BRiM: A Modular Bicycle-Rider Modeling Framework

Stienstra, T.J.; Brockie, S.G.; Moore, J.K.

DOI
10.59490/660179a06bf1082286458109

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the 5th Symposium on the Dynamics and Control of Single-track Vehicles

The Evolving Scholar Vol. 3, 2023

http://dx.doi.org/10.59490/660179a06bf1082286458109

Bicycle and Motorcycle Dynamics 2023
Symposium on the Dynamics and Control of Single Track Vehicles
18–20 October 2023, Delft University of Technology, The Netherlands

Type of the Paper: Conference Paper

Revised

BRiM: A Modular Bicycle-Rider Modeling Framework
[version 2; peer reviewed]

Timótheüs J. Stienstra[1], Samuel G. Brockie[1], Jason K. Moore[1,*]

[1] Faculty of Mechanical, Maritime and Materials Engineering (3mE), Delft University of Technology, The Netherlands; j.k.moore@tudelft.nl, ORCID 0000-0002-8698-6143

[*] corresponding author

Name of Editor: Edwin De Vries
First published: 19/09/2023
Last published: 02/04/2024
Citation: Stienstra, T., Brockie, S. & Moore, J. (2024). BRiM: A Modular Bicycle-Rider Modeling Framework [version 2; peer reviewed]. The Evolving Scholar - BMD 2023, 5th Edition.

Abstract:

The development of computationally efficient and validated single-track vehicle-rider models has traditionally required handcrafted one-off models. Here we introduce *BRiM*, a software package that facilitates building these models in a modular fashion while retaining access to the mathematical elements for handcrafted modeling when desired. We demonstrate the flexibility of the software by constructing the Carvallo-Whipple bicycle model with different numerical parameters representing different bicycles, modifying it with a front fork suspension travel model, and extending it with moving rider arms driven by joint torques at the elbows. Using these models we solve a lane-change optimal control problem for six different model variations which solve in mere seconds on a modern personal computer. Our tool enables flexible and rapid modeling of single-track vehicle-rider models that give precise results at high computational efficiency.

Keywords:

Bicycle Dynamics, Brim, Computational Modeling, Open-source, Sympy, Simulation, Trajectory Tracking Problem

ISSN: 2667-2812

 http://dx.doi.org/10.59490/660179a06bf1082286458109

Introduction

Throughout the 200 year history of the bicycle, numerous researchers have developed mathematical models to investigate various aspects of bicycle dynamics and rider control (Schwab and Meijaard, 2013). These models have contributed valuably to our understanding of self-stability (Meijaard et al., 2011), active and passive rider control (Moore, 2012; Schwab et al., 2012; Sharp, 2008), and the identification of specific eigenmodes (Sharp, 1976). These insights have also informed the development of bicycles with improved stability, handling, and comfort (Plöchl et al., 2012).

Mathematical bicycle models have been created using both numeric and symbolic approaches, and a combination of the two. Furthermore, these have been facilitated by many different languages and software packages. For example, Meijaard et al. (2007) used the dynamics modeling software *SPACAR* (van Soest et al., 1992) to numerically derive the equations of motion (EoMs) for the Carvallo-Whipple bicycle model. The same thing has been done symbolically by Sharp (2008), who used Matlab's Symbolic Toolbox (The MathWorks Inc., 2023), and by Moore (2012) using both the symbolic dynamics package *AUTOLEV* (Levinson and Kane, 1990) and Python's computer algebra package *SymPy* (Meurer et al., 2017).

One commonality between these numeric and symbolic approaches is that they often handcraft the EoMs derivation. This approach is advantageous in that the implementer controls the choice of coordinates that are used, often optimizing the simplicity of the resulting EoMs. Indeed, handcrafted EoMs offer the highest reachable computational performance (Rosenthal and Sherman, 1986).

Many research grade software packages, like *SPACAR*, *Simbody* (Sherman et al., 2011), and *ADAMS* (Ryan, 1990), efficiently form the EoMs for a multibody system, but these resulting EoMs exist solely in the form of numeric computer code and lack accessibility to be interrogated. Modern physics engines, like *MuJoCo* (Todorov et al., 2012) and *Nimble* (Werling et al., 2021), also offer numerical computation of high-accuracy dynamics. However, these have not gained traction in bicycle dynamics research, perhaps due to the focus on general applicability with varying assumptions in contact dynamics. *FastBike* (Dynamotion, 2023) simulates single-track vehicles, but remains proprietary, and *JBike6* (Dressel, 2006) only calculates the eigenvalues of one model.

Symbolics offer transparency, allowing for a clear understanding of the underlying equations and enabling greater flexibility in manipulating and interpreting the resulting expressions. There is scope for a symbolic approach to result in more performant code through processes like term rewriting to minimize the number of expressions in the EoMs (Gowda et al., 2022). One downside is that model derivation is typically more expensive than with numeric approaches. However, for bicycle models this cost is often still only of the order of seconds and can be amortized if the resulting model is cached and reused.

Despite the extensive research, creating accurate and performant mathematical bicycle models remains a common challenge. As pointed out by Meijaard et al. (2007), numerous published models exhibit mistakes in their derivation. Nowadays, many researchers use the linearized Carvallo-Whipple model (Meijaard et al., 2007) as starting point and extend it to incorporate additional features such as tire models (Limebeer and Sharp, 2006; Plöchl et al., 2012; Schwab and Meijaard, 2013) or rider attachments (Moore, 2012). This approach is error prone (Schwab and Meijaard, 2013), time consuming, and hinders the development of more complex models. It also reduces research dissemination and reproducibility as models may not be compatible with one another due to the usage of different conventions and programming languages, or dependence on closed-source software.

BRiM targets two main user groups: researchers wanting to use accessible and validated bicycle-rider models ("*model users*"), and researchers developing their own modifications of, or extensions to, bicycle-rider models ("*model developers*"). To address the needs of the former group, *BRiM* provides a library of composable bicycle and bicycle-rider models that users can apply to their own research questions directly out of the box. Users can trust the accuracy of these as, where possible, they are prevalidated, in addition to the source code being open for review and critique. For the second user group, *BRiM* provides a framework for researchers to develop their own modular submodels and seamlessly integrate these into their bicycle and bicycle-rider models. Through the usage of *BRiM*, these extensions can more easily be shared between researchers due to the common tooling.

