DIPLOMARBEIT

Objektorientierte Modellierung und dynamische Co-Simulation mit CATIA V6 am Beispiel von Kraftfahrzeugsystemen

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

unter der Leitung von Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Horst Ecker
sowie Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker

am Institut für Mechanik und Mechatronik

eingereicht an der Technischen Universität Wien
Fakultät für Maschinenwesen und Betriebswissenschaften

von

Peter Smolek

0625538

Pramergasse 8-10/2/16, 1090 Wien

Datum

Unterschrift

Danksagung

Ich möchte Prof. Horst Ecker für die gewissenhafte Betreuung meiner Diplomarbeit danken. Auch in stressigen Zeiten stand er mir immer mit Rat zur Seite. In diesem Rahmen möchte ich mich bei Prof. Felix Breitenecker bedanken, der die interfakultäre Zusammenarbeit in einer freundlichen Atmosphäre ermöglicht hat.

Mein größter Dank gebührt Dipl.-Ing. Bernhard Heinzl für die unzähligen Stunden an Unterstützung im Laufe der Diplomarbeit. Sowohl in inhaltlicher als auch organisatorischer Hinsicht konnten mit seiner Hilfe sämtliche Probleme aus dem Weg geräumt werden.

Ich danke Dipl.-Ing. Nikolas Popper für die Untersützung der dwh GmbH, ohne deren Know-How und Infrastruktur die Arbeit nicht möglich gewesen wäre.

Ein besonderes Dankeschön geht an meine Freundin Cornelia Spreitzer, die auch in schwierigen Zeiten immer für mich da war und an mich geglaubt hat.

Zu guter Letzt möchte ich mich herzlich bei meinen Eltern bedanken. Ohne sie wäre mein Studium nicht möglich gewesen. Sie haben mich in meinen Entscheidungen immer unterstützt und mir so die Möglichkeit gegeben, meinen eigenen Weg zu finden.

Kurzfassung

Die Vernetzung von verschiedenen Stationen der virtuellen Produktentwicklung eröffnet neue Wege für Innovation und Effizienz. Beispielsweise trägt eine Verbindung von Computer-Aided Design (CAD) mit physikalischer Systemsimulation dazu bei, bereits in der Entwurfsphase systematisch Auswirkungen eines Bauteils auf das Gesamtsystem analysieren zu können. Ein derartiger Zugang erfordert neuartige Simulationsansätze, etwa im Rahmen einer Cooperative-Simulation (Co-Simulation). In diesem Sinne ist das Ziel der Arbeit, die Möglichkeiten der Co-Simulation mit CATIA V6 Dynamic Behavior Modeling zu beleuchten. Aufbauend auf den Grundlagen der akausalen, objektorientierten Modellierung mit Modelica werden mögliche Implementierungen von Co-Simulation diskutiert.

Zwei verschiedene Ansätze der Einbindung von CATIA V6 in die Berechnung, vor allem in Hinblick auf den Zugriff auf Geometrie-Daten aus CAD-Modellen, werden vorgestellt und bewertet. Auf Basis von Studien zu Implementierung, auftretenden Fehlern und Flexibilität der Auswahl an Simulatoren wird das Building Controls Virtual Test Bed (BCVTB) als Software-Backbone dem Functional Mockup Interface (FMI) vorgezogen. Um eine effiziente Simulation zu gewährleisten, wird eine Methode zur Parallelisierung der Co-Simulation untersucht.

Als Anwendungsbeispiel demonstriert das Modell eines Kraftfahrzeugs die Möglichkeiten der Co-Simulation in Verbindung mit CATIA V6. Der Einsatz der Modelica-Sprache in der Dynamic-Behavior-Modeling-Umgebung erlaubt es dabei, die notwendigen mechanischen Komponenten objektorientiert zu implementieren. Spezieller Wert wird auf die Umsetzung der Räder gelegt, vor allem im Hinblick auf flexible Einsetzbarkeit des gesamten Modells. Der Antrieb wird als datenbasiertes Modell über eine Motorkennlinie in MATLAB aufgebaut, das Hauptgetriebe ist durch die Parameter der Getriebeübersetzung enthalten. Die Wahl des Gangs kann durch einen Statechart in Stateflow übersichtlich abgebildet werden. Verschiedene Szenarienrechnungen beleuchten das Verhalten des Modells genauer und zeigen Vor- und Nachteile sowie Anwendbarkeit und Grenzen der gewählten Modellierungsansätze auf. Aufbauend auf den verwendeten Methoden werden dabei Möglichkeiten aufgezeigt, wie Co-Simulation mit CATIA V6 zukünftig im Prozess der Produktentwicklung eingesetzt werden kann.

Abstract

The connection of the different aspects in the process of virtual product development offers new potential for innovation and efficiency. For instance, the combination of Computer-Aided Design (CAD) and physical systems simulation helps analysing the effects of a component on the entire system in the design stage. This idea requires advanced approaches in computational engineering, for example through Cooperative-Simulation (Co-Simulation). In this context the main goal of this study is to investigate the possibilities of Co-Simulation with CATIA V6 Dynamic Behavior Modeling. Starting with the basic principles of acausal, object-oriented modeling with Modelica, methods of implementing a Co-Simulation are discussed. Two different approaches for integrating CATIA V6, especially with regard to accessing the geometry data of CAD models, are presented and assessed. By examining methods of implementation, simulation errors and flexibility of simulator choices, the Building Controls Virtual Test Bed (BCVTB) is preferred to the Functional Mockup Interface (FMI). To ensure efficient computation, a method for parallelisation of a Co-Simulation is investigated. As an example, the model of a motor vehicle demonstrates the possibilities of Co-Simulation in connection with CATIA V6. The application of the Modelica language allows an object-oriented description of mechanical components. Special attention is paid to the model of the wheel, specifically on the flexible usage with regard to the whole automobile. The engine is implemented as a data-based model in MATLAB, based on its characteristic behaviour. The main gear box is incorporated by inclusion of the transmission ratios. The selection of the gear can be represented as a state chart in Stateflow. Various scenarios are simulated and the behavior of the model with the pros and cons of the chosen approaches are discussed. The possibilities and limitations of the applicability of the model are shown. With the presented methods, future potential of using Co-Simulation with CATIA V6 for virtual product development arises.

Inhaltsverzeichnis

1. Einleitung

Der Prozess der Produktentwicklung wird zunehmend von vielen verschiedenen Einflussfaktoren bestimmt. Die Komplexität der Produkte wächst stetig an, etwa im Hinblick auf Ausprägung und Varianten oder auch der zunehmenden Interdisziplinarität. Zusätzlich werden die Prozesse in der Entwicklung durch die Verteilung auf verschiedene Standorte und die Abhängigkeit von Zulieferern immer vernetzter. Für die Planung, Auslegung und Simulation gibt es unzählige Softwaretools, die wieder über eigene Schnittstellen und Datenformate verfügen, was zur Komplexität des Gesamtprozesses beiträgt.

Das Product Lifecycle Management (PLM) ist ein Konzept, um sämtliches Wissen über ein Produkt zu verwalten und es über die gesamte Lebensspanne zu begleiten. Alle beteiligten Stationen verfügen über dieselbe Wissensbasis und arbeiten verzahnt miteinander. Über den Lebenszyklus, von der Entwicklung über Produktion sowie Gebrauch, Nachgebrauch und Recycling, werden die anfallenden Informationen zentral gespeichert. Das Ziel ist es, Produktkomplexität bewältigbar zu machen und dadurch Innovationen zu fördern. In diesem Kontext wird die Entwicklung von Produkten genauer betrachtet.

Die Idee der integrativen virtuellen Produktentwicklung besteht darin, sämtliche relevanten Informationen zur Verfügung zu haben und diese allen Phasen der Entstehung zugänglich zu machen. Für diesen Ansatz ist es notwendig, das Produkt selbst in den Fokus zu stellen. Die verschiedenen Werkzeuge wie Computer Aided Design, Computer Aided Engineering oder Systemsimulationen werden auf das Produkt abgestimmt. Von den ersten Auslegungsrechnungen über die Erstellung von technischen Zeichnungen bis zu Simulationen stehen alle Beteiligten in Kontakt und tauschen Informationen aus. Auf diese Art kann nicht nur das Verhalten des Systems frühzeitig untersucht werden, sondern auch neue innovative Lösungsansätze gefunden werden.

In dieser Rolle kann Cooperative-Simulation helfen, Aussagen über das Verhalten von Komponenten zu treffen, die noch nicht gefertigt wurden. Co-Simulation erlaubt verschiedene Methoden der Beschreibung von Systemen zu nutzen. Damit ist es möglich, Modelle zu erstellen, die signalflussbasiert, objektorientiert, auf Daten beruhend, statischer oder dynamischer Natur sind. Die an den Modellen arbeitenden Personen können die jeweils für die Problemstellung am besten geeigneten Werkzeuge benutzen. Dadurch werden Ressourcen gespart, da sich nicht alle Expertinnen und Experten auf ein Programm einigen müssen, sondern ihre Präferenzen und Erfahrungen ausnutzen können. Der zweite Vorteil besteht darin, dass komplexe Problemstellungen sehr einfach abgebildet werden können, die sonst schwer zu beschreiben wären. Umfassende Modelle

erlauben es zusätzlich Wechselwirkungen und Rückkopplungen zwischen Teilkomponenten zu berücksichtigen, die in Einzelsimulationen sonst nicht erfassbar wären. Beispiele hierfür sind Multidomain-Anwendungen, das Simulieren steifer Systeme oder das Event Handling. Auf diese Art können sehr heterogene Systeme effizient modelliert werden.

Das Hinzuziehen von CATIA V6 bindet den Bereich des Computer Aided Design in den Prozess mit ein. Bei der Entwicklung von mechanischen Komponenten ist dieses Werkzeug seit vielen Jahren weit verbreitet. Eine enorme Menge an Informationen ist in den zugrundeliegenden Dateien vorhanden. Diese Informationen für eine Simulation nutzbar zu machen, trägt den Gedanken der integrativen virtuellen Produktentwicklung weiter. Durch das in CATIA V6 Release 2012 implementierte Tool für die dynamische Systemsimulation, genannt Dynamic Behavior Modellng (DBM), wird die Verbindung von Computer Aided Design und dynamischer Systemsimulation möglich. Die DBM-Umgebung basiert auf der Modelica-Sprache und kann daher für die objektorientierte Modellierung physikalischer Systeme herangezogen werden. Durch diese Schritte wird der Weg zu einem Knowledge-Enabled Engineering geebnet.

Das Ziel der Arbeit besteht darin, die Möglichkeiten von Co-Simulation mit CATIA V6 zu ergründen. Im Speziellen ist die Integration der Informationen aus dem Computer Aided Design von Interesse. Es werden zwei unterschiedliche Ansätze herangezogen, eine Co-Simulation aufzusetzen und die Vor- und Nachteile diskutiert. Zur Demonstration der umfassenden Einsatzmöglichkeiten wird ein typisches Anwendungsbeispiel für die dynamische Systemsimulation mechanischer Komponenten implementiert und getestet.

2. Grundlagen der objektorientierten Modellierung physikalischer Systeme

In Naturwissenschaft und Technik ist eines der wesentlichen Ziele, das Verhalten von Prozessen und Systemen zu verstehen. Dieses Verständnis ist notwendig, um Phänomene, die in der Natur vorkommen, technisch nutzbar zu machen, oder bestehende Technologien auf Fehler oder Verbesserungspotential zu untersuchen. Um diese hochkomplexen Vorgänge einer Untersuchung zugänglich zu machen, müssen Annahmen getroffen werden. Teilsysteme, bei denen vermutet werden kann, dass sie zum betrachteten Effekt nichts beitragen, werden vernachlässigt. Andere Aspekte werden detailliert abgebildet, um deren Einfluss sichtbar zu machen. Dieses Vorgehen kann im Allgemeinen als Modellierung oder Modellbildung bezeichnet werden.

Bei sämtlichen verschiedenen Arten der wissenschaftlichen Untersuchung ist ein derartiges Verfahren vonnöten. Bei **experimenteller Behandlung** wird etwa ein Windkanal benutzt, um an einem maßstäblichen Replikat den Luftwiderstand zu untersuchen. Bei **theoretischen Modellen** werden eventuell Einflüsse der Schwerkraft vernachlässigt oder ein Vakuum vorausgesetzt. Auch bei der sogenannten dritten Säule der Wissenschaft, der **Simulationstechnik**, ist ein solches Vorgehen von entscheidender Bedeutung.

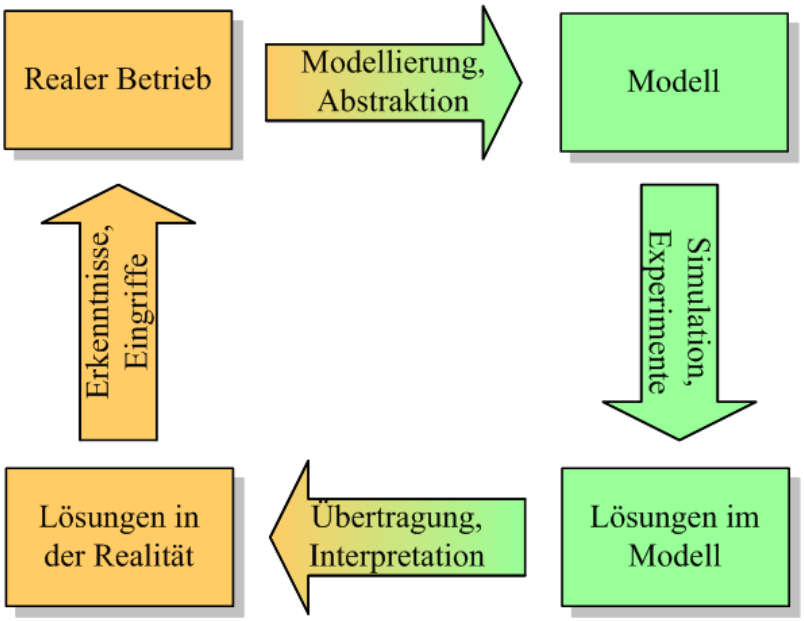


Abbildung 2.1: Modellierungskreislauf.

Um die Natur mittels Computersimulationen zu untersuchen, beschäftigt sich die Modellbildung mit der Frage, wie das Systemverhalten digital abgebildet werden kann. Wie in Abbildung 2.1 angedeutet, handelt es sich dabei um einen iterativen Prozess. Die Realität wird abstrahiert, mit diesem Modell können in weiterer Folge virtuelle Experimente durchgeführt werden. Die Ergebnisse der Simulation werden mit jenen von realen Prozessen verglichen und die Erkenntnisse daraus fließen in eine Überarbeitung des Modells.

Im Bereich der Modellbildung und Simulation gibt es verschiedene Möglichkeiten, die Wirklichkeit abzubilden. Eine naheliegende besteht darin, Gleichungen, die aus theoretischen Ansätzen gewonnen werden, am Computer abzubilden und zu lösen. Um das Vorgehen zu verdeutlichen und zwei verschiedene Ansätze vorzustellen, wird ein einfaches Beispiel herangezogen. Man betrachte eine Feder, auf der eine Masse ruht (siehe Abbildung 2.2). Die Masse wird aufgrund der Fallbeschleunigung eine Kraft auf die Feder ausüben, die dadurch gestaucht wird. Zuerst werden die Gleichungen, die das System beschreiben, angegeben, in diesem Fall das Federgesetz und das zweite und dritte Newton'sche Axiom:

$$F_F = c(l - x); \quad F = ma = mg; \quad F_F = F.$$

Aufgrund der eindimensionalen Betrachtungsweise wird auf die Vektorschreibweise verzichtet. Der Ausdruck F_F stellt die Federkraft dar, mit der Federkonstanten c und der ungedehnten Länge l. Die Variable ist hier die x-Koordinate. Die zweite Gleichung bezieht sich auf die Masse m. Die Beschleunigung a ist gleich der Fallbeschleunigung g, die in negative x-Richtung wirkt. Da der stationäre Zustand berechnet werden soll, werden andere Beschleunigungen null gesetzt.

2.0.1. Kausale Betrachtungsweise

Zusammengesetzt ergibt sich die algebraische Gleichung, die das Gesamtsystem beschreibt:

$$c(l - x) = mg$$

Damit ist allerdings die Modellierung noch nicht abgeschlossen. Die Parameter c, l und g werden als konstant und bekannt angenommen, nicht aber m und x. Die Frage, die sich stellt, ist, ob über die Gewichtskraft die Stauchung der Feder zu berechnen ist oder ob die Masse durch Messung des Federwegs bestimmt werden soll. Diese zwei Betrachtungen

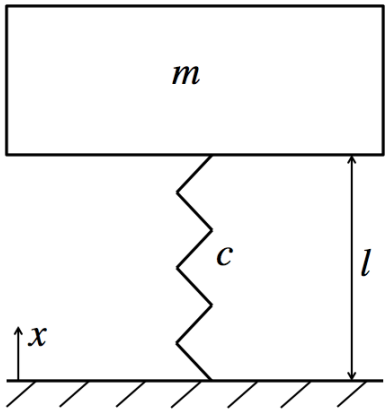


Abbildung 2.2: Darstellung der Komponenten eines Feder-Masse-Systems.

führen zu unterschiedlichen Formeln:

$$x := l - \frac{mg}{c}; \quad m := \frac{c(l-x)}{g}.$$

Je nach gesuchtem Ergebnis können diese Modelle berechnet werden. Der letzte Rechenschritt entspricht einer kausalen Betrachtungsweise. Mit dem Zuweisungsoperator (:=) wird angedeutet, dass der Variablen auf der linken Seite der Wert der Berechnung auf der rechten Seite zugewiesen wird. Bei diesem einfachen Beispiel wird deutlich, dass die Umformungen von Hand vorgenommen werden müssen, bevor das Modell (also die Gleichung) implementiert werden kann. Bei einem komplexen System wird das Zusammensetzen der Gleichungen ungleich aufwendiger. Die Berechnung von x oder m führt zu zwei unterschiedlichen Modellen und falls eine Komponente im Gesamtsystem ausgewechselt werden soll, müssen die Gleichungen neu erstellt werden.

2.0.2. Akausale Betrachtungsweise

Einen anderen Ansatz liefert hier die akausale objektorientierte Modellierung [1]. Die Idee besteht darin, dass Komponenten als Objekte abgebildet werden, die über Eigenschaften und Verhalten verfügen. Über standardisierte Schnittstellen können sie mit der Umgebung interagieren. Das Gesamtmodell kann dann durch Verbindung der Objekte zusammengesetzt werden. Die Komponenten sollen als sogenannte *Klassen* definiert werden können, was die Wiederverwendbarkeit ermöglicht. Die Bauteile selbst werden nur von ihren physikalischen Eigenschaften beschrieben, also im Beispiel der Feder $F_F = c(l - x)$. Die Modellierung endet damit, dass sämtliche modellrelevanten Gleichungen aufgeschrieben werden. Es sei hier auf die Bedeutung des Gleichheitszei-

chens hingewiesen, das tatsächlich eine Gleichheit beschreibt und keine Zuweisung (im Vergleich dazu angedeutet mit dem Zuweisungsoperator :=). Die Sortierung der Gleichungen und Festlegung der Reihenfolge der Berechnung ist dadurch nicht mehr Teil der Modellierung sondern Teil der Berechnung.

Zusätzlich kann eine grafische Benutzeroberfläche dazu dienen, das Erstellen eines Modells schnell und einfach zu gestalten. Die bereits erwähnten Klassen verfügen über grafische Repräsentationen. Bei der Modellierung werden Instanzen dieser Klassen generiert und miteinander verbunden. Das so erzielte Ergebnis ist anschaulich und leicht kommunizierbar. Dadurch können Fehler vermieden oder nachträglich leichter entdeckt werden. Zur Verdeutlichung ist das beschriebene Beispiel in Modelica umgesetzt worden, in Abbildung 2.3 ist das Modell in der grafischen Oberfläche zu sehen.

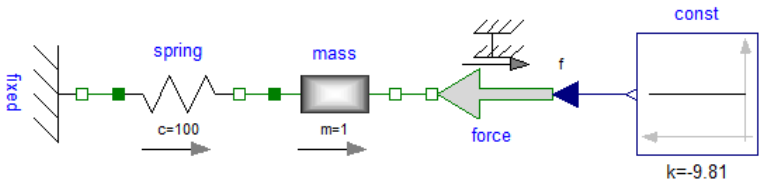


Abbildung 2.3: Modell des Feder-Masse-Systems in Modelica.

2.1. Grundlagen von Modelica

Modelica ist eine Sprache, mit der Systeme akausal beschrieben werden können. Die Idee stammt aus der Dissertation von Hilding Elmqvist [2] und wird von der non-profit Organisation *Modelica Association* [3] entwickelt. Der Aufbau von Modelica ist objektorientiert, Komponenten werden über Klassen beschrieben. Diese können verschiedene Eigenschaften und Verhalten haben, ausgedrückt durch Variablendefinitionen und Gleichungen. Sie können beliebig ineinander integriert werden. Dadurch ist ein hierarchischer Aufbau möglich. Komponenten werden lokal beschrieben, die Komplexität eines Modells resultiert daraus, dass mehrere Komponenten verbunden werden. Zusammengefasst haben Klassen folgende Eigenschaften:

- eine grafische Repräsentation,
- Spezifikationen von Konnektoren,
- Definitionen und Deklarationen von Parametern und Variablen und
- akausale Gleichungen zur Beschreibung des Verhaltens der Komponente.

Die grafische Repräsentation einer Komponente dient dazu, dass leicht zu erkennen ist, um welche Klasse es sich handelt. Dies erleichtert nicht nur die Modellbildung sondern auch das Einarbeiten in ein bereits bestehendes Modell. Bei Klassen, die von Grund auf entwickelt werden, empfiehlt es sich, auch den grafischen Layer zu berücksichtigen.

Zu den wichtigsten Definitionen der Modelica-Sprache zählen die Konnektoren. Über diese Schnittstellen werden verschiedene Komponenten miteinander verbunden, wodurch die Position im Gesamtsystem festgelegt wird. Modelica definiert verschiedene Typen von Konnektoren, die verschiedenen physikalischen Systemen entsprechen, zum Beispiel Strömungsmechanik, Elektrotechnik, et cetera. Jeder dieser Konnektortypen besitzt eine Fluss- (flow) und eine Potentialgröße (effort). In Tabelle 2.1 sind flow und effort für verschiedene Disziplinen aufgelistet.

Physikalisches Gebiet	**Potentialgröße**	**Flussgröße**
Mechanische Translation	Länge	Kraft
Mechanische Rotation	Winkel	Drehmoment
Elektrotechnik	Spannung	Stromstärke
Thermodynamik	Temperatur	Wärmestrom
Strömungsmechanik	Druck	Volumenstrom

Tabelle 2.1: Unterschiedliche Fluss- und Potentialgrößen der physikalischen Gebiete in Modelica.

Zwei oder mehr verbundene Konnektoren müssen den Kirchhoff'schen Gesetzen gehorchen. Am Beispiel der mechanisch-translatorischen Bibliothek bedeutet das, dass die Potentialgröße, also der Ort aller verbundenen Komponenten, gleich sein muss. Die Flussgröße Kraft muss aufsummiert null ergeben. Dies ist einleuchtend, wenn man das Beispiel von mehreren Starrkörpern, an denen Kräfte angreifen, heranzieht. Der Ortsvektor aller Körper muss im Verbindungspunkt derselbe sein, während die Flussgröße dem Kräftegleichgewicht gehorchen muss.

Bei der Deklaration der Parameter und Variablen wird ebenso das Prinzip der physikalisch korrekten Abbildung weiterverfolgt. Wie auch bei den Konnektoren angewandt, gibt es eine Umsetzung von physikalischen Einheiten in Modelica. Dadurch kann einer Variable bei der Deklaration eine Einheit mitgegeben werden. Auf eine fehlende Konsistenz der Einheiten wird beim Kompilieren hingewiesen.

Die Definition der Gleichungen bestimmt das Verhalten einer Klasse. Wie bereits erwähnt, werden Gleichungen nicht als Zuweisungen verstanden sondern als Äquivalenzen

im wahrsten Sinne des Wortes. Es ist nicht notwendig zu wissen, welche Variable aus welcher Gleichung zu berechnen ist. Auch die vertikale Anordnung der Gleichungen ist nicht von Bedeutung. Dies wird dadurch möglich, dass die Gleichungen erst beim Berechnen der Simulation sortiert werden. Es können nur lokale Variablen der Klasse verwendet werden, was die Allgemeingültigkeit der Komponente gewährleistet. Am Beispiel des Feder-Masse-Schwingers wird nicht in den jeweiligen Komponenten festgelegt, welche Größen vorgegeben und welche gesucht werden. Erst das zusammengesetzte System ergibt das vollständige Verhalten.

Wie eingangs erwähnt ist Modelica eine Sprache und kein Simulator. Es gibt eine Vielzahl an kommerzieller oder frei verfügbarer Software, die den Modelica-Standard nutzen. Eines dieser Programme ist das gleichzeitig mit Modelica entstandene Tool Dymola. Durch die lange Entwicklungszeit und robusten Solver [4] eignet es sich gut für den Einsatz bei der Simulation von mechanischen Systemen.

2.2. Die Modelica Standard Library

Die Modelica Standard Library stellt eine umfangreiche Bibliothek an Komponenten dar, die mit dem Modelica-Standard erstellt wurden. Sie ist frei zugänglich und kann daher von jedem Modelica-fähigen Simulator benutzt werden. Die einzelnen Bibliotheken decken einen großen Teil an physikalischen Disziplinen ab, wie Elektrotechnik, Strömungsmechanik oder Thermodynamik. Die für diese Studie relevanten Klassen werden genauer besprochen, die Struktur der Modelica Standard Library ist in Abbildung 2.4 zu sehen.

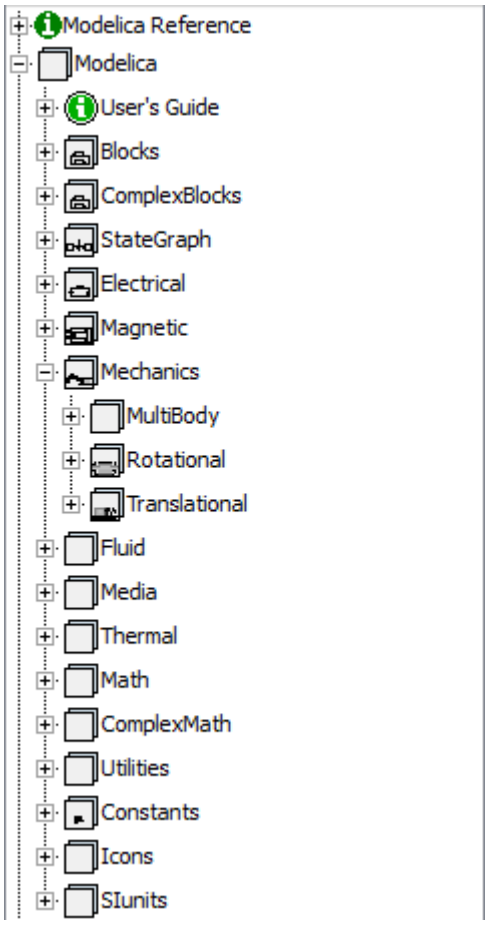


Abbildung 2.4: Modelica Standard Library

Die allgemeine Blocks Library beinhaltet Komponenten, die für die Manipulation von kausalen Signalen zuständig sind. Darunter befinden sich mathematische Operationen wie

Gain, Summation oder Multiplexer sowie Quellen für Signale, etwa Konstanten, Winkelfunktionen oder Rampen. Es gibt Elemente, die für die Manipulation von diskreten Informationen zuständig sind, wie Delays und Holds. Speziell für das Gebiet der Regelungstechnik gibt es den Bereich der kontinuierlichen Komponenten, mit denen Übertragungsfunktionen, Integratoren und dergleichen implementiert werden können. Bei den Konnektoren handelt es sich ausschließlich um gerichtete Größen wie Fließkommazahlen, Integer oder Bool'sche Operatoren.

Der für die Modellierung mechanischer Komponenten wichtigste Teil der Modelica Standard Library ist die Mechanics Bibliothek. Die drei untergeordneten Bereiche sind mechanisch-translatorisch, mechanisch-rotatorisch und MultiBody. Die Komponenten der ersten zwei Untergruppen beschreiben eindimensionale Systeme. Die MultiBody-Library [5] bildet hingegen die völlige Bandbreite von mechanischen Systemen ab, indem jeder Konnektor sechs Freiheitsgrade besitzt. Der Nachteil daran ist, dass eine große Anzahl an Gleichungen gelöst werden muss, was die Performance dementsprechend verschlechtert. Bei komplexen Problemstellungen ist der Einsatz der MultiBody Library häufig unumgänglich. Einzelne Komponenten der eindimensionalen Bibliotheken wie Federn oder Dämpfer können zusätzlich eingesetzt werden. Modelle werden in der dreidimensionalen Betrachtungsweise durch zwei Hauptgruppen von Komponenten aufgebaut: Parts und Joints. Parts sind im Prinzip mechanische Bauteile wie Massen oder auch komplexe Teile wie Räder. Joints sind Gelenke, die den Zweck haben, gewisse Freiheitsgrade zu sperren. Komplettiert wird die Bibliothek durch Komponenten für Kräfte, Sensoren zum Messen physikalischer Größen und dem speziellen World Block, der für den Koordinatenursprung sowie Richtung und Größe der Fallbeschleunigung zuständig ist. Eine genauere Diskussion der MultiBody Konnektoren erfolgt in Abschnitt 5.

Neben der Modelica Standard Library gibt es Bibliotheken, die von Drittanbietern entwickelt wurden und zum Teil nur kommerziell erhältlich sind. Ein für das betrachtete Beispiel relevantes Paket ist die Powertrain-Library [6], die von Dassault Systèmes [7] vertrieben wird. Der Großteil der enthaltenen Komponenten ist sehr spezialisiert, einzelne können jedoch für die Modellierung hilfreich sein. Hervorzuheben sind hier Getriebeelemente im Allgemeinen und Differentiale im Speziellen. Auch Radkomponenten können für simple Simulationen herangezogen werden.

3. Grundlagen der Co-Simulation

In der Simulationstechnik ist die Beschreibung von Problemen und deren effiziente Lösung das zentrale Thema. Flexibilität in der Beschreibung kann sowohl die Effizienz erhöhen, aber auch zur Abbildung von Phänomenen, die bis dato nicht erfassbar sind, beitragen. Um diese Anpassbarkeit zu erreichen, kann der Ansatz der Cooperative Simulation (Co-Simulation) herangezogen werden. Darunter versteht man, dass mehrere Simulatoren zeitgleich an ein und demselben Problem arbeiten. Jedes der Programme kann eigene Modellbeschreibungen und Lösungsalgorithmen verwenden.

Die Natur abzubilden, um sie einer Berechnung zugänglich zu machen, ist kein leichtes Unterfangen. Dies kann auf ganz unterschiedliche Arten passieren, manche sind einfach zu erfassen und intuitiv, andere bilden die Realität besser ab. Verschiedene technische und naturwissenschaftliche Disziplinen haben daher unterschiedliche Modellierungsansätze, die sich zum Teil gravierend unterscheiden. In der Regelungstechnik werden Modelle meist signalflussbasiert aufgebaut, in der Epidemiologie werden häufig Agenten eingesetzt. Die Möglichkeit, diese Vielfalt bei der Bearbeitung komplexer Systeme beizubehalten, ist daher ein wichtiger Fortschritt. Zusätzlich wird dadurch bei kollaborativer Zusammenarbeit verschiedener Spezialisten ermöglicht, dass die Werkzeuge benutzt werden können, mit denen die jeweiligen Personen vertraut sind. In dieselbe Kerbe schlägt auch die Wahl des Lösungsalgorithmus (Solver). Die Funktionsweise der Solver spielt eng mit der Beschreibung des Modells zusammen. Durch verschiedene Modellierungsansätze entstehen auch unterschiedliche numerische Problemstellungen, die gelöst werden müssen. Neben den Vorteilen, die spezialisierte Software bei der Modellbildung bietet, kann die Co-Simulation sogar noch mehr leisten. Durch die Verbindung von mehreren Programmen lassen sich Probleme angehen, die durch einzelne Softwarelösungen schwer oder sogar gar nicht bearbeitbar wären. Als Beispiele können hier hybride Modelle – die Verbindung von diskreter und kontinuierlicher Betrachtungsweise, etwa in Bezug auf event handling – oder steife Systeme erwähnt werden.

Der Ansatz der Co-Simulation verlangt es, dass verschiedene Programme während der Laufzeit miteinander Informationen austauschen. Diese Daten dienen den Simulatoren als Eingangswerte. Abbildung 3.1 zeigt einen schematischen Aufbau der Systemarchitektur. Jede Co-Simulation benötigt eine Kontrollinstanz, welche die gesamte Simulation steuert. Dieser Backbone kann selbst ein Simulator sein, oder einzig und allein für den Datenaustausch zuständig sein. Mit ihm stehen die anderen Simulationsprogramme in Verbindung und tauschen zu bestimmten Zeitpunkten Informationen aus. Dieser

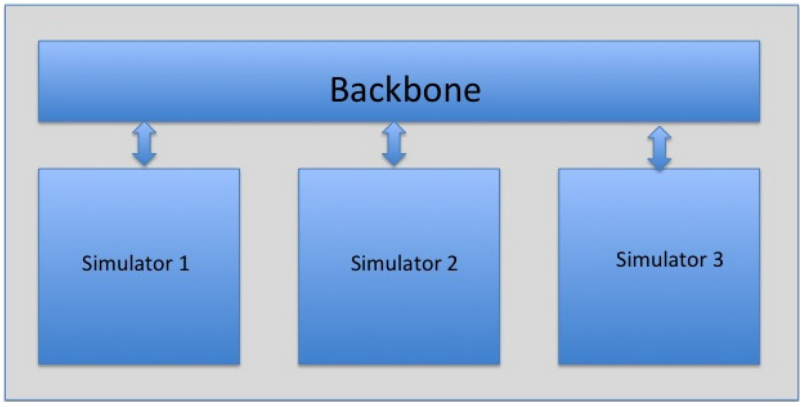


Abbildung 3.1: Schematische Darstellung des Aufbaus einer Co-Simulation.

Datenaustausch kann nach einem fixen Zeitintervall geschehen, aber auch dynamische Zeitspannen sind möglich. Die Solver der verwendeten Programme können zwischen den Intervallen unabhängig voneinander arbeiten. Die einzige Ausnahme besteht durch den Datenaustausch: Zu diesen Zeitpunkten müssen sämtliche Simulationen einen Zeitschritt haben, damit die zu kommunizierenden Daten berechnet werden können.

Zur Umsetzung einer Co-Simulation gibt es mittlerweile eine Vielzahl an kommerziellen aber auch frei zugänglichen Softwarelösungen. Für die Auswahl an Programmen ist das wichtigste Kriterium die Kompatibilität mit den Simulatoren, die eingesetzt werden sollen. Auch andere Faktoren wie Effizienz, Erweiterbarkeit oder Anschaffungskosten müssen berücksichtigt werden. Im Folgenden werden zwei Möglichkeiten vorgestellt, Co-Simulation zu betreiben.

3.1. Co-Simulation mittels Functional Mockup Interface

Eine der Optionen, eine Co-Simulation aufzusetzen, ist mit Hilfe des Functional Mockup Interface (FMI), siehe [8]. Dieser Standard wurde im Zuge des MODELISAR-Projekts entwickelt, das im Dezember 2011 endete. Das Ziel war die Entwicklung von Electronic Control Units, um die Steuerung von Motoren und Getrieben von Kraftfahrzeugen zu verbessern. Dies führte zum Ansatz des Modell-Austauschs, der über ein standardisiertes Format einfach einsetzbar ist. Der FMI-Standard wird mittlerweile als Projekt der Modelica Association weiterentwickelt, die Version 2.0 soll demnächst veröffentlicht werden [10].

Das Functional Mockup Interface vereinfacht das kollaborative Zusammenarbeiten an ein und demselben Projekt immens. Verschiedene Teile können unabhängig voneinander entwickelt und einfach aneinander gekoppelt werden. Diese Koppelung ist auf Anwen-

derebene einfach nutzbar.

Um Co-Simulation betreiben zu können, ist es notwendig, dass verschiedene Simulationsprogramme miteinander kommunizieren können. Dieser Datenaustausch erweist sich allerdings als schwierig, da die unterschiedlichen Softwarelösungen andere Programmiersprachen oder Standards benutzen, die im Normalfall nicht kompatibel sind. Einige Programme benutzen einfache firmeneigene Interfaces, um eine abweichende Modellbeschreibung zu ermöglichen. Zum Beispiel ist es in Simulink [11] möglich, Maschinencode (unter anderem MATLAB, C und C++) als S-Funktion (system function) zu integrieren. Diese Schnittstellen erreichen allerdings sehr schnell ihre Grenzen, da die Kompatibilität zu anderen Programmen eingeschränkt ist. Eine detaillierte Aufstellung der speziellen Eigenschaften ist in [8] nachzulesen.

Das Functional Mockup Interface kann hier Abhilfe schaffen. Durch die einheitliche Darstellung der Modelle können unterschiedliche Softwarelösungen herangezogen werden, um ein System zu modellieren. Im Gegensatz zur Struktur in Abbildung 3.1 gibt es keinen dezidierten Backbone, der nur für die Kommunikation eingerichtet ist. Stattdessen wird eine Functional Mockup Unit (FMU) in einen Simulator importiert. Diese eigenständige Einheit kommuniziert mit dem restlichen Modell über gerichtete Ein- und Ausgänge. Im FMI-Standard gibt es zwei Möglichkeiten, ein Modell als FMU an ein anderes Programm zu übergeben. Die erste ist per Model Exchange das gesamte Modell zu exportieren. Die zweite Möglichkeit besteht darin, ein alleinstehendes, kompiliertes Modell mitsamt Solver für Co-Simulation auszugeben.

Model Exchange

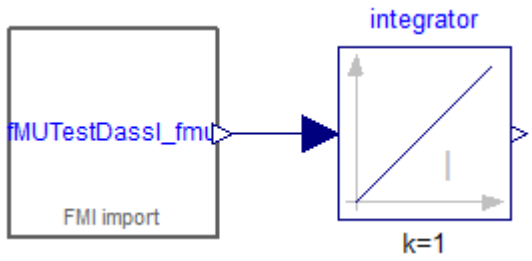
Bei FMUs, die mit der Option Model Exchange erstellt wurden, kann jedes andere Programm, das über eine FMI-Schnittstelle verfügt, dieses Modell einbinden. Das Ändern von Parametern zur genaueren Anpassung, aber auch für Parameterstudien, ist durch den Model Exchange möglich. Das Gesamtsystem wird global mit dem Algorithmus des Wirtprogramms gelöst, auch das importierte Modell. Dies hat den Vorteil, dass es keinen Unterschied zwischen Zeitschritt und Datenaustauschrate gibt, da bei jedem Schritt des globalen Solvers auch Daten ausgetauscht werden. Es ist dadurch nicht möglich, mehrere Algorithmen für spezielle Probleme zu nutzen. Um dieser Tatsache gerecht zu werden, existiert die Möglichkeit mittels FMI auch Co-Simulation zu betreiben.

Co-Simulation

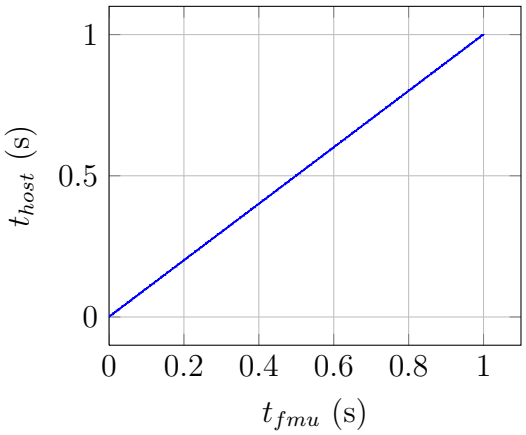
Die zweite Methode, FMI-Schnittstellen zu nutzen, ist mit der Option Co-Simulation. Auf diese Art werden mehr als nur Informationen über das Modell geteilt. Das bedeutet, dass der Solver mit der FMU exportiert und bei der Simulation auf das Teilmodell angewandt wird. Eine auf diese Art importierte FMU arbeitet daher nicht mit dem Gleichungslöser des Wirtprogramms, es wird also tatsächlich Co-Simulation betrieben. Im Anwendungsfall ist es natürlich interessant, wie über FMI die Schrittweiten und Datenaustauschraten bestimmt werden.

3.1.1. Untersuchungen zur Schrittweitenwahl

Zu diesem Zweck werden zwei Softwareprodukte mit FMI-Anbindung herangezogen und unterschiedliche Kombinationen untersucht. Der erste Versuch beschränkt sich auf den Einsatz von Dymola. Darin wird eine FMU mit der Option Co-Simulation erstellt und in ein anderes Dymola-Modell integriert, siehe Abbildung 3.2 (a). Die Einstellungsmöglichkeiten von FMUs in Dymola sind begrenzt. Es ist nicht möglich, eine unabhängige Datenaustauschrate zu definieren, bei jedem Schritt des globalen Solvers werden Daten ausgetauscht.



(a) Modell in Dymola mit der importierten FMU.

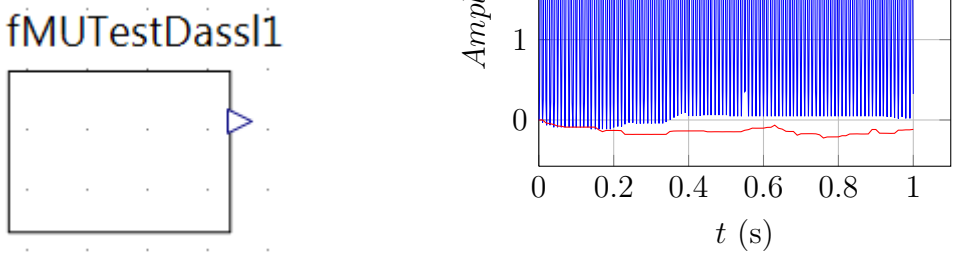


(b) Zeitschritte von Dymola als Host aufgetragen über jenen der FMU.

Abbildung 3.2: Modell und Ergebnisse der Simulation mit Dymola unter Einbindung einer FMU.

Der Standardsolver von Dymola (DASSL) arbeitet mit variabler Schrittweite, im Normalfall mit vielen Zeitschritten. Dementsprechend überdeckt der häufige Datenaustausch den Einfluss des Gastsolvers massiv. Dies ist in Abbildung 3.2 (b) daran zu erkennen,

dass die Anzahl der Schrittweiten von FMU und Host nahezu gleich sind. Die Vorgabe, dass der Datenaustausch dem Wirtsystem folgt, hat als Konsequenz, dass auch die FMU zum selben Zeitpunkt einen Zeitschritt haben muss. Durch die hohe Frequenz der Datenpunkte ist es nicht zu erzielen, dass die FMU mehr Zeitschritte absolviert. Auch beim Übergang auf einen Solver mit fixer Schrittweite und dem manuellen Reduzieren der Anzahl der Rechenschritte folgt die FMU mit. Daher kann auch kein Einfluss des exportierten Solvers der FMU festgestellt werden. Es ist zu sehen, dass es zwischen den Zeitschritten des Gastprogramms und der FMU kaum Unterschiede gibt. Die FMU macht lediglich sechs Schritte mehr.



