# Analytical Derivatives of Rigid Body Dynamics Algorithms

J. Carpentier

# Analytical Derivatives of Rigid Body Dynamics Algorithms

Justin Carpentier
Laboratoire d'Analyse et d'Architecture des Systèmes
Université de Toulouse
7 avenue du Colonel Roche, Toulouse, FRANCE
Email: justin.carpentier@laas.fr

***Abstract*— Rigid body dynamics is a well-established methodology in robotics. It can be exploited to exhibit the analytic form of kinematic and dynamic functions of the robot model. Two major algorithms, namely the recursive Newton-Euler algorithm (RNEA) and the articulated body algorithm (ABA), have been proposed so far to compute inverse dynamics and forward dynamics in a few microseconds. However, computing their derivatives remains a costly process, either using finite differences (costly and approximate) or automatic differentiation (difficult to implement and suboptimal). As computing the derivatives becomes an important issue (in optimal control, estimation, co-design or reinforcement learning), we propose in this paper new algorithms to efficiently compute them using closed-form formulations. We first explicitly differentiate RNEA, using the chain rule and adequate algebraic differentiation of spatial algebra. Then, using properties about the derivative of function composition, we show that the same algorithm can also be used to compute the derivatives of the forward dynamics with marginal additional cost. To this end, we finally introduce a new algorithm to compute the inverse of the joint-space inertia matrix, without explicitly computing the matrix itself. The algorithms have been implemented in our open-source C++ framework called Pinocchio. The reported benchmarks, based on several robot models, display computational costs varying between 3 microseconds (for a 7-dof arm) up to 17 microseconds (for a 36-dof humanoid), i.e. outperforms state-of-the-art results.**

## I. Introduction

Rigid body dynamics algorithms are now a well-established framework at the heart of many robotic applications. They are also gaining in popularity in related domains as biomechanics and computer animation. This is mostly due to their ability to compute in a generic and efficient way the kinematic and dynamic quantities that describe the motion of poly-articulated systems such as robots, human skeletons or avatars. They are now essential for the control and the stabilization of quadrupedal and humanoid robots [11, 16, 18]. At the same time, optimal control and trajectory optimization are becoming standard approaches to control complex robotic systems [27, 17], generate human-like or avatar motions [29, 23], or in the context of simultaneous design and control of robots [28, 9, 15] for instance. They mostly rely on an accurate integration of the forward dynamics together with the differentiation of this same dynamics with respect to the state, parameters and control variables of the system. Notably, most of the computation time of these optimal control algorithms is precisely spent on this very last step.

Evaluating the partial derivatives of the dynamics can be performed in several manners. The simplest way is to use the finite differences, i.e. calling several times the input dynamics while adding a small increment on the input variables. The main advantage is to systematize the derivation process by considering the input function as a black box. It comes at the price of calling $n + 1$ times the input function (with $n$ the number of input variables). It is also sensitive to numerical rounding errors. Yet, this approach has shown to be fast fast enough to be applied on real system [32, 17] but requires fine parallelization. Another methodology which have been proposed by Garofalo et al. [12] is to analytically derive the Lagrangian equation of motion. Such a derivation gives a better insight into the structure of the derivatives but it leads to dense computations and fails to efficiently exploit the sparsity induced by the kinematic model, in a similar way than rigid body dynamics algorithms do. A last method is to rely on automatic differentiation of rigid body dynamics algorithms as done in Drake [31] and more recently exploited by Giftthaler et al. [13]. The idea is to overload the scalar type of input variables and, knowing the derivatives of basic functions like $\cos, \sin, \exp$, etc., to obtain the partial derivatives of a given output with respect to the input variables by applying the chain rule formula in a systematic way. However, automatic differentiation may require intermediate computations which are hard to avoid or to simplify. To overcome this issue, one solution is to use code generation (to remove redundant computations) as done in [13].

We rather propose to tackle these issues by analytically deriving these algorithms in order to speed up the computation of the derivatives, and also gain a better insight into their mathematical structure. We are then able to exploit the inherent structure of spatial algebra (e.g. the cross product operator) at the root of rigid-body-dynamics algorithms, while the aforementioned approaches are in fact not able to do. A similar approach has been proposed in Lee et al. [19], where the authors derive the Lie theoretic formulation of inverse dynamics in the context of serial chaines with closed loops. This paper differs by also considering poly-articulated systems made of several sub-chains.

This paper is made of two concomitant contributions. In a first contribution we establish in a concise way the analytical derivatives of the inverse dynamics through the derivation

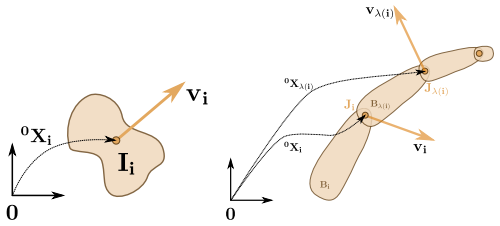


Fig. 1: Spatial notations used all along this paper.

of the so-called recursive Newton-Euler algorithm [20, 8]. The second contribution concerns the analytical derivatives of the forward dynamics: we demonstrate that these derivatives can be directly deduced from the derivatives of the inverse dynamics with only a minor additional cost. We provide all these derivatives inside our C++ framework for rigid-body systems called Pinocchio [5].

Based on the standard notations of rigid-body dynamics (recalled in Sec. II), we make explicit in Sec. III the partial derivatives of the recursive Newton-Euler algorithm (RNEA). Sec. IV then explains how the derivatives of the forward dynamics can be computed from RNEA derivatives. Benchmarks are reported in Sec. V.

## II. Rigid Body Dynamics Notations

Spatial algebra allows to write in a concise manner the kinematics (velocity, acceleration, etc.) and dynamics (force, momenta, etc.) quantities that describe the motion of a rigid body. All over this paper, we use the spatial notations and conventions which have been introduced and popularized in robotics by Featherstone [8]. They are now at the root of many efficient and mature software packages such as HuMAnS [34], RBDL [10], METAPOD [25], Bullet [6] or MuJoCo [32], just to name a few.

Hereafter, we recap all these notations that we exploit later in Sec. III in order to derive the analytical expressions of the partial derivatives of the recursive Newton-Euler algorithm.

### A. Spatial quantities and notations for an isolated rigid body

*1) Placement quantity:* If we consider an isolated body $B_i$ in space endowed with a fixed frame with index $i$ (see Fig. 1), it is firstly described by its *placement* quantity, denoted ${}^{i}\boldsymbol{X}_0$ by Featherstone [8], where the subscript 0 corresponds to the index of the world frame. ${}^{i}\boldsymbol{X}_0$ belongs to $\mathrm{SE}(3)$, the so-called special Euclidian group of rigid transformation of dimension 3.

*2) Kinematics quantities:* The instantaneous time derivative of the placement quantity ${}^{i}\boldsymbol{X}_0$ is given by its *spatial velocity* $\boldsymbol{v}_i$ which belongs to the so-called tangent space of $\mathrm{SE}(3)$, denoted $\mathrm{se}(3)$. A spatial velocity is nothing more than the instantaneous linear and angular velocity of the rigid body.

In a similar way, we can define the *spatial acceleration* of a rigid body, denoted by $\boldsymbol{a}_i$ as the instantaneous time derivative of the spatial velocity.

