Mixed Use of Analytical Derivatives and Algorithmic Differentiation for NMPC of Robot Manipulators

Alejandro Astudillo, Justin Carpentier, Joris Gillis, Goele Pipeleers, Jan Swevers

Preprints, First IFAC Modeling, Estimation and Control Conference
October 24-27, 2021. Online and UT Austin, USA

Mixed Use of Analytical Derivatives and Algorithmic Differentiation for NMPC of Robot Manipulators ⋆

Alejandro Astudillo * **Justin Carpentier** ** **Joris Gillis** *
Goele Pipeleers * **Jan Swevers** *

* *MECO Research Team, Dept. of Mechanical Engineering, KU Leuven. Flanders Make - DMMS-M, 3001 Leuven, Belgium. (e-mail: {alejandro.astudillovigoya, joris.gillis, goele.pipeleers, jan.swevers}@kuleuven.be).*
** *Inria, École normale supérieure, CNRS, PSL Research University, 75006 Paris, France (email: justin.carpentier@inria.fr)*

Abstract: In the context of nonlinear model predictive control (NMPC) for robot manipulators, we address the problem of enabling the mixed and transparent use of algorithmic differentiation (AD) and efficient analytical derivatives of rigid-body dynamics (RBD) to decrease the solution time of the subjacent optimal control problem (OCP). Efficient functions for RBD and their analytical derivatives are made available to the numerical optimization framework CasADi by overloading the operators in the implementations made by the RBD library Pinocchio and adding a derivative-overloading feature to CasADi. A comparison between analytical derivatives and AD is made based on their influence on the solution time of the OCP, showing the benefits of using analytical derivatives for RBD in optimal control of robot manipulators.

Keywords: Analytical derivatives, algorithmic differentiation, robot manipulators, optimal control, predictive control.

1. INTRODUCTION

The computational cost of solving nonconvex optimal control problems (OCP) restricts the real-time implementation of nonlinear model predictive controllers (NMPC). Algorithms that converge to solutions of the underlying OCP with a reduced number of iterations still face the common issue of evaluating expensive, nonlinear functions of system dynamics and, in general, their derivatives. In NMPC of robot manipulators, this evaluation represents a larger bottleneck than the algebra of the OCP solver since manipulators are subjected to highly-nonlinear rigid-body dynamics (RBD) and operating constraints.

Optimal control of robot manipulators is a key component in a wide range of industrial applications, e.g. space manipulation (Giordano et al. (2019)), robotic surgery (Su et al. (2020)). These applications drive a need for fast numerical optimal control and hence the need for efficient functions for RBD and their derivatives arises. This paper illustrates the potential of using computationally-efficient RBD functions and derivatives to reduce the solution time of OCPs arising in NMPC of robot manipulators, without loosing flexibility about the types of OCPs that can be solved.

Rigid-body dynamics libraries (RBDL), including Robotran (Docquier et al. (2013)), Drake (Tedrake (2019)), RobCoGen (Giftthaler et al. (2017)) and Pinocchio (Carpentier et al. (2019)) implement efficient RBD algorithms. Such libraries usually depend on algorithmic differentiation (AD) tools to obtain derivatives for numerical optimization. Pinocchio, in contrast, implements its own efficient algorithms for analytical derivatives for RBD. Crocoddyl, introduced in Mastalli et al. (2020), is a framework for optimal control of robots that exploits such differentiation in Pinocchio. It implements differential dynamic programs but does not handle constraints in a unified way as done in sequential quadratic programs (SQP). Other numerical optimization frameworks, such as CasADi (see Andersson et al. (2019)), automatically apply AD to supply a numerical solver with any derivatives it needs. One can easily build OCP frameworks on top of such generic foundation, e.g. OpenOCL (Koeneman et al. (2019)), Rockit (Gillis et al. (2020)). Adding tailored RBD derivatives to the core of a generic framework like CasADi would nevertheless have a limited impact since they are application specific.

Current practice involves either adding AD support to RBDL (see Tedrake (2019); Giftthaler et al. (2017)), or implementing RBDL on top of AD frameworks (see Gjerde Johannessen et al. (2019); Millard et al. (2020)) rather than combining the power of generic AD and tailored derivative routines in RBDL libraries. However, there is still uncertainty about how such combination contributes to reduce the solution time of nonconvex OCPs.

⋆ The authors would like to thank Flanders Make SBO MULTIROB: "Rigorous approach for programming and optimal control of multi-robot systems", FWO project G0A6917N of the Research Foundation - Flanders (FWO - Flanders), and KU Leuven-BOF PFV/10/002 Centre of Excellence: Optimization in Engineering (OPTEC) for supporting this research.

The contributions of this paper are twofold. It explores and enables the mixed use of AD and analytical derivatives in a particular optimization framework that can emit C code for efficient evaluation. Moreover, it demonstrates the significant contribution of analytical derivatives of RBD to reduce the solution time of OCPs for robotic manipulators. To the best of the authors' knowledge, this is the first implementation that allows to use both analytical derivatives and AD within a numerical optimization framework in the context of NMPC for robot manipulators.
This paper is organized as follows. Notation and preliminary concepts on rigid-body dynamics and optimal control are presented in Section 2. Section 3 explains the differentiation of rigid-body dynamics. Next, the software framework is presented in Section 4. Experiments and results are discussed in Section 5. We close the paper with concluding remarks.

