Efficient Derivative Evaluation for Rigid-body Dynamics based on Recursive Algorithms subject to Kinematic and Loop Constraints

Manuel Kudruss[1], Paul Manns*[2], and Christian Kirches*[2]

[1]Visteon Electronics Germany GmbH, Amalienbadstraße 41a, 76227 Karlsruhe, Germany, mkudruss@visteon.com
[2]Institute for Mathematical Optimization, Technische Universität Braunschweig, 38106 Braunschweig, Germany {paul.manns,c.kirches}@tu-bs.de

April 30, 2019

Abstract

Simulation, optimization and control of robotic and bio-mechanical systems depend on a mathematical model description, typically a rigid-body system connected by joints, for which efficient algorithms to compute the forward or inverse dynamics exist. Models that e.g. include spring-damper systems are subject to both kinematic and loop constraints. Gradient-based optimization and control methods require derivatives of the dynamics, often approximated by numerical differentiation (FD). However, they greatly benefit from accurate gradients, which promote faster convergence, smaller iteration counts, improved handling of nonlinearities or ill-conditioning of the problem formulations, which are particularly observed when kinematic constraints are involved. In this article, we apply algorithmic differentiation (AD) to propagate sensitivities through dynamics algorithms. To this end, we augment the computational graph of these algorithms with derivative information. We provide analytic derivatives for elementary operations, in particular matrix factorizations of the descriptor form of the equation of motions with additional constraints, which yields a very efficient derivative evaluation for constrained dynamics. The proposed approach is implemented within the free software package Rigid Body Dynamics Library (RBDL), which heavily employs so-called *Spatial Transformations* in its implementation of the dynamics algorithms. Thus, manipulations of *Spatial Transformations* are treated as elementary operations. The efficiency is improved further by sparsity exploitation. We validate and benchmark the implementation against its FD counterpart for a lifting motion of a human model.

Index Terms — Numerical algorithms, Robotics, Control applications

*PM and CK acknowledge funding by DFG through Priority Programme 1962. CK acknowledges financial support by BMBF, grants 05M2016-MoPhaPro, 05M17MBA-MOReNet, and 61210304-ODINE. PM acknowledges funding by the EC within the H2020 project Spexor (GA 687662).

1 Introduction

MODEL-BASED optimal control has led to impressive open-loop motion generation results for robotic systems, cf. [1, 2]. Many potential applications, like e.g. whole-body control and particularly, stability estimation and correction, ask for closed-loop motion control in real-time. Recent publications [3] show inspiring progress, but the realization in real-time remains an open challenge. Another emerging trend is to apply machine learning techniques for motion generation, c.f. [4]. Either approach relies on the efficient derivative evaluation for rigid-body dynamics (RBD) of complex multi-body systems (MBSs) subject to contacts and collisions. Furthermore, AD codes facilitate the use of certain optimal control approaches like discrete mechanics and optimal control (DMOC) [5], which requires fine first derivatives because its NLP formulation already requires them for constraint vector computation, which in turn requires the evaluation of second derivatives in gradient-based solvers. As FD of second derivatives is quite inexact, its use should be limited to the constraint Jacobian.

1.1 Related Work

Regarding the aforementioned applications, we have successfully combined DMOC with the presented AD code to solve a human bending and lifting motion in [6] and Oehler et al. have employed our code for stable obstacle traversal of mobile ground robots in [7].

Recursive algorithms are the methods of choice to evaluate common RBD quantities of MBSs subject to kinematic trees, c.f. [8]. They also form the basis for most approaches to derivative computations of RBD, and several codes are available to this end, e.g. *Simbody*[1], *drake*[2], *Metapod*[3], *Pinocchio*[4], and *RBDL*[5] [9], which implements the recursive algorithms relying on the ideas from [8]. The library *GEAR*[6] is based on [10] and implements an early approach employing a LIE group formulation for RBD. In [11], a derivation based on LAGRANGIAN formulation is presented. The library *drake* offers an AD interface using template programming. Recent codes for derivative evaluation of RBD are presented in [12, 13, 14, 15]. The implementation of [12] is realized in the *C++*-library *MBSlib*[7] by employing the operator-overloading based AD tool *ADOL-C*. Following the same idea, source code transformation is applied in [13, 14] to code generated by the RBD control toolbox *ct*[8]. [15] presents efficient derivative computations by differentiating *Pinocchio*'s recursive algorithms, and uses insights into the structure of MBSs derivatives to obtain an efficient implementation of the Recursive NEWTON-EULER Algorithm (RNEA).

1.2 Contributions of this Article

We augment the computational graph (CG) of state-of-the-art recursive algorithms to evaluate derivatives wrt. to the inputs. We follow the AD principle

[1] github.com/simbody/simbody, [2] github.com/RobotLocomotion/drake,
[3] github.com/laas/metapod, [4] stack-of-tasks.github.io/pinocchio, [5] rbdl.bitbucket.org,
[6] github.com/junggon/gear, [7] github.com/SIM-TU-Darmstadt/mbslib,
[8] https://bitbucket.org/adrlab/ct

and treat algorithms as composites of elementary operations, for which we provide analytic derivatives. Manipulations of *Spatial Transformations*, see Sect. 3.1 and 3.2, serve as elementary AD operations. The augmented CG holds derivative information by propagating directions using the chain rule.

We extend this approach to MBSs with constrained (e.g. contact) dynamics, for which we require matrix factorizations to solve the equations of motion. This case has not been addressed before. Our methodology can be implemented on top of RBD codes, which is demonstrated for the aforementioned open source library RBDL [9]. The applicability of our work for optimal control problems (OCPs) is shown for a lifting motion of a human model. Furthermore, we demonstrate the benefit of appropriate sparsity exploitation.

2 Modes of Derivative Evaluation

We introduce the different modes of derivative evaluation and provide details on the expected approximation quality and computational effort.

