Science Arts & Métiers (SAM)

is an open access repository that collects the work of Arts et Métiers Institute of Technology researchers and makes it freely available over the web where possible.

This is an author-deposited version published in: https://sam.ensam.eu
Handle ID: .http://hdl.handle.net/10985/17376

To cite this version :

Mohammad SAFEEA, Pedro NETO, Richard BEAREE - Reducing the Computational Complexity of Mass-Matrix Calculation for High DOF Robots - In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Espagne, 2018 - 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) - 2018

Any correspondence concerning this service should be sent to the repository Administrator : scienceouverte@ensam.eu

Arts et Métiers Institute of Technology

Reducing the computational complexity of mass-matrix calculation for high DOF robots

Mohammad Safeea, Richard Bearee, *Senior Member*, *IEEE*, and Pedro Neto, *Member*, *IEEE*

***Abstract*—Increasingly, robots have more degrees of freedom (DOF), imposing a need for calculating more complex dynamics. As a result, better efficiency in carrying out dynamics computations is becoming more important. In this study, an efficient method for computing the joint space inertia matrix (JSIM) for high DOF serially linked robots is addressed. We call this method the Geometric Dynamics Algorithm for High number of robot Joints (GDAHJ). GDAHJ is non-symbolic, preserve simple formulation, and it is convenient for numerical implementation. This is achieved by simplifying the way to recursively derive the mass-matrix exploiting the unique property of each column of the JSIM and minimizing the number of operations with $O(n^2)$ complexity. Results compare favorably with existing methods, achieving better performance over state-of-the-art by Featherstone when applied for robots with more than 13 DOF.**

***Index Terms*—mass-matrix, dynamics, Geometric Dynamics Algorithm for High number of robot Joints (GDAHJ), high DOF robots.**

I. Introduction

DYNAMICS of robots is an important topic since that it is highly involved in their design, simulation and control. Owing to its importance this subject had been studied extensively in the past thirty years. Thus, several algorithms and methods had been developed to calculate robot dynamics [1] [2]. Nevertheless, this subject remains till this day open for extensive research while every year there are new studies being published, methods and algorithms being proposed.

Robot dynamics can be described by one of two formulations:

1) Operational space formulation. In this formulation the dynamics equations are referenced to the manipulator end-effector. In a pioneering study this approach was described and used to control PUMA600 robot [3]. It is also applied for the combined application of motion and force control [4]. Algorithms for efficient robot dynamics calculations based on operational space formulation are presented in [5] and [6].
2) Joint space formulation. This formulation describes the dynamics of robot in joint space. This formulation manifests the effect of the joints' positions, velocities and accelerations on the torques and vice-versa.

The mathematical formulation of the inverse dynamics in joint space [7, 8, 9, 10] is given by:

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

Where $\boldsymbol{\tau}$ is the joints' torques vector, $\boldsymbol{q}$ is the joints' positions vector, $\dot{\boldsymbol{q}}$ is the vector of joints' angular velocities, $\ddot{\boldsymbol{q}}$ is the vector of joints' angular accelerations, $\mathbf{A}(\boldsymbol{q})$ is 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 joints' torques due to gravity. As described in [11], equation (1) can be extended to include contact forces, joints elasticity, friction, actuators inertias and dynamics. $\mathbf{A}(\boldsymbol{q})$ is an $n \times n$ matrix, in which n is the number of robot's joints considering that each joint has one degree of freedom (DOF), it is symmetric, positive definite and has the property of being a function of only joints' positions. $\mathbf{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is an $n \times n$ matrix, function of joints' positions and velocities, and describes the centrifugal and Coriolis effects on joints' torques.

One of the earliest methods used to deduce the equations of robot dynamics was the one based on Lagrangian formulation. This method is well described in the literature. A methodology for deducing the dynamics of gear-driven serially linked robot by using Lagrangian formulation is described in [12]. This study took into consideration the effects of the driving motors. The Lagrangian formulation is widely used as the bases for automatic generation of equations of robot dynamics in symbolic form. Most recent toolboxes for generating equations of robot dynamics using Lagrangian formulation are in [13].