2. PRELIMINARIES

This section introduces some notation and preliminary concepts on rigid-body dynamics and numerical optimal control. We then motivate the need for efficient expressions for rigid-body dynamics and their derivatives in the context of optimal control problems.

2.1 Notation

For a function $f(w) : \mathbb{R}^n \to \mathbb{R}^m$, $w \mapsto f(w)$, we denote $\mathbf{J_f}(w) := \frac{\partial f}{\partial w}(w) \in \mathbb{R}^{m\times n}$ as the Jacobian, $\nabla f(w) = \mathbf{J_f}(w)^\top$ as the gradient, $\nabla_{w_1} f(w) := \frac{\partial f}{\partial w_1}(w)$ as the directional derivative of $f(w)$ along w_1, and $\mathbf{H_f}(w) := \nabla^2 f(w)$ as the Hessian. The superscripts c and a indicate differentiation via algorithmic or analytical differentiation, respectively.

2.2 Rigid-Body Dynamics

Robot manipulators can be represented as kinematic chains, i.e. chains of rigid-bodies connected by joints. Joints impose constraints on how a rigid-body moves with respect to neighbour bodies in the chain. For fully actuated systems, RBD can be expressed using the Lagrangian formalism (Murray et al. (1994))

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + G(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{J_c}(\mathbf{q})^\top \mathbf{f}^{\text{ext}}, \quad (1)$$

where $\mathbf{q}$ stands for the joint position vector, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are its first and second order derivatives, respectively, $\boldsymbol{\tau}$ is the generalized joint torque, M is the joint-space inertia matrix, C is the Coriolis matrix, G encloses gravity effects, $\mathbf{J_c}$ is the contact Jacobian, and $\mathbf{f}^{\text{ext}}$ is the stack of external forces. Dependency on $(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ is dropped henceforth to increase readability.

Let us briefly introduce the algorithms to compute inverse dynamics, forward dynamics and joint-space inertia matrix. Inverse dynamics (ID) computes the torque $\boldsymbol{\tau}$ needed to produce a certain acceleration $\ddot{\mathbf{q}}$ for a rigid body with given kinematics $(\mathbf{q},\dot{\mathbf{q}})$ and subject to $\mathbf{f}^{\text{ext}}$. The most efficient algorithm to evaluate the ID equation

$$\boldsymbol{\tau} = M\ddot{\mathbf{q}} + C\dot{\mathbf{q}} + G - \mathbf{J_c}^\top \mathbf{f}^{\text{ext}} = ID(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}},\mathbf{f}^{\text{ext}}) \quad (2)$$

is the recursive Newton-Euler algorithm (RNEA), see Featherstone (2008). This algorithm avoids the explicit computation of M and exploits the sparsity induced by the kinematic chain. The RNEA has a computational complexity of $O(n_b)$, where n_b is the number of bodies composing the rigid-body system.
Analogous to ID, forward dynamics (FD) computes the acceleration $\ddot{\mathbf{q}}$ of a rigid-body as

$$\ddot{\mathbf{q}} = M^{-1}(\boldsymbol{\tau} + \mathbf{J_c}^\top \mathbf{f}^{\text{ext}} - C\dot{\mathbf{q}} - G) = FD(\mathbf{q},\dot{\mathbf{q}},\boldsymbol{\tau},\mathbf{f}^{\text{ext}}). \quad (3)$$

The articulated-body algorithm (ABA), generalized in Featherstone (2008), is one of the most efficient algorithms to compute FD. It avoids the explicit computation of M^{-1} and has a complexity of $O(n_b)$.
The joint-space inertia matrix M is a positive-definite matrix of special interest when determining the ill-conditioning of RBD or computing $\nabla_{\ddot{\mathbf{q}}} ID$, for instance. It can be computed with the composite-rigid-body algorithm (CRBA), see Walker and Orin (1982), with a complexity of $O(n_b^2)$. Its inverse M^{-1} is relevant when computing $\nabla_{\boldsymbol{\tau}} FD$ or solving FD naively, and can be directly computed with ABA as shown in Carpentier and Mansard (2018).

2.3 Numerical Optimal Control

In this paper we are interested in solving OCPs of the form

$$\begin{aligned}
\underset{w\,\in\,W}{\text{minimize}} \quad & V_N(x_N) + \sum_{k=0}^{N-1} V(x_k,u_k) && \text{(4a)}\\
\text{subject to} \quad & x_0 = p, && \text{(4b)}\\
& x_{k+1} = \xi(x_k,u_k), && \text{(4c)}\\
& \boldsymbol{\tau}_{min} \le r(x_k,u_k) \le \boldsymbol{\tau}_{max}, && \text{(4d)}\\
& \zeta(x_N) \le 0, && \text{(4e)}\\
& \zeta(x_k,u_k) \le 0, \qquad k \in [0,N-1], && \text{(4f)}
\end{aligned}$$