Definition 1 (Differentiability and (Directional) Derivative)
Let $\boldsymbol{f} : \mathcal{U} \subseteq \mathbb{R}^n \to \mathbb{R}^m$, $\mathcal{U}$ open and $\boldsymbol{x} \in \mathcal{U}$. $\boldsymbol{f}$ is *(Fréchet) differentiable* at $\boldsymbol{x}$ if a linear map $\mathrm{D}f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ exists s.t.

$$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{v}) = \boldsymbol{f}(\boldsymbol{x}) + \mathrm{D}\boldsymbol{f}(\boldsymbol{x})\boldsymbol{v} + \boldsymbol{r}(\boldsymbol{x};\boldsymbol{v})$$

for all $\boldsymbol{v}$ s.t. $\boldsymbol{x}+\boldsymbol{v} \in \mathcal{U}$ and $\frac{\boldsymbol{r}(\boldsymbol{x};\boldsymbol{v})}{||\boldsymbol{v}||} \underset{\boldsymbol{v}\to\boldsymbol{0}}{\to} \boldsymbol{0}$. The uniquely defined operator $\mathcal{U} \ni \boldsymbol{x} \mapsto \mathrm{D}\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^{m\times n}$ is called the *(first-order) derivative* of $\boldsymbol{f}$ at $\boldsymbol{x}$. Let $\boldsymbol{f} : \mathcal{U} \subseteq \mathbb{R}^n \to \mathbb{R}^m$ and $\boldsymbol{x} \in \mathcal{U}$, $\mathcal{U}$ open, and $\boldsymbol{v} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\}$. If $h \mapsto \boldsymbol{f}(\boldsymbol{x}+h\boldsymbol{v})$ is differentiable at $h = 0$,

$$\partial\boldsymbol{f}(\boldsymbol{x};\boldsymbol{v}) = \lim_{h\to 0} \frac{\boldsymbol{f}(\boldsymbol{x}+h\boldsymbol{v}) - \boldsymbol{f}(\boldsymbol{x})}{h}$$

is called the *(first-order) directional derivative* of $\boldsymbol{f}$ at $\boldsymbol{x}$ along $\boldsymbol{v}$.

2.1 Numerical Differentiation

The Taylor expansion of $\boldsymbol{f}(\boldsymbol{x})$ yields one-sided (forward) and the (improved) central finite difference schemes

$$\begin{aligned} \partial_{+h}\boldsymbol{f}(\boldsymbol{x};\boldsymbol{d}) &= \tfrac{\boldsymbol{f}(\boldsymbol{x}+h\boldsymbol{d})-\boldsymbol{f}(\boldsymbol{x})}{h} + O(h), \\ \partial_{\pm h}\boldsymbol{f}(\boldsymbol{x};\boldsymbol{d}) &= \tfrac{\boldsymbol{f}(\boldsymbol{x}+h\boldsymbol{d})-\boldsymbol{f}(\boldsymbol{x}-h\boldsymbol{d})}{2h} + O\left(h^2\right) \end{aligned}$$

for directional derivative approximation. Regarding computational effort, they cost twice the effort of the nominal function evaluation and a Jacobian evaluation costs $1+n$ (∂_{+h}) and $2n$ ($\partial_{\pm h}$) times the effort of the nominal evaluation.

While FD is easily implemented on top of nominal function evaluations, accuracy is lost as it critically depends on the magnitude $h\,||d||$. Even for optimal perturbations in ∂_{+h} and $\partial_{\pm h}$, their accuracy is bounded by only one half of the significant digits of $\boldsymbol{f}(\boldsymbol{x})$ for ∂_{+h} and two thirds for $\partial_{\pm h}$.

2.2 Algorithmic Differentiation

In contrast, AD builds on the realization of $\boldsymbol{f}(\boldsymbol{x})$ in a CG by repeatedly applying the chain rule of calculus to the sequence of arithmetic operations stored.

Algorithm 1: Zero-order forward sweep.

input : $\varphi_{1-n}, \dots, \varphi_k$, $\boldsymbol{x}$
output : $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$

$v_{[1-n:0]} = \boldsymbol{x}_{[1:n]}$
for $i = 1 : k$ **do**
  $v_i = \varphi_i(v_{j \prec i})$
end
$\boldsymbol{y}_{[1:m]} = v_{[k-m+1:k]}$

Algorithm 2: First order forward sweep.

input : $\varphi_{1-n}, \dots, \varphi_k$, $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$
output : $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, $\dot{\boldsymbol{y}} = \partial \boldsymbol{f}(\boldsymbol{x}; \dot{\boldsymbol{x}})$

$v_{[1-n:0]} = \boldsymbol{x}_{[1:n]}$
$\dot{v}_{[1-n:0]} = \dot{\boldsymbol{x}}_{[1:n]}$
for $i = 1 : k$ **do**
  $v_i = \varphi_i(v_{j \prec i})$
  $\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_{j \prec i})\ \dot{v}_j$
end
$\boldsymbol{y}_{[1:m]} = v_{[k-m+1:k]}$
$\dot{\boldsymbol{y}}_{[1:m]} = \dot{v}_{[k-m+1:k]}$

Table 1: Computational effort and quality of derivatives $\partial \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{d})$ for a single direction $\boldsymbol{d}$ or several directions $\boldsymbol{D}$.

	Effort [#eval($\boldsymbol{f}(\boldsymbol{x})$)]		Precision in
Type	$\boldsymbol{d} \in \mathbb{R}^{\cdot}$	$\boldsymbol{D} \in \mathbb{R}^{\cdot \times n}$	machine prec. ϵ
$\partial_{+h} \boldsymbol{f}(\boldsymbol{x}; \cdot)$	2	$1 + \quad n$	$\gtrsim \sqrt{\epsilon}$
$\partial_{\pm h} \boldsymbol{f}(\boldsymbol{x}; \cdot)$	2	$1 + 2\,n$	$\gtrsim \sqrt[3]{\epsilon^2}$
$\partial_{ad} \boldsymbol{f}(\boldsymbol{x}; \cdot)$	$\leqslant \frac{5}{2}$	$\leqslant 1 + \frac{3}{2} n$	$\gtrsim \epsilon$

This augments the evaluation procedure, such that derivatives are evaluated simultaneously with the nominal function $\boldsymbol{f}$. This allows to automatically compute derivatives of arbitrary order and accurate to machine precision. Following this, $\boldsymbol{f}$ can be represented by a sequence $\{\varphi_1, \dots, \varphi_k\}$ of elementary operations. Thus, $\boldsymbol{f}$ can be evaluated by Alg. 1 for given inputs $\boldsymbol{x}$ using the slice operator $[i : j]$ and the dependence relation $\prec$ specified below.