(a) Modell in SimulationX mit der importierten FMU.

(b) Ausgang der FMU: Blau: Kommunikationsintervall $1ms$, rot: $10ms$.

Abbildung 3.3: Modell und Ergebnisse der Simulation mit SimulationX unter Einbindung einer FMU.

Als zweiter Versuch wird das Programm SimulationX 3.5 der ITI GmbH [9] herangezogen. Das Besondere daran ist, dass in SimulationX eine Möglichkeit besteht, für jede FMU eine Datenaustauschrate festzulegen. Dies hat den Vorteil, dass eine FMU mit niedrigen zeitlichen Gradienten den Datenaustausch reduzieren kann, das Wirtsystem arbeitet inzwischen mit dem letzten Wert. Der Nachteil besteht darin, dass es nicht möglich ist, eine variable Datenaustauschrate festzulegen, wie das bei Dymola der Fall ist. In Abbildung 3.3 (a) ist das Modell zu sehen, das nur aus der importierten FMU besteht. In Abbildung 3.3 (b) ist der Ausgang der FMU aufgetragen, in rot bei einem Kommunikationsintervall von $10ms$ und in blau mit $1ms$. Es ist zu erkennen, dass der Datenaustausch einen großen Einfluss auf die Qualität des Ergebnisses besitzt. Bei der geringen Austauschrate kann der hochfrequente Ausgang nicht mehr aufgelöst werden.

Ein trade-off gegenüber der Rechenzeit ist unausweichlich. Durch Variation der Solver, Schrittweiten des Gast- und Wirtsystems als auch Datenaustauschraten wurde versucht, die internen Vorgänge der FMU zu beleuchten. Dazu wurden die Zeitschritte des Gastsystems aufgenommen und mit denen des Wirtsystems verglichen. Auch hier zeigt sich, dass die Wahl des Solvers der FMU keinen Einfluss auf das Ergebnis hat. Dies kann darauf zurückzuführen sein, dass ein simples Beispiel verwendet wurde (Integration eines Sinussignals) und dass dadurch verschiedene Solver quasi gleiche Ergebnisse liefern. Dieselbe Situation zeigt sich bei der Wahl der Schrittweite beim Export der FMU.

3.2. Co-Simulation mit BCVTB

Eine weitere Möglichkeit, Co-Simulation zu betreiben, ist mittels spezieller Software, die als Backbone fungiert. Ein wichtiger Punkt ist, dass mehreren gleichzeitig laufenden Simulationsprogrammen die Möglichkeit gegeben wird, zur Laufzeit Daten untereinander auszutauschen. Eines dieser Programme ist das Building Controls Virtual Test Bed (BCVTB) [12], entwickelt von Lawrence Berkeley National Laboratory an der University of California. BCVTB basiert auf Ptolemy II, einer modularen Softwareumgebung für die Simulation heterogener Systeme.

Ptolemy II [13] ist ein open-source Rahmenwerk und wird an der University of California in Berkeley entwickelt. Heterogen in diesem Sinn bedeutet, dass Systeme aus vielen verschiedenartig beschaffenen Subsystemen bestehen. Diese verschiedenen Charakteristiken führen dazu, dass Ptolemy einen hierarchischen Ansatz wählt, der hohe Flexibilität in der Modellierung lässt. So ist es zum Beispiel möglich, Subsysteme mit eigenen lokalen Kontrollstrukturen zu versehen, also auch mit verschiedenen Solvern.

3.2.1. Aufbau von BCVTB

Die Komponenten eines Modells werden in BCVTB *Actors* genannt. Sie stellen Einheiten dar, die eine bestimmte Funktion haben wie eine Datenquelle, ein Differenzierer oder ein Multiplexer. In der grafischen Oberfläche werden sie verbunden, was den gerichteten Signalstrom zwischen den Actors festlegt. Der für die Co-Simulation entwickelte *Simulator Actor* soll besonders erwähnt werden. Er steht stellvertretend für ein Simulationsprogramm, das an BCVTB angebunden wird. Neben systemrelevanten Einstellungen werden hier die Informationen für das Anstoßen der Simulatoren übergeben. Der Actor ist damit die Schnittstelle für die Kommunikation. Ein enges Zusammenspiel zwischen den Parametern des Simulator Actors und den angebundenen Programmen ist notwen-

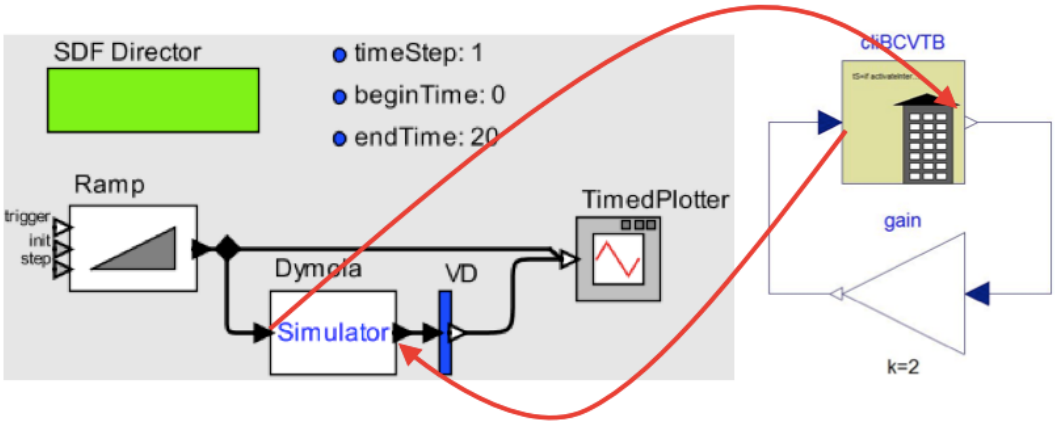


Abbildung 3.4: Darstellung der Funktionsweise von BCVTB. Links: Oberfläche von BCVTB, rechts: Dymola-Modell mit BCVTB-Block.

dig, damit die Berechnung stattfinden kann. In Abbildung 3.4 sind exemplarisch drei Actors zu sehen: eine Rampe als Quelle des Signals, ein Simulator Actor und ein Plotter zur Ausgabe. Beim Datenaustausch während der Simulation werden die Eingangssignale des Simulator-Actors an Dymola geschickt und scheinen dort als Output aus dem BCVTB-Block auf. Dessen Inputs werden umgekehrt zurück an BCVTB übergeben.

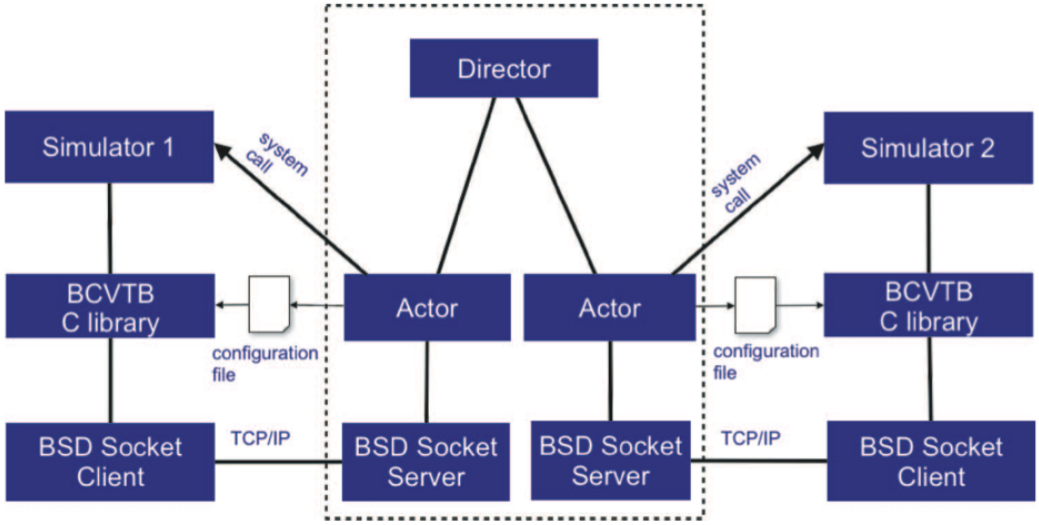


Abbildung 3.5: Darstellung der Architektur von BCVTB aus [14].

In BCVTB wird die Simulation über *Directors* gesteuert, sie legen Simulationsparameter wie Schrittweite und Solver fest. Von besonderem Interesse ist hier die *Synchronous Data Flow* (SDF) Domäne, die durch den gleichnamigen Director gesteuert wird. Unter

SDF versteht man, dass ein übergeordneter periodischer Takt festlegt, wann die Daten zwischen den Actors ausgetauscht werden. Genauer gesagt warten sämtliche Actors, bis sie den Auftrag bekommen zu „feuern“, und bearbeiten dann den Zeitschritt. Bei der nächsten Iteration sind diese Daten für den nachgelagerten Block als Eingang vorhanden. Modelle in BCVTB können hierarchisch angeordnet werden. Diese Submodelle können für sich wieder einen Director besitzen. Das Besondere daran ist, dass dadurch verschiedene Kommunikationsintervalle in den Subsystemem definiert werden können. Diese Möglichkeit eignet sich im Speziellen hervorragend für den Einsatz mit Co-Simulation, da so zum Beispiel steife Systeme behandelt werden können.

Die Kommunikation zwischen BCVTB und den Clients wird über Berkley Software Distribution (BSD) sockets [14] durchgeführt. Diese Art der Verbindung verwendet im Prinzip ein TCP/IP-Protokoll und erlaubt Kommunikation zwischen laufenden Prozessen. Zur Veranschaulichung der Architektur von BCVTB wird ein Bild aus [14] herangezogen, das in Abbildung 3.5 zu sehen ist.

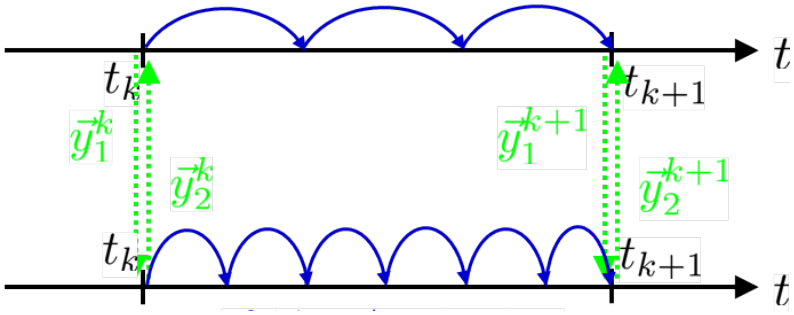


Abbildung 3.6: Funktionsprinzip der Co-Simulation nach dem Jakobi-Typ.

Der Director steuert die gesamte Simulation, vor allem die Simulator-Actors, die darunter angeordnet sind. Sie rufen die jeweiligen Simulatoren auf, über C-Bibliotheken stellen die BSD-Sockets die Kommunikationsschnittstelle dar. Der Bereich innerhalb der strichlierten Linie ist Teil des Kernprogramms BCVTB.

Die Art, wie Daten ausgetauscht werden, entspricht dem Jakobi-Typ. Darunter versteht man, dass die Simulatoren quasi-parallel arbeiten und die Informationen erst am Ende jedes Zeitschritts ausgetauscht werden. Zwischen diesen Punkten müssen die Programme ihre jeweiligen Eingangsdaten aus den letzten Werten extrapolieren. Eine Darstellung dieses Prinzips ist in Abbildung 3.6 zu sehen. Die grünen Pfeile zeigen die Kommunikation der Daten. Nur zu fixen Zeitpunkten können Informationen ausgetauscht werden. Die blauen Pfeile stellen die Zeitschritte der zwei Simulatoren dar. Die Zeitschrittweiten der Prozesse müssen nicht ident sein, zum Kommunikationszeitpunkt müs-

sen aber beide einen Berechnungspunkt setzen.

3.2.2. Überblick über Simulatoren

Die Möglichkeiten, mit BCVTB Co-Simulation zu betreiben, sind dadurch beschränkt, welche Programme mit BCVTB kommunizieren können. Folgend wird ein kurzer Überblick über die wichtigsten Tools gegeben, die dieses Kriterium erfüllen und für die Anwendung der Simulation von mechanischen Systemen einsetzbar sind. Eines der mächtigsten und vielfältigsten Werkzeuge ist MATLAB. Der überwiegende Einsatz ist im Bereich von numerischen Berechnungen, es ist aber auch symbolisches Rechnen möglich. MATLAB eignet sich unter anderem für das Einlesen und Aufbereiten großer Datenmengen. Dadurch kann die MATLAB-Anbindung an BCVTB zum Beispiel dafür genutzt werden, datenbasierte Modelle effizient umzusetzen. Auch die Möglichkeit, symbolische Operationen durchzuführen, ist bei dieser Anwendung von Vorteil.

Vor allem für regelungstechnische Anwendungen eignet sich Simulink sehr gut. So können zum Beispiel Fahrzeugregelungen oder auch die Regelung thermischer Systeme leicht implementiert werden. Aufbauend darauf soll Stateflow nicht unerwähnt bleiben. Stateflow nutzt eine grafische Oberfläche, um Statecharts abzubilden. Dies kann etwa bei einer Schaltlogik genutzt werden, die aufgrund verschiedener Informationen den passenden Gang auswählt und die dafür notwendigen Informationen ausgibt. Mit Stateflow ist also eine elegante Erweiterung auf diskrete Vorgänge in kontinuierlichen Systemen möglich.

Der wohl wichtigste Partner in der Simulation mechanischer Systeme ist Dymola. Aufbauend auf der Modelica Standard Library lassen sich dreidimensionale mechanische Starrkörpersysteme in der MultiBody-Domäne leicht umsetzen. Modelica ermöglicht es, Modellbibliotheken selbst zu entwickeln oder von Drittanbietern zu nutzen. Es gibt eine Handvoll weiterer Software, deren Anbindung BCVTB derzeit unterstützt und deren Anwendung für die betrachteten Problemstellungen allerdings wenig Relevanz hat.

3.2.3. Anbindung von Dymola und MATLAB an BCVTB

Die Anbindung der verschiedenen Programme gestaltet sich unterschiedlich komplex, da diese stark von den jeweiligen Simulatoren abhängt. Die notwendigen Funktionen werden mit der Installation von BCVTB mitgeliefert und befinden sich im Installationsordner. Um MATLAB einzubeziehen, muss zuerst auf diesen Ordner verwiesen werden. Die drei Funktionen sind für die Erstellung der Verbindung mittels BSD-sockets, den Datenaustausch und für das Beenden der Verbindung verantwortlich. Angestoßen wird die

Simulation standardmäßig durch das Ausführen einer MATLAB-Instanz über BCVTB selbst. Dies entspricht im Prinzip nichts anderem als dem Aufrufen des Programms aus der command line. So können auch zusätzliche Informationen wie die Datenaustauschrate übermittelt werden.

Die Anbindung von Dymola ist etwas umständlicher. Um dieselben Funktionen, die für die Kommunikation mit BCVTB zuständig sind, aufzurufen, muss auf C-Funktionen zurückgegriffen werden. Auch diese werden bei der Installation bereitgestellt, müssen allerdings in das Arbeitsverzeichnis von Dymola überführt werden. Dies geschieht im Normalfall dadurch, dass BCVTB selbst den Kopiervorgang übernimmt, und zwar beim Aufruf des Simulator Actors. Dieser ruft ein Batch-Script auf, das die Dateien kopiert. Danach wird ein Modelica-Script-File gestartet, das den eigentlichen Simulationsstart anstößt. Auf der Seite von Dymola gibt es einen eigenen Block, der die Kommunikation übernimmt. Dieser wird mit der Buildings Library [15] mitgeliefert, die ursprünglich zur thermischen Simulation von Gebäuden dient. Der BCVTB-Block (Abbildung 3.4 rechts) sorgt, ähnlich wie in MATLAB, für die Kommunikation über BSD-sockets.

3.2.4. Kontrollinstanz

Durch die Anwendung von Co-Simulation entsteht aufgrund der verschiedenen Programme eine große Anzahl an Dateien. Daher ist es wichtig, eine Struktur zu verfolgen, die Übersicht gewährleistet und die Nachvollziehbarkeit zu späteren Zeitpunkten unterstützt. Weitergehend sinnvoll ist der Aufbau einer Kontrollinstanz, die der gesamten Simulation vorsteht. Dies wird dadurch notwendig, dass es nicht ohne Weiteres möglich ist, in BCVTB sämtliche simulationsrelevanten Parameter zentral vorzugeben. Daten, wie zum Beispiel die Zeitschrittweite, müssen in jedem Subprogramm getrennt eingestellt werden. Dies ist für eine effiziente Nutzung eines Modells nicht von Vorteil, da oft viele Simulationen mit unterschiedlichen Parametern getestet werden müssen, um ein befriedigendes Ergebnis zu erzielen. Weitergehend ist es eventuell von Interesse, Eigenschaften des Modells zu verändern, um optimale Ergebnisse zu erzielen, zum Beispiel in Form einer Parameteroptimierung mit genetischen Algorithmen. In Abbildung 3.7 ist das Schema aus Abbildung 3.1 um die Kontrollinstanz erweitert worden. Über externe Dateien werden sowohl simulationsrelevante Informationen übergeben als auch die Ergebnisse gespeichert.

Um dies zu erreichen, wurde eine Kontrollinstanz in MATLAB erarbeitet, die den Vorgaben genügt. Die Wahl, MATLAB zu benutzen, beruht auf zwei Tatsachen. Erstens ist es unkompliziert möglich, Daten in externen Dateien zu speichern, und zweitens können

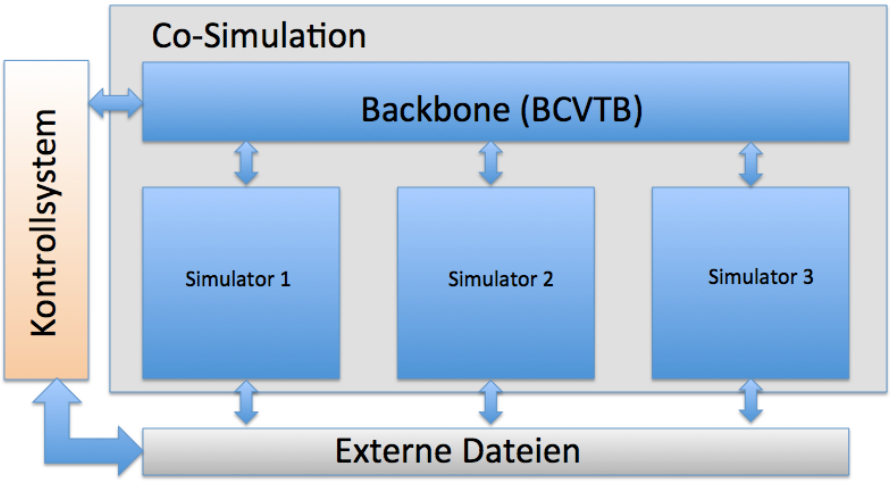


Abbildung 3.7: Schematische Darstellung des Aufbaus einer Co-Simulation mit Kontrollinstanz.

über Funktionsaufrufe sehr einfach Argumente an das System zum Ausführen übergeben werden. In Anhang A ist der Sourcecode des Kontrollskripts zu sehen. Zuerst werden die Simulationsparameter definiert, die wichtigsten sind Beginn- und Endzeitpunkt der Simulation und die Zeitschrittweite. Zusätzlich können weitere Variablen definiert werden, falls ein Wert für eine Parameteroptimierung angepasst werden soll. Auch Angaben über die Arbeitsverzeichnisse der einzelnen Programme müssen getroffen werden, damit die richtigen Dateien für die Simulation aufgerufen werden können. Eine Ordnerstruktur zu implementieren ist kein Problem. Zu beachten ist jedenfalls, dass jedem Subprogramm ein eigenes Arbeitsverzeichnis zur Verfügung gestellt werden soll, damit es zu keinem Konflikt bezüglich Dateinamen kommt. Vor allem bei den Konfigurationsdaten für die Socket-Kommunikation kommt es sonst zu Problemen. Die Simulationsparameter werden als eigene Dateien gespeichert, um allen Clients Zugriff auf die Werte zu gestatten. Der Dateityp kann im Prinzip frei gewählt werden, es bietet sich wegen der einfachen Schreib- und Lesbarkeit allerdings das Format Comma-Seperated Value (csv) an. Zum Start der Simulation wird der Aufruf von BCVTB mit sämtlichen Argumenten an das System übergeben. Die Argumente können praktischerweise über die command line an BCVTB übergeben werden und dort als normale Konstante gehandhabt werden. Damit müssen in BCVTB selbst die Simulationsparameter nicht aus den Dateien ausgelesen werden. Der Aufruf von BCVTB muss mit dem Argument „run“ durchgeführt werden, damit die Simulation auch tatsächlich gestartet wird. Über den Simulationsaufruf kann eine Schleife gelegt werden, die bei Beendigung eines Simulationsdurchgangs den zu

variierenden Wert anpasst und die Simulation erneut aufruft. Der Quellcode des Kontrollscripts ohne Optimierung ist in Anhang A nachzulesen.

Sobald BCVTB aufgerufen ist, kann die Simulation starten. Die jeweiligen Simulator Actors stoßen die Subprogramme an (siehe Abbildung 3.5). Die Simulationsparameter können nicht von BCVTB übergeben werden, daher wird auf die csv-Dateien zurückgegriffen. Bei der Anbindung von MATLAB gestaltet sich die Sache einfach. Die Parameter können aus den Files ausgelesen werden bevor die Berechnung startet. Einzig der Verzeichnispfad der Dateien muss bekannt sein. Dieser kann vorab fixiert und dadurch in die MATLAB-Datei geschrieben werden. Alternativ kann im Kontrollskript eine Umgebungsvariable, die den Pfad beinhaltet, erstellt werden. Diese kann dann von MATLAB ausgelesen werden. In Dymola gestaltet sich die Parameterübergabe schwieriger. Es ist nicht möglich, die Simulationsparameter direkt an Dymola zu übergeben, nachdem es bereits von BCVTB gestartet wurde. Das Auslesen wäre bereits Teil der Simulation, was zu einer Fehlermeldung führt. Wie bereits erwähnt, wird Dymola aber nicht direkt von BCVTB gestartet sondern indirekt über ein Modelica-Skript, siehe Listing 1. Darin ist es möglich, die Start- und Endzeit auszulesen und an Dymola zu übergeben. Die Zeitschrittweite, oder eigentlich in diesem Fall die Datenaustauschrate, muss dem BCVTB-Block im Dymola-Modell selbst übergeben werden. Dies geschieht, indem die Dateien im Modell selbst erneut ausgelesen werden. Auch weitere, zum Beispiel zu optimierende Parameter können auf diese Weise in das Dymola-Submodell eingespeist werden.

```
1 openModel("model.mo");
2 beginTimeString := Modelica.Utilities.Streams.readFile("../beginTime.csv")
3 beginTimeReal := Modelica.Utilities.Strings.scanReal(beginTimeString[1])
4 endTimeString := Modelica.Utilities.Streams.readFile("../endTime.csv")
5 endTimeReal := Modelica.Utilities.Strings.scanReal(endTimeString[1])
6 simulateModel("model", startTime=beginTimeReal, stopTime=endTimeReal);
7 exit();
```

Listing 1: Modelica-Skript zum Auslesen der Simulationsparameter und Start der Simulation.

3.3. Parallelisierung mit BCVTB

Einer der Vorteile, der sich durch Co-Simulation mit BCVTB bietet, ist, dass Programme eigenständig an ihren Teilmodellen arbeiten können. Im Gegensatz zu FMI kann auf diese Weise die Simulation auf verschiedene Rechner verteilt werden, wodurch die Rechenzeit reduziert werden kann. Dies geschieht dadurch, dass die Subprogramme, die

durch BCVTB angestoßen werden, sich nicht lokal auf demselben Betriebssystem befinden. Es sind Szenarien denkbar, wobei mehrere Server zur Berechnung herangezogen werden oder mehrere virtuelle Maschinen eines Großrechners, die einzelne Subprogramme beinhalten. Zur Abgrenzung sei das Aufteilen von Rechenaufgaben eines einzigen Programms auf mehrere CPUs erwähnt, was aber hier nicht thematisiert wird.

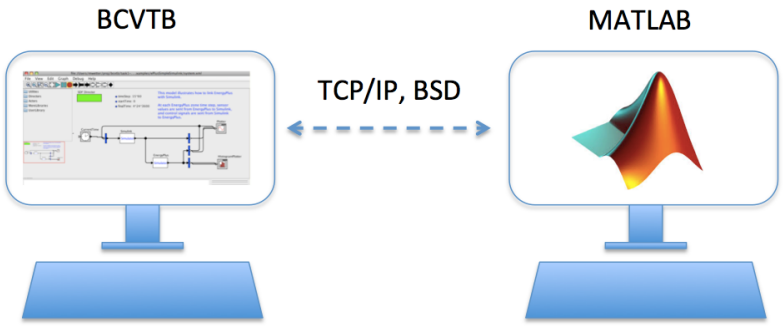


Abbildung 3.8: Kommunikation bei der Parallelisierung der Co-Simulation.

Die Parallelisierung wird dadurch möglich, dass durch die Erweiterung von Ptolemy II in BCVTB der Simulator Actor xml-Dateien erstellt, die wiederum beliebige ausführbare Dateien aufrufen können (nachzulesen in [12]). Die Kommunikation über BSD-Sockets entspricht ohnehin einer TCP/IP-Schnittstelle und kann daher auch über ein Netzwerk genutzt werden. Unter dem Windows Betriebssystem, auf dem bei dieser Studie BCVTB betrieben wird, entspricht dieses Vorgehen einem Aufruf eines Batch-Files[1]. In BCVTB reicht es daher im Simulator-Actor statt des Namens eines bestimmten Programms ein Batch-File zu übergeben. Durch Erweiterungen lässt sich dieses Verhalten auch auf Netzwerkbefehle erweitern. Um dies zu erreichen, wird die frei zugängliche PuTTY Software benutzt, die den Zugang zu externen Rechnern über das Secure Shell (SSH) Protokoll erlaubt. Im Speziellen werden die zwei Module PSCP für das Kopieren von Daten und PLINK für das Ausführen von command line arguments benutzt. Um das Starten eines Subprogramms auf externen Rechnern zu demonstrieren, wird das Vorgehen am Beispiel von MATLAB gezeigt. Die Umsetzung auf andere Gastprogramme ist möglich, im Falle von Dymola sogar einfacher, da dieses ohnehin von einem Batch-File gestartet wird. Um die Flexibilität dieses Vorgehens zu zeigen, wird der MATLAB-Client auf dem MacOS X 10.8 Betriebssystem ausgeführt, die Co-Simulation ist in dem Fall eine plattformübergreifende Anwendung. Der externe Rechner (egal ob virtuelle oder physische Maschine) wird Server genannt.

[1] Als Batch wird unter Windows eine zeilenweise abgearbeitete Reihung von Systemkommandos bezeichnet.

Damit die Simulation durchgeführt werden kann, muss MATLAB sämtliche relevanten Dateien erhalten. Es wird der Ansatz gewählt, alle Files an den Remote Server zu übergeben, da die Arbeit am Modell lokal erfolgen soll und nur die Berechnung extern ausgeführt wird. Die zu übergebenden Dateien sind daher das MATLAB-File selbst, die Dateien der Simulationsparameter und die Informationen zur BSD-Kommunikation. Letztere wird von BCVTB für jedes Gastprogramm erstellt und beinhaltet Informationen zur Verbindung, also Hostname und Port. Die Datei muss im Arbeitsverzeichnis des Simulators am Server abgelegt werden, sie wird in weiterer Folge nach ihrem Namen als socket.cfg bezeichnet. Der Quelltext des Batch-Files ist in Anhang B nachzulesen.

Um eine SSH-Verbindung aufbauen zu können, muss eine Authentifizierung am Server stattfinden. Sie geschieht im Normalfall durch die manuelle Eingabe eines Passworts. Da dies bei jedem Simulationslauf für jedes Subprogramm, das extern ausgeführt werden soll, notwendig wäre, wird auf public key authentification zurückgegriffen. Es wird ein Schlüsselpaar erzeugt, das aus einem öffentlichen (public) und einem privaten (private) key besteht. Der Server bekommt den öffentlichen Schlüssel, mit dem es möglich ist den geheimen private key bei der Verbindung zu überprüfen. Damit diese Überprüfung automatisiert im Hintergrund ablaufen kann, wird das ebenfalls im PuTTY Paket enthaltene Programm *Pageant* benutzt. Nachdem die notwendigen Daten dem Server zur Verfügung gestellt wurden, kann die Simulation wie erwähnt per SCP gestartet werden. Bei der Simulation kann es zu Fehlern kommen, wenn der Name des Hosts dem Server nicht bekannt ist. Dies ist ein Problem, weil im socket.cfg File die IP-Adresse des Hosts nicht aufscheint, sondern nur der Name. Durch Verknüpfung der IP-Adresse mit dem Hostnamen in den Einstellungen des Servers kann das Problem behoben werden.

Durch die Art, wie BCVTB im Normalfall MATLAB aufruft, tritt ein weiteres Problem auf. Beim direkten Start von MATLAB werden von BCVTB Systemvariablen übergeben. Dies geschieht unbemerkt im Hintergrund. Im Fall des entfernten Zugriffs ruft BCVTB allerdings ein Batch-File auf, daher werden diese Variablen nicht an MATLAB weitergegeben. Die Systemvariablen werden allerdings für die Einbindung der MATLAB-Bibliothek für die Socket-Verbindung gebraucht. Deshalb ist es notwendig, Modifikationen am MATLAB-File durchzuführen. Es müssen zwei Systemvariablen gesetzt werden, BCVTB_OS und BCVTB_HOME. Die erste gibt an, welches Betriebssystem dem Prozess zugrunde liegt. Unter Windows muss hier „windows“ gesetzt werden, unter Linux oder MacOS kann das Setzen entfallen, da von vornherein der korrekte Wert genommen wird. Die Variable BCVTB_HOME muss zum Verzeichnis der BCVTB-Installation zeigen. Dies wird benötigt, um die MATLAB-Bibliothek nutzen zu können. Schlussendlich

ist es notwendig, den Befehl für das Einbinden der Bibliothek anzugeben. Der Pfad muss hier auf den lib/util Ordner zeigen, nicht auf lib/matlab. Um den MATLAB-Aufruf sauber abzuschließen, muss im MATLAB-File der „exit“ Befehl ausgeführt werden. Damit wird verhindert, dass die offene MATLAB-Instanz weitere Aufrufe behindert. In Anhang C ist das grundlegende Konstrukt zur Remote-Simulation unter MacOS nachzulesen.

Nach Beendigung der Simulationen ist es wünschenswert, das log-File vom Server auf den Host zu holen, um eine vollständige Dokumentation der Simulation zentral verfügbar zu haben. Mit diesem letzten SCP-Befehl ist das Batch-File vollständig ausgeführt, die Simulation abgeschlossen und sämtliche Subprogramme inklusive BCVTB werden wieder geschlossen. Ergebnisse können vor dem Schließen entweder von den Subprogrammen oder von BCVTB in externe Dateien geschrieben werden. Die MATLAB-Kontrollinstanz kann abschließend für das Postprocessing der Daten genutzt werden. Speziell das Visualisieren und der Vergleich von Daten mehrerer Simulationsdurchgänge ist dadurch leicht möglich.

Als Demonstrationsbeispiel wird ein Szenario aus Abschnitt 6 herangezogen. Das MATLAB-Modell wird über das vorgestellte Verfahren an einen zweiten Rechner weitergegeben und dadurch die Simulation auf zwei Computern parallel ausgeführt. Bei der Simulation, bei der alle Programme auf einem Rechner laufen, ergibt sich eine Rechenzeit von 16.3 Sekunden. Dieselbe Simulation dauert auf zwei Rechner verteilt 16.6 Sekunden. Entgegen den Erwartungen steigt die Dauer bei diesem Beispiel an, wenn auch nur wenig. Dieses Ergebnis resultiert aus dem Umstand, dass die Simulation mit MATLAB auf einen Rechner mit geringerer Rechenleistung ausgelagert wurde, was sich klarerweise auf die Performance auswirkt. Außerdem führt die Netzwerkkommunikation zu zusätzlicher Latenz. Der Vorteil der verteilten Simulation kann bei diesem Beispiel nicht ausgespielt werden, da der Großteil der rechenintensiven Tätigkeit weiterhin von derselben Maschine bewältigt wird. Erst wenn das Modell eine gleichmäßige Aufteilung zulässt, kommt die zusätzliche Rechenleistung zum Tragen.

Die einfache Parallelisierung von Co-Simulationen mit BCVTB ist damit möglich. Die Anpassungen, die an den Modellen vorgenommen werden müssen, um eine Co-Simulation zu realisieren, sind gering. Damit kann ein Modell entwickelt und erst später die Entscheidung getroffen werden, ob die Simulation auf mehreren Rechnern ausgeführt wird. Vor allem, wenn Simulationen sehr umfangreich sind, ist diese Möglichkeit hilfreich.

3.4. Fehler durch den Informationsfluss in BCVTB

Wie bereits erwähnt arbeiten die Actors in BCVTB mit Ein- und Ausgangssignalen. Alle Blöcke erhalten an ihren Inputs Informationen. Zu jeder Iteration werden die internen Berechnungen durchgeführt und die gewonnenen Daten dem Outputport zur Verfügung gestellt. Für den nachfolgenden Actor stellt dies wiederum den Input dar. In der darauffolgenden Iteration bearbeitet der nächste Actor diesen Input zu einem Output und so weiter (siehe Abbildung 3.9). Das hat zur Folge, dass bei einer seriellen Verbindung mehrerer Module Informationen mehrere Zeitschritte brauchen, um vom Anfang bis zum Ende zu laufen.

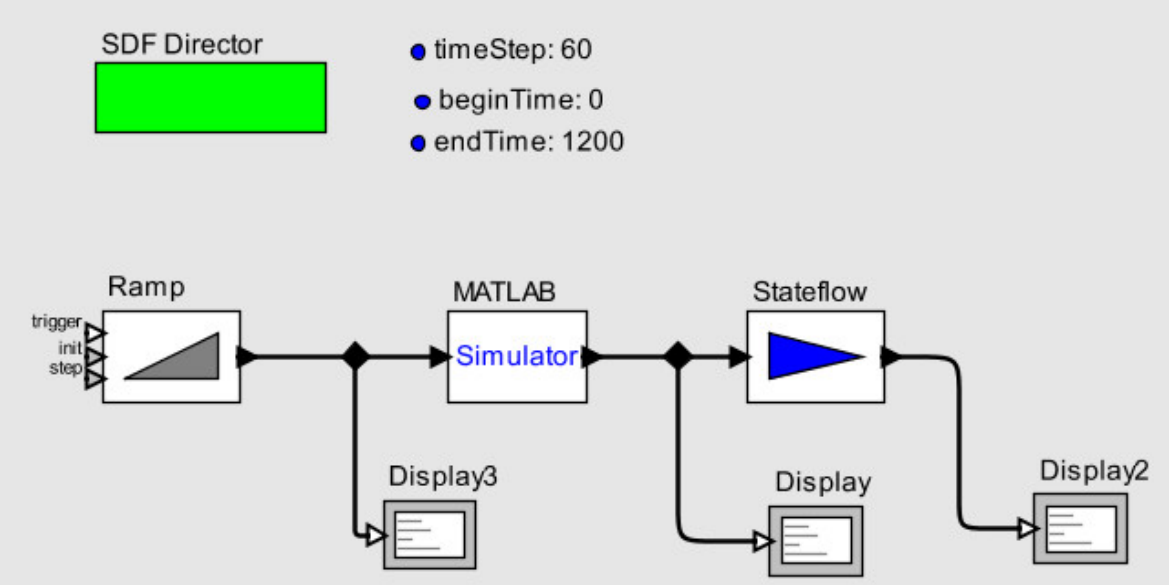


Abbildung 3.9: Modell zum Test der Verzögerung des Informationsflusses in BCVTB.

Um die Auswirkungen dieser Tatsache zu beleuchten, wird ein einfaches Beispiel betrachtet. In BCVTB wird eine Co-Simulation implementiert, die MATLAB und Stateflow als Gastprogramme beinhaltet. Damit zeigen sich mögliche Probleme auch bei der Umsetzung einer komplexen Variante. In der Abbildung 3.9 sieht man das Modell in BCVTB. Als Eingang in das System wurde ein Rampenelement gewählt, das mit einem Startwert von 2000 initialisiert wird und bei jeder Iteration den Output um den Wert 200 erhöht. Diese Quelle wird als Input auf den Simulator-Actor aufgeprägt, der eine MATLAB-Instanz aufruft. In MATLAB ist eine Art datenbasierte Kennlinie implementiert. Der Actor erhält einen Inputwert und gibt den zur Kennlinie passenden Output aus. Die Kennlinie ist so beschaffen, dass bei steigendem Eingang auch der Output größer wird. Bei einem Eingangswert von 3500 wird der maximale Output von 30000 erreicht,

der auch bei höherem Input konstant bleibt. Es ist zu erwähnen, dass die verwendeten Zahlenwerte keine physikalische Relevanz besitzen und nur aus Anschauungsgründen gewählt wurden. Der Output aus dem Simulator-Block ist wieder der Eingang in den nächsten Block, nämlich einen Simulator Actor, der Stateflow aufruft. In Stateflow ist eine simple Entscheidungsstruktur abgebildet, die entscheidet, ob ein Wert von 30000 bereits erreicht wurde. Sollte das der Fall sein, wird der Output auf -1 gesetzt, ansonsten auf $+1$. Initialisiert werden die beiden Simulator-Blöcke sowohl mit Output als auch Input null. Die drei Display-Blöcke dienen der Ausgabe der Daten.

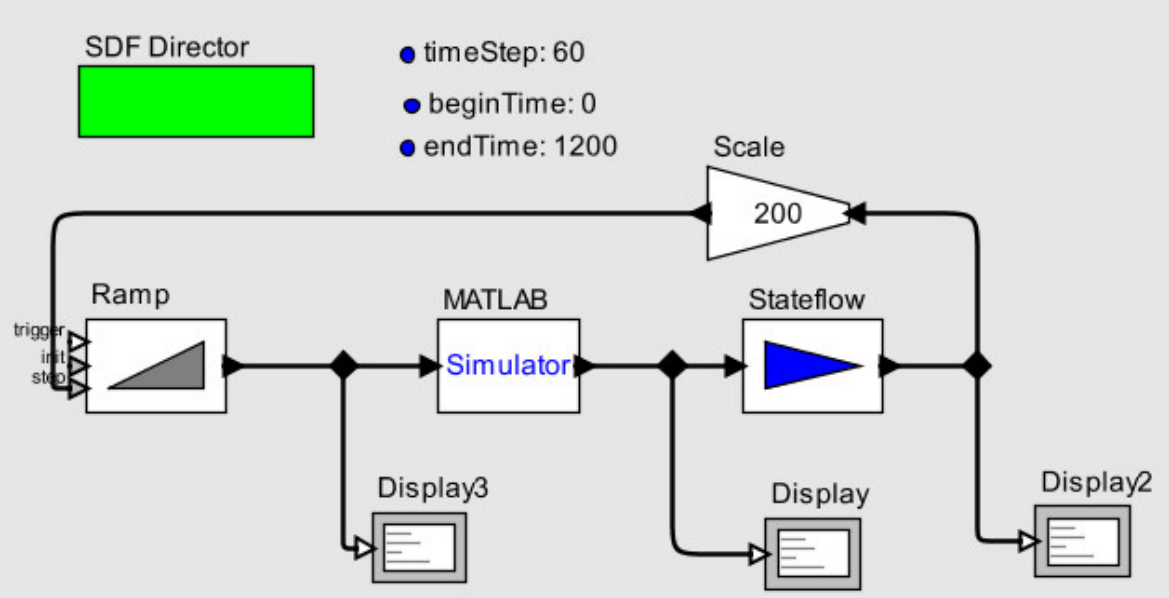


Abbildung 3.10: Das Modell in Abbildung 3.9 wurde um eine Feedbackschleife erweitert: Die Entscheidung aus Stateflow wird genutzt, um die Steigung der Rampe zu regeln.

Nach dem Start der Simulation erfolgt zuerst die Zuweisung der Initialisierung und danach die erste Iteration. Während der Iteration arbeiten die Blöcke mit den Daten an ihrem jeweiligen Input, also den Werten der Initialisierung des vorhergehenden Blocks. Der Output der Rampe wird in der ersten Iteration gemäß der vorgegebenen Steigung erhöht. MATLAB konnte im ersten Schritt bereits mit dem Wert der Rampe arbeiten und kann daher einen sinnvollen Wert weitergeben. Stateflow kann diese Information aber erst in der zweiten Iteration nutzen. Zu diesem Zeitpunkt kann Stateflow mit einem sinnvollen Input arbeiten, obwohl die Rampe schon zur Zeit null die Information besaß. Anders ausgedrückt, dauert es zwei Iterationen, bis die Information von der Rampe bei Stateflow angelangt ist. Jeder weitere nachgeschaltete Block würde wieder einen

Zeitschritt verzögert die Informationen erhalten.

Iteration	Ramp Output	MATLAB Output	Stateflow Output
0	2000	0	0
1	2000	13074	1
2	2200	13074	1
3	2400	15672	1
4	2600	18518	1
5	2800	21482	1
6	3000	24328	1
7	3200	26824	1
8	3400	28736	1
9	3600	29858	1
10	3800	30000	1
11	4000	30000	-1
12	3800	30000	-1
13	3600	30000	-1
14	3400	30000	-1
15	3200	29858	-1
16	3000	28736	-1
17	2800	26824	-1
18	2600	24328	-1
19	2400	21482	-1
20	2200	18518	-1

Tabelle 3.1: Ergebnisse der Simulation mit Feedback-Schleife.

