Closed-form method for the inertia-weighted input matrix utilizing O(n) forward dynamics

G. Krög[1], H. Gattringer[1], A. Müller[1]

[1] Institute of Robotics
Johannes Kepler University Linz
Altenberger Straße 69, 4040 Linz, Austria
[gabriel.kroeg, hubert.gattringer, a.mueller]@jku.at

ABSTRACT

This paper presents a closed-form method to evaluate the inertia-weighted input matrix, which is the map between inputs and system accelerations, using intermediate results from an $O(n)$ method, the Implicit Inversion Method (IIM). This matrix can be used to greatly reduce the calculation times for optimal control problems which include the Equations of Motion (EoM), as the relationship between inputs and accelerations is fixed for a certain time step and can therefore be evaluated a priori. That means that the EoM only need to be solved once for the whole problem instead of having to solve the equations in every iteration of the optimization. The method presented in this paper especially targets the case that the system dynamics are implemented using an $O(n)$ method and takes advantage of that by reusing variables that occurred in the evaluation of that method. Using these quantities, it is possible to calculate the inertia-weighted input matrix without having to convert the system to minimal form. Utilizing the shape of the resulting equation, it is even possible to avoid having to explicitly invert any matrices.

Keywords: dynamics, recursive dynamics algorithms, robotics, numerical methods, optimal control.

1 INTRODUCTION

In operational-space control like [1], the dynamics of the robot is often incorporated as constraints like

$$\begin{aligned} \min_{\ddot{\mathbf{q}},\mathbf{u}} \frac{1}{2} \left\| \mathbf{J}_t\ddot{\mathbf{q}} + \dot{\mathbf{J}}_t\dot{\mathbf{q}} - \dot{\mathbf{V}}_{t,c} \right\|^2 \\ \text{s.t. } \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{B}(\mathbf{q})\mathbf{u}, \end{aligned} \tag{1}$$

to ensure that the commanded behavior is physically feasible. In this kind of optimization problem, both the accelerations $\ddot{\mathbf{q}}$ and inputs $\mathbf{u}$, which can contain both actuation torques and interaction forces here, are used as optimization variables. Since the relationship between these is given through the Equations of Motion (EoM) of the robot, they have to be solved for the accelerations in every iteration of the optimization to make sure that the result remains dynamically consistent. Because of this, the inclusion of system dynamics is a very costly feature as the system mass matrix $\mathbf{M}$ needs to be inverted every time. To mitigate this, the system dynamics can be evaluated before starting the optimization, eliminating the system accelerations from the problem as they linearly depend on the inputs like in [2] or [3]. This results in problems like

$$\min_{\mathbf{u}} \frac{1}{2} \left\| \mathbf{J}_t\mathbf{M}^{-1}\mathbf{B}\mathbf{u} - \mathbf{J}_t\mathbf{M}^{-1}\mathbf{h} + \dot{\mathbf{J}}_t\dot{\mathbf{q}} - \dot{\mathbf{V}}_{t,c} \right\|^2, \tag{2}$$

where the EoM only needs to be solved once for every time step, since except for the optimization variables $\mathbf{u}$, all other occurring quantities only depend on desired values or the current state. With this, they can be considered as constant since this optimization has to be evaluated for every time step separately. This means that the system dynamics now only need to be evaluated once per optimization instead of the hundreds if not thousands of times that would be necessary otherwise, depending on how many iterations are needed.

For multi-body systems with many degrees of freedom (DOF), efficient methods to solve the EoM like the various existing recursive dynamics algorithms, such as the Articulated-Body Algorithm (ABA) by Featherstone [4] or a similar $O(n)$-method by Bremer [5], which is used in this work, show their advantages in terms of computational time. Since the standard form of these methods is unable to produce mentioned decoupling of the acceleration terms however, the inertia-weighted input matrix (IWIM) $\hat{\mathbf{B}} = \mathbf{M}^{-1}\mathbf{B}$, which is the linear map between the desired system inputs and the task accelerations, as can be seen in (2), has to be derived separately. Previous work on a similar topic was done in [6] and [7], which are based on methods from [8], where different derivations of the inverse operational-space inertia matrix (OSIM) was shown. These approaches however, treat the contact forces separately from motor torques, as in these cases, the inertia matrix of one or more specific manipulators is the topic of interest. This paper presents an algorithm which enables the calculation of the IWIM for systems with chain or tree topology using intermediate results obtained by solving an $O(n)$ method, called the Implicit Inversion Method (IIM) here.

