2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

A Modified Newton-Euler Method for Dynamic Computations in Robot Fault Detection and Control

Alessandro De Luca Lorenzo Ferrajoli
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Ariosto 25, 00185 Roma, Italy
deluca@dis.uniroma1.it lorenzo.ferrajoli@tiscali.it

***Abstract*— We present a modified recursive Newton-Euler method for computing some dynamic expressions that arise in two problems of fault detection and control of serial robot manipulators, and which cannot be evaluated numerically using the standard method. The two motivating problems are: *i)* the computation of the residual vector that allows accurate detection of actuator faults or unexpected collisions using only robot proprioceptive measurements, and *ii)* the evaluation of a passivity-based trajectory tracking control law. The modified Newton-Euler algorithm generates factorization matrices of the Coriolis and centrifugal terms that satisfy the skew-symmetric property. The computational advantages with respect to numerical evaluation of symbolically obtained dynamic expressions is illustrated on a $7R$ DLR lightweight manipulator.**

I. Introduction

Computational issues are a major aspect in the implementation of model-based robot control laws. In order to meet real-time constraints, two standard options are usually considered for serial manipulators: customized Lagrangian methods [1], [2] and recursive Newton-Euler methods [3].

The first approach requires the derivation of the robot dynamics in closed symbolic form based on kinetic and potential energy computation, followed by a conversion into a real-time code so as to optimize the number of elementary operations to be evaluated numerically [4], [5]. The conversion can be performed with automatic tools and/or involves customization to the specific robot manipulator under consideration [6]. The second approach is more systematic and uses the general (forward) recursive expressions of the robot differential kinematics and those of the (backward) force/torque balance equations for each link. While these two approaches have been refined over the years, with a variety of algorithmic implementations that share a linear complexity $O(N)$ in terms of the number N of joints (differing by the leading coefficient) [3], [7], [8], Newton-Euler methods are generally recognized to be superior when N becomes large.

In this paper, we consider a modified Newton-Euler method that solves two motivating problems in robot control and fault detection, which escape the use of the standard method for computing the dynamic terms involved.

The first problem arises in the implementation of an efficient approach to the *sensorless detection* of generic faults acting on the robot, e.g., actuator faults [9], unexpected collisions [10], [11], or unmodeled friction [12]. In these works, a so-called *residual* vector signal is computed based on the variation of the generalized momentum, and then used to detect on line the occurring fault. In order to avoid numerical differentiation of the inertia matrix $\boldsymbol{M}(\boldsymbol{q})$ [10] or partial derivatives of its elements [13], the expression of the residual requires the evaluation of a quadratic velocity term of the form $\boldsymbol{C}^T(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ (note the transpose), where $\boldsymbol{C}$ is a matrix factorizing the Coriolis and centrifugal terms and satisfying the skew-symmetric property for $\dot{\boldsymbol{M}} - 2\boldsymbol{C}$. This special velocity term cannot be evaluated numerically through the standard Newton-Euler method.

The second problem arises in *passivity-based control* design, both in nominal conditions [14] and in the adaptive case [15]. In particular, a dynamic term $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_r$ appears in these control laws, with $\dot{\boldsymbol{q}}_r$ being a modified reference velocity. Again, this term cannot be computed numerically through the standard Newton-Euler method.

We present here a modification of the recursive Newton-Euler algorithm for robot inverse dynamics that takes as input *four* vector quantities, typically a position $\boldsymbol{q}$, two possibly different velocities $\dot{\boldsymbol{q}}$ and $\dot{\boldsymbol{q}}_a$ (this is the novelty), and an acceleration $\ddot{\boldsymbol{q}}$, apart from a standard scalar parameter α which specifies the presence or not of gravity. Using this modified algorithm, we are able to solve numerically the above mentioned passivity-based control problem in one algorithmic sweep, with complexity $O(N)$, and the sensorless detection problem in N of such elementary sweeps.

It is worth to point out that some of the presented developments may be used also for computing the inverse dynamics of robots with elastic joints, where higher order derivatives of the link variables appear [16].

The paper is organized as follows. In Sect. II, we introduce and formalize the two motivating problems. In Sect. III, we recall the standard Newton-Euler method and its use, and set up the recursive notation. The new method is presented in Sect. IV and its properties are analyzed. A comparative numerical test on a $7R$ lightweight robot by DLR [17] is reported in Sect. V, where the numerical method is evaluated against a Lagrangian approach in terms of computational times. On-going work on open issues and possible extensions are discussed in the concluding section.

II. MOTIVATING PROBLEMS

Consider a robot manipulator with an open kinematic chain of N moving rigid bodies, and let $\boldsymbol{q} \in \mathbb{R}^N$ be the vector of generalized coordinates. The dynamic model is

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{u}, \tag{1}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is the symmetric, positive definite robot inertia matrix, $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the Coriolis and centrifugal vector (with quadratic dependence on $\dot{\boldsymbol{q}}$), $\boldsymbol{g}(\boldsymbol{q})$ is the gravity vector, and $\boldsymbol{u}$ is the vector of external generalized forces performing work on $\boldsymbol{q}$. For simplicity, in the following we will neglect friction and other dissipative terms.