Um die Problematik zu verdeutlichen, wird das Modell um eine Feedbackschleife erweitert. Das neue Modell ist in Abbildung 3.10 zu sehen. Je nach der Entscheidung aus Stateflow, ob der Grenzwert bereits erreicht wurde, wird die Steigung der Rampe auf entweder +200 (Grenze noch nicht erreicht) oder −200 (Grenze wurde erreicht) gestellt. Wenn die Information an der Rampe gleichzeitig allen Blöcken zur Verfügung stünde, würde in MATLAB genau einmal der Grenzwert erreicht, Stateflow würde die Steigung ändern und der Wert der Rampe würde abnehmen, weswegen auch MATLAB den Grenzwert nicht mehr erreichen würde. Durch die beschriebene Verzögerung liefert MATLAB den Grenzwert von 30000 allerdings fünf Mal zurück, siehe Tabelle 3.1. Dies ist darauf zurückzuführen, dass alleine Stateflow zwei Iterationen auf die Information warten muss. Dadurch arbeitet die Rampe mit der falschen Steigung und nimmt weiter zu. Wenn die geänderte Steigung beim Rampenblock ankommt, dauert es wieder, bis der Wert 3500 unterschritten wird. Das bedeutet, dass wegen der Zeitverzögerung der Informationsweitergabe Rückkopplungen fehlerbehaftet sind. Es ist zu erwähnen, dass bei Iteration

eins die Rampe die Steigung null hat. Dies geschieht aufgrund der Initialisierung von Stateflow auf den Wert null. Dadurch geht lediglich ein Zeitschritt verloren, es ändert an der Problematik nichts.

Es ist offensichtlich, dass dieses Verhalten Fehler in den Simulationsergebnissen erzeugt. Im Falle eines rein seriell arbeitenden Modells kann der Einfluss auf das Ergebnis eventuell nachträglich korrigiert werden. Es ist wichtig, dass die Simulationszeit ausreichend lang gewählt wird, da die Simulatoren im hinteren Teil der seriellen Schaltung in den ersten Iterationen nicht sinnvoll arbeiten können. Noch deutlicher wird der Fehler, wenn der zweite Fall eintritt, nämlich eine Rückkopplung im System vorhanden ist. Die eigentliche Idee, dass bei Erreichen der Grenze sofort die Steigung reduziert wird, funktioniert nicht exakt, man schießt also über das Ziel hinaus. Dieser Fehler kann minimiert werden, indem die Kommunikationsrate erhöht wird. In Fällen, wo die Genauigkeit von großer Bedeutung ist, kann es zweckmäßig sein, über Subsysteme und eigene Directors die Anzahl der Iterationen nur in bestimmten Teilen des Modells zu erhöhen. Es sei noch einmal erwähnt, dass der Fehler dadurch nur reduziert, nicht aber eliminiert wird.

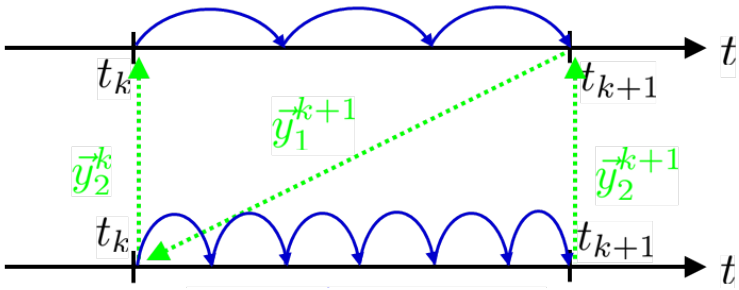


Abbildung 3.11: Funktionsprinzip der Co-Simulation nach dem Gauss-Seidl-Typ.

Das Problem entsteht durch die grundsätzliche Funktionsweise, wie BCVTB Zeitschritte durchläuft. Damit eine Rückkopplung fehlerfrei funktioniert, müssten alle Teilsimulationen zwischen den Kommunikationsintervallen warten, bis der vorherige Teil fertig gerechnet hat. Dieses Vorgehen entspricht einer Co-Simulation nach dem Gauss-Seidl-Typ, das in Abbildung 3.11 zu sehen ist. Dadurch wird die Simulation allerdings erheblich verlangsamt. Dieser Effekt ist vor allem dann gravierend, wenn mehrere Simulatoren eine Rechenzeit in ähnlicher Größenordnung haben. Statt parallel rechnen zu können, müssen sie aufeinander warten, daher steigt die gesamte Rechenzeit des Systems multiplikativ an.

4. Co-Simulation mit CATIA V6

Im Gebiet des konstruktiven Maschinenbaus ist es seit vielen Jahren der Standard, Computer Aided Design (CAD) für die virtuelle Produktentwicklung zu nutzen. Der Ansatz, dynamische Systemsimulation für die Berechnung heranzuziehen, ist im Gegensatz dazu noch eher ein Novum. Dassault Systèmes hat mit CATIA V6 eine Möglichkeit geschaffen, diese zwei Teilbereiche miteinander zu verknüpfen. In der aktuellen Version ist in CATIA V6 eine Modelica Simulationsumgebung implementiert worden, sie nennt sich Dynamic Behavior Modeling (DBM). Damit ist es möglich, neben den umfangreichen Methoden der CAD Konstruktion auch Systemsimulationen umzusetzen.

Das Ziel dieser Arbeit ist es, einen Schritt weiter zu gehen. Die Möglichkeit, die DBM Umgebung mit den Modelica Libraries nutzen zu können, bietet in dem Zusammenhang die Chance, CATIA V6 für eine Co-Simulation zugänglich zu machen. Der Vorteil darin besteht, das mächtige Tool der CAD Konstruktion und damit Informationen der CAD-Modelle in eine Systemsimulation einbetten zu können. Bisher war es notwendig, diese zwei Schienen parallel zu fahren, was nicht nur zusätzlichen Aufwand bei der Implementierung bedeutet – ein Großteil der physikalischen Daten muss doppelt eingegeben werden – sondern auch, dass mögliche Synergieeffekte verloren gehen.

Im Detail ist der Ansatz folgender: Können Daten aus der CAD-Konstruktion für eine dynamische Systemsimulation herangezogen werden, dann stehen sie automatisch zur weiteren Verarbeitung zur Verfügung. Die Daten müssen nicht separat eingegeben werden, was den Arbeitsablauf vereinfacht und Fehler vermeidet. Durch die Integration dieser mechanischen Systemsimulation in eine Co-Simulation können die anderen Subprogramme ebenfalls auf die Informationen zugreifen. Die Verbindung von spezieller Software kann daher die Schwächen der anderen Programme ausgleichen. Ein Beispiel ist die Simulation von gesamten Fahrzeugmodellen. Die Geometriedaten und Massenparameter, die im Zuge der Entwicklung der Komponenten ohnehin als CAD-Dateien vorliegen, können im Idealfall für eine mechanische Systemsimulation verwendet werden. In weiterer Folge können über Co-Simulation Modelle anderer Hersteller eingebunden werden, wie zum Beispiel die Motorsteuerung, Klimasysteme, elektrische Systeme oder dergleichen. Durch die flexible Anbindung können die Modelle von Subsystemen in unterschiedlichen Programmiersprachen oder Simulatoren erzeugt worden sein. Es werden etwa Regelungen klassisch in Simulink als gerichtete Signalflussgraphen umgesetzt. Diese verschiedenen Modelle können dann zu einem Gesamtsystem zusammengesetzt werden, um das vollständige Fahrzeug zu simulieren. Da die Subsysteme untereinander lediglich

Daten austauschen, die inneren Vorgänge allerdings in sich geschlossen sind, ist die so gewonnene Simulation robust und wenig fehleranfällig.

Die folgende Studie zeigt, welche Möglichkeiten momentan in Verbindung mit CATIA V6 existieren, diesen Anforderungen gerecht zu werden. Vor allem auf die Implementierung der Co-Simulation wird besonderer Wert gelegt, da insbesondere dadurch Flexibilität bei der Modellierung zu erreichen ist.

4.1. Dynamic Behavior Modeling

In CATIA V6 wird die dynamische Systemsimulation über die Dynamic Behavior Modeling Workbench aufgesetzt. Im Prinzip entspricht diese von den Fähigkeiten her einer Dymola Instanz. Die Funktionalität wurde allerdings in das in CATIA übliche Design integriert, dementsprechend ergeben sich Änderungen in der Bedienung. CATIAs Graphical User Interface (GUI) verfolgt einen Ein-Fenster-Ansatz. Das bedeutet, dass CATIA immer in einem Hauptfenster läuft, sämtliche Unterfunktionen spielen sich in Subfenstern ab. Der offensichtliche Nachteil entsteht dadurch, dass Funktionen wie die Windows Tastenkürzel nicht dafür benutzt werden können, um zwischen Fenstern umzuschalten.

Die vermutlich größte Änderung ergibt sich daraus, dass ab CATIA V6 sowohl das Lizenz- als auch das Dateimanagement nicht lokal sondern über einen externen Server ablaufen. Bei der Lizenzverwaltung ist dieses Vorgehen nicht ungewöhnlich, die Verwaltung der Daten auszulagern, allerdings schon und spiegelt den Trend des Cloudcomputings wider. Der daraus resultierende Vorteil ist, dass sämtliche Benutzer Zugriff auf dieselben Dateien haben (sofern die Berechtigung vorhanden ist). Damit lässt sich kollaboratives Arbeiten leichter umsetzen. Auch ein Versionskontrollsystem ist im Product Lifecycle Management (PLM) integriert und das automatische Backup von Daten ist auf einem zentralen Server wesentlich leichter und effizienter durchführbar. Nachteile dieses Systems ergeben sich hauptsächlich aus der restriktiven Datenverwaltung, so ist es zum Beispiel nur mit sehr großem Aufwand und Administratorrechten möglich, Dateien vollständig zu löschen. Wichtig ist zu erwähnen, dass das lokale Ex- und Importieren von Dateien sehr wohl möglich ist, als praktikable Lösung für das tägliche Arbeiten ist es aber nicht anzusehen. Durch das zentrale Datenmanagement muss jede Bibliothek, die benutzt werden soll, auch auf dem Server abgespeichert werden.

Zusammengefasst kann die DBM-Umgebung im Prinzip ähnlich zur Dymola-Software verwendet werden. Dies alleine bringt noch keine weiteren Vorteile, die interessanten Aspekte kommen hinzu, wenn man eine Ebene höher geht.

4.2. RFLP-Environment

Im Zuge der Idee, einen ganzheitlichen Produktentstehungsprozess zu gestalten, wurde in CATIA V6 der Virtual Product Management (VPM) Functional Logical (FL) Editor eingeführt. Dies ist eine Weiterführung des PLM-Gedankens und dient dazu, sämtliche produktrelevanten Informationen zentral an einem Ort zu bündeln. Im FL-Editor gibt es vier Bereiche, die in der Produktentstehung von Bedeutung sind und in aufsteigender Reihenfolge den Detailgrad des Produkts darstellen. An oberster Stelle steht der Bereich „Requirements“, der die Anforderungen an das Produkt textuell hinterlegt. Weiters werden die funktionalen Eigenschaften im Bereich „Functional“ definiert. Für die Anwendung wichtige Punkte sind „Logical“ und „Physical“. Die logischen Eigenschaften legen fest, wie verschiedene Teile des Ganzen in Bezug zueinander stehen. Im letzten Bereich werden die tatsächlichen physischen Eigenschaften der einzelnen Teile hinterlegt. Zusammengefasst wird dieses Vorgehen in der Requirements, Functional, Logical, Physical (RFLP) Umgebung, die den Ausgangspunkt für die Integration von CAD-Daten in die mechanische Systemsimulation darstellt. In Abbildung 4.1 ist angedeutet, wie die RFLP- mit der DBM-Umgebung zusammenhängt und welche Workbenches verwendet werden.

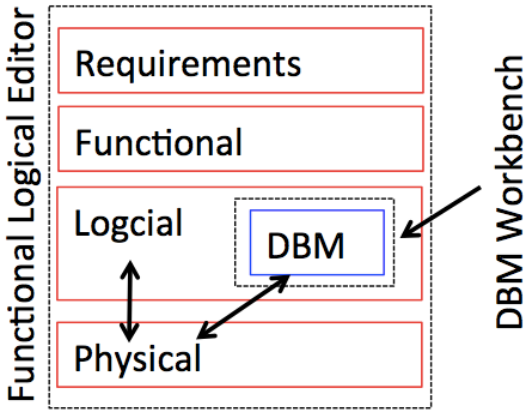


Abbildung 4.1: Struktur der RFLP-Umgebung und Zusammenhang mit DBM. Strichliert sind die verwendeten Workbenches angegeben.

Die Bereiche „Requirements“ und „Functional“ werden in dieser Studie nicht betrachtet, da keine Produktentwicklung an sich stattfindet. Dementsprechend bezieht sich die Untersuchung auf die Bereiche „Logical„ und „Physical“. Die logische Architektur wird mittels Logical References (LR) aufgebaut. Diese repräsentieren logische Teilbereiche, die beliebig angeordnet werden können. Es ist möglich, Referenzen ineinander zu verschachteln und zu verbinden. Die Verbindungen werden über Ports und Connections

verwirklicht. Der Verwendungszweck eines Ports ist fest definiert. Die Möglichkeiten sind Ausgang, Eingang, Aus- und Eingang sowie ungerichtet. Ports können mit Connections verbunden werden. Abbildung 4.2 zeigt eine exemplarische Logikstruktur. Es ist zu erkennen, dass sich eingebettet in einer LR zwei weitere Referenzen befinden, was die hierarchische Anordenbarkeit demonstriert. In dem Beispiel sind die Ports als ungerichtet definiert, was an den rechteckigen Endpunkten der Verbindung erkennbar ist.

Den logischen Referenzen kann in weiterer Folge ein dynamisches Verhalten in Form eines Modelica-Modells hinterlegt werden. Mit der DBM-Workbench kann dieses Modell erstellt werden, es entspricht einer Modelica Library. Die logischen Ports werden als Modelica Interfaces automatisch in das Modell eingefügt. Es ist damit eine weitere Hierarchieebene über dem DBM-Modell geschaffen worden. Der ohnehin hierarchische Aufbau von Modelica selbst wird dadurch nicht eingeschränkt, sondern nur untergeordnet.

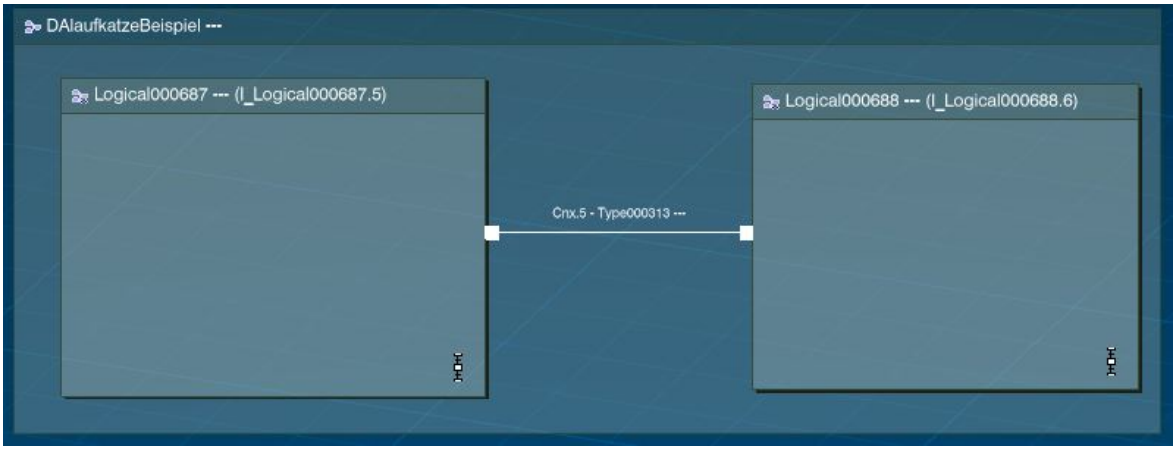
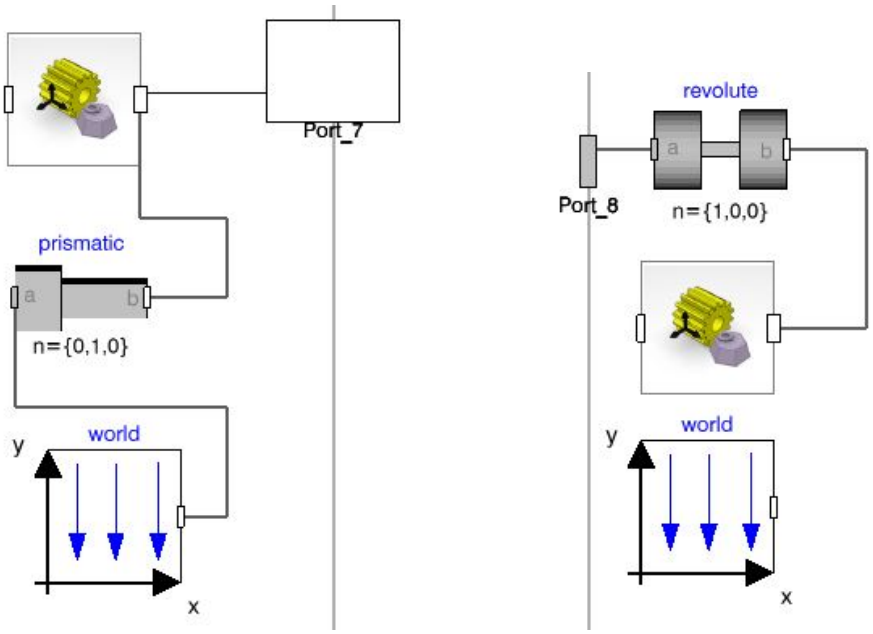


Abbildung 4.2: Eine exemplarische Logikstruktur. Ineinander verschachtelte Logical References (LR), denen ein dynamisches Systemverhalten hinterlegt ist.

Die logischen Referenzen können um eine geometrische Komponente erweitert werden. Spezielle CAD Modelle namens 3DRepresentation können den LR hinterlegt werden und stellen die geometrisch-physikalische Repräsentation einzelner Bauteile dar. Die Integration von 3DParts wird in der Version CATIA V6 R2013x noch nicht unterstützt, was die derzeitigen Möglichkeiten einschränkt. Es wurde von den Entwicklern allerdings angekündigt, dieses Manko in zukünftigen Versionen zu beheben. Die Parameter, die damit in das System eintreten, sind geometrische und Trägheitsdaten.

Um diese Informationen zu nutzen, ist es notwendig, die Geometriedaten mit den Verhaltensmodellen zu koppeln. Dafür existiert ein eigener Block CATIA_3DBody, der

für diese Verbindung zuständig ist. Die Komponenten können als MultiBody Objekte gesehen werden, sind also nur über die passenden Interfaces zu verbinden. Sobald in einem DBM-Modell ein CAD-Element verlinkt ist, kann die Simulation nicht mehr in der DBM Umgebung ausgeführt werden. Es ist einzig und allein möglich, ein auf diese Art erstelltes Modell in der übergeordneten RFLP Umgebung zu benutzen, da nur hier alle notwendigen Daten zusammenlaufen. Diese Tatsache schränkt die Möglichkeiten, Modelle für eine Co-Simulation zu nutzen, erheblich ein. Ein weiteres Problem ist, dass es in der verwendeten Version von CATIA nicht möglich ist, vorhandene 3DParts in die Simulation einzubinden.



(a) Schlitten. (b) Pendel.

Abbildung 4.3: Die DBM-Modelle für Schlitten und Pendel, die den Logical References aus Abbildung 4.2 hinterlegt sind.

Um eine Simulation mit Logical References, DBM und den angeschlossenen physikalischen Daten zu illustrieren, wird ein Demonstrationsbeispiel herangezogen. In diesem Beispiel wurde ein einfaches Pendel modelliert, das auf einem reibungsfrei beweglichen Schlitten befestigt ist. Der gesamte Aufbau ist ungedämpft und bewegt sich nur aufgrund der initialen Auslenkung des Pendels. Die logische Struktur in Abbildung 4.2 zeigt zwei Logikreferenzen, die über ungerichtete Ports verbunden sind. Die Ports wurden deshalb gewählt, da aufgrund des mechanischen Verhaltens die MultiBody Bibliothek verwendet wird. Die linke LR enthält das DBM Modell des Schlittens, an dem ein Pendel befestigt ist. Das Pendel selbst ist in der rechten Referenz untergebracht. Die Symbole in den

rechten unteren Ecken der LR zeigen, dass ihnen jeweils ein DBM-Modell hinterlegt ist, diese sind in Abbildung 4.3 zu sehen.

Das Modell des Schlittens besteht aus einem CATIA_3DPart, der sich translatorisch frei bewegen kann. Der Körper ist als Quader umgesetzt und besitzt einen Anknüpfungspunkt, an dem das Pendel befestigt ist. Dieses Modell alleine führt keine Bewegung aus, die Verbindung zu der anderen Logikreferenz induziert das dynamische Verhalten.

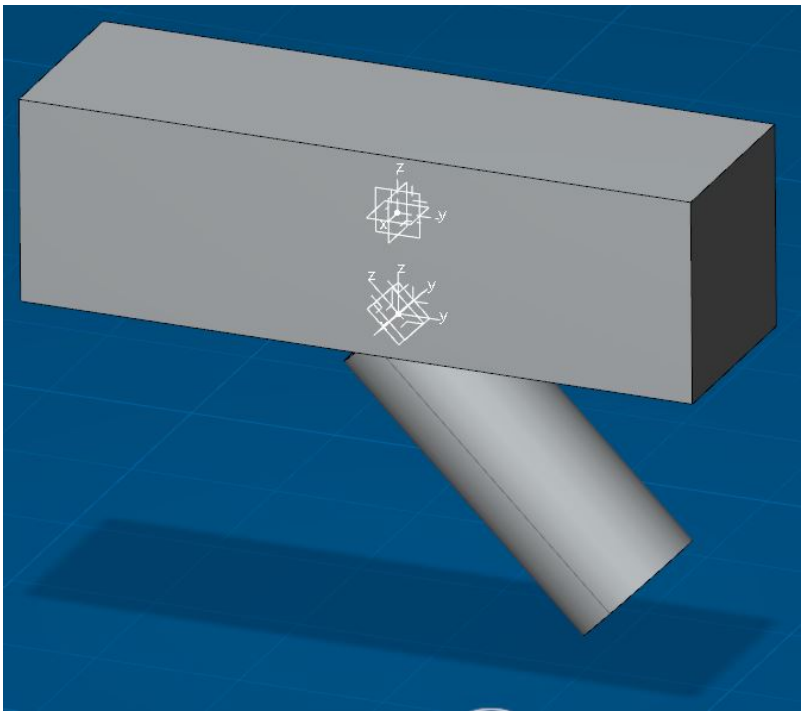


Abbildung 4.4: Ein Bild der Animation der CAD-Komponenten. Der Quader stellt den translatorisch beweglichen Schlitten, der Zylinder das rotatorisch bewegliche Pendel dar.

Das zweite Modell ist das des Pendels. Ein zweiter CATIA_3DPart, ein Zylinder, ist an einem revoluteJoint befestigt, um die Rotation zu ermöglichen. Das Pendel besitzt einen Anknüpfungspunkt an einem Ende. Am revoluteJoint ist die Anfangsbedingung festgelegt, dass das Pendel um 45° ausgelenkt ist. Die Verbindung zwischen logischer Struktur und DBM-Modell wird über Port_7 und Port_8 realisiert.

Zur Veranschaulichung des Systems ist in Abbildung 4.4 ein Bild der Animation der Simulation abgebildet. Auf der oberen Seite ist der Schlitten als Quader erkennbar, an dessen Unterseite das zylinderförmige Pendel hängt. Die weißen Koordinatensysteme bilden die Anknüpfungspunkte der CATIA_3DParts zu anderen Komponenten. Bei genauem Betrachten ist zu sehen, dass auf der Unterseite des Quaders zwei Koordina-

tensysteme mit demselben Ursprung, aber um die y-Achse verdreht, zu sehen sind. Dies entspricht den miteinander verbundenen Punkten des Schlittens und des Pendels, welche sich aufgrund des revoluteJoints um die y-Achse verdrehen können. Es ist zu erwähnen, dass es keine Kollisionskontrolle zwischen den zwei CAD-Teilen gibt. Für eine physikalisch korrekte Modellierung wäre eine Implementierung davon allerdings wünschenswert. In Abbildung 4.5 sind zwei Ergebnisse einer fünf Sekunden langen Simulation abgebildet. Die rote Kurve zeigt die Auslenkung des Quaders in Bezug auf den Nullpunkt in Metern, die blaue die Auslenkung des Pendels in Radiant. Die Einheiten werden im Ausgabefenster nicht dargestellt, die Anpassungsmöglichkeiten sind begrenzt.

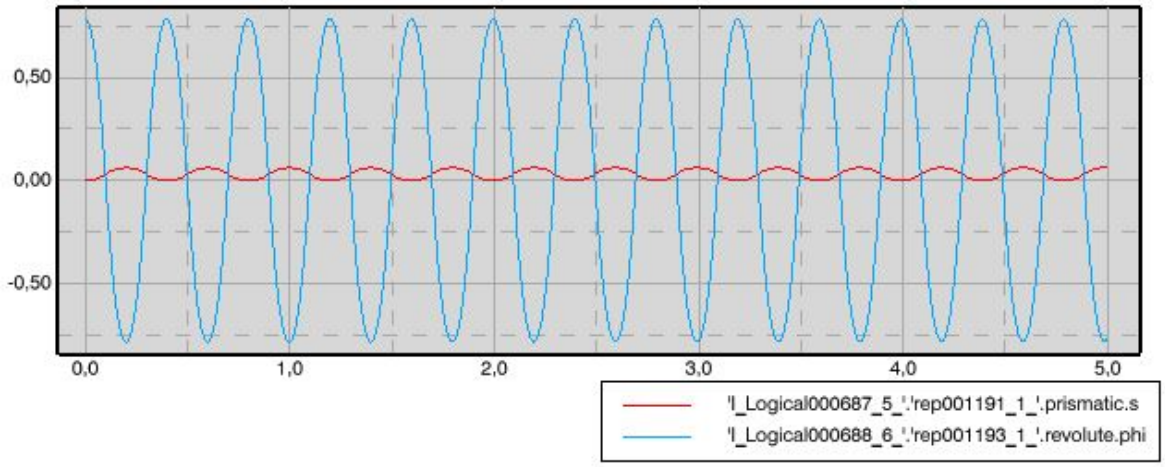


Abbildung 4.5: Das Plotter Fenster in CATIA. Die rote Kurve zeigt die Auslenkung des Schlittens, die blaue den Winkel des Pendels.

Die hierarchische Anordnung der logischen Referenzen, aber auch der anderen Komponenten in der RFLP Umgebung spiegeln denselben Ansatz wie in Modelica wieder. Es wird dadurch möglich, erstellte Modelle in andere Simulationen zu integrieren und dadurch Komponenten wiederzuverwerten. Angewandt auf das Beispiel könnte in einem weiteren Modell ein Doppelpendel simuliert werden. Dazu kann die Logikreferenz mitsamt den Geometrie- und Verhaltenseigenschaften importiert werden.

4.3. CATIA und FMI

Die Möglichkeiten für dynamische Simulationen in CATIA sollen durch die Anbindung an eine Co-Simulation erweitert werden. Der offensichtliche Anfangspunkt ist die Integration von CATIA V6 mit FMI, da auch die zugrundeliegende Software Dymola eine FMI-Schnittstelle bietet. Bei der Benutzung von Functional Mockup Units (FMU) ist

zuerst die Wahl zu treffen, welcher Simulator für die Gesamtsimulation verwendet wird und welche anderen Programme nur die FMUs beisteuern. Diese Entscheidung schlägt sich in der Frage nieder, ob in CATIA FMUs ex- oder importiert werden sollen. Beim Import dient CATIA selbst als Simulationsoberfläche und steuert die Co-Simulation. Im Fall des Exports wird ein Modell in CATIA implementiert und anschließend per FMI-Schnittstelle ausgegeben. Ein anderes Programm kann diese FMU anschließend einbetten.

4.3.1. Export über FMI in CATIA

Die erste Variante, die hier betrachtet werden soll, ist des Exports von FMUs in CATIA. Es gibt zwei verschiedene Funktionen, die den Namen „FMU-Export“ tragen. Einerseits ist es in der Dynamic Behavior Modeling Umgebung möglich, ein Modell als FMU zu exportieren. Diese Funktionalität bietet dieselben Möglichkeiten und Beschränkungen wie in Dymola. Damit ist durch den Einsatz von CATIA noch kein zusätzlicher Nutzen entstanden. Eine Möglichkeit, die Grenzen des Machbaren zu erweitern, ist das Miteinbeziehen von CATIA_3DParts, um die Informationen von CAD-Komponenten auch außerhalb von CATIA nutzen zu können. Wie sich zeigt, ist es möglich, solche Modelle als FMU zu exportieren. Bei der Integration der FMU in ein Dymola Modell zeigte sich allerdings, dass damit keine lauffähigen Simulationen zu erstellen sind. Es entsteht dasselbe Problem wie beim Ausführen dieses Modells in der DBM-Umgebung, nämlich dass die CAD-Komponenten nicht gefunden werden. Die Restriktion, dass CAD-Komponenten nur in Simulationen in der RFLP-Umgebung genutzt werden können, kann dadurch nicht umgangen werden. Der naheliegende Ansatz ist, zu versuchen, ob man in der RFLP-Umgebung einen FMU-Export erzielen kann, und es gibt tatsächlich eine Funktionalität unter diesem Titel. Die Bezeichnung stellte sich allerdings als irreführend heraus und es besteht nicht die Möglichkeit, ein Modell in der Functional Logical Workbench als FMU zu exportieren. Es ist lediglich möglich, eine vorher importierte FMU wieder zu extrahieren.

Es lässt sich zusammenfassen, dass es nicht möglich ist, über die aktuell in Dymola umgesetzten Möglichkeiten des FMU-Exports hinaus die Vorteile von CATIA in anderen Simulatoren zu nutzen. Um auf die Informationen der CAD-Daten zugreifen zu können, ist also weiterhin die Functional Logical Workbench unumgänglich. Damit verbleibt die zweite Möglichkeit mit dem FMI zu interagieren, nämlich CATIA selbst als Simulator für eine Co-Simulation zu nutzen und fremde FMUs einzubetten.

4.3.2. Import von FMUs in CATIA

Auch beim Import von FMUs bietet CATIA zwei Methoden. In der Functional Logical Workbench gibt es eine eigene Funktion, beim Erstellen eines Behaviors in einer Logical Reference eine FMU zu definieren. Es wird im Vergleich zu einem normalen DBM-Modell keine Library erstellt, in der gearbeitet werden kann. Stattdessen wird eine FMU direkt in die Logical Reference eingesetzt, wie gewohnt kann diese über logische Verbindungen an andere LR angeknüpft werden. Für die Zuordnung der In- und Outputs der FMU an die Ports gibt es ein einfaches Tool. Dies ist vor allem erforderlich, wenn nach dem Import zusätzliche Ports hinzugefügt werden beziehungsweise die automatische Zuordnung nicht das gewünschte Ergebnis erzielt. Auf diese Weise erstellte Simulationen können im Normalfall ausgeführt werden. Das System ist allerdings sehr fehleranfällig und kleine Schwierigkeiten können zum Verlust des ganzen Modells führen. Ein Beispiel dafür ist, dass bei erfolgloser Simulation die FMU offenbar nicht korrekt zurückgesetzt wird. Bei erneutem Ausführen der Berechnung kann die FMU nicht mehr korrekt initialisiert werden. Um diesen Fehler zu beheben, muss das gesamte Modell neu aufgerufen werden. Es ist anzunehmen, dass mit weiterer Entwicklung der Software diese Probleme behoben werden, im jetzigen Stadium gestaltet sich das Benutzen dieser Funktionen als schwierig.

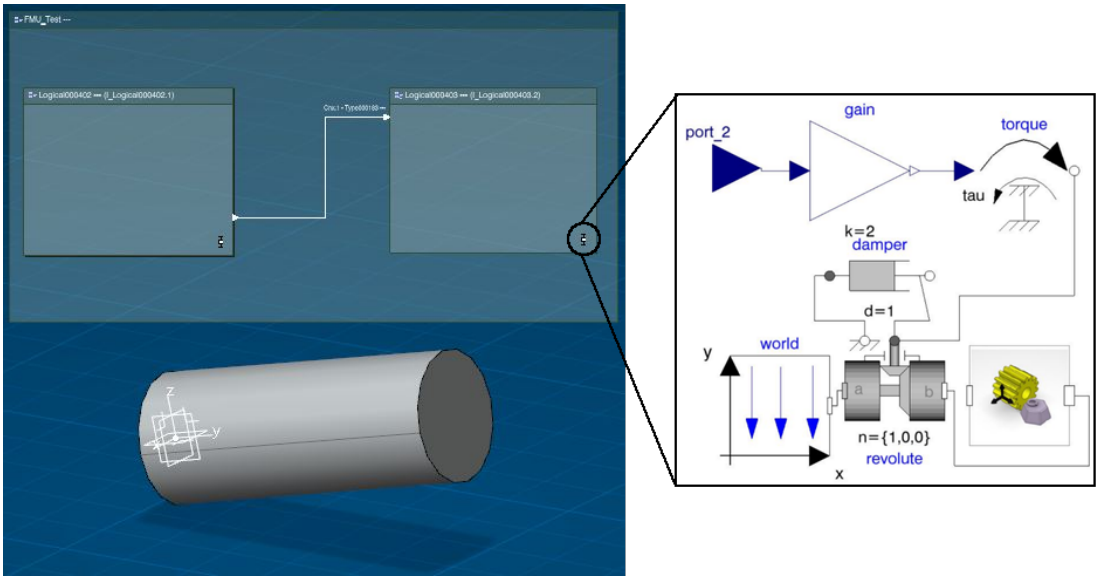
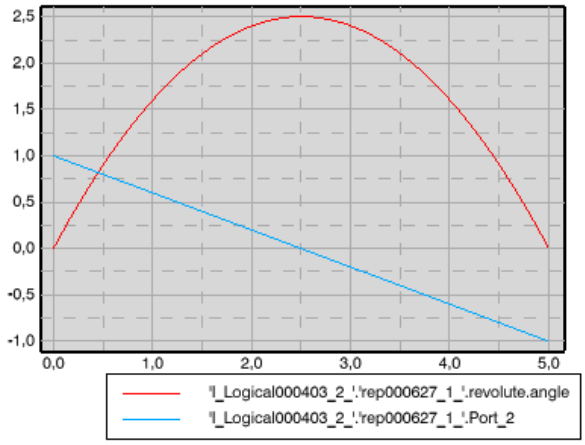


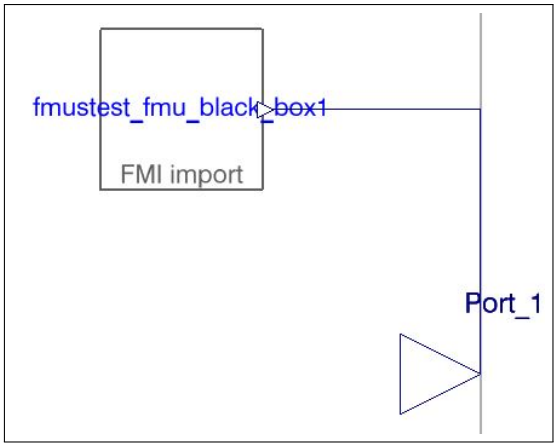
Abbildung 4.6: Auf der linken Seite ist der Aufbau des Zylindermodells in der RFLP-Umgebung zu sehen, rechts das der rechten Logical Reference hinterlegte DBM-Modell. In der linken LR ist die FMU importiert, die das Drehmoment vorgibt.

Die zweite Möglichkeit, eine FMU in CATIA zu importieren, findet sich in der Dynamic

Behavior Modeling Workbench selbst. Wie auch beim Export entspricht diese Funktion der von Dymola. Eine FMU kann importiert und in einem DBM-Modell instanziiert werden. Die Schnittstellen werden direkt als Modelica-Interfaces dargestellt und diese können mit dem restlichen Modell verbunden werden. Um die FMU in einer Simulation in der RFLP-Umgebung nutzen zu können, ist lediglich das gewohnte Vorgehen der Kopplung an logical ports notwendig. Auf diese Art lässt sich eine Simulation sowohl in der DBM- als auch in der FL-Workbench ausführen. Dieselbe Funktionalität, die der FMU-Import in der RFLP-Umgebung erzielt, kann so robuster umgesetzt werden. Bei diesem Vorgehen gibt es ebenfalls noch Probleme, so ist es nicht ohne Weiteres möglich eine importierte FMU durch eine andere zu ersetzen. Auch hier treten Fehler bezüglich der Instanziierung auf, die ein effizientes Arbeiten erschweren.



(a) Plotter Fenster in CATIA V6. Die rote Kurve beschreibt den Winkel des Zylinders in Radiant über der Zeit, die blaue das Drehmoment.



(b) DBM-Teilmodell, in dem eine FMU instanziiert ist, die mit einem logical port verbunden ist.

Abbildung 4.7: Plot eines Simulationsergebnisses und FMU Implementierung des Zylindermodells.

Um die Struktur der Co-Simulation mittels FMI zu demonstrieren, wird ein Beispiel herangezogen. Ein Modell eines bewegten Zylinders wird in CATIA umgesetzt, die Größe des auf den Zylinder wirkenden Moments wird von einer FMU erzeugt. Um zusätzliche CAD-Daten einzubinden, wird mit der RFLP-Umgebung gearbeitet, der FMI-Import erfolgt allerdings im DBM-Modell, wie zuvor beschrieben. In Abbildung 4.6 sind die Modelle in den jeweiligen Umgebungen zu sehen. Auf der linken Seite sieht man die Functional Logical Workbench, in der die logische Struktur vorgegeben ist. In der linken logischen Referenz wird die FMU hinterlegt, in der rechten befindet sich der in CATIA umgesetzte Anteil des Systems. Die Linien deuten an, dass das DBM-Modell der Logical

Reference hinterlegt ist. Darin wird der mechanische Teil des Gesamtmodells beschrieben. An der Weltkomponente befindet sich drehbar gelagert ein CATIA_3DPart Block, der den Zylinder darstellt. Im Revolute Joint wirkt ein rotatorischer Dämpfer der Bewegung entgegen, die durch ein im Ursprung wirkendes Moment verursacht wird. Die Größe des Moments wird von außerhalb bezogen, genauer gesagt von der FMU. Die Functional Mockup Unit wurde in Dymola erstellt und beinhaltet im Wesentlichen eine Rampe, die von 1 bis -1 innerhalb von fünf Sekunden ansteigt. Wie zuvor erwähnt wird die FMU nicht über die FMI-Anbindung der RFLP-Umgebung eingebunden sondern in einem DBM-Modell instanziiert. Die Verbindung zum Rest des Systems erfolgt klarerweise über logical connections. Die Verbindungen sind als gerichtete Ports ausgeführt, da dieses Verhalten zu den Modelica Interfaces passt. Das Ergebnis eines Simulationslaufs und die Implementierung der FMU in einem DBM-Modell sind in Abbildung 4.7 zu sehen.

Das Beispiel zeigt, wie mit CATIA V6 mittels FMI-Schnittstelle Co-Simulation mit Einbindung von CAD-Daten durchgeführt werden kann. Der Vorteil bei dieser Methode ist, dass das FMI bereits in CATIA implementiert ist und daher keine extra Software benutzt werden muss. Die beschränkte Anzahl an Simulatoren mit FMI-Schnittstelle beschränkt jedoch die Flexibilität in der Simulatorauswahl.

4.4. CATIA und BCVTB

Die zweite Methode, um CATIA einer Co-Simulation zugänglich zu machen, ist mit Hilfe des Building Controls Virtual Test Bed (BCVTB). Wie in Abschnitt 3.2 beschrieben gibt es bereits eine Kopplung von BCVTB und Dymola. Die Tatsache, dass die DBM-Umgebung praktisch eine Dymola Umgebung darstellt, kann daher genutzt werden. Die für die Kommunikation mit BCVTB notwendige Modelica Buildings Library kann ohne Probleme in CATIA importiert werden. Die Modellierung in der DBM-Umgebung erfolgt dann wie in Dymola. Die Herausforderung bei der Verbindung von CATIA mit BCVTB besteht darin, die Schritte zu finden, die nötig sind, um CATIA per Systemkommandos zu erreichen und die Simulation auszuführen.

Dafür wird die Infrastruktur der Kommunikation zwischen BCVTB und Dymola adaptiert und auf CATIA angewandt. Wie bereits erklärt geschieht dies durch ein Batch-File, das von BCVTB aufgerufen wird. Mithilfe des Files werden Dymola alle notwendigen Dateien bereitgestellt. Für das Vorgehen ist es zuerst wichtig zu verstehen, dass es in CATIA nicht möglich ist, Arbeitsverzeichnisse zu definieren, wie das in Dymola der Fall ist. Alle Dateien, die für Simulationen mit der DBM-Umgebung wichtig sind, werden im

Verzeichnis user/AppData/Local/DassaultSystemes/CATTemp/SVE abgelegt. Dies ist daher auch der Ordner, in dem CATIA beim Start einer Co-Simulation über BCVTB die dafür notwendigen Daten sucht. Zunächst werden die vier für die Socket-Kommunikation benötigten Dateien in das Arbeitsverzeichnis durch das Batch-Skript kopiert (es handelt sich um bcvtb.h, bcvtb_modelica.dll, bcvtb_modelica.exp und bcvtb_modelica.lib, die im BCVBT Installationsverzeichnis unter lib/modelica gefunden werden können). Zusätzlich müssen die Konfigurationseinstellungen für die Kommunikation mittels BSD-Sockets geteilt werden, dafür wird das socket.cfg File ebenfalls in das Arbeitsverzeichnis kopiert. Soweit die Parallele zu Dymola. Es zeigt sich allerdings, dass CATIA weitere C-Bibliotheken benötigt, um die für die Kommunikation mit BCVTB notwendigen Funktionen aufrufen zu können. Diese müssen daher ebenfalls in das Arbeitsverzeichnis gestellt werden, hier handelt es sich um die Dateien bcvtb.dll, zu finden im Ordner lib/util, und libexpat.dll in lib/windows/expat/lib. Mit diesen insgesamt sechs Files besitzt CATIA sämtliche Informationen, um die Kommunikation mit BSD-sockets durchzuführen. Unter Listing 2 ist der Quelltext des Batch-Files zu sehen.

```
1 @echo Copying files needed by CATIA
2 copy "C:\Program Files\bcvtb\lib\modelica\bcvtb.h" "C:\Users\user\AppData\
    Local\DassaultSystemes\CATTemp\SVE"
3 copy "C:\Program Files\bcvtb\lib\modelica\bcvtb_modelica.*" "C:\Users\user
    \AppData\Local\DassaultSystemes\CATTemp\SVE"
4 copy "socket.cfg" "C:\Users\user\AppData\Local\DassaultSystemes\CATTemp\
    SVE"
5 copy "C:\Program Files\bcvtb\lib\util\bcvtb.dll" "C:\Users\user\AppData\
    Local\DassaultSystemes\CATTemp\SVE"
6 copy "C:\Program Files\bcvtb\lib\windows\expat\lib\libexpat.dll" "C:\Users
    \user\AppData\Local\DassaultSystemes\CATTemp\SVE"
7 @echo Connecting to CATIA
8 catia.vbs
```

Listing 2: Sourcecode des Batch-Files für die Kommunikation mit CATIA, das von BCVTB aufgerufen wird.