To summarize, our contributions are twofold. First is the development and release of the open-source package *BRiM* for bicycle-rider modeling, including its library of composable bicycle-rider models and frameworks for users to implement their own extensions, as well as utilities for parametrization, simulation, and visualization. Second is the demonstration of *BRiM* by formulating and solving multiple novel trajectory tracking optimization problems of bicycle and bicycle-rider models, including a comparison of the minimized steer torques required for a lane change maneuver for three different bicycle geometries, plus a comparison between a rigid fork and front suspension bike, and a comparison between the steering torque and elbow torques, for the same maneuver.

ISSN: 2667-2812

 http://dx.doi.org/10.59490/660179a06bf1082286458109

Software Overview

BRiM is built on top of the `physics.mechanics` module of *SymPy*, a feature-rich and well-tested codebase for creating symbolic EoMs for complicated multibody systems. *BRiM*, via *SymPy*, uses Kane's method (Kane and Levinson, 1985) to form the EoMs of the system. Unlike classical methods for multibody dynamics, like Newton-Euler, Lagrange, and Hamilton, Kane's method leads directly to simpler EoMs for a system (Kane and Levinson, 1980; Rosenthal and Sherman, 1986). It was also designed to be systematic, making it easy to translate into a computational implementation. While other algorithms exist for efficiently computing a system's dynamics (Featherstone, 2008; Jain, 2010), these have been optimized for numerics. Consequently, Kane's method is a popular algorithm when deriving symbolic EoMs (Levinson and Kane, 1990).

BRiM provides three different levels of abstraction for constructing and interacting with bicycle-rider models. All three levels can be used interchangeably with one another, which is possible because they all either directly use, or abstract down to using, the symbolic building blocks defined with *SymPy*. At a high level, aimed predominantly at model users, *BRiM* offers a library of prebuilt modeling components that can be easily connected in a modular fashion to create a full bicycle or bicycle-rider model with minimal boilerplate code. At this level, the user is able to, for example, swap out a knife-edge wheel model for a toroidal wheel model in a single statement.

An intermediate-level interface allows the *SymPy* constructs for joints, bodies, and loads to be used directly to replace, extend, and augment models that do not exist at the component level. This level also allows model developers to readily transform these custom models into *BRiM* constructs that can then be used at the component level with an equally light interface as model components present within the package, thus making them suitable for sharing with, and use by, model users.

For advanced users, *BRiM* also allows direct manipulation of the *SymPy* equations, like the customization of the direction cosine matrices between reference frames. This enables modelers to make optimizations to the system's EoMs where their insight can simplify the model definition. It also allows the generated EoMs to be interrogated and transformed, which can be invaluable when debugging or optimizing custom models.

BRiM Core

BRiM has three types of components, models, connections, and load groups, with which a multibody dynamics system is described. A "*model*" is an object that represents a specific system within defined system boundaries. A model can be standalone or composed of multiple submodels. Each model encapsulates the relations and behaviors of the system, allowing for a modular and hierarchical tree representation where parent models do not know the details of their submodel children. Each subsystem is treated independently within its respective system boundary, which is simpler to model. Once the subsystems have been defined separately, they can be merged to form the complete system, with parents defining the interactions between their children. This modular approach not only simplifies the organization of the model but also promotes reusability and flexibility in system design.

A "*connection*" is a utility of a parent model to define relationships between two or more child submodels. Connections provide three key features. Firstly, connections enable parent models to define the interactions between their child submodels in a modular fashion. Secondly, connections enable the reuse of these possibly complex interactions. An example of both is the interaction between a wheel and the ground, where multiple complex descriptions of tire models are possible. Using connections these tire models can be defined once and then be reused in all bicycle models in *BRiM*. Lastly, connections allow new properties to be incorporated into a model. For example, when modeling a bicycle with a leaning rider, the axis should be defined about which the rider leans. However, if a bicycle is being modeled without a rider then this axis is not needed and so not considered.

A "*load group*" is a collection of related loads (forces and torques), which can be applied to an associated model or connection. Loads are separated from models because it is unlikely that a model will universally require specific loads in all circumstances. For example, consider a load group whose task is to provide the inputs to steer a bicycle model. The simplest option here would be to apply a simple time-varying torque actuator acting about the steering axis. If a rider model was also being used, a more complex load group could consist of two time-varying torque actuators acting one each at the rider's right and left elbows.

After a user has configured a model using submodels, connections, and loads, *BRiM* uses the following five-step algorithm to define the multibody system and establish all relationships required to form the system's EoMs. First, parent models associate their child submodels to their connections, which ensures that all connections have access to the information they require. Next, the "*define objects*" step creates objects such as symbols and reference frames, without defining any relationships between them. After this, the "*define kinematics*" step establishes relationships between the objects' orientations/positions, velocities, and accelerations. Next,

ISSN: 2667-2812

 http://dx.doi.org/10.59490/660179a06bf1082286458109

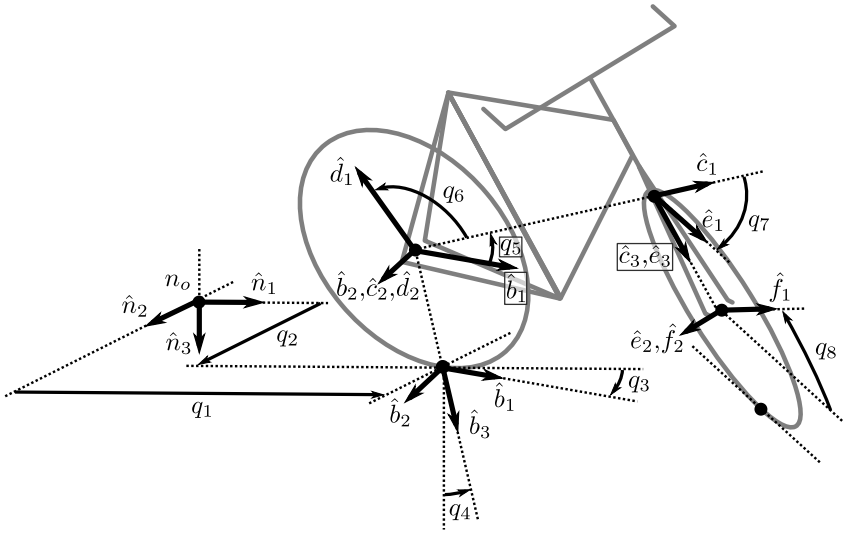

Figure 1. Configuration of the Carvallo-Whipple bicycle model following the convention from Moore (2012), where q_1 and q_2 are the perpendicular distances of the rear contact point in the ground plane, q_3, q_4, and q_5 are the yaw, roll, and pitch angles of the rear frame relative to the ground, q_7 is the steering rotation angle, and q_6 and q_8 are the rear and front wheel rotation angles.

the "*define loads*" step specifies the forces and torques acting on the system. Lastly, the "*define constraints*" step computes the holonomic and nonholonomic constraints on the system.

