FH Vorarlberg
Vorarlberg University of Applied Sciences

Modeling of a Motorcycle in Dymola/Modelica

Master's in Mechatronics

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

Dornbirn, 2009

Supervisors
Dipl.-Ing. Dr. Franz Geiger - FH Vorarlberg
Prof. Dr. François E. Cellier - ETH Zürich
Dirk Zimmer MSc - ETH Zürich

Submitted by
Thomas Schmitt BSc

Abstract

This thesis introduces a new and freely available *Modelica* library for the purpose of simulation, analysis and control of bicycles and motorcycles (single-track vehicles). The library is called *MotorcycleLib* and focuses on the modeling of virtual riders based on automatic controller design. For the vehicles, several models of different complexity have been developed. To validate these models and their driving performance, virtual riders are provided. The main task of a virtual rider is to track either a roll angle profile or a pre-defined trajectory using path preview information. Both methods are implemented and several test tracks are also included in the library. Regarding the stability of uncontrolled vehicles, an eigenvalue analysis is performed in order to obtain the self-stabilizing area. A key task for a virtual rider is to stabilize the motorcycle. To this end, the controller has to generate a suitable steering torque based on the feedback of appropriate state variables (e.g. lean angle and lean rate). One major problem in controlling two-wheeled (single-track) vehicles is that the coefficients of the controller are strongly velocity dependent. This makes the manual configuration of a controller laborious and error-prone. To overcome this problem, an automatic calculation of the controller's coefficients is desired. This calculation requires a preceding eigenvalue analysis of the corresponding uncontrolled vehicle. This enables a convenient controller design, and hence several control laws that ensure a stable driving behavior are provided. The corresponding output represents a state feedback matrix that can be directly applied to ready-made controllers which are the core of virtual riders. The functionality of this method is illustrated by several examples in the library.

Kurzfassung

In dieser Arbeit wird eine neue, frei verfügbare *Modelica* Bibliothek für die Simulation, Analyse und Regelung von Fahrrädern und Motorrädern (einspurige Fahrzeuge) vorgestellt. Die Bibliothek nennt sich *MotorcycleLib*. Das Hauptaugenmerk liegt dabei auf der Modellierung von virtuellen Fahrern welche auf einer automatischen Reglerauslegung basieren. Für die einspurigen Fahrzeuge wurden sämtliche Modelle unterschiedlicher Komplexität entwickelt. Um die Modelle und damit das Fahrverhalten validieren zu können, werden virtuelle Fahrer zur Verfügung gestellt. Deren Hauptaufgabe besteht nun darin, entweder ein Neigungswinkelprofil oder eine vordefinierte Strecke zu verfolgen. Um letzeres zu realisieren, schaut der Fahrer, wie es auch in der Realität der Fall ist, eine gewisse Distanz voraus. Beide Methoden wurden implementiert und können auf verschiedenen Teststrecken getestet werden. Um die Stabilität der Fahrzeuge überprüfen zu können, wird eine Eigenwertanalyse durchgeführt, mit Hilfe derer festgestellt werden kann, in welchem Bereich das Fahrzeug eine selbststabilisierende Wirkung erzielt. Eine weitere wichtige Aufgabe des virtuellen Fahrers besteht darin, das Fahrzeug zu stabilisieren, das heißt, es aufrecht zu halten. Dazu muss ein Regler, welcher den Kern des virtuellen Fahrers darstellt, aufgrund der Rückführung geeigneter Zustandsvariablen (z.B. Neigungswinkel und Neigungsrate) ein passendes Drehmoment auf den Lenker des Fahrzeugs ausüben. Eine große Herausforderung, welche es zu überwinden gilt, stellen dabei die geschwindigkeitsabhängigen Reglerkoeffizienten dar. Diese Problematik ist sehr zeitaufwändig und erschwert die manuelle Einstellung der Regler. Um solche Probleme zu überwältigen, wird eine automatische Auslegung der Reglerkoeffizienten vorgestellt, welche auf einer zuvor durchgeführten Eigenwertanalyse basiert. Dadurch können die Regler auf eine sehr komfortable Art und Weise erstellt werden. Um verschiedene Reglerauslegung durchführen zu können, wurden spezielle Verfahren entwickelt. Das Ergebnis dieser Reglerauslegung ist eine Rückführungsmatrix, welche direkt auf vorgefertigte virtuelle Fahrer angewendet werden kann. Die Funktionalität dieser Methode wird anhand zahlreicher Beispiele aus der Bibliothek demonstriert.

Acknowledgment

My enthusiasm for mathematical modeling of physical systems began in the 2^{nd} semester of the bachelor's degree program in 2005. Our mathematics teacher, Peter Pichler, taught us the theory of differential equations. He is an excellent teacher with a particular proficiency for imparting knowledge. At the same time, while attending the physics lecture, Stefan Mohr, our physics lecturer, described a spring-damper system by means of differential equations. Just like Peter Pichler, he is able to impart knowledge in a manner that really motivated me. Many thanks to both of you! I guess that is one of the reasons why I am currently sitting here writing this thesis.

Let me continue. At the end of the 4^{st} semester our signal theory lecturer, Reinhard Schneider, wanted us to derive the equations of a crane-crab system and afterwards transform it into a state-space representation. A new milestone had been established. In a single blow, it was possible to transfer the mathematical model into a computer program and thus simulate it. Thank you Reinhard Schneider especially for this most useful task!

In the 5^{th} semester, Hannes Wachter, Markus Andres and I successfully completed a semester abroad in Linköping/Sweden. There we completed a modeling and simulation course and a project based on this lecture. Our lecturer was Måns Östring; a young and motivated guy. He taught us to model physical systems from a bond graphical point of view. Mycket brå! On top of that, I noticed that mathematical modeling of physical systems is the topic I am interested in. Thank you Måns for this lecture! Thank you Hannes and Markus for the nice time in Sweden!

During the master's degree program, we had two seminars that served as preparation for the Master's Thesis. I used both seminars to deepen my knowledge regarding modeling and simulation. At this time, I found the book "Continuous System Modeling" by François Cellier. I asked Reinhard Schneider without any expectations whether we could visit him. At this time, I thought that professor Cellier was still in Tucson/Arizona. A few days later, Reinhard Schneider sent me the link of his homepage. To my surprise, I discovered that he is a lecturer at the ETH-Zürich. While exploring his homepage, I found a section called "Proposals for MS Thesis Topics". Another couple of days later Markus Andres, and I tried to phone in. Okay, it was summer: Maybe he was on vacation. Thus, we wrote him an email! Let me shorten the whole story.

Professor Cellier was quite encouraged concerning our request. Johannes Steinschaden, the head of our degree program, the “girls” from the international office and Professor Cellier made it possible to carry out this Master’s Thesis. Thanks a lot for all of your efforts! As a preparation Johannes Steinschaden made it possible to attend Professor Cellier’s course “mathematical modeling of physical systems” at the ETH-Zürich. Thus, Markus Andres, Mathias Schneller and I drove to Zürich every week in order to learn as much as possible. Thanks to all of you. Special thanks to Markus and Mathias for the great time!

Special thanks to my supervisor, Franz Geiger, at the “Vorarlberg University of Applied Sciences” for your lectures on modern control engineering, the most useful robotics lab sessions and your great support outside of the lectures. Again, many thanks to my supervisor, François Cellier, at the ETH-Zürich for the interesting lectures and the support during the thesis. Many thanks to Dirk Zimmer for your visits and your great support! Although you are currently writing your Ph.D. thesis, you always had time for us!

Many thanks to my family! You always believed in me! Many thanks to Markus Andres’ family who treated me like a family member during the whole thesis! Thanks to little U. You made me feel happy all the time!

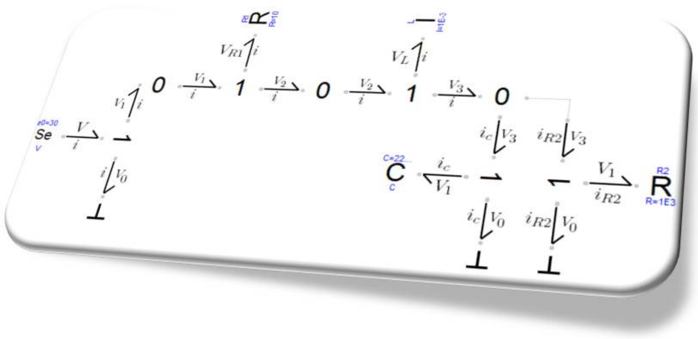
Most of all I want to thank my girlfriend Yvonne. Especially in the last couple of months I did not have any time for you but you are still in love with me.

Contents

List of Figures

List of Tables

1 Introduction



1.1 Motivation

In recent years, more and more detailed and partly animated models of vehicles are being used in practice. These models serve on the one hand to support the design of new vehicle types. Ever shorter development periods of new vehicle types (models) force the designers to replace tests that used to be performed in the past on prototypes of the new vehicles by simulation runs to be executed before a prototype is ever build. On the other hand, it has become customary to employ mathematical simulations of vehicles also for training purposes. For example, driving a vehicle on an icy road is being trained in Sweden on a computer by means of a detailed and realistic vehicle model.

Among the vehicle models, models of bicycles and motorcycles turn out to be particularly delicate. Whereas a four-wheeled vehicle remains stable on its own, the same does not hold true for a single-track (two-wheeled) vehicle. For this reason, the stabilization of such a vehicle, a control issue, requires special attention. The simulation has to account for the inclination of the vehicle in a curve as well as for the shift of the center of gravity of the driver. Such models are currently not yet being offered in the public libraries of Dymola/Modelica.

1.2 Structure of the Thesis

The purpose of the last section of this chapter is to give a brief outline to the mathematical modeling of physical systems. Chapter 2 is intended to demonstrate the basics of the physical behavior of single-track vehicles. This chapter focuses on the physical effects that cause an uncontrolled version of a bicycle or motorcycle to stay upright. In Chapter 3, the bicycle and motorcycle models used in this library are explained. In the beginning, three basic models are introduced, i.e. they are made of rigid bodies, with ideal tires and so on. At the end, two advanced models are explained, i.e. suspensions, front twist frame flexibility and non-ideal tire characteristics like slip, and aerodynamics are taken into account.

In Chapter 4, the stability of an uncontrolled version of the basic bicycle and motorcycle models is discussed. So far, the user of this library is able to perform such an analysis in order to find the self-stabilizing region of the vehicle.

Chapter 5 provides the controller models needed in order to develop a virtual rider. In this chapter, several approaches to design suitable controllers are introduced, e.g. classic design, state-space design, linear quadratic regulator (LQR) design. In Chapter 6, the former developed controllers are incorporated into virtual riders. Basically, two different kinds of virtual riders are introduced. Namely riders that are capable of either tracking a roll angle profile (open-loop method) or a pre-defined trajectory (closed-loop method). In order to validate the performance of a virtual rider several environments are provided. That is, for the open-loop method, test tracks are introduced (e.g. 90°-curve, moose test, s-chicane). To cover the closed-loop method, a randomly generated path is included in the library.

The purpose of Chapter 7 is to get familiar with the *MotorcycleLib*. Thus, several different examples are provided in the library. In the documentation itself it is explained how some of them were established. This information is provided to the user in the form of a user's guide.

1.3 Basic Structure of the Library

The top layer of the *MotorcycleLib* is depicted in Figure 1.1. For each single-track vehicle a separate sub-package is provided. The basic bicycle sub-package is composed of a rigid rider and a movable rider sub-package. Both include the corresponding wrapped model and a function in order to perform an eigenvalue analysis. The basic motorcycle sub-package also includes a wrapped model and an eigenvalue analysis function. The structure of the advanced motorcycle sub-package is much more detailed since each part (e.g. front frame) is created in a fully object-oriented fashion. It is composed of a *parts* and an *aerodynamics* sub-package. The parts sub-package includes several different front frames and swinging arms, a rear frame, the rider's upper body, a torque

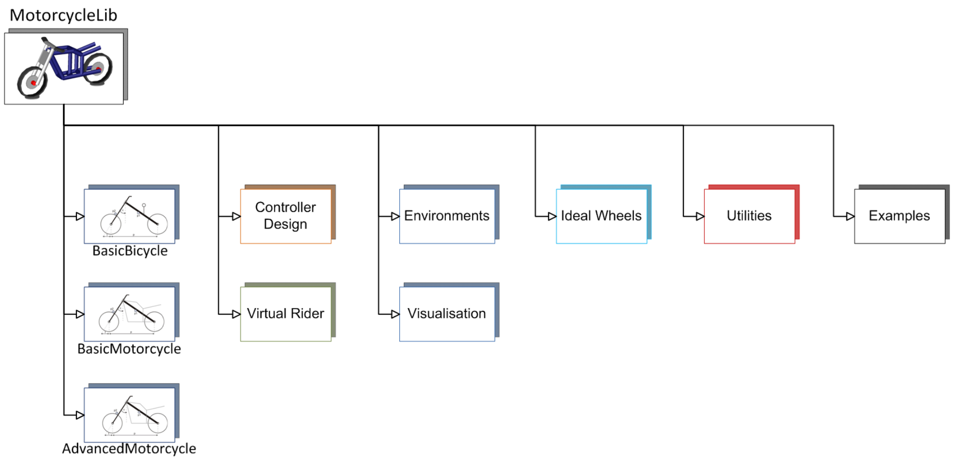


Figure 1.1: Library structure - top layer

source (engine), the elasto-gap and a utilities sub-package. Here models of characteristic spring and damper elements are stored. The aerodynamics sub-package includes a lift force, a drag force and a pitching moment model.

The controller design sub-package contains pole placement functions in order to design appropriate controllers. The virtual rider sub-package includes, among others, a virtual rigid rider and a virtual movable rider sub-package. In both the riders are capable of either tracking a roll angle profile or a pre-defined trajectory. To this end, several different controllers (e.g. classic, state-space and LQR) are incorporated into the virtual riders.

The environments sub-package provides tracks for both roll angle tracking and path tracking. In addition, the models for single-point path tracking are included. The visualization sub-package provides the graphical information for the environments sub-package. In the ideal wheels sub-package the visualization of the rolling objects from D. Zimmer's *MultiBondLib* [ZC06] were modified such that the appearance is similar to real motorcycle wheels. The utilities sub-package provides some additional functions and models which are partly used in the library. The purpose of the examples sub-package is to provide several different examples that demonstrate how to use the library.

1.4 Utilized Libraries

The bicycle and motorcycle models of this library are based on the *MultiBondLib* [ZC06] and F. Cellier's *BondLib* [CN05]. The wheels and tires of the vehicles are either provided by the *MultiBondLib* or M. Andres' *WheelsAndTires* [And] library. The former ones are ideal, so-called "knife-edge" wheels whereas non-ideal effects (e.g. slip) are taken into account in the latter ones. Thus, depending on the particular model, both libraries are utilized. The *Modelica Standard Library* [Ass09] provides the basic elements, e.g. for the controller models. For the controller design itself, among others, the *Modelica Linear Systems* library is used. To be able to perform an eigenvalue analysis, the *Linear Systems* library is taken into account.

1.5 Competitors

In 2006, F. Donida et al. introduced the first Motorcycle Dynamics Library in Modelica [DFS$^+$06] and [DFST08]. In contrast to the *MotorcycleLib*, it is based on the Modelica *MultiBody* library [OEM03]. The library focuses on the tire/road interaction. Moreover, different virtual riders (rigidly attached to the main frame or with an additional degree of freedom allowing the rider to lean sideways) capable of tracking a roll angle and a target speed profile are presented. Until now these virtual riders include fixed structure controllers only [DFST08]. This means that the virtual rider stabilizes the vehicle only correctly within a small velocity range. Using the *MotorcycleLib* this major deficiency can be overcome. Furthermore, to validate the motorcycle's performance, the virtual rider is capable of either tracking a roll angle profile (open-loop method) or a pre-defined path (closed-loop method).

1.6 Mathematical Modeling of Physical Systems

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

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

1.6.1 Differential and Algebraic Equations

In general, a physical system can be described by DAE's. The most basic approach is to find the system of DAE's manually by setting up the equations of the elements that the system is composed of and combining them in an appropriate manner. As soon as a proper description of the system is found, the DAE's have to be transformed into a state-space representation to be solved. The state-space representation is a description of the system by means of ordinary differential equations (ODE's) i.e. a system of 2^{nd} order can be described with 2 ordinary differential equations (see Chapter 5.2.1).

1.6.2 Bond Graphs

Another approach to model physical systems are bond graphs. They are based on energy flow through systems with the basic laws of energy conservation applied. These laws state that energy can only be affected by three mechanisms: It can be *stored*, *transported* or *converted*. Energy flow is the derivation of energy by time which is also referred to as power. In every physical system, power is the product of an effort and a flow variable, e.g. for electrical systems the effort is the voltage, the flow is the current.

The energy flow is represented in a graphical manner by directed harpoons as shown in Figure 1.2 and is mathematically described by the relation $P = e \cdot f$. A bond is thus the

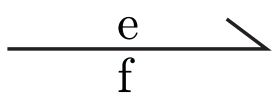


Figure 1.2: A directed harpoon (bond) to model energy flow.

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

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

Multi Bond Graphs

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

1.6.3 Object-Oriented Modeling

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

Modelica

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

Dymola

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

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

2 Bicycle and Motorcycle Dynamics

This chapter is intended to show the basics of the physical behavior of bicycles and motorcycles (single-track) vehicles. The focus of this chapter lies in the physical effects that cause an uncontrolled version of a single-track vehicle to stay upright. Unlike four-wheeled vehicles, single-track vehicles are unstable when stationary. Under certain conditions, a self-stabilizing area appears when the vehicle moves in a forward direction. This area mainly depends on the geometry and mass distribution of the vehicle but as well on gyroscopic effects.

In the following, some important terms and definitions are introduced. Afterwards, the gyroscopic effects are described. Finally, the importance of trail is discussed.

2.1 Terms and Definitions

Bicycles and motorcycles are described with reference to the terms and definitions depicted in Figure 2.1.

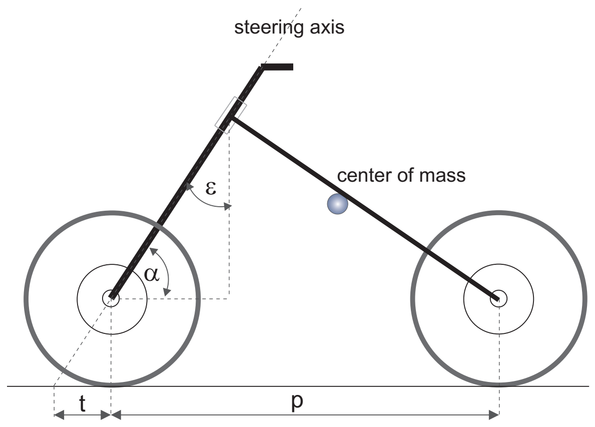


Figure 2.1: Terms and definitions of bicycles and motorcycles

The wheelbase p is the distance between front wheel and rear wheel contact point. The trail t is the distance between the front wheel contact point and the point of intersection of the steering axis with the ground line (horizontal axis). The head angle α, also called caster angle, is the inclination between the steering axis and ground line. It is also common to define the head angle $\epsilon = \frac{\pi}{2} - \alpha$ as the inclination between the steering axis and vertical axis. However, this depends on the convention.

2.2 Gyroscopic Effects

From a physical point of view, a wheel is nothing other than a gyroscope. In the following, the so-called gyroscopic effects are demonstrated with reference to Figure 2.2.

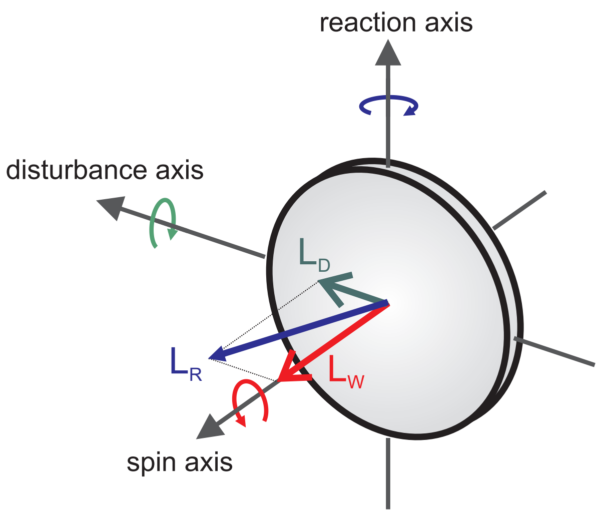


Figure 2.2: Demonstration of gyroscopic effects

Let us say that the wheel performs a counter-clockwise rotation about the spin axis. This results in an angular momentum L_W in direction of the spin-axis. The angular momentum of a rigid body is defined as

$$L = I \cdot \omega \tag{2.1}$$

Where I is the moment of inertia and ω is the angular velocity.

Let us now apply a counter-clockwise torque (e.g. caused by side wind) about the disturbance axis. This generates an additional angular momentum L_D. As a consequence of L_D, the wheel tries to elude and thus performs a clockwise rotation about the reaction axis. This effect is referred to as *precession* of a gyroscope. The rotation about the reaction axis stops as soon as the spin axis is coincident with the resulting angular momentum L_R. For a better understanding, Figure 2.3 depicts each step of this sequence.

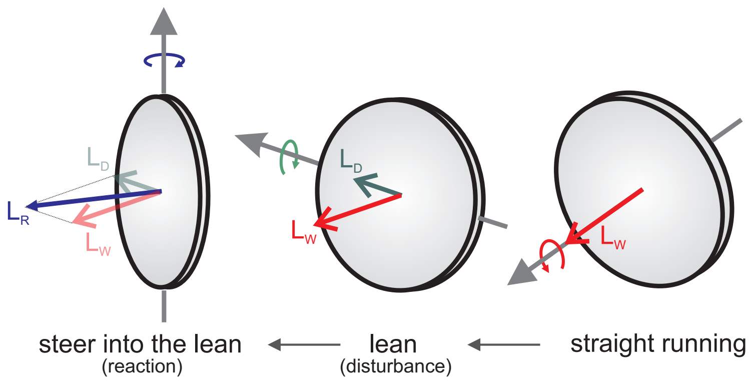


Figure 2.3: Stepwise demonstration of gyroscopic effects

This effect is utilized by human cyclists. To enter a *left turn*, it is not mandatory to apply a steering torque. As described in Figure 2.2, it is absolutely sufficient to lean to the left. Due to gyroscopic effects the front wheel of the bicycle steers into the lean. Thus it is possible to enter a turn without touching the handlebar of the bicycle (free-hand). Figure 2.4 illustrates that this effect also works vice-versa.

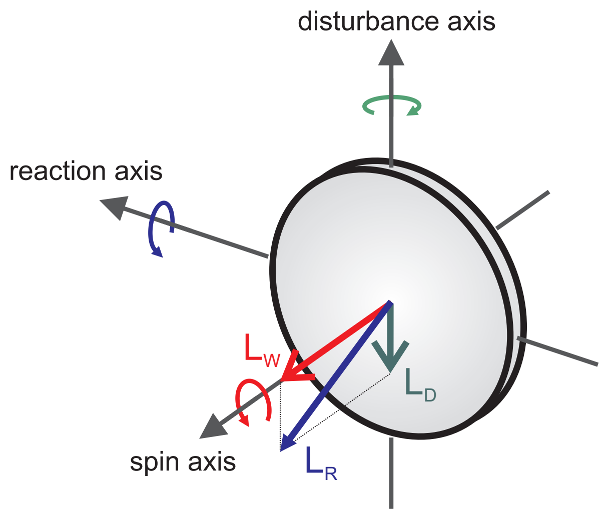


Figure 2.4: Demonstration of gyroscopic effects

In order to enter a left turn, one has to steer to the right initially. Again, due to gyroscopic forces, this causes the vehicle to lean to the left. This effect is usually referred to as countersteering and is utilized by both motorcycle and bicycle riders.

Straight Running

Due to geometry and gyroscopic forces, Klein and Sommerfeld [KS10] found out that a single-track vehicle is self-stabilizing within a certain velocity range. In this range, an interaction of the effects mentioned in Figure 2.2 and 2.4 takes place. Hence, the vehicle performs a tail motion in the longitudinal direction. Below this area, the steering deflections caused by gyroscopic forces are too small in order to generate enough centrifugal force. Thus the amplitude of the tail motion increases and the vehicle falls over. Although, these interactions are damped by the trail (refer to Section 2.3) it is still impossible to achieve stable behavior. Hence the rider has to apply a steering torque to ensure the vehicle stays upright. Above this area, for high speeds, the gyroscopic forces are almost unnoticeable for the rider. That is, the amplitude of the tail motion is close to zero. More precisely, although the vehicle feels stable, after a certain time, it falls over like a capsizing ship. However, by applying a steering torque it is rather simple to stabilize the vehicle. In most cases, it is sufficient that one solely touches the handle bars in order to stabilize the vehicle. To this end, in Chapter 4 the straight running capabilities are considered from a more theoretical point of view.

In reality, due to friction, the velocity of the vehicle decreases and hence, after a certain time, the vehicle becomes unstable. For a detailed description regarding gyroscopic effects, refer to [KS10].

Turning

During a turn, a single-track vehicle has a specific lean angle depending on the velocity and the radius of the turn. In the following, let us derive the relations between velocity, turn radius and lean angle. The forces acting on a single-track vehicle are shown in Figure 2.5.

In order to prevent the bicycle from capsizing, the moments caused by gravitational force F_g and centrifugal force F_c have to cancel each other out.

$$F_g \cdot x = F_c \cdot y \tag{2.2}$$

With $F_g = m \cdot g$ and $F_c = \frac{m \cdot v^2}{R}$, where R is the radius of the turn, m is the mass of the vehicle including the rider and g is the gravity.

The lean angle is given by

$$tan(\phi) = \frac{F_c}{F_g} = \frac{F_{lat}}{N} \tag{2.3}$$

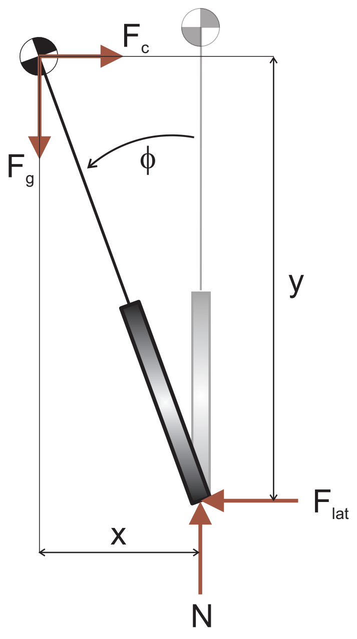


Figure 2.5: Forces that act on a single-track vehicle during a turn

where N is the normal force ($N = F_c$) and F_{lat} is the so-called cornering or lateral force. This force ensures that the wheel does not slide away. In order to ensure stable behavior, the following equilibrium condition must hold:

$$F_{lat} = F_c \tag{2.4}$$

Thus for a given velocity and turn radius, the corresponding lean angle results in:

$$\phi = atan\left(\frac{v^2}{R \cdot g}\right) \tag{2.5}$$

For an ideal bicycle (infinitesimally thin wheels, no slip, no friction, etc.) the lean angle is a function of velocity and turn radius. No matter what size or weight the vehicle is. In reality, the cornering force is limited by friction. The maximum cornering force therefore results in

$$F_{lat,max} = m \cdot g \cdot \mu \tag{2.6}$$

By inserting Equation 2.6 into Equation 2.5, it follows

$$tan(\phi_{max}) = \mu \tag{2.7}$$

Hence, in reality the maximum lean angle only depends on the friction μ.

2.3 The Importance of Trail

As already mentioned, the trail is significantly involved in the process of stabilizing the vehicle. The stabilizing effect caused by the trail is explained with reference to Figure 2.6.

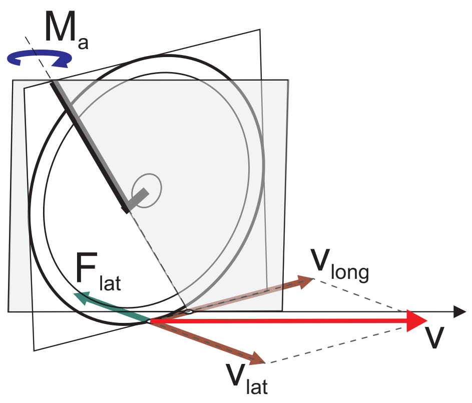


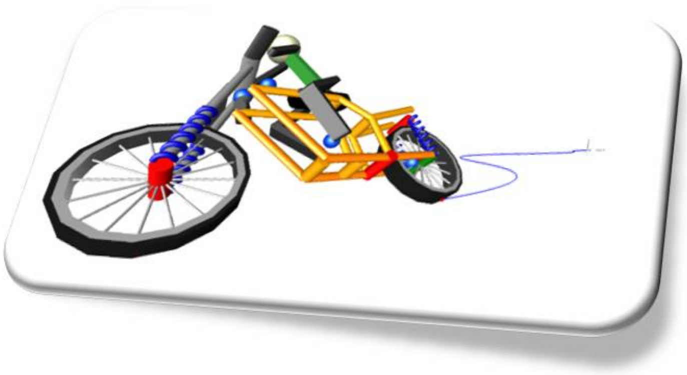
Figure 2.6: Aligning moment caused by the trail

Let us say that the vehicle drives with a constant velocity v. Furthermore, let us say that due to a disturbance (e.g. side wind) the vehicle leans to the left. Thus, the front wheel is turned to the left as well (as depicted in Figure 2.6). Now a cornering force F_{lat} appears. This force in combination with the trail generates an aligning moment M_a that turns the wheel back. That is the trail has the function of a lever arm. The more trail, the longer the lever arm and the more aligning moment is generated. For a detailed description, refer to [Cos06].

Thus, mountain bikes have less trail in order to be more agile. Consequently, the stability decreases. Vehicles with more trail are less agile but more stable. For this reason, while riding a vehicle with more trail one has to apply higher steering torques in order to turn.

Finally, the center of mass and the wheelbase are taken into account. Basically, a high center of mass makes it easier to balance the vehicle than a short one. The wheelbase p is a crucial factor for the time needed in order to reach the desired lean. The higher p, the larger the lateral distance is that the center of mass has to cover and the more time it takes to reach the desired lean angle.

3 Bicycle and Motorcycle Modeling



This chapter is intended to introduce the bicycle- and motorcycle models that are included in the *MotorcycleLib*. Firstly, a basic 3 degree of freedom (d.o.f.) bicycle model is described. The intended bicycle model was developed by Schwab et al. [SMP05]. This model consists of four rigid bodies connected via revolute joints (hinges). The wheels are considered to be ideal. Secondly, this model is extended by an additional d.o.f. allowing the rider to lean sideways. This extension is also based on a bicycle model recently introduced by Schwab et al. [SKM08].

Thirdly, a 4 d.o.f. motorcycle model which was introduced by V. Cossalter [Cos06] is described. Basically, V. Cossalter's model is the same as the one introduced by R. S. Sharp in 1971 [Sha71]. This model allows a lateral displacement of the rear frame since the wheels are no longer ideal. Due to the fact that the wheels of the *MultiBondLib* [ZC06] are ideal, the model is reduced to 3 d.o.f.. Later, with reference to the *WheelsAndTires* library [And], it is possible to consider non-ideal effects of wheels and tires and thus simulate the lateral displacement of the wheels caused by tire slip.

Finally, two more complex models are described. The first model was originally developed by C. Koenen during his Ph.D. Thesis [Koe83]. R. S. Sharp and D. J. N. Limebeer introduced the SL2001 model which is based on Koenen's model [SL01]. They reproduced Koenen's model as accurately as possible and described it by means of multi bodies. The model developed in this library is based on the SL2001 motorcycle. The second model is based on an improved more state-of-the-art version of the former one developed by R. Sharp, S. Evangelou and D. J. N. Limebeer [SEL04]. Another detailed description of these models can be found in S. Evangelou's Ph.D. Thesis [Eva03]. As with the former 4 d.o.f. model, these models only include all degrees of freedom in combination with the *WheelsAndTires* library. Again, without this library, several freedoms are inhibited.

A very detailed historical background regarding motorcycle models can be found in S. Evangelou's Ph.D. Thesis [Eva03] or in an article published by D. J. N. Limebeer and R. Sharp [LS06]. A very detailed history of bicycle steer and dynamic studies can be found in [MPRS07].

3.1 Basic Bicycle Model - Rigid Rider

3.1.1 Definition of Bicycles

Although bicycles are composed of several different parts, in the simplest case the mechanical model consists of four rigid bodies. A rear frame including the rigidly attached rider, a front frame including the front fork and handle bar assembly and two ideal knife-edge wheels touching the ground at a single contact point. The front and rear wheels are attached to the front and rear frame revolute joints. Figure 3.1 depicts the model of a basic bicycle.

The wheels assumed to be infinitesimally thin (knife-edge) without slippage in the longitudinal and lateral direction. The two frames are connected via an inclined revolute joint (steering axis). Each wheel is connected to the frame using a revolute joint. Since a rigid body in space has 6 d.o.f., one would think that the basic bicycle in total has 24 d.o.f. in total. However, on the one side, each revolute joint inhibits 5 d.o.f., namely all translational d.o.f. and two rotational d.o.f., and on the other side, each contact point between wheel and ground inhibits 3 d.o.f., namely one translational d.o.f. in the normal direction (in direction of the z-axis) and two rotational freedoms (about the x- and z-axis). Hence the total number of d.o.f. is $24 - 3 \cdot 5 - 2 \cdot 3 = 3$.

The remaining 3 d.o.f. are:

- the roll angle ϕ of the rear frame
- the steering angle δ, which is the rotation between the front frame with respect to the rear frame about the inclined steering axis
- θ_r, which is the rotation of the rear wheel with respect to the rear frame

The origin (0) of the global co-ordinate system is at the contact point of the rear wheel. The orientation of the rear frame with respect to the global co-ordinate system is clearly defined by means of three angular rotations (see Figure 3.1):

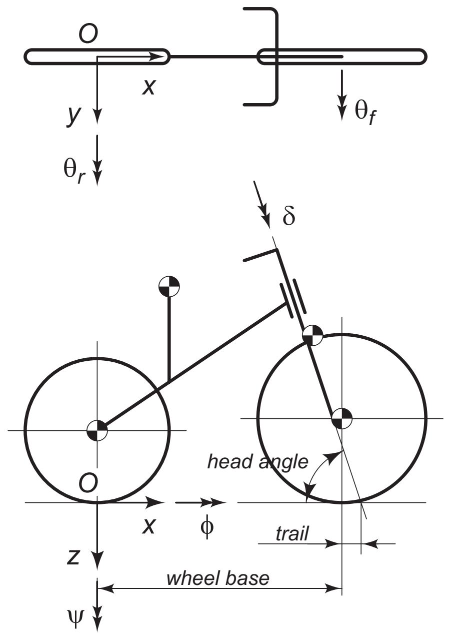


Figure 3.1: Model of a simple 3 d.o.f. bicycle (Source: [SMP05], p.30).

- a yaw rotation ψ about the global z-axis
- a roll rotation ϕ about the rotated x-axis
- a pitch rotation θ about the rotated y-axis

3.1.2 Geometry of Bicycles

The geometry of bicycles is described with the following parameters: The wheel base p, the trail t, the head angle α and the radii of the wheels (refer to Chapter 2.1).

3.1.3 Model of the 3 d.o.f. Bicycle

The model of the basic bicycle used in this documentation was presented by Schwab et al. in 2005 [SMP05]. This model is based on Whipple's model [Whi99]. To develop a model within the Dymola environment, a different co-ordinate system is introduced (see Figure 3.2). The parameters of Schwab's benchmark bicycle are listed in Table 3.1.

Parameter	**Symbol**	**Value**
wheel base	p	1.02m
trail	t	0.08m
head angle	α	atan(3)
forward speed	v	*variable* [m/s]
Rear Wheel		
radius	r_{rw}	0.3m
mass	m_{rw}	2kg
inertia tensor	$(I_{rw\_xx}, I_{rw\_yy}, I_{rw\_zz})$	(0.06, 0.06, 0.12) kgm^2
Rear Frame		
center of mass	(x_{rf}, y_{rf}, z_{rf})	(0.3, 0.9, 0) m
mass	m_{rf}	85kg
inertia tensor	$\begin{pmatrix} I_{rf\_xx} & I_{rf\_xy} & 0 \\ I_{rf\_yx} & I_{rf\_yy} & 0 \\ 0 & 0 & I_{rf\_zz} \end{pmatrix}$	$\begin{pmatrix} 9.2 & 2.4 & 0 \\ 2.4 & 2.8 & 0 \\ 0 & 0 & 11 \end{pmatrix} kgm^2$
Front Frame		
center of mass	(x_{ff}, y_{ff}, z_{ff})	(0.3, 0.7, 0) m
mass	m_{ff}	4 kg
inertia tensor	$\begin{pmatrix} I_{ff\_xx} & I_{ff\_xy} & 0 \\ I_{ff\_yx} & I_{ff\_yy} & 0 \\ 0 & 0 & I_{ff\_zz} \end{pmatrix}$	$\begin{pmatrix} 0.0546 & -0.0162 & 0 \\ -0.0162 & 0.0114 & 0 \\ 0 & 0 & 0.06 \end{pmatrix} kgm^2$
Front Wheel		
radius	r_{fw}	0.35m
mass	m_{fw}	3kg
inertia tensor	$(I_{fw\_xx}, I_{fw\_yy}, I_{fw\_zz})$	(0.14, 0.14, 0.28) kgm^2
Steering Axis		
position	(x_{st}, y_{st}, z_{st})	(0.8, 0.9, 0) m

Table 3.1: Parameters for the basic bicycle model depicted in Figure 3.1

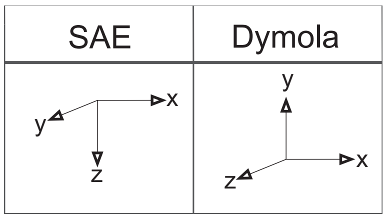


Figure 3.2: Conventions on co-ordinate systems. Left: co-ordinate system defined by the SAE (Society of Automotive Engineers); Right: Dymola specific co-ordinate system

Remark:

The parameter values in Table 3.1 are different from those described in Schwab's paper since Dymola uses a different co-ordinate system.

Unwrapped Model

The model of the bicycle is built by means of the *MultiBondLib* [ZC06] (see Figure 3.3).

Each of the four rigid bodies includes a mass and a corresponding inertia tensor. The position of the front and rear frame masses as well as the position of the steering axis are defined by means of *fixed translation* elements (e.g. position of the rear frame's center of mass is given by $r = \{-0.3, 0.6, 0\}$). Compared to Table 3.1, one could think that the second element of r, namely $r_{12} = 0.6$ is incorrect. However, since the fixed translation element defines the distance between the rear wheel's center-point and the steering axis, the rear wheel's radius ($r_{RW} = 0.3m$) has to be subtracted in order to get a correct result. This is due to the fact that the origin of the global co-ordinate system is at the contact point of the rear wheel. The minus sign of the first element indicates that the vehicle drives in the negative x direction.

Wrapped Model

For the sake of usability, models composed of several parts are wrapped. A wrapped version of the basic bicycle is shown in Figure 3.4.

Compared to Figure 3.3, the wrapped model includes several new components. The additional components used in this model are inputs, outputs, interfaces, sensors, torque sources, a constant source, logical switches and actuated revolute joints instead of the standard ones. On the one side, the actuated revolute joints are used to measure the steering angle and the angular velocity of the rear wheel and on the other side, to provide appropriate control inputs, i.e. a steering torque that has to be applied by the

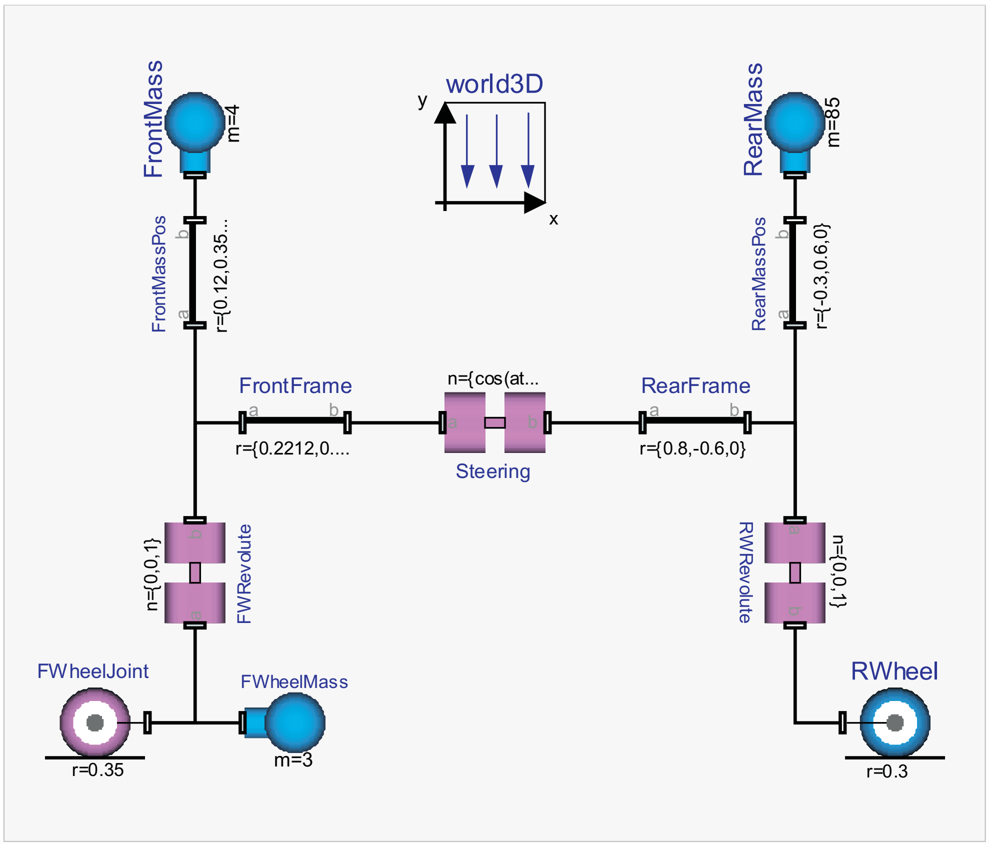
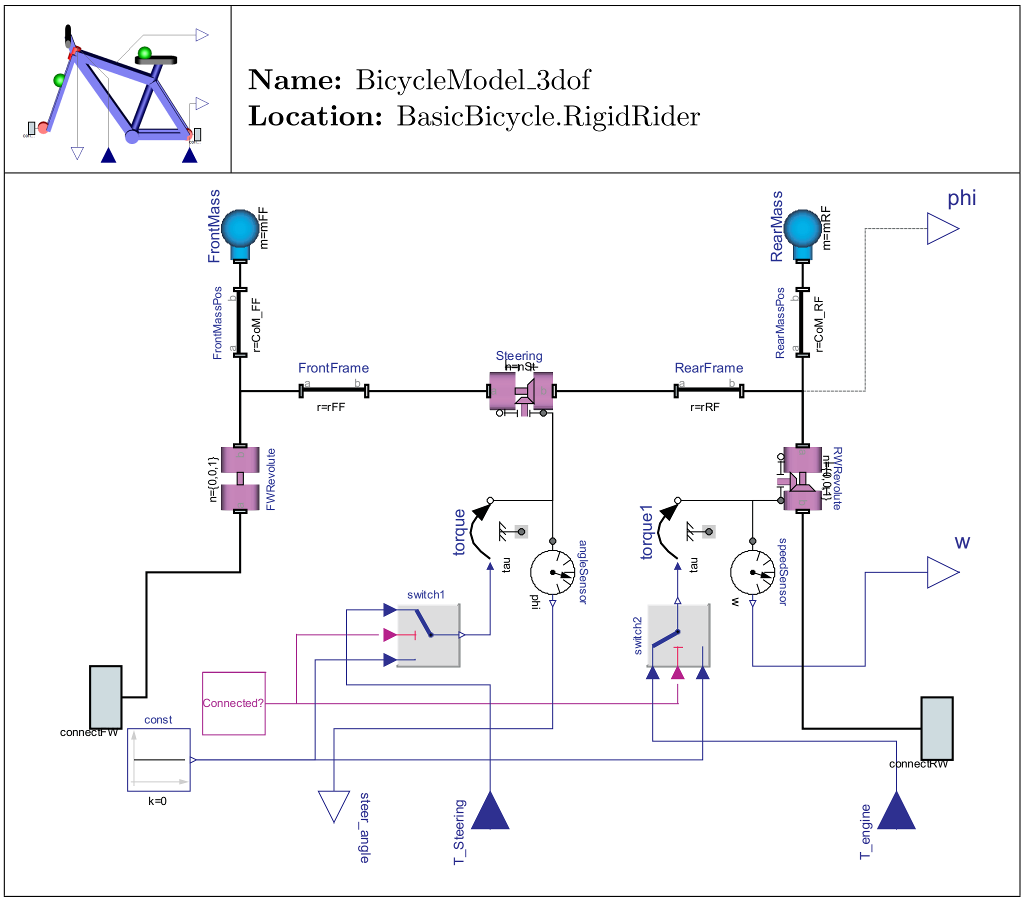


Figure 3.3: Model of a simple 3 d.o.f. bicycle

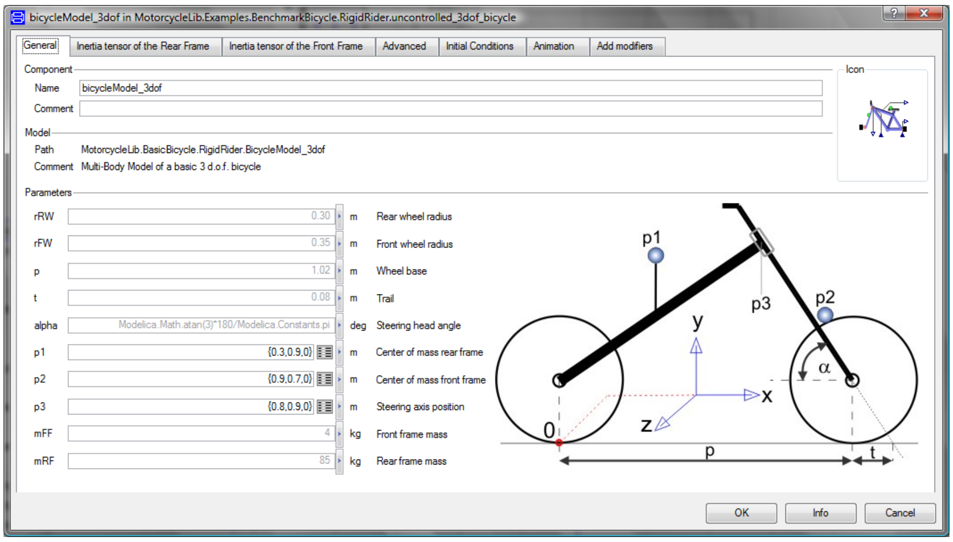
rider in order to track either a lean angle profile or a pre-defined path and a torque source to control the motorcycle's forward velocity. The front and rear wheel are not part of the model anymore. Hence, to connect wheels to the motorcycle, interfaces are provided. The outputs `phi`, `w` and `steer_angle` as well as the inputs `T_Steering` and `T_engine` are needed for control purposes (see Chapter 5). The logical switches between the two inputs and the torque sources prevent an error message if the inputs are disconnected (e.g. an uncontrolled version of the motorcycle has neither inputs nor outputs). With reference to the wrapped model, the user is able to enter the parameters listed in Table 3.1. A "double-click" on the model opens the following parameter window (see Figure 3.5).

With reference to the picture of the bicycle in the parameter window describing the relevant data to be entered and Table 3.1, the bicycle can be modeled in a straightforward fashion. After entering the relevant parameters, either D. Zimmer's [ZC06] or M. Andres' [And] wheels are connected to the model. In the next step the simulation of the uncontrolled bicycle can be started. In order to simulate the model of the bicycle,



Name: BicycleModel_3dof
Location: BasicBicycle.RigidRider

Figure 3.4: Wrapped model of a simple 3 d.o.f. bicycle



bicycleModel_3dof in MotorcycleLib.Examples.BenchmarkBicycle.RigidRider.uncontrolled_3dof_bicycle

General | Inertia tensor of the Rear Frame | Inertia tensor of the Front Frame | Advanced | Initial Conditions | Animation | Add modifiers

Component
Name: bicycleModel_3dof
Comment:

Model
Path: MotorcycleLib.BasicBicycle.RigidRider.BicycleModel_3dof
Comment: Multi-Body Model of a basic 3 d.o.f. bicycle

Parameters

rRW	0.30	m	Rear wheel radius
rFW	0.35	m	Front wheel radius
p	1.02	m	Wheel base
t	0.08	m	Trail
alpha	Modelica.Math.atan(3)*180/Modelica.Constants.pi	deg	Steering head angle
p1	{0.3,0.9,0}	m	Center of mass rear frame
p2	{0.9,0.7,0}	m	Center of mass front frame
p3	{0.8,0.9,0}	m	Steering axis position
mFF	4	kg	Front frame mass
mRF	85	kg	Rear frame mass

Figure 3.5: Parameter window of the basic bicycle

the ideal wheels of the *MultiBondLib* are connected to the model (see Figure 3.6).

Visualization

The parts of the *MultiBondLib* are animated by default. The animation result of the basic bicycle is shown in Figure 3.7. To improve the quality of the animation, Dymola offers a *Visualizers* package capable of visualizing 3-dimensional objects used for the animation. In order to get a proper animation of the model, the pre-defined visualization is turned off. An example of a new defined shape is given below:

```
Modelica.Mechanics.MultiBody.Visualizers.Advanced.Shape FWSuspLeft(
  shapeType="box",
  r_shape={0,0,-0.050} "Offset in order to move the object",
  color={0,0,100},
  width=0.03 "Width of visual object",
  height=0.03 "Height of visual object",
  lengthDirection=Steering.n "Vector in length direction",
  length=-FrontFrame.length "Length of visual object",
  r=FrontFrame.frame_b.P.x "Origin of visual object",
  R=MB.Frames.Orientation(T=FrontFrame.frame_b.P.R,w=zeros(3));
```

The *lines of code* shown above are used to visualize the left tube of the front forks.

The result of the modified visualization is depicted in Figure 3.8.

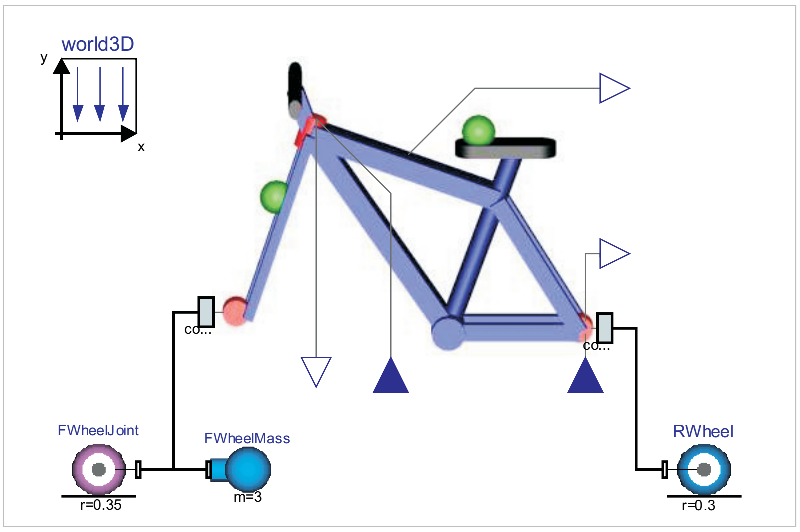


Figure 3.6: Wrapped 3 d.o.f. bicycle with ideal wheels

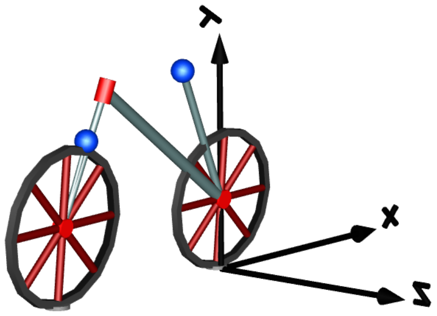


Figure 3.7: Automatically generated animation result of the basic 3 d.o.f. bicycle

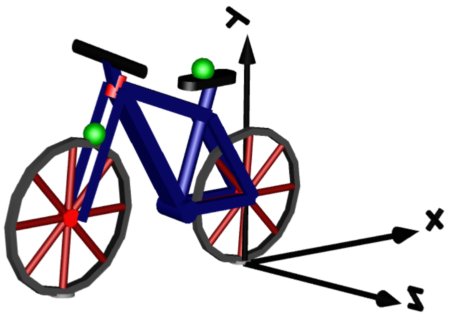


Figure 3.8: Modified animation result of the basic 3 d.o.f. bicycle

3.1.4 State Selection

State Variables of Mechanical Bond Graphs

The natural state variables of bond graphs are potential and flow variables. State variables itself describe the state of a dynamical system. Such variables are always outputs of integrators. In the following, the state variables of mechanical bond graphs are derived.

Newton's second law for a single force acting on a mass is given by:

$$F = m \cdot a = m \cdot \frac{dv}{dt} \tag{3.1}$$

The integral form of the equation above is obtained by solving it for the velocity v. Hence,

$$v = \frac{1}{m} \int F \cdot dt \tag{3.2}$$

The linear equation of a spring element is given by:

$$F = k \cdot x \tag{3.3}$$

Differentiating both sides of the equation results in:

$$\frac{dF}{dt} = k \frac{dx}{dt} = k \cdot v \tag{3.4}$$

Now, both sides are multiplied by dt

$$dF = k \cdot v \cdot dt \tag{3.5}$$

By taking the integral of both sides, F results in:

$$F = k \int v \cdot dt \tag{3.6}$$

Hence, the natural state variables of mechanical bond graphs are forces (torques) and (angular) velocities. Unfortunately the state variables of mechanical systems are (angular) positions and (angular) velocities. However, in a bond graphic representation the (angular) positions are not needed in order to get a complete description of the dynamics of the system. But without any positional information we do not know whether two bodies occupy the same space. Thus the *BondLib* and the *MultiBondLib* are using the positions and velocities of a body as state variables.