In section 2, a brief overview of the task-space optimal control problem is given as a motivation for this new formulation. Section 3 then covers the method used to obtain the dynamic model and the $O(n)$ algorithm that is used here to solve the EoM, showing the problems that arise from using this method. A short overview of the two mentioned existing methods is given in section 4 and the main contribution of this work, the IIM, is then presented in section 5. Building on this, simulation results of several models and runtime comparisons are shown in section 6. Finally, section 7 concludes this paper.

2 TASK CONTROL OPTIMIZATION PROBLEM

A generic optimization problem for task control in operational-space as shown in [3] is written as (1) with the task Jacobian $\mathbf{J}_t$ giving the relationship between a task twist $\mathbf{V}_t$ and the minimal velocities $\dot{\mathbf{q}}$, $\mathbf{V}_t = \mathbf{J}_t\dot{\mathbf{q}}$. The commanded time derivative of the task twist $\dot{\mathbf{V}}_{t,c}$ contains all the necessary information about the desired task trajectory. To ensure that the resulting output is physically feasible, the EoM are considered in the constraints, which is computationally very expensive if both accelerations and inputs are used as optimization variables, as mentioned above. A more efficient approach is to compute

$$\ddot{\mathbf{q}} = -\mathbf{M}^{-1}\mathbf{h} + \mathbf{M}^{-1}\mathbf{B}\mathbf{u} \tag{3}$$

before solving (1), as shown in(2). That way, the system accelerations $\ddot{\mathbf{q}}$ can be split up into the components

$$\ddot{\mathbf{q}} = \ddot{\mathbf{q}}_0 + \ddot{\mathbf{q}}_u, \tag{4}$$

the free acceleration $\ddot{\mathbf{q}}_0 = -\mathbf{M}^{-1}\mathbf{h}$ that only depends on the current system state and the accelerations caused by the inputs, $\ddot{\mathbf{q}}_u$, which depend linearly on $\mathbf{u}$ through the IWIM, so $\ddot{\mathbf{q}}_u = \hat{\mathbf{B}}\mathbf{u}$. For the derivative of the task twist, this means

$$\dot{\mathbf{V}}_t = \mathbf{J}_t\ddot{\mathbf{q}} + \dot{\mathbf{J}}_t\dot{\mathbf{q}} = \mathbf{J}_t(\ddot{\mathbf{q}}_u + \ddot{\mathbf{q}}_0) + \dot{\mathbf{J}}_t\dot{\mathbf{q}} = \mathbf{J}_t\hat{\mathbf{B}}\mathbf{u} + \dot{\mathbf{V}}_{t,0}, \tag{5}$$

where $\dot{\mathbf{V}}_{t,0} = \mathbf{J}_t\ddot{\mathbf{q}}_0 + \dot{\mathbf{J}}_t\dot{\mathbf{q}}$ are the free task accelerations that result from the system behavior without inputs. The matrix $\mathbf{J}_t\hat{\mathbf{B}}$ is the map from the system inputs to the operational-space accelerations, which is a generalization of the cross-coupling inverse OSIM from [3]. Due to this separation, the nonlinear constraint that is the EoM can be directly incorporated into the cost function (2) which speeds up the optimization considerably, as the result is an unconstrained quadratic problem and the set of optimization variables is smaller.

Still, $\ddot{\mathbf{q}}_0$ needs to be evaluated, which leads to the question what approach should be used to solve the system dynamics.

3 $O(n)$ FORWARD DYNAMICS ALGORITHM

To efficiently handle the EoM of more complex systems, recursive dynamics algorithms have proven to be the method of choice, as some of them are computationally very efficient. The one described in the following is the $O(n)$-method presented in [5], which is closely related to the ABA described in [4].

3.1 Subsystem Formulation

The Subsystem formulation is a modeling approach where the complete system is regarded as an assembly of smaller subsystems which are attached to one another in certain coupling points, usually the joints that connect them. These subsystems are derived separately and can then be assembled as seen in

$$\sum_i \left(\frac{\partial \dot{\mathbf{y}}_i}{\partial \dot{\mathbf{q}}}\right)^\top (\mathbf{M}_i \ddot{\mathbf{y}}_i + \mathbf{h}_i(\dot{\mathbf{y}}_i, \mathbf{q}, \mathbf{u}_i)) = 0 \tag{6}$$

and then solves the dynamics recursively on the subsystem level, thus avoiding large matrix dimensions. This modeling approach offers great flexibility as additional components can be added without having to change the preexisting subsystems.