The Lagrangian formulation is a straight forward approach that treats the robot as a whole and utilizes its Lagrangian, a function that describes the energy of the mechanical system:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{2}$$

Where $\mathcal{L}$ is the Lagrangian function, $\mathcal{T}$ is the kinetic energy and $\mathcal{U}$ is the potential energy. The function described previously is formulated in terms of the generalized coordinates $\boldsymbol{q}$. By differentiating that function we can derive an expression of the associated generalized forces $\boldsymbol{v}$:

$$\boldsymbol{v} = \frac{d}{dt}\left(\frac{\delta \mathcal{L}}{\delta \dot{\boldsymbol{q}}}\right)^{\mathrm{T}} - \left(\frac{\delta \mathcal{L}}{\delta \boldsymbol{q}}\right)^{\mathrm{T}} \tag{3}$$

Even though the Lagrangian formulation can be considered as a straight forward approach, the method requires partial differentiation. Despite the fact that symbolic manipulation methods have been utilized to perform the differentiation [14],

This work was supported in part by the Portuguese Foundation for Science and Technology (FCT) project COBOTIS (No 32595), Portugal 2020 project DM4Manufacturing POCI-01-0145-FEDER-016418 by UE/FEDER through the program COMPETE2020, and the Portuguese Foundation for Science and Technology (FCT) SFRH/BD/131091/2017.

Mohammad Safeea is with the Department of Mechanical Engineering, University of Coimbra, Portugal and with Arts et Métiers, ParisTech, France, e-mail: ms@uc.pt.

Richard Bearee is with Arts et Métiers, ParisTech, Lille, France, e-mail: Richard.BEAREE@ENSAM.EU.

Pedro Neto is with the Department of Mechanical Engineering, University of Coimbra, Coimbra, Portugal, e-mail: pedro.neto@dem.uc.pt.

the method still lacks the efficiency in terms of execution-time. This can be clearly noticed when the robot presents a relatively high number of DOF as noted in [9] and most remarkably in [15], where the author performed comparison of execution-times required to run simulations based on dynamical models derived by Newton-Euler recursive technique and Euler-Lagrange technique. It was reported execution-times difference of order of magnitude which clearly put the case in favor of the Newton-Euler recursion method.

The formulation of robot-specific dynamics using Kane's dynamical equations is in [16]. In this study the authors argue that using Lagrange method to compute dynamics produces huge equations resulting in slow execution and costly computations, while the Recursive Newton-Euler is a generalized method that might perform unnecessary calculations on specific robot. Thus, a faster execution algorithm with less computational-cost could be achieved if robot-specific equations are carefully deduced. The study elaborates in step by step manner the methodology for deriving dynamics equations of Stanford manipulator starting from Kane's dynamical equations. Nevertheless, the method requires a knowledgeable analyst to take on a pencil and paper in hand and work out the equations of a specific robot. A comprehensive review of Kane's equations and Gibbs-Appell equations is in [17].

A computationally efficient Newton-Euler recursive method is described in [18]. This method is performed in two phases: the first phase (forward propagation) during which the accelerations and velocities of robot links are calculated, and the second phase (backward propagation) where torques and forces are calculated. The method proved to be very efficient for calculating the inverse dynamics. However, the calculations are carried out implicitly such that the inertia matrix cannot be retrieved directly. It is shown in [19] that the inertia matrix $\mathbf{A}(\boldsymbol{q})$ can be calculated from the model of the inverse dynamics by assigning a unit value to one element of the joints' accelerations vector and assigning a zero value to the remaining elements, including the joints' velocities and the gravity term. In such scenario the associated column of the inertia matrix can be calculated, and by iterating the procedure through all of the elements of the joints' acceleration vector the inertia matrix is achieved. This method was later renamed composite-rigid-body algorithm (CRBA), by Featherstone [20]. Using CRBA to calculate the inertia matrix proved to be computationally efficient, especially if the calculations are performed in links-attached local frames. Computer code of the algorithm based on 6D or spatial vectors algebra is available in [20]. A comprehensive review of spatial vectors and Plücker basis is in [21] and in chapter 2 of [22].

In this study we propose GDAHJ as a method to calculate JSIM for articulated bodies with relatively high number of DOF, GDAHJ achieves better efficiency over state-of-the-art method, the famous CRBA. This increase in efficiency is achieved through minimizing the number of operations that have $O(n^2)$ computational complexity. In GDAHJ the number of computations associated with the quadratic terms are reduced to the minimum value possible, from $16n^2$ in the case of CRBA to $5.5n^2$ for GDAHJ.