In each define step, a model first calls the define step for each of its submodels using a depth-first traversal, which is required because parent models may use properties of their submodels. Next, the model defines itself while triggering the define step of connections. These are initiated manually by the parent model because it is possible for there to be two-way dependence between parent models and their associated connections in a single define step. Finally, the model calls the define step for each of its associated load groups.

Upon completion of the above steps, all relationships within the model are defined. At this stage, the end-user can export the model to a `System` instance from the `physics.mechanics` module of *SymPy*, which can automatically form the EoMs.

BRiM Models

While it is possible to implement any bicycle model using *BRiM*, *BRiM* distributes a complete implementation of the Carvallo-Whipple bicycle model (Carvallo, 1899; Whipple, 1899) following the parameterization convention of Moore (2012). The Carvallo-Whipple bicycle is widely recognized as the lowest-order bicycle model to have been repeatedly validated by multiple authors (Kooijman et al., 2008; Schwab and Meijaard, 2013; Sharp, 2008). The parametrization convention of Moore, as shown in Figure 1, has been chosen due to its configuration independence, which is preferred when forming nonlinear EoMs of a bicycle, (Peterson, 2013). By default, the Carvallo-Whipple bicycle consists of a ground and four bodies: a rear wheel, a rear frame, a front frame, and a front wheel. Each body is assumed to be rigid and all bodies are attached using pin joints. The rear wheel's contact point is defined within the ground plane and the rear frame is oriented in yaw-roll-pitch rotation relative to the ground. The front wheel's contact point is constrained to the ground using a holonomic constraint, and nonholonomic constraints are applied to both wheels to enforce no-slip conditions. Further details of the Carvallo-Whipple bicycle model's definition are found in Moore (2012).

To facilitate the implementation of extensions, like toroidal-shaped wheels and tire models (Limebeer and Sharp, 2006; Schwab and Meijaard, 2013), to the Carvallo-Whipple bicycle in a modular fashion, *BRiM* splits the bicycle model into separate models for each body. This division results in a total of five submodels: ground, rear wheel, front wheel, rear frame, and front frame. The modularity of the tire models is achieved by using connections to describe the interaction between the ground and a wheel. A schematic overview is shown in Figure 2.

Published rider models tend to either fall into the category of upper-body models (Moore, 2012) or pedaling models (Park et al., 2022). To support varying complexities of both types of model, *BRiM* segments the rider into optional submodels and connections.

ISSN: 2667-2812

 http://dx.doi.org/10.59490/660179a06bf1082286458109

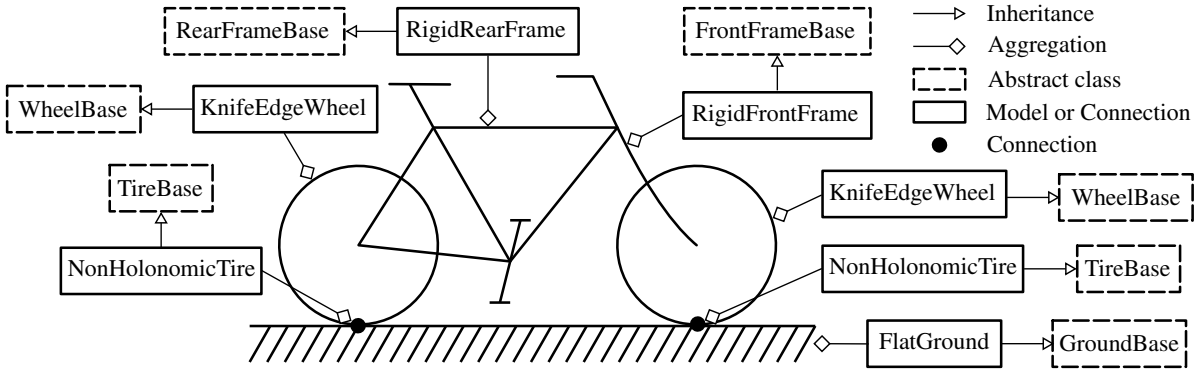

Figure 2. Schematic of the Carvallo-Whipple bicycle in *BRiM* aggregating from its constituent parts. The blueprint of these parts is described in abstract classes and is copied through inheritance.

The only mandatory submodel is the pelvis because it is shared by all rider models. The other five submodels forming the rider are the torso, left and right legs, and left and right arms. The relationships between these are described by five connections representing the various joints: the left and right hips, the left and right shoulders, and the sacrum connecting the torso to the pelvis. To extend a bicycle model with a rider, connections at the saddle, the handlebars, and the pedals are required.

Benchmarks

A convenient proxy for the computational performance of the derived EoMs is the number of floating point operations required to evaluate them. The output of *BRiM* has been compared to handcrafted EoMs of the Carvallo-Whipple bicycle by Moore (2012); Stienstra (2023a), both of which were also derived using *SymPy*. The version by Moore utilizes writing the equations as one might on paper to reduce the complexity resulting in 2198 operations. The version by Stienstra mainly utilized the intermediate-level interface in *SymPy* resulting in 2389 operations. *BRiM*'s output is in between the two containing 2291 operations, highlighting that *BRiM* is able to produce efficient EoMs on par with handcrafted derivations despite its modularity and user-friendly interface.

Demonstrations Methodology

To demonstrate the wide applicability, capabilities, and potential of *BRiM*, we solved a series of related trajectory tracking optimization problems for bicycle and bicycle-rider models. The goal here is to showcase the modularity and extensibility of *BRiM* with examples, and evidence that it can be readily used in novel applications, reducing the barriers to generating new results.

Bicycle and Bicycle-Rider Models

In the demonstrations, the same trajectory is tracked but the bicycle (or bicycle-rider) model involved differed. All six of these models were built entirely using *BRiM*. The specifics of the model used in each of the six optimization cases are outlined below.