Definition 2 (Slice operator and indexing, $i : j$)
The slice operator $i : j$ generalizes indexing of a vector to extract several segments or elements where i is the lower, j the upper bound. A slice of a vector $\boldsymbol{x}$ is defined as

$$\boldsymbol{x}[i : j] = [x_i, x_{i+1}, x_{i+2}, \dots, x_{i+l}]\,,$$

where $l \in \mathbb{N}$ the largest integer s.t. $i + l \leqslant j$. For $\boldsymbol{x} \in \mathbb{R}^n$ the indices i, j are optional in the slice of a vector $\boldsymbol{x}[i : j]$. Alternatively, when used as an operator, slicing represents an implicit loop over the dimension p, e.g., for $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{m \times m}$ and $\dot{\boldsymbol{A}}, \dot{\boldsymbol{B}}, \dot{\boldsymbol{C}} \in \mathbb{R}^{p \times m \times m}$

$$\dot{\boldsymbol{C}}[:] = \dot{\boldsymbol{A}}[:]\boldsymbol{B} + \boldsymbol{B}\dot{\boldsymbol{C}}[:] \Leftrightarrow \dot{\boldsymbol{C}}_i = \dot{\boldsymbol{A}}_i \boldsymbol{B} + \boldsymbol{B}\dot{\boldsymbol{C}}_i,\ 0 \leqslant i < p$$

Definition 3 (Precedence Relation $\prec$)
The relation $j \prec i$ indicates that a quantity v_i depends on the v_j for indices $j < i$, i.e., at least one element of the tuple $v_j = \varphi_j(v_{k \prec j})$ is an argument of the function φ_i. Each instruction φ_i can be interpreted by a node in a directed acyclic graph, the computational graph. There is an edge from φ_i to φ_j iff $j \prec i$.

2.3 Forward Mode of AD

The forward mode of AD augments the evaluation procedure in Alg. 1 to compute a directional derivative $\dot{\boldsymbol{y}} = \partial \boldsymbol{f}(\boldsymbol{x}; \dot{\boldsymbol{x}})$. Therefore, tangential information is propagated by applying the chain rule to every intermediate evaluation, i.e.,

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_{j \prec i}) \dot{v}_j \equiv \sum_{j \prec i} \partial_{v_j} \varphi_i(v_{j \prec i}; \dot{v}_j),$$

where we assume that the elemental tangent operation given by $\frac{\partial \varphi_i}{\partial v_j}(v_{j \prec i})\dot{v}_j$ exists for each elementary operation φ_i. In contrast to symbolic differentiation, the intermediate partials $\dot{v}_i = \frac{\partial \varphi_i}{\partial v_j}(u_i)\dot{u}_i$ are not accumulated as an expression of growing complexity and then evaluated but evaluated first and then accumulated. The augmented evaluation procedure is presented in Alg. 2 which can be extended to compute multiple directional derivatives in one sweep.

In contrast to FD, the derivatives computed via AD are accurate to machine precision, see [16]. Regarding computational effort, the costs to compute the directional derivative $\partial \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{d}) \in \mathbb{R}^m$ along a single direction $\boldsymbol{d}$ with AD are bounded by 2.5 times the costs for the nominal evaluation $\boldsymbol{f}(\boldsymbol{x})$, see [16]. The effort of propagating p directions, i.e., $\boldsymbol{D} \in \mathbb{R}^{n \times p}$, to compute $\partial \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{D}) \in \mathbb{R}^{m \times p}$ is bounded by $1 + \frac{3}{2}p$ times the effort of the nominal evaluation of the function $\boldsymbol{f}$, i.e., it scales linearly in number of directions p. Therefore, AD is expected to be slightly more expensive than FD in the worst case. Practical applications show that this is often not the case.

We summarize the theoretical information on computational complexity and accuracy in Table 1.

3 Evaluation of Rigid-Body Dynamics

The well known descriptor form of the equation of motion of a MBS subject to external contacts expressed via kinematic constraints is given by

$$\begin{bmatrix} \boldsymbol{H}(\boldsymbol{q}) & \boldsymbol{G}(\boldsymbol{q})^T \\ \boldsymbol{G}(\boldsymbol{q}) & 0 \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ -\boldsymbol{\lambda}_c \end{bmatrix} = \begin{bmatrix} \boldsymbol{S}^T\boldsymbol{\tau} - \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \\ -\boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{bmatrix} \tag{1}$$

with generalized positions, velocities, accelerations and forces $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\tau} : \mathbb{R} \to \mathbb{R}^{n^{\mathrm{DoF}}}$, a s.p.d. joint space inertia $\boldsymbol{H} : \mathbb{R}^{n^{\mathrm{DoF}}} \to \mathbb{R}^{n^{\mathrm{DoF}} \times n^{\mathrm{DoF}}}$, the nonlinear effects vector $\boldsymbol{c} : \mathbb{R}^{n^{\mathrm{DoF}}} \times \mathbb{R}^{n^{\mathrm{DoF}}} \to \mathbb{R}^{n^{\mathrm{DoF}}}$, and a selection matrix $\boldsymbol{S} \in \mathbb{R}^{n^{\mathrm{DoF}} \times n^{\mathrm{DoF}}}$ mapping the joint torques $\boldsymbol{\tau}$ to the actuated degrees of freedom (DoFs). We focus on scleronomous holonomic constraints of the form $\boldsymbol{g}(\boldsymbol{q}(t)) = 0$ with $\boldsymbol{g} : \mathbb{R}^{n^{\mathrm{DoF}}} \to \mathbb{R}^{n^{\mathrm{G}}}$, where a contact force $\boldsymbol{\lambda}_c : \mathbb{R} \to \mathbb{R}^{n^{\mathrm{G}}}$ acts on the system by means of the transposed contact Jacobian $\boldsymbol{G}$. The term $\boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{\partial \dot{\boldsymbol{g}}(\boldsymbol{q})}{\partial t}\dot{\boldsymbol{q}}$ is called the contact Hessian. The same holds for infinitesimal contact events.

Each of the involved vector- or matrix-valued quantities has to be evaluated considering the kinematic topology of the respective MBS and the descriptor form of the equation of motions has to be solved by using linear algebra routines employing matrix decomposition techniques.