The Coriolis and centrifugal term in (1) can be factorized as

$$\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}, \tag{2}$$

where $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is any $N \times N$ matrix such that the following skew-symmetric property holds:

$$\boldsymbol{x}^T \left(\dot{\boldsymbol{M}}(\boldsymbol{q}) - 2\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \right) \boldsymbol{x} = 0, \quad \forall \boldsymbol{x} \in \mathbb{R}^N. \tag{3}$$

Property (3) is equivalent to the existence of the following expression for the time derivative of the inertia matrix:

$$\dot{\boldsymbol{M}}(\boldsymbol{q}) = \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{C}^T(\boldsymbol{q}, \dot{\boldsymbol{q}}). \tag{4}$$

Typically, the factorization matrix $\boldsymbol{C}$ is built from Christoffel symbols of the second kind [18]. Note however that the factorization matrix in (2) obeying to (3) is not uniquely defined.

A. Sensorless fault detection

In [10], [12], [19]. the presence of a fault acting on the robot dynamics has been modeled by expressing the vector of external generalized forces $\boldsymbol{u}$ in (1) as

$$\boldsymbol{u} = \boldsymbol{u}_c + \boldsymbol{u}_f, \tag{5}$$

where $\boldsymbol{u}_c$ is the commanded (control) torques and $\boldsymbol{u}_f$ represents unexpected and/or unmodeled (and unmeasurable) fault actions on the robot that may be occasional or persistent. Examples in this class include actuator faults, collisions with the environment at any robot location along the structure, or friction at the joints/transmissions.

In order to detect the occurrence of one of such potential faults using only proprioceptive robot measurements (i.e., $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$) and the available commands $\boldsymbol{u}_c$, the following N-dimensional residual vector can be computed

$$\boldsymbol{r}(t) = \boldsymbol{K}_I \left[\boldsymbol{p}(t) - \int_0^t \left(\boldsymbol{u}_c + \boldsymbol{C}^T(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{r} \right) ds \right], \tag{6}$$

where $\boldsymbol{K}_I > \boldsymbol{0}$ is a diagonal matrix and

$$\boldsymbol{p} = \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{7}$$

is the robot generalized momentum. Assuming the robot at rest at time $t = 0$, we set $\boldsymbol{r}(0) = \boldsymbol{0}$. Using eqs. (1), (4–5), and (7), it has been shown that the time evolution of the residual vector $\boldsymbol{r}$ is given by

$$\dot{\boldsymbol{r}} = -\boldsymbol{K}_I \boldsymbol{r} + \boldsymbol{K}_I \boldsymbol{u}_f.$$

Therefore, $\boldsymbol{r}$ is 'observing' (a filtered and component-wise decoupled version of) the generalized force $\boldsymbol{u}_f$ resulting from the fault. When this residual grows over a threshold, detection is triggered. Vector $\boldsymbol{r}$ can be further used, respectively, to identify the type of actuator fault [20], to let the robot react safely to an unexpected collision [10], or to compensate unmodeled friction in the motion control law [12].

B. Passivity-based control

Beside the most effective feedback linearization method for the accurate tracking of a desired differentiable trajectory $\boldsymbol{q}_d(t) \in C^2$, a very popular nonlinear control approach is based on the passivity characteristics of the robot dynamics [15], [21], related to the skew-symmetric property (4). The main advantages of passivity-based control are: *i)* a better use of the available actuator torques since complete cancellation of the dynamic terms is avoided; *ii)* a more robust behavior in the presence of uncertainties; and, *iii)* the possibility of designing adaptive versions to cope with unknown dynamic parameters. The classical expression of a passivity-based trajectory tracking control law for (1) is

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}_r + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_r + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{K}_P \boldsymbol{e} + \boldsymbol{K}_D \dot{\boldsymbol{e}}, \tag{8}$$

where $\boldsymbol{e} = \boldsymbol{q}_d - \boldsymbol{q}$ is the trajectory error, $\boldsymbol{K}_P$ and $\boldsymbol{K}_D$ are positive definite (diagonal) gain matrices, $\dot{\boldsymbol{q}}_r$ is the modified reference velocity

$$\dot{\boldsymbol{q}}_r = \dot{\boldsymbol{q}}_d + \boldsymbol{\Lambda}\boldsymbol{e}, \quad \boldsymbol{\Lambda} > \boldsymbol{0},$$

and the factorization matrix $\boldsymbol{C}$ in (8) should satisfy eq. (3).

