A Geometric Dynamics Algorithm for Serially Linked Robots

Mohammad Safeea[1,2][0000−0003−2312−4594], Pedro Neto[1][0000−0003−2177−5078], and Richard Béarée[2][0000−0002−6186−0755]

[1] Department of Mechanical Engineering University of Coimbra, Portugal
{ms@,pedro.neto@dem.}uc.pt
[2] Arts et Métiers, LISPEN, Lille, France
richard.bearee@ensam.eu

Abstract. This study introduces the Geometric Dynamics Algorithm (GDA) for representing the dynamics of serially linked robots. GDA is non-symbolic, preserves simple formulation, and is convenient for numerical implementation. GDA-based algorithms are deduced for efficient calculation of various dynamic quantities including (1) joint space inertia matrix (JSIM) (2) Coriolis matrix (3) centrifugal matrix (4) and the time derivative of JSIM. The proposed algorithms were analyzed in terms of their computational complexity. Results compare favorably with other methods.

Keywords: Dynamics · serially linked robots · geometric dynamic algorithm.

1 Introduction

Inverse dynamics equation of robots is a complex multi-variable function of its joints torques velocities accelerations and positions. The joint space formulation of this equation is:

$$\boldsymbol{\tau} = \mathbf{A}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g} \tag{1}$$

Where $\boldsymbol{\tau}$ is the vector of robot's joints torques, $\boldsymbol{q}$ is the vector of joints positions, $\dot{\boldsymbol{q}}$ is the vector of joints angular velocities, $\ddot{\boldsymbol{q}}$ is the vector of joints angular accelerations, $\mathbf{A}(\boldsymbol{q})$ is the joint space inertia matrix of the robot, $\mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the joint space Coriolis matrix of the robot, and $\boldsymbol{g}$ is the vector of joint's torques due to gravity.

One of the earliest methods used to deduce the equations of robot dynamics is based on Lagrangian formulation. This method is well described in the literature [1]. However, the method requires partial differentiation, usually done off-line on a computer using a symbolic math software [2]. In addition, for robots with high degrees of freedom (high DOF) the generated equations are enormous resulting in slow execution [3]. The formulation of robot-specific dynamics using Kane's equations is in [4], the authors demonstrated the methodology for deducing dynamics equations of a robot by hand. Newton-Euler recursive method is described in [5]. It includes forward propagation for calculating links accelerations

followed by a backward propagation for calculating joints torques. The method is very efficient for calculating the torques vector. However, the calculations are carried out implicitly such that the dynamic matrices (inertia, Coriolis, centrifugal) cannot be retrieved directly. Composite Rigid Body Algorithm (CRBA) [6] is an efficient method for calculating the mass matrix of robots. Though, it can not be used for calculating other dynamic matrices. Calculating Coriolis matrix is important for control applications, for example in passivity-based control as noted in [7]. In [8], the Coriolis matrix, $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, according to the notation of that paper, was used for calculating collision detection signal. In that study the Coriolis matrix transpose appears in the term $\mathbf{C}^{\mathrm{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ of equation (22), as such Coriolis matrix has to be calculated explicitly in real time for performing real time reaction control. Coriolis matrix can be deduced in a symbolic form by utilizing Euler-Lagrange formulation through partial differentiation and by utilizing Christoffel symbol of the first kind.

In this study we propose a novel method for representing the dynamics of robots. In our method, the dynamics quantities are calculated as contributions of frames' effects in what we call the frame injection effect. In such a case, the dynamics of each link is described using a separate equation linear in $\ddot{\boldsymbol{q}}$, expressed in a mathematical formulation resembling the manipulator's inverse dynamics equation. This facilitates (as shown later) the process of calculating: joint space inertia matrix (JSIM), Coriolis matrix, centrifugal matrix, and the time derivative of JSIM, (TD-JSIM), for serially linked robots. Accompanying code for various algorithms described in this article is available publicly in the repository [9].

