Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

Proximal and Sparse Resolution of Constrained Dynamic Equations

Justin Carpentier
Inria, École normale supérieure
CNRS, PSL Research University
75005 Paris, France
Email: justin.carpentier@inria.fr

Rohan Budhiraja
Inria Paris
75012 Paris, France
Email: rohan.budhiraja@inria.fr

Nicolas Mansard
LAAS-CNRS, ANITI
University of Toulouse
31400 Toulouse, France
Email: nicolas.mansard@laas.fr

***Abstract*—Control of robots with kinematic constraints like loop-closure constraints or interactions with the environment requires solving the underlying constrained dynamics equations of motion. Several approaches have been proposed so far in the literature to solve these constrained optimization problems, for instance by either taking advantage in part of the sparsity of the kinematic tree or by considering an explicit formulation of the constraints in the problem resolution. Yet, not all the constraints allow an explicit formulation and in general, approaches of the state of the art suffer from singularity issues, especially in the context of redundant or singular constraints. In this paper, we propose a unified approach to solve forward dynamics equations involving constraints in an efficient, generic and robust manner. To this aim, we first (i) propose a proximal formulation of the constrained dynamics which converges to an optimal solution in the least-square sense even in the presence of singularities. Based on this proximal formulation, we introduce (ii) a sparse Cholesky factorization of the underlying Karush–Kuhn–Tucker matrix related to the constrained dynamics, which exploits at best the sparsity of the kinematic structure of the robot. We also show (iii) that it is possible to extract from this factorization the Cholesky decomposition associated to the so-called Operational Space Inertia Matrix, inherent to task-based control frameworks or physic simulations. These new formulation and factorization, implemented within the Pinocchio library, are benchmark on various robotic platforms, ranging from classic robotic arms or quadrupeds to humanoid robots with closed kinematic chains, and show how they significantly outperform alternative solutions of the state of the art by a factor 2 or more.**

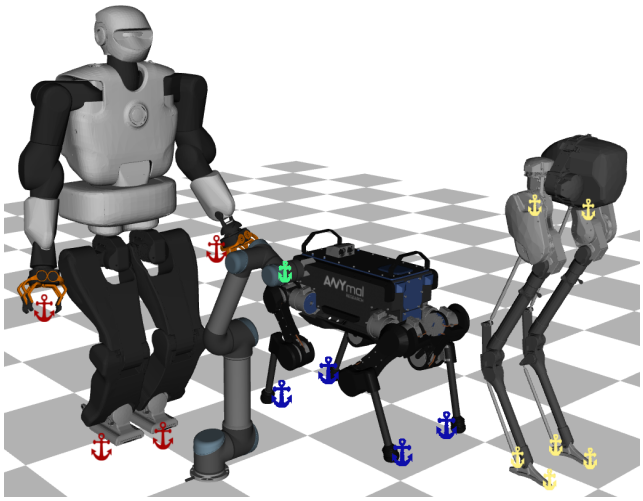


Fig. 1. Robotic systems may be subject to different types of constraints: point contact constraints (quadrupeds), flat foot constraints (humanoids), closed kinematic chains (parallel robots, here the 4-bar linkages of Cassie) or even contact with the end effectors (any robot). Each colored "anchor" here shows a possible kinematic constraint applied on the dynamics of the robot. In this paper, we introduce a generic approach to handle all these types of constraints, contacts and kinematic closures, in a unified and efficient manner, even in the context of ill-posed or singular cases.

I. Introduction

As soon as a robot makes contacts with the world or is endowed with loop closures in its design, its dynamics is governed by the constrained equations of motion. From a phenomenological point of view, these equations of motion follow the so-called least-action principle, also known under the name of the Maupertuis principle which dates back to the 17^{th} century. This principle states that the motion of the system follows the closest possible acceleration to the free-falling acceleration (in the sense of the kinetic metric) which fulfils the constraints. In other words, solving the constrained equations of motion boils down to solving a constrained optimization problem where forces acts as the Lagrange multipliers of the motion constraints.

This principle has been exploited by our community since the seminal work of Barraf [1], which is here our main source of inspiration. He initially proposed to formulate the dynamics with maximal coordinates (i.e. each rigid body is represented by its 6 coordinates of motion) as a sparse constrained optimization problem, and proposed an algorithm to solve it in linear time. While maximal coordinates are interesting for their versatility and largely used in simulation [2], working directly in the configuration space with generalized coordinates presents several advantages [16] that we propose to exploit in this paper.