*3) Kinetic quantities:* If the rigid body is also provided with a mass distribution, we may define its *spatial inertia* $\boldsymbol{I}_i$ characterized by the mass, the center of mass and the rotational inertia of the body. This spatial inertia enables us to introduce two additional quantities which quantify the dynamic properties of the rigid body: (i) the *spatial momentum* given by $\boldsymbol{h}_i \overset{\text{def}}{=} \boldsymbol{I}_i\boldsymbol{v}_i$ which stacks the linear and angular momenta expressed in the body frame; (ii) the *spatial force* given by $\boldsymbol{f}_i \overset{\text{def}}{=} \boldsymbol{I}_i\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{h}_i$ which is nothing more than the time derivative of the spatial momentum quantity. The $\times^*$ operator is made explicit in the next paragraph.

*4) Group actions:* Both spatial velocity and acceleration belong to a specific group, called by Featherstone the group of *motions* [8] which elements are generically denoted by $\boldsymbol{m}$. This means that they have in common similar operators to operate on other spatial quantities. In a similar way, Featherstone encompasses spatial momentum and force into the group of *forces* with elements denoted by $\boldsymbol{f}$. Spatial inertia can then be seen as an operator which maps from the space of motions to the space of forces.

Placement object may act on both motion and force quantities. If we note $A$ and $B$ two Cartesian frames and ${}^{A}\boldsymbol{X}_B$ their relative placement, these operations are respectively denoted by [8]:

$$\boldsymbol{m}_A = {}^{A}\boldsymbol{X}_B\,\boldsymbol{m}_B \tag{1a}$$

$$\boldsymbol{f}_A = {}^{A}\boldsymbol{X}^*_B\,\boldsymbol{f}_B \tag{1b}$$

which reads as "${}^{A}\boldsymbol{X}_B$ *acts on the motion* $\boldsymbol{m}_B$ *expressed in frame* $B$ *to return another motion* $\boldsymbol{m}_A$ *expressed in frame* $A$*"*.

Motion object may also operate on both motion and force objects through the notion of *spatial cross product* operations, which are in some sense similar to classic time derivatives:

$$\dot{\boldsymbol{m}} = \boldsymbol{v}_A \times \boldsymbol{m} \tag{2a}$$

$$\dot{\boldsymbol{f}} = \boldsymbol{v}_A \times^* \boldsymbol{f} \tag{2b}$$

where $\dot{\boldsymbol{m}}$ is the time variation of $\boldsymbol{m}$ and both $\boldsymbol{m}$ and $\boldsymbol{f}$ are spatial quantities which are fixed when they are expressed in frame $A$ moving at the spatial velocity $\boldsymbol{v}_A$. Following these notations, the time derivative of an inertia element $\boldsymbol{I}_A$ attached to a frame A is given by:

$$\dot{\boldsymbol{I}}_A = \boldsymbol{v}_A \times^* \boldsymbol{I}_A - \boldsymbol{I}_A\boldsymbol{v}_A\times \tag{3}$$

Finally, Featherstone has shown in his book [8, p.28] that:

$${}^{A}\dot{\boldsymbol{X}}_B\,\boldsymbol{m} = (\boldsymbol{v}_B - \boldsymbol{v}_A) \times {}^{A}\boldsymbol{X}_B\,\boldsymbol{m} \tag{4}$$

which corresponds to the time variation of the action operator of the placement ${}^{A}\boldsymbol{X}_B$ onto a motion $\boldsymbol{m}$. This last relation will be of primal importance in Sec. III.

All these operators have an explicit expression in the book of Featherstone [8], while further details on the mathematical structure can be found in Murray et al. [24].

### B. Spatial quantities and notations for poly-articulated systems

A poly-articulated system like a humanoid robot or a robotic arm is composed of several rigid bodies that are linked together through articulations called joints (see Fig. 1).

A joint can be seen as a constraint which limits the relative displacement between two consecutive bodies: the relative transformation $\boldsymbol{X}_{\text{J}}$ governed by the joint only covers a subset of $\text{SE}(3)$. The relative transformation between the frame $i$ and the frame of the parent body $\lambda(i)$ is given by ${}^{i}\boldsymbol{X}_{\lambda(i)} \overset{\text{def}}{=} \boldsymbol{X}_{\text{J}}\,\boldsymbol{X}_T(i)$ where $\boldsymbol{X}_T(i)$ is the placement of the joint with respect to the frame attached to $\lambda(i)$.

This partial covering of rigid transformation space can be also observed at the joint motion level. If we note $\boldsymbol{q}_i$ the minimal coordinates representation of the transformation $\boldsymbol{X}_{\text{J}}$ and $\dot{\boldsymbol{q}}_i$ the instantaneous time variations of $\boldsymbol{q}_i$, the instantaneous joint velocity $\boldsymbol{v}_{\text{J}}$ is given by:

$$\boldsymbol{v}_{\text{J}} \overset{\text{def}}{=} \frac{\partial \boldsymbol{X}_{\text{J}}}{\partial \boldsymbol{q}_i}\dot{\boldsymbol{q}}_i = \boldsymbol{S}_i\dot{\boldsymbol{q}}_i \tag{5}$$

where $\boldsymbol{S}_i \overset{\text{def}}{=} \frac{\partial \boldsymbol{X}_{\text{J}}}{\partial \boldsymbol{q}_i}$ is the joint motion subspace matrix whose columns only covers a subpart of the tangent space $\text{se}(3)$ when evaluated at $\boldsymbol{q}_i$. In the previous equations, we have considered that the joint subspace motion $\boldsymbol{S}_i$ are independent from the time, which is the case for most of the joints used in robotics. Yet, this hypothesis is not a limitation as such dependency can be easily handled by adding a drift term $\boldsymbol{\sigma}_{\text{J}}(t, \boldsymbol{q}_i)$ in the expression of $\boldsymbol{v}_{\text{J}}$ in Eq. (5).

We may again derive $\boldsymbol{v}_{\text{J}}$ with respect to time to get:

$$\boldsymbol{a}_{\text{J}} = \boldsymbol{S}_i\ddot{\boldsymbol{q}}_i + \boldsymbol{c}_{\text{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\text{J}} \tag{6}$$

where $\boldsymbol{c}_{\text{J}}$ collects all the drifting terms due to the variations of $\boldsymbol{S}_i$ with respect to $\boldsymbol{q}_i$ and $\ddot{\boldsymbol{q}}_i$ is time derivative of $\dot{\boldsymbol{q}}_i$. In other terms, we have:

$$\boldsymbol{c}_{\text{J}} \overset{\text{def}}{=} \left(\dot{\boldsymbol{q}}_i^T \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_i}\right)\dot{\boldsymbol{q}}_i \tag{7}$$

where $\frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_i}$ is a tensor expression often equal to zero for most of the classic joints. A closed form computation of $\dot{\boldsymbol{q}}_i^T \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_i}$ is given by:

$$\dot{\boldsymbol{q}}_i^T \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_i} = \sum_{k=1}^{n_i} \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_i^k}\dot{\boldsymbol{q}}_i^k \tag{8}$$

with $n_i$ the number of degrees of freedom (dof) of joint $i$ and $\boldsymbol{q}_i^k$ is the $k$-th component of the configuration vector $\boldsymbol{q}_i$. We refer again to [8] for further details on these terms.