Symbolic Pre-Processing

Since Dymola works with a-causal sets of equations, symbolic pre-processing is performed in order to simulate the system. If algebraic loops or structural singularities appear, Dymola offers algorithms capable of handling such problems. The former ones are eliminated with tearing algorithms (Tarjan algorithm). The latter ones are eliminated by means of the Pantelides algorithm. Concerning the set of equations in combination with potential algebraic loops and/or structural singularities, the efficiency of the resulting simulation is affected (e.g. if a structural singularity appears, several new equations are introduced in order to eliminate it).

Selection of State Variables

Beside the points mentioned above, the selection of state variables is also very important for the efficiency of the resulting simulation.

Sometimes many possible sets of state variables exist. In this case, Dymola uses dynamic state selection. This means that appropriate states are chosen at run-time. Usually, this leads to additional equations since Dymola uses the Pantelides algorithm to perform this operation. An undesirable side effect is that the simulation time increases. To avoid dynamic state selection, joints offer the parameter `enforceStates` to declare state variables. The following code-segment of the equation layer demonstrates the state selection via the `enforceStates` parameter:

```
parameter Boolean enforceStates =  false;
Real phi(stateSelect = if enforceStates then StateSelect.always
else StateSelect.prefer);
```

If the parameter `enforceStates = true`, the selected state variables are always the relative (angular) position and (angular) velocity of a joint.

The state vector of the basic bicycle is given by:

$$x = \begin{pmatrix} RWheel.xA \\ RWheel.xB \\ leanAngle \\ leanRate \\ FWRevolute.phi \\ FWRevolute.w \\ Steering.phi \\ Steering.w \\ RWRevolute.phi \\ dynamic\ state \end{pmatrix} = \begin{pmatrix} x_{long} \\ x_{lat} \\ \phi \\ \dot{\phi} \\ \varphi_{FW} \\ \dot{\varphi}_{FW} \\ \delta \\ \dot{\delta} \\ \varphi_{RW} \\ dynamic\ state \end{pmatrix}$$

The first two states are preferred states of the rear wheel if `enforceStates = false`. The states δ, $\dot{\delta}$, ϕ and $\dot{\phi}$ are responsible for the stability of the motorcycle. In Chapter 4, these states are needed to perform an eigenvalue analysis. Furthermore, the states are used to design a controller. For the states φ_{FW} and $\dot{\varphi}_{FW}$ the parameter `enforceStates` of the front wheel revolute joint was set `true`. The parameter φ_{RW} is a preferred state of the rear wheel and the last state is dynamically selected by Dymola.

3.2 Basic Bicycle Model - Movable Rider

This section is intended to extend the former described model by an additional d.o.f. allowing the rider's upper body to lean sideways. Schwab et al. introduced the model of a 4 d.o.f. bicycle in their recently published paper [SKM08]. Figure 3.9 shows the extended model of the bicycle. The additional d.o.f. is represented by the angle γ. Apart from the additional d.o.f. the definition of the bicycle is equal to the former one.

The parameters for the benchmark bicycle are listed in Table 3.2.

3.2.1 Model of the 4 d.o.f. Bicycle

Wrapped Model

The wrapped model is depicted in Figure 3.10. Roughly speaking, the model is based on the former one with a bunch of new parts representing the rider's upper body.

A "double-click" on the model opens the parameter window shown in Figure 3.11.

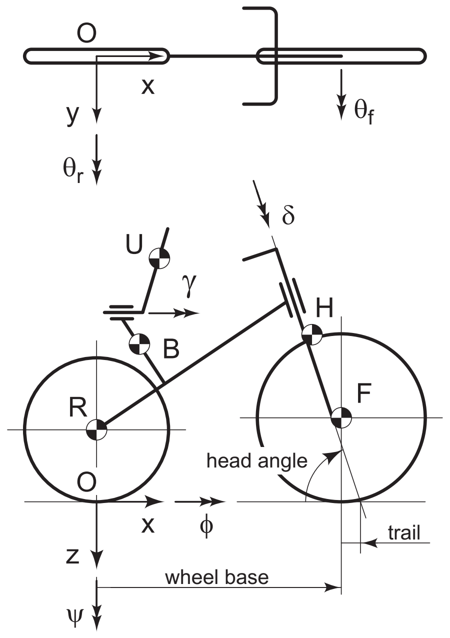
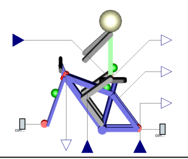


Figure 3.9: Model of a simple 4 d.o.f. bicycle (Source: [SKM08], p.2).

Parameter	Symbol	Value
Rear Frame (including lower rider body)		
center of mass	(x_{rf}, y_{rf}, z_{rf})	(0.345, 0.765, 0) m
mass	m_{rf}	34kg
inertia tensor	$\begin{pmatrix} I_{rf\_xx} & I_{rf\_xy} & 0 \\ I_{rf\_yx} & I_{rf\_yy} & 0 \\ 0 & 0 & I_{rf\_zz} \end{pmatrix}$	$\begin{pmatrix} 3.869 & 1.3 & 0 \\ 1.3 & 1.272 & 0 \\ 0 & 0 & 4.667 \end{pmatrix} kgm^2$
Front Frame		
center of mass	(x_{ub}, y_{ub}, z_{ub})	(0.27, 0.99, 0) m
mass	m_{ub}	51 kg
inertia tensor	$\begin{pmatrix} I_{ub\_xx} & I_{ub\_xy} & 0 \\ I_{ub\_yx} & I_{ub\_yy} & 0 \\ 0 & 0 & I_{ub\_zz} \end{pmatrix}$	$\begin{pmatrix} 4.299 & 1.444 & 0 \\ 1.444 & 1.413 & 0 \\ 0 & 0 & 5.186 \end{pmatrix} kgm^2$

Table 3.2: Parameters for the extended bicycle model depicted in Figure 3.9



Name: BicycleModel_4dof
Location: BasicBicycle.MovableRider

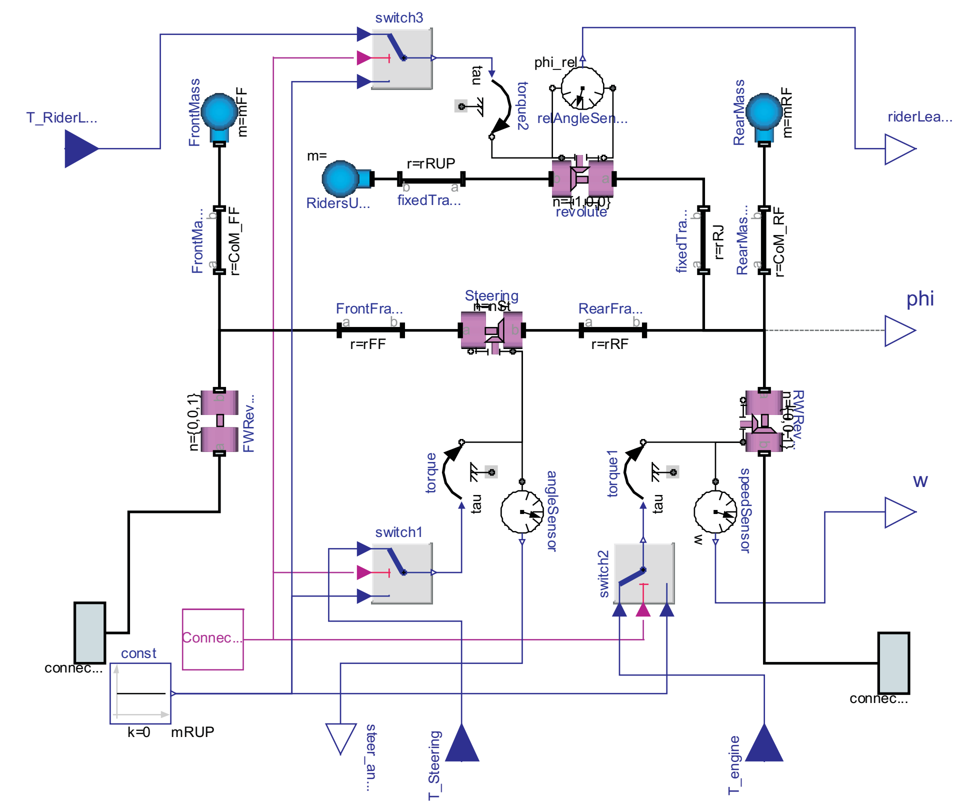
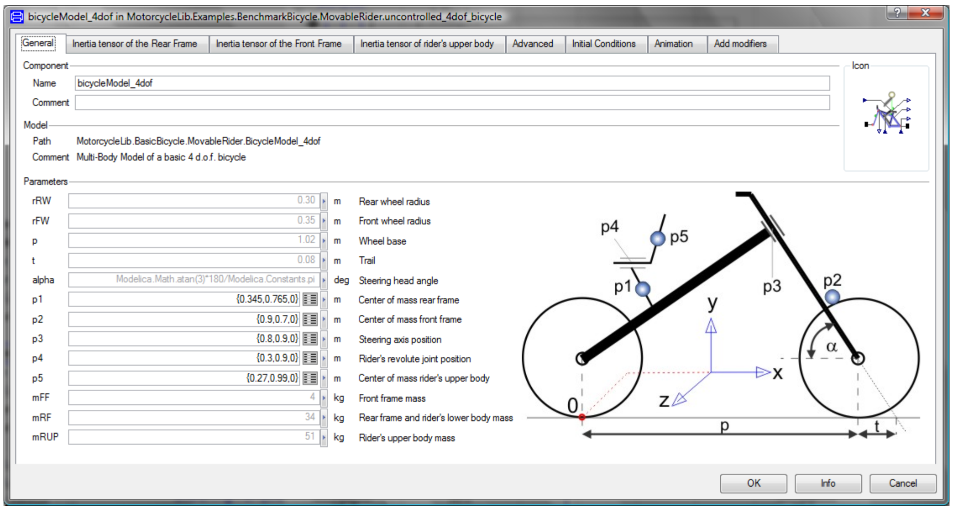


Figure 3.10: Wrapped model of the extended 4 d.o.f. bicycle


bicycleModel_4dof in MotorcycleLib.Examples.BenchmarkBicycle.MovableRider.uncontrolled_4dof_bicycle

General | Inertia tensor of the Rear Frame | Inertia tensor of the Front Frame | Inertia tensor of rider's upper body | Advanced | Initial Conditions | Animation | Add modifiers

Component

Name	bicycleModel_4dof
Comment	

Model

Path	MotorcycleLib.BasicBicycle.MovableRider.BicycleModel_4dof
Comment	Multi-Body Model of a basic 4 d.o.f. bicycle

Parameters

rRW	0.30	m	Rear wheel radius
rFW	0.35	m	Front wheel radius
p	1.02	m	Wheel base
t	0.08	m	Trail
alpha	Modelica.Math.atan(3)*180/Modelica.Constants.pi	deg	Steering head angle
p1	{0.345,0.765,0}	m	Center of mass rear frame
p2	{0.9,0.7,0}	m	Center of mass front frame
p3	{0.8,0.9,0}	m	Steering axis position
p4	{0.3,0.9,0}	m	Rider's revolute joint position
p5	{0.27,0.99,0}	m	Center of mass rider's upper body
mFF	4	kg	Front frame mass
mRF	34	kg	Rear frame and rider's lower body mass
mRUP	51	kg	Rider's upper body mass

Figure 3.11: Parameter window of the 4 d.o.f. bicycle

Visualization

The animation of the bicycle is depicted in Figure 3.12.

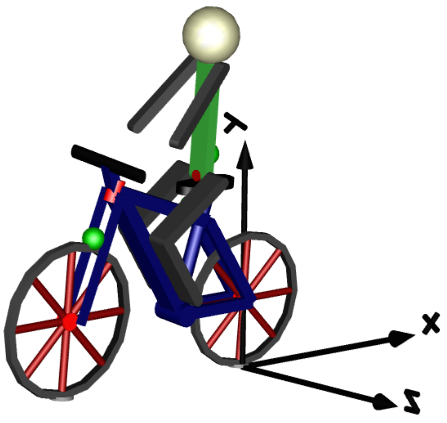


Figure 3.12: Modified animation result of the 4 d.o.f. bicycle

3.2.2 State Selection

The state selection is based on the 3 d.o.f. model. Additionally, the states γ and $\dot{\gamma}$ which represent the lean angle and lean rate of the rider's upper body relative to the rear frame are selected.

3.3 Basic Motorcycle Model

3.3.1 Definition of Motorcycles

The motorcycle introduced in this section uses the same definitions as the basic bicycle. As already mentioned it is based on V. Cossalter's model [Cos06]. In the beginning, the motorcycle has 3 d.o.f.. Later on, with reference to the *WheelsAndTires* [And] library, the model can be extended by an additional d.o.f. allowing the motorcycle's rear frame to move laterally. That means that lateral slip of the wheels is incorporated.

3.3.2 Geometry of Motorcycles

Figure 3.17 depicts the geometrical definition of V. Cossalter's motorcycle model ([Cos06], page 262).

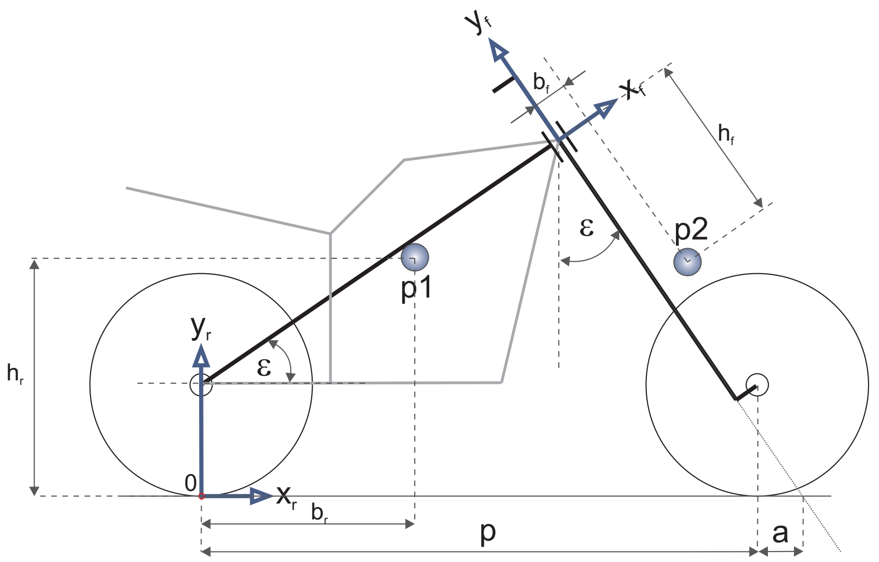


Figure 3.13: Geometric description of a basic 3 d.o.f. motorcycle

In contrast to the bicycle model, the position of the steering axis is not given anymore. This position is automatically calculated in the background. Furthermore, a second co-ordinate system is introduced to describe the geometry of the front frame. The origin of this co-ordinate system is the position of the steering axis revolute joint, and the direction of the y_f-axis is coincident with the steering axis. The reason for the orientation of the front frame co-ordinate system is to avoid *products of inertia*. The inertia tensor of a body in space is given by:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

If the axes of a co-ordinate system are coincident with the principle axes of a *symmetrical* body, the products of inertia are eliminated and the remaining elements are the *principal moments of inertia*:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

3.3.3 Model of the Basic Motorcycle

The objective is to feed the model with exactly the same parameters as listed in Table 3.3. Since Dymola uses world (global) co-ordinates to describe the whole geometry of a multibody system, some parameters of the motorcycle have to be re-calculated in order to develop the model. Furthermore, the calculation of the steering axis co-ordinates should be done automatically in the background. If the front frame center of mass co-ordinates are entered, the re-calculation of the center of mass position with reference to the global co-ordinate system has to be done automatically as well.

Remark

Since the wheels are considered to be ideal, the cornering and cambering stiffness of the front and rear wheel are neglected.

Calculation of the Steering Axis Co-Ordinates

The following calculations are done with reference to Figure 3.14.

The normal trail a_n is given by:

$$a_n = cos(\epsilon) \cdot a = 0.1034m \tag{3.7}$$

Parameter	**Symbol**	**Value**
wheel base	p	1.414m
trail	a	0.116m
caster angle	ϵ	27° (0.47124 rad)
Rear Wheel		
radius	r_{rw}	0.305m
axial inertia	I_{rw_zz}	1.05 kgm^2
cornering stiffness	$K_{\lambda r}$	15.8 kN/rad
cambering stiffness	$K_{\varphi r}$	1.32 kN/rad
Rear Frame - (x_r, y_r) co-ordinate system		
center of mass	$(b_r, h_r, 0)$	(0.48, 0.616, 0) m
mass	m_{rf}	217.5 kg
inertia tensor	$\begin{pmatrix} I_{rf\_xx} & I_{rf\_xy} & 0 \\ I_{rf\_yx} & I_{rf\_yy} & 0 \\ 0 & 0 & I_{rf\_zz} \end{pmatrix}$	$\begin{pmatrix} 31.2 & 1.74 & 0 \\ 1.74 & 21.08 & 0 \\ 0 & 0 & 0 \end{pmatrix} kgm^2$
Front Frame - (x_f, y_f) co-ordinate system		
center of mass	$(b_f, h_f, 0)$	(0.024, 0.461, 0) m
mass	m_{ff}	30.7 kg
inertia tensor	$(I_{ff\_xx}, I_{ff\_yy}, I_{ff\_zz})$	(1.23, 0.44, 0)kgm^2
steering damper	c	6.8 $\frac{Nm \cdot s}{rad}$
Front Wheel		
radius	r_{fw}	0.305m
axial inertia	I_{fw_zz}	0.72 kgm^2
cornering stiffness	$K_{\lambda f}$	11.2 kN/rad
cambering stiffness	$K_{\varphi f}$	0.94 kN/rad

Table 3.3: Parameters for the basic motorcycle model depicted in Figure 3.17

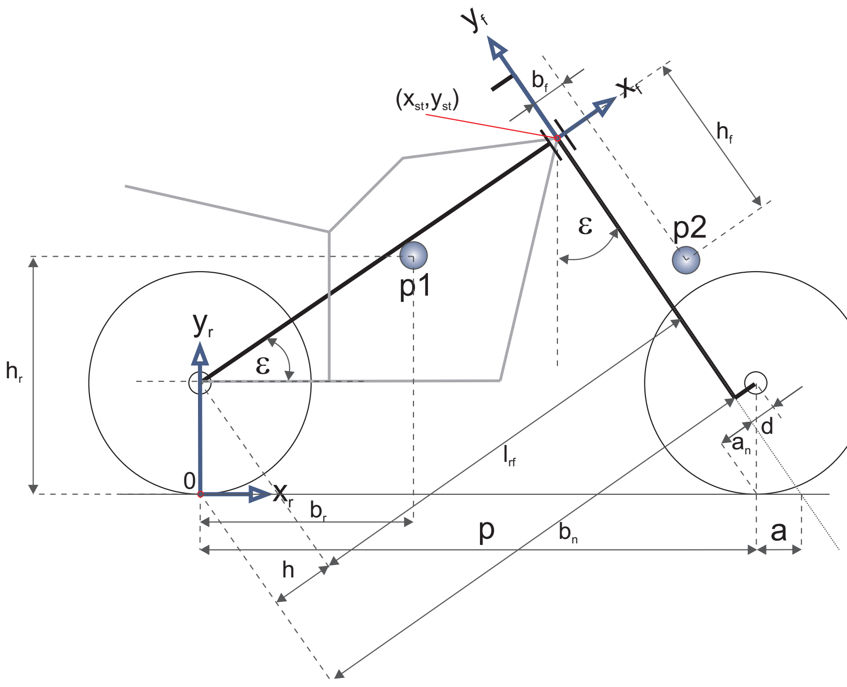


Figure 3.14: Detailed geometric description of a basic 3 d.o.f. motorcycle

The normal rear trail b_n is

$$b_n = (p + a) \cdot cos(\epsilon) = 1.3632m \tag{3.8}$$

By introducing the auxiliary variable

$$h = sin(\epsilon) \cdot r_{rw} = 0.1385m \tag{3.9}$$

the length of the rear frame becomes

$$l_{rf} = b_n - h = 1.2248m \tag{3.10}$$

Hence, the x and y co-ordinates of the steering axis are

$$x_{st} = cos(\epsilon) \cdot l_{rf} = 1.0913m \tag{3.11}$$

$$y_{st} = sin(\epsilon) \cdot l_{rf} + r_{rw} = 0.8610m \tag{3.12}$$

Calculation of the Front Frame Co-Ordinates

Figure 3.15 depicts the front frame of the motorcycle with the appropriate co-ordinates to calculate the center of mass.

By means of another auxiliary variable

$$dx = p + a - x_{st} = 0.4387m \tag{3.13}$$

the length of the line between the steering axis and the intersection between the steering axis and x-axis is given by

$$l_{ff} = \frac{dx}{sin(\epsilon)} = 0.9664m \tag{3.14}$$

with

$$h_2 = tan(\epsilon) \cdot b_f = 0.0122m \tag{3.15}$$

and

$$h_3 = l_{ff} - h_f + h_2 = 0.5176m \tag{3.16}$$

The y-coordinate of the front frame center of mass becomes

$$y_f = cos(\epsilon) \cdot h_3 = 0.4612m \tag{3.17}$$

Finally, with two additional auxiliary variables

$$x_1 = sin(\epsilon) \cdot h_f = 0.2093m \tag{3.18}$$

$$x_2 = cos(\epsilon) \cdot b_f = 0.0214m \tag{3.19}$$

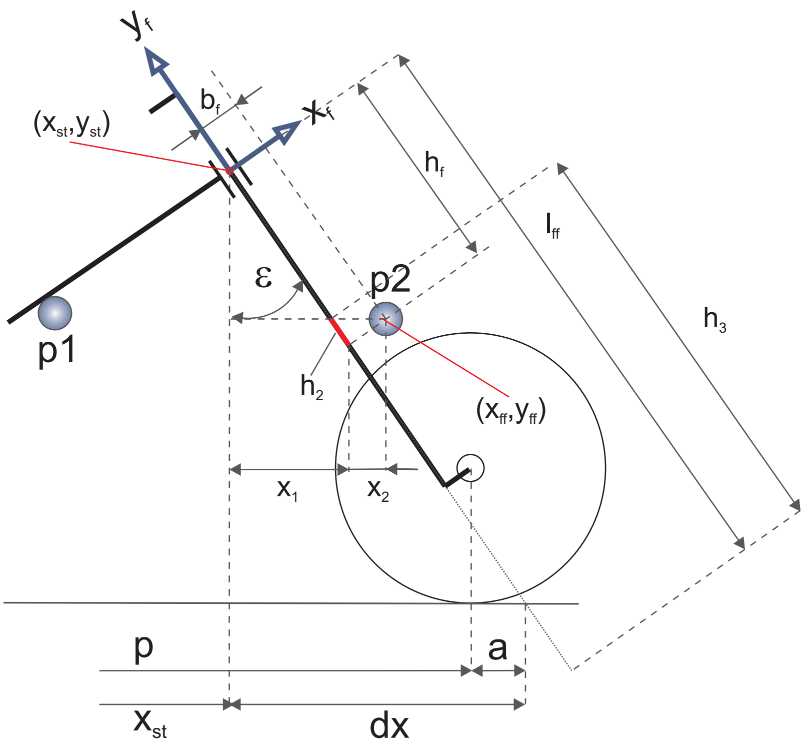


Figure 3.15: Detailed geometric description of the front frame of a basic 3 d.o.f. motorcycle

the x co-ordinate is given by

$$x_f = x_{st} + x_1 + x_2 = 1.3220m \tag{3.20}$$

Hence, the center of mass $p2 = (x_f, y_f, 0) = (1.3220, 0.4612, 0)m$.

Transformation of the Front-Frame-Inertia Tensor

The last step in order to develop a model of the motorcycle is to transform the body co-ordinates of the front frame inertia tensor into world (global) co-ordinates. As mentioned before, the principal axes of the front frame are coincident with the co-ordinate system (x_f, y_f). To obtain an inertia tensor valid for Dymola's world co-ordinate system, it is rotated about the z-axis. The amount of rotation is equal to the value of the caster angle.

The inertia tensor of the front frame co-ordinate system is:

$$I = \begin{bmatrix} 1.23 & 0 & 0 \\ 0 & 0.44 & 0 \\ 0 & 0 & 0 \end{bmatrix} kgm^2$$

The rotation matrix about the z-axis is given by:

$$R = \begin{bmatrix} cos(\epsilon) & -sin(\epsilon) & 0 \\ sin(\epsilon) & cos(\epsilon) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

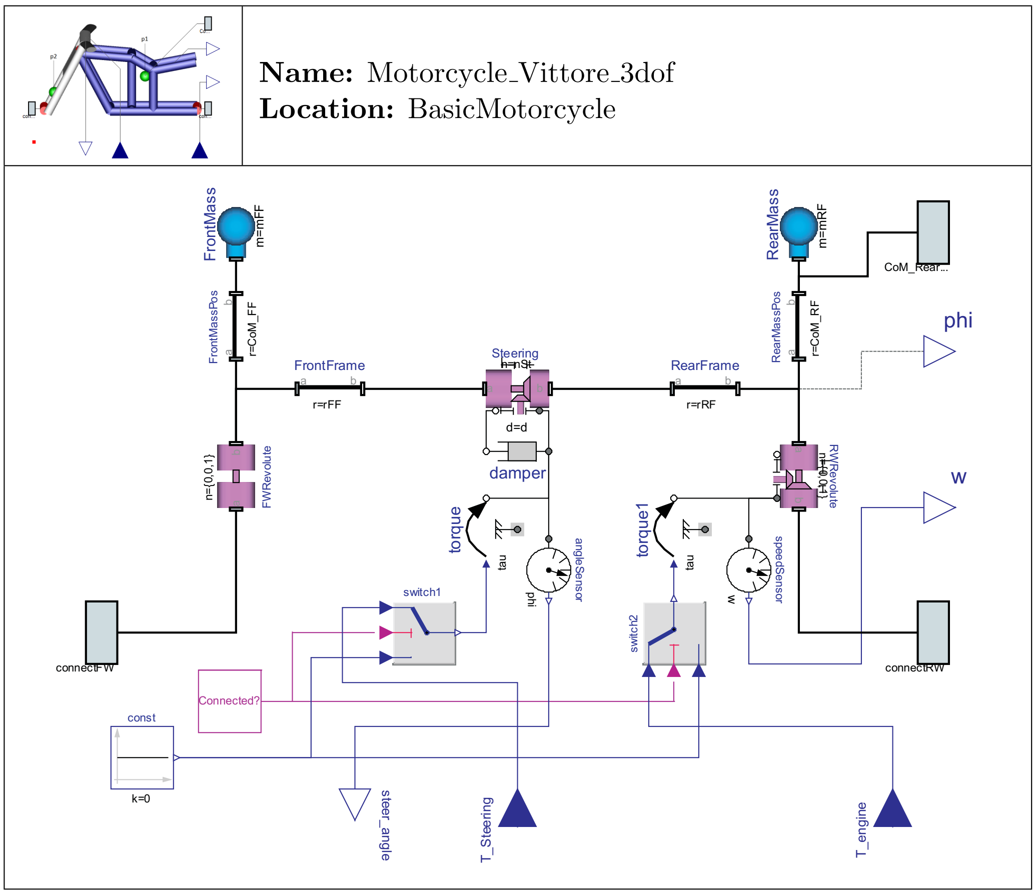
By definition, the inertia tensor valid for Dymola's world co-ordinate system becomes:

$$I\_new = R' \cdot I \cdot R = \begin{bmatrix} 1.0672 & 0.3196 & 0 \\ 0.3196 & 0.6028 & 0 \\ 0 & 0 & 0 \end{bmatrix} kgm^2 \tag{3.22}$$

Wrapped Model

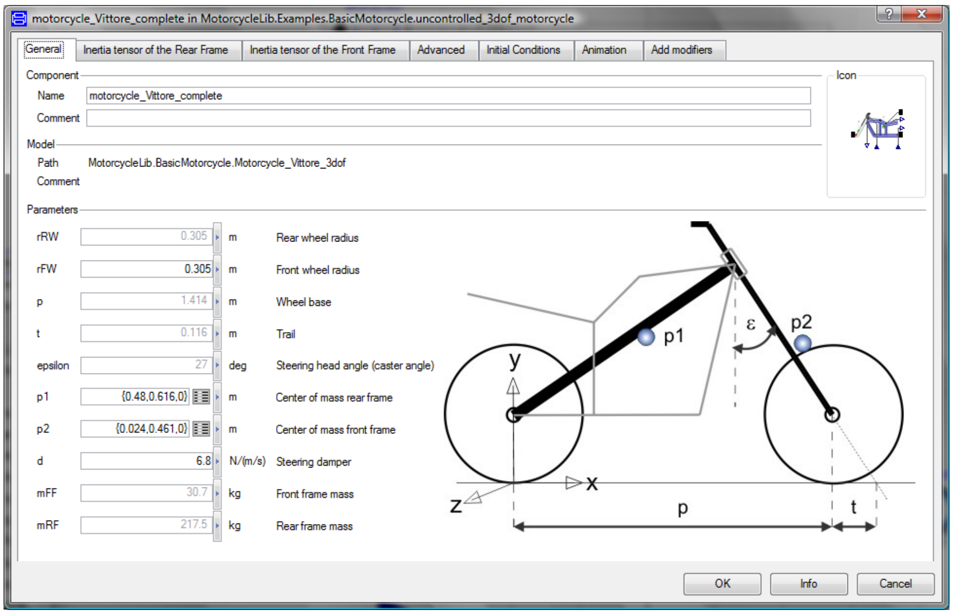
The wrapped model of the basic motorcycle is shown in Figure 3.16

A "double-click" on the model opens the parameter window depicted in Figure 3.17.



Name: Motorcycle_Vittore_3dof
Location: BasicMotorcycle

Figure 3.16: Wrapped model of a 3 d.o.f motorcycle



motorcycle_Vittore_complete in MotorcycleLib.Examples.BasicMotorcycle.uncontrolled_3dof_motorcycle

Component

Name: motorcycle_Vittore_complete

Comment:

Model

Path: MotorcycleLib.BasicMotorcycle.Motorcycle_Vittore_3dof

Comment:

Parameters

rRW	0.305	m	Rear wheel radius
rFW	0.305	m	Front wheel radius
p	1.414	m	Wheel base
t	0.116	m	Trail
epsilon	27	deg	Steering head angle (caster angle)
p1	{0.48,0.616,0}	m	Center of mass rear frame
p2	{0.024,0.461,0}	m	Center of mass front frame
d	6.8	N/(m/s)	Steering damper
mFF	30.7	kg	Front frame mass
mRF	217.5	kg	Rear frame mass

Figure 3.17: Parameter window of the 3 d.o.f. motorcycle

Visualization

Unlike former visualizations of the bicycles, the wheels of [And] are now connected to the motorcycle model. The model in combination with the wheels is depicted in Figure 3.18.

Again, for esthetical purposes, the motorcycle model is visualized (see Figure 3.19).

3.3.4 State Selection

The states of the motorcycle are the same as for the basic bicycle model.

3.4 Advanced Motorcycle Models

In general, a motorcycle includes two different modes (refer to ([Cos06], page 260 or [LS06], page 57):

- *In-plane* modes
- *out-of-plane* modes

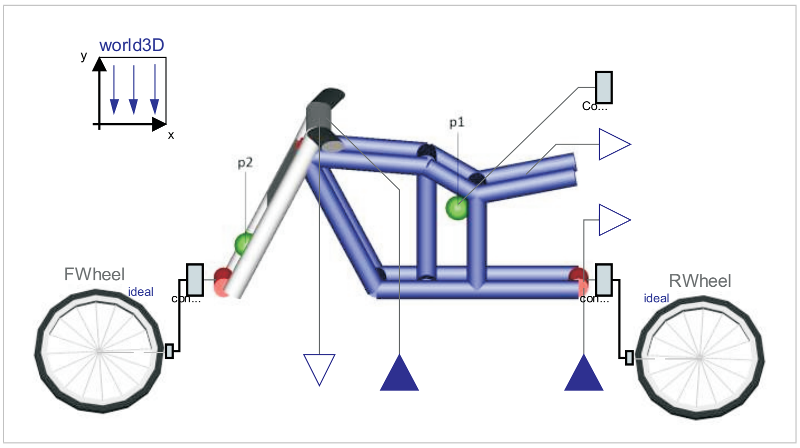


Figure 3.18: Wrapped 3 d.o.f. motorcycle with ideal wheels of the *WheelsAndTires* library

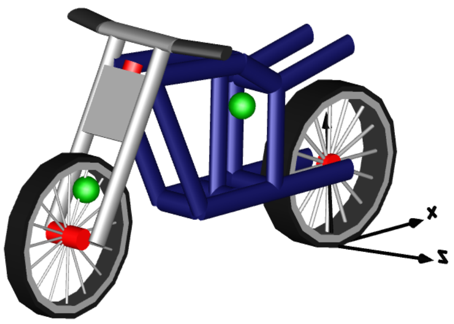


Figure 3.19: Modified animation result of the 3 d.o.f. motorcycle

The in-plane modes involve motions in the vertical direction (e.g. suspension and wheel motion), whereas the out-of-plane modes involve e.g. yaw, lean and steering angle lateral displacement. Hence, the in-plane modes are related to riding comfort, while the out-of-plane modes are related to stability and handling of motorcycles.

Until now, several simple 3 and 4 d.o.f. models were developed. Due to the fact that the parts are rigid (i.e. no twist frame flexibility) and neither rear nor front suspensions were implemented, no in-plane modes appeared so far. In the following, two high-fidelity models developed by Sharp et al. are introduced [SL01] and [SEL04].

3.4.1 Definition of the SL2001 Motorcycle

The basic components of the SL2001 model [SL01] are (refer to [Eva03], page 70/181):

- a front and a rear frame connected via an inclined revolute joint
- a front and a rear wheel
- yaw, roll and lateral translation freedoms of the rear frame
- one d.o.f. for the steering axis and another one for the twist axis relative to the rear frame
- non-ideal tire characteristics[1] (e.g. slip)
- aerodynamic forces
- an additional d.o.f for the rider's upper body to lean sideways
- main frame bounce, pitch and suspension freedoms

3.4.2 Geometry of the SL2001 Motorcycle

Figure 3.20 depicts the model of Sharp and Limebeer, published in 2001.

The geometry of the motorcycle is described with reference to the co-ordinates listed in Table 3.4.

The model is composed of six parts:

1. a front frame including the front forks and handle bar assembly
2. a rear frame including the lower rigid body of the rider
3. a swinging arm including the rear suspension
4. the rider's upper body

[1]The motorcycle model will be evaluated with reference to D. Zimmer's ideal tires and M. Andres' non-ideal tires.

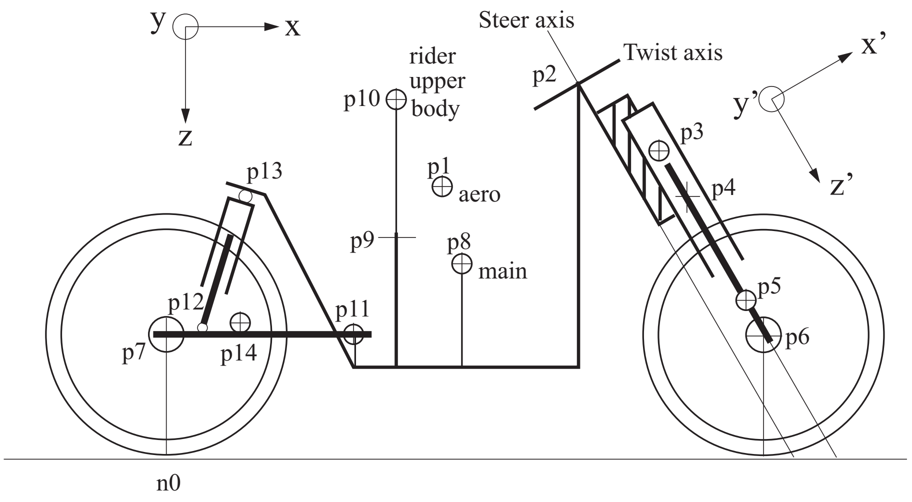


Figure 3.20: Geometric description of the SL2001 motorcycle model (Source:[SL01], p.126).

Name	Coordinates	Description
p1	(x1, y1, 0)	Aerodynamics center of pressure
p2	(x2, y2, 0)	Steer and twist axis joint with rear frame
p3	(x3, y3, 0)	Center of mass front frame
p4	(x4, y4, 0)	Connection between front frame and front suspension
p5	(x5, y5, 0)	Center of mass front suspension
p6	(x6, y6, 0)	Front wheel attachment point
p7	(x7, y7, 0)	Rear wheel attachment point
p8	(x8, y8, 0)	Center of mass of the rear frame
p9	(x9, y9, 0)	Attachment point rider's upper body to rear frame
p10	(x10, y10, 0)	Center of mass rider's upper body
p11	(x11, y11, 0)	Attachment point swinging arm to rear frame
p12	(x12, y12, 0)	Attachment point rear suspension to swinging arm
p13	(x13, y13, 0)	Attachment point rear suspension to rear frame
p14	(x14, y14, 0)	Center of mass swinging arm

Table 3.4: Co-ordinates of the SL2001 motorcycle model

5. a rear wheel and
6. a front wheel

The parameter values of the SL2001 motorcycle can be found on: `http://www3.imperial.ac.uk/controlandpower/research/motorcycles/programs`

3.4.3 Definition of Sharp's Improved Motorcycle Model

This model has the same number of bodies and freedoms as the SL2001 model ([Eva03], page 124/181). Among others, some new components such as a telelever front suspension and a chain drive are considered. However, both are not included in the *MotorcycleLib*. Additionally, a monoshock rear suspension as can be found in modern motorcycle designs was taken into account. In order to develop a model of a monoshock rear suspension, some additional co-ordinates are needed.

3.4.4 Geometry of Sharp's Improved Motorcycle Model

Sharp's improved motorcycle model is illustrated in Figure 3.21. The geometry is

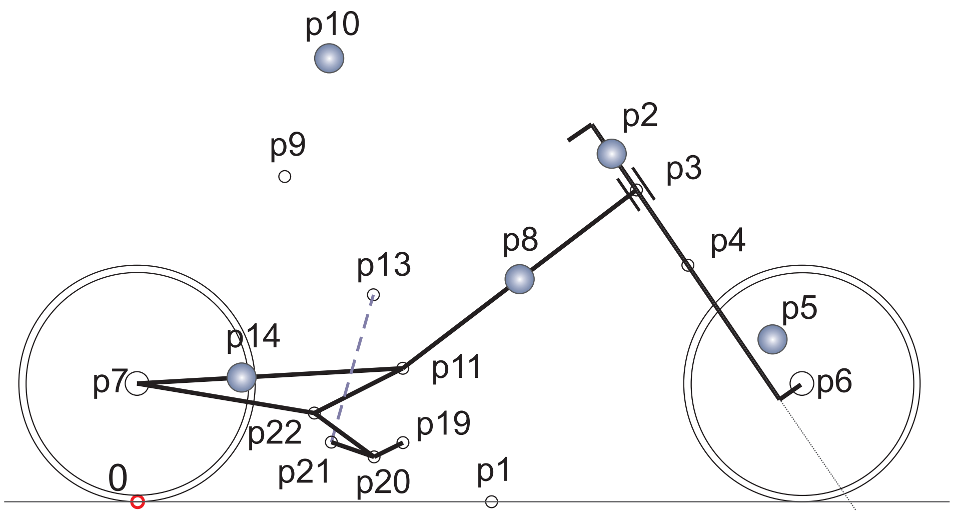


Figure 3.21: Geometry of Sharp's improved motorcycle model

described with reference to the co-ordinates listed in Table 3.5. The parameter values of the improved motorcycle can be found in the paper [SEL04].

Name	**Coordinates**	**Description**
p1	(x1, y1, 0)	Aerodynamic forces reference point
p2	(x2, y2, 0)	Steer and twist axis joint with rear frame
p3	(x3, y3, 0)	Center of mass front frame
p4	(x4, y4, 0)	Connection between front frame and front suspension
p5	(x5, y5, 0)	Center of mass front suspension
p6	(x6, y6, 0)	Front wheel attachment point
p7	(x7, y7, 0)	Rear wheel attachment point
p8	(x8, y8, 0)	Center of mass of the rear frame
p9	(x9, y9, 0)	Attachment point rider's upper body to rear frame
p10	(x10, y10, 0)	Center of mass rider's upper body
p11	(x11, y11, 0)	Attachment point swinging arm to rear frame
p13	(x13, y13, 0)	Attachment point rear suspension to rear frame
p14	(x14, y14, 0)	Center of mass swinging arm
p19	(x19, y19, 0)	Swinging link pivot for rear suspension
p20	(x20, y20, 0)	Lower connection link for rear suspension
p21	(x21, y21, 0)	Rear Suspension/Link Attachment
p22	(x22, y22, 0)	Upper connecting link pivot for rear suspension

Table 3.5: Co-ordinates of Sharp's improved motorcycle model

3.4.5 Models of the Advanced Motorcycles

Compared to the basic models explained so far, models containing in-plane dynamics are much more complicated. Thus, the wrapping of these models is really essential.

Figure 3.22 depicts a simplified version of the SL2001 model (i.e. the twist axis is not included, there are no *limit stops* for the suspensions, no actuated revolute joints, no sensors, etc.). From Figure 3.22 one can see the interfaces defined in order to get useful objects to wrap the models.

Wrapped Models

Each of the six parts is designed in a fully object-oriented fashion.

Front Frame

The front frame includes the steering and twist axis, the front mass, a steering damper, a spring/damper element for the twist axis, a telescopic version of the front suspension, the front wheel revolute joint an two interfaces. The twist axis is needed to consider torsional compliance. The center of mass of the front frame mass is modeled with a *fixed translation* element. The front frame is connected to the front wheel and the rear frame. To design a controller, the steering angle is measured and a steering torque input is provided. If the motorcycle is uncontrolled, a logic switch prevents an error message if the inputs are disconnected.

Front Suspension

Two different kinds of telescopic front suspensions are provided. The classic version of the telescopic front suspension has the spring/damper-elements at the bottom and the rigid part at the top. The second one is an upside down version of the former one. Furthermore, new spring and damper elements are developed in order to implement real spring/damper-characteristics.

Figure 3.24 shows the classic version of the telescopic front suspension.

The front suspensions include a *fixed translation* element representing the rigid part, a mass connected to another fixed translation to model the position of the center of mass and an *actuated prismatic joint* which defines the translational direction of the spring/damper-elements. The parts connected to the prismatic joints are a linear spring and a linear damper and two *elasto-gaps*. The latter ones are needed in order to limit the elongation of the front suspension (see Chapter 3.4.5). The front suspension

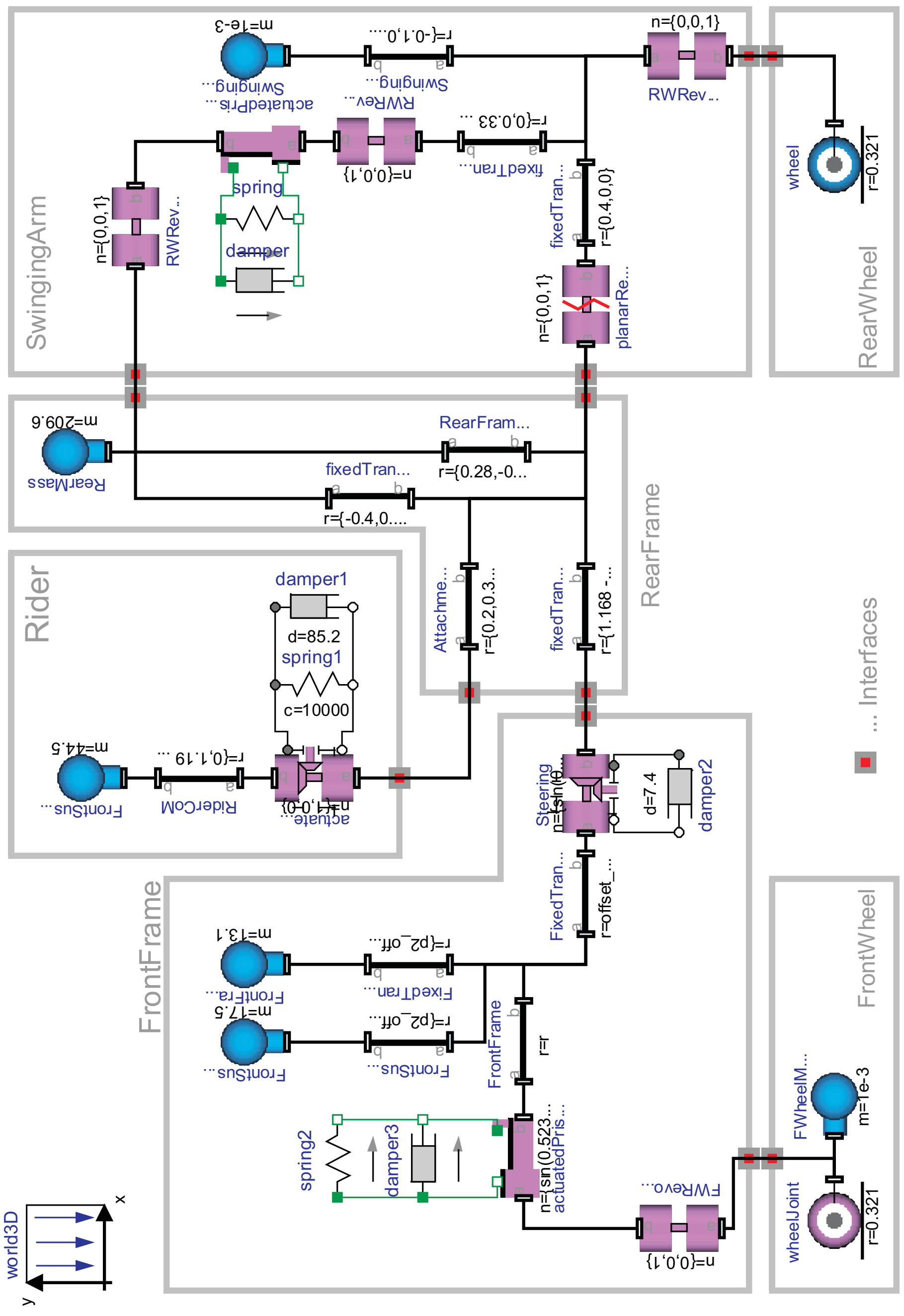


Figure 3.22: Simplified unwrapped model of the SL2001 model

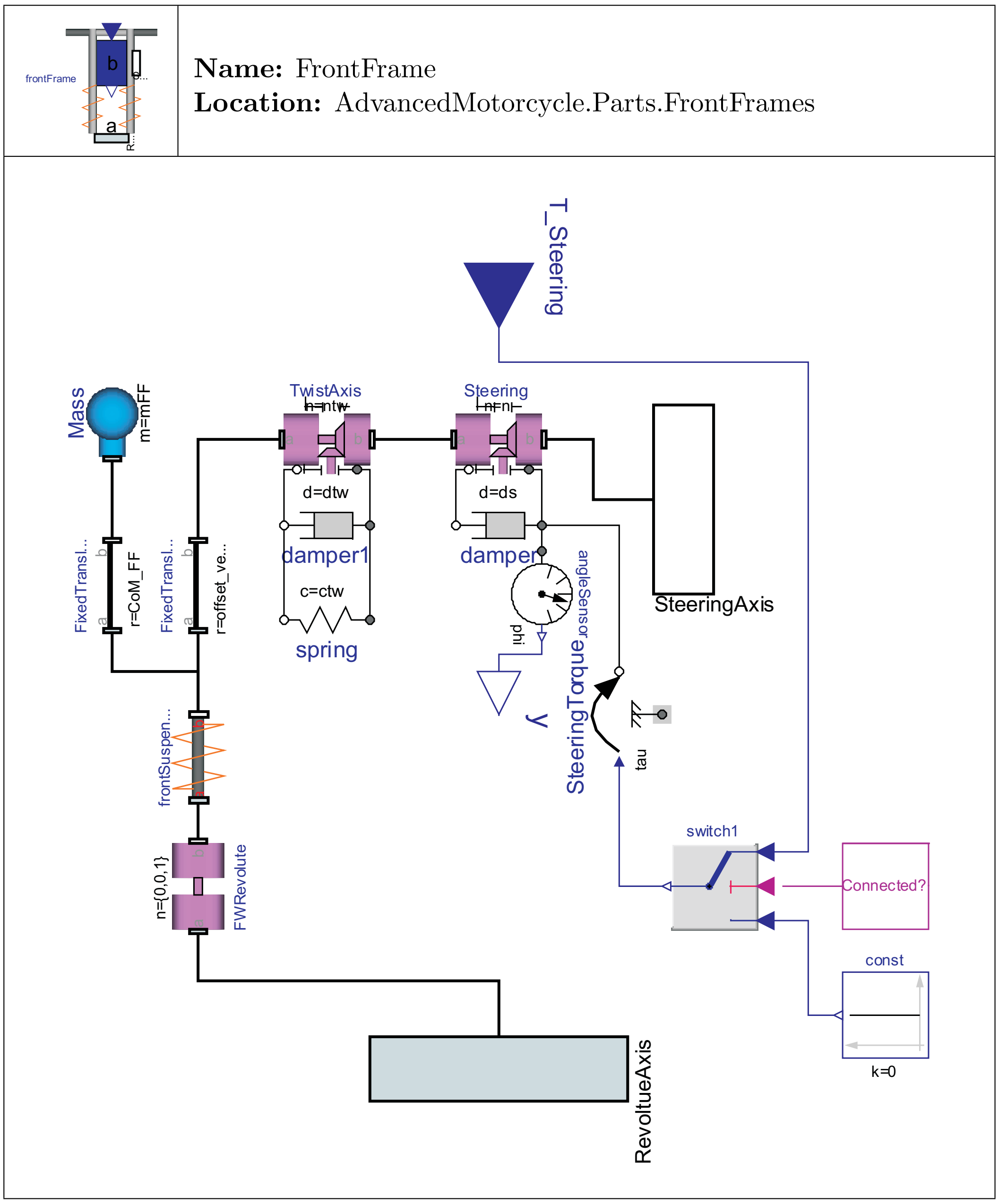
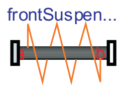

Name: FrontFrame
Location: AdvancedMotorcycle.Parts.FrontFrames

Figure 3.23: Wrapped model of the front frame



Name: FrontSuspension
Location: AdvancedMotorcycle.Parts.FrontFrames.
FrontSuspensions

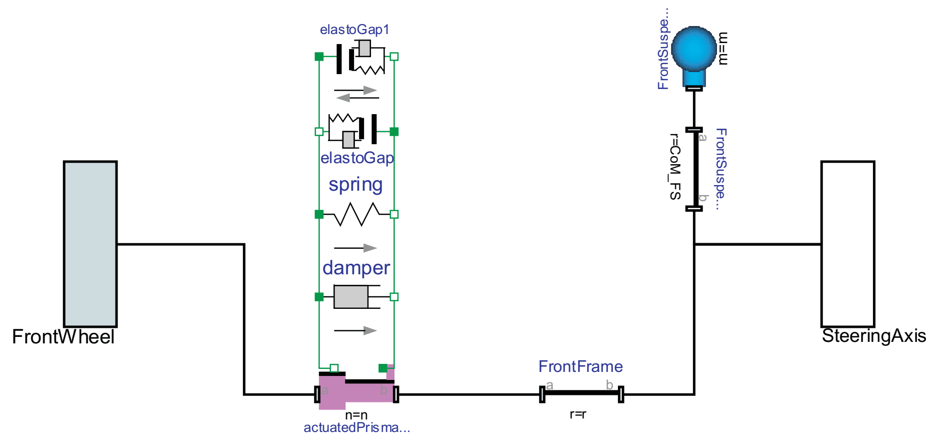


Figure 3.24: Model of a classic telescopic front suspension

provides two interfaces that are used to connect it to the front wheel and the twist axis.

The model of the upside down telescopic front suspension is illustrated in Figure 3.25

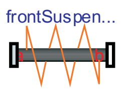
Contrary to the former two models, the next front suspension model depicted in Figure 3.26 implements characteristic spring/damper-elements.

The model of the characteristic spring is shown in Figure 3.27.

The model of the characteristic spring is a slightly modified version of the one in F. Cellier's *BondLib* [CN05]. The spring length is measured at the signal output of the two conversion elements (*translational to bond graph* and *bond graph to translational*). Both signals are fed into a sum block. Additionally, a constant source defines the unstretched spring length s_{rel0}. The total length is then fed into a table. The table includes the characteristic curve of the spring and thus returns the corresponding force ($F = f(x)$) for each length. In other words, the spring constant k is substituted with a characteristic function and thus $F = f(x)$ instead of $F = k \cdot x$. This force is finally fed into a modulated source. The *zero-junction* ensures that this is the only force acting on the spring.

The model of the characteristic damper is illustrated in Figure 3.28.

The linear damper model of the *BondLib* has no modulated source since it simply satisfies the linear equation $F = b \cdot v$ (where b is the so-called damper constant).