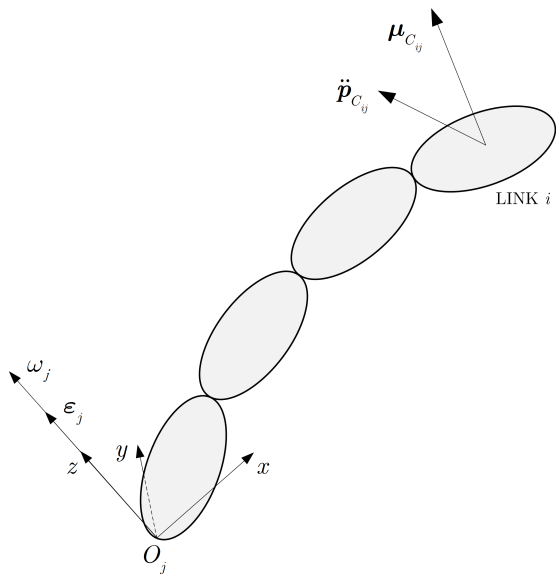

Figure 1. Inertial moment $\boldsymbol{\mu}_{\mathrm{C}ij}$ and linear acceleration $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ of centre of mass of link i transferred by frame j

II. Theory and principals

The proposed algorithm builds on what we call the frame injection effect, Figure 1, in which each frame j attached to joint j will transfer to link i a linear acceleration into its centre of mass and an inertial moment around its centre of mass. In this study we notate them by $\ddot{\boldsymbol{p}}_{\mathrm{C}ij}$ and $\boldsymbol{\mu}_{\mathrm{C}ij}$, respectively. 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 centre of link i. The same subscript notation will hold throughout this study for denoting frame-link interaction of cause-and-effect unless stated otherwise.

A. Link's acceleration due to the single-frame rotation

Each frame j transfers to link i three acceleration vectors tangential acceleration, normal acceleration and Coriolis acceleration. The first of which is shown in Figure 2, it is due to the angular acceleration of frame j:

$$\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} = \boldsymbol{\varepsilon}_j \times \boldsymbol{p}_{\mathrm{C}ij} \tag{4}$$

Where $\ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij}$ is the tangential acceleration of the centre of mass of link i due to the rotation of frame j, the symbol $\times$ is used to denote the cross product (the same notation of the cross product will hold throughout this study) and $\boldsymbol{p}_{\mathrm{C}ij}$ is the vector connecting the origin of frame j and the centre of mass of link i. $\boldsymbol{\varepsilon}_j$ is the angular acceleration of link j:

$$\boldsymbol{\varepsilon}_j = \ddot{q}_j \boldsymbol{k}_j \tag{5}$$

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.

Concerning the normal acceleration, each frame j transfers to link i a normal acceleration due to its rotation, Figure 2:

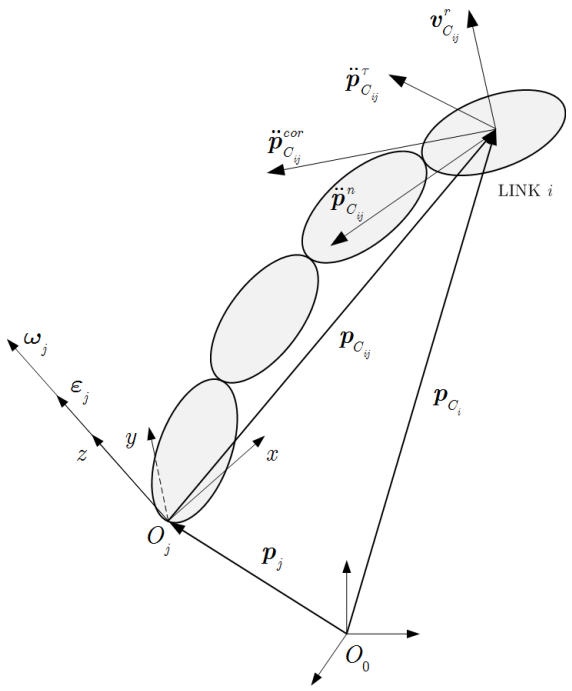

Figure 2. Tangential acceleration of centre of mass of link i transferred by frame j

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

Where $\boldsymbol{\omega}_j$ is the angular velocity of link j due to the rotational effect of joint j. It is given by:

$$\boldsymbol{\omega}_j = \dot{q}_j \boldsymbol{k}_j \tag{7}$$

We can rewrite the equation of the normal acceleration transferred to link i due to frame j by:

$$\ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} = \boldsymbol{k}_j \times (\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij})\dot{q}_j^2 \tag{8}$$

The third acceleration transferred is Coriolis acceleration, Figure 2, in which each frame j transfers to link i Coriolis acceleration $\ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij}$:

$$\ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} = 2\boldsymbol{\omega}_j \times \boldsymbol{v}^{r}_{\mathrm{C}ij} \tag{9}$$