with prediction horizon $N \in \mathbb{N}$, state vector $x_k \in \mathbb{R}^{n_x}$, input vector $u_k \in \mathbb{R}^{n_u}$ and decision variable $w = [x_0^\top, u_0^\top, \cdots\ x_{N-1}^\top, u_{N-1}^\top, x_N^\top]^\top \in \mathbb{R}^{n_w}$ belonging to the closed set $W := \{w \in \mathbb{R}^{n_w} : w_{min} \le w \le w_{max}\}$. In addition, $p \in \mathbb{R}^{n_x}$ is the state vector estimation, $\xi(x_k,u_k) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \mapsto \mathbb{R}^{n_x}$ is a discrete-time representation of the system, $r(x_k,u_k) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \mapsto \mathbb{R}^{n_b}$ imposes constraints on $\boldsymbol{\tau}$, $V_N(x_N) : \mathbb{R}^{n_x} \mapsto \mathbb{R}$, $V(x_k,u_k) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \mapsto \mathbb{R}$ define the objective function, and $\zeta(x_k,u_k) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \mapsto \mathbb{R}^{n_\zeta}$, $\zeta(x_N) : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_\zeta}$ are path and terminal constraints, respectively. Constraints (4c) arise from the shooting intervals introduced by the multiple-shooting method.
In direct optimal control, (4) is converted into an equivalent nonlinear program (NLP) with objective function $f(w)$, equality constraints $h(w) = 0$ and inequality constraints $g(w) \le 0$. Such NLP can be solved using derivative-based optimization, with first-order methods, i.e. require the evaluation of up to first-order derivatives, or second-order methods, i.e. require the evaluation of up to second-order derivatives. A well-known second-order method is the SQP method. This method approximates an optimal solution w^* by iterating the decision variable as $w_{k+1} = w_k + d_k$, where d_k is the solution of a quadratic program (QP). For every SQP iteration, at least one QP is solved, requiring the evaluation of ∇f, h, ∇h, g, ∇g and the Hessian $\mathbf{H}_{\mathcal{L}}$ of the Lagrangian of the NLP.

2.4 Bottleneck in Dynamics Evaluation

Within the framework of MPC, OCP (4) is solved for a new value of p at every sample instant. Therefore, the

solution time of (4) must be less than a sample time T_s to allow real-time implementations. For robot manipulators, the state vector is usually defined as $x_k \coloneqq [\mathbf{q}^\top, \dot{\mathbf{q}}^\top]^\top$. Common choices for u_k are $u_k \coloneqq \boldsymbol{\tau}$ and $u_k \coloneqq \ddot{\mathbf{q}}$. When the former is chosen, $\xi(x_k, u_k)$ in (4c) becomes a discretization of FD, while $r(x_k, u_k)$ in (4d) is a linear mapping from u_k to $\boldsymbol{\tau}$. Conversely, when $u_k \coloneqq \ddot{\mathbf{q}}$ and assuming the robot manipulator is fully actuated, the differential flatness property can be exploited. Hence, the multiple-shooting constraints are governed by the dynamics of a double integrator, which are cheap to evaluate, while $r(x_k, u_k)$ becomes ID. Constraints (4f) and (4e) define path and terminal constraints on the forward kinematics of the robot, which are cheaper to compute than the RBD.
Note that, whether $u_k \coloneqq \boldsymbol{\tau}$ or $u_k \coloneqq \ddot{\mathbf{q}}$, the evaluation of FD or ID and their derivatives may well become a bottleneck. The reason is that the evaluation of such highly-nonlinear functions is computationally expensive and is required to compute h and ∇h for $u_k \coloneqq \boldsymbol{\tau}$, or g and ∇g for $u_k \coloneqq \ddot{\mathbf{q}}$, while solving the NLP equivalent to OCP (4).

3. JACOBIAN OF RIGID-BODY DYNAMICS EXPRESSIONS FOR NUMERICAL OPTIMIZATION

The problem of computing the derivatives of RBD for their use in OCPs is discussed in this section. The first subsection presents the commonly used method of AD. We then give details on the efficient computation of analytical derivatives for RBD and the complexity of higher-order derivatives.

3.1 Algorithmic Differentiation

Algorithmic differentiation (AD) is a method to compute the derivatives of functions by applying the Leibniz's chain rule to expression-graph representations of such functions. There are two basic approaches for AD: the forward mode and the reverse mode.
For a function $\vartheta(x) : \mathbb{R}^{n_{\text{in}}} \mapsto \mathbb{R}^{n_{\text{out}}}$, the *forward mode* computes a Jacobian-times-vector product, i.e. a directional derivative, $\hat{y} \coloneqq \mathbf{J}_\vartheta \hat{x}$ with a seed $\hat{x}$ and a computational complexity comparable to evaluating $\vartheta(x)$. The *reverse mode* computes a Jacobian-transposed-times-vector product $\bar{x} \coloneqq \mathbf{J}_\vartheta^\top \bar{y}$ with a seed $\bar{y}$, also with a complexity comparable to evaluating $\vartheta(x)$.
By choosing the seeds as slices of a unit matrix, each sweep of the forward mode computes one column of the Jacobian $\mathbf{J}_\vartheta \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$, while a sweep of the reverse mode computes one row of $\mathbf{J}_\vartheta$. Therefore, the computation of $\mathbf{J}_\vartheta$ requires n_{in} sweeps of forward mode or n_{out} sweeps of reverse mode.