	**Name:** FrontSuspensionUSD **Location:** AdvancedMotorcycle.Parts.FrontFrames. FrontSuspensions

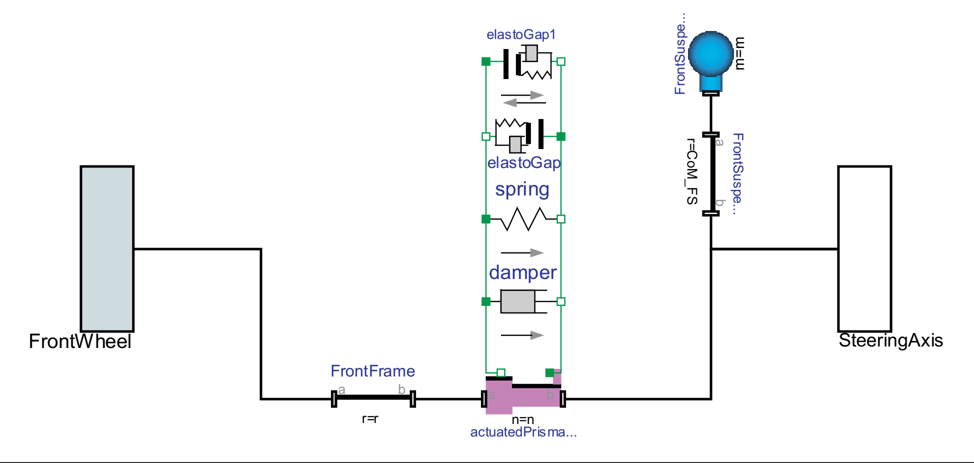
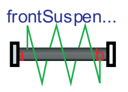


Figure 3.25: Model of an upside down telescopic front suspension

	**Name:** FrontSuspensionUSD_Ind **Location:** AdvancedMotorcycle.Parts.FrontFrames. FrontSuspensions

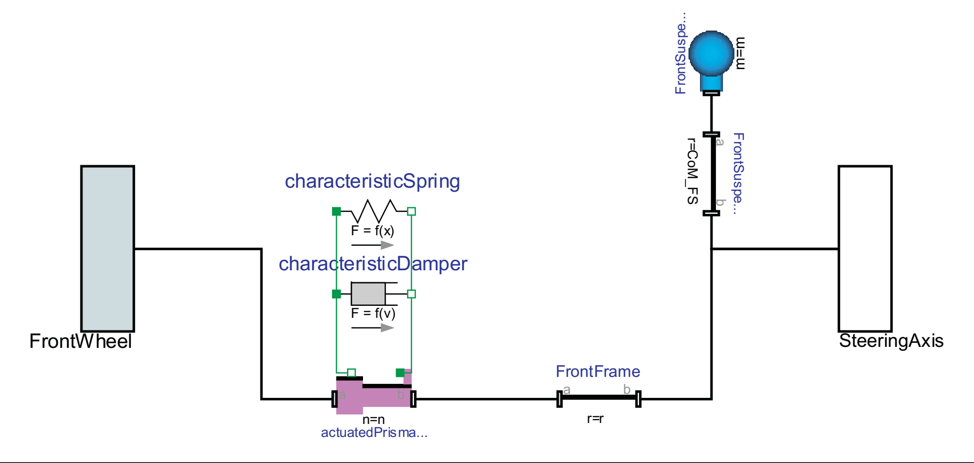
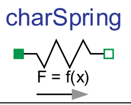


Figure 3.26: Model of a classic telescopic front suspension using characteristic spring/-damper-elements



Name: CharacteristicSpring
Location: AdvancedMotorcycle.Parts.Utilities

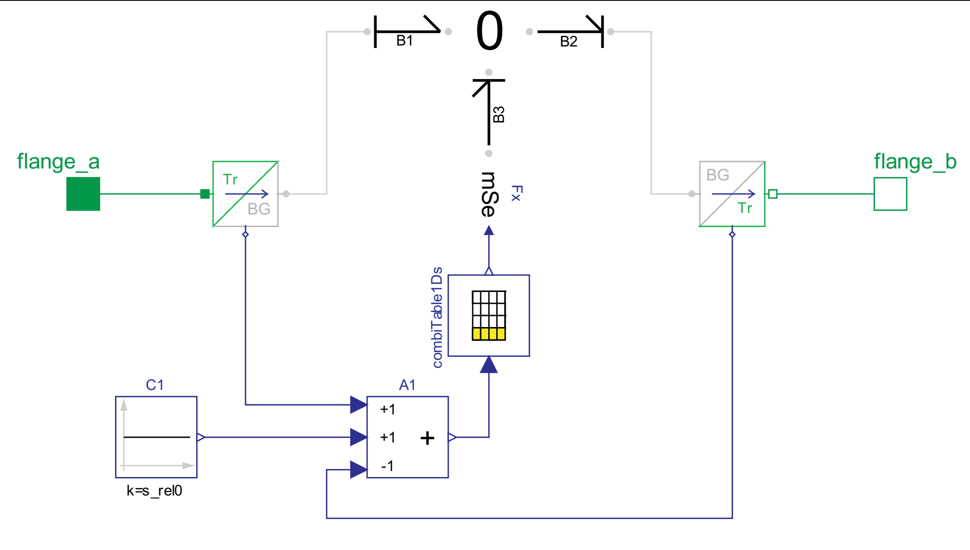
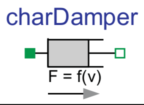


Figure 3.27: Model of a characteristic spring



Name: CharacteristicDamper
Location: AdvancedMotorcycle.Parts.Utilities

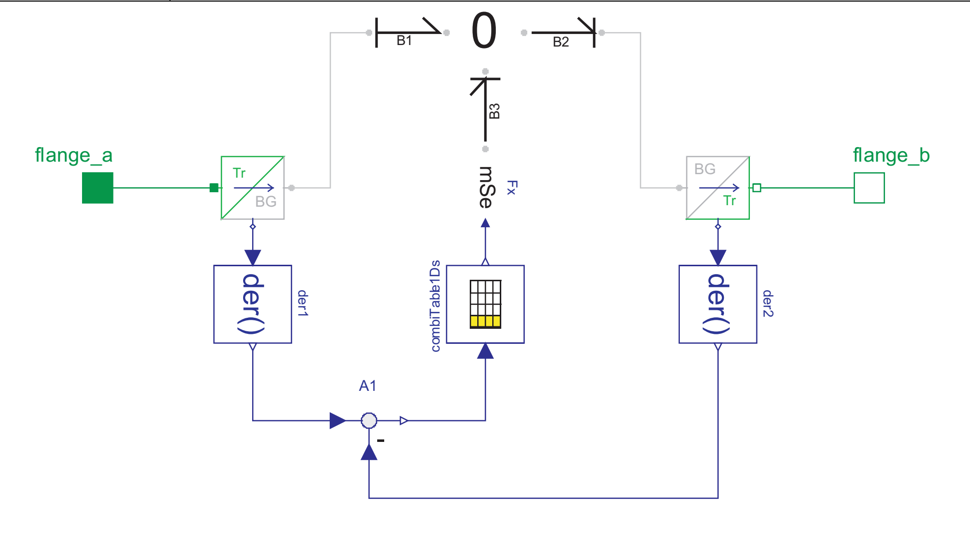


Figure 3.28: Model of a characteristic damper

Again, the length is provided at the outputs of the conversion elements. In order to determine the velocities, both signals are fed into a *derivative block*. The output of the downstream feedback block is the total velocity. This velocity is fed into a table which includes the characteristic curve of the damper element. Mathematically, this relation is described by the simple equation $F = f(v)$. Again, to ensure that just one source is acting on the element, a zero-junction is used.

Swinging Arm

Figure 3.29 depicts the model of a classic swinging arm.

Name: SwingingArm_classic
Location: AdvancedMotorcycle.Parts.SwingingArms

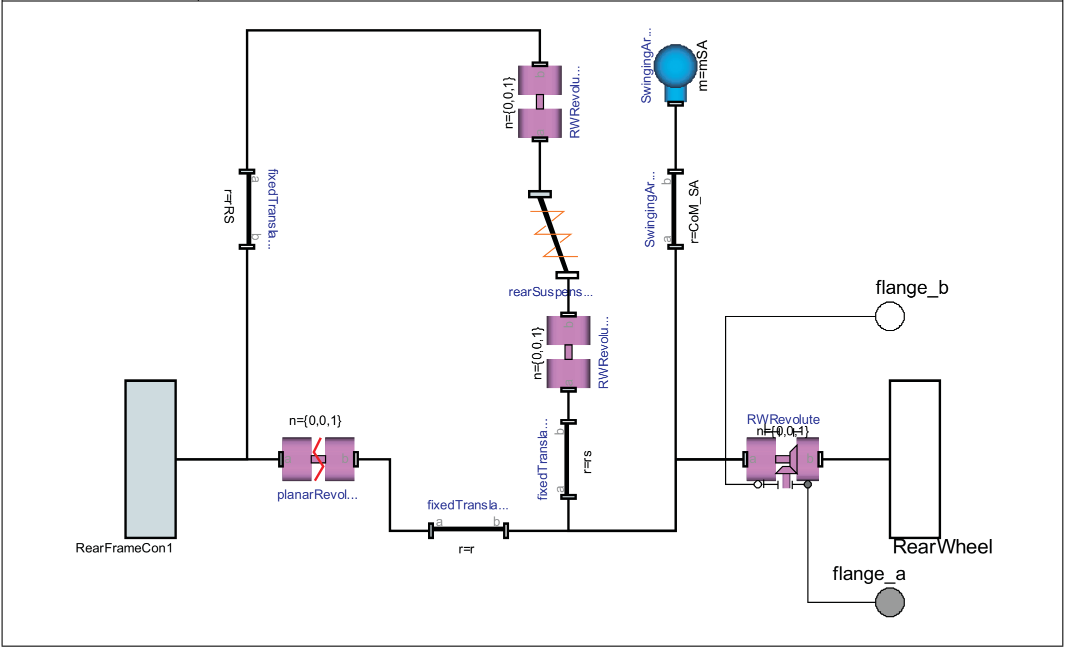
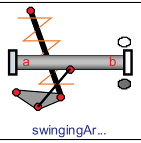


Figure 3.29: Wrapped model of a classic swinging arm

The swinging arm is hinged to the rear frame and the rear wheel. The center of mass position is modeled via a fixed translation element and a mass element. The rear suspension is connected to revolute joints. The model involves a closed kinematic loop. To overcome problems with kinematic loops, the *MultiBondLib* provides *revolute cut-joints*. By substituting the revolute joint with a revolute cut-joint, the model becomes solvable.



Name: SwingingArm_rocker
Location: AdvancedMotorcycle.Parts.SwingingArms

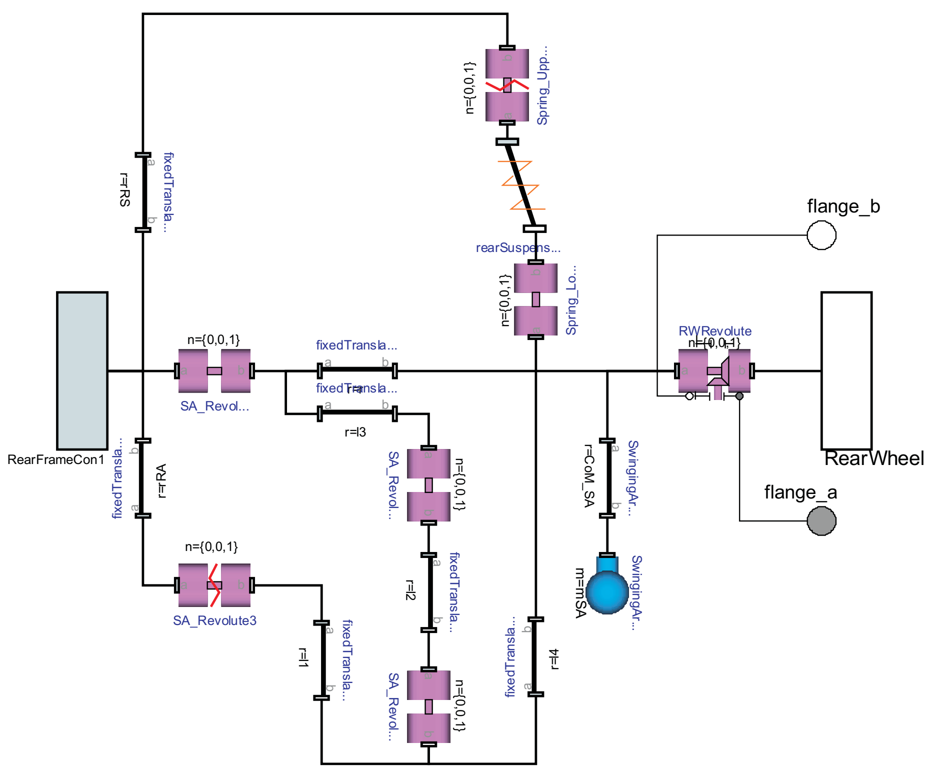


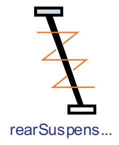
Figure 3.30: Wrapped model of a rocker chassis swinging arm (monoshock system)

In Figure 3.30 the model of a rocker chassis swinging arm is shown.

The rocker-chassis swinging arm (monoshock system) is a modern re-design of the former one (twin-shock system). One of the greatest advantages is that only one shock absorber (rear suspension) is used. This part is connected via a mechanical linkage to the rear frame and the swinging arm of the motorcycle. This model involves two closed kinematic loops and thus two revolute cut-joints are implemented in the model.

Rear Suspension (Massless)

Again, two different versions of rear suspensions are provided. The standard version (Figure 3.31) includes linear spring/damper elements and the second version (Figure 3.32) characteristic spring/damper-elements.



Name: RearSuspension_classic
Location: AdvancedMotorcycle.Parts.SwingingArms.
RearSuspensions

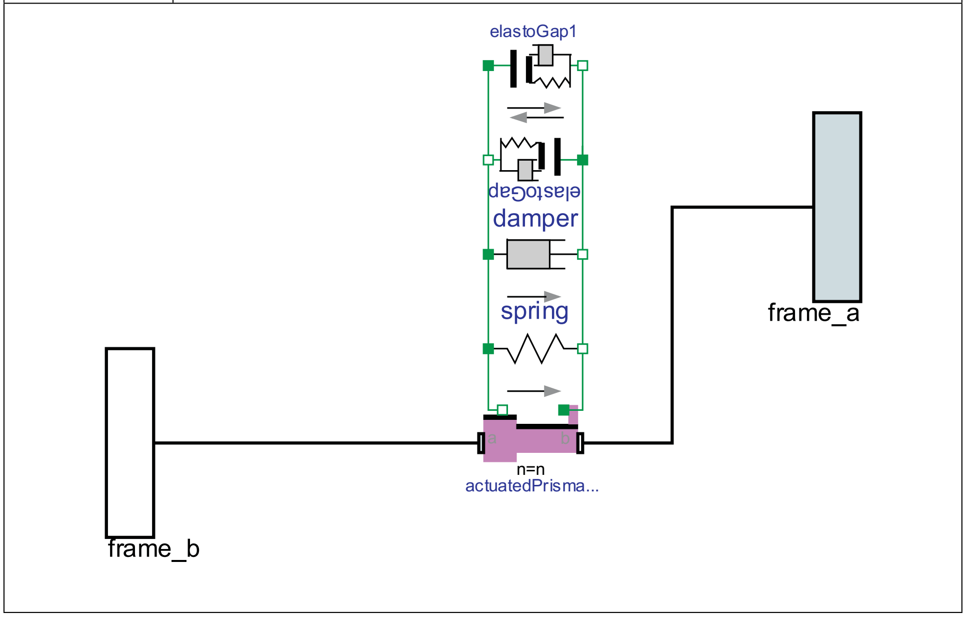


Figure 3.31: Model of the standard rear suspension

Rear Frame

The rear frame includes five interfaces to connect the following parts:

Name: RearSuspension_classic
Location: AdvancedMotorcycle.Parts.SwingingArms.
RearSuspensions

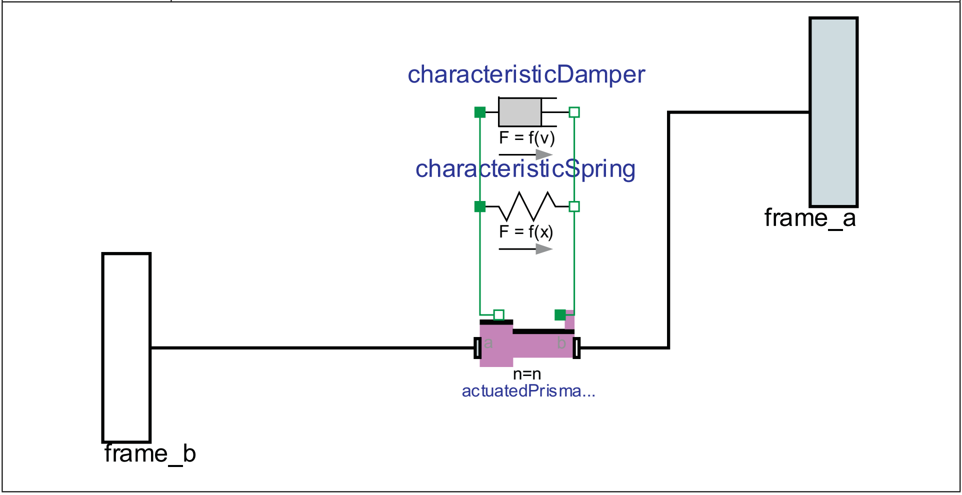


Figure 3.32: Model of the characteristic rear suspension

1. rear frame to swinging arm
2. rear frame to steering axis
3. rear frame to rider's upper body
4. center of pressure (CoP) to aerodynamic forces
5. center of mass (CoM) to external disturbances (e.g. side wind)

The wrapped model of the rear frame is depicted in Figure 3.33. In order to design a controller, the lean angle is measured. In addition, the lean angle and the lean rate can be selected as states. This is done with the boolean parameter `enforceStates`. The lean angle is calculated by the scalar product:

```
eAxis = transpose(RearFrameCoM.frame_a.P.R)*{0,0,1};
leanAngle = -arcsin({0,1,0}*eAxis);
phi = leanAngle;
leanRate = der(leanAngle);
```

Rider's Upper Body

The rider's upper body is hinged to the motorcycle's rear frame via an actuated revolute joint. The spring/damper-elements are used to model realistic rider behavior. To design

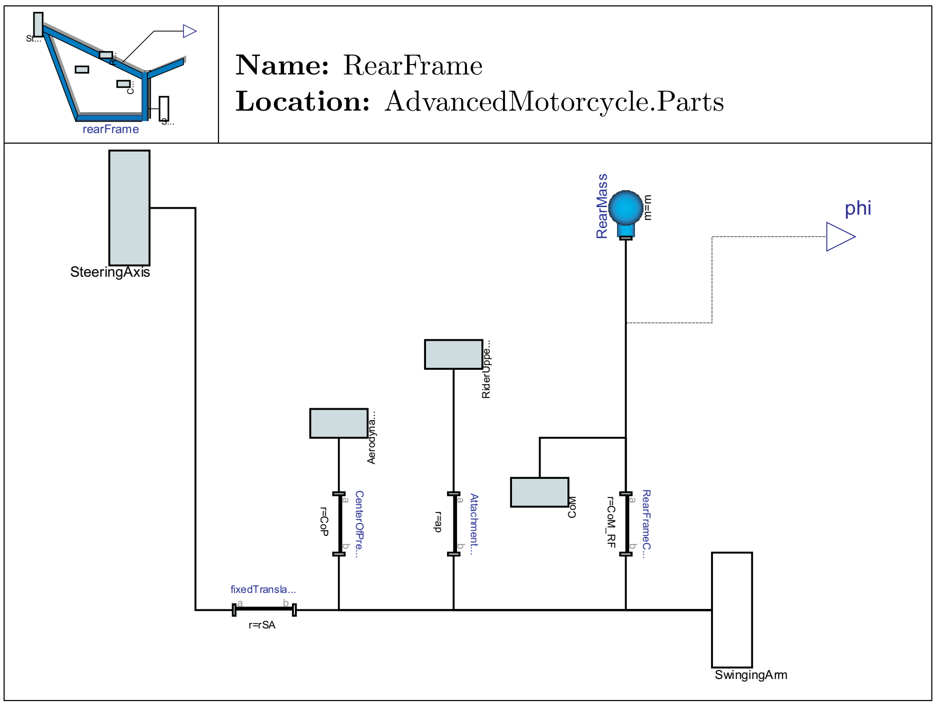

Name: RearFrame
Location: AdvancedMotorcycle.Parts

Figure 3.33: Model of the rear frame

a controller, a torque input `T_RdierLean` is applied. A sensor is needed to measure the lean angle of the rider's upper body relative to the rear frame (see Figure 3.34).

Elasto-Gap

The elasto-gap is depicted in Figure 3.35

This model is needed to ensure that the elongation of the suspensions is limited to a defined length. As long as the relative length of the spring s_{rel} (e.g. Figure 3.31) is smaller than s_{rel0}, the elasto-gap is inactive. That is, the output y of the *greater*-block is false. Consequently, the output of the switch is equal to u_3 and hence a pair of force sources with the same magnitude but opposite signs is applied in order to compensate the spring/damper forces. As soon as s_{rel} is greater than s_{rel0}, the elasto-gap becomes active. That is, the output of the *greater*-block is false which in turn sets the output of the switch equal to u_1. Thus, the force sources are inactive.

Visualization

The animation results of the SL2001 and the improved version of it are depicted in Figure 3.36 and 3.37 respectively. The models are again in combination with wheels

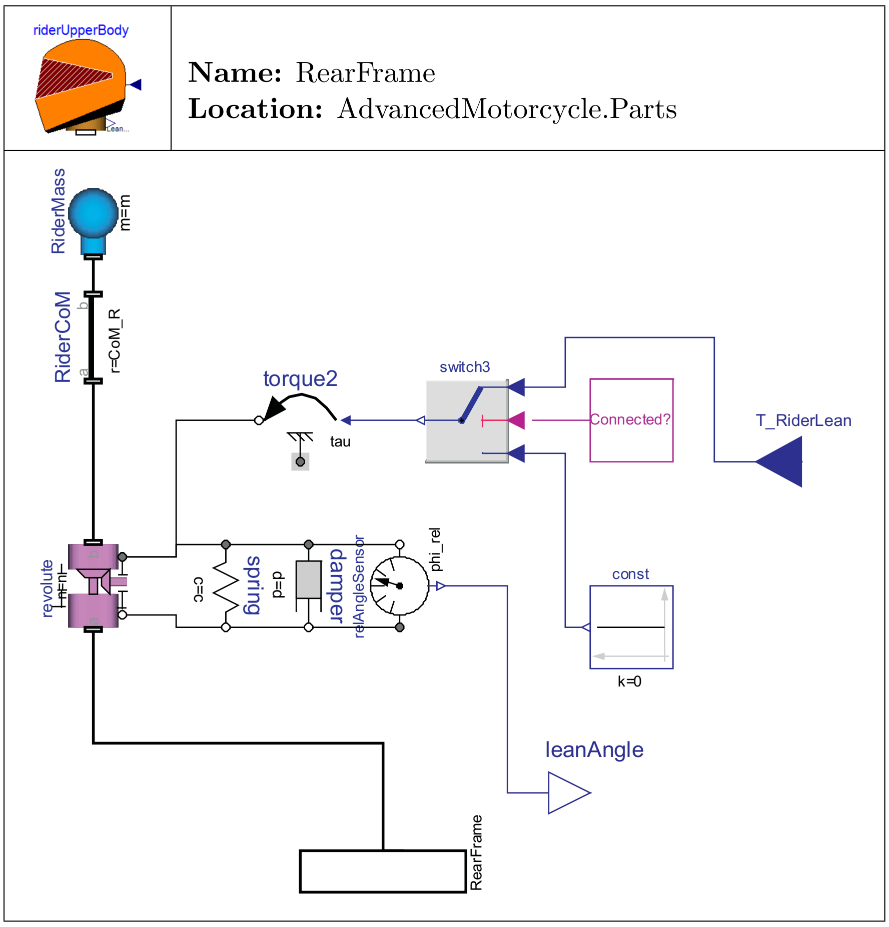
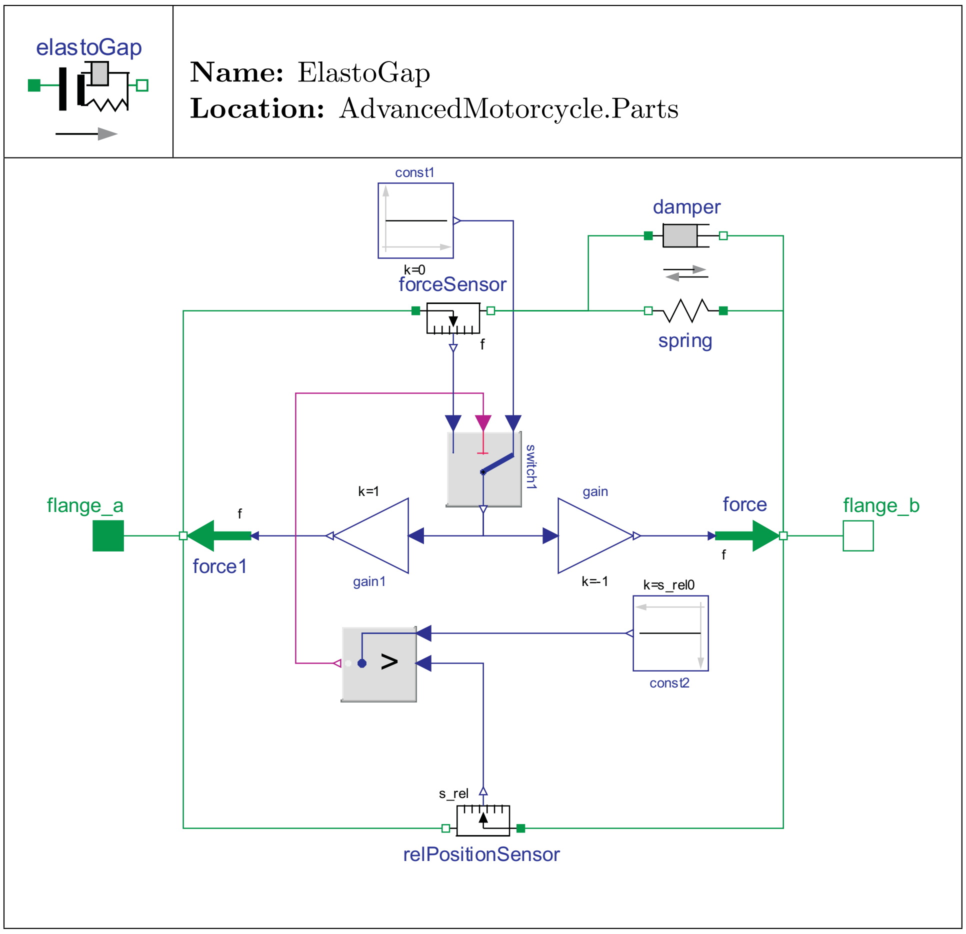

Name: RearFrame
Location: AdvancedMotorcycle.Parts

Figure 3.34: Model of rider's upper body



Name: ElastoGap
Location: AdvancedMotorcycle.Parts

Figure 3.35: Elasto-gap model

from the *WheelsAndTires* library.

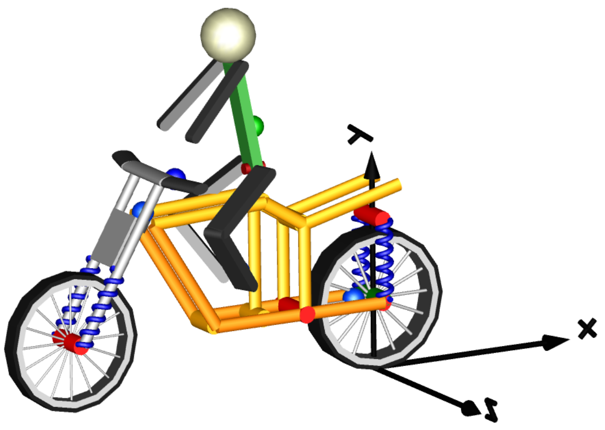


Figure 3.36: Animation result of the SL2001 motorcycle

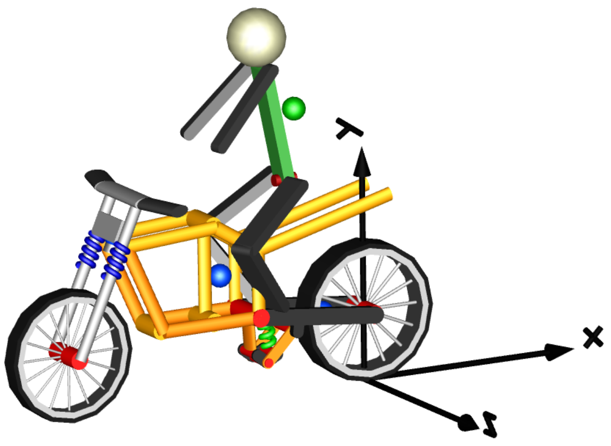


Figure 3.37: Animation of Sharp's improved motorcycle model

3.4.6 Aerodynamics

In order to obtain more realistic results, in-plane aerodynamics are included. These are the lift and drag force and a pitching moment. The lift force is described with the following equation:

$$F_{lift} = \frac{1}{2}\rho \cdot v^2 \cdot A \cdot C_L \tag{3.23}$$

Where ρ is the air density, v is the motorcycle's velocity, A is the frontal area of the front frame and C_L is the so-called lift coefficient.

The drag force is given by:

$$F_{drag} = \frac{1}{2}\rho \cdot v^2 \cdot A \cdot C_D \tag{3.24}$$

Where C_D is the drag coefficient. All the other variables are equal to those of Equation 3.23.

The pitching moment results in:

$$M_{pitch} = \frac{1}{2}\rho \cdot v^2 \cdot A \cdot C_P \cdot p \tag{3.25}$$

Where C_P is the so-called pitching coefficient and p is the wheelbase.

The lift and the drag force act on the center of pressure (CoP), whereas the pitching moment is acting on the center of mass (CoM) of the rear frame (see Figure 3.33). The lift force model is shown in Figure 3.38.

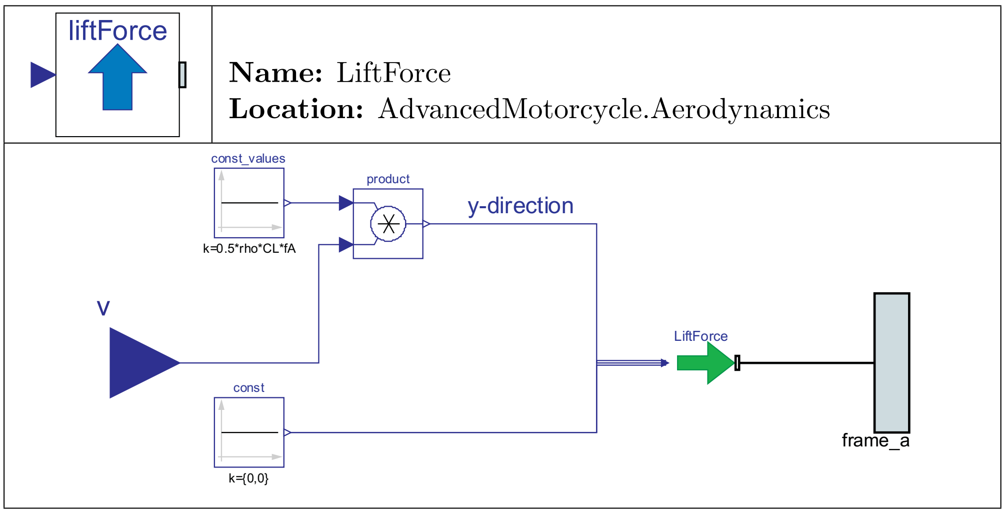
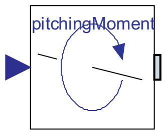


Figure 3.38: Lift force model

The lift force acts in the y direction. The model of the drag force is almost the same, except that the force points in the z direction. The pitching moment model is illustrated in Figure 3.39.

	**Name:** LiftForce **Location:** AdvancedMotorcycle.Aerodynamics

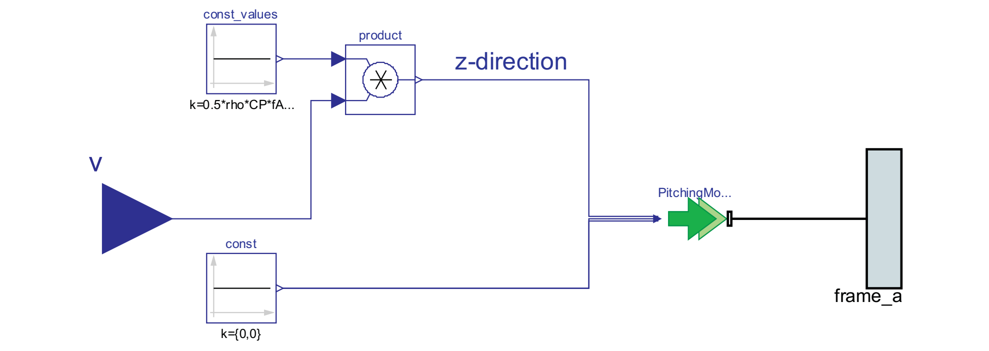
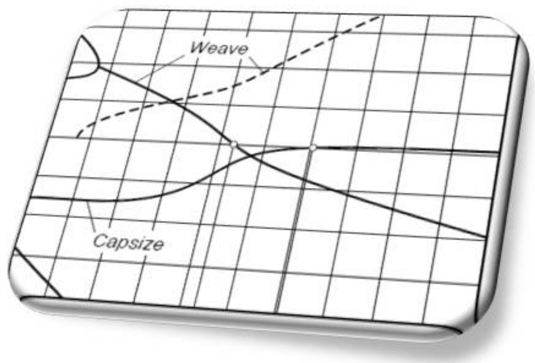


Figure 3.39: Pitching moment model

4 Eigenvalue Analysis



An eigenvalue analysis is performed in order to determine the self-stabilizing area of an uncontrolled bicycle or motorcycle. For this purpose, the state variables of the vehicle, which are responsible for stability, are of interest. In case of 3 d.o.f. models, these are steer angle δ, lean angle ϕ and their derivatives.

$$x = \begin{pmatrix} \delta \\ \dot{\delta} \\ \phi \\ \dot{\phi} \end{pmatrix} \tag{4.1}$$

In the case of vehicles with an additional d.o.f. allowing the rider's upper body to lean sideways, the state variables γ and $\dot{\gamma}$ are also taken into account.

All the other state variables (e.g. lateral- and longitudinal position) of the state vector described in Section 3.1.4 have no influence on the stability of single-track vehicles. Now, the four eigenvalues (one for each state variable) are calculated as a function of the vehicle's forward velocity $\lambda = f(v)$ (e.g. $v = 10ms^{-1}$ to $v = 50ms^{-1}$). Thus, for each specific velocity the model is linearized. The result of such an analysis are three

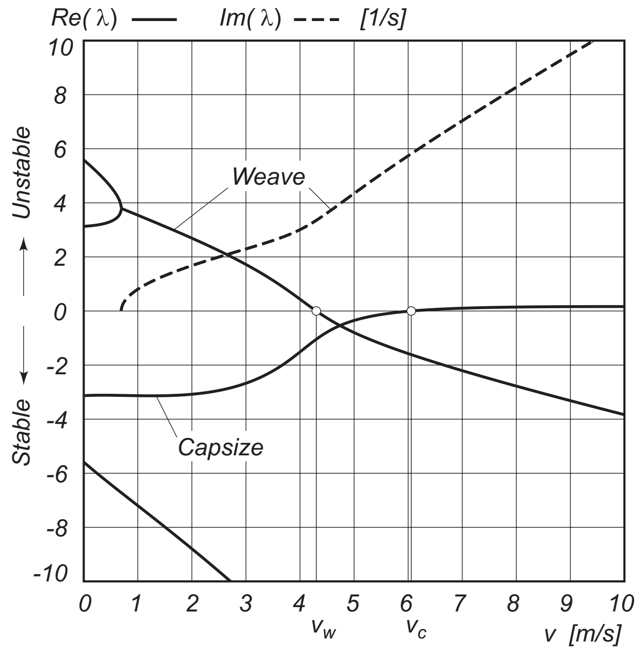


Figure 4.1: Stability analysis of an uncontrolled bicycle. The stable region is determined by eigenvalues with a negative real part. Here it is from 4.3 ms^{-1} to 6.0 ms^{-1}. Source: ([SMP05], p.31)

different forward velocities at which the motion of the vehicle changes qualitatively. Figure 4.1 depicts a typical result of such an analysis.

The first velocity is below the stable region. The second velocity is within and the third one above the stable region. In the following, the modes of single-track vehicles are explained.

4.1 Modes of Single-Track Vehicles

Figure 4.1 depicts the eigenvalues λ as a function of the velocity ($\lambda = f(v)$). Positive eigenvalues or more precisely eigenvalues with a positive real part correspond to unstable behavior while eigenvalues with a negative real part correspond to stable behavior. Eigenvalues including an imaginary part emphasize that the system is oscillating whereas eigenvalues without an imaginary part are non-oscillating. A stable region exists, if and only if all real parts of the eigenvalues are negative. In Figure 4.1 the stable region lies within the range $4.3ms^{-1} \leq v \leq 6ms^{-1}$.

4.1.1 Weave Mode

The weave mode begins at zero velocity. This mode is non-oscillating in the beginning and after a certain value passes over into a oscillating motion. In this mode the bicycle performs a tail motion, i.e. it sways about the normal direction (z-axis). The non-oscillating motion at very low speeds states that the bicycle is too slow to perform a tail motion and thus falls over like an uncontrolled inverted pendulum. As soon as it passes a certain value of approximately $v_w = 0.7ms^{-1}$, the real parts of the eigenvalues merge and two conjugate complex eigenvalues appear. Hence, a tail motion emerges. This motion is still unstable, but becomes stable as soon as the real eigenvalues cross the real axis. This happens at a weave speed of about $v_w = 4.3ms^{-1}$. For all velocities greater than v_w, this motion is stable.

4.1.2 Capsize Mode

The capsize mode is a non-oscillating motion, which corresponds to a real eigenvalue dominated by lean. As soon as the bicycle speed passes the upper limit of the stable region of about $v_c = 6ms^{-1}$, it falls over like a capsizing ship. However, above the stable region stable region, the bicycle is easy to stabilize although the real eigenvalues are positive. In the paper [SMP05] of Sharp et al. this motion is called “mildly unstable” as long as the absolute value of the eigenvalues is smaller than $2s^{-1}$.

4.1.3 Castering Mode

The castering mode is a non-oscillating mode, which corresponds to a real negative eigenvalue dominated by steer. In this mode the front wheel has the tendency to turn towards the direction of the traveling vehicle.

Apart from the modes explained above, some other modes exists. For instance, the so-called *Wobble mode* is an oscillating mode which affects just the front wheel and front frame. For a detailed description of all possible modes that emerge at single-track vehicles, refer to [Cos06] or [LS06].

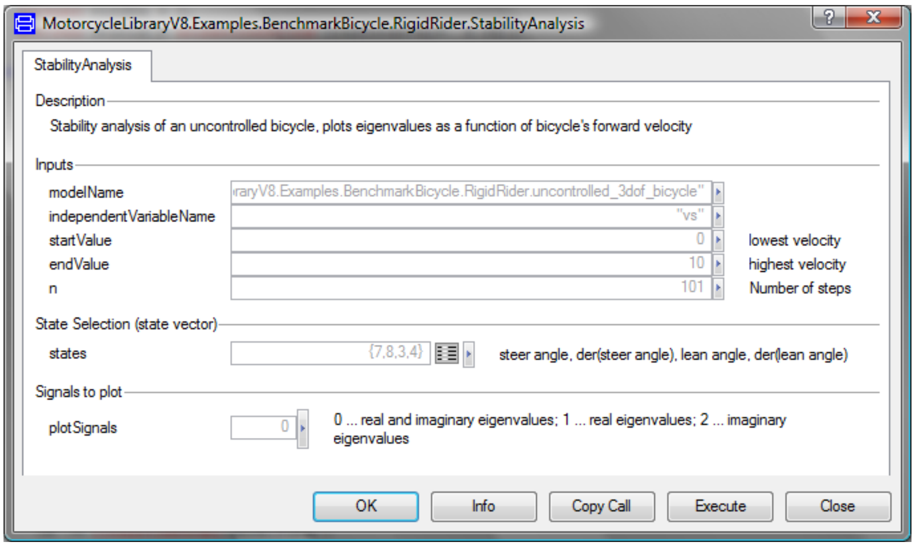
4.2 Linearization of the Models

A model of a single-track vehicle is linearized in order to perform a stability analysis. The output of the linearization is a linear state-space representation of the model. The linearization is done automatically using the Modelica `linearize` function. This function is part of the *linear systems* sub-package which is part of the *Modelica Standard Library*. With respect to this state-space model appropriate eigenvalues are calculated. Therefore, the reduced state vector introduced in Equation 4.1 is used.

4.3 An Eigenvalue Analysis for the Basic Bicycle and Motorcycle Models

4.3.1 A Function for the Eigenvalue Analysis

To be able to perform an eigenvalue analysis, specific functions for the basic bicycle and motorcycle models are provided. The code of such a function can be found in Appendix B.1. To execute this function one has to select it in the *package browser* of Dymola, click on the right mouse button and select *call function*. After executing the function, the following window appears (see Figure 4.2).


MotorcycleLibraryV8.Examples.BenchmarkBicycle.RigidRider.StabilityAnalysis

StabilityAnalysis

Description
Stability analysis of an uncontrolled bicycle, plots eigenvalues as a function of bicycle's forward velocity

Inputs

modelName	raryV8.Examples.BenchmarkBicycle.RigidRider.uncontrolled_3dof_bicycle"	
independentVariableName	"vs"	
startValue	0	lowest velocity
endValue	10	highest velocity
n	101	Number of steps

State Selection (state vector)

states	{7,8,3,4}	steer angle, der(steer angle), lean angle, der(lean angle)

Signals to plot

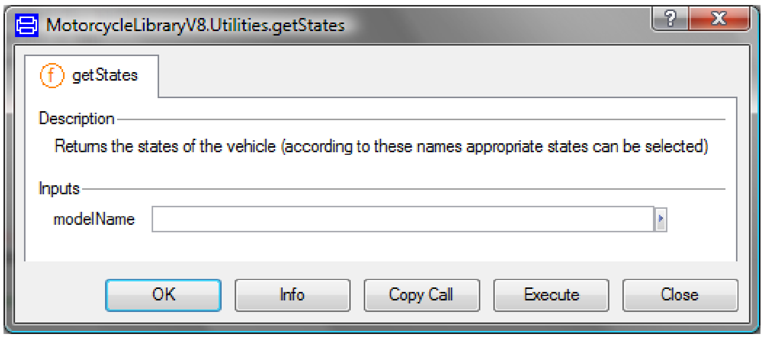
plotSignals	0	0 ... real and imaginary eigenvalues; 1 ... real eigenvalues; 2 ... imaginary eigenvalues

Figure 4.2: Parameter window of the eigenvalue analysis function

The first parameter are inputs. Firstly, the user has to enter the model name (e.g. MotorcycleLib.Examples.BenchmarkBicycle.RigidRider.uncontrolled_3dof_bicycle). This can be done by simply *drag and drop* the model into the input field. The input variable name which represents the sweep velocity is set to "vs" by default. This parameter is defined in the model in order to sweep the velocity. In the next two input fields, the start and end velocities are set. The last input value states how many steps are performed. The more steps are performed, the more accurate are the results. The next parameter is the state-vector. It includes the necessary states for the analysis.

Remark

If the states are not known, the function `getStates` (MotorcycleLib.Utilities) should be executed first (see Figure 4.3). After the user has entered the model name, the function



Description
Returns the states of the vehicle (according to these names appropriate states can be selected)
Inputs
modelName

Figure 4.3: Parameter window of the getStates function

returns the states of the model. The state names and the corresponding position in the state-vector are plotted in the command window. The code of the function can be found in Appendix B.3.

Finally, by means of the last input field, the signals to plot can be chosen.

4.3.2 Results

Basic Bicycle Models

In the following figures some typical results are shown. Figure 4.4 depicts the eigenvalues analysis of the basic bicycle which is based on Schwab's benchmark bicycle [SMP05].

Figure 4.5 illustrates the eigenvalue analysis of a 4 d.o.f. bicycle.

The result of the eigenvalue analysis presented by Schwab et al. [SKM08] is shown in Figure 4.6

In their recently published paper, they state that with reference to the movable rider, the weave speed changes from $v_w \approx 4.292ms^{-1}$ to $v_w \approx 4.533ms^{-1}$, whereas the capsize speed changes from $v_c \approx 6.024ms^{-1}$ to $v_w \approx 6.037ms^{-1}$. The stable region of the basic bicycle is almost the same ($v_w = 4.28ms^{-1}$, $v_c = 5.996ms^{-1}$). Contrary to the basic bicycle, the stable region of the 4 d.o.f bicycle is from $v_w = 4.497ms^{-1}$ to $v_c = 6.01ms^{-1}$. Although the values are a little bit less compared to Schwab's model, the difference is negligible.

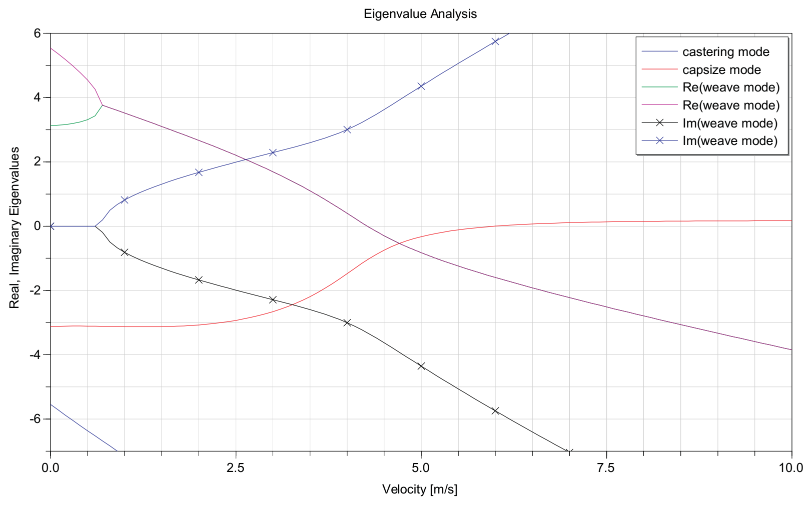


Figure 4.4: Stability analysis of the uncontrolled basic bicycle. The stable region is determined by eigenvalues with a negative real part. The results are identical to Schwab's result (see Figure 4.1)

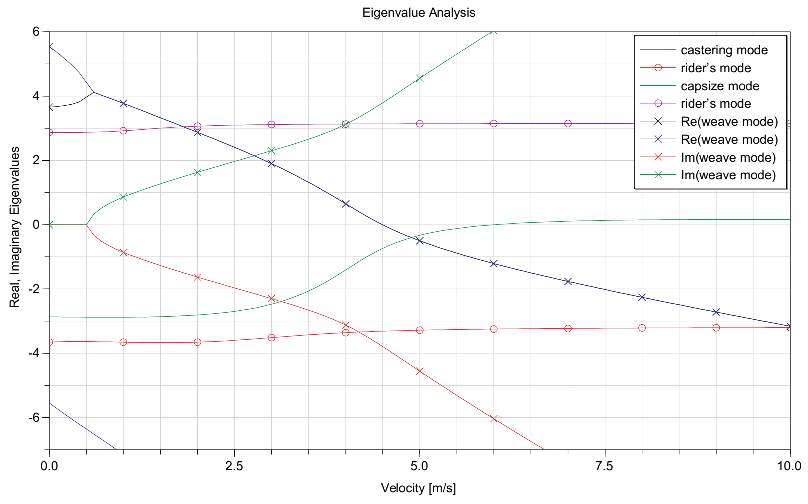


Figure 4.5: Stability analysis of the uncontrolled 4 d.o.f. bicycle

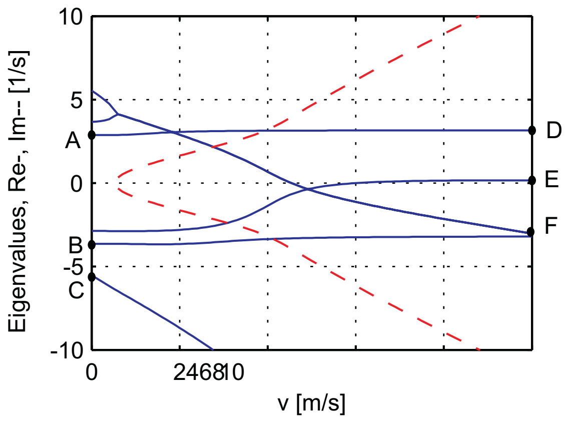


Figure 4.6: Stability analysis of the uncontrolled 4 d.o.f. bicycle introduced by Schwab et al. (Source: [SKM08], p.6)

Basic Motorcycle Model

The eigenvalue analysis for the basic motorcycle is depicted in Figure 4.7.

4.4 An Eigenvalue Analysis for the Advanced Motorcycle Models

4.4.1 A MATLAB Function for the Eigenvalue Analysis

In the following, an eigenvalue analysis for the SL2001 model in combination with the ideal wheels of the *MultiBondLib* [ZC06] is carried out by means of MATLAB. Due to a lack of time a Modelica function in order to perform an eigenvalue analysis is not provided.

After the model of the SL2001 motorcycle is simulated, it is linearized. In order to do this, one has to click on the simulation tab and choose the linearize command.

After the linearization, Dymola automatically creates a file called *dslin.mat*. This file includes all the necessary information of the linear state-space model. With the command `load dslin.mat` within the MATLAB environment, the file is loaded into the workspace. By typing “xuyName” into the command window, the names of the state variables appear. With the command `[A,B,C,D]=tloadlin('dslin.mat')` the file *dslin.mat* is loaded and the state-space model is stored into the matrices A, B, C

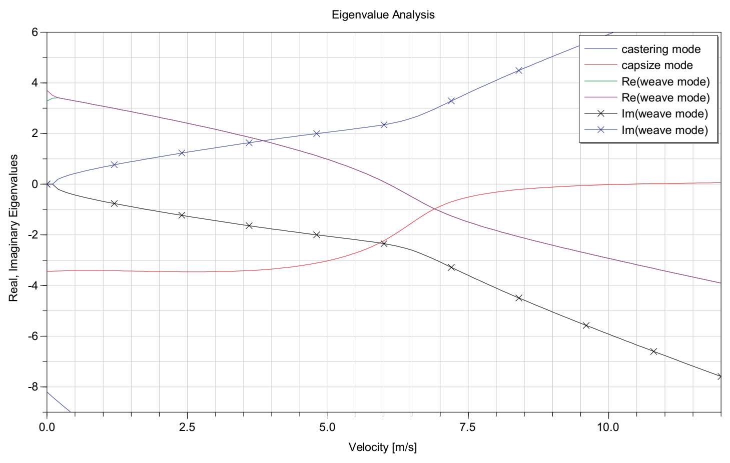


Figure 4.7: Stability analysis of the uncontrolled 3 d.o.f. motorcycle. The stable region is determined by eigenvalues with a negative real part. Here it is from 6.08 ms^{-1} to 10.32 ms^{-1}.

and D. Now, the reduced state vector is built. Therefore appropriate states have to be selected.

Compared to the 3 and 4 d.o.f. basic bicycle, and the basic motorcycle, the SL2001 motorcycle includes much more state variables which are responsible for the stability. In total, the vehicle includes 18 state variables, whereas 15 have an influence on the stability. In addition to the state variables mentioned so far, the following have to be taken into account: The states of the front frame's revolute joint which is connected to the front wheel, the states of the twist axis' revolute joint, the states of the swinging arm's revolute joint which is connected to the rear wheel, the states of the swinging arm's actuated prismatic joint and the angular rotation about the z-axis of the rear wheel.

In the following, the eigenvalues are calculated for a velocity range from $v = 0ms^{-1}$ to $v = 17ms^{-1}$. To this end, the SL2001 motorcycle model has to be linearized for the velocities $v = 0, 1, 2, ..., 17ms^{-1}$.

4.4.2 Results

The results are depicted in Figure 4.8.

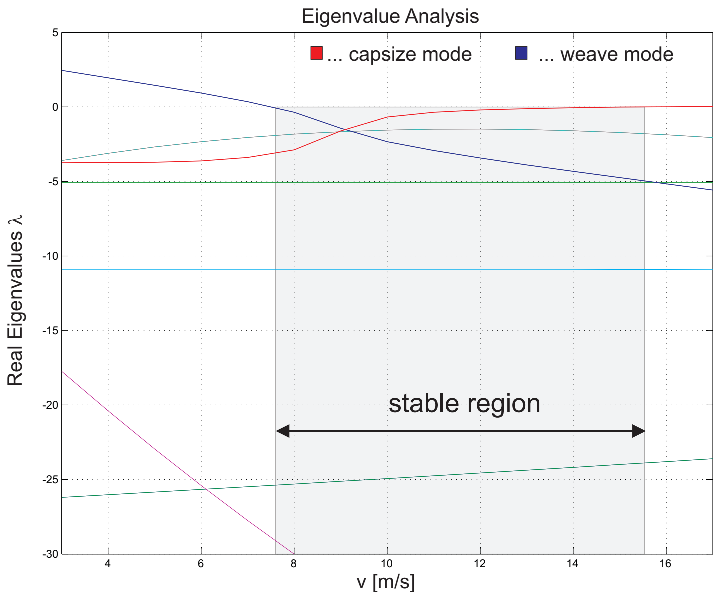
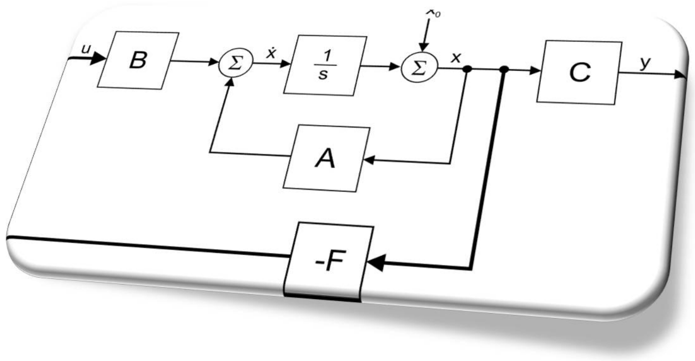


Figure 4.8: Stability analysis of the SL2001 motorcycle model. The self-stabilizing region lies within the range $7.5ms^{-1} \leq v \leq 15.75ms^{-1}$.