Some constraints can be put under an explicit form, i.e. there exists a reduced parametrization of the configuration that is free of constraints. This is often the case for classical kinematic closures [37, 16]. Yet explicit formulation is not always possible, and in particular is not possible for the common case of contact constraints [42]. We address here the more generic case where the constraints are written under an implicit form i.e. the configuration should nullify a set of equations, which makes it possible to handle any kind of

design or contact constraints, or both together (see Fig. 1).

While recursive formulation exists and are predominant for unconstrained dynamics (both inverse [40, 35] and forward [13]) and for decomposing and inverting the joint-space inertia matrix (JSIM) [14], little effort seems to have been put in formulating a recursive algorithm in reduced coordinates for the constrained dynamics. Yet the constrained system is directly inverted in several trajectory-optimization implementation. In [6], the constrained dynamics is explicitly solved in the context of trajectory optimization. In [24], a hierarchical trajectory optimization is also inverting directly the constrained system with real-time applications. In [33], a similar approach is used and extended to account for constrained impact dynamics and switching constraints.

In [1], the links between the constrained optimization formulation and the so-called operational-space inertia matrix (OSIM) are also exhibited. OSIM is essential in many formulation of inverse-dynamics based robot control [28, 42, 38]. Several algorithms have proposed recursive formulations to compute this matrix efficiently [34, 45].

In this paper, we use the connection between constrained dynamics and OSIM to propose a recursive algorithm for tackling both simultaneously and in an efficient way. We extend the existing sparse Cholesky decomposition of the JSIM [16] to compute together the decomposition of the OSIM and the decomposition of the "KKT" linear system arising from the constrained dynamics. We then take advantage of classical methods of constrained optimization to best solve this linear system. In particular, we explain how proximal optimization is suitable in this case, both for its ability to handle ill-conditioned problems (which arise near singularity or in the case of redundant constraints) and its efficiency to converge to accurate situation.

The remainder of this paper is organized as follows. In Sec. II, we first introduce the notations and the standard approaches for solving constrained dynamics problem in robotics as well as the notions of proximal operators, at the core of the proposed approach. We then present in Sec. III the proximal reformulation of the constrained dynamics problem. Based on this proximal reformulation, we show in Sec. IV how the branch-induced sparsity of the robot kinematic structure can be exploited. This sparsity can be used to sensibly reduce the overall computational complexity for solving constrained dynamics problems. We benchmark the proposed approach in Sec. V and showcase its application on standard robotics problems. Sec. VI concludes the paper.

II. Background

In this section, we first review the concepts of constrained dynamics and proximal algorithms which are at the core of the contributions and introduce the main notations subsequently employed. In this paper, we consider poly-articulated rigid body systems, i.e. a collection of rigid bodies (also named links) connected by joints and which form a kinematic structure subject to constraints. These constraints can be of various nature: closed loop constraints (enforced by the presence of joints which constrain the relative motion of two independent branches of the kinematic tree), contact constrains (occurring when a robot makes a contact with another robot or with the surrounding environment), etc.

A. Constrained dynamics

Lagrangian dynamics. While the dynamics of rigid bodies is well described when written in maximal coordinates [1], i.e. by considering the position and orientation of each body and explicitly enforcing the joint constraints, Lagrangian dynamics [30] is usually preferred in robotics. It provides a compact (minimal) formulation of the equations of motion and efficient algorithms in generalized coordinates. Such algorithms are also called rigid-body dynamics algorithms [16] in the literature, and have been introduced in order to exploit at best the sparsity of both the system kinematics and the underlying spatial operations. For a given poly-articulated system, the Lagrangian dynamics reads as follows:

$$M(\boldsymbol{q})\dot{\boldsymbol{v}} + \underbrace{c(\boldsymbol{q},\boldsymbol{v}) + g(\boldsymbol{q})}_{b(\boldsymbol{q},\boldsymbol{v})} = \boldsymbol{\tau}, \tag{1}$$