- Optimization #1: This is the default Carvallo-Whipple bicycle model without the inertial effects of a rider. It uses the parameter values from a Batavus Browser Dutch style city bicycle (Moore, 2012), `"Browser"`. The constituent submodels used to create this instance are shown in Figure 2. The system's inputs are a torque applied to the rear wheel about the wheel hub axis (the "*propulsion/braking torque*") and a torque actuator applied between the handlebars and the rear frame such that they are actuated about the steering axis (the "*steering torque*").
- Optimization #2: This is the same as optimization #1 but with the parameter values from a Bianchi Pista steel frame track bicycle (Moore, 2012), `"Pista"`.
- Optimization #3: This is the same as optimization #1 but with the parameter values from a Gary Fisher hard-tail mountain bicycle (Moore, 2012), `"Fisher"`.

ISSN: 2667-2812

- Optimization #4: This modifies optimization #3 by replacing the rigid front frame with a fork suspension on soft settings.
- Optimization #5: This is a bicycle-rider model. It extends optimization #1 by including the inertial effects of a rigidly attached rider including arms, where the shoulders are modeled to allow flexion and rotation, and the elbows are pin joints.
- Optimization #6: This modifies optimization #5 by replacing the "*steering torque*" with a pair of torque actuators at the elbows of the rider model (the "*elbow torques*").

Complete scripts for constructing these six models, plus formulating and running the trajectory tracking optimization problem, can be found in the GitHub repository for this paper: `github.com/mechmotum/brim-bmd-2023-paper`.

Equation Generation

BRiM automatically generates the EoMs of the system. For the base Carvallo-Whipple bicycle model (optimizations #1–3), which follows the convention of Moore (2012), *BRiM* produces a system defined in terms of eight generalized coordinates and eight generalized speeds. For the optimizations, we choose a set of seven ($Q = 7$) of the generalized coordinates ($\mathbf{q}_{ind}$) and three ($U = 3$) generalized speeds ($\mathbf{u}_{ind}$) to be independent. These are

$$\begin{aligned}\mathbf{q}_{ind} &= [q_1, q_2, q_3, q_4, q_6, q_7, q_8] \ , \\ \mathbf{u}_{ind} &= [u_4, u_6, u_7] \ .\end{aligned}$$

With these, we can formulate the 16 state equations that will enforce the system dynamics in the optimization problem. The first eight are the kinematic differential equations that map the time derivative of each generalized coordinate to the corresponding generalized speed

$$\dot{q}_i = u_i \qquad i \in \mathbb{N} \cap [1, 8] \ . \tag{1}$$

The next three are dynamic differential equations, each one corresponding to the time derivative of the three independent generalized speeds

$$\mathbf{M}\dot{\mathbf{u}}_{ind} - \mathbf{k} = \mathbf{0} \, , \tag{2}$$

where $\mathbf{M}$ is the 3×3 mass matrix of the bicycle model and $\mathbf{k}$ is its length-3 forcing column vector encompassing both externally applied forces and forces due to various velocity effects.

This is followed by a single differential algebraic equation that enforces the holonomic constraint to keep the front wheel in contact with the ground. Finally, four differential algebraic constraints, two each per wheel, enforce the nonholonomic constraints that provide the no-slip conditions for the front and rear wheels.

In optimization #4, one extra independent generalized coordinate (q_s) describing the suspension travel and one extra independent generalized speed (u_s) describing the suspension velocity are introduced. Consequently, $Q = 8$ and $U = 4$. This results in the addition of one extra kinematic differential equation involving $\dot{q}_s$ and one extra dynamic differential equation involving $\dot{u}_s$.

Optimizations #5–6, use the same base generalized coordinates and generalized speeds as optimizations #1–3. However, due to the addition of the rider, an additional six dependent generalized coordinates are introduced. For each of the two arms, two are for the flexion and rotation of the shoulder joint and one is for the flexion of the elbow joint. Additionally, each of these dependent generalized coordinates maps to a dependent generalized speed so an additional six of these are also required. Q and U are unchanged because no independent generalized coordinates or speeds are introduced. To keep the hands attached to the handlebar grips, six more holonomic constraints are used. In terms of additional state equations to enforce the dynamics, six additional kinematic differential equations are introduced, alongside six additional differential algebraic equations to enforce the holonomic constraints.

Let $\mathbf{r}$ be the vector of model inputs and R be its length. For optimizations #1–5, $R = 2$ and $\mathbf{r} = [T_p, T_s]$, where T_p and T_s are the propulsion/braking and steering torques respectively. For optimization #6, $\mathbf{r} = [T_p, T_l, T_r]$ and $R = 3$, where T_l and T_r are the left and right elbow torques respectively.

ISSN: 2667-2812

Tracking Problem

In our trajectory tracking optimization problems, the task is to follow a target ground path representing a lane change. The path is defined by the equation

$$\bar{q}_2\left(q_1\right) = \begin{cases} 0\,, & q_1 < X_s \\ \frac{1}{2} X_2 \left(1 - \cos\left(\frac{q_1 - X_s}{X_1 - 2X_s}\pi\right)\right)\,, & X_s \leq q_1 \leq X_1 - X_s\,, \\ X_2\,, & X_1 - X_s < q_1 \end{cases} \tag{3}$$

where q_1 and q_2 are the states describing the longitudinal and lateral position of the rear wheel contact point in the Newtonian coordinate system respectively, $\bar{q}_2$ is the target lateral position as a function of q_1, X_1 and X_2 are the longitudinal length and lateral displacement of the target path respectively, and X_s is the length of straight at the beginning and end of the path.

The objective function J simultaneously minimizes both the tracking error and the input magnitudes, to find controls that minimize the approximated input energy used in the maneuver. It is defined as

$$J = \int_{t_0}^{t_F} (1 - w)\left(q_2 - \bar{q}_2\left(q_1\right)\right)^2 dt + w \sum_{i=1}^{R} \int_{t_0}^{t_F} r_i^2 dt\,, \tag{4}$$

where t_0 and t_F are the initial and final times, r_i is the ith input variable, and w is a weighting parameter between the squared tracking error and the sum of squared inputs. We selected $w = 2.5 \cdot 10^{-3}$, which aims to target a mean tracking error of $0.025\,\mathrm{m}$ for an approximate torque of $0.5\,\mathrm{N\,m}$.