Remark. Apart from other model terms that appear also in standard computations of robot inverse dynamics, the critical aspect of eq. (6) and eq. (8) stands in the need, respectively, of $\boldsymbol{C}^T(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$, with the transpose of the factorization matrix $\boldsymbol{C}$, and of $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_r$, with different inner and outer velocities. Indeed, a symbolic computation based on the Lagrange equations and Christoffel symbols allows to evaluate the above terms as well. On the other hand, it is apparent that these two terms cannot be numerically computed by the standard (recursive) Newton-Euler method, as highlighted in the next section. This motivated our further developments for an efficient real-time computation of eq. (6) or eq. (8), especially for large N.

III. STANDARD NEWTON-EULER METHOD

We briefly recall here the standard recursive Newton-Euler (N-E) method of [3], also in order to set up the notation for the modified one. For compactness, in this and the following section we will consider only the case of open kinematic chains with rotational joints. Moreover, no special attention is paid at this stage to the most efficient implementation of the N-E algorithm.

The N-E algorithm can be described succinctly as a vector function $\mathbf{NE}_\alpha$ of *three* ordered vector inputs $\boldsymbol{a}_i \in \mathbb{R}^N$, $i = 1, 2, 3$, parametrized by a scalar $\alpha \in \{g_0, 0\}$ accounting for

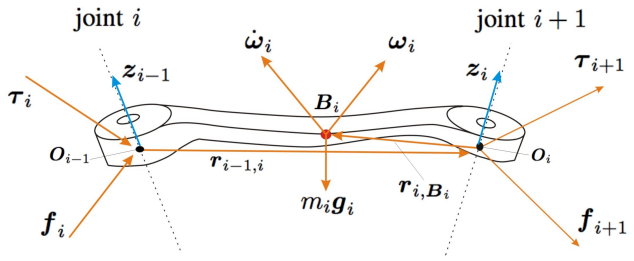

Fig. 1. Definitions related to link i

presence or absence of the acceleration $g_0 = 9.81$ [m/s^2] due to gravity:

$$\mathbf{NE}_\alpha(\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3). \tag{9}$$

For instance, the inverse dynamics computation associated to a desired differentiable motion $\boldsymbol{q}_d(t)$ in the gravity field is expressed as

$$\boldsymbol{u}_d(t) = \mathbf{NE}_{g_0}(\boldsymbol{q}_d(t), \dot{\boldsymbol{q}}_d(t), \ddot{\boldsymbol{q}}_d(t)), \tag{10}$$

providing as output the desired joint torque $\boldsymbol{u}_d(t)$ at time t.

With reference to the generic link i, $i = 1, \ldots, N$, shown in Fig. 1, let $\boldsymbol{z}_{i-1}$ be the unitary vector along the axis of joint i (as in the standard Denavit-Hartenberg convention [18]), $\boldsymbol{r}_{i-1,i} = \boldsymbol{O}_i - \boldsymbol{O}_{i-1}$ and $\boldsymbol{r}_{i,\boldsymbol{B}_i} = \boldsymbol{B}_i - \boldsymbol{O}_i$, where $\boldsymbol{O}_i$ and $\boldsymbol{B}_i$ are respectively the origin of the i-th reference frame and the center of mass of link i. Furthermore, let $\boldsymbol{f}_i$ and $\boldsymbol{\tau}_i$ be the forces and torques transmitted from link $i-1$ to link i. Finally, let m_i and $\boldsymbol{I}_i$ be the mass and baricentral inertia tensor of link i, and $\boldsymbol{g}_i$ the gravity vector acting on its center of mass. Note that all quantities are expressed in the reference frame attached to the current link (*moving frames*), so that, e.g., $\boldsymbol{I}_i$ will be a constant.

The *forward recursion* on the angular velocity and acceleration, and on the linear acceleration of the frame origin and center of mass of each link is given by:

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} + \dot{q}_i \boldsymbol{z}_{i-1} \tag{\textbf{AVR}}$$

$$\dot{\boldsymbol{\omega}}_i = \dot{\boldsymbol{\omega}}_{i-1} + \ddot{q}_i \boldsymbol{z}_{i-1} + \dot{q}_i \boldsymbol{\omega}_{i-1} \times \boldsymbol{z}_{i-1} \tag{\textbf{AAR}}$$

$$\ddot{\boldsymbol{O}}_i = \ddot{\boldsymbol{O}}_{i-1} + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i-1,i} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i-1,i}) \tag{\textbf{LAR'}}$$

$$\ddot{\boldsymbol{B}}_i = \ddot{\boldsymbol{O}}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i,\boldsymbol{B}_i} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i,\boldsymbol{B}_i}), \tag{\textbf{LAR}}$$

for $i = 1, \ldots, N$, with the initialization $\boldsymbol{\omega}_0 = \dot{\boldsymbol{\omega}}_0 = \boldsymbol{0}$ and $\ddot{\boldsymbol{O}}_0 = -\alpha \boldsymbol{g}_0 / \|\boldsymbol{g}_0\|$ (fixed base, including gravity acceleration when present).