Das beschriebene Vorgehen sorgt dafür, dass sämtliche gebrauchte Dateien am richtigen Ort abgelegt werden. Der Zugriff auf CATIA erfolgt aber nicht über das Batch-File selbst sondern über den Umweg eines Visual Basic Skripts. Zur Erinnerung, Dymola wird standardmäßig von einem Batch-File aufgerufen und mittels eines Modelica Skripts werden weitere Kommandos wie Start der Simulation übergeben. Bei Simulationen mit CATIA V6 ist es nicht sinnvoll, bei jedem Simulationsstart das Programm zu starten, da dieser Prozess selbst auf einem schnellen Computer mehrere Minuten dauern kann,

was die Simulationsdauer stark verlängert. Es wird daher ein Ansatz gewählt, auf eine bereits laufende CATIA Instanz zuzugreifen und dort die Simulation zu starten. Dies ist mit dem Component Object Model (COM) möglich. Die Technik wurde von Microsoft entwickelt und ist plattform- und programmiersprachenunabhängig, was sich für die Anwendung von Co-Simulation eignet. Es stellt sich heraus, dass CATIA die Kommunikation über die COM-Schnittstelle unterstützt, wobei CATIA selbst als COM-Server fungiert. Da die COM-Schnittstelle plattformunabhängig ist, kann sie ebenfalls dazu genutzt werden, nicht nur die Verbindung zwischen CATIA und BCVTB herzustellen sondern im selben Schritt auf die Makro Funktionalität zuzugreifen. Makros können in CATIA unter anderem mit Visual Basic Script (VBS) aufgesetzt werden. Diese Sprache wird daher auch für den Aufbau der Kommunikation mittels COM benutzt, im selben Schritt können dann Kommandos wie „Simulation starten“ an CATIA gesandt werden. In Listing 3 ist der Quelltext eines VBS zu sehen, das ein Objekt erstellt, das auf die laufende CATIA Instanz verweist.

```
1 Dim CATIA
2 Set CATIA = CreateObject("CATIA.Application")
3 CATIA.StartCommand "execute"
```

Listing 3: Sourcecode des Visual Basic Skripts, das vom Batch-File aufgerufen wird.

In weiterer Folge können Befehle übergeben werden, in diesem Beispiel wird lediglich die Simulation gestartet. Die Funktion „StartCommand“ ist in diesem Fall besonders hilfreich, da sich dadurch sämtliche Befehle ausführen lassen, die auch in CATIA über die Eingabezeile eingegeben werden können. Zusätzlich können sämtliche Fähigkeiten der Makros genutzt werden, denkbar ist es zum Beispiel, Dateien aufzurufen, zu schließen oder sogar Änderungen am Modell vorzunehmen. Der Aufruf des Visual Basic Scripts ist in der letzten Zeile des Batch-Files zu finden (siehe Listing 2). Damit sind sämtliche Glieder in der Kette zwischen BCVTB und CATIA vollständig und die Kommunikation kann stattfinden.

Zur Veranschaulichung wird ein einfaches Beispiel implementiert, das die Kommunikation zwischen CATIA und BCVTB zeigen soll. Das Modell besteht im Wesentlichen darin, dass in BCVTB ein Signal erzeugt, an CATIA geschickt und dort verstärkt und zur Auswertung wieder retourniert wird. In Abbildung 4.8 ist die Implementierung in BCVTB zu sehen. In der linken oberen Ecke ist der Synchronous Data Flow Director zu sehen, der die Simulation steuert. Die Simulationsparameter sind daneben als Variablen definiert. Das Anfangssignal wird als Rampe ausgeführt, der Startwert liegt bei 2000 und das Inkrement bei 100 pro Iteration. Im Simulator-Actor wird das Batch-File aufge-

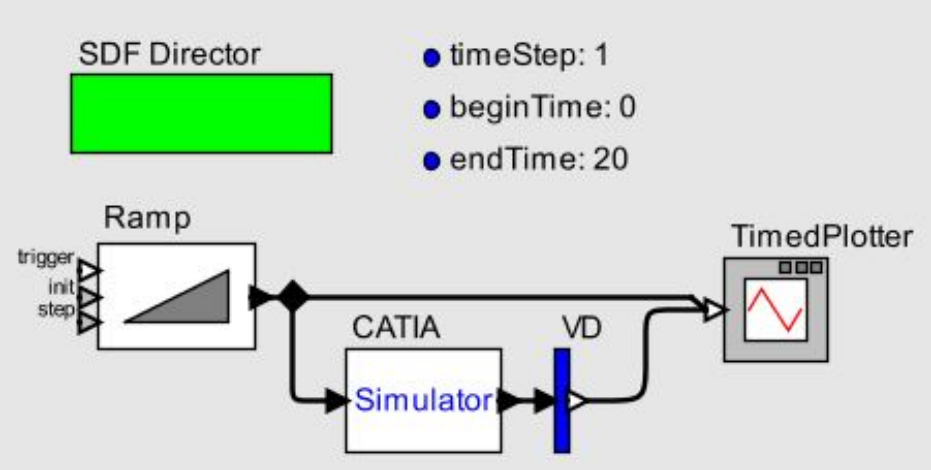


Abbildung 4.8: Testbeispiels zur Demonstration der Co-Simulation in BCVTB.

rufen, das wiederum über das VB-Skript die Simulation in CATIA anstößt. Über einen Vector Disassembler (VD) wird der Datenstrom für die Auswertung vorbereitet. Im *TimedPlotter* wird dann das ursprüngliche Signal mit dem Output aus CATIA verglichen. Abbildung 4.9 zeigt das simple Modell, das in CATIA implementiert ist. Der BCVTB Block steuert den Datenaustausch. Der Datenstrom wird in ein Gain-Element geschickt und dadurch verstärkt. Anschließend werden die Informationen wieder an BCVTB zurückgegeben. Zur Auswertung der Ergebnisse werden das Ein- und das Ausgangssignal aus CATIA im zeitlichen Verlauf dargestellt. Abbildung 4.10 zeigt das Output Fenster des TimedPlotter.

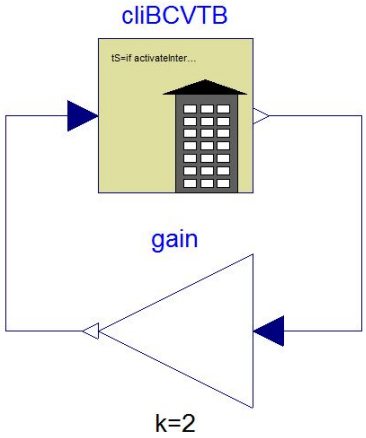


Abbildung 4.9: In CATIA DBM implementiertes Teilmodell des Testbeispiels. Die Daten werden von BCVTB erhalten, verstärkt und wieder retourniert.

Als rote Linie ist die Quelle aufgetragen. Die Initialisierung und die Steigung entspre-

chen den Vorgaben. Das durch CATIA modifizierte Signal ist als blau gepunktete Linie zu erkennen. Vorgegeben durch das Gain-Element im DBM-Modell wird das Rampensignal verdoppelt. Die Initialisierung des Ausgabewerts zum Startpunkt in CATIA ist mit null vorgegeben. Das in Abschnitt 3.4 beschriebene Verhalten ist auch in diesem Beispiel gut erkennbar. Der korrekte Wert hinkt der Simulation um einen Zeitschritt nach. Gut abzulesen ist dieses Phänomen am letzten Zeitschritt: Der doppelte Wert der Rampe beträgt 8000, dieser wird jedoch noch nicht erreicht, dafür wäre eine weitere Iteration erforderlich.

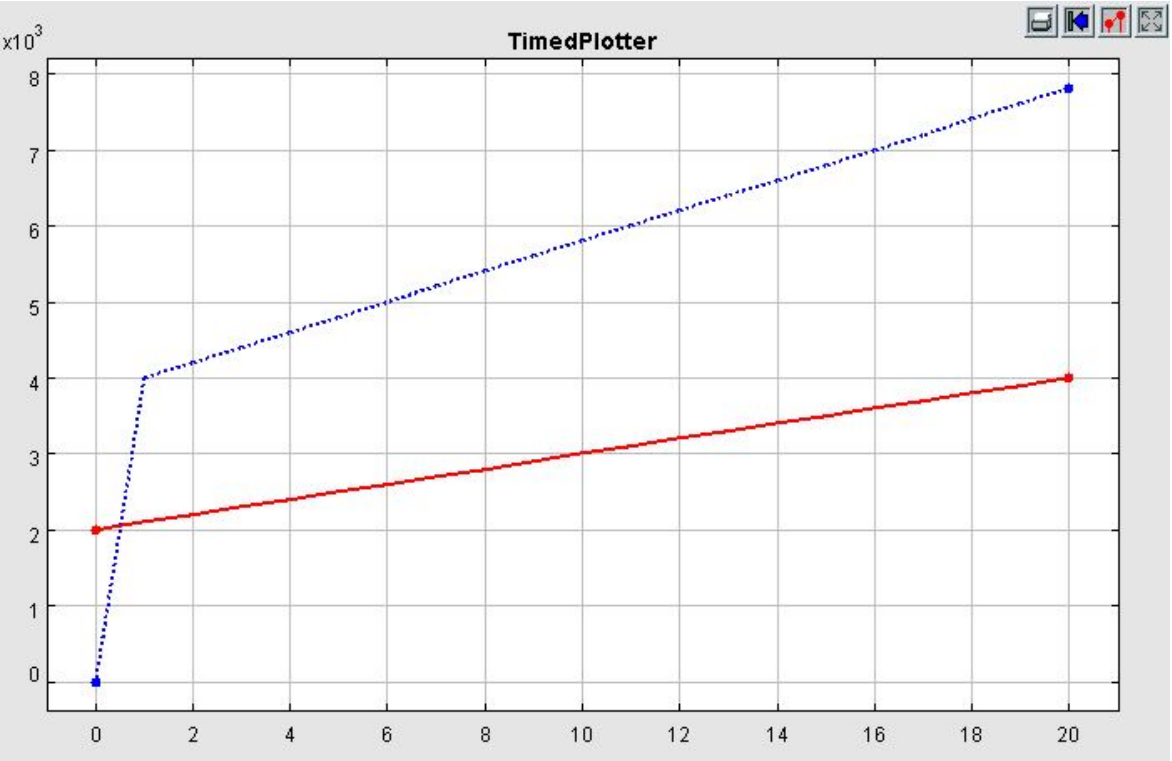


Abbildung 4.10: Output Fenster des TimedPlotter im BCVTB-Modell. Die rote Linie zeigt den Verlauf der Rampe, die blau gepunktete die Rückmeldung aus CATIA.

Die bisherigen Vorbereitungen zur Co-Simulation gehen von der Parallele zwischen Dymola und CATIA, also dem Dynamic Behavior Modeling, aus. Diese Ähnlichkeiten machen Co-Simulation mittels BCVTB möglich. Die Nutzung der speziellen Vorteile von CATIA, nämlich von CAD-Daten in einer Systemsimulation, sind aber in der DBM-Umgebung nicht möglich. Dafür muss der Übergang auf die Requirements, Functional, Logical, Physical (RFLP) Umgebung gemacht werden. Mit der Zuordnung von DBM-Modellen zu Logical References in der Functional Logical Workbench scheint dies leicht

umsetzbar zu sein. Die Realität sieht leider anders aus. Die Funktionalität der Kommunikation mit BCVTB ist bei der Simulation in der RFLP-Umgebung nicht erreichbar. Das Problem scheint darin zu bestehen, dass CATIA die C-Funktionen, die für die BSD-sockets notwendig sind, nicht korrekt aufrufen kann. Aufgrund der verschlossenen Architektur von CATIA ist es nicht möglich gewesen, tiefere Einblicke zu erhalten oder sogar eine Lösung für das Problem zu finden. Die Hoffnung besteht, dass dieses Problem in zukünftigen Versionen von CATIA behoben wird.

4.5. Schlussfolgerungen

Wie sich die Situation im Moment darstellt, ist es grundsätzlich möglich, CATIA V6 in eine Co-Simulation einzubinden. Damit ist ein wichtiger Schritt getan, um Systemsimulation der Auslegung mechanischer Komponenten in CATIA näher zu bringen. Nichtsdestotrotz fehlen gewisse Teile in der Kette, um die Vorteile wirklich nutzen zu können. Sowohl Dynamic Behavior Modeling als auch die RFLP-Umgebung stecken noch in den Kinderschuhen. Das zeigt sich vor allem darin, dass eine Vielzahl an Programmfehlern das Arbeiten erschwert. Aber auch die in der PLM-Lösung von CATIA angepriesenen Features sind noch nicht zur Gänze implementiert. Als wichtigstes Beispiel sei auf die Integration der CAD-Daten in der RFLP-Umgebung hingewiesen. Sinnvollerweise sollen vorhandene CAD-Modelle in diese Plattform integriert werden können. Dies ist allerdings noch nicht möglich, da spezielle Teile eigens für dieses Ziel erstellt werden müssen. Der Vorteil der Integration der zwei Gebiete CAD und Systemsimulation wird dadurch fast gänzlich zunichte gemacht, da ein großer Zusatzaufwand entsteht. Sowohl die Nutzung von FMI als auch die Anbindung mittels BCVTB ist zwar möglich, aber eingeschränkt. Das Functional Mockup Interface ist von den Features am flexibelsten, die Anwendung wird durch eine Flut an Programmfehlern allerdings zur Qual für den Anwender. Es besteht außerdem das grundsätzliche Problem von FMI, dass die Auswahl der Simulatoren von der Kompatibilität mit der Schnittstelle abhängt. Vor allem das Produkt MATLAB, das Dank der allgemeinen Nutzbarkeit und weiter Verbreitung von großer Bedeutung ist, kann auf diese Art nicht angebunden werden, da es keine FMI-Schnittstelle bietet. In weiterer Folge wird in dieser Arbeit die Co-Simulation mit FMI nicht weiterverfolgt. Die Kopplung mittels BCVTB präsentiert sich in der Nutzbarkeit wesentlich besser. Das Benutzen von Modelica-Komponenten in der DBM-Umgebung und die dadurch umsetzbare Co-Simulation funktioniert stabil und ohne große Probleme. Die fehlende Möglichkeit, dieses Vorgehen auch auf die RFLP-Umgebung zu übertragen, ist auch hier einschränkend. Dadurch, dass es aber ohnehin nicht möglich ist CAD Daten im gewohn-

ten Format in die Simulation zu integrieren, ist die Notwendigkeit, die RFLP-Umgebung zu nutzen gering. Der Vorteil, dass durch die Implementierung von Dymola eine Vielzahl an Unternehmen, die bereits CATIA nutzen, die Möglichkeit bekommen Systemsimulationen durchzuführen, ist aber weiterhin ungebrochen. Um die Vorteile der Anwendung hervorzustreichen, wird die Umsetzung einer Co-Simulation an einem Beispiel aus der Fahrzeugindustrie gezeigt.

5. Modellierung von Fahrzeugkomponenten in Modelica

Die vorgestellten Möglichkeiten der Co-Simulation sollen im Zuge eines größeren Anwendungsbeispiels demonstriert werden. Die Verwendung von CATIA V6 DBM legt nahe, dass sich dafür komplexe mechanische Systeme eignen.

Die in den folgenden Kapiteln vorgestellte Anwendung bezieht sich auf die Simulation eines Kraftfahrzeugs. Die mechanischen Komponenten werden mit Hilfe der Modelica-Sprache modelliert. Die Flexibilität der Co-Simulation wird dazu genutzt, datenbasierte Modelle in MATLAB zu implementieren und in weiterer Folge durch Einbeziehung von Stateflow das kontinuierliche Modell um eine diskrete Komponente zu erweitern. Dadurch entsteht eine hybride Co-Simulation unter der Einbeziehung von vier unterschiedlichen Programmen. Die detaillierte Vorstellung der Gesamtsimulation befindet sich in Abschnitt 6.

Für die Vorbereitung der Co-Simulation müssen die mechanischen Komponenten in Modelica abgebildet werden. In diesem Abschnitt werden die einzelnen Submodelle vorgestellt und die Überlegungen dahinter erklärt.

5.1. Radmodell

Eine der wichtigsten Komponenten des Systems Fahrzeug/Straße ist die Verbindung zwischen diesen beiden Teilen – das Rad. Hier treten das ruhende und das bewegte System in Wechselwirkung und dadurch wird der Vortrieb des Fahrzeugs erzeugt. Das Drehmoment aus dem Motor wirkt als Kraft über den Radius des Rades auf die Straße. Dadurch ergibt sich die Beschleunigung des Fahrzeugs. Durch die kinematische Kopplung von Rad und Straße ergibt sich der Zusammenhang zwischen Raddrehzahl und Geschwindigkeit.

In Abbildung 5.1 sind die Kräfte und Momente eingezeichnet, die auf das Rad wirken. Auf die Achse wirkt die Gewichtskraft F_G sowie die Trägheitskraft F_T, die die Beschleunigung der Aufbaumasse beinhaltet. Aufgrund des Motors wirkt über den Antriebsstrang das Antriebsmoment M_A auf das Rad. Auf der Straßenseite wirken Antriebskraft F_X und Radlast F_Y. Das Rad selbst führt sowohl eine translatorische als auch eine rotatorische Bewegung aus, die über die Haftbedingung an der Straße miteinander kinematisch gekoppelt sind.

In der Modelica Standard Library gibt es keine Komponente, die den oben beschriebenen mechanischen Eigenschaften entspricht. Die auf Antriebsstränge spezialisierte PowerTrain-Library bietet zwar mehrere Radmodelle an, diese sind aber nur sehr eingeschränkt nutzbar, da sie auf eindimensionalen mechanischen Verbindungen beruhen.

Für eine detaillierte und dadurch hinreichend komplexe Beschreibung des Problems ist es jedoch notwendig auf die MultiBody-Library zurückzugreifen. Daher wurde der Ansatz gewählt, das Radmodell von Grund auf zu entwickeln. Das Rad wird als eigene Komponente modelliert, die wegen des objektorientierten Ansatzes beliebig oft wiederverwendet werden kann.

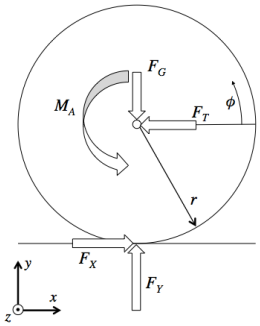


Abbildung 5.1: Kräfte und Momente auf das Rad.

Im ersten Schritt der Modellierung wird die Umsetzung des Drehmoments in eine Vortriebskraft und die Implementierung der kinematischen Kopplung in einer eigenen Komponente umgesetzt. Dieser Block kann später in ein Radmodell integriert werden. Durch diesen Ansatz ergibt sich nicht nur eine übersichtliche Struktur, vor allem die Erweiterbarkeit wird dadurch erleichtert.

Im zweiten Schritt werden Kontaktpunkte zur Umgebung definiert. Im betrachteten Fall sind das die Verbindung zur Seitenwelle, über die das Antriebsmoment übertragen wird, die Radaufhängung, an der die Karosserie über das Feder-Dämpfer-System angeschlossen ist, und die Auflagefläche des Reifens auf der Straße, die dem Momentanpol der Rollbewegung entspricht.

5.1.1. MultiBody-Konnektoren

Die Betrachtungsweise in Modelica basiert auf akausalen Verbindungen, die jeweils eine Potential- als auch eine Flussgröße beinhalten. Bei elektrischen Verbindungen ist das zum Beispiel Stromstärke als Fluss- und Spannung als Potentialgröße (genauer beschrieben in Abschnitt 2). Dieselbe Logik kommt bei der MultiBody-Library zum Tragen, wobei die Anzahl der Kenngrößen höher ist [5]. Konnektoren, oder auch Interfaces genannt, besitzen in der MultiBody-Library als Potentialgrößen Abstand und Winkel und als Flussgrößen Kraft und Drehmoment und werden als Frames bezeichnet (siehe Abbildung 5.2).

Jeder Frame besitzt einen relativen Lagevektor, der als r_0 bezeichnet wird und die Potentialgröße für die Translation darstellt. Dieser wird relativ zu anderen Frames derselben Komponente angegeben. Die zweite Potentialgröße, der Winkel, wird über die Verdrehung im Raum, oder auch Orientierung, angegeben. Die Orientierung wird mittels einer Transformationsmatrix T und der relativen Winkelgeschwindigkeit w angegeben, zusammen bilden sie die Orientation R. Auch hier handelt es sich um eine relative Rotation von einem Frame in einen anderen.

$$r\_0 = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}; R.T = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix}; R.w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

Die Flussgrößen sind Kraft- und Drehmomentvektoren, die auf den Frame wirken. Sie werden als f und t bezeichnet. Das interne Verhalten von Komponenten in der MulitBody-Library wird durch Beziehungen der Fluss- und Potentialgrößen an den Schnittstellen gestaltet. Es ist zu bedenken, dass nicht alle Größen vorgegeben werden können, da durch die Verbindung der Komponente mit der Umgebung Zustände der Interfacevariablen aufgezwungen werden. Ein überbestimmtes System kann nicht simuliert werden, selbst wenn sich die Zustände nicht ausschließen.

5.1.2. Modell der Kraftumsetzung

Für das Modell der Kraftumsetzung werden drei Modelica-Interfaces definiert, siehe Abbildung 5.2. Das erste, genannt $frame\_b$, beschreibt den Auflagepunkt von Reifen und Straße. Ein weiteres, $frame\_a$, stellt die Verbindung zu den Aufbaumassen dar. Über das dritte Interface $flange\_a$ wird die Antriebsleistung eingebracht. Da Kräfte und Momente in verschiedene Richtungen wirken, ist es sinnvoll die zwei ersten Interfaces als MultiBody-Frames umzusetzen. Da bei der Übertragung des Antriebs nur Moment und Verdrehwinkel von Interesse sind, ist es zweckmäßig, dieses Interface als eindimensional rotatorisches umzusetzen. Dieses Interface wir als Flange bezeichnet. Folgend werden die Zusammenhänge der Fluss- und Potentialgrößen beschrieben.

Die Lage der zwei MultiBody-Frames spielt bei der Kraftumsetzung keine Rolle. Es wird angenommen, dass sich beide Interfaces an derselben Stelle befinden. Im Kontext zum restlichen Radmodell befinden sich dementsprechend beide Frames an der Kontaktfläche mit der Straße. Bei der Flange-Schnittstelle ist aufgrund der benutzten rotatorischen Bibliothek keine Lage definiert.

Die Orientierung des Frames zwischen Rad und Straße ändert sich während der Si-

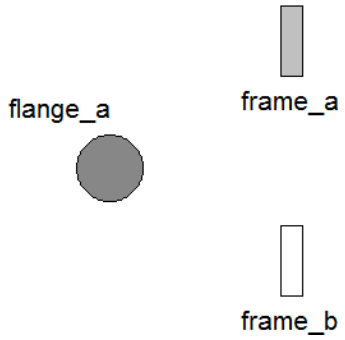


Abbildung 5.2: Das grafische Modell der Kraftumsetzung. Die MultiBody-Konnektoren $frame\_a$ und $frame\_b$ und der eindimensional rotatorische $flange\_a$.

mulation nicht. Dies wird nicht im Radmodell definiert, da der Frame die Orientation R durch die Verbindung mit der Straße erhält. Beim Integrieren des Rades in ein Modell ist es dadurch nicht möglich, die Komponente zu benutzen, ohne sie mit dem Boden zu verbinden. Die Orientierung der Radaufhängung wird mit der des Momentanpols gleichgesetzt. Dies ist notwendig, damit von der Radaufhängung ein Moment übertragen werden kann. Sollte das nicht erwünscht sein, können im Anschluss über ein Gelenk die Drehfreiheitsgrade freigegeben werden. Durch diesen Ansatz ist das Modell flexibel einsetzbar. Der Verdrehwinkel des Flanges, an dem der Antrieb angeschlossen ist, kann klarerweise nicht starr bleiben, da die Antriebswelle rotiert. Der Zusammenhang zwischen Winkel und Position der Straße stellt die kinematische Kopplung zum Boden dar.

$$flange\_a.\varphi = -\frac{frame\_b.r\_0[1]}{r}$$

Der Ausdruck $frame\_b$ bezeichnet den MultiBody-Frame, an dem die Kraftumsetzung stattfindet, $flange\_a$ ist die Verbindung zum Antrieb. Die Variable $frame\_b.r\_0[1]$ bezeichnet die x-Komponente des relativen Lagevektors, r ist der Radius des Rades, der dem Modell als Parameter mitgegeben wird. Das negative Vorzeichen ergibt sich aus der Orientierung des Winkels. Die Verdrehung in negativer φ-Richtung resultiert in einem positiven Versatz in x-Richtung. Mit Lagevektor und Orientierung der MultiBody-Frames und dem Winkel des Flanges sind die Abhängigkeiten der Potentialvariablen vollständig. Dadurch ist die kinematische Kopplung zwischen der Rotation und der translatorischen Bewegung des Rades vollzogen. Eine Änderung in der Position des Rades bewirkt ein Verdrehen desselben und vice versa.

Mit den bisherigen Informationen ist ein schlupffrei abrollendes Rad umgesetzt. Das ist jedoch für ein angetriebenes Rad nicht ausreichend, da zusätzlich das Antriebsmo-

ment auf die Straße übertragen werden muss. Dies wird über die Flussgrößen der Interfaces durchgeführt. Die Zusammenhänge zwischen den Frames werden als Kräfte- und Momentengleichgewichte festgelegt. Am Momentanpol $frame\_b$ soll der Kraftumsatz stattfinden. In y-Richtung müssen die Gewichtskraft und, im Falle der vertikalen Beschleunigung, Beschleunigungskräfte aufgenommen werden. Die Kräftebilanz mit den Größen aus Abbildung 5.1 ergibt

$$F_Y = F_G.$$

Das Kräftegleichgewicht in x-Richtung kann äquivalent aufgestellt werden:

$$F_X = F_T.$$

Das Momentengleichgewicht um die Radachse liefert eine Bedingung für F_X:

$$F_X = -\frac{M_A}{r}.$$

Zusammengesetzt ergibt sich

$$F_T = -\frac{M_A}{r}. \tag{5.1}$$

Um diese Beziehungen zu implementieren, muss man betrachten, wie das Rad mit dem Boden verbunden werden kann. Der Boden bildet den Absolutpunkt der gesamten Simulation und ist dementsprechend mit der Welt-Komponente verankert. Es ist also notwendig, dem Rad die Möglichkeit zu bieten sich im Vergleich zur Umgebung zu bewegen. Dies wird im Fahrwegmodell umgesetzt, das detailliert in Abschnitt 5.3 beschrieben wird. Die einfachste Umsetzung der Relativbewegung ist über ein prismaticJoints, das die x-Richtung als Freiheitsgrad entsperrt. Damit ist es aber nicht möglich, dass der Reifen Kräfte in die Bewegungsrichtung übertragen kann, die Reaktionskraft F_X wird daher null. Diesen Umstand kann man sich zunutze machen, indem Gleichung (5.1) formal umgeschrieben wird zu

$$0 = F_T + \frac{M_A}{r}.$$

Mit $F_X = 0$ kann der Kraftumsatz konsistent angegeben werden:

$$\begin{pmatrix} F_X \\ F_Y \end{pmatrix} = \begin{pmatrix} F_T \\ F_G \end{pmatrix} + \begin{pmatrix} \frac{M_A}{r} \\ 0 \end{pmatrix}.$$

Die Auflagerreaktionen F_X und F_Y sind jene, die an der Verbindung zwischen Straße und Rad greifen, also in $frame\_b$. Die Kräfte aufgrund von beschleunigten Massen wirken

am zweiten MultiBody Interface, genannt $frame\_a$. Dadurch, dass F_X auf konstant null gehalten wird, kann die Verbindung zwischen Antriebsmoment und Massen wirken. Das umgesetzte Drehmoment erzeugt in weiterer Folge eine translatorische Beschleunigung der Aufbaumassen, was die Bewegung des Fahrzeugs darstellt.

Zuletzt wird vorgegeben, dass an $frame\_b$, also zwischen Straße und Rad, keine Momente übertragen werden können. Das Interface $frame\_a$ kommt ohne weitere Definitionen aus. Hier ist anzumerken, dass $frame\_a$ Momente aufnehmen kann. Diese Tatsache geht einher mit der Vorgabe, dass sich dessen Orientierung nicht ändern darf. Betrachtet man den Spezialfall eines einzigen Rades, an dem eine Last mit Schwerpunkt nicht in der Radachse an der Aufhängung montiert ist, treten Schwierigkeiten auf. In der Realität würde diese Konstruktion nicht in der Lage sein, stabil zu stehen. Da im Modell an der Radaufhängung allerdings die Momente der Last aufgenommen werden können, befindet sich das Modell in einem stabilen Zustand (siehe Abschnitt 5.1.4). Diese Tatsache ist für die Simulation von Fahrzeugen mit zwei oder mehr Rädern nicht von Bedeutung, sollte aber bei Anwendungen mit nur einem Rad in Betracht gezogen werden. In Listing 4 ist der Modelica-Quelltext zu sehen. Es ist zu erwähnen, dass die Vorzeichen von denen der Herleitung abweichen, da Modelica eine andere Vorzeichenkonvention verwendet.

```
1  model basicWheel
2    parameter Modelica.SIunits.Length r=0.5 "Wheel radius";
3    Modelica.Mechanics.MultiBody.Interfaces.Frame_b frame_b
4    Modelica.Mechanics.MultiBody.Interfaces.Frame_a frame_a
5    Modelica.Mechanics.Rotational.Interfaces.Flange_a flange_a
6  equation
7    Connections.branch(frame_b.R, frame_a.R);
8    flange_a.phi = -frame_b.r_0[1]/r;
9    frame_a.r_0 = frame_b.r_0;
10   frame_a.R = frame_b.R;
11   frame_b.f = -frame_a.f+{flange_a.tau/r,0,0};
12   frame_b.t = zeros(3);
13 end basicWheel;
```

Listing 4: Quelltext des Kraftübertragungsmodells in Modelica.

5.1.3. Erweitertes Radmodell

Die einfachste Realisierung eines Radmodells ist mit dem Modell der Kraftübertragung leicht umzusetzen. Zuerst müssen die jeweiligen Interfaces definiert werden. Diese entsprechen jenen des Submodells, also ein Frame bei der Kontaktfläche, ein Frame, der der

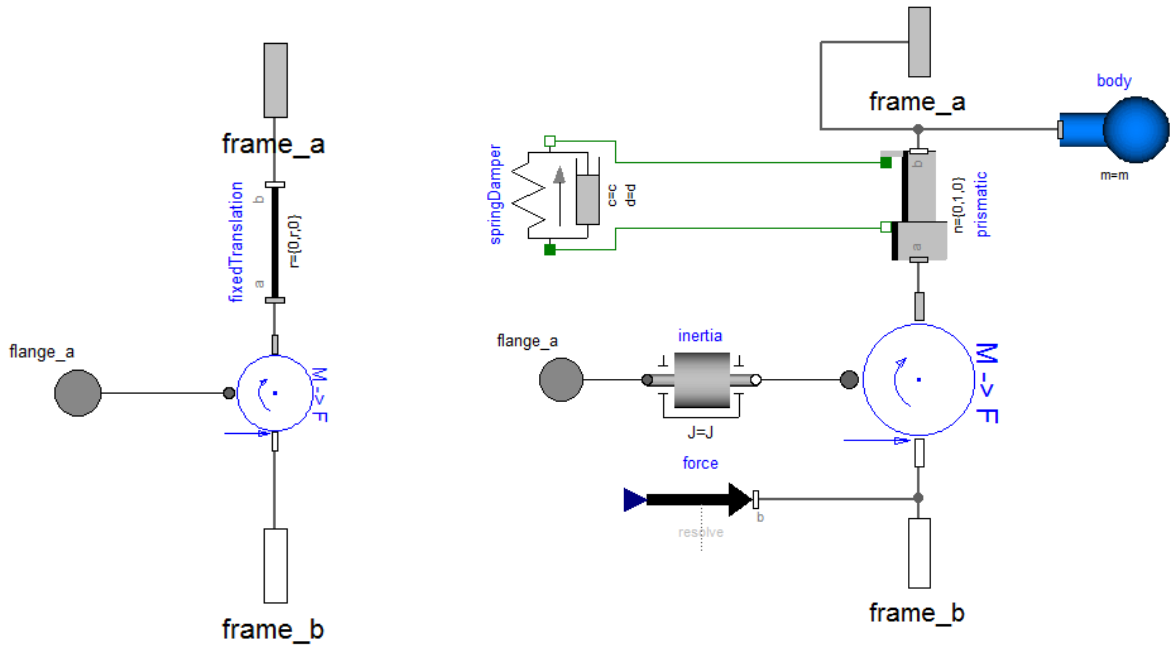
Radaufhängung entspricht, und der Flange zur Übertragung des Drehmoments. Sinnvollerweise setzt die Radaufhängung in der Radachse an, dementsprechend muss dieser Frame mit einem fixedTranslation-Element um den Radius nach oben versetzt werden. Damit ist ein angetriebenes, abrollendes Rad umgesetzt, siehe Abbildung 5.3 (a). Um zusätzliche Eigenschaften eines Rades abzubilden, soll dieses Modell erweitert werden (Abbildung 5.3 (b)), was Dank der gewählten Hierarchie leicht möglich ist.

Zuerst sollen noch die elastischen Eigenschaften des Reifens abgebildet werden. Dies kann dadurch erreicht werden, indem das zuvor eingeführte fixedTranslation-Element durch ein prismaticJoint ersetzt wird. Damit wird der Freiheitsgrad in y-Richtung freigegeben. An die Hilfsinterfaces des Gelenks kann ein lineares Feder-Dämpfer-Element angebracht werden, der Wert für die Steifigkeit wird als c, der für die Dämpfung als d bezeichnet. Durch die leichten Modifikationen wird der gewünschte Effekt erzielt, dem Rad elastische Eigenschaften zu geben. Damit ist genau genommen der Radius des Rades nicht mehr konstant. Da bei der Modellierung ein schlupffreies Rollen angenommen wird, wird auch der veränderliche Radius bei der Kraftübertragung nicht berücksichtigt. Als zusätzliche Erweiterung werden Masse m_R und Massenträgheit des Rades J_R hinzugefügt. Auch dies ist einfach durch die Addition einer Punktmasse in der Radachse und eines Inertia-Elements am Flange machbar.

Als dritte und aufwendigste Detaillierung des Modells wird der Einfluss des Rollwiderstands implementiert. Er wirkt aufgrund der Energiedissipation bei der dynamischen Abplattung des Reifens beim Rollen und tritt als Kraft entgegen der Bewegungsrichtung in Erscheinung. Die Kraft kann über ein force-Element hinzugefügt werden, das an der Kontaktfläche zwischen Straße und Rad angreift. Die Berechnung der Amplitude wird textuell implementiert. Als Ausgangsbasis wird zur Bestimmung der geschwindigkeitsabhängige Rollwiderstand nach [16] herangezogen:

$$F_R = fF_Z; f = f_0 + Kv^2. \tag{5.2}$$

Der Rollwiderstand F_R hängt direkt mit der Vertikalkraft F_Z zusammen. Der Faktor f berechnet sich über einen konstanten Wert f_0 und einem Term, der über einen Koeffizienten K von der Geschwindigkeit v abhängt. Die Gleichung (5.2) wird zur Berechnung leicht angepasst. Erstens erzeugt die Konstante f_0 auch bei stillstehendem Rad eine Widerstandskraft, was unphysikalisch ist. Die Gleichung (5.2) gilt daher eigentlich nur bei $v \neq 0$, was zu einer Unstetigkeit führen würde. Um dieser Tatsache entgegenzuwirken, wird auch f_0 um einen geschwindigkeitsproportionalen Anteil erweitert. Dafür wird die



(a) Einfaches Modell des Rades.

(b) Erweitertes Modell des Rades.

Abbildung 5.3: Einfaches und erweitertes Radmodell.

Arcustangens-Funktion benutzt, die mit $\pi/2$ normiert wird. Als Konsequenz ist bei kleinen Geschwindigkeiten die Widerstandskraft auch klein. Der Arkustangens konvergiert rasch gegen $\pi/2$ und der geschwindigkeitsabhängige Term verliert den Einfluss.

Zweitens geht über das Quadrat der Geschwindigkeit deren Richtung verloren. Dadurch wird das Quadrat ersetzt durch das Produkt aus Geschwindigkeit und Betrag der Geschwindigkeit. Die erweiterte Beziehung (5.2) ist daher:

$$F_R = fF_Z; f = f_0 \frac{\arctan(v)}{\pi/2} + Kv|v|. \tag{5.3}$$

Zum Vergleich sind in Abbildung 5.4 die Funktionen (5.2) in blau und (5.3) in rot aufgetragen.

Das Verhalten des Rades ist somit um drei Eigenschaften erweitert worden, wie in Abbildung 5.3 zu sehen ist. Auf ähnliche Art kann das Modell weiter ausgestaltet werden. Diese Vorgehensweise zeigt die in Abschnitt 2 vorgestellten Vorteile der objekorientierten Modellierung.

Zum Abschluss ist auch die grafische Komponente der Modellerstellung nicht zu ver-

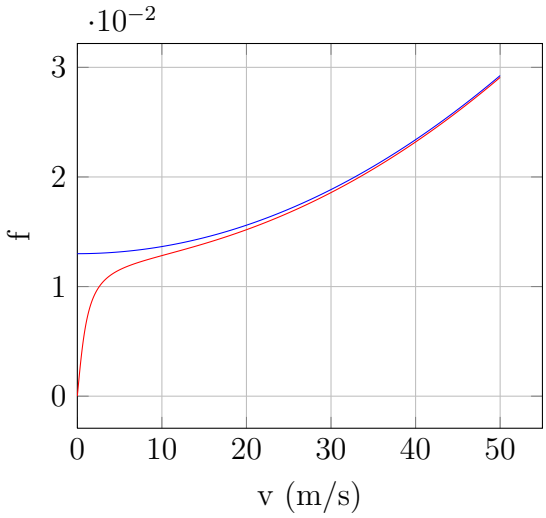
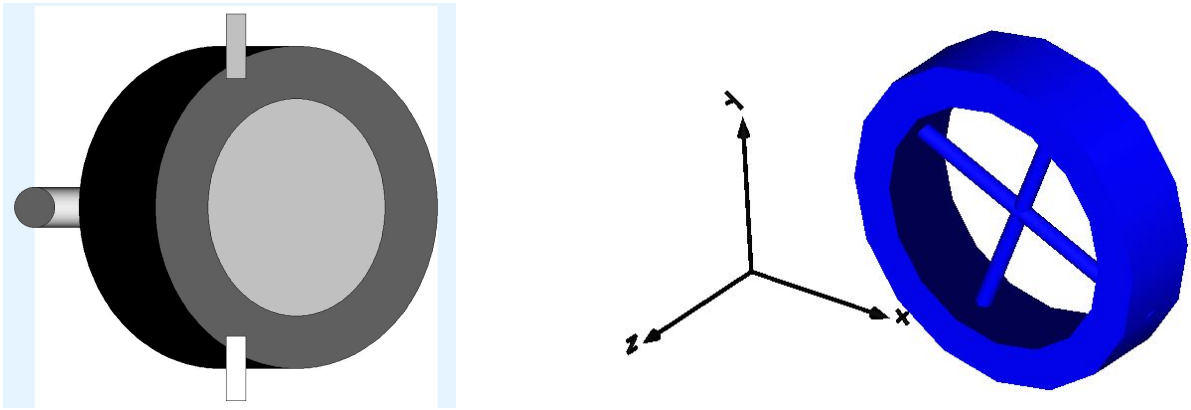


Abbildung 5.4: Vergleich der Funktionen des Rollwiderstands, (5.2) in Blau und (5.3) in Rot.



(a) Das Icon bei der Modellierung.

(b) Darstellung bei der Animation.

Abbildung 5.5: Repräsentationen des Rades.

nachlässigen. Daher wurde sowohl eine grafische Repräsentation im Modell als auch eine während der Animation erstellt. Das Icon, wie in Abbildung 5.5 (a) zu sehen, wird angezeigt, wenn eine Instanz des Rades in einem Modell verwendet wird. In Grau beziehungsweise Weiß sind die drei Interfaces zu erkennen. An der Unterseite befindet sich der Frame, der als Verbindung zwischen Rad und Straße fungiert, oben ist die Radaufhängung angebracht. Die Verbindung zum Antrieb wird mit einem grauen Zylinder verdeutlicht, der eine Welle darstellen soll. Das zweite Bild zeigt die Animation des Reifens. Hier wurde darauf geachtet, dass die Rotation des Reifens gut erkennbar ist.

5.1.4. Anwendungsbeispiel des Radmodells

Um zu veranschaulichen, wie das Rad in eine Simulation eingebaut werden kann, ist es sinnvoll, ein einfaches Beispiel zu betrachten. Im Fokus soll das Rad an sich stehen, dementsprechend sind die restlichen Komponenten sehr vereinfacht abgebildet. In Abbildung 5.6 ist das Modelica-Modell zu sehen. Als zentraler Punkt zeigt sich das Rad.

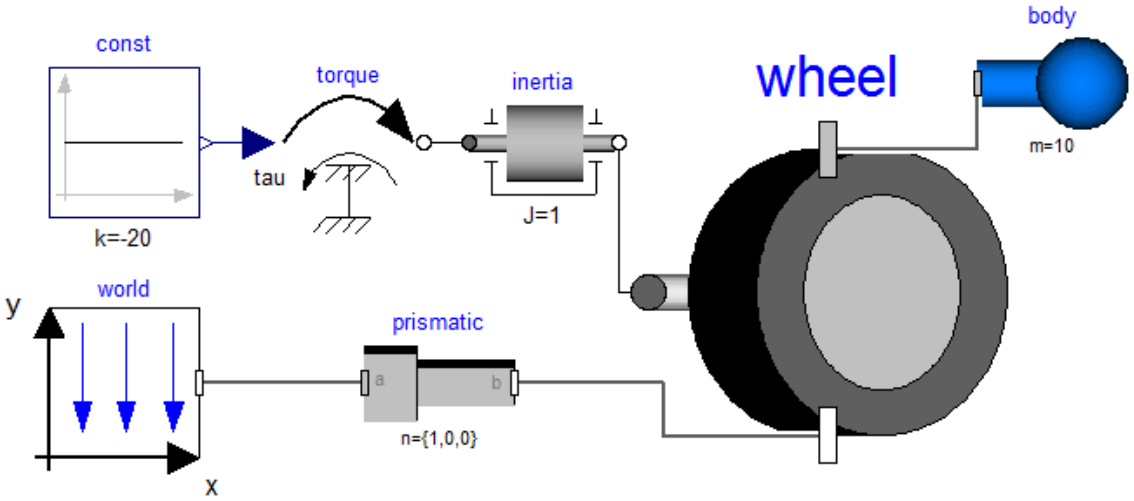
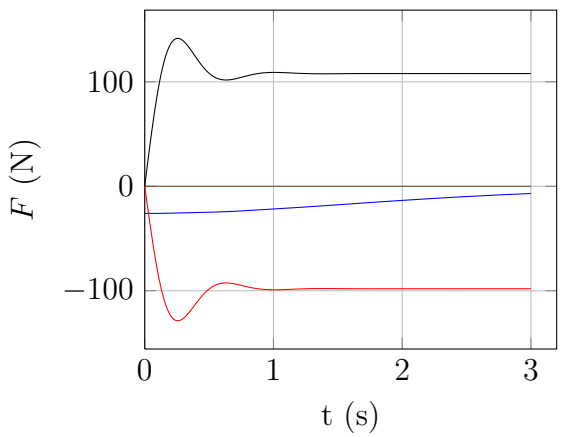


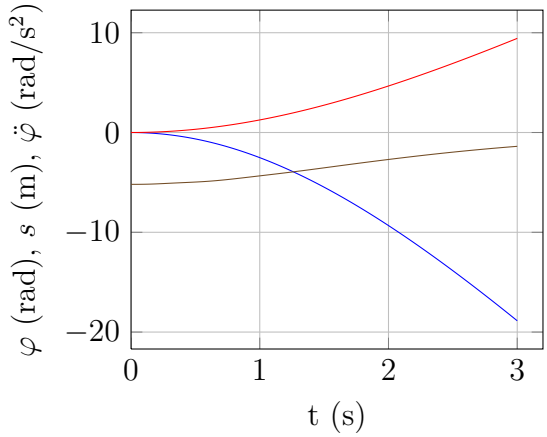
Abbildung 5.6: Modelica-Modell zum Test des Rades.