We add constraints defining the initial and final state of the system's motion to be in the nominal configuration and at a nominal travel speed. In optimizations #1–6 we enforce

$$\begin{aligned} q_1\left(t_1\right) &= 0.0 & q_1\left(t_F\right) - X_1 &= 0.0 & q_6\left(t_1\right) &= 0.0 & \\ q_2\left(t_1\right) &= 0.0 & q_2\left(t_F\right) - X_2 &= 0.0 & q_7\left(t_1\right) &= 0.0 & q_7\left(t_F\right) &= 0.0 \\ q_3\left(t_1\right) &= 0.0 & q_3\left(t_F\right) &= 0.0 & q_8\left(t_1\right) &= 0.0 & \\ q_4\left(t_1\right) &= 0.0 & q_4\left(t_F\right) &= 0.0 & & & \end{aligned}$$

where the second node at t_1 is used for the initial state because the inputs do not affect the first node when using backward Euler discretization. Additionally, in optimization #4 the generalized coordinate and generalized speed for the suspension are constrained at t_1 such that the suspension compression and weight through the handlebars are in equilibrium.

Initial Guess

When solving an optimization problem using direct collocation, the initial guess provided can influence the convergence properties of the problem, or even whether the problem solves at all (Betts, 2010). For consistency between the six optimization cases, we used equivalent initial guesses, only accounting for the inclusion or exclusion of different states and inputs as decision variables.

The initial guess in each optimization was created using a forward simulation that aimed to simultaneously satisfy $\tilde{q}_1\left(t_0\right) = 0$, $\tilde{q}_2\left(t_0\right) = 0$, $\tilde{q}_1\left(t_F\right) = X_1$, and $\tilde{q}_2\left(t_F\right) = X_2$. The tilde above a variable denotes that this is the initial guess corresponding to that variable. The forward simulation from t_0 until t_F was conducted under null inputs ($\tilde{r}_i = 0, i \in \mathbb{N} \cap [1, R]$). Due to the null inputs, the initial state required modification such that the bicycle rides diagonally between the endpoints:

$$\tilde{q}_3\left(t_0\right) = \arctan\left(\frac{X_2}{X_1}\right) \qquad \tilde{u}_4\left(t_0\right) = 0 \qquad \tilde{u}_6\left(t_0\right) = -\frac{\sqrt{X_1^2 + X_2^2}}{r_R\left(t_F - t_0\right)} \qquad \tilde{u}_7\left(t_0\right) = 0\,.$$

As the system dynamics are defined as differential algebraic equations, the `IDA` differential algebraic equation solver (Gardner et al., 2022; Hindmarsh et al., 2005) from the *ODES* Scikit (Malengier et al., 2018) was used to conduct the forward simulation. The resulting initial guesses are dynamically feasible, with the only active instance constraint violations being

$$q_3\left(t_0\right) = \arctan\left(\frac{X_2}{X_1}\right) \neq 0 \qquad q_3\left(t_F\right) = \arctan\left(\frac{X_2}{X_1}\right) \neq 0\,.$$

ISSN: 2667-2812

 http://dx.doi.org/10.59490/660179a06bf1082286458109

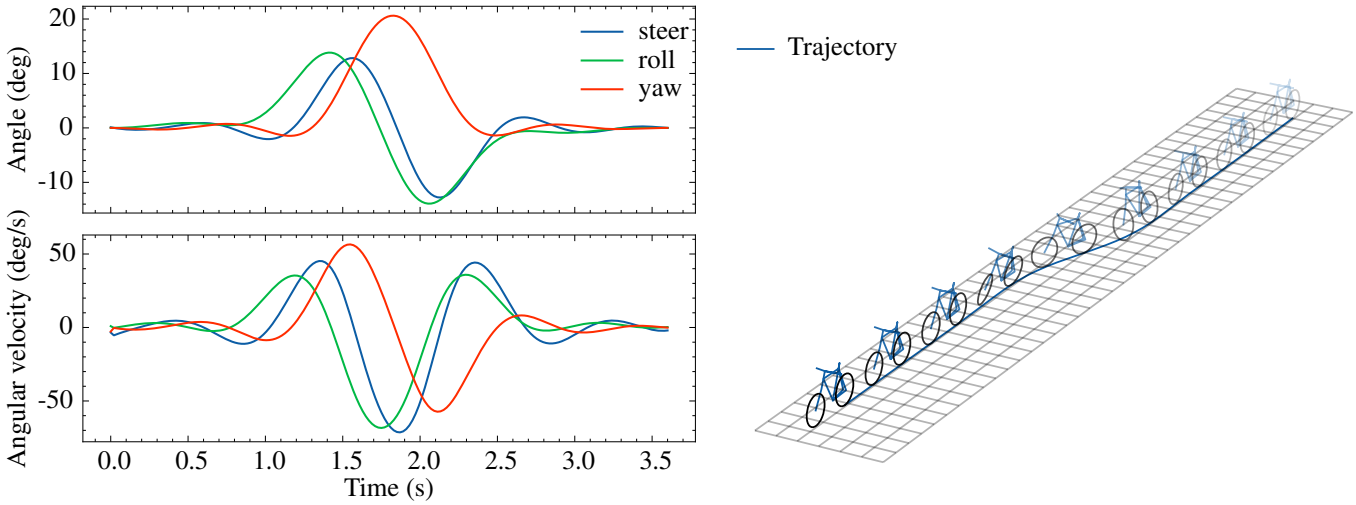

Figure 3. State plot and time-lapse of optimization #1 created using *SymMePlot* (Stienstra, 2023b).

Optimization

We solved each of the six trajectory tracking optimization problems using *opty* (Moore and van den Bogert, 2018), a Python package for solving optimization problems involving dynamic systems using direct collocation (Betts, 2010). *opty* is designed with *SymPy* in mind and thus interfaces seamlessly with *BRiM*. *opty* transcribes the optimization problem into a nonlinear programming problem, which is then solved numerically using the interior-point solver *Ipopt* (Biegler and Zavala, 2009). Each optimization problem was transcribed with 180 nodes, resulting in a node every $0.02\,\mathrm{s}$, and the dynamics were collocated using the backward Euler method. *Ipopt*'s default settings were used with the exception of enabling gradient-based scaling.

Results

All optimizations were successfully solved to the *Ipopt* converge tolerance from the stated initial guesses. Figure 3 shows the steer, roll, and yaw angle and angular velocity states for the optimal solution to optimization #1. Additionally, Figure 3 includes a sequence image of the Browser bicycle traversing the optimal trajectory. This was produced using *SymMePlot* (Stienstra, 2023b), a visualization library build on top of *SymPy*, which has been integrated with *BRiM*. A comparison of the optimal trajectories for all six optimizations is shown in Figure 4, along with comparisons of the optimal steer and roll angle states.