Remark

An eigenvalue analysis for Sharp's improved motorcycle model [SEL04] is carried out in the same manner.

5 Controller Design



In this chapter, the *controller design* is carried out with reference to several different approaches. The controllers are either developed for rigid or movable riders. In the beginning, three controllers valid for models composed of rigid riders are introduced. The first controller is built with classic elements (i.e. PI- or PID-elements). In order to design the controller, the lean angle is fed back. The output of the controller is the appropriate steering torque. The second controller is based on a state-space representation. Therefore, the lean angle and the lean rate are fed back to generate a suitable steering torque. The third controller is also based on a state-space representation. Additionally, the steer angle and the steer rate are taken into account. Hence, a controller design with reference to a preceding eigenvalue analysis is possible. The next controller is an extension of the former one. Again, based on a preceding eigenvalue analysis, a controller valid for a specific velocity range is developed. Finally, two alternative approaches to the former one are introduced.

The next controller is based on a state-space representation valid for models including a movable rider. To develop a controller for such a model, an additional torque is needed. This torque has to be applied by the upper body of the rider in order to balance the vehicle.

The state-space controller design itself is done with the pole placement technique. Until now, Modelica does not offer a pole placement function. Hence, an own developed pole placement function based on *Ackermann's formula* is presented. Therefore, it is possible to model single-input, single-output (SISO) systems. Unfortunately, a pole placement function for multiple-input, multiple-output systems is still missing. The design of controllers related to models composed of rigid riders is done by means of the basic motorcycle model, whereas the design for models composed of movable riders is done with the 4 d.o.f. bicycle model.

Furthermore, two approaches based on a linear quadratic regulator (LQR) are introduced. The first one is carried out with reference to a rigid rider, whereas the second one is valid for a movable rider. In order to change the velocity during the simulation (e.g. tracking a velocity profile), a simple velocity controller is introduced at the end of this chapter.

5.1 Classic Controller

In the simplest case the stability controller of a single-track vehicle is designed by means of a PID-controller. Therefore the lean angle of the rear frame is measured and fed back in order to generate an appropriate steering torque. Figure 5.1 depicts the basic structure of a controller/plant configuration.

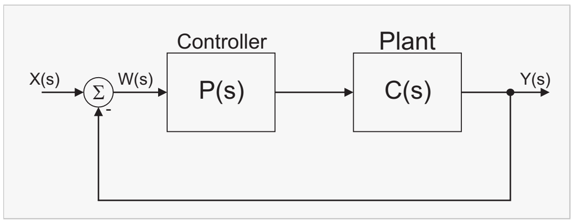


Figure 5.1: Block diagram of a controller/plant configuration

The corresponding model of the basic motorcycle is depicted in Figure 5.2.

The controller coefficients are obtained by trial and error. Sharp used the same controller structure for several of his motorcycles (refer to [SL01] and [SEL04]). An example of the controlled basic motorcycle can be found in the examples sub-package of the library.

5.2 State-Space Controller

In this section, several different state-space controllers are introduced. In order to design such a controller, either the lean angle and the lean rate or the lean angle, the lean rate plus the steer angle and the steer rate are fed back. The former one is a

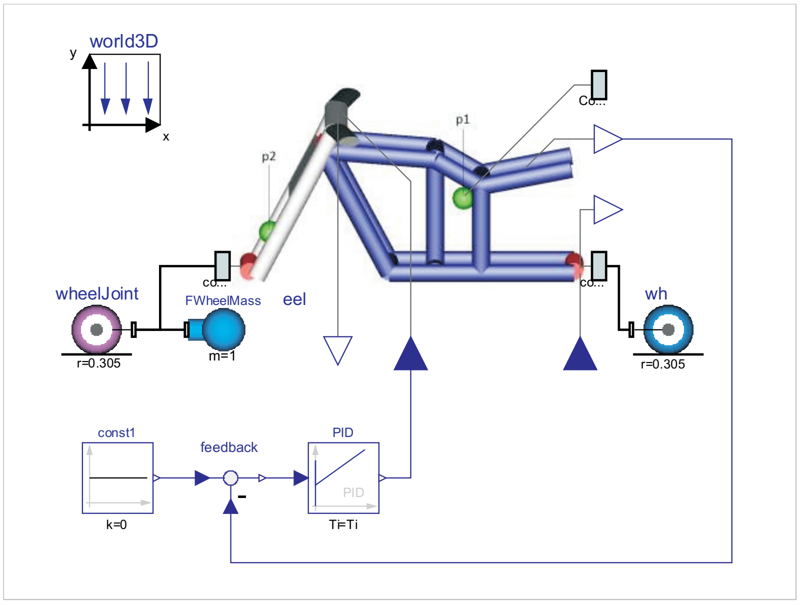


Figure 5.2: Controlled motorcycle model. The lean angle is fed back into a PID controller which returns a steering torque.

simple approach which will be explained by means of MATLAB. The latter one is based on a preceding eigenvalue analysis. The result of this analysis for the uncontrolled motorcycle is modified in such a way that the real parts of the eigenvalues for the controlled (closed-loop) system are all located in the left-half plane. Therefore, a pole placement function, which is based on *Ackermann's formula*, was developed. This function calculates the coefficients of the controller (i.e. the coefficients of the state feedback matrix). Therefore an offset, in order to shift the poles towards the left-half plane, has to be entered. Until now, all of the introduced controllers are just valid for one specific velocity. Thus a third controller valid for a specific velocity range is developed.
Finally, two individual approaches are introduced. Instead of defining an offset for the whole region, each region is controlled individually. This means that each region is controlled by appropriate control laws.

5.2.1 Short Introduction to State-Space Design

In general, the state-space representation of a linear system is given by:

$$\dot{x} = A \cdot x + B \cdot u, \qquad x(0) = x_0 \tag{5.1}$$

$$y = C \cdot x + D \cdot u \tag{5.2}$$

where x is a $(n \times 1)$ state vector, y is an $(m \times 1)$ output vector, A is referred to as system matrix with a dimension of $(n \times n)$, B is an $(n \times r)$ input matrix, C is an $(m \times n)$ output matrix and D the "feedthrough" matrix with a dimension of $(m \times r)$. Usually D is set to zero, except if the output directly depends on the input. The state vector x at time $t = 0$ includes the initial conditions, sometimes referred to as initial disturbances x_0. In a state-space representation, a system of n^{th} order is described with n differential equations of 1^{st} order. For a detailed introduction, refer to [Föl08].

The block diagram of a system described in state-space is illustrated in Figure 5.3

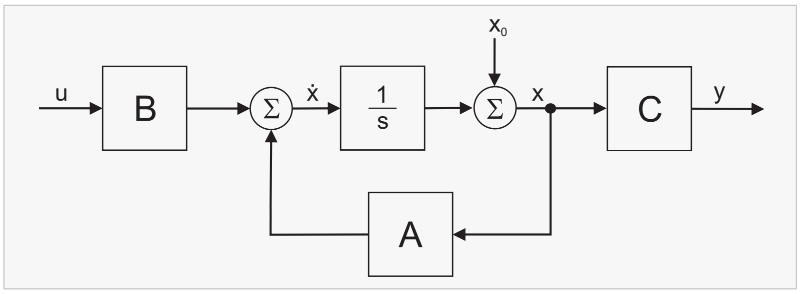


Figure 5.3: Block diagram of a system described in state-space

One major advantage of state-space control compared to classic control is that each state of the system can be controlled. In order to control the system, the state vector x is fed back. The state feedback control law for a linear time-invariant system is given by:

$$u(t) = -F \cdot x(t) \tag{5.3}$$

where F is a constant matrix.

By substituting u of Equation 5.1 with Equation 5.3, the state equation results in

$$\dot{x} = A \cdot x - B \cdot F \cdot x \tag{5.4}$$

or

$$\dot{x} = (A - B \cdot F)x \tag{5.5}$$

The block diagram of the equation above is shown in Figure 5.4.

The elements of the state feedback matrix F have to be chosen in such a way that the *initial disturbances*[1] $x_0(t)$ for $t \to +\infty$ converge towards zero

$$\lim_{t \to \infty} x(t) = 0 \tag{5.6}$$

[1]In order to compensate arbitrary disturbances which appear spontaneously, an additional methodology is needed (refer to Föllinger [Föl08] chapter 13.8). This is due to the fact that the state-space representation originates from the theory of differential equations.

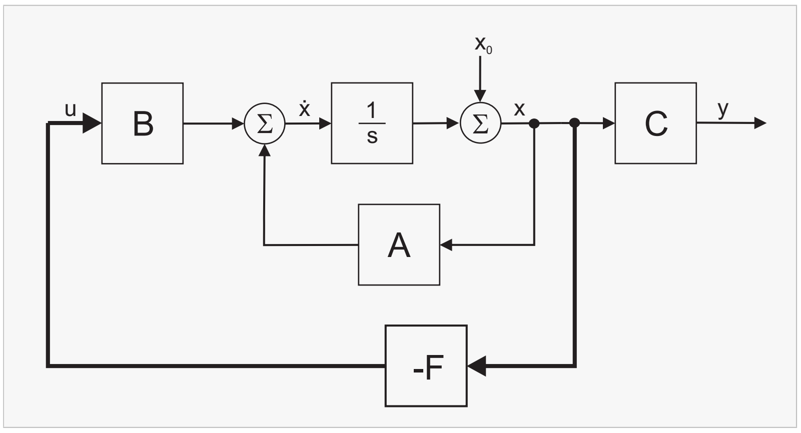


Figure 5.4: State feedback

and that the system becomes stable.

Remark

Concerning the state variables, the state feedback matrix is equivalent to a proportional controller. A D portion (D controller) is directly or indirectly obtained from the plant. Directly means that both a variable and its derivative are states of the system.

The main task of the state feedback control is to find appropriate coefficients for the state feedback matrix F in order to achieve the desired dynamic behavior of the system. One method which fulfills all the requirements is the so-called pole placement technique (refer to [Föl08] or [Unb07]). Figure 5.5 illustrates the graphical interpretation.

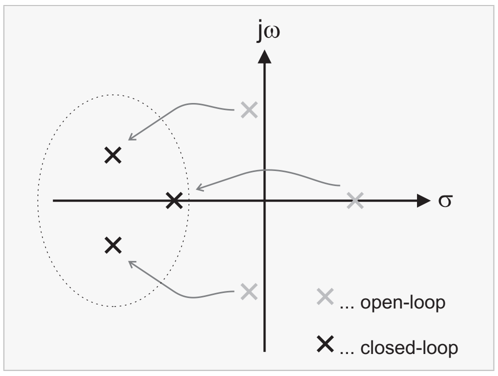


Figure 5.5: Graphical interpretation of the pole placement technique. Eigenvalues (poles) of the system located in the left-half plane correspond to a stable behavior.

5.2.2 Lean Angle Control

In the following, a simple state-space controller is developed. As already mentioned in the beginning, the lean angle ϕ and the lean rate $\dot{\phi}$ are fed back in order to generate an appropriate steering torque. Concerning the block diagram of the state feedback controller in Figure 5.4, the block diagram valid for this task looks like (see Figure 5.6):

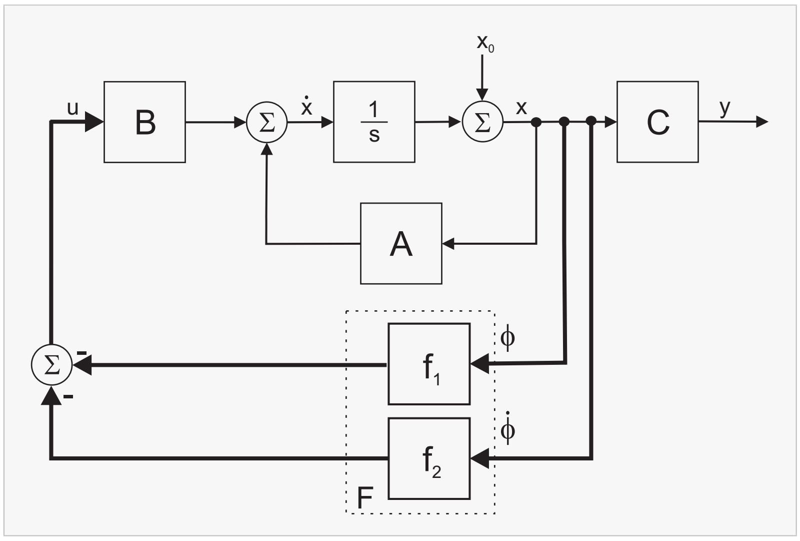


Figure 5.6: Simple state-space controller for the basic motorcycle model, where ϕ and $\dot{\phi}$ are two states of the state vector introduced in Chapter 3.1.4.

The corresponding state feedback control law results in:

$$u = -\begin{pmatrix} f_1 & f_2 \end{pmatrix} \cdot \begin{pmatrix} \phi \\ \dot{\phi} \end{pmatrix} \tag{5.7}$$

In order to design such a controller, the basic motorcycle model is linearized. The output of the linearization is a linear state-space representation of the model. The controller design for the linearized model is carried out with MATLAB. The model to be linearized is depicted in Figure 5.7.

The additional input in Figure 5.7 is used to notify Dymola that an external input is used. For the following controller design, the motorcycle has an initial velocity of $8ms^{-1}$ and an initial lean angle ϕ of 5°. After the model is simulated, it is linearized. In order to do this, one has to click on the simulation tab and choose the linearize command.

After the linearization, the file *dslin.mat* is generated (refer to Chapter 4.4.1). Again, in order to reduce the state vector, appropriate states have to be selected. These are

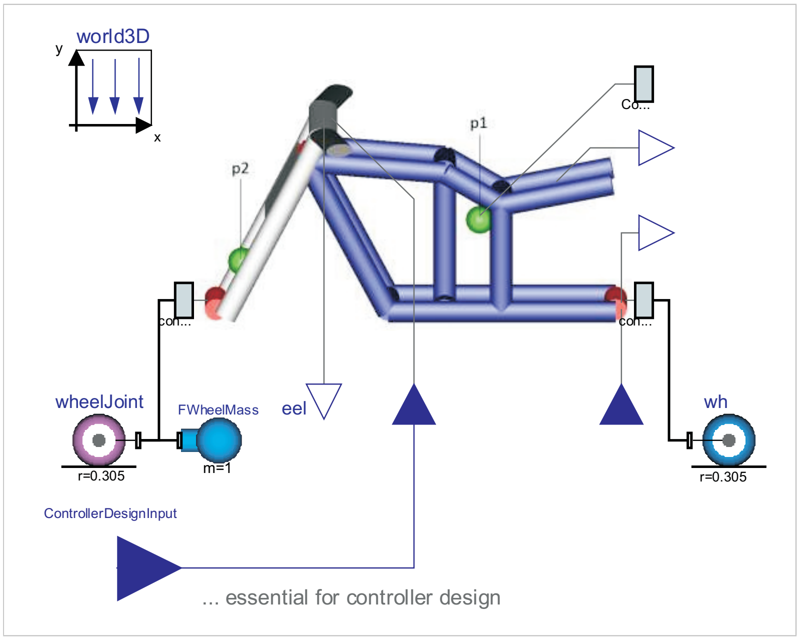


Figure 5.7: Basic motorcycle model used to design a controller

the lean angle ϕ and the lean rate $\dot{\phi}$. The following *lines of code* are used in order to build the reduced state-space model for the controller design:

```
states = [1, 2];    % 1 ... lean angle, 2 ... lean rate

Arel = A(states, states);
Brel = B(states,:);
```

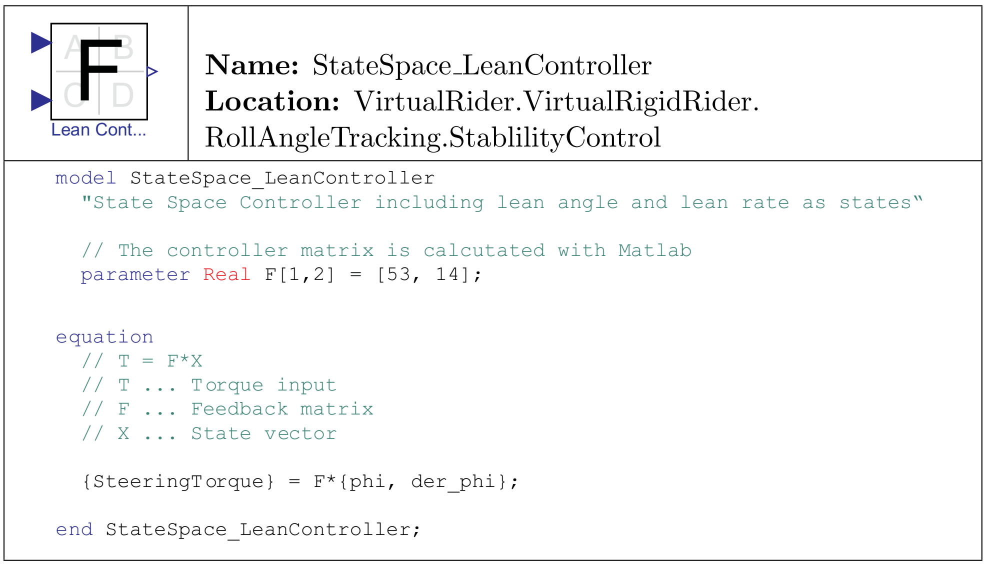
Now, the poles of the system are computed. This is done via the `p = eig(Arel)` command. The poles of the linearized motorcycle model for a speed of $8ms^{-1}$ are p = (2.9180, -3.7585). In the next step, the desired location of the poles is determined. Let us say that the desired location of the poles is at $p1 = -3$ and $p2 = -5$. In order to do this, MATLAB provides the `place()`-function. The output of this function is the desired state feedback matrix F.

```
F = place(Arel, Brel, [p1, p2])
```

For the basic motorcycle model with a forward velocity of $v = 8ms^{-1}$, the coefficients of the state feedback matrix are:

```
F = [624.6933, 172.2334]
```

The MATLAB *m-file* which was used in this section can be found in Appendix C.1. The corresponding Modelica model of the controller is shown in Figure 5.8



Name: StateSpace_LeanController
Location: VirtualRider.VirtualRigidRider.
RollAngleTracking.StablilityControl

```
model StateSpace_LeanController
  "State Space Controller including lean angle and lean rate as states"

  // The controller matrix is calcutated with Matlab
  parameter Real F[1,2] = [53, 14];


equation
  // T = F*X
  // T ... Torque input
  // F ... Feedback matrix
  // X ... State vector

  {SteeringTorque} = F*{phi, der_phi};

end StateSpace_LeanController;
```

Figure 5.8: Model of a simple state-space controller. The inputs (blue) are the lean angle and lean rate, the output (white) is the steering torque

5.2.3 State-Space Controller Based on a Preceding Eigenvalue Analysis

By means of a preceding eigenvalue analysis, the controller is designed in a much more physical fashion. In the following approach, the controller design is carried out for a specific velocity. So far, a state-space controller based on two inputs was developed. Both inputs are states which are responsible for the motorcycle's stability. But there are still two more states which affect the motorcycle's stability as well. These are the steer angle δ and the steer rate $\dot{\delta}$. All the other states of the motorcycle do not affect the stability and are thus useless for the controller design.

A major deficiency of the former controller is the fact that the poles have no physical interpretation. The coefficients of the state feedback matrix F were calculated for a motorcycle velocity of $8ms^{-1}$. With respect to the eigenvalue analysis of the uncontrolled motorcycle (see Figure 4.7), it is clearly within the stable range. Thus, both poles should be negative but one of them is positive although the vehicle is self-stabilizing. Hence, the interpretation of the poles can never be a physical one. On the contrary, if all states which are responsible for the stability are taken into account, the physical meaning of the poles is definitely given. More precisely, the state variables of the vehicle are exactly the one introduced in Equation 4.1. As stated before, it is now possible to design a controller based on a preceding eigenvalue analysis. In the following, three different approaches are introduced.

The block diagram of the state-space controller based on the eigenvalue analysis is depicted in Figure 5.9.

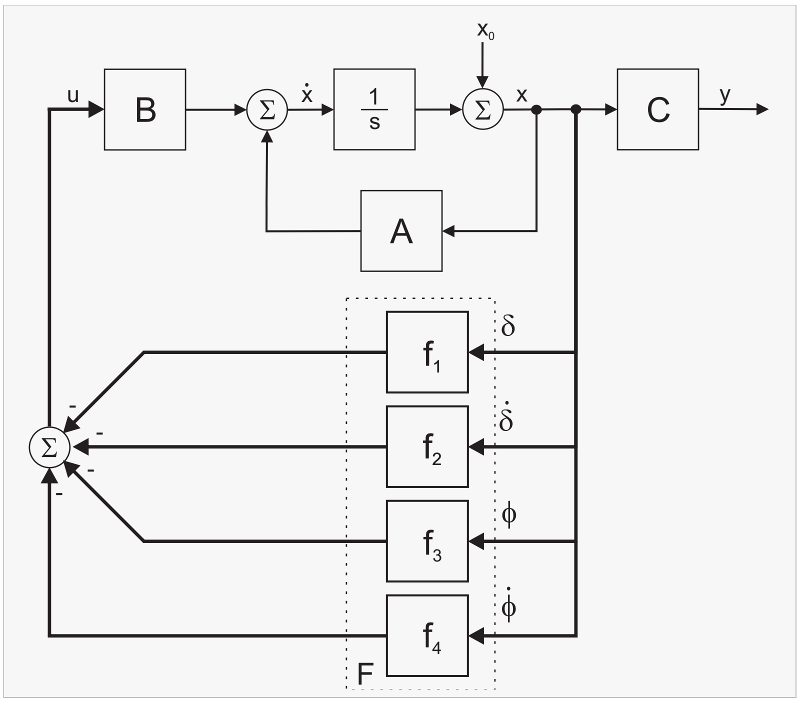


Figure 5.9: State-space controller for the basic motorcycle model

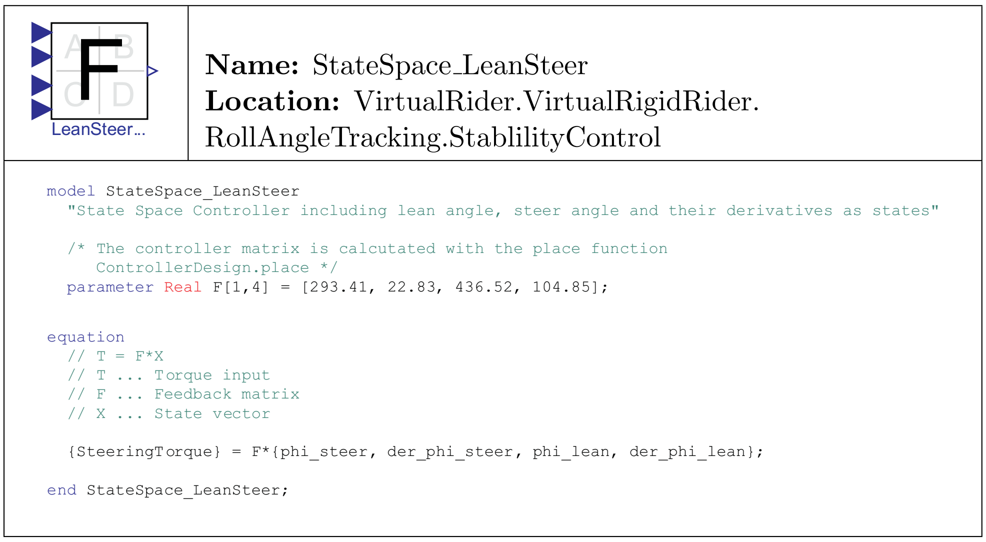
The state feedback control law is given by:

$$u = -\begin{pmatrix} f_1 & f_2 & f_3 & f_4 \end{pmatrix} \cdot \begin{pmatrix} \delta \\ \dot{\delta} \\ \phi \\ \dot{\phi} \end{pmatrix} \tag{5.8}$$

The corresponding Modelica model is shown in Figure 5.10.

Approach: Define an Offset in Order to Shift the Poles

In the following, all poles are shifted by the same value into the left-half plane. Therefore an offset is defined. Let us say that the poles of the motorcycle for an arbitrary velocity are $poles = [p_1, p_2, p_3, p_4]$. Now, all poles are shifted by means of an offset d. Thus, the desired pole locations are: $p_{desired} = [p_1 - d, p_2 - d, p_3 - d, p_4 - d]$. Figure 5.11 depicts the graphical interpretation.



Name: StateSpace_LeanSteer
Location: VirtualRider.VirtualRigidRider.
RollAngleTracking.StablilityControl

```
model StateSpace_LeanSteer
  "State Space Controller including lean angle, steer angle and their derivatives as states"

  /* The controller matrix is calcutated with the place function
     ControllerDesign.place */
  parameter Real F[1,4] = [293.41, 22.83, 436.52, 104.85];


equation
  // T = F*X
  // T ... Torque input
  // F ... Feedback matrix
  // X ... State vector

  {SteeringTorque} = F*{phi_steer, der_phi_steer, phi_lean, der_phi_lean};

end StateSpace_LeanSteer;
```

Figure 5.10: Wrapped model of a state-space controller. The inputs (blue) are the steer angle, lean angle and their derivatives, the output (white) is the steering torque

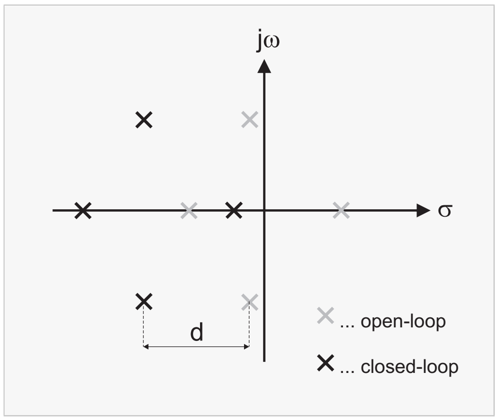


Figure 5.11: Graphical interpretation of the offset d. All poles are shifted by the same value into the left-half plane

The former introduced lean angle controller was designed with MATLAB. Now, a Modelica pole placement function based on *Ackermann's formula* is introduced.

$$F = t_n \cdot S_n^{-1} \cdot \phi(A) \tag{5.9}$$

where

$$t_n = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix}$$

is a vector of length n which is used to store the last row of the inverted controllability matrix

$$S_n = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$

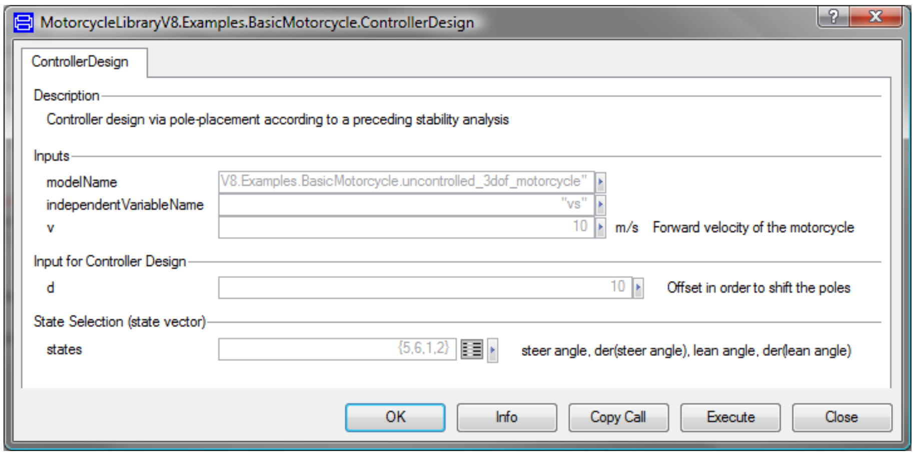
and

$$\phi(A) = A^n + a_1 \cdot A^{n-1} + a_2 \cdot A^{n-2} + \cdots + a_{n-1} \cdot A + a_n \cdot \mathbb{1}$$

is the characteristic polynomial of the system matrix A. The coefficients a_m are chosen in such a way that λ_1, λ_2, ..., λ_n are the eigenvalues (poles) of the polynomial

$$(s - \lambda_1) \cdot (s - \lambda_2) \cdots (s - \lambda_n) = s^n + a_1 \cdot s^{n-1} + \cdots + a_{n-1} \cdot s + a_n$$

The source code of the `place` function can be found in Appendix B.3. By executing this function, the following window appears (see Figure 5.12).


MotorcycleLibraryV8.Examples.BasicMotorcycle.ControllerDesign

ControllerDesign

Description

Controller design via pole-placement according to a preceding stability analysis

Inputs

modelName	V8.Examples.BasicMotorcycle.uncontrolled_3dof_motorcycle"		
independentVariableName	"vs"		
v	10	m/s	Forward velocity of the motorcycle

Input for Controller Design

d	10	Offset in order to shift the poles

State Selection (state vector)

states	{5,6,1,2}	steer angle, der(steer angle), lean angle, der(lean angle)

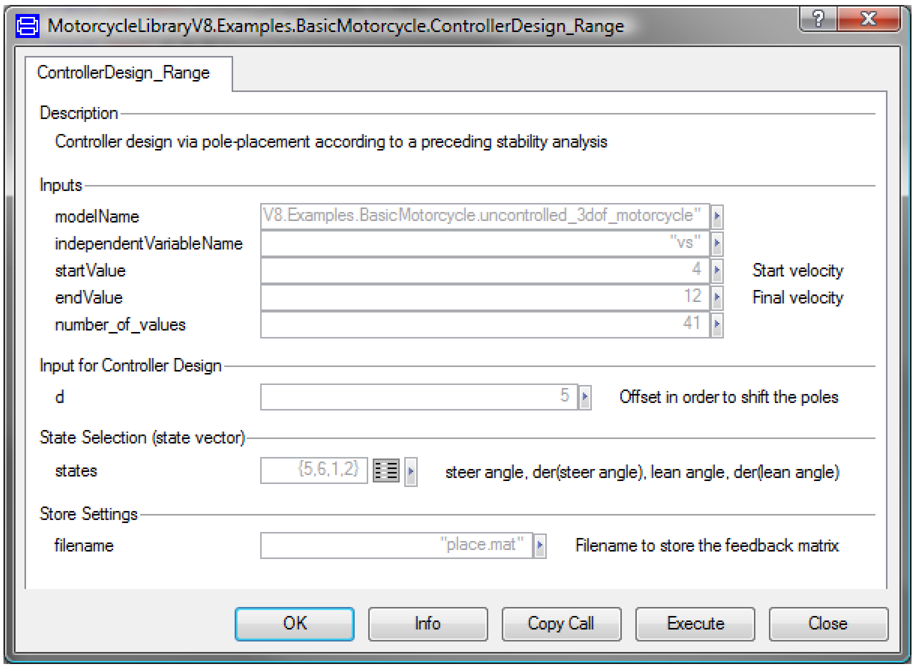
Figure 5.12: Parameter window of the Modelica place function

In order to calculate the coefficients of the state feedback matrix, one has to enter the velocity of the motorcycle, the offset and the appropriate states. The function returns the coefficients of the state feedback matrix which are displayed in the command window.

5.2.4 State-Space Controller for a Specific Velocity Range

1st Approach: Define an Offset in Order to Shift the Poles

The following state-space controller has exactly the same structure as the former one but is valid for a specific velocity range. After the `placeRange` function is executed, the parameter window in Figure 5.13 appears. The source code of this function can be found in Appendix B.3.



MotorcycleLibraryV8.Examples.BasicMotorcycle.ControllerDesign_Range

ControllerDesign_Range

Description

Controller design via pole-placement according to a preceding stability analysis

Inputs

modelName	V8.Examples.BasicMotorcycle.uncontrolled_3dof_motorcycle"	
independentVariableName	"vs"	
startValue	4	Start velocity
endValue	12	Final velocity
number_of_values	41	

Input for Controller Design

d	5	Offset in order to shift the poles

State Selection (state vector)

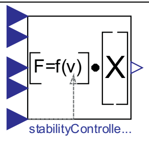
states	{5,6,1,2}	steer angle, der(steer angle), lean angle, der(lean angle)

Store Settings

filename	"place.mat"	Filename to store the feedback matrix

Figure 5.13: Parameter window of the Modelica placeRange function

Compared to the `place` function, additional parameters are needed. Instead of the velocity, the start and end value have to be entered. Now, all eigenvalues are shifted by the same value into the left-half plane. The output of the `placeRange` function is a matrix including as many rows as defined by `number_of_values`. Where each row includes the appropriate state feedback matrix. By default this matrix is stored in a file called *place.mat* and automatically fed into a table of a ready-made state-space controller (see Figure 5.14). In order to perform several simulations using the same vehicle, arbitrary filenames can be entered to store the results.



Name: StabilityController_table
Location: VirtualRider.VirtualRigidRider.
RollAngleTracking.StablilityControl

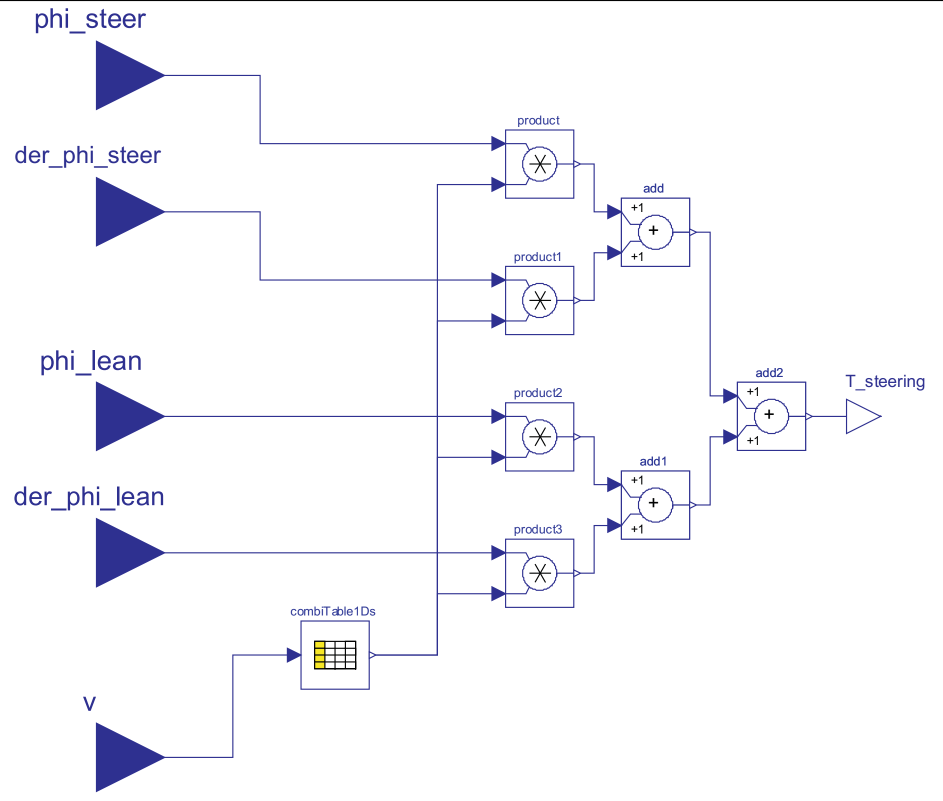


Figure 5.14: Wrapped model of a state-space controller for a specific velocity range. The inputs (blue) are the steer angle, lean angle and their derivatives and the forward velocity of the motorcycle, the output (white) is the steering torque. The table includes the state feedback matrix coefficients which by default are stored in *place.mat*

Figure 5.15 shows the stable and unstable regions of the motorcycle which were computed via an eigenvalue analysis. Figure 5.16 depicts the result of the controller de-

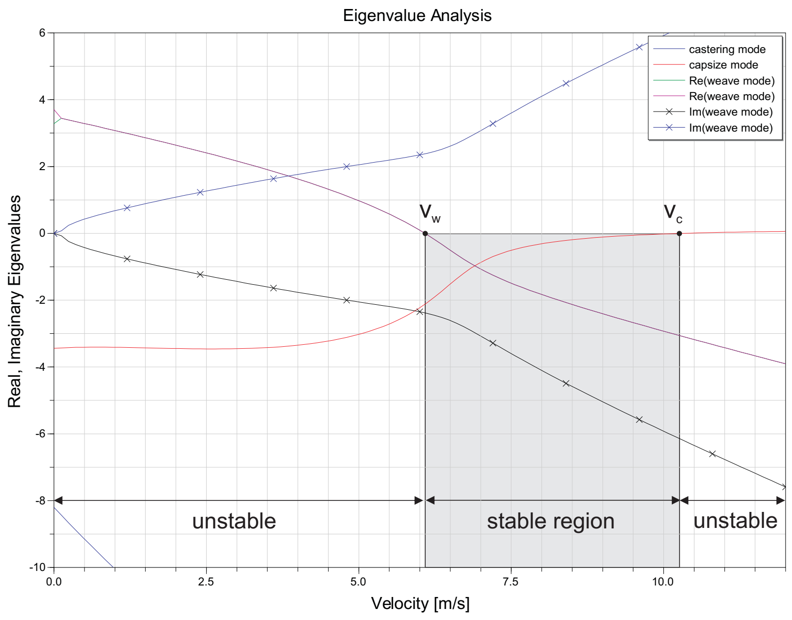


Figure 5.15: Controller design with reference to a preceding eigenvalue analysis. All eigenvalues are shifted by the same offset d

sign.

2nd Approach: Individual Controller

The former approach is based on an offset in order to shift all poles towards the left-half plane (i.e. to make the system faster).
But what are the pros of this method apart from the simple implementation? In other words: Is it really necessary to shift all poles towards the left-half plane?
Concerning the eigenvalue analysis, the answer to this question is rather simple. Within the stable region, the motorcycle needs no control and thus no offset. Above the stable region (for velocities higher than v_c) the behavior of the motorcycle is dominated by the capsize mode. Hence, it is absolutely sufficient to shift just this pole towards the left-half plane and leave all other poles unchanged. Below the stable region (for velocities lower than v_w) the instability of the motorcycle is caused by the weave mode (see Figure 5.17). To ensure stable behavior the two real parts of the conjugate complex poles have to be shifted towards the left-half plane. Now, a control law for the regions below and after the stable region is set up:

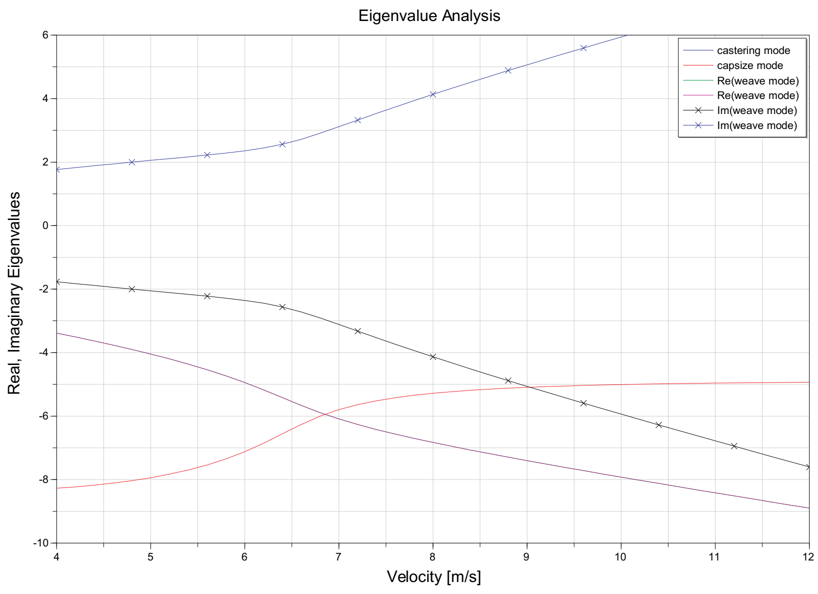


Figure 5.16: Result of the controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where the offset is $d = 5$

$$\text{control law} \begin{cases} v < v_w : & d = d_w \cdot (v_w - v) \\ v_w < v < v_c : & d = 0 \\ v_c < v : & d = d_c \cdot (v - v_c) \end{cases} \tag{5.10}$$

Figure 5.18 shows the result of the individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$.

3^{rd} Approach: Improved Individual Controller

Although the results of the individual controller are rather good, there is still potential for improvements. For velocities equal to v_w or v_c, the eigenvalues which are responsible for stability are close or equal to zero. To be more precise, for such velocities the stability of the motorcycle is critical since a real part equal to zero has no damping. Somewhere in the stable region, the weave and the capsize mode have an intersection point v_i. Instead of the previous control law, the improved control law results in:

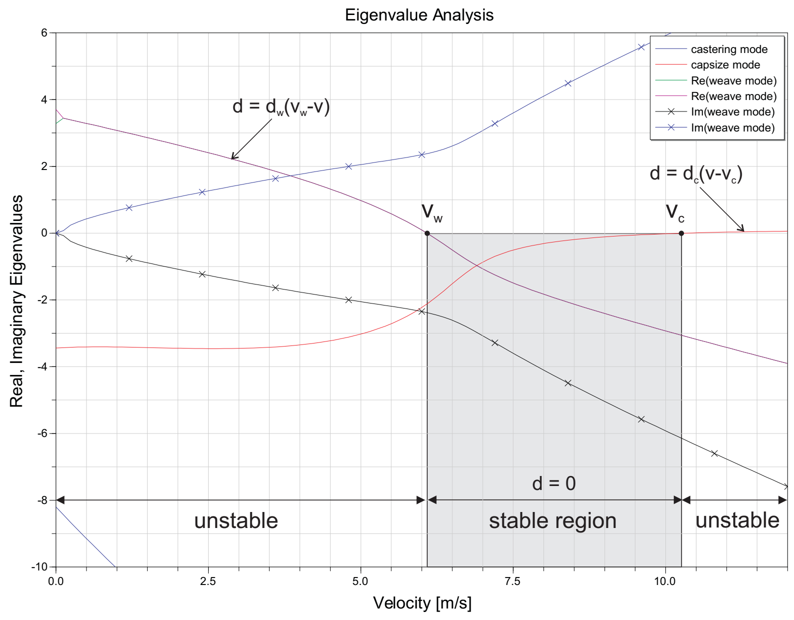


Figure 5.17: Controller design with reference to a preceding eigenvalue analysis. The stable region is left unchanged - within the region below v_w, the weave mode eigenvalues are modified - within the region above v_c, the capsize mode eigenvalue is modified.

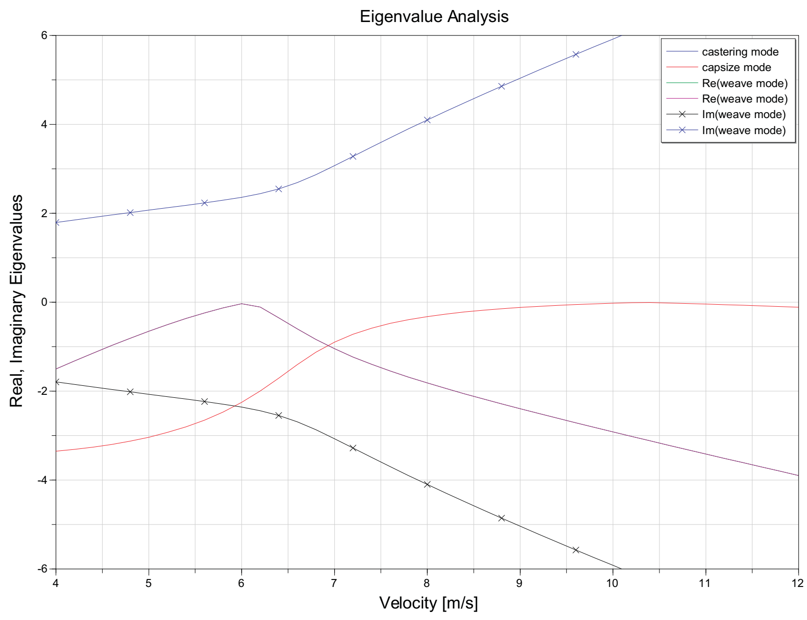


Figure 5.18: Result of the individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where $v_w = 6.1ms^{-1}$, $v_c = 10.3ms^{-1}$, $d_w = 1.5$ and $d_c = 0.1$

$$\textit{control law} \begin{cases} v < v_i : & d = d_0 + d_w \cdot (v_i - v) \\ v_i < v : & d = d_0 + d_c \cdot (v - v_i) \end{cases} \tag{5.11}$$

A graphical interpretation of the control law is illustrated in Figure 5.19.

Figure 5.20 depicts the result of the improved individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$.

5.3 State-Space Controller Valid for Models Composed of Movable Riders

So far, the states $x = (\delta \; \dot{\delta} \; \phi \; \dot{\phi})^T$ of the controller introduced in Section 5.2.3 were fed back in order to generate an appropriate steering torque. Now, two more states are taken into account, namely the lean angle γ and the lean rate $\dot{\gamma}$ of the rider relative to the rear frame. These states are also fed back to generate a suitable torque which is applied by the rider in order to balance the vehicle. The corresponding control law is

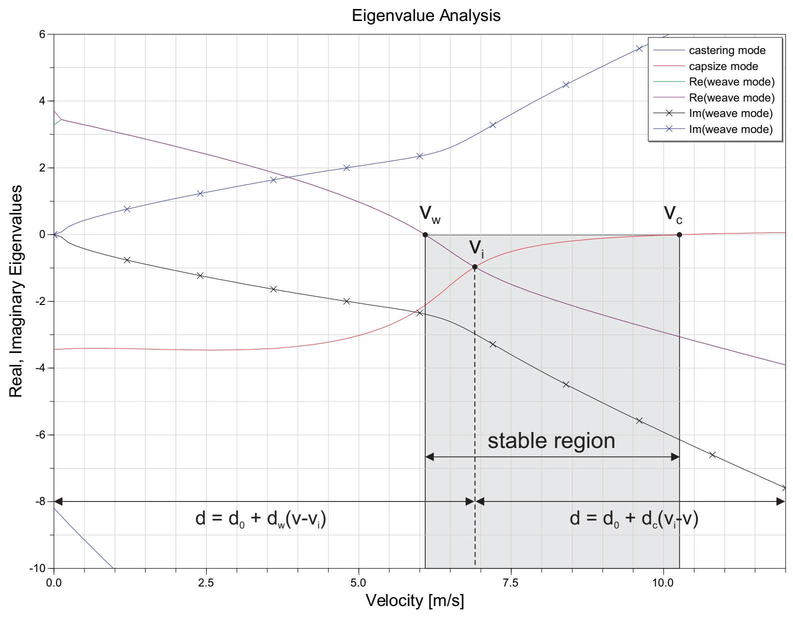


Figure 5.19: Controller design with reference to a preceding eigenvalue analysis. Within the region below v_i, the weave mode eigenvalues are modified - within the region above v_i, the capsize mode eigenvalue is modified. In addition the weave and capsize eigenvalues can be shifted by a value equal to d0.

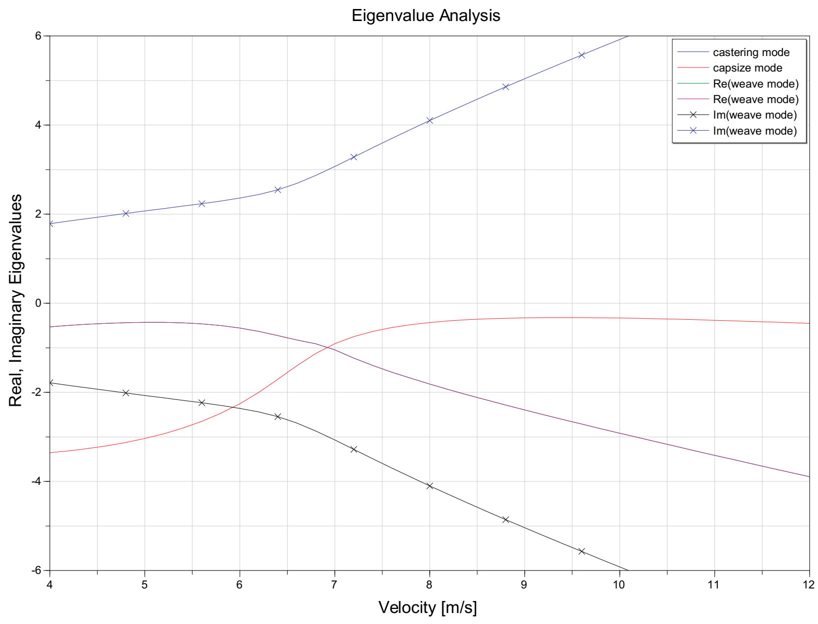


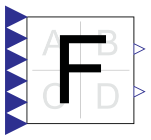
Figure 5.20: Result of the improved individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where $v_i = 6.9ms^{-1}$, $d_w = 0.75$, $d_c = 0.1$ and $d_0 = 0$

given by:

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = - \begin{pmatrix} f_{11} & f_{12} & f_{13} & f_{14} & f_{15} & f_{16} \\ f_{21} & f_{22} & f_{23} & f_{24} & f_{25} & f_{26} \end{pmatrix} \cdot \begin{pmatrix} \delta \\ \dot{\delta} \\ \phi \\ \dot{\phi} \\ \gamma \\ \dot{\gamma} \end{pmatrix} \tag{5.12}$$

Basically, the controller design is carried out in the same way as it was described in Section 5.2.2. The model of the controller is shown in Figure 5.21.

The corresponding MATLAB *m-file* can be found in Appendix C.2. Unlike in former controller designs, a second offset `offset_rider` is introduced in order to move the poles of the rider independently of the other ones. However, this was just a suggestion on how to design the controller.



Name: StateSpaceController
Location: VirtualRider.VirtualMovableRider.
RollAngleTracking.StabilityControl

```
model StateSpaceController

  // The controller matrix is calcutated with Matlab
  parameter Real F[2,6] = [ 34.2606,    1.6523,   2   2.7179,    3.4349,   10.2637,    3.1373;
                           118.7795,   29.6039,  16   9.5166,   64.2751,  246.6806,   84.0309];


equation
  // T = F*X
  // T ... Torque input
  // F ... Feedback matrix
  // X ... State vector

 {SteeringTorque, RiderLeanTorque} = F*{SteerAngle, SteerRate, LeanAngle, LeanRate, RiderLeanAngle, RiderLeanRate};

end StateSpaceController;
```

Figure 5.21: Model of a state-space controller. The inputs (blue) are the steer angle, lean angle, rider's lean angle and their derivatives, the outputs (white) are the steering torque and a torque that has to be applied by the upper body of the rider in order to lean sideways

5.4 A Linear Quadratic Regulator (LQR) as a Solution of Optimal Control

Another possibility to design a controller is to use a linear quadratic regulator (LQR). Until now, each controller was designed in such a way that the closed-loop system becomes stable. In order to fulfill this requirement the poles of the closed-loop system were placed into the left-half plane. This was done using several different approaches. However, the basic idea is always the same: The more negative the location of the poles, the faster the system and the more *control energy* (e.g. steering torque) is needed. So far, the control energy (cost) of the system was never taken into account, i.e. it was implicitly assumed that the control energy does not play an important role. Unfortunately, this is everything else but close to reality (e.g. a steering torque of 500Nm can never be applied by a human).

To take the control energy into account right from the start of the controller design, the optimal control theory provides several methods in order to minimize it. For linear time-invariant (LTI) systems the LQR method is the most suitable one.

Basic Idea

Again, the plant is considered to be a linear system which is described in terms of Equation 5.1. The state feedback matrix F has to be chosen in such a way that:

1. the closed-loop system is not too slow and not too much oscillating

2. the control energy is minimized

The quality criterion (cost functional) in order to fulfill (1) takes the following form:

$$J = \frac{1}{2} \int_{0}^{\infty} x^T(t)\; Q\; x(t)\; dt \tag{5.13}$$

where x is the state vector and Q is a symmetric, positive semidefinite matrix

Analog to (1) the quality criterion of (2) is given by:

$$J = \frac{1}{2} \int_{0}^{\infty} u^T(t)\; R\; u(t)\; dt \tag{5.14}$$

where u is the input vector and R is a symmetric, positive definite matrix

In order to fulfill both quality criteria Equation 5.13 and 5.14 are combined. The resulting cost functional is given by:

$$J = \frac{1}{2} \int_{0}^{\infty} \left[x^T(t)\; Q\; x(t) + u^T(t)\; R\; u(t) \right] dt \tag{5.15}$$

where Q and R are weighting matrices.

To minimize the cost functional J, the control law $u = -F \cdot x$ (Equation 5.3) is modified. F is substituted by

$$F = -R^{-1} \cdot B^T \cdot P \tag{5.16}$$

where P is found by solving the following equation

$$A^T \cdot P + P \cdot A - P \cdot B \cdot R^{-1} \cdot B^T \cdot P + Q = 0 \tag{5.17}$$

which is referred to as continuous time *algebraic riccati equation.*

For a detailed description of the equations above, refer to Föllinger [Föl08] or Lunze [Lun06]. To this end it is not necessary to understand the theory in detail since the algorithm, in order to find the optimal state feedback matrix, is provided by many programs (e.g. MATLAB, Scilab, Octave).

Due to the absence of a Modelica LQR function, this problem is solved with the `lqr()`-function provided by MATLAB.

The main task is to find appropriate weighting factors for Q and R to fulfill (1) and (2). In order to do that, some design goals have to be specified. According to that goals, Q and R are modified as long as the results are such that the requirements are fulfilled. This is a laborious and time consuming task. Nevertheless, in the following it is carried out for the basic bicycle model.