where $\boldsymbol{q} \in \mathcal{Q} \simeq \mathbb{R}^{n_q}$, $\boldsymbol{v} \in \mathcal{T}_q Q \simeq \mathbb{R}^{n_v}$ and $\boldsymbol{\tau} \in \mathcal{T}_q^* Q \simeq \mathbb{R}^{n_v}$ are respectively the joint configuration vector, the joint velocity vector and the joint torque vector, while $\mathcal{Q}$ is the joint configuration space, $\mathcal{T}_q Q$ and $\mathcal{T}_q^* Q$ are the tangent and the dual tangent spaces of $\mathcal{Q}$ respectively[1]. $\dot{\boldsymbol{v}}$ is the time derivative of $\boldsymbol{v}$ and corresponds to the joint acceleration vector. $M(\boldsymbol{q})$ stands for the joint space inertia matrix (JSIM), $c(\boldsymbol{q},\boldsymbol{v})$ denotes the Coriolis and centrifugal effects and $g(\boldsymbol{q})$ is the generalized gravity vector. We will drop the dependencies to $\boldsymbol{q}$ and $\boldsymbol{v}$ in the rest of the paper for brevity.

Constrained dynamics. As we discussed in the introduction, the system dynamics obeys the least-action principle when subject to constraints. This stipulates that the acceleration of the constrained system is the closest to the free acceleration of the system according to the kinetic metric, while satisfying the constraints. Mathematically, this principle reads:

$$\min_{\dot{\boldsymbol{v}}} \quad \frac{1}{2}\|\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_{\text{free}}(\boldsymbol{q},\boldsymbol{v},\boldsymbol{\tau})\|_M^2 \tag{2a}$$

$$\text{subject to} \quad f_c(\boldsymbol{q}) = 0, \tag{2b}$$

where $\dot{\boldsymbol{v}}_{\text{free}}(\boldsymbol{q},\boldsymbol{v},\boldsymbol{\tau}) \stackrel{\text{def}}{=} M^{-1}(\boldsymbol{\tau} - b)$ is the free acceleration of the system without constraint, $\|x\|_M = \sqrt{x^t M x}$ is the kinetic metric and $f_c : \mathcal{Q} \to \mathbb{R}^m$ is the implicit constraint function of dimension m. In order to solve (2) in $\dot{\boldsymbol{v}}$, one needs to derive (2b) twice with respect to time in order to make the

[1] It is important to notice that joint configuration, velocity and torque vectors may be of different dimensions, depending on the nature of the joints. For instance, in the presence of a free-floating joint, its configuration vector is typically represented by the stack of one 3d vector for the translation and one 4d vector representing the orientation as a quaternion ($n_q = 7$), while the joint velocity corresponds to the concatenation of a linear velocity and an angular velocity vectors ($n_v = 6$).

constraint explicitly dependant of $\dot{\boldsymbol{v}}$, which gives us:

$$\min_{\dot{\boldsymbol{v}}} \quad \frac{1}{2}\|\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_{\text{free}}\|_M^2 \tag{3a}$$

$$\text{subject to} \quad J_{f_c}(\boldsymbol{q})\dot{\boldsymbol{v}} + \underbrace{\dot{J}_{f_c}(\boldsymbol{q},\boldsymbol{v})\boldsymbol{v}}_{\gamma_{f_c}(\boldsymbol{q},\boldsymbol{v})} = \boldsymbol{a}_c^*, \tag{3b}$$

where $J_{f_c}(\boldsymbol{q}) \in \mathbb{R}^{m\times n_v}$ is the Jacobian of the constraint f_c and $\gamma_{f_c}(\boldsymbol{q},\boldsymbol{v})$ is the acceleration drift. We also introduce the term $\boldsymbol{a}_c^*$ in (3b) to account for a desired constraint acceleration, which might include for instance some corrective terms when the constraint (2b) is not fully satisfied (for example, the Baumgarte stabilization method [18].) It is worth mentioning at this stage that the proposed analysis also generalises to non-holonomic constraints, i.e. constraints which also exhibit a dependency in $\boldsymbol{v}$.