The optimal controls are shown in Figure 5. Comparisons are shown for the optimal propulsion/braking torque for all optimizations and optimal steering torque for optimizations #1–5. For optimization #6, where there was no steering torque, the optimal left and right elbow torques are included. A final subplot compares the tracking error of the optimal trajectory in each problem.

Figure 4 shows that all bicycles follow roughly the same trajectory. Optimizations #1–4, which were riderless, have consistent states and inputs, with the largest difference being seen in optimization #1 where the bicycle is the Batavus Browser. Optimization #5–6, which included the rider, exhibit trajectories with longer transitions and smoother states while also having an increased input cost. Comparing optimizations #5 and #6, it can be seen that the optimal torques in optimization #6 are of larger magnitude while resulting in almost the same trajectory and states.

Table 1 provides a range of metrics related to the optimization problem solves for each of the six optimization problems. These include the achieved optimal cost, and the tracking and input costs that contributed to these, along with the mean tracking error for each optimal trajectory. The number of iterations taken by *Ipopt* is provided to measure the convergence performance, and by association, the computational performance, in solving each problem.

ISSN: 2667-2812

http://dx.doi.org/10.59490/660179a06bf1082286458109

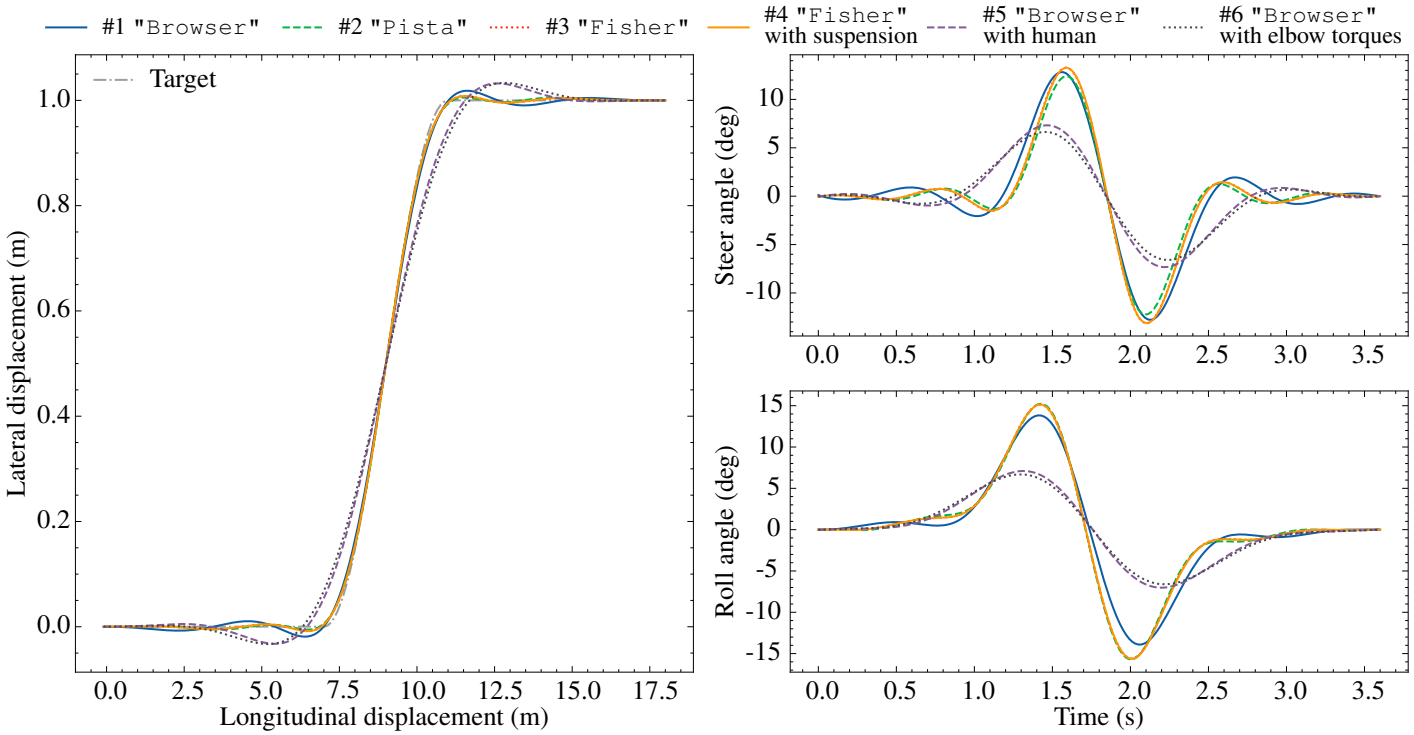

Figure 4. Comparison of the errors between the optimal and target trajectories, the optimal steer angles, and optimal roll angles for optimizations #1–6. Note that the trajectory plot has non-equal axes.