5.4.1 1^{st} Approach: Minimization of the Lean Angle

In a first approach the lean angle is minimized. Since the bicycle has just one input, namely the steering torque $u = T_{steering}$ applied by the rider, the R matrix is reduced to a scalar. In the simplest case $R = 1$. The state vector $x = (\delta\ \dot{\delta}\ \phi\ \dot{\phi})^T$ has a size of (4×1) and hence Q is a (4×4) matrix. Now, the weightings of Q have to be chosen such that ϕ is minimized. Thus Q results in:

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \leftarrow \mathit{min}\ \phi$$

The value of R which is equal to one, states that the control energy has the same weighting as the lean angle. More precisely, the lean angle is minimized by the same value as the steering torque.

In the following, a controller design for a velocity range from $v = 1ms^{-1}$ to $10ms^{-1}$ is carried out. In order to calculate the coefficients, the basic bicycle model has to be linearized for each velocity. After each linearization, the procedure introduced in Chapter 5.2.2 is executed. The only difference is that four states are needed and that the `place()`-function is replaced by an `lqr()`-function.

```
states = [7,8,3,4];     % Steer Angle, Steer Rate, Lean Angle, Lean Rate

Arel = A(states, states);
Brel = B(states,:);
```

The lines of code in order to calculate the state feedback matrix F are shown below.

```
R = 1;                 % single input

q1  = 0;               % weighting factor steer angle
q2  = 0;               % weighting factor steer rate
q3  = 1;               % weighting factor lean angle
q4  = 0;               % weighting factor lean rate
Q = [q1 0 0 0; 0 q2 0 0; 0 0 q3 0; 0 0 0 q4];

F = lqr(Arel, Brel, Q, R)
```

The MATLAB *m-file* can be found in Appendix C.3.

The model of the LQR is shown in Figure 5.22.

Name: LQR_LeanSteer
Location: VirtualRider.VirtualRigidRider.RollAngleTracking.StablilityControl

```
model LQR_LeanSteer
  "LQR including lean angle, steer angle and their derivatives as states"

  // The calculation of the feedback matrix coefficients is done with MATLAB
  parameter Real F[1,4] = [9.5166, 1.9268, 150.4986, 48.1777];

equation
  // T = F*X
  // T ... Torque input
  // F ... Feedback matrix
  // X ... State vector

  {SteeringTorque} = F*{phi_steer, der_phi_steer, phi_lean, der_phi_lean};

end LQR_LeanSteer;
```

Figure 5.22: Model of linear quadratic regulator (LQR). The calculation of the coefficients is carried out with MATLAB.

The results of the simulation are shown in Figure 5.23

This approach was also carried out by Schwab et al. in their recently published paper [SKM08] on page 4/8. Fortunately the results are the same. Figure 5.24 shows the steering torque for velocities of $v = 1ms^{-1}$ and $v = 4ms^{-1}$. For initial lean angles greater than 11.6° and a velocity of $1ms^{-1}$, the torque gains unrealistic high values. Furthermore, it can be seen that the real parts of the weave mode eigenvalues have their maximum at a velocity of $v = 4ms^{-1}$. Thus, the amplitude of the steering torque is lower. Since the lean angle is minimized, almost no steering torque is required for velocities equal to $4ms^{-1}$. For velocities equal to $5ms^{-1}$ the weave eigenvalues of the uncontrolled bicycle are the same as for the controlled version. Hence the steering torque is almost zero. In that case, the coefficients of the state feedback matrix are $F = [0.6038,\ 0.0424,\ 0.6934,\ 0.0420]$.

5.4.2 2^{nd} Approach: Minimization of Lean and Steer Angle

In order to minimize the lean and the steer angle, the weightings of the Q matrix are modified as follows:

$$Q = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \leftarrow min\ \delta \\ \\ \leftarrow min\ \phi \\ \\ \end{array}$$

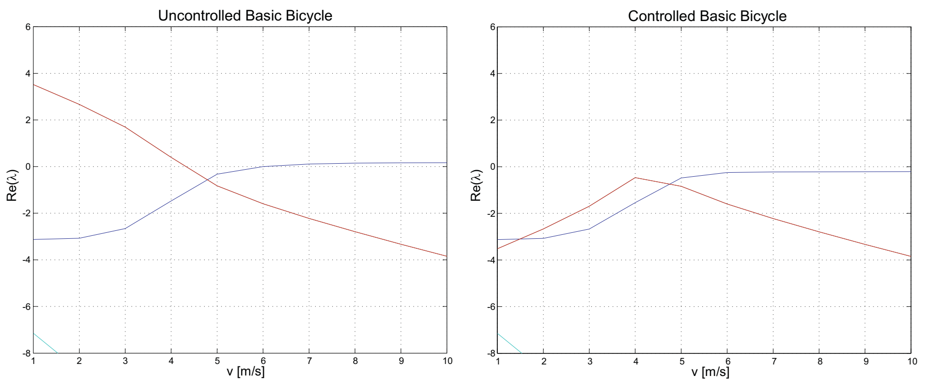


Figure 5.23: LQR design - minimization of the lean angle: Left plot - eigenvalue analysis for the uncontrolled version of the bicycle; Right plot - eigenvalues of the controlled bicycle

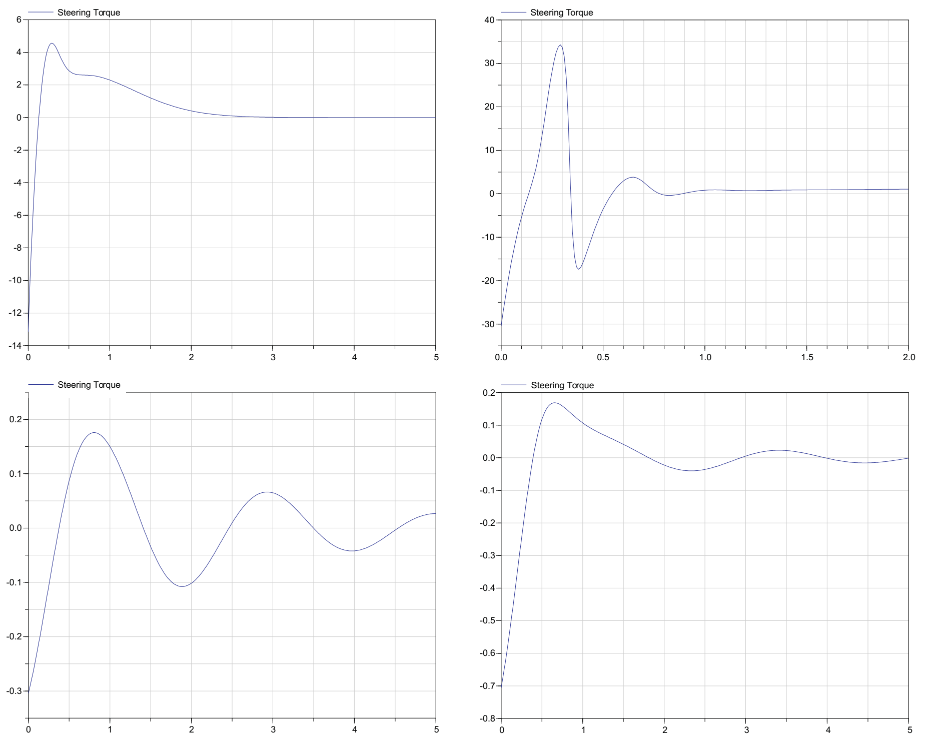


Figure 5.24: LQR design - steering torque needed in order to stabilize the bicycle; Top left - Initial lean angle $\phi = 5°$, $v = 1ms^{-1}$; Top right - Initial lean angle $\phi = 11.6°$, $v = 1ms^{-1}$; Bottom left - Initial lean angle $\phi = 5°$, $v = 4ms^{-1}$; Bottom right - Initial lean angle $\phi = 11.6°$, $v = 4ms^{-1}$

Figure 5.25 shows a comparison between the first and the second approach.

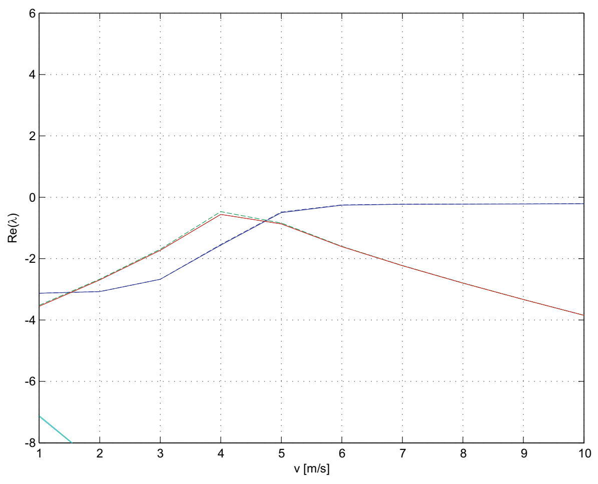


Figure 5.25: LQR design - minimization of the lean and steer angle: The dashed lines are the results of the 1st approach.

Regarding Figure 5.25, one can see that the additional minimization of the steer angle does not make a useful contribution.

5.4.3 3rd Approach: Minimize the State Vector x

For the sake of simplicity, every LQR design starts with minimizing the whole state vector and the input by the same value. Thus, Q is equal to a 4×4 identity matrix, whereas R = 1 again. The result is shown in Figure 5.26

The result is a little bit different compared to the first two approaches, especially below the weave speed v_w. However, the applied torque for low speeds is almost the same.

5.4.4 4th Approach: Minimization by Uneven Weightings

The last approach is used to vary the matrix R. At first R is set to a value greater than one $R = 10$. The Q matrix is the same as the one introduced in the first approach. The result is shown in Figure 5.27.

Unfortunately, a weighting of R = 1 was almost the best possible case. Hence, the control energy cannot be decreased any more. On the contrary, weightings of R much smaller than one would cause a faster system and much more control energy in order to keep the bicycle upright.

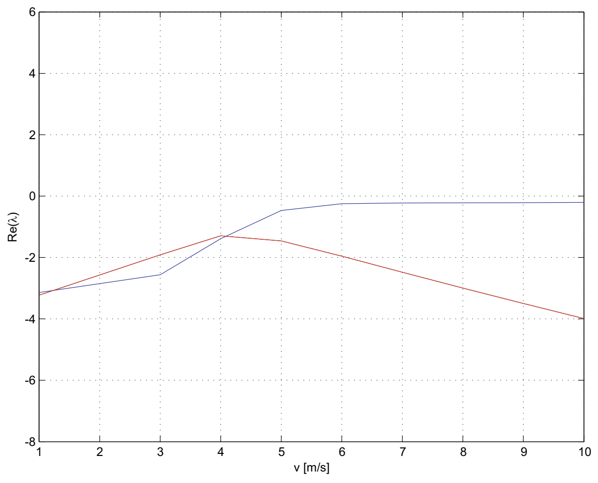


Figure 5.26: LQR design - minimization of the state vector

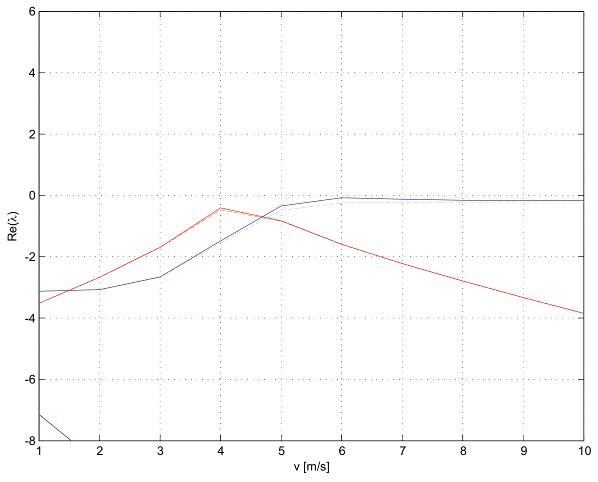


Figure 5.27: LQR design - varying weightings of R. The dashed lines are the results of the first approach. Although $R = 10$, the results are almost the same

5.5 LQR valid for Models composed of Movable Riders

In the following, an LQR valid for the 4 d.o.f. bicycle is designed. The control law and thus the model of the controller are equal to those introduced in Section 5.3. Contrary to the LQR design in Section 5.4, the weighting matrix R now has size (2×2) and Q is increased to size (6×6). Regarding the control energy as well as the controller performance, the following approach is carried out.

$$Q = \begin{array}{cl} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} & \begin{array}{l} \\ \\ \leftarrow \mathit{min}\ \phi \\ \\ \leftarrow \mathit{min}\ \gamma \\ \\ \end{array} \end{array}$$

and

$$R = \begin{array}{cl} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{array}{l} \leftarrow \mathit{min}\ u_1 \\ \leftarrow \mathit{min}\ u_2 \end{array} \end{array}$$

where u_1 is the steering torque and u_2 is the body torque that has to be applied by the upper body of the rider. The calculation is done with MATLAB. The *m-file* can be found in Appendix C.2. In the following, the coefficients of the state feedback matrix for a bicycle velocity of $3ms^{-1}$ are calculated. The state feedback matrix results in:

```
F = [26.6202    2.2106   26.9760    0.2602  −39.3326  −12.2478;
    −11.9325   −0.7383   −8.8855    4.5996   43.8729   13.6827]
```

The simulation result is depicted in Figure 5.28.

Regarding the results of Figure 5.28 it can be seen that the behavior of the controlled bicycle is rather good. In order to increase the controller performance, that is, to model an aggressive rider, the coefficients Q(3,3) and Q(5,5) have to be increased. However, this could result in additional control energy. The behavior of the bicycle with R(1,1) = R(2,2) = 0.1 and Q(3,3) = Q(5,5) = 100 is shown in Figure 5.29.

5.6 Velocity Control

The wrapped model of a simple velocity controller is shown in Figure 5.30.

The inputs are the set-value of the velocity and the angular velocity of the motorcycle's rear wheel. The output of the gain block with the value $\frac{1}{r}$ is the corresponding angular

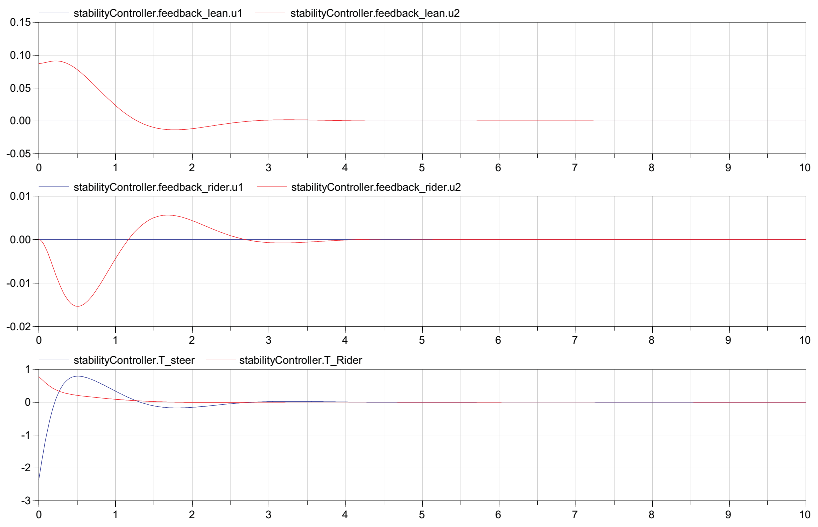


Figure 5.28: Simulation results of the 4 d.o.f. controlled bicycle model

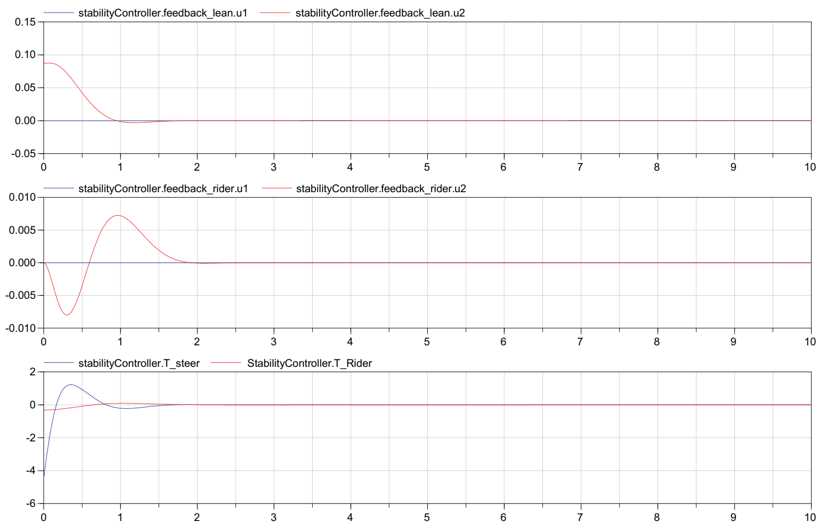
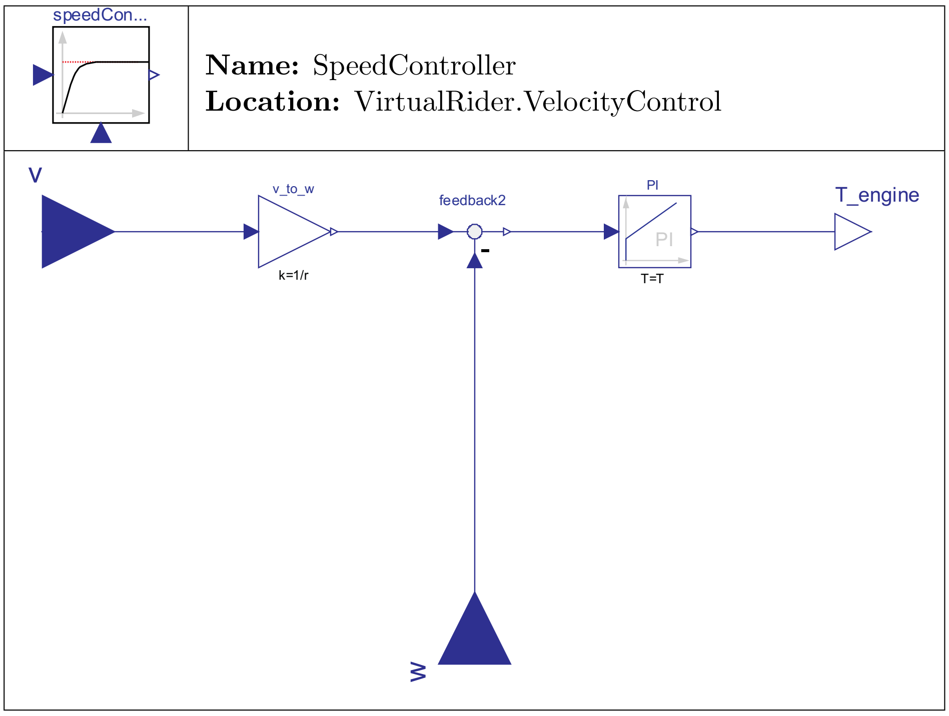


Figure 5.29: Optimized simulation results of the 4 d.o.f. controlled bicycle model

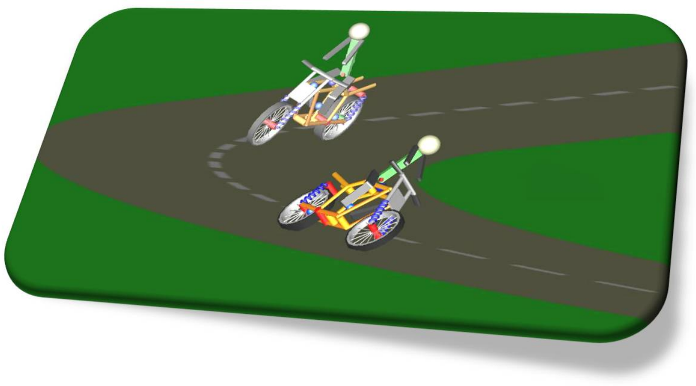


Name: SpeedController
Location: VirtualRider.VelocityControl

Figure 5.30: Wrapped model of a simple velocity controller.

velocity set-value. The output of the model is the appropriate torque to drive the rear wheel.

6 Development of a Virtual Rider



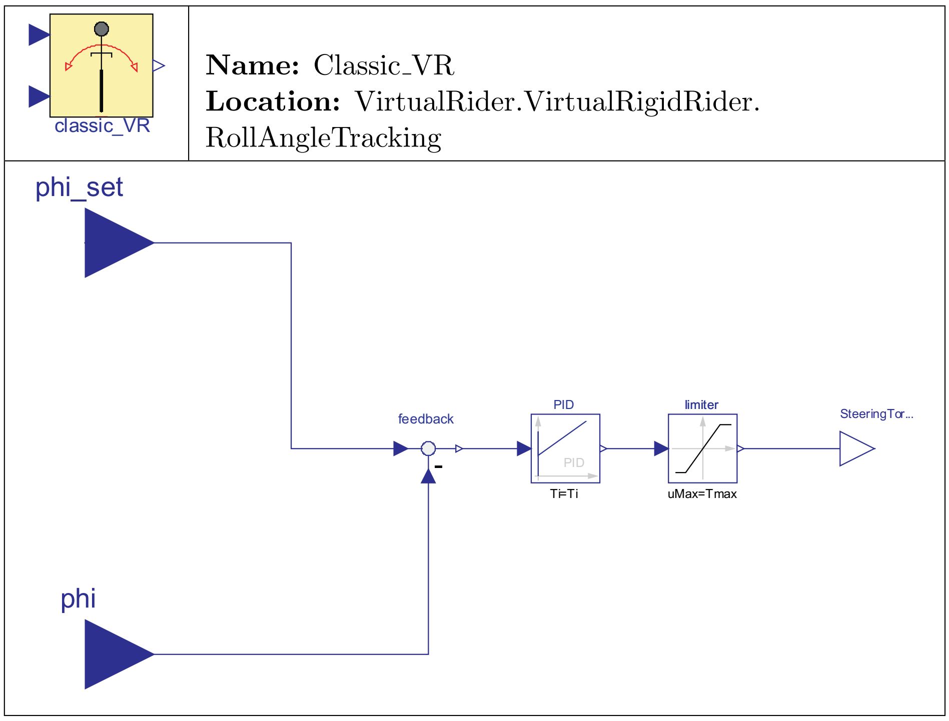
To validate the motorcycle's performance, a virtual rider is essential. The main purpose of a virtual rider is to track either a roll angle profile (open-loop method) or a pre-defined path (closed-loop method). In the simplest case, the rider is considered to be an inert mass rigidly attached to the main (rear) frame of the motorcycle (e.g. basic motorcycle model). More realistic models introduce an additional degree of freedom that enables the rider's upper body to lean sideways. Another essential task of a virtual rider is to stabilize the motorcycle either for a specific velocity or within a specific velocity range. Therefore, appropriate state variables (e.g. lean (roll) angle and lean rate) are fed back to generate a suitable steering torque. Several possibilities exist to cope with such a control task. The simplest case is by classic controller design, i.e. the use of PI-, PD- or PID-elements. Modern approaches are based on a state-space representation.

In the following chapter, several different virtual riders capable of either tracking a roll angle profile or a pre-defined path are introduced. For a virtual rider capable of tracking a roll angle profile, the stability controllers which were introduced in Chapter 5 are used. Instead of the previous set values, which were all set to zero, a roll angle profile is used. For a virtual rider capable of tracking a pre-defined path, classic-, state-space- and combinations of classic- and state-space controllers are applied. To develop a more realistic virtual rider, *single-point path preview* is performed. This means that the virtual rider looks a pre-defined distance ahead in order to follow the path. It is worth noting that a similar deviation pattern is actually observed from human riders. In the simplest case, a stability controller is extended by a PI-controller. The input of the PI-Controller is the lateral position of the rear frame's center of mass. The output is an additional steering torque which is added to the existing one. Afterwards, the PI-controller is replaced by a state-space controller. Contrary to the former approach, the derivative of the lateral position is additionally taken into account. Finally, a linear quadratic regulator (LQR) is developed.

6.1 Roll Angle Tracking

6.1.1 Classic Design

The simplest implementation of a virtual rider was implicitly introduced in Chapter 5.1. For the sake of usability, this model is wrapped (see Figure 6.1). The additional *limiter*



Name: Classic_VR
Location: VirtualRider.VirtualRigidRider.
RollAngleTracking

Figure 6.1: Wrapped model of a simple virtual rider. The inputs (blue) are the lean angle set-value and lean angle, the output (white) is the steering torque

block is used to ensure realistic torque inputs. In Chapter 5, all set-values were equal to zero. Let us now apply a lean angle set-value in order to follow a desired profile. In the simplest case, a constant value is entered. Figure 6.2 show the controlled model of the basic motorcycle.

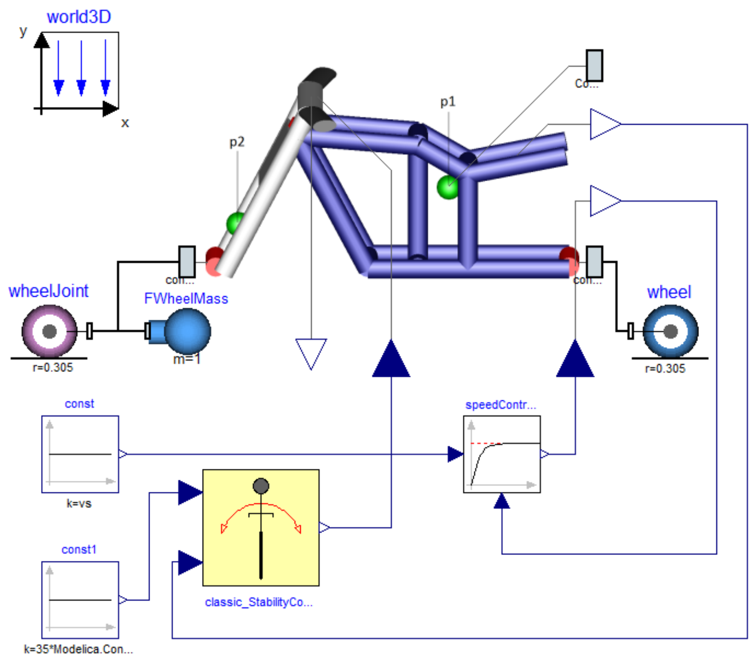


Figure 6.2: Wrapped model of the controlled basic motorcycle

Figure 6.3 shows the simulation result for a set value $x_{set} = 35°$ and a constant velocity of $v = 6ms^{-1}$. The corresponding signals are shown in Figure 6.4. The coefficients of the PID controller were determined by trial-and-error ($k = 20$, $T_i = 2.5$ and $T_d = 10^{-3}$). The maximum torque that can be applied by the rider was set to $T_{max} = 5Nm$.

6.1.2 State-Space Design

Controller Extensions

In order to track a roll angle profile, the state-space controllers introduced in Chapter 5 are extended by a so-called *reference input*. So far, no reference input was used, i.e. the set-value of the state variables was zero (see Figure 5.4) This raises the following question:

Why do all states converge towards zero without explicitly using a reference input equal to zero?

Let us apply a reference input to the block diagram in Figure 5.4. The result is shown in Figure 6.5.

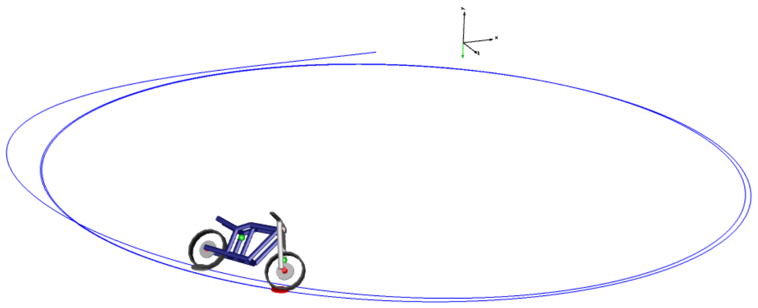


Figure 6.3: Animation result of the basic motorcycle model with a classic virtual rider tracking a 35° roll angle profile with a forward velocity of $6ms^{-1}$

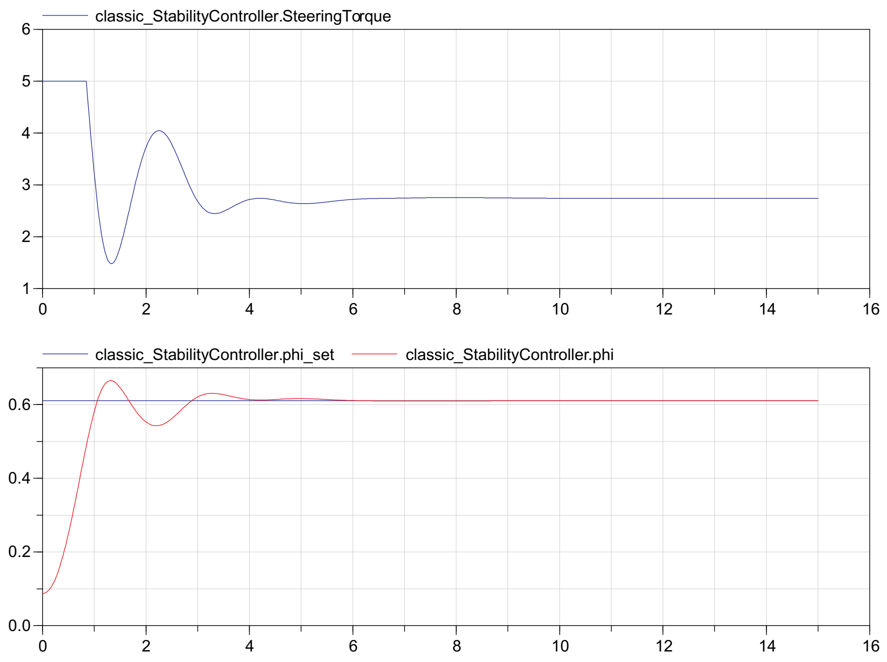


Figure 6.4: Simulation result of the basic motorcycle model with a classic virtual rider tracking a 35° roll angle profile with a forward velocity of $6ms^{-1}$. The upper plot shows the steering torque, the lower plot shows the set-value and the actual lean angle of the motorcycle

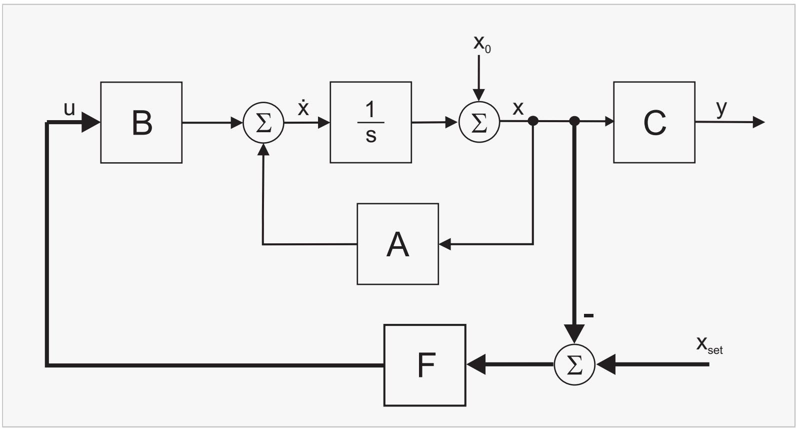


Figure 6.5: State feedback with an additional reference input

Mathematically the control law introduced in Equation 5.3 results in:

$$u = F \cdot (x_{set} - x) \tag{6.1}$$

If a reference input $x_{set} = 0$ is applied, then $u = -F \cdot x$ which is nothing else but the original control law without any reference input. Thus, in a state-space design without a reference input all states converge towards zero.

A Simple Virtual Rider

Let us apply a reference input to the state-space controller depicted in Figure 5.6. The modified block diagram is shown in Figure 6.6. The reference input x_{set} is the desired lean angle profile. The derivative block provides the corresponding reference input for the lean rate $\dot{\phi}$. In order to develop models with less inputs, the block diagram in Figure 6.6 is rebuilt (see Figure 6.7).

The wrapped model is shown in Figure 6.8.

A Virtual Rider Based on a Preceding Eigenvalue Analysis

In the following, a virtual rider based on the state-space controller introduced in Chapter 5.2.3 is developed, i.e. to be able to follow a roll angle profile, the steer angle is additionally taken into account. In other words, for a given *lean angle set-value* x_{set}, the corresponding steer angle has to be calculated. The relation between steer and lean

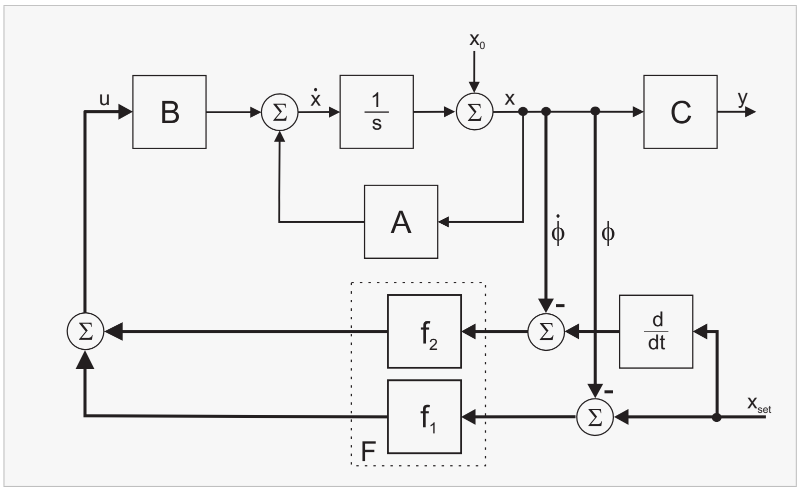


Figure 6.6: Block diagram of a simple virtual rider

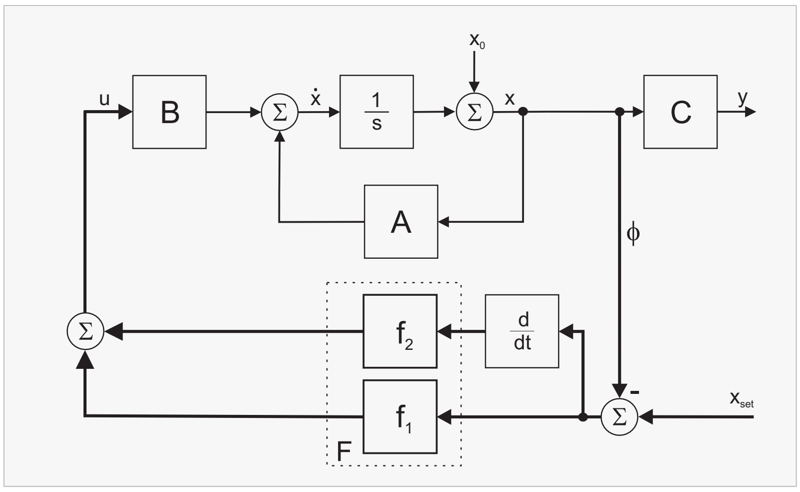
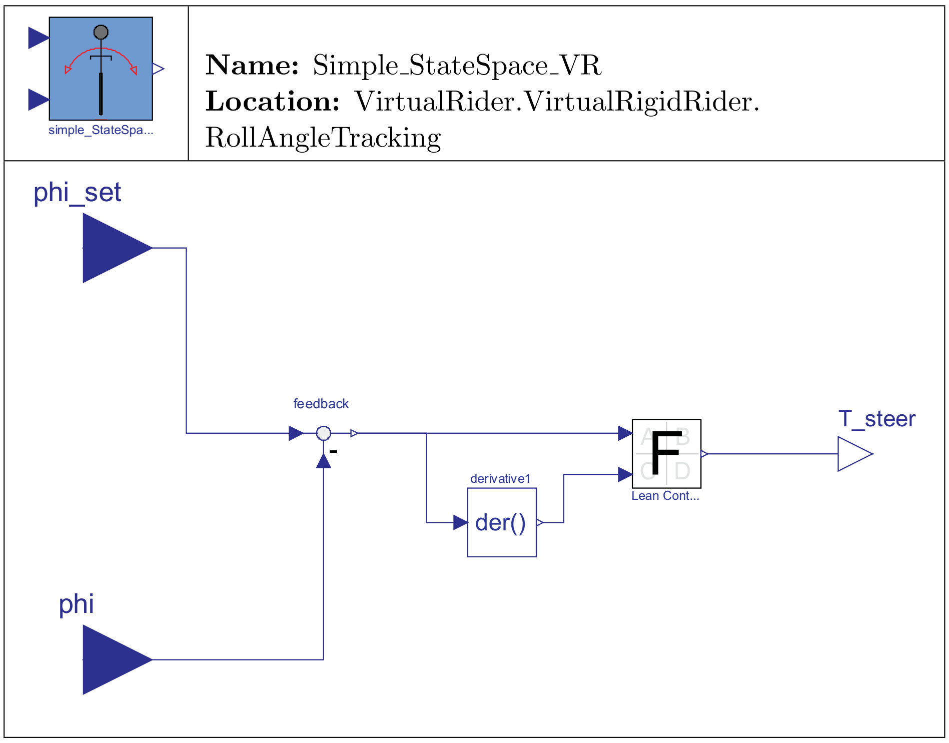


Figure 6.7: Another possible block diagram of a simple virtual rider



Name: Simple_StateSpace_VR
Location: VirtualRider.VirtualRigidRider.
RollAngleTracking

Figure 6.8: Wrapped model of a simple virtual rider composed of a simple state-space controller. The inputs (blue) are the lean angle and lean rate, the output (white) is the steering torque

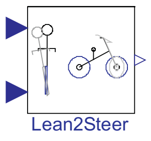
angle, under the assumption of small perturbations, i.e. $sin(\phi) \approx \phi$, is given by:

$$\delta = atan\left(\frac{p \cdot cos(\phi)}{R \cdot tan(\epsilon)}\right) \tag{6.2}$$

Where p is the wheel base, ϵ is the caster angle, $\phi = x_{set}$,

$$R = \frac{v^2}{g \cdot tan(\phi)} \tag{6.3}$$

is the radius of the curve (refer to Equation 2.5) and g is the gravity. The corresponding model is depicted in Figure 6.9. Once again, the block diagram, which was introduced in



Name: lean2steer
Location: VirtualRider.Utilities

```
model lean2steer
  "Calculation of the steer angle for a given lean angle"

  import SI = Modelica.SIunits;
  import CO = Modelica.SIunits.Conversions;
  import MA = Modelica.Math;

  parameter SI.Distance p = -1.414 "wheel base";

  parameter CO.NonSIunits.Angle_deg eps = -27
    "Caster angle (Steering head angle)";
  final parameter SI.Angle eps_rad = CO.from_deg(eps);

  SI.Distance R "Curve radius";

equation
  R = noEvent( if abs(lean) < 10e-6 then 0 else v^2 / (9.81*tan(lean)));
  steer = noEvent( if abs(R) < 10e-6 then 0 else MA.atan(p*cos(lean) / (R*cos(eps_rad))));


equation

end lean2steer;
```

Figure 6.9: Model that calculates the steer angle for a given lean angle ($\delta = f(\phi)$). The inputs (blue) are the lean angle and the velocity of the motorcycle, the output (white) is the corresponding steer angle.

Chapter 5.9, is modified (see Figure 6.10). The wrapped model is shown in Figure 6.11.

A Virtual Rider Based on a Preceding Eigenvalue Analysis Valid for a Specific Velocity Range

To design such a virtual rider the incorporated controller of the virtual rider shown in Figure 6.11 is replaced by the controller introduced in Chapter 5.14 in order to work

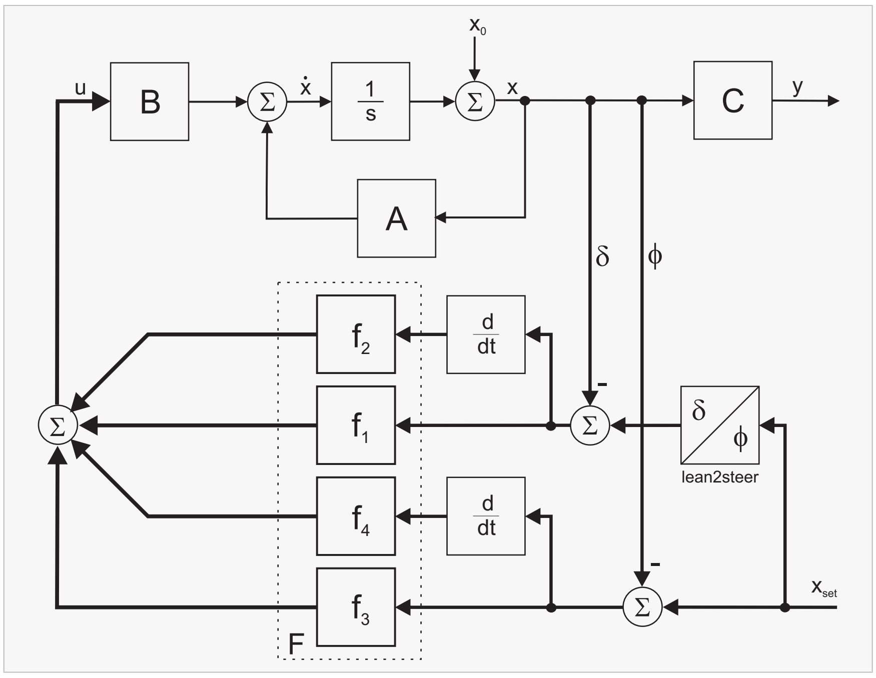


Figure 6.10: Block diagram of a virtual rider composed of a state-space controller and an additional block in order to calculate the corresponding steer angle for a given lean angle

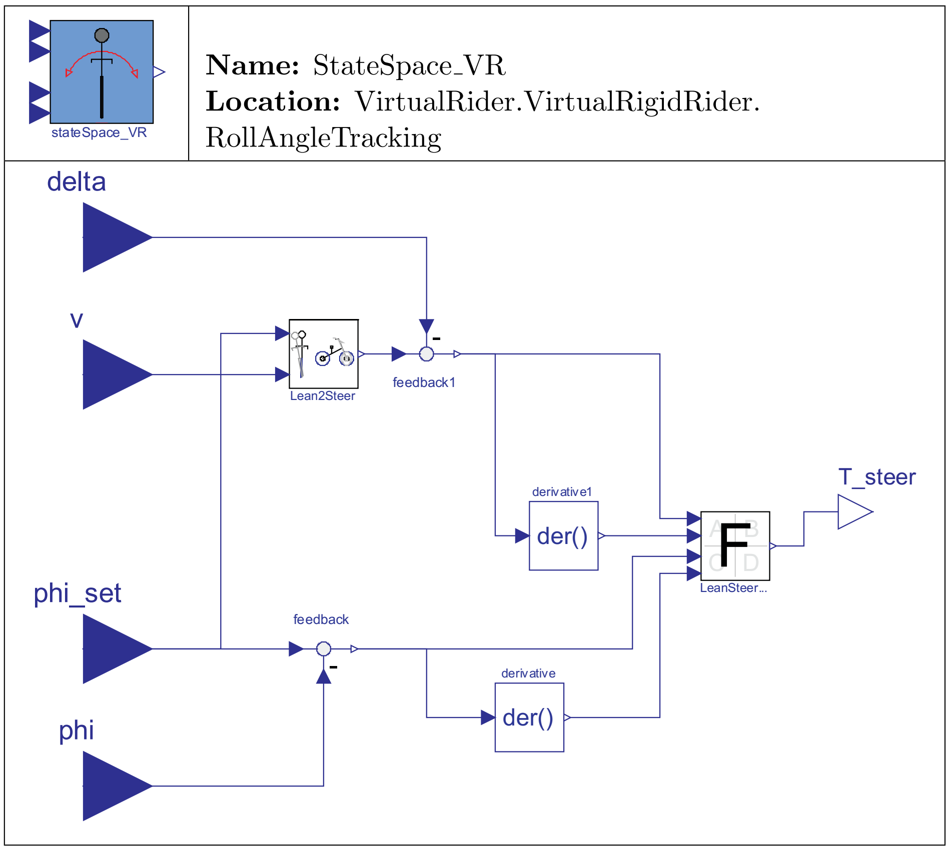
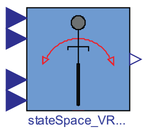


Figure 6.11: Wrapped model of the virtual rider composed of a state-space controller. The inputs (blue) are the lean and steer angle, the lean angle set-value and the velocity of the motorcycle, the output (white) is the steering torque

correctly within a user-defined velocity range. The wrapped model is illustrated in Figure 6.12.



Name: StateSpace_VR_Range
Location: VirtualRider.VirtualRigidRider.
RollAngleTracking

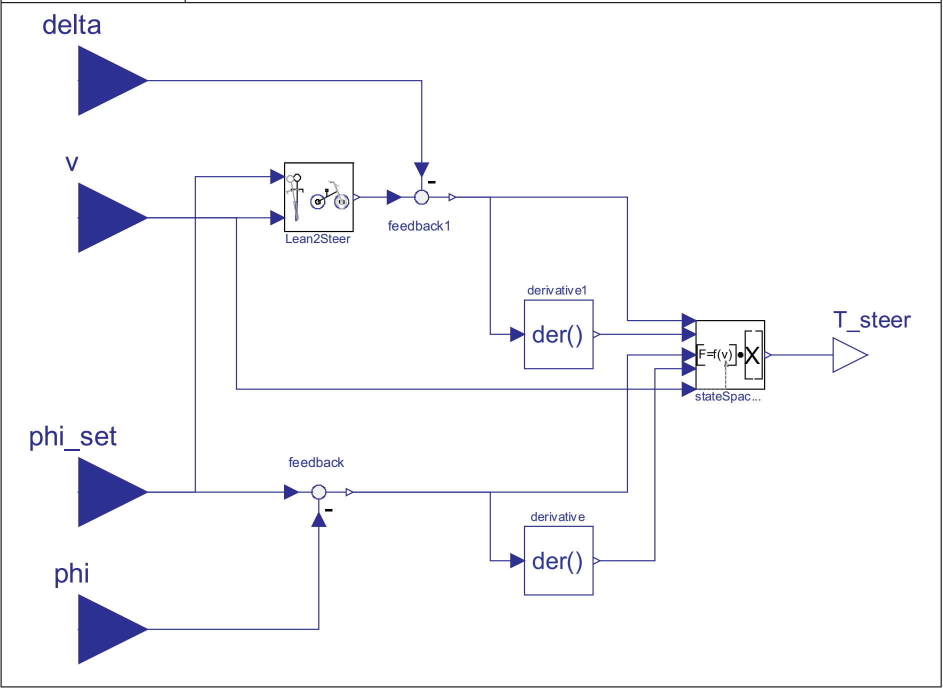


Figure 6.12: Wrapped model of the virtual rider composed of a state-space controller for a user defined velocity range. The inputs (blue) are the lean and steer angle, lean angle set-value and the velocity of the motorcycle, the output (white) is the steering torque

6.1.3 State-Space Design for Models Composed of Movable Riders

The wrapped model of such a controller is shown in Figure 6.13. Apart from that, it works in the same way as all the other virtual riders described so far.

A detailed example of a controlled 4 d.o.f bicycle model can be found in Chapter 7.1.2.

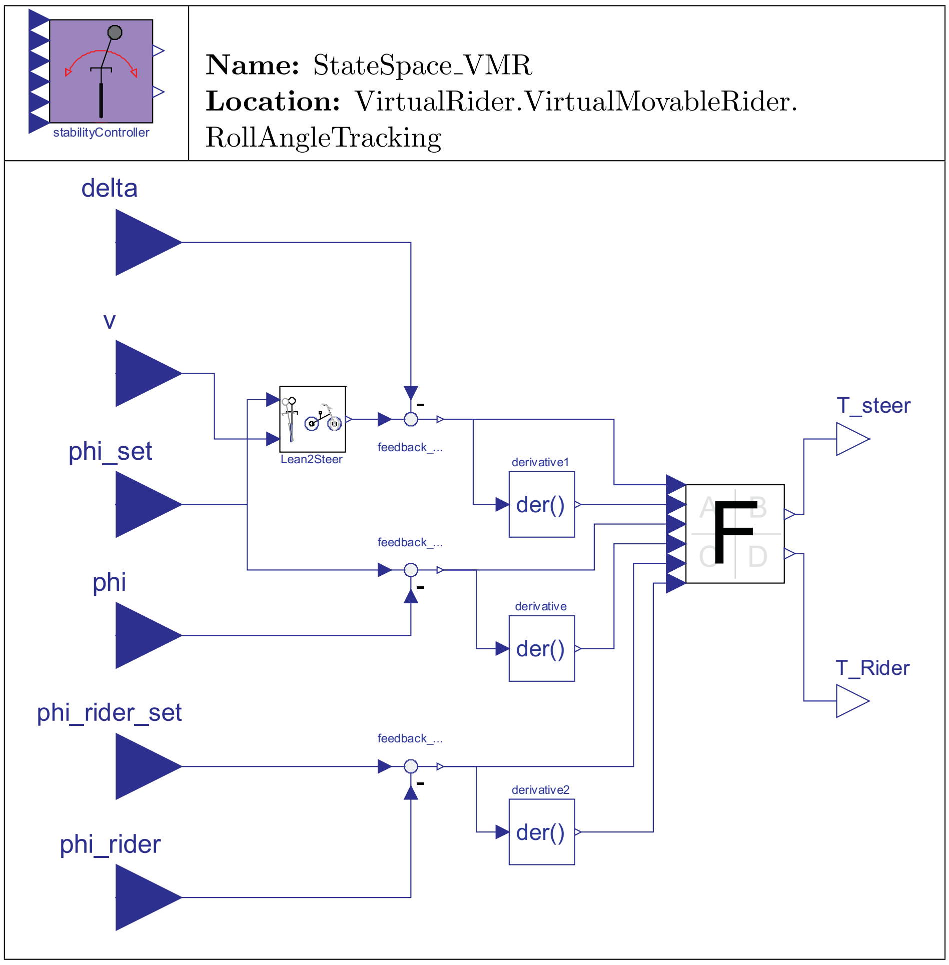

Name: StateSpace_VMR
Location: VirtualRider.VirtualMovableRider.RollAngleTracking

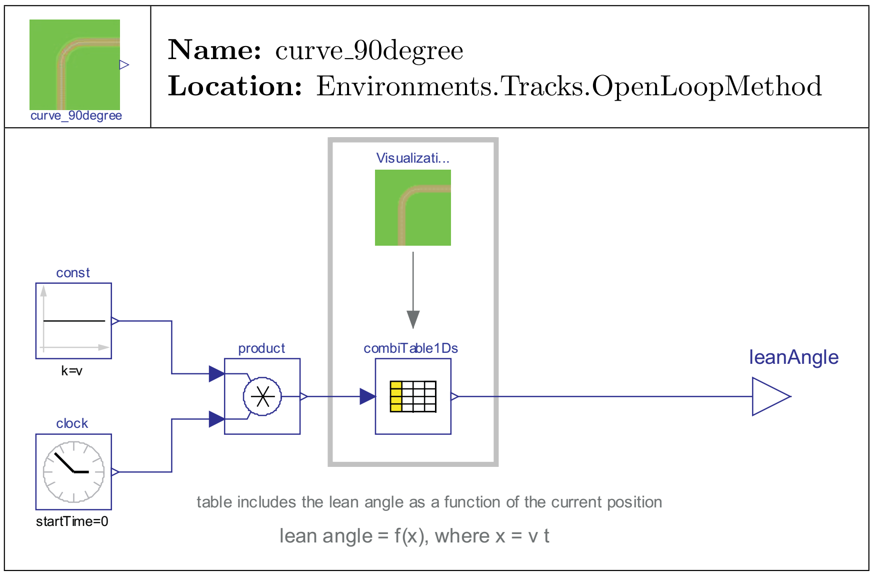
Figure 6.13: Wrapped model of the virtual movable rider composed of a state-space controller. The inputs (blue) are the lean and steer angle, lean angle set-value, velocity, rider's lean angle and the set-value of rider's lean angle, the outputs (white) are the steering torque and the torque applied by the upper body of the rider.

6.1.4 Optimal Control - LQR Design

In Chapter 5.4, a controller based on an LQR that stabilizes the motorcycle was developed. The structure of the virtual rider model is basically the same as the one presented in Figure 6.11 except that the state-space controller is replaced by the LQR introduced in Figure 5.22. As always, the main task of the controller is to stabilize the motorcycle and simultaneously track a roll angle profile. Thus, the weightings of R and Q have to be modified until the results are suitable.

6.1.5 Environments

Several test tracks are provided in order to validate the motorcycle's performance. In the following, the model of a 90°-curve is explained. The wrapped model is shown in Figure 6.14. The actual position of the vehicle is the product of *velocity* and *actual*



Name: curve_90degree
Location: Environments.Tracks.OpenLoopMethod

curve_90degree
Visualizati...
const
k=v
product
combiTable1Ds
leanAngle
clock
startTime=0
table includes the lean angle as a function of the current position
lean angle = f(x), where x = v t

Figure 6.14: Wrapped model of a 90°-curve

simulation time. The *Visualization* model includes the picture of the 90°-curve. The dimensions of the curve are shown in Figure 6.15.