Lagrangian of the constrained dynamics. Problem (3) corresponds to an equality-constrained quadratic program. The associated Lagrangian equation is:

$$L(\dot{\boldsymbol{v}},\boldsymbol{\lambda}) = \frac{1}{2}\|\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_{\text{free}}\|_M^2 + \boldsymbol{\lambda}^t\left(J_{f_c}\dot{\boldsymbol{v}} + \boldsymbol{\gamma}_{f_c} - \boldsymbol{a}_c^*\right), \tag{4}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ corresponds to the Lagrange multipliers associated to the constraint (3b). The solution of the equality-constrained problem (3) is given by the following minmax problem:

$$(\dot{\boldsymbol{v}}^*,\boldsymbol{\lambda}^*) = \arg\min_{\dot{\boldsymbol{v}}}\max_{\boldsymbol{\lambda}} L(\dot{\boldsymbol{v}},\boldsymbol{\lambda}). \tag{5}$$

Its solution can be derived from the so-called Lagrange optimality conditions, corresponding to the notion of saddle points [3] where the gradients $\nabla_{\dot{\boldsymbol{v}}}L$ and $\nabla_{\boldsymbol{\lambda}}L$ should be both equal to zero. The solution of these first-order necessary conditions are given by the system of equations:

$$\underbrace{\begin{bmatrix}\mathbf{0}_{m\times m} & J_{f_c}\\ J_{f_c}^t & M\end{bmatrix}}_{K(\boldsymbol{q})}\begin{bmatrix}\boldsymbol{\lambda}\\ \dot{\boldsymbol{v}}\end{bmatrix} = \begin{bmatrix}-\gamma_{f_c} + a_c^*\\ M\dot{\boldsymbol{v}}_{\text{free}}\end{bmatrix}, \tag{6}$$

where $K(\boldsymbol{q})$ is the so-called Karush–Kuhn–Tucker (KKT) matrix. The blocks of the left-hand side are ordered in the opposite way to what can be found in the literature. This particular choice is going to be key for the decomposition we are proposing in Sec. IV. Assuming that M is always positive definite – as is the case for all physical systems, due to their strictly positive mass distribution – this system of equations (6) may have zero, one or an infinite number of solutions, depending on the rank of the constraint Jacobian J_{f_c} and whether $-\gamma_{f_c} + a_c^*$ lies in the range-space of J_{f_c}. For symmetric positive definite M, the solutions of (6) are given by:

$$\dot{\boldsymbol{v}} = \dot{\boldsymbol{v}}_{\text{free}} - M^{-1}J_{f_c}^t\boldsymbol{\lambda}, \tag{7}$$

and:

$$\underbrace{J_{f_c}M^{-1}J_{f_c}^t}_{\Lambda^{-1}(\boldsymbol{q})}\boldsymbol{\lambda} = \underbrace{J_{f_c}\dot{\boldsymbol{v}}_{\text{free}} + \gamma_{f_c}}_{a_{c,\text{free}}(\boldsymbol{q},\boldsymbol{v},\boldsymbol{\tau})} - a_c^*, \tag{8}$$

where $a_{c,\text{free}}(\boldsymbol{q},\boldsymbol{v},\boldsymbol{\tau})$ is the acceleration of the free dynamics in the space of the constraint and $\Lambda^{-1}(\boldsymbol{q})$ is the so-called inverse operational space inertia matrix (OSIM) [28], also named in other communities the Delassus matrix [12].

Standard resolution of saddle-point problems. When J_{f_c} is rank deficient, which is the case in the context of redundant contacts or when a robot reaches a kinematic singularity, $K(\boldsymbol{q})$ does not admit an inverse and the problem is said to be ill-posed. Advanced numerical methods such as null-space projection, Schur complement, iterative or Krylov subspace methods [3] are then needed to approximately solve (6). Yet, while being very generic, these common strategies cannot exploit the underlying sparsity of the robot kinematics and may require a large number of iterations before converging to a feasible solution for a given tolerance. In Sec. III, we instead propose a proximal reformulation of (4) which is both efficient and robust to singularity or rank deficiency issues of the constraint Jacobian. This proximal reformulation enables us to introduce a relaxed version of (6) which is guaranteed [41] to converge to the optimal solution (if it exists) within a few iterations.

In [15], Featherstone proposed to leverage the sparse Cholesky factorization of the joint space inertia matrix [14] to solve (3). By writing $M = UU^t$, (with U a sparse upper triangular matrix with a sparsity structure induced by the kinematic tree), the overall idea is to sparsely compute Λ^{-1} as the product of two rectangular matrices A^tA with $A = U^{-1}J_{f_c}^t \in \mathbb{R}^{n_v\times m}$. He then exploits a dense Cholesky factorization of Λ^{-1} to solve (8) and finally retrieve $\dot{\boldsymbol{v}}$ from (7). While this approach is generic and efficient, it does not handle the aforementioned singular or redundant cases. To overcome these limitations, based on the proximal resolution of the constrained dynamics, we introduce in Sec. IV an efficient algorithm to compute the sparse Cholesky factorization of the KKT matrix K which efficiently exploits the sparsity induced by the kinematic tree. We also show how this decomposition is connected to the Cholesky factorization of Λ^{-1}.

