) & \text{else} \\ \quad \text{if Contact} \\ 0 & \text{else} \end{cases} \quad (3.43)$$

The derivation of these equations can be found in [ZO09].

The results of this model are shown in Figure 3.20. Although the value for v_{Ad} changes

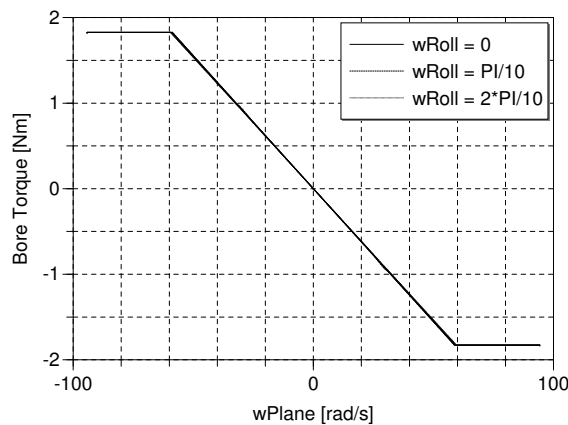


Figure 3.20: The bore torque depending on the yaw rate at different pitch angle velocities in the model from [ZO09].

slightly during the simulation, the pitch rotation velocity is too low to get big differences in v_{Ad} that would result in a curve with a lower slope. The reason for that is the offset in v_{Ad} that comes from the `softMaxTol` function and the parameters of the *Combined Friction* model which can be seen in Figure 3.16 on Page 41. This also means that the maximum values for the bore torque are reached at a relatively high turning speed of the tire compared to the foregoing model. This can be seen in Figure 3.20 as well when looking at the scaling of the x-axis. For higher pitch rotation velocities, the behaviors of the models become more similar.

3.5.5 Camber Force

The effect of camber force has its reason in the lean angle of a wheel. The elements in the tread shuffle cannot follow their natural path because they stick to the ground during the contact phase. That effect produces a force that adds up to the lateral force with its origin shown in Figure 3.21.

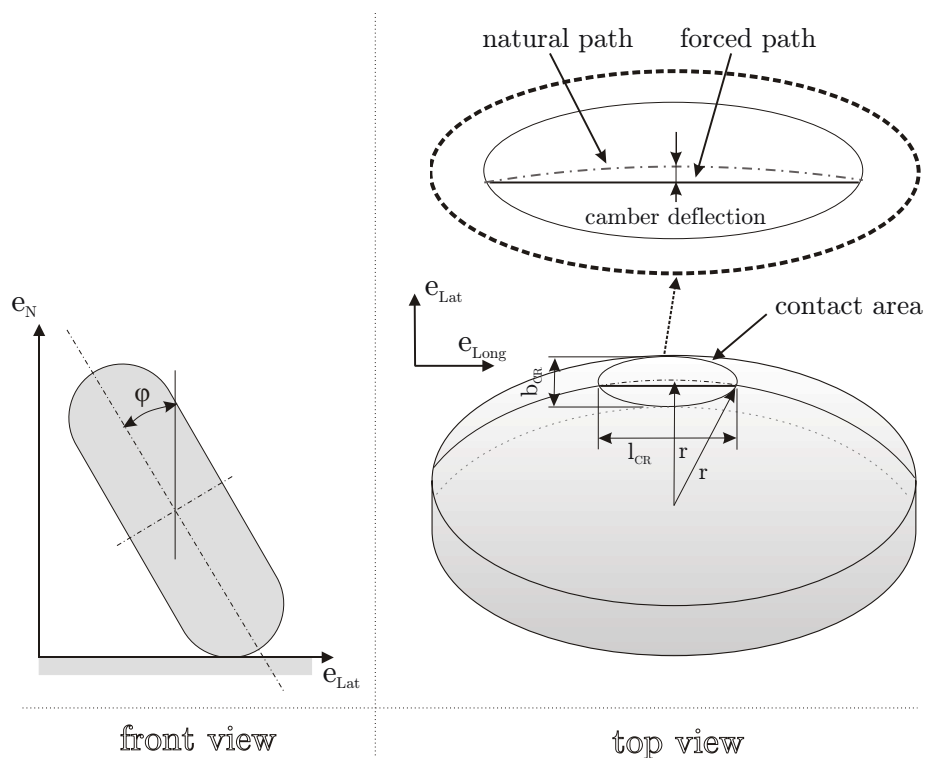


Figure 3.21: View of a wheel with lean angle and highlighted contact area including a sketch of the maximal camber deflection.

As pointed out by [ZO09] and presented in [Ril07], [Pac06] the force that results from a camber angle can be modeled by an additional sliding velocity, which adds up to the lateral sliding velocity. This is a convenient way because all the properties of the friction model stay valid automatically.

Regarding the implementation it would be possible to introduce a force as well, even in parallel with the additional sliding velocity. This could be done by adding the force to the `frictionBase` class and the `CamberForceBase`. The models covering the effects of occurring camber angles would then determine either the additional sliding velocity or the force directly and set the other one to zero. The approach using the force was not

used in the library due to the fact that the technique based on the additional sliding velocity is more convenient and common.

An application one could seriously worry about is, when the tire is not able to move laterally because of restrictions outside the tire model, but a lateral speed arises from a camber angle. In this case the additional slip velocity is transformed to lateral sliding velocity by Equation (3.13) and results in the corresponding force based on the friction model. Therefore, this application is not a problem.

3.5.5.1 Camber Force Base and No Camber Force

The model `CamberForceBase` ensures that variable `vLatCamber` is calculated somehow. Like in the foregoing models *No Camber Force* just sets `vLatCamber` to 0.

3.5.5.2 Simple Camber Speed

The *Simple Camber Speed* model is based on the assumption mentioned in the introduction. It computes the speed of the parts in the contact area arising from the forced path and adds it to the lateral sliding velocity. The equations are based on geometric properties that are depicted in Figure 3.21.

Following Pythagoras' rule for right-angled triangles, one can find the maximum of the camber deflection to be

$$CamberDeflection_{max} = r - r\sqrt{1 - \left(\frac{l_{CR}}{2r}\right)^2} \quad (3.44)$$

$$CamberDeflection = CamberDeflection_{max} \cdot \sin(\varphi) \quad (3.45)$$

Now the actual speed that adds up to the lateral sliding velocity is calculated by⁷

$$v_{LatCamber} = \begin{cases} camberGain \cdot camberDeflection \left(\frac{v_{RotLong}}{2 \cdot l_{CR}} \right) & \text{if} \\ \quad \text{Contact AND } l_{CR} > 0 \\ 0 \\ \text{else} \end{cases} \quad (3.46)$$

Equation (3.46) is based on the fact that the time of contact $t_{Contact}$ can be approximately calculated from the length of the contact area l_{CR} and the speed due to pitch

⁷Inserting (3.47) in (3.48) leads to (3.46).

rotation $v_{RotLong}$ by the equation

$$t_{Contact} \approx \frac{l_{CR}}{v_{RotLong}} \quad (3.47)$$

and the average speed $v_{LatCamber}$ therefore becomes

$$v_{LatCamber} \approx camberGain \cdot \frac{camberDeflection}{2 \cdot t_{Contact}} \quad (3.48)$$

assuming a triangular velocity curve during the time of deflection. With the parameter `camberGain` the effect of the model can be scaled easily to fit measured data. The result of the model is shown in Figure 3.22 in Section 3.5.5.3 depicting the additional lateral speed due to the lean angle. This is possible as both models have the same results qualitatively.

3.5.5.3 Combined Camber Speed

The foregoing model was based on assumptions that were taken from [ZO09] and modified them partially. *Combined Camber Speed* is taken directly from [ZO09] and utilizes the same equations for the calculation of the camber deflections. The following equations differ a little, with the argumentation for that to be looked up in [ZO09]. The `camberDeflection` is computed by Equation (3.44) and (3.45) again

$$v_{LatCamber} = \begin{cases} camberGain \cdot camberDeflection \cdot v_{RotLong} & \text{if} \\ \text{Contact AND } l_{CR} > 0 & \\ 0 & \text{else} \end{cases} \quad (3.49)$$

Figure 3.22 depicts the sinusoidal characteristics of the camber speed with respect to the camber angle. The horizontal line on the left of the figure results from the initial tilting of the wheel when applying the lean angle without pitch angle velocity and therefore as well with out camber force. As mentioned before the resulting speed is added to the lateral sliding velocity in the *Friction Base* model and therefore the characteristics of the slip models are directly influencing the resulting force.

3.5.6 Influence of the Load on the Friction Coefficient

As shown in [Ril07] the normal load of the tire has a nonlinear influence on the friction properties. Usually a doubled normal load does not enable the tire to transmit the

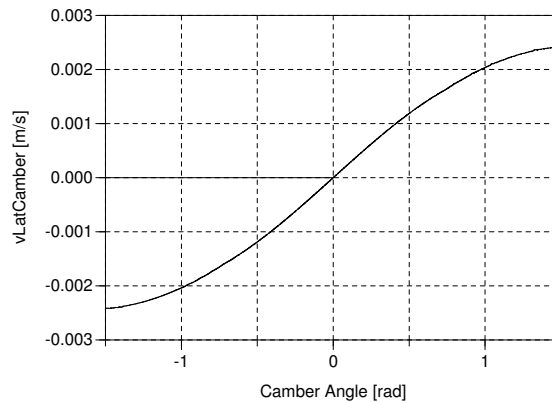


Figure 3.22: The lateral velocity due to the camber angle.

double forces in longitudinal and lateral direction. This fact shall be accounted for in the more complex of the following models.

3.5.6.1 Friction Load Influence Base

Besides the icon the *Friction Load Influence Base* defines the factor `loadInfluence` to be defined by the models that extend this base.

3.5.6.2 Linear Load Influence

Linear Load Influence is a quite simple model defining the factor `loadInfluence` to be 1. This results in a tire capable of transmitting driving forces which rise linearly with the normal load. Although this is not a very realistic model, it makes it possible to understand complex driving situations more easily and is therefore very useful for the modeler when trying to understand more complex simulations e.g. when searching for errors.

3.5.6.3 Quadratic Load Influence

The *Quadratic Load Influence* model from [Z009] makes the coefficient of friction μ dependent on the normal load. This combined with the fact that μ gets multiplied with the normal load in the reasonable models for slip properties is the reason for the quadratic behavior of the overall model with a maximum of the contact forces at f_{NSat} . If the normal force rises further, the contact forces stay constant due to a decreasing friction coefficient that is determined by the else clause of Equation (3.50).

The decrease of the friction coefficient is specified by three values. These are the nominal load $f_{Nominal}$, the friction coefficient at nominal load μ_N and the friction coefficient at the double of the nominal load μ_{2N} . The used equations are the following.

$$\text{loadInfluence} = \begin{cases} (2\mu_N - \mu_{2N}) - (\mu_N - \mu_{2N}) \cdot \frac{f_N}{f_{NNominal}} & \text{if } f_N \leq f_{NSat} \\ \frac{(2\mu_N - \mu_{2N})^2}{4 \cdot (\mu_N - \mu_{2N})} \cdot \frac{f_{NNominal}}{f_N} & \text{else} \end{cases} \quad (3.50)$$

$$f_{NSat} = f_{NNominal} \cdot \frac{2\mu_N - \mu_{2N}}{2 \cdot (\mu_N - \mu_{1N})} \quad (3.51)$$

Figure 3.23 shows the values of the `loadInfluence` factor as well as the corresponding longitudinal force at full sliding. Full sliding has been chosen because then the friction coefficient does not change during the simulation due to varying slip. The used values are the defaults of the model which are $f_N = 3kN$, $\mu_N = 1$ and $\mu_{2N} = 0.75$. Figure 3.23(a) shows the factor in an usual range of f_N , whereas Figure 3.23(b) shows the behavior of the model when very big normal loads are acting. These are not representative for usual simulation cases, but for quick transient events it is important that the friction coefficient still has positive values.

3.5.7 Overturning Torque

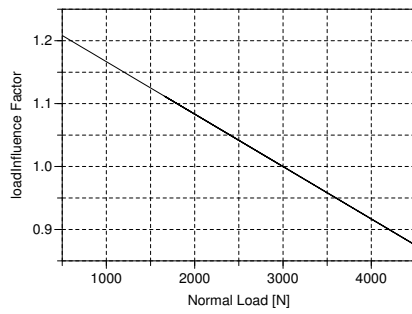
The overturning torque tries to reduce the lean angle of the tire by a counteracting torque.

3.5.7.1 Overturing Torque Base and No Overturing Torque

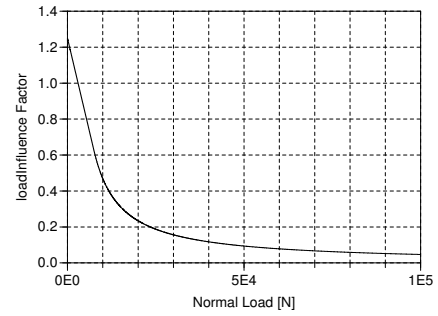
The *Overturing Torque Base* model ensures `tOverturn` to be calculated and *No Overturing Torque* can be used to set it to 0 if this effect is not accounted for in the overall model.

3.5.7.2 Linear Overturing Torque

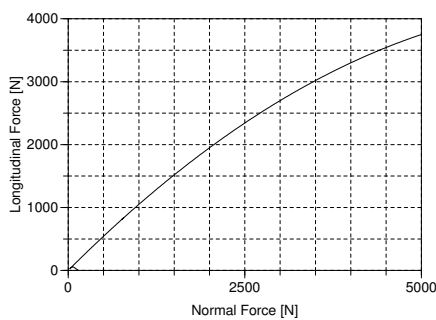
This effect arises due to the varying deflection of the contact area. This effect can be modeled as shown in [Ril07]. The model developed there is linearly dependent on the lean angle φ (`phi`).



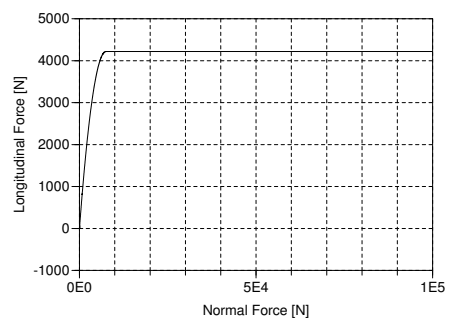
(a) $\text{loadInfluence} = f(f_N)$ for usual values of f_N .



(b) $\text{loadInfluence} = f(f_N)$ for very big values of f_N .



(c) $f_{Long} = f(f_N)$ for usual values of f_N at full sliding.



(d) $f_{Long} = f(f_N)$ for very big values of f_N at full sliding.

Figure 3.23: The load influence factor and longitudinal friction force in dependence on the normal load f_N .

As mentioned before the relationship between the lean angle φ and the overturning torque can be modeled linearly, as shown in Figure 3.24. Other factors that influence the value of the torque are radial stiffness and the width of the contact area. The equations used are the following.

$$c_{Overturn} = \frac{1}{12} \cdot b_{CR}^2 \cdot c_R \quad (3.52)$$

$$t_{Overturn} = c_{Overturn} \cdot \varphi \quad (3.53)$$

3.5.8 Self Aligning Torque

The self aligning torque is the result of the lateral force f_{Lat} which does not act on the center of the contact area in longitudinal direction. This distance is called the

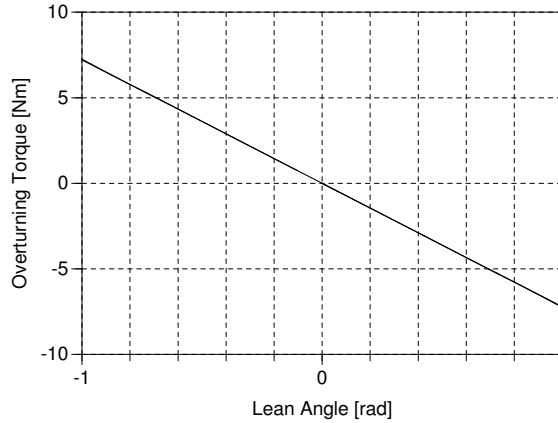


Figure 3.24: The overturning torque in dependence on the lean angle φ .

pneumatic trail due to the reasons of this trail. The force and the mentioned positional shift l_{Trail} produce a torque that tries to reduce the slip angle. How this can be modeled is shown in [MW03] very well. They find that l_{Trail} mainly depends on the slip angle.

The following models do a piecewise polynomial interpolation introduced by [Ril07]. They build on the lateral slip s_{Lat} , the slip of adhesion s_{Ad} and full sliding s_{Sl} as a base for the calculation of l_{Trail} rather than the slip angle α . This is possible due to the fact that the relation

$$\tan(\alpha) = s_{Lat} \quad (3.54)$$

is quite linear for low values of α . For values of α that would result in a nonlinear relation the values of l_{Trail} get 0, therefore this effect has no noticeable influence on the result.

3.5.8.1 Self Aligning Torque Base and No Self Aligning Torque

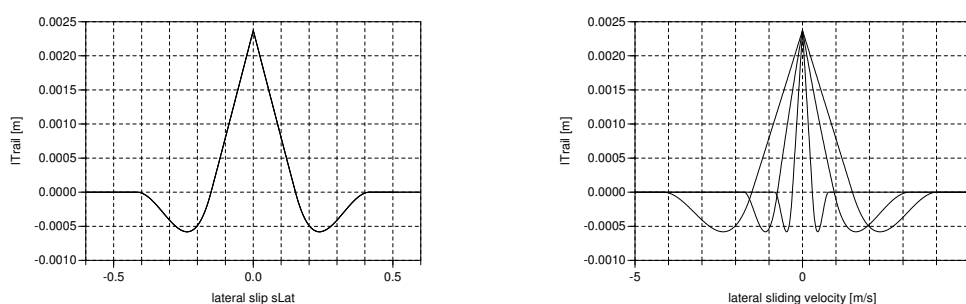
Self Aligning Torque Base ensures that variable `tAlign` is calculated somehow. Like in the foregoing models *No Self Aligning* sets `tAlign` to 0.

3.5.8.2 Self Aligning Torque Rill

As mentioned in the introduction and shown in Figure 3.25, the model of Rill uses the lateral slip as base for the calculation of the dynamic trail. The used interpolation is the following.

$$l_{Trail} = \begin{cases} \frac{1}{6} \cdot l_{CR} \cdot \left(1 - \frac{|s_{Lat}|}{s_{Ad}}\right) & \text{if} \\ |s_{Lat}| < s_{Ad} \text{ AND Contact} \\ -\frac{1}{6} \cdot l_{CR} \cdot \frac{|s_{Lat}| - s_{Ad}}{s_{Ad}} \left(\frac{s_{Sl} - |s_{Lat}|}{s_{Sl} - s_{Ad}}\right)^2 & \text{if} \\ |s_{Lat}| > s_{Ad} \text{ AND } |s_{Lat}| < s_{Sl} \text{ AND Contact} \\ 0 \text{ else} \end{cases} \quad (3.55)$$

Figure 3.25(b) shows the pneumatic trail in dependence on the lateral sliding velocity.



(a) The dynamic trail by [Ril07] depending on the lateral slip.

(b) The dynamic trail by [Ril07] depending on the lateral sliding velocity.

Figure 3.25: The dynamic trail depending on different physical quantities.

The differing curves arise from the fact that in the equation of the slip (3.10) the pitch angle velocity is accounted for inversely. As this velocity changes during the simulation that was carried out resulting in Figure 3.25, the dynamic trail does not directly depend on lateral sliding velocity.

In Figure 3.26, one can see the influence of the friction model on the dynamic trail l_{Trail} . Figure 3.26 depicts the result of a simulation with the friction model *Combined Friction*. The characteristic point of the friction curve s_{Ad} and s_{Sl} can change depending on the situation of the tire. As these are used for the borders of the interpolation for the pneumatic trail, this influences the curve for the pneumatic trail. The simulation was carried out with a translational speed of $2ms^{-1}$ (resulting in the wider curve) and $10ms^{-1}$ (the tighter curve) respectively. For all the other models, either the slip or the sliding velocities of the characteristic curve are constants and they therefore result in a behavior depicted in Figure 3.25(a) either based on the lateral slip (for

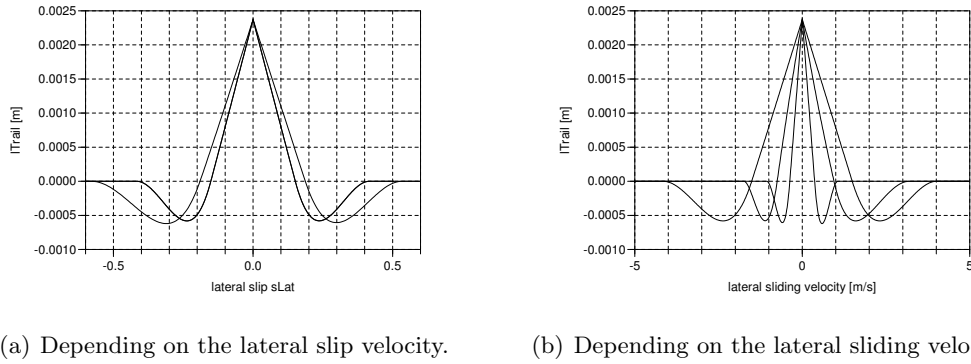


Figure 3.26: The pneumatic trail by [ZO09] depending on different variables with the *Combined Friction* model for the slip properties.

RillFriction) or the sliding velocities (for *Sliding Dry Friction* and *Sliding Dry Friction Longitudinal*).

3.6 Geometry

All geometric classes have two main tasks to fulfill. The first is to determine the contact point properties⁸ including the vector \mathbf{rCP} that points from the center of the rim⁹ to the contact point. Secondly, all have to compute the unit vectors that are important in many other objects. As the unit vectors are calculated similarly, independent of the actual geometric properties, these equations are implemented in the base model, as shown in Section 3.6.1.

The first task - the determination of \mathbf{rCP} is depending on the actual geometry of the tire, the normal vector of the surface at the contact point and the lean angle of the tire described by \mathbf{ePlane} . Therefore the contact point has to be found, which can be a non-trivial task in case of an uneven surface. In case of the *Wheels and Tires* library a quite simple approach was developed, which is described in Section 3.6.5 on Page 64. Although the same code for the finding of the contact point is used in all the models, it was copied to the three different geometric classes. This is reasonable due to the fact that the base class can be extended by classes using other techniques for finding the contact point totally different.

Figure 3.27 depicts the `penetrationDepth`, which is calculated in all geometric classes

⁸A full listing of the quantities gathered in the contact point properties can be found in Figure 3.6 on Page 24.

⁹In case of a dynamic belt the center point of the belt.

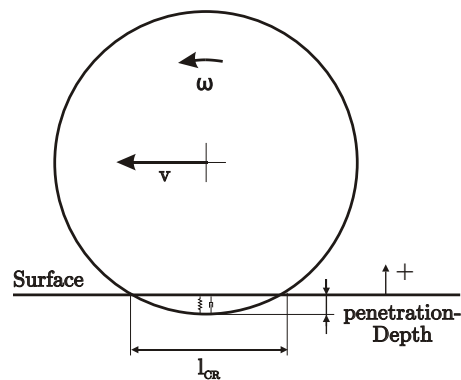


Figure 3.27: A tire with an ideally stiff belt penetrating the surface.

in a similar manner. It is used in the *Vertical Dynamics* classes to calculate the normal force acting on the contact point. The *No Elevation* class is a special case here as it sets the `penetrationDepth` to a value rather than using it to calculate a force¹⁰. More details on the *Vertical Dynamics* Classes can be found in 3.9 starting on Page 72. Positive values of the `penetrationDepth` indicate a tire lifted from the ground, whereas negative values denote a real penetration in the surface.

3.6.1 Geometry Base

The graphical model of the *Geometry Base* can be found in Figure 3.8 on Page 26. It can be seen that the unit vectors are calculated in this class because `UV` is an output. This is done based on `eAxis` as this is an input to the model from a measurement in the *Rim* Class. Furthermore, `eN` is determined utilizing the `get_eN`¹¹ function. The

¹⁰In this case Figure 3.8 on Page 26 has a wrong causality indicated for the `penetrationDepth` what is not a problem for Dymola and therefore disregarded for the sake of understandability of the mainly used models.

¹¹The `get_eN` function is part of the *Surface* class described in Chapter 5. It returns the normal vector on the *Surface* at the current position of the tire.

equations used for computing the other unit vectors are the following.

$$d_{Long} = e_N \times e_{Axis} \quad (3.56)$$

$$e_{Long} = \frac{d_{Long}}{\sqrt{d_{Long} \cdot d_{Long}}} \quad (3.57)$$

$$d_{Lat} = e_{Long} \times e_N \quad (3.58)$$

$$e_{Lat} = \frac{d_{Lat}}{\sqrt{d_{Lat} \cdot d_{Lat}}} \quad (3.59)$$

$$d_{Plane} = e_{Long} \times e_{Axis} \quad (3.60)$$

$$e_{Plane} = \frac{d_{Plane}}{\sqrt{d_{Plane} \cdot d_{Plane}}} \quad (3.61)$$

The graphical view of all the classes in the *Geometry* package is the same, as only equations are added to realize their functionality.

3.6.2 Flat Disc Model

The first and simplest model has the geometry of a flat disc and it is shown in Figure 3.28. The width of the model is just used for the visualization and is not accounted

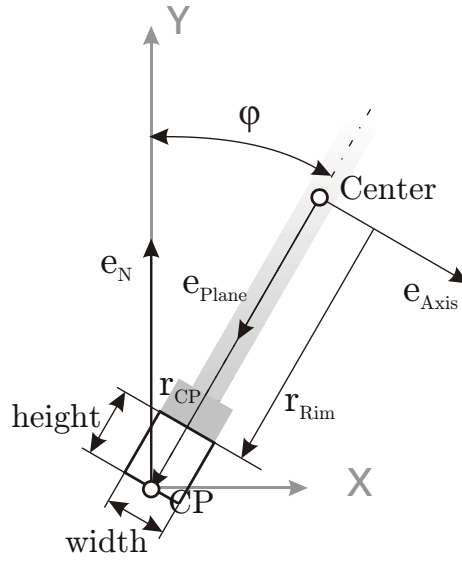


Figure 3.28: Geometric properties of the ideal tire.

for in the geometrical calculations.

The equations that are used to find the contact point properties are the following.

$$r_{CP} = \begin{cases} r_{Wheel} \cdot e_{Plane} - penetrationDepth \cdot e_N & \text{if} \\ Contact & \\ r_{Wheel} \cdot e_{Plane} & \text{else} \end{cases} \quad (3.62)$$

$$penetrationDepth = (x_{Belt} - x_{CP} + r_{CP}) \cdot e_N \quad (3.63)$$

$$Contact = penetrationDepth \leq 0 \quad (3.64)$$

$$l_{CR} = \begin{cases} \sqrt{8 \cdot r_{Wheel} \cdot penetrationDepth} & \text{if} \\ Contact & \\ 0 & \text{else} \end{cases} \quad (3.65)$$

$$b_{CR} = 0 \quad (3.66)$$

With r_{CP} being the vector pointing from the belts center point to the contact point, r_{Wheel} the overall radius of the wheel ($r_{Rim} + height$ of the belt), x_{Belt} being the absolute position of the belt, x_{CP} the absolute position of the contact point and l_{CR} , b_{CR} being the length and the width of the contact area respectively.

The if clauses are necessary due to the fact that the `penetrationDepth` gets positive values if the tire lifts from the ground. That would lead to wrong results for `rCP`.

3.6.3 Circular Tire

The geometry of the circular tire is depicted in Figure 3.29. From the figure one can directly recognize that the `rCP` depends on the lean angle φ of the tire. This has to be accounted for in the computation. One can also see that the radius of the circular element is called `height`. This is done to enable a simple class change from the ideally slim belt to the circular belt.

The resulting equations are the following.

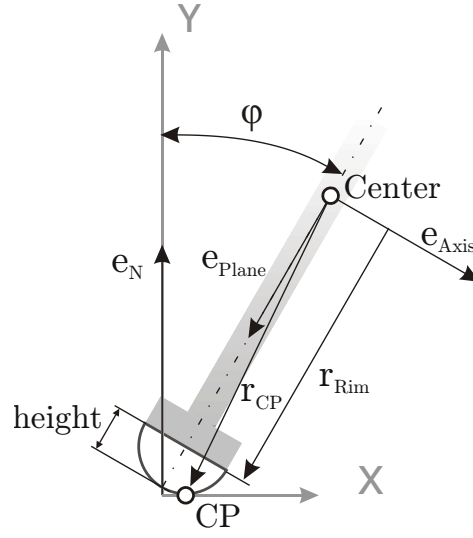


Figure 3.29: Geometric properties of the circular tire.

$$r_{CP} = \begin{cases} r_{Rim} \cdot e_{Plane} - height \cdot e_N - penetrationDepth \cdot e_N & \text{if} \\ Contact & \\ r_{Rim} \cdot e_{Plane} - height \cdot e_N & \text{else} \end{cases}$$

(3.67)

$$penetrationDepth = (x_{Belt} - x_{CP} + r_{CP}) \cdot e_N$$

(3.68)

$$Contact = penetrationDepth \leq 0$$

(3.69)

$$l_{CR} = \begin{cases} \sqrt{8 \cdot (r_{Rim} + height) \cdot (-penetrationDepth)} & \text{if} \\ Contact & \\ 0 & \text{else} \end{cases}$$

(3.70)

$$b_{CR} = \begin{cases} \sqrt{8(r_{Rim} + height) \cdot (-penetrationDepth)} & \text{if} \\ \quad Contact \text{ AND } (-penetrationDepth) < height \\ 2 \cdot height & \text{if} \\ \quad Contact \text{ AND } (-penetrationDepth) > height \\ 0 & \text{else} \end{cases} \quad (3.71)$$

For the visualization of the tire (or the belt), a torus element is used. It extends the *Visualizer Base* and computes a donut like shape with an parameter **opening** angle that makes it useful for the visualization of a tire. To visualize it correctly, a relative rotation matrix **RRel** has to be calculated which is done by the following equation. This is a simplified version of Equation (7.5) in [Zim06].

$$R_{Rel} = R_{Belt}[1, :] \cdot R_{Belt}[1, :]' - skew(R_{Belt}[1, :]) \quad (3.72)$$

3.6.4 Belted Tire

The most complex geometry that is possible with the library is the belted tire shown in Figure 3.30. It features a tire that has a radius describing the tread area as well as side walls, both configurable easily by the parameters *width*, *height* and *r* (radius). Out of these three parameters, adding the information of the rim's radius, the other geometrical aspects are computed. As well a detection whether the tire rolls on the corner between the side wall and radius is implemented.

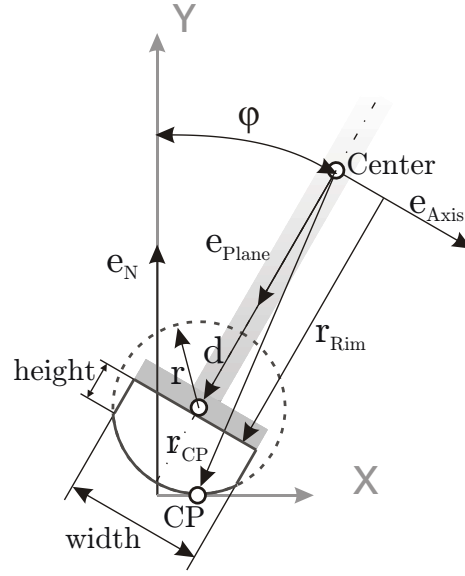


Figure 3.30: Geometric properties of a tire with side walls and tread area.