The matrix $\mathbf{M}_i$ is the corresponding mass matrix and $\mathbf{h}_i$ the dynamics vector of the subsystem i, which also includes the effects of the input $\mathbf{u}_i$ here. The variables $\dot{\mathbf{y}}_i$ respectively $\ddot{\mathbf{y}}_i$ are a set of velocities/accelerations, through which the full dynamics of subsystem i can be described. These velocities

$$\dot{\mathbf{y}}_i = \begin{pmatrix} \mathbf{v}_{o,i}^\top & \boldsymbol{\omega}_{o,i}^\top & \dot{\mathbf{q}}_i^\top \end{pmatrix}^\top \tag{7}$$

consist of the coupling point velocity $\mathbf{v}_{o,i}$, the angular velocity of the subsystem frame i, $\boldsymbol{\omega}_{o,i}$ and the internal velocities of the subsystem, $\dot{\mathbf{q}}_i$ as can be seen in fig. 1.

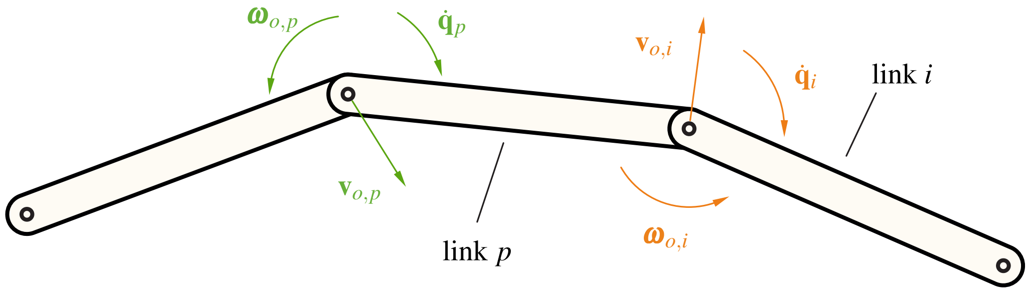

Figure 1. Visualization of the elements of $\dot{\mathbf{y}}_i$

The consecutive $\dot{\mathbf{y}}_i$ can be calculated recursively as

$$\dot{\mathbf{y}}_i = \mathbf{T}_{ip}\, \dot{\mathbf{y}}_p + \mathbf{F}_i\, \dot{\mathbf{q}}_i, \tag{8}$$

with the index p indicating the parent subsystem of i, the velocity propagation matrix $\mathbf{T}_{ip}$ from p to i and $\mathbf{F}_i$ the matrix that incorporates the free motion of the subsystem i into $\dot{\mathbf{y}}_i$. This leads to the global relationship

$$\begin{pmatrix} \dot{\mathbf{y}}_1 \\ \dot{\mathbf{y}}_2 \\ \vdots \\ \dot{\mathbf{y}}_N \end{pmatrix} = \begin{pmatrix} \mathbf{F}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{T}_{21}\mathbf{F}_1 & \mathbf{F}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{T}_{N1}\mathbf{F}_1 & \mathbf{T}_{N2}\mathbf{F}_2 & \cdots & \mathbf{F}_N \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_1 \\ \dot{\mathbf{q}}_2 \\ \vdots \\ \dot{\mathbf{q}}_N \end{pmatrix} = \mathbf{F}\dot{\mathbf{q}} \tag{9}$$

between the set of all velocities $\dot{\mathbf{y}}_i$ and the minimal system velocities $\dot{\mathbf{q}}$, which is a lower triangular block matrix. This property of $\mathbf{F}$ will be exploited later on in the IIM.

3.2 The $O(n)$ algorithm

The solution of the system dynamics as shown in (1) has a computational complexity of $O(n^3)$ due to the necessary inversion of the mass matrix and the dimensions of the system matrices. One way to mitigate these problems is to split the system into a number of smaller components which are then used to calculate the system dynamics. Using this principle, a number of recursive algorithms, like [5] or [4] have been developed which can reduce the computational complexity to $O(n)$. They do require increased overhead however, which is why they only evaluate faster than the minimal form from $n \approx 9$ on.

The $O(n)$-method introduced by Bremer in [5] consists of three recursions. In the first one, the kinematics is evaluated from the base subsystem outwards as can be seen in (8). Then, in the following inward recursion, the articulated-body inertia $\mathbf{M}_i^*$ and bias forces $\mathbf{h}_i^*$, as they are named in the ABA, are computed as

$$\mathbf{M}_p^* = \mathbf{M}_p + \sum_{i\in\{s_j(p)\}} \mathbf{T}_{ip}^{*\top}\mathbf{M}_i^*\mathbf{T}_{ip} \qquad \mathbf{h}_p^* = \mathbf{h}_p + \sum_{i\in\{s_j(p)\}} \mathbf{T}_{ip}^{*\top}\left(\mathbf{M}_i^*\dot{\mathbf{T}}_{ip}\dot{\mathbf{y}}_p + \mathbf{h}_i^*\right), \tag{10}$$

$$\text{with} \quad \mathbf{T}_{ip}^* = (\mathbf{I} - \mathbf{M}_i^*\mathbf{F}_i\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)^\top\mathbf{T}_{ip} \quad \text{and} \quad \mathbf{M}_{Ri} = \mathbf{F}_i^\top\mathbf{M}_i^*\mathbf{F}_i. \tag{11}$$