2 Theory and principals

The proposed algorithm depends on what we call the frame injection effect, introduced in our previous work [10]. Each frame j attached to joint j transfers to link i a linear acceleration into its center of mass $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ and an inertial moment around its center of mass $\boldsymbol{\mu}_{\mathrm{C}ij}$, Fig. 1 (a). This transfer is due to the rotational effect of joint j around its axes of rotation, or the z axis of frame j according to modified Denavit Hartenberg (MDH) designation. This cause and effect relationship between frame j and link i is referred to by the subscript ij in $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ and $\boldsymbol{\mu}_{\mathrm{C}ij}$, while the C in the subscript is used to refer to the mass center of link i, the same subscript notation will hold throughout this paper for denoting frame-link interaction of cause-and-effect unless stated otherwise.

2.1 Link's acceleration due to the single-frame rotation

It can be proved that each frame j transfers to the center of mass of link i three acceleration vectors tangential ($\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij}$), normal ($\ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij}$) and Coriolis ($\ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij}$). The first of which is shown in Fig. 1 (b), it is due to the angular acceleration of frame j, and it can be calculated from:

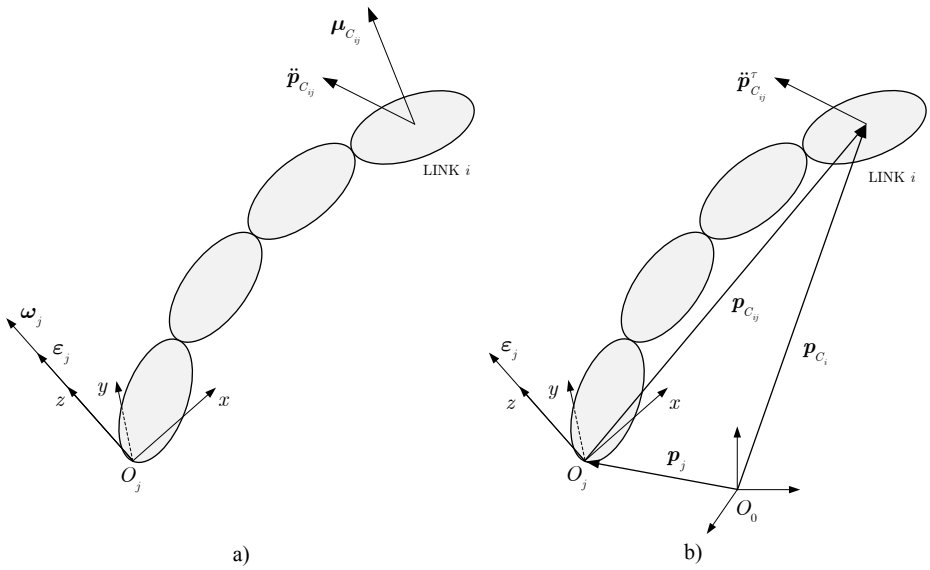

Fig. 1. (a) Inertial moment and linear acceleration transferred to link i by frame j. (b) Tangential acceleration of center of mass of link i transferred by frame j.

$$\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} = \boldsymbol{\varepsilon}_j \times \boldsymbol{p}_{\mathrm{C}ij} \tag{2}$$

Where $\boldsymbol{p}_{\mathrm{C}ij}$ is the vector connecting the origin of frame j and the center of mass of link i, $\times$ is the cross product, and $\boldsymbol{\varepsilon}_j$ is the angular acceleration of link j:

$$\boldsymbol{\varepsilon}_j = \ddot{q}_j \boldsymbol{k}_j \tag{3}$$

Where $\boldsymbol{k}_j$ is the unit vector associated with the z axis of joint j, and $\ddot{q}_j$ is the angular acceleration of that joint. The normal acceleration is shown in Fig. 2 (a), and it is given by:

$$\ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} = \boldsymbol{\omega}_j \times (\boldsymbol{\omega}_j \times \boldsymbol{p}_{\mathrm{C}ij}) \tag{4}$$

Where $\boldsymbol{\omega}_j$ is the angular velocity of link j due to the rotation of joint j:

$$\boldsymbol{\omega}_j = \dot{q}_j \boldsymbol{k}_j \tag{5}$$

Thus, we can rewrite the equation of the normal acceleration:

$$\ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} = \boldsymbol{k}_j \times (\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij}) \dot{q}_j^2 \tag{6}$$