Therefore the equations become as follows.

if Contact then

$$r_{CP} = \begin{cases} d \cdot e_{Plane} - r \cdot e_N - penetrationDepth \cdot e_N \\ \quad \text{if } (abs(\varphi_{act}) < \varphi_{max}) \\ (r_{Rim} + height)e_{Plane} + x \cdot e_{Axis} - penetrationDepth \\ \quad \text{else AND } (\varphi_{act} > \varphi_{max}) \\ (r_{Rim} + height)e_{Plane} - x \cdot e_{Axis} - penetrationDepth \cdot e_N \\ \quad \text{else} \end{cases} \quad (3.73)$$

else

$$r_{CP} = \begin{cases} d \cdot e_{Plane} - r \cdot e_N \\ \quad \text{if } (abs(\varphi_{act}) < \varphi_{max}) \\ (r_{Rim} + height) \cdot e_{Plane} + x \cdot e_{Axis} \\ \quad \text{else AND } (\varphi_{act} > \varphi_{max}) \\ (r_{Rim} + height) \cdot e_{Plane} - x \cdot e_{Axis} \\ \quad \text{else} \end{cases} \quad (3.74)$$

$$\varphi_{Act} = \frac{\pi}{2} - arccos(e_{Axis} \cdot (-e_N)) \quad (3.75)$$

$$penetrationDepth = (x_{Belt} - x_{CP} + r_{CP}) \cdot e_N \quad (3.76)$$

$$Contact = penetrationDepth < 0 \quad (3.77)$$

$$l_{CR} = \begin{cases} \sqrt{8 \cdot (r_{Rim} + r \cdot (-penetrationDepth))} \\ \quad \text{if } Contact \\ 0 \\ \quad \text{else} \end{cases} \quad (3.78)$$

$$b_{CR} = \begin{cases} \sqrt{8r \cdot (-penetrationDepth)} \\ \quad \text{if } Contact \text{ AND } (-penetrationDepth) < r - height \\ 2 \cdot r \\ \quad \text{if } Contact \text{ AND } (-penetrationDepth) > r - height \\ 0 \\ \quad \text{else} \end{cases} \quad (3.79)$$

Again the matrix `RRe1` is computed for the visualization of the torus element. This is done the same way as in Equation (3.72). Some additional equations are set to find

the following final parameters.

$$x = \frac{width}{2} \quad (3.80)$$

$$y = rRim + height \quad (3.81)$$

$$d = y - \sqrt{r^2 - x^2} \quad (3.82)$$

$$\varphi_{max} = \frac{\pi}{2} - \arctan\left(\frac{\sqrt{r^2 - x^2}}{x}\right) \quad (3.83)$$

$$\varphi_{Belt} = \arctan\left(\frac{x}{y - d}\right) \quad (3.84)$$

With x being the x coordinate of the outmost possible contact point of the tire, y the y coordinate of the outmost possible contact point of the tire, d the distance from the rims center to the midpoint of the circle describing the tire's curvature, whereas φ_{max} lean angle at which the tire's contact point is at the outmost possible point and φ_{Belt} being half opening angle of the torus used for the visualization.

3.6.5 Finding the Contact Point

To prevent unnecessary computational effort in case of a flat surface that will be sufficient in many cases, the parameter `FlatSurface` has been integrated in the `Surface` class. This parameter can be used in geometric classes to find the contact point which is described by the vector \mathbf{x}_{CP} pointing from the world's coordinate system to the contact point CP in a very simple way. This is done by the following equations

$$x_{CP} = \begin{pmatrix} x_{Belt}[1] + r_{CP}[1] \\ 0 \\ x_{Belt}[3] + r_{CP}[3] \end{pmatrix} \quad (3.85)$$

$$e_N = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad (3.86)$$

The main simplification is that the normal vector always points into the positive y axis and does not have to be determined by a function. The second element of \mathbf{x}_{CP} can be

set to zero directly what is another simplification.

In the *surface* class the two functions `get_eN` and `get_elevation` have to be defined. Depending on the way the actual surface is defined, they can have very different algorithms in their body, the geometric class does not have to account for. The *Environment* package containing the *surface* is described in Section 5.2.1 on Page 102 in more detail.

On uneven surfaces, the following equations are used to find the contact point.

$$x_{CP} = \left\{ \begin{array}{l} x_{Belt}[1] + r_{CP}[1] \\ \text{Surface.get_elevation}(x_{CP}[1], x_{CP}[3]) \\ x_{Belt}[3] + r_{CP}[3] \end{array} \right\} \quad (3.87)$$

$$e_N = \text{Surface.get_eN}(x_{CP}[1], x_{CP}[3]) \quad (3.88)$$

Obviously there is no iteration used to find the contact point. This is not necessary due to the fact that the geometry of the tire is well defined and in combination with the unit vectors the ideal contact point can be calculated by geometric equations starting from the belt's center point \mathbf{x}_{CP} . The `penetrationDepth` is calculated in the *geometry* class as well and is used to correct the ideal vector \mathbf{r}_{CP} to the real one. Adding the vector \mathbf{r}_{CP} to the position of the rim \mathbf{x}_{Belt} leads to the position of the contact point \mathbf{x}_{CP} . In case of an uneven surface this is done for the x and z coordinates, whereas the y coordinate is determined by the `get_elevation` function. Furthermore the normal vector \mathbf{e}_N can be found from the x and z position of the contact point. So everything needed for further calculation is defined in a quite simple fashion.

Basically, the vector \mathbf{x}_{CP} is just used for the functions `get_elevation`, `get_eN` and `get_mu`. It is not set in the translational part of the *Contact Bond*, as the contact point's position is defined by \mathbf{x}_{Belt} adding \mathbf{r}_{CP} . To be precise, the *Contact Bond's* frame position is not equal to \mathbf{x}_{CP} at all times as the vector \mathbf{x}_{CP} is pointing on the surface at all times whereas the *Contact Bond's* frame is on the tires surface (lifting with the tire from the ground).

3.7 Contact Physics

The *Contact Physics* model applies forces and torques on the contact point, as well as measuring its velocities. All these physical quantities get transferred via the *Contact Point Connector* to the models determining friction and vertical dynamic properties.

Measuring and setting of speeds and forces has to be done in an acasual way as ideal models set velocities rather than applying forces.

The second thing determined by the *Contact Physics* class is the dynamic behavior of the contact point as some models developed [Pac06] introduce a flexibility of the contact point in longitudinal and lateral direction. Two different models are implemented, both described in the following sections.

3.7.1 Contact Physics Base

The *Contact Physics Base* features the variable definitions and the connections to the *Tire Bus* as well as the *Contact Point Connector*. It contains everything that can be seen in the shaded part in Figure 3.31.

3.7.2 Contact Bond No Dynamics

As mentioned above, the model applies forces and torques and measures the speed of the contact point. This can be seen in Figure 3.31. The `mSe` elements `fSource` and `tSource` act as sources of force or torque respectively, whereas the `Df` element `vSens` measures the speed of the contact point. Important to now is that these bond graphic elements are a-causal. This means that e.g. a velocity sensor can apply a velocity to the bond as well as measure one. The main difference between the two items is that the sensor contains more equations that set the not measured quantity (in the actual case of the `Df` element the effort) to zero, whereas the source does not set the other quantity.

Another thing contained is the `MBG_defaults` item with the “3” in it. This element defines the contained multi bond elements to have a size of three, defining the three dimensional space the model is calculated in. This makes e.g. the source element apply three torques and the sensors measure three quantities.

Another important element in the model is the `mTF` element. It does the transformation of the translational quantities from the world coordinate system to the body-coordinate system that is used for calculations at the contact point. The body coordinate system in the contact point is defined by the three unit vectors `eLong`, `eLat` and `eN`. These three unit vectors are used to build the rotation matrix that is fed into the `mTF` element by the Equation (3.89)¹². This body coordinate system is also used to set the rotation matrix of the flange connector of the *Contact Physics*'s frame. This is done in order to enable the modeler to simply connect parts (like arrows for visualization) to the frame in a convenient way.

¹²Torques (and angular velocities) do not need any transformation as they are given in the body coordinate system by the definition of the *Multi Bond Library*, whereas translational quantities are resolved in the world coordinate system.

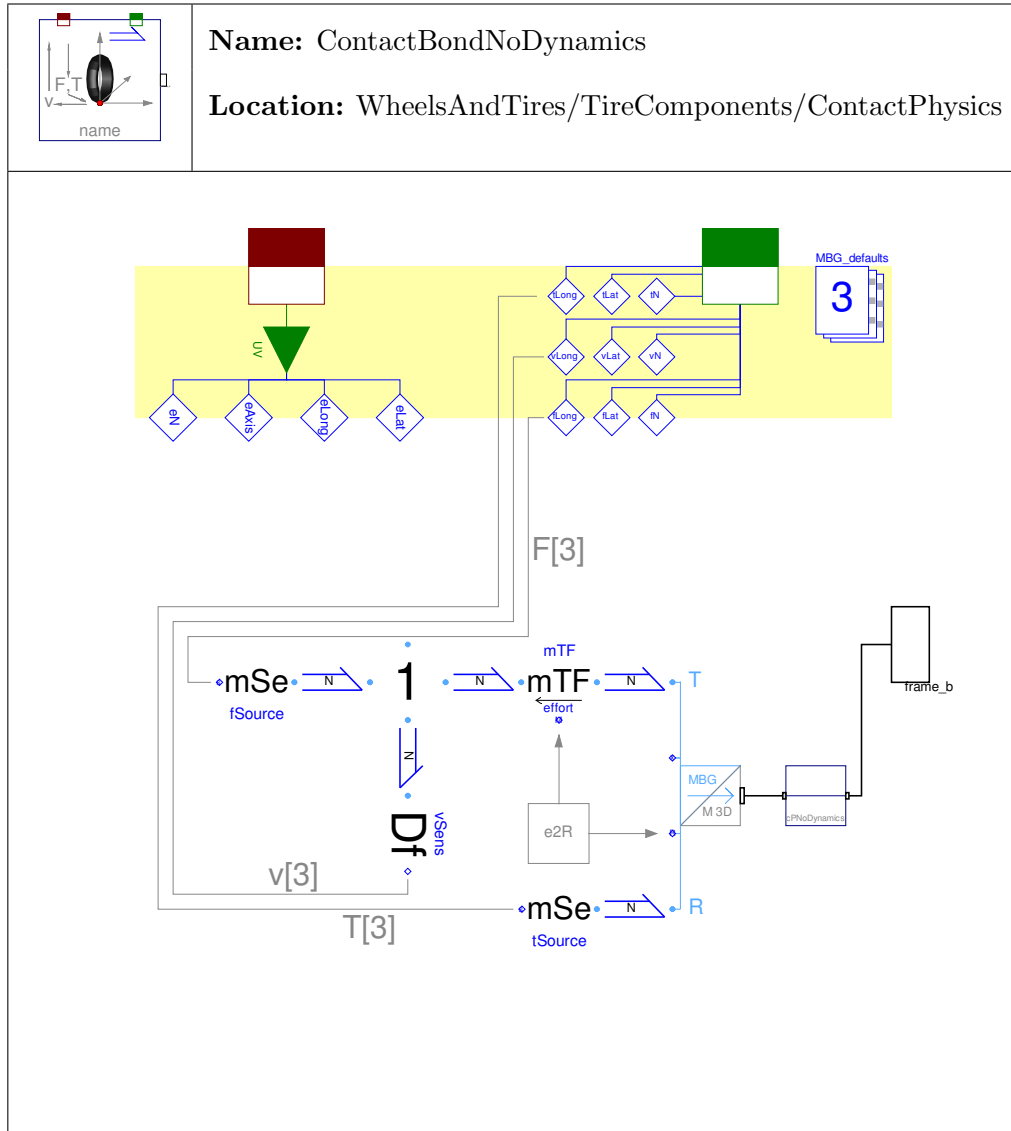


Figure 3.31: The model applying forces and torques to the contact point and measuring its velocities without any dynamics.

$$R = \left\{ \begin{array}{ccc} eLong & eN & eLat \end{array} \right\}' \quad (3.89)$$

The model `cpNoDynamics` can be found between the `MBG2Mech` element and `frame_b` does not feature anything else but a rigid connection between its two frames and is therefore not described any closer.

3.7.3 Contact Bond With Dynamics

The only difference between the *Contact Bond With Dynamics* and the one without is the model for the contact point dynamics. In this model, the rigid connection from *Contact Bond No Dynamics* is replaced by the model shown in Figure 3.32.

The *Contact Point Second Order* model features two *Modulated Actuated Prismatic* elements that apply forces computed by the connected spring damper elements. The *Modulated Actuated Prismatic* elements are different to the usual prismatic elements as their direction of operation is determined by a vector that is an input to the model. A more detailed description can be found in Section 3.11.2.5 on Page 91. The main purpose of these elements is to ensure that the dynamics point to the lateral and longitudinal direction regardless of what happens during initialization or with dynamic elements in the *Belt Dynamics*.

Using the *Contact Bond With Dynamics* together with an uneven surface will cause Dymola to respond with an error message. This arises due to the fact that for used spring damper models the relative position and velocity have to be calculated which implies a derivation of the elevation (from the *Surface* class). This is not possible due to the structure of the used *Surface* class.

The initialization is done in a quite simple way setting the relative distance and speed of the *Modulated Actuated Prismatic* elements to zero.

3.8 Center to Contact Point

The *Center To Contact Point* model is a model without a nice physical interpretation. It is kind of a *Fixed Translation* element known from the *Modelica Standard Library* [Ass09]. This model is built on multi bond graph techniques and is therefore derived from the *Fixed Translation* in the *Multi Bond Lib* [Zim06]. It shall describe the translation between the contact point and the belt's center point.

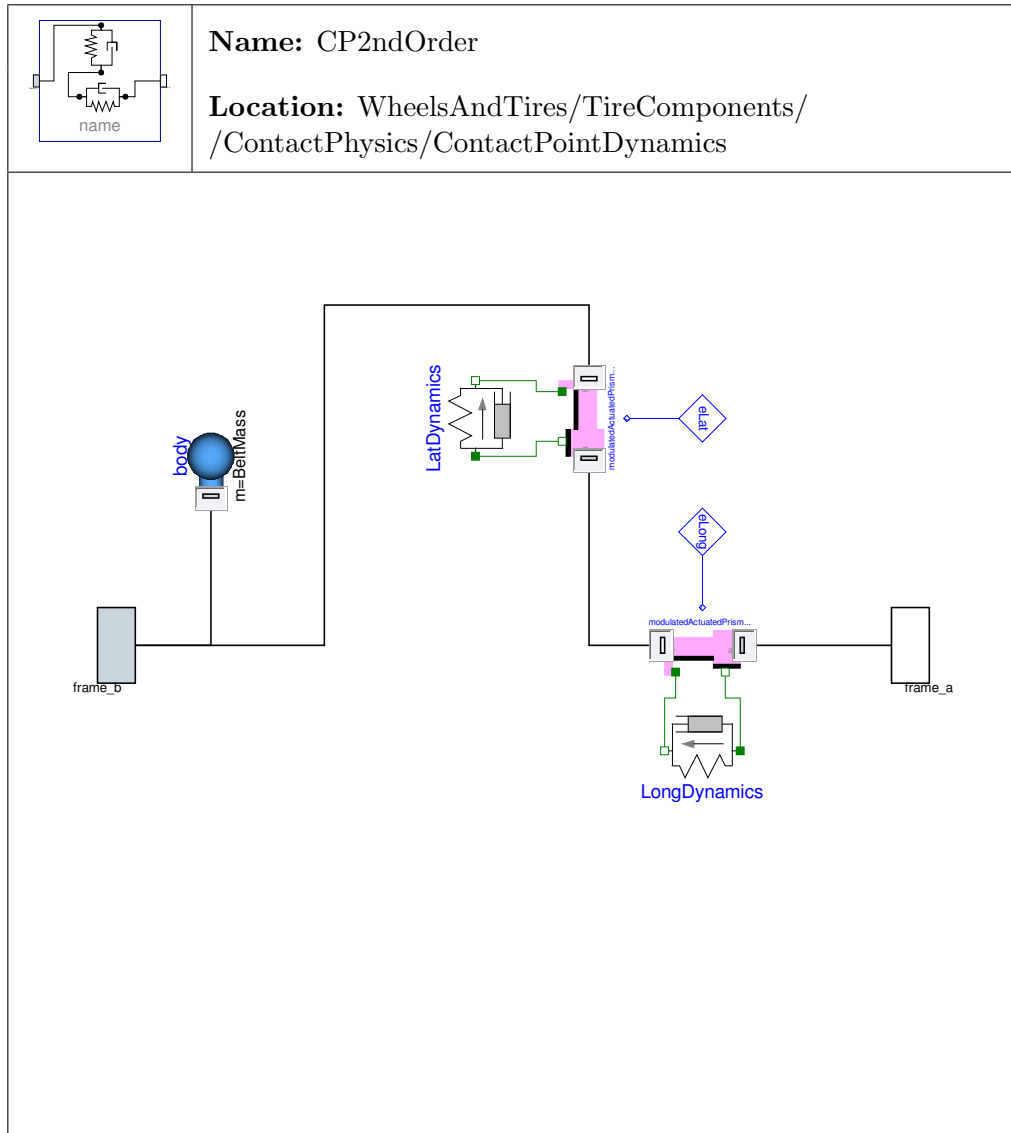


Figure 3.32: Model that enables the contact point to shift in longitudinal and lateral direction.

3.8.1 Center to Contact Point Base

The base class does not feature more than the frames and the interface to the *Tire Bus* the definition of the variable r (r_{CP}) needed from the communication structure.

3.8.2 Center to Contact Point Bond

There are three main differences between the *Center To Contact Point* model and the *Fixed Translation*. The first of which is that the *Center To Contact Point* model does not have fixed dimensions as the vector r_{CP} (r in Figure 3.33) depends on the `penetrationDepth` as well as the normal vectors `ePlane` and `eN`. The second difference is that it must not rotate even if `frame_a` is connected to a frame with a pitch angle velocity. The last is that the two frames have different rotation matrices. To cover all these demands a special model was created, shown in Figure 3.33.

To realize the required features mentioned at the beginning of this section, two new multi bond elements had to be created. These are the `mTranslationalTF_r` (modulated translational transformer radius) and the `mTranslation_r` (modulated translation radius). The first one adds a variable vector to a positional vector of the `frame_a` setting the geometrical relation between `frame_a` and `frame_b`. A more precise description can be found in Section 3.11.2.4. The `mTranslationalTF_r` is an element computing the relation between torques and forces acting on a translational element depending on the dimensions and orientation of the element. A more detailed description can be found in Section 3.11.2.3.

The second requirement to not rotate with the pitch angle velocity is fulfilled by the `mSF` element on the lower left of the model. It measures the angular velocity about the local z axis and subtracts it from the speeds at the following 0 junction. This results in an element that does not react on pitch angle velocity but does so for the other two angular velocities.

The different rotation matrices emerge from the contact bond (`frame_b`) and from the *Rim/Belt Dynamics* (`frame_a`)¹³. Different matrices are used to enable a more plausible description of the two frames. It was considered to have a single rotation matrix solution for which the frame of the contact point would be described by the same rotational matrix as the rim. This would lead to a rotating frame of the contact point, which is inconvenient as the frame of the contact point would rotate around the unit vector `eAxis`. Additionally, this structure is likely to cause higher computational effort during simulation. Therefore, the solution containing two rotation matrices was chosen and thus the *Center to Contact Point* model has to handle these two rotation matrices. The *Modulated Effort Transformer* `mTF_efford` calculates the rotational quantities from the belt's body coordinate system to the world system. The same is done

¹³Note that the Model is flipped horizontally when built in the wheels model.

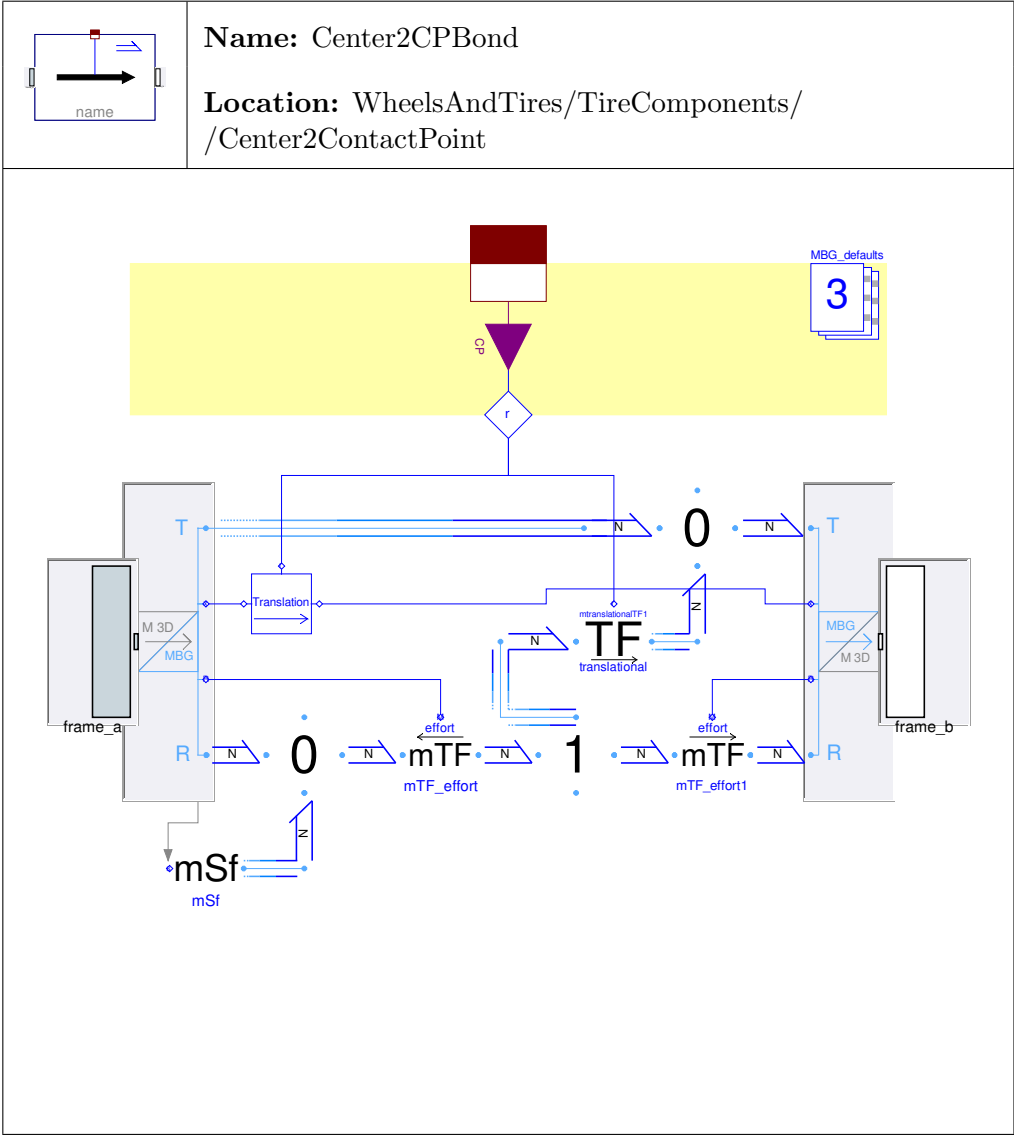


Figure 3.33: Bond graphic model for the translation from the center of the rim/belt to the contact point.

by `mTF_efford1` for the quantities of the contact point. Additionally, the *Modulated Translational Transformer* `translational` connects the rotational and translational quantities (both in the world coordinate system) as in the *Fixed Translation* model.

3.9 Vertical Dynamics

The base class just defines the two bus connections to the *Tire Bus* and the *Contact Point Properties*. Nothing more is defined here due to the essentially different structure of the models extending the base.

The `penetrationDepth` is calculated in the *Geometry Class*. It can either be set to a certain value as in the *No Elevation* class or be the base for the calculation of the normal force `fN`. It is important to notice that `penetrationDepth` as well as `fN` are directed as `eN` is.

3.9.1 No Elevation

This class (depicted in Figure 3.34) differs from the other elevation classes due to the fact that not a normal force is computed as a function of the penetration depth and speed. It imposes a holonomic constraint, that defines the values of `penetrationDepth` and therefore also its derivative the penetration speed to be zero. This implies that there is contact at all time. The necessary force is then calculated from that holonomic constraint.

An important characteristic of this model is that it has to derive the `penetrationDepth` that depends on the `get_elevation` function. As this function is not derivable analytically, it is not possible to use this model on uneven surfaces. To be more precise, the parameter `flatSurface` has to be `true`.

3.9.2 Kelvin Elevation

This class is the first one to enable the tire to lift from the ground. The force-free elevation is realized with the *Elevation Gap* described in detail in Section 3.11.1.7 on Page 3.11.1.7. The model compensates the force of the connected mechanical system, when there is no contact to the ground. The external mechanical system is connected via the lower two mechanical connectors `flange_a_extension` and `flange_b_extension`. A problem regarding the so-called “sticking effect” is explained in more detail in Section 3.11.1.7 on Page 87.

The *Kelvin Elevation* shown in Figure 3.35 is basically a model that combines the *Elevation Gap* model with a parallel spring damper component. This is a simple but

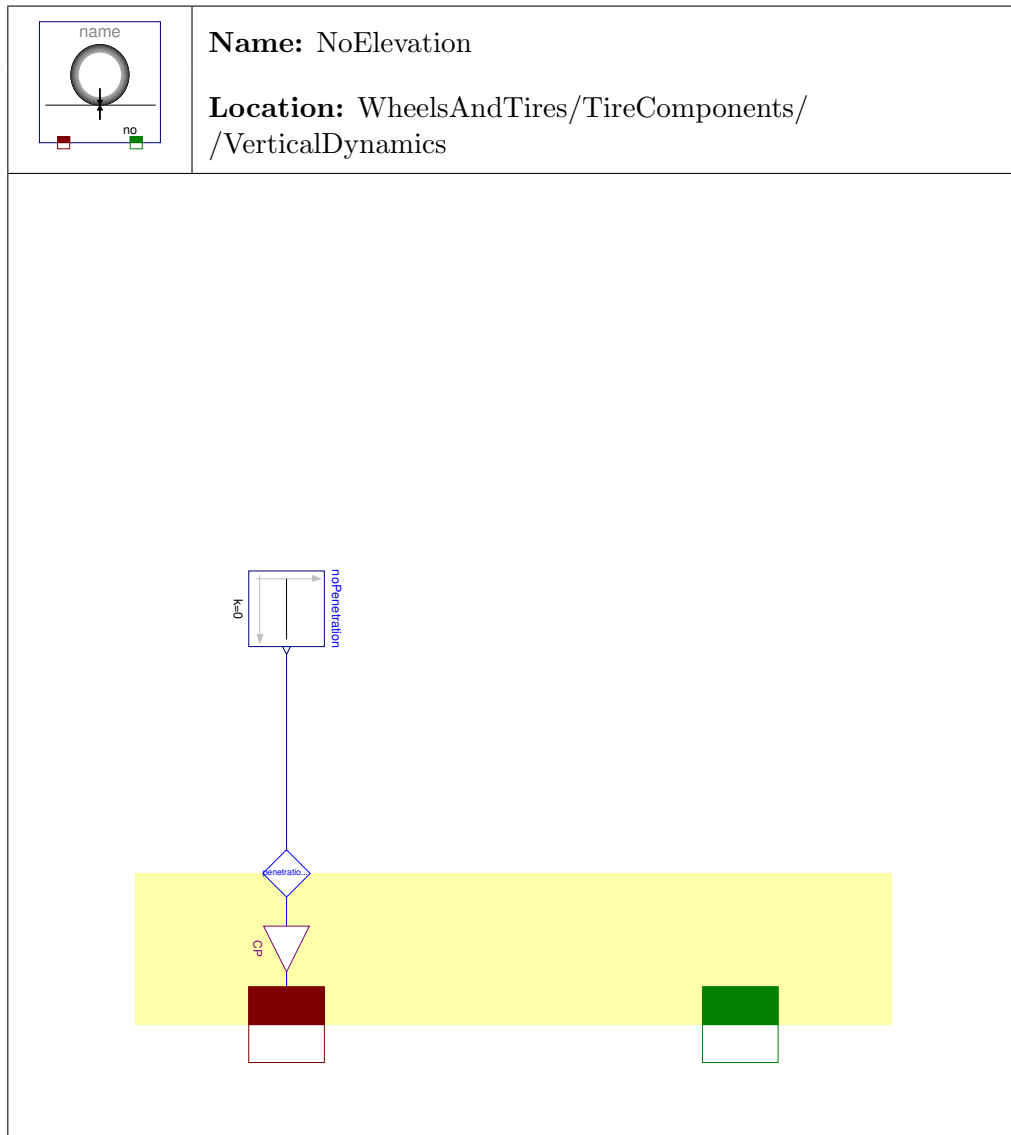


Figure 3.34: Model that makes the tire stick to the ground.

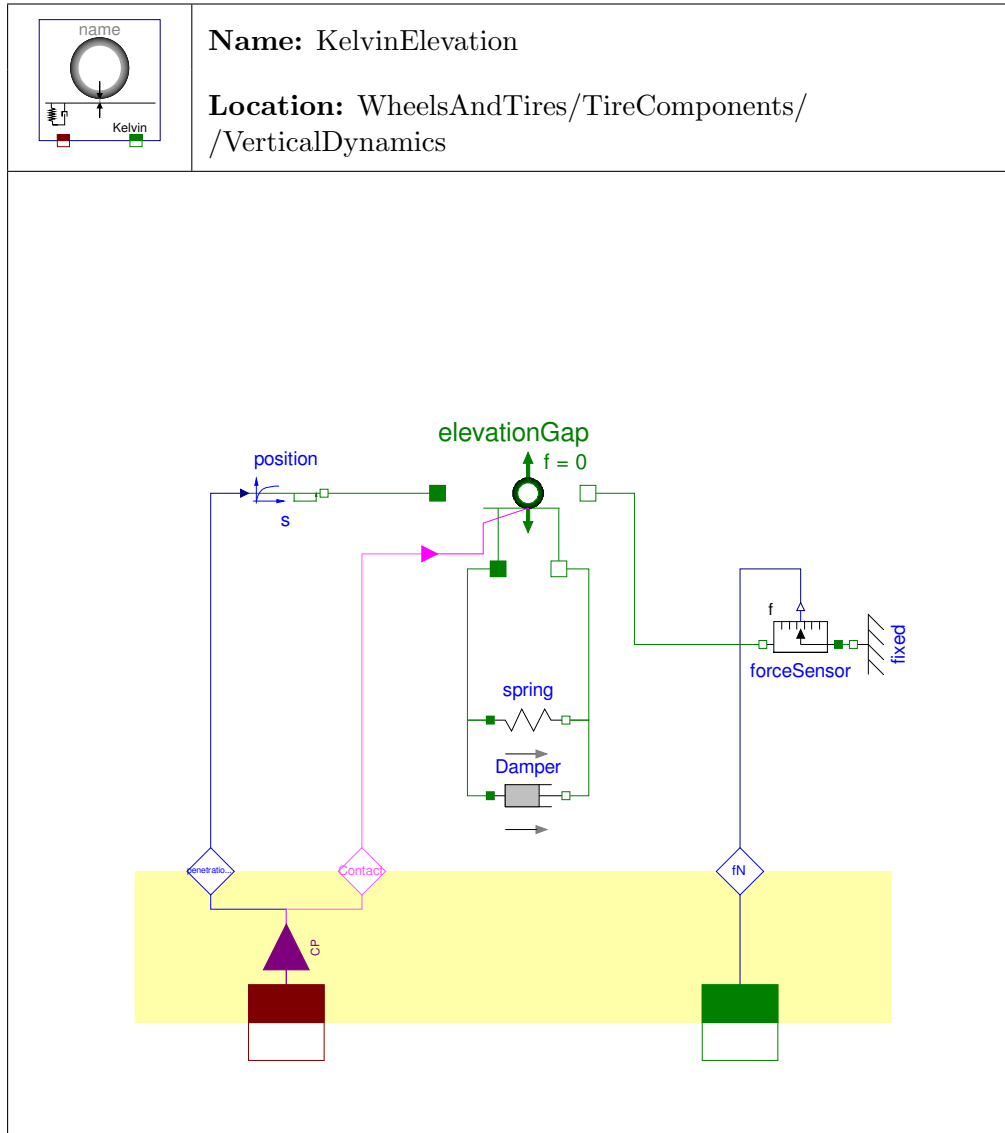


Figure 3.35: Model of the vertical dynamics with possible elevation and a parallel spring damper elements realizing vertical dynamics.

yet realistic way to model vertical dynamics of a tire and is quite familiar to modelers as well.