The *backward recursion* on dynamic forces and torques is given by:

$$\boldsymbol{f}_i = \boldsymbol{f}_{i+1} + m_i \ddot{\boldsymbol{B}}_i \tag{\textbf{FR}}$$

$$\begin{aligned} \boldsymbol{\tau}_i \;=\; & \boldsymbol{\tau}_{i+1} - \boldsymbol{f}_i \times (\boldsymbol{r}_{i-1,i} + \boldsymbol{r}_{i,\boldsymbol{B}_i}) \\ & + \boldsymbol{f}_{i+1} \times \boldsymbol{r}_{i,\boldsymbol{B}_i} + \boldsymbol{I}_i \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times (\boldsymbol{I}_i \boldsymbol{\omega}_i) \end{aligned} \tag{\textbf{TR}}$$

$$u_i = \boldsymbol{\tau}_i^T \boldsymbol{z}_{i-1}, \tag{\textbf{FP}}$$

for $i = N, \ldots, 1$, with the initialization $\boldsymbol{f}_{N+1} = \boldsymbol{\tau}_{N+1} = \boldsymbol{0}$ (no contact with the environment at the end-effector level).

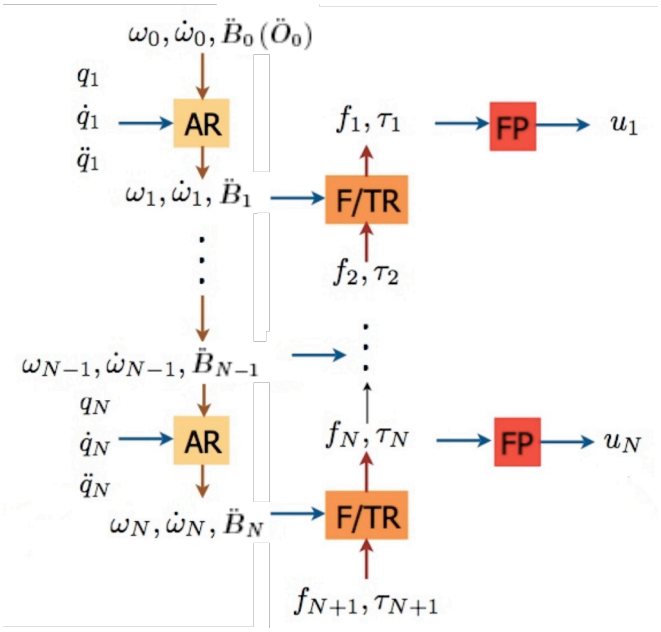

Fig. 2. Standard recursive Newton-Euler algorithm (modified from [18])

The overall computational scheme is summarized in Fig. 2, where the "**AR**" blocks include eqs. (**AVR**), (**AAR**), (**LAR'**), and (**LAR**).

A. *Use of the* NE *algorithm*

One sweep of the standard N-E algorithm can be also used to compute separately the various dynamic terms in (1). For instance, the gravity vector is obtained as

$$\boldsymbol{g}(\boldsymbol{q}) = \mathbf{NE}_{g_0}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0}) \tag{11}$$

while the i-th column of the inertia matrix

$$\boldsymbol{M}_i(\boldsymbol{q}) = \mathbf{NE}_0(\boldsymbol{q}, \boldsymbol{0}, \hat{\boldsymbol{v}}_i), \tag{12}$$

where $\hat{\boldsymbol{v}}_i = \begin{pmatrix} 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \end{pmatrix}^T$ is the i-th vector of the canonical base of $\mathbb{R}^N$. Moreover, the generalized momentum in eq. (7) is given by

$$\boldsymbol{p} = \mathbf{NE}_0(\boldsymbol{q}, \boldsymbol{0}, \dot{\boldsymbol{q}}), \tag{13}$$

while the well-known feedback linearization control law is obtained from

$$\boldsymbol{u}_{FL} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{a} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \mathbf{NE}_{g_0}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{a}), \tag{14}$$

where $\boldsymbol{a} = \ddot{\boldsymbol{q}}_d + \boldsymbol{K}_P \boldsymbol{e} + \boldsymbol{K}_D \dot{\boldsymbol{e}}$. However, from

$$\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathbf{NE}_0(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{0}) \tag{15}$$

one can neither extract the matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ to be transposed in eq. (6) —and also not its single elements or columns— nor compute the mixed velocity term $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_r$ needed in eq. (8).