Coriolis acceleration $\ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij}$ is shown in Fig. 2 (b), it can be calculated from:

$$\ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} = 2\boldsymbol{\omega}_j \times \boldsymbol{v}^{r}_{\mathrm{C}ij} \tag{7}$$

Where $\boldsymbol{v}^r_{\mathrm{C}ij}$ is the velocity transferred to the center of mass of link i from frames $j+1$ up to frame i, and the superscript r is to denote that this is a relative velocity. $\boldsymbol{v}^r_{\mathrm{C}ij}$ can be calculated from:

$$\boldsymbol{v}^r_{\mathrm{C}ij} = \sum_{k=j+1}^{i} \boldsymbol{\omega}_k \times \boldsymbol{p}_{\mathrm{C}ik} \tag{8}$$

As such the total linear acceleration transferred by frame j to the center of mass of link i is given by:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}ij} = \ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} \tag{9}$$

2.2 Link's inertial moment due to the single-frame rotation

It can be proved that each frame j will transfer to link i three inertial moments, the first of the inertial moments transferred is due to angular acceleration of frame j:

$$\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} = (\mathbf{R}_i \mathbf{I}^i_i \mathbf{R}^{\mathrm{T}}_i)\boldsymbol{\varepsilon}_j \tag{10}$$

Where $\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij}$ is the moment transferred by frame j into link i due to frame's j angular acceleration, $\mathbf{R}_i$ is the rotation matrix of frame i relative to base frame, $\mathbf{I}^i_i$ is the inertial tensor of link i around its center of mass represented in frame i.

The second inertial moment $\boldsymbol{\mu}^{n}_{\mathrm{C}ij}$ is due to centrifugal effect:

$$\boldsymbol{\mu}^{n}_{\mathrm{C}ij} = \frac{1}{2}(\mathbf{L}_i \boldsymbol{\omega}_j) \times \boldsymbol{\omega}_j \tag{11}$$

Where $\mathbf{L}_i$ is a 3×3 matrix that is calculated from:

$$\mathbf{L}_i = \mathbf{R}_i(\mathrm{tr}(\mathbf{I}^i_i)\mathbf{1}_3 - 2\mathbf{I}^i_i)\mathbf{R}^{\mathrm{T}}_i \tag{12}$$

Where the subscript in $\mathbf{L}_i$ is to notate that the matrix calculated pertains to link i, $\mathrm{tr}(\mathbf{I}^i_i)$ is the trace of the inertial tensor, and $\mathbf{1}_3$ is the identity matrix.

The third inertial moment $\boldsymbol{\mu}^{cor}_{\mathrm{C}ij}$ is due to Coriolis effect:

$$\boldsymbol{\mu}^{cor}_{\mathrm{C}ij} = (\mathbf{L}_i \boldsymbol{\omega}_j) \times \boldsymbol{\omega}^r_{ij} \tag{13}$$

Where $\boldsymbol{\omega}^r_{ij}$ can be calculated from:

$$\boldsymbol{\omega}^r_{ij} = \sum_{k=j+1}^{i} \boldsymbol{\omega}_k \tag{14}$$

Thus, the total inertial moment transferred to link i around its center of mass due to the rotational effect of frames j is given by:

$$\boldsymbol{\mu}_{\mathrm{C}ij} = \boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} + \boldsymbol{\mu}^{n}_{\mathrm{C}ij} + \boldsymbol{\mu}^{cor}_{\mathrm{C}ij} \tag{15}$$