Accurate first-order derivatives of FD or ID can be computed with AD. Both FD and ID are functions mapping from $\mathbb{R}^{3n_b}$ to $\mathbb{R}^{n_b}$, assuming that there are no external forces $\mathbf{f}^{\text{ext}}$ affecting the rigid-body, and $\mathbf{q}$ is the same size as its tangent vector $\dot{\mathbf{q}}$, e.g. $\mathbf{q}$ is not a quaternion. Since, $n_{\text{in}} = 3n_b > n_{\text{out}} = n_b$, reverse mode of AD is used to differentiate the RBD.
Let us define $\mathbf{J}^c_{FD}$ and $\mathbf{J}^c_{ID}$ as the Jacobian of FD and ID, respectively, computed with reverse mode. The computation of such Jacobians has a complexity of $O(n_b^2)$, due to the $O(n_b)$ complexity of both FD and ID, and the need for n_b sweeps of reverse mode to compute a Jacobian.

3.2 Analytical Derivatives

Contrary to AD, which evaluates both the values and their derivatives for a expression-graph from atomic expressions, the analytical differentiation of RBD directly operates at the level of spatial operations that describe the dynamics of rigid-body systems (Carpentier and Mansard (2018)). For instance, the time derivative of the rotation matrix R associated to a frame rotating at a speed ω is given by $\dot{R} = R\,[\omega]_\times$, where $[.]_\times$ is the skew operator. While from a mechanical point of view this relation seems basic, AD would have to recover it by differentiating each individual element of R. Therefore, analytical derivatives are able to exploit the inherent sparsity of the spatial operations to compute the derivatives of both FD and ID, or any other related quantities. In other words, the granularity of the operations is shifted from atomic expressions to spatial expressions. In addition to that, and as shown in Carpentier and Mansard (2018), the evaluation complexity of RBD algorithms can also be lowered by exploiting some simplifications which appears from the recursion of the dynamic equations themselves or by exploiting the intimate relations between forward and inverse dynamics derivatives, to skip for instance the evaluation of complex tensorial quantities.
Let us define the Jacobian of FD (respectively ID) computed with analytical derivatives as $\mathbf{J}^a_{FD}$ ($\mathbf{J}^a_{ID}$). The computational complexity of evaluating the partial derivatives of FD (ID), as in Carpentier and Mansard (2018), is $O(n_b n_d)$ where n_d is the depth of the kinematic tree. Note that for serial robot manipulators $n_d = n_b$. Consequently, the computational complexity of $\mathbf{J}^a_{FD}$ and $\mathbf{J}^a_{ID}$ for serial robot manipulators is $O(n_b^2)$. This is the same complexity as that from the Jacobians $\mathbf{J}^c_{FD}$ and $\mathbf{J}^c_{ID}$, computed with AD. However, analytical derivatives exploit the sparsity in the function by preserving the structured sparsity of RNEA and directly differentiating the spatial operators, then reducing the number of atomic operations in the Jacobian evaluations, and thereby reducing the evaluation time of $\mathbf{J}^a_{ID}$ and $\mathbf{J}^a_{FD}$.

3.3 Higher-order Derivatives

A family of second-order methods requires the computation of second-order derivatives to populate Hessians in the NLP solution algorithm. The computation of $\mathbf{H}_\mathcal{L}$ requires, for instance, computing the Hessian $\mathbf{H}^c_{FD}$ of $\gamma \coloneqq \lambda^\top FD$, where λ are the Lagrange multipliers corresponding to the equality constraints in the NLP. The Hessian of a scalar-valued function can be computed as the Jacobian of the gradient $\nabla\gamma$. Thus,

$$\mathbf{H}^c_{FD} = \mathbf{J}_{\nabla\gamma}, \quad \text{with} \quad \nabla\gamma = \mathbf{J}^{c^\top}_{FD}\lambda, \tag{5}$$

where reverse mode of AD is applied recursively to, first, directly compute $\nabla\gamma$ as a Jacobian-transposed-vector product and next, compute $\mathbf{J}_{\nabla\gamma}$, with a total complexity of $O(n_b)$. Contrarily, if $\mathbf{J}^a_{FD}$ has already been computed via analytical derivatives, the Hessian of γ is computed as

$$\mathbf{H}^a_{FD} = \mathbf{J}_{\nabla\gamma}, \quad \text{with} \quad \nabla\gamma = (\lambda^\top \mathbf{J}^a_{FD})^\top, \tag{6}$$

where the computation of $(\lambda^\top \mathbf{J}^a_{FD})^\top$ needs to be performed before applying reverse mode of AD to $\nabla\gamma$. In this case,

computing $\nabla\gamma$ has a complexity of $O(n_b^2)$ due to matrix multiplication and transposition, while computing $\mathbf{J}_{\nabla\gamma}$ has a complexity of $O(n_b)$.