## III. Analytical derivatives of the Recursive Newton-Euler Algorithm

In this section, we derive the analytical expressions of the partial derivatives of the inverse dynamics (ID) function in the context of rigid-body systems. Inverse dynamics allows to compute the generalized torque $\boldsymbol{\tau}$ to apply on a rigid-body system ($model$) in order to produce a desired generalized acceleration $\ddot{\boldsymbol{q}}$ giving the current generalized position $\boldsymbol{q}$ and velocity $\dot{\boldsymbol{q}}$ of the system together with the stack external forces $\boldsymbol{f}^{\text{ext}}$:

$$\boldsymbol{\tau} = \text{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{f}^{\text{ext}})$$

**Algorithm 1 -** Pseudo code of the recursive Newton-Euler algorithm as exposed by Featherstone [8, p. 96]

*Initialization step:*

$\boldsymbol{v}_0 = \boldsymbol{0}$

$\boldsymbol{a}_0 = -\boldsymbol{a}_g$ // initialize the world spatial acceleration with the gravity field value

*Forward pass:*

**for** $i = 1$ **to** $N_B$ **do**

  $[\boldsymbol{X}_{\text{J}}, \boldsymbol{S}_i, \boldsymbol{v}_{\text{J}}, \boldsymbol{c}_{\text{J}}] = \text{jcalc}(\text{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$

  ${}^{i}\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\text{J}}\,\boldsymbol{X}_T(i)$

  $\boldsymbol{v}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\text{J}}$

  $\boldsymbol{a}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\ddot{\boldsymbol{q}}_i + \boldsymbol{c}_{\text{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\text{J}}$

  $\boldsymbol{h}_i = \boldsymbol{I}_i\boldsymbol{v}_i$

  $\boldsymbol{f}_i = \boldsymbol{I}_i\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{h}_i - \boldsymbol{f}_i^{\text{ext}}$

**end**

*Backward pass:*

**for** $i = N_B$ **to** $1$ **do**

  $\boldsymbol{\tau}_i = \boldsymbol{S}_i^T\boldsymbol{f}_i$

  **if** $\lambda(i) \neq 0$ **then**

    $\boldsymbol{f}_{\lambda(i)} = \boldsymbol{f}_{\lambda(i)} + {}^{\lambda(i)}\boldsymbol{X}_i^*\boldsymbol{f}_i$

  **end**

**end**

Using the Lagrangian formalism, the inverse dynamics reads as:

$$\boldsymbol{\tau} = M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) - \sum_i J_i(\boldsymbol{q})^T\boldsymbol{f}_i^{\text{ext}} \tag{9}$$

where $M$ stands for the joint space inertia matrix, $C$ is the Coriolis matrix, $\boldsymbol{g}$ encompasses the gravity effects and $J_i$ are the frame Jacobian where external forces are expressed. For clarity reasons, we remove from these quantities the explicit dependency on the input $model$.

Efficient algorithms have been proposed in the literature to solve the inverse dynamics problem [8]. The most efficient one still remains the RNEA (see Alg. 1) whose complexity is linear in the number of bodies composing the rigid-body system. It was originally proposed by Luh et al. [20] in the 80's and generalized by Featherstone [8] to exploit the computational structure of spatial algebra.

The partial derivatives of the inverse dynamics then correspond to the variations of the torque output according to the input variables $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$. With the Lagrangian notations, these partial derivatives correspond to:

$$\frac{\partial\,\text{ID}}{\partial \boldsymbol{q}} = \frac{\partial M}{\partial \boldsymbol{q}}\ddot{\boldsymbol{q}} + \frac{\partial C}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}} - \sum_i \frac{\partial J_i^T}{\partial \boldsymbol{q}}\boldsymbol{f}_i^{\text{ext}} \tag{10a}$$

$$\frac{\partial\,\text{ID}}{\partial \dot{\boldsymbol{q}}} = \frac{\partial C}{\partial \dot{\boldsymbol{q}}}\dot{\boldsymbol{q}} + C \tag{10b}$$

$$\frac{\partial\,\text{ID}}{\partial \ddot{\boldsymbol{q}}} = M \tag{10c}$$

where we drop the dependency on input variables for better clarity. At this stage, several observations come:

(i) the quantities $\frac{\partial M}{\partial \boldsymbol{q}}$, $\frac{\partial C}{\partial \boldsymbol{q}}$, $\frac{\partial J_i^T}{\partial \boldsymbol{q}}$ and $\frac{\partial C}{\partial \dot{\boldsymbol{q}}}$ which appear in (10a) and (10b) are large tensor matrices that are hard to explicit and they require larger capacity storage than the three partial derivatives we wish to compute. Some closed-form expressions of these tensors have been proposed in [12] using the Lagrangian formalism, requiring intensive computations.

(ii) the partial derivative of the torque with respect to the joint acceleration quantity (10c) is simply the joint space inertia matrix. They already exist efficient algorithms to compute this last quantity as the composite rigid body algorithm (CRBA) originally proposed by Walker and Orin [33].

In the following, we aim at exploiting the simplicity and the efficiency of the rigid-body dynamics algorithms like RNEA and CRBA in order to derive the analytical expressions of the partial derivatives $\frac{\partial \text{ID}}{\partial \boldsymbol{q}}, \frac{\partial \text{ID}}{\partial \dot{\boldsymbol{q}}}, \frac{\partial \text{ID}}{\partial \ddot{\boldsymbol{q}}}$ using spatial notations while avoiding complex computations as the aforementioned tensor expressions.

### A. The recursive Newton-Euler algorithm

As previously mentioned, RNEA is the most effective way to solve the inverse dynamics problem by exploiting the structured sparsity induced by the kinematic model. It is a two-pass algorithm which propagates the kinematic quantities in a first forward pass (similar to a forward kinematics), then collect the torque contribution of the subtrees in a second backward pass. The Algorithm 1 recaps the spatial operations performed inside the recursive Newton-Euler algorithm. Compared to Featherstone [8, p. 96], we assume here that the external contact forces are already expressed in the joint frames[1].

In the forward pass of RNEA, jcalc is the function which computes the forward kinematics at the joint level and for a given joint type jtype$(i)$, according to its current configuration vector $\boldsymbol{q}_i$ and the corresponding velocity vector $\dot{\boldsymbol{q}}_i$. $\boldsymbol{X}_\text{J}, \boldsymbol{S}_i, \boldsymbol{v}_\text{J}$ and $\boldsymbol{c}_\text{J}$ are spatial quantities output by jcalc and they only depend on the current joint configuration $\boldsymbol{q}_i$ and velocity $\dot{\boldsymbol{q}}_i$. While $\boldsymbol{v}_i, \boldsymbol{a}_i$ and $\boldsymbol{f}_i$ are also dependent to the motions of the parent bodies by forward recursion.

In the backward pass of RNEA, both $\boldsymbol{f}_i$ and $\boldsymbol{\tau}_i$ quantities are also subject to the dynamics of the subtree rooted at the joint $i$ through the backward recursion.

### B. Generic partial derivatives of the recursive Newton-Euler algorithm

In what follows, we derive in a generic manner the computations performed by the RNEA. We will not derive RNEA with respect to $\ddot{\boldsymbol{q}}$ because we already know Eq. (10c) that $\frac{\partial \text{ID}}{\partial \ddot{\boldsymbol{q}}} = M$, and CRBA is already very efficient to perform this precise computation.

We denote by $\boldsymbol{u}$ an arbitrary vector which stands either for $\boldsymbol{q}$ or $\dot{\boldsymbol{q}}$. And we use the chain rule formula in order to derive the basic spatial operations performed in Algorithm 1. For a better clarity, we separate the derivations for the forward and the backward passes.