3.1 Primer on Spatial Algebra for Rigid-Body Dynamics

We lean on the notation in [9] and refer to it concerning more detailed descriptions of the involved quantities. A comprehensive introduction into the concepts of Spatial Algebra can be found in the textbook of Featherstone [8]. For the reader's convenience, we revisit the variable definitions from [9] in Table 2 and briefly explain the important quantities.

We assume a MBS consisting of n^{B} connected rigid bodies. A connection consists of two bodies and the connecting joint which defines their relative motion. Furthermore, we assume a kinematic tree topology of the MBS, i.e., we exclude kinematic loops (except for possible contacts with the environment)

Table 2: Variable definitions of a rigid multi-body model.

λ_i	Parent body index for joint i connecting body i with body λ_i
$\kappa(i)$	The set of joints that influence (,i.e., support) body i
$\boldsymbol{S}_i$	Motion space matrix for joint i
${}^i\boldsymbol{X}_0$	Spatial transformation from global frame to frame of body i
$\boldsymbol{X}_{\mathbf{T}i}$	Spatial transformation from parent of body i to frame of joint i
${}^i\boldsymbol{X}_{\lambda_i}$	Spatial transformation from parent of body i to body i
${}^{\lambda_i}\boldsymbol{X}_i^*$	Spatial adjoint transformation body i to its parent body
$\boldsymbol{I}_i$	Spatial inertia of body i
$\boldsymbol{I}_i^c$	Composite body inertia of body i

and presume a *root* body. We enumerate each body from the root up to n^{B} consecutively.

In order to access the above-defined topology, we define for each joint $i = 1, \ldots, n^{\mathrm{B}}$ the index λ_i of its parent body and the set of its influencing joints indices $\kappa(i)$.

There are two fundamental elements of spatial algebra, spatial motions $\hat{\boldsymbol{v}} \in \mathsf{M}^6$ and spatial forces $\hat{\boldsymbol{f}} \in \mathsf{F}^6$. A mapping $\mathcal{D}_O : \mathbb{R}^6 \to \mathsf{M}^6$ and $\mathcal{E}_O : \mathbb{R}^6 \to \mathsf{F}^6$ can be defined by using Plücker bases $\mathcal{D}_O, \mathcal{E}_O$, c.f. [17]. Spatial inertia $\hat{\boldsymbol{I}} : \mathsf{M}^6 \to \mathsf{F}^6$ defines a mapping between motion and force spaces. Spatial velocity $\boldsymbol{v}_i$, velocity-dependent acceleration $\boldsymbol{c}_i$ and spatial acceleration $\boldsymbol{a}_i$ of body i are elements of spatial motion space M^6. Spatial force $\boldsymbol{f}_i$ acting on the parent body is element of spatial force space F^6.

The joints define the relative motion and the respective motion DoFs are defined by the motion space matrix of joint i denoted as $\boldsymbol{S}_i$. We restrict to single-DoF joints for convenience, but multi-DoF joints can easily be derived by looking up the details in [8]. Following this, the motions subspace matrices are given by $\boldsymbol{S} = e_i \in \mathbb{R}^6$ where e_i denotes the i-th canonical unit basis vector and $i = 1, 2, 3$ for rotational joints around the x, y, z-axis and $i = 4, 5, 6$ for prismatic joints along the x, y, z-axis. The spatial inertia term $\boldsymbol{I}$ is constant and can be neglected from here on.

Spatial algebra defines a cross product motion vectors by $\times$ and another for forces by $\times^*$. For $\hat{\boldsymbol{v}} = \left[\begin{smallmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{smallmatrix}\right]$, they read

$$\hat{\boldsymbol{v}}\times = \left[\begin{smallmatrix} \boldsymbol{\omega}\times & \boldsymbol{0} \\ \boldsymbol{v}_O\times & \boldsymbol{\omega}\times \end{smallmatrix}\right], \quad \hat{\boldsymbol{v}}\times^* = \left[\begin{smallmatrix} \boldsymbol{\omega}\times & \boldsymbol{v}_O\times \\ \boldsymbol{0} & \boldsymbol{\omega}\times \end{smallmatrix}\right]. \tag{2}$$

The critical quantities during derivative evaluations are the spatial or PLÜCKER transformations denoted by ${}^i\boldsymbol{X}_j$, which represent transformations between the Cartesian coordinate frames at bodies (or joints) i and j. They have a special form, which can be exploited numerically.

If a coordinate frame B is translated by $\boldsymbol{r} \in \mathbb{R}^3$ and rotated by $\boldsymbol{E} \in \mathcal{SO}(3)$ relatively to a coordinate frame A, then the respective spatial transformation ${}^B\boldsymbol{X}_A(\boldsymbol{E}, \boldsymbol{r})$ and its adjoint operator ${}^B\boldsymbol{X}_A^*(\boldsymbol{E}, \boldsymbol{r})$ read

$${}^B\boldsymbol{X}_A(\boldsymbol{E}, \boldsymbol{r}) = \left[\begin{smallmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{r}\times & \boldsymbol{E} \end{smallmatrix}\right], \quad {}^B\boldsymbol{X}_A^*(\boldsymbol{E}, \boldsymbol{r}) = \left[\begin{smallmatrix} \boldsymbol{E} & -\boldsymbol{E}\boldsymbol{r}\times \\ \boldsymbol{0} & \boldsymbol{E} \end{smallmatrix}\right].$$

Here, $\mathcal{SO}(3)$ denotes the special orthogonal group of rotational matrices and $\boldsymbol{r}\times$ denotes the skew symmetric cross-product matrix of the translational vector $\boldsymbol{r}$

given by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}.$$

Spatial transformations transform elements of the motion space M^6 while spatial forces in F^6 are transformed by the adjoint operator. Their structure can be exploited to evaluate the respective operations like matrix-matrix or matrix-vector products efficiently, c.f. [8].

3.2 Derivatives of Spatial Transformations

Now, we introduce manipulations of spatial transformations as elementary operations in the AD sense. Spatial transformations are products of two matrices in special form, which leads to Prop. 1.