3.9.3 Gehmann Elevation

The *Gehmann Elevation* model combines the *Elevation Gap* with a more complex model for rubber modeling [MW03]. Here one can see the advantage of the *Elevation Gap* model that makes it very easy to use the components of the 1D mechanics bond graph library to define vertical dynamics. The only difference to the *Kelvin Elevation* is that the *Damper* is replaced by a serial spring damper element, what is the reason why the *Gehmann Elevation* is not depicted separately.

However, in this case there is one problem related to the bond graph library that cannot compute the behavior of a spring and a damper connected in series directly. The simpler possibility to overcome this problem is to add a very light weight mass between the two components. However, this introduces additional oscillations to the system which is usually of very low damping. So to overcome this problem a bond graphic model of the whole spring damper system connected in series has been created that is now used to calculate the behavior of the Gehmann model for rubber. This bond graphic model is shown in Figure 3.42 on Page 88. It is based on the model of a simple spring as it has been modeled in the 1D translational library of the *Bond Lib* [CN05].

3.9.4 Elasto Elevation

The *Elasto Elevation* (shown in Figure 3.36) is derived from the Standard Library of Modelica in Version 3.0 [Ass07]. The problem concerning sticking was elegantly solved there. So this model features a parallel spring damper system that acts without sticking. Another advantage of that model is that it does not introduce a discontinuity, that usually arises from the dampers force which is proportional to the speed of movement. In the event that the tire moves to the road with a certain speed, the damper force gets compensated until `Contact` gets true. From that point on the spring force rises continuously but the damping force already has a certain value $\neq 0$ due to the speed of movement. So in general and if there is no more complex behavior to be modeled, the *Elasto Elevation* model is the best choice for the vertical dynamics.

The model is defined by equations exclusively as no standard elements can be used to model the described behavior. The gray elements shown in Figure 3.36 are simple lines representing the following equations.

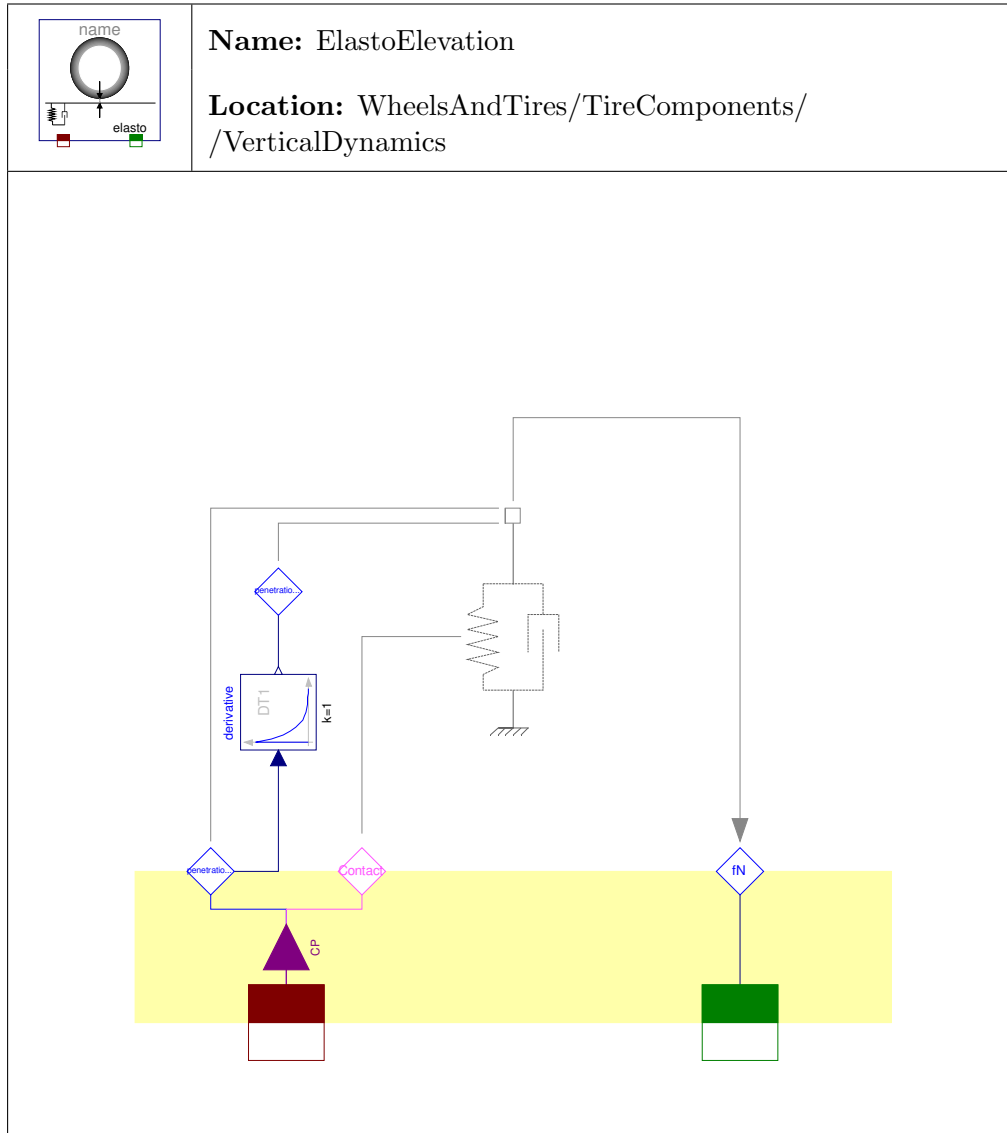


Figure 3.36: Model of the vertical dynamics with possible elevation and a special spring damper combination to overcome the sticking effect.

$$\begin{aligned}
& \text{if Contact} == 1 \left\{ \begin{array}{l} f_c = cR \cdot \text{penetrationDepth} \\ f_d = dR \cdot \text{penetrationSpeed} \\ f_N = \begin{cases} 0 & \text{if } (f_c + f_d) \geq 0 \\ f_c + \max(f_c, f_d) & \text{else} \end{cases} \end{array} \right. \\
& \text{else} \left\{ \begin{array}{l} f_c = 0 \\ f_d = 0 \\ f_N = 0 \end{array} \right.
\end{aligned} \tag{3.90}$$

3.10 Belt Dynamics

The models described in this section are intended to provide a dynamically behaving belt. To be more precise, it shall model the flexibility of the tire. This is realized by connecting an ideal tire model defined by the *Geometry Class* to the ideal rim in different ways. The models are called *Belt Dynamics* models because the Belt suits the description by the ideal geometry best. All models except the *Rigid Belt* model add a considerable amount of complexity to the simulation. This challenges the modeler to exactly know what is going on in the model.

Every belt model is realized to describe the relationship between the center point of the rim and the (virtual) center point of the belt. This is considered to be the most general approach of modeling. This is also the reason for the *Center to CP* model to be necessary, because it realizes the translation between the center of the belt to the contact point.

3.10.1 Belt Dynamics Base

The base model defines the used connections to the *Tire Bus*, necessary sensors and UV as an input. The basic elements of the icon are defined in the base as well. Nothing more is done here. The model looks exactly the same as shown in Figure 3.37 on Page 79 without the rigid connection between the two frames and is therefore not depicted separately.

3.10.2 Rigid Belt

This model is used to provide an ideal belt for the tire models that do not imply a dynamic belt behavior. Hence, the model is shown in Figure 3.37 and is obviously quite simple.

3.10.3 Torsion Belt

The torsion belt is useful for simulations where just longitudinal properties are of interest. For this purpose, a quite simple model is presented in Figure 3.38. An *Actuated Revolute* is used to model the flexibility combined with a parallel spring damper element for the determination of the belt's behavior. As no more precise model than the parallel spring damper system was found in literature, no more complex models for the behavior have been added, but expanding the model should not be any problem.

The initialization was again done simply by setting the initial angle of the revolute φ to zero as well as the angular velocity ω .

3.10.4 Longitudinal Lateral Belt

This model has a certain similarity with the *Contact Bond With Dynamics* presented in Section 3.7.3 on Page 68 and is depicted in figure 3.39. Therefore, it also does not work in combination with an uneven surfaces which's function computing the elevation is not derivable analytically. The main difference is that the inbuilt flexibility is at the center of the rim, not at the contact point. The other functions are quite similar, letting the belt translate in longitudinal and lateral direction. The difference between the models results from the different forces acting on spring and damper elements, due to the translational element *Center To Contact Point* that connects the contact point and belt center. This is also the reason why using the *Longitudinal Lateral Belt* in combination with the *Contact Bond With Dynamics* does not make much sense.

For the initialization, the longitudinal and lateral positions as well as the corresponding velocities for `frame_a` and `frame_b` have been set to the same value in longitudinal and lateral direction. This is done that way due to the fact that these are the directions where the only difference would be possible.

3.10.5 Quad Order Two Belt

The *Quad Order Two Belt* is a quite intuitive model. The *Rim* model in Figure 3.40 consists of four fixed translations that connect the center of the rim to the outer diameter of the rim. The pretensioned springs connect the outer radius of the rim with another four fixed translations that are part of the belt. These again model the ideal

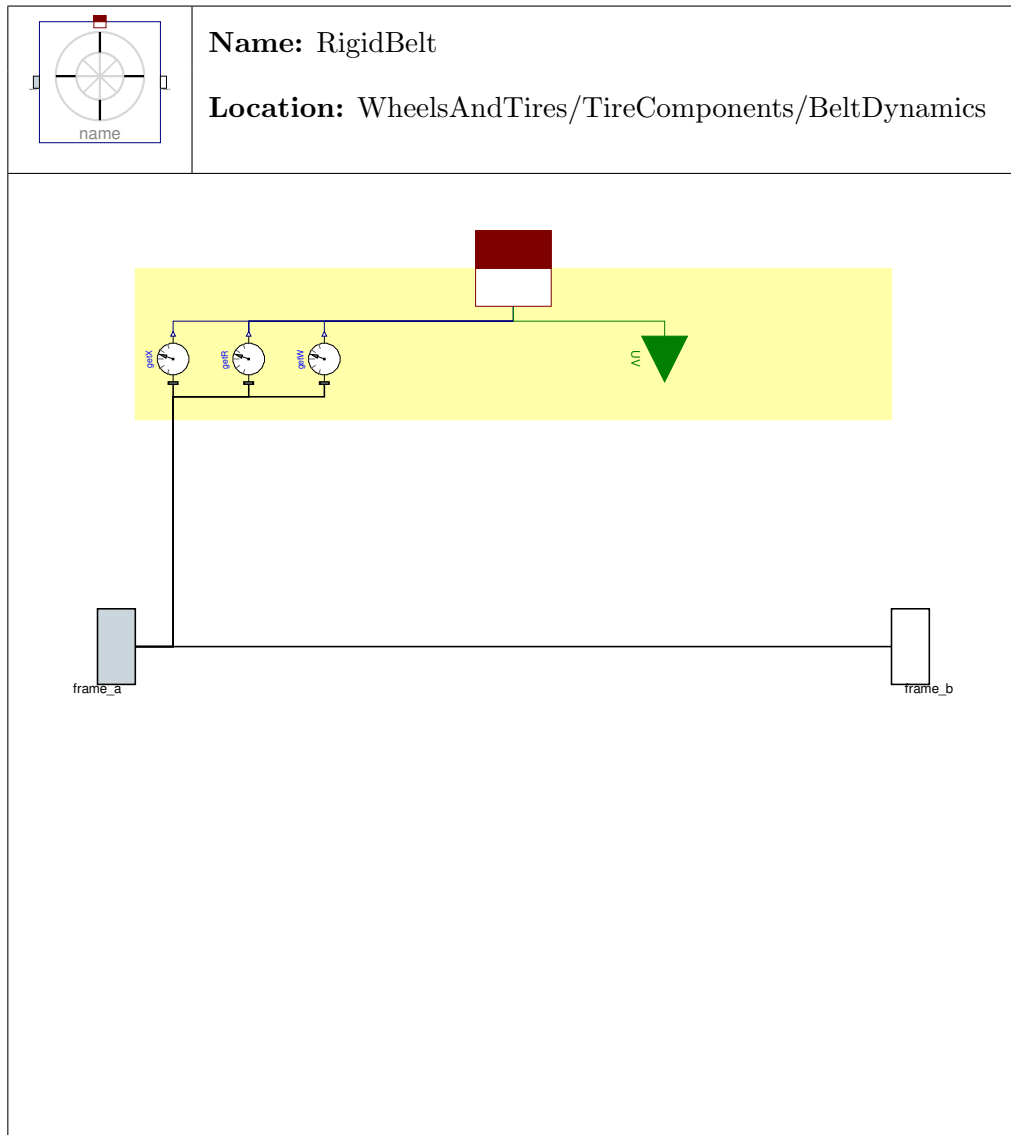


Figure 3.37: Model of an ideal belt without any dynamics.

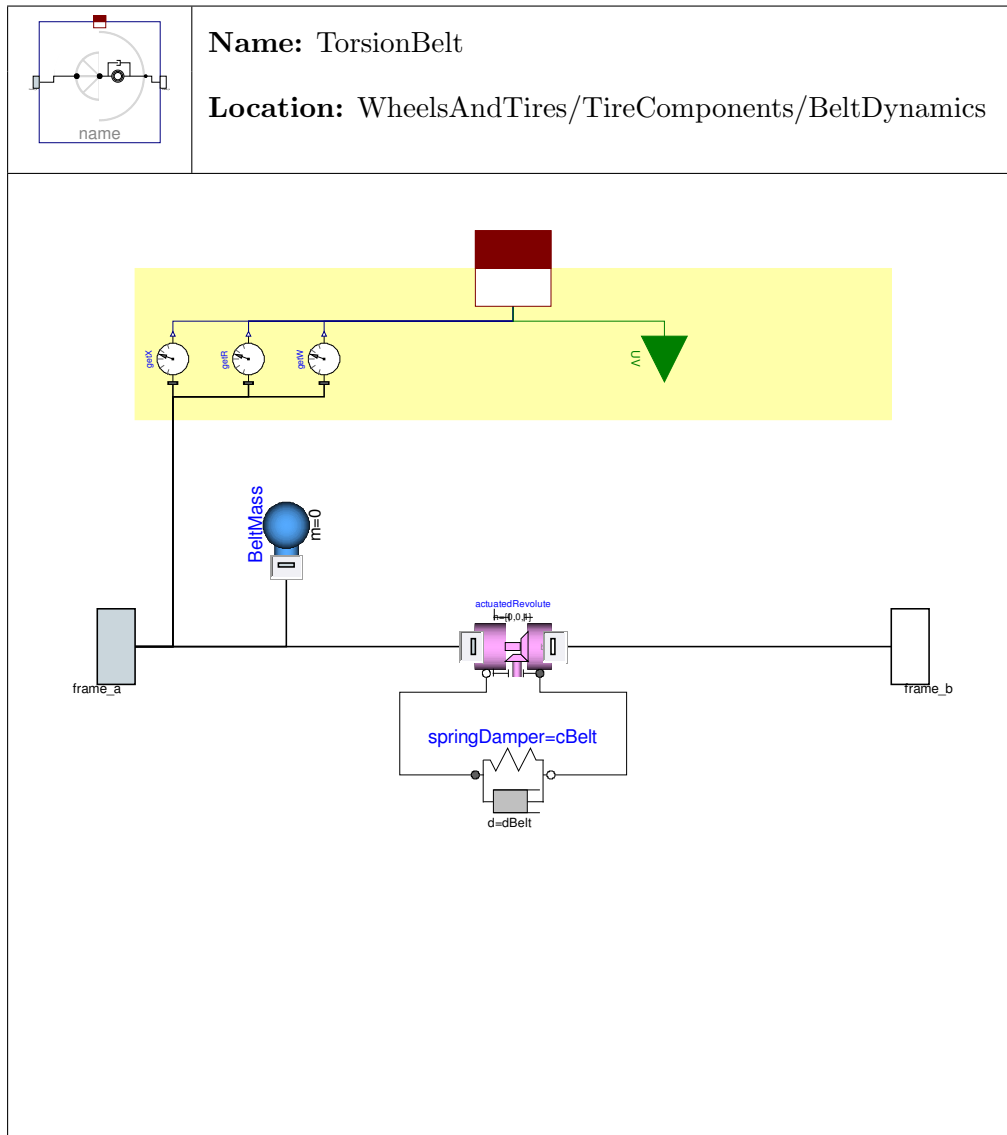


Figure 3.38: Model of a belt able to rotate around its spinning axis.

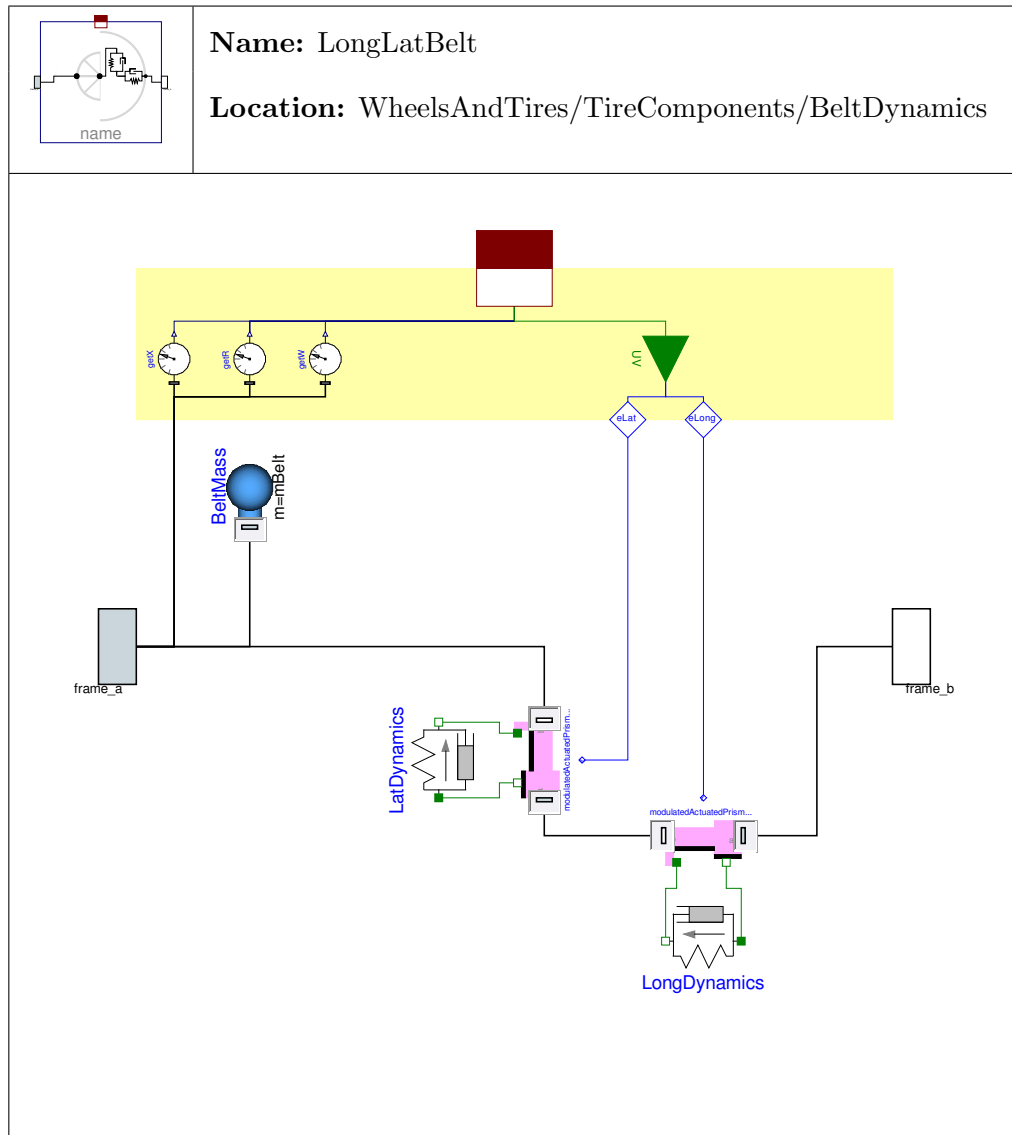


Figure 3.39: Model of a belt able to translate in longitudinal and lateral direction.

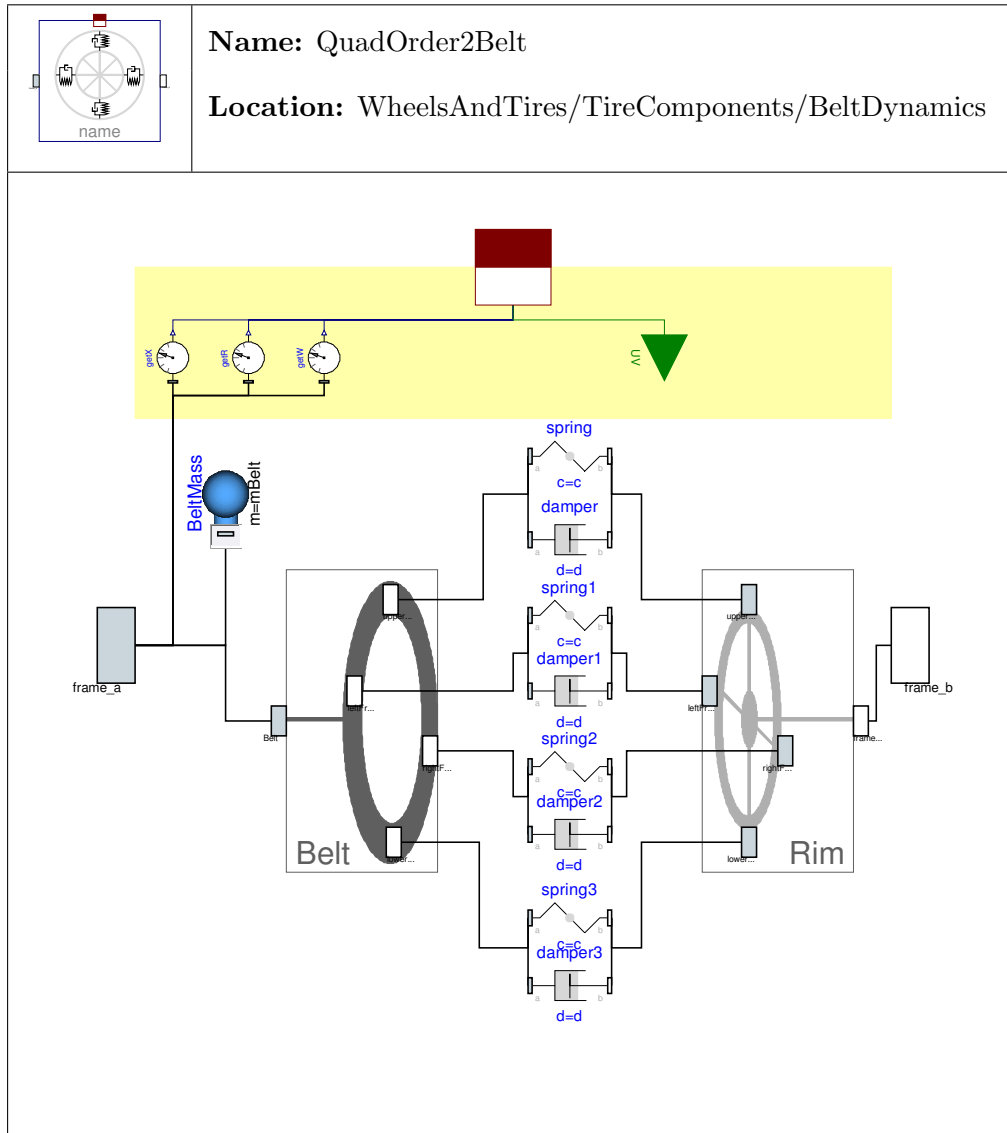


Figure 3.40: Model of a belt defined by four spring damper systems and eight translational elements.

connection from the outer diameter of the belt to the virtual center point of the belt that is the “output” of the model. The springs model the belt in a way that their unstretched length is the height of the side wall without any inflation pressure. The actual tension is reasoned by the inner pressure of the tire.

The model is probably not a very good description of a real tire though, because a deformation at the contact point does move the whole ideally stiff belt due to the ideal translational elements in the `Belt` model. Additionally for every deformation all the four spring/dampers combinations respond with a force what is not realistic either. Without the dampers or with a zero damping constant, the tire will never stop oscillating when there is no contact with the ground, which is unrealistic as well. So further consideration is necessary to make this model really useful. It was still left in the library due to the fact that the initialization is non-trivial. This way the *Quad Order Two Belt* can be used as a working base for further extensions.

The initialization is non-trivial because `Dymola` will initialize the `BeltMass` with angle and position set to $\{0,0,0\}$ regardless of where the rim is when no manual adjustments are made. The translational properties as well as the rotational velocities can simply be initialized in the same way it was done in the preceding models by putting the following equations in the initialization section.

$$frame_a.P.x = frame_b.P.x \quad (3.91)$$

$$frame_a.P.v = frame_b.P.v \quad (3.92)$$

$$frame_a.P.w = frame_b.P.w \quad (3.93)$$

Problems come up with the angle of the mass that cannot be aligned with

$$frame_a.P.R = frame_b.P.R \quad (3.94)$$

because the rotation matrix `R` is a highly redundant description resulting in a overspecified system of equations for the initialization problem. One possibility to overcome this is to compute the three Cartesian angles and then the `BeltMass` has to be initialized with these starting angles. This can be done by adding the following code in the equation section.

```
when {initial()} then
  Ra = transpose(frame_b.P.R);
  phi[2] = atan2(-Ra[3,1], sqrt(Ra[1,1]^2+Ra[2,1]^2));

  if abs(phi[2]) < PI/2-tol or abs(phi[2]) > PI/2+tol then
    phi[3] = atan2(Ra[2,1]/cos(phi[2]), Ra[1,1]/cos(phi[2]));
    phi[1] = atan2(Ra[3,2]/cos(phi[2]), Ra[3,3]/cos(phi[2]));
  elseif phi[2] > 0 then
```

```
    phi[3] = 0;
    phi[1] = atan2(Ra[1,1],Ra[2,2]);
else
    phi[3] = 0;
    phi[1] = -atan2(Ra[1,1],Ra[2,2]);
end if;
end when;
```

These equations can be found e.g. in [Cra89]. The singularity at $\text{phi}[2] = \pi/2$ is caught by an `assert` statement as there is no unique solution and one angle has to be set to a guess value. The equations are put in the `when {initial()}` clause in the equation section, because if they were used in the initial equations these variables would have to be given dummy values in the `equation` section.

One problem with the dynamic belts is that the relative rotation of the belt is not taken care of in the code section that is responsible for finding the contact point. Therefore when using the *Quad Order 2 Belt*, errors in the determination of the contact point will occur. The solution was not further investigated on because the *Quad Order 2 Belt* has to be revised anyway. Probably using the relative rotational matrix of the two frames would lead to a correct result somehow.

3.11 Utilities

This section describes the models that were created because they were needed to model certain special effects and are not contained in the *Bond Library* (`BondLib`) or *Multi Bond Library* (`MultiBondLib`). This distinction was made in the library as well, splitting the utilities into *Additional Bond Graph* elements and *Additional Multi Bond* elements.

This section also describes the *Icons* that are useful for quick changes in the library's optical design.

3.11.1 Additional Bond Graphs

The following sections describe the elements used for the tire models, but which are not contained in the *Bond Library* [CN05].

3.11.1.1 Angle

There are two reasons for this model to exist. The first is a design issue that restricts the use of the original *Angle* element of the *Bond Library*. This issue can lead to a problem with one equation to many in the system of equations due to the connection from `phi_ref` to `BG2Rot.phi` (the real port of the “BG → Rot” element in Figure 3.41). Therefore this connection was canceled in this element. The second reason is that an *approximated derivative block* was used instead of an analytically calculating one. The main reason for that is consistency with the element *Position* (Section 3.11.1.3) where an analytical derivative block leads to problems in certain models.

3.11.1.2 Angular Velocity

The *Angular Velocity* block is added to the library for the same reasons as the *Angle* block. It is the one equation too many when connected e.g. to an actuated revolute joint.

The model is the same as for the *Angle* class without the `derivative` block.

3.11.1.3 Position

The *Position* model has the same background as the *Angle*. The main difference is that here translational elements are used instead of rotational ones.

3.11.1.4 Position With S

The *Position With S* element is basically the *Position* element with the additional equation for the position added from the input to the “BG → Tr”¹⁴ element. So the only difference between the *Position With S* element and the original *Position* from the *Bond Library* element is the non ideal derivative block used instead of the ideal one. This is done because the element is used in the *Vertical Dynamics* classes. The functions that are used to determine elevation and normal vector of the surface are not derivable analytically (what the ideal `derivative` block does). Therefore, the non ideal block has been used introducing just little difference to the ideal solution. The time constant which is the reason for the non ideal behavior, is a parameter that can be set to very small values. For the default value, a time constant was used that enables good accurateness with very little influence on simulation time.

¹⁴The translational version of the *BG → Rot* element in Figure 3.41.

