Analytical Inverse of the Joint Space Inertia Matrix

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

***Abstract*—In this short report, we briefly introduce a new algorithm to compute the analytical inverse of the joint space inertia matrix. Such inverse may be useful for several applications like the computation of the analytical derivatives of the forward dynamics. We also provide a free C++ implementation of this algorithm inside our C++ framework for rigid-body dynamics computations called Pinocchio.**

I. Content

The algorithm that we introduce now allows to compute in recursive manner (i.e. by exploiting the sparsity induced by the kinematic tree) the inverse of the joint space inertia matrix, without explicitly computing the joint space inertia matrix itself. It formally exploits the computations done in the articulated body algorithms by removing the affine terms. To the best of our knowledge, this is the first time that this algorithm is introduced and detailed.

This algorithm is composed of three passed, similarly to ABA. We use the same notations that have been used so far in our paper on the analytical derivatives [2] and which follows the ones introduced in [4]. We denote by M_{inv} the inverse of M et we use the notation $M_{\text{inv}}[i,:]$ to designate the rows of M_{inv} which correspond to the joint i. Using this notation, $M_{\text{inv}}[i, \text{subtree}(i)]$ are the columns of $M_{\text{inv}}[i,:]$ supported by the joint i including i itself. In addition, we exploit the Pythonic notation $M_{\text{inv}}[i, i\ :]$ to express all the columns of $M_{\text{inv}}[i,:]$ starting at index i until the last column of M_{inv}. We also exploit the fact that M^{-1} is symmetric.

In Algo. 1, $\boldsymbol{F}_i$ is the force-set collecting the contributions of the supporting tree rooted at i and $\boldsymbol{P}_i$ is a motion-set which contains the contributions of all the parents of joint i.

II. Implementation

This algorithm is made freely available in our open-source C++ framework called Pinocchio [3]. Pinocchio allows to compute in efficient manner the forward and inverse dynamics of poly-articulated systems such as humanoid robots, robotic manipulators, quadrupedal robots. Pinocchio is now at the hearth of many softwares developed in the Gepetto team at LAAS-CNRS for planing and control of robots [5, 1]. In addition, Pinocchio comes with a complete Python interface for easy code prototyping.

Acknowledgments

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

References

[1] Justin Carpentier and Nicolas Mansard. Multi-contact locomotion of legged robots. *Submitted to IEEE Transaction on Robotics*, 2018.
[2] Justin Carpentier and Nicolas Mansard. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and Systems*, 2018.
[3] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. Pinocchio: fast forward and inverse dynamics for poly-articulated systems, 2015–2018. URL https://stack-of-tasks.github.io/pinocchio.
[4] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2008.
[5] Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylene Campana, Nicolas Mansard, and Florent Lamiraux. HPP: A new software for constrained motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

Algorithm 1 - Pseudo code of the algorithm to directly compute the inverse of the joint space inertia matrix and which is inspired from ABA exposed by Featherstone [4, p. 132] and follows the same notations.

```
First forward pass:
for $i = 1$ to $N_B$ do
    $[\boldsymbol{X}_\mathrm{J}, \boldsymbol{S}_i] = \mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$
    ${}^{i}\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_\mathrm{J}\, \boldsymbol{X}_T(i)$
    $\boldsymbol{I}_i^A = \boldsymbol{I}_i$
end
Backward pass:
for $i = N_B$ to $1$ do
    $\boldsymbol{U}_i = \boldsymbol{I}_i^A \boldsymbol{S}_i$
    $\boldsymbol{D}_i = \boldsymbol{S}_i^T \boldsymbol{U}_i$
    $M_\mathrm{inv}[i, i] = \boldsymbol{D}_i^{-1}$
    $M_\mathrm{inv}[i, \mathrm{subtree}(i)] = M_\mathrm{inv}[i, \mathrm{subtree}(i)] - \boldsymbol{D}_i^{-T} \boldsymbol{S}_i^T \boldsymbol{F}_i[:, \mathrm{subtree}(i)]$
    if $\lambda(i) \neq 0$ then
        $\boldsymbol{F}_{\lambda(i)}[:, \mathrm{subtree}(i)] = \boldsymbol{F}_{\lambda(i)}[:, \mathrm{subtree}(i)] + {}^{\lambda(i)}\boldsymbol{X}_i^* \boldsymbol{U}_i M_\mathrm{inv}[i, \mathrm{subtree}(i)]$
        $\boldsymbol{I}_i^a = \boldsymbol{I}_i^A - \boldsymbol{U}_i \boldsymbol{D}_i^{-1} \boldsymbol{U}_i^T$
        $\boldsymbol{I}_{\lambda(i)}^A = \boldsymbol{I}_{\lambda(i)}^A + {}^{\lambda(i)}\boldsymbol{X}_i^*\, \boldsymbol{I}_i^a\, {}^{i}\boldsymbol{X}_{\lambda(i)}$
    end
end
Second forward pass:
for $i = 1$ to $N_B$ do
    if $\lambda(i) \neq 0$ then
        $M_\mathrm{inv}[i, i:] = M_\mathrm{inv}[i, i:] - \boldsymbol{D}_i^{-1} \boldsymbol{U}_i^T\; {}^{i}\boldsymbol{X}_{\lambda(i)} \boldsymbol{P}_{\lambda(i)}[i, i:]$
    end
    $\boldsymbol{P}_i[i, i:] = \boldsymbol{S}_i M_\mathrm{inv}[i, i:]$
    if $\lambda(i) \neq 0$ then
        $\boldsymbol{P}_i[i, i:] = \boldsymbol{P}_i[i, i:] + {}^{i}\boldsymbol{X}_{\lambda(i)} \boldsymbol{P}_{\lambda(i)}[i, i:]$
    end
end
```