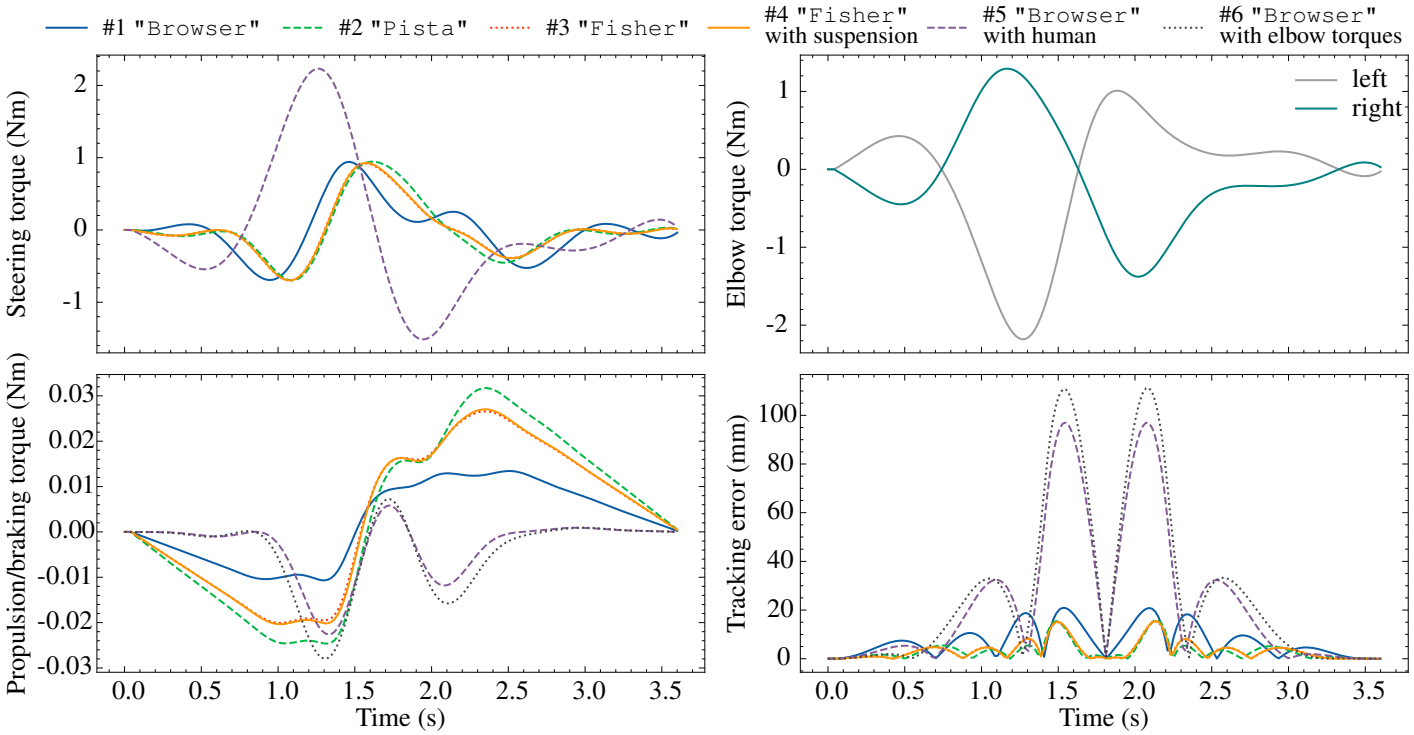

Figure 5. Comparison of the optimal inputs for optimizations #1–6. Note that #6 is missing in the steering torque graph because elbow torques are used.

ISSN: 2667-2812

 http://dx.doi.org/10.59490/660179a06bf1082286458109

Optimization	Optimal cost	Tracking cost	Input cost	Mean tracking error (m)	# NLP iterations	Time in *Ipopt* (s)
#1	1.579×10^{-3}	3.267×10^{-4}	5.011×10^{-1}	9.526×10^{-3}	47	1.2
#2	1.434×10^{-3}	8.676×10^{-5}	5.392×10^{-1}	4.909×10^{-3}	76	2.3
#3	1.276×10^{-3}	9.468×10^{-5}	4.726×10^{-1}	5.128×10^{-3}	74	1.9
#4	1.288×10^{-3}	9.494×10^{-5}	4.772×10^{-1}	5.135×10^{-3}	77	2.4
#5	1.253×10^{-2}	5.175×10^{-3}	2.948	3.792×10^{-2}	25	3.5
#6	1.656×10^{-2}	6.864×10^{-3}	3.883	4.367×10^{-2}	23	2.5

Table 1. Solution and convergence metrics for optimization #1–6. Computational metrics were produced using an *Intel(R) Core(TM) i9-13900K CPU @ 3.00GHz*.

Discussion

In the optimal controls for all six problems, countersteering at the start acts to roll the bicycle toward the centerline of the target path while steering the bicycle away from it. This is to be expected as the mass center must be inside the wheels during a turn for the centrifugal forces and centripetal accelerations to balance the bicycle.

All optimal trajectories generally cross the target path in the middle. This can be explained by the target path's rotational symmetry and the periodicity constraints imposed at t_0 and t_F in all of the optimization problems. The propulsion/braking torque remains low in each optimization, as the path can mostly be followed without braking. The increased tracking error of optimization #1 compared to the other riderless optimization #2–4 can be explained by the lower agility of the Batavus Browser bicycle. It is designed to be a stable city bicycle and is almost twice as heavy as the Gary Fisher and Bianchi Pista bicycles.

The solutions to optimizations #3 and #4 are similar by multiple measures. Qualitatively, the optimal states (Figure 4) and inputs (Figure 5) appear nearly identical. As the target trajectory follows a path on flat ground, it should be expected that there will be very little to no travel in the suspension during the maneuver, and therefore the presence of suspension should have little effect on the results.

Comparing the solutions of optimizations #1 and #5, the shape of the optimal trajectories differs with much larger deviations from the target path at the transitions between the straights and the chicane being exhibited in optimization #5. This is to be expected as the addition of the rider results in a large increase in the mass and inertia of the system, and an increase in the height of the mass center. The optimal cost is approximately eight times as large in optimization #5, which can be attributed in equal parts to both the tracking and input costs.

The trajectories in optimizations #5 and #6 have the same shape, which is expected as the inertia is the same. Only the method of how the steering torque is applied differs. In optimization #6, the pair of elbow torques are relatively symmetrical to one another. It is more efficient in terms of the objective function for both elbow torques contribute equally to steering. Still, the combined magnitude of the elbow torques is relatively larger than the steering torque in optimization #5 because the efficiency of each arm is dependent on the steering angle.

Conclusion

This paper has presented *BRiM*, an open-source package for bicycle-rider modeling. *BRiM* enables model users to readily create bicycle and bicycle-rider models from its library of composable submodels. It also facilitates model developers to implement their own models and extensions, which can be used in tandem with all other aspects of the package. Additionally, benchmarking of the EoMs generated by *BRiM* shows that these are of comparable performance to handcrafted EoMs.

BRiM's functionality has been demonstrated by formulating and solving multiple novel trajectory tracking optimization problems of bicycle and bicycle-rider models. A comparison of the minimized steer torques required for a lane change maneuver for three different bicycle geometries has showcased how *BRiM* enables simple reparametrization of models. The modularity of *BRiM* has been illustrated by comparing a bicycle model with a rigid fork to one with suspension. By attaching a rider that can be actuated in a different way to a bicycle, *BRiM*'s models have been shown to be easily extended and modified.

A limitation of *BRiM* is that it forces model developers to use an existing interface for augmented model components. These

ISSN: 2667-2812

interfaces can limit flexibility, especially if a component requires additional properties to be interacted with. While connections largely solve this problem, an over-usage of them can result in a cluttering of the implementation code.