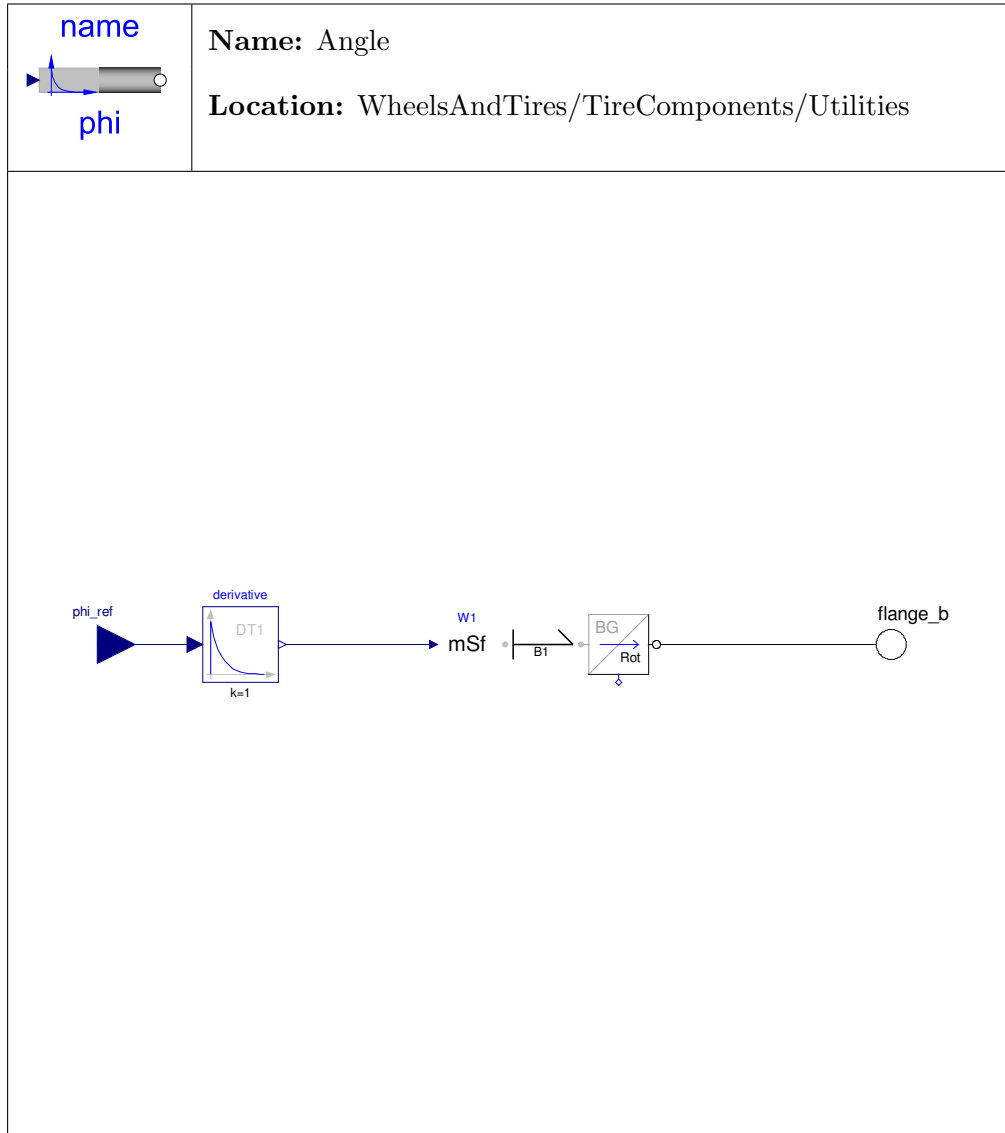


Figure 3.41: Alternative model of an angle source with one less equation compared to the original one from the *Bond Library* and a non ideal derivative block.

3.11.1.5 Velocity

The velocity block leaves out the superfluous equation contained in the original *Speed* block of the *Bond Library* for the position to allow simulation.

3.11.1.6 Serial Spring Damper

In contrast to the *Modelica Standard Library*, the *Bond Library* does not allow a serial connection of a spring and a damper element without a mass in between the two elements. Therefore, an element was developed that is used in the *Gehmann Elevation* model. It is derived from the original model of a spring in the *Bond Library* and shown in Figure 3.42.

3.11.1.7 Elevation Gap

The model shown in Figure 3.43 allows a force free lift of the tire from the ground. This is realized by a force sensor that measures the force generated by the externally connected elements in order to supply an inversely connected source of force if there is no contact to the ground. This makes it possible to have external 1D translational elements determining the vertical dynamics of the tire in the case of contact which is very convenient. E.g. the *Kelvin Elevation* uses a simple spring damper system for the vertical dynamics, which can be extended easily to more complex ones like the *Gehmann Elevation*.

There is one disadvantage of this model. This is that in the case of decreasing `penetrationDepth`, the damper introduces a force that “pulls” the tire back onto the street. This happens because of the damper’s characteristic. If this force gets bigger than the force of the spring that “presses” the tire away from the street, the tire “sticks” on the street, what is not the real physical behavior. This is a serious issue if the damping coefficient is big compared to the spring constant. As tires usually have very low damping coefficients and are relatively stiff, that fact is neglected in favor of the comfortable way to specify more complex vertical dynamics. If there are problems due to the sticking effect, a model presented in Section 3.9.4 on Page 75 overcomes this effect utilizing a modeling technique from the Modelica 3.0 Library. Still both versions are part of the library because both have advantages. One is simple to customize, the other has no sticking.

3.11.2 Additional Multi Bond Graphs

The models contained in this section are created as they are necessary in the tire models and not contained in the *Multi Bond Library*.

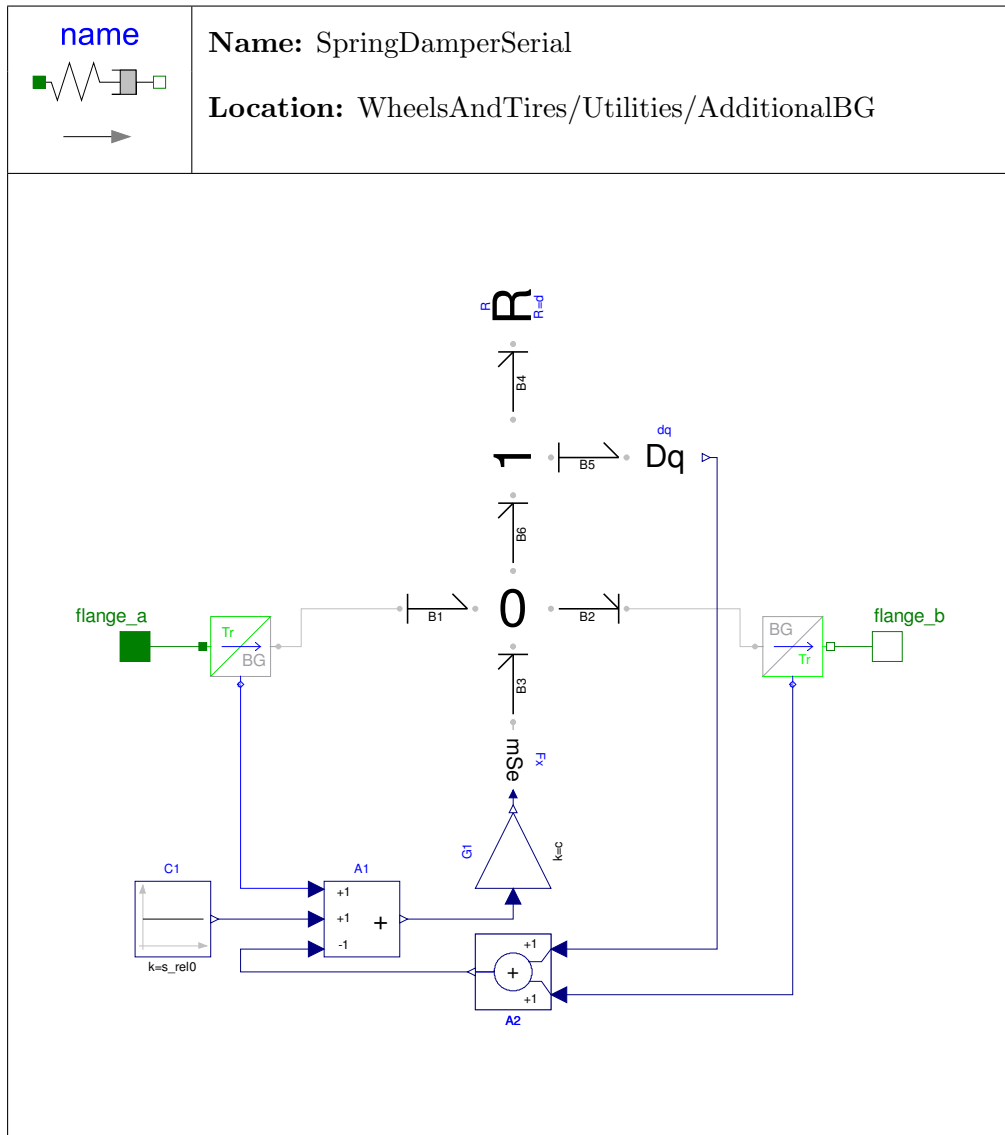


Figure 3.42: Bond graphic model of a serial spring damper system.

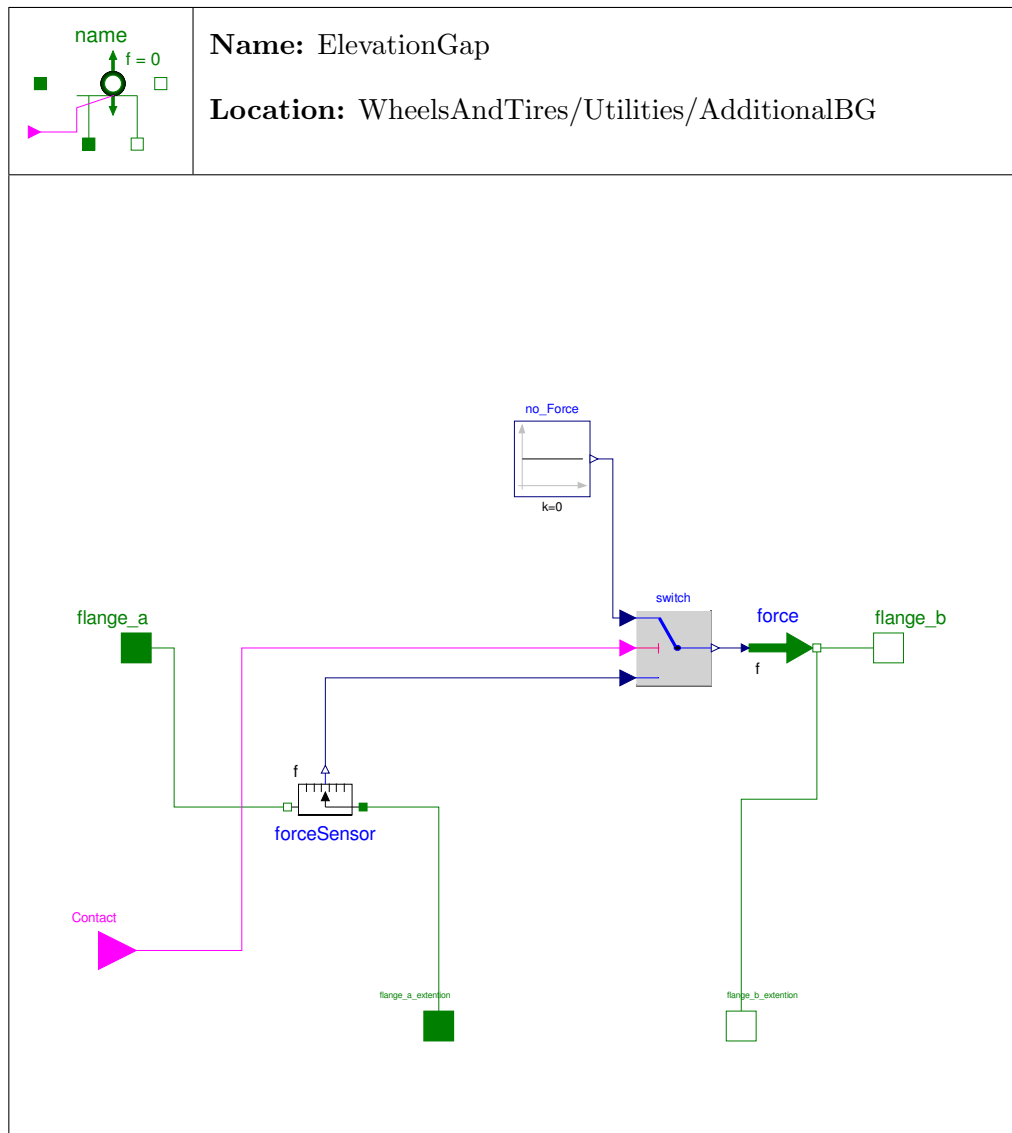


Figure 3.43: Model enabling a force free lift from the ground (with sticking effect).

3.11.2.1 BooleanSignal

The *Boolean Signal* (depicted in Figure 3.44) is created to handle the **Contact** signal

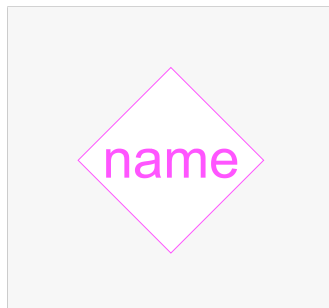


Figure 3.44: The icon of the *Boolean* signal.

in an a-causal way in the communication structure described in Section 3.3 on Page 23. It is simply the Boolean version of the *Real Signal* in the *Multi Bond Library*.

3.11.2.2 Modulated Effort Transformer 2

The *Modulated Effort Transformer 2* is the modulated version of the *Effort Transformer TF_effort* in the *Multi Bond Library*. The equations used are

$$e_B = M \cdot e_A \quad (3.95)$$

$$f_A = M' \cdot f_B \quad (3.96)$$

whereas M is an input not a parameter in the modulated version, e is the effort of the corresponding input A or B and f is the adequate flow variable.

3.11.2.3 Modulated Translational Transformer

The *Modulated Translational Transformer* is again the modulated version of the *Translational Transformer* in the *Multi Bond Library*. It computes a cross product connecting forces and torques as well as velocities and angular velocities in an a-causal way.

There are actually two versions of this model, one using a radius (translational) that is the input to the model, the other using an amplification in combination with directional information that represents the radius. These are used in different models depending on which suits the application better.

The equations for the “radius version” are the following

$$f_2 = \text{cross}(f_1, r) \quad (3.97)$$

$$e_1 = \text{cross}(r, e_2) \quad (3.98)$$

whereas the “amplification version” uses equations with an additional variable `ampl`.

$$f_2 = \text{cross}(f_1, \text{ampl} \cdot r) \quad (3.99)$$

$$e_1 = \text{cross}(\text{ampl} \cdot r, e_2). \quad (3.100)$$

3.11.2.4 Modulated Translation

The *Modulated Translation* is the element that relates the position of the two frames depending on \mathbf{r} , a vector that connects the two. As in Section 3.11.2.3, there is a version with the vector as a “real” (Equation (3.101)) vector and a second version using an amplification in combination with directional information (Equation (3.102)) representing the vector, where either one can be used depending on the application. For Equation (3.101) the input vector \mathbf{eR} is normalized and the connections are tested whether something is connected, and are set to default values if not.

$$x_2 = x_1 + r \quad (3.101)$$

$$x_2 = x_1 + (eR \cdot \text{ampl}) \quad (3.102)$$

3.11.2.5 Modulated Actuated Prismatic

The model shown in Figure 3.45 is derived from the *Actuated Prismatic* element in the *Multi Bond Library* [Zim06]. The difference is that for the original prismatic element the direction of action can be set via parameter values and the environment determines how the model behaves during simulation. E.g. if the model is attached to a revolute joint the direction of action changes with the angle of the revolute joint, which makes total sense from the physical or mechanical point of view.

For some models of tires though these elements have to act in given directions independently of what happens at the frames they are connected to. That is exactly what the *Modulated Actuated Prismatic* does. The input \mathbf{r} sets the direction of action independent of the connected frames. This may not be a reasonable model in terms of mechanics, but it is very useful in tire modeling e.g. due to the fact that no initialization is needed and it is guaranteed to act e.g. in the longitudinal direction in every case.

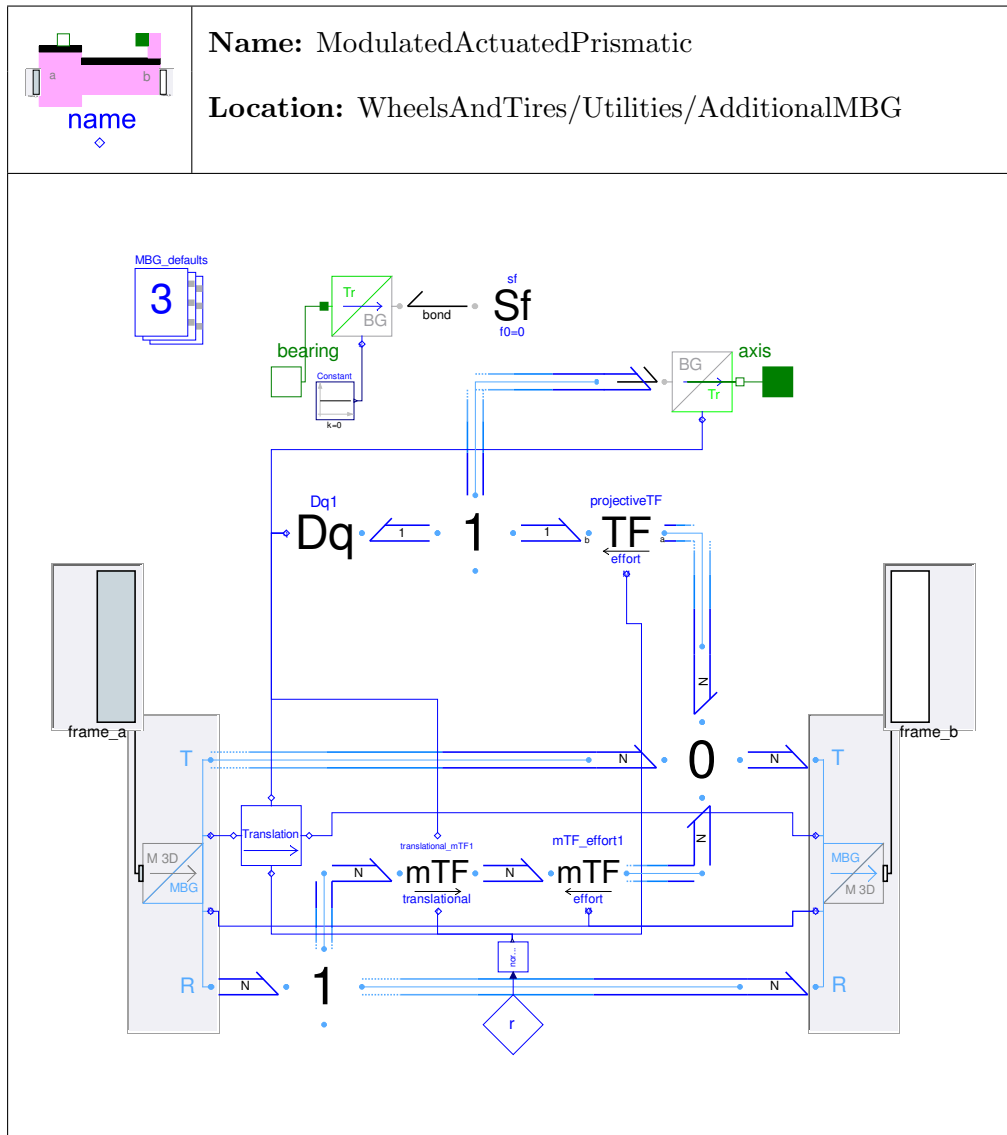


Figure 3.45: Model of a *Modulated Actuated Prismatic* element.

3.11.2.6 Absolute Sensor

The *Absolute Sensor* is just a very slight modification of the original element in the *Multi Bond Library*. The difference is that the original element sets the unit of the output to Newton, which does not suit the application in the *Wheels And Tires Library*. So this section has been deleted from the original model and this is the only difference.

3.11.2.7 Normalize

This model is just a graphical version of the Modelica function *normalize* and is intended to be used in graphical models to connect the contained elements more conveniently. It is done for vectors of dimension 3 only, which makes it exclusively suitable for 3D applications.

3.11.3 Icons

Figure 3.46 shows the icons used for the corresponding packages and models. Models in the library are created extending these base models to ensure a common look and enabling a fast changes in the optical design of the library.

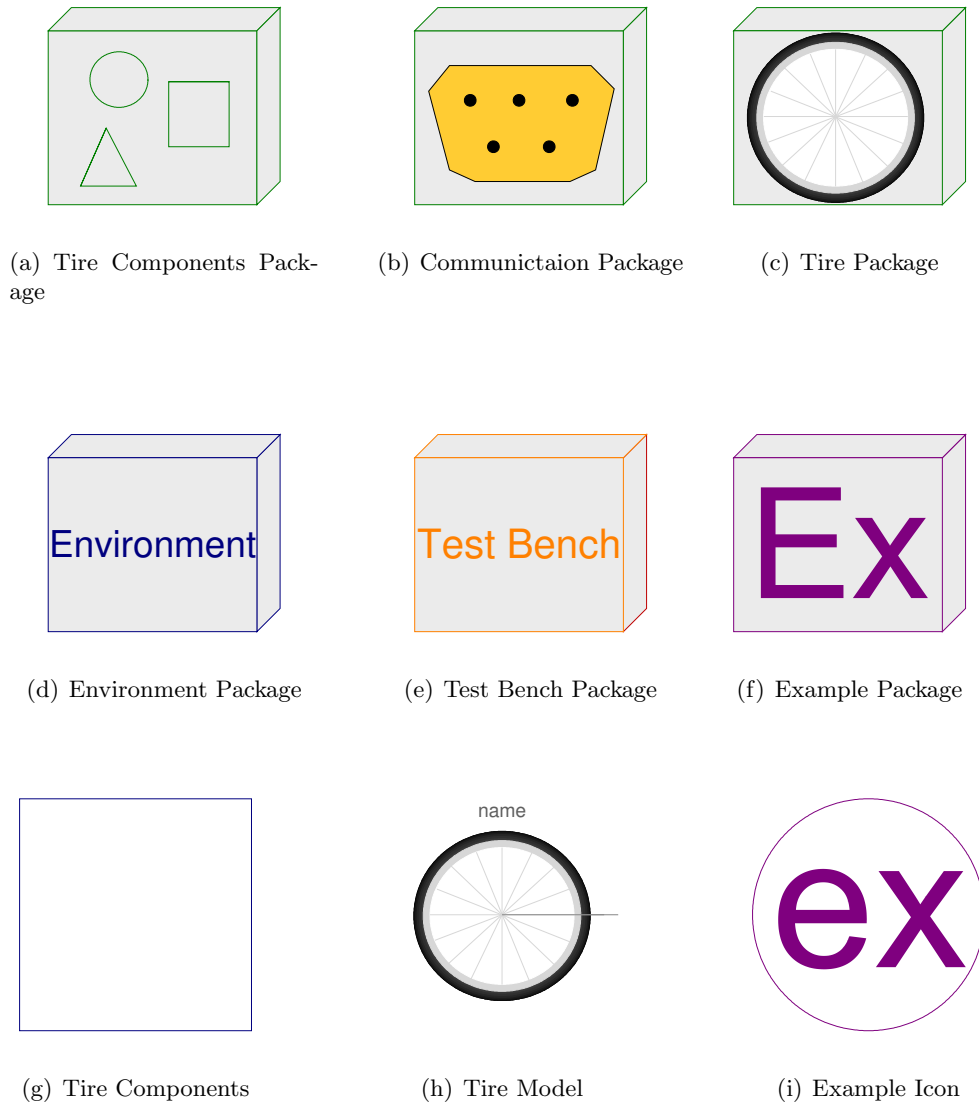


Figure 3.46: Icons used in the library.

4 Provided Tire Models

All tires presented in this chapter are basically built up like the ideal tire in Figure 3.4 on Page 21. All connect seven objects defining the tire's properties with three different connectors. Two of these are the customized bus connections described in Section 3.3, the other is the rigid connection between frames defined by the *Multi Bond Library*. These rigid connections transfer mechanical properties and describe an ideal connection between the two connected frames. Different properties can simply be realized by choosing different classes. As the interfaces between the classes are standardized, no further adaptation is necessary. If no parameter passing from the top level is realized, the tire is usable already. Although for convenient use parameter passing combined with a well engineered initialization is a postulate. How parameters can be aggregated is shown in Section 4.2, whereas the different strategies of initialization are demonstrated in Section 4.3. The ready-made tire models delivered with the library are presented in the following section.

4.1 Predefined Models

Figure 4.1 shows the different tire models and the components they utilize.

For different combinations of frictional components, a class has to be created. Therefore, the amount of combinations of different frictional components was kept as small as possible, creating the following frictional classes.

- *Ideal Friction* – `IdealFriction`, modeling ideal frictional behavior.
- *Dry Friction with Roll Resistance* – `DryFrictionRollRes`, adding slip based on the sliding dry friction model and constant rolling resistance.
- *Advanced Friction* – `AdvancedFriction`, the most advanced version of friction possible with the current version of the library including all covered frictional effects.

Looking at Figure 4.1, two elements in brackets are found for the *Linear Overturning Torque*. These models will compute the overturning torque to be zero because a width of the tire is required to get an overturning stiffness $\neq 0$.

Classes	Sub Classes	Tires	Ideal	Slipping	Advanced	Dynamic	Advanced	Dynamic	Ideal	Slipping	Advanced	LongLatDynamics	QuadDynamics	
			Slim				Circ.		Belt					
Belt Dynamics	Rigid Belt		x	x	x	x	x		x	x	x			
	Torsion Belt							x						
	Longitudinal Lateral Belt											x		
	Quad Order 2 Belt												x	
Contact Physics	Contact Bond without Dynamics		x	x	x		x	x	x	x	x	x	x	
	Contact Bond with Dynamics					x								
Center 2 Contact Point	Center 2 CP Bond		x	x	x	x	x	x	x	x	x	x	x	
Elevation and Vertical Dynamics	No Elevation		x						x					
	Kelvin Elevation									x	x			
	Gehmann Elevation			x									x	
	Elasto Elevation				x	x	x	x					x	
Friction	Slip Properties	No Slip	x						x					
		Sliding Dry Friction Longitudinal												
		Sliding Dry Friction		x							x			
		Rill Friction												
		Combined Friction			x	x	x	x				x	x	x
	Load Influence	Linear Load Influence	x	x						x	x			
		Quadratic Load Influence			x	x	x	x				x	x	x
	Roll Resistance	No Roll Resistance	x							x				
		Constant Roll Resistance		x							x			
		Speed Depending Roll Resistance			x	x	x	x				x	x	x
	Bore Torque	No Bore Torque	x	x						x	x			
		Linear Bore Torque												
		Combined Bore Torque			x	x	x	x				x	x	x
	Self Aligning Torque	No Self Aligning Torque	x	x						x	x			
		Self Aligning Torque Rill			x	x	x	x				x	x	x
	Camber Force	No Camber Force	x	x						x	x			
Linear Camber Force														
Combined Camber Force				x	x	x	x				x	x	x	
Overturning Torque	No Overturning Torque	x	x						x	x				
	Linear Overturning Torque			(x)	(x)	x	x				x	x	x	
Geometry	Flat Disc		x	x	x									
	Circular Tire					x	x	x						
	Belt Tire									x	x	x	x	
Rim	Spoked Rim		x	x	x	x	x	x	x	x	x	x	x	

Figure 4.1: Predefined tires available in the *Wheels and Tires* Library.

4.2 Parameter Aggregation

As many of the parameters are commonly used in every tire model, a *Commons* package has been created to gather all these parameters. The different tire models can then simply extend these partial models and will feature all the parameters from these files. So there is a central place to change parameters in one go, instead of doing it in every tire model.

The created “parameter” files are the following.

- `ParametersCommon` – *Parameters Common*
- `ParametersDryRollRes` – *Parameters Dry Friction with Roll Resistance*
- `ParametersAdvancedFriction` – *Parameters Advanced Friction*

The first of these gathers the parameters that are common for every tire model, like the rim mass or different variables for the visualization. *Parameters Dry Friction with Roll Resistance* model contains all the parameters used by the *Dry Friction with Roll Resistance* as the last (partial) model does for the *Advanced Friction* class. These parameter models can be used in combination with the corresponding friction class, independent of the other used classes. No further gathering was realized because the cost-benefit ratio is inadequate.

Generally, it has to be mentioned that some of the parameters have to be adapted to suit the actual application. But as the user is expected to be familiar with tire modeling or will even add own models to the library, that is not considered to be a problem.

4.3 Initialization

The initialization is quite different for ideal models and their slipping counterparts. The major differences regarding initialization are that predefined ideal models are not able to lift from the ground and obviously do not slip. Therefore, two different initialization strategies have been implemented, which unfortunately could not be aggregated for all models as is possible with the parameters. This is not possible due to the fact that variables cannot be initialized in a different model than they are used in. For this reason, every tire has its own equations for initialization.

Another very important aspect of the initialization is that every model adding dynamics to the system like the *Contact Point Dynamics* or some kind of belt dynamics is initialized in a way suitable for the unloaded steady state directly in the model adding the dynamics. That makes it possible to just have two different initializations. So every tire provided in this library will initialize with a `PenetrationDepth` of 0 then start to penetrate the ground to build up the necessary normal force.

4.3.1 Initialization of Non-Slipping Tires

The initialization of the tires with ideal slip behavior is relatively straightforward. The approach used in the *Wheels and Tires* Library is based on the models of the ideal tire from the *Multi Bond Library* [Zim06]. It can be done by setting the rotational speed of the rim mass. The holonomic constraint caused by the ideal slip behavior and the *Center to Contact Point* model directly converts this into a translational velocity of the rim mass. As it can be more convenient to set the translational velocity, an option called `useTranslationalVelocities` was added to the initialization, enabling the user to set the translational velocities directly. Again the model converts the translational velocity into rotational speed by the model geometry automatically when setting the translational velocity of the rim mass.

There are a couple of restrictions to these ideal models that become obvious when looking at the code more closely. The first is the *No Elevation* class fixing the contact point somewhere in the x-z-plane. Therefore, the initialization just sets coordinates in x and z direction not considering inputs that would put the tire above the surface in the air or penetrating in the ground as this is not possible. This also applies for the translational speed in vertical direction, as the tire cannot move in this direction. Another thing important to notice is that for the usage of translational velocities in the initialization, the yaw rate is set to 0 to ensure a reasonable start of the tire.

4.3.2 Initialization of Slipping Tires

Tires that can slip have to be initialized in a more complex way to cover all possible combinations of translational velocities, sliding velocities and angular velocities. A tire that is initialized with a speed about its rotational axis will directly transform this rotational speed to sliding velocity as the rim mass is initialized with a translational speed of 0 in that particular case. This will make the tire slip for some time, transforming the rotational energy to translational with losses due to the slip. This is not convenient in general, as the user usually wants to initialize a model with an exact translational speed e.g. for stability analysis of a bicycle. Therefore a more complex version of initialization was implemented. To enable the user to switch from ideal models to this more complex conveniently, an additional variable `useAdvancedInitialization` was added. This disables the simple initialization used for nonslipping tires that is possible for the slipping ones as well. The user is now able to set two of the possible velocities¹ manually, letting the model determine the third automatically. So the user can set e.g. the translational speed to a certain value and the slip to zero, making the model behave as the ideal version at least during the initialization.

Another option called `onSurface` is provided to enable the user to make the tire initialize on the surface that can be uneven in the event of these models. The tire is

¹Which are the rotational velocities, the translational velocities and the sliding velocities.

initialized with the rim mass at height 0 by default otherwise. This would stretch the spring of the vertical dynamics to an unrealistically big value making the tire respond to the very big normal force resulting in a very big acceleration normal to the surface. This can be overcome with the initialization of the position. Problematic with this version of initialization is that e.g. the positions of 3 of the tires of a simple car model are defined by holonomic constraints from the position of the fourth tire. Therefore, initializing the position of all tires of this model can be problematic. In this case the `onSurface` option becomes useful. The option is automatically ignored if the position is initialized directly by the user.

There are two more properties that have to be kept in mind with the advanced initialization. The first is that the normal component is ignored in the initialization of translational velocities. Secondly in the case of setting the rotational speed, the normal component it is set to 0. Generally, it is suggested to initialize the tire on a flat part of the surface.