B. Proximal algorithms

In convex optimization, proximal algorithms [41] are a broad class of optimization techniques to find the optimum of convex functions, potentially non-smooth.

Proximal operators. Proximal algorithms are naturally built on the notion of proximal operators, defined for a proper and lower semi-continuous convex function $f : \mathcal{X} \to \mathbb{R} \cup [-\infty;\ +\infty]$ as:

$$\text{prox}_{f,\alpha}(y) \stackrel{\text{def}}{=} \arg\min_{x\in\mathcal{X}} f(x) + \frac{1}{2\alpha}\|x-y\|_2^2, \tag{9}$$

where $\alpha \in \mathbb{R}^{+*}$ can be assimilated to a step-size. In terms of interpretation, the additional term $\frac{1}{2\alpha}\|x-y\|_2^2$ acts as a smoothing of the function f, transforming the initial minimization problem over f into a strongly convex problem. In addition, many functions (typically norm functions) admit closed-form solutions for their proximal operator, while directly solving $\arg\min_x f(x)$ is potentially hard or even impossible. Proximal algorithms typically iterate over the proximal operators,

following the recursion:

$$x_{k+1} = \text{prox}_{f,\alpha}(x_k), \tag{10}$$

until a given convergence criteria or a certain number of iterations are reached. In general, this results in a cascade of simpler problems to solve, at the price of possibly requiring a large number of iterations before converging to the solution of the original problem with a desired precision.

Application to linear least-square problems. A typical application of proximal algorithms is for linear least-square regressions, when one seeks $x \in \mathbb{R}^n$ which best explains the system of equations $Ax = b$ in the least-square sense, where $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$ is possibly rank deficient:

$$x^* = \arg\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|_2^2 \tag{11}$$

where $f(x) = \frac{1}{2}\|Ax - b\|_2^2$. A standard approach for computing the least-square solution goes through the calculation of the generalized inverse of A, denoted $A^\dagger$, resulting in: $x^* = A^\dagger b$. The computation of this generalized inverse in turn requires to manually select a certain threshold associated with the smallest singular value of A. This impacts the quality of the solution, especially in the context of ill-posed problems. Instead, proximal algorithms do not require such choice of threshold and iterate over an always well-posed problem and converges to the optimal solution of the original problem [41]. The proximal operator associated to the least-square problem reads:

$$\text{prox}_{f,\alpha}(x_k) = \arg\min_x \frac{1}{2}\|Ax - b\|_2^2 + \frac{1}{2\alpha}\|x - x_k\|_2^2, \tag{12}$$

and admits an analytical solution of the form:

$$x_{k+1} = \text{prox}_{f,\alpha}(x_k) = \left(A^t A + \frac{1}{\alpha} I_n\right)^{-1} \left(A^t b + \frac{1}{\alpha} x_k\right), \tag{13}$$

where I_n is the identity matrix of dimension n. It appears then for $\alpha > 0$, the matrix $A^t A + \frac{1}{\alpha} I_n$ is always symmetric and positive definite. The iterative procedure only requires one matrix inversion or factorization (typically, a Cholesky decomposition), and one can show that it converges to the optimal solution at a linear rate with respect to α [41]. A typical stopping criteria is given by $\|A^t (Ax_k - b)\|_\infty < \epsilon$, where ϵ corresponds to a desired accuracy in terms of optimality of the current solution x_k. In practice, for most classical optimization problems in robotics, only a dozen of iterations are really needed to find an optimal solution with sufficient accuracy, a typical value for ϵ being 10^{-8}. In addition, proximal algorithms offer the ability of being easily warm-started, and in the context of linear problems, they can be assimilated to an iterative-refinement procedure. It is also worth noticing that, in the case where $x_k = \mathbf{0}_n$, Eq. (12) reduces to the so-called Tikhonov regularization method.