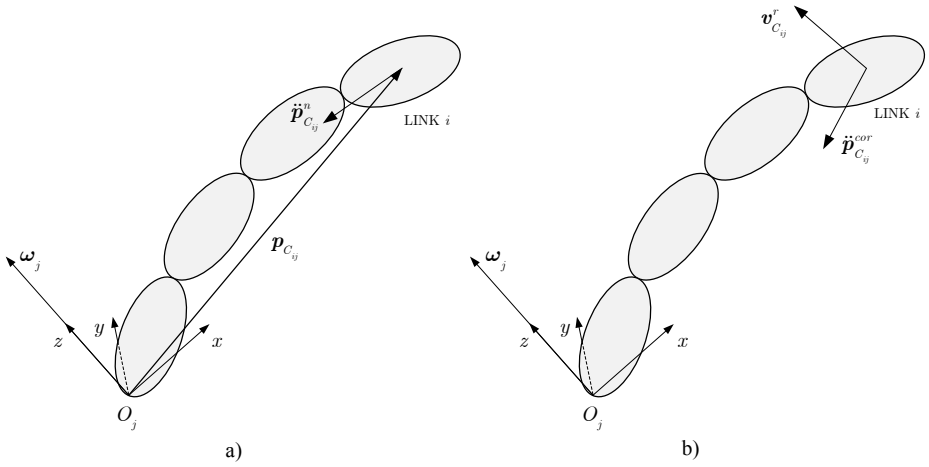

Fig. 2. (a) Normal acceleration of center of mass of link i transferred by frame j. (b) Coriolis acceleration of center of mass of link i transferred by frame j.

2.3 Dynamical representation of a single-link

Each link i can be represented dynamically by an equivalent acceleration of its center of mass $\ddot{\boldsymbol{p}}_{\mathrm{C}i}$ and an inertial moment around its center of mass $\boldsymbol{\mu}_{\mathrm{C}i}$ Fig. 3 (a). Where $\ddot{\boldsymbol{p}}_{\mathrm{C}i}$ is given by:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}i} = \sum_{j=1}^{i} \ddot{\boldsymbol{p}}_{\mathrm{C}ij} \tag{16}$$

Or by substituting $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ with its value from (9) we get:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}i} = \sum_{j=1}^{i} \ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} \tag{17}$$

From the previous equation we notice that the total acceleration of the center of mass of link i can be rewritten in a form similar to the equation of inverse dynamics by using a matrix vector notation as in the following:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}i} = \mathbf{C}_i \ddot{\boldsymbol{q}} + \mathbf{D}_i \dot{\boldsymbol{q}} \tag{18}$$

Where $\mathbf{C}_i$ is $3 \times n$ matrix and the j^{th} column vector of this matrix is given by:

$$\mathrm{col}_j(\mathbf{C}_i) = \frac{\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij}}{\ddot{q}_j} = \boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij} \tag{19}$$

The subscript i attached to the matrix $\mathbf{C}_i$ is used to notate that the matrix pertains to link i. The robot's links has n of $\mathbf{C}$ matrices each pertains to one link. $\mathbf{D}_i$ is also a $3 \times n$ matrix, and the subscript i in $\mathbf{D}_i$ is used as before. $\mathbf{D}_i$ columns can be calculated from:

$$\mathrm{col}_j(\mathbf{D}_i) = \boldsymbol{k}_j \times (\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij})\dot{q}_j + 2\boldsymbol{k}_j \times \boldsymbol{v}^r_{\mathrm{C}ij} \tag{20}$$

Using the same reasoning $\boldsymbol{\mu}_{\mathrm{C}i}$ can be calculated:

$$\boldsymbol{\mu}_{\mathrm{C}i} = \sum_{j=1}^{i} \boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} + \boldsymbol{\mu}^{n}_{\mathrm{C}ij} + \boldsymbol{\mu}^{cor}_{\mathrm{C}ij} \tag{21}$$

Again we notice that the contribution of $\ddot{\boldsymbol{q}}$ to the inertial moment is associated only with $\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij}$. Consequently, we can rearrange (21):

$$\boldsymbol{\mu}_{\mathrm{C}i} = \mathbf{U}_i\ddot{\boldsymbol{q}} + \mathbf{V}_i\dot{\boldsymbol{q}} \tag{22}$$

Where $\mathbf{U}_i$ is a $3 \times n$ matrix, each column vector of this matrix is given by:

$$\mathrm{col}_j(\mathbf{U}_i) = (\mathbf{R}_i\mathbf{I}^i_i\mathbf{R}^{\mathrm{T}}_i)\boldsymbol{k}_j \tag{23}$$

Similarly, $\mathbf{V}_i$ is a $3 \times n$ matrix, each column vector of this matrix is given by:

$$\mathrm{col}_j(\mathbf{V}_i) = \frac{1}{2}(\mathbf{L}_i\boldsymbol{\omega}_j) \times \boldsymbol{k}_j + (\mathbf{L}_i\boldsymbol{k}_j) \times \boldsymbol{\omega}^r_{ij} \tag{24}$$

As a result, each link is represented by a linear acceleration of its center of mass, and an inertial moment around its center of mass. Their mathematical equation is linearized in $\ddot{\boldsymbol{q}}$ and $\dot{\boldsymbol{q}}$, resembling the manipulator's inverse dynamics equation.