4.4 Defining New Tires

For the user who has to define new tire models, it is suggested to copy the most similar version of a tire that is available in the library. Afterwards, the user can exchange the objects that were adapted or created by right-clicking on the object to be replaced and choosing *Change Class* from the Dymola context menu. If the used simulation environment does not support this function the model can also be exchanged in the Modelica code directly which is equal to the *Change Class* option. When changing sub-classes a good approach is to duplicate the most similar class and then modify the equations or models contained. This way, models are likely to not have basic errors, as all the parameters are left in the code which can be useful and the names in the model stay the same. Basically, the names of every model can be changed because the communication structure does not care about the names of the objects, but the initialization is done utilizing some of the names in the described example tires. Therefore, difficulties can arise with the initialization when changing object names.

As mentioned before, the initialization of components adding dynamics to the model is usually done in the dynamic model directly making them exchangeable, without having to care about the initialization of the overall model. So it is suggested that future models, which add dynamics to the model, are initialized in the same way. Although e.g. initializing everything from a central code section could add some clarity, the initialization structure would have to be adapted with every change in the tire's structure which is important for the initialization.

5 Environment

The environment in the current version of the library is limited to even or uneven surfaces which the tire can have contact to. A very simple approach to describe an even surface is to set the contact point to a constant height and the frictional coefficient to a constant value as well. This simplifies some of the essential parts of tire modeling drastically. Although it may not be the very core of the library, it was decided to create an uneven surface, even if it is still a rather simple one. The main aspect of this part is to create a kind of framework that guides future modelers through expanding the library with more complex surfaces that the tire model is compatible with.

The following sections describe the *Surface Base* that is intended to form the framework. Afterwards, the section *Surface* describes the implementation chosen in the *Wheels And Tires* library.

5.1 Surface Base

This partial model defines the functions necessary to calculate the behavior of the tire, including the following.

- `get_eN_Base` – returns the normal vector of the surface at the actual x and z coordinates.
- `get_elevation_Base` – returns the y coordinate of the surface at the actual x and z coordinates.
- `get_mu_Base` – returns the frictional coefficient at the actual x and z coordinates.

From the description above, one can see that inputs for all functions are the x and z coordinates. How the results of these functions are determined is left open and can be defined by the user.

5.2 Surface Class

Basically the method to be implemented must be able to compute elevation (the y coordinate shown in Figure 5.1) and the normal vector at every point of the surface. There are many substantially different approaches whereas the chosen one is a rather simple and conveniently configurable one.

5.2.1 The used Method

The implemented version of the *Surface* in the *Wheels and Tires* Library is based on the method shown in the following section and introduced in [AS93]. It is an interpolation in a unit square based on four y values and the eight corresponding partial derivatives. A sketch of the unit square is shown in Figure 5.1.

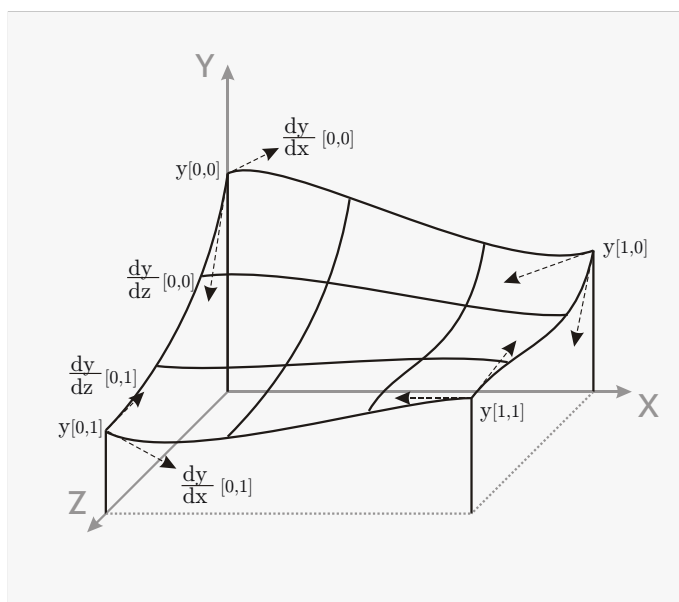


Figure 5.1: The foundation for the interpolation used to compute the elevation y and normal vector in a square.

5.2.1.1 Limitation

Limitations are given due to the single contact point model used. The radii describing the surface have to be significantly greater than the tire's radius. This ensures that only a single contact point occurs.

As the model uses the introduced interpolation structure, rather than e.g. a two dimensional mathematical function, the results are not derivable analytically by Dymola. This limits the application of this surface model to *Vertical Dynamics* that have a non ideal derivative block to calculate the normal velocity of the ideal penetration depth. These are all models in the *Wheels and Tires* library except the *No Elevation* class.

5.2.1.2 Interpolation in one rectangle element

Starting from Figure 5.1, the value of y as well as $\frac{dy}{dx}$ and $\frac{dy}{dz}$ at an arbitrary point in the square can be found by the following equations. The equations are a modified version of the ones in [AS93] in order to have y and the corresponding partial derivatives as an output rather than z . This better suits the simulation environment Dymola.

$$y(x, z) = (\mathbf{HY})^T \underbrace{\begin{bmatrix} y(0, 0) & y(1, 0) & \frac{dy}{dz}(0, 0) & \frac{dy}{dz}(1, 0) \\ y(0, 1) & y(1, 1) & \frac{dy}{dz}(0, 1) & \frac{dy}{dz}(1, 1) \\ \frac{dy}{dx}(0, 0) & \frac{dy}{dx}(1, 0) & 0 & 0 \\ \frac{dy}{dx}(0, 1) & \frac{dy}{dx}(1, 1) & 0 & 0 \end{bmatrix}}_{\mathbf{G}} (\mathbf{HY}) \quad (5.1)$$

with

$$\mathbf{X} = \begin{Bmatrix} x^3 \\ x^2 \\ x \\ 1 \end{Bmatrix} \quad \mathbf{Y} = \begin{Bmatrix} y^3 \\ y^2 \\ y \\ 1 \end{Bmatrix} \quad \mathbf{H} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -1 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \quad (5.2)$$

This is the basis for the function `get_elevation` that returns y at every possible position in the rectangle as an output.

In order to compute the normal vector, the following functions are useful.

$$\frac{dy}{dz}(x, z) = (d\mathbf{HX})^T \mathbf{G}(\mathbf{HZ}) \quad (5.3)$$

$$\frac{dy}{dx}(x, z) = (\mathbf{HX})^T \mathbf{G}(d\mathbf{HZ}) \quad (5.4)$$

with

$$d\mathbf{H} = \begin{bmatrix} 0 & 6 & -6 & 0 \\ 0 & -6 & 6 & 0 \\ 0 & 3 & -4 & 1 \\ 0 & 3 & -2 & 0 \end{bmatrix} \quad (5.5)$$

from what the normal vector can be computed by

$$\mathbf{n}(x, y) = \begin{pmatrix} 0 \\ \frac{dy}{dz}(x, z) \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ \frac{dy}{dx}(x, z) \\ 0 \end{pmatrix} \quad (5.6)$$

with the normal vector resulting in

$$\mathbf{e}_n(x, z) = \frac{\mathbf{n}(x, z)}{|\mathbf{n}(x, z)|} \quad (5.7)$$

which is the mathematical background behind the `get_eN` function.

The transpose signs in the equation above indicate that a scalar product is to be used at these places. Whereas this is done that way in MATLAB, Dymola does automatically compute a scalar product with `vector · vector` as it distinguishes between vectors and matrices.

5.2.1.3 Combining multiple Rectangles of arbitrary size

In order to be able to describe more complex surfaces than the ones shown in Section 5.2.1.2, a possibility was created to add multiple surfaces to form one big surface. The major problem regarding this strategy is that the functions presented in the foregoing chapter just work with a single unit square.

The interpolation method is based on a square element with a side length of 1. This is not very convenient to describe bigger surfaces which are reasonable when modeling motorcycles or cars. The *Surface* class therefore enables the user to define the total length and width of the Surface as well as the amount of grid points representing the vertices of the (in general) rectangular elements shown in Figure 5.2(a). Therefore, the values returned by the `get_rectangle` function have to be scaled to unitary values which is done by the following equations.


```

rectangleX := LengthX/(nu-1);
localXNorm := localX/rectangleX;
X :={localXNorm^3,localXNorm^2,localXNorm,1};

```

Hereby `LengthX` denotes the total length of the surface, `nu` the number of grid points in the `x` direction and `localX` is the position in the currently “active” rectangular element. These values are then used to compute `X` and do the interpolation. The same is done for the `z` coordinate.

The number of connected rectangles is defined by `[nu, nv]` as depicted in Figure 5.2. The size of the corresponding matrices of `y`, $\frac{dy}{dx}$ and $\frac{dy}{dz}$ obviously has to match the

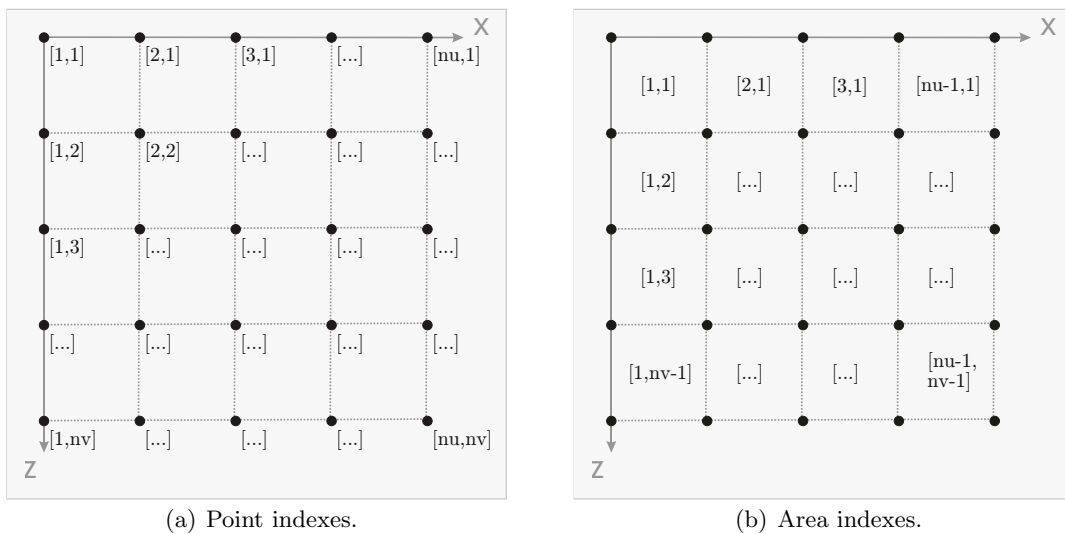


Figure 5.2: Coordinates for multiple rectangles assembled to form a more complex surface.

values of `nu`, `nv`.

Therefore, the function `get_rectangle` was created and returns the values given below, as this is useful for both functions `get_elevation` and `get_eN`.

- `xIndex` – The index in `x` direction, denoting the area which the tire is in (see Figure 5.2(b)).
- `zIndex` – Same as `xIndex` for the `z` direction.
- `inArea` – A boolean variable denoting whether the tire is in the area of the defined surface¹.

¹If the tire is not in the defined area, the elevation is set to 0 and the normal vector is set to `{0, 1, 0}`. This can be troublesome when the defined surface reaches below the elevation of 0, as then the

- `localX` – The position in x direction in the current rectangle.
- `localZ` – The same as `localX` in the z direction.

Figure 5.3 shows the parameter window for the configuration of a *Surface* class. Here

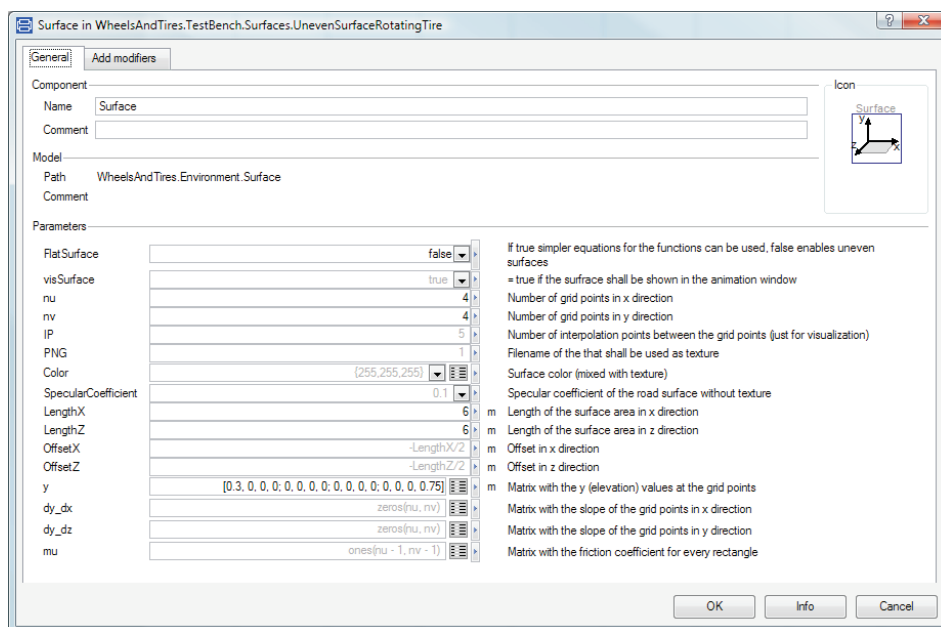


Figure 5.3: The parameter window for customizing the *Surface*'s properties.

a quite simple one is shown with just two values $\neq 0$ and all partial derivatives $= 0$.

5.2.1.4 The Visualization

For the visualization of the *Surface*, the *Surface Material* class is used. This is a class provided with Dymola enabling a visualization of parametric surfaces.

To utilize this class, one has to set the number of grid points `nu` and `nv` as they are shown in Figure 5.2(a). To be able to visualize the surface, one has to set x , y and z coordinates for every point in that grid. As the surfaces in the *Wheels and Tires* library have a regular grid, this can be done by a double loop for the x and z coordinates in a linear fashion. Building a grid for the visualization with exactly as many grid points used to define the single sections of the surface results in a visualization shown in Figure 5.4. This will return the same simulation results as the one in Figure 5.5 but with a confusing visualization. Therefore the parameter `IP` (see Figure 5.3 on Page 106)

limitation for a single contact point is not fulfilled. If the tire was at a elevation > 0 , it would simply fall down hitting a virtual and not visualized “floor” at 0 height.

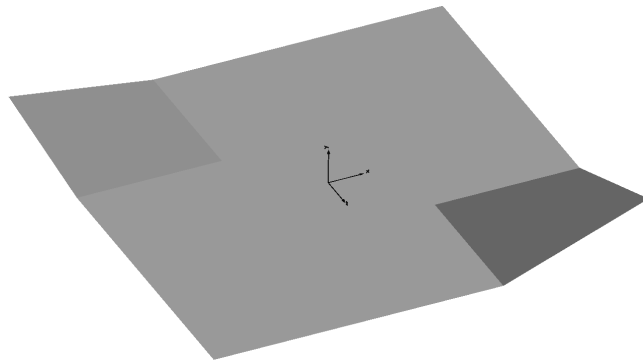


Figure 5.4: The result of the surface configuration shown in Figure 5.3 without any further interpolation for the visualization without a PNG as texture.

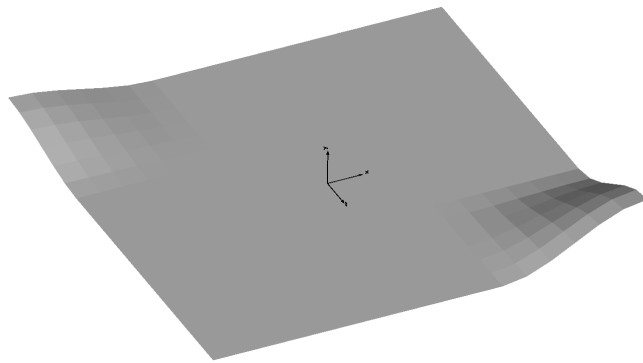


Figure 5.5: The result of the surface configuration shown in Figure 5.3 without a PNG as texture and five interpolation points between the vertices.

has been introduced. This adds IP points between the original grid points providing a much smoother visualization.

To determine the single y values for the visualization, basically the same function as shown in Section 5.2.1.2 on Page 103 is used. The full code for this visualization is provided in Appendix A.2.3.

It is important to notice that the interpolation just has optical reasons. The real values of y and eN the simulation is based on are determined totally smoothly and independent of the visualization.

5.2.1.5 The Flat Surface Option

As mentioned in Section 3.9.1 on Page 72, the models that have to derive the `penetrationDepth` are not usable in combination with these parametric surfaces. Therefore an option has been created that is especially useful if flat surfaces are simulated which is often adequate for vehicle dynamics simulations. If this option is enabled two major changes happen throughout the model. The first is that the `Surface` class disables the parameters `y`, `dy_dx` and `dy_dz` and switches to a visualization of exclusively flat surfaces. The second difference lets the `Geometry` class set the `y` coordinate of the contact point to 0. This enables the model to derive the `penetrationDepth` and therefore suitable for the *No Elevation* class. Additionally, `eN` is set to `{0,1,0}`.

5.2.1.6 The frictional coefficient

Another thing that can be customized with the *Surface* class is the frictional coefficient. This is done by the parameter `mu` which can be found in Figure 5.3 on Page 106, for all of the rectangles depicted in Figure 5.2(b) on Page 105. The transition between the single rectangles is a hard one so no kind of interpolation between the rectangles is implemented. Thus, the frictional coefficient changes step like when crossing a border between two rectangle elements.

5.2.2 Changing the Surface Class

Changing the *Surface* class could bring the advantage that a derivable version of a surface could be implemented. This would enable the usage of analytical derivation throughout the *Vertical Dynamics* class.

The most reliable version to implement a new *Surface* class would extend the existing *Surface Base*. This would ensure compatibility with the rest of the tire model. In order to make the existing models work with the new *Surface* class, one would have to remove or rename the existing one. This has to be done as the parts of the tire communicating with the *Surface* class imply the definition found below.

```
protected
    outer WheelsAndTires.Environment.Surface Surface;
```

This covers the model path as well as the name so either all models implementing these lines of code have to be changed or the original surface has to be removed or renamed².

²Renaming will change the definitions in the models containing this statement, what has to be considered.

5.2.3 Implementation Details

This section presents a method of parameter passing that is very useful in the *Surface* class and is therefore presented quickly. Additionally two minor implementation details are mentioned afterwards.

In order to get functions in Dymola using parameters of a superposed model, it is possible to utilize protected functions. This can be done in the following way.

```
public
  function get_eN = get_eN_protected (
    Parameter1 = Parameter1,
    Parameter2 = Parameter2);
```

Where `get_eN_protected` is the function implementing the “real” body. The code section above is just useful to pass parameters to the function without the user caring about these. The protected function then receives these values as inputs as follows.

```
protected
  function get_eN_protected
    extends get_eN_Base; // adds x and z as inputs and eN[3] as output
    input Real Parameter1;
    input Real Parameter2;
  algorithm
    ...
  end get_eN_protected;
```

The result of this, at first sight slightly obscure construction, is that the user is able to call `get_eN(x,z)` with the function gathering the parameters from the *Surface* on its own. Otherwise, the call would have to include all the parameters from the *Surface* which does not make sense at all, because the *Geometry* class would have to know about the parameters of the *Surface* class.

Another important aspect describing the functionality of the *Surface* class can be seen when looking at the last code section. The base function is extended by the `get_eNprotected` function in order to add standardized in- and outputs.

To move the surface in the *x/z* plane, the two parameters `offsetX` and `offsetZ` have been introduced (see Figure 5.3 on Page 106) which basically define the position of the left upper corner of the grid shown in Figure 5.2(a) on Page 105.

The full code of the *Surface* class is shown in Appendix A.2.

6 Simulation Results

This section covers the models in the *Wheels and Tires* library that can be simulated directly. These are split up into two packages.

- The *Test Bench* covers models that are provided to verify basic functions of the tire. These include the *Elevation*, *Friction Effects*, *Surfaces* and *Plotting Functions*.
- The *Examples* package includes models that are provided to show the usage of the tires in combination with a vehicle.

6.1 The Test Bench

Models contained in the *Test Bench* are designed to verify fundamental properties of the tested tires. The packages contained are listed and described below.

- *Elevation* – Two models to test basic properties of the *Elevation* objects. The first is rather simple and the latter a little more complex.
- *Friction Effects* – Eight models to test the objects of the *Friction* class.
- *Surfaces* – Three models to gain some knowledge about the usage of the *Surface* class and its configuration as well as checking its functionality.
- *Plotting Functions* – One function to plot the behavior of different frictional classes. Possible results are shown in Figures 3.15 and 3.16 on Pages 40 and 41.

All of the models in the test benches for *Elevation* and *Friction Effects* are basically built up like the one shown in Figure 6.1. They consist of *Actuated Prismatic*s to stabilize the tire. *Actuated Revolute*s are used to let the tire spin or (not in the depicted model) have lean angle or e.g. turn around the y axis to exhibit bore torque. The other models built on the same basis, should therefore be self-explanatory and are thus not depicted in particular.

The models provided are suited to the tire models used. So when changing tire parameters or objects of the tire, one will have to reconfigure at least the parameters of the test bench models as well.

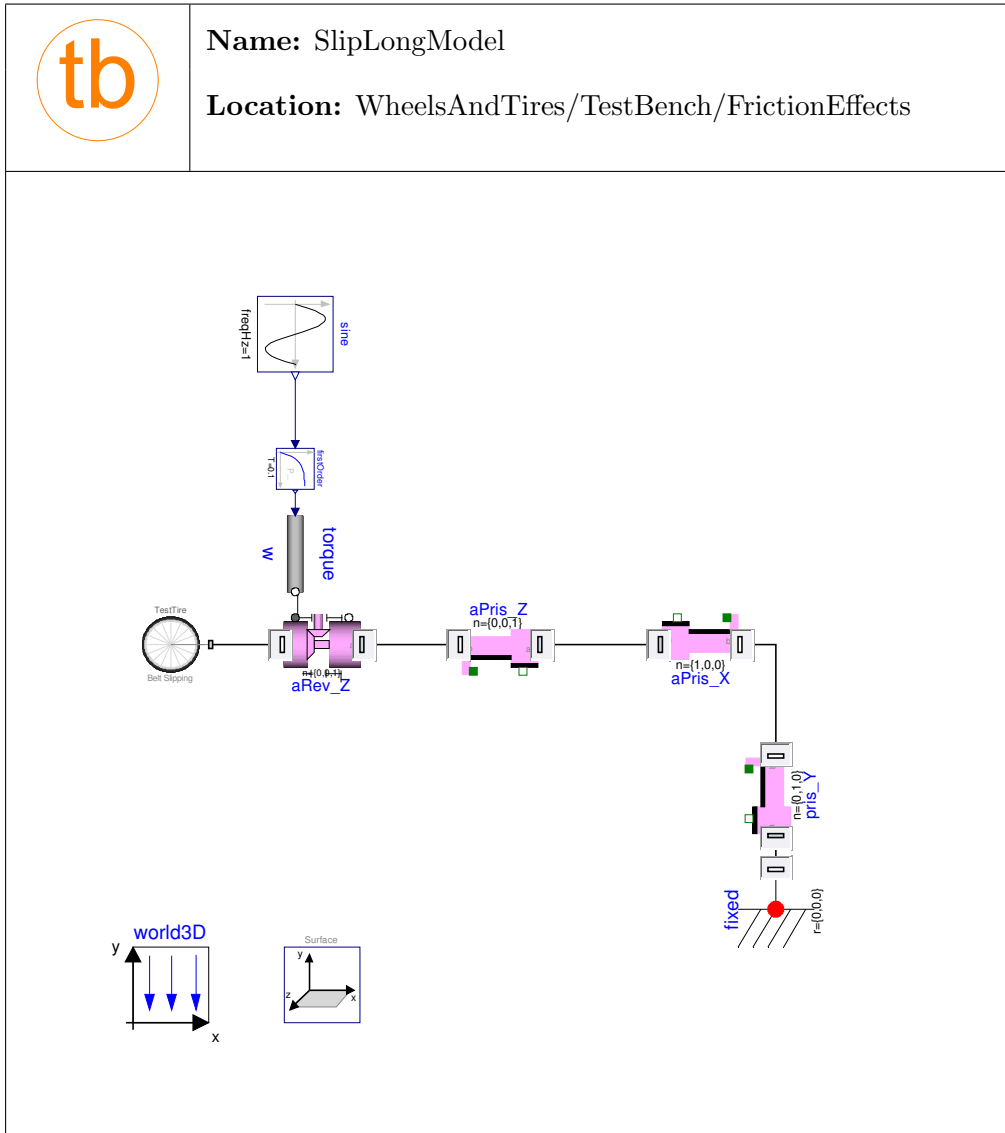


Figure 6.1: Model that applies a driving torque making the tire slip.

To give an impression of how the simulation results look like, Figures 6.2 to 6.5 provide

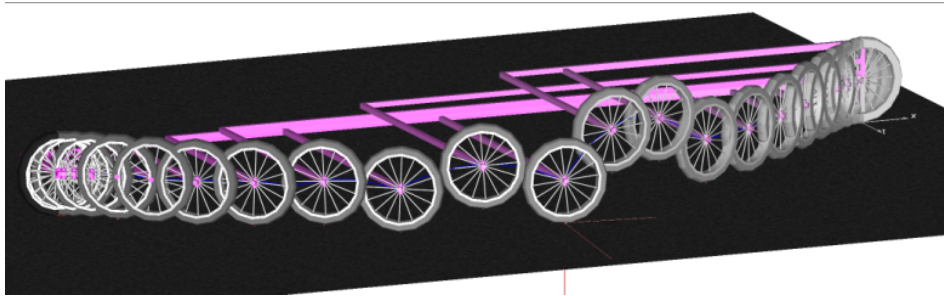


Figure 6.2: Tire elevating from the ground while driving a curve. Result of *Elevation Advanced*.

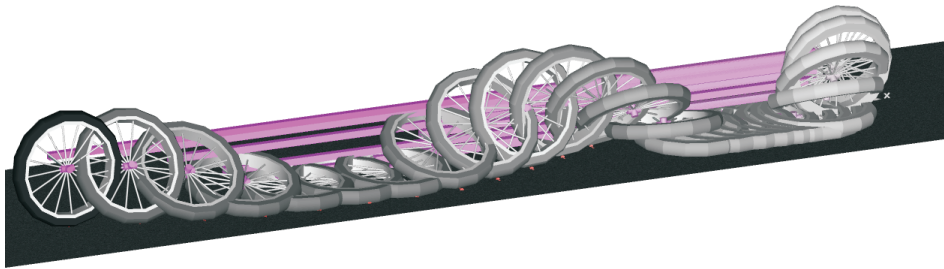


Figure 6.3: Tire's overturning torque *Test Bench* result.

some screen shots of simulation results from the *Test Bench* files. The “history frames” option in the “Animation Setup” was turned on to provide some idea of the behavior of the tires during the simulation. Unfortunately the transparency setting of the surface becomes deactivated when enabling the history frames. A transparent surface would make it possible to see the reaction forces of the tire.

The simulation of the *Test Bench* models covering frictional properties takes about $0.13ms/Interval$ to $0.25ms/Interval$. The fastest model is the *Self Aligning Torque* test that computes $10s$ with $2500Intervals$ in $0.33s$. The slowest is the *Advanced Elevation* test taking about $0.6s$ for $2500Intervals$ covering $10s$ of simulation time. The situation changes when simulating the *Surface* test models that take between 1.8 and $35.5ms/Interval$. So the slowest model takes about $4.5s$ for $10s$ of simulation time for three tires. This shows that the simulation times strongly depend on the application of the wheel models making general predictions difficult.

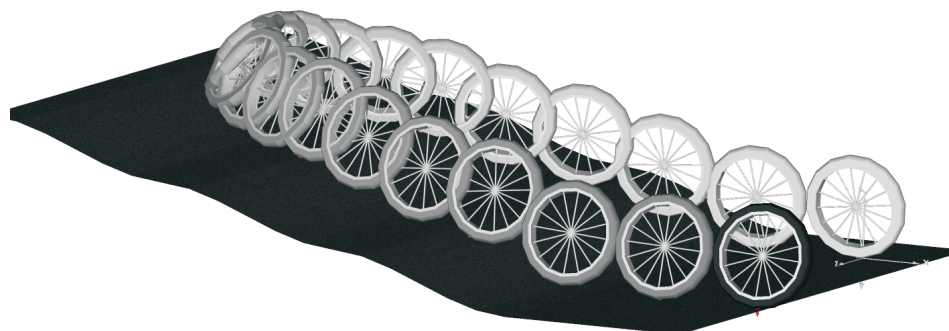


Figure 6.4: A tire rolling up an uneven surface with an initial lean angle (resulted from *Uneven Surface Rotating Tire*).

6.2 Provided Examples

Six examples are provided with the library. Five of the Examples are two-wheeled (single-track) vehicles, whereas three are unsprung bicycle models composed of rigid elements exclusively. The other two model sprung motorcycles including front and rear suspension. For both vehicles, models are provided to analyze the uncontrolled stability, by comparing two bicycles in one model directly, e.g. with different tire properties for geometry and friction as shown in Figures 6.6 and 6.7 for the bicycle. The other two models are “nice to show” models with the bicycle being accelerated by a strong torque to make the front wheel lift from the ground as shown in Figure 6.8 and the motorcycle jumping over a gap as depicted in Figure 6.9.

In contrast to the motorcycle the bicycle example has a proper visualization. For the motorcycle (and the four-wheeled vehicle), no further effort was spent regarding the visualization, as it is not the crucial aspect of the examples. Both two-wheeled models (found in the *Used Models* package) were provided by Thomas Schmitt and were part of the development of the *Motorcycle Library* [Sch09].

The sixth example is a very basic unsprung model of a four-wheeled vehicle with a predefined steering angle profile and driving torques acting on the rear tires. The result of the quite simple input signals is a rather curious behavior of the vehicle with the car starting to drift after 22s of the simulation, due to an impulse in the driving torque (shown in Figure 6.10). This is “reacted” on by a full breaking of all wheels marking the wheels slide to standstill of the vehicle.

Regarding the simulation speeds of the examples, it can be stated that the bicycle models take about half of the simulated time for the computation of the result. The

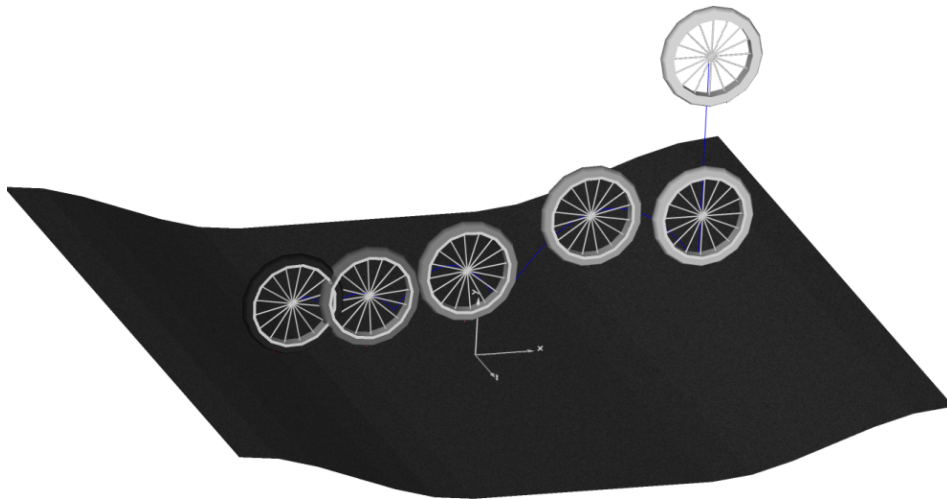


Figure 6.5: Tire dropping on an uneven surface (resulted from *Uneven Surface Dropping Tire*).