Um das System zu simulieren, ist es notwendig den Frame auf der Unterseite des Rades, also $frame\_b$, mit der Straße zu verbinden. Dies ist der einzige Konnektor, der verbunden werden muss, und wo in weiterer Folge mit der World-Komponente der Bezug zum absoluten Welt-Koordinatensystem hergestellt wird. Dazwischen muss sich zumindest ein prismaticJoint befinden, damit sich das Rad relativ zur Umwelt in x-Richtung bewegen kann. Das ist der Minimalaufbau des Bodens. Die World-Komponente definiert den absoluten Nullpunkt des Systems und die Richtung der Gravitation, hier in negativer y-Richtung gewählt. Am Frame der Radaufhängung $frame\_a$, der am Bild am oberen Ende des Reifens abgebildet ist, ist ein Körper befestigt, der die Aufbaumasse des Fahrzeugs darstellt. Die Kenngrößen sind mit der Masse $m_C = 10kg$ und dem Vektor von Aufhängung zum Schwerpunkt $r_C = (-1, 1, 0)^T m$ angegeben. Diese Geometrie wurde so gewählt, um zu demonstrieren, dass das dabei entstehende Drehmoment tatsächlich von der Radaufhängung aufgenommen wird und daher keinen Einfluss auf $frame\_b$ hat. Am rotatorischen Interface greift das Antriebsmoment an. Am Flange ist eine rotierende Masse in Form eines eindimensionalen Inertia-Elements angebracht, um das Trägheitsmoment einer Welle abzubilden. Die Massenträgheit wird mit $J_W = 1kgm^2$ angenommen. An dieser Welle greift ein Drehmoment von $M = 20Nm$ an, das das Rad

antreibt. Das negative Vorzeichen ergibt sich dadurch, dass für einen Vortrieb in positiver x-Richtung ein negatives Moment um die z-Achse aufgebracht werden muss. Die Kenngrößen des Rades werden mit $r = 0.5m$, $m_R = 1kg$, $J_R = 0.1kgm^2$, $c = 1000N/m$, $d = 100Ns/m$, $K = 0.01$ und $f_0 = 10^{-6}$ angegeben.



(a) Kräfte an der Radaufhängung und der Kontaktfläche zum Boden, in x-Richtung: blau F_T, braun F_X und in y-Richtung: rot F_G, schwarz F_Y.



(b) Winkel des Rades in Radiant (blau), zurückgelegte Weglänge in Meter (rot) und Winkelbeschleunigung in rad/s^2 (braun).

Abbildung 5.7: Ergebnisse der Simulation des Anwendungsbeispiels des Rades.

In Abbildung 5.7 (a) sind neben den Kräften in x-Richtung auch die vertikalen Kraftkomponenten eingezeichnet. An den Kurven ist der Einfluss des Feder-Dämpfer-Elements leicht erkennbar. Im eingeschwungenen Zustand ist zu sehen, dass die Werte betragsmäßig nicht gleich groß sind. Das liegt daran, dass an der Radaufhängung, also $frame\_a$, das Gewicht des Rades nicht enthalten ist. An $frame\_b$ wirkt im Gegensatz dazu die gesamte Masse.

In Abbildung 5.7 (b) sind Winkel, Ortskoordinate und die Winkelbeschleunigung abgebildet. In der dreisekündigen Simulation hat das Rad 9.43 Meter zurückgelegt, der entsprechende Winkel beträgt $-18.87rad$ oder $-1081°$ beziehungsweise drei Umdrehungen.

Auf dem linken Bild ist zu erkennen, wie die Kräfte an den Frames des Reifens in x- und y-Richtung wirken. Wie vom prismaticJoint gefordert, wirken an $frame\_b$ keine Kräfte in Bewegungsrichtung. Dies ist passend bezüglich der oben beschrieben Funktionsweise des Radmodells. An $frame\_a$ wirkt eine Kraft entgegen der Beschleunigung und verstärkt durch die Masse des Aufbaus von $m_C = 10kg$ und des Rades von $m_R = 1kg$ wirkt demnach zum Zeitpunkt $t = 0$ eine Kraft von $F_{CX} = -26N$. Um den Wert nach-

vollziehen zu können, benötigt man die Beschleunigung des Gefährts, die man über die Winkelbeschleunigung erhält. Diese errechnet sich über den Drallsatz in folgender Weise:

$$J_{ges}\ddot{\varphi} = M_A - F_X r,$$

wobei J_{ges} das gesamte Massenträgheitsmoment bezeichnet, im Beispiel ist $J_{ges} = J_R + J_W = 1.1kgm^2$. Es ist zu erwähnen, dass implizit angenommen wird, dass die Rotation um eine Trägheitshauptachse der Welle stattfindet. Durch die eindimensionale Betrachtungsweise ist dies auf natürliche Weise gegeben.

Die Reaktionskraft F_X entspricht im Prinzip der Reibungskraft, die der translatorischen Beschleunigung entgegenwirkt. Dementsprechend kann man die Kraft angeben als

$$F_X = m_{ges}a.$$

wobei m_{ges} die gesamte bewegte Masse ist und a die Beschleunigung. Nimmt man weiters die Beziehung $\ddot{\varphi} = \frac{a}{r}$ zur Hand, kann man die Winkelbeschleunigung berechnen:

$$\ddot{\varphi} = \frac{M_A}{(J_{ges} + m_{ges}r^2)} = -5.1948rad/s^2.$$

Der Wert der berechneten Winkelbeschleunigung stimmt mit dem der Simulation überein. Die Beschleunigung ist weiter $a = \ddot{\varphi}r = -2.5974m/s^2$. Die Kraft am $frame\_a$ in x-Richtung ergibt $F_X = m_C a = -25.974N$ und bestätigt damit die Simulation. Für Zeitpunkte $t > 0$ weichen die Werte von den Berechneten ab, da aufgrund des Rollwiderstands die Beschleunigung abnimmt. Dieser Einfluss ist in der Abbildung 5.7 sowohl bei der Winkelbeschleunigung als auch bei der Trägheitskraft sichtbar. Es ist anzumerken, dass im Modell $F_X = 0$ gilt, bei der analytischen Berechnung die Kraft allerdings nicht verschwindet. Es ist daher zu sehen, dass durch die gewählte Modellierung die Größen richtig berechnet werden.

Das Anwendungsbeispiel soll einerseits zeigen, wie das vorgestellte Modell einzusetzen ist, wie es funktioniert und wo die Grenzen liegen. Andererseits ist auch die Überprüfung der Plausibilität durch den Vergleich mit analytischen Lösungen wichtig. Es konnte gezeigt werden, dass das Radmodell den Anforderungen des Einsatzes in einer Gesamtfahrzeugsimulation entspricht und die Werte plausibel erscheinen. Die nächste Aufgabe ist, die an diesem zentralen Bauteil gekoppelten Elemente passend aufzubauen.

5.2. Fahrzeugmodell

Die nächste Komponente, die betrachtet wird, ist die Aufbaumasse des Fahrzeugs. Die Aufgabe des Antriebs besteht darin, das Fahrzeug zu beschleunigen. Die Trägheitskraft der Fahrzeugmasse sowie verschiedene Verlustgrößen wirken gegen diese Bewegung. Für die Berechnung des Rollwiderstands sind die Radlasten entscheidend. Auch die Kraft, die sich durch das Auf- oder Abwärtsfahren ergibt, hängt von der Masse des Fahrzeugs ab. Zusätzlich soll auch der Luftwiderstand der Karosserie berücksichtigt werden.

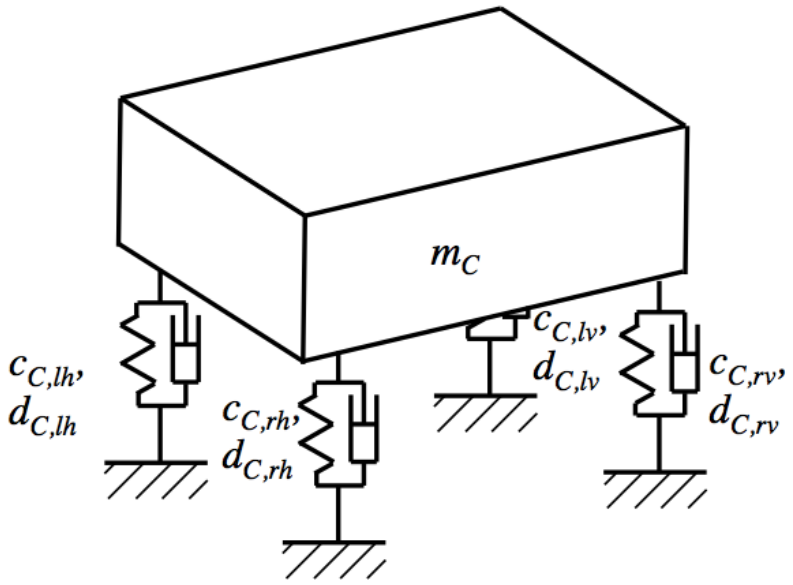


Abbildung 5.8: Starrkörpermodell des Fahrzeugaufbaus mit Massen m, Steifigkeiten der Aufbaufedern c und Dämpfungen der Aufbaudämpfer d. Indices: C... Chassis, r... rechts, l... links, h... hinten, v... vorne.

Abbildung 5.8 zeigt schematisch das Starrkörpermodell eines Fahrzeugs. Die Aufbaumassen werden in m_C zusammengefasst. Sie ruhen auf den Aufbaufedern und sind mit den vier Rädern verbunden, ausgedrückt durch die Auflager. Die Indices zeigen die Position an, mit l und r für links und rechts und v und h für vorne und hinten. Das Fahrzeugmodell muss lediglich das Chassis und die Aufbaufedern berücksichtigen, die Räder mitsamt Elastizitäten sind bereits im Radmodell enthalten. In Abbildung 5.9 ist die Implementierung des Fahrzeugmodells in Modelica dargestellt. Wie durch Abbildung 5.8 angedeutet, wird der Aufbau voll dreidimensional gestaltet. Das Koordinatensystem wird wie bei der Beschreibung des Radmodells mit x in Fahrtrichtung, y vertikal und z quer zur Bewegung festgelegt. Für die Länge in x-Richtung sind die zwei Body-Blöcke zuständig. Der linke der beiden stellt die Distanz zwischen Hinterachse und Schwer-

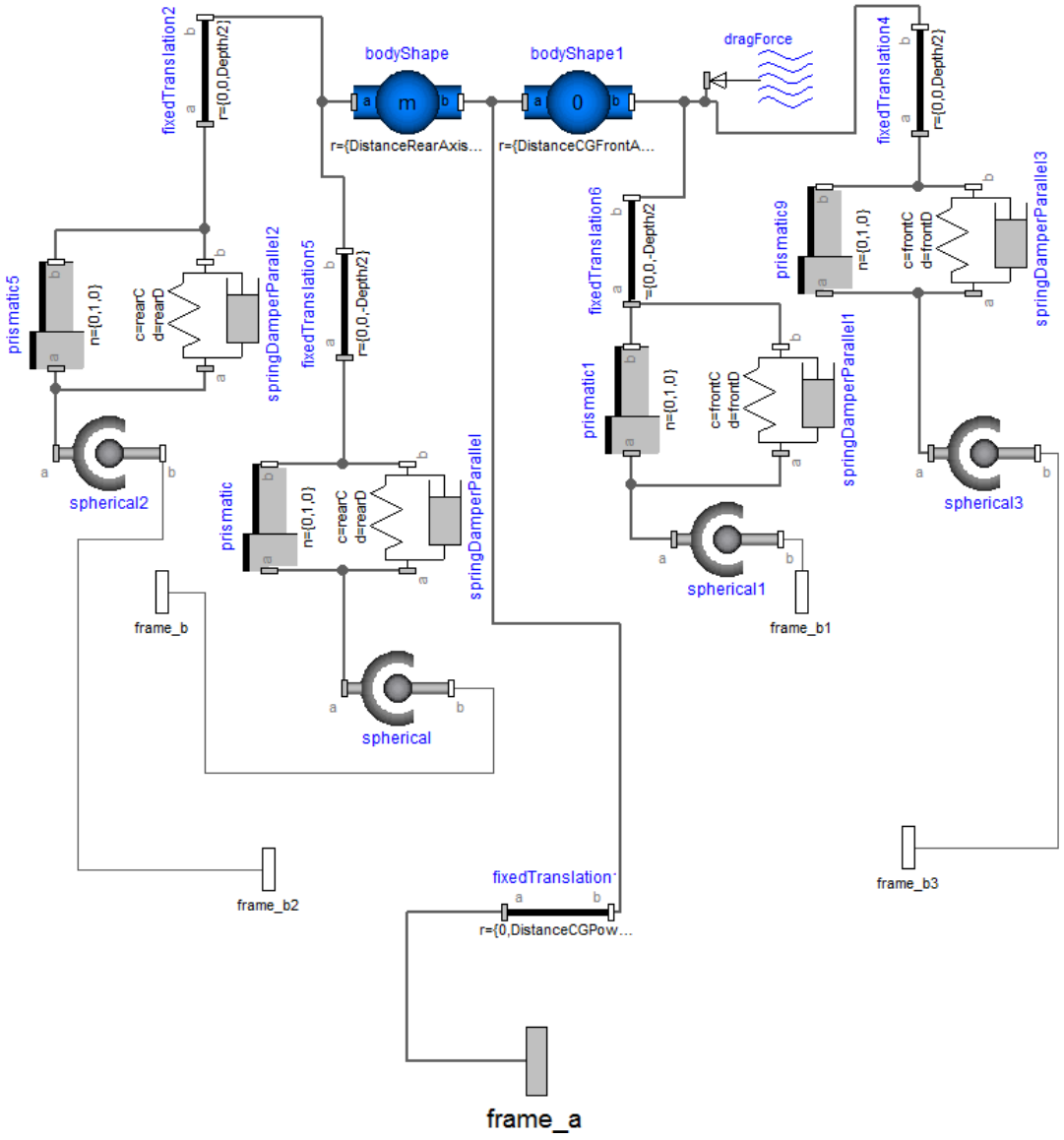


Abbildung 5.9: Starrkörpermodell des Fahrzeugaufbaus in Modelica.

punkt dar. In diesem Element wird auch die Masse des Fahrzeugs definiert. Der zweite Body-Block ist masselos und führt vom Schwerpunkt des Fahrzeugs bis zur Vorderachse. Die Aufteilung in zwei Blöcke ist notwendig, da im Schwerpunkt auch die Verankerung des Antriebsstrangs angreift. Wäre das nicht der Fall, würde die Lage des Schwerpunkts dementsprechend verfälscht. Außerdem ist es dadurch möglich, auch komplexe Geometrien der Aufbaumassen umzusetzen und ebenfalls im Schwerpunkt angreifen zu lassen. Die Aufhängung ist als Interface $frame\_a$ ausgeführt. Falls gewünscht kann der Antriebsstrang vertikal versetzt befestigt werden. Diese Möglichkeit wird mit einem

fixedTransaltion-Element geschaffen, das zwischen Schwerpunkt und Frame angebracht ist.

Die Länge in z-Richtung wird über vier fixedTranslation-Blöcke berücksichtigt. Ihre Länge entspricht jeweils der halben Tiefe des Fahrzeugs. An ihren Enden greifen die Aufbaufedern an. Sie sind ausgeführt als Feder-Dämpfer-Elemente, die mit einem prismaticJoint parallel geschalten sind. Das Gelenk hat die Funktion, dass die Aufbaufeder immer normal zur Karosserie steht. Vor die Verbindung mit den Rädern ist ein Kugelgelenk geschaltet, um die Übertragung eines Drehmoments zu verhindern und dadurch ein Sperren dieser Freiheitsgrade zu vermeiden. Die Verbindung zur Radaufhängung ist mit vier $frame\_b$ ausgeführt. Die Aufbaufeder stellt damit die Länge in y-Richtung dar und wird durch die ungedehnte Länge der Feder definiert. Diese muss derart angepasst werden, dass mit den jeweiligen Steifigkeiten und der Masse die gewünschte Schwerpunktshöhe erreicht wird. An einem der vier prismaticJoints wird die Anfangsbedingung der Höhe vorgegeben. Es ist zu bedenken, dass Längen im Fahrzeugmodell nur relativ angegeben werden können. Dadurch kann die zusätzliche Höhe des Schwerpunkts durch den Radius des Rades nicht berücksichtigt und muss manuell aus den Werten herausgerechnet werden.

5.2.1. Luftwiderstand

Eine der größten Verlustkomponenten bei einem bewegten Fahrzeug ist der Luftwiderstand. Der Luftwiderstand ist eine Kraft, die entgegen der Bewegungsrichtung aufgrund von verschiedenen Phänomen wie Fluidreibung, Beschleunigung der Luft und Verwirbelungen wirkt. Die Form des Körpers hat einen großen Einfluss auf den Verlust und Zahlenwerte können praktisch nur empirisch bestimmt werden. Der Einfluss wird im Normalfall durch die dimensionslose Kenngröße des Strömungswiderstandskoeffizienten c_W angegeben [17]. Dieser ist definiert durch die Gleichung

$$c_W = \frac{F_W}{\rho \frac{v^2}{2} S},$$

mit F_W als die Widerstandskraft, S die angeströmte Oberfläche, ρ die Dichte des Fluids und v die Anströmgeschwindigkeit. Über diese Definition von kann mit den passenden Zahlenwerten der Luftwiderstand berechnet werden. Abbildung 5.10 zeigt das grafische Modell. An der Verbindungsstelle $frame\_a$ wird die Geschwindigkeit gemessen. An dieser Stelle wird im Endeffekt auch die Kraft aufgebracht. Die Berechnung selbst passiert in textueller Form. Der Quelltext ist in Listing 5 zu sehen.

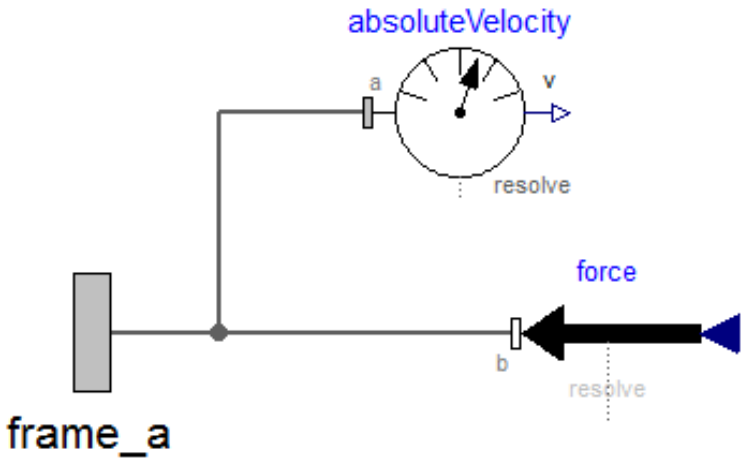


Abbildung 5.10: Grafisches Modell zur Berechnung des Luftwiderstands in Modelica.

Neben den Parameterdefinitionen ist vor allem die Zeile mit der Berechnung der Kraft interessant. Es ist zu sehen, dass die Geschwindigkeit nicht wie gefordert einfach quadriert wird. Dies resultiert daraus, dass durch die gerade Hochzahl das Vorzeichen verloren ginge. Damit würde der Luftwiderstand bei beiden Bewegungsrichtungen in dieselbe Richtung wirken. Um dies zu verhindern, wird der Umweg über folgende mathematische Schreibweise gemacht:

$$\vec{v}^2 = \vec{v}|\vec{v}|.$$

Der vektorielle Betrag ist in Listing 5 auf einen Skalar angewandt. Auf diese Art wird die Richtung der Kraft gewahrt.

```
1 parameter Modelica.SIunits.Density rho=1 "Density of air for drag
    calculation";
2 parameter Modelica.SIunits.Area S=1 "Area of car front for drag
    calculation";
3 parameter Real cx=1 "Drag coefficient";
4 force.force=-{(rho*S*cx/2)*absoluteVelocity.v[1]*(absoluteVelocity.v[1]^2)
    ^(0.5),0,0};
```

Listing 5: Ausschnitt aus dem Quelltext der Berechnung des Luftwiderstands in Modelica.

In Abbildung 5.9 ist zu sehen, dass der Luftwiderstandsblock (dragForce) an der Vorderseite des Fahrzeugs angreift. Für das Vorwärtsfahren ist dieses Verhalten erwünscht, nicht jedoch für das Rückwärtsfahren. Der Unterschied des Kraftangriffs tritt jedoch erst ein, wenn aufgrund von großen Beschleunigungen das Fahrzeug um die z-Achse geneigt

wird. Da beim Rückwärtsfahren keine starken Beschleunigungen erwartet werden, ist dieser Fehler zu vernachlässigen.

5.3. Fahrwegmodell

Der fehlende Teil für ein einfaches aber funktionsfähiges Gesamtfahrzeugmodell ist der Boden. Der Boden muss verschiedene wichtige Aufgaben erfüllen. Prinzipiell definiert er die Fahrbahn und stellt damit die Umgebung dar. Außerdem sind diverse systemrelevante Funktionen notwendig, um das Modell als Ganzes zum Laufen zu bringen.

Jeder Auflagepunkt eines Rades verfügt über drei Freiheitsgrade im Raum, nämlich die translatorischen. Insgesamt besitzt das Fahrzeug also zehn Freiheitsgrade. Ein voll räumliches Modell ist die flexibelste und allgemeingültige Modellierung, allerdings können damit numerische Instabilitäten auftreten. Abgesehen davon werden bei der Berechnung einfacher Fälle Gleichungen gelöst, die keinen Einfluss auf das Ergebnis haben. Um solche Schwierigkeiten vermeiden zu können, werden die Eigenschaften des Bodens so gewählt, dass alle durch das Fahrzeug- und Reifenmodell abbildbaren Szenarien ermöglicht werden. Die folgenden Vereinfachungen werden daher getroffen:

- Die Räder können sich nur in x- und y-Richtung bewegen.
- Es ist keine Kurvenfahrt möglich.
- Die Räder rollen schlupffrei auf dem Boden.
- Das Höhenprofil wird an der Vorder- und Hinterachse vorgegeben.
- Die Aufstandspunkte der Räder werden mit den Abständen initialisiert.
- Die Räder rollen immer auf horizontalem Boden.

Die erste Vereinfachung bewirkt, dass das Fahrzeug quasi auf Schienen fährt. In z-Richtung ist die Bewegung gesperrt, es treten leistungslose Zwangskräfte auf. Dadurch werden Bewegungen, die aufgrund kleiner Kräfte oder numerischer Fehler auftreten, aufgefangen und ein seitliches „Rutschen“ des Fahrzeugs vermieden. Bei zwei der Räder muss der Freiheitsgrad in z-Richtung freigegeben werden, damit kein statisch unbestimmtes System entsteht, das in der Simulation nicht aufgelöst werden könnte.

Durch die Restriktionen des ersten Punktes ergibt sich, dass keine Kurvenfahrt möglich ist. Die Modellierung dieser wäre im vorgestellten Modell prinzipiell möglich, würde aber

eine große Anzahl an Modifikationen erfordern. Zum Beispiel müssten die Räder drehbar gestaltet werden, was wiederum einen weiteren Input für die Lenkstange benötigte.

Die Beschaffenheit des Untergrunds wird in der Betrachtung idealisiert angenommen. Der Reifen haftet auf dem Boden, es tritt kein Schlupf auf. Dadurch ist es auch nur sehr eingeschränkt möglich, das Fahren auf unterschiedlichen Untergründen darzustellen. Eine Erweiterung des Modells um einen Schlupf wäre in Folgearbeiten denkbar.

Prinzipiell könnte der Untergrund zwischen linkem und rechtem Rad unterschiedlich beschaffen sein. Diese Tatsache wird allerdings in der vorliegenden Betrachtung ebenfalls außen vor gelassen, da dieser Umstand bei normalem Fahrbetrieb vernachlässigbar ist. Dementsprechend argumentiert sich die Vereinfachung, das Höhenprofil an beiden Achsen vorzugeben. Die zwei Räder einer Achse sind zu jeder Zeit auf derselben Höhe.

In der Realität muss der Boden keine Informationen über die Geometrie des Fahrzeugs besitzen. Es wäre auch in Modelica denkbar den Abstand der Räder nicht fest vorzuschreiben, sondern rein über Gelenke festzulegen. Dieser Ansatz scheitert an der Problematik, dass bei der Simulation die richtigen Initialisierungsdaten schwer oder gar nicht gefunden werden könnten. Um dem entgegenzuwirken, werden die Auflagepunkte an den Positionen initialisiert, an denen sich die Räder zu Beginn der Simulation befinden müssen. Der Nachteil darin besteht, dass Achsabstand und Breite des Fahrzeugs auch dem Fahrwegmodell bekannt sein müssen.

Die letzte der aufgelisteten Restriktionen ist gleichzeitig die problematischste. In der Natur ändert sich bei unterschiedlicher Steigung des Höhenprofils der Auflagepunkt am Reifen, indem er am Umfang verschoben wird. Dadurch ergeben sich Anteile der Schwerkraft in beziehungsweise gegen die Fahrtrichtung. Bei den Modellen des Rades und des Bodens ist diese Forderung schwer umsetzbar. Die Lösung dieses Problems besteht darin, die Räder virtuell auf einer flachen Oberfläche rollen zu lassen und die Belastung durch das Gewicht des Fahrzeugs als äußere Kraft aufzubringen, siehe Abschnitt 5.3.2.

5.3.1. Umsetzung des Fahrwegmodells

In Modelica wird ein Modell erstellt, das den oben gestellten Anforderungen unter Berücksichtigung der gewählten Vereinfachungen gerecht wird. In Abbildung 5.11 ist die grafische Umsetzung zu sehen. Zur einfacheren Orientierung wurden die wesentlichen Komponenten bei den Auflageflächen der Räder mit punktierten Linien gekennzeichnet. Jeder Auflagepunkt eines Reifens ist über einen $frame\_b$ definiert. Diese Interfaces werden über die notwendigen prismaticJoints mit der Umgebung verbunden. Der absolute Punkt der Gesamtsimulation, also die World-Komponente, wird über den $frame\_a$ an

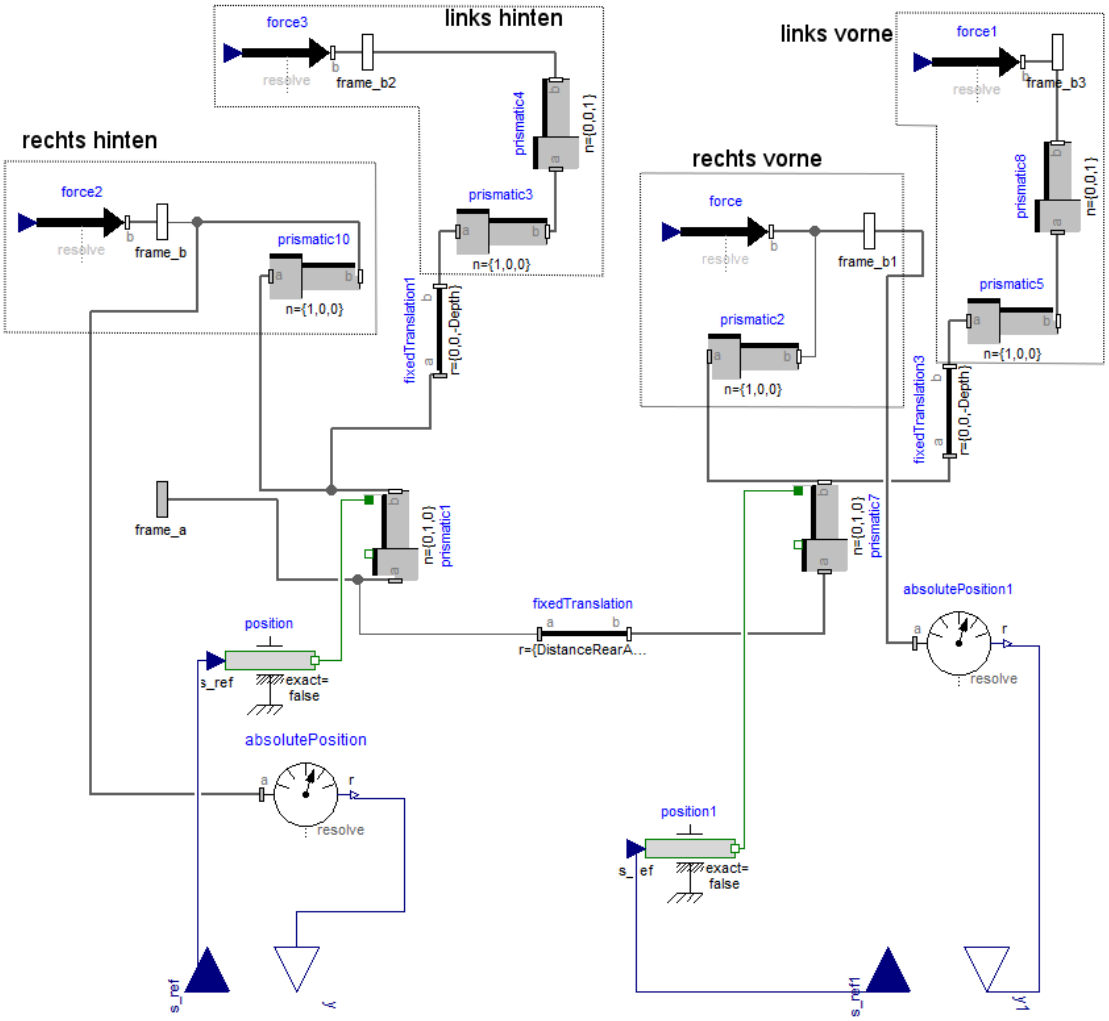


Abbildung 5.11: Fahrwegmodell in Modelica.

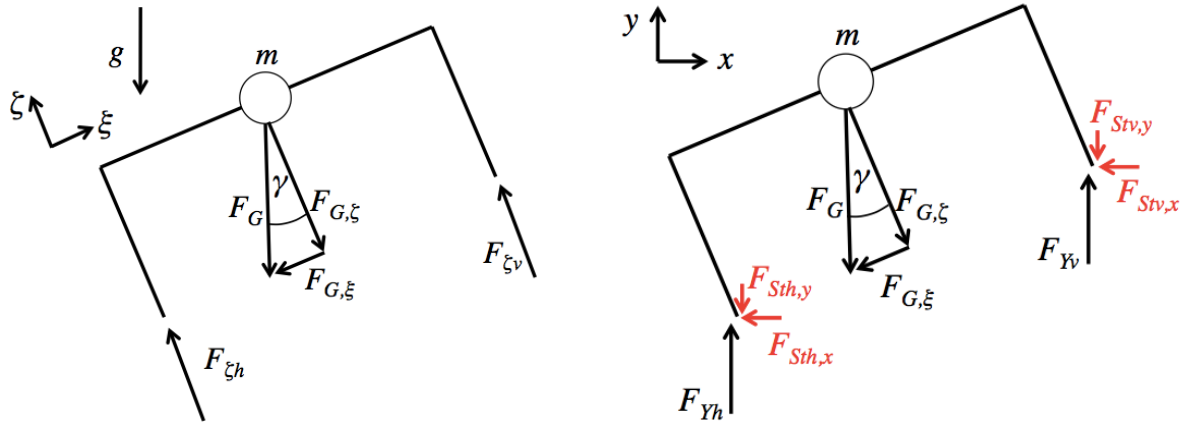
das Fahrwegmodell angeschlossen. Vom Koordinatenursprung ausgehend ist ein Gelenk angebracht, das die translatorische Bewegung in y-Richtung, also vertikal, erlaubt. Ein zweites dieser Joints ist durch ein fixedTranslation-Element in x-Richtung versetzt angebracht. An diesen beiden Gelenken wird das Höhenprofil der Achsen vorgegeben. An den prismaticJoints wird über die translatorischen Hilfsinterfaces – in Modelica als grüne Quadrate dargestellt – über einen Positionsaktuator die y-Koordinate vorgegeben. Der Wert der Position wird extern bezogen und daher über ein Input-Interface eingelesen. An den vertikalen prismaticJoints sind die Verbindungen zu den vier Rädern angebracht. Es ist zu sehen, dass sowohl rechtes Vorder- als auch Hinterrad über lediglich ein weiteres Gelenk verbunden sind, das die x-Richtung frei gibt. Die beiden linken Reifen haben als zusätzlichen Freiheitsgrad auch die z-Koordinate, um ein statisch unbestimmtes Sys-

tem zu vermeiden. Die beiden Gelenke sind um die Breite des Fahrzeugs in negativer z-Richtung verschoben, was wieder über ein fixedTranslation-Element ausgeführt ist.

An den Auflagepunkten der beiden rechten Räder wird die Position gemessen. Die x-Komponente dieses Lagevektors wird dann über ein Output-Interface aus dem Fahrwegmodell ausgegeben. Dies ist notwendig, da die vertikale Auslenkung der Achsen davon abhängt, wie weit sich das Fahrzeug bewegt hat. Es wird also ein Höhenprofil extern vorgegeben, wobei das Fahrwegmodell meldet, an welchen Positionen sich die Achsen befinden.

5.3.2. Kräfte aufgrund der Steigung der Fahrbahn

In Abbildung 5.11 ist zu sehen, dass an allen vier Auflagepunkten der Räder eine Kraft angreift. Wie bei den Vereinfachungen erwähnt, stehen die Räder immer auf einer horizontalen Fläche, auch wenn sich diese in unterschiedlicher Höhe befindet. Daher muss der Steigungswiderstand als äußere Kraft eingebracht werden. Aufgrund der lokalen Betrachtung in Modelica können nur Größen verwendet werden, die im Fahrwegmodell auftreten. Deshalb kann die Gewichtskraft, die im Schwerpunkt des Fahrzeugs angreift, nicht herangezogen werden. Die Gewichtsinformation ist allerdings in den Radlasten enthalten. Gemeinsam mit dem Steigungswinkel γ kann dann der Steigungswiderstand berechnet werden.



(a) Darstellung der Kraefte im ξ-ζ-Koordinatensystem.

(b) Darstellung der Kräfte im x-y-Koordinatensystem.

Abbildung 5.12: Schematische Darstellung des geneigten Fahrzeugs.

Für die Berechnung werden die Vorgänge bei geneigter Fahrbahn im statischen Fall betrachtet, es handelt sich daher um eine Momentanaufnahme. Die variable Länge der

Aufbaufederung wird in den folgenden Ausführungen nicht berücksichtigt. Die Veränderungen der Schwerpunktslage und die dynamischen Kräfte durch die Federungen fließen automatisch über die Radlasten ein. In Abbildung 5.12 ist eine ebene Darstellung des Problems gezeigt.

Im linken der beiden Bilder wurde ein Koordinatensystem gewählt, das an die Neigung der Fahrbahn angepasst ist. Diese Vorstellung entspricht den in der Realität vorkommenden Verhältnissen. Die tatsächlich auftretenden Kräfte sind normal zum Boden und werden mit $F_{\zeta v}$ und $F_{\zeta h}$ bezeichnet. Die Indices v und h beziehen sich auf die Vorder- und Hinterachse, die unterschiedlich große Reaktionskräfte aufweisen. Der Komponente $F_{G,\xi}$ entspricht keine Reaktionskraft und führt zu einer beschleunigten Bewegung des Fahrzeugs in Richtung $-\xi$.

Die Gewichtskraft teilt sich in Folge der Neigung der Straße unterschiedlich auf die beiden Auflagepunkte auf. Diese Aufteilung wird durch das Modell der Aufbaumassen abgebildet und spiegelt sich in den Radlasten wider. Dazu werden die Verhältnisse an der Hinterachse genauer betrachtet, wie im Kräfteplan in Abbildung 5.13 zu sehen ist.

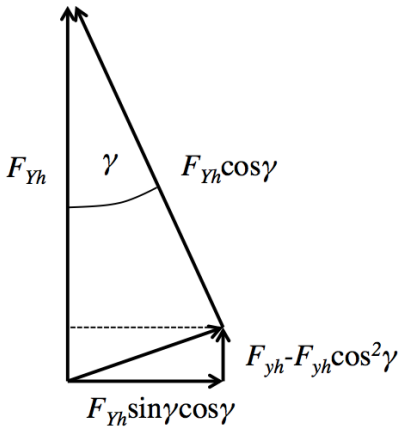


Abbildung 5.13: Trigonometrische Überlegungen zur Radlast an der Hinterachse bei der Berechnung des Steigungswiderstands.

Die im Modell vorhandene Radlast F_{Yh} muss so korrigiert werden, dass die tatsächlich auftretenden Kraft $F_{\zeta h}$ wirkt. Dafür wird der Vektor F_{Sth} von F_{Yh} abgezogen, die zusätzlich eingebrachte Kraft wird mit dem Index St für Steigung versehen.

$$F_{Sth,x} = F_{Yh} \sin\gamma \cos\gamma$$

$$F_{Sth,y} = F_{Yh} - F_{Yh} \cos^2\gamma.$$

Äquivalent kann mit den Kräften an der Vorderachse verfahren werden. Die Kraft F_{Yh}

stellt die Radlast an der Hinterachse dar, die Aufteilung auf die zwei Räder muss daher noch erfolgen. In Modelica werden die Anteile der Gewichtskraft der vier Räder ohnehin getrennt bestimmt, dementsprechend können die Auflagerreaktionen der vier $frame\_b$ zur Berechnung der Kräfte herangezogen werden. Exemplarisch wird in Listing 6 die Definition der Kraft an dem rechten Vorderrad gezeigt (siehe Abbildung 5.11).

```
1 force.force ={Modelica.Math.sin(gamma)*Modelica.Math.cos(gamma)*frame_b1.f
    [2], Modelica.Math.cos(gamma)^2*frame_b1.f[2]-frame_b1.f[2],0};
```

Listing 6: Quelltext der Berechnung der Kraft aufgrund der Fahrbahnneigung.

Es ist anzumerken, dass sich aufgrund der unterschiedlichen Koordinatendefinition in Modelica ein Vorzeichen ändert. Der Winkel γ kann im Fahrwegmodell leicht gebildet werden, da die Positionen der Vorder- und Hinterachse bekannt sind. Über den Achsabstand und den Höhenunterschied ergibt sich die Steigung. Dies impliziert die Annahme, dass sich die lokale Fahrbahnneigung an den Radaufstandspunkten annähernd aus der mittleren Fahrbahnneigung ergibt.

Damit sind alle Anforderungen, die an das Fahrwegmodell gestellt wurden, erfüllt. Mit den vorgestellten Komponenten ist es bereits möglich, eine Simulation des Gesamtfahrzeugs zu erstellen. Es wurde bis jetzt allerdings noch nicht spezifiziert, wie der Antrieb des Fahrzeugs integriert werden kann.

5.4. Modell des Antriebsstrangs

Als letzte Komponente wird ein simples Modell eines Antriebsstrangs implementiert. Wie bei allen vorgestellen Teilen kann auch dieser aufbauend auf die zu untersuchenden Eigenschaften erweitert werden. Den Eingang in den Antriebsstrang stellt das Drehmoment dar. Es wird von außen drehzahlabhängig vorgegeben. Zur Vereinfachung werden sowohl Motor als auch Hauptgetriebe nicht in Modelica abgebildet sondern parametrisch berücksichtigt, siehe Abschnitt 6. Das betrachtete Fahrzeug soll angetriebene Vorderräder besitzen, dementsprechend wird das Antriebsmoment über ein Differentialgetriebe und zwei Seitenwellen an die Räder geliefert. Abbildung 5.14 zeigt das Modell.