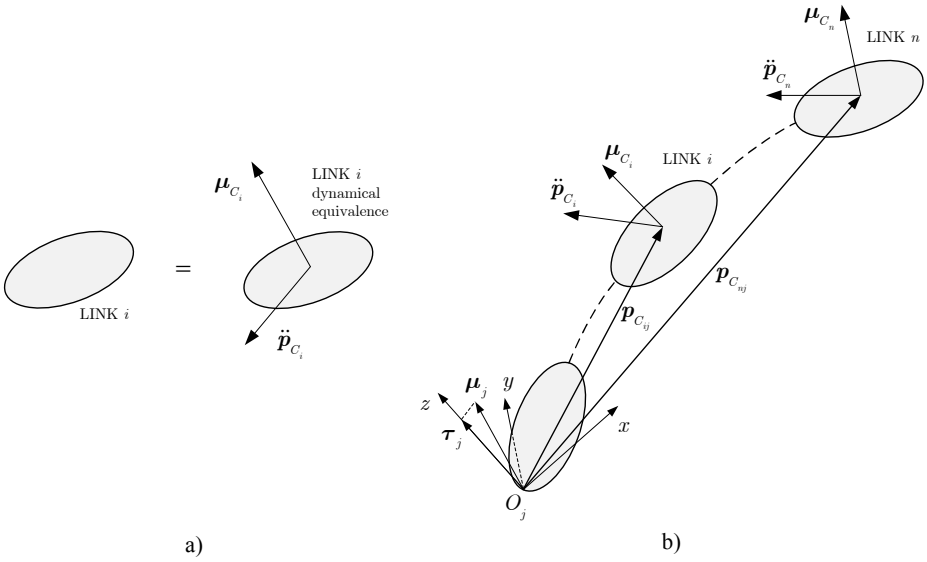

Fig. 3. (a) Dynamical representation of each link. (b) Moment acting on joint j due to robot's motion.

2.4 Moment acting on a joint due to robot dynamics

From figure 3 (b), the total moment $\boldsymbol{\mu}_j$ acting on joint j is:

$$\boldsymbol{\mu}_j = \sum_{i=j}^{n} \boldsymbol{\mu}_{Ci} + m_i \boldsymbol{p}_{Cij} \times \ddot{\boldsymbol{p}}_{Ci} \tag{25}$$

Which can be rearranged in a form resembling the inverse dynamics equation:

$$\boldsymbol{\mu}_j = \mathbf{G}_j \ddot{\boldsymbol{q}} + \mathbf{H}_j \dot{\boldsymbol{q}} \tag{26}$$

Where $\mathbf{G}_j$ is $3 \times n$ matrix, each column k of this matrix is given by:

$$\mathrm{col}_k(\mathbf{G}_j) = \sum_{i=j}^{n} \mathrm{col}_k(\mathbf{U}_i) + m_i \boldsymbol{p}_{Cij} \times \mathrm{col}_k(\mathbf{C}_i) \tag{27}$$

In a similar way, each column k of matrix $\mathbf{H}_j$ can be calculated from:

$$\mathrm{col}_k(\mathbf{H}_j) = \sum_{i=j}^{n} \mathrm{col}_k(\mathbf{V}_i) + m_i \boldsymbol{p}_{Cij} \times \mathrm{col}_k(\mathbf{D}_i) \tag{28}$$

Subscript j in $\mathbf{H}_j$ and $\mathbf{G}_j$ is to denote that these matrices are associated with joint j.