IV. Modified Newton-Euler method

The modified N-E algorithm can be described succinctly as a vector function $\mathbf{NE}^*_\alpha$ of *four* vector inputs $\boldsymbol{a}_i \in \mathbb{R}^N$, $i = 1, 2, 3, 4$, still parametrized by the same scalar $\alpha \in \{g_0, 0\}$:

$$\mathbf{NE}^*_\alpha(\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4). \tag{16}$$

The basic idea is to allow more freedom by splitting the quadratic appearance of the velocity vector $\boldsymbol{a}_2 = \dot{\boldsymbol{q}}$ within (9) in two distinct inputs, $\boldsymbol{a}_2 = \dot{\boldsymbol{q}}$ and an auxiliary $\boldsymbol{a}_3 = \dot{\boldsymbol{q}}_a$. with the additional trivial condition that, for $\dot{\boldsymbol{q}}_a = \dot{\boldsymbol{q}}$, it is

$$\mathbf{NE}^*_\alpha(\boldsymbol{q}, \dot{\boldsymbol{q}}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \mathbf{NE}_\alpha(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}). \tag{17}$$

Moreover, for a correct computation of eqs. (6) and (8), we need to enforce the skew-symmetric property (3). In particular, the algorithm $\mathbf{NE}^*$, when invoked with $\dot{\boldsymbol{q}}_a = \hat{\boldsymbol{v}}_i$ and $\alpha = 0$, should generate

$$\boldsymbol{C}^*_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathbf{NE}^*_0(\boldsymbol{q}, \dot{\boldsymbol{q}}, \hat{\boldsymbol{v}}_i, \boldsymbol{0}), \tag{18}$$

namely the i-th column of a matrix

$$\boldsymbol{C}^*(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{pmatrix} \boldsymbol{C}^*_1 & \boldsymbol{C}^*_2 & \dots & \boldsymbol{C}^*_N \end{pmatrix} \tag{19}$$

that has to satisfy

$$\dot{\boldsymbol{M}}(\boldsymbol{q}) - \boldsymbol{C}^*(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{C}^{*T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{0}, \quad \forall \boldsymbol{q}, \dot{\boldsymbol{q}}. \tag{20}$$

With this in mind, we introduce an auxiliary angular velocity $\boldsymbol{\omega}_{a_i}$ of link i, which is dependent on $\dot{q}_{a_1}, \dots, \dot{q}_{a_i}$ through the recursive relation

$$\boldsymbol{\omega}_{a_i} = \boldsymbol{\omega}_{a_{i-1}} + \dot{q}_{a_i} \boldsymbol{z}_{i-1}. \tag{**AVR***}$$

Moreover, the following terms in the standard N-E recursions with quadratic dependence on velocity will be modified:

1) the third term on the right-hand side of (**AAR**)
$$\dot{q}_i \boldsymbol{\omega}_{i-1} \times \boldsymbol{z}_{i-1};$$
2) the third term on the right-hand side of (**LAR'**)
$$\boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i-1,i});$$
3) the third term on the right-hand side of (**LAR**)
$$\boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i,\boldsymbol{B}_i});$$
4) the fifth term on the right-hand side of (**TR**)
$$\boldsymbol{\omega}_i \times (\boldsymbol{I}_i \boldsymbol{\omega}_i).$$

In the first one we have $\dot{q}_i$ multiplying $\boldsymbol{\omega}_{i-1}$, which is function of $\dot{q}_j$ ($j = 1, ..., i-1$), whereas in the remaining ones we have vector products. In these vector products, one factor will be made dependent on $\dot{\boldsymbol{q}}$ while the other on $\dot{\boldsymbol{q}}_a$. Note that in all the above four cases, the result will change depending on the order of substitutions. In fact, in the modified eq. (**AAR**)

$$\dot{q}_i \boldsymbol{\omega}_{a_{i-1}}(\dot{q}_{a_1}, \dots, \dot{q}_{a_{i-1}}) \neq \dot{q}_{a_i} \boldsymbol{\omega}_{i-1}(\dot{q}_1, \dots, \dot{q}_{i-1}), \tag{21}$$

while in the modified eqs. (**LAR'**), (**LAR**) and (**TR**) we will have a sign inversion due to anti-symmetric property of the vector product ($\boldsymbol{v} \times \boldsymbol{w} = -(\boldsymbol{w} \times \boldsymbol{v})$).

As for the angular acceleration recursion, we may substitute eq. (**AAR**) with one of the two following options, indexed by 0 and 1:

$$\dot{\boldsymbol{\omega}}_i = \dot{\boldsymbol{\omega}}_{i-1} + \ddot{q}_i \boldsymbol{z}_{i-1} + \dot{q}_i \boldsymbol{\omega}_{a_{i-1}} \times \boldsymbol{z}_{i-1} \tag{**AAR**$^*_0$}$$

or

$$\dot{\boldsymbol{\omega}}_i = \dot{\boldsymbol{\omega}}_{i-1} + \ddot{q}_i \boldsymbol{z}_{i-1} + \dot{q}_{a_i} \boldsymbol{\omega}_{i-1} \times \boldsymbol{z}_{i-1}. \tag{**AAR**$^*_1$}$$