Am Eingang in das System, der über ein Input-Interface gestaltet ist, wird der Zahlenwert des Drehmoments an ein Torque-Element übergeben. Für die korrekte Drehrichtung muss das Signal durch den Gain das Vorzeichen wechseln. Das Antriebsmoment wirkt in x-Richtung, die beiden anderen Komponenten des Torque-Vektors werden auf null gesetzt. Die Antriebswelle wird zweigeteilt dargestellt, in der Abbildung als driveShaftLeft und driveShaftRight bezeichnet. Zwischen den zwei Körpern, die die Wel-

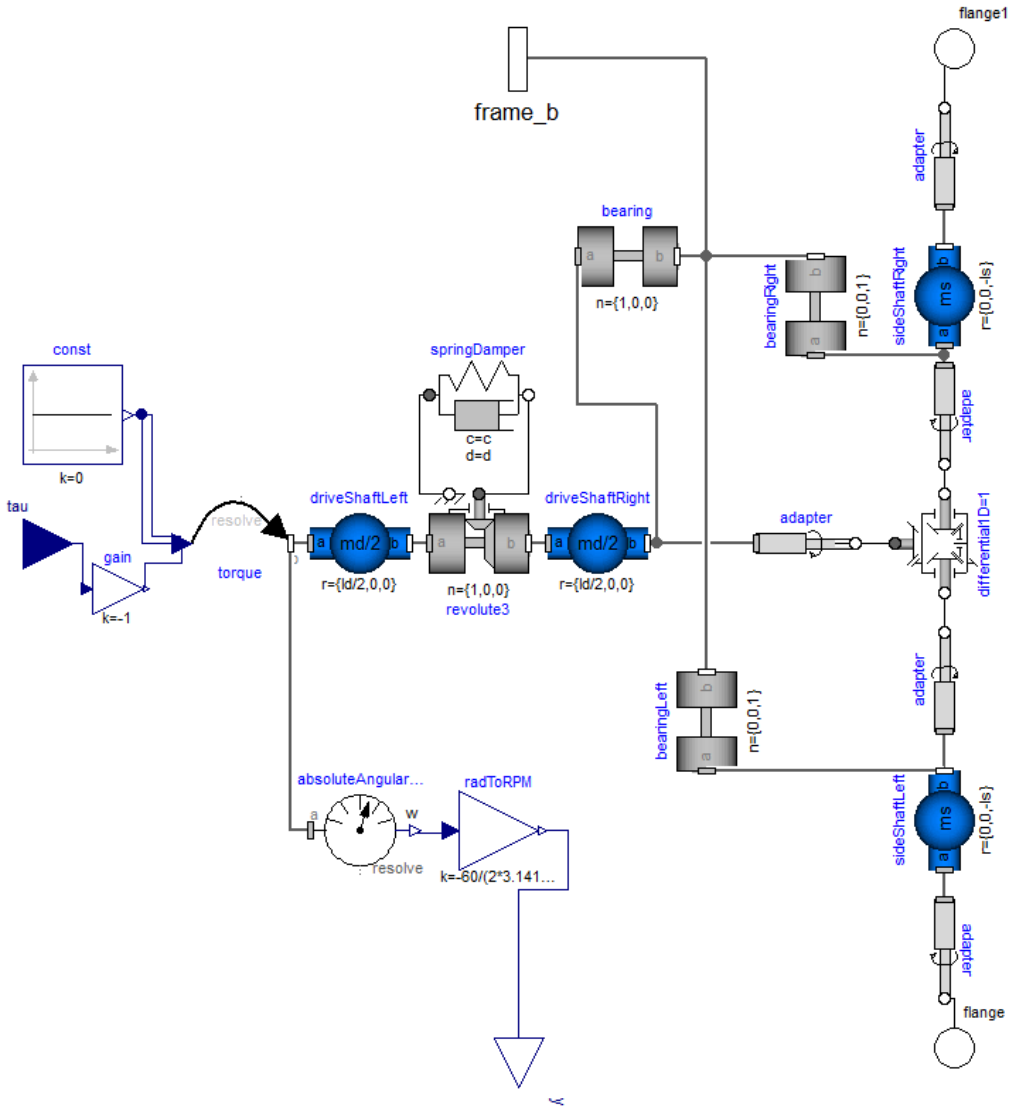


Abbildung 5.14: Antriebsstrangmodell in Modelica.

le darstellen, ist ein rotatorisches Gelenk angebracht, an dessen support-Flanges ein Drehfeder-Drehdämpfer-Element angebracht ist. Dieses Konstrukt realisiert die elastischen Eigenschaften der Welle. Über einen Adapter (siehe weiter unten) werden Drehmoment und Drehzahl auf ein Differentialgetriebe geleitet. Diese Komponente ist in der PowerTrain-Library enthalten und stellt eine Momentenaufteilung und Drehzahlbilanz dar. Außerdem ist es möglich eine Übersetzung anzugeben. Nach dem Differential wird die Antriebsleistung über die zwei Seitenwellen an zwei Flange-Interfaces übergeben. Drei revoluteJoints stellen als Lagerungen die Verbindung des rotatorischen Systems mit dem umgebenden Fahrzeugmodell dar. Sie sind mit einem $frame\_b$ verbunden,

der wiederum der Halterung des Antriebsstrangs entspricht. An der rechten Seite der Antriebswelle wird über ein Messgerät die Winkelgeschwindigkeit abgegriffen. Sie wird durch ein Gain-Element von Hand in die Einheit Umdrehungen pro Minute umgerechnet und über einen Output zur Verfügung gestellt.

Der Antriebsstrang stellt prinzipiell ein rein rotatorisches System dar. Es ist jedoch vor allem für die Anwendung mit CATIA 3DParts wünschenswert, die physischen Komponenten als volle MultiBody-Körper umzusetzen. Dieses Vorgehen verlangt es, dass zwischen den zwei verschiedenen Konnektorenarten gewechselt werden kann. Diese Aufgabe übernehmen die eigens dafür kreierten Adapter-Elemente.

5.4.1. Adapter zur Konvertierung von MultiBody auf rotatorische Interfaces

Wie in Abschnitt 2 vorgestellt, werden in Modelica gleichartige Konnektoren verbunden, um so das Gesamtsystem zusammenzustellen. Verschiedenartige Interfaces können hingegen nicht ohne Weiteres gekoppelt werden. Deshalb wird eine Klasse erstellt, die der Konvertierung von unterschiedlichen Interfaces dient.

Im Gegensatz zu den eindimensionalen mechanischen Interfaces werden bei MultiBody-Konnektoren sowohl translatorische als auch rotatorische Informationen abgebildet. Die Herausforderung ist daher, die relevanten Fluss- und Potentialgrößen an ein anderes Interface zu übergeben und die übrigen auf sinnvolle Werte zu setzen. Die Klasse, die im Weiteren als Adapter bezeichnet wird, soll zwei Konnektoren besitzen: einen MultiBody $frame\_a$ und einen rotatorischen $flange\_b$. Ein Interface der rotatorischen Bibliothek hat als Potentialgröße den Winkel φ und als Flussgröße das Drehmoment τ. Da die rotatorische Library eindimensional aufgebaut ist, muss als erstes eine Drehrichtung festgelegt werden. In dieser Achse wird eine Drehbewegung des MultiBody-Konnektors auf den Flange übertragen. Dafür wird als Parameter der Vektor n definiert, der die Drehrichtung vorgibt. Über diese Definition kann dem Wert τ die dementsprechende Komponente des Drehmoments des MultiBody-Interfaces zugewiesen werden. Die restlichen Einträge des Momentenvektors werden auf null gesetzt, da der Drehung um die anderen Achsen kein Drehmoment entgegenwirken kann. Auch dem Kraftvektor von $frame\_a$ können keine Kräfte entgegenwirken da vom rotatorischen Interface keine translatorischen Größen übertragen werden können.

Der Winkel φ muss über die Funktion MultiBody.Frames.angularVelocity1 aus dem Orientation-Konstrukt des $frame\_a$ gewonnen werden. Die Funktion extrahiert die Winkelgeschwindigkeiten ω_i in allen Raumrichtungen. Der Winkel φ wird dann nach Bildung des inneren Produkts mit n durch Integration der Winkelgeschwindigkeit be-

rechnet. Der Umweg über ω ist deshalb notwendig, da beim Messen der Winkel immer in den Grenzen $[-180°; 180°]$ ausgegeben wird. Dieses Verhalten ist nicht erwünscht. Durch die vorgestellte Implementierung wird bei der Integration der Winkelgeschwindigkeit der Winkel fortgeführt, ohne die 2π-Periodizität zu berücksichtigen. Letztlich ist zu erwähnen, dass die noch fehlende Potentialgröße r_0 (siehe Abschnitt 2) nicht vorgegeben werden kann, da die Position von außen aufgeprägt wird. In Listing 7 ist der Quelltext der Komponente in Modelica zu sehen.

```
1  model adapter
2    Modelica.Mechanics.MultiBody.Interfaces.Frame_a frame_a;
3    Modelica.Mechanics.Rotational.Interfaces.Flange_b flange_b;
4    Modelica.SIunits.AngularVelocity w[3];
5    parameter Real n[3]={1,0,0} "Direction of rotation";
6  equation
7    Connections.root(frame_a);
8    frame_a.f = zeros(3);
9    zeros(3) = flange_b.tau*n+frame_a.t;
10   w= Modelica.Mechanics.MultiBody.Frames.angularVelocity1(frame_a.R);
11   der(flange_b.phi) =w*n;
12 end adapter;
```

Listing 7: Sourcecode der Adapter-Klasse im textuellen Layer von Modelica.

Damit sind sämtliche Komponenten für ein Gesamtfahrzeugmodell aufgebaut. Im nächsten Abschnitt wird gezeigt, wie die Implementierung in Modelica in eine Co-Simulation eingebaut werden kann und welche Vorteile dieses Vorgehen mit sich bringt.

6. Co-Simulation eines Fahrzeugsmodells

Aufbauend auf den Untersuchungen der Möglichkeiten von Co-Simulation mit CATIA V6 wird ein Modell eines Gesamtfahrzeugs erstellt. Die mechanischen Komponenten wurden mit Hilfe der Modelica-Sprache abgebildet und können in CATIA simuliert werden. Dafür müssen die einzelnen Teile zuerst zu einem Modell zusammengestellt werden. Die Simulation wird über BCVTB (Building Controlls Virtual Test Bed) gesteuert. Vor allem die Verteilung der Datenströme und die Kommunikationsintervalle werden darin definiert. Die zwei fehlenden Teilebereiche sind der Antrieb des Fahrzeugs und das Profil der Fahrbahn. Der Antrieb wird als Kennlinie in MATLAB implementiert. Zusätzlich zur Motorkennlinie wird das Hauptgetriebe über die Übersetzungsparameter ebenfalls außerhalb von CATIA eingestellt. Die Wahl des Gangs wird über ein Statechart in Stateflow vorgegeben, die Informationen werden dann an MATLAB geschickt und verarbeitet und im Endeffekt an CATIA übergeben. Der Fahrbahnverlauf wird in MATLAB definiert und während der Simulation wird die aktuelle Fahrbahnhöhe der Vorder- und Hinterachse, abhängig von der aktuellen Position, ausgelesen. In Abbildung 6.1 ist der Aufbau der Co-Simulation schematisch dargestellt.

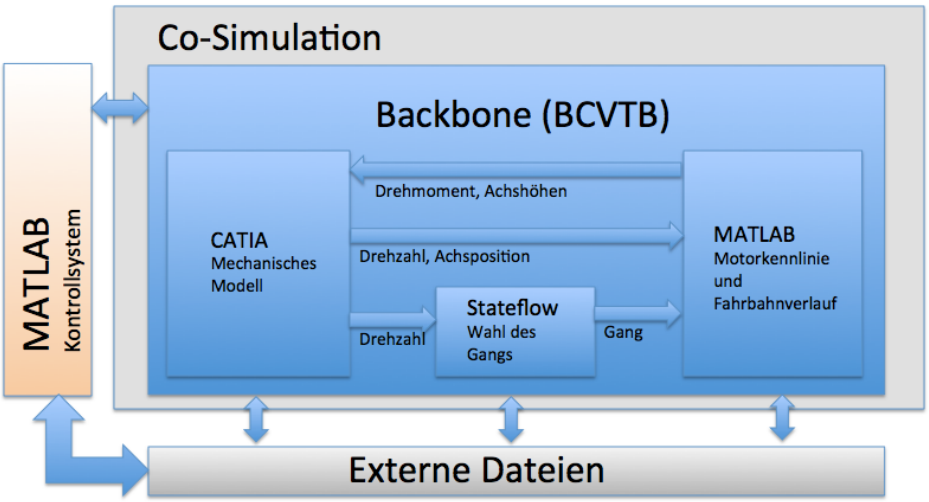


Abbildung 6.1: Schematischer Aufbau der Co-Simulation des Fahrzeugs.

Um die Simulation zentral steuern zu können, wird über BCVTB eine weitere Ebene eingeführt, die die notwendigen Parameter an die anderen Programme weitergibt. Die Funktionsweise des Kontrollskripts wurde bereits ausführlich in Abschnitt 3.2.4 beschrieben. Durch die Verwendung von CATIA können nicht mehr alle Simulationsparameter vorgegeben werden, da auf eine aktive Instanz zugegriffen werden muss. Es müssen also

die notwendigen Werte zwei Mal eingestellt werden. Da es sich lediglich um Anfangs- und Endpunkt der Simulation handelt, ist die Einschränkung gering.

Außerdem wird ein weiterer Parameter definiert, der eine Verzögerung des Beginns des Antriebsmoments darstellt. Diese Vorgabe kommt im MATLAB-Modell zu tragen. Die Größe des Drehmoments wird auf null gesetzt, bis der vorgegebene Zeitpunkt erreicht wird. Auf diese Art kann sich das System einschwingen, da die Initialisierung der Federn mit der ungedehnten Federlänge geschieht und dadurch erst einen stabilen Punkt erreichen kann. Das hilft die gewonnenen Ergebnisse einfacher interpretieren zu können, da sich verschiedene Phänomene nicht mehr überlagern.

Folgend werden die einzelnen Elemente der Co-Simulation im Detail beschrieben und die Abhängigkeiten beleuchtet. Um das gesamte Modell auf Plausibilität zu überprüfen, werden unterschiedliche Szenarien simuliert.

6.1. Das Backbone der Simulation: BCVTB

Über das Building Controls Virtual Test Bed wird die gesamte Simulation angestoßen und der Datenverkehr zwischen den Teilsimulationen gesteuert. Es erfüllt für die Co-Simulation damit die Rolle des Backbones. Die Anzahl an Aus- und Eingängen wird prinzipiell in den einzelnen Simulatoren selbst definiert. In BCVTB müssen die Datenströme daran angepasst verteilt werden. Vor allem die Dimension der Vektoren muss beachtet werden. In Abbildung 6.2 ist das Modell zu sehen.

In der linken oberen Ecke befindet sich der Synchronous-Data-Flow-Director, der das Systemverhalten regelt. Ihm müssen die Kenngrößen für den Datenaustausch übergeben werden. Die darunter angeordneten Parameter besitzen lediglich Default-Werte, die tatsächlichen Größen werden über das Kontrollskript aus MATLAB übermittelt. Der Simulator, in dem das mechanische System berechnet wird, ist CATIA, der dazugehörige Actor ist am oberen Rand angebracht. Die Daten, die aus CATIA ausgelesen werden, sind die Drehzahl sowie die x-Positionen der Hinter- und Vorderachse. In dem VectorDisassambler VD1 wird der Vektor der Länge drei in die Einzelkomponenten aufgespalten. Die drei Signale werden an MATLAB weitergegeben, Stateflow erhält nur die Drehzahl. Vor jedem der Simulator-Actor-Blöcke müssen die Komponenten in einen Vektor gefasst werden. Dies geschieht mit den VectorAssemblern VA1 und VA2. In Stateflow ist das Fahrermodell implementiert, in dem abhängig von der Drehzahl der Gang des Schaltgetriebes bestimmt wird (mehr dazu in Abschnitt 6.4). Der aktuelle Gang wird ebenfalls an MATLAB übermittelt. Dieser Simulator-Actor bekommt also einen Eingangsvektor mit vier Größen übergeben. Das MATLAB-Modell bestimmt aufgrund der Drehzahl und des

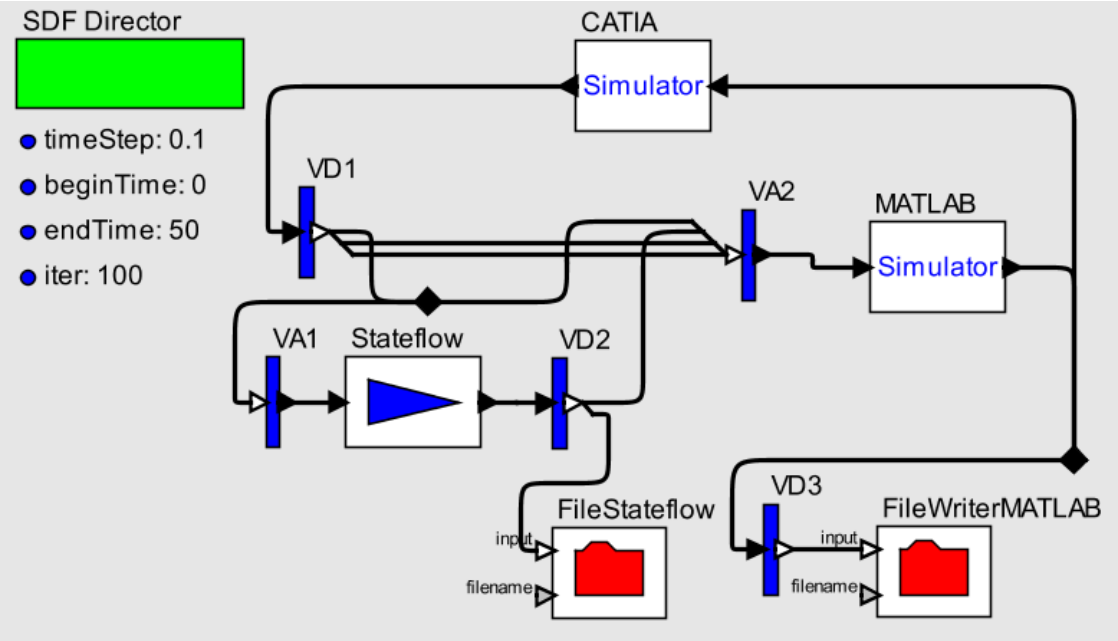


Abbildung 6.2: Schaltbild der Co-Simulation in BCVTB mit den drei Clients CATIA, MATLAB und Stateflow.

Gangs das Antriebsmoment und mit den Informationen der Achsenposition die jeweiligen Werte des Höhenprofils. Diese drei Größen werden an CATIA übergeben. Damit ist der Kreislauf geschlossen. Zusätzlich werden Ausgabewerte von Stateflow und MATLAB in Dateien geschrieben, um nachträglich eine leichte Kontrolle zu ermöglichen.

Das Modell in BCVTB steuert lediglich die Verteilung der Daten, führt aber selbst keine Berechnungen durch. In den nächsten Abschnitten wird auf die Funktionsweise der Modelle in MATLAB und Stateflow genauer eingegangen.

6.2. Gesamtfahrzeugmodell in Modelica

Die bereits erstellten Klassen werden in einem Modell in CATIA instanziiert. Es ist zu beachten, dass alle notwendigen Bibliotheken zuvor in die CATIA-Datenbank eingelesen werden müssen. Die Symbole der Komponenten, die für jede Klasse implementiert wurden, erleichtern die Orientierung im zusammengesetzten System. Abbildung 6.3 zeigt das Modell im grafischen Layer. Am oberen Ende der Grafik ist die Darstellung einer Karosserie erkennbar. Dieses Bild stellt das Fahrzeugmodell dar. Darin sind die Aufbaumassen enthalten sowie der Block für die Berechnung des Luftwiderstands. Die Verbindung zu den Rädern erfolgt über die Aufbaufedern. Die vier erkennbaren weißen Frames können mit der Radaufhängung verbunden werden. Die Vorderseite des Fahrzeugs ist rechts,

angedeutet durch die Form des Chassis. Diese Ausrichtung wurde gewählt, damit sie der Fahrtrichtung in der Animation entspricht.

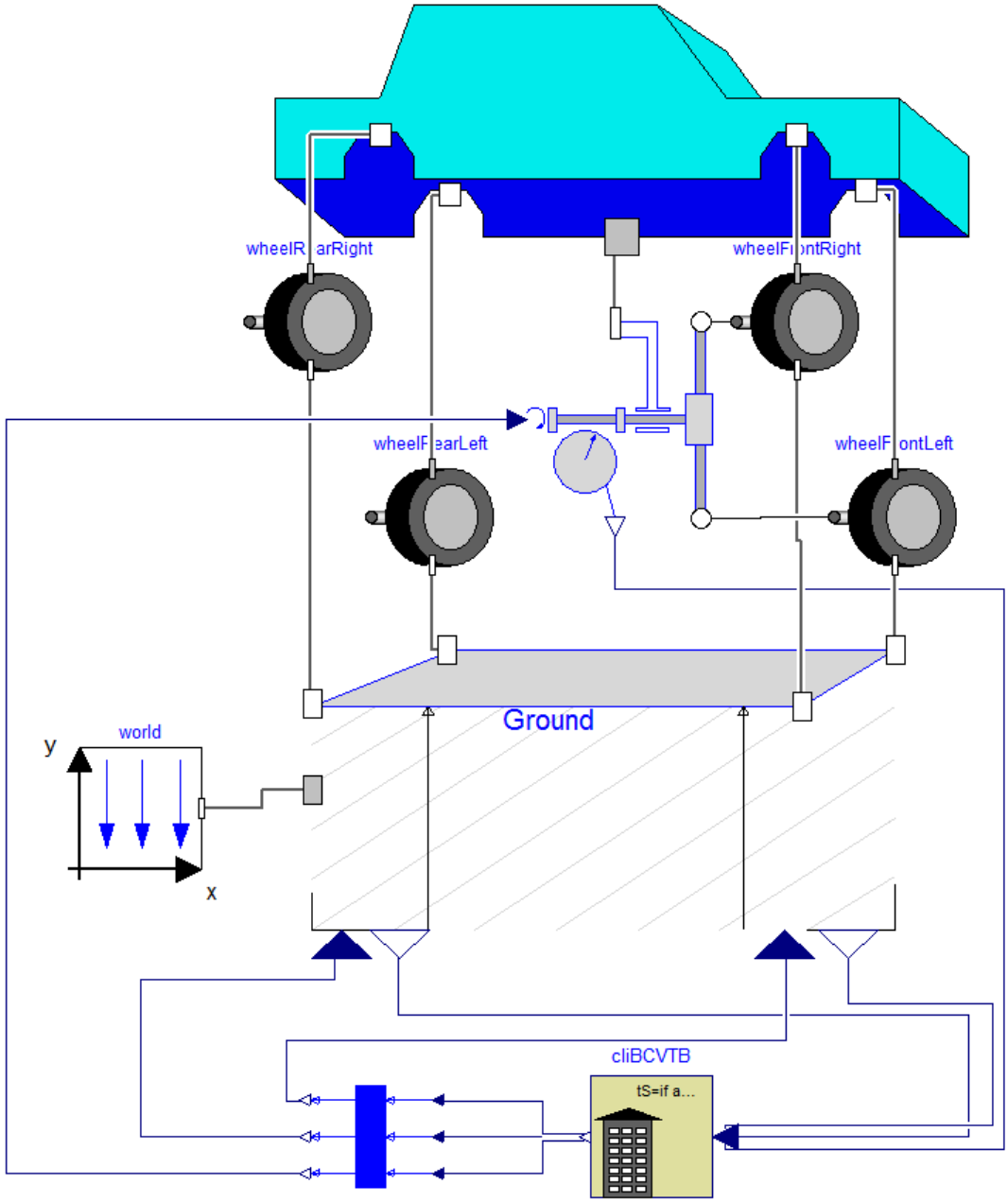


Abbildung 6.3: Zusammengesetztes Fahrzeugmodell in Modelica.

An den Aufbaufedern sind die vier Räder angebracht. Sie sind Instanzen derselben Klasse. Über die im Radmodell definierten Parameter können ihnen unterschiedliche Eigenschaften verliehen werden. Das Fahrzeug soll mit einem Vorderradantrieb ausgestattet sein, dementsprechend können die rotatorischen Konnektoren der Hinterräder

frei bleiben. An den Vorderrädern bringen die Seitenwellen des Antriebsstrangs die Antriebsleistung ein.

Der Antriebsstrang ist über einen MultiBody-Konnektor am Fahrzeugmodell befestigt. Damit wird gewährleistet, dass die mechanischen Eigenschaften abseits der Rotation der Wellen ins System einfließen. Die Größe des Drehmoments wird von MATLAB vorgegeben, dementsprechend bezieht der Antriebsstrang sein Eingangssignal aus dem BCVTB-Block. Der gemessene Wert der Drehzahl wird zu diesem Zweck an die anderen Simulatoren übergeben.

Das Fahrwegmodell stellt die Auflagefläche des Fahrzeugs dar. Der Untergrund ist über den grauen Frame mit der World-Komponente verbunden. Somit ist festgelegt, dass der Boden das ruhende System ist und sich das Fahrzeug relativ dazu bewegt. Die Richtung der Schwerkraft wird in negativer y-Richtung gezählt. Auf diese Art stimmen die in den einzelnen Klassen beschriebenen Koordinatensysteme mit dem der Gesamtsimulation überein. Die Auflagepunkte der Räder sind als weiße Frames zu erkennen. Es ist notwendig, dass die Räder an den jeweils dazu angeordneten Verbindungspunkten angeschlossen werden, also zum Beispiel das linke hintere Rad des Fahrzeugs auch links hinten am Boden steht. Daher ist die grafische Repräsentation auf die Art gestaltet, dass ein Verwechseln der Konnektoren erschwert wird. An der Unterseite des Fahrwegmodells werden die Lagekoordinaten der Achsen in x-Richtung an BCVTB übergeben. In MATLAB wird aus dem Höhenprofil der passende Wert ausgelesen und an das Fahrwegmodell übergeben.

Die Ausgänge aus dem BCVTB-Block werden nicht direkt an die passenden Stellen im Gesamtfahrzeugmodell übertragen, sondern zuerst in eine zusätzliche Komponente geleitet. Diese hat die Aufgabe das Signal aufzuarbeiten. Der Aufbau dieses Elements wird in weiterer Folge näher erläutert.

6.2.1. Aufarbeitung des Signals aus dem BCVTB-Block

Die Kommunikation zwischen BCVTB und CATIA geschieht zu diskreten Zeitpunkten. Sie sind äquidistant über die Simulationszeit verteilt, also nicht adaptiv. Im Normalfall ist es nicht notwendig eine Anzahl an Kommunikationen in der Größenordnung der Anzahl an Zeitschritten des numerischen Solvers zu haben. Dieses Verhalten hat zur Folge, dass während der Simulation CATIA nicht mit aktuellen Daten rechnen kann, sondern jene Eingangsdaten aus der jeweils letzten Kommunikation zu einer Treppenfunktion extrapoliert werden müssen. Die Länge, an denen der Wert konstant ist, entspricht den Kommunikationsintervallen. Dadurch treten allerdings Probleme auf, da die Funktion

nicht stetig ist. Bei der Übertragung des Drehmoments kann dieses Verhalten eine Anregung der Drehschwingungsdynamik im Antriebsstrang erzeugen. Bei der Übergabe des Höhenprofils sind die Auswirkungen sogar noch deutlicher merkbar. Das eigentlich glatte Fahrbahnprofil wird künstlich „holprig“, was zur Auswirkung hat, dass den Radlasten starke Schwankungen aufgeprägt werden. Ein Vergleich der Signale ist in Abbildung 6.4 zu sehen.

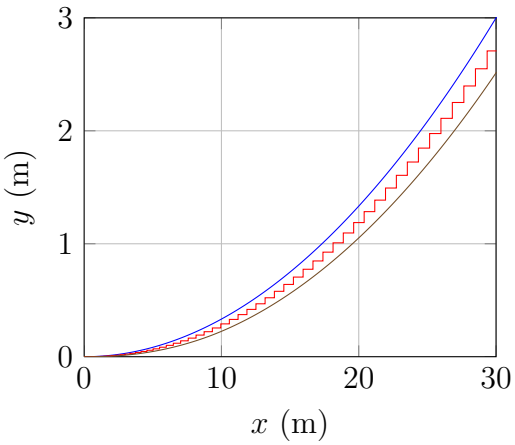


Abbildung 6.4: Vergleich der Funktionen der Höhenprofile. Blau: Funktion in MATLAB, rot: Treppenfunktion in CATIA, braun: Output der Glättungsfunktion gemäß Abbildung 6.5.

Um dem unnatürlichen Verhalten entgegenzuwirken, soll das Signal aufbereitet werden. Der Ansatz besteht darin die Funktion zu glätten. Dies muss während der Laufzeit passieren, dementsprechend kann nur auf den aktuellen und vergangene Zeitschritte zugegriffen werden, nicht aber auf zukünftige. Aus diesem Grund wird der Ansatz gewählt einseitige Differenzenverfahren zu verwenden. Im Prinzip kann die Ordnung frei gewählt werden. Es ist aber zu bedenken, dass durch die Glättung das Eingangssignal verzögert ausgegeben wird, mit steigender Ordnung nimmt dieser Effekt zu. Diese Verzögerung im Zeitbereich resultiert aus der Extrapolation der Funktion und wirkt sich über die Geschwindigkeit des Fahrzeugs auch auf das Fahrbahnprofil aus. Als Kompromiss werden Differenzen zweiter Ordnung für die Glättung des Antriebsmoments und dritter Ordnung für den Höhenverlauf verwendet. Eine Glättung zweiter Ordnung liefert eine stetig differenzierbare Funktion, die Ableitung davon ist allerdings nur stetig. In der zweiten Ableitung können also noch kleine Sprünge auftreten. Durch Versuche zeigt sich, dass der Ansatz ausreichend geeignet ist die numerischen Fehler zu glätten. Eine Diskussion des Einflusses auf die Vorgabe des Fahrbahnprofils folgt in Abschnitt 6.5.

Die Implementierung wird für finiten Rückwärtsdifferenzen zweiter Ordnung gezeigt. Für die Glättung dritter Ordnung ist der Ansatz gleich, die Anzahl der Blöcke und die Werte sind dementsprechend anzupassen. Für die Implementierung werden Elemente aus der diskreten Bibliothek der Modelica Standard Library benutzt, um damit die Auswertung der Funktion an vorhergehenden Zeitschritten zu bestimmen. Dafür muss die Kommunikationsschrittweite bekannt sein. Diese Vorgabe wird dadurch erfüllt, dass die Information aus der passenden Datei ausgelesen wird, in der das Kontrollskript die Information abgespeichert hat. Mit den Funktionswerten zweier vergangener Kommunikationen kann das Differenzenschema gebildet werden:

$$\frac{\partial^2 f_i}{\partial t^2} \approx \frac{f_i - 2f_{i-1} + f_{i-2}}{\Delta t^2}.$$

Die zweite Ableitung der Funktion kann in Folge zwei Mal integriert werden, um auf die geglättete Funktion zu kommen. Die Implementierung in Modelica ist in Abbildung 6.5 zu sehen.

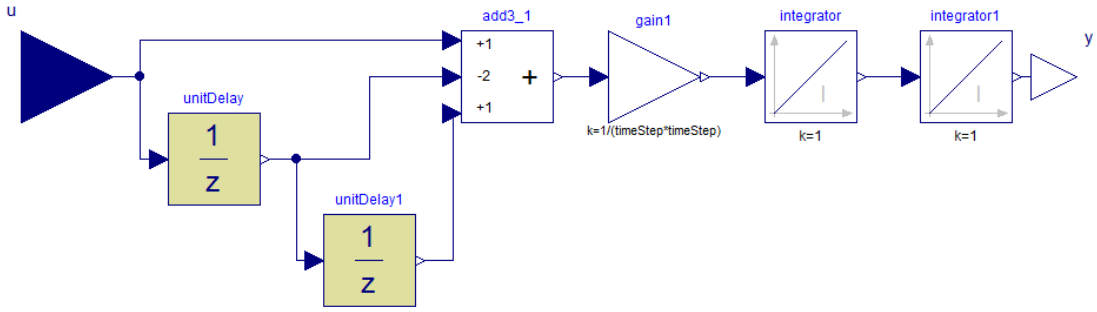


Abbildung 6.5: Modell zur Signalglättung während der Laufzeit in Modelica.

Im Block, der die Signalaufbereitung in Abbildung 6.3 übernimmt, sind drei Instanzen dieser Komponente enthalten, eine zweiter und zwei dritter Ordnung, siehe Abbildung 6.6. Die drei Signale aus BCVTB werden unterschiedlich behandelt, um etwa das Höhenprofil stärker zu glätten als das Drehmoment.

6.3. Datenbasierte Modelle in MATLAB

Das Modell in MATLAB hat zwei verschiedene Aufgaben zu bewältigen. Einerseits wird die Kennlinie des Antriebs mitsamt Hauptgetriebe ausgelesen. Andererseits wird das Höhenprofil vorgegeben. Die zwei Tätigkeiten laufen getrennt voneinander ab und benötigen die Informationen der jeweils anderen nicht. Es wäre denkbar diese Teilbereiche auf

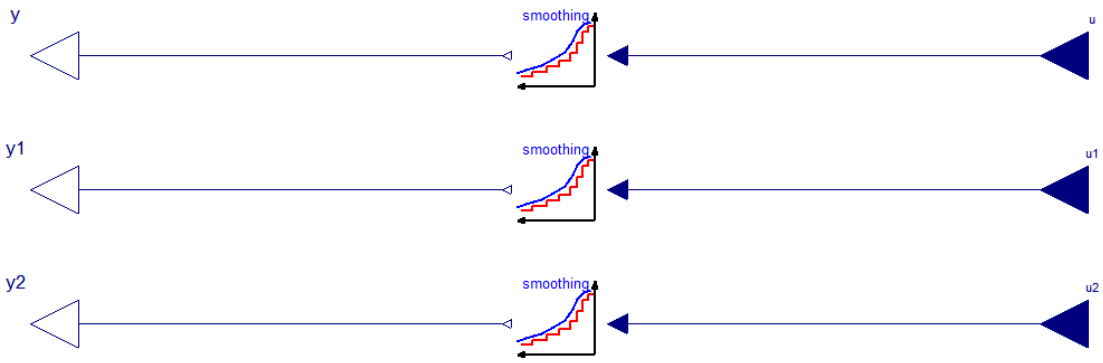


Abbildung 6.6: Klasse zur Aufarbeitung der Eingangssignale.

zwei Simulatoren aufzuspalten. Das hätte den Vorteil unterschiedliche Kommunikationsintervalle einsetzen zu können. Der Nachteil bestünde darin, dass eine zusätzliche Instanz weiteren Arbeitsspeicher allozieren würde. Da die zwei Vorgänge ähnliche Änderungsraten aufweisen, ist es nicht notwendig die Prozesse zu trennen. Stattdessen gestattet die Kombination das Gesamtsystem simpler zu gestalten. Bei den folgenden Ausführungen werden die Teilbereiche wenn möglich dennoch getrennt behandelt.

6.3.1. Motorkennlinie

Der Antrieb des Fahrzeugs wird parametrisch vorgegeben. Genauer gesagt wird das Verhalten des Motors als Kennlinie dargestellt. Das Ausgangsdrehmoment des Motors hängt im Normalfall von der Drehzahl und der Gaspedalstellung ab. Um auf Werte für letztere zu kommen, kann als Vereinfachung angenommen werden, dass das Gaspedal immer maximal betätigt wird. Mit dieser Annahme fällt eine Variable weg und die Kennlinie vereinfacht sich auf den eindimensionalen Fall. Um auf plausible Motorkennwerte zu kommen, wird das Example 2 im Appendix A von [16] herangezogen. Als Näherung wird die Kurve eines Dieselmotors vereinfacht angenommen. Zwischen den Drehzahlen $n = 2000U/min$ und $n = 4000U/min$ wirkt konstantes Moment von $M = 250Nm$. Bis zur kritischen Drehzahl von $n = 5000U/min$ fällt die Kurve dann linear auf $M = 200Nm$ ab. Die unteren Drehzahlbereiche können in der Realität nicht genutzt werden. In der Simulation soll es allerdings möglich sein das Anfahren des Fahrzeugs darstellen zu können. Der Anfahrvorgang ist in der Realität äußerst komplex, zum Beispiel müssen das Einkuppeln, das Schleifen der Kupplung, etc. berücksichtigt werden. Um diesen Problemen aus dem Weg zu gehen, wird angenommen, dass auch in den unteren Drehzahlbereichen das maximale Drehmoment übertragen werden kann. Die Drehzahl-Drehmoment-Kurve

ist in Abbildung 6.7 zu sehen.

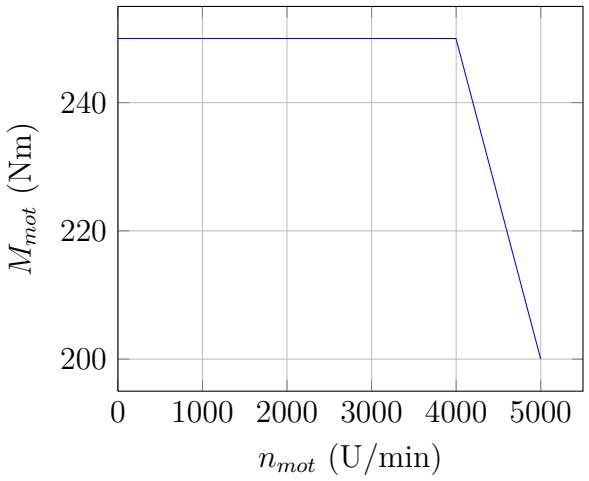


Abbildung 6.7: Kennlinie des Motors in MATLAB.

6.3.2. Hauptgetriebe

Zusätzlich zum Motor wird das Hauptgetriebe in MATLAB abgebildet. Die einfachste Methode, die verschiedenen Übersetzungen zu berücksichtigen, ist das jeweilige Übersetzungsverhältnis direkt einzubringen. Die Drehzahl am Abtrieb, die in CATIA gemessen wird, wird um den entsprechenden Faktor erhöht. Mit dieser Größe wird das Drehmoment aus der Kennlinie ausgelesen und vor der Weitergabe ebenfalls um das Übersetzungsverhältnis vergrößert. Damit ist τ definiert über $n_A = n_{mot}\tau$, wobei n_A die Drehzahl nach dem Getriebe darstellt. Die Zahlenwerte der fünf Getriebestufen stammen ebenfalls aus [16]:

$$\tau = (0.2667, 0.4474, 0.6588, 0.8834, 0.9728).$$

Die Anordnung der Übersetzungsverhältnisse im Vektor τ ist in der Reihenfolge vom ersten Gang (erster Eintrag) bis zum fünften (letzter Eintrag). Die Struktur als Vektor ist gewählt, damit die Gleichungen in MATLAB allgemeingültig sind und der eingelegte Gang den Index des Vektors bestimmt.

Die Definitionen der Kennlinie und der Übersetzungsverhältnisse sollen aus dem Programm, das während der Co-Simulation läuft, ausgelagert werden. Dieser Schritt ist sinnvoll, um einerseits die Simulationszeit zu reduzieren und andererseits eine Trennung der Vorgabe der Motorparameter vom Programm für die tatsächliche Co-Simulation zu gewährleisten. Als zusätzlicher Vorteil ergibt sich, dass die Informationen damit allgemeingültig vorliegen und im Bedarfsfall von anderen Simulatoren benutzt werden kön-

nen. Die Umsetzung besteht darin, dass die Kennlinie in einem eigenen MATLAB-File aufgebaut wird. Die Funktionen von Drehzahl und Drehmoment und die Werte der Übersetzungsparameter werden dann in csv-Dateien geschrieben. Das eigentliche Programm für die Simulation liest die Werte ein.

6.3.3. Höhenprofil der Fahrbahn

Das Höhenprofil der Fahrbahn wird ebenfalls in einem Hilfsprogramm erstellt. Die Kontur der Landschaft ist im allgemeinen Fall eine Funktion in allen drei Raumrichtungen mit $y = y(x, z)$. Da aber bereits im Fahrwegmodell vorgegeben wird, dass Unterschiede in der Höhe zwischen den linken und rechten Rädern vernachlässigt werden, vereinfacht sich die Beziehung zu $y = y(x)$. Ein Beispiel für ein Höhenprofil ist in Abbildung 6.4 zu sehen.

Zur Erstellung des Profils wird ein eigenes Skript erstellt, siehe Listing 8. Für die einfache Definition der Fahrbahn sollen Punkte vorgegeben werden können. Zwischen diesen Koordinaten wird die Kurve linear interpoliert. Damit ist es sehr einfach möglich eine gewünschte Form des Höhenprofils anzugeben. Im einfachsten Fall reicht es aus, zwei Punkte anzugeben. Die Kurven des Höhenprofils werden ebenfalls in csv-Dateien geschrieben. Mit dieser Methode ist es auch denkbar GPS-Daten für die Bestimmung des Profils heranzuziehen.

```
1  numberOfPoints=10000;
2  DataPoints = csvread('landscapeData.csv');
3  x = linspace(DataPoints(1,1), DataPoints(end, 1), numberOfPoints);
4  y = zeros(1,length(x));
5  lastindex = 1;
6  for i=1:size(DataPoints, 1)
7      newindex = find(x==interp1(x,x, DataPoints(i,1),'nearest'));
8      interval = newindex-lastindex;
9      if interval
10         y(lastindex:newindex) = linspace(DataPoints(i-1,2), DataPoints(i
       ,2), interval+1);
11     end
12     lastindex = newindex;
13 end
14 csvwrite('landscape_x.csv',x);
15 csvwrite('landscape_y.csv',y);
```

Listing 8: Quelltext des MATLAB Programms für die Erstellung des Fahrbahnprofils.

6.3.4. MATLAB-Datei zur Co-Simulation

Die von den Hilfsprogrammen erstellten Informationen werden in die MATLAB-Datei für die Co-Simulation eingelesen. Zusätzlich zu den bisher beschriebenen Werten wird ein Verzögerungswert festgelegt. Er gibt an, ab welcher Simulationszeit der Antrieb die Arbeit aufnimmt. Dieses Vorgehen ist sinnvoll, um dem mechanischen System in CATIA Zeit zu geben, die statische Gleichgewichtslage zu erreichen. In Listing 9 ist ein Teil des Quelltextes der Simulation in MATLAB zu sehen.

Im betrachteten Ausschnitt wurden sowohl Variablendefinitionen als auch die Aufrufe der Simulation und das Errorhandling weggelassen. In der while-Schleife finden die Berechnungen statt. Bei jedem Durchlauf dieser Sequenz wird die Zeit um eine Kommunikationsschrittweite erhöht, bis BCVTB das Ende der Simulation bekannt gibt. Im ersten Schritt wird abgefangen, dass der Wert des eingelegten Gangs null ist. Im Standardfall wird vorgegeben, dass der erste Gang eingelegt ist.

```
1  ...
2  while (simulate) %Inputs: 1:Drehzahl, 2 Gang, 3: Position der Hinterachse,
3                   % 4:Position der Vorderachse
4  ...
5    if (simulate)
6          gear = input(2);
7          if (gear == 0)
8                gear=1;
9          end
10         if(simTimWri <= startDelay)
11               output(1) = 0;
12         else
13               output(1) = interp1(n,M,input(1)/(tau(fix(gear))),'linear',
14               'extrap')/(tau(fix(gear))); %Drehmoment
15         end
16         output(2) = interp1(x, y, input(3), 'linear','extrap');
17             %Hoehe der Hinterachse
18         output(3) = interp1(x, y, input(4), 'linear','extrap');
19             % Hoehe der Vorderachse
20         simTimWri = simTimWri + delTim;
21     end
22 end
23 exit;
```

Listing 9: Auszug aus dem Quelltext des MATLAB-Programms für die Co-Simulation.

Darauf folgt die Berechnung der Ausgangsgrößen. Falls die Simulationszeit geringer ist

als die gewählte Verzögerung des Motorstarts, wird kein Drehmoment ausgegeben. Ansonsten wird aus der Drehmoment-Drehzahl-Kurve das Moment ausgelesen und mit dem Übersetzungsverhältnis beaufschlagt. Die Drehzahl wird aus dem ersten Input bezogen und ebenfalls durch das Getriebe manipuliert. Die anderen zwei Outputs sind die Höhen der Vorder- und Hinterachse, die aus der x-y-Kurve mit den Inputs zwei und drei ausgelesen werden. Die Ausgabewerte liegen im Normalfall zwischen zwei Datenpunkten und werden daher linear interpoliert. Die Option „extrap“ erlaubt es zusätzlich die Kurven außerhalb des Definitionsgebiets zu extrapolieren. Zuletzt wird die Zeit aktualisiert.