The lean angle profile data for a specific vehicle are fed into the table. In the first column of this table, the positional information (traveled distance) is stored. This can be done in two different ways. The data can be entered directly or by means of a `record`[1]. The second column includes the corresponding lean angles that are calculated

[1]A Modelica record is used to structure data of primitive data types (e.g. Real, Integer, String, etc.) It is similar to a **struct** in C or MATLAB

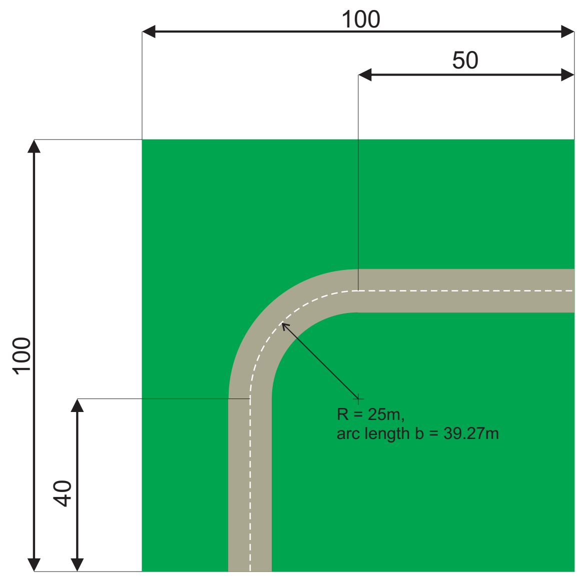


Figure 6.15: Dimensions of the 90°-curve

with reference to Equation 2.5. In Table 6.1, the lean angle profile of a 90°-curve for the basic motorcycle is listed.

Path $x = v \cdot t$ [m]	**Lean Angle** ϕ [°]
0	0
50 - 5	0
50 + 1	$atan\left(\frac{v^2}{R*g}\right)$
51 + 39.27 - 5	$atan\left(\frac{v^2}{R*g}\right)$
50 + 39.27 + 1	0
100	0

Table 6.1: Lean angle profile of a 90°-curve valid for the basic motorcycle model

The results of a motorcycle tracking a 90°-curve are shown in Chapter 7.2.

6.2 Path Tracking

In order to follow a pre-defined path, once again the controller has to be modified. Figure 6.16 shows the basic structure of a path preview controller. The controller is

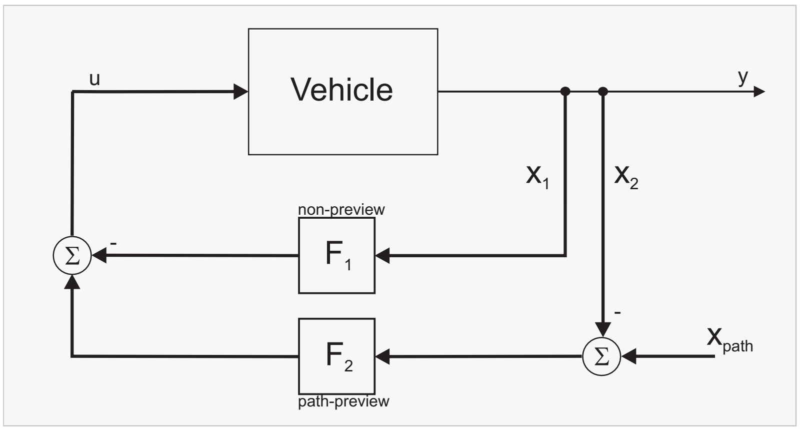


Figure 6.16: Basic structure of a path preview controller

composed of a *non-preview* and a *path preview* part. The former one is responsible for the motorcycle's stability. The latter one is an additional controller which constraints the vehicle to follow a pre-defined path. The corresponding control law results in:

$$u = -\begin{pmatrix} F_1 & F_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{6.4}$$

Where x_1 and x_2 are state vectors. The elements of x_1 depending on the particular controller design, e.g. $x_1 = (\phi\ \dot{\phi})^T$. In the simplest case, x_2 is the lateral position x_{lat} a pre-defined distance $x_{preview}$ ahead of the vehicle (see Figure 6.17). The path (roadway) is defined by a lateral profile i.e. the x-direction is fixed.

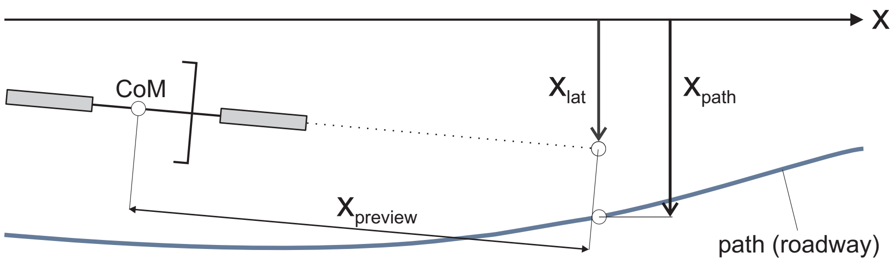
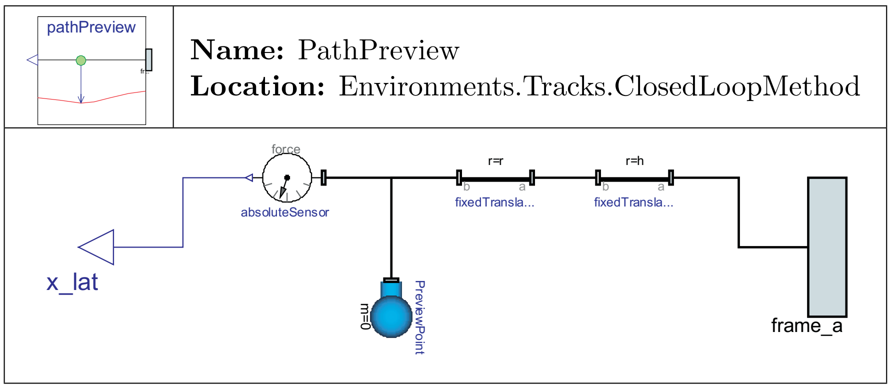


Figure 6.17: Single-point path preview

Such a path preview controller is determined with reference to several different approaches. To this end, three different approaches are introduced. Firstly, the controller designed in Chapter 5.9 is extended by a PI-controller. Secondly, a state-space controller is developed. Thirdly, an approach based on optimal control is introduced. At the end of this chapter, an alternative approach to single-point path preview, namely multi point path preview is mentioned.

To emulate a real human rider, *single-point path preview* is performed, i.e. the rider looks a pre-defined distance ahead. The path preview model is shown in Figure 6.18. Since the path preview model is connected to the rear frame's center of mass, its height



Name: PathPreview
Location: Environments.Tracks.ClosedLoopMethod

Figure 6.18: Wrapped path preview model

has to be subtracted in order to measure the correct lateral distance. The second *fixed translation* element is used to define the preview distance. The lateral distance is measured with a positional sensor. The *simple mass* element (m = 0) is used to indicate (visualize) the preview distance the rider is looking ahead.

6.2.1 Path Generation

The path generation is done with MATLAB. It is based on an approach introduced by R. S. Sharp and V. Valtetsiotis [SV01]. Basically, a random signal is generated. Afterwards, the offset is eliminated. In the next step, the signal is filtered in such a way that the result is a smooth path. A typical path is shown in Figure 6.19. The MATLAB m-file can be found in Appendix C.4.

This path is stored in a *mat-file* which is fed into the model shown in Figure 6.20. The traveled path of the motorcycle is calculated via $x = v \cdot t$. In other words, the actual position of the vehicle is the product of velocity and actual simulation time. This position is fed into the *Position* table. Thus, for each actual position of the vehicle, the corresponding lateral deviation is provided by the table. Regarding Figure 6.19, the model is just valid for very small lateral deviations (y-values) since the first column of the table includes the values of the fixed x-axis. Instead of these values, the traveled path has to be calculated. Since the values in the m-file are discrete, the length of each path element is calculated and summed up in order to get the real traveled path (Figure 6.21).

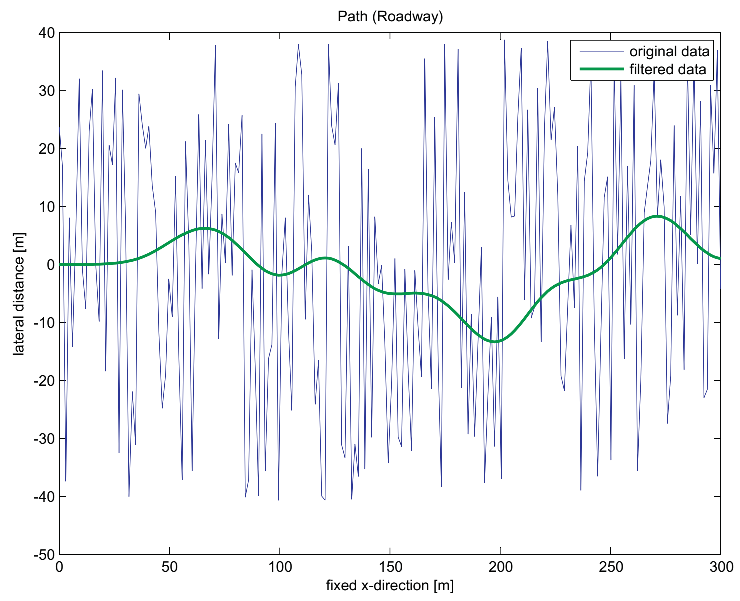


Figure 6.19: Path generated with MATLAB

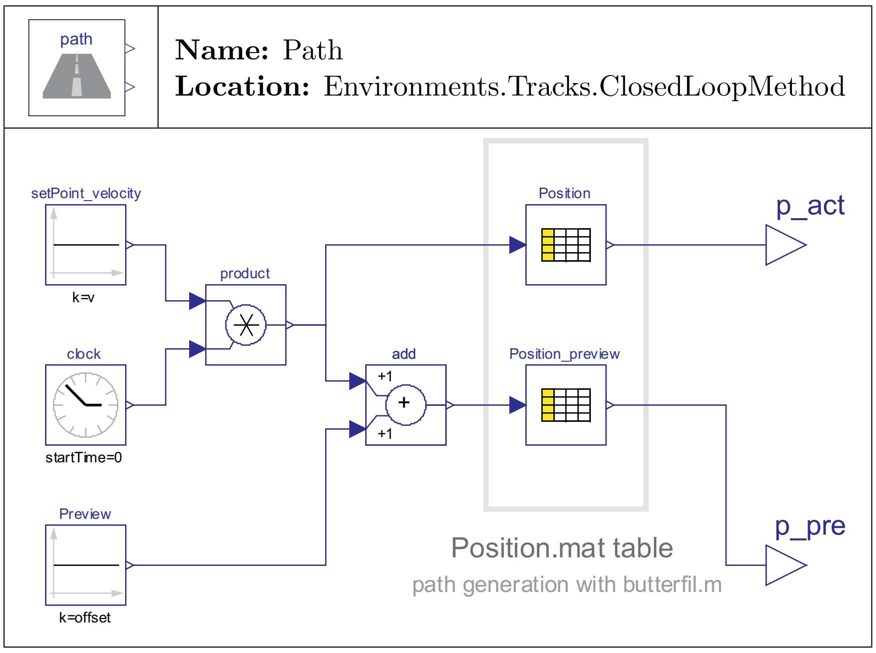


Figure 6.20: Wrapped path model

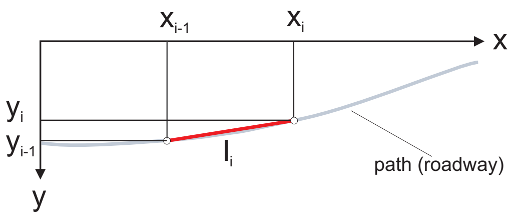


Figure 6.21: Calculation of the traveled path of the vehicle

According to Figure 6.21, the length of a single element is given by

$$l(i) = \sqrt{(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2} \tag{6.5}$$

Hence, the traveled path results in

$$path = \sum_{i=0}^{n} l(i) \tag{6.6}$$

By substituting the fixed x-values with the real path-values, the error is eliminated. Regarding the path preview performed by the rider, an offset is added to the actual position of the vehicle. The value of the offset is equal to the preview distance the rider is looking ahead. The position of the path preview point is fed into a second table called *Position_preview*. The output of this table serves as input for the virtual rider.

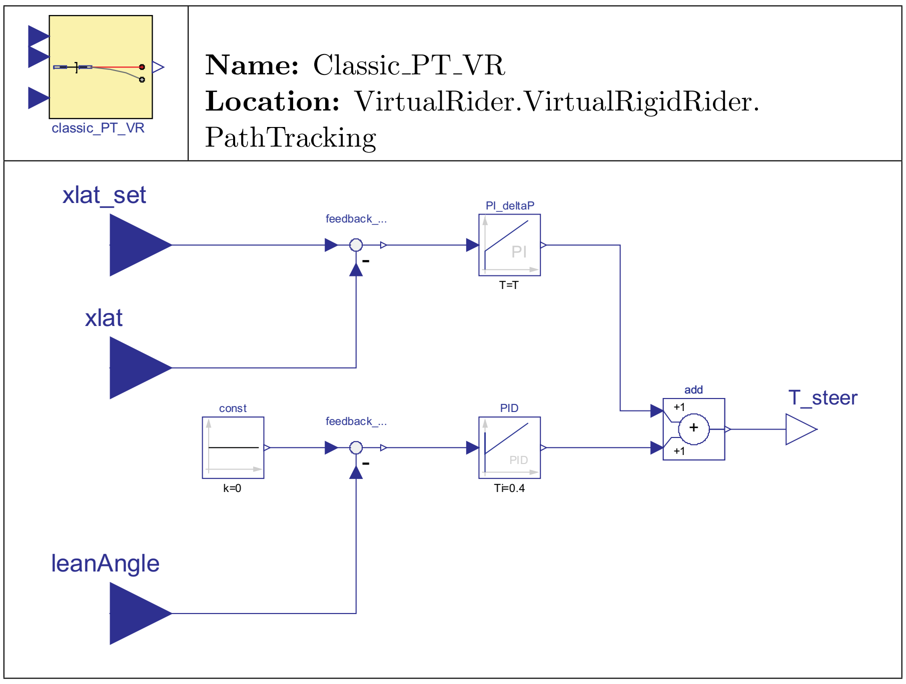
6.2.2 Classic Design

According to Figure 6.16, the model of a simple path preview controller composed of a PID stability controller and a PI controller to cover the path preview purposes is depicted in Figure 6.22.

The lean angle set-value is set to zero to ensure that the vehicle stays upright. The coefficients of the PID and the PI-controller have to be chosen such that the vehicle follows the pre-defined path without capsizing.

6.2.3 State-Space Design

For the sake of completeness, Figure 6.23 shows the basic structure of a state-space path preview controller.



Name: Classic_PT_VR
Location: VirtualRider.VirtualRigidRider.
PathTracking

Figure 6.22: Wrapped model of a simple path preview controller. The inputs (blue) are the path, lateral position and lean angle, the output (white) is the steering torque

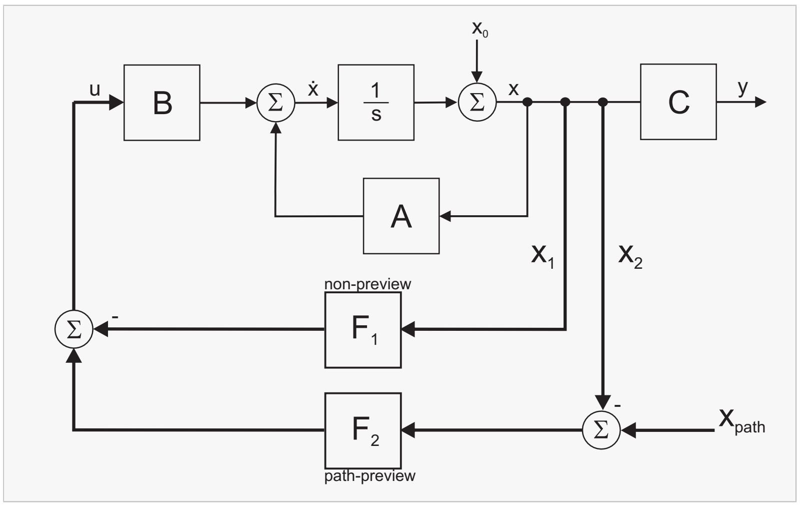
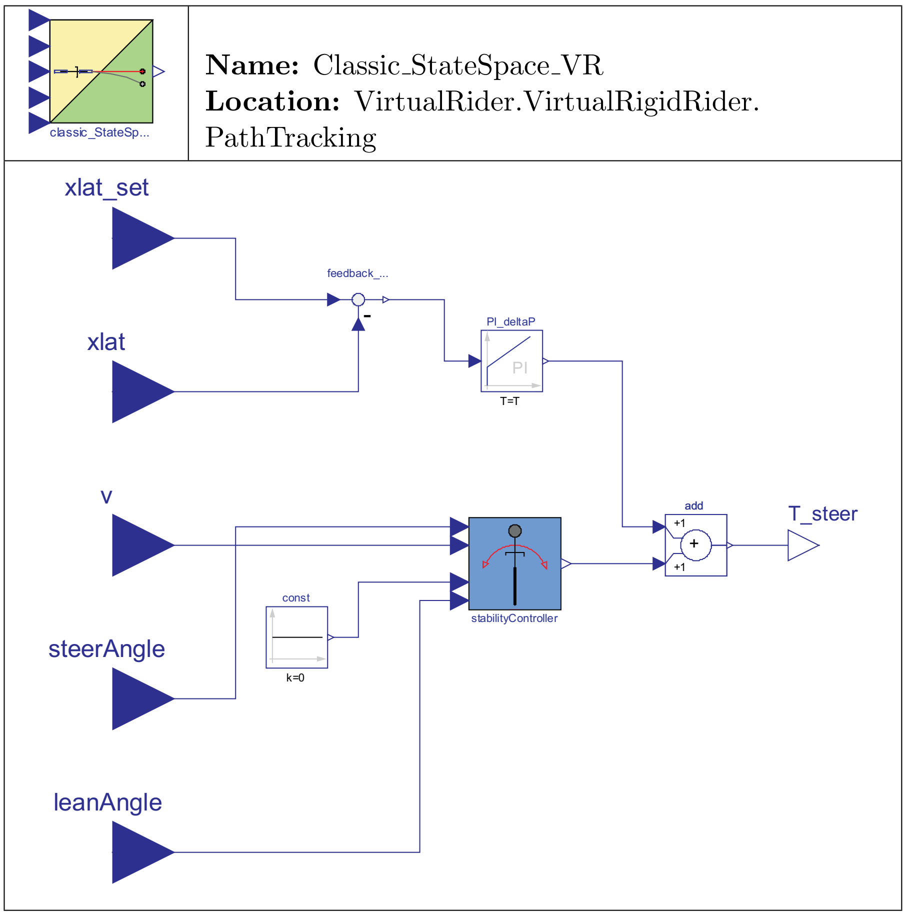


Figure 6.23: Basic structure of a state-space path preview controller

Combination of a State-Space and a Classic Controller

In a first approach a PID-controller was used to ensure that the motorcycle stays upright. Now, the PID controller is replaced by a state-space controller based on a preceding eigenvalue analysis. This is conform to the model shown in Figure 6.8. The wrapped model of the virtual rider is shown in Figure 6.24.



Name: Classic_StateSpace_VR
Location: VirtualRider.VirtualRigidRider.
PathTracking

Figure 6.24: Wrapped model of a state-space path preview controller. The inputs (blue) are the path, lateral position, velocity, steer angle and lean angle, the output (white) is the steering torque

Full State-Space Design

So far, two virtual riders capable of tracking a pre-defined path were developed. The first was composed of classic elements. For the second one, the PID-controller was replaced by a state-space controller (see Figure 6.8) due to the convenient physical interpretation of the poles. For the following virtual rider, the “path preview” controller is based on a state-space controller as well. The two state vectors needed to design the controller are thus: $x_1 = (\delta\ \dot{\delta}\ \phi\ \dot{\phi})^T$ and $x_2 = (x_{lat}\ \dot{x}_{lat})^T$. The state vector x_2 includes the lateral position and lateral rate. In order to measure the lateral rate, the path preview model has to be modified. Figure 6.25 depicts the necessary modifications for such a model. The lateral acceleration is measured with a sensor and is integrated twice.

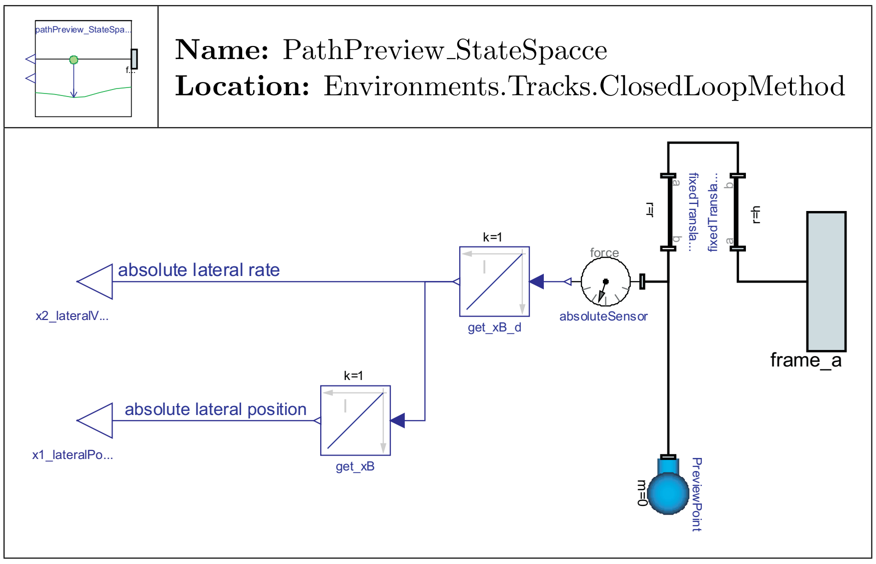
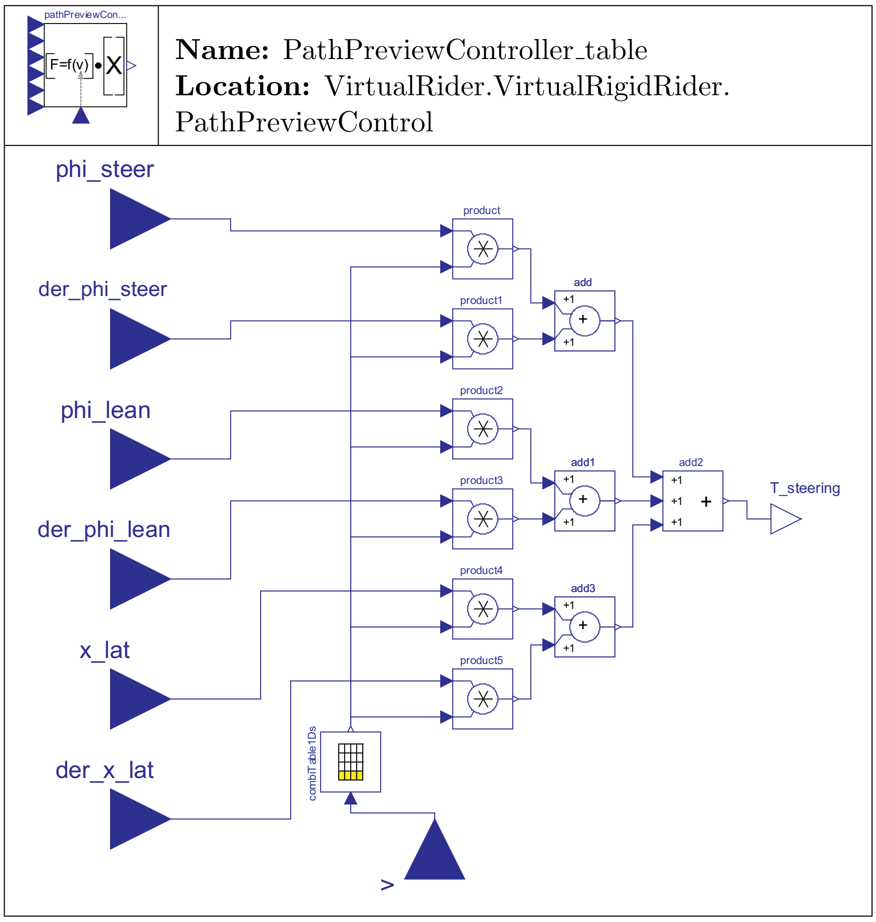


Figure 6.25: Wrapped path preview model for state-space design. The model is connected to the rear frame’s center of mass

Thus, the output of the first integrator introduces the lateral rate, and the output of the second one, the lateral displacement (position). Instead of using two integrators to get both the lateral position and the lateral rate, they could be measured directly by means of a position and a velocity sensor. However, in order to develop a controller, the elements of the state-vector x_2 have to be state variables. Fortunately, each output of the integrators of the path preview model is a state variable by default. To design a controller, the model of the uncontrolled motorcycle shown in Figure 5.7 is extended by the path preview model.

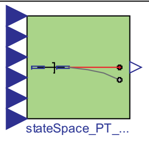
The library includes both a virtual rider valid for a specific velocity and another one valid for a specific velocity range. The calculation of the state feedback matrix coefficients is done with the `place_pathPreview` or the `placeRange_pathPreview` function. Contrary to the function `place_Range_offset`, an additional offset `offset_lat` is included. This offset is used to change the location of the poles of the state-vector x_2.

To be able to track a pre-defined path, the controller introduced in Figure 5.14 has to be modified such that the state vector x_2 is incorporated. The modified wrapped model is shown in Figure 6.26. This model is now integrated into the virtual rider shown in



Name: PathPreviewController_table
Location: VirtualRider.VirtualRigidRider.
PathPreviewControl

Figure 6.26: Wrapped model of a state-space path preview controller valid for a specific velocity range



Name: StateSpace_PT_VR_Range
Location: VirtualRider.VirtualRigidRider.
PathTracking

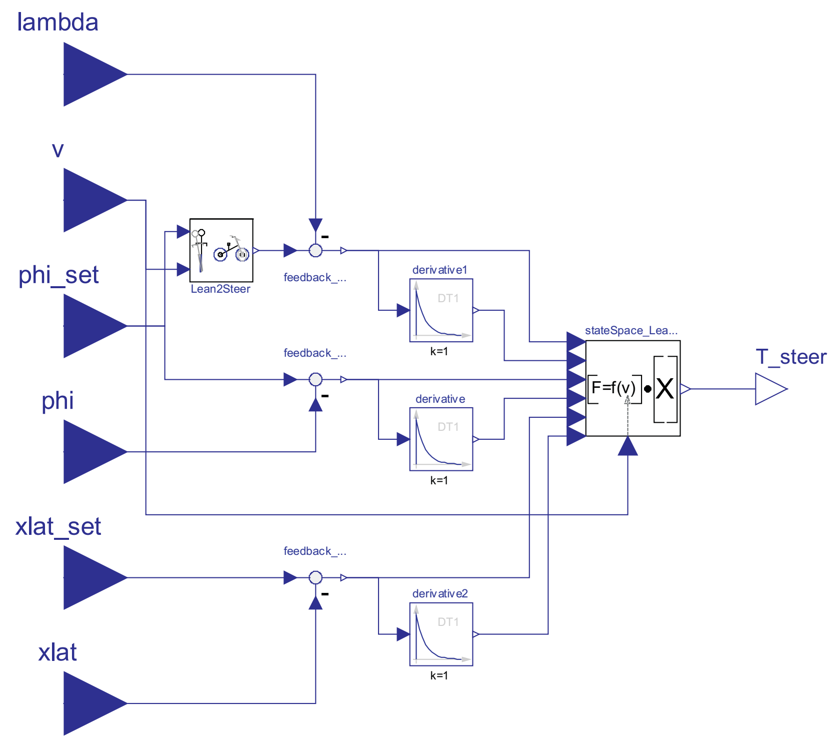


Figure 6.27: Wrapped model of a virtual rider composed of a state-space controller capable of stabilizing the motorcycle and tracking a pre-defined trajectory. The model is valid for a specific velocity range. The inputs (blue) are the lean angle set-value, lean angle, steer angle, path, lateral position, and velocity, the output (white) is the steering torque

Figure 6.27.

A detailed example can be found in Chapter 7.2.

6.2.4 Optimal Preview Control - LQR Design

The design of a virtual rider capable of tracking a pre-defined trajectory can be done in the same manner as in Chapter 5.4. Again, the main task is to find the coefficients of the weighting matrices R and Q such that the vehicle follows the pre-defined path without capsizing. The system has still one input and hence R is again a scalar. The size of Q is increased by (2×2) due to the state-vector x_2. Hence, Q is a (6×6)-matrix. In a first approach, let us minimize the lateral deviation x_{lat}, the lean angle ϕ and the control energy (steering torque) u.

Thus

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \delta \\ \dot{\delta} \\ \phi \\ \dot{\phi} \\ x_{lat} \\ \dot{x}_{lat} \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \\ \\ \leftarrow min\ \phi \\ \\ \leftarrow min\ x_{lat} \\ \\ \end{array}$$

and $R = 1 \quad \leftarrow min\ u$

The controller design is done with MATLAB. Basically, the m-file is similar to the one presented in Chapter 5.4 except that Q is a (6×6)-matrix. The wrapped model of the virtual rider composed of an LQR is quite similar to the model shown in Figure 6.27. Instead of a state-space controller valid for a specific velocity range, an LQR valid for a specific velocity is implemented. The model of this controller has the same structure as the one shown in Figure 5.22. Additionally, the states x_{lat} and $\dot{x}_{lat}$ have to be taken into account. The model used for the simulation is shown in Figure 6.28. The simulation result for a randomly generated path, a preview distance of 6m and a forward velocity of $6ms^{-1}$ is depicted in Figure 6.29.

The simulation result in Figure 6.29 shows that the virtual rider is too slow in order to follow the path. Furthermore, the lateral deviation of the rider is higher than the lateral profile of the path. Thus, the weighting coefficient $Q(5,5)$ has to be increased in order to minimize the path error. Let us set $R = 0.1$ and $Q(5,5) = 100$. The result is shown in Figure 6.30.

Figure 6.30 shows the improved result. The lateral deviation of the motorcycle is now even lower than the lateral profile of the path. Obviously, the rider was not too slow. It was rather a wrong weighting of R and Q. Thus, let us again set $R = 1$ in order to

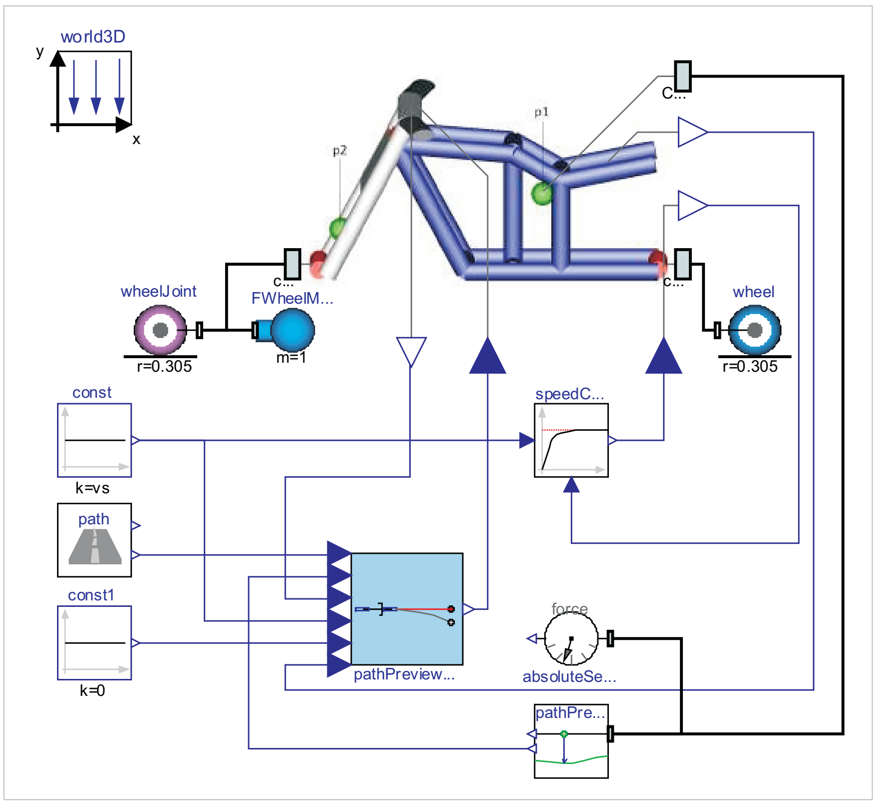


Figure 6.28: Model of the basic motorcycle with a virtual rider composed of an LQR

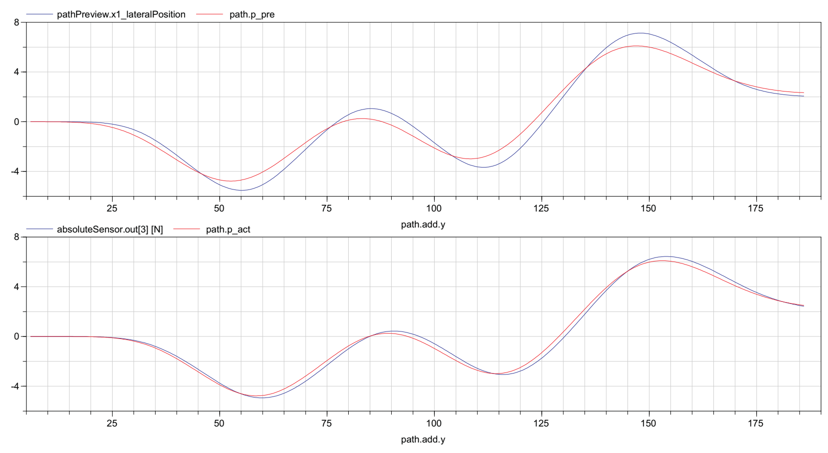


Figure 6.29: Simulation result of the basic motorcycle model with a virtual rider composed of an LQR. Upper plot: The red signal is the path a pre-defined distance ahead, the blue signal is the preview distance of the rider. Lower plot: The red signal is the actual path, the blue signal is the traveled path measured at the rear frame's center of mass

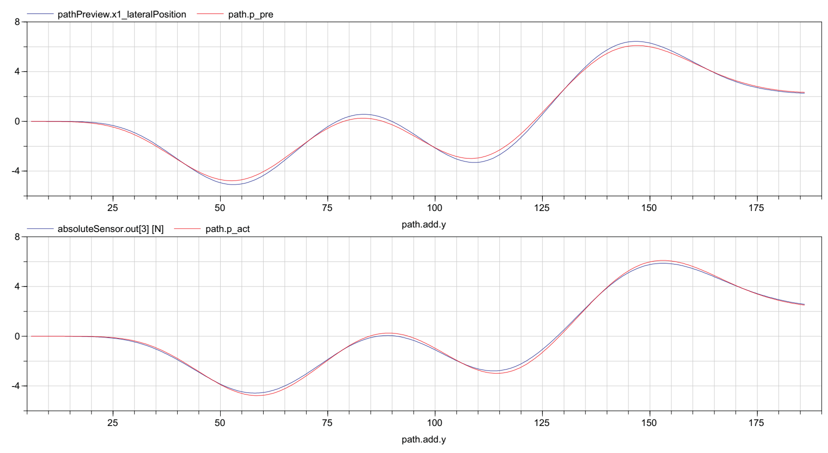


Figure 6.30: Improved simulation result of the basic motorcycle model with a virtual rider composed of an LQR

reduce the applied steering torque. The response of the rider caused by the weighting coefficient $Q(5,5) = 1$ was too low whereas $Q(5,5) = 100$ was already too fast. Thus, let us set $Q(5,5) = 10$. The improved results are shown in Figure 6.31. Out of this simulation results it can be seen that the rider is now perfectly tracking on the path without any deviation.

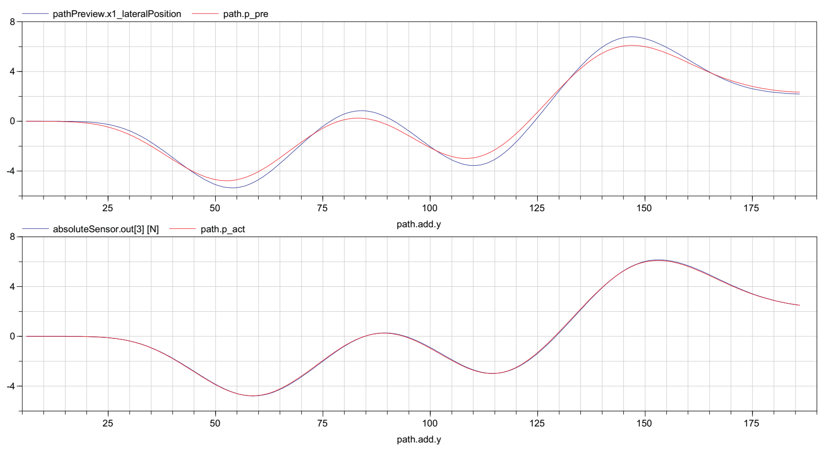


Figure 6.31: Another improved simulation result of the basic motorcycle model with a virtual rider composed of an LQR

In this approach, the lateral position and its derivative were taken into account. Additionally, maybe one wishes to minimize the attitude angle (yaw angle) differences. In order to do this, the angle of the current path has to be calculated. Afterwards, this angle is compared with the yaw angle of the motorcycle and minimized by appropriate weighting coefficients. Thus $x_2 = (x_{lat}\ \dot{x_{lat}}\ \psi\ \dot{\psi})^T$.

Remark

The weightings have to be adjusted until the results are suitable. Without any knowledge regarding linear control theory, this task is laborious and time consuming. Compared to classic minimization problems, optimal control is different. It is not possible to minimize both the state variables (performance of the controller) and the control energy. It is rather a trade-off between control energy and performance.

6.2.5 LQR Design for Models Composed of Movable Riders

The design of such a controller is carried out in exactly the same way as shown in Section 6.2.4. Of course, R is again increased to size (2×2), whereas Q becomes a

(8×8) matrix. The state-space controller and the corresponding virtual rider are both included in the library.

6.2.6 Optimal Preview Control with Multi-Point Preview

To emulate the behavior of a human rider, single-point preview is performed by the virtual rider. To improve the quality of a virtual rider, multi-point preview has to be implemented. This means that the rider gathers knowledge of the whole lateral profile of the path (see [SV01] on page 102 and 105, respectively).

R. S. Sharp is very experienced in this area. In one of his first papers regarding optimal preview control, he implemented a virtual car driver capable of tracking a pre-defined path by means of multi-point preview [SV01]. In the paper [Sha07a], he adapted the optimal preview control theory to the benchmark bicycle introduced by Schwab et al. [SMP05]. Furthermore, at about the same time, he applied the theory to a motorcycle [Sha07b]. The motorcycle itself is nothing other than the improved version of his and D. J. N. Limebeer's SL2001 model, presented in [SEL04].

However, due to a "lack of time" this approach is not implemented in the library.

7 Examples

The purpose of this chapter is to demonstrate the usage of the *MotorcycleLib*.

7.1 Benchmark Bicycle

7.1.1 Rigid Rider

Let us start with a simple example of an uncontrolled 3 d.o.f. bicycle model (see Figure 7.1. A “double-click” on the model opens the parameter window in Figure 3.5.

Name: uncontrolled_3dof_bicycle
Location: Examples.BenchmarkBicycle.RigidRider

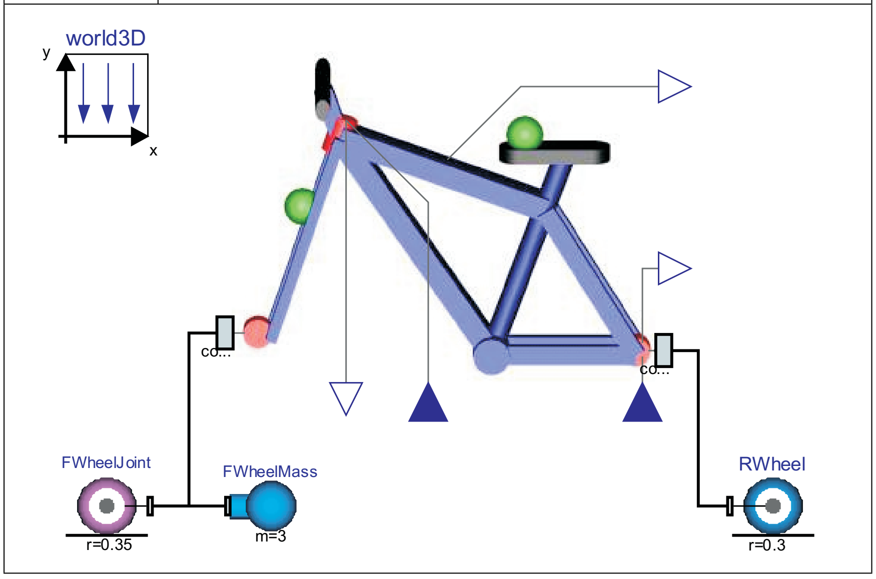


Figure 7.1: Example: uncontrolled 3 d.o.f. benchmark bicycle

Now, the vehicle specific parameters are entered. For this example, the parameters of Table 3.1 were used. For the velocity of the bicycle, let us create a global variable.

```
import SI = Modelica.SIunits;
parameter SI.Velocity vs = 3;
```

The value of the global variable `vs` is now set as the initial velocity of the rear wheel. Again, a “double-click” on the rear wheel model opens the parameter window. In order to define the initial conditions, one has to click on the initialization tab. Now several input-fields appear. The global variable `vs` is entered into the *w_start* field. Since the input has to be an angular velocity, the following values are entered:

```
vs*{0,0,1/0.3*180/Modelica.Constants.pi}
```

The factor $\frac{1}{0.3}$, where 0.3 is the rear wheel's radius, is used to calculate the corresponding angular velocity ($\omega = \frac{v}{r}$). The factor $\frac{180}{\pi}$ is needed to convert radians into degrees. In the next step the model is simulated. Figure 7.2 shows the animation result for a velocity of $3ms^{-1}$.

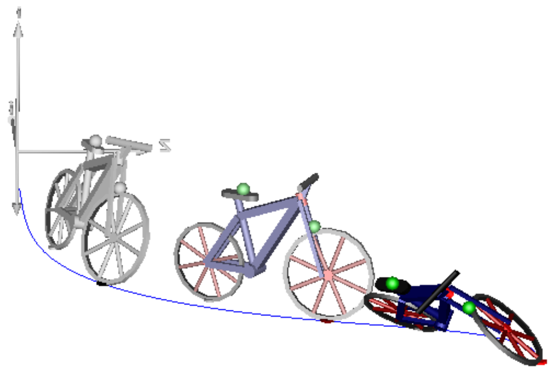


Figure 7.2: Animation result of the uncontrolled 3 d.o.f. benchmark bicycle. After about 2s the vehicle falls over like an uncontrolled inverted pendulum.

In the next step we perform an eigenvalue analysis to determine the self-stabilizing area. Therefore the function `stabilitAnylysis` (Examples.BenchmarkBicycle.RigidRider) is executed. This is done with by click on the right mouse button in the package browser. Now, the window in Figure 4.2 appears. All the other steps needed in order to get the result were already performed in Chapter 4. The result of the eigenvalue analysis is depicted in Figure 4.4.

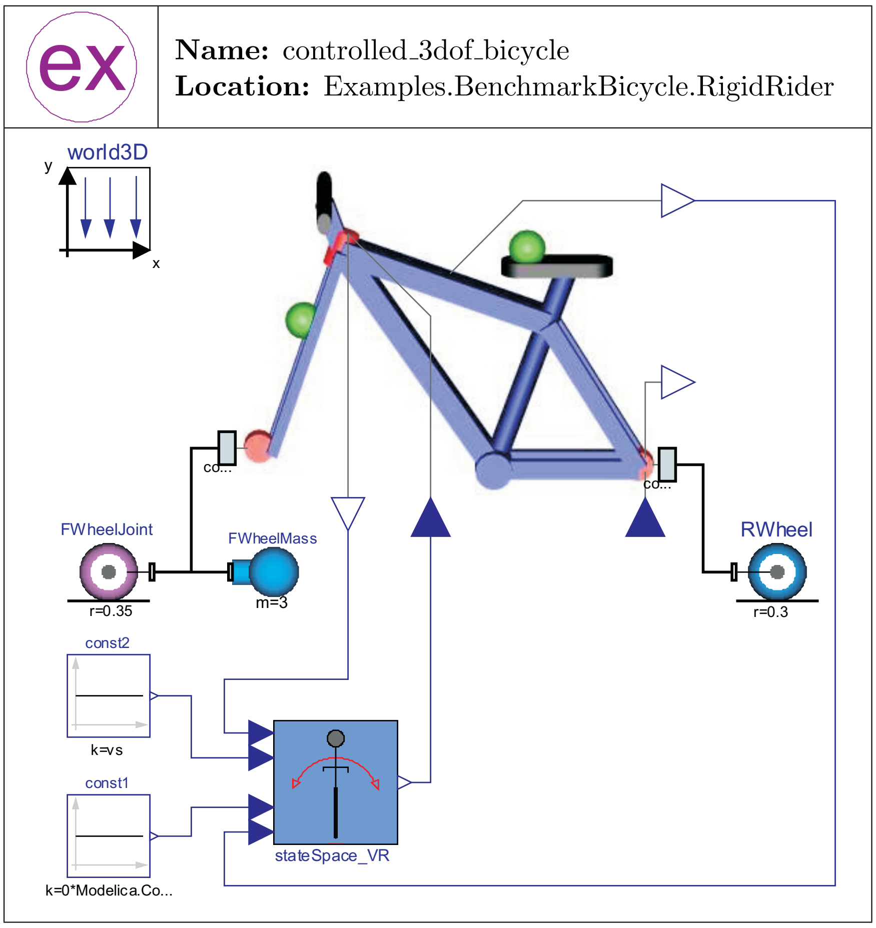
In the next step let us calculate the coefficients of the state feedback matrix for a specific velocity. For this purpose the `ControllerDesign`[1] function (Examples.BenchmarkBicycle.RigidRider) is executed. According to the results of the eigenvalue analysis, it

[1]Every function in the example sub-package includes a function call of the appropriate function.

can be seen that the vehicle is truly unstable for a velocity of $3ms^{-1}$. In the following, the vehicle is stabilized for a velocity of $3ms^{-1}$. Out of Figure 4.4, one can see that an offset greater than 1.7 is sufficient in order to achieve stable behavior. Let us cover the behavior of a cautious rider and thus set the value of the offset equal to 2. The higher the offset, the more aggressive the rider. Finally, the state variables are entered. If they are not similar to the default values, the function `getStates` (Utilities) should be executed in order to get the appropriate states. The output of the function call is displayed in the command window.

```
[15.9523615979054, 1.81106337719821, 21.5057512252129, 2.1654011773943]
```

The model of the controlled 3 d.o.f. bicycle is shown in Figure 7.3. To simulate the



Name: controlled_3dof_bicycle
Location: Examples.BenchmarkBicycle.RigidRider

Figure 7.3: Example: controlled 3 d.o.f. benchmark bicycle

model, one has to copy the state feedback matrix (displayed in the command window) and insert it into the controller. Figure 7.4 shows the animation result for a velocity of $3ms^{-1}$ and an initial lean angle of 5°.

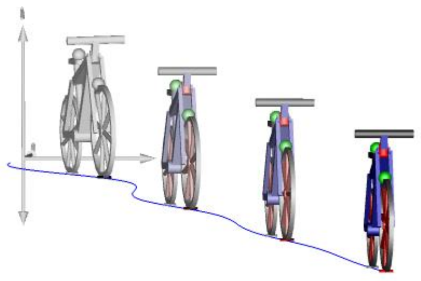


Figure 7.4: Animation result of the controlled 3 d.o.f. benchmark bicycle

Regarding the animation result, it can be seen that an offset $d = 2$ corresponds to a slow (cautious) rider. For an offset equal to $d = 5$, the state feedback matrix coefficients become:

```
[37.4451900093418, 4.01922050142088, 93.1718854159252, 23.3342372363612]
```

A comparison of the controller performance and the appropriate control energy (steering torque) for two different offsets is depicted in Figure 7.5. The results are as expected.

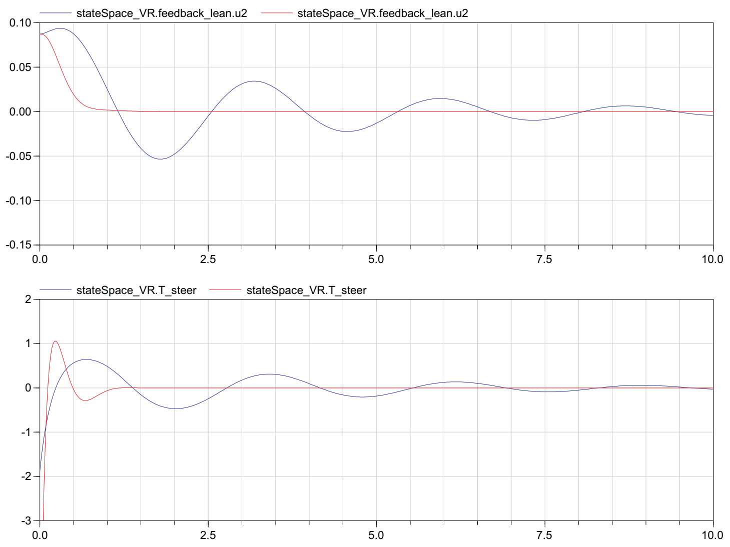


Figure 7.5: Comparison of the controller performance and the appropriate control energy (steering torque) for two different offsets: the blue signals are valid for an offset d = 2, whereas the red signals are valid for d = 5.

The more offset, the more steering torque is needed and the faster the system.

7.1.2 Movable Rider

For the second example, the same steps as in Example 7.1 are performed. Due to the absence of a MIMO pole placement function in the library, the calculation of the state feedback matrix is done with MATLAB. The corresponding *m-file* can be found in Appendix C.2. The model of the uncontrolled 4 d.o.f. bicycle can be found in Examples.BenchamarkBicycle.MovableRider. The additional input `RiderTorque` is needed for the state-space controller design. The results of the eigenvalue analysis are shown in Figure 4.5. Contrary to Example 7.1, an offset greater than 1.9 is required to ensure stable behavior. Again, an offset of 2 is used. A second offset `offset_rider` is needed to move the additional eigenvalues introduced by the movable upper body of the rider. According to Figure 4.5 an offset greater than 3.11 is needed to ensure stable behavior. Let us set the offset equal to 4.

Now, the model of the uncontrolled 4 d.o.f. bicycle is linearized for a velocity of $3ms^{-1}$ (see Chapter 5.2.2 for the model linearization). The eigenvalues for a velocity of $3ms^{-1}$ are:

```
p =

 -10.5364
   3.1144
   1.8988 + 2.3000i
   1.8988 - 2.3000i
  -3.5115
  -2.4742
```

The state feedback matrix is calculated by means of the following lines of code:

```
  offset = 2;
  offset_rider = 4;
  p1 = p(1) - offset;
  p2 = p(2) - offset_rider;
  p3 = p(3) - offset;
  p4 = p(4) - offset;
  p5 = p(5) - offset;
  p6 = p(6) - offset_rider;

  poles = [p1, p2, p3, p4, p5, p6];

  disp('Controller Matrix F:');
  F = place2(Arel, Brel, poles)
```

The coefficients of the state feedback matrix result in:

```
F =
  16.5667    1.6496   21.8597    1.3027    0.6104    0.1844
 −79.5174  −11.4260 −109.7068    4.2119   65.8985   30.5641
```

In the next step, the state feedback matrix is inserted into the controller of the model shown in Figure 7.6. The animation results are depicted in Figure 7.7.

Name: controlled_4dof_bicycle
Location: Examples.BenchmarkBicycle.MovableRider

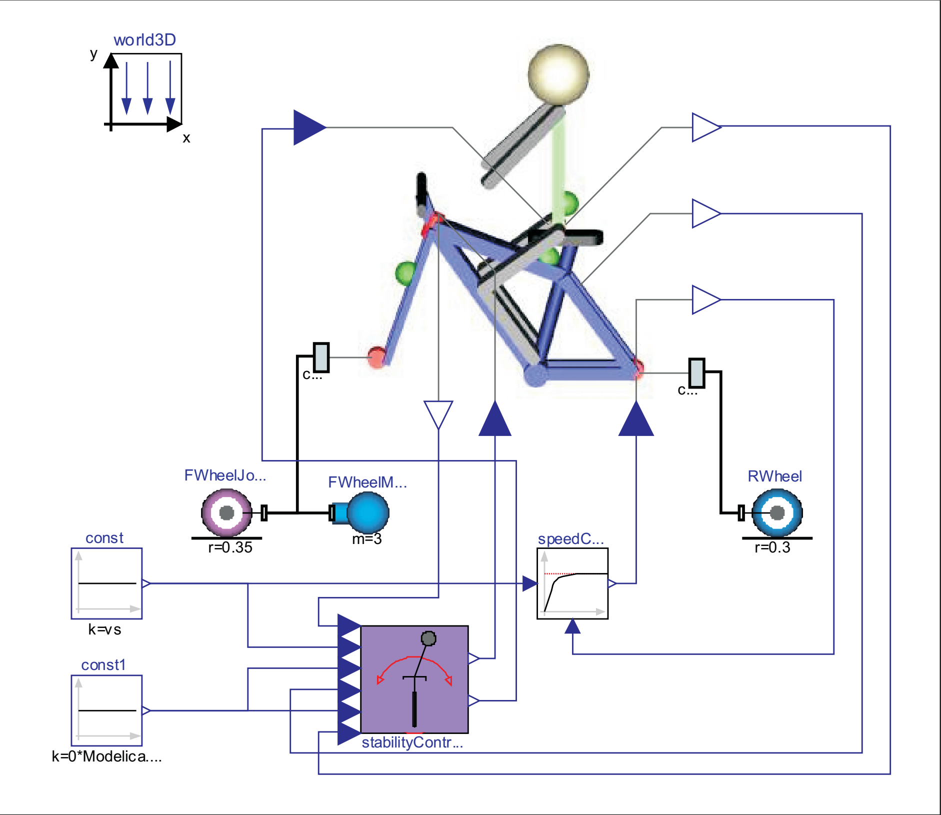


Figure 7.6: Example: controlled 4 d.o.f. benchmark bicycle

Remark

The controller design carried out in this example is just one possible approach. For the sake of simplicity, two different offsets were used. However, in an ideal scenario one should only move those eigenvalues that lead to unstable behavior.