[1]Most of sensors used in robotics to estimate the action of external forces (e.g. force-torque sensors or tactile skins) provide measurements already expressed in a frame attached to the joints themselves.

**Algorithm 2 -** Partial derivatives of the RNEA forward pass inner loop.

1: $\left[\frac{\partial \boldsymbol{X}_\text{J}}{\partial \boldsymbol{u}}, \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{u}}, \frac{\partial \boldsymbol{v}_\text{J}}{\partial \boldsymbol{u}}, \frac{\partial \boldsymbol{c}_\text{J}}{\partial \boldsymbol{u}}\right] = \text{jcalc}(\text{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$

2: $\frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \boldsymbol{u}} = \frac{\partial \boldsymbol{X}_\text{J}}{\partial \boldsymbol{u}} \boldsymbol{X}_T(i)$

3: $\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}} = \frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \boldsymbol{u}} \boldsymbol{v}_{\lambda(i)} + {}^i\boldsymbol{X}_{\lambda(i)} \frac{\partial \boldsymbol{v}_{\lambda(i)}}{\partial \boldsymbol{u}} + \frac{\partial \boldsymbol{v}_\text{J}}{\partial \boldsymbol{u}}$

4: $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{u}} = \boldsymbol{I}_i \frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}}$

5: $\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{u}} = \frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \boldsymbol{u}} \boldsymbol{a}_{\lambda(i)} + {}^i\boldsymbol{X}_{\lambda(i)} \frac{\partial \boldsymbol{a}_{\lambda(i)}}{\partial \boldsymbol{u}} + \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{u}} \ddot{\boldsymbol{q}}_i + \frac{\partial \boldsymbol{c}_\text{J}}{\partial \boldsymbol{u}} + \frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}} \times \boldsymbol{v}_\text{J} + \boldsymbol{v}_i \times \frac{\partial \boldsymbol{v}_\text{J}}{\partial \boldsymbol{u}}$

6: $\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}} = \boldsymbol{I}_i \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{u}} + \frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}} \times^* \boldsymbol{h}_i + \boldsymbol{v}_i \times^* \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{u}}$

**Algorithm 3 -** Partial derivatives of the RNEA backward pass inner loop.

1: $\frac{\partial \boldsymbol{\tau}_i}{\partial \boldsymbol{u}} = \frac{\partial \boldsymbol{S}_i^T}{\partial \boldsymbol{u}} \boldsymbol{f}_i + \boldsymbol{S}_i^T \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}}$

2: **if** $\lambda(i) \neq 0$ **then**

3: $\quad \frac{\partial \boldsymbol{f}_{\lambda(i)}}{\partial \boldsymbol{u}} = \frac{\partial \boldsymbol{f}_{\lambda(i)}}{\partial \boldsymbol{u}} + \frac{\partial^{\lambda(i)} \boldsymbol{X}_i^*}{\partial \boldsymbol{u}} \boldsymbol{f}_i + {}^{\lambda(i)}\boldsymbol{X}_i^* \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}}$

4: **end**

*1) Partial derivatives of the forward pass:* Algorithm 2 shows the partial derivatives of the spatial quantities involved in the forward pass of the RNEA. We deliberately omit the partial derivative of the joint placement variables $\boldsymbol{X}_T(i)$ with respect to $\boldsymbol{u}$ as it is a fixed quantity independent from the kinematic variables $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. The same rule applies for the spatial inertias $\boldsymbol{I}_i$ on line 6.

Due to the forward recursion, and similarly to the observations made on RNEA in Sec. III-A, we might see on lines 3 and 5 that both $\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}}$ and $\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{u}}$ directly depend on the partial derivatives of their parent bodies through $\frac{\partial \boldsymbol{v}_{\lambda(i)}}{\partial \boldsymbol{u}}$ and $\frac{\partial \boldsymbol{a}_{\lambda(i)}}{\partial \boldsymbol{u}}$. The other partial derivatives $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{u}}$ and $\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}}$ directly rely on the internal partial derivatives of the current joint $i$ and indirectly to the parent bodies motion via $\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}}$ and $\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{u}}$.

It is also worth to notice that on one side $\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}}$ and $\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{u}}$ are motion-sets, namely collections of motion vectors which have been stacked inside a matrix. And on the other side, $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{u}}$ and $\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}}$ are force-sets, that are collections of force vectors also stacked inside a matrix.

*2) Partial derivatives of the backward pass:* Algorithm 3 depicts how the partial derivatives of the joint torque $\frac{\partial \boldsymbol{\tau}_i}{\partial \boldsymbol{u}}$ are affected by the variations of the joint motion subspace

$\boldsymbol{S}_i$ and the variation of the force-set supported by joint $i$. This backward loop mostly propagates the partial derivatives computed in the forward pass towards the kinematic tree.

It is important to notice at this stage that the structured sparsity of RNEA is also preserved by directly applying the chain rule on it.

### C. Simplifying expressions

Depending on the value of $\boldsymbol{u}$, some partial derivatives present in Algorithms 2 or 3 may vanish because they are independent from either $\boldsymbol{q}$ or $\dot{\boldsymbol{q}}$. We now detail theses simplifications for each single line of Algorithms 2 and 3 in order to give in the end, a complete and directly applicable version of the recursive derivatives. This is certainly the most technical part of this paper.

*1)* **Algorithm 2, line 1:** $\boldsymbol{X}_{\text{J}}$ and $\boldsymbol{S}_i$ only depends on the configuration $\boldsymbol{q}_i$ of joint $i$. It follows that $\frac{\partial \boldsymbol{X}_{\text{J}}}{\partial \boldsymbol{u}}$ and $\frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{u}}$ are only non-zero for $\boldsymbol{u} = \boldsymbol{q}_i$. The values of $\frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_i}$ depend on the type of joint (given by jtype$(i)$) and they are mostly equal to zero (e.g. revolute, prismatic, free-flyer joints). And as introduced in Sec. II, $\frac{\partial \boldsymbol{X}_{\text{J}}}{\partial \boldsymbol{q}_i}$ is nothing more than the motion subspace $\boldsymbol{S}_i$.

We can deduce from Sec. II that $\frac{\partial \boldsymbol{v}_{\text{J}}}{\partial \boldsymbol{q}_i} = \dot{\boldsymbol{q}}_i^T \frac{\partial \boldsymbol{S}_i}{\partial \boldsymbol{q}_i}$ and $\frac{\partial \boldsymbol{v}_{\text{J}}}{\partial \dot{\boldsymbol{q}}_i} = \boldsymbol{S}_i$. $\dfrac{\partial \boldsymbol{c}_{\text{J}}}{\partial \boldsymbol{u}}$ for both $\boldsymbol{q}_i$ and $\dot{\boldsymbol{q}}_i$ correspond to the evaluation of complex tensor expressions, often equal to zero, that is why we omit their analytical expressions for brevity.

*2)* **Algorithm 2, line 2:** As shown in the next paragraph, the result of $\dfrac{\partial \boldsymbol{X}_{\text{J}}}{\partial \boldsymbol{u}} \boldsymbol{X}_T(i)$ is not required and can be skipped. What it is important to notice here is that $\frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \dot{\boldsymbol{q}}_k} = 0$ for any $k$ because $\boldsymbol{X}_{\text{J}}$ depends only on the joint configuration $\boldsymbol{q}_i$.