The set $\{s_j(p)\}$ denotes all subsystems that have p as its parent. Finally, the system accelerations are obtained through the final outwards recursion

$$\ddot{\mathbf{q}}_i = -\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top\left[\mathbf{M}_i^*(\mathbf{T}_{ip}\ddot{\mathbf{y}}_p + \dot{\mathbf{T}}_{ip}\dot{\mathbf{y}}_p) + \mathbf{h}_i^*\right] \qquad \text{with} \qquad \ddot{\mathbf{y}}_i = \mathbf{T}_{ip}\ddot{\mathbf{y}}_p + \dot{\mathbf{T}}_{ip}\dot{\mathbf{y}}_p + \mathbf{F}_i\ddot{\mathbf{q}}_i + \dot{\mathbf{F}}_i\dot{\mathbf{q}}_i. \tag{12}$$

The problem here is that this method gives just the resulting accelerations without the possibility of splitting it up the way it was done in (4). Therefore, an additional method is required to calculate the IWIM.

4 EXTENSION FROM OPERATIONAL-SPACE INERTIA MATRIX TO INERTIA-WEIGHTED INPUT MATRIX

The two methods mentioned in the introduction were developed for the calculation of the inverse OSIM $\mathbf{\Lambda}^{-1} = \mathbf{J}_t^\top\mathbf{M}^{-1}\mathbf{J}_t$, which defines the dynamic behavior of a robot manipulator regarding forces acting on the end effector and is a subset of the IWIM in task coordinates, $\mathbf{J}_t\hat{\mathbf{B}} = \begin{bmatrix}\mathbf{J}_t\mathbf{M}^{-1}\mathbf{J}_a^\top & \mathbf{\Lambda}^{-1}\end{bmatrix}$. Therefore, the OSIM can only take into account the contact forces at the respective manipulators and not actuator torques.

By splitting the input $\mathbf{u}$ up into motor torques $\boldsymbol{\tau}$ and contact forces $\mathbf{F}_c$, the input matrix $\mathbf{B}$ also has to be separated into $\mathbf{B} = \begin{pmatrix}\mathbf{J}_a^\top & \mathbf{J}_t^\top\end{pmatrix}$, with the actuation jacobian $\mathbf{J}_a^\top$ and the task jacobian which was introduced earlier. Inserting this relationship and the accelerations from (1) into (5) gives the manipulator accelerations

$$\dot{\mathbf{V}}_t = \mathbf{J}_t\mathbf{M}^{-1}\left(\mathbf{J}_t^\top\mathbf{F}_c + \mathbf{J}_a^\top\boldsymbol{\tau}\right) + \dot{\mathbf{V}}_{t,0} = \mathbf{\Lambda}^{-1}\mathbf{F}_c + \mathbf{J}_t\mathbf{M}^{-1}\mathbf{J}_a^\top\boldsymbol{\tau} + \dot{\mathbf{V}}_{t,0}. \tag{13}$$

The methods explained in the following are assuming the term $\mathbf{J}_t\mathbf{M}^{-1}\mathbf{J}_a^\top\boldsymbol{\tau} + \dot{\mathbf{V}}_{t,0}$ is already known, as mentioned in [9], section 2.5.4. This can be done through the $O(n)$-method for example.

The question now is, whether these methods can be adapted to return $\mathbf{J}_t\hat{\mathbf{B}}$ without knowledge of $\mathbf{M}$, using the subsystem quantities $\mathbf{M}_i$ and $\mathbf{h}_i$.

Extended-Force-Propagator Algorithm (EFPA) The EFPA by Wensing et al. [7] is an extension of the Force Propagation method from [8] and is a recursive algorithm. While it is very efficient at calculating $\mathbf{\Lambda}^{-1}$, it is noticeably harder to extend its functionality to the calculation of $\mathbf{J}_t\mathbf{M}^{-1}\mathbf{J}_a^\top$ since the desired input is the global input $\mathbf{u}$ for each recursive step and not the specific input of a certain subsystem $\mathbf{u}_i$. Therefore, this algorithm is not applicable for this paper.

Extended Unit Force Method (eUFM) Another way of calculating the inverse OSIM is shown in [6]. It is based on the Unit Force Method of [8], but generalized in the sense that it does not rely on assumptions to simplify the system dynamics. Because of this, it will be called the extended Unit Force Method later on.