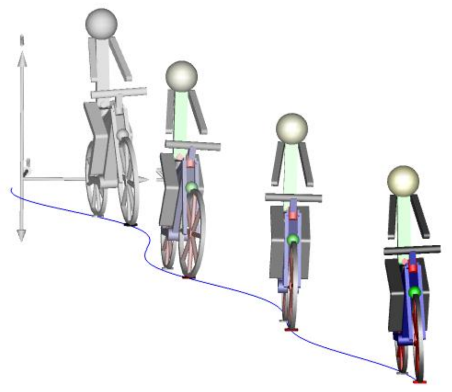


Figure 7.7: Animation result of the controlled 4 d.o.f. benchmark bicycle

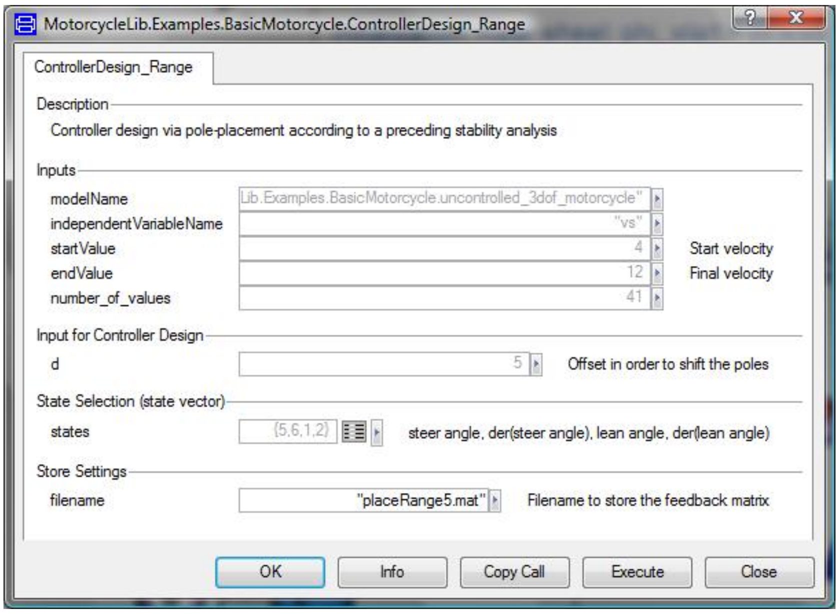
7.2 Basic Motorcycle

7.2.1 Roll Angle Tracking

The purpose of the third example is to track a roll angle profile of a 90°-curve with several different velocities. In the first step, an eigenvalue analysis is performed. This is done with the `StabilityAnalysis` function that can be found in Examples.BasicMotorcycle. The results are shown in Figure 4.7.

In the next step the feedback matrices (one for each velocity) are automatically calculated by means of the function `ControllerDesign_Range`. Let us calculate the feedback matrices within the range $4ms^{-1} \leq v \leq 12ms^{-1}$ and for an offset $d = 5$. If the function is executed, the window in Figure 7.8 appears. The result of the function is a matrix stored in the file `placeRange5.mat`. The matrix itself has 41 rows, where each row includes the coefficients of the state feedback matrix for the corresponding velocity. The filename is now inserted into the controller of the model shown in Figure 7.9.

Now, the roll angle profile has to be defined. The dimensions of the curve are shown in Figure 6.15. Fortunately, the parameters for the basic motorcycle model were already defined (see Table 6.1). The animation result for a motorcycle with a velocity of $v = 10ms^{-1}$ is illustrated in Figure 7.10. The corresponding signals are shown in Figure 7.11. In the middle plot of Figure 7.11 it can be seen that the rider steers into the opposite direction (countersteering) in order to enter a turn. The peaks in the steering torque signal (lower plot) are due to the ideal derivative blocks included in the virtual rider.


MotorcycleLib.Examples.BasicMotorcycle.ControllerDesign_Range

ControllerDesign_Range

Description

Controller design via pole-placement according to a preceding stability analysis

Inputs

modelName	Lib.Examples.BasicMotorcycle.uncontrolled_3dof_motorcycle"	
independentVariableName	"vs"	
startValue	4	Start velocity
endValue	12	Final velocity
number_of_values	41	

Input for Controller Design

d	5	Offset in order to shift the poles

State Selection (state vector)

states	{5,6,1,2}	steer angle, der(steer angle), lean angle, der(lean angle)

Store Settings

filename	"placeRange5.mat"	Filename to store the feedback matrix

Figure 7.8: Parameter window of the ControllerDesign_Range function

Remark

The peaks can be diminished if the ideal derivative blocks are replaced by real (approximated) derivative blocks. But one has to keep in mind that results of the controller design (e.g. Figure 5.16) are only plotted correctly with ideal derivative blocks. This is due to the time constant of the real derivative blocks.

7.2.2 Path Tracking

The path tracking capabilities of the basic motorcycle were already tested in Chapter 6.2.4. In order to design a virtual rider composed of a state-space controller based on a preceding eigenvalue analysis, the function `ControllerDesign_pathPreview` (Examples.BasicMotorcycle) has to be executed. The utilized model is depicted in Figure 7.12.

Again, the coefficients of the state feedback matrix were automatically calculated for an offset $d = 5$. Additionally, an offset $d_{lat} = 5$ is required in order to keep the motorcycle on the desired path. The results are shown in Figure 7.13.

Name: controlled_3dof_motorcycle_Curve
Location: Examples.BasicMotorcycle

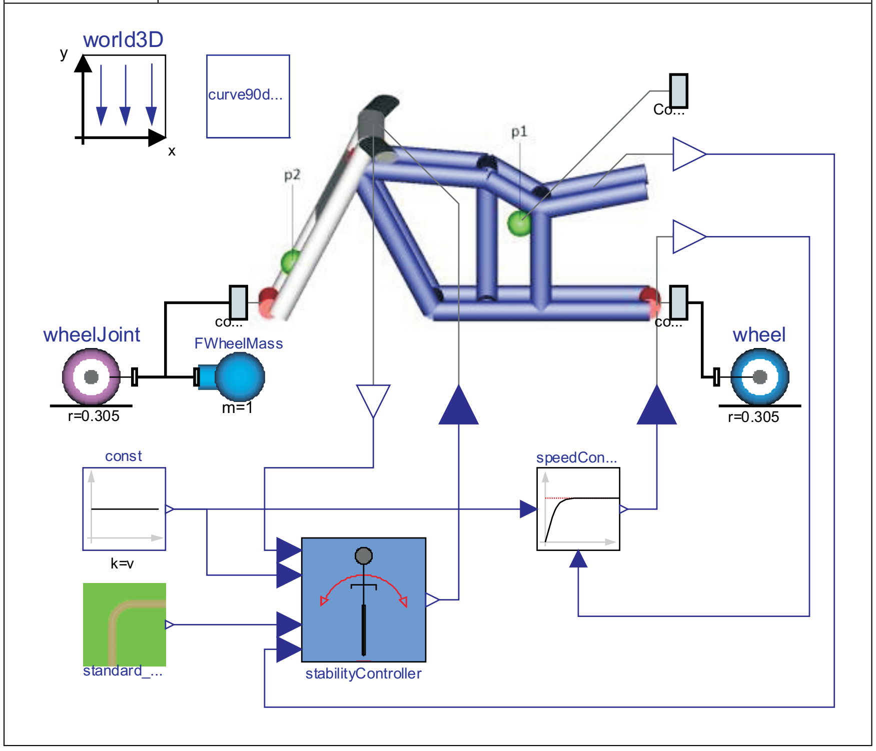


Figure 7.9: Example: controlled 3 d.o.f. motorcycle

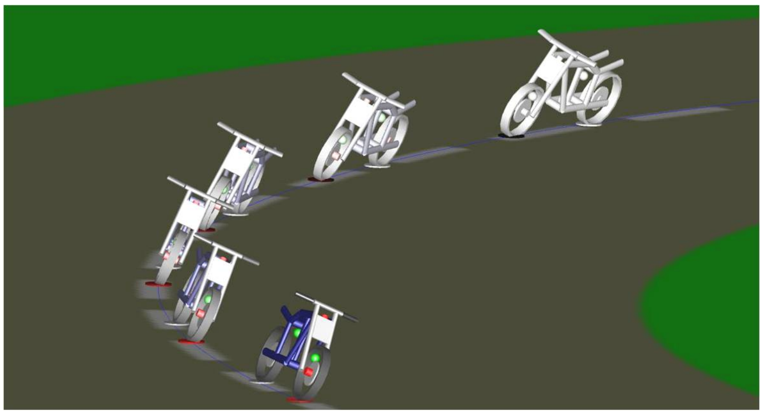


Figure 7.10: Animation result of the basic motorcycle tracking a roll angle profile with a velocity of $10ms^{-1}$

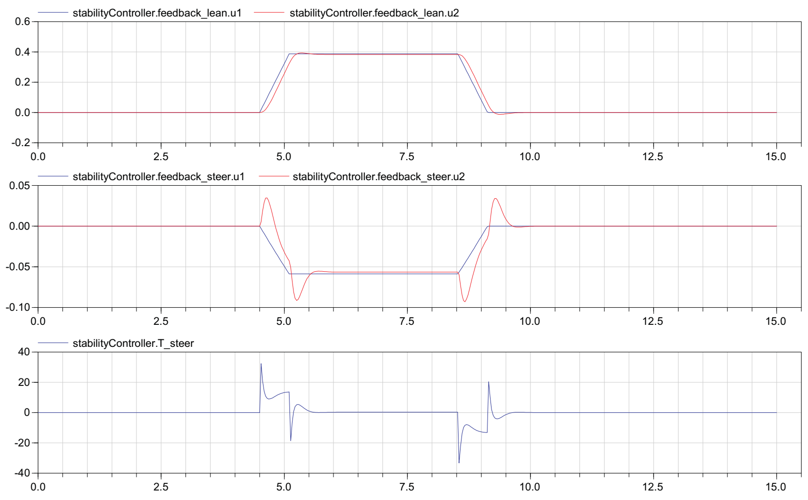


Figure 7.11: Signals of the controlled 3 d.o.f motorcycle. Upper plot: lean angle; middle plot: steer angle; lower plot: steering torque

7.3 Advanced Motorcycle

Finally, an example of an advanced motorcycle is presented. The model of an uncontrolled version of Sharp's improved motorcycle is shown in Figure 7.14. In this model, in-plane modes and aerodynamics are considered. The animation result for a velocity of $6ms^{-1}$ is shown in Figure 7.15.

As with the basic models, it is now possible to design a controller. This can be done in two different ways. In the simplest case, the rider solely applies a steering torque to the handle bars. Another possibility would be to consider both the steering torque and the upper body lean torque. Again, the controller design is based on a preceding eigenvalue analysis. Unfortunately, a Modelica function is not available. Therefore, this task has to be performed using MATLAB (refer to Chapter 4.4.1).

Name: controlled_3dof_motorcycle_closedLoop_Range
Location: Examples.BasicMotorcycle

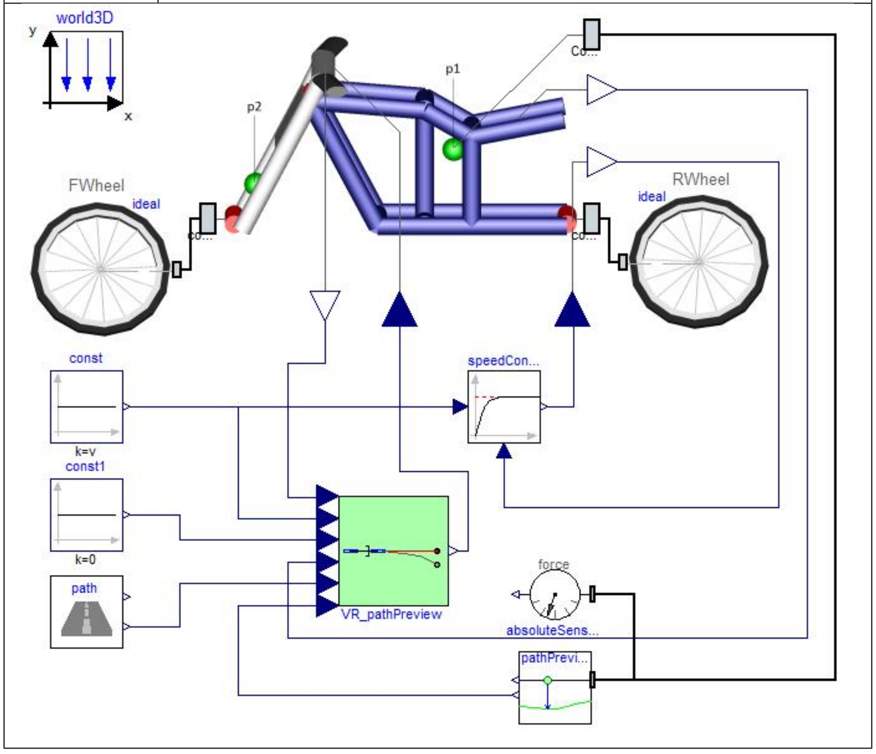


Figure 7.12: Example: model of a 3 d.o.f. motorcycle tracking a pre-defined path

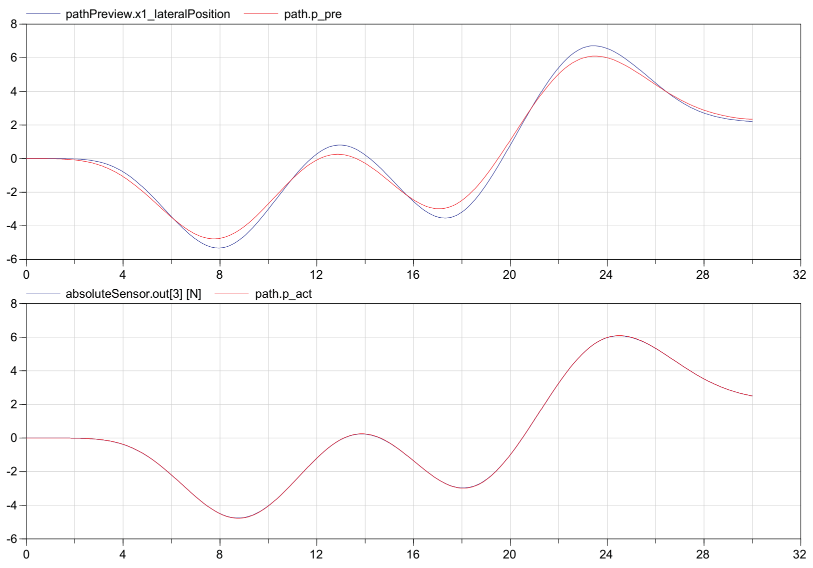


Figure 7.13: Simulation result of a motorcycle tracking a pre-defined path. Upper plot: The red signal is the path a pre-defined distance ahead, the blue signal is the preview distance of the rider. Lower plot: The red signal is the actual path, the blue signal is the traveled path measured at the rear frame's center of mass

Name: Suzuki_GSX_R1000
Location: Examples.AdvancedMotorcycle.Sharp2004

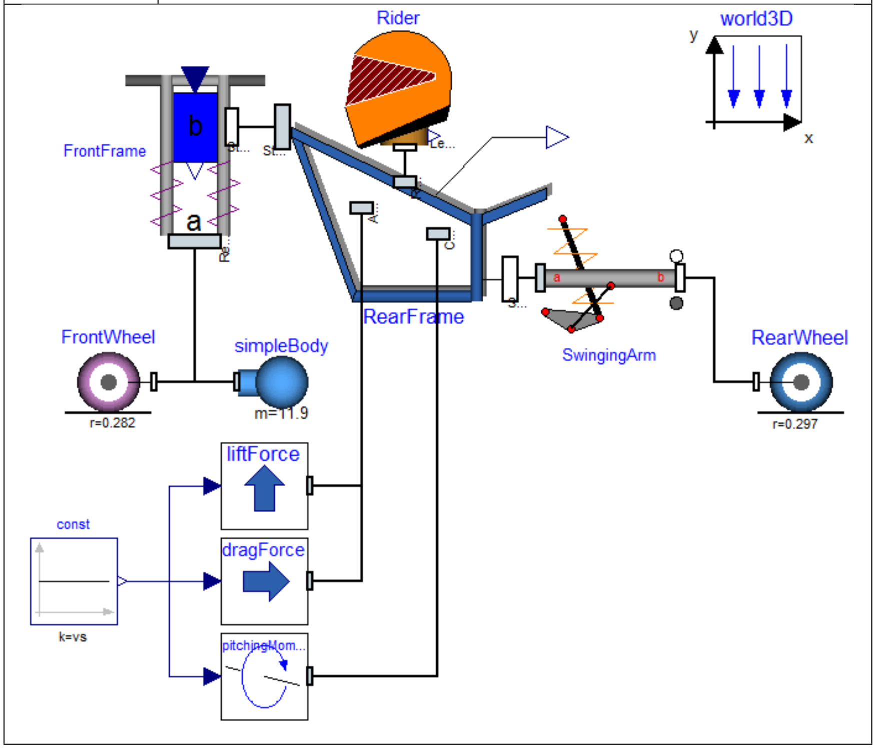


Figure 7.14: Example: uncontrolled advanced motorcycle model (Sharp's improved model)

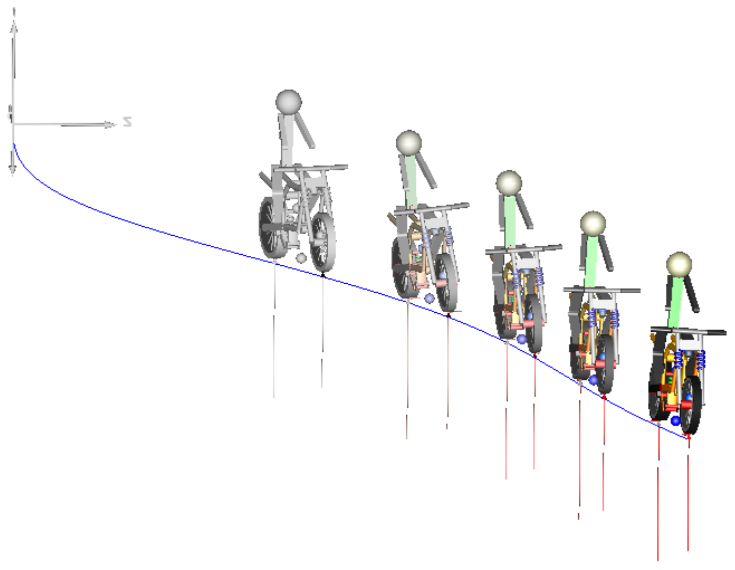


Figure 7.15: Animation result of an uncontrolled improved motorcycle

8 Conclusions

Chapter 2 of the Master's Thesis is intended to discuss the basics of bicycle and motorcycle dynamics. The purpose of this chapter is to provide a brief explanation of the interaction between gyroscopic effects and the geometry (e.g. the trail) of single-track vehicles. This knowledge is really essential to describe several possible motions of single-track vehicles and thus understand its stability behavior. It should be mentioned that not only the gyroscopic effects contribute to the stability; it is rather the trail, or more precisely the interaction of the whole geometry that is crucial for the stability.

In Chapter 3, several different single-track vehicle models are introduced. Basically, two different kinds of models are provided. Those that include out-of-plane modes and those that include both in-plane and out-of-plane modes. Roughly speaking, out-of-plane modes are related to stability and handling of single-track vehicles, whereas in-plane modes are dealing with riding comfort. Furthermore, it is important to keep in mind that models connected to ideal wheels include so-called holonomic constraints. Such constraints are based on location and, with single-track vehicles, prevent them from sinking into the ground. The wheels used in this library are either provided by D. Zimmer's *MultiBondLib* or by M. Andres' *WheelsAndTires* library. The former ones are ideal, whereas in the latter ones non-ideal effects such as slip can be considered. For the bicycle, either a 3 or a 4 degree of freedom (d.o.f.) out-of-plane model is included in the library. The former ones are composed of four rigid bodies, namely a front and a rear frame and two wheels, connected via revolute joints. The wheels are infinitesimally thin (knife-edge). The latter ones introduce an additional degree of freedom that allows the rider's upper body to lean sideways. Both models are based on those introduced by Schwab et al.. The next out-of-plane model is a 3 degree of freedom motorcycle model based on V. Cossalter. By using M. Andres' non-ideal wheels, several additional degrees of freedoms can be added to the system depending on one's needs. To incorporate in-plane modes, two advanced motorcycle models both based on Sharp et al. are included in the library. Such models are composed of a front frame including the front forks and handle bar assembly, a rear frame including the lower rigid body of the rider, a swinging arm including the rear suspensions, the rider's upper body, a front and a rear wheel. Furthermore, several additional freedoms due to twist frame flexibility, suspensions, non-ideal tire models and aerodynamics are included. Compared to the former models, each body is created in a fully object-oriented fashion.

In Chapter 4, an eigenvalue analysis of an uncontrolled version of either a bicycle or a motorcycle model is performed. The result of such an analysis are the eigenvalues of the motorcycle as a function of the velocity. In other words, due to the results the self-stabilizing region of the vehicle is illustrated. With the knowledge of Chapter 2, these results can be perfectly interpreted. Furthermore, this task is beneficial for the optimization of the vehicle's geometry. By changing the geometry or the center of mass' locations of a vehicle, the eigenvalues of the system are changing as well. It is thus possible to optimize the design of a vehicle regarding self-stability. The simulation results of an eigenvalue analysis for the 3 and the 4 d.o.f. bicycle model have been compared to those established by Schwab et al.. Fortunately, the results perfectly agree with those of Schwab.

However, the main task of the master's thesis was to develop a virtual rider. In order to validate a single-track vehicle, a virtual rider is essential. A virtual rider has to fulfill two tasks in order to validate the performance of the vehicle. Firstly, it has to be ensured that the motorcycle stays upright. Secondly, it has to be possible to track either a roll angle profile or a pre-defined trajectory. In terms of stability, Chapter 5 introduces several different approaches for the controller design. Although classic controllers are presented and are also included in the library, the focus lies in state-space controller design. In the simplest case, the lean angle and the lean rate of the vehicle are fed back in order to generate an appropriate steering torque. However, since a physical interpretation of such a system is not possible, an alternative approach was developed. This approach is based on a preceding eigenvalue analysis. This means that exactly the same state variables, namely the steer angle, the lean angle and their derivatives, are used to design the controller. Of course, if the upper body of the rider is movable, the states γ and $\dot{\gamma}$ are taken into account as well. Thus, a physical interpretation of the poles is available. One major problem in controlling single-track vehicles is that the coefficients of the controller are strongly velocity dependent. This makes the manual configuration of a controller laborious and error-prone. As already mentioned, the eigenvalues are a function of the velocity, i.e. the trajectory of each eigenvalue is thus perfectly known. With this information, a controller design can be done in a straightforward manner. To this end, three different approaches were developed. In the first approach, all eigenvalues (poles) of the system are simply shifted towards the left-half plane by the same value. As this is not mandatory, two improved control laws have been established. Both are based on solely shifting those poles towards the left half-plane that are unstable. The results are several pole placement functions that automatically calculate the controller coefficients, i.e. the elements of the state feedback matrix. The algorithm of the function is based on *Ackermann's formula*. The corresponding output represents a state feedback matrix that can be directly applied to ready-made controllers which are the core of virtual riders. Finally, several approaches based on optimal control, more precisely on linear quadratic regulators (LQR), are introduced. One major advantage of optimal control contrary to pole placement is that the control energy is taken into account right from the start. However, it is difficult to say, whether an LQR or a pole placement design is preferable. Basically, both designs have their benefits. If one is more familiar with the physical interpretation of the system, pole placement is a convenient choice. Concerning the LQR design, it is very important to keep in mind that compared to classic minimization problems, optimal control is different. It is not possible to minimize both the state variables (performance of the controller) and the control energy (e.g. steering torque). It is rather a trade-off between control energy and performance.

The virtual riders composed of former developed controllers are presented in Chapter 6. For virtual riders capable of tracking a roll angle profile, several test tracks are provided. So far, the set-values were always set to zero to ensure stable behavior. Instead of these set-values, the roll angle profile (e.g. of a standard 90°-curve) is fed into the virtual rider. Since each vehicle has its own specific profile, some records including such profiles are provided. In order to track a pre-defined trajectory using path preview information, a randomly generated path is included in the library. This path is defined by its lateral profile. To emulate the behavior of a human rider, single-point preview is performed by the virtual rider, i.e. the rider looks a pre-defined distance ahead in order to follow the path. It is worth noting that a similar deviation pattern is actually observed from human riders. In order to track such a path, the controllers of Chapter 5 have been extended. In the simplest case, the lateral position of the rear frame's center of mass is fed back in order to generate an additional steering torque that keeps the vehicle on the desired path. For the utilized state-space and LQR controllers, the lateral rate is additionally taken into account. For the state-space path tracking controller, the pole placement function valid for a specific velocity is extended in order to design an appropriate controller. Due to a lack of time solely one approach was developed. At the end of this chapter, multi-point path preview is mentioned. Compared to single-point path preview, it is more realistic since a real human rider does not solely look at a single point. The rider rather gathers information of the whole road (path) in order to track it.

For the sake of usability, the library includes several examples. In the documentation itself, it is explained how some of them were established. This information is provided in the form of a user's guide. For instance, as soon as the vehicle-specific parameters are entered, one has to decide whether ideal or non-ideal wheels are used. Afterwards, an eigenvalues analysis is carried out. Regarding the self-stabilizing region of the vehicle, it is now possible to optimize the geometry (e.g. more trail increases the self-stabilizing area). If the calculation is based on a preceding eigenvalue analysis, the controller design is done with one of the implemented pole placement functions. The output of such a function are the automatically calculated coefficients of the state feedback matrix which are either displayed on the command window or stored in a *mat-file*. Now, the performance of the vehicle is evaluated. The library provides several examples to all the tasks mentioned above.

9 Further Work

Eigenvalue Analysis

The library provides functions to perform an eigenvalue analysis for the basic bicycle and motorcycle models. The main problem of these functions are the velocity dependent eigenvalues of the system matrix A (see Figure 4.1). As a consequence, the indices of the eigenvalues, which are stored in a vector, are changing as well. Thus, one has to sort the eigenvalues in a proper manner, which is cumbersome. Since the system matrix A of the basic bicycle and motorcycle models includes solely one conjugated complex pole pair, an algorithm capable of sorting the eigenvalues was feasible. However, the system matrix of the advanced motorcycle includes at least 5 conjugated complex pole pairs. Thus, it is much more complicated to provide an appropriate algorithm which ensures that the indices are not changing.

Additionally it would be of great interest to provide a universally valid function.

Controller Design

So far, a pole placement function for SISO systems has been established. However, a pole placement function for MIMO systems and a LQR function are still missing.

One possible pole placement algorithm valid for MIMO systems, which is not yet implemented, was developed by F. Cellier [Cel].

In addition, the pole placement functions that are provided in the library were especially developed for single-track vehicle models. Thus, it would be convenient to establish a function that can be independently used for each plant.

Steady State Error

Concerning the state variables, the state feedback matrix is equivalent to a proportional controller. A D portion (D controller) is directly or indirectly obtained from the plant. However, an integral portion to eliminate the steady-state error is missing. A possible solution is shown in Figure 9.1.

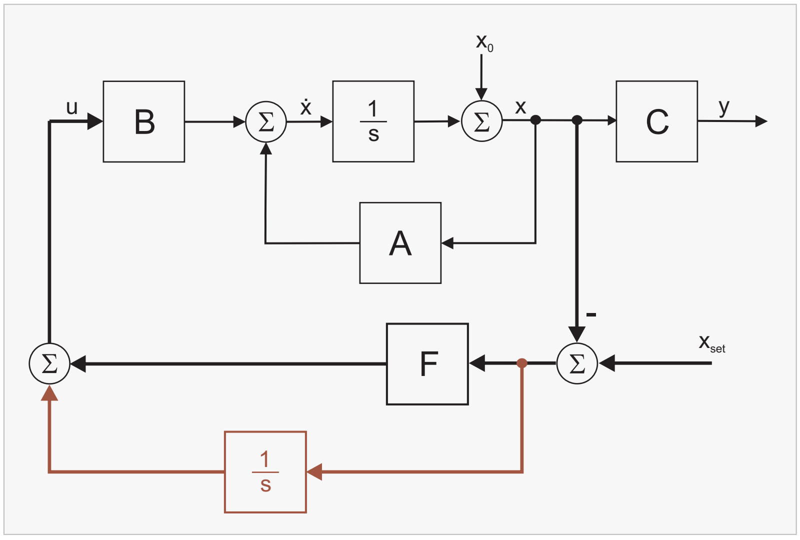


Figure 9.1: An additional integral portion in order to eliminate the steady-state error

The influences of the additional integral controller have not been tested yet. Since an integrator introduces a pole in the origin, it should be tested how this affects the whole systems. Moreover, it should be tested whether or not it is really sufficient to include integral controllers in such systems.

Path Preview

The library provides virtual riders that are capable of tracking a pre-defined path using single-point path preview. But a real human driver does not look solely at a single point a pre-defined distance ahead of the vehicle. A real rider gathers information of the whole path. Due to a lack of time, multi-point preview is not implemented in the library. The theoretical background of multi-point preview is provided in a paper by Sharp [SV01].

Path Tracking

The library provides a lateral profile that is described by means of global co-ordinates. The path generation is done with MATLAB. In a first step, one could establish a Modelica function in order to generate the path within the library. Furthermore, the path could be transformed into the rider's view. That is to say that the path is described by means of body co-ordinates which make it possible to generate complex paths. The theoretical background of this task can be found in [SV01].

Additionally it would be of great interest to consider an improved path generation. For example, the path could be composed of specific ready-made elements that are plugged together (e.g. straight elements, quadrant elements, etc.).

Bibliography

[And] M. Andres. Object-oriented modeling of wheels and tires. Master's Thesis, 2009.

[Ass07] Modelica Association. Modelica 3.0 language specification. http://www.modelica.org/documents/ModelicaSpec30.pdf, 2007.

[Ass09] Modelica Associantion. Modelica standard library. http://www.modelica.org/libraries/Modelica, 2009.

[Cel] F. E. Cellier. A numerically stable algorithm for pole placement of single/input and multi/input systems. Technical report, Dept. of Electrical & Computer Engineering, Univ. of Arizona/Tucson.

[Cel91] Francois E. Cellier. *Continuous System Modeling.* Springer, 1991.

[CN05] F. E. Cellier and À. Netbot. The modelica bond-graph library. In *Proceedings of the 4th International Modelica Conference, Hamburg*, pages 57–65, 2005.

[Cos06] V. Cossalter. *Motorcycle Dynamics.* 2006. 2nd edition.

[DFS$^+$06] F. Donida, G. Ferreti, S. M. Savaresi, M. Tanelli, and F. Schiavo. Motorcycle dynamics library in modelica. *Proceedings of the Fifth International Modelica Conference*, 5:157–166, 2006.

[DFST08] F. Donida, G. Ferreti, S. M. Savaresi, and M. Tanelli. Object-oriented modeling and simulation of a motorcycle. *Mathematical and Computer Modelling of Dynamic Systems*, 14, No. 2:79–100, 2008.

[Eva03] S. Evangelou. *The control and stability analysis of two-wheeled road vehicles.* PhD thesis, Imperial College London, September, 2003.

[Föl08] Otto Föllinger. *Regelungstechnik, Einführung in die Methoden und ihre Anwendungen.* Hüthig, 2008. 10. durchgesehene Auflage.

[Fri04] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley & Sons, 2004.

[Koe83] C. Koenen. *The dynamic behaviour of motorcycles when running straight ahead and when cornering.* PhD thesis, Delft University, 1983.

[KS10] F. Klein and A. Sommerfeld. Über die theorie des kreisels. *Quarterly Journal of Pure and Applied Mathematics*, Chapter 9, Section 8:863–884, Leipzig, 1910.

[LS06] D. J. N. Limebeer and R. S. Sharp. Bicycles, motorcycles and models. *IEEE Control Systems Magazine*, pages 34–61, October, 2006.

[Lun06] Jan Lunze. *Regelungstechnik II, Mehrgrößensysteme, Digitale Regelung.* Springer, 2006. 4. neu bearbeitete Auflage.

[McB05] Robert Thomas McBride. *System Analysis through Bond Graph Modeling.* PhD thesis, University of Arizona, 2005.

[MPRS07] J. P. Meijaard, J. M. Papadopoulos, A. Ruina, and A. L. Schwab. Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review. *R. Soc*, page 63, 2007.

[OEM03] M. Otter, H. Elmqvist, and S. E. Mattsson. The new modelica multibody library. *Proceedings of the third International Modelica Conference, Linköping*, 3, 2003.

[Sch09] Thomas Schmitt. Modeling of a motorcycle in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.

[SEL04] R. S. Sharp, S. Evangelou, and D. J. N. Limebeer. Advances in the modelling of motorcycle dynamics. *Multibody System Dynamics*, Volume 12:251–283, 2004.

[Sha71] R. S. Sharp. The stability and control of motorcycles. *Journal Mechanical Engineering Science*, Volume 13:316–329, 1971.

[Sha07a] R. S. Sharp. Optimal stabilization and path-following controls for a bicycle. *Proc. IMechE*, 221:34–61, 2007.

[Sha07b] R. S. Sharp. Motorcycle steering control by road preview. *Journal of Dynamic Systems, Measurement, and Control*, 129:415–428, July, 2007.

[SKM08] A. L. Schwab, J. D. G. Kooijman, and J. P. Meijaard. Some recent developments in bicycle dynamics and control. *Fourth European Conference on Structural Control*, page 8, 2008.

[SL01] R. S. Sharp and D. J. N. Limebeer. A motorcycle model for stability and control analysis. *Multibody System Dynamics*, Volume 6:123–142, 2001.

[SMP05] A. L. Schwab, J. P. Meijaard, and J. M. Papadopoulos. Benchmark results on the linearized equations of motion of an uncontrolled bicycle. *KSME International Journal of Mechanical Science and Technology*, pages 292–304, 2005.

[SV01] R. S. Sharp and V. Valtetsiotis. Optimal preview car steering control. *Vehicle System Dynamics*, Supplement 35:101–117, 2001.

[Unb07] Heinz Unbehauen. *Regelungstechnik II, Zustandsregelungen, digitale und nichtlineare Regelsysteme.* Hüthig, 2007. 9. durchgesehene und korrigierte Auflage.

[Whi99] F. J. W. Whipple. The stability of the motion of a bicycle. *Quarterly Journal of Pure and Applied Mathematics*, Volume 30:312–348, 1899.

[ZC06] D. Zimmer and F. E. Cellier. Multibond graph library. *Proceedings of the Fifth International Modelica Conference*, pages 559–568, 2006.

[Zim06] Dirk Zimmer. A modelica library for multibond graphs and its application in 3d-mechanics. Master's thesis, ETH Zürich, 2006.

Sworn declaration

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

Thomas Schmitt, Dornbirn, August 27, 2009

A Paper Submitted to the Modelica Conference 2009

A Virtual Motorcycle Rider Based on Automatic Controller Design

Thomas Schmitt
Vorarlberg Univ. of Appl. Sc.
Austria
Thomas.Schmitt@students.fhv.at

Dirk Zimmer
ETH Zürich
Switzerland
DZimmer@Inf.ETHZ.CH

François E. Cellier
ETH Zürich
Switzerland
FCellier@Inf.ETHZ.CH

Abstract

This paper introduces a new and freely available *Modelica* library for the purpose of simulation, analysis and control of bicycles and motorcycles (single-track vehicles). The library is called *MotorcycleLib* and focuses on the modeling of virtual riders based on automatic controller design.

For the single-track vehicles, several models of different complexity have been developed. To validate these models and their driving performance, virtual riders are provided. The main task of a virtual rider is to track either a roll angle profile or a pre-defined trajectory using path preview information. Both methods are implemented and several test tracks are also included in the library.

Keywords: virtual rider; automatic controller design; state-space controller, bicycle and motorcycle modeling; pole placement

1 Introduction

Among the vehicle models, models of bicycles and motorcycles turn out to be particularly delicate. Whereas a four-wheeled vehicle remains stable on its own, the same does not hold true for a single-track (two-wheeled) vehicle. For this reason, the stabilization of such a vehicle, a control issue, requires special attention.

A key task for a virtual rider is to stabilize the vehicle. To this end, a controller has to generate a suitable steering torque based on the feedback of appropriate state variables of the vehicle (e.g. lean angle and lean rate). One major problem in controlling single-track vehicles is that the coefficients of the controller are strongly velocity dependent. This makes the manual configuration of a controller laborious and error-prone. To overcome this problem, an automatic calculation of the controller's coefficients is desired. This calculation requires an eigenvalue analysis of the corresponding uncontrolled vehicle which is performed in order to determine the self-stabilizing area. The library includes the means for such an analysis and its results can be interpreted by three different modes that qualitatively describe the vehicle's motion [12]. This enables a convenient controller design and hence several control laws that ensure a stable driving behavior are provided. The corresponding output represents a state feedback matrix that can be directly applied to ready-made controllers which are the core of virtual riders. The functionality of this method is illustrated by several examples in the library.

In 2006, F. Donida et al. introduced the first Motorcycle Dynamics Library in Modelica [5] and [4]. The library focuses on the tire/road interaction. Moreover different virtual riders (rigidly attached to the main frame or with an additional degree of freedom (d.o.f.) allowing the rider to lean sideways) capable of tracking a roll angle and a target speed profile are presented. Until now these virtual riders include fixed structure controllers only [4]. This means that the virtual rider stabilizes the vehicle only correctly within a small velocity range.

Using the automatic controller design functions provided by the *MotorcycleLib* this major deficiency can be overcome. Furthermore, to validate the motorcycle's performance, the virtual rider is capable of either tracking a roll angle profile (open-loop method) or a pre-defined path (closed-loop method).

2 Bicycle and Motorcycle Models

The mathematical modeling of single-track vehicles is a challenging task which covers a wide range of models of varying complexity. The library provides several single-track vehicle models of different complexity. The models are composed of multibody elements and are based on bond graphs [2] and multi-bond graphs [17]. Basically two types of models are provided. Some include out-of-plane modes only, while others include both in-plane and out-of-plane modes. Roughly speaking, out-of-plane modes are related to stability and handling of single-track vehicles whereas in-plane modes are dealing with riding comfort. The wheels used in this library are either provided by D. Zimmer's *MultiBondLib* [17] or by M. Andres' *WheelsAndTires* library [1]. The former are ideal, whereas in the latter models non-ideal effects such as slip can be considered. For the bicycle, both 3 and 4 d.o.f. out-of-plane mode models are included in the library. The former are composed of four rigid bodies, namely a front frame, a rear frame including a rigidly attached rider and two wheels, connected via revolute joints. The wheels are infinitesimally thin (knife-edge). The latter introduce an additional d.o.f. that allows the rider's upper body to lean sideways. Both models are based on those introduced by Schwab et al. [12] and [11].

The out-of-plane mode motorcycle model is a 4 d.o.f. model that is based on a model established by V. Cossalter [3]. Basically, V. Cossalter's model is the same as the one introduced by R. S. Sharp in 1971 [13]. This model allows a lateral displacement of the rear frame since the wheels are no longer ideal. Due to the fact that the wheels of D. Zimmer's *MultiBondLib* [17] are ideal, the model is reduced to 3 d.o.f.Later, with reference to the *WheelsAndTires* library [1], it is possible to consider non-ideal effects of wheels and tires and thus simulate the lateral displacement of the wheels caused by tire slip. The animation of a 3 d.o.f. motorcycle is depicted in Figure 1. To incorporate in-

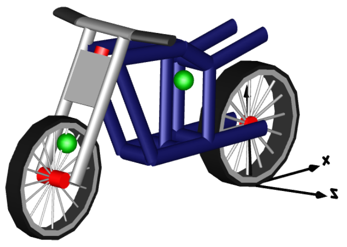


Figure 1: Animation of a 3 d.o.f. motorcycle model

plane modes two more complex models are included in the library. The first model was originally developed by C. Koenen during his Ph.D. Thesis [9]. R. S. Sharp and D. J. N. Limebeer introduced the SL2001 model which is based on Koenen's model [15]. They reproduced Koenen's model as accurately as possible and described it by means of multibodies. The model developed in this library is based on the SL2001 motorcycle. The second model is based on an improved more state-of-the-art version of the former developed by R. Sharp, S. Evangelou and D. J. N. Limebeer [14]. A very detailed description of these models can be found in S. Evangelou's Ph.D. Thesis [6]. Such models are composed of a front frame including the front forks and handle bar assembly, a rear frame including the lower rigid body of the rider, a swinging arm including the rear suspensions, the rider's upper body, a front and a rear wheel. Furthermore several additional freedoms due to twist frame flexibility at the steering head, suspensions, non-ideal tire models and aerodynamics are taken into account. The animation of the SL2001 model is depicted in Figure 2. In con-

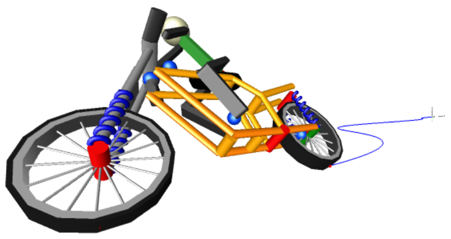


Figure 2: Animation of the SL2001 motorcycle model

trast to the former models each body is created in a fully object-oriented fashion. As with the out-of-plane models these models only include all degrees of freedom in combination with the *WheelsAndTires* library. Without this library several freedoms are inhibited.

It is important to keep in mind that vehicles in combination with ideal wheels include so called holonomic constraints. Such constraints are based on location and in case of single-track vehicles prevent them from sinking into the ground.

3 Eigenvalue Analysis

Due to geometry and gyroscopic forces Klein and Sommerfeld [8] found out that a single-track vehicle is self-stabilizing within a certain velocity range. That is, the vehicle performs a tail motion in the longitudinal direction. Below this range the steering deflections caused by gyroscopic forces are too small in order to generate enough centrifugal force. Thus the amplitude of the tail motion increases and the vehicle falls over. Although, these interactions are damped by the trail[1] it is still impossible to achieve stable behavior. Hence

[1]The trail is the distance between the front wheel contact point and the point of intersection of the steering axis with the ground line (horizontal axis).

the rider has to apply a steering torque to ensure that the vehicle stays upright. Above this range, for high speeds, the gyroscopic forces are almost unnoticeable for the rider. That is, the amplitude of the tail motion is close to zero. More precisely, although the vehicle feels stable, after a certain time, it falls over like a capsizing ship. However, by applying a steering torque it is rather simple to stabilize the vehicle. In most cases it is sufficient that one solely touches the handle bars in order to compensate for the instabilities.

An eigenvalue analysis is performed in order to determine the self-stabilizing range of an uncontrolled bicycle or motorcycle. For this purpose the state variables of the vehicle that are responsible for stability are of interest. These are the steer angle δ, the lean angle ϕ, and their derivatives.

$$x = \begin{pmatrix} \delta \\ \dot{\delta} \\ \phi \\ \dot{\phi} \end{pmatrix}$$

In case of vehicles with an additional d.o.f. allowing the rider's upper body to lean sideways, the state variables γ and $\dot{\gamma}$ are also taken into account, where γ is the lean angle of the rider's upper body relative to the rear frame and $\dot{\gamma}$ the corresponding lean rate. All the other state variables (e.g. lateral- and longitudinal position) of the state vector have no influence on the stability of single-track vehicles. Now, the eigenvalues (one for each state variable) are calculated as a function of the vehicle's forward velocity $\lambda = f(v)$ (e.g. $v = 10ms^{-1}$ to $v = 50ms^{-1}$). Thus, for each specific velocity the model is linearized. The result of such an analysis are three different velocity ranges at which the motion of the vehicle changes qualitatively. Figure 3 depicts a typical result of such an analysis. The first velocity range is below the stable region, the second one is within, and the third one above the stable region. Positive eigenvalues, or more precisely eigenvalues with a positive real part, correspond to unstable behavior whereas eigenvalues with a negative real part correspond to stable behavior. Eigenvalues including an imaginary part emphasize that the system is oscillating whereas eigenvalues without an imaginary part are non-oscillating. A stable region exists, if and only if all real parts of the eigenvalues are negative. In the following, the modes of single-track vehicles are explained with reference to Figure 3.

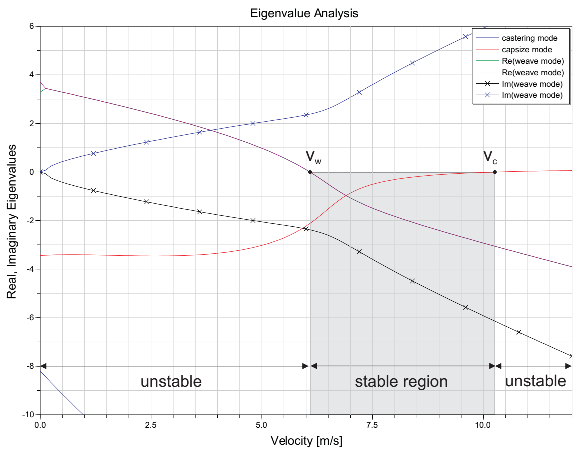


Figure 3: Result of the eigenvalue analysis for a 3 d.o.f. motorcycle model. The stable region is determined by eigenvalues with a negative real part. Here it is from $v_w = 6.1ms^{-1}$ to $v_c = 10.3ms^{-1}$.

3.1 Weave Mode

The weave mode begins at zero velocity. This mode is non-oscillating in the beginning and after a certain velocity passes over into an oscillating motion. The non-oscillating motion at very low speeds states that the bicycle is too slow to perform a tail motion and thus falls over like an uncontrolled inverted pendulum. As soon as it passes a certain value of approximately $v_w = 0.12ms^{-1}$ the real parts of the eigenvalues merge and two conjugate complex eigenvalues appear. Hence, a tail motion in the longitudinal direction emerges. This motion is still unstable but becomes stable as soon as the real parts of the eigenvalues cross zero. This happens at a velocity of about $v_w = 6.1ms^{-1}$. For all velocities greater than v_w this motion is stable.

3.2 Capsize Mode

The capsize mode is a non-oscillating motion that corresponds to a real eigenvalue dominated by the lean. As soon as the bicycle speed passes the upper limit of the stable region of about $v_c = 10.3ms^{-1}$, it falls over like a capsizing ship. However, above the stable region the bicycle is easy to stabilize although the real eigenvalue is positive. In the paper [12] of Sharp et al. this motion is called "mildly unstable" as long as the absolute value of the eigenvalues is smaller than $2s^{-1}$.

3.3 Castering Mode

The castering mode is a non-oscillating mode that corresponds to a real negative eigenvalue dominated by the steer. In this mode the front wheel has the tendency to turn towards the direction of the traveling vehicle.

4 Controller Design

4.1 An Introduction to State-Space Design

In general, the state-space representation of a linear system is given by:

$$\dot{x} = A \cdot x + B \cdot u, \quad x(0) = x_0 \tag{1}$$

$$y = C \cdot x + D \cdot u \tag{2}$$

where x is a $(n \times 1)$ state vector, y is a $(m \times 1)$ output vector, A is referred to as system matrix with a dimension of $(n \times n)$, B is a $(n \times r)$ input matrix, C is a $(m \times n)$ output matrix and D the "feedthrough" matrix with a dimension of $(m \times r)$. Usually D is set to zero, except if the output directly depends on the input. The state vector x at time $t = 0$ includes the initial conditions, sometimes referred to as initial disturbances x_0. The block diagram of a system described in state-space is illustrated in Figure 4. One major advantage of state-space control compared to classic control is that each state of the system can be controlled. In order to control the system, the state vector x is fed back. The state feedback control law for a linear time-invariant system is given by:

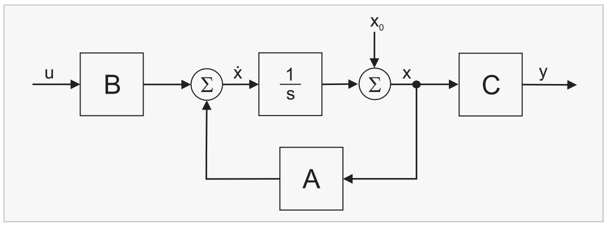


Figure 4: Block diagram of a system described in state-space

$$u(t) = -F \cdot x(t) \tag{3}$$

where F is a constant matrix.

By substituting u of Equation 1 with Equation 3 the state equation results in

$$\dot{x} = A \cdot x - B \cdot F \cdot x = (A - B \cdot F)x \tag{4}$$

The block diagram of the equation above is shown in Figure 5.

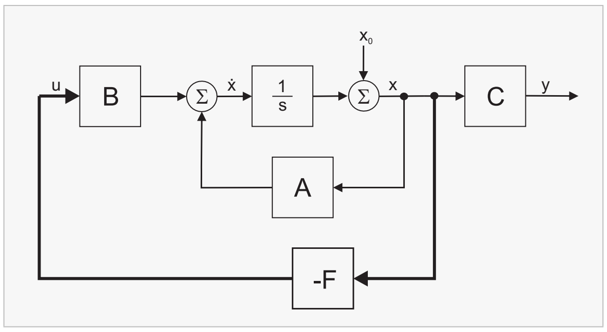


Figure 5: State feedback

The elements of the state feedback matrix F have to be chosen in such a way that the *initial disturbances* $x_0(t)$ for $t \to \infty$ converge towards zero

$$\lim_{t \to \infty} x(t) = 0 \tag{5}$$

and that the system becomes stable.

The main task of the state feedback control is to find appropriate coefficients for the state feedback matrix F in order to achieve the desired dynamical behavior of the system. One method that fulfills all the requirements is the so-called pole placement technique (refer to [7]). Figure 6 illustrates the graphical interpretation.

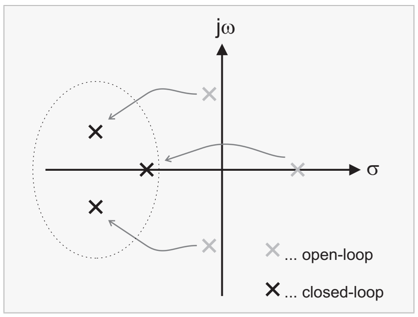


Figure 6: Graphical interpretation of the pole placement technique. Eigenvalues (poles) of the system located in the left-half plane correspond to stable behavior.

4.2 State-Space Controller Design Based on a Preceding Eigenvalue Analysis

The library includes several different stabilizing controllers. Although classic controllers and linear quadratic regulators (LQR) are included in the library, the focus lies in state-space controller design via the pole placement technique. In the simplest case the lean angle and the lean rate of the vehicle are fed back in order to generate an appropriate steering torque. However, since a physical interpretation of these eigenvalues is not possible an alternative approach is introduced in this paper. This approach is based on a preceding eigenvalue analysis. That is, exactly the same state variables, namely the steer angle δ, the lean angle ϕ, and their derivatives, are used to design the controller (see Figure 7). Of course if the upper body of

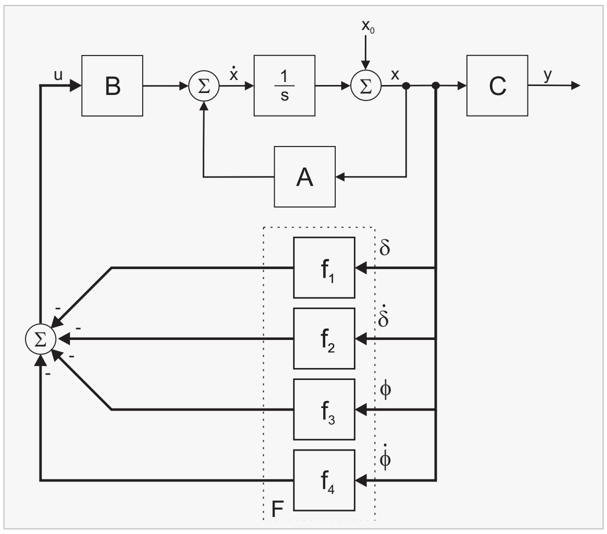


Figure 7: State-space controller based on a preceding eigenvalue analysis

the rider is movable, the states γ and $\dot{\gamma}$ are taken into account as well. Thus, a physical interpretation of the poles is available.

As already mentioned the eigenvalues are a function of the velocity, i.e. the trajectory of each eigenvalue is thus perfectly known. With this knowledge the velocity dependent coefficients of the state feedback matrix can be conveniently calculated. To this end, three different approaches were developed. In the first approach all eigenvalues (poles) of the system are simply shifted towards the left-half plane (see figure 6) by the same value (offset). A typical result for the controlled version of the 3 d.o.f. motorcycle is illustrated in Figure 8.