*3)* **Algorithm 2, line 3:** It is already well-known and detailed in [8] that $\frac{\partial \boldsymbol{v}_i}{\partial \dot{\boldsymbol{q}}}$ corresponds to the kinematic Jacobian of the joint $i$. Concerning the derivatives with respect to $\boldsymbol{q}$, we can observe that ${}^i\boldsymbol{X}_{\lambda(i)} \frac{\partial \boldsymbol{v}_{\lambda(i)}}{\partial \boldsymbol{q}}$ corresponds to the action of the relative transformation ${}^i\boldsymbol{X}_{\lambda(i)}$ onto the columns of the motion-set $\frac{\partial \boldsymbol{v}_{\lambda(i)}}{\partial \boldsymbol{q}}$, which can do by exploiting the sparsity induced by the kinematic tree. And the term $\frac{\partial \boldsymbol{v}_{\text{J}}}{\partial \boldsymbol{u}}$ for $\boldsymbol{u} = \boldsymbol{q}$ has already been detailed.

The most delicate part concerns the first term $\frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \boldsymbol{u}} \boldsymbol{v}_{\lambda(i)}$ with $\boldsymbol{u} = \boldsymbol{q}_i$ (otherwise the term is equal to zero). From Eq. (4), we have:

$$\frac{d\, {}^i\boldsymbol{X}_{\lambda(i)}}{dt} \boldsymbol{m} = -\boldsymbol{v}_{\text{J}} \times {}^i\boldsymbol{X}_{\lambda(i)}\, \boldsymbol{m} = \left({}^i\boldsymbol{X}_{\lambda(i)}\, \boldsymbol{m}\right) \times \boldsymbol{S}_i \dot{\boldsymbol{q}}_i \quad (11)$$

by definition of $\boldsymbol{v}_{\text{J}}$ in Eq. (5). We can also show that:

$$\frac{d\, {}^i\boldsymbol{X}_{\lambda(i)}}{dt} \boldsymbol{m} = \left( \frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \boldsymbol{q}_i} \dot{\boldsymbol{q}}_i \right) \boldsymbol{m} \quad (12)$$

As these two expressions (11) and (12) are linear with respect to the components $\dot{\boldsymbol{q}}_i^k$ of $\dot{\boldsymbol{q}}_i$ and they are valid for any motion $\boldsymbol{m}$, we may identify the operator expression $\frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \boldsymbol{q}_i} \boldsymbol{v}_{\lambda(i)}$ to be:

$$\frac{\partial\, {}^i\boldsymbol{X}_{\lambda(i)}}{\partial \boldsymbol{q}_i} \boldsymbol{v}_{\lambda(i)} = \left({}^i\boldsymbol{X}_{\lambda(i)}\, \boldsymbol{v}_{\lambda(i)}\right) \times \boldsymbol{S}_i \quad (13)$$

where this operation must be understood as a column-wise operation of the spatial vector ${}^i\boldsymbol{X}_{\lambda(i)}\, \boldsymbol{v}_{\lambda(i)}$ on the motion-set $\boldsymbol{S}_i$. A more rigorous demonstration can be done using the formalism of Lie groups and Lie algebra [24], which would require the introduction of additional notations that are out of the scope of this paper.

*4)* **Algorithm 2, line 4:** The computations done on this line are simply the action of the inertia $\boldsymbol{I}_i$ on the motion-sets $\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}}$.

*5)* **Algorithm 2, line 5:** The operations done on this line look very similar to the ones performed on line 3, especially for the three first terms. We have already seen how to evaluate $\frac{\partial \boldsymbol{c}_{\text{J}}}{\partial \boldsymbol{u}}$ in Sec. III-C1. It is worth to notice again that for most of the joints, $\frac{\partial \boldsymbol{v}_{\text{J}}}{\partial \boldsymbol{q}_i} = 0$ and we have for $\boldsymbol{u} = \dot{\boldsymbol{q}}_i$, $\frac{\partial \boldsymbol{v}_{\text{J}}}{\partial \dot{\boldsymbol{q}}_i} = \boldsymbol{S}_i$ and $\boldsymbol{v}_i \times \boldsymbol{S}_i = 0$, which implies that most of the time, the last term of line 5 is equal to zero.

*6)* **Algorithm 2, line 6:** This line presents basic operations on motion and force sets. All the same, it is importance to notice that in $\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{u}} \times^* \boldsymbol{h}_i$, the operator $\boldsymbol{m} \times^* \boldsymbol{h}_i$ acting on any motion $\boldsymbol{m}$, can be interpreted as a linear operator with a spatial skew matrix representation.

*7)* **Algorithm 3, line 1:** For most of the joints, $\frac{\partial \boldsymbol{S}_i^T}{\partial \boldsymbol{q}_i} = 0$. And it appears that $\boldsymbol{S}_i^T \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}}$ is simply a matrix product, which can be evaluated following the sparsity induced by the kinematic tree.

*8)* **Algorithm 3, line 3:** Following the same reasoning than in Sec. III-C3, we can show that:

$$\frac{\partial^{\lambda(i)} \boldsymbol{X}_i^*}{\partial \boldsymbol{u}} \boldsymbol{f}_i = {}^{\lambda(i)}\boldsymbol{X}_i^* \left( \boldsymbol{S}_i \times^* \boldsymbol{f}_i \right) \quad (14)$$

which has to be again interpreted as a column-wise operator of the columns of $\boldsymbol{S}_i$ on the spatial force $\boldsymbol{f}_i$.

### D. Direct outcome of these derivations

Finally, a direct outcome of these computations is the analytical expressions of the partial derivatives of the forward kinematics, trough the quantities $\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{q}}$, $\frac{\partial \boldsymbol{v}_i}{\partial \dot{\boldsymbol{q}}}$ and $\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{q}}$, $\frac{\partial \boldsymbol{a}_i}{\partial \dot{\boldsymbol{q}}}$. Indeed, these four last terms refer to the partial derivatives of the spatial velocity and spatial acceleration of the joint $i$ with respect to the joint configuration and velocity vectors. They come for free with the direct derivation of RNEA.

## IV. Analytical derivatives of the Forward Dynamics

Forward dynamics (FD) is the reciprocal of inverse dynamics. In other words, it computes the generalized acceleration $\ddot{\boldsymbol{q}}$ of the rigid-body system according to the current generalized position $\boldsymbol{q}$, velocity $\dot{\boldsymbol{q}}$, torque input $\boldsymbol{\tau}$ and external forces $\boldsymbol{f}^{\text{ext}}$:

$$\ddot{\boldsymbol{q}} = \text{FD}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}, , \boldsymbol{f}^{\text{ext}})$$

Using the Lagrangian notations, the forward dynamics reads:

$$\ddot{\boldsymbol{q}} = M(\boldsymbol{q})^{-1} \left( \boldsymbol{\tau} - C(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q}) + \sum_i J_i(\boldsymbol{q})^T \boldsymbol{f}_i^{\text{ext}} \right) \quad (15)$$

Similarly to inverse dynamics, efficient recursive rigid-body algorithms have been proposed to solve Eq. (15). One of the most efficient one is the ABA, introduced by Featherstone in the 80's [7]. Similarly to RNEA, the algorithmic complexity of ABA is linear in the number of bodies composing the rigid-body system. One of the main feature of ABA is to not rely on the explicit inverse of the joint space inertia matrix $M$, allowing to save computation times.

Yet, ABA is much more complex than RNEA as it is composed of three main recursions that we briefly summarize here. In the first recursion, the kinematic quantities are propagated along the tree structure. The second recursion corresponds to a backward pass where the spatial forces which act on bodies are computed from the joint torque input. In the last recursion, the spatial accelerations of bodies are then deduced, allowing to compute the joint acceleration vector.