Es ist zu sehen, dass die parametrische Modellierung verhältnismäßig einfach aufgebaut ist. Einerseits ist das eine Konsequenz aus den beschriebenen Vereinfachungen, andererseits allerdings auch aufgrund der kompakten Art, wie MATLAB mit Daten umgehen kann. Dieses Beispiel lässt einen Einblick in die Möglichkeiten, die unter Einbeziehung von MATLAB denkbar sind, zu. Die Erweiterung auf Kennlinien mehrerer Dimensionen oder mit wesentlich größeren Datenmengen ist ohne viel Aufwand machbar.

6.4. Stateflow

In Statflow wird ein einfaches Fahrermodell implementiert. Die Informationen, die daraus gewonnen werden, werden dann dem MATLAB-Modell für die Berechnung des Drehmoments zur Verfügung gestellt. Es wurde bereits vorgestellt, dass das Gaspedal während der gesamten Simulation als maximal betätigt angenommen wird, was genau genommen bereits eine Modellierung des Fahrers ist. In weiterer Folge müssen Vorgaben gemacht werden, wie das Schaltverhalten aussieht. Dafür wird die Drehzahl-Drehmoment-Kurve des Motors herangezogen. Die ursprüngliche Annahme war, dass der Motor mit Drehzahlen von $n = 2000U/min$ bis $n = 5000U/min$ läuft. Unter dieser Voraussetzung wird definiert, dass der Fahrer bei einer Motordrehzahl von $n = 4500U/min$ auf den nächsthöheren Gang schaltet. Beim Hinunterschalten wird eine Grenzdrehzahl von $n = 2500U/min$ angenommen.

Bei der Umsetzung des Modells ist zuerst darauf zu achten, dass Stateflow die Messung der Drehzahl direkt aus dem CATIA-Modell des Antriebsstrangs bezieht. An dieser Stelle liegt allerdings nicht die Drehzahl des Motors sondern der Antriebswelle vor. Daher müssen die zuvor definierten Werte über die Übersetzungsverhältnisse des Schaltgetriebes angepasst werden. Aus praktischen Gründen werden die Grenzdrehzahlen gerundet. Abbildung 6.8 zeigt das Statechart der Schaltlogik in Stateflow und die jeweiligen Grenzdrehzahlen an den Übergangslinien.

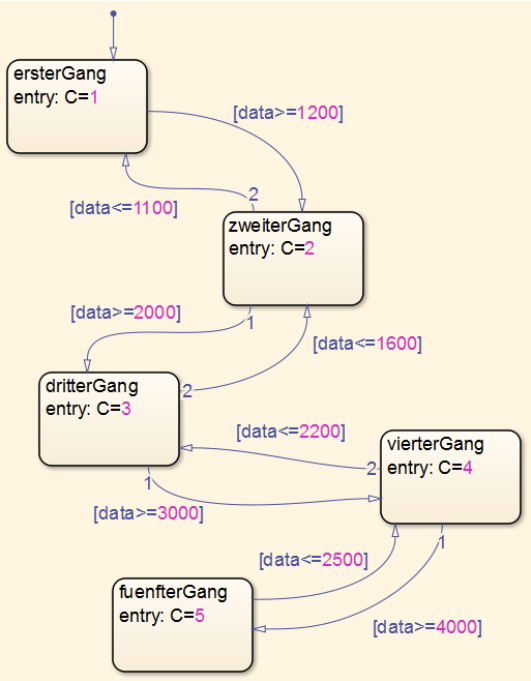


Abbildung 6.8: Statechart des Modells der Schaltlogik in Stateflow.

Die fünf Rechtecke stellen die unterschiedlichen Zustände dar, die im Modell angenommen werden können. An der linken oberen Seite ist der Ausgangspunkt beim Start der Simulation. Wenn der Grenzwert der Drehzahl erreicht wird, wird in den jeweils nächsten Zustand gewechselt.

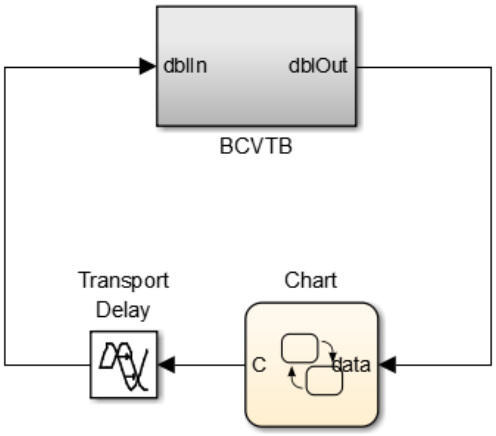


Abbildung 6.9: Simulink Modell mit eingebettetem Statechart und BCVTB Block.

Stateflow überprüft bei jedem Schritt, ob die notwendigen Kriterien erfüllt sind, und passt dementsprechend den Output an. Für das zustandsabhängige Umschalten werden

in Stateflow State-Events definiert. Eine Abbildung von diskreten Zustands-Übergangs-Modellen in Modelica wäre sehr aufwendig, in Stateflow ist die Lösung sehr elegant möglich.

Eingebettet wird das Diagramm in ein Simulink-Modell. An dieser Stelle wird die Verbindung mit BCVTB hergestellt und die Daten ausgetauscht. Im Kreislauf ist zusätzlich ein Delay-Block zwischengeschalten. Er ist dafür zuständig, die auftretende algebraische Schleife aufzubrechen. Abbildung 6.9 zeigt das Simulink Modell.

Das Modell in Simulink besteht nur aus Komponenten, die einmal zu jedem Kommunikationsschritt eine Tätigkeit ausführen. Es bringt keine zusätzlichen Vorteile zwischen den Intervallen zu evaluieren, ob sich die Drehzahl geändert hat, da dieser Input aus dem jeweils letzten zur Verfügung stehenden Datenwert bis zum nächsten Datenaustausch konstant extrapoliert wird. Es kann daher ein diskreter Solver benutzt werden, der pro Kommunikationsschritt einen Zeitschritt tätigt. Mit der Verbindung des diskreten Modells in Stateflow mit dem kontinuierlichen in CATIA entsteht eine hybride Co-Simulation.

6.5. Szenarienrechnungen

Das vorgestellte Modell eines Fahrzeugs wird für die Simulation unterschiedlicher Szenarien herangezogen. Dafür ist es notwendig die Komponenten mit sinnvollen Werten zu belegen. Aus der Literatur werden Beispiele für Fahrzeuge herangezogen.

6.5.1. Systemparameter

Das betrachtete Fahrzeug wird in Anlehnung an Example 2 in [16] gewählt. Die Masse des gesamten Fahrzeugs wird daher mit $m_C = 1150kg$ angegeben. Enthalten sind hier sämtliche Komponenten inklusive der Räder. Die Masse, die dem Modell der Karosserie übergeben wird, muss daher noch um die der Räder reduziert werden. Diese Struktur wird in Anlehnung an Abbildung 5.8 gewählt, die nur zwischen diesen Massen unterscheidet. Daher wird auch den Komponenten des Antriebsstrangs keine Masse gegeben, da diese bereits in der Aufbaumasse enthalten ist.

Die geometrischen Verhältnisse (siehe Abbildung 6.10) sind so bestimmt, dass der Achsabstand $l_{ges} = 2.66m$ beträgt. Der Abstand zwischen Vorderachse und Schwerpunkt beträgt $l_{v,SP} = 1.064m$, die Höhe des Schwerpunkts vom Boden aus gesehen soll etwa $h_{SP} = 0.57m$ betragen. Diese Länge kann allerdings nicht direkt vorgegeben werden, da sie sich aufgrund der Deformation der Aufbaufedern ergibt. Aus diesem Grund wird die

ungedehnte Federlänge mit einem etwas größeren Wert $h_{SP} = 0.63m$ definiert, um die Längenreduktion zu berücksichtigen. Diese Kalibrierung ist notwendig, da das Modell den Parameter der undeformierten Federlänge verlangt. Im Modell der Aufbaumassen muss diese Größe um den Radius reduziert werden, da nur die relative Länge zur Radachse im Modell vorgegeben werden kann. Die Breite des Fahrzeugs beträgt $l_z = 1.83m$.

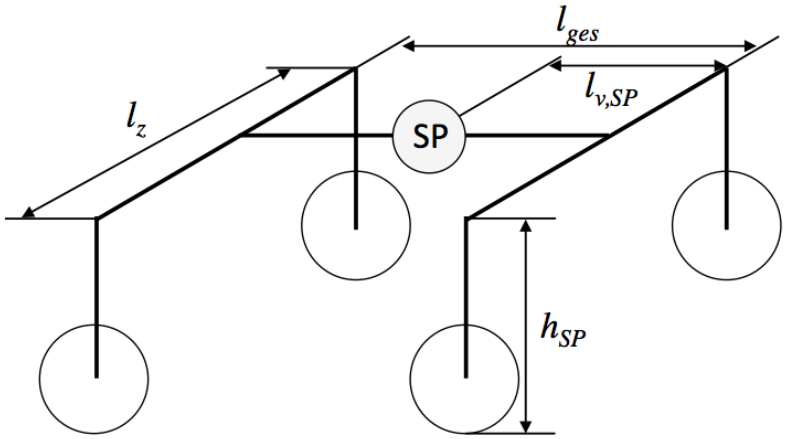


Abbildung 6.10: Geometrie des Fahrzeugs.

Die Daten der Aufbaufedern werden aus [18] entnommen und an einer Achse jeweils gleich angenommen. Die Größen an der Hinterachse betragen $c_h = 30000N/m$ und $d_h = 3500Ns/m$, die der Vorderachse $c_v = 25000N/m$ und $d_v = 2400Ns/m$. Mit diesen Kenngrößen ergibt sich eine Hubeigenfrequenz der Karosserie von etwa $f = 1.5Hz$, was einem praxisgerechten Wert entspricht. Die Werte für die Berechnung des Luftwiderstandes werden aus [16] entnommen. Für die Dichte der Luft wird von einer Höhe von $500m$ ausgegangen. Die Werte für den Luftwiderstand ergeben sich zu $\rho_{Luft} = 1.168kg/m^3$, $c_x = 0.36$ und $S = 2.06m^2$.

Die Räder werden mit Daten aus [18] parametrisiert. Zusätzlich zu den Rädern selbst sind bereits Massen der Aufhängung berücksichtigt. Bei den Vorderrädern betragen die Masse $m_{R,v} = 30kg$ und das Trägheitsmoment $J_{R,v} = 2kgm^2$, bei den Hinterrädern $m_{R,h} = 25kg$ und $J_{R,h} = 1.5kgm^2$. Die Kenngrößen für die Berechnung des Rollwiderstands werden mit dem geschwindigkeitsabhängigen Term $K = 6.5 \cdot 10^{-6}$ und dem konstanten Einfluss $f_0 = 0.013$ gewählt. Die Steifigkeit der Reifen wird aus [18] entnommen und beträgt $c_R = 200kN/m$. Zur Berechnung des Dämpfungsparameters wird die

Bestimmungsgleichung des Dämpfungsfaktors [19] benötigt:

$$D_R = \frac{d_R}{2\sqrt{c_R m_R}}.$$

Es wird angenommen, dass ein alleinstehendes Rad nach der Anregung nur kurz schwingt und wegen der eigenen Dämpfung schnell zur Ruhe kommt. Daher wird für die Dämpfung ein Lehr'sches Dämpfungsmaß von $D_R = 0.5$ angenommen, um ein schnell abklingendes, unterkritisches Schwingen zu erlauben. Mit dieser Annahme errechnen sich die Dämpfungsparameter zu $d_{R,v} = 2449Ns/m$ und $d_{R,h} = 2236Ns/m$.

Schlussendlich fehlen noch die Kenngrößen des Antriebsstrangs. Wie bereits erwähnt, werden die Komponenten masselos angenommen, da sämtliche Massen in der Aufbaumasse enthalten sind. Die Möglichkeit, Körper masselos zu gestalten, ist äußerst praktisch und nicht selbstverständlich. Bei der analytischen Berechnung kann es passieren, dass die Massenmatrix singulär wird und dadurch kein Ergebnis erzielt werden kann. Durch die Modelica-Sprache werden die Abhängigkeiten zwischen Teilsystemen allein durch Bilanzgleichungen angegeben. Wenn ein Teil der Gleichung zu null wird, fällt dieser aus der Berechnung heraus. Die Massenmatrix kann allerdings trotzdem singulär werden. Dass eine Berechnung weiterhin möglich ist, liegt an dem robusten Solver. Der Nachteil an der Vorgehensweise ist, dass Gleichungen bis zur Evaluierung mitgenommen werden und sich damit der Rechenaufwand erhöht. In weiterer Folge werden die elastischen Eigenschaften des Antriebsstrangs genauer beleuchtet.

6.5.2. Elastisches Verhalten des Antriebsstrangs

Zur Auslegung der elastischen Komponenten des Antriebsstrangs werden einfache Annahmen getroffen und an einer Beispielsimulation des Gesamtmodells in Modelica getestet. Der Antriebsstrang ist zwar mit vier Körpern ausgeführt, soll aber als Zwei-Massen-Schwinger simuliert werden (siehe Abschnitt 5.4). Dafür werden die Trägheiten der Seitenwellen sowie die des rechten Teils der Antriebswelle vernachlässigt. Damit ist nur noch ein Körper mit Trägheitsdaten im Modell des Antriebsstrang dezidiert vorhanden. Der zweite Teil des Schwingers ist die Fahrzeugmasse selbst, die über den Radius der Räder auf den Antriebsstrang wirkt. Es sind daher zwei Trägheiten im System vorhanden, die über eine Drehfeder verbunden sind. Sie stellt die Elastizität der Antriebswelle dar. Auf der Abtriebsseite berechnet sich das Massenträgheitsmoment zu

$$J_C = m_C r^2 = 71.875 kgm^2.$$

mit der Wagenmasse m_C und dem Radius r. Auf der Antriebsseite soll das Massenträgheitsmoment von Schwungrad und Motor einen Einfluss auf das Verhalten des Antriebsstrangs haben. Es wird mit einer Trägheit von $J_M = 0.1kgm^2$ vorgegeben, die auf der linken Seite der Drehfeder wirkt. Um auf einen realistischen Wert für die Elastizität zu kommen wird, die Berechnung für die Eigenfrequenz ω des Drehschwingers herangezogen:

$$\omega = \sqrt{\frac{c}{J_C} + \frac{c}{J_M}}.$$

Der Wert c beschreibt die Drehsteifigkeit, J_C und J_M beziehen sich auf die Massenträgheitsmomente auf beiden Seiten der Drehfeder. Das System soll zunächst danach ausgelegt werden, dass eine Eigenfrequenz von $f = 3Hz$ gilt. Dieser Wert wird angenommen, um das an und für sich nichtlineare Phänomen des Schieberuckelns abbilden zu können. Mit $\omega = 2\pi f$ berechnet sich die Drehsteifigkeit zu

$$c = \frac{\omega^2 J_C J_M}{J_C + J_M} = 35.48Nm/rad.$$

Mit den vorgestellten Parametern wird eine Simulation durchgeführt. Zur Vereinfachung wird das komplexe Verhalten des Antriebs durch ein konstantes Eingangsdrehmoment von $M_A = 250Nm$ ersetzt. Weiters wird das Profil der Fahrbahn als eben angenommen. In Abbildung 6.11 ist der relative Verdrehwinkel der Drehfeder zu sehen.

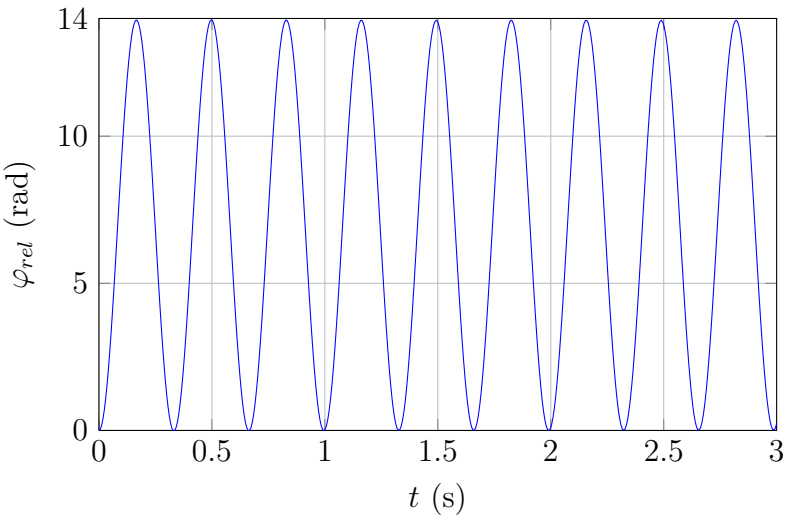


Abbildung 6.11: Verdrehwinkel der Drehfeder φ_{rel} über der Zeit t.

Es ist zu erkennen, dass die geforderte Frequenz von $3Hz$ gut eingehalten wird. Das Problem, das sich durch diese Herangehensweise ergibt, ist, dass der relative Verdreh-

winkel sehr groß wird. Ein Winkel von $\varphi_{rel} = 14rad \approx 800°$ entspricht mehr als zwei ganzen Umdrehungen. Eine normale Welle würde einer derartigen Verdrehung natürlich nicht standhalten beziehungsweise bei den anliegenden Momenten diese nicht zulassen.

Daher wird der Ansatz für die Auslegung verworfen und stattdessen die maximale Auslenkung von $\varphi_{rel} = 0.1°$ bei einem maximalen Drehmoment von $M_A = 937Nm$ festgelegt. Das Antriebsmoment entspricht jenem des Motormodells im ersten Gang. Damit erhält man eine Drehsteifigkeit von

$$c = \frac{M_A}{\varphi_{rel}} = 551000Nm/rad.$$

Mit diesem Wert ergibt sich allerdings eine Eigenfrequenz von ca. $380Hz$. Bei Simulationszeiten von mehreren Sekunden werden die Schwingungen aufgrund der niedrigen Amplitude vom Solver nicht berechnet. Falls spezielles Interesse am Drehschwingungsverhalten besteht, kann die Dauer der Simulation entsprechend herabgesetzt werden. Um dieses Verhalten zu demonstrieren, ist in Abbildung 6.12 (blaue Linie) wieder der

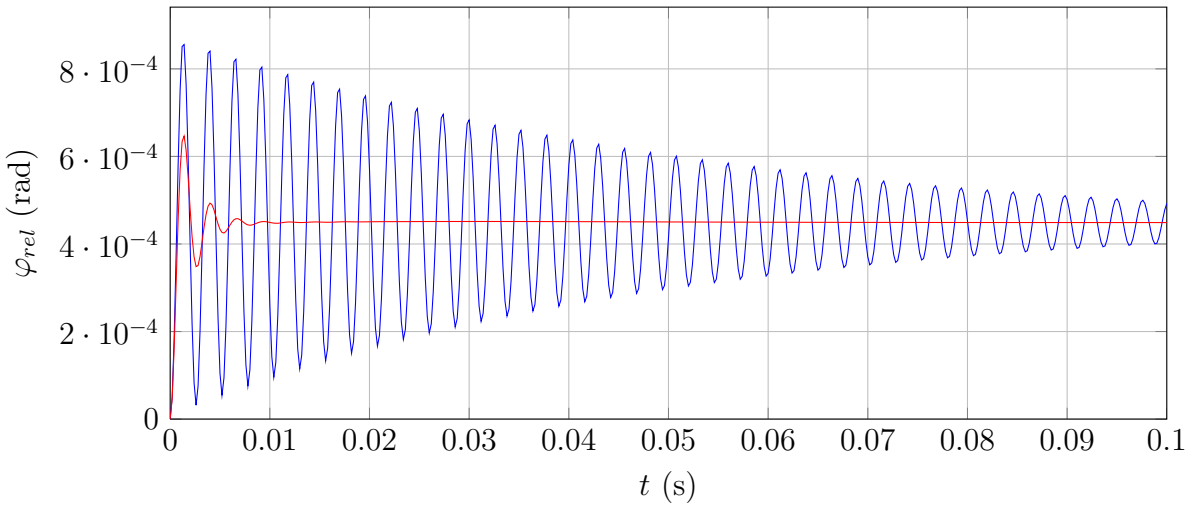


Abbildung 6.12: Verdrehwinkel der Drehfeder φ_{rel} über der Zeit t: In Blau $d = 0Nms/rad$, in Rot mit $d = 100Nms/rad$.

Relativwinkel zu sehen, dieses Mal wurde die Simulationsdauer mit $t_{Ende} = 0.1s$ verkürzt gewählt. Es ist zu erkennen, dass etwa 38 Schwingungen in einer Zehntelsekunde auftreten, was der Frequenz von $380Hz$ entspricht. Um Probleme, die aufgrund des Schwingungsverhaltens auftreten können, zu vermeiden, wird zusätzlich zur Steifigkeit eine Dämpfung angegeben. Diese soll erreichen, dass die Schwingung schnell abklingt, aber kein aperiodischer Fall auftritt. Zu diesem Zweck wurde der Wert mit $d = 100Nms/rad$

gewählt. Die rote Linie in Abbildung 6.12 zeigt den Verdrehwinkel mit der zusätzlichen Dämpfung.

In den weiteren Untersuchungen werden die kleinskaligen Änderungen keine Rolle spielen, dementsprechend werden hohe Frequenzen nicht genauer betrachtet. Durch den robusten Solver in CATIA DBM sind die eben vorgestellten steifen Systeme weiter behandelbar. Es ist jedoch zu bedenken, dass weitere Änderungen im Modell hier zu Problemen führen können.

6.5.3. Szenario 1: Das Fahren auf ebener Fahrbahn

Durch die Studien des Elastizitätsverhaltens des Antriebsstrangs sind die Parameter des Modells definiert. In weiterer Folge wird das Verhalten des Gesamtmodells während der Co-Simulation beleuchtet. Als erstes und einfachstes Beispiel wird das Fahren auf ebener Fahrbahn präsentiert. Die Einflüsse der Bestimmung der Fahrbahnhöhe fallen somit weg. Dieser Fall kann bei späteren Szenarien als Referenz herangezogen werden.

Die Verzögerungszeit des Motordrehmoments ist mit $t_{start} = 1s$ vorgegeben. Bis dahin hat das mechanische Modell Zeit eine Ruheposition einzunehmen. Bei t_{start} beginnt das Motordrehmoment an der Antriebswelle zu wirken. Die Startzeit der Simulation ist bei $t_0 = 0s$, das Ende ist bei $t_{Ende} = 10s$ vorgegeben. Während der Berechnung stößt BCVTB 1000 Mal die Kommunikation zwischen den Subprogrammen an.

Zuerst werden die Daten betrachtet, die CATIA mit BCVTB austauscht. Das Drehmoment und die Drehzahl sind in Abbildung 6.13 gezeigt. Die in blau gezeichnete Kurve stellt das Antriebsmoment dar, die rote die gemessene Drehzahl an der Antriebswelle. Bis zum Zeitpunkt $t = 1s$ liegt kein Antrieb vor. Danach springt das Motormoment auf den vorgegebenen Wert von $M = 250Nm$ und umgerechnet durch die Übersetzung des ersten Gangs das Antriebsmoment auf $M_A = 937Nm$. Das resultiert darin, dass sich das Fahrzeug in Bewegung setzt und damit auch die Drehzahl an der Seitenwelle zunimmt. Bei Annäherung an die Grenzdrehzahl weicht das Antriebsmoment vom konstanten Wert ab und sinkt linear. Wenn die Schaltdrehzahl erreicht wird, fällt das Eingangsmoment schlagartig ab und nimmt den konstanten Wert des zweiten Gangs an. An der Drehzahl ist diese Änderung in deren Steigung zu sehen, die zum Zeitpunkt des Schaltens einen Knick macht. Während der zehnsekündigen Simulation treten insgesamt drei Schaltvorgänge auf, am Ende ist der vierte Gang eingekuppelt. Da sich die Übersetzung immer mehr einem eins-zu-eins-Verhältnis annähert, sinkt auch das Drehmoment. Die Drehzahl erreicht am Ende der Simulation einen Wert von $n = 3820U/min$.

Die nächsten Ergebnisse, die betrachtet werden, sind die Radlasten an den Vorder-

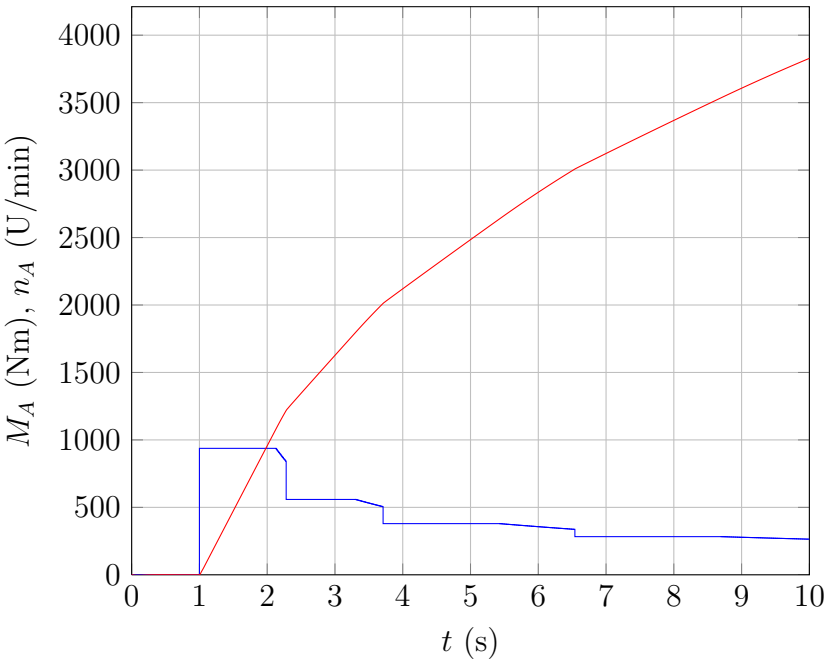


Abbildung 6.13: Motorkenngrößen der Simulation auf ebener Fahrbahn. Blau: Antriebsmoment in Nm, rot: Antriebsdrehzahl in U/min.

und Hinterrädern. Die blaue Linie in Abbildung 6.14 zeigt die vertikale Komponente der Kraft an einem Rad der Hinterachse, die rote ein Rad der Vorderachse. Zu Beginn der Simulation sieht man den Einschwingvorgang von Aufbaufederung und Reifen. Beim Start der Beschleunigung des Fahrzeugs ist zu erkennen, dass noch kein stabiler Zustand erreicht wurde. Durch den Sprung in der Antriebsleistung springt auch die Beschleunigung, was sich wiederum auf die Radlasten auswirkt. In der Zeit nach den Schaltvorgängen ist das Schwingungsverhalten der elastischen Komponenten zu erkennen. Mit höherem Gang wird der Unterschied der Kräfte zwischen den zwei Achsen größer. Dieser Unterschied resultiert daraus, dass der Schwerpunkt nicht symmetrisch angeordnet ist, sondern weiter vorne liegt. Bei einer Simulation des stillstehenden Fahrzeugs zeigt sich, dass die eingependelte Vertikalkraft bei den Vorderrädern bei $F_{Y,v} = 2273N$ und an der hinteren Achse bei $F_{Y,h} = 3367N$ liegt. Mit steigendem Gang und sinkender Beschleunigung nähern sich die Kurven diesen Werten an, erreichen sie aber erst, wenn die Last- und Antriebskräfte gegengesetzt gleich groß werden, also die Beschleunigung verschwindet.

Die Kinematik des Fahrzeugs ist in Abbildung 6.15 zu sehen. Insgesamt legt es in den zehn Sekunden eine Distanz von $s = 196m$ zurück und erreicht eine Endgeschwindigkeit von $v = 122km/h$. Eine Geschwindigkeit von $v = 100km/h$ wird zum Zeitpunkt $t = 7.03s$ erreicht, also $\Delta t = 6.03s$ nach Beginn des Beschleunigungsvorgangs. Dieser Wert

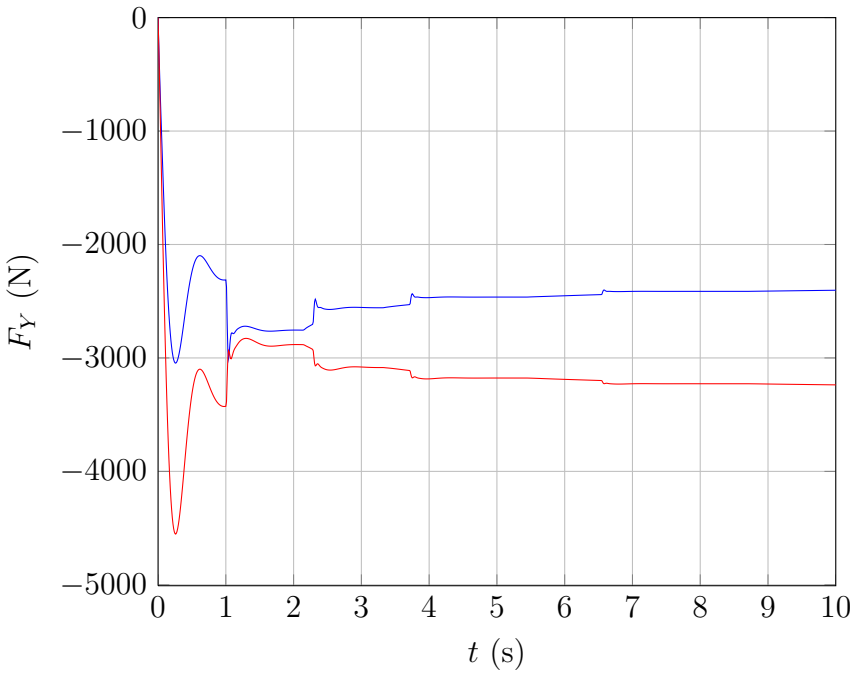


Abbildung 6.14: Radlasten bei der Simulation auf ebener Fahrbahn. Blau: Hinterrad, rot: Vorderrad.

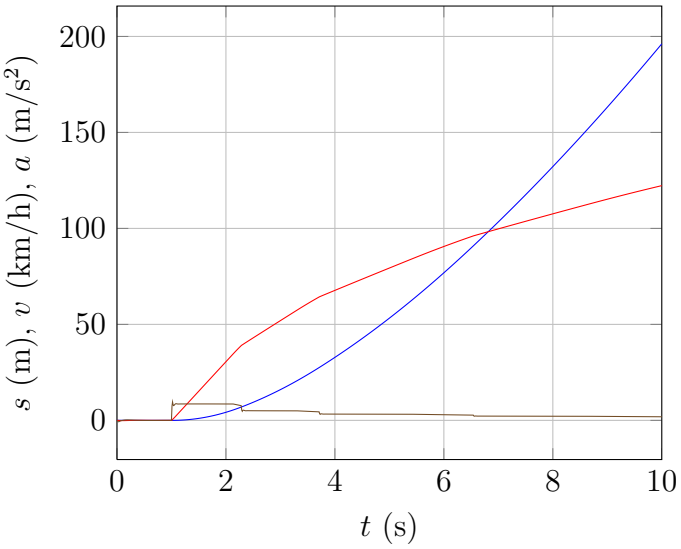


Abbildung 6.15: Kinematik des Fahrzeugs auf ebener Fahrbahn. Blau: zurückgelegte Weglänge in m, rot: Geschwindigkeit in km/h, braun: Beschleunigung in m/s^2.

ist für ein Mittelklassefahrzeug sehr niedrig. Diese Tatsache resultiert vor allem daraus, dass die Leistungskurve in den unteren Drehzahlbereichen stark vereinfacht angenommen

wurde. Das Fehlen eines Modells des Einkuppelvorgangs und die hohen Drehmomente zu Beginn führen zu diesem Ergebnis.

6.5.4. Szenario 2: Anstieg der Fahrbahn

Als nächstes Szenario wird eine ansteigende Fahrbahn wie in Abbildung 6.16 herangezogen. Das Profil wird so gewählt, dass sich zu Beginn das Fahrzeug auf ebenem Boden befindet, damit es nicht zu rollen beginnt. Nach einer Distanz von $s = 10m$ wird eine relativ starke Steigung von $k = 0.2$ vorgegeben. Die Änderung der Steigung erfolgt abrupt, es tritt eine nicht differenzierbare Stelle auf, die in CATIA geglättet wird. Die Simulationsparameter werden gleich wie bei der Referenzsimulation gewählt, also ein Kommunikationsintervall von 0.01.

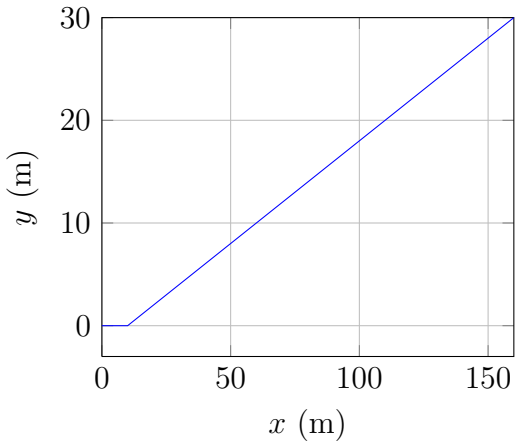


Abbildung 6.16: Fahrbahnprofil mit linearer Steigung von 20% ab $x = 10m$.

Das Modell reagiert äußerst sensibel auf die Vorgabe des Fahrbahnprofils (Vergleich Abbildung 6.17). Dieses Verhalten rührt daher, dass durch die Festsetzung eines Ortes Zwangskräfte auftreten, um das System auf der vorgegebenen geometrischen Bahn zu halten. Die vorgegebene Lage muss zwei Mal differenziert werden, um die Beschleunigung zu erhalten. Die Beschleunigung definiert allerdings die Trägheitskräfte. Daher können kleine, sogar aus der numerischen Berechnung resultierende Fehler zu großen Reaktionskräften führen. Im schlimmsten Fall kommt es zu einer Resonanz und der Fehler schaukelt sich über den Simulationsverlauf auf. Je besser die Glättung des Signals ist, desto eher kann dieser Fehler vermindert werden.

In Abbildung 6.17 sind zwei Ergebnisse derselben Variablen aus zwei verschiedenen Simulationsläufen abgebildet. Die Linien zeigen die Verläufe der Radlast an einem hinteren Rades, in Rot bei der Simulation mit 1000 Kommunikationsschritten, in Blau mit

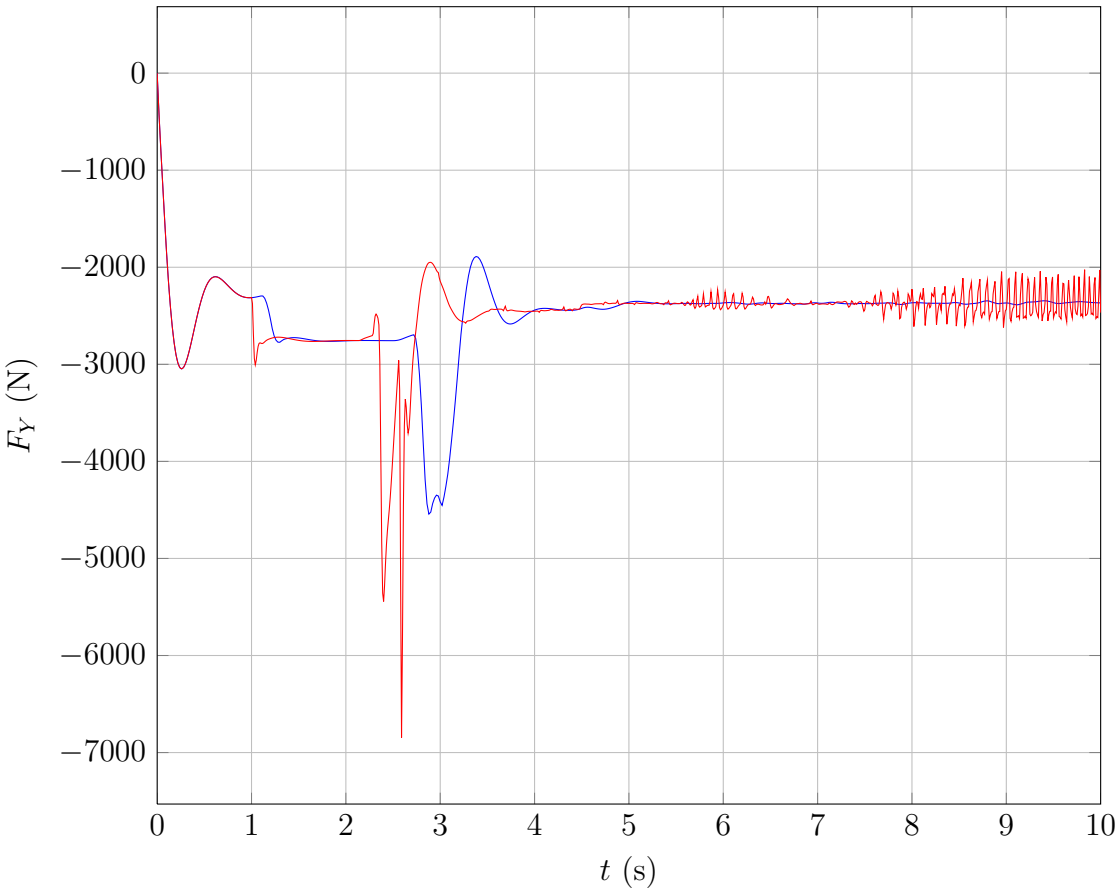


Abbildung 6.17: Radlast auf ansteigender Fahrbahn. Rote Linie: 1000 Kommunikationsschritte, blaue Linie: 100.

lediglich 100. Es ist auf einen Blick zu erkennen, dass die Änderung dieses Simulationsparameters einen großen Einfluss auf das Ergebnis besitzt. Zu Beginn der Simulation sind die Kurven nahezu ident. Das resultiert daraus, dass noch keine relevanten Informationen von BCVTB an CATIA gesandt wurden. Zum Zeitpunkt $t = 1s$ wird wieder der Antrieb aktiviert, hier sieht man bereits die Verzögerung der blauen im Vergleich zur roten Linie. Dieser Umstand ist nicht verwunderlich. Wie in Abschnitt 3.4 ausführlich beschrieben, benötigt die Information in BCVTB mindestens einen Zeitschritt, um das gewünschte Ziel zu erreichen. Der Einfluss wird bei größeren Intervallen damit stärker. Bei der Vorgabe des Drehmoments ist dieser Effekt störend, aber nicht dramatisch. Das System wird lediglich langsamer.

Bei der Steuerung des Fahrbahnprofils ist die Auswirkung zu Beginn der Steigung zu sehen. Es zeigt sich ein zeitlicher Verzug, sogar wesentlich gravierender. Hier ist diese Totzeit allerdings unproblematisch, da dadurch der Hügel ein paar Meter zurückversetzt

wird. Wird mit echten GPS-Daten gearbeitet, wäre eine Unschärfe in ähnlicher Größenordnung sowieso nicht zu vermeiden. Der Fehler, der aus der verspäteten Reaktion auf Geschwindigkeitsänderungen resultiert, ist ebenfalls vernachlässigbar klein, da sich die Beträge der Beschleunigungen in Grenzen halten.

Problematisch wird der Einfluss der Kommunikationsschrittweite bei den resultierenden Kräften aufgrund kleiner Fehler in der Vorgabe der Fahrbahn. In Abbildung 6.17 sieht man, dass zu Beginn der Steigung ein großer Ausschlag auftritt. Dieses Verhalten ist physikalisch korrekt und daher erwünscht. Die Größenordnungen der Radlasten aus den zwei Simulationen unterscheiden sich allerdings stark. Bei der hohen Schrittweite wird eine maximale Kraft von etwa $F_Y = 4500N$ erreicht, während es bei der Rechnung mit wesentlich geringerer Intervalllänge $F_Y = 6800N$ sind. Eine weitere Auswirkung ist, dass sich in der Simulation unterschiedliche Schwingungen ergeben. Der Effekt tritt bei beiden Fällen ein, allerdings sind sowohl Frequenz als auch Amplitude bei hoher Kommunikationsrate wesentlich größer. Außerdem ist dabei ein Verstärkungseffekt zu beobachten. Dieses Verhalten ist nicht erwünscht und führt im schlimmsten Fall zur Divergenz der Lösung. Es werden außerdem die Ergebnisse verfälscht, da die Radlasten zur Berechnung von zwei Widerstandsgrößen herangezogen wird.

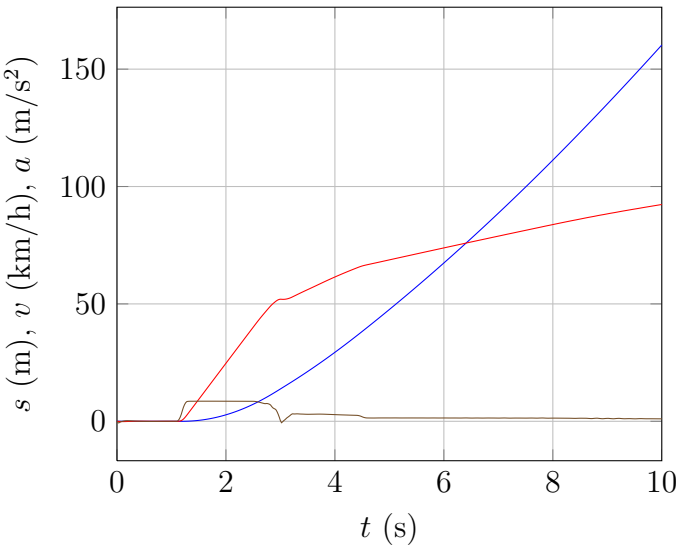


Abbildung 6.18: Kinematik des Fahrzeugs auf linear ansteigender Fahrbahn. Blau: zurückgelegte Weglänge in m, rot: Geschwindigkeit in km/h, braun: Beschleunigung in m/s^2.

Als Konsequenz des Verhaltens kann die hohe Kommunikationsfrequenz nicht aufrecht erhalten werden. Als Kompromiss aus beiden Fehlerquellen wird mit einer Rate

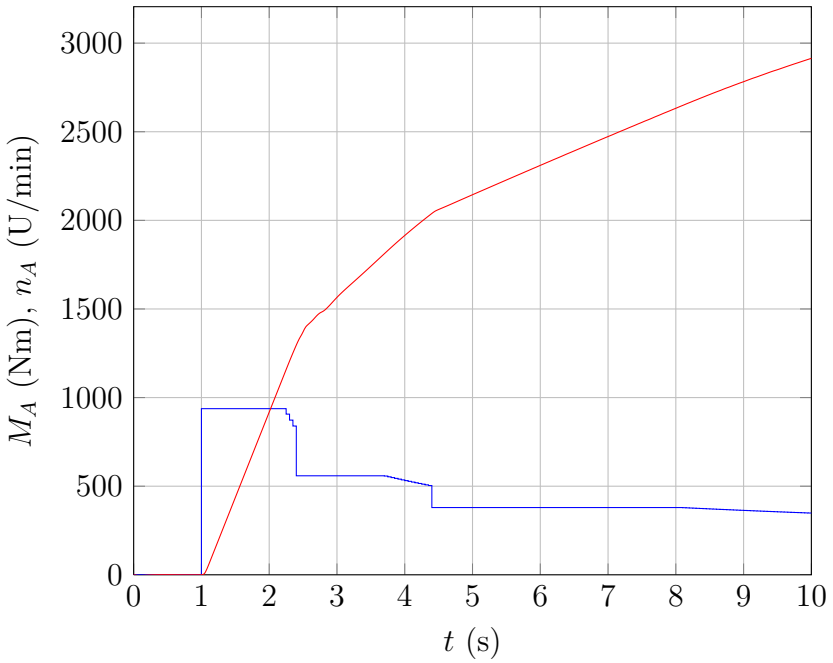


Abbildung 6.19: Motorkenngrößen der Simulation auf linear ansteigender Fahrbahn. Blau: Antriebsmoment in Nm, rot: Antriebsdrehzahl in U/min.

von 200 Kommunikationen pro zehn Sekunden gearbeitet. In Folgearbeiten könnte über Möglichkeiten nachgedacht werden, die Vorgabe eines Höhenprofils auf andere Art durchzuführen. Ein Ansatz wäre nicht die Fahrbahnsteigung selbst vorzugeben, sondern den Winkel, der am Rad anliegt.