Proposition 1 (Spatial Transformation Derivative)
Let A and B be frames with spatial velocities $\boldsymbol{v}_A, \boldsymbol{v}_B \in \mathsf{M}^6$, for any motion vector $\boldsymbol{m} \in \mathsf{M}^6$ or force vector $\boldsymbol{f} \in \mathsf{F}^6$, the derivative of the spatial transforms are given by

$$\partial_t({}^B\boldsymbol{X}_A){}^A\boldsymbol{m} = {}^B(\boldsymbol{v}_A - \boldsymbol{v}_B) \times {}^B\boldsymbol{X}_A{}^A\boldsymbol{m}, \tag{3}$$

$$\partial_t({}^A\boldsymbol{X}_B^*){}^B\boldsymbol{f} = {}^A\boldsymbol{X}_B^{*B}(\boldsymbol{v}_B - \boldsymbol{v}_A) \times^* {}^B\boldsymbol{f}. \tag{4}$$

If A and B are connected by a single-DoF joint defined by motion matrix $\boldsymbol{S}$, i.e., the $\boldsymbol{X}$ is parametrized by $q(t)$, this yields

$$\partial_t({}^B\boldsymbol{X}_A\boldsymbol{m}) = [({}^B\boldsymbol{X}_A\boldsymbol{m})\times]\boldsymbol{S}\dot{q}, \tag{5}$$

$$\partial_t({}^A\boldsymbol{X}_B^*\boldsymbol{f}) = {}^A\boldsymbol{X}_B^*[\times^*\boldsymbol{f}]\boldsymbol{S}\dot{q}, \tag{6}$$

where we define $[\times^*\boldsymbol{f}]$ as the matrix encoding the operation $\cdot \times^* \boldsymbol{f}$ because $\times^*$ is not skew symmetric, see (2).

Proof A proof is given in [8] for the first case, the second follows analogously. ∎

Proposition 2 (Derivative of Spatial Inertia Transformation)
Let A and B be frames with spatial velocities $\boldsymbol{v}_A, \boldsymbol{v}_B \in \mathsf{M}^6$ and $\boldsymbol{I}$ the spatial inertia of a body in A, the derivative of the transformation ${}^A\boldsymbol{I} = {}^A\boldsymbol{X}_B^*\,{}^B\boldsymbol{I}\,{}^B\boldsymbol{X}_A$ of $\boldsymbol{I}$ from B to A is given by

$$\partial_t({}^A\boldsymbol{I}) = [{}^A(\boldsymbol{v}_B - \boldsymbol{v}_A)\times^*]{}^B\boldsymbol{I} - {}^B\boldsymbol{I}[{}^A(\boldsymbol{v}_B - \boldsymbol{v}_A)\times].$$

Proof The claim follows from [8] and using Prop. 1. ∎

Note that the derivative of a spatial transform from Prop. 1 is no longer a spatial transform, $\boldsymbol{X} \equiv \boldsymbol{X}(\boldsymbol{E}, \boldsymbol{r})$, because the identity $\boldsymbol{E}^T\boldsymbol{E} = \boldsymbol{1}$ does not hold for $\partial\boldsymbol{E}$ in general, i.e.,

$$\partial({}^B\boldsymbol{X}_A)(\boldsymbol{E}, \boldsymbol{r}; \partial\boldsymbol{E}, \partial\boldsymbol{r}) \neq \boldsymbol{X}(\partial\boldsymbol{E}, \partial\boldsymbol{E}\boldsymbol{r} + \boldsymbol{E}\partial\boldsymbol{r}).$$

The property of the codomain is lost because the directional derivative $\partial\boldsymbol{E}(p; d) \in \mathbb{R}^{3\times3}$ of $\boldsymbol{E} : \mathbb{R}^p \to \mathbb{R}^{3\times3}$ mapping to $\mathcal{SO}(3)$ is not necessarily an element of $\mathcal{SO}(3)$ anymore.

The structure allows to store and compute spatial transforms, their adjoints, inverses and concatenations in motion and force space efficiently. Because the associated derivatives are straightforward consequences of the results just derived, they serve as elementary operations in the AD methodology.

3.3 Derivatives of Recursive Algorithms

Revisiting the equation of motion in (1), for each of the involved quantities an efficient recursive algorithm is available, e.g. the Composite Rigid-Body Algorithm (CRBA) for the evaluation of $\boldsymbol{H}(\boldsymbol{q})$ or RNEA for $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$.

The joint space inertia matrix $\boldsymbol{H}(\boldsymbol{q})$ can be efficiently computed by means of the Composite Rigid-Body Algorithm. It computes the non-zero entries by recursively assembling the required composite rigid-body inertia $\boldsymbol{I}^c$ backwards up to the root body. In Alg. 3, we present the derivative of CRBA as *pseudo* code. Its correctness follows from Prop. 1, 2 and derivative calculus.

For the analysis and the derivative of RNEA, we have to refer to [15] due to page limitations. The rows of contact Jacobian $\boldsymbol{G}$ for contact frame A of body j is computed by transforming the joint motion space matrices to the coordinate frame A by

$$ {}^{A}\hat{\boldsymbol{G}}(\boldsymbol{q}) = {}^{A}\boldsymbol{X}_0 \sum_{i \in \kappa(j)} {}^{0}\boldsymbol{X}_i(\boldsymbol{q}_i)\boldsymbol{S}_i. \tag{7} $$

The contact Hessian $\boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, as the time derivative of contact Jacobian $\boldsymbol{G}$ follows (7) in its computation. The computation of the respective derivatives follows again from Prop. 1 and derivative calculus.