4. SOFTWARE FRAMEWORK

Having discussed how to compute derivatives of RBD and their importance in the context of numerical optimal control, let us now consider the software implementation of an interface developed to close the gap between (i) a general framework for numerical optimization and AD and (ii) the efficient implementation of analytical derivatives tailored for application-specific numerical evaluation.

4.1 Numerical Optimization Framework - CasADi

CasADi is an open tool for numerical optimization and AD. It is based on a symbolic framework which constructs expression-graphs from functions and algorithms. Every node in the graph represents an atomic operation. These graphs are automatically differentiable by performing AD on them. As a numerical optimization framework, CasADi implements algorithms to solve (non)convex OCPs and interfaces other numerical optimization solvers. CasADi also features a native support for code-generation.

4.2 Rigid Body Dynamics Library - Pinocchio

Pinocchio is an open-source RBDL which features efficient implementations of RBD algorithms such as RNEA, ABA, and CRBA. It also implements the analytical derivatives of RBD mentioned in Section 3. The algorithms implemented in Pinocchio exploit sparsity, with specific spatial operators for different types of joints, and static polymorphism to reduce their evaluation time, outperforming other RBDLs (Carpentier et al. (2019); Neuman et al. (2019)).

4.3 Development of an Interface Framework

CasADi, being a general optimization and AD framework, is not aimed to include RBD features in its core. Moreover, the algorithms in Pinocchio are not tailored to be evaluated symbolically. Hence, we built a C++ interface [1] that generates CasADi expression-graphs from the RBD algorithms in Pinocchio by using operator-overloading (Phipps and Pawlowski (2012)). The interface includes methods to compute analytical derivatives of RBD and to export them as serialized CasADi functions, which can then be imported by any software tool that is compatible with CasADi such as MPC tools in Python (Lucia et al. (2017)), in MATLAB (Chen et al. (2019)), or in C (Verschueren et al. (2019)).
A new feature that allows the user to overload derivatives with custom user-defined derivative expressions was added to CasADi. The feature is implemented as an option called *custom_jacobian*, settable by the user for any CasADi function. Hereby, CasADi is set to use analytical derivatives for RBD function calls, while the remainder of the constraint computation (e.g. the discretization step) still uses regular AD. This hybrid setup works transparently, with regular CasADi features such as C code generation still fully functional.

[1] Available on GitHub: *https://git.io/JnN5T*

5. RESULTS

Having discussed how to generate computationally efficient RBD derivatives and how to use them within a numerical optimization framework, this section addresses the assessment of such expressions with respect to AD and its contribution into the solution time of a nonconvex OCP. Each test case in this section is executed using Ubuntu 18.04 in a laptop with an Intel Core i7-8850H CPU. The reported evaluation times are the average time after executing each test case 100000 times. Each function is code-generated and then compiled using LLVM 9.0.0 with compilation flags `-O3` and `-march=native`, unless stated otherwise.

5.1 Benchmark on Dynamics Expressions and Derivatives

We first evaluate the performance of the RBD derivatives over five robot models [2] in terms of number of atomic operations and evaluation time. The assessment in terms of atomic operations is highlighted in Table 1. The results show a consistent reduction of atomic operations for most of the Jacobians computed by analytical derivatives with respect to AD. This reduction is mainly due to the sparsity handling in Pinocchio's implementations. Moreover, there is a reduction in the number of expensive operations like sine, cosine, and division in $\mathbf{J}^a_{FD}$ and $\mathbf{J}^a_{ID}$. For instance, the number of sin and cos operations in both $\mathbf{J}^a_{FD}$ and $\mathbf{J}^a_{ID}$ is n_b while in $\mathbf{J}^c_{FD}$ and $\mathbf{J}^c_{ID}$ is $2n_b$ for all the evaluated robots. Similarly, the number of divisions is reduced for all cases of $\mathbf{J}^a_{FD}$ compared to those in $\mathbf{J}^c_{FD}$, from a reduction of 25.0% in the double pendulum up to 93.3% in Atlas. Jacobians of ID have no division operations. For detailed information on the count of operation types, see the supplementary material [3].

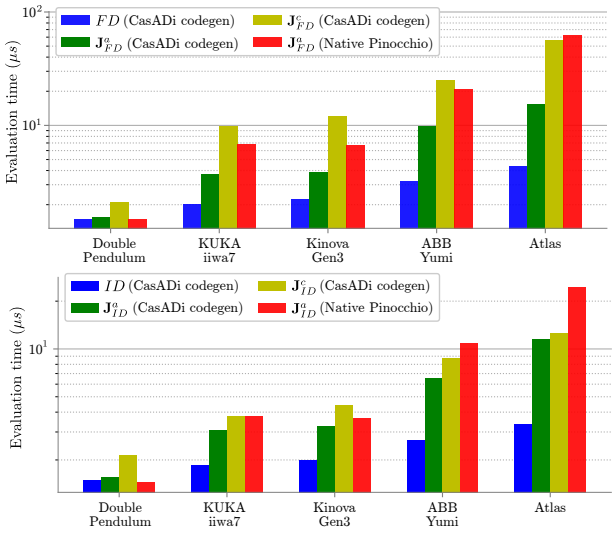

Fig. 1. Comparison of evaluation time for Jacobians of FD (top) and ID (bottom) on 5 different robot models with and without code-generation (codegen). The scales are logarithmic.