### *A. Lagrangian expressions of the partial derivatives of forward dynamics*

The partial derivatives of the forward dynamics correspond to the variations of the joint acceleration $\ddot{\boldsymbol{q}}$ with respect to the input variables $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau})$, which gives:

$$\begin{aligned}\frac{\partial \text{FD}}{\partial \boldsymbol{q}} &= \frac{\partial M^{-1}}{\partial \boldsymbol{q}}\left(\boldsymbol{\tau} - C\dot{\boldsymbol{q}} - \boldsymbol{g} + \sum_i J_i^T \boldsymbol{f}_i^{\text{ext}}\right) \\ &\quad + M^{-1}\left(\sum_i \frac{\partial J_i^T}{\partial \boldsymbol{q}}\boldsymbol{f}_i^{\text{ext}} - \frac{\partial C}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}\right) \qquad (16a)\\ \frac{\partial \text{FD}}{\partial \dot{\boldsymbol{q}}} &= M^{-1}\left(\frac{\partial C}{\partial \dot{\boldsymbol{q}}}\dot{\boldsymbol{q}} + C\right) \qquad (16b)\\ \frac{\partial \text{FD}}{\partial \boldsymbol{\tau}} &= M^{-1} \qquad (16c)\end{aligned}$$

From these expressions, some remarks can be raised:

(i) we already mentioned in Sec. III the difficulties to explicitly compute the tensors quantities $\frac{\partial C}{\partial \boldsymbol{q}}$, $\frac{\partial C}{\partial \dot{\boldsymbol{q}}}$ and $\frac{\partial M}{\partial \boldsymbol{q}}$. A similar comment holds for $\frac{\partial M^{-1}}{\partial \boldsymbol{q}}$ which can also be deduced from $\frac{\partial M}{\partial \boldsymbol{q}}$ through the relation:

$$\frac{\partial M^{-1}}{\partial \boldsymbol{q}} = -M^{-1}\frac{\partial M}{\partial \boldsymbol{q}}M^{-1}$$

(ii) from Eq. (10c) and Eq. (16c), we can observe that the partial derivatives of the inverse and forward dynamics with respect to their third input argument ($\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ respectively) are inverse one from each other. This leads to the following mathematical relation:

$$\frac{\partial \text{FD}}{\partial \boldsymbol{\tau}} = M^{-1} = \frac{\partial \text{ID}}{\partial \ddot{\boldsymbol{q}}}^{-1} \qquad (17)$$

### *B. Link between analytical derivatives of forward and inverse dynamics*

Due to the three recursions present in ABA, deriving analytically ABA is much more laborious than in the case of RNEA, as it involves additional intermediate computations. Despite that, we show hereafter how the partial derivatives of the forward dynamics can be obtained from the partial derivatives of the inverse dynamics, and hence from the ones of RNEA which have been introduced in Sec. III.

As aforementioned, forward dynamics is the reciprocal of inverse dynamics, which means that these two functions are linked by the following identity:

$$\text{ID} \circ \text{FD} = id \qquad (18)$$

where $\circ$ denotes the composition operator and $id$ corresponds to the identity function (i.e. for any input $\boldsymbol{x}$, $id(\boldsymbol{x}) = \boldsymbol{x}$). Evaluated at any given entry $(\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0, \boldsymbol{\tau}_0)$, Eq. (18) also reads:

$$\text{ID}(model, \boldsymbol{q}_0, \dot{\boldsymbol{q}}_0, \text{FD}(model, \boldsymbol{q}_0, \dot{\boldsymbol{q}}_0, \boldsymbol{\tau}_0)) = \boldsymbol{\tau}_0 \qquad (19)$$

and for convenience in the notations, we set:

$$\ddot{\boldsymbol{q}}_0 \overset{\text{def}}{=} \text{FD}(model, \boldsymbol{q}_0, \dot{\boldsymbol{q}}_0, \boldsymbol{\tau}_0) \qquad (20)$$

We omit the here the dependency on external forces $\boldsymbol{f}^{\text{ext}}$ for better readability. Applying the chain rule formula on Eq. (19), we obtain the following point-wise equality:

$$\frac{\partial \text{ID}}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\ddot{\boldsymbol{q}}_0} + \frac{\partial \text{ID}}{\partial \ddot{\boldsymbol{q}}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\ddot{\boldsymbol{q}}_0}\frac{\partial \text{FD}}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\boldsymbol{\tau}_0} = \frac{\partial \boldsymbol{\tau}_0}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\ddot{\boldsymbol{q}}_0} \qquad (21)$$

where $\boldsymbol{u}$ indistinctly denotes either $\boldsymbol{q}$ or $\dot{\boldsymbol{q}}$. As $\boldsymbol{\tau}_0$ is given and fixed, we have:

$$\frac{\partial \boldsymbol{\tau}_0}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\ddot{\boldsymbol{q}}_0} = 0 \qquad (22)$$

for any value of $\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0$ and $\ddot{\boldsymbol{q}}_0$. And we know from Eq. (10c) that:

$$\frac{\partial \text{ID}}{\partial \ddot{\boldsymbol{q}}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\ddot{\boldsymbol{q}}_0} = M(\boldsymbol{q}_0) \qquad (23)$$

which leads to:

$$\frac{\partial \text{FD}}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\boldsymbol{\tau}_0} = -M^{-1}(\boldsymbol{q}_0)\frac{\partial \text{ID}}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{q}_0,\dot{\boldsymbol{q}}_0,\ddot{\boldsymbol{q}}_0} \qquad (24)$$

as the joint space inertia matrix is always invertible.

It follows from Eqs. (17) and (24) that the partial derivatives of the forward dynamics can be directly deduced from the derivatives of the inverse dynamics. To the best of our knowledge, this is the first time that this specific relation between the partial derivatives of forward and inverse dynamics is highlighted and exploited in order to simplify the underlying computations.

To summarize the proposed approach, we have shown that it is sufficient to compute the inverse of the joint space inertia matrix and the partial derivatives of inverse dynamics, in order to get the partial derivatives of forward dynamics. It is also important to notice at this stage that, if we have a direct access to the partial derivatives of forward dynamics, it will be possible to directly compute from these derivatives the partial derivatives of inverse dynamics. This is made possible through the inherent relations (18) and (24) that link together inverse and forward dynamics as well as their partial derivatives.

TABLE I: Summary table of mean computation times for derivatives of forward and inverse dynamics.

| | KUKA-LWR | HyQ | Atlas |
|---|---|---|---|
| ID | 1.20 us | 2.14 us | 5.51 us |
| analytical derivatives of ID | 3.34 us | 7.01 us | 16.72 us |
| finite differences of ID | 21.26 us | 88.52 us | 452.46 us |
| FD | 1.78 us | 4.28 us | 9.81 us |
| analytical derivatives of FD | 5.78 us | 14.24 us | 45.20 us |
| finite differences of FD | 22.67 us | 94.23 us | 470.14 us |
| $M^{-1}$ dedicated algorithm | 1.82 us | 4.86 us | 12.70 us |
| $M^{-1}$ Cholesky factorization | 1.88 us | 5.82 us | 28.29 us |

### C. Computing the inverse of the joint space inertia matrix

The last difficulty lies in the computation of the inverse of the joint space inertia matrix denoted $M^{-1}$. The standard approach consists in first computing the joint space inertia matrix $M$ using CRBA and then performing its sparse Cholesky decomposition by employing a dedicated algorithm proposed in [8, p. 112]. Such a decomposition can be written as:

$$M = L\,D\,L^T \tag{25}$$

where $L$ is a lower triangular matrix and $D$ is a diagonal matrix. It follows from Eq. (25) that the expression of the inverse of the joint space inertia matrix is given by:

$$M^{-1} = L^{-T}\,D^{-1}\,L^{-1} \tag{26}$$

Getting the numerical expression of $M^{-1}$ can be achieved using for instance a forward substitution on the columns of the identity matrix.