Figure 6.6: Two similar bicycles with non-slipping tires differing in their geometric properties. The upper bicycle is equipped with ideally slim tires whereas the lower one has a belted tire having a width of 2cm.

motorcycle jumping over the gap takes about 250s of calculation time for 16s of actual simulation. The reason for this is the more complex structure of the motorcycle than the one of the bicycle. The undamped four wheeled models compute the results in a little less time than the simulation time. All of the models used non-dynamic tires either with *slipping* or the *advanced* friction models. All simulations have been carried out on a Core2Duo clocked at 2GHz equipped with 4GB RAM, computing 250 output intervals per second. It has to be mentioned that no big effort was spent regarding the optimization of simulation speed of the overall models. A more suitable combination of state variables would most likely lead to a considerable enhancement in speed.



Figure 6.7: Two similar bicycles with slipping tires differing in their frictional properties. The one driving the inner circle is equipped with the *Advanced Friction* class, whereas the outer bicycle is equipped with the *Dry Friction with Constant Roll Resistance* class.



Figure 6.8: A bicycle with non-ideal tires accelerated by a torque on the rear tire, with the front tire lifting from the ground.

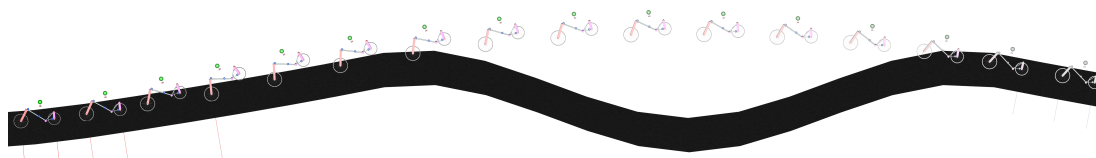


Figure 6.9: A motorcycle jumping over a gap.

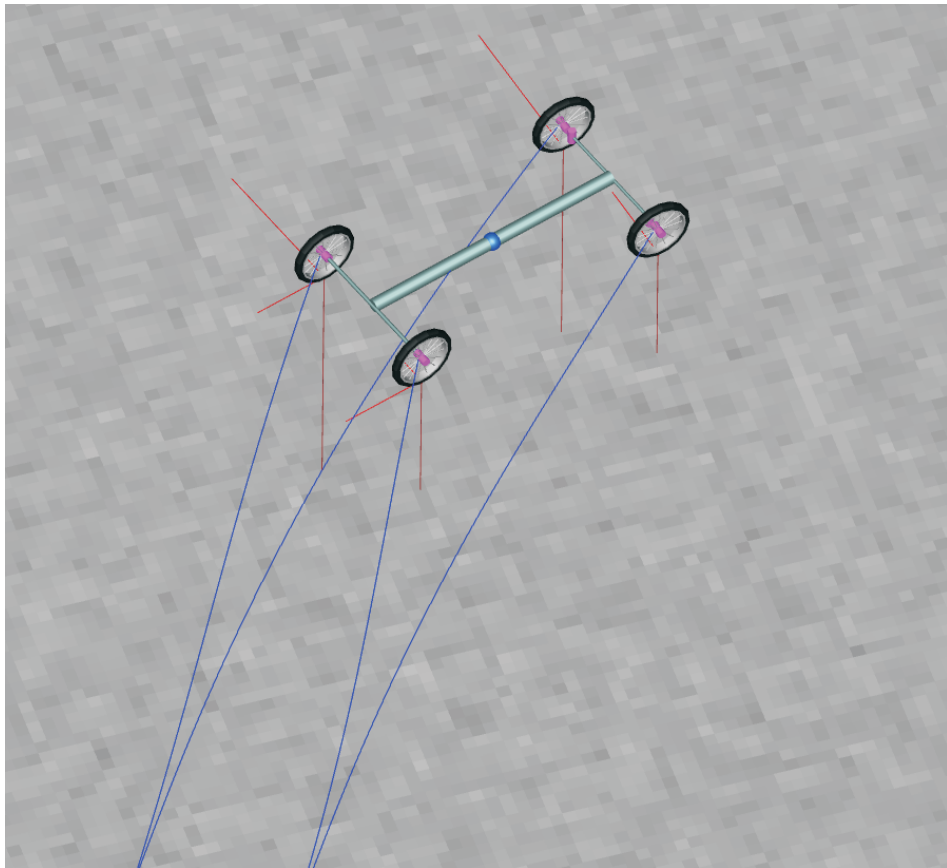


Figure 6.10: Result of the *Four Wheeled Vehicle* example after 22.5s (vehicle is drifting)

7 Conclusions

The goal of the Master's Thesis was to create a truly object-oriented tire library with a hierarchical composition. The necessary decomposition of properties results in a wheel model consisting of seven well-defined and truly self-sufficient classes (see Chapter 3). To ease the use of these classes a communication structure tailored to the special needs arising from the decomposition was introduced. Two of the classes have a further hierarchical decomposition, namely the *Friction* and the *Contact Physics* class. Whereas the latter one could be discussed to get split up into two separate classes, the *Friction* class is one of the key enhancements making it possible for the future modeler to focus on a special part of frictional properties exclusively.

Tire models can be built from scratch which is absolutely manageable as well as the even simpler and therefore recommended possibility to adapt ready-made tires presented in Chapter 4. The resulting models can be combined with simple (even) or more complex surfaces (see Chapter 5) without changing the tire model. Moreover, a number of test bench models are provided, enabling a test of single tire properties. Finally, models of two-wheeled single-track and four-wheeled vehicles are provided, showing the capabilities of the library (see Chapter 6).

Still, the lack of comparison to verified simulation results and measurement data is considered to be a drawback as some malfunctions could probably not be identified. Moreover the application of the library is limited due to the fact that the 3D mechanics connectors of the *Multi Bond Library* (which the library was built upon) and the *Multi Body Library* (that is part of the Modelica Standard library) are incompatible. This restricts the use of the tires to models built with elements from the *Multi Bond Library*.

To sum up, the library enables a quick and convenient building of easily customizable tire models. There are some further enhancements possible, but the structure of the tire models should persist as it can easily be extended, fulfilling the major requirements that were demanded at the beginning of the work.

8 Further Work

The following section accumulates some thoughts that arose during the creation of the library. Every paragraph intends to present one improvement to the library which is considered to be reasonable. It is split into the sections covering functional enhancements, better usability and one section regarding computational effort.

Usability

A very important factor to enhance the library's usability would be to add more suitable default parameters. Especially for inexperienced users this would be a great advantage. Still, the finding of suiting parameters is a very time consuming task especially for persons not very familiar with the topic. Apart from that a number of real-world tires could be implemented as a reference if parameter data is freely available.

A further enhancement regarding usability could be achieved by using parameters, which the average user is more familiar with. So e.g. it would be possible to provide a (non-linear) relationship between the inflation pressure of the tire and the vertical stiffness for the *Vertical Dynamics* model. This would enable the user to enter pressure values which nearly everybody is familiar with, rather than having to think about a stiffness and damping constant.

One task to improve the exchangeability of the geometric classes would be to unify the parameters of these models. This was not done in the library in order to have parameters that the modeler is more familiar with, especially for the *Belted Tire*. Still, one could discuss whether it makes more sense to use the ISO standard tire codes (P215/65R15...) as parameters or to make it more exchangeable with different geometries.

For more complex tire models, the number of parameters rises quickly. This can lead to a cumbersome adaptation of parameters when switching from one tire to a different one. Records of parameters could enhance this aspect as well as creating a separate class for every single tire. Both versions have advantages over the other and if the library is extended with more ready-made tires, one should think about which version is the better one.

Using *Surfaces* in models with working directories other than the directory the *Wheels and Tires* library is saved in will cause the textures (PNGs) of the surfaces to disappear. This is caused by the fact that the *Surface Material* class searches for the image files in the `./images` subdirectory of the working directory. Therefore, the modeler has to copy the PNG files to the modeling directory or use the gray surfaces that are created without a PNG. No possibility to overcome this problem was investigated during the thesis, but still it would be an advantage.

Initializing the model together with a mass (not the simple mass from the *Multi Bond Library*s) causes inconvenient behavior during the initialization. This could probably be overcome by a correct usage of the `defineRoot` statement. If that is not possible, it could be considered to set angles and translational positions separately to overcome this issue at least partly.

Function

Probably the most important part that is not realized in the current version of the library are the tire models by Pacejka [Pac06]. This would be an extension to the *friction class* that could be done quite quickly, greatly enhancing the applicability of the library. Moreover this would add the possibility to model non-symmetric tread profile, which results in an offset to the frictional curve shown in Section 3.5.2.

In some simulations, not the whole 3D tire model is necessary. Sometimes a simplified version in 2D or even 1D would be enough e.g. when simulating the behavior of a single suspension with a road profile as an input. This would decrease computational effort drastically whilst barely reducing the result's quality. Still, this would most likely cause some major changes in the library and therefore has to be thought of carefully before starting the redesign or extension.

Many relations that influence the tire's behavior are not considered in the library. So e.g. there is an influence of the driving speed to the damping of the *Vertical Dynamics* model and the friction coefficient. There are quite a lot of these effects that are not modeled arising from the complex structure of a tire making it difficult for a person not very familiar with the topic to decide which effects are more important than others. Therefore, either an experienced person or users should demand certain improvements.

A very beneficial part to improve would be the *Surface* class. If it was made derivable, that would enable modifications of the *Vertical Dynamics* class replacing the non-ideal derivative block with an ideal one.

In order to have a more correct result in combination with the actual *Surface*, it would be possible to use conditional declarations to either use ideal or non-ideal derivation blocks in the elevation and "long-lat" dynamic models. This could be made directly

dependent on the `flatSurface` option. Apart from that this would make it possible to use all models on all possible surfaces without the user having to adapt models.

Another useful enhancement for the surface class would be to create multiple surfaces instead of a single one. This would enable the modeler to create more appealing visualizations and simply show e.g. different frictional properties indicated by different textures. Moreover this would enable a reduced stretching of the used image files (PNGs), which would be another optical enhancement.

A kind of environment-builder concept featuring different parts e.g. standardized road elements, or even some aesthetically pleasing elements like houses or trees with a proper visualization would greatly enhance the libraries optics. To make more complex parts like road elements possible, a different concept of friction-coefficient determination would have to be introduced.

An interesting aspect would be to include the temperature rise due to losses in the tires. Still this a quite complex enhancement because models would be needed that compute the overall losses, and the cooling by the surrounding air would have to be included as well. Still the change of temperature of the air inside the tire has a noticeable influence on the pressure (with respect to rough calculations). Based on measurements one could as well include the influence of the warming of the tire's rubber composition. It would be possible to separate the vertical dynamics into models for inflation pressure and the rubber. With the temperature included in the model the generation of longitudinal and lateral forces could be made dependent on the temperature as well.

For simulations that include freely moving tires, problems occur during the initialization. The vector `dLong` is reportedly set to 0 in the dimensions 1 and 3, as these values are not initialized. This can cause a division by zero in the equation normalizing `dLong` to `eLong` because all elements of `dLong` can become zero when setting dimensions 1 and 3 to zero. This is ignored, as the solver is able to handle this problem and it therefore has no influence on the result. Attempts to account for the displayed warning by setting `dLong[1]` to a reasonable value during initialization cause Dymola to get an over-specified initial problem.

One possibility to enhance the usage of more complex surfaces is presented in Chapter 10 of [Pac06]. Rather than the smoothly interpolated surfaces implemented in the *Wheels and Tires* library, this would enable the models to respond on more unsteady surfaces correctly. But as mentioned in [BA03], the implementation of this “filter” is quite complex.

There are different definitions of slip found in literature. An enhancement would be to implement these different definitions that could then be used by the different *Slip Properties* classes. This would enable an exact comparability to different available simulation results.

Speed

One quite important aspect would be to review the model structure with regards to simulation efficiency. Some effort was spent selecting appropriate state variables with the *enforce states* parameter turned on, but there are plenty of possibilities to further enhance the models' efficiency.

Investigations which part of the model adds immoderate computational effort would be very interesting. These analyses are possible with Dymola and would most likely be highly revealing. Other possibilities would be to do an in-depth presentation of the library to experienced Modelica users who would probably be able to directly identify problems regarding computational efficiency. E.g. a test regarding the filtering times of the non-ideal derivative blocks, in order to find the right balance between accuracy and speed would be very interesting.

Bibliography

- [AJ02] J. Andreasson and J. Jarlmark. Modularised tyre modelling in modelica. In *Proceedings of the Second International Modelica Conference, Oberpfaffenhofen, Germany*, pages 267–274, 2002.
- [And03] Johan Andreasson. Vehicledynamics library. In *Proceedings of the Third International Modelica Conference, Linköping, Sweden*, pages 11–18, 2003.
- [AS93] G. Aumann and K. Spitzmüller. *Computerorientierte Geometrie*. BI-Wiss.-Verl, 1993.
- [Ass07] Modelica Association. Modelica 3.0 language specification. <http://www.modelica.org/documents/ModelicaSpec30.pdf>, 2007.
- [Ass09] Modelica Association. Modelica standard library v3.0. <http://www.modelica.org/libraries/Modelica>, 2009.
- [BA03] Mats Beckmann and Johan Andreasson. Wheel model library for use in vehicle dynamic studies. In *Proceedings of the third Modelica Conference, Linköping, Sweden*, pages 385–392, 2003.
- [BSGR08] P. Bohara, A. Saha, P. Ghosh, and M. Roopak. Tyre performance prediction through fea. Hasetri - Hari Shankar Singhania Elasto-mer and Tyre Research Institute, 2008.
- [Cel91] Francois E. Cellier. *Continuous System Modeling*. Springer, 1991.
- [CN05] F. E. Cellier and Å. Netbot. The modelica bond-graph library. In *Proceedings of the 4th International Modelica Conference, Hamburg*, pages 57–65, 2005.
- [Cra89] John J. Craig. *Introduction to Robotics Mechanics and Control*. Addison Wesley Longman, 1989. Second Edition.
- [Dix96] John C. Dixon. *Tires, Suspension and Handling*. Cambridge University Press, 1996. Second Edition.
- [Fri04] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley & Sons, 2004.

- [Lei09] Günter Leister. *Fahrzeugreifen und Fahrwerkentwicklung*. Vieweg + Teubner, 2009. 1. Auflage.
- [McB05] Robert Thomas McBride. *System Analysis through Bond Graph Modeling*. PhD thesis, University of Arizona, 2005.
- [MW03] Manfred Mitschke and Henning Wallentowitz. *Dynamik der Kraftfahrzeuge*. Springer Verlag, VDI, 2003. 4. Auflage.
- [Pac06] Hans B. Pacejka. *Tyre and Vehicle Dynamics*. Butterworth-Heinemann, 2006. Second Edition.
- [Ril07] Georg Rill. *Simulation von Kraftfahrzeugen*. Vieweg-Verlag, 2007. genehmigter Nachdruck.
- [Sch09] Thomas Schmitt. Modeling of a motorcycle in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.
- [Zim06] Dirk Zimmer. A modelica library for multibond graphs and its application in 3d-mechanics. Master's thesis, ETH Zürich, 2006.
- [ZO09] Dirk Zimmer^a and Martin Otter^b. Real-time models for wheels and tires in an object-oriented modeling framework. Accepted for publication in Vehicle Dynamics, 2009.

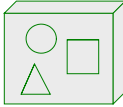

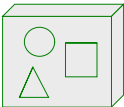


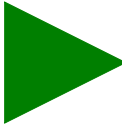
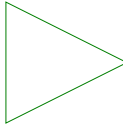

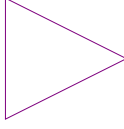

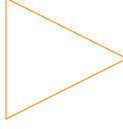
Sworn declaration

I hereby declare under oath that this master's thesis is the product of my own independent work. All content and ideas drawn directly or indirectly from external sources are indicated as such. The thesis has not been submitted to any other examining body and has not been published.


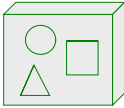
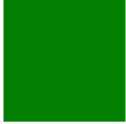
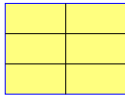

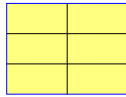

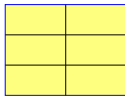

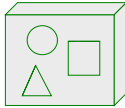
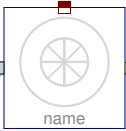
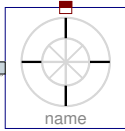
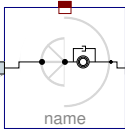
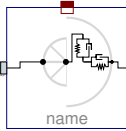
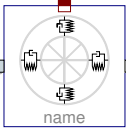

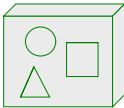
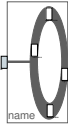
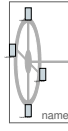
Markus Andres, Dornbirn, August 27, 2009

A Appendix

A.1 Library Structure

Top Level Package	Second L. Package	Third L. Package	Contained Models
 Tire Components			
	 Communication		    Tire Bus Contact Point Connector Unit Vector Input Unit Vector Output     Contact Properties Input Contact Properties Output Sensor Values Input Sensor Values Output


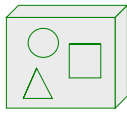
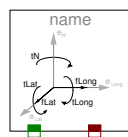
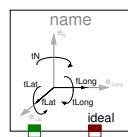
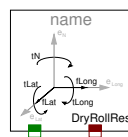
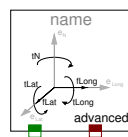

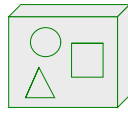


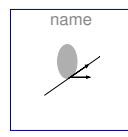
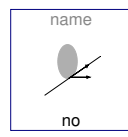

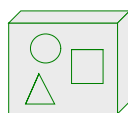
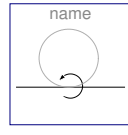
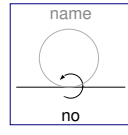
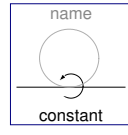
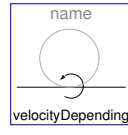

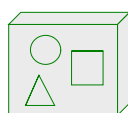
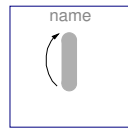
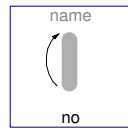
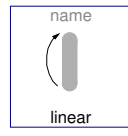
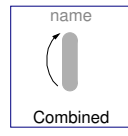
continued on next page

Top Level	Second L.	Third L.	Contained Models
		 <p>Signals</p>	 <p>Unit Vector Signal</p>  <p>Unit Vectors</p>  <p>Contact Properties Signal</p>  <p>Contact Properties</p>  <p>Sensor Values</p>  <p>Sensor Values</p> <p>Signal</p>
	 <p>Belt Dynamics</p>		 <p>Belt Dynamics Base</p>  <p>Rigid Belt</p>  <p>Torsion Belt</p>  <p>Long Lat Belt</p>  <p>Quad Order 2 Belt</p>
		 <p>Used Models</p>	 <p>Quad Belt</p>  <p>Quad Rim</p>

continued on next page

Top Level	Second L.	Third L.	Contained Models
	 Contact Physics		 name name name Contact Physics Base Contact Physics No Dynamics Contact Physics With Dynamics
		 Contact Point Dynamics	 name name name Contact Point Dynamics Base Contact Point Dynamics No Dynamics Contact Point Dynamics Second Order
		 Center To Contact Point	 name name Center To Contact Point Base Center To Contact Point Bond
	 Vertical Dynamics		 name name name name name Elevation Base No Elevation Kelvin Elevation Gehmann Elevation Elasto Elevation


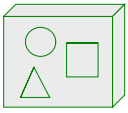
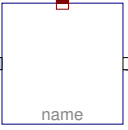
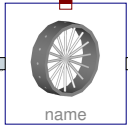




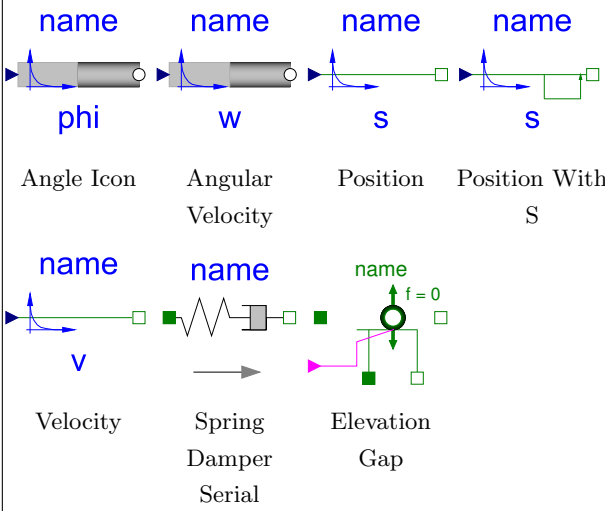
continued on next page

Top Level	Second L.	Third L.	Contained Models			
	  Friction		 Friction Base	 Ideal Friction	 Dry Friction With Roll Resistance	 Advanced Friction
	  Slip Properties		 Get mu.vSlip	 SoftMaxTol	 Slip Base	 No Slip
	  Rolling Resistance		 Rolling Resistance Base	 No Rolling Resistance	 Constant Rolling Resistance	 Speed Depending Rolling Resistance
	  Bore Torque		 Bore Torque Base Icon	 No Bore Torque	 Linear Bore Torque	 Combined Bore Torque



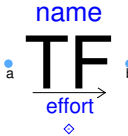
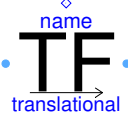
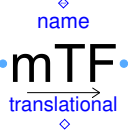
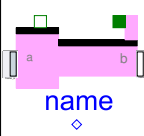

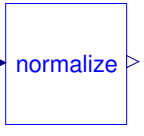

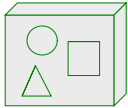
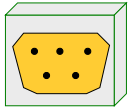
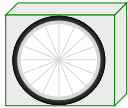




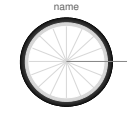


continued on next page

Top Level	Second L.	Third L.	Contained Models			
		 Camber Force	 Camber Force Base	 No Camber Force	 Simple Camber Speed	 Combined Camber Speed
		 Load Influence	 Friction Load Influence Base	 Linear Load Influence	 Quadratic Load Influence	
		 Overturing Torque	 Overturing Torque Base	 No Overturing Torque	 Linear Overturing Torque	
		 Self Aligning Torque	 Self Aligning Torque Base	 No Self Aligning Torque	 Self Aligning Torque Rill	
	 Geometry		 Geometry Base	 Flat Disc	 Circular Tire	 Belt Tire

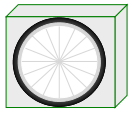

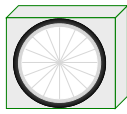

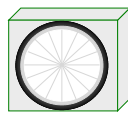


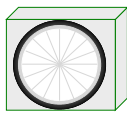





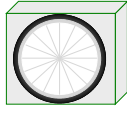


continued on next page

Top Level	Second L.	Third L.	Contained Models
		 Rim	 Rim Base  Spoked Rim
	 Utilities		
		 Additional Bond Graph	 <p> name name name name phi w s s Angle Icon Angular Velocity Position Position With S name name name v Spring Damper Serial Elevation Gap f = 0 </p>


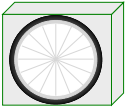







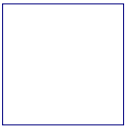




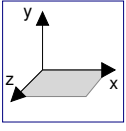




continued on next page

Top Level	Second L.	Third L.	Contained Models
	↳	 Additional Multi Bond Graph	 Boolean Signal  modulated Transformer 2 Effort  Modulated Translational Transformer Radius  Modulated Translational Amplification  Modulated Actuated Prismatic  Absolute Sensor  Normalize
	↳	 Icons	 Tire Components Package  Communication Package  Tire Package  Environment Package  Test Bench Package  Example Package  Tire Components  Tire Model  Test Bench Model  Example Model






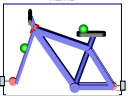
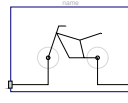









continued on next page

Top Level	Second L.	Third L.	Contained Models
 Tires			
	 Commons		Parameters Common Parameters Dry Friction with Rolling Resistance Parameters Advanced Friction
		 Types	 Initialization of Slipping Tires
	 Slim		    Ideal Slipping Advanced Dynamic
	 Circular		  Advanced Dynamic

continued on next page

Top Level	Second L.	Third L.	Contained Models
	 Belt		<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <small>name</small>  Ideal </div> <div style="text-align: center;"> <small>name</small>  Slipping </div> <div style="text-align: center;"> <small>name</small>  Advanced </div> <div style="text-align: center;"> <small>name</small>  Longitudinal Lateral Dynamics </div> </div> <div style="text-align: center; margin-top: 20px;"> <small>name</small>  Quad Dynamics </div>
 Environment			
	 Surface Base		<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  get_eN_Base </div> <div style="text-align: center;">  get_elevation_Base </div> <div style="text-align: center;">  get_mu_Base </div> </div>
	<small>name</small>  Surface		<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  get_eN </div> <div style="text-align: center;">  get_elevation </div> <div style="text-align: center;">  get_mu </div> <div style="text-align: center;">  get_rectangle </div> </div>

continued on next page

Top Level	Second L.	Third L.	Contained Models
 Examples			 Uncontrolled Bicycle Uncontrolled Bicycle Torque Uncontrolled Motorcycle Uncontrolled Motorcycle Torque  Four Wheeled Vehicle
	 Used Models		  Bicycle Motorcycle
 Visualization			   name Torus Surface Visualization Arrow
	 Base Classes		   Surface Material Visual Base Visualizer Base

A.2 Complete Code for the Surface Class

A.2.1 WheelsAndTires.Environment.SurfaceBase

```
partial model SurfaceBase
  import SI = Modelica.SIunits;

  partial function get_eN_Base
    input SI.Position x;
    input SI.Position z;
    output Real eN[3];
  end get_eN_Base;

  partial function get_elevation_Base
    input SI.Position x;
    input SI.Position z;
    output SI.Position elevation;
  end get_elevation_Base;

  partial function get_mu_Base
    input SI.Position x;
    input SI.Position z;
    output Real mu;
  end get_mu_Base;

end SurfaceBase;
```

A.2.2 WheelsAndTires.Environment.Surface

```
model Surface

  extends WheelsAndTires.Environment.SurfaceBase;
  import SI = Modelica.SIunits;
  import MB = Modelica.Mechanics.MultiBody;
  import WheelsAndTires.Visualization.BaseClasses.SurfaceMaterial;

  parameter Boolean FlatSurface = true
    "If true simpler equations for the functions can be used,
     false enables uneven surfaces";
  parameter Boolean visSurface = true
    "= true if the surfrace shall be shown in the animation window";
```

```
parameter Integer nu(final min=2) = 4
  "Number of grid points in x direction";
parameter Integer nv(final min=2) = 4
  "Number of grid points in z direction";
parameter Integer IP(final min=0) = 5
  "Number of interpolation points between the grid points
  (just for visualization)";

parameter Real PNG = 1
  "Filename of the that shall be used as texture";
parameter MB.Types.Color Color= {255,255,255}
  "Surface color (mixed with texture)";
parameter MB.Types.SpecularCoefficient SpecularCoefficient = 0.1
  "Specular coefficient of the road surface without texture";

parameter SI.Length LengthX = 50
  "Length of the surface area in x direction";
parameter SI.Length LengthZ = 50
  "Length of the surface area in z direction";
parameter SI.Length OffsetX = -LengthX/2 "Offset in x direction";
parameter SI.Length OffsetZ = -LengthZ/2 "Offset in z direction";

parameter SI.Distance y[nu,nv] = zeros(nu,nv)
  "Matrix with the y (elevation) values at the grid points"
  annotation(Dialog(enable= not FlatSurface));
parameter Real dy_dx[nu,nv] = zeros(nu,nv)
  "Matrix with the slope of the grid points in x direction"
  annotation(Dialog(enable= not FlatSurface));
parameter Real dy_dz[nu,nv] = zeros(nu,nv)
  "Matrix with the slope of the grid points in y direction"
  annotation(Dialog(enable= not FlatSurface));
parameter Real mu[nv-1,nv-1] = ones(nv-1,nv-1)
  "Matrix with the friction coefficient for every rectangle";

final parameter Integer nuIP = nu+(nu-1)*IP
  "Number of the grid values in x direction with interpolation";
final parameter Integer nvIP = nv+(nv-1)*IP
  "Number of the grid values in y direction with interpolation";

public
function get_eN = get_eN_protected (
  FlatSurface = FlatSurface,
  yMatrix=y,
```

```

dy_dxMatrix=dy_dx,
dy_dzMatrix=dy_dz,
nu=nu,
nv=nv,
LengthX=LengthX,
LengthZ=LengthZ,
OffsetX=OffsetX,
OffsetZ=OffsetZ);
protected
function get_eN_protected
extends get_eN_Base;
// adds x and z as inputs and eN[3] as output
input Boolean FlatSurface;
input SI.Distance yMatrix[:,:];
input Real dy_dxMatrix[:,:];
input Real dy_dzMatrix[:,:];
input Integer nu;
input Integer nv;
input SI.Length LengthX;
input SI.Length LengthZ;
input SI.Distance OffsetX;
input SI.Distance OffsetZ;
protected
Integer xIndex;
Integer zIndex;
SI.Distance localX;
SI.Distance localZ;
Boolean inArea;
SI.Distance rectangleY[2,2];
Real rectangleDy_dx[2,2];
Real rectangleDy_dz[2,2];
Real X[4];
Real Z[4];
Real G[4,4];
constant Real H[4,4] = [2,-3, 0, 1;-2, 3, 0, 0;
  1,-2, 1, 0; 1,-1, 0, 0];
constant Real dH[4,4] = [0, 6,-6, 0; 0,-6, 6, 0;
  0, 3,-4, 1; 0, 3,-2, 0];
Real dy_dx;
Real dy_dz;
SI.Distance factorX;
SI.Distance factorZ;
SI.Length localXNorm;

```

```

    SI.Length localZNorm;
    SI.Length rectangleX;
    SI.Length rectangleZ;
    Real dN[3];
algorithm

(xIndex, zIndex, inArea, localX, localZ) :=
get_rectangle(x=x,z=z,nu=nu,nv=nv,LengthX=LengthX,
    LengthZ=LengthZ,OffsetX=OffsetX,OffsetZ=OffsetZ);

if inArea and not FlatSurface then

    rectangleX :=LengthX/(nu-1);
    rectangleZ :=LengthZ/(nv-1);

    rectangleY :=yMatrix[xIndex:xIndex + 1, zIndex:zIndex + 1];
    rectangleDy_dx :=dy_dxMatrix[xIndex:xIndex + 1,
        zIndex:zIndex + 1];
    rectangleDy_dz :=dy_dzMatrix[xIndex:xIndex + 1,
        zIndex:zIndex + 1];
    G :=[rectangleY, rectangleDy_dz*rectangleZ;
        rectangleDy_dx*rectangleX, zeros(2, 2)];

    localXNorm :=localX/rectangleX;
    localZNorm :=localZ/rectangleZ;

    X :={localXNorm^3,localXNorm^2,localXNorm,1};
    Z :={localZNorm^3,localZNorm^2,localZNorm,1};

    dy_dx := (dH*X)*G*(H*Z)/rectangleX;
    dy_dz := (H*X)*G*(dH*Z)/rectangleZ;

    dN :=cross({0,dy_dz,1}, {1,dy_dx,0});
    eN :=dN/sqrt(dN*dN);
else
    eN :={0,1,0};
end if;

end get_eN_protected;

public
function get_elevation = get_elevation_protected (
    FlatSurface = FlatSurface,