III. Proximal Resolution of Constrained Dynamic Equations

In this section, we show how the application of the proximal operator over the dual variables of the constrained dynamics Lagrangian (4) lead to a well-posed iterative optimization process, converging to the optimal solution of (3), even in the presence of singular or redundant constraints, and show how it differs from the classic augmented Lagrangian method.

A. The augmented Lagrangian approach

A classic approach for solving equality-constrained quadratic program is the augmented Lagrangian method (ALM) [43]. ALM consists of augmenting the Lagrangian function (4) with a L2 penalization of the linear constraint (3b). In the context of the constrained dynamics (3), this augmentation reads:

$$L_A(\dot{\boldsymbol{v}}, \boldsymbol{\lambda}) = L(\dot{\boldsymbol{v}}, \boldsymbol{\lambda}) + \frac{\mu}{2}\|J_{f_c}\dot{\boldsymbol{v}} + \gamma_{f_c} - a_c^*\|_2^2, \tag{14}$$

where μ is a positive penalty parameter. In order to find the saddle-point problem associated to (14), ALM alternates between the minimization of $L_A(\dot{\boldsymbol{v}}, \boldsymbol{\lambda})$ with respect to the primal variable $\dot{\boldsymbol{v}}$:

$$\begin{aligned}\dot{\boldsymbol{v}}_{k+1} = \left(M + \mu J_{f_c}^t J_{f_c}\right)^{-1} \big(&M\dot{\boldsymbol{v}}_{\text{free}} \\ &- J_{f_c}^t \left(\boldsymbol{\lambda}_k + \mu(\gamma_{f_c} - a_c^*)\right)\big)\end{aligned} \tag{15}$$

and the classic rule to update the multipliers:

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \mu\left(J_{f_c}\dot{\boldsymbol{v}}_{k+1} + \gamma_{f_c} - a_c^*\right). \tag{16}$$

This optimization scheme has been proven to converge with a rate of convergence directly related to the value of μ [4]. In the context of equality-constrained quadratic program, the rate of converge is linear in μ, also meaning that large values of μ will decrease the actual number of iterations needed to reach a desired precision. Yet, in the context of classic ALM, the value of μ also affects the conditioning of $M + \mu J_{f_c}^t J_{f_c}$, which is also impacted by the condition number of $J_{f_c}^t J_{f_c}$, being potentially high in the context of singular or over-constrained problems. From a numerical point of view, this behavior will limit the applicability of the approach for these aforementioned cases.

B. The proximal Lagrangian

An alternative solution to ALM consists of using a proximal reformulation of the minmax problem associated with the constrained problem Lagrangian (4), leading to:

$$\text{prox}_{L,\mu^{-1}}(\dot{\boldsymbol{v}}_k, \boldsymbol{\lambda}_k) = \arg\min_{\dot{\boldsymbol{v}}} \max_{\boldsymbol{\lambda}} L(\dot{\boldsymbol{v}}, \boldsymbol{\lambda}) - \frac{\mu}{2}\|\boldsymbol{\lambda} - \boldsymbol{\lambda}_k\|_2^2, \tag{17}$$

where $\boldsymbol{\lambda}_k$ is the current estimate of the multipliers[2]. We could have also introduced a proximal term associated to the primal variable $\dot{\boldsymbol{v}}$, but this is not required, as M is already a strictly symmetric positive-definite matrix, hence $\frac{1}{2}\|\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_{\text{free}}\|_M^2$ is strictly convex. Solving the saddle-point problem associated with (17) leads to the primal/dual system of equations:

$$\underbrace{\begin{bmatrix} -\mu I_m & J_{f_c} \\ J_{f_c}^t & M \end{bmatrix}}_{K_\mu(\boldsymbol{q})} \begin{bmatrix} \boldsymbol{\lambda} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} -\gamma_{f_c} + a_c^* - \mu\boldsymbol{\lambda}_k \\ M\dot{\boldsymbol{v}}_{\text{free}} \end{bmatrix}. \tag{18}$$

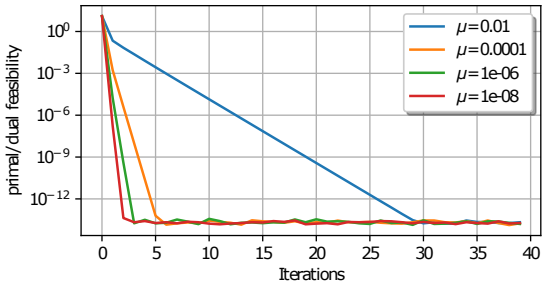


Fig. 2. Converge rate of the primal-dual proximal formulation for different values of μ for typical equality-constrained quadratic program. The convergence rate is linear in μ as suggested by the theory [41] and the convergence rate is inversely proportional to the value of μ.