Where $\boldsymbol{\omega}_j$ is as described previously in equation (7), and $\boldsymbol{v}^{r}_{\mathrm{C}ij}$ is the velocity transferred to the centre of mass of link i from frames $j+1$ up to frame i. Here, the r in the superscript is to denote that this is a relative velocity, and C in the subscript is used to refer to the mass centre of link i, so that $\boldsymbol{v}^{r}_{\mathrm{C}ij}$ can be calculated from:

$$\boldsymbol{v}^{r}_{\mathrm{C}ij} = \sum_{k=j+1}^{i} \boldsymbol{\omega}_k \times \boldsymbol{p}_{\mathrm{C}ik} \tag{10}$$

The total linear acceleration transferred by frame j to the centre of mass of link i is given by:

$$\ddot{\boldsymbol{p}}_{\mathrm{C}ij} = \ddot{\boldsymbol{p}}^{\tau}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{n}_{\mathrm{C}ij} + \ddot{\boldsymbol{p}}^{cor}_{\mathrm{C}ij} \tag{11}$$

B. Link's inertial moment due to single-frame effect

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 and it is given by:

$$\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} = (\mathbf{R}_i \mathbf{I}^i_i \mathbf{R}^{\mathrm{T}}_i)\boldsymbol{\varepsilon}_j \tag{12}$$

While $\boldsymbol{\mu}^{\tau}_{\mathrm{C}ij}$ 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 in relation to base frame, and $\mathbf{I}^i_i$ is 3×3 inertial tensor of link i around its centre of mass represented in frame i.

The second inertial moment transferred from frame j to link i is due to centrifugal effect:

$$\boldsymbol{\mu}^{n}_{\mathrm{C}ij} = \frac{1}{2}(\mathbf{L}_i \boldsymbol{\omega}_j) \times \boldsymbol{\omega}_j \tag{13}$$

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

$$\mathbf{L}_i = \mathbf{R}_i(\mathrm{tr}(\mathbf{I}^i_i)\mathbf{1}_3 - 2\mathbf{I}^i_i)\mathbf{R}^{\mathrm{T}}_i \tag{14}$$

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 transferred from frame j to link i is due to Coriolis effect:

$$\boldsymbol{\mu}^{cor}_{\mathrm{C}ij} = (\mathbf{L}_i \boldsymbol{\omega}_j) \times \boldsymbol{\omega}^{r}_{ij} \tag{15}$$

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

$$\boldsymbol{\omega}^{r}_{ij} = \sum_{k=j+1}^{i} \boldsymbol{\omega}_k \tag{16}$$

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

$$\boldsymbol{\mu}_{\mathrm{C}ij} = \boldsymbol{\mu}^{\tau}_{\mathrm{C}ij} + \boldsymbol{\mu}^{n}_{\mathrm{C}ij} + \boldsymbol{\mu}^{cor}_{\mathrm{C}ij} \tag{17}$$

III. Joint space inertia matrix for high DOF (JSIMHJ)

The GDAHJ algorithm calculates the joint space inertia matrix for robots with high DOF quite efficiently, this increase in efficiency is achieved through minimizing the number of operations that has $O(n^2)$ computational complexity to a minimum, according to our knowledge GDAHJ is the most efficient method for high DOF robots ever proposed till now.

Starting from the basic interpretation of JSIM columns, the mathematical equations of GDAHJ algorithm can be deduced. Where as described in section 3.2 of [1], each column j of the JSIM can be interpreted as: the torques acting on the various joints of the robot, due to the unit acceleration of joint j, giving that the angular velocities of all of the joints are equal to zero. In Figure 3 we show the free body diagram of one link of the robot, with the inertial moments and inertial forces acting on it.

Following the previous definition of column j of JSIM, we can calculate that column as the following: (1) choose a joint

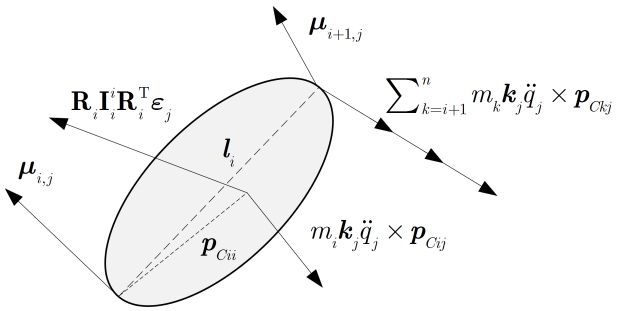

Figure 3. Inertial forces and moments acting on link i due to angular acceleration of joint j.

j, and (2) write the balance equation of a link i from the robot. By referring to Figure 3, the balance equation of link i:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j} + (\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}})\boldsymbol{k}_j\ddot{q}_j + m_i\hat{\boldsymbol{p}}_{\mathrm{C}ii}(\ddot{q}_j\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij}) +$$

$$\hat{\boldsymbol{l}}_i \sum_{k=i+1}^{n} m_k(\ddot{q}_j\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}kj}) \tag{18}$$