2.5 Robot's dynamics, joint space inertia matrix, Coriolis and centrifugal matrices

The torque acting on joint k due to robot dynamics is calculated from projecting $\boldsymbol{\mu}_k$, derived in the previous section, onto the z axis of joint k as in the following:

$$\boldsymbol{\tau}_k = \boldsymbol{k}_k^{\mathrm{T}} \boldsymbol{\mu}_k \tag{29}$$

If we designate the inertial matrix by the symbol $\mathbf{A}$, then each row k of it, $\mathrm{col}_k(\mathbf{A}^{\mathrm{T}})$, is calculated from:

$$\mathrm{col}_k(\mathbf{A}^{\mathrm{T}}) = \boldsymbol{k}_k^{\mathrm{T}} \mathbf{G}_k \tag{30}$$

Using the same notion, each row k of Coriolis matrix $\mathbf{B}$, or $\mathrm{col}_k(\mathbf{B}^{\mathrm{T}})$ can be calculated from:

$$\mathrm{col}_k(\mathbf{B}^{\mathrm{T}}) = \boldsymbol{k}_k^{\mathrm{T}} \mathbf{H}_k \tag{31}$$

Thus, the inverse dynamics equation of the robot defined by the inertial matrix $\mathbf{A}(\boldsymbol{q})$ and Coriolis matrix $\mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, has been derived.

Using the same principals developed in this section the joint space centrifugal matrix can be calculated, this is done by considering only the terms resulting from Centrifugal accelerations during the calculation of matrices $\mathbf{D}_i$, $\mathbf{V}_i$ and $\mathbf{H}_j$.

2.6 Geometrical calculation of the time derivative of inertia matrix

The time derivative of JSIM shows up as a by-product when calculating the time derivative of the generalized momentum in [8], which has applications in collision detection, while as noted in [7] the author had to change the formulation of the equation describing the rate of change of the generalized momentum in order to avoid the numerical differentiation of JSIM since calculating it numerically introduces errors which destroy its symmetric property. In this section we describe the methodology for deducing an efficient and fast algorithm of $O(n^2)$ for calculating TD-JSIM on the bases of the frame injection principal. The proposed algorithm was implemented in MATLAB® and the computational cost of this algorithm is presented in operation count section of this paper.

It has been shown previously in (30) that each row of JSIM is given by:

$$\mathrm{col}_k(\mathbf{A}^{\mathrm{T}}) = \boldsymbol{k}_k^{\mathrm{T}} \mathbf{G}_k$$

Thus, the time derivative of matrix $\mathbf{A}$ is calculated:

$$\mathrm{col}_k(\dot{\mathbf{A}}^{\mathrm{T}}) = \dot{\boldsymbol{k}}_k^{\mathrm{T}} \mathbf{G}_k + \boldsymbol{k}_k^{\mathrm{T}} \dot{\mathbf{G}}_k \tag{32}$$

Since $\boldsymbol{k}_k^{\mathrm{T}}$ is of a constant magnitude then its time derivative is given by:

$$\dot{\boldsymbol{k}}_k^{\mathrm{T}} = (\boldsymbol{\omega}_k^0 \times \boldsymbol{k}_k)^{\mathrm{T}} \tag{33}$$

Where $\boldsymbol{\omega}_k^0$ is the angular velocity of frame k. Since that the derivative of matrix $\mathbf{G}_k$ can be defined by the derivative of its columns then by considering (27), each column of $\dot{\mathbf{G}}_k$ is calculated from:

$$\mathrm{col}_j(\dot{\mathbf{G}}_k) = \sum_{i=k}^{n} \mathrm{col}_j(\dot{\mathbf{U}}_i) + m_i \dot{\boldsymbol{p}}_{\mathrm{C}ik} \times \mathrm{col}_j(\mathbf{C}_i)$$

$$+ m_i \boldsymbol{p}_{\mathrm{C}ik} \times \mathrm{col}_j(\dot{\mathbf{C}}_i) \tag{34}$$

While it can be shown that the derivative terms inside the summation of the previous equation are equal to:

$$\mathrm{col}_j(\dot{\mathbf{U}}_i) = \boldsymbol{\omega}_i^0 \times (\mathbf{R}_i \mathbf{I}_i^i \mathbf{R}_i^{\mathrm{T}} \boldsymbol{k}_j) - (\mathbf{R}_i \mathbf{I}_i^i \mathbf{R}_i^{\mathrm{T}})((\boldsymbol{\omega}_i^0 - \boldsymbol{\omega}_j^0) \times \boldsymbol{k}_j) \tag{35}$$