[2] The robot models are: a 2-DoF double pendulum, a 7-DoF KUKA iiwa7, a 7-DoF Kinova Gen3, a 14-DoF ABB Yumi (dual manipulator), and a 30-DoF Atlas (humanoid robot). This robots are assumed to have no joints other than revolute joints.
[3] Supplementary material available on: https://git.io/JnNd4

Table 1. Number of atomic operations in forward and inverse dynamics functions and their Jacobians computed both with AD ($\mathbf{J}^c_{FD}, \mathbf{J}^c_{ID}$) and analytical derivatives ($\mathbf{J}^a_{FD}, \mathbf{J}^a_{ID}$).

Robot	n_b	Number of atomic operations					
		FD	$\mathbf{J}^c_{FD}$	$\mathbf{J}^a_{FD}$	ID	$\mathbf{J}^c_{ID}$	$\mathbf{J}^a_{ID}$
Double pendulum	2	85	**292**	327	65	**162**	241
KUKA iiwa7	7	2675	33160	**11874**	1150	9472	**7739**
Kinova Gen3	7	3424	40051	**14565**	1492	11955	**9426**
ABB Yumi	14	6208	99890	**36401**	2416	26264	**22064**
Atlas	30	11564	703094	**164975**	4627	74628	**52604**

If we now turn to the comparison in terms of evaluation time, Fig. 1 shows that, for all code-generated cases, analytical derivatives outperform AD in the computation of Jacobians of RBD. FD and ID are included in the comparison as a reference for the evaluation time of their Jacobians. For code-generated functions, the Jacobian $\mathbf{J}^a_{FD}$ had, on average, an evaluation time 57.99% lower than $\mathbf{J}^c_{FD}$, while the evaluation time of $\mathbf{J}^a_{ID}$ was on average 21.20% lower than $\mathbf{J}^a_{ID}$. Note that the evaluation time of $\mathbf{J}^a_{FD}$ and $\mathbf{J}^a_{ID}$ from native Pinocchio, i.e. without code-generation, are on average 2.35 (respectively 1.42) times slower with respect to their code-generated version. This highlights the potential for speed-up when combing code-generation and appropriate compiler flags.

Two unexpected findings stand out from Fig. 1: (i) for a double pendulum, the evaluation time of $\mathbf{J}^a_{FD}$ (and $\mathbf{J}^a_{ID}$) was lower than in $\mathbf{J}^c_{FD}$ (and $\mathbf{J}^c_{ID}$) despite having a larger number of operations (see Table 1), and (ii) for robots with a small number of bodies (i.e. $n_b \leq 7$) the evaluation time of $\mathbf{J}^a_{FD}$ (and $\mathbf{J}^a_{ID}$) is comparable to evaluating FD (and ID respectively). We suspect an explanation would involve the exact nature and complexity of each atomic operation (see supplementary material).

5.2 Contour-following on a 7-DoF Robot Manipulator

To assess whether and how analytical derivatives contribute to the solution time of OCPs arising from NMPC of robot manipulators, we present a test case of a contour-following task for a 7-DoF Kinova Gen3 robot. For this test case, an NMPC with an underlying OCP of the form (4) is executed, first without overloading RBD derivatives, i.e. using AD, and then overloading RBD derivatives with analytical derivatives. The derivatives in the rest of the OCP are computed with AD. Following the approach in Van Duijkeren (2019), the functions in (4) are defined as

$$V = \left\| \begin{bmatrix} \dot{s}_k - \dot{s}^{ref}_k \\ e_p(\mathbf{q}_k, s_k) \\ \mathbf{q}_k \\ \dot{\mathbf{q}}_k \end{bmatrix} \right\|^2, \quad V_N = \left\| \begin{bmatrix} \mathbf{q}_N \\ \dot{\mathbf{q}}_N \\ e_p(\mathbf{q}_N, s_N) \end{bmatrix} \right\|^2, \qquad (7)$$
$$\zeta = ||e_p(\mathbf{q}_k, s_k)||^2 - \rho^2,$$

where s is a path parameter variable subject to double integrator dynamics that augment the state vector as $x := [\mathbf{q}^\top, \dot{\mathbf{q}}^\top, s, \dot{s}]^\top$, $e_p(\mathbf{q}, s)$ is a funtion for end-effector's position error, $\rho = 0.01$ is an upper bound to e_p, and the prediction horizon is $N = 16$. The speed at which the end-effector follows a reference path is governed by $\dot{s}^{ref}_k$. Recall from section 2.4 that the selection of u as $\dot{\mathbf{q}}$ or $\boldsymbol{\tau}$ determines the expressions for ξ in (4c) and r in (4d). If $u := [\ddot{\mathbf{q}}^\top, \ddot{s}]^\top$ (OCP-ID), then ξ is a discretized representation of a double integrator with appropiate dimensions while r is ID. Contrarily, if $u := [\boldsymbol{\tau}^\top, \ddot{s}]^\top$ (OCP-FD), ξ is a discretized representation of FD stacked with the double integrator dynamics of s, and r is a linear mapping from u to $\boldsymbol{\tau}$. Note that u is augmented with $\ddot{s}$, and the discretized representations in ξ are obtained by applying the explicit Runge-Kutta method.