To get the inverse OSIM $\mathbf{\Lambda}^{-1} = \mathbf{J}_t\mathbf{M}^{-1}\mathbf{J}_t^\top$, the matrix $\mathbf{M}^{-1}\mathbf{J}_t^\top$ is calculated first, since this term directly results from the EoM

$$\ddot{\mathbf{q}} = -\mathbf{M}^{-1}\left(\mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{J}_a^\top\boldsymbol{\tau}\right) + \mathbf{M}^{-1}\mathbf{J}_t^\top\mathbf{F}_c, \tag{14}$$

which can also be written as $\ddot{\mathbf{q}} = \ddot{\mathbf{q}}_{[0]} + \mathbf{M}^{-1}\mathbf{J}_t^\top\mathbf{F}_c$, where $\ddot{\mathbf{q}}_{[0]}$ are the system accelerations according to the current state and actuator torques, which are calculated using the $O(n)$-method.

The matrix $\mathbf{M}^{-1}\mathbf{J}_t^\top$ can then be calculated column-wise by choosing the force input as a unit vector $\mathbf{e}_i$ whose i-th entry is 1. When solving for the accelerations now, the result is

$$\ddot{\mathbf{q}}_{[i]} = \ddot{\mathbf{q}}_{[0]} + \mathbf{M}^{-1}\mathbf{J}_t^\top\mathbf{e}_i, \tag{15}$$

which means that the difference $\ddot{\mathbf{q}}_{[i]} - \ddot{\mathbf{q}}_{[0]}$ equates to the i-th column of $\mathbf{M}^{-1}\mathbf{J}_t^\top$. Therefore, by repeating this process for each entry of the contact force vector, the entire matrix is obtained. To get the OSIM now, this result only needs to be multiplied by $\mathbf{J}_t$.

While this method is less efficient at calculating the OSIM than the EFPA, it uses a global input vector. Through this, the input can be chosen as $\mathbf{u} = \begin{pmatrix}\boldsymbol{\tau}^\top & \mathbf{F}_c^\top\end{pmatrix}^\top$, which means that the dynamics can now be rewritten to (3) with $\mathbf{B} = \begin{bmatrix}\mathbf{J}_a^\top & \mathbf{J}_t^\top\end{bmatrix}$. For this case, the eUFM now returns $\hat{\mathbf{B}}$ instead of $\mathbf{M}^{-1}\mathbf{J}_t^\top$.

The main problem of this method is that the $O(n)$-algorithm has to be evaluated $m+1$ times, for $\mathbf{u} \in \mathbb{R}^m$, resulting in a computational complexity of $O(mn)$. This makes the method impracticable for greater numbers of inputs as the computation times rise linearly with the number of inputs additionally to the DOF.

5 IMPLICIT INVERSION METHOD FOR COMPUTING THE INERTIA-WEIGHTED INPUT MATRIX

Since most systems typically have a number of inputs that is close to if not higher than their DOF, especially when including contact forces, it is worthwhile to think about more efficient methods to solve this problem.

To get a linear relation between inputs and system accelerations, it is assumed that the subsystem dynamics vectors $\mathbf{h}_i$ from (6) are linear in the corresponding subsystem inputs $\mathbf{u}_i$, thus enabling a split into $\mathbf{h}_i = \mathbf{h}_{i,0} + \mathbf{h}_{i,u}$ with $\mathbf{h}_{i,u} = -\mathbf{B}_i\mathbf{u}_i$. Looking at the propagation formula for the bias forces in (10), the same kind of split can be introduced here

$$\begin{aligned}&\mathbf{h}_p^* = \mathbf{h}_{p,0}^* + \mathbf{h}_{p,u}^*\\ &\quad\text{with}\quad \mathbf{h}_{p,u}^* = \mathbf{h}_{p,u} + \sum_{i\in\{s_j(p)\}} \mathbf{T}_{ip}^{*\top}\mathbf{h}_{i,u}^* = -\mathbf{B}_p\mathbf{u}_p + \sum_{i\in\{s_j(p)\}} \mathbf{T}_{ip}^{*\top}\mathbf{h}_{i,u}^*\end{aligned} \tag{16}$$

From this, a global form $\mathbf{h}_u^* = \begin{pmatrix}\mathbf{h}_{1,u}^{*\,\top} & \cdots & \mathbf{h}_{N,u}^{*\,\top}\end{pmatrix}^\top = -\mathbf{B}^*\mathbf{u}$ with $\mathbf{u}$ as the entire system input can then be derived as

$$\mathbf{B}^* = \begin{pmatrix}\mathbf{I} & \mathbf{T}_{21}^{*\top} & \cdots & \mathbf{T}_{N1}^{*\top}\\ \mathbf{0} & \mathbf{I} & \cdots & \mathbf{T}_{N2}^{*\top}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I}\end{pmatrix}\operatorname{diag}(\mathbf{B}_i), \tag{17}$$

where the matrices $\mathbf{T}^*_{ip}$ propagate as $\mathbf{T}^*_{ac} = \mathbf{T}^*_{ab}\mathbf{T}^*_{bc}$.