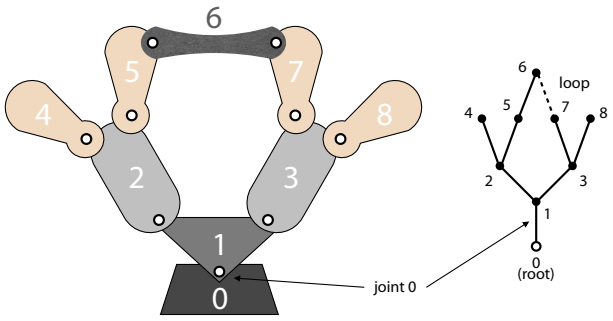


Fig. 3. Illustration of kinematic chain equipped with a closed loop. The whole kinematic chain can be represented by a tree where bodies correspond to nodes and the joints to the arcs connections two nodes. In this work, the closed-loop constraints are considered as implicit constraints and are solved using the formulation proposed in Sec. III.

Contrary to (6), here $K_\mu(\boldsymbol{q})$ is always nonsingular, thanks to the presence of the nondegenerated top-left corner block. It should be noted that this is similar to the case of the aforementioned linear least-square proximal resolution. In turn, the regularized KKT matrix $K_\mu(\boldsymbol{q})$ makes the problem (18) always well-posed, defined by a unique solution. In addition, we recover in $K_\mu(\boldsymbol{q})$ the Hessians associated to the saddle-point of the constrained optimization problem (3). M, which is strictly positive-definite, is related to the primal variables and $-\mu I_m$, which is strictly negative-definite for $\mu > 0$, is related to the dual variables. It is also worth mentioning that, when μ goes to 0, one retrieves the original KKT problem (6), likely ill-posed.

In the end, iterating over (17) with a constant penalty factor $\mu > 0$, by following the update rule $(\dot{\boldsymbol{v}}_{k+1}, \boldsymbol{\lambda}_{k+1}) = \text{prox}_{L,\mu^{-1}}(\dot{\boldsymbol{v}}_k, \boldsymbol{\lambda}_k)$, converges to the optimal solution of (3). In practice, the proposed approach only requires very few iterations to converge to a tolerance closed to the machine precision [41], as seen in Fig. 2. Indeed, and contrary to the classic augmented Lagrangian approach previously described, the conditioning of K_μ is in overall improved, enforcing the numerical stability of the proposed approach, at the price of solving a system of equations of larger dimension. Besides, while the inverse of K_μ exhibits an analytical formula of the form:

$$K_\mu^{-1}(\boldsymbol{q}) = \begin{bmatrix} -\Lambda_\mu & \Lambda_\mu J_{f_c} M^{-1} \\ M^{-1} J_{f_c}^t \Lambda_\mu & M^{-1} - M^{-1} J_{f_c}^t \Lambda_\mu J_{f_c} M^{-1} \end{bmatrix}, \tag{19}$$

with:

$$\Lambda_\mu(\boldsymbol{q}) \stackrel{\text{def}}{=} \left(J_{f_c} M^{-1} J_{f_c}^t + \mu I_m\right)^{-1} \tag{20}$$

corresponding to the damped OSIM, it is costly to evaluate in general. Indeed, it requires several matrix inversions: one to compute the damped OSIM matrix Λ_μ, and another one to get the the joint space inertia matrix inverse M^{-1}, even though efficient algorithms have been proposed so far [25, 7]. To overcome these computational limitations, we propose to leverage the branch-induced sparsity of K_μ by deriving in the next section new rigid body dynamic algorithms to compute its sparse Cholesky decomposition and perform the related sparse matrix operations.

[2]The '$-$' term appearing in front of $\|\boldsymbol{\lambda} - \boldsymbol{\lambda}_k\|_2^2$ comes from the maximization of the Lagrangian (4) with respect to the dual variable $\boldsymbol{\lambda}$.