Where $\boldsymbol{\mu}_{i,j}$ is the total moment acting on joint i due to the acceleration of joint j only. $\boldsymbol{l}_i$ is the vector connecting the origin of frame i to the proceeding frame's origin, the little hat notation above the vector is used to denote the skew symmetric operator associated with that vector. From the definition given in the previous section of column i, we substitute $\ddot{q}_j$ by its value $\ddot{q}_j = 1$. Then the modified balance equation is:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j} + (\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}})\boldsymbol{k}_j + m_i\hat{\boldsymbol{p}}_{\mathrm{C}ii}(\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}ij})$$

$$+\hat{\boldsymbol{l}}_i \sum_{k=i+1}^{n} m_k(\boldsymbol{k}_j \times \boldsymbol{p}_{\mathrm{C}kj}) \tag{19}$$

While:

$$\boldsymbol{p}_{\mathrm{C}ij} = \boldsymbol{p}_{\mathrm{C}i} - \boldsymbol{p}_j \tag{20}$$

And:

$$\boldsymbol{p}_{\mathrm{C}kj} = \boldsymbol{p}_{\mathrm{C}k} - \boldsymbol{p}_j \tag{21}$$

We substitute the values of $\boldsymbol{p}_{\mathrm{C}ij}$ and $\boldsymbol{p}_{\mathrm{C}kj}$ into (19), and we fix:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j}$$

$$+\left(\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}} - m_i\hat{\boldsymbol{p}}_{\mathrm{C}ii}\hat{\boldsymbol{p}}_{\mathrm{C}i} - \hat{\boldsymbol{l}}_i\left(\sum_{k=i+1}^{n} m_k\hat{\boldsymbol{p}}_{\mathrm{C}k}\right)\right)\boldsymbol{k}_j$$

$$-\left(m_i\boldsymbol{p}_{\mathrm{C}ii} + (\sum_{k=i+1}^{n} m_k)\boldsymbol{l}_i\right) \times (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{22}$$

We define the vector $\boldsymbol{\eta}_i$ by:

$$\boldsymbol{\eta}_i = m_i\boldsymbol{p}_{\mathrm{C}ii} + (\sum_{k=i+1}^{n} m_k)\boldsymbol{l}_i \tag{23}$$

Algorithm 1 Calculating joint space inertia matrix entries, algorithm is based on eq (31).

For $i = 1 : n$
  For $j = 1 : i$
% calculating $\mathbf{A}_{i,j}$ will require two vector inner products and one
% scalar addition with total cost $(3n^2 + 3n)m + (2.5n^2 + 2.5n)a$
    $\mathbf{A}_{i,j} = \boldsymbol{k}_j^{\mathrm{T}}\boldsymbol{d}_i + \boldsymbol{t}_j^{\mathrm{T}}\boldsymbol{y}_i$
    $\mathbf{A}_{j,i} = \mathbf{A}_{i,j}$
  End
End

And we define the matrix operator $\boldsymbol{\kappa}_i$ by:

$$\boldsymbol{\kappa}_i = -m_i\hat{\boldsymbol{p}}_{\mathrm{C}ii}\hat{\boldsymbol{p}}_{\mathrm{C}i} - \hat{\boldsymbol{l}}_i\left(\sum_{k=i+1}^{n} m_k\hat{\boldsymbol{p}}_{\mathrm{C}k}\right) \tag{24}$$

Then we write:

$$\boldsymbol{\mu}_{i,j} = \boldsymbol{\mu}_{i+1,j} + \left(\mathbf{R}_i\mathbf{I}_i^i\mathbf{R}_i^{\mathrm{T}} + \boldsymbol{\kappa}_i\right)\boldsymbol{k}_j - (\boldsymbol{\eta}_i) \times (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{25}$$

By performing a recursion on previous equation from link n to link i, and noticing that $\boldsymbol{\mu}_{n+1,j} = \mathbf{0}$ we get:

$$\boldsymbol{\mu}_{i,j} = \left(\sum_{k=i}^{n}\left(\mathbf{R}_k\mathbf{I}_k^k\mathbf{R}_k^{\mathrm{T}} + \boldsymbol{\kappa}_k\right)\right)\boldsymbol{k}_j - \left(\sum_{k=i}^{n}\boldsymbol{\eta}_k\right) \times (\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{26}$$