And:

$$\dot{\boldsymbol{p}}_{\mathrm{C}ik} = \boldsymbol{v}_{Ci} - \boldsymbol{v}_k \tag{36}$$

Where $\boldsymbol{v}_{\mathrm{C}i}$ is the linear velocity of the center of mass of link i and $\boldsymbol{v}_k$ is the linear velocity of origin of frame k.

$$\mathrm{col}_j(\dot{\mathbf{C}}_i) = (\boldsymbol{\omega}_j^0 \times \boldsymbol{k}_j) \times \boldsymbol{p}_{\mathrm{C}ij} + \boldsymbol{k}_j \times (\boldsymbol{v}_{\mathrm{C}i} - \boldsymbol{v}_j) \tag{37}$$

Thus the TD-JSIM can be calculated. Using MATLAB®, the equations of this section were implemented in an efficient $O(n^2)$ algorithm for calculating TD-JSIM available in [9]. We also want to mention here that with slight modification

of the algorithm proposed, the term $\mathbf{C}^{\mathrm{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ in equation (22) of [8] can be calculated numerically and efficiently from the relation

$$\mathbf{C}^{\mathrm{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \dot{\mathbf{A}}\dot{\boldsymbol{q}} - \boldsymbol{b}$$

Where $\boldsymbol{b}$ is the Coriolis vector. This calculation can be done efficiently, by taking an advantage of the fact that several parameters of robot dynamics are calculated at the same time when $\dot{\mathbf{A}}$ is being calculated, thus $\boldsymbol{b}$ can be calculated as a by-product while calculating $\dot{\mathbf{A}}$, the cost analysis of the proposed algorithms is described in operation count section.

3 Implementation and results

To assess the performance of GDA, a comparison with well established algorithms was performed. While Robotics Toolbox for MATLAB® (RTB) [11], CRBA algorithm by Featherstone [12] and the symbolic-numeric method in [13], were used for comparison, we want to mention here that after a lengthy search for robotics toolboxes, RTB, is the only toolbox that we could find on MATLAB® that can calculate Coriolis matrix numerically.

Accompanying MATLAB® code is provided in the repository [9], where we developed the following functions:

1- GetMassMatrixGDA: calculates joint space inertia matrix.

2- GetCoriolisMatrixGDA: calculates joint space Coriolis matrix..

3- GetCentrifugalMatrixGDA: calculates joint space centrifugal matrix.

4- GetDerivativeOfMassMatrixGDA: calculates the time derivative of JSIM.

5- GetCTdqGDA: calculates the time derivative of JSIM, Coriolis vector, and $\mathbf{C}^{\mathrm{T}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ as described before.

The repository also contains a detailed breakdown on the computational complexity for the implemented algorithms, or the number of floating point operations, additions and multiplications, as a function of the number of DOF of the robot. Table 1 shows a summary, for comparison the table lists also the computational complexity of the other algorithms. For constructing joint space inertia and Coriolis matrices, RTB invokes several calls of the efficient recursive Newton-Euler, as described in [14]. As such RTB constructs JSIM column by column. This is done by assigning a unity value to only one element of joint's acceleration vector, while assigning zeros to remaining elements, joints velocities and gravity term, then recursion is performed, and the associated column vector of the inertia matrix is calculated. The same procedure is repeated for all of JSIM columns, and since that the computational complexity of recursive Newton-Euler is $O(n)$ and that constructing the inertia matrix requires n recursions, then the total computational complexity of the algorithm is of $O(n^2)$. For calculating Coriolis matrix RTB uses the same methodology it applies for calculating JSIM, that is by performing several recursions, at each recursion the angular accelerations and gravity term are set to zero, while one of the possible combinations of $(\dot{q}_i, \dot{q}_j)$ is utilized, as such RTB invokes $n^2/2$ recursions, and the total computational complexity is of $O(n^3)$.

Table 1 also shows that the Composite Rigid Body Algorithm (CRBA) is more efficient in calculating the mass matrix. In contrast, the proposed method can be used to calculate the inertia matrix, Coriolis matrix, Centrifugal matrix, and TD-JSIM matrix with computational complexity of $O(n^2)$. An added advantage of GDA is that when the aforementioned matrices are being evaluated robot dynamics represented by inertial moments and accelerations of the links are being calculated as a by-product of the computations.