OCP-FD (OCP-ID) is solved with a variation of the SQP method called the sequential convex quadratic programming method (SCQP) (Verschueren et al. (2016)). This method exploits the convexity in the so-called convex-over-nonlinear functions (i.e. a function composition of an outer convex function and a inner nonlinear function) in the objective and constraints of the OCP. This method avoids computing the exact Hessian $\mathbf{H}_{\mathcal{L}}$ and instead creates an approximation based on the Jacobian of inner nonlinear functions and the Hessian of the outer convex function, which is a scalar for this problem. Thereby, there is no need to compute second-order derivatives when solving OCP-FD or OCP-ID with SCQP. The interested reader is referred to Verschueren et al. (2016) for more information on the selection of these functions. We use QRQP (Andersson et al. (2019)) to solve the inner QP subproblems arising from the SCQP method.

Fig. 2 presents the comparison, in terms of evaluation time, of OCP-FD and OCP-ID being solved with and without analytical derivatives of RBD overloading their AD counterpart. In this figure, there is a clear trend on the OCP solution with analytical derivatives on RBD being faster to solve than those fully depending on AD. In fact, solution of OCP-FD with analytical derivatives on RBD is 1.29 times faster than OCP-FD with AD on RBD, while OCP-ID with analytical derivatives on RBD is only 1.09 faster than its AD counterpart. These results are consistent with those from Fig. 1, where the evaluation time of $\mathbf{J}^a_{FD}$ differ with the evaluation time of $\mathbf{J}^c_{FD}$ in a greater proportion than the difference between evaluation time of $\mathbf{J}^a_{ID}$ and $\mathbf{J}^c_{ID}$.

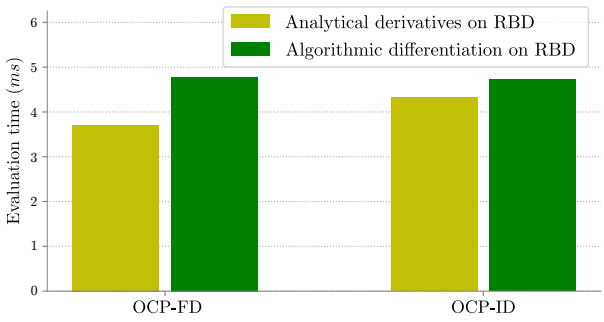

Fig. 2. Comparison of evaluation time of OCP-FD and OCP-ID with and without using analytical derivatives on RBD.

We also compare the computation of the Hessian of $\gamma_{FD} := \lambda^\top FD$ and $\gamma_{ID} := \lambda^\top ID$ as required in the solution of OCP-FD and OCP-ID with SQP. Unlike SCQP, SQP does require the computation of second-order derivatives due to the exact Hessian computation of $\mathbf{H}_\mathcal{L}$. The comparison of $\mathbf{H}^a_{FD}$, $\mathbf{H}^c_{FD}$, $\mathbf{H}^a_{ID}$, $\mathbf{H}^c_{ID}$ in terms of both evaluation time and number of atomic operations, is shown in Table 2.

Table 2. Evaluation time and number of atomic operations of the Hessian of FD and ID computed both with AD ($\mathbf{H}^c_{FD}, \mathbf{H}^c_{ID}$) and analytical derivatives ($\mathbf{H}^a_{FD}, \mathbf{H}^a_{ID}$).

Hessian	Evaluation time (μs)	Number of atomic operations
$\mathbf{H}^a_{FD}$	84.33	135390
$\mathbf{H}^c_{FD}$	**58.84**	**84888**
$\mathbf{H}^a_{ID}$	56.76	77683
$\mathbf{H}^c_{ID}$	**38.31**	**30770**

As Table 2 shows, $\mathbf{H}^c_{FD}$ and $\mathbf{H}^c_{ID}$ computed with AD, have fewer atomic operations and a faster evaluation than $\mathbf{H}^a_{FD}$ and $\mathbf{H}^a_{ID}$, which first-order derivatives are obtained with analytical derivatives. This result is expected, since the computation of $\mathbf{H}^a_{FD}$ and $\mathbf{H}^a_{ID}$ based on the precomputed Jacobian $\mathbf{J}^a_{FD}$ ($\mathbf{J}^a_{ID}$) has a complexity of $O(n_b^2)$, while the computation of $\mathbf{H}^c_{FD}$ and $\mathbf{H}^c_{ID}$ solely based on AD has a complexity of $O(n_b)$, as shown in Section 3.3.