Algorithm 3: CRBA Forward Sweep

input : $\boldsymbol{q}$, $\partial\boldsymbol{q}$
output : $\boldsymbol{H}, \partial\boldsymbol{H} = \partial\text{CRBA}(\boldsymbol{q}; \partial\boldsymbol{q})$

$\boldsymbol{H} = \boldsymbol{0}$ // Nominal
$\partial\boldsymbol{H}[:] = \boldsymbol{0}$
for $i = 1, \ldots, n_B$ **do**
  $[\boldsymbol{X}_{\mathbf{J}i}, \boldsymbol{S}_i, \boldsymbol{c}_{\mathbf{J}i}] = \text{jcalc}(\text{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$ // Nominal
  ${}^{i}\boldsymbol{X}_{\lambda_i} = \boldsymbol{X}_{\mathbf{J}i}\boldsymbol{X}_{\mathbf{T}i}$ // AD computes derivatives, ED reuses quantities
  $\boldsymbol{I}_i^c = \boldsymbol{I}_i$ // Nominal
  $\partial\boldsymbol{I}_i^c[:] = \boldsymbol{0}$
end
for $i = n_B, \ldots, 1$ **do**
  if $\lambda_i \neq 0$ **then**
    $\boldsymbol{I}^c = {}^{\lambda_i}\boldsymbol{X}_i^* \boldsymbol{I}_i^{c\,i}\boldsymbol{X}_{\lambda_i}$ // Nominal
    $\boldsymbol{I}_{\lambda_i}^c = \boldsymbol{I}_{\lambda_i}^c + \boldsymbol{I}^c$ // Nominal
    $\partial\boldsymbol{I}_{\lambda_i}^c[:] = \partial\boldsymbol{I}_{\lambda_i}^c[:] + [{}^{i}S_i\partial\boldsymbol{q}[:]\times^*]\boldsymbol{I}^c - \boldsymbol{I}^c[{}^{i}S_i\partial\boldsymbol{q}[:]\times] + {}^{\lambda_i}\boldsymbol{X}_i^*\partial\boldsymbol{I}_i^c[:]{}^{i}\boldsymbol{X}_{\lambda_i}$
  end
  $\boldsymbol{F} = \boldsymbol{I}_i^c\boldsymbol{S}_i$ // Nominal
  $\partial\boldsymbol{F}[:] = \partial\boldsymbol{I}_i^c[:]\boldsymbol{S}_i$ // $\partial\boldsymbol{S}_i = \boldsymbol{0}$ for 1-DoF joints
  $\boldsymbol{H}_{ii} = \boldsymbol{S}_i^T\boldsymbol{F}$ // Nominal
  $\partial\boldsymbol{H}_{ii}[:] = \boldsymbol{S}_i^T\partial\boldsymbol{F}[:]$ // $\partial\boldsymbol{S}_i = \boldsymbol{0}$ for 1-DoF joints
  $j = i$
  while $\lambda_i \neq 0$ **do**
    $\boldsymbol{F} = {}^{\lambda_i}\boldsymbol{X}_j^*\boldsymbol{F}$ // Nominal
    $\partial\boldsymbol{F} = {}^{\lambda_i}\boldsymbol{X}_j^*(\partial\boldsymbol{F} - [\times^*\boldsymbol{F}]S_j\partial\boldsymbol{q})$ // Matrix-vector product
    $j = \lambda_i$ // Nominal
    $\boldsymbol{H}_{ij} = \boldsymbol{F}^T\boldsymbol{S}_j$ // Nominal
    $\boldsymbol{H}_{ji} = \boldsymbol{H}_{ij}^T$ // Nominal
    $\partial\boldsymbol{H}_{ij}[:] = \partial\boldsymbol{F}^T[:]\boldsymbol{S}_j$ // $\partial\boldsymbol{S}_i = \boldsymbol{0}$ for 1-DoF joints
    $\partial\boldsymbol{H}_{ji}[:] = \partial\boldsymbol{H}_{ij}^T[:]$
  end
end

3.4 Derivatives of Contact Dynamics

We briefly derive the derivative of a linear system solution. This is especially important as the derivative of the linear system solves that arise in the com-

Algorithm 4: Solution of Descriptor Form Derivative.

input : $\boldsymbol{q}$, $\partial\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\partial\dot{\boldsymbol{q}}$, $\boldsymbol{\tau}$, $\partial\boldsymbol{\tau}$, CS
output : $\ddot{\boldsymbol{q}}$, $\partial\ddot{\boldsymbol{q}}$, $\boldsymbol{\lambda}_c$, $\partial\boldsymbol{\lambda}_c = \partial\mathrm{FD}(\boldsymbol{q}, \partial\boldsymbol{q}, \dot{\boldsymbol{q}}, \partial\dot{\boldsymbol{q}}, \boldsymbol{\tau}, \partial\boldsymbol{\tau}, \mathrm{CS})$

$\boldsymbol{H}, \partial\boldsymbol{H} = \partial\,\mathrm{CRBA}(\boldsymbol{q}, \partial\boldsymbol{q})$
$\boldsymbol{c}, \partial\boldsymbol{c} = \partial\,\mathrm{RNEA}(\boldsymbol{q}, \partial\boldsymbol{q}, \dot{\boldsymbol{q}}, \partial\dot{\boldsymbol{q}}, \boldsymbol{0}, \boldsymbol{0})$
$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{H} & \boldsymbol{G}^T \\ \boldsymbol{G} & 0 \end{bmatrix}, \partial\boldsymbol{K} = \begin{bmatrix} \partial\boldsymbol{H} & \partial\boldsymbol{G}^T \\ \partial\boldsymbol{G} & 0 \end{bmatrix},$
$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{S}^T\boldsymbol{\lambda}_c - \boldsymbol{c} \\ -\boldsymbol{\gamma} \end{bmatrix}, \partial\boldsymbol{b} = \begin{bmatrix} \boldsymbol{S}^T\partial\boldsymbol{\lambda}_c - \partial\boldsymbol{c} \\ -\partial\boldsymbol{\gamma} \end{bmatrix},$
$\boldsymbol{x} = (\ddot{\boldsymbol{q}}^T, \boldsymbol{\lambda}_c^T)^T = \mathrm{solve}(\boldsymbol{K}, \boldsymbol{b}),$
$(\partial\ddot{\boldsymbol{q}}^T, \partial\boldsymbol{\lambda}_c^T)^T = \mathrm{solve}(\boldsymbol{K}, \partial\boldsymbol{b} - \partial\boldsymbol{K}\boldsymbol{x}),$

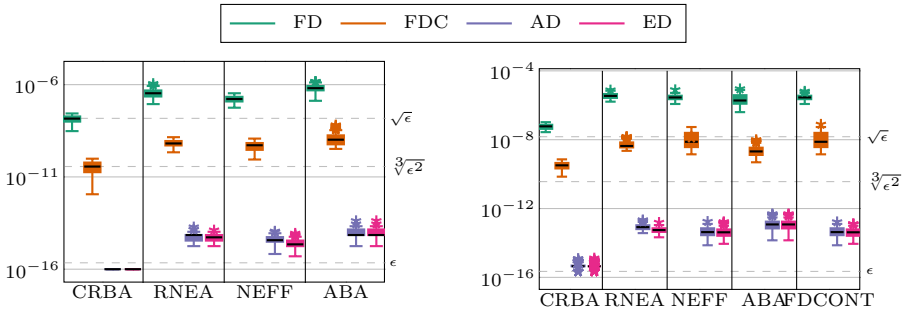
putation of constrained dynamics can be accelerated considerably using these insights. This will be demonstrated computationally in Section 4.2.