Table 1. Operation count for proposed method and other methods, m stands for multiplication and a for addition.

Method	Matrix	Cost
GDA	JSIM	$(18n^2 + 49n)m + (16n^2 + 37n - 3)a$
GDA	Coriolis	$(36n^2 + 49n - 3)m + (34.5n^2 + 32.5n - 6)a$
GDA	Centrifugal	$(15n^2 + 58n)m + (13.5n^2 + 39.5n - 3)a$
GDA	TD-JSIM	$(46.5n^2 + 124.5n - 36)m + (51.5n^2 + 117.5n - 41)a$
CRBA	JSIM	$(10n^2 + 22n - 32)m + (6n^2 + 37n - 43)a$
Symbolic-Numeric	JSIM-Coriolis	$(\frac{3}{2}n^3 + \frac{35}{2}n^2 + 9n - 16)m + (\frac{7}{6}n^3 + \frac{23}{2}n^2 + \frac{64}{3}n - 28)a$
RTB	JSIM	$O(n^2)$
RTB	Coriolis	$O(n^3)$

4 Conclusion

In this paper we proposed a novel algorithm for representing the dynamics of serially linked robots and calculating its joint space inertia matrix, Coriolis matrix, centrifugal matrix, and the time derivative of joint space inertia matrix. In addition we described the frame injection principal. A comparison between the proposed algorithm against other well established algorithms was made, the performance of the proposed algorithm was discussed in operation count section of this paper. We perceive that GDA's way of representing robot dynamics in a mathematical form resembling the equation of the inverse dynamics as intuitive, simple and easy to implement. It's remarkable efficiency is reflected by the $O(n^2)$ algorithm deduced for calculating the time derivative of joint space inertia matrix. By using this methodology other parameters of robot dynamics can be calculated efficiently in a like manner to what had been presented in this paper.

Acknowledgments

This research was partially supported by Portugal 2020 project DM4Manufacturing POCI-01-0145-FEDER-016418 by UE/FEDER through the program COMPETE 2020, and the Portuguese Foundation for Science and Technology (FCT) SFRH/BD/131091/2017 and COBOTIS (PTDC/EME- EME/32595/2017).

References

1. Lorenzo Sciavicco, Bruno Siciliano, and Luigi Villani. Lagrange and newton-euler dynamic modeling of a gear-driven robot manipulator with inclusion of motor inertia effects. *Advanced robotics*, 10(3):317–334, 1995.
2. Charles P Neuman and John J Murray. Symbolically efficient formulations for computational robot dynamics. *Journal of robotic systems*, 4(6):743–769, 1987.
3. Wisama Khalil. Dynamic modeling of robots using recursive newton-euler techniques. In *ICINCO2010*, 2010.
4. Thomas R Kane and David A Levinson. The use of kane's dynamical equations in robotics. *The International Journal of Robotics Research*, 2(3):3–21, 1983.
5. John YS Luh, Michael W Walker, and Richard PC Paul. On-line computational scheme for mechanical manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76, 1980.
6. Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.
7. Alessandro De Luca and Lorenzo Ferrajoli. A modified newton-euler method for dynamic computations in robot fault detection and control. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3359–3364. IEEE, 2009.
8. Alessandro De Luca, Alin Albu-Schaffer, Sami Haddadin, and Gerd Hirzinger. Collision detection and safe reaction with the dlr-iii lightweight manipulator arm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1623–1630. IEEE, 2006.
9. Matlab code for GDA. https://github.com/Modi1987/The-Geometric-Dynamics-Algorithm-GDA-
10. M. Safeea, R. Bearee, and P. Neto. Reducing the computational complexity of mass-matrix calculation for high dof robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5614–5619, Oct 2018.
11. Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer Science & Business Media, 2011.
12. R. Featherstone. Spatial vectors and rigid-body dynamics.
13. M Vukobratović, Shi-Gang Li, and N Kirćanski. An efficient procedure for generating dynamic manipulator models. *Robotica*, 3(03):147–152, 1985.
14. Peter I Corke et al. A computer tool for simulation and analysis: the robotics toolbox for matlab. In *Proc. National Conf. Australian Robot Association*, pages 319–330, 1995.