A similar separation can be done with the system accelerations.

When looking at $\ddot{\mathbf{y}}_i = \frac{\partial \dot{\mathbf{y}}_i}{\partial \dot{\mathbf{q}}}\ddot{\mathbf{q}} + \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \dot{\mathbf{y}}_i}{\partial \dot{\mathbf{q}}}\right)\dot{\mathbf{q}}$ and inserting (4), one gets $\ddot{\mathbf{y}}_i = \ddot{\mathbf{y}}_{i,0} + \left(\frac{\partial \dot{\mathbf{y}}_i}{\partial \dot{\mathbf{q}}}\right)\ddot{\mathbf{q}}_u$, which can be used to rewrite (12) to $\ddot{\mathbf{q}}_i = \ddot{\mathbf{q}}_{i,0} - \ddot{\mathbf{q}}_{i,u}$ with

$$\ddot{\mathbf{q}}_{i,u} = -\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top \left[\mathbf{M}_i^*\mathbf{T}_{ip}\left(\frac{\partial \dot{\mathbf{y}}_p}{\partial \dot{\mathbf{q}}}\right)\ddot{\mathbf{q}}_u + \mathbf{h}^*_{i,u}\right]. \tag{18}$$

Extending the resulting formula for $\ddot{\mathbf{q}}_{i,u}$ to all system accelerations, one gets

$$\ddot{\mathbf{q}}_u = -\mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top\mathbf{M}_i^*)(\mathbf{F} - \mathrm{diag}(\mathbf{F}_i))\ddot{\mathbf{q}}_u - \mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)\mathbf{h}^*_{i,u}. \tag{19}$$

which can be simplified to

$$\left[\mathbf{I} + \mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top\mathbf{M}_i^*)(\mathbf{F} - \mathrm{diag}(\mathbf{F}_i))\right]\ddot{\mathbf{q}}_u = \mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)\mathbf{B}^*\mathbf{u}. \tag{20}$$

With (11), the expression $\mathbf{I} - \mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top\mathbf{M}_i^*\mathbf{F}_i)$ vanishes, leaving

$$\ddot{\mathbf{q}}_u = \left[\mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top\mathbf{M}_i^*)\mathbf{F}\right]^{-1}\mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)\mathbf{B}^*\mathbf{u} = \mathbf{\Gamma}^{-1}\mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)\mathbf{B}^*\mathbf{u} \tag{21}$$

The matrix $\mathbf{\Gamma}$ inherits its structure from $\mathbf{F}$, i.e. it is a lower triangular block matrix. Moreover, thanks to $\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top\mathbf{M}_i^*\mathbf{F}_i = \mathbf{I}$, all the block matrices on the diagonal of $\mathbf{\Gamma}$ are identity matrices, leading to a lower unitriangular structure, i.e. a lower triangular matrix with only ones as its diagonal entries. This guarantees that $\mathbf{\Gamma}$ will always be invertible.

Using this special structure, the matrix inversion can be done through the cascaded inversion of submatrices of $\mathbf{\Gamma}$. Following [10], the inverse of matrices of the shape

$$\mathbf{A}_i = \begin{pmatrix} \mathbf{A}_p & \mathbf{0} \\ \mathbf{C}_i & \mathbf{I} \end{pmatrix} \tag{22}$$

is

$$\mathbf{A}_i^{-1} = \begin{pmatrix} \mathbf{A}_p^{-1} & \mathbf{0} \\ -\mathbf{C}_i\mathbf{A}_p^{-1} & \mathbf{I} \end{pmatrix}, \tag{23}$$

as long as the diagonal block matrices are quadratic, which can be guaranteed in this case. Starting with the upper left submatrix $\mathbf{A}_1 = \mathbf{I}$ and then gradually expanding, until $\mathbf{A}_N = \mathbf{\Gamma}$. This way, the inverse can be obtained without any explicit matrix inversion, which motivates to call this approach the IIM.

The resulting matrix can then be simplified to

$$\mathbf{\Gamma}^{-1} = \mathbf{I} - \mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top\mathbf{M}_i^*\mathbf{T}_{ip})\hat{\mathbf{T}}\mathrm{diag}(\mathbf{F}_i). \tag{24}$$

The matrix $\hat{\mathbf{T}}$ can be derived as

$$\hat{\mathbf{T}} = [\ \mathbf{t}_{ij}\ ]_{i,j=1..N} \qquad \text{with} \qquad \mathbf{t}_{ij} = \begin{cases} \mathbf{I} & j = p(i) \\ \mathbf{T}^*_{p(i),j} & j \in \mathscr{P}(i) \setminus p(i) \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{25}$$

where $p(i)$ denotes the predecessor of i and $\mathscr{P}(i)$ the set of all subsystems preceding i according to the system topology.