```

```
yMatrix=y,  
dy_dxMatrix=dy_dx,  
dy_dzMatrix=dy_dz,  
nu=nu,  
nv=nv,  
LengthX=LengthX,  
LengthZ=LengthZ,  
OffsetX=OffsetX,  
OffsetZ=OffsetZ);
```

```
protected
```

```
function get_elevation_protected  
  extends get_elevation_Base;  
  // adds x and z as inputs and elevation as output  
  input Boolean FlatSurface;  
  input SI.Distance yMatrix[:,:];  
  input Real dy_dxMatrix[:,:];  
  input Real dy_dzMatrix[:,:];  
  input Integer nu;  
  input Integer nv;  
  input SI.Length LengthX;  
  input SI.Length LengthZ;  
  input SI.Distance OffsetX;  
  input SI.Distance OffsetZ;
```

```
protected
```

```
  Integer xIndex;  
  Integer zIndex;  
  SI.Distance localX;  
  SI.Distance localZ;  
  Boolean inArea;  
  SI.Distance rectangleY[2,2];  
  Real rectangleDy_dx[2,2];  
  Real rectangleDy_dz[2,2];  
  Real X[4];  
  Real Z[4];  
  Real G[4,4];  
  SI.Distance rectangleX;  
  SI.Distance rectangleZ;  
  Real localXNorm;  
  Real localZNorm;  
  constant Real H[4,4] =  
    [2,-3, 0, 1; -2, 3, 0, 0; 1,-2, 1, 0; 1,-1, 0, 0]  
    "Matrix used for the interpolation";
```

```

algorithm
  (xIndex, zIndex, inArea, localX, localZ) :=
    get_rectangle(x=x,z=z,nu=nu,nv=nv,LengthX=LengthX,
      LengthZ=LengthZ,OffsetX=OffsetX,OffsetZ=OffsetZ);

  if inArea and not FlatSurface then
    rectangleX := LengthX/(nu-1);
    rectangleZ := LengthZ/(nv-1);

    rectangleY :=yMatrix[xIndex:xIndex + 1, zIndex:zIndex + 1];
    rectangleDy_dx :=dy_dxMatrix[xIndex:xIndex + 1,
      zIndex:zIndex + 1];
    rectangleDy_dz :=dy_dzMatrix[xIndex:xIndex + 1,
      zIndex:zIndex + 1];
    G :=[rectangleY, rectangleDy_dz*rectangleZ;
      rectangleDy_dx*rectangleX, zeros(2, 2)];

    localXNorm :=localX/rectangleX;
    localZNorm :=localZ/rectangleZ;

    X :={localXNorm^3,localXNorm^2,localXNorm,1};
    Z :={localZNorm^3,localZNorm^2,localZNorm,1};

    elevation := (H*X)*G*(H*Z);
  else
    elevation := 0;
  end if;

end get_elevation_protected;

public
function get_mu = get_mu_protected (
  muMatrix=mu,
  nu=nu,
  nv=nv,
  LengthX=LengthX,
  LengthZ=LengthZ,
  OffsetX=OffsetX,
  OffsetZ=OffsetZ);

protected
function get_mu_protected

```



```
    extends get_mu_Base; // adds x and z as inputs and mu as output
    input Real muMatrix[:,:]; //additional input(s)
    input Integer nu;
    input Integer nv;
    input SI.Length LengthX;
    input SI.Length LengthZ;
    input SI.Distance OffsetX;
    input SI.Distance OffsetZ;
protected
    Integer xIndex;
    Integer zIndex;
    Boolean inArea;
algorithm
    (xIndex,zIndex,inArea) :=
    get_rectangle(x=x,z=z,nu=nu,nv=nv,LengthX=LengthX,LengthZ=LengthZ,
    OffsetX=OffsetX,OffsetZ=OffsetZ);
    if inArea then
        mu := muMatrix[xIndex,zIndex];
    else
        mu := 1;
    end if;
end get_mu_protected;

public
function get_rectangle
    input SI.Position x;
    input SI.Position z;
    input Integer nu;
    input Integer nv;
    input SI.Length LengthX;
    input SI.Length LengthZ;
    input SI.Distance OffsetX;
    input SI.Distance OffsetZ;

    output Integer rectX;
    output Integer rectZ;
    output Boolean inArea;
    output SI.Distance localX;
    output SI.Distance localZ;

protected
    SI.Distance rectangleX;
    SI.Distance rectangleZ;
```

```
algorithm
  rectX := integer(ceil(((x-OffsetX)/LengthX)*(nu-1)));
  rectZ := integer(ceil(((z-OffsetZ)/LengthZ)*(nv-1)));

  if rectX > nu-1 or rectX < 1 or rectZ > nv-1 or rectZ < 1 then
    inArea :=false;
  else
    inArea :=true;
  end if;

  rectangleX := LengthX/(nu-1);
  rectangleZ := LengthZ/(nv-1);

  // getting local position
  (in the rectangle the tire is in at the time of calling)
  localX :=mod(x - LengthX - OffsetX, rectangleX);
  localZ :=mod(z - LengthZ - OffsetZ, rectangleZ);

end get_rectangle;

protected
WheelsAndTires.Visualization.SurfaceVisualization
  surfaceVisualizationNotFlat(
    nu=nu,
    nv=nv,
    IP=IP,
    PNG=PNG,
    Color=Color,
    SpecularCoefficient=SpecularCoefficient,
    LengthX=LengthX,
    LengthZ=LengthZ,
    OffsetX=OffsetX,
    OffsetZ=OffsetZ,
    y=y,
    dy_dx=dy_dx,
    dy_dz=dy_dz) if visSurface and not FlatSurface;

WheelsAndTires.Visualization.SurfaceVisualization
  surfaceVisualizationFlat(
    nu=nu,
    nv=nv,
    IP=IP,
```

```
PNG=PNG,  
Color=Color,  
SpecularCoefficient=SpecularCoefficient,  
LengthX=LengthX,  
LengthZ=LengthZ,  
OffsetX=OffsetX,  
OffsetZ=OffsetZ,  
y=zeros(nu,nv),  
dy_dx=zeros(nu,nv),  
dy_dz=zeros(nu,nv)) if visSurface and FlatSurface;  
end Surface;
```

A.2.3 WheelsAndTires.Visualization.SurfaceVisualization

```
model SurfaceVisualization  
  
import SI = Modelica.SIunits;  
import MB = Modelica.Mechanics.MultiBody;  
import WheelsAndTires.Visualization.BaseClasses.SurfaceMaterial;  
  
parameter Integer nu(final min=2) = 4  
  "Number of grid points in x direction";  
parameter Integer nv(final min=2) = 4  
  "Number of grid points in y direction";  
parameter Integer IP(final min=0) = 5  
  "Number of interpolation points between the grid points  
  (just for visualization)";  
  
parameter Real PNG = 1  
  "Filename of the that shall be used as texture";  
parameter MB.Types.Color Color= {255,255,255}  
  "Surface color (mixed with texture)";  
parameter MB.Types.SpecularCoefficient SpecularCoefficient = 0.1  
  "Specular coefficient of the road surface without texture";  
  
parameter SI.Length LengthX = 10  
  "Length of the surface area in x direction";  
parameter SI.Length LengthZ = 10  
  "Length of the surface area in z direction";  
parameter SI.Length OffsetX = -LengthX/2 "Offset in x direction";  
parameter SI.Length OffsetZ = -LengthZ/2 "Offset in z direction";
```

```

parameter SI.Distance y[nu,nv] = zeros(nu,nv)
  "Matrix with the y (elevation) values at the grid points";
parameter Real dy_dx[nu,nv] = zeros(nu,nv)
  "Matrix with the slope of the grid points in x direction";
parameter Real dy_dz[nu,nv] = zeros(nu,nv)
  "Matrix with the slope of the grid points in y direction";

final parameter Integer nuIP = nu+(nu-1)*IP
  "Number of the grid values in x direction with interpolation";
final parameter Integer nvIP = nv+(nv-1)*IP
  "Number of the grid values in y direction with interpolation";

WheelsAndTires.Visualization.BaseClasses.SurfaceMaterial SM(
  nu = nuIP,
  nv = nvIP,
  Extra = 10*(PNG+10*100),
  Material=vector([Color/255;SpecularCoefficient]));

protected
  outer MultiBondLib.Mechanics3D.World3D world3D;
  SI.Position x "actual x value for the interpolation";
  SI.Position z "actual z value for the interpolation";
  SI.Position X[4] "vector with powers of x (for the interpolation)";
  SI.Position Z[4] "vector with powers of Z (for the interpolation)";
  constant Real H[4,4] = [ 2,-3, 0, 1;
    -2, 3, 0, 0;
    1,-2, 1, 0;
    1,-1, 0, 0]
    "Matrix used for the interpolation";
  Real G[4,4] "Matrix used for the interpolation";
  SI.Length rectangleX "Scaling factor for dy_dx";
  SI.Length rectangleZ "Scaling factor for dy_dz";

algorithm
when {initial() and world3D.enableAnimation} then

  rectangleX := LengthX/(nu-1);
  rectangleZ := LengthZ/(nv-1);

  // loop for the single rectangles
  (Interpolation just works within one rectangle)
  for m in 1:nu-1 loop
    for n in 1:nv-1 loop

```

```

// creating the grid in x and z direction
for i in 1:nuIP loop
  for j in 1:nvIP loop
    SM.x[i,j] := LengthX*(i - 1)/(nuIP - 1) + OffsetX;
    SM.z[i,j] := LengthZ*(j - 1)/(nvIP - 1) + OffsetZ;
  end for;
end for;

// definition of the matrix G for the interpolation
G :=[y[m:m+1, n:n+1], dy_dz[m:m+1, n:n+1]*rectangleZ;
     dy_dx[m:m+1, n:n+1]*rectangleX, zeros(2, 2)];

// interpolation of the y (elevation)
// values for the single rectangles
for i in 1:IP+2 loop
  for j in 1:IP+2 loop
    x := mod((SM.x[i,j] - OffsetX)/rectangleX *
             * (1-(LengthX/1e6)), 1);
    z := mod((SM.z[i,j] - OffsetZ)/rectangleZ *
             * (1-(LengthX/1e6)), 1);
    /* *(1-(LengthX/1e6)) ensures that the first argument
       of the mod() will not reach one. This would cause x and
       z to get 0 what is wrong as it should be 1. This is
       considered the best solution as it causes invisibly
       small difference in the visualization. */
    X := {x^3,x^2,x,1};
    Z := {z^3,z^2,z,1};
    SM.y[(m-1)*(IP+1)+i, (n-1)*(IP+1)+j] := (H*X)*G*(H*Z);
  end for;
end for;

end for;
end for;

end when;

end SurfaceVisualization;

```

A.3 Paper submitted to Modelica Conference 2009

The following pages contain the paper submitted to the Modelica Conference in Como on the 21. and 22. September 2009.

Object-Oriented Decomposition of Tire Characteristics based on semi-empirical Models

Markus Andres
Vorarlberg Univ. of Appl. Sc.
Austria

Markus.Andres@students.fhv.at

Dirk Zimmer
ETH Zürich
Switzerland

DZimmer@Inf.ETHZ.CH

François E. Cellier
ETH Zürich
Switzerland

FCellier@Inf.ETHZ.CH

Abstract

This article introduces a new and freely available Modelica library, called *Wheels and Tires*, for modeling wheels and tires. The contained models are intended to be used in vehicle simulations, where computational performance is a major concern. Semi-empirical single contact point models are well suited for this kind of applications and are therefore applied in the presented library.

The *Wheels and Tires* library provides a tool to quickly build custom tire models, and allows a convenient customization of existing models. This is achieved by a modular and expandable design system utilizing well established models. In addition, a set of ready-made models is provided to allow a quick insight in the used modeling structure and to enable a direct application in vehicle models. The final version of the library will be published as a free library via the Modelica website as well as the website of F. E. Cellier.

Keywords: object-oriented tire modeling; object-oriented tyre modelling; semi-empirical tire model; tire decomposition

1 Introduction

During the last decades a fairly large number of tire models of varying levels of complexity suiting basically differing fields of application have been developed. These range from simple non-slipping tires to very complex FEA (finite element analysis) models for performance prediction [6].

The library developed is intended to be used in simulations that cover entire vehicles, therefore computational effort is an important issue. Hence, the selection of the appropriate level of detail for the used models is essential for the overall simulation performance. Semi-empirical single contact point models provide a

very good trade-off between accuracy and computational effort. Such models are based on physical considerations, like those emerging from multibody dynamics. These physical aspects get enhanced with empirical formulas representing measurement results that cover e.g. friction and slip characteristics. Two of these semi-empirical models are commonly accepted and widely used. These are TMeasy by G. Rill [11] and the magic-formula model by H. B. Pacejka [10]. However, both are often implemented in a flat and mainly unstructured fashion, which makes them difficult to understand and maintain. Customizing these models for particular situations or expanding them in order to cover new aspects of tires can be cumbersome and is often error-prone.

A paper by D. Zimmer and M. Otter [14] builds on the previously mentioned models and demonstrates how models of varying levels of complexity can be integrated within the object-oriented framework of Modelica. However, the object orientation in these models limits itself primarily to their external interfaces. The models themselves continue to be mostly flat. For instance the most complex tire model created, defines approximately 200 equations [14] and is a good example showing the difficulties that arise from the common flat structure.

Another example for a quite flat structure can be found in the freely available but outdated Vehicle Dynamics library [2]. There, a wheel-base model gets extended with friction models of [11] and [10], but not much further effort was spent regarding object-orientation.

In [1], a tire model is modularized in hub, belt and road elements. A further enhancement is made in [5] by redesigning the model's structure as well as enabling uneven road surfaces and losing contact to the ground due to enhanced vertical dynamics. This modularization is well defined, but still the different aspects of friction are summarized in the *Tyre-Road* class

and can not be customized easily. Moreover the libraries presented in [14], [1] and [5] are not freely available.

The newly developed library takes the object-orientation even further than in [5]. Therefore the focus of this research effort concerns itself less with modeling new tire properties, but more with an improved organization of existing knowledge. This will enable future modelers to conveniently customize the models to their own purposes.

2 Basic Considerations

2.1 A Closer Look at Tires

In motion dynamics of vehicles the forces exerted by the tire-road contact are of major importance. This section is intended to provide basic knowledge about tires buildup. For a more detailed information the reader is referred to [8], [9], [10] or [11].

The modern tire is a complex construction resulting from clashing requirements. They basically have to carry the vertical load, transmit forces to accelerate (and slow down) the vehicle and generate cornering forces to guide the vehicle through curves securely. This has to be fulfilled under a large variety of environmental conditions with a long life time ensured. The rolling resistance has to be as small as possible, with damping and acoustic properties suiting modern demands. As one can imagine there is no optimal solution to this problem resulting in a large variety of different tires for varying demands.

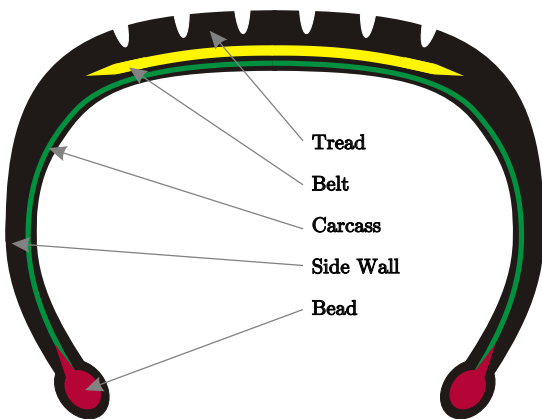


Figure 1: Basic structure of a tire.

A quite basic design example for a tire is depicted in Figure 1. For today's passenger cars steel-belt tires are used exclusively, which differ in construction only marginally, when treated from such a basic point of

view. On the inside of the tire a coating (not depicted in Figure 1) inherits the function of the tube, preventing the over-pressurized air to leak to the outside. The *Bead* is usually built of steel wire with synthetic rubber components, ensuring a tight fit of the tire on the rim, allowing a reliable operation under difficult conditions e.g. when driving over a curbstone. The rubber elements building the sidewall strongly affect the vertical dynamics of the tire and are important when it comes to handling precision and stability. The *carcass* is the element absorbing the tension from the inflation pressure. Therefore, it has to be protected from damage, which is ensured by the *side wall*. The *tread* is responsible for the force generation by establishing a reliable contact to road and is therefore a very central element of the tire. Its composition is a major factor when it comes to the frictional properties of the tire. The *tread* is reinforced by the steel *belt* that enhances mileage and reduces the rolling resistance. Overall a mixture of more than 20 rubber composites form the tire, which makes them quite difficult to describe as well as enabling the tire engineer to adjust the tire properties to different needs.

2.2 Definition of Coordinate Systems

Figures 2 to 5 are intended to present basic as-

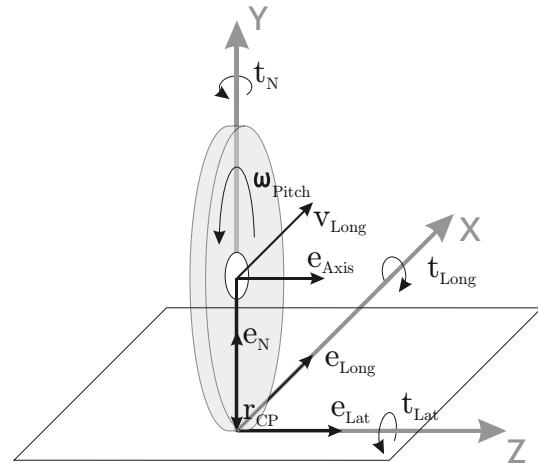


Figure 2: The unitary vectors of the tire without a lean angle and the vector r_{CP} pointing from the rim's center to the contact point.

sumptions made. Figure 2 shows the two coordinate systems used to describe the orientation of a wheel. The contact point's coordinate system is described by e_{Long} , e_{Lat} and e_N , whereas e_{Long} , e_{Axis} and e_{Plane} ¹ form the rim's system. Figure 4 shows the standardized

¹Pointing in the opposite direction of e_N in Figure 2 and shown in Figure 3 and 4.

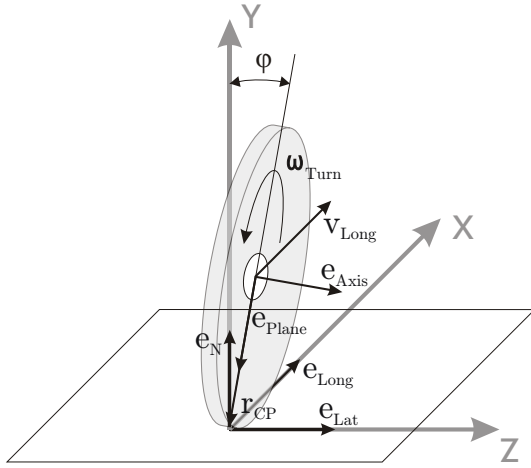


Figure 3: The unitary vectors of the tire with a lean angle φ of 10° .

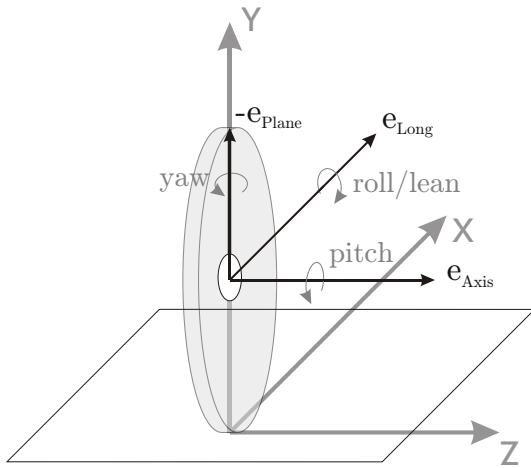


Figure 4: Standardized names of the angles and angular velocities.

names of the angels and their corresponding angular velocities.

An enlarged view of the contact point is depicted in Figure 5. It shows important properties like the translational velocity of the contact point in longitudinal (v_{Long}) and lateral (v_{Lat}) directions, the overall velocity v and the slip angle α . The sliding velocity in longitudinal $v_{SlipLong}$ direction is calculated by $(\omega \times r_{CP}) \cdot e_{Long} + v_{Long}$.

3 Object-oriented Tire Modeling

Section 3.1 is intended to demonstrate the considerations that lead to the actual structure of the tire model. Afterwards Section 3.2 explains the resulting structure from the modeler's point of view. Sections 3.3 to 3.10 shallowly introduce the classes forming the tire.

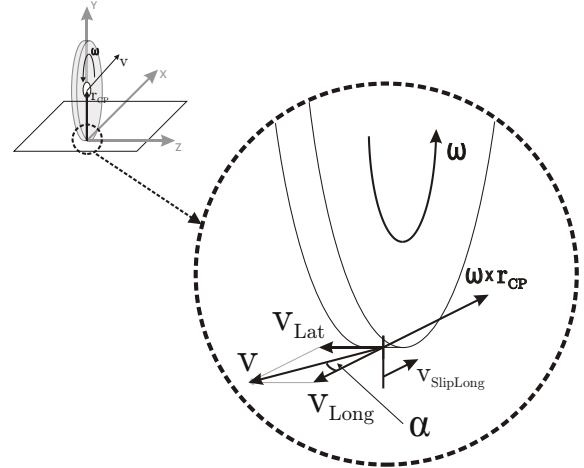


Figure 5: An enlarged view of the contact point with sliding velocities.

3.1 Decomposition into Objects

Thinking about wheels, the first division of the model is quite obvious, as there are two physical components: the rim and the tire. The rim does not need to be split up into further objects, as its properties can be modeled in a quite simple fashion in a semi-empirical tire model. This and the main properties of tires regarding modeling are shown in Figure 6.

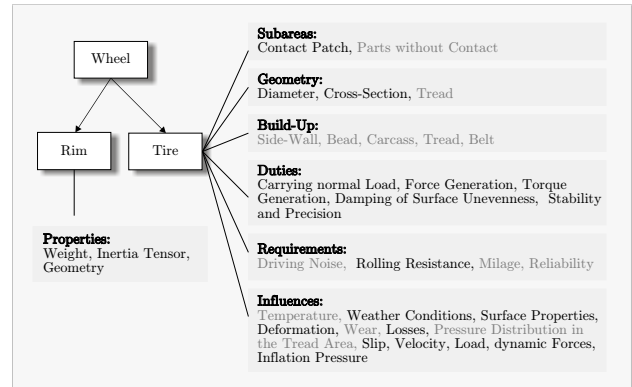


Figure 6: Composition of wheels and properties of tires. Properties depicted in gray are neglected in the presented tire model.

The tire does require a much closer look concerning its properties. Modeling every single of the properties shown in Figure 6 by a class of its own is an infeasible task, suggesting a certain grouping of properties. Due to the semi-empirical single contact point model one constraint is fixed. There has to be a model describing the contact point. It is named the *Contact Physics* model.

One of the most challenging tasks when modeling a tire, is to calculate the forces the tire excites in dif-

ferent driving situations. There are several different models trying to describe the relations resulting in the forces that act on the contact point. Hence a decision was made to create a class that gathers the different effects that are responsible for the generation of forces and torques acting on the contact point. Due to its origin it is called the *Friction* class, resulting in Figure 7.

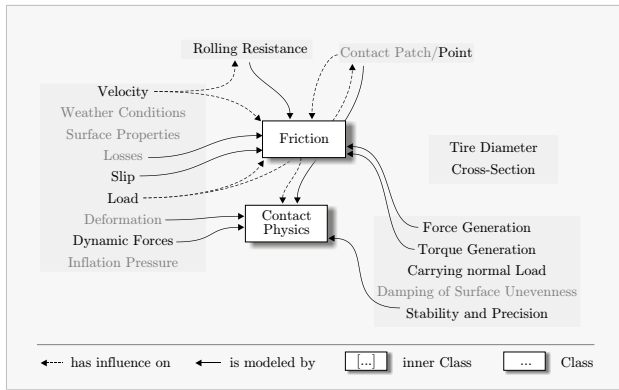


Figure 7: Properties from Figure 6 shown in relation to the corresponding *Tire Component* classes which are closely related with the semi-empirical contact point model. Properties depicted in gray are considerably simplified in the model.

Still, absolutely basic properties of the tire are not yet part of the model as shown in Figure 7. The probably most obvious are the *Tire Diameter* and the *Cross-Section*. These properties basically define the geometry of the tire, which shall be changeable conveniently in the final tire model, allowing basically different geometric properties of the tire. Therefore a *Geometry* class is introduced, defining the positional relation between the tire hub and the contact point and some other properties.

The next very basic duty the tire has to fulfill, is the *Carrying of normal Load* resulting in a normal force. Due to the changing requirements to these aspects, the effects are described in a class named *Vertical Dynamics*.

Until now, all possible tire models would be totally rigid. To enable a certain deformation of the tire, the *Belt Dynamics* class is introduced. It allows a flexibility of the still rigid tire, defined by the *Geometry* class, related to the tire's hub.

The wheel is now modularized into six classes including the *Rim* class which is not depicted in Figure 8, as it was shown in Figure 6. Section 3.2 explains the implemented version of these classes from the modeler's point of view. It also explains why the final model of the tire contains seven classes, introduc-

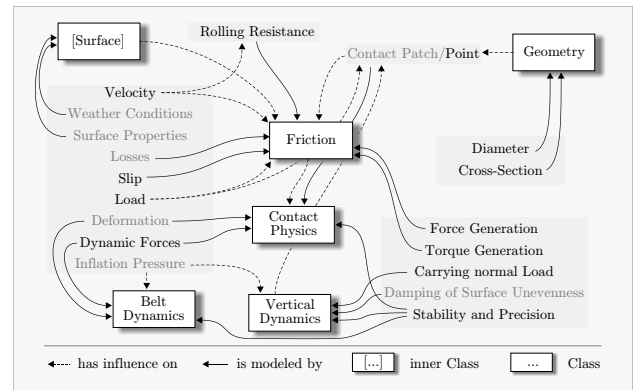


Figure 8: Final decomposition of the tire.

ing a *Center to Contact Point* class.

Still there is one class found in Figure 8 that has not been introduced until now. This is justified as it is an *inner* model and does not form a part of the actual tire model. It realizes uneven surfaces and allows a position-depending friction coefficient. This model is described in Section 5.

3.2 The Tire Model's Structure

The tire is split up into seven objects shown in Figure 9. It consists of objects modeling

- *Vertical Dynamics*,
- *Friction*,
- *Geometry*,
- *Contact Physics*,
- the translation from *Center to Contact Point*,
- *Belt Dynamics* and
- the *Rim*.

The single classes are described in the following sections. Here the relations between the classes shall be illustrated.

The connection to the superordinate vehicle model is established by the three-dimensional frame named *Tire Hub* depicted in Figure 9. The *Tire Hub*, a standard element of the *MultiBondLib* [13], is rigidly connected to the model of the *Rim*. The rim's frame connected to the *Tire Hub* therefore models the center-point of the rim. The "output" of the *Rim* again models the center-point of the *Rim*, and is connected to the *Belt Dynamics* model. In this model the relative movement between the rigid belt and the rim can be described. The "output" of the *Belt Dynamics* is again

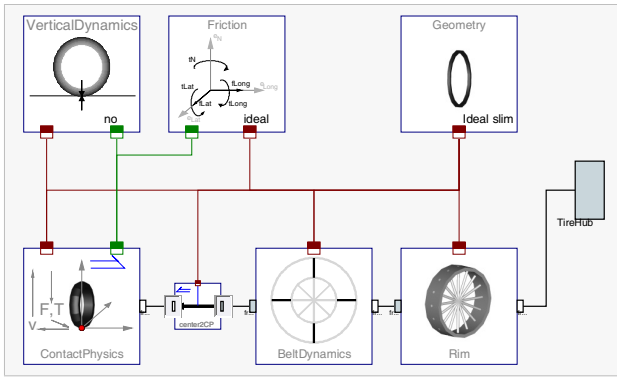


Figure 9: Model of the ideal tire showing the seven used objects and the communication structure on the top level. The *Tire Bus* is colored red whereas the *Contact Point Connector* is green.

positioned at the center-point of the belt. Therefore an element is needed to realize the translation from this point to the contact point. As this cannot be modeled by a standard element, the *Center To Contact Point* model has been created. It connects the *Contact Physics* model that applies forces and torques to the contact point. This lower part of the model represents the mechanically connected part of the model. The upper three classes are not directly connected by a mechanical connector, although the *Contact Point Connector* implies kind of a mechanical connection. The *Vertical Dynamics* class determines if and how the tire is able to lift from the ground and how it responds to normal load. The *Friction* class determines the longitudinal and lateral forces as well as the torques that act on the contact point. Finally, the *Geometry* class determines the unitary vectors shown in Figure 2 and the position of the contact point depending on the actual geometry, position and orientation of the tire.

The visualization is implemented in the corresponding object, e.g. the rim is visualized in the *Rim* object and the tire is visualized in the *Geometry* class. This makes it possible to recognize basic changes to the models in the animation directly.

All of the featured classes that model a certain type of effect are extended from the corresponding base class, ensuring that the necessary output is calculated. This guarantees also that all models stay exchangeable not depending on the implementation of the class. In this way, the user can add new classes or adapt existing ones being sure that the other elements stay unchanged and remain exchangeable.

3.3 Rim Class

The *Rim Class* is a rather simple model, as the rim can be modeled ideally just consisting of a body that has a mass and an inertia tensor.

3.4 Friction

For the sake of object oriented modeling, the different modeled effects are put into subclasses (see Figure 10 as an example) that are of varying complexity. The

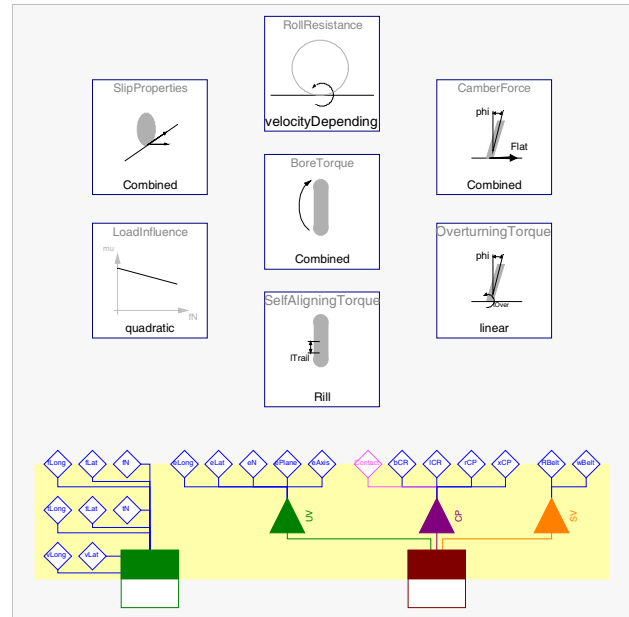


Figure 10: Model of the advanced friction.

communication between the classes is established using *inner/outer* statements. Therefore no connections between the sub-models are visible in Figure 10.

To use a certain combination of frictional effects in a tire model, a class gathering these effects has to be created as shown in Figure 10. This class can then e.g. replace the *Ideal Friction* depicted in Figure 9. In the library three different combinations of *Friction* classes are composed, forming an *Ideal Friction*, one simple *Dry Friction With Rolling Resistance* but without e.g. *Bore Torque* or *Camber Force* and one *Advanced Friction* that is shown in Figure 10. New frictional models can be created easily, which is the reason why only three frictional classes were included in the library.

3.5 Geometry

All geometric classes have two main tasks to fulfill. The first is to determine the contact point properties including the vector r_{CP} that points from the center of the rim to the contact point and the penetration depth.

Secondly the unit vectors shown in Figure 2 get computed by the *Geometry* class. To enable that functionality it utilizes the *Surface*'s functions *get_eN* and *get_elevation* establishing the connection from the tire to the surface.