To hide the complexity in the previous equation, we denote the terms between parenthesis by:

$$\boldsymbol{b}_i = \left(\sum_{k=i}^{n}\boldsymbol{\eta}_k\right) \tag{27}$$

And

$$\mathbf{D}_i = \sum_{k=i}^{n}\left(\mathbf{R}_k\mathbf{I}_k^k\mathbf{R}_k^{\mathrm{T}} + \boldsymbol{\kappa}_k\right) \tag{28}$$

Substituting (27) and (28) in (26) yields:

$$\boldsymbol{\mu}_{i,j} = \mathbf{D}_i\boldsymbol{k}_j - \hat{\boldsymbol{b}}_i(\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{29}$$

For calculating the (i, j) entry of JSIM, $\mathbf{A}_{i,j}$, we project $\boldsymbol{\mu}_{i,j}$ on the z axes of joint i, or in other words we multiply (29) by the unit vector $\boldsymbol{k}_i^{\mathrm{T}}$:

$$\mathbf{A}_{i,j} = \boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\mu}_{i,j} = \boldsymbol{k}_i^{\mathrm{T}}\mathbf{D}_i\boldsymbol{k}_j - \boldsymbol{k}_i^{\mathrm{T}}\hat{\boldsymbol{b}}_i(\boldsymbol{k}_j \times \boldsymbol{p}_j) \tag{30}$$

By noticing that each entry i, j of the JSIM, or $\boldsymbol{k}_i^{\mathrm{T}}\boldsymbol{\mu}_{i,j}$ is a scalar, then we can transpose the previous equation without loss of generality:

$$\mathbf{A}_{i,j} = \boldsymbol{k}_j^{\mathrm{T}}\left(\mathbf{D}_i^{\mathrm{T}}\boldsymbol{k}_i\right) - \left(\boldsymbol{k}_j \times \boldsymbol{p}_j\right)^{\mathrm{T}}\left(\hat{\boldsymbol{b}}_i^{\mathrm{T}}\boldsymbol{k}_i\right)$$

$$= \boldsymbol{k}_j^{\mathrm{T}}\left(\mathbf{D}_i^{\mathrm{T}}\boldsymbol{k}_i\right) + \left(\boldsymbol{k}_j \times \boldsymbol{p}_j\right)^{\mathrm{T}}\left(\hat{\boldsymbol{b}}_i\boldsymbol{k}_i\right) \tag{31}$$

In such a way we have decoupled the dependency between indexes i and j. Moreover, we limited the cross-coupling

Table I
OPERATION COUNT FOR PROPOSED METHOD AND OTHER METHODS, m STANDS FOR MULTIPLICATION AND a FOR ADDITION.

Method	Quantity	Cost	Reference
GDAHJ	JSIM	$(3n^2+88n-3)m+(2.5n^2+95.5n-18)a$	Proposed
CRBA	JSIM	$(10n^2+22n-32)m+(6n^2+37n-43)a$	[23] and [22] EQ. 10.3
Symbolic-Numeric	JSIM-Coriolis	$(\frac{3}{2}n^3+\frac{35}{2}n^2+9n-16)m+(\frac{7}{6}n^3+\frac{23}{2}n^2+\frac{64}{3}n-28)a$	[24]
RNEA†	Inverse dynamics	$(93n-108)m+(81n-100)a$	[22] Table 10.1

† Recursive Newton-Euler Algorithm (RNEA) is designed for calculating the inverse dynamics. Nevertheless, if the algorithm is invoked by passing $\dot{\boldsymbol{q}}=\mathbf{0}$, by ignoring the gravity vector, and by passing an angular acceleration vector with all elements equal to zero except for an element $\ddot{q}_j=1$, then the corresponding j column of JSIM is produced. In such a case, by invoking this algorithm n times, all columns of JSIM are calculated, and the resulting computational cost is $(93n^2-x_1n)m+(81n^2-x_2n)a$, which is very expensive in the $O(n^2)$ part of the algorithm.

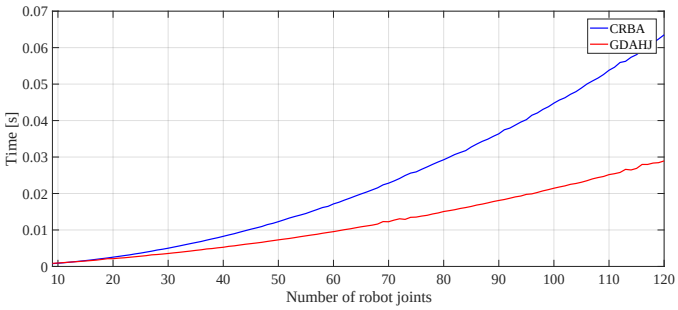

Figure 4. Execution time results for GDAHJ vs CRBA.