Similarly, one can define the following three couples of new recursions:

$$\ddot{\boldsymbol{O}}_i = \ddot{\boldsymbol{O}}_{i-1} + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i-1,i} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_{a_i} \times \boldsymbol{r}_{i-1,i}) \tag{**LAR'**$^*_0$}$$

$$\ddot{\boldsymbol{O}}_i = \ddot{\boldsymbol{O}}_{i-1} + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i-1,i} + \boldsymbol{\omega}_{a_i} \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i-1,i}) \tag{**LAR'**$^*_1$}$$

$$\ddot{\boldsymbol{B}}_i = \ddot{\boldsymbol{O}}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i,\boldsymbol{B}_i} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_{a_i} \times \boldsymbol{r}_{i,\boldsymbol{B}_i}) \tag{**LAR**$^*_0$}$$

$$\ddot{\boldsymbol{B}}_i = \ddot{\boldsymbol{O}}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{r}_{i,\boldsymbol{B}_i} + \boldsymbol{\omega}_{a_i} \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_{i,\boldsymbol{B}_i}) \tag{**LAR**$^*_1$}$$

$$\boldsymbol{\tau}_i = \boldsymbol{\tau}_{i+1} - \boldsymbol{f}_i \times (\boldsymbol{r}_{i-1,i} + \boldsymbol{r}_{i,\boldsymbol{B}_i}) + \boldsymbol{f}_{i+1} \times \boldsymbol{r}_{i,\boldsymbol{B}_i} + \boldsymbol{I}_i \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times (\boldsymbol{I}_i \boldsymbol{\omega}_{a_i}) \tag{**TR**$^*_0$}$$

$$\boldsymbol{\tau}_i = \boldsymbol{\tau}_{i+1} - \boldsymbol{f}_i \times (\boldsymbol{r}_{i-1,i} + \boldsymbol{r}_{i,\boldsymbol{B}_i}) + \boldsymbol{f}_{i+1} \times \boldsymbol{r}_{i,\boldsymbol{B}_i} + \boldsymbol{I}_i \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_{a_i} \times (\boldsymbol{I}_i \boldsymbol{\omega}_i) \tag{**TR**$^*_1$}$$

Accordingly, an indexed family of recursive algorithms $\mathbf{NE}^*_{\alpha, i_1 \dots i_4}$ can be formulated through the following sequences of steps

1) **AVR**
2) **AVR***
3) **AAR**$^*_{i_1}$
4) **LAR'**$^*_{i_2}$
5) **LAR**$^*_{i_3}$
6) **FR**
7) **TR**$^*_{i_4}$
8) **FP**,

with $i_j \in \{0, 1\}$ for $j = 1, \dots, 4$. Each computational scheme will be similar to the one in Fig. 2, with the same $O(N)$ complexity.

As a consequence, $2^4 = 16$ different versions of the algorithm are possible and it needs to be checked whether none, one, many, or all of them will lead to a factorization matrix $\boldsymbol{C}^*(\boldsymbol{q}, \dot{\boldsymbol{q}})$, as defined by eqs. (18– 19), that obeys to the skew-simmetric property (20). For this, one can proceed element-wise, verifying in symbolic form if

$$\begin{aligned} &\frac{\partial M_{ij}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} - C^*_{ij} - C^*_{ji} = 0, \\ &\text{for } j = 1, ..., N, \ i = 1, ..., N - j + 1, \end{aligned} \tag{22}$$

where

$$M_{ij}(\boldsymbol{q}) = \hat{\boldsymbol{v}}_i^T \mathbf{NE}^*_{\alpha, i_1 \dots i_4}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0}, \hat{\boldsymbol{v}}_j), \tag{23}$$

and

$$C^*_{ij}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \hat{\boldsymbol{v}}_i^T \mathbf{NE}^*_{\alpha, i_1 \dots i_4}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \hat{\boldsymbol{v}}_j, \boldsymbol{0}). \tag{24}$$

Symbolic computations have been performed in the MATLAB® environment, using the *Symbolic Math Toolbox*. The whole family of recursive algorithms has been checked

on the fully general dynamic model of open chain manipulators with up to $N = 7$ rotational joints. It has been found that only two cases satisfy in general the required conditions (22), namely $\mathbf{NE}^*_{\alpha,0001}$ and its dual $\mathbf{NE}^*_{\alpha,1110}$. Any of these two can be used as the final modified N-E algorithm $\mathbf{NE}^*_\alpha$.

We finally note that there where specific instances of manipulators in which the factorization matrix $\boldsymbol{C}^*$ computed with the new algorithm but in symbolic form was found to be equal to the matrix $\boldsymbol{C}$ computed through the Christoffel symbols. For other manipulators, this was not the case (despite the satisfaction of the skew-symmetric property). It can be concluded that the designed $\mathbf{NE}^*_\alpha$ algorithm does not cover all admissible factorizations of the Coriolis and centrifugal terms.