Definition 4 (Solution of a Linear System)
Consider $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, $\boldsymbol{B} \in \mathbb{R}^{N\times M}$ and $\boldsymbol{X} \in \mathbb{R}^{N\times M}$ that solves linear equation system $\boldsymbol{A}\,\boldsymbol{X} = \boldsymbol{B}$, where we denote the solution operator $\boldsymbol{X} = \boldsymbol{A}^{-1}\,\boldsymbol{B}$ by

$$\boldsymbol{X} = \mathrm{solve}(\boldsymbol{A}, \boldsymbol{B}). \tag{8}$$

Proposition 3 (Derivative of a Linear System)
The derivative of the solution of the setting of Def. 4 with respective directions ∂A, ∂X and ∂B is $0 = \partial\boldsymbol{A}\boldsymbol{X} + \boldsymbol{A}\partial\boldsymbol{X} - \partial\boldsymbol{B}$. Recapping (8), the derivative is given by $\partial\boldsymbol{X} = \mathrm{solve}(\boldsymbol{A}, \partial\boldsymbol{B} - \partial\boldsymbol{A}\boldsymbol{X})$.

Proof TAYLOR-expansion of each quantity up to order one and coefficient-wise compare yield the claim. ■



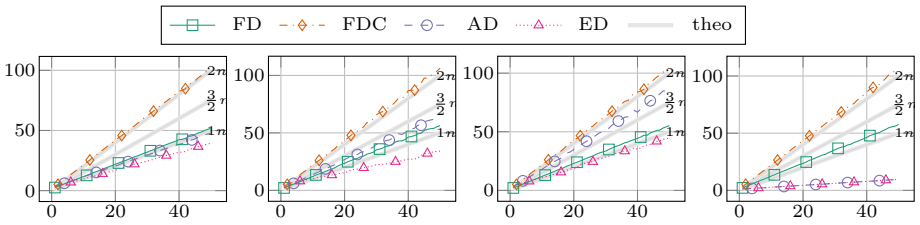
(a) Cart-pendulum model in minimal coordinates.

(b) Cart-pendulum model with constraints.

Figure 1: Errors against analytic derivatives of FD, FDC, AD and ED for different algorithms and both cart-pendulum models.

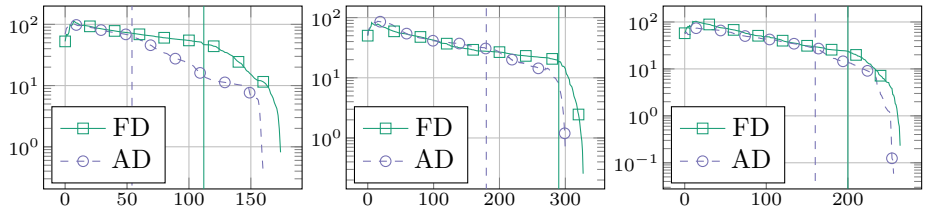
4 Numerical Results

We present the validation and benchmarking results of our AD implementation. We use the notion ED for AD including structure and sparsity exploitation of RBD computations.



(a) Forward dyn. (b) Nonlinear effects (c) CRBA (d) Solve of (1)

Figure 2: Effort [#eval($\boldsymbol{f}(\boldsymbol{x})$)] vs. propagated directions [n] of FD, FDC, AD and ED for a multi-pendulum model.



(a) 3 kg (b) 6 kg (c) 9 kg

Figure 3: Selfconvergence plots of SQP performance using AD and FD for different masses for the biomechanical OCP: distance of current iterate $\boldsymbol{x}^n$ to final iterate $\boldsymbol{x}^\star$ [$\|\boldsymbol{x}^n - \boldsymbol{x}^\star\|_2$] versus iteration counter [n].

4.1 Evaluation of Derivative Quality

To quantify the differences between derivative information computed via AD, ED and FD and analytic derivatives, we consider a cart-pendulum benchmark model in two versions: one with kinematic constraints and one without in minimal coordinates. The contact model is defined as a free-floating box with three DoFs, x, y position and orientation around y-axis, and one DoF for the rotational motion of the pendulum relative to the cart. The contact can be resolved directly into a minimal coordinate formulation with two DoFs.

We evaluated 10 000 uniform random samples of the inputs $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\tau} \in [-\boldsymbol{\pi}, \boldsymbol{\pi}]^{n^{\text{DoF}}}$ and the respective directions $\boldsymbol{D} \in \mathbb{R}^{n^{\text{DoF}} \times 3n^{\text{DoF}}}$ for both models and each algorithm, i.e., CRBA, RNEA, Nonlinear Effects based on RNEA (NEFF), Articulated Body Algorithm (ABA) and Forward Dynamics Constraints Direct (FDCONT). The latter is not run for the minimal coordinate model.

The expected accuracies are $\sqrt{\epsilon} \approx 1.054 \cdot 10^{-8}$ for FD and $\sqrt[3]{\epsilon^2} \approx 2.310 \cdot 10^{-11}$ for FDC, see Tab. 1. In contrast, we expect the AD and ED derivatives to be around ϵ.

Fig. 1a shows the resulting errors for the cart-pendulum model in minimal coordinates, while Fig. 1b shows those for the cart-pendulum model with kinematic constraints.

The approximation errors cluster for all algorithms. For both models, CRBA showed the best accuracy, while ABA showed the worst. Both AD and ED show an accuracy of $\approx 1 \cdot 10^{-14}$. FD and FDC behave similar: mean accuracies of $\approx 5 \cdot 10^{-6}$ and $\approx 1 \cdot 10^{-8}$ are achieved in the worst case for the contact model.