For the system accelerations due to inputs, this gives

$$\ddot{\mathbf{q}}_u = \mathrm{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)[\mathbf{I} - \mathrm{diag}(\mathbf{M}_i^*\mathbf{T}_{ip})\hat{\mathbf{T}}\,\mathrm{diag}(\mathbf{F}_i\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)]\mathbf{B}^*\mathbf{u} \tag{26}$$

As can be seen from this formula, the only inverses remaining are those of $\mathbf{M}_{Ri}$, which were already calculated during the execution of the $O(n)$-method.

This yields the final result $\ddot{\mathbf{q}} = \ddot{\mathbf{q}}_0 + \hat{\mathbf{B}}\mathbf{u}$ with $\ddot{\mathbf{q}}_0$ obtained by running the $O(n)$-method once without any inputs. Through that, all variables that are necessary to calculate the global matrices $\hat{\mathbf{T}}, \mathbf{B}^*$ and subsequently $\hat{\mathbf{B}}$ are computed in advance. Finally, the IWIM $\hat{\mathbf{B}}$ is obtained as

$$\hat{\mathbf{B}} = \operatorname{diag}(\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)[\mathbf{I} - \operatorname{diag}(\mathbf{M}_i^*\mathbf{T}_{ip})\hat{\mathbf{T}}\operatorname{diag}(\mathbf{F}_i\mathbf{M}_{Ri}^{-1}\mathbf{F}_i^\top)]\mathbf{B}^*. \tag{27}$$

6 SIMULATION RESULTS

To get a better understanding of the performance of this new method, it is used to simulate different models with an increasing number of degrees of freedom. The runtime is then compared against the solution of the eUFM which was explained in section 4 and the $O(n)$-method as explained in section 3.2 for reference. The latter is only there as a baseline for the runtime, since as mentioned, this method is unable to explicitly calculate the IWIM.

All algorithms are implemented in Matlab and the runtime is calculated as the average time it takes to evaluate the accelerations of a model 100 times with randomized values for the initial states and inputs for each run.

6.1 Simulation: n-link Chain

The model is a n-Link pendulum with n ranging from 1 to 50. For the results shown in fig. 2 (a), only the last link was actuated, i.e. the number of inputs remained constant.

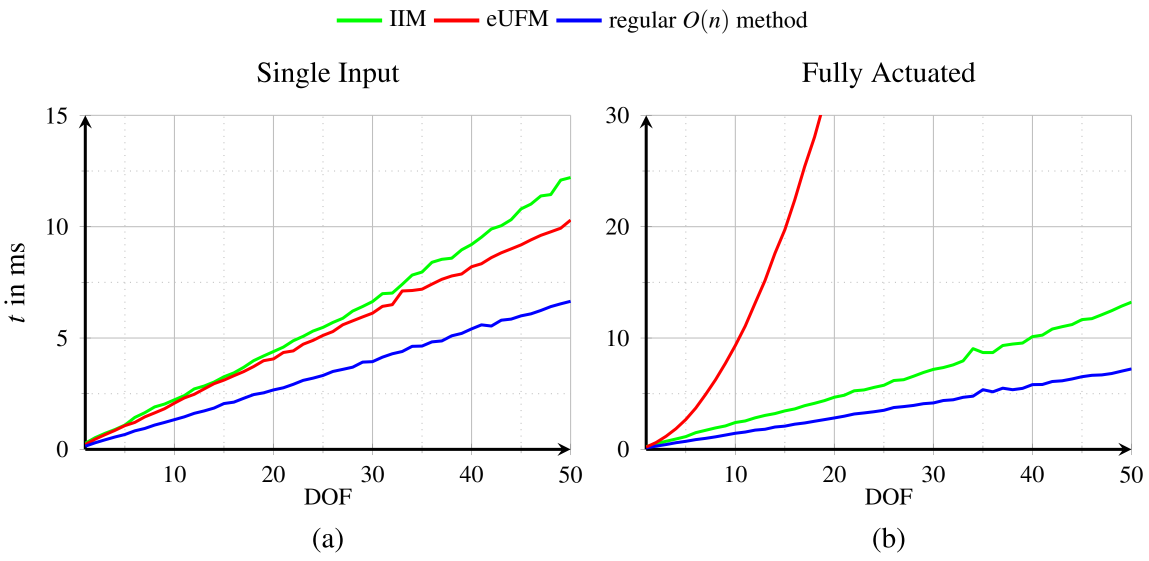

Figure 2. Run time comparisons for the forward dynamics evaluation n-link pendulum models

As can be seen here, the IIM is slightly worse than the eUFM in this case. However, having only a single input is a pathological case for any system, so the same n-link pendulum is modeled as a fully actuated system next. The results for these simulations are shown in fig. 2 (b). When looking at the results for the eUFM here, it becomes clear why this method should be avoided for systems with more inputs, the resulting computational cost becomes prohibitive.