A. *Use of the* $\mathbf{NE}^*$ *algorithm*

The modified N-E algorithm can be immediately used for solving the two original problems that motivated this work. In particular, the passivity-based control law in eq. (8) is obtained by the single sweep

$$\boldsymbol{u} = \mathbf{NE}^*_{g_0}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \dot{\boldsymbol{q}}_r, \ddot{\boldsymbol{q}}_r) + \boldsymbol{K}_P \boldsymbol{e} + \boldsymbol{K}_D \dot{\boldsymbol{e}}, \qquad (25)$$

thus with $O(N)$ computational complexity, similar to the result obtained in [22]. On the other hand, the residual in eq. (6) needs $N + 2$ sweeps of the algorithm, namely two for computing the generalized momentum

$$\boldsymbol{p} = \mathbf{NE}^*_0(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0}, \dot{\boldsymbol{q}}) \qquad (26)$$

and the gravity vector

$$\boldsymbol{g}(\boldsymbol{q}) = \mathbf{NE}^*_{g_0}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0}), \qquad (27)$$

and N sweeps

$$\dot{\boldsymbol{q}}^T \boldsymbol{C}^*_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}^T \mathbf{NE}^*_0(\boldsymbol{q}, \dot{\boldsymbol{q}}, \hat{\boldsymbol{v}}_i, \boldsymbol{0}), \quad i = 1, \ldots, N, \quad (28)$$

for computing component-wise the vector $\boldsymbol{C}^{*T}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$. Its overall complexity is thus $O(N^2)$.

V. Numerical Test

As a selected test for comparing the efficiency of the modified N-E algorithm with respect to the evaluation of symbolically obtained dynamic terms, we have considered one of the series of $7R$ lightweight robots (LWR) developed at DLR [17]. For this robot, we have computed the matrix $\boldsymbol{C}^*(\boldsymbol{q}, \dot{\boldsymbol{q}})$ with the $\mathbf{NE}^*$ algorithm for a given argument $\boldsymbol{q}, \dot{\boldsymbol{q}}$, as well as evaluated the symbolic matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ obtained using Christoffel symbols. Indeed, since both $\boldsymbol{C}$ and $\boldsymbol{C}^*$ satisfy the skew-symmetric property, this comparison is consistent. The full dynamic model in closed form was previously obtained using a Lagrangian approach, with the kinematic and dynamic parameters taken from [23]. The conventional Denavit-Hartenberg kinematic parameters for this arm are reported in Table I. Uniform mass distribution has been assumed for each link, with diagonal and constant link inertia matrices (when expressed in the barycentric frame associated to each link). Motor inertias, as reflected through the square of the gear ratios, have been added on the diagonal of matrix $\boldsymbol{M}(\boldsymbol{q})$.

i	α_i	a_i	d_i	θ_i
1	$\frac{\pi}{2}$	0	L	q_1
2	$-\frac{\pi}{2}$	0	0	q_2
3	$\frac{\pi}{2}$	0	$2L$	q_3
4	$-\frac{\pi}{2}$	0	0	q_4
5	$-\frac{\pi}{2}$	0	$2L$	q_5
6	$\frac{\pi}{2}$	0	0	q_6
7	0	a_7	0	q_7

TABLE I
KINEMATIC PARAMETERS FOR THE DLR LWR-III ARM

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
matrice_s_dlr	1	2.201 s	2.201 s	

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
NE_0001_LWR	7	0.105 s	0.067 s	
mats_NE	1	0.105 s	0.000 s	
vprod	490	0.038 s	0.038 s	

Fig. 3. Execution times for the closed-form function (top) and for the $\mathbf{NE}^*$-based function (bottom)

In particular, two MATLAB *m*-functions were programmed, one containing the closed form expression of $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, called *matrice_s_LWR*, and another, called *mats_NE*, that computes $\boldsymbol{C}^*(\boldsymbol{q}, \dot{\boldsymbol{q}})$ column-wise with the $\mathbf{NE}^*$ algorithm being invoked $N = 7$ times, as explained in Sect. IV-A. The programs were run on an Intel® Core Duo 1.83 GHz machine with 1 GB of RAM, under MAC OS X®. For comparing the execution times, the MATLAB *profiler* utility was used, which is based on CPU time measurements.

The obtained results for this test are shown in Fig. 3, where *NE_0001_LWR* is the core $\mathbf{NE}^*$ algorithm function and *vprod* is an internal function that computes vector products in $\mathbb{R}^3$. The "total time" is the effective time needed for (all) the calls of the profiled function, while the "self-time" is the time necessary to execute the profiled function itself, excluding the time needed by other functions possibly called therein. The achieved improvement by the proposed numerical method is about 20 times.

VI. Conclusions

Motivated by two computational problems that arise in a recent approach to fault detection and in the classical passivity-based tracking control law for robot manipulators, we introduced a novel version of the recursive Newton-Euler

numerical algorithm. The modified algorithm allows to compute some dynamic terms in which there is a need of splitting the quadratic dependence on the generalized velocity in two different quantities, the robot velocity $\dot{\boldsymbol{q}}$ and an auxiliary velocity vector $\dot{\boldsymbol{q}}_a$. Moreover, the algorithm provides the numerical version of a factorization matrix for the Coriolis and centrifugal terms which automatically satisfies the skew-symmetric property. The improved efficiency of the proposed numerical method has been tested on a robot with 7 rotational dofs, and is expected to be even superior when N increases.