We note that error propagation can easily account for a loss of 2-3 orders of magnitude in precision, c.f. similar observations in [18], and occurs for all modes of derivative evaluation. The results confirm that AD improves the derivative accuracy substantially, in our setup by up to 5 orders of magnitude.

4.2 Benchmarking the Derivative Evaluation

We have evaluated the runtimes of AD, ED as well as FD, FDC using a 10 DoF multipendulum model, similar to the one from [9], for an increasing number of propagated directions n for the algorithms FDCONT, NEFF, CRBA and the solution of (1). FDC evaluates slowest, in line with the expectation $1 + 2n$. FD performs a little worse than the expectation of $1 + n$. The AD implementation performs similar to FD for FDCONT, and between FD and FDC for NEFF and CRBA. In the latter case, it exceeds the theoretical expectation of $1 + 1.5n$. For these three algorithms, the structure exploitation implemented in ED pays off and ED outperforms AD and FD. The biggest benefit of AD and ED over a black-box FD evaluation is obtained in the solution of (1). The reuse of the factorization significantly decreases the runtime. AD and ED take less than a quarter of the time of FD. The results are shown in Fig. 2.

4.3 Application to optimal control of a bio-mechanical model

We have used our AD implementation to optimally control a two-phase bending and lifting motion for an 8 DoF human model in the sagittal plane, see [6], and with varying weights attached to the hand during lifting. We followed a direct and all-at-once approach and solved the resulting NLP using an SQP method. Fig. 3 a)-c) show its convergence from the same initialization for different masses to lift in the second phase. The SQP method consistently required fewer iterations using AD compared to FD in all cases (up to 10 % for (b)). Entry into the rapid full step local convergence phase happened earlier in all cases, yielding a faster descent.

5 Conclusion

We proposed and implemented a new AD approach to evaluate derivative information of recursive algorithms for RBD, and demonstrated its applicability for optimal control.

Regarding the quality of the derivatives, each derivative evaluation mode followed the theoretical expectations when tested against analytic derivative computations. The implementations AD and ED exhibited a fine performance up to machine precision.

Regarding the speed of the derivative evaluations, the implementation AD, which follows the chain rule plainly, performs between FD and FDC in most of the considered cases, and mostly a little better than the theoretical expectation. We observe that ED, which utilizes the structure of the RBD algorithms to avoid unnecessary computations, outperforms the FD computation in terms of speed in all of the considered cases. We highlight that employing Prop. 3 in the linear system solution speeds up AD and ED considerably.

In contrast to the presented literature that focuses on RNEA, we augmented the algorithms CRBA and ABA, the latter being considered as *complex* in [15] due to the occurring tensor-valued quantities. Furthermore, none of the existing publications treats the contact case in MBS.

While our code exhibits superior runtime performance for contact dynamics, the overall implementation, especially the one for kinematic quantities, has still room for improvements by further tailoring the structure exploitation per joint type. Furthermore, we aim to support custom and multi-DoF joints as well as the torque-based muscle approximation model from [19] in future work to broaden the class of models to which the code is applicable.

References

[1] M. Sreenivasa, M. Millard, and M. Felis et al., "Optimal control based stiffness identification of an ankle-foot orthosis using a predictive walking model," *Frontiers in Computational Neuroscience*, vol. 11, no. 23, 2017. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fncom.2017.00023/abstract

[2] D. Clever, M. Harant, and K. Mombaur et al., "Cocomopl: A novel approach for humanoid walking generation combining optimal control, movement primitives and learning and its transfer to the real robot hrp-2," *IEEE RA-L*, vol. 2, no. 2, pp. 977–984, 2017.

[3] J. Koenemann, A. Del Prete, and Y. Tassa et al., "Whole-body Model-Predictive Control applied to the HRP-2 Humanoid," in *IROS*, 2015.

[4] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids," in *IROS*, pp. 5307–5314.

[5] O. Junge, J. E. Marsden, and S. Ober-Blöbaum, "Discrete mechanics and optimal control," *IFAC Proc. Vol.*, vol. 38, no. 1, pp. 538–543, 2005.

[6] P. Manns and K. Mombaur, "Towards discrete mechanics and optimal control for complex models," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4812–4818, 2017.

[7] M. Oehler, S. Kohlbrecher, and O. von Stryk, "Whole-body planning for obstacle traversal with autonomous mobile ground robots," in *RAAD19 (to appear)*, 2019.

[8] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer, 2008.

[9] M. Felis, "RBDL: an Efficient Rigid-Body Dynamics Library using Recursive Algorithms," *Autonomous Robots*, vol. 41, no. 2, pp. 495–511, 2017.

[10] J. Kim, "Lie group formulation of articulated rigid body dynamics," Tech. Rep., 2012.

[11] G. Garofalo, C. Ott, and A. Albu-Schäffer, "On the closed form computation of the dynamic matrices and their differentiations," in *IROS*, 11 2013, pp. 2364–2359.

[12] J. Wojtusch, J. Kunz, and O. v. Stryk, "MBSlib – An Efficient Multibody Systems Library for Kinematics and Dynamics Simulation, Optimization and Sensitivity Analysis," *IEEE RA-L*, vol. 1, no. 2, pp. 954–960, 7 2016.

[13] M. Neunert, M. Giftthaler, and M. Frigerio et al., "Fast derivatives of rigid body dynamics for control, optimization and estimation," in *SIMPAR*, 12 2016, pp. 91–97.

[14] M. Giftthaler, M. Neunert, and M. Stäuble et al., "Automatic differentiation of rigid body dynamics for optimal control and estimation," *Advanced Robotics*, vol. 31, no. 22, 09 2017.

[15] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *RSS*, 2018.

[16] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008, vol. 105.

[17] R. Featherstone, "Plucker basis vectors," in *ICRA*, 2006, pp. 1892–1897.

[18] S. Walter, "Structured higher-order algorithmic differentiation in the forward and reverse mode with application in optimum experimental design," Ph.D. dissertation, Humboldt-Universität zu Berlin, 2012.

[19] M. Millard, A. L. Emonds, and M. H. et al., "A reduced muscle model and planar musculoskeletal model fit for the simulation of whole-body movements," *Journal of Biomechanics*, 2019, to appear.