However, we found out that this approach is not the most efficient way to compute the inverse of the joint space inertia matrix. Indeed, it requires the computation of the joint space inertia matrix itself with its Cholesky decomposition, which are not required in the calculation of the partial derivatives of forward dynamics, as shown in Sec. IV-B.

To overcome these limitations, we have developed a dedicated algorithm to efficiently compute $M^{-1}$ by exploiting the sparsity induced by the kinematic tree, and without requiring the computation of the $M$ itself. This algorithm is a rewriting of ABA where we have omitted the affine terms like Coriolis and gravity effects that are normally evaluated by ABA. We also exploit the fact that $M^{-1}$ is a symmetric matrix, which means that it is sufficient to compute and store its upper or lower triangular part. Due to the space limitation, we provide all the details of this three-pass algorithm in the companion report [3]. And without investigating precisely its operational cost, we have found that computing $M^{-1}$ with this algorithm can be in practice up to twice faster than the standard approach for robots with numerous degrees of freedom, as shown in Sec. V.

## V. Results

In this section, we report the performances of our analytical derivatives compared to the finite differences approach. We do these benchmarks for various robots: the 7-dof robotic arm KUKA-LWR, the 18-dof quadruped robot HyQ and the 36-dof humanoid robot ATLAS.

All our derivatives have been implemented in C++ and we use the popular Eigen library [14] (version 3.3.4) for linear algebra computations. All the benchmarks have been performed on a 2.2 GHz quad-core Intel Core i7 processor using LLVM 9.0.0 as C++ compiler. We have done the computations on a single core of the CPU and we have disabled the turbo-boost option in order to obtain consistent timing measurements all along the benchmark process. While it may be possible to parallelize finite differences, most of the current robots do have enough computational ressources to do it. We then decided to implement them on a single core.

### A. Benchmark on the partial derivatives

For each robot, we randomly sample generalized configuration, velocity and acceleration vectors. And we measure over $10^5$ samples the mean time spent for the basic algorithms themselves (RNEA or ABA) as well as the mean times required to evaluate their derivatives analytically and using finite differences.

Tab. I and Fig. 2 collects all these computation times, that we comment in what follows.

*1) Benchmark of inverse dynamics derivatives:* From the three first rows of Tab. I, it is interesting to notice that running the analytical derivatives of inverse dynamics lasts at most 3 times the time spent in computations by the inverse dynamics itself. And we have a ratio from 7 to 26 with respect to finite differences.

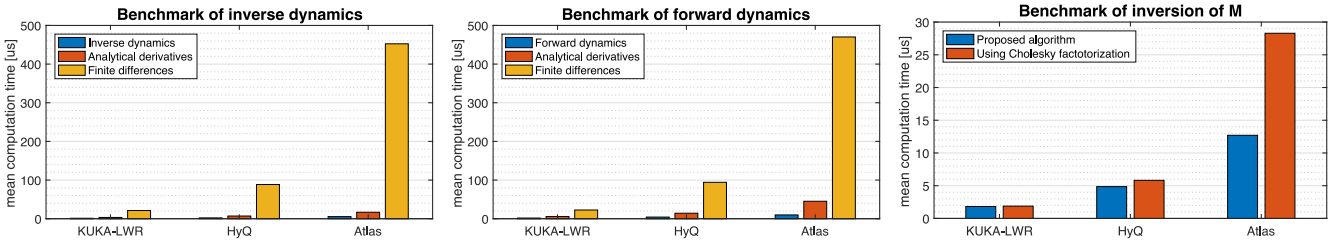
*2) Benchmark of forward dynamics derivatives:* Similarly to inverse dynamics, performing analytical derivatives is at much $4$ times slower than a call to the forward dynamics function. This time, the ratio between analytic derivation and finite differences goes from $4$ up to $10$. This difference of performances between analytical derivatives of forward and inverse dynamics are mainly due to the additional cost of computing the $M^{-1}$.

From Fig. 2, we can observe that the computational complexity of finite differences grows like 2 times the number of degrees of freedom, which is expected by the method. Indeed, we are only evaluating the finite differences with respect to the joint configuration and velocity vectors, as the last partial derivative is analytic.

Giftthaler et al. [13] have also reported some computation times for the HyQ robot, using automatic differencing and code generation. We obtain similar timings for the inverse dynamics ($5.06$ us against $6.92$ us[2]). But our analytical derivatives of forward dynamics shows better performances ($8.72$ us against $20.52$ us), again without using code generation from our side. This difference in performances is certainly due to the computational cunning on the derivatives of function composition that we have highlighted in Sec. IV.

*3) Benchmark on the joint space inertia matrix inversion:* From Fig. 2c, it appears that our algorithm to directly compute the inverse of $M$ outperforms the standard Cholesky decomposition approach for systems having a relatively high

[2]both obtained without modeling the free-flyers, following benchmarks of [13]




(a) Benchmark of inverse dynamics (b) Benchmark of forward dynamics (c) Benchmark of joint space inertia matrix inversion

Fig. 2: Comparison of mean computation times for derivatives of forward and inverse dynamics.

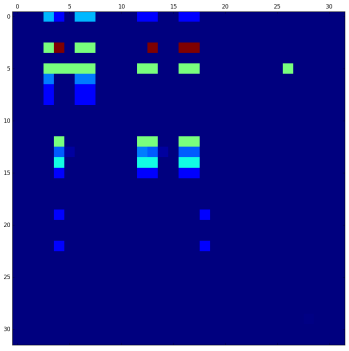


Fig. 3: Sparsity errors while computing the finite differences.

number of dof and remains competitive for robots having few dof.

### B. Numerical precision

It is well-known that methods using finite-differences are subject to numerical rounding errors. And this phenomena is amplified when the function to differentiate involves highly non-linear functions like $\cos, \sin$ and $\exp$ for instance. In what follows, we want to illustrate this aspect and to show how analytical derivatives in the context of inverse and forward dynamics have a larger numerical accuracy than the finite differences method.

For that purpose, we evaluate the partial derivatives of both forward and inverse dynamics at a given random configuration $\boldsymbol{q}_0$, with $\dot{\boldsymbol{q}}_0 = \ddot{\boldsymbol{q}}_0 = \mathbf{0}$ and we set the gravity vector to be also equal to zero, while omitting external forces contributions. In this precise case, the outputs of Eqs. (9) and (15) vanish, and the partial derivatives of ID and FD with respect to $\dot{\boldsymbol{q}}$ too. Yet, as finite differences add a small increment in the input vectors in order to evaluate derivatives, it appears that the result is not uniformly equal to zero as depicted by Fig. 3.

### C. Source code implementation

All the aforementioned analytical derivatives have been implemented in our rigid-body dynamics frameworks called Pinocchio [5]. Pinocchio implements fast forward and inverse dynamics algorithms and their analytical derivatives, for poly-articulated systems doted with a free-floating base or not. It also provides Python bindings for efficient code prototyping. Pinocchio is now at the hearth of the planing and control algorithms [21, 4] of the Gepetto team at LAAS-CNRS.