Three different *Geometry* classes are provided for ideally *Slim*, *Circular* tires with cross-section being modeled as a semi-circle, and a “*Belt*” profile with side walls and a sector of a circle modeling the tread area.

3.6 Contact Physics

The *Contact Physics* model applies forces and torques on the contact point, as well as measuring its velocities. All these physical quantities are connected to the models determining frictional and vertical dynamic properties via the *Contact Point Connector*. Measuring and setting of speeds and forces has to be done in an a-casual way as ideal models set velocities rather then apply forces.

The second property determined by the *Contact Physics* class is the dynamic behavior of the contact point. It can introduce flexibility of the contact point in longitudinal and lateral direction.

3.7 Center to Contact Point

The *Center To Contact Point* class is a model without a nice physical interpretation. It is kind of a *Fixed Translation* element known from the *Modelica Standard Library*. The model is built using multi-bond graphs and therefore derived from the *Fixed Translation* in the *MultiBondLib* [13]. It describes the connection between the contact point and the belt's center point.

3.8 Vertical Dynamics

The models in the *Vertical Dynamics* package determine the behavior of the tire normal to the surface. The normal force is related to the penetration depth computed in the *Geometry* class. The *Vertical Dynamics* models can either set the penetration depth to a certain value, or use it as a base for the calculation of the normal force f_N .

There are basically three different approaches to model vertical dynamics in the library. One is to not allow any elevation from the ground or penetration into it introducing a holonomic constraint. The second utilizes an *ElevationGap* model that compensates forces of a attached 1D mechanical model when the tire lifts from the ground. It is a derivation from the *ElastoGap*

model found in the *BondLib* [7]. It makes the vertical dynamics easily changeable by a modification of the 1D mechanical system. The third is a derivation of the *ElastoGap* model of the *Modelica Standard Library 3.0*. It overcomes the sticking effect of the *Elevation-Gap* but is harder to adapt to different dynamics.

3.9 Belt Dynamics

The models described in this section allow the tire to have a certain flexibility. This is realized by connecting an ideal (virtual) belt to the ideal rim in a flexible fashion. All models except the *Rigid Belt* model add a considerable amount of complexity to the simulation. Different models are provided to realize the dynamic behavior. One allows translation of the belt in longitudinal and lateral direction, another models a rotational degree of freedom around the axis of rotation, both defining the dynamics by 1D mechanical elements. The last model is the most complex with the dynamics defined by four ideally stiff translational elements for rim and belt respectively, connected by 3D spring damper systems.

3.10 Communication Structure

The tire's objects compute a bunch of different variables, some of which are used in different objects as well. The *Tire Bus*, with a connector as shown in Figure 11, gathers the variables used in most of the objects. Additionally a division into records of similar variables ensures a better overview and allows a more convenient graphical connection. For each of these sets of variables separate inputs and outputs have been created. This makes it possible to show, in which objects variables are computed or just used (see Figure 11). The second communication structure is the

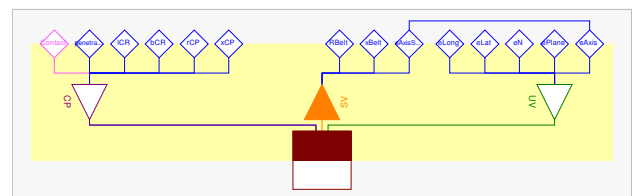


Figure 11: Separation of the *Tire Bus* connector (lower quadratic element) within the *Geometry Class* in an input for *Sensor Values* (SV) and outputs for *Contact Properties* (CP) and *Unit Vectors* (UV). The rhombuses define protected variables used in the model.

connector that contains the velocities as well as the forces and torques that act on the contact point. It

is named *Contact Point Connector*. This connector is implemented in an a-causal manner due to the requirements of this connection. It can be found connecting *Contact Physics*, *Vertical Dynamics* and *Friction*. Both bus connectors are illustrated in Figure 9.

4 Provided Tire Models

All tires are basically built up like the ideal tire in Figure 9. In the library eleven ready-made tires are provided for a quick application and a better understanding of the model structure. Four models of slim tires include three rigid versions and one with a dynamically behaving contact point. Two tires with semi-circular cross section are provided, one rigid and the other having rotational dynamics. The so-called belted tires feature three rigid models differing in frictional behavior. The other two belted tires behave dynamically.

5 Environment

The *Environment* in the current version of the library is limited to even or uneven but slowly changing surfaces, which is a limitation arising from the single contact point model. Basically the method to be implemented has to be able to compute elevation (the y coordinate shown in Figure 12) and the normal vector on the surface. There are many significantly different approaches, whereas the one chosen is rather simple but still conveniently configurable.

5.1 Surface Base

The *Surface Base* (Section 5.1) class ensures compatibility with future enhancements. This partial inner model defines the functions necessary to calculate the behavior of the tire. These include the following functions.

- *get_eN_Base* – returning the normal vector of the surface with the actual x and z coordinates as inputs.
- *get_elevation_Base* – returns the y coordinate of the surface (x and z are inputs again).
- *get_mu_Base* – returns the frictional coefficient at the x and z coordinates of the contact point.

5.2 Surface Class

The implemented version of the *Surface* in the *Wheels and Tires* library is based on the method shown in [4]. It is an interpolation in a unit square based on four y values and the eight corresponding partial derivatives. A sketch of the unit square is shown in Figure 12. To enable the user to define more complex

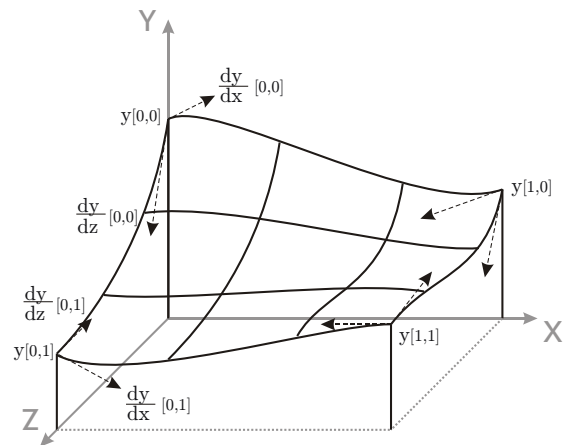


Figure 12: The basis for the interpolation used to find the elevation y and normal vector in one square.

surfaces an arbitrary amount of interpolated elements can be combined to one surface of definable size. The only fact limiting the amount of rectangles is simulation time, especially the time consumed for compilation that rises quickly with growing surfaces.

6 Simulation Results

The following sections are intended to give an impression of what the simulation results look like. Therefore, results from the *Test Bench* package and from the *Example* package are depicted.

6.1 Test Bench

Figures 13 to 16 show results from different models in the *Test Bench* package. The models are used to test basic functionalities of the tire and the surface classes.

6.2 Examples

Six examples are included in the library demonstrating the application of the models in vehicles. Five of the Examples are two-wheeled (single-track) vehicles

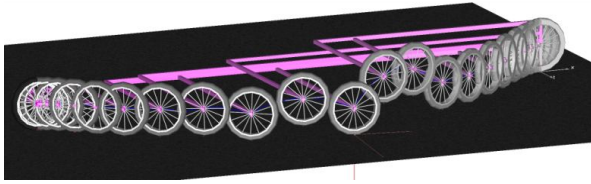


Figure 13: Tire elevating from the ground while driving a curve.

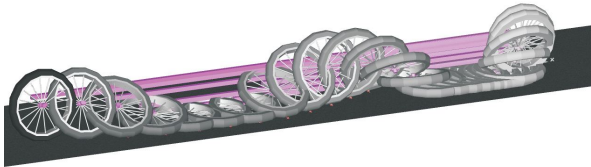


Figure 14: Tire's overturning torque *Test Bench* result.

that are provided by [12] and copied to the *Used Models* in the *Examples* package of the *Wheels and Tires* library. Three of the five examples are unsprung bicycle models composed of rigid elements exclusively. The other two models are sprung motorcycles including front and rear suspension. For both vehicles, models are provided to analyze the uncontrolled stability, e.g. with different tire properties for geometry and friction as shown in Figures 17 and 18.

The other two models are “nice to show” models with the bicycle being accelerated by a strong torque to make the front wheel lift from the ground as shown in Figure 19 and the motorcycle jumping over a gap as depicted in Figure 20.

The last example is a very basic unsprung model of a four-wheeled vehicle with a predefined steering angle profile and driving torques acting on the rear tires. An impulse in the driving torque makes the vehicle-model slide after a certain simulation time, with the

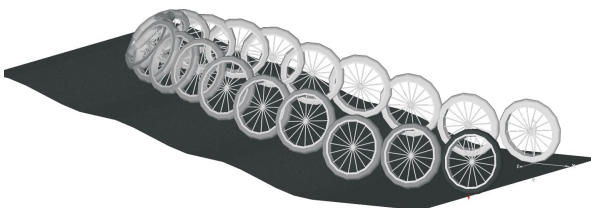


Figure 15: A tire rolling up an uneven surface with an initial lean angle.

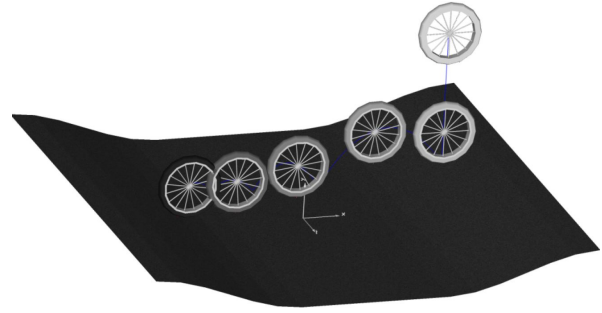


Figure 16: Tire dropping on an uneven surface.



Figure 17: Two similar bicycles with non-slipping tires differing in their geometric properties. The upper bicycle is equipped with ideally slim tires, whereas the lower one driving a narrowing curve has a belted tire with a width of 2cm.

model “reacting” by a full breaking, which makes the wheels slide to a standstill of the vehicle.

Regarding the simulation speeds of the examples it can be stated, that the bicycle models take about half of the simulated time for the computation of the result. The motorcycle jumping over the gap takes about 250s of calculation time for 16s of actual simulation. The reason for that is the more complex structure of the motorcycle in comparison with the bicycle. The again undamped four-wheeled models compute the results in a little less time than the simulation time. All models used non-dynamic tires either with *slipping* or the *advanced* friction models. The simulations have



Figure 18: Two similar bicycles with slipping tires differing in their frictional properties. The one driving the inner circle is equipped with the *Advanced Friction* class, whereas the outer bicycle is equipped with the *Dry Friction with Constant Roll Resistance* class.

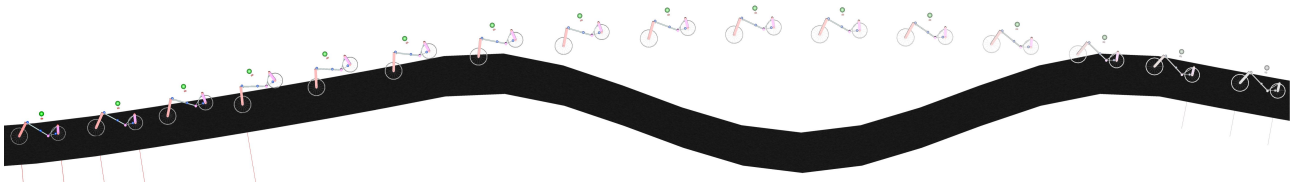


Figure 20: A motorcycle jumping over a gap.



Figure 19: A bicycle with non-ideal tires accelerated by a torque on the rear tire, with the front tire lifting from the ground.

been carried out on a Intel Core2Duo clocked at 2GHz and equipped with 4GB RAM, computing 250 output intervals per second. It has to be mentioned that no big effort was spent regarding the optimization of simulation speed of the overall models. A more suitable combination of state variables would most likely lead to a considerable enhancement in speed.

7 Library Structure

This section introduces the top level packages that are contained in the *Wheels and Tires* library and are depicted in Figure 21.

The *Tire Components* package covers the classes that the *Tires* are built of. The contained sub-packages are described in Sections 3.3 to 3.10. These classes form the ready-made tires contained in the *Tires* package as one example shows for the *Ideal Tire* as depicted in Figure 9.

The *Environment* package contains the *Surface Base* as well as a possible implementation for even and uneven surfaces. It is described in Section 5. The *Test Bench* and the *Example* Package are top-level packages as well. They are provided to test the tire models' functionality and get an insight into the usage of the models. A short description can be found in Sections 6.1 and 6.2 respectively. Finally the *Visualization* package gathers a few models used to enable the animation of all necessary parts of the library.

8 Conclusion

To sum up, the library enables a quick and convenient building of easily customizable tire models. There are some further enhancements possible but the structure of the tire models should persist as it is well expandable, fulfilling the major requirements that were demanded at the beginning of the work. Still the lack of comparison to real applications and measurement data is considered to be a drawback as some minor malfunctions could probably not be identified.

For the use of the models in real-time applications a further optimization of the computational efficiency would be desirable to ensure quick enough simulation. Furthermore a prediction of the influences that different objects have on the computational effort of the overall tire would be of advantage.

A more detailed description of the library can be found in the corresponding master's thesis [3].

References

- [1] J. Andreasson and J. Jarlmark. Modularised tyre modelling in modelica. In *Proceedings of the Second International Modelica Conference, Oberpfaffenhofen, Germany*, pages 267–274, 2002.
- [2] Johan Andreasson. Vehicledynamics library. In *Proceedings of the Third International Modelica Conference, Linköping, Sweden*, pages 11–18, 2003.
- [3] Markus Andres. Object-oriented modeling of wheels and tires in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.
- [4] G. Aumann and K. Spitzmüller. *Computerorientierte Geometrie*. BI-Wiss.-Verl, 1993.
- [5] Mats Beckmann and Johan Andreasson. Wheel model library for use in vehicle dynamic studies. In *Proceedings of the third Modelica Conference, Linköping, Sweden*, pages 385–392, 2003.

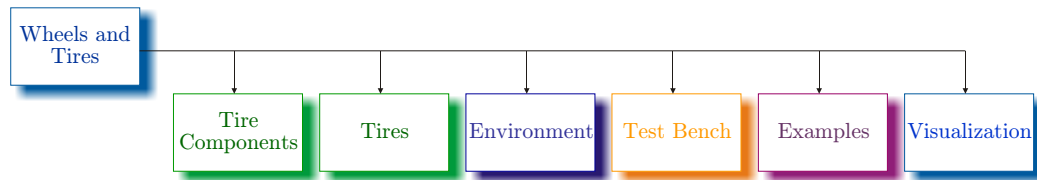


Figure 21: The library's top level packages.

- [6] P. Bohara, A. Saha, P. Ghosh, and M. Roopak. Tyre performance prediction through fea. Hasetri - Hari Shankar Singhania Elasoimer and Tyre Research Institute, 2008.
- [7] F. E. Cellier and À. Netbot. The modelica bond-graph library. In *Proceedings of the 4th International Modelica Conference, Hamburg*, pages 57–65, 2005.
- [8] John C. Dixon. *Tires, Suspension and Handling*. Cambridge University Press, 1996. Second Edition.
- [9] Günter Leister. *Fahrzeugreifen und Fahrwerkentwicklung*. Vieweg + Teubner, 2009. 1. Auflage.
- [10] Hans B. Pacejka. *Tyre and Vehicle Dynamics*. Butterworth-Heinemann, 2006. Second Edition.
- [11] Georg Rill. *Simulation von Kraftfahrzeugen*. Vieweg-Verlag, 2007. genehmigter Nachdruck.
- [12] Thomas Schmitt. Modeling of a motorcycle in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.
- [13] Dirk Zimmer. A modelica library for multibond graphs and its application in 3d-mechanics. Master's thesis, ETH Zürich, 2006.
- [14] Dirk Zimmer and Martin Otter. Real-time models for wheels and tires in an object-oriented modeling framework. Accepted for publication in *Vehicle Dynamics*, 2009.

A.4 Paper submitted to “Tag der Mechatronik 2009”

The following pages contain the paper submitted to “Tag der Mechatronik 2009” in the Vorarlberg University of Applied Sciences, Dornbirn.

FH VORARLBERG: OBJECT-ORIENTED MODELING OF WHEELS AND TIRES IN DYNOMLA/MODELICA

Markus Andres

Abstract: The Master's Thesis introduces a new Modelica library designed for modeling and simulation of wheels and tires as part of a complete vehicle model. The library is created in a fully object-oriented fashion with a modular and expandable design system. This enables a more flexible and maintainable modeling and simulation of wheels and tires utilizing well established models. It will be made freely available.

Key words: object-oriented tire modeling; object-oriented tyre modelling; semi-empirical tire model; tire decomposition

1. INTRODUCTION

During the last decades a fairly large number of tire models of varying complexity levels suiting strongly differing fields of application have been developed. These range from simple non-slipping tires to very complex FEA (finite elements analysis) models for performance prediction.

The library developed is intended to be used in simulations that cover entire vehicles, therefore computational effort is an important issue. Hence, the selection of the appropriate level of detail for the used models is essential for the overall simulation performance. Semi-empirical single contact point models provide a very good tradeoff between accuracy and computational effort. Such models are based on physical considerations, like those emerging from multi-body dynamics. These physical aspects get enhanced with empirical formulas representing measurement results that cover e.g. friction and slip characteristics.

Two of these semi-empirical models are commonly accepted and widely used. These are TMeasy by G. Rill (Rill, 2007) and the magic-formula model by H. B. Pacejka (Pacejka, 2006). However, both are often implemented in a flat and mainly unstructured fashion, which makes them difficult to understand and maintain. Customizing these models for particular situations or expanding them in order to cover new aspects of tires can be hard and is often error-prone.

A paper by (Zimmer & Otter, 2009) builds on the previously mentioned models and demonstrates how varying levels of complexity can be integrated within the object-oriented framework of Modelica. However, the object orientation in these models limits itself primarily to their external interfaces. The models themselves continue to be mostly flat. For instance the most complex tire model created, defines approximately 200 equations (Zimmer & Otter, 2009) and is a good example showing the difficulties arising from the common flat structure.

In (Andreasson & Jarlmark, 2002) a tire model is modularized in hub, belt and road elements. A further enhancement is made in (Beckmann & Andreasson, 2003), enabling uneven road surfaces and loosing of contact to the ground. This modularization is well defined, but still the frictional properties are summarized in the road class and can not be customized easily.

The newly developed library takes the object-orientation further than in (Beckmann & Andreasson, 2003). Therefore the focus of this research effort concerns itself less with modeling new tire properties, but more with an improved organization of

existing knowledge utilizing established models developed by other researchers. This will enable engineers to conveniently customize the free models to their own purposes and add further tire models and additional properties of tires.

2. OBJECT-ORIENTED TIRE MODELING

Figure 1 shows the most important properties of wheels and tires regarding modeling, demonstrating that a gathering of tire properties is necessary to design a manageable model. The result of this aggregation is shown in Figure 2, depicting five of the seven objects forming the final wheel model. The inner *Surface* is disregarded as it is not part of the tire model directly.

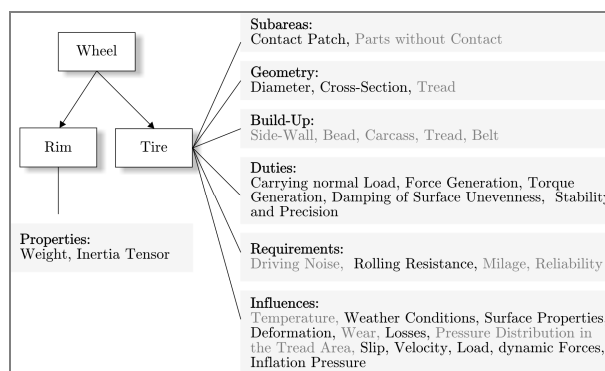


Figure 1: Properties of a Wheel split in Rim and Tire, whereas the gray properties are not covered in the resulting model.

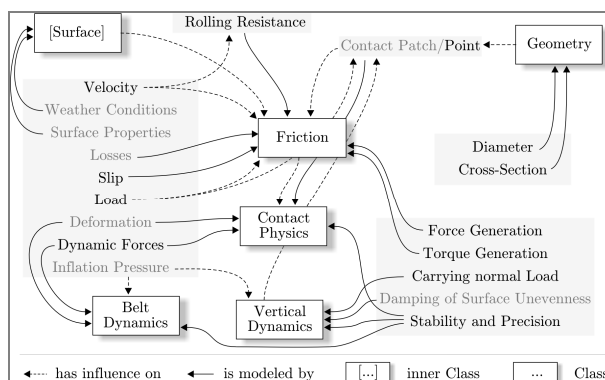


Figure 2: Aggregation of modeled tire properties.

Figure 3 shows the implementation of the structure developed in Figure 2 in the simulation environment Dymola. It includes the *Rim* class from Figure 2 and a *Center to Contact Point* class which is necessary due to the structure of the model.

Looking at Figure 3 the connection to the super-ordinate vehicle model is established by the three dimensional frame named *Tire Hub*. The *Rim*'s right frame is rigidly connected to the *Tire Hub*, therefore modeling its center-point. The "output" of the *Rim* again models its center-point, which is connected to the *Belt Dynamics* model. In this model the relative movement

between the rigid belt and the rim can be described. The “output” of the *Belt Dynamics* is again positioned at the center-point of the belt. Therefore an element is needed to model the translation from this point to the contact point. This is realized by the *Center To Contact Point* class, which connects the *Contact Physics* model that applies forces and torques to the contact point, as well as measuring its velocities. The latter both elements are modeled utilizing multi bond graph techniques (Zimmer, 2006). This lower part of the model represents the mechanically connected part of the model. The *Vertical Dynamics* class determines if and how the tire can lift from the ground and how it responds to normal load. The *Friction* class computes the longitudinal and lateral forces as well as the torques which act on the contact point. Finally, the *Geometry* class provides the unitary vectors and the position of the contact point depending on the actual geometry.

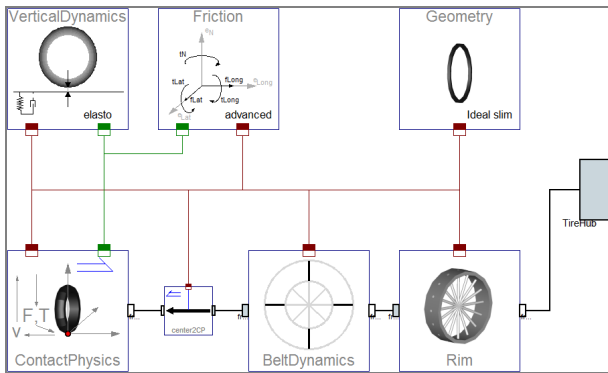


Figure 3: Model of the ideally slim slipping tire showing the seven used objects and the communication structure.

Figure 3 also shows the specifically developed *Communication Structure*, tailored to the special needs and user friendliness. It features the *Tire Bus* connecting all classes and the *Contact Point Connector* which can be found connecting *Contact Physics*, *Vertical Dynamics* and *Friction* models.

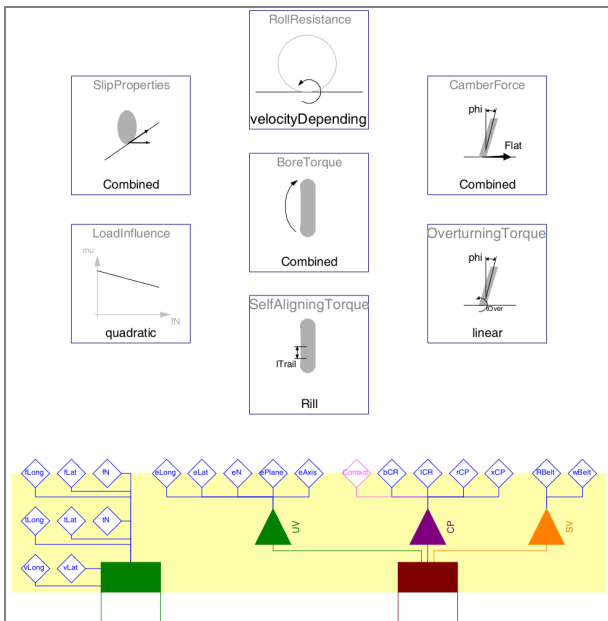


Figure 4: Model of the Advanced Friction from Figure 3

As an example, the model of the *Friction* class depicted in Figure 3, is expanded in Figure 4. For the sake of object oriented modeling, the effects are put into objects of varying complexity. The internal Communication is realized by *inner/outer* statements, therefore no connections are visible.

The connection to other objects is realized by the shaded part of the model, where the busses are split into single variables. As throughout the whole library, all featured classes modeling a certain type of effect extend the corresponding base classes. This ensures that the necessary output is calculated and that models are exchangeable independent of their implementation.

3. RESULTS

All of the library’s tires build upon the same structure as the slim slipping tire in Figure 3. In the library eleven ready-made tires are provided for a quick application and understanding of the model's structure. Additionally *Test Bench* models (e.g. Figure 5) are provided to check the basic functionality of tires and the *Surface* package. Latter one enables the simulation of conveniently definable uneven road profiles. To get an impression of how to use the tires in vehicle simulations, six models are gathered in the *Examples* package including single track vehicles (e.g. Figure 6) and a four-wheeled vehicle.

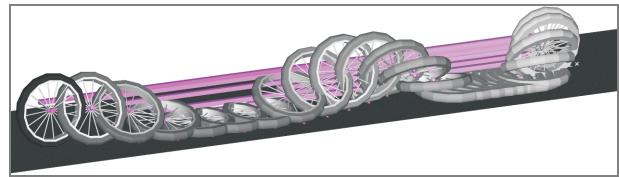


Figure 5: Result of the *Camber Force Test Bench*.

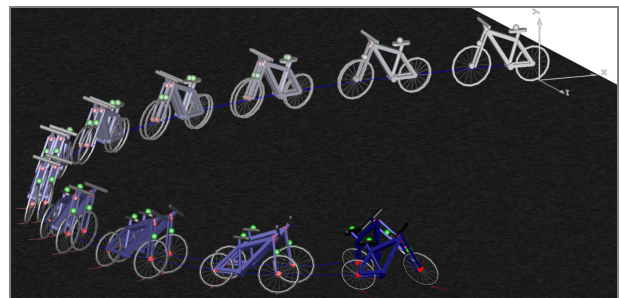


Figure 6: Identical uncontrolled bicycles from (Schmitt, 2009) with tires differing in frictional properties (*Examples* package).

9. CONCLUSION

To sum up the library enables a convenient building of easily customizable tire models. There are some further enhancements possible, but the structure of the tire models should persist as it is well expandable, fulfilling the requirements which were demanded at the beginning of the work.

10. REFERENCES (EXCERPT)

Andreasson J. & Jarlmark J. (2002). Modularised Tyre Modelling in Modelica, In: *Proceedings of the Second International Modelica Conference, Oberpfaffenhofen, Germany*, pp. 267-274

Beckmann, M. & Andreasson, J. (2003). Wheel model library for use in vehicle dynamic studies, In: *Proceedings of the 3rd Modelica Conference, Linköping, Sweden*, pp 385-392

Pacejka, H. B. (2006). *Tyre and Vehicle Dynamics Second Edition*. Butterworth-Heinemann

Rill, G. (2007). *Simulation von Kraftfahrzeugen*. Vieweg

Schmitt, T. (2009). *Modeling of a Motorcycle in Dy-mola/Modelica*. Master’s Thesis

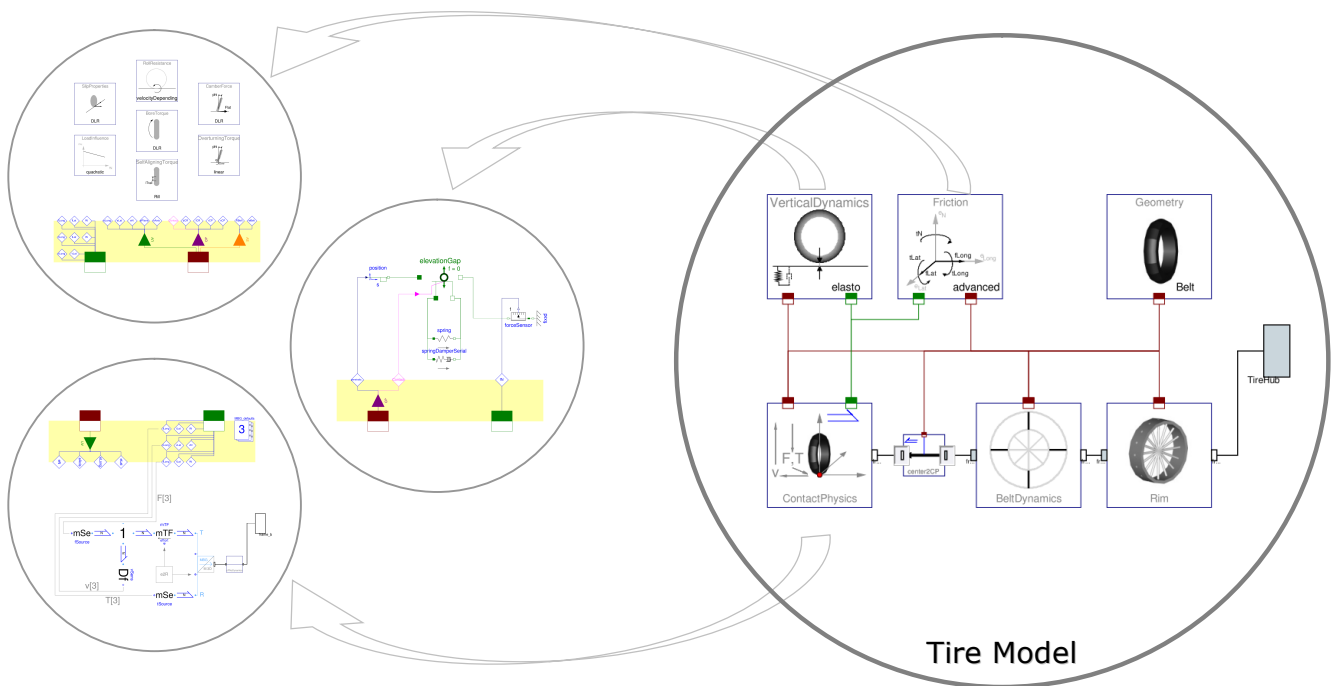
Zimmer D. (2006). *A Modelica Library for Multibond Graphs and its Application in 3D-Mechanics*. Master’s Thesis

Zimmer D. & Otter M. (2009). *Real-time models for wheels and tyres in an object-oriented modeling framework*. Accepted for publication in *Vehicle Dynamics*

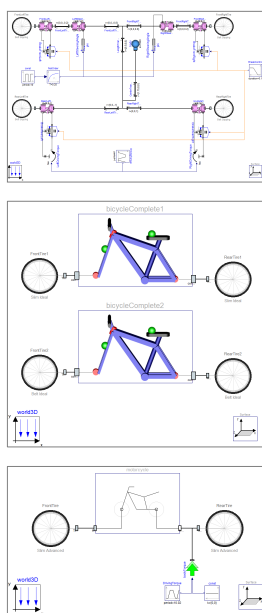
Object-Oriented Modeling of Wheels and Tires in Dymola/Modelica

The *Wheels and Tires* library provides a tool to quickly build custom tire models. This is realized by a modular and expandable design system utilizing well established models. In addition, a set of ready-made models is provided to get a quick insight in the used modeling structure and to enable a direct application in vehicle models. The final version of the library will be published as a free library via the Modelica website as well as the website of F. E. Cellier.

Tire Model Structure



Modeling



Simulation