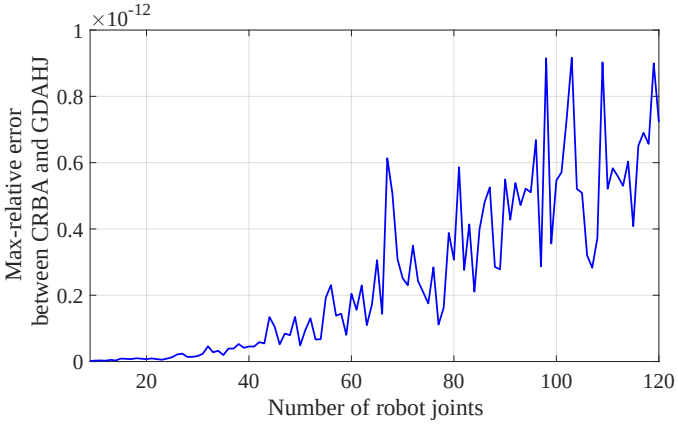

Figure 5. Relative error in computation for results achieved using GDAHJ and CRBA.

interaction between joint j and bodies i into a minimum. The previous equation states that the effect of acceleration of each joint j is limited to the terms $\boldsymbol{k}_j^{\mathrm{T}}$, and $\boldsymbol{t}_j=\left(\boldsymbol{k}_j\times\boldsymbol{p}_j\right)$. The effect of the articulated bodies from link n to link i is manifested by the terms $\left(\mathbf{D}_i^{\mathrm{T}}\boldsymbol{k}_i\right)$ and $\left(\hat{\boldsymbol{b}}_i^{\mathrm{T}}\boldsymbol{k}_i\right)$. The terms $\boldsymbol{d}_i=\left(\mathbf{D}_i^{\mathrm{T}}\boldsymbol{k}_i\right)$ and $\boldsymbol{y}_i=\left(\hat{\boldsymbol{b}}_i^{\mathrm{T}}\boldsymbol{k}_i\right)$ can be calculated with an $O(n)$ algorithm using multiple recursions, while the mass matrix entries can be calculated with minimum quadratic cost using the nested loop in Algorithm 1.

The nested loop in Algorithm 1 has the minimal quadratic cost. This cost results from two vector-inner products and one scalar addition, with a cost $(3n^2+3n)m+(2.5n^2+2.5n)a$, where m stands for multiplication and a stands for addition. Thus, the $O(n^2)$ computational cost is optimized.

IV. IMPLEMENTATION AND RESULTS

To prove the validity of the proposed method, GDAHJ, and to assess its execution-time performance, a comparison with well established algorithms was performed, namely with CRBA method.

Table I shows the computational complexity of the proposed algorithm against state-of-the-art algorithm, measured in the number of floating point operations (additions and multiplications) as function of n, the number of DOF of the robot. The operation count for CRBA reported in Table I pertains to the most efficient version of this algorithm [22]. The results reported in Table I for GDAHJ do not include the number of operations required to perform the direct kinematics of $O(n)$, since that most of robotics operations require direct kinematics calculation, otherwise the cost of the direct kinematics can be added. It can be inferred that GDAHJ performs better than CRBA for articulated bodies (serially connected) that have more than 13 DOF.

To confirm the theoretical results, both algorithms CRBA and GDAHJ were implemented in Matlab, and a comparison in terms of execution time between the two algorithms was performed. Numerical tests were carried out by considering a manipulator in which the mass of each link was generated randomly in the range [1,10] kg. The inertial tensors of each link were generated as random positive definite matrix in which each element of the matrix is in the range [1,10] $kg.m^2$. Denavit-Hartenberg parameters of each link were generated randomly as well as the pose of the robot (joint angles). Afterwards, the JSIM of the manipulator was calculated twice, once using CRBA method and another time using GDAHJ method. Figure 4 shows the results in terms of execution time and Figure 5 shows the results in terms of numerical error of the calculation between the two methods. From the figures we notice that GDAHJ performs better than CRBA in terms of execution time for high DOF robots (more than 13 DOF). The maximum ratio of time of execution CRBA/GDAHJ acheived is 2.2, which is less than the theoretical limit 16/5.5. Again, the time required for performing the direct kinematics for GDAHJ is not taken into consideration in the plots.