## VI. Conclusion

The paper proposes the first generalization of the efficient rigid-body-dynamics algorithms to compute their derivatives. Our approach leads to a very efficient algorithm, easy to implement, and able to compute the derivatives of the inverse and direct dynamics. The complexity is linear in the number of bodies. Computing the derivative of these two functions has an algorithm cost about 3-4 times larger than evaluating the function itself. As a side contribution, we also proposed an original algorithm to compute the inverse of the mass matrix.

All this theoretical work comes with practical implementation: we provide a complete open-source implementation in C++ implementing all these algorithms and that can be run using a URDF model of the robot. We used it to benchmark the proposed algorithms on several robot models. Obviously, by keeping a linear complexity, we are much faster than finite differences (about 40 times faster on a humanoid robot). We have also shown that analytical derivatives is important to properly capture the sparsity of the resulting matrices, that finite differences fail to properly achieve. We used the same benchmark (HyQ model) on a similar CPU than the best implementation proposed yet [13]. While we do not need to rely on code-generation software, our algorithm is 30% faster for the inverse dynamics (5 us versus 7 us) and 60% faster for the direct dynamics (8 us versus 20 us).

The capability to write simple and super-efficient algorithms [8] to compute inverse and direct dynamics has an important impact in enabling roboticists to develop complex model-based methods in many aspects of our domain. We similarly believe that the extension of these algorithms, with similar complexity (in implementation and cost), will have a similar impact. Optimal control and model-predictive control (MPC) rely on gradient computations of the robot dynamics to iteratively improve the robot future trajectory [30]. The most efficient MPC solvers are yet implemented using finite differences [29, 23]. To prevent the outrageous cost, parallelization has to be enforced for computing the derivatives, which led to the use of high-performance computers, often in the cloud, when implemented in a real-time set up [17]. Based on the

benchmarks that we reported in the paper, it seems feasible to implement whole-body MPC for a full humanoid with computation frequency of 100Hz.

Similarly to control, optimal estimation (e.g. maximum likelihood) is often written as an optimization problem where the derivatives of the dynamics are important [1, 2, 26]. Differentiating the dynamics is also important in co-design, where the mechanical design of the robot is optimized (once more using gradient-based iterations) [28, 15]. We have yet proposed the derivatives of the dynamics with respect to the robot state and control variables. The proposed method directly extends to the derivatives with respect to the model parameters (masses, lengths, etc). Finally, the derivatives give also important information about the variability of the robot behavior and might be useful, if available for cheap, in reinforcement learning and deep policy optimization [22].

## ACKNOWLEDGMENTS

This work is supported by the RoboCom++ FLAG-ERA JTC 2016 proposal.

## REFERENCES

[1] M. Benallegue and F. Lamiraux. Humanoid flexibility deformation can be efficiently estimated using only inertial measurement units and contact information. In *IEEE International Conference on Humanoid Robots (Humanoids)*, Madrid, Spain, 2014.

[2] Michael Bloesch, Marco Hutter, Mark Hoepflinger, Stefan Leutenegger, Christian Gehring, C. David Remy, and Roland Siegwart. State estimation for legged robots - consistent fusion of leg kinematics and IMU. In *Robotics: Science and Systems*, Sydney, Australia, 2012.

[3] Justin Carpentier. Analytical inverse of the joint space inertia matrix. Technical report, Laboratoire d'Analyse et d'Architecture des Systèmes, 2018.

[4] Justin Carpentier and Nicolas Mansard. Multi-contact locomotion of legged robots. *Submitted to IEEE Transaction on Robotics*, 2018.

[5] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. Pinocchio: fast forward and inverse dynamics for poly-articulated systems, 2015–2018. URL https://stack-of-tasks.github.io/pinocchio.

[6] Erwin Coumans. Bullet Physics Simulation. In *ACM SIGGRAPH 2015 Courses*, 2015.

[7] Roy Featherstone. The calculation of robot dynamics using articulated-body inertias. *The International Journal of Robotics Research*, 1983.

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

[9] Roy Featherstone. Quantitative measures of a robots physical ability to balance. *The International Journal of Robotics Research*, 2016.

[10] Martin L Felis. RBDL: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 2017.

[11] Michele Focchi, Andrea Del Prete, Ioannis Havoutis, Roy Featherstone, Darwin G Caldwell, and Claudio Semini. High-slope terrain locomotion for torque-controlled quadruped robots. *Autonomous Robots*, 2017.

[12] Gianluca Garofalo, Christian Ott, and Alin Albu-Schaffer. On the closed form computation of the dynamic matrices and their differentiations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[13] Markus Giftthaler, Michael Neunert, Markus Stäuble, Marco Frigerio, Claudio Semini, and Jonas Buchli. Automatic differentiation of rigid body dynamics for optimal control and estimation. *Advanced Robotics*, 2017.

[14] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010. URL http://eigen.tuxfamily.org.

[15] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint Optimization of Robot Design and Motion Parameters using the Implicit Function Theorem. In *Robotics: Science and Systems*, Cambridge, Massachusetts, 2017.

[16] Alexander Herzog, Nicholas Rotella, Sean Mason, Felix Grimminger, Stefan Schaal, and Ludovic Righetti. Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, 2016.

[17] Jonas Koenemann, Andrea Del Prete, Yuval Tassa, Emanuel Todorov, Olivier Stasse, Maren Bennewitz, and Nicolas Mansard. Whole-body model-predictive control applied to the HRP-2 humanoid. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[18] Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[19] Sung-Hee Lee, Junggon Kim, Frank Chongwoo Park, Munsang Kim, and James E Bobrow. Newton-type algorithms for dynamics-based robot movement optimization. *IEEE Transactions on robotics*, 2005.

[20] J Luh, M Walker, and R Paul. Resolved-acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Control*, 1980.

[21] Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylene Campana, Nicolas Mansard, and Florent Lamiraux. HPP: A new software for constrained motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[22] Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, Berkeley, USA, July 2014.

[23] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on*

*Graphics (TOG)*, 2012.

[24] Richard M Murray, Zexiang Li, S Shankar Sastry, and S Shankara Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[25] Maximilien Naveau, Justin Carpentier, Sébastien Barthelemy, Olivier Stasse, and Philippe Souères. METAPOD — Template META-PrOgramming applied to dynamics: CoP-CoM trajectories filtering. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2014.

[26] Simona Nobili, Marco Camurri, Victor Barasuol, Michele Focchi, Darwin Caldwell, Claudio Semini, and Maurice Fallon. Heterogeneous sensor fusion for accurate state estimation of dynamic legged robots. In *Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[27] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 2014.

[28] Guilhem Saurel, Justin Carpentier, Nicolas Mansard, and Jean-Paul Laumond. A simulation framework for simultaneous design and control of passivity based walkers. In *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2016.

[29] Gerrit Schultz and Katja Mombaur. Modeling and optimal control of human-like running. *IEEE/ASME Transactions on mechatronics*, 2010.

[30] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[31] Russ Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016. URL http://drake.mit.edu.

[32] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[33] Michael W Walker and David E Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 1982.

[34] Pierre-Brice Wieber, Florence Billet, Laurence Boissieux, and Roger Pissard-Gibollet. The HuMAnS toolbox, a homogenous framework for motion capture, analysis and simulation. In *International Symposium on the 3D Analysis of Human Movement*, 2006.