Further development of *BRiM* continues. Work is currently underway to add further models to *BRiM*, improve the package's ability to interface with simulation and optimization tools like *opty*, and develop further teaching materials like a tutorial workshop. Examples of models currently under development include tire models and rider models actuated by musculotendons.

An obvious next step is for other researchers to use *BRiM* in their own work. This would provide valuable feedback, which could help guide the future development of the package. *BRiM*'s permissive open-source license also means that its source code can be reviewed and critiqued by others, ensuring the accuracy and validity of its models, which will benefit all users and their work. Our hope is that *BRiM* can become a foundational tool for bicycle dynamics research, improving the ease, speed, and accuracy of work done in this area, by helping researchers to develop and share their models in the future.

Acknowledgments

This project has been made possible in part by CZI grant CZIF2021-006198 and grant DOI https://doi.org/10.37921/240361looxoj from the Chan Zuckerberg Initiative Foundation (funder DOI 10.13039/100014989).

References

Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming*. SIAM.

Biegler, L. T. and Zavala, V. M. (2009). Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3):575–582.

Carvallo, E. (1899). *Théorie du mouvement du monocycle et de la bicyclette.* Gauthier-Villars.

Dressel, A. E. (2006). The Benchmarked Linearized Equations of Motion for an Idealized Bicycle (Implemented in Software and Distributed via the Internet). Master Thesis, Cornell University.

Dynamotion (2023). Fastbike. https://www.dynamotion.it/en/software/fastbike-software/.

Featherstone, R. (2008). *Rigid Body Dynamics Algorithms*. Springer.

Gardner, D. J., Reynolds, D. R., Woodward, C. S., and Balos, C. J. (2022). Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*.

Gowda, S., Ma, Y., Cheli, A., Gwóźzdź, M., Shah, V. B., Edelman, A., and Rackauckas, C. (2022). High-performance symbolic-numerics via multiple dispatch. *ACM Communications in Computer Algebra*, 55(3):92–96.

Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396.

Jain, A. (2010). *Robot and multibody dynamics: analysis and algorithms*. Springer Science & Business Media.

Kane, T. R. and Levinson, D. A. (1980). Formulation of equations of motion for complex spacecraft. *Journal of Guidance and Control*, 3(2):99–112.

Kane, T. R. and Levinson, D. A. (1985). *Dynamics, theory and applications*. McGraw Hill.

Kooijman, J. D. G., Schwab, A. L., and Meijaard, J. P. (2008). Experimental validation of a model of an uncontrolled bicycle. *Multibody System Dynamics*, 19(1):115–132.

Levinson, D. A. and Kane, T. R. (1990). AUTOLEV—a new approach to multibody dynamics. In *Multibody Systems Handbook*, pages 81–102. Springer.

Limebeer, D. and Sharp, R. (2006). Bicycles, motorcycles, and models. *IEEE Control Systems Magazine*, 26(5):34–61.

ISSN: 2667-2812

Malengier, B., Kišon, P., Tocknell, J., Abert, C., Bruckner, F., and Bisotti, M.-A. (2018). ODES: a high level interface to ODE and DAE solvers. *The Journal of Open Source Software*, 3(22):165.

Meijaard, J., Papadopoulos, J. M., Ruina, A., and Schwab, A. (2007). Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(2084):1955–1982.

Meijaard, J. P., Papadopoulos, J. M., Ruina, A., and Schwab, A. L. (2011). History of thoughts about bicycle self-stability. Technical report, Cornell.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A. (2017). SymPy: Symbolic computing in Python. *PeerJ Computer Science*, 3:e103.

Moore, J. K. (2012). *Human Control of a Bicycle*. PhD thesis, University of California, Davis.

Moore, J. K. and van den Bogert, A. J. (2018). opty: Software for trajectory optimization and parameter identification using direct collocation. *Journal of Open Source Software*.

Park, S., Caldwell, G. E., and Umberger, B. R. (2022). A direct collocation framework for optimal control simulation of pedaling using opensim. *Plos one*, 17(2):e0264346.

Peterson, L. (2013). *Bicycle dynamics: Modelling and experimental validation*. PhD thesis, University of California Davis.

Plöchl, M., Edelmann, J., Angrosch, B., and Ott, C. (2012). On the wobble mode of a bicycle. *Vehicle System Dynamics*, 50(3):415–429.

Rosenthal, D. and Sherman, M. (1986). High Performance Multibody Simulations via Symbolic Equation Manipulation and Kane's Method. *Journal of the Astronautical Sciences*, 34:223–239.

Ryan, R. (1990). ADAMS—multibody system analysis software. *Multibody Systems Handbook*, pages 361–402.

Schwab, A. L. and Meijaard, J. P. (2013). A review on bicycle dynamics and rider control. *Vehicle System Dynamics*, 51(7):1059–1090.

Schwab, A. L., Meijaard, J. P., and Kooijman, J. D. (2012). Lateral dynamics of a bicycle with a passive rider model: Stability and controllability. *Vehicle System Dynamics*, 50(8):1209–1224.

Sharp, R. S. (1976). The Dynamics of Single Track Vehicles. *Vehicle System Dynamics*, 5(1-2):67–77.

Sharp, R. S. (2008). On the Stability and Control of the Bicycle. *Applied Mechanics Reviews*, 61(6).

Sherman, M. A., Seth, A., and Delp, S. L. (2011). Simbody: Multibody dynamics for biomedical research. *Procedia Iutam*, 2:241–261.

Stienstra, T. J. (2023a). BRiM: A Modular Bicycle-Rider Modeling Framework. Master Thesis, Delft University of Technology.

Stienstra, T. J. (2023b). SymMePlot. https://github.com/tjstienstra/symmeplot.

The MathWorks Inc. (2023). Symbolic Toolbox. https://www.mathworks.com.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

van Soest, A. J., Schwab, A. L., Bobbert, M. F., and van Ingen Schenau, G. J. (1992). SPACAR: a software subroutine package for simulation of the behavior of biomechanical systems. *Journal of biomechanics*, 25(10):1219–1226.

Werling, K., Omens, D., Lee, J., Exarchos, I., and Liu, C. K. (2021). Fast and Feature-Complete Differentiable Physics for Articulated Rigid Bodies with Contact. *arXiv*, page 15.

Whipple, F. J. (1899). The stability of the motion of a bicycle. *Quarterly Journal of Pure and Applied Mathematics*, 30(120):312–348.

ISSN: 2667-2812