When comparing the computational times of fig. 2 at 50 DOF, the results of which are listed in table 1, the increase stays below 10% for the IIM and the $O(n)$-method, while for the eUFM the computational time rises by $\approx 40\%$ for each additional input.

Table 1. Average computational times for a 50-link pendulum

	Single Input	Fully Actuated	
IIM	12.21	13.22	ms
eUFM	10.30	205.85	ms
$O(n)$	6.65	7.26	ms

6.2 Application example: A humanoid robot

Finally, the method is tested on a more practical system, a humanoid robot with 24 DOF and 42 inputs. The large number of inputs stems from 8 contact points that are defined for the robot, so there are 24 contact forces and 18 motor torques. Again, the accelerations are calculated for a set of 100 random states and inputs. The results of these simulations are listed in table 2 and show the same trends as were observed in the n-link pendulum. The IMM takes roughly 30% longer than the regular $O(n)$-method, while the eUFM performs much worse, as expected here due to the large number of inputs.

Table 2. Average computational times for a humanoid robot

	Humanoid	
IIM	4.13	ms
eUFM	67.20	ms
$O(n)$	2.72	ms

7 CONCLUSIONS

This paper introduces the Implicit Inversion Method, which utilizes the subsystem formulation for the EoM to effectively calculate the inertia-weighted input matrix $\mathbf{M}^{-1}\mathbf{B}$, which embeds the input variables into the system dynamics, without having knowledge of the system inertia matrix $\mathbf{M}$. It is used in conjunction with an $O(n)$-algorithm to evaluate the free system dynamics and uses intermediate results from the algorithm.

The derivation of the IWIM leads to an intermediate result for which a matrix inversion is still necessary, but the unitriangular structure of $\mathbf{\Gamma}$ guarantees invertibility. This further enables the inversion of the matrix without actually taking an inverse apart from one that was already known from the evaluation of the $O(n)$-method, hence the name of this algorithm.

To check the performance of this method, run time comparisons were made between the IIM and an extended form of the Unit Force Method. For single-input systems, the current implementation of the IMM is slower than the Unit Force Method, but for multiple inputs it becomes much faster. This point might be improved further through a more efficient implementation in the future, e.g. in C++ instead of the current Matlab version.

ACKNOWLEDGMENTS

This work has been supported by the "LCM - K2 Center for Symbiotic Mechatronics" within the framework of the Austrian COMET-K2 program.

REFERENCES

[1] Khatib, O.: A unified approach for motion and force control of robot manipulators: The operational space formulation. IEEE Journal on Robotics and Automation (1987)

[2] Wensing, P.M., Orin, D.E.: Improved Computation of the Humanoid Centroidal Dynamics and Application for Whole-Body Control. International Journal of Humanoid Robotics **13**(01) (March 2016) 1550039

[3] Wensing, P.M., Orin, D.E.: Generation of dynamic humanoid behaviors through task-space control with conic optimization. In: 2013 IEEE International Conference on Robotics and Automation. (2013)

[4] Featherstone, R.: Rigid Body Dynamics Algorithms. Springer (2008)

[5] Bremer, H.: Elastic Multibody Dynamics. Springer (2008)

[6] Gattringer, H., Müller, A., Pucher, F., Reiter, A.: O(n) algorithm for elastic link/joint robots with end-effector contact. In: IUTAM Symposium on Intelligent Multibody Systems – Dynamics, Control, Simulation. (2019)

[7] Wensing, P., Featherstone, R., Orin, D.E.: A reduced-order recursive algorithm for the computation of the operational-space inertia matrix. In: 2012 IEEE International Conference on Robotics and Automation. (2012)

[8] Lilly, K.: Efficient Dynamic Simulation of Robotic Mechanisms. Springer Science & Business Media (1993)

[9] Siciliano, B., Khatib, O.: Springer Handbook of Robotics. Volume 200. Springer-Verlag Berlin Heidelberg (2008)

[10] Lu, T.T., Shiou, S.H.: Inverses of 2×2 block matrices. Computers & Mathematics With Applications **43** (2002) 119–129