A metric-value was defined to measure the relative error. The proposed metric-value is calculated by the overall-sum of

the absolute values of the differences taken on all the cells of the mass matrix calculated by CRBA against its counterpart calculated by GDAHJ. Then, this sum is divided by the maximum absolute matrix-element value from the calculated JSIM:

$$e = \frac{sum(sum(abs(\mathbf{A_{GDAHJ}} - \mathbf{A_{CRBA}})))}{max(max(abs((\mathbf{A_{GDAHJ}} + \mathbf{A_{CRBA}})/2)))} \quad (32)$$

Where the implementation $sum(sum(arg))$ returns the sum of all elements of the argument matrix (arg), $abs(arg)$ is a function that returns a matrix where all its elements have the absolute value of their counterpart of the argument matrix argument matrix (arg). From the graph in Figure 5 we notice that the error is small and can be attributed to numerical rounding errors.

V. Conclusion

In this study we proposed GDAHJ, a novel algorithm for efficient calculation of JSIM for serially linked robots, the algorithm achieves better efficiency over state-of-the-art when calculating JSIM for hyper-joint manipulators. This increase in efficiency is achieved through minimizing the number of operations associated with the $O(n^2)$. In such a case, the number of computations associated with the quadratic terms are reduced to the minimum value possible, from $(16n^2)$ in the case of CRBA to $(5.5n^2)$ for the proposed algorithm. At the end comparison between the proposed algorithm against state of the art CRBA was made, the performance of the proposed algorithm was discussed in operation count section of this study. On a theoretical level, this study demonstrates the minimum bound for the $O(n^2)$ operations required for calculating JSIM. Future work will focus on reducing the number of operations associated with $O(n)$ of the algorithm.

References

[1] R. Featherstone and D. Orin, "Robot dynamics: equations and algorithms," in *ICRA*, 2000, pp. 826–834.

[2] C. Balafoutis, "A survey of efficient computational methods for manipulator inverse dynamics," *Journal of Intelligent and Robotic Systems*, vol. 9, no. 1-2, pp. 45–71, 1994.

[3] O. Khatib, "Dynamic control of manipulator in operational space," in *Proc. 6th IFToMM World Congress on Theory of Machines and Mechanisms*, 1983, pp. 1128–1131.

[4] O. . Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 1, pp. 43–53, 1987.

[5] K.-S. Chang, "Efficient algorithms for articulated branching mechanisms: dynamic modeling, control, and simulation," Ph.D. dissertation, Citeseer, 2000.

[6] K.-S. Chang and O. Khatib, "Efficient recursive algorithm for the operational space inertia matrix of branching mechanisms," *Advanced Robotics*, vol. 14, no. 8, pp. 703–715, 2001.

[7] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2009.

[8] J. J. Craig, *Introduction to robotics: mechanics and control*. Pearson Prentice Hall Upper Saddle River, 2005, vol. 3.

[9] W. Khalil, "Dynamic modeling of robots using recursive newton-euler techniques," in *ICINCO2010*, 2010.

[10] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB*. Springer Science & Business Media, 2011, vol. 73.

[11] M. W. Spong, "Control of robots and manipulators," *The control handbook*, pp. 1339–1351, 1996.

[12] L. Sciavicco, B. Siciliano, and L. Villani, "Lagrange and newton-euler dynamic modeling of a gear-driven robot manipulator with inclusion of motor inertia effects," *Advanced robotics*, vol. 10, no. 3, pp. 317–334, 1995.

[13] M. Toz and S. Kucuk, "Dynamics simulation toolbox for industrial robot manipulators," *Computer Applications in Engineering Education*, vol. 18, no. 2, pp. 319–330, 2010.

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

[15] H. Hoifodt, "Dynamic modeling and simulation of robot manipulators: the newton-euler formulation," 2011.

[16] T. R. Kane and D. A. Levinson, "The use of kane's dynamical equations in robotics," *The International Journal of Robotics Research*, vol. 2, no. 3, pp. 3–21, 1983.

[17] H. Baruh, *Analytical dynamics*. MacGraw-Hill, 1999.

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

[19] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, no. 3, pp. 205–211, 1982.

[20] R. Featherstone, "Spatial vectors and rigid-body dynamics." [Online]. Available: http://royfeatherstone.org/spatial/

[21] ——, "Plucker basis vectors," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 1892–1897.

[22] ——, *Rigid body dynamics algorithms*. Springer, 2014.

[23] ——, "Efficient factorization of the joint-space inertia matrix for branched kinematic trees," *The International Journal of Robotics Research*, vol. 24, no. 6, pp. 487–500, 2005.

[24] M. Vukobratović, S.-G. Li, and N. Kirćanski, "An efficient procedure for generating dynamic manipulator models," *Robotica*, vol. 3, no. 03, pp. 147–152, 1985.