6. CONCLUSION AND FUTURE WORK

In this paper we have shown the benefits of using computationally-efficient functions for both RBD and their derivatives when aiming to reduce the solution time of OCPs involving robot manipulators. Relevant state-of-the-art implementations from a numerical optimization framework and a RBDL were combined and enhanced to allow the transparent use of analytical derivatives of RBD in OCP solution algorithms, without excluding the use of AD for the remainder of the functions in the algorithms. The results of this study indicate that using tailored, analytical derivatives of RBD widely contributes to reduce the solution time of OCPs arising in the context of NMPC of robot manipulators. We showed, however, that computing second-order derivatives of RBD by recursively applying AD leads to more efficient functions compared to applying AD to first-order analytical derivatives.
Future work should focus on the implementation of efficient second-order analytical derivatives of RBD, a quantitative comparison between the presented framework and other optimal control frameworks using RBD, as well as benchmarks of the effect of analytical derivatives within different derivative-based optimization algorithms.

REFERENCES

Andersson, J.A.E., Gillis, J., Horn, G., Rawlings, J.B., and Diehl, M. (2019). CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1), 1–36.

Carpentier, J. and Mansard, N. (2018). Analytical Derivatives of Rigid Body Dynamics Algorithms. In *Robotics: Science and Systems (RSS 2018)*. Pittsburgh, United States.

Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., and Mansard, N. (2019). The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, 614–619. IEEE.

Chen, Y., Bruschetta, M., Picotti, E., and Beghi, A. (2019). Matmpc-A matlab based toolbox for real-time nonlinear model predictive control. *2019 18th European Control Conference, ECC 2019*, 3365–3370.

Docquier, N., Poncelet, A., and Fisette, P. (2013). ROBOTRAN: a powerful symbolic generator of multibody models. *Mechanical Sciences*, 4(1), 199–219.

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

Giftthaler, M., Neunert, M., Stäuble, M., Frigerio, M., Semini, C., and Buchli, J. (2017). Automatic differentiation of rigid body dynamics for optimal control and estimation. *Advanced Robotics*, 31(22), 1225–1237.

Gillis, J., Vandewal, B., Pipeleers, G., and Swevers, J. (2020). Effortless Modeling of Optimal Control Problems With Rockit. In *39th Benelux Meeting on Systems and Control*, 138. Elspeet.

Giordano, A.M., Ott, C., and Albu-Schäffer, A. (2019). Coordinated Control of Spacecraft's Attitude and End-Effector for Space Robots. *IEEE Robotics and Automation Letters*, 4(2), 2108–2115.

Gjerde Johannessen, L.M., Hauan Arbo, M., and Gravdahl, J.T. (2019). Robot Dynamics with URDF & CasADi. In *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*. IEEE.

Koeneman, J., Licitra, G., Alp, M., and Diehl, M. (2019). OpenOCL - Open Optimal Control Library. In *Robotics Science and Systems*.

Lucia, S., Tătulea-Codrean, A., Schoppmeyer, C., and Engell, S. (2017). Rapid development of modular and sustainable nonlinear model predictive control solutions. *Control Engineering Practice*, 60(April 2016), 51–62.

Mastalli, C., Budhiraja, R., Merkt, W., Saurel, G., Hammoud, B., Naveau, M., Carpentier, J., Righetti, L., Vijayakumar, S., and Mansard, N. (2020). Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Millard, D., Heiden, E., Agrawal, S., and Sukhatme, G.S. (2020). Automatic Differentiation and Continuous Sensitivity Analysis of Rigid Body Dynamics. URL `http://arxiv.org/abs/2001.08539`.

Murray, R.M., Sastry, S.S., and Zexiang, L. (1994). *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1st edition.

Neuman, S.M., Koolen, T., Drean, J., Miller, J.E., and Devadas, S. (2019). Benchmarking and Workload Analysis of Robot Dynamics Algorithms. *IEEE International Conference on Intelligent Robots and Systems*, 5235–5242.

Phipps, E. and Pawlowski, R. (2012). Efficient Expression Templates for Operator Overloading-Based Automatic Differentiation. In *Recent Advances in Algorithmic Differentiation*, 309–319. Springer Berlin Heidelberg.

Su, H., Ovur, S.E., Li, Z., Hu, Y., Li, J., Knoll, A., Ferrigno, G., and De Momi, E. (2020). Internet of Things (IoT)-based Collaborative Control of a Redundant Manipulator for Teleoperated Minimally Invasive Surgeries. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 9737–9742. IEEE.

Tedrake, R. (2019). Drake: Model-based design and verification for robotics. URL `https://drake.mit.edu`.

Van Duijkeren, N. (2019). *Online Motion Control in Virtual Corridors - For Fast Robotic Systems*. Ph.D. thesis, KU Leuven.

Verschueren, R., Frison, G., Kouzoupis, D., van Duijkeren, N., Zanelli, A., Novoselnik, B., Frey, J., Albin, T., Quirynen, R., and Diehl, M. (2019). acados: a modular open-source framework for fast embedded optimal control.

Verschueren, R., van Duijkeren, N., Quirynen, R., and Diehl, M. (2016). Exploiting convexity in direct Optimal Control: a sequential convex quadratic programming method. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, 1099–1104.

Walker, M.W. and Orin, D.E. (1982). Efficient Dynamic Computer Simulation of Robotic Mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 104(3), 205–211.