On-going work is devoted to addressing the relationships between the factorization matrices provided by the algorithm, the one generated by Christoffel symbols, and the complete set of all such matrices satisfying the skew-symmetric property. Also, a more complete numerical comparison with customized symbolic computations is under way. On the other hand, the inclusion of prismatic joints in the novel algorithm is rather straightforward.

As a possible extension of this approach, we foresee its application to the inverse dynamics problem for robots with elastic joints [16], [24]. In that case, the evaluation of higher-order derivatives of dynamic model terms expressed in symbolic form could be replaced by a suitable variation of the proposed numerical recursive method.

ACKNOWLEDGEMENTS

This work has been funded by the European Commission's Sixth Framework Programme as part of the project PHRIENDS under grant no. 045359, and by the MIUR project PRIN 2007 SICURA.

REFERENCES

[1] J. Hollerbach, "A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity", *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 10, pp. 730–736, 1980.

[2] J. J. Murray and C. P. Neuman, "Organizing customized robot dynamics algorithms for efficient numerical evaluation", *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 115–125, 1988.

[3] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators", *ASME J. of Dynamic Systems, Measurement, and Control*, vol. 102, no. 2, pp. 69–76, 1980.

[4] J. Koplik and M. C. Leu, "Computer generation of robot dynamics equations and the related issues", *J. of Robotic Systems*, vol. 3, no. 3, pp. 301–319, 1986.

[5] P. K. Khosla and T. Kanade, "Real-time implementation and evaluation of model-based controls on CMU DD arm", in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1986, pp. 1546–1555.

[6] C. P. Neuman and J. J. Murray, "Symbolically efficient formulations for computational robot dynamics", *J. of Robotic Systems*, vol. 4, no. 6, pp. 743–769, 1987.

[7] L. Sciavicco, B. Siciliano, and L. Villani, "Lagrange and Newton-Euler dynamic modeling of a gear-driven rigid robot manipulator with inclusion of motor inertia effects", *Advanced Robotics*, vol. 10, pp. 317–334, 1996.

[8] R. Featherstone and D. E. Orin, "Dynamics", in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., pp. 35–65. Springer, 2008.

[9] A. De Luca and R. Mattone, "Actuator failure detection and isolation using generalized momenta", in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2003, pp. 634–639.

[10] A. De Luca, A. Albu-Schäffer, S. Haddadin, and G. Hirzinger, "Collision detection and safe reaction with the DLR-III lightweight robot arm", in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 1623–1630.

[11] A. De Luca and L. Ferrajoli, "Exploiting robot redundancy in collision detection and reaction", in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2008, pp. 3299–3305.

[12] L. Le Tien, A. Albu-Schäffer, A. De Luca, and G. Hirzinger, "Friction observer and compensation for control of robots with joint torque measurements", in *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2008, pp. 3789–3795.

[13] A. De Luca and R. Mattone, "Sensorless robot collision detection and hybrid force/motion control", in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2005, pp. 1011–1016.

[14] M. Takegaki and S. Arimoto, "A new feedback method for dynamic control of manipulators", *ASME J. of Dynamic Systems, Measurement, and Control*, vol. 102, pp. 119–125, 1981.

[15] J.-J. Slotine and W. Li, "On the adaptive control of robot manipulators", *Int. J. of Robotics Research*, vol. 6, no. 3, pp. 49–59, 1987.

[16] A. De Luca and W. Book, "Robots with flexible elements", in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., pp. 287–319. Springer, 2008.

[17] G. Hirzinger, A. Albu-Schäffer, M. Hähnle, I. Schaefer, and N. Sporer, "On a new generation of torque controlled light-weight robots", in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2001, pp. 3356–3363.

[18] L. Sciavicco and B. Siciliano, *Modeling and Control of Robot Manipulators*, Springer, London, 2nd edition, 2000.

[19] A. De Luca and R. Mattone, "An adapt-and-detect actuator FDI scheme for robot manipulators", in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2004, pp. 4975–4980.

[20] A. De Luca and R. Mattone, "An identification scheme for robot actuator faults", in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2005, pp. 1127–1131.

[21] W. K. Chung, L.-C. Fu, and S.-H. Hsu, "Motion control", in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., pp. 133–159. Springer, 2008.

[22] J. Yuan and B. Yuan, "Recursive computation of the Slotine-Li regressor", in *Proc. American Control Conf.*, 1995, pp. 2327–2331.

[23] A. Albu-Schäffer, *Regelung von Robotern mit elastischen Gelenken am Beispiel der DLR-Leichtbauarme*, PhD thesis (in German), Technische Universität München, 2001.

[24] A. Albu-Schäffer and G. Hirzinger, "A globally stable state-feedback controller for flexible joint robots", *Advanced Robotics*, vol. 15, no. 8, pp. 799–814, 2001.