In den Abbildungen 6.18 und 6.19 sind die charakteristischen Größen der Simulation mit der angepassten Kommunikationsschrittweite aufgetragen. Es wird insgesamt aufgrund der Steigung eine im Vergleich zu Szenario 1 geringere Distanz von $s = 159m$ zurückgelegt. Die maximale Geschwindigkeit wird zum Ende der Simulation erreicht und beträgt $v = 91km/h$. Am Beginn des Hügels erfährt die Geschwindigkeit einen kleinen Rückgang, der allerdings nicht ausreicht, um zurückzuschalten. An der Kurve des Eingangsmoments hat sich nicht viel geändert, außer dass lediglich der dritte Gang erreicht wird, was aufgrund der niedrigen Geschwindigkeit einleuchtend ist.

6.5.5. Szenario 3: Parabolisch ansteigende Fahrbahn

Um die Eigenschaften des Modells weiter zu untersuchen, wird der Fahrbahnverlauf geändert. Anstelle der linearen Steigung wird ein parabolischer Straßenverlauf vorgegeben,

siehe Abbildung 6.20. Die Kurve entspricht der Funktion

$$y(x) = \begin{cases} \frac{x^2}{300}, & x \geq 0 \\ 0, & x < 0 \end{cases} .$$

Die Form wird gewählt, um zu zeigen, dass durch den erhöhten Widerstand beim Bergauffahren das Fahrzeug nicht unbegrenzt beschleunigen kann. Wenn die Steigung zu groß wird, sinkt die Drehzahl wieder und ein Herunterschalten wird notwendig. In Abbildung 6.21 sind die Simulationsergebnisse für den Antrieb zu sehen.

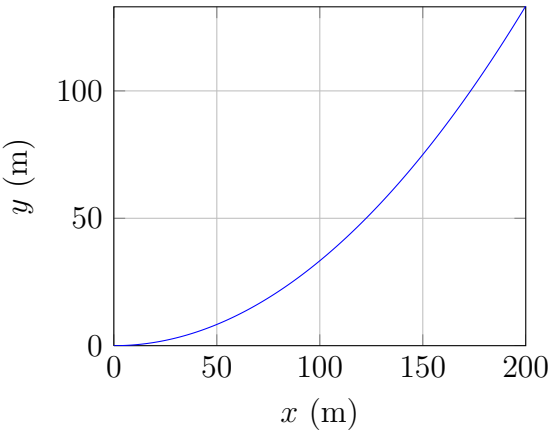


Abbildung 6.20: Fahrbahnprofil mit parabolischer Steigung.

Mit der blauen Kurve ist die Drehzahl der Antriebswelle aufgetragen. Nach zwei Schaltvorgängen, die sich durch Knicke erkennbar geben, erreicht die Drehzahl ein Maximum. Dadurch, dass die Steigung weiter zunimmt, muss die Drehzahl abnehmen. Wenn die Drehzahl unter einen Grenzwert fällt, wird der Gang gewechselt, um das Antriebsmoment zu erhöhen. Die blaue Linie knickt dadurch wieder ab, sinkt aber weiter. Bei einer längeren Simulationslaufzeit würde wieder der erste Gang erreicht werden.

Die rote Kurve zeigt das Antriebsmoment, wie es vom BCVTB-Block ausgegeben wird. Zu Beginn erinnert das Verhalten an das der Referenzsimulation. Dadurch, dass im dritten Gang die Drehzahl nicht mehr weiter steigt, wird auch nicht weiter geschalten. Beim Erreichen der unteren Grenzdrehzahl wird wieder zurück in den zweiten Gang geschalten. Dieser Sprung passiert instantan, da in der Motorkennlinie kein abnehmendes Moment in den unteren Drehzahlbereichen auftritt.

Zur Veranschaulichung des Fahrbahnverlaufs in CATIA wird in Abbildung 6.22 der Output aus BCVTB für die Höheninformationen der Hinterachse gezeigt. Zusätzlich ist das geglättete Signal abgebildet. Am Vergleich ist zu sehen, dass in der Tat aus der Trep-

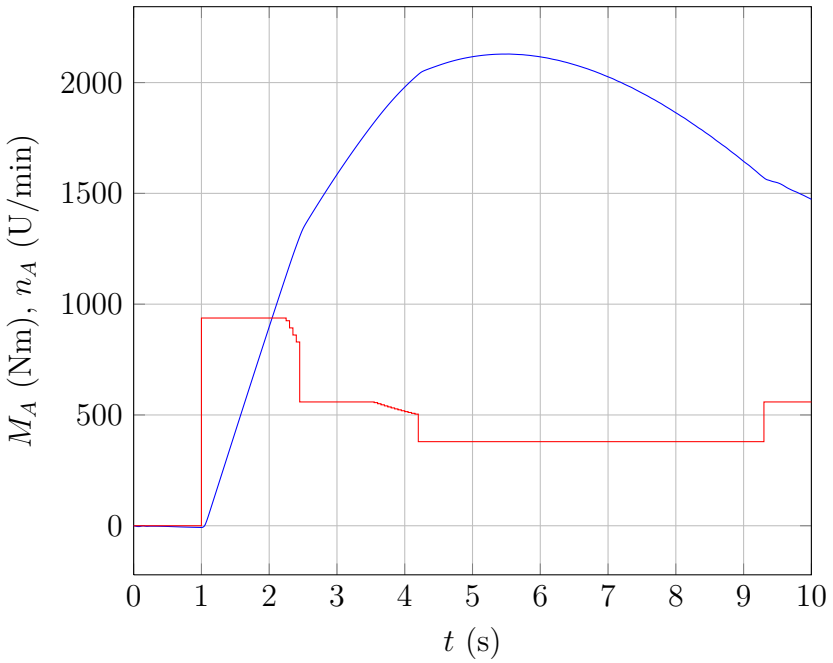


Abbildung 6.21: Kenngrößen des Antriebs bei der Simulation mit parabolischem Fahrbahnverlauf. Blaue Linie: Antriebsdrehzahl in U/min, rote: Antriebsdrehmoment in Nm.

penfunktion eine glatte Kurve entsteht. Sie ist in der Zeit nach hinten versetzt, reagiert also langsamer auf Änderungen. Dieses Verhalten tritt vor allem bei der Glättung des Antriebsmoments auf, da die Sprünge beim Schalten verschleppt werden.

Das Verlauf des Höhenprofils ist keine Parabel, obwohl das in der Definition der Fahrbahn gefordert wird. Der Umstand erklärt sich damit, dass nicht das Fahrbahnprofil selbst abgebildet ist, sondern der Verlauf der Höhe über die Zeit. Daher hat die Geschwindigkeit des Fahrzeugs einen Einfluss. Hier zeigt sich auch das wirkliche Verhalten des Glättungsmechanismus. Er ist nicht dafür da, die durch die Kommunikation entstandene Treppenfunktion in den Ausgangszustand zurückzuversetzen, sondern das von der Geschwindigkeit abhängige Signal zu glätten. Hier zeigen sich ebenso Schwächen, da bei großen Beschleunigungen auch die geglättete Funktion nicht die selbe Qualität aufweisen kann wie bei konstanter Geschwindigkeit. Aus diesem Umstand resultiert mitunter das unerwünschte Verhalten, das im vorigen Szenario besprochen worden ist.

Wie in Abbildung 6.23 zu sehen ist, nimmt die zurückgelegte Distanz erwartungsgemäß weiter ab und liegt bei $s = 123m$, wobei eine Höhe von $h = 49m$ erreicht wird. Die maximale Geschwindigkeit wird bei $t = 4.88s$ erreicht und beträgt $v = 65km/h$.

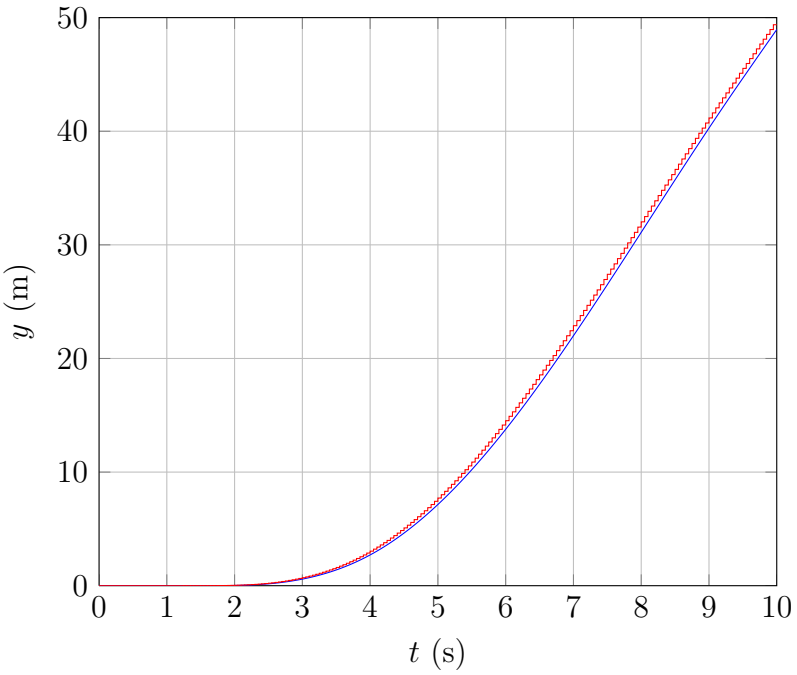


Abbildung 6.22: Höhe der Hinterachse in y-Richtung bei der Simulation mit parabolischem Straßenverlauf. Rot: Signal aus BCVTB, blau: geglättete Funktion.

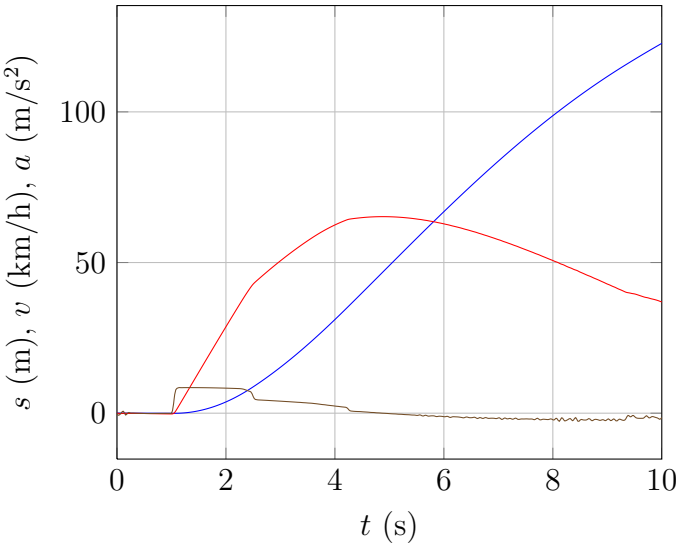


Abbildung 6.23: Kinematik des Fahrzeugs auf parabolisch ansteigender Fahrbahn. Blau: zurückgelegte Weglänge in m, rot: Geschwindigkeit in km/h, braun: Beschleunigung in m/s^2.

6.5.6. Szenario 4: Kuppe und abschüssige Fahrbahn

Der letzte Beispielfall ist der der abschüssigen Fahrbahn. Der gewählte Fahrbahnverlauf ist ähnlich dem des Beispiels aus Szenario 2. Zu Beginn startet das Fahrzeug auf einer Ebene. Nach $s = 20m$ beginnt die Fahrbahn abzufallen. Wie im anderen Beispiel ist die Kante nicht abgerundet. Die Steigung des anschließenden abfallenden Teils beträgt $k = -0.2$. Der Fahrbahnverlauf ist in Abbildung 6.24 zu sehen.

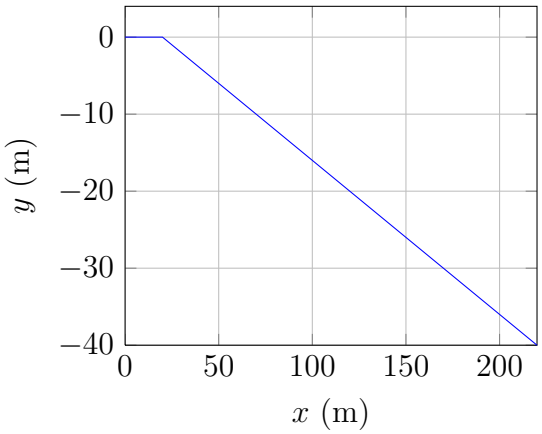


Abbildung 6.24: Fahrbahnprofil mit abschüssiger Steigung $k = -0.2$ ab $x = 20m$.

Das Fahren über eine Kuppe kann problematisch sein und zu unphysikalischen Simulationsergebnissen führen. Die Kuppe ist dadurch verwirklicht, dass von der zuerst ebenen Fahrbahn eine Steigung in negativer y-Richtung vorgegeben wird. Beim Überfahren der Kuppe treten Beschleunigungskräfte auf, die der Schwerkraft entgegenwirken. Wenn an einem der Räder dieser Fall eintritt, verliert es die Haftung mit der Straße und hebt ab. Deshalb sind als erstes Ergebnis die Radlasten an Vorder- und Hinterachse in Abbildung 6.25 zu sehen.

Die beiden Kurven zeigen prinzipiell ein ähnliches Verhalten wie bei der Referenzsimulation. Am Anfang ist das Einschwingen zu sehen. Zum Zeitpunkt des Starts des Antriebs verschieben sich die Kräfte zueinander. Da die Kante erst später als in Szenario 2 auftritt, kann das Schwingungsverhalten länger beobachtet werden. Mit der Verschiebung der Gleichgewichtsposition ist das Schalten vom ersten in den zweiten Gang erkennbar. Der große Ausschlag bei $t = 3.2s$ stellt den Beginn der Steigung dar. Es ist offensichtlich, dass die beiden Radlasten weit über den Nullpunkt hinweg geschoben werden. In der Realität würde das Rad den Kontakt mit dem Boden verlieren und abheben. An dieser Stelle stößt das Modell an seine Grenzen. Die Kraft, die das Fahrzeug von der Fahrbahn wegbewegen würde, wird vom Fahrwegmodell aufgenommen und in die

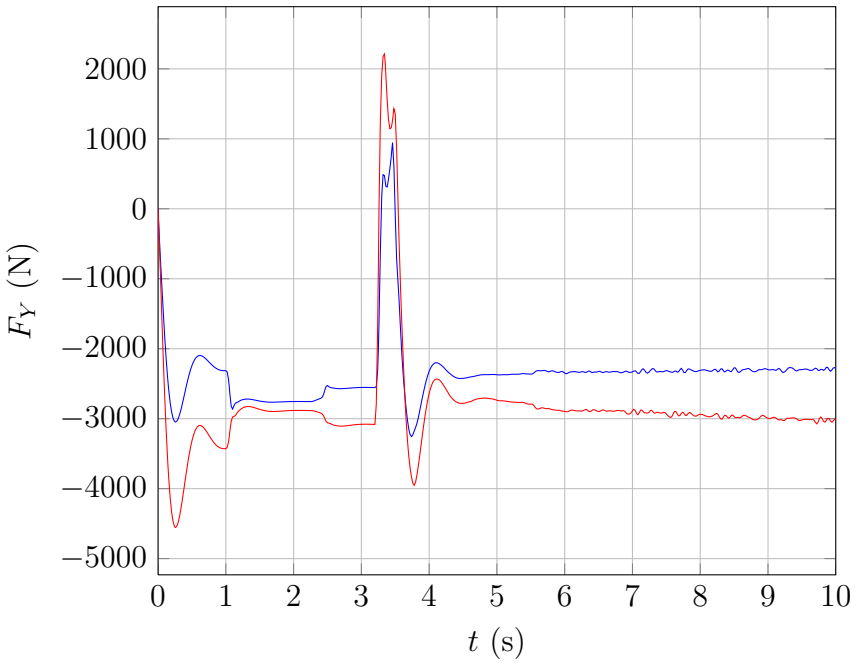


Abbildung 6.25: Radlasten an den Vorder- und Hinterrädern. Blau: hinten, rot: vorne.

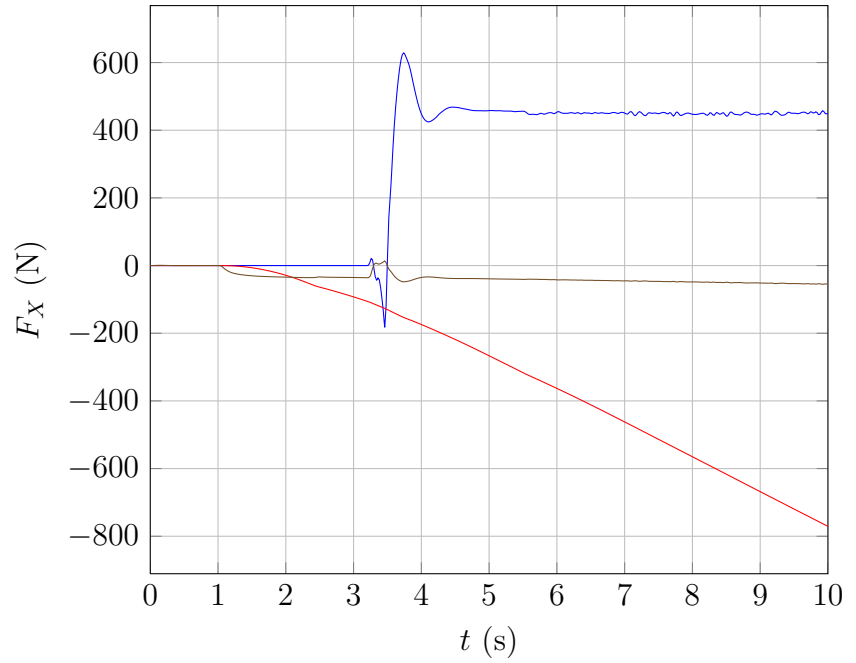


Abbildung 6.26: Die Widerstandskräfte bei der Simulation mit abschüssiger Fahrbahn. Blau: Fahrbahnsteigung, rot: Luftwiderstand, braun: Rollwiderstand.

World-Komponente geleitet. Die Umsetzung des fallenden Fahrzeugs wird nicht weiter behandelt. Eine mögliche Implementierung wäre, dass über Co-Simulation zwischen zwei

Modelica-Modellen gewechselt wird, dies entspräche dann einem strukurvariablen Modell. Je nachdem, welcher Fall auftritt, könnten unterschiedliche Modelle zur Berechnung herangezogen werden.

Zur Demonstration der Widerstandskräfte werden in Abbildung 6.26 die Kräfte in x-Richtung an einem Hinterrad gezeigt. Die blaue Kurve zeigt den Widerstand aufgrund der Fahrbahnsteigung. Am Vorzeichen der Kraft ist zu erkennen, dass es sich bei abfallender Fahrbahn nicht um einen Widerstand handelt, sondern um eine treibende Kraft. Dieses Ergebnis ist zu erwarten, da die abschüssige Fahrbahn das Fahrzeug beschleunigt. Der Richtungswechsel an der Kante resultiert aus der Richtung der Normalkraft, wie oben beschrieben. Da die Steigung beim abfallenden Teil konstant ist, pendelt sich auch die resultierende Kraft auf einen konstanten Wert ein. Die Schwingung nach dem Zeitpunkt $t = 5s$ erklärt sich aus dem Verhalten der vertikalen Kraft, die stark von kleinen Fehlern beeinflusst wird (siehe Szenario 2).

Die rote Linie beschreibt den Luftwiderstand des Fahrzeugs. Durch die quadratische Abhängigkeit von der Geschwindigkeit steigt die Widerstandskraft mit fortschreitender Simulationsdauer stark an.

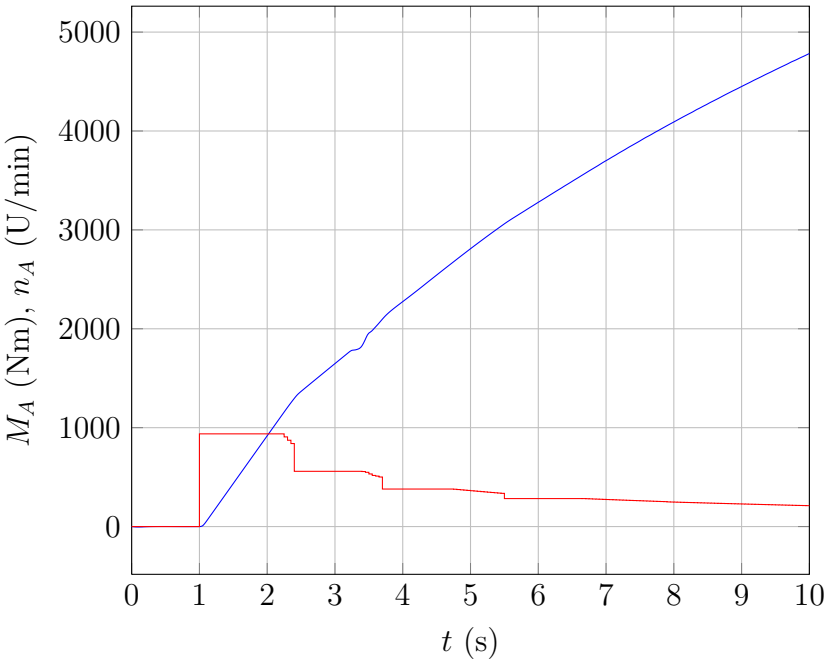


Abbildung 6.27: Kenngrößen des Antriebs bei der Simulation mit linear abfallendem Fahrbahnverlauf. Blaue Linie: Antriebsdrehzahl in U/min, rote: Antriebsdrehmoment in Nm.

Im Gegensatz dazu ist mit braun der Rollwiderstand aufgetragen. Dieser besitzt

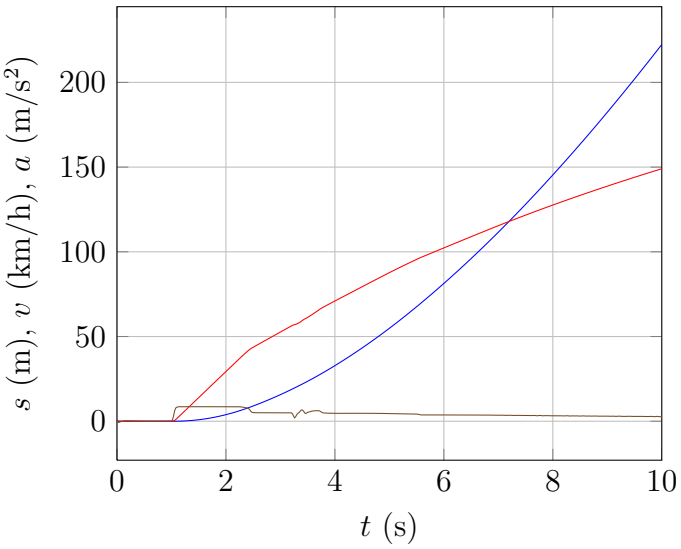


Abbildung 6.28: Kinematik des Fahrzeugs auf linear abschüssiger Fahrbahn. Blau: zurückgelegte Weglänge in m, rot: Geschwindigkeit in km/h, braun: Beschleunigung in m/s^2.

einen geschwindigkeitsabhängigen Term und einen konstanten, der künstlich über die Arkustangens-Funktion an die Geschwindigkeit gekoppelt wird. Dieses Verhalten ist zu sehen, wenn sich das Fahrzeug in Bewegung setzt. Der konstante Wert wird schnell erreicht, die lineare Abhängigkeit von der Geschwindigkeit fällt wegen des kleinen Beiwerts schwach aus.

In den Abbildung 6.27 und 6.28 sind die charakteristischen Größen der Simulation zu sehen. Mit $s = 222m$ legt das Fahrzeug auf der abschüssigen Fahrbahn die größte Distanz aus allen Szenarien zurück. Auch die Geschwindigkeit ist mit $v = 149km/h$ die, größte die erreicht wurde. Durch die hohe Drehzahl wird in den fünften und letzten Gang geschaltet und die Drehzahl am Ende der Simulation ist mit $n = 4783U/min$ nahe am Maximum.

6.6. Schlussfolgerungen

Die vorgestellten Szenarien demonstrieren die wesentlichen Eigenschaften des Modells. Die mechanischen Komponenten in Modelica arbeiten in ihren Grenzen so wie erwartet. Der Einsatz der Co-Simulation zeigt, dass komplexe Problemstellungen einfach angegangen werden können. Die Grenzen wurden aufgezeigt und Probleme wurden besprochen. Im Speziellen ist die Wahl der Kommunikationsschrittweite problematisch und sollte in weiteren Studien näher untersucht werden. Hier kann die Änderung des Fahrwegmodells

Abhilfe schaffen. Das Ziel ist ein direktes Vorgeben der Position zu vermeiden, da durch die doppelte Ableitung kleine Fehler akkumuliert werden.

Das vorgestellte Modell eines Fahrzeugs stellt ein Rahmenwerk dar. Je nachdem, welche Bereiche des Fahrzeugs von Interesse sind, können diese Systemteile detaillierter abgebildet und in die Gesamtsimulation eingebaut werden. Wenn zum Beispiel das Schwingungsverhalten des Antriebsstrangs betrachtet werden soll, kann genau diese Komponente detailreicher modelliert werden. Die restlichen Teilgebiete werden übernommen. Damit muss man sich keine Gedanken über Last oder Antrieb machen, da diese bereits implementiert sind und die gewünschten Ergebnisse liefern. In weiteren Schritten können andere Teilaspekte ausformuliert werden, etwa eine genauere Abbildung des Getriebes oder Motors oder die Erweiterung des Modells auf eine Multidomain-Anwendung über thermische, strömungsmechanische oder elektrische Systeme.

7. Zusammenfassung und Ausblick

Die vorgestellten Studien zeigen, wie mithilfe von CATIA V6 Co-Simulation betrieben werden kann. Ausgehend von dem Ziel einer ganzheitlichen Produktentwicklung wurden Anforderungen festgelegt. Darauf folgend wurden die Möglichkeiten, die Modelica für die Systemsimulation durch die objektorientierte Modellierung bietet, beleuchtet. Vor allem die Fähigkeit, universell einsetzbare Komponenten zu erstellen und diese hierarchisch in ein Simulationsmodell zu integrieren, ist von großer Bedeutung. Dies wird durch die Verwendung von akausalen Beschreibungen physikalischer Vorgänge möglich.

Das mächtige Werkzeug, das Modelica darstellt, wird in eine Cooperative-Simulation integriert, um weitere Arten der Modellbeschreibung zuzulassen. Es wurden zwei Methoden vorgestellt, mithilfe derer eine Co-Simulation aufgesetzt werden kann. Das Functional Mockup Interface beschreibt einen Standard für den Datenaustausch zwischen Programmen durch Functional Mockup Units. Eine so erzeugte Simulation benötigt keinen eigenen Backbone, sondern kommt mit einem Simulator aus, der FMUs importiert. Einen anderen Ansatz verfolgt das Building Controls Virtual Test Bed. Dieses stellt den bei FMI fehlenden Backbone dar und sorgt für die Kommunikation der Clients. Die mit BCVTB kompatiblen Simulatoren eignen sich gut für die in dieser Arbeit betrachtete Anwendung, im Vergleich zu FMI ist die Anbindung allerdings aufwändiger. Das vergleichsweise einfache Verteilen der Co-Simulation auf mehrere Rechner kann bei rechenintensiven Modellen Ressourcen einsparen.

Durch die Art, wie in BCVTB die Zeitschritte abgearbeitet werden, ergeben sich aber auch Probleme, auf die Rücksicht genommen werden muss. Die dadurch entstehende Verzögerung verlangsamt die Systemdynamik, wodurch Änderungen gewisser Kenngrößen verzögert weitergegeben werden. Der zusätzliche Fehler muss bei der Co-Simulation beachtet werden und kann durch eine geringere Zeitschrittweite lediglich reduziert werden. Dieser Umstand resultiert in der Limitierung der Einsatzmöglichkeiten von BCVTB. Beim Einsatz von FMI tritt dieses Problem nicht auf, die Auswahl an Simulatoren ist jedoch zur Zeit sehr gering. Alternativ können auch andere Ansätze des Datenaustauschs in Betracht gezogen werden. Bei der Wahl der Methode, eine Co-Simulation umzusetzen, müssen diese Faktoren berücksichtigt und auf den geplanten Einsatz abgestimmt werden.

In CATIA V6 sind Computer Aided Design und, mit der kürzlich implementierten Dynamic Behavior Modeling Umgebung, auch dynamische Systemsimulation in einem Softwarepaket vereint. Dadurch eignet sich CATIA dazu, die ohnehin durch Modelica

und Co-Simulation umfassenden Möglichkeiten weiter zu vergrößern. Über die FMI-Schnittstelle lassen sich Simulationen umsetzen, die ihre geometrisch-physikalischen Parameter direkt aus den CAD-Modellen beziehen. Es zeigt sich, dass die neue Requirements, Functional, Logical, Physical (RFLP) Umgebung noch in den Kinderschuhen steckt und dadurch manche Features nur schwer nutzbar sind. Die Integration von CATIA in eine Co-Simulation über den Backbone BCVTB ist in der Wahl der Simulatoren wesentlich flexibler. Die Nutzung der CAD-Daten ist aber wegen der fehlenden Kompatibilität mit der RFLP-Umgebung bis dato nicht erreichbar. Es ist zwar möglich, Geometrie- und Masseninformationen der CAD-Daten händisch in die Systemsimulation einzubringen, die einfache Kopplung ist aber nicht durchführbar. Es ist allerdings davon auszugehen, dass in zukünftigen Versionen von CATIA eine derartige Anbindung möglich sein wird.

Die Möglichkeiten der Co-Simulation mit CATIA V6 wurden genutzt, um ein Kraftfahrzeug exemplarisch abzubilden. Die mechanischen Komponenten sind mithilfe von Modelica erstellt worden. Die einzelnen Teilbereiche für ein Gesamtfahrzeug sind mit einem niedrigen Detaillierungsgrad, aber unter Rücksichtnahme auf eine hohe Flexibilität bei der Erweiterbarkeit, umgesetzt. Vor allem auf die Modellierung der Räder wurde Wert gelegt.

Für die Co-Simulation wurden außerdem datenbasierte Modelle für Motorkennlinie und Fahrbahn in MATLAB abgebildet und nach Informationen aus CATIA dementsprechend ausgegeben. Dieses Tool eignet sich wegen des einfachen Umgangs mit großen Datenmengen ideal dafür. Mithilfe von Stateflow können diskrete Ereignisse leicht abgebildet werden. Dadurch kann die Schaltlogik als Statechart umgesetzt werden. In Simulationen verschiedener Szenarien wurde das Zusammenspiel der einzelnen Simulatoren beleuchtet und die Lösungen auf Plausibilität überprüft. Gewisse Teilaspekte wie das Verhalten des Antriebsstrangs oder die Auswirkung der Kommunikationsrate, wurden genauer besprochen.

Das Gesamtmodell des Fahrzeugs stellt ein Rahmenwerk dar, jeder der Teilaspekte ist leicht erweiter- oder austauschbar. Auf diese Art können Untersuchungen zu Phänomenen, die einzelne oder mehrere Komponenten betreffen, leicht durchgeführt werden, indem die jeweiligen Teile detaillierter modelliert werden. Durch den Einsatz der Co-Simulation können Verhalten abgebildet werden, die in einem einzelnen Simulator schwer oder gar nicht darstellbar sind. Vor allem kann man mittels Co-Simulation umfassendere Modelle simulieren und damit zusätzliche Rückkopplung und Wechselwirkung zwischen einzelnen Teilbereichen erfassen, deren Einflüsse in Einzelsimulationen nicht ersichtlich

gewesen wären. Beispielsweise wurde mit der Einbeziehung von Stateflow gezeigt, dass durch Statecharts einfach Entscheidungen abgebildet werden können. Die Tatsache, dass MATLAB in eine Simulation integriert werden kann, spiegelt die hohe Bandbreite an Problemstellungen wider, die dadurch bearbeitet werden können. In weiteren Studien wäre es sinnvoll, den nachteiligen Einfluss der Kommunikationsschrittweite auf die Simulation zu reduzieren. Speziell durch eine Erweiterung des Fahrwegmodells können teilweise Verbesserungen erreicht werden.

Trotz der noch fehlenden Integration der Geometriedaten in die Co-Simulation ist die Wahl von BCVTB sinnvoll. Das vorgestellte Rahmenwerk bietet die Möglichkeit, eine derartige Anbindung nutzen zu können, sobald diese Funktionalität in CATIA umgesetzt ist. Der Aufwand der Adaptierung wird voraussichtlich gering sein, da nur ein kleiner Schritt notwendig ist, um die Verbindung herzustellen. Sobald dieser Schritt gemacht ist, können bestehende CAD-Daten herangezogen werden, um die einzelnen Teile mit physischen Komponenten zu füllen. Mit dem bestehenden Rahmen können Einzelteile direkt nach der Entwicklung den ersten Tests unterzogen werden, wie sie sich im Gesamtsystem verhalten. Dabei handelt es sich nicht nur um einfache Belastungsabschätzungen, sondern auch um das dynamische Zusammenspiel der Komponenten. Durch elastische Elemente können Schwingungseigenschaften untersucht werden, zum Beispiel ob mit den gewählten Parametern aufgrund einer Anregung Resonanz auftritt. Es ist damit nicht notwendig, gewisse Einschränkungen bei der Modellbildung in Kauf zu nehmen, wenn die fraglichen Systembereiche bereits durch vorhandene Module abgebildet sind. Der Teilbereich, der von Interesse ist, wird lediglich in den Gesamtverbund des Modells eingebunden.

In zukünftigen Entwicklungen lässt sich dieses System noch weiter denken. Durch die Zusammenstellungszeichnungen in CATIA sind Informationen über die Interaktion von Einzelteilen vorhanden. Es muss daher möglich sein, direkt aus diesen Daten ein physikalisches Modell automatisch aufzubauen. Auf diese Art können dynamische Systemsimulationen mit geringem Aufwand erstellt werden, wodurch mehr Zeit in die Auswertung von Ergebnissen fließen kann. Damit wird Zeit und Geld gespart und die Entwicklung hochwertiger Fahrzeugkomponenten vorangetrieben. Durch die Möglichkeit, dynamische Simulationen bereits in die frühen Phasen der Produktentwicklung zu integrieren, können Fehler frühzeitig erkannt werden. Die Anzahl an Prototypen, die gebaut werden müssen, wird dadurch reduziert. Es ist sogar denkbar, noch einen Schritt weiter zu gehen. Statt nur Informationen aus den CAD-Daten in eine Simulation zu integrieren, lässt sich auch die Geometrie selbst dynamisch ändern. Damit kann Co-Simulation noch

früher in die Entwicklung von Komponenten einbezogen werden, um eine Optimierung der Geometrie zu erreichen. Nach jedem Simulationsdurchlauf können die Parameter angepasst und mit den neuen Daten die Simulationen erneut durchgeführt werden. Auf diese Art werden Faktoren und Wechselwirkung zwischen Komponenten für die optimale Auslegung einbezogen, die mit klassischen Methoden nicht abbildbar sind.

Auf dem Weg zu einem integrierten umfassenden Product Lifecycle Management werden sämtliche Informationen aus unterschiedlichen Bereichen über die Lebenszeit des Produkts zusammengeführt. Durch den Aufbau einer zentralen Wissensbasis ist es möglich, Informationen von CAD und CAE während der Produktentwicklung gemeinsam zu nutzen. Dadurch könnten in Zukunft zusätzlich zu geometrischen Informationen auch Daten zu thermischen, ökonomischen oder ökologischen Eigenschaften integriert werden.

Die aktuellen Möglichkeiten mögen im Moment noch begrenzt einsetzbar sein. Durch die vorgestellten Ideen kann aber jetzt schon abgeschätzt werden, in welche interessanten Richtungen sich die integrierte Produktwicklung bewegen kann. Die Bausteine sind bereits vorhanden, es liegt nur daran, sie richtig zusammenzusetzen.

Literatur

[1] Peter Fritzson: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, John Wiley & Sons Inc, 2004, print.

[2] Hilding Elmqvist: Structured Model Language for Large Continuous Systems. Dissertation, Lund Institute of Technology (now called: Lund University), 1978

[3] https://www.modelica.org/association, besucht am 26.11.2013

[4] François E. Cellier, Ernesto Kofman: „Differential Algebraic Equation Solvers.“, Continuous System Simulation, Springer Science & Business Media, Inc, 2006, 319-396, print.

[5] Martin Otter et al.: The New Modelica MultiBody Library, DLR; Dynasim, pp. 311-330, Proceedings of the 3rd International Modelica Conference, Linköping, November 3-4, 2003

[6] Martin Otter et al.: Package PowerTrain: A Modelica library for modeling and simulation of vehicle power trains, Modelica Workshop 2000, Lund, Sweden, pp. 23-32.

[7] http://www.3ds.com, besucht am 26.11.2013

[8] T. Blochwitz et al.: The Functional Mockup Interface for Tool independent Exchange of Simulation Models, Proceedings of the 8th International Modelica Conference, Pages 105-114. Linköping University Press. 8th International Modelica Conference, 20.-22.03. 2011, Dresden. ISBN 978-91-7393-096-3. ISSN 1650-3740

[9] http://www.simulationx.com, besucht am 26.11.2013

[10] Modelica Association: Functional Mock-up Interface for Model Exchange and Co-Simulation, 2.0 Beta 4, 2012, www.fmi-standard.org

[11] MathWorks: Developing S-Functions, 2013, www.mathworks.com

[12] M. Wetter and P. Haves: A Modular Building Controls Virtual Test Bed for the Integration of Heterogeneous Systems, Third National Conference of IBPSA-USA Berkeley, California July 30 – August 1, 2008

[13] Johan Eker et al.: Taming Heterogeneity—The Ptolemy Approach, Proceedings of the IEEE, VOL. 91, NO. 1, January 2003 127

[14] M. Wetter: Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed , 2011, Journal of Building Performance Simulation, 4(3):185-203, 2011.

[15] M. Wetter et al.: Modelica Buildings Library, 2012, Journal of Building Performance Simulation, (pre-print) 2013

[16] Giancarlo Genta: Motor Vehicle Dynamics, World Scientific Publishing Co. Pte. Ltd Singapore, 1997, print

[17] Bruce R. Munson, Donald F. Young, Theodore H. Okiishi: Fundamentals of Fluid Mechanics, Third Edition, John Wiley & Sons, Inc., 1998, p 591

[18] Ryszard Andrzejewski, Jan Awrejcewicz: Nonlinear Dynamics of a Wheeled Vehicle, Springer Science+Business Media, inc, 2005, p 201

[19] Horst Czichos, Manfred Hennecke: HÜTTE - das Ingenieurwissen, 33. Auflage, Springer Berlin Heidelberg New York, 2008, p E53

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

A. Kontrollsystem in MATLAB

```
%% Defining the Arguments for the Simulation
beginTime = 0;
endTime = 2400;
timeStep = 60;
xml_file = 'testmodelmatlab.xml';
additional_args='';

%% Parameter Gain for Paramtervariation
gain = 20;


%% Building the command line argument
sp = ' ';
working_dir = 'C:\Users\user\BCVTB\test\';
bcvtb_dir = '"C:\Program Files (x86)\bcvtb\bin\BCVTB.jar"';
command = 'java -jar';
command_arg = '-run';
var1 = horzcat('-beginTime', sp, num2str(beginTime));
var2 = horzcat('-endTime', sp, num2str(endTime));
var3 = horzcat('-timeStep', sp, num2str(timeStep));
command_line = horzcat(command, sp, bcvtb_dir, sp, command_arg, sp,
    working_dir, xml_file, sp, var1, sp, var2, sp, var3, sp,
    additional_args)
%% Writing the arguments to a .csv file
csvwrite (strcat(working_dir, 'beginTime.csv'),beginTime);
csvwrite (strcat(working_dir, 'endTime.csv'),endTime);
csvwrite (strcat(working_dir, 'timeStep.csv'),timeStep);
csvwrite (strcat(working_dir, 'gain.csv'),gain);
%% execute
system (command_line);
```

Listing 10: Implementierung eines Kontrollsystems in MATLAB zur Steuerung einer Co-Simulation mit BCVTB.

B. Batch-File für den MATLAB-Aufruf

```
1 pscp socket.cfg user@128.0.0.1:/Users/user/Documents/Software/matlab
2 pscp ../beginTime.csv user@128.0.0.1:/Users/user/Documents/Software/matlab
3 pscp ../timeStep.csv user@128.0.0.1:/Users/user/Documents/Software/matlab
4 pscp Model.m user@128.0.0.1:/Users/user/Documents/Software/matlab
5 plink -ssh user@128.0.0.1 "cd /Users/user/Documents/Software/matlab ; /
      Applications/MATLAB_R2011b.app/bin/matlab -nojvm -nosplash -logfile
      matlab.log -r Model"
6 pscp user@128.0.0.1:/Users/user/Software/matlab/matlab.log %CD%
```

Listing 11: Sourcecode des Batch-Files mit dem MATLAB auf einem MacOS-Betriebssystem aufgerufen wird.

C. Beispielcode zur Remote-Simulation mit MATLAB

```
1 %% Program for Co-Simulation
2 %Declarations
3 setenv('BCVTB_OS', '')
4 setenv('BCVTB_HOME', '/Applications/bcvtb')
5 addpath( strcat(getenv('BCVTB_HOME'), '/lib/matlab'));
6 loadlibrary('/Applications/bcvtb/lib/util/libbcvtb',@bcvtb)
7 beginTime = csvread('beginTime.csv');
8 timeStep = csvread('timeStep.csv');
9 retVal    = 0;
10 flaWri    = 0;
11 flaRea    = 0;
12 simTimWri = beginTime;
13 simTimRea = 0;
14 delTim = timeStep;
15 sockfd = establishClientSocket('socket.cfg');
16 simulate = true;
17 while (simulate)
18 ...    %The simulation takes place
19 end
20 exit
```

Listing 12: Beispielcode für eine entfernt ausgeführte Co-Simulation mit MATLAB auf dem MacOS-Betriebssystem.