Two improved control laws have been established. Both are based on solely shifting those poles towards the left-half plane that are unstable. Within the stable region the motorcycle needs no control and thus no offset. Above the stable region (for velocities greater than v_c) the behavior of the bicycle is dominated by the capsize mode. Hence, it is absolutely sufficient to shift just this pole towards the left-half plane and leave all other poles unchanged. Below the stable region, for velocities lower than v_w, the instability of the motorcycle is caused by the weave mode (see Figure 9). To en-

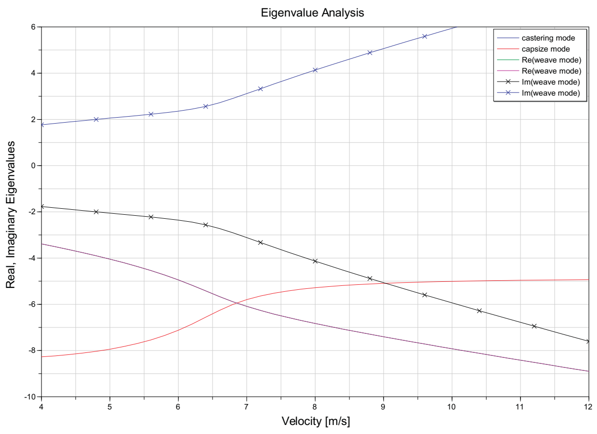


Figure 8: Result of the controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where the offset is $d = 5$.

sure stable behavior the two real parts of the conjugate complex poles have to be shifted towards the left-half plane. Now, a control law for the regions below and above the stable region is set up:

$$\textit{control law} \begin{cases} v < v_w: & d = d_w \cdot (v_w - v) \\ v_w < v < v_c: & d = 0 \\ v_c < v: & d = d_c \cdot (v - v_c) \end{cases}$$

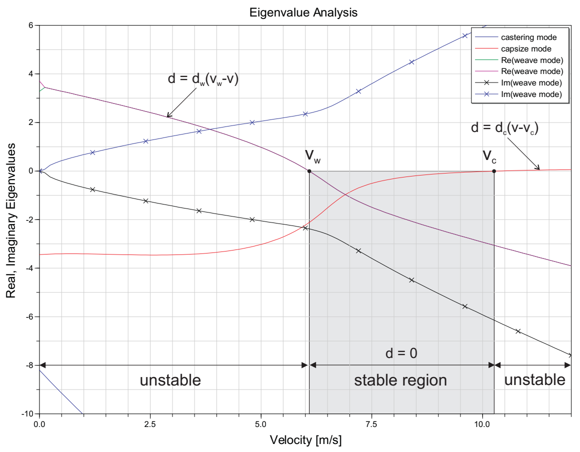


Figure 9: Controller design with reference to a preceding eigenvalue analysis. The stable region is left unchanged - below v_w, the weave mode eigenvalues are modified - above v_c, the capsize mode eigenvalue is modified.

Figure 10 shows the result of the individual controller design. Although the results of the individual controller are rather good, there is still potential for improvements. For velocities equal to v_w or v_c the eigenvalues that are responsible for stability are close or equal to zero. To be more precise, for such velocities the stability of the motorcycle is critical since a

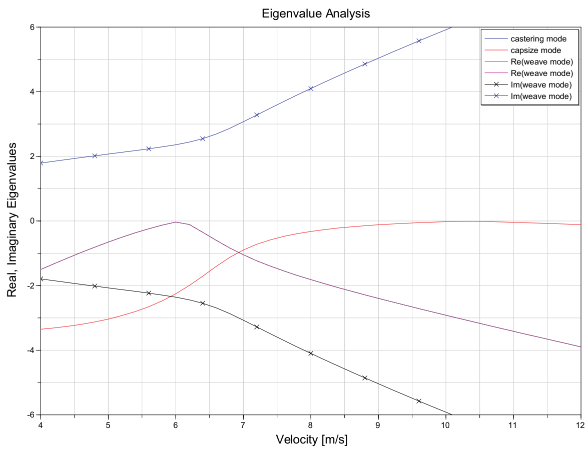


Figure 10: Result of the individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where $v_w = 6.1ms^{-1}$, $v_c = 10.3ms^{-1}$, $d_w = 1.5$ and $d_c = 0.1$.

real part equal to zero has no damping. Somewhere in the stable region the weave and the capsize mode have an intersection point v_i. Instead of the previous control law, the improved control law results in:

$$\mathit{control\ law} \begin{cases} v < v_i: & d = d_0 + d_w \cdot (v_i - v) \\ v_i < v: & d = d_0 + d_c \cdot (v - v_i) \end{cases}$$

A graphical interpretation of the control law is illustrated in Figure 11.

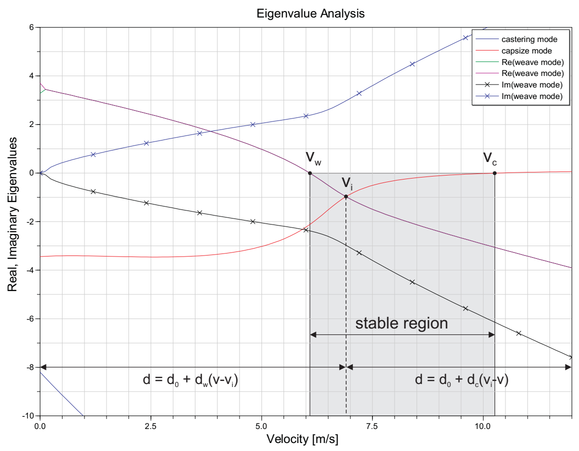


Figure 11: Controller design with reference to a preceding eigenvalue analysis. Below v_i, the weave mode eigenvalues are modified - Above v_i, the capsize mode eigenvalue is modified. In addition, the weave and capsize eigenvalues can be shifted by an offset d_0.

Figure 12 depicts the result of the improved individual controller design.

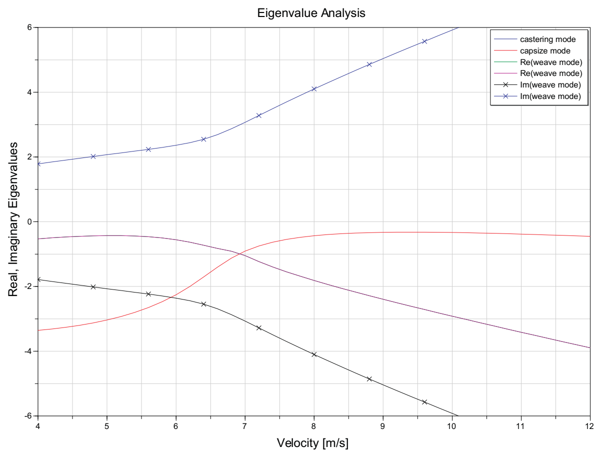


Figure 12: Result of the improved individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where $v_i = 6.9ms^{-1}$, $d_w = 0.75$, $d_c = 0.1$ and $d_0 = 0$

4.3 Results

The results are several pole placement functions that automatically calculate the controller coefficients, i.e. the elements of the state feedback matrix. The corresponding output represents a state feedback matrix that can be directly applied to ready-made controllers. The algorithm of the functions is based on *Ackermann's formula*. Unfortunately, it is just valid for single-input, single-output (SISO) systems. In order to design a multiple-input, multiple-output (MIMO) controller, e.g. for vehicles including rider's capable of leaning sideways, a MATLAB *m-file* based on the *place*-function is provided.

Finally, the coefficients of the state feedback matrix are automatically fed into a ready-made controller which is incorporated into a virtual rider. With respect to the virtual rider the vehicle's performance can now be evaluated.

5 Development of a Virtual Rider

5.1 Roll Angle Tracking

For virtual riders capable of tracking a roll angle profile, several test tracks are provided. So far, no reference input was used, i.e. the set-value of the state variables was zero. Instead of a set-values equal to zero, the roll angle profile (e.g. of a standard 90°-curve) is fed into the virtual rider. The corresponding block diagram is illustrated in Figure 13. Since each vehicle has its own specific profile, some records including such profiles are provided. The corresponding Mod-

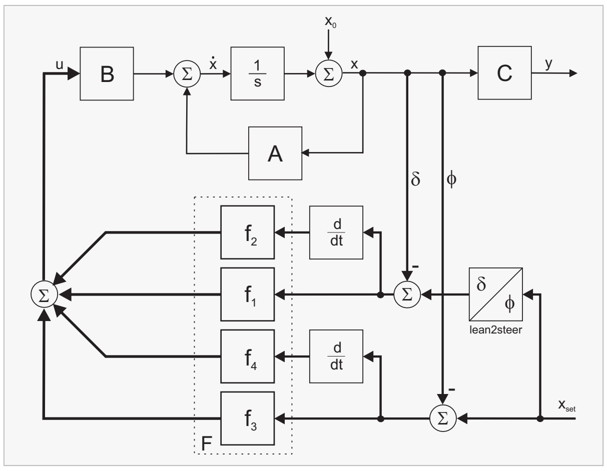


Figure 13: Block diagram of a virtual rider composed of a state-space controller and an additional block in order to calculate the corresponding steer angle. The reference input x_{set} is the desired roll angle profile.

elica model is depicted in Figure 14. The incorporated

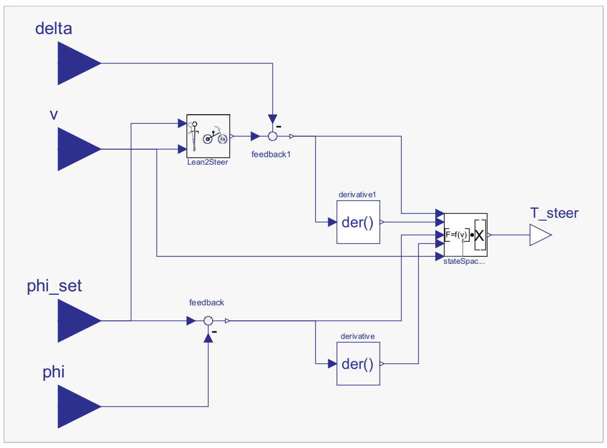


Figure 14: Wrapped model of the virtual rider composed of a state-space controller for a user defined velocity range. The inputs (blue) are lean and steer angle, lean angle set-value and the velocity of the motorcycle, the output T_{steer} (white) is the steering torque

controller is shown in Figure 15.

5.2 Path Tracking

In order to track a pre-defined trajectory using path preview information, a randomly generated path is included in the library. The path generation was done with MATLAB. This path is defined by its lateral profile [16]. To emulate the behavior of a human rider, single-point path preview is performed by the virtual rider. That is, the rider looks a pre-defined distance ahead in order to follow the path. It is worth noting that

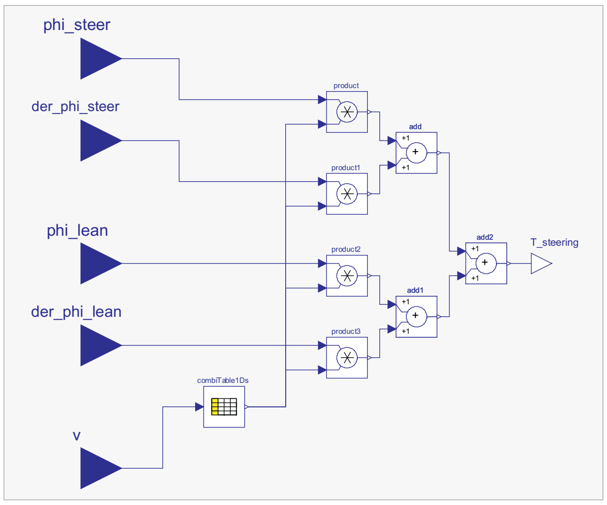


Figure 15: Wrapped model of a state-space controller for a specific velocity range. The table includes the state feedback matrix coefficients which by default are stored in *place.mat*

a similar deviation pattern is actually observed from human riders. In order to track a path, the controllers have to be extended [16] (see Figure 16). In the sim-

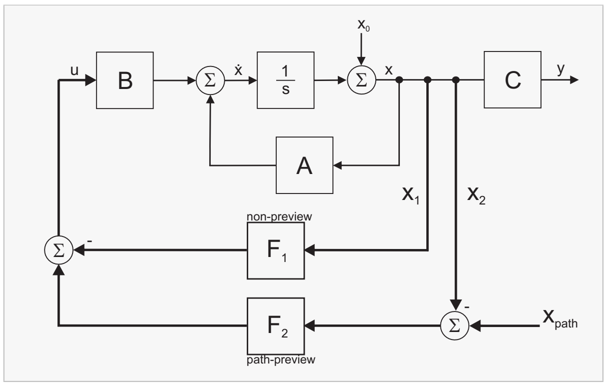


Figure 16: Basic structure of a state-space path preview controller. The state vector x_1 includes the states that are responsible for the stability (non-preview), whereas $x_2 = (x_{lat}\ \dot{x}_{lat})^T$ includes the states required for path tracking.

plest case the lateral position x_{lat} of the rear frame's center of mass is fed back in order to generate an additional steering torque that keeps the vehicle on the desired path. For the utilized state-space controllers the lateral rate $\dot{x}_{lat}$ is additionally taken into account. The corresponding Modelica model is basically the same as the one depicted in Figure 14. To cover the path tracking capabilities two additional inputs, namely x_{lat} and $\dot{x}_{lat}$ are included. For the state-space path tracking controller the pole placement functions were extended in order to conveniently design such a controller.

6 Examples

The first example demonstrates a 3 d.o.f. motorcycle stabilized by a virtual rider. The animation of the uncontrolled vehicle with a velocity of $4ms^{-1}$ is depicted in Figure 17. In order to determine the self-stabilizing range of the motorcycle an eigenvalue analysis is carried out. The results are shown in Figure 3. According to these results it can be seen that the vehicle is truly unstable for a velocity of $4ms^{-1}$. Furthermore, it can be seen that an offset of $d = 2$ is absolutely sufficient to achieve stable behavior. With this information the coefficients of the state feedback matrix are calculated. For this purpose the pole placement function based on the first approach is executed. The corresponding output is stored in the controller of the ready-made virtual rider introduced in Figure 14. The model of the controlled motorcycle is depicted in Figure 18. The animation of the controlled vehicle is shown in Figure 19.

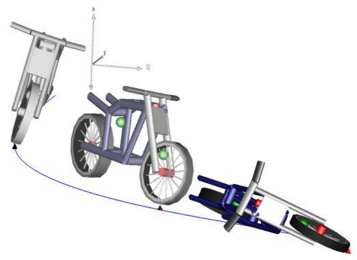


Figure 17: Animation result of the uncontrolled 3 d.o.f. motorcycle. After about 2s the motorcycle falls over like an uncontrolled inverted pendulum

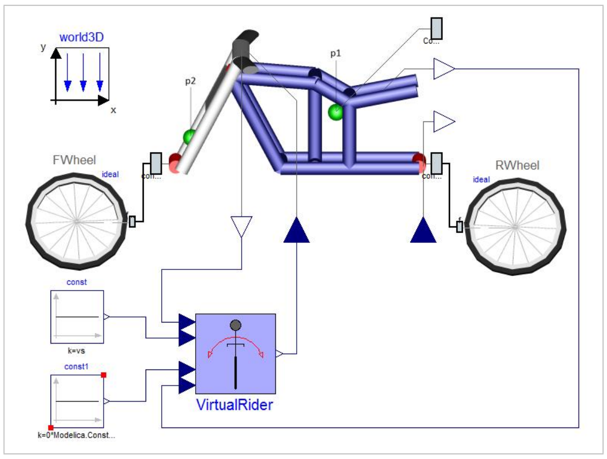


Figure 18: Example: controlled 3 d.o.f. motorcycle. The wrapped model of the virtual rider corresponds to Figure 14.

In the second example the motorcycle tracks a roll angle profile. The utilized model is depicted in Figure 18. Instead of a *constant source* block, the model of a 90°-curve is included. The coefficients of the state feedback matrix were automatically calculated for an offset $d = 5$. The resulting eigenvalues are equal to those depicted in Figure 8. The animation result is depicted in Figure 20.

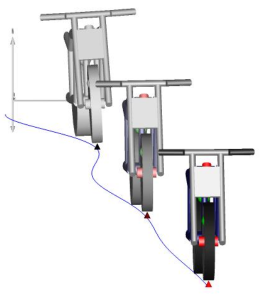


Figure 19: Animation result of the controlled 3 d.o.f. motorcycle

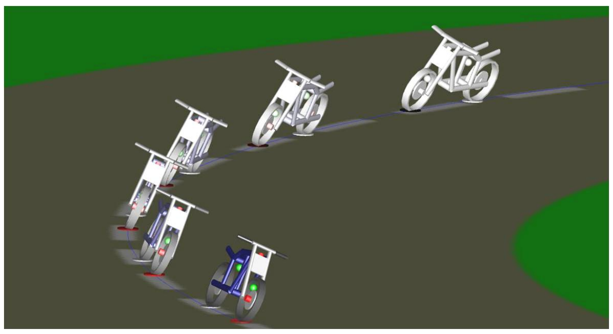


Figure 20: Animation result of a 3 d.o.f. motorcycle tracking a 90°-curve

In the last example the motorcycle tracks a predefined path. The utilized model is depicted in Figure 21.

Again, the coefficients of the state feedback matrix are automatically calculated for an offset $d = 5$. Additionally, an offset $d_{lat} = 5$ is needed in order to keep the motorcycle on the desired path. The results are shown in Figure 22.

7 Structure of the Library

The structure of the *MotorcycleLib* is depicted in Figure 23. For each single-track vehicle a separate sub-package is provided. The basic bicycle sub-package is composed of a rigid rider and a movable rider sub-package. Both include the corresponding wrapped model and a function in order to perform an eigenvalue analysis. The basic motorcycle sub-package also

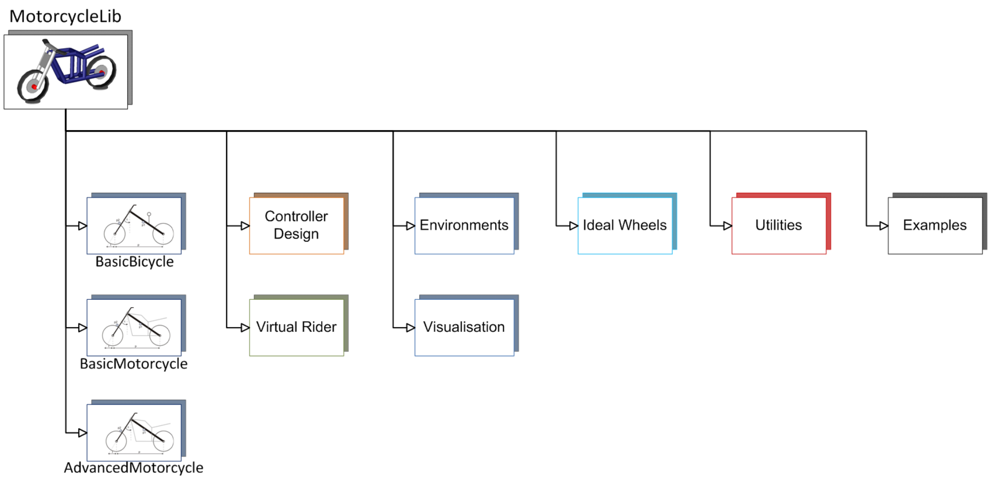


Figure 23: Library structure

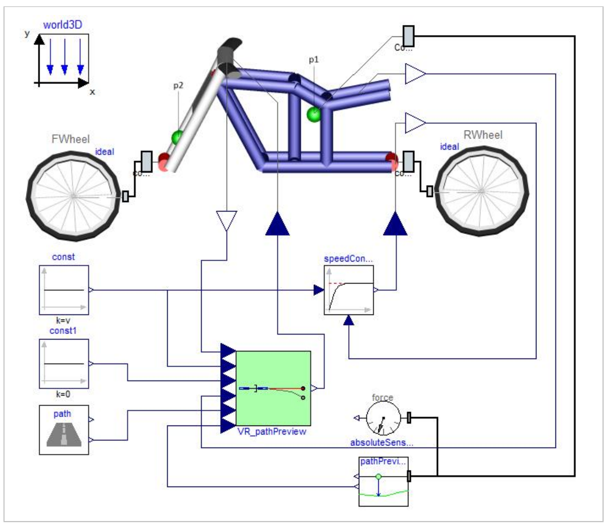


Figure 21: Example: model of a 3 d.o.f. motorcycle tracking a pre-defined path

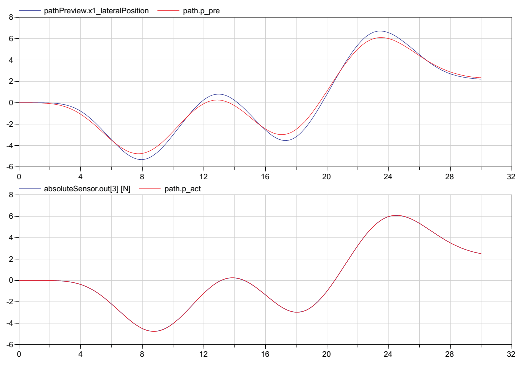


Figure 22: Simulation result of a motorcycle tracking a pre-defined path. Upper plot: The red signal is the path a pre-defined distance ahead, the blue signal is the preview distance of the rider. Lower plot: The red signal is the actual path, the blue signal is the traveled path measured at the rear frame's center of mass

includes a wrapped model and an eigenvalue analysis function. The structure of the advanced motorcycle sub-package is much more detailed since each part (e.g. front frame) is created in a fully object-oriented fashion. It is composed of a parts and an aerodynamics sub-package. The parts sub-package includes several different front and swinging arms, a rear frame, the rider's upper body, a torque source (engine), an elasto-gap and a utilities sub-package. In the latter one, models of characteristic spring and damper elements are stored. The aerodynamics sub-package includes a lift force, a drag force and a pitching moment model.

The controller design sub-package contains pole placement functions in order to design appropriate controllers. The virtual rider sub-package includes, among others, a virtual rigid rider and a virtual movable rider sub-package. In both the riders are capable of either tracking a roll angle profile or a pre-defined trajectory. To this end, several different controllers (e.g. classic, state-space and LQR) are incorporated into the virtual riders.

The environments sub-package provides tracks for both roll angle tracking and path tracking. In addition, the models for single-point path tracking are included. The visualization sub-package provides the graphical information for the environments sub-package. In the ideal wheels sub-package the visualization of the rolling objects from D. Zimmer's *MultiBondLib* were modified such that the appearance is similar to real motorcycle wheels. The utilities sub-package provides some additional functions and models which are partly used in the library. The purpose of the examples sub-package is to provide several different examples that demonstrate how to use the library.

8 Conclusion

The library provides appropriate eigenvalue functions for the basic bicycle and motorcycle models. Beside the controller design such an analysis is beneficial for the optimization of the vehicle's geometry. By changing the geometry or the center of mass' locations of a vehicle, the eigenvalues of the system are changing as well. It is thus possible to optimize the design of a vehicle regarding self-stability.

Furthermore, due to the results of the eigenvalue analysis it is now possible to conveniently design a state-space controller valid for a specific velocity range of the vehicle. Thus, for the calculation of the state feedback matrix coefficients, a pole placement function was developed. In order to design an LQR, MATLAB functions are provided.

To test the performance of the vehicles, the virtual riders are capable of tracking both, a roll angle profile and a pre-defined path. Therefore, several test tracks are included in the library.

A very detailed description of this paper can be found in the corresponding master's thesis [10].

References

[1] M. Andres. Object-oriented modeling of wheels and tires. Master's Thesis, 2009.

[2] F. E. Cellier and À. Netbot. The modelica bond-graph library. In *Proceedings of the 4th International Modelica Conference, Hamburg*, pages 57–65, 2005.

[3] V. Cossalter. *Motorcycle Dynamics*. 2006. 2nd edition.

[4] F. Donida, G. Ferreti, S. M. Savaresi, and M. Tanelli. Object-oriented modeling and simulation of a motorcycle. *Mathematical and Computer Modelling of Dynamic Systems*, 14, No. 2:79–100, 2008.

[5] F. Donida, G. Ferreti, S. M. Savaresi, M. Tanelli, and F. Schiavo. Motorcycle dynamics library in modelica. *Proceedings of the Fifth International Modelica Conference*, 5:157–166, 2006.

[6] S. Evangelou. *The control and stability analysis of two-wheeled road vehicles*. PhD thesis, Imperial College London, September, 2003.

[7] Otto Föllinger. *Regelungstechnik, Einführung in die Methoden und ihre Anwendungen*. Hüthig, 2008. 10. durchgesehene Auflage.

[8] F. Klein and A. Sommerfeld. Über die theorie des kreisels. *Quarterly Journal of Pure and Applied Mathematics*, Chapter 9, Section 8:863–884, Leipzig, 1910.

[9] C. Koenen. *The dynamic behaviour of motorcycles when running straight ahead and when cornering*. PhD thesis, Delft University, 1983.

[10] Thomas Schmitt. Modeling of a motorcycle in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.

[11] A. L. Schwab, J. D. G. Kooijman, and J. P. Meijaard. Some recent developments in bicycle dynamics and control. *Fourth European Conference on Structural Control*, page 8, 2008.

[12] A. L. Schwab, J. P. Meijaard, and J. M. Papadopoulos. Benchmark results on the linearized equations of motion of an uncontrolled bicycle. *KSME International Journal of Mechanical Science and Technology*, pages 292–304, 2005.

[13] R. S. Sharp. The stability and control of motorcycles. *Journal Mechanical Engineering Science*, Volume 13:316–329, 1971.

[14] R. S. Sharp, S. Evangelou, and D. J. N. Limebeer. Advances in the modelling of motorcycle dynamics. *Multibody System Dynamics*, Volume 12:251–283, 2004.

[15] R. S. Sharp and D. J. N. Limebeer. A motorcycle model for stability and control analysis. *Multibody System Dynamics*, Volume 6:123–142, 2001.

[16] R. S. Sharp and V. Valtetsiotis. Optimal preview car steering control. *Vehicle System Dynamics*, Supplement 35:101–117, 2001.

[17] D. Zimmer and F. E. Cellier. Multibond graph library. *Proceedings of the Fifth International Modelica Conference*, pages 559–568, 2006.

B Appendix: Modelica Functions

B.1 Eigenvalue Analysis

```
function CalculateEigenvaluesBicycle
  "Stability analysis of an uncontrolled bicycle, plots eigenvalues
   as a function of bicycle’s forward velocity"

  extends Modelica.Icons.Function;
  input String modelName;
  input String independentVariableName;
  input Real startValue "lowest velocity";
  input Real endValue "highest velocity";
  input Integer number_of_values;

input Real states[4] = {5,6,1,2}
    "|State Selection (state vector)| steer angle, der(steer angle),
     lean angle, der(lean angle)";

  input Integer plotSignals = 0
    "|Signals to plot| 0 ... real and imaginary eigenvalues;
     1 ... real eigenvalues; 2 ... imaginary eigenvalues";

protected
  Boolean hd = LinearSystems.Internal.SetHideDymosim();
  Boolean OK=linearizeModel(modelName);
  Real[number_of_values] values = linspace(startValue, endValue,
  number_of_values);

  Real nxMat[1, 1] = readMatrix("dslin.mat", "nx", 1, 1);
  Integer ABCDsizes[2] = readMatrixSize("dslin.mat", "ABCD");
  Integer nx = integer(nxMat[1, 1]);
  Integer nu = ABCDsizes[2] - nx;
  Integer ny = ABCDsizes[1] - nx;
  LinearSystems.StateSpace ABCD(
    nx = nx,
    ny = ny,
    nu = nu);

  /* Matrix which includes the relevant statevariables to compute the
     Eigenvalues
     Simulation window: type importInitial() then linearize the model
     and open the function "dslin.mat"
     type xuyName to get the selceted StateVariables */
  Real Arelevant[4,4];
  Real EigenValuesi[4, 2] = fill(0, 4, 2);
  Real EV_history[4,number_of_values] = fill(0,4,number_of_values);
  Real EV_history_imag[4,number_of_values] = fill(0,4,number_of_values);
  Real EV_real[4];
  Real EV_imag[4];
  Real EV_sort[4];
  Real EV_sort_imag[4];

  Integer n;
  Integer k;
  Integer m;

  String stateNames[nx] "Number of system states";
  Integer window = 0;
  Boolean status = animationOnline(loadInterval=  0.05);

algorithm
  // plot settings
  window := createPlot(id=  1,
                       position=   {2, 3, 750, 450},
                       y=   fill("", 0),
                       heading=  "Eigenvalue Analysis",
                       range=  {(-2.0), 12.0, 6.0, (-10.0)},
                       autoscale=  false,
                       autoerase=  false,
                       autoreplot=  true,
                       description=  false,
                       grid=  true,
                       color=  true,
                       online=  false,
                       legendLocation=  5,
                       legendHorizontal=  false,
                       leftTitle=  "Real, Imaginary Eigenvalues",
                       bottomTitle=  "Velocity [m/s]");

  ABCD := LinearSystems.linearize(modelName);
  EV_sort :={0,0,0,0};
  for i in 1:number_of_values loop
    SetVariable(independentVariableName, values[i]);
    OK := linearizeModel(modelName);
    if OK then
      ABCD := LinearSystems.readStateSpace("dslin.mat");
      stateNames := ABCD.xNames;

      /* Select the relevant states for the stability analysis
         here: Arelevant = {steer angle, steer rate, lean angle,
         lean rate} */
      Arelevant := (ABCD.A)[states, states];
      EigenValuesi := Modelica.Math.Matrices.eigenValues(Arelevant);
      // Eigenvalues are stored in two separate vectors
      EV_real := EigenValuesi[:, 1];
      EV_imag := EigenValuesi[:, 2];

      // control variables
      k := 0;
      m := 1;

      n :=size(EV_imag, 1);

      /* Remark:
           Since the function Modelica.Math.Matrices.eigenValues()
           orders the Eigenvalues not always in the same way it is
           necessary to re-order the Eigenvalues stored in the
           Eigenvalue-Vector EigenValuesi.
         Algorithm:
           1. check whether the eigenvalues include imaginary parts
             -> if an imaginary part appears, store the values in the
                last row of the vector EV_sort and EV_sort_imag
             -> else store the eigenvalues in ascending order
           2. sort the values in ascending order
             -> if all elements of EV_imag are zero sort all rows
             -> else (if imaginary parts appear) sort the first four
                rows and leave the remaining rows unchanged
       */
```

```
      // Part 1
      for j in 1:n loop
        if EV_imag[j] <> 0 then
          EV_sort[n-k] := EV_real[j];
          // build also a new vector which stores the imaginary
          // values <> 0
          EV_sort_imag[n-k] := EV_imag[j];
          k := k+1;
          //p := i;
        else
          EV_sort[m] := EV_real[j];
          EV_sort_imag[m] := EV_imag[j];
          m := m+1;
        end if;
      end for;

      // Part 2
      if EV_imag[1] == 0 and EV_imag[2] == 0 and EV_imag[3] == 0 and
         EV_imag[4] == 0 then
         EV_history[:, i] := Modelica_LinearSystems.Math.Vectors.
         sort(EV_sort);
         EV_history_imag[3:4, i] := EV_sort_imag[3:4];
      else
        EV_history[1:2, i] := Modelica_LinearSystems.Math.Vectors.
        sort(EV_sort[1:2]);
        EV_history[3:4, i] := EV_sort[3:4];
        EV_history_imag[3:4,i] := Modelica_LinearSystems.Math.Vectors.
        sort(EV_sort_imag[3:4]);
      end if;

    end if;
  end for;

   if plotSignals == 0 then
     // plot the real eigenvalues
     plotArrays(x= values[1:number_of_values],
               y= transpose(EV_history[:,1:number_of_values]),
               title= "Eigenvalue Analysis",
               legend= {"castering mode", "capsize mode",
                        "Re(weave mode)", "Re(weave mode)"},
               style= {0});
     // plot the imaginary eigenvalues
     plotArrays(x= values[1:number_of_values],
               y= transpose(EV_history_imag[3:4,1:number_of_values]),
               title= "Eigenvalue Analysis",
               legend= {"Im(weave mode)", "Im(weave mode)"},
               style= {1});
    elseif plotSignals == 1 then
      plotArrays(x= values[1:number_of_values],
               y= transpose(EV_history[:,1:number_of_values]),
               title= "Eigenvalue Analysis",
               legend= {"castering mode", "capsize mode",
                        "Re(weave mode)", "Re(weave mode)"},
               style= {0});
    else
      plotArrays(x= values[1:number_of_values],
               y= transpose(EV_history_imag[3:4,1:number_of_values]),
               title= "Eigenvalue Analysis",
               legend= {"Im(weave mode)", "Im(weave mode)"},
               style= {1});
    end if;

  writeMatrix("stability.mat","eigenvalues",
  [values,transpose(EV_history[:,:])]);
  Advanced.HideDymosim := hd;

end CalculateEigenvaluesBicycle;´
```

B.2 getStates Function

```
function getStates
  "Returns the states of the vehicle (according to these names
   appropriate states can be selected)"

  extends Modelica.Icons.Function;
  import Modelica.Utilities;

  input String modelName;

protected
  Boolean hd = LinearSystems.Internal.SetHideDymosim();
  Boolean OK=linearizeModel(modelName);
  Real nxMat[1, 1] = readMatrix("dslin.mat", "nx", 1, 1);
  Integer ABCDsizes[2] = readMatrixSize("dslin.mat", "ABCD");
  Integer nx = integer(nxMat[1, 1]);
  Integer nu = ABCDsizes[2] - nx;
  Integer ny = ABCDsizes[1] - nx;

  LinearSystems.StateSpace ABCD(
    nx = nx,
    ny = ny,
    nu = nu);

algorithm
  ABCD := LinearSystems.linearize(modelName);
  stateNames := ABCD.xNames;

  // print the states to command window
  for i in 1:nx loop
    Modelica.Utilities.Streams.print(" ");
    Modelica.Utilities.Streams.print("Nr." + integerString(i));
    Modelica.Utilities.Streams.print(stateNames[i]);
  end for;

  Advanced.HideDymosim := hd;

  annotation (Documentation(info="<html>
<p>
This function is used to get the states of a vehicle.
</p>
</html>"));
end getStates;
```

B.3 place Function

```
function place
  "calculates the state feedback matrix of a state-space system
   w.r.t. pole placement"

  import Modelica_LinearSystems.Math.Complex;
  import Modelica.Utilities.Streams;
  import SI = Modelica.SIunits;
  extends Modelica.Icons.Function;
  input String modelName;
  input String independentVariableName = "vs";

  input SI.Velocity v = 10 "Forward velocity of the motorcycle";

  input Real offset_real = 10
    "|Inputs for Controller Design| Real part of the offset";
  input Real offset_imag = 0
    "|Inputs for Controller Design| Imaginary part of the offset";

  input Real states[4] = {3, 4, 11, 13}
    "|State Selection (state vector)| steer angle, der(steer angle),
     lean angle, der(lean angle)";

  output Real F[1, 4] "State Feedback Matrix";

protected
  Complex offset = Complex(re=offset_real, im=offset_imag);

  Boolean hd = LinearSystems.Internal.SetHideDymosim();
  Boolean OK=linearizeModel(modelName);
  Real nxMat[1, 1] = readMatrix("dslin.mat", "nx", 1, 1);
  Integer ABCDsizes[2] = readMatrixSize("dslin.mat", "ABCD");
  Integer nx = integer(nxMat[1, 1]);
  Integer nu = ABCDsizes[2] - nx;
  Integer ny = ABCDsizes[1] - nx;

  LinearSystems.StateSpace ABCD(
    nx = nx,
    ny = ny,
    nu = nu);

  Real Arel[4,4];
  Real Brel[4,1];
  Real con[4,4];
  Real eig[4, 2] = fill(0, 4, 2); // each element has the value 0
  Real t[1,4];
  Complex p[1,4];
  Complex h[1,4];
  Complex c[1,5];
  Real c_real[1,5];
```

```
algorithm
  ABCD := LinearSystems.linearize(modelName);
  SetVariable(independentVariableName, v);

  OK := linearizeModel(modelName);
  if OK then
    ABCD := LinearSystems.readStateSpace("dslin.mat");

    // state selection to calculate the relevant matrices
    //states := {4, 9, 7, 8};
    //states := {3,4,11,13};
    Arel := (ABCD.A)[states, states];
    Brel := (ABCD.B)[states,:];

    // compute the controlability matrix
    con := [Brel, Arel*Brel, Arel^2*Brel, Arel^3*Brel];

    // store the last row of con in t
    t := [0, 0, 0, 1]*Modelica.Math.Matrices.inv(con);
    //t := [0, 0, 0, 1]/con;
    // calculate eigenvalues of Arel
    eig := Modelica.Math.Matrices.eigenValues(Arel);

    eiga := Complex(re=  eig[1,1], im=  eig[1,2]);
    eigb := Complex(re=  eig[2,1], im=  eig[2,2]);
    eigc := Complex(re=  eig[3,1], im=  eig[3,2]);
    eigd := Complex(re=  eig[4,1], im=  eig[4,2]);
    eig2 := [eiga, eigb, eigc, eigd];
    eig4 := Complex(re=  eig[:,1], im=  eig[:,2]);

    // pole placement according to a user defined offset
    // (store values in a complex eigenvector)
    pa := Complex.'-'(eiga,offset);
    pb := Complex.'-'(eigb,offset);
    pc := Complex.'-'(eigc,offset);
    pd := Complex.'-'(eigd,offset);
    p := [pa, pb, pc, pd];

    // the following complex numbers c1 ... c5 (stored in c) and
    // h1 ... h4 (stored in h) are used
    // to calculte the coefficients of Arel
    c1 := Complex(re=1, im=0);
    c2 := Complex(re=0, im=0);
    c3 := Complex(re=0, im=0);
    c4 := Complex(re=0, im=0);
    c5 := Complex(re=0, im=0);
    c := [c1, c2, c3, c4, c5];

    h1 := Complex(re=0, im=0);
    h2 := Complex(re=0, im=0);
    h3 := Complex(re=0, im=0);
    h4 := Complex(re=0, im=0);
    h := [h1, h2, h3, h4];

    // calculate the coefficients of Arel
    for i in 1:4 loop
      if i == 1 then
        h1 := Complex.'*'(pa, c1);
        c2 := Complex.'-'(c2, h1);
      elseif i == 2 then
        h1 := Complex.'*'(pb, c1);
        h2 := Complex.'*'(pb, c2);
        c2 := Complex.'-'(c2, h1);
        c3 := Complex.'-'(c3, h2);
      elseif i == 3 then
        h1 := Complex.'*'(pc, c1);
        h2 := Complex.'*'(pc, c2);
        h3 := Complex.'*'(pc, c3);
        c2 := Complex.'-'(c2, h1);
        c3 := Complex.'-'(c3, h2);
        c4 := Complex.'-'(c4, h3);
      else
        h1 := Complex.'*'(pd, c1);
        h2 := Complex.'*'(pd, c2);
        h3 := Complex.'*'(pd, c3);
        h4 := Complex.'*'(pd, c4);
        c2 := Complex.'-'(c2, h1);
        c3 := Complex.'-'(c3, h2);
        c4 := Complex.'-'(c4, h3);
        c5 := Complex.'-'(c5, h4);
      end if;
      h := [h1, h2, h3, h4];
      c := [c1, c2, c3, c4, c5];
    end for;
  end if;
  // remove the imaginary part of the coefficient vector c
  c_real := [c1.re, c2.re, c3.re, c4.re, c5.re];
  // calculate the state feedback matrix F
  Modelica.Utilities.Streams.print(" ");
  Modelica.Utilities.Streams.print("State Feedback Matrix F");
  F := t*[c_real[1, 5]*identity(4) + c_real[1, 4]*Arel
       + c_real[1, 3]*Arel^2 + c_real[1, 2]*Arel^3 + Arel^4];

end place;
```

B.4 placeRange Function

In the following the `placeRange` function valid for a specific offset is presented. The basic structure of the other pole placement functions is almost the same.

```
function placeRange_offset
  "calculates a set of feedback matrices for a user selected velocity
   range of a state-space system w.r.t. pole placement"

  import Modelica_LinearSystems.Math.Complex;
  import Modelica.Utilities.Streams;
  import SI = Modelica.SIunits;
  extends Modelica.Icons.Function;

  input String modelName;
  input String independentVariableName = "vs";
  input Real startValue = 5;
  input Real endValue = 30;
  input Integer number_of_values = 25;

  input Real d = 10
    "|Input for Controller Design| Offset in order to shift the poles";

  input Real states[4] = {3, 4, 11, 13}
    "|State Selection (state vector)| steer angle, der(steer angle),
    lean angle, der(lean angle)";

  input String filename = "place.mat"
    "|Store Settings| Filename to store the state feedback matrix";

  output Real F[1, 4] "State Feedback Matrix";
```

```
protected
  Boolean hd = LinearSystems.Internal.SetHideDymosim();
  Real[number_of_values] values = linspace(startValue, endValue,
  number_of_values);

  Complex offset = Complex(re=d, im=0);

  Real EV_history[4,number_of_values] = fill(0, 4, number_of_values);
  Real EV_history_imag[4,number_of_values] = fill(0, 4,
  number_of_values);

  Real EV_real[4];
  Real EV_imag[4];
  Real EV_sort[4];
  Real EV_sort_imag[4];

  Integer n;
  Integer k;
  Integer m;

  Real nxMat[1, 1] = readMatrix("dslin.mat", "nx", 1, 1);
  Integer ABCDsizes[2] = readMatrixSize("dslin.mat", "ABCD");
  Integer nx = integer(nxMat[1, 1]);
  Integer nu = ABCDsizes[2] - nx;
  Integer ny = ABCDsizes[1] - nx;

  LinearSystems.StateSpace ABCD(
    nx = nx,
    ny = ny,
    nu = nu);

  Real Arel[4,4];
  Real Brel[4,1];
  Real F_table[number_of_values, 4] = fill(0, number_of_values, 4);
  Real con[4,4];
  Real eig[4, 2] = fill(0, 4, 2);
  Real t[1,4];
  Complex p[1,4];
  Complex h[1,4];
  Complex c[1,5];
  Real c_real[1,5];

algorithm
```

```
ABCD := LinearSystems.linearize(modelName);

for i in 1:number_of_values loop
  SetVariable(independentVariableName, values[i]);
  OK := linearizeModel(modelName);
  if OK then
    ABCD := LinearSystems.readStateSpace("dslin.mat");
    // state selection to calculate the relevant matrices
    //states := {4, 9, 7, 8};
    Arel := (ABCD.A)[states, states];
    Brel := (ABCD.B)[states,:];

    // compute the controlability matrix
    con := [Brel, Arel*Brel, Arel^2*Brel, Arel^3*Brel];

    // store the last row of con in t
    t := [0, 0, 0, 1]*Modelica.Math.Matrices.inv(con);

    // calculate eigenvalues of Arel
    eig := Modelica.Math.Matrices.eigenValues(Arel);
    eig_test :=eig;
    // Eigenvalues are stored in two separate vectors
    EV_real := eig[:, 1];
    EV_imag := eig[:, 2];

    // control variables
    k := 0;
    m := 1;

    n :=size(EV_imag, 1);
    /* Remark:
        Since the function Modelica.Math.Matrices.eigenValues()
        orders the Eigenvalues not always in the same way it is
        necessary to re-order the Eigenvalues stored in the
        Eigenvalue-Vector eig.
      Algorithm:
        1. check whether the eigenvalues include imaginary parts
          -> if an imaginary part appears, store the values in the
             last row of the vector EV_sort and EV_sort_imag
          -> else store the eigenvalues in ascending order
        2. sort the values in ascending order
          -> if all elements of EV_imag are zero sort all rows
          -> else (if imaginary parts appear) sort the first four
```

```
         rows and leave the remaining rows unchanged
*/

// Part 1
for j in 1:n loop
  if EV_imag[j] <> 0 then
    EV_sort[n-k] := EV_real[j];
    // build also a new vector which stores the imaginary
    // values <> 0
    EV_sort_imag[n-k] := EV_imag[j];
    k := k+1;
    //p := i;
  else
    EV_sort[m] := EV_real[j];
    EV_sort_imag[m] := EV_imag[j];
    m := m+1;
  end if;
end for;

// Part 2
if EV_imag[1] == 0 and EV_imag[2] == 0 and EV_imag[3] == 0
   and EV_imag[4] == 0 then
   EV_history[:, i] := Modelica_LinearSystems.Math.Vectors.
   sort(EV_sort);
   EV_history_imag[3:4, i] := EV_sort_imag[3:4];
else
  EV_history[1:2, i] := Modelica_LinearSystems.Math.Vectors.
  sort(EV_sort[1:2]);
  EV_history[3:4, i] := EV_sort[3:4];
  EV_history_imag[3:4,i] := Modelica_LinearSystems.Math.Vectors.
  sort(EV_sort_imag[3:4]);
end if;

eiga := Complex(re=  EV_history[1,i], im=  EV_history_imag[1,i]);
eigb := Complex(re=  EV_history[2,i], im=  EV_history_imag[2,i]);
eigc := Complex(re=  EV_history[3,i], im=  EV_history_imag[3,i]);
eigd := Complex(re=  EV_history[4,i], im=  EV_history_imag[4,i]);
eig2 := [eiga, eigb, eigc, eigd];

// store eigenvalues such that F has correct coefficients
eig3 := [eiga, eigd, eigc, eigb];
eiga_s := eiga;
eigb_s := eigd;
```

```
eigc_s := eigc;
eigd_s := eigb;

// pole placement according to a user defined offset
// (store values in a complex eigenvector)
pa := Complex.'-'(eiga_s,offset);
pb := Complex.'-'(eigb_s,offset);
pc := Complex.'-'(eigc_s,offset);
pd := Complex.'-'(eigd_s,offset);
p := [pa, pb, pc, pd];

// the following complex numbers c1 ... c5 (stored in c)
// and h1 ... h4 (stored in h) are used
// to calculte the coefficients of Arel
c1 := Complex(re=1, im=0);
c2 := Complex(re=0, im=0);
c3 := Complex(re=0, im=0);
c4 := Complex(re=0, im=0);
c5 := Complex(re=0, im=0);
c := [c1, c2, c3, c4, c5];

h1 := Complex(re=0, im=0);
h2 := Complex(re=0, im=0);
h3 := Complex(re=0, im=0);
h4 := Complex(re=0, im=0);
h := [h1, h2, h3, h4];

// calculate the coefficients of Arel
for i in 1:4 loop
  if i == 1 then
    h1 := Complex.'*'(pa, c1);
    c2 := Complex.'-'(c2, h1);
  elseif i == 2 then
    h1 := Complex.'*'(pb, c1);
    h2 := Complex.'*'(pb, c2);
    c2 := Complex.'-'(c2, h1);
    c3 := Complex.'-'(c3, h2);
  elseif i == 3 then
    h1 := Complex.'*'(pc, c1);
    h2 := Complex.'*'(pc, c2);
    h3 := Complex.'*'(pc, c3);
    c2 := Complex.'-'(c2, h1);
    c3 := Complex.'-'(c3, h2);
```

```
          c4 := Complex.'-'(c4, h3);
        else
          h1 := Complex.'*'(pd, c1);
          h2 := Complex.'*'(pd, c2);
          h3 := Complex.'*'(pd, c3);
          h4 := Complex.'*'(pd, c4);
          c2 := Complex.'-'(c2, h1);
          c3 := Complex.'-'(c3, h2);
          c4 := Complex.'-'(c4, h3);
          c5 := Complex.'-'(c5, h4);
        end if;
        h := [h1, h2, h3, h4];
        c := [c1, c2, c3, c4, c5];
      end for;

      // remove the imaginary part of the coefficient vector c
      c_real := [c1.re, c2.re, c3.re, c4.re, c5.re];

      // calculate the state feedback matrix F
      F := t*[c_real[1, 5]*identity(4) + c_real[1, 4]*Arel +
      c_real[1, 3]*Arel^2 + c_real[1, 2]*Arel^3 + Arel^4];
      F_table[i, :] := F[1,:];
    end if;
  end for;

  // store the state feedback matrices in a table
  writeMatrix(filename,"Ftable",[values,F_table[:,:]]);
  // DataFiles.readMATmatrix("place.mat", "F_table");

end placeRange_offset;
```

C Appendix: MATLAB Functions

C.1 Pole Placement - States: ϕ and $\dot{\phi}$

```
%% LeanControl.m
% design of a lean controller for the basic motorcycle model

clc;

%% load dslin.math and show the names of the states, inputs and outputs
load dslin.mat
xuyName

%% States
% 1.  leanAngle
% 2.  leanRate

%% Input
% u1: Steering torque

[A,B,C,D]=tloadlin('dslin.mat');
%%
% Define the relevant A Matrix depending on the states needed for the
% controller
states = [1, 2];

Arel = A(states, states);
Brel = B(states,:);

%% Is the System controlable?
% Compute the controlability matrix
Co = ctrb(Arel, Brel);

% If the rank of the matrix is equal to the system's states then the system
% is controlable
rang = rank(Co);
[m, n] = size(Arel);
if (rang < n)
    disp('r < n —> System is not controlable!');
else
```

```
    disp('System controlable!');
end

%% Controller design
disp(' ')
disp('Compute the poles (p) of the vehicle:')
p = eig(Arel)

p1 = - 3;
p2 = - 5;

disp('Controller Matrix F:');
F = place(Arel, Brel, [p1, p2])

% check the pole locations
p_real = eig(Arel-Brel*F)
```

C.2 Pole Placement for Models Composed of a Movable Rider

```
%% Bicycle_StateSpaceController_LeanSteer_RiderLean.m
% State-Space Controller for a 4 d.o.f bicycle model
% the output of this function is a state feedback matrix calculated in two
% different ways:
%   - Pole placement:   place()
%   - Optimal control:  lqr()

clc;

%% load dslin.math and show the names of the states, inputs and outputs
load dslin.mat;
xuyName;

%% Input
% u1: Steering torque
% u2: Rider lean torque

[A,B,C,D]=tloadlin('dslin.mat');
%%
% Define the relevant A Matrix depending on the states needed for the
% controller

% state selection
% states = [steer angle, steer rate
%           lean angle, lean rate,
%           rider lean angle, rider lean rate];

states = [7, 8, 3, 4, 9, 10];
Arel = A(states, states);
Brel = B(states,:);
```

```
%% Is the System controlable?
% Compute the controlability matrix
Co = ctrb(Arel, Brel);

% If the rank of the matrix is equal to the system's states then the system
% is controlable
rang = rank(Co);
[m, n] = size(Arel);
if (rang < n)
    disp('r < n --> System is not controlable!');
else
    disp('System controlable!');
end

%% Controller design
% to design a simple state-space controller we first compute the root locus
% of the system --> does not work - only for SISO Systems
% compute the pole-zero map of the system

% pz = pzmap(vehicle)
disp(' ')
disp('Compute the poles (p) of the vehicle:')
p = eig(Arel)

% pole plalement according to pole-zero map

offset = 5;
offset_rider = 3;
p1 = p(1) - offset;
p2 = p(2) - offset_rider;
p3 = p(3) - offset;
p4 = p(4) - offset;
p5 = p(5) - offset;
p6 = p(6) - offset_rider;

poles = [p1, p2, p3, p4, p5, p6];

disp('Controller Matrix F:');
F = place2(Arel, Brel, poles)

% LQR Design
R = [1 0; 0 1];
% introducing Q matrix
q1 = 0;
q2 = 0;
q3 = 1;
q4 = 0;
q5 = 1;
q6 = 0;

Q = diag([q1, q2, q3, q4, q5, q6]);
```

```
F_lqr = lqr(Arel, Brel, Q, R)

% check the pole location
p_real = eig(Arel-Brel*F)
```

C.3 LQR Design

```
%% LQR_Controller.m
% State-Space Controller for the basic bicycle model

%close all;
clc;

%% load dslin.math and show the names of the states, inputs and outputs
load dslin.mat;
xuyName;

%% Input
% u1: Steering torque

[A,B,C,D]=tloadlin('dslin.mat');
%%
% Define the relevant A Matrix depending on the states needed for the
% controller

states = [7,8,3,4];     % Steer Angle, Steer Rate, Lean Angle, Lean Rate

Arel = A(states, states);
Brel = B(states,:);

%% Is the System controlable?
% Compute the controlability matrix
Co = ctrb(Arel, Brel);

% If the rank of the matrix is equal to the system's states then the system
% is controlable
rang = rank(Co);
[m, n] = size(Arel);
if (rang < n)
    disp('r < n --> System is not controlable!');
else
    disp('System controlable!');
end

%% Poles of the system
%poles = pole(vehicle)
poles = eig(Arel)

%% LQR Design
% define the penalizing matrices Q and R
% simplest case -> unity matrices

R = 1;              % single input
% introducing q matrix
q1 = 0;             % weighting factor steer angle
q2 = 0;             % weighting factor steer rate
q3 = 1;             % weighting factor lean angle
q4 = 0;             % weighting factor lean rate
Q = [q1 0 0 0; 0 q2 0 0; 0 0 q3 0; 0 0 0 q4];

F = lqr(Arel, Brel, Q, R)

%% Poles of the closed loop system
p_real = eig(Arel-Brel*F)
```

C.4 Path Generation

```
% ButterFil.m
% Butterworth filtered path generation
% A path is generated w.r.t. random numbers
% These random numbers are filtered in such a way that the result is a
% smooth trackable path

clear all;
clc;

%n = 91;
n = 200;
t=linspace(0,50,n);
v = 6;
x = v*t;

y = 80*rand(size(x));
% Offset elimination
y = y - mean(y);
% Butterworth Filter
%[b,a] = butter(5,0.12);
[b,a] = butter(7,0.05);

f = filter(b,a,y);
figure(1)
plot(x, y, x, f)
title('Path (Roadway)');
xlabel('fixed x-direction [m]');
ylabel('lateral distance [m]');
legend('original data', 'filtered data')

%% Curvature calcutlation
% symbolic precalculation
```

```
Δ = x(2)−x(1);
en = diff(f, 2)/Δ^2;
den = sqrt(1 + (diff(f)/Δ).^2 ).^3;

C = en./den(1:n−2);
figure(2)
plot(x(1:n−2), C, 'o')

%% motorcycle lean angle
g = 9.81;
phi = atan(v^2*C/g);
figure(3)
plot(x(1:n−2), phi)

%% Path angle calculation
psi = diff(f)/Δ;
figure(4)
plot(x(1:n−1), psi)

%% store path and curvature in table functions
% Positon Table
pos_tab = [x(1:n−2)', f(1:n−2)'];
save('Position.mat', 'pos_tab', '−v4');

% Curvature Table
cur_tab = [x(1:n−2)', C(1:n−2)'];
save('Curvature.mat', 'cur_tab', '−v4');

% Lean Angle
lean_tab = [x(1:n−2)', phi(1:n−2)'];
save('LeanAngle.mat', 'lean_tab', '−v4');

% yaw angle
yaw_tab = [x(1:n−2)', psi(1:n−2)'];
save('YawAngle.mat', 'yaw_tab', '−v4');
```