IV. Sparse Cholesky Factorization

Rigid body dynamics algorithms [16] is popular in robotics for their ability to exploit *at best* the branch-induced sparsity of robot kinematic trees, and their simplicity in terms of software implementation. This has resulted in many efficient and open-source frameworks rooted on these algorithms [39, 17, 10, 29]. The exploitation of sparsity is particularly interesting and important when computing the JSIM, with many efficient algorithms to compute it [16], among which, the composite rigid body algorithm (CRBA) appears to be the one with the lowest computational complexity.

Additionally, some contexts may require the computation of the JSIM inverse M^{-1} in addition to the value of the JSIM M itself (like task-space inverse dynamics or robot simulation to name a few). In [14], Featherstone suggested to leverage the already computed JSIM in order to retrieve an expression of its inverse through its Cholesky factorization. He notably introduced several algorithms to compute this decomposition and to perform the related matrix-vector operations, while exploiting at best the induced sparsity. In this section, we extend the Featherstone algorithms and introduce a generic approach to compute the Cholesky decomposition of K_μ and to perform the related matrix-vector operations.

The rest of this section is organized as follows. We first introduce the main notations for the rigid body dynamics algorithms and recall the Featherstone algorithms to decompose the JSIM. We then introduce the Cholesky decomposition of K_μ and present the related algorithms. We conclude this section by showing the connection of the newly introduced decomposition and discuss potential generalization of the proposed approach.

A. Kinematic structure and notations

Whether a robot holds or not loop-constraints in its mechanism, its joint structure can always be depicted by an open kinematic tree, on top of which loop constraints apply if exist. Fig. 1 depicts robots with different kinematic structures, where

bodies are represented by nodes and the joints correspond to the arcs connecting two nodes, as illustrated in Fig. 3. The connectivity of this kinematic tree can be described by a set p of positive indexes which relates a body index i to the index of its body parent $p(i)$ inside the kinematic tree. The indexes in p follow a regular numbering scheme, meaning that $\forall\, i, 0 \leq p(i) < i$, the 0 index referring to the root body. In the case where the robot is made of a unique serial chain, we have the special property that $p(i) = i - 1$. In addition, we denote by $s(i)$ the set of of bodies supported by the body i (aka the descendant of i), including the index of body i itself.

In the case where the kinematic tree contains joints with several DoFs, one needs to consider an extended set of indexes for both parent set p and support sets $s(i)$ used by the matrix factorization algorithms, as discussed in details in [14, 16]. Hereafter, in order to simplify the explanations and without loss of generality, we assume that the kinematic tree is only composed of single DoF joints. We consider a kinematic tree made of N bodies and joints, and the loop joints are considered as constraints and handled as geometric constraints as proposed in Sec. III. In particular, for single DoF joints, we have $N = n_v$. The implementation of our method, reported in Sec. V, is done with the more general case enabling multi-DOF joints.

B. Sparse Cholesky factorization of the JSIM

The Cholesky factorization of the JSIM is given by $M = UDU^t$, with U an upper triangular matrix with ones on its main diagonal and D a diagonal matrix made of positive elements. From this decomposition, one can retrieve an expression of the JSIM inverse as $M^{-1} = U^{-t}D^{-1}U^{-1}$. To compute U, it is worthwhile to analyze first the sparsity pattern of the JSIM for a branched kinematic tree. The matrix entries of the JSIM are given by the following formula:

$$M_{i,j} = \begin{cases} S_i^t I_i^c S_j & \text{if } j \in s(i) \\ S_j^t I_j^c S_i & \text{if } i \in s(j) \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

where $S_i \in \mathbb{R}^6$ is the joint motion axis and I_i^c is the composite rigid body inertia associated to the joint i. We refer to [16] for further details. Following (21), it appears that $M_{i,j} = 0$ whenever i and j are located on different branches, which translates to:

$$i \notin s(j) \text{ and } j \notin s(i) \Rightarrow M_{i,j} = M_{j,i} = 0. \tag{22}$$

Additionally, as shown in [14], U exhibits the same sparsity pattern as M, which is fundamental for efficient resolution of multi-branched systems like humanoid robots. Another way to write it down is that:

$$i \notin s(j) \Rightarrow U_{j,i} = 0. \tag{23}$$

As observed in [14], it is important to seek for an upper decomposition of the form $M = UDU^t$, and not for a (more classical) lower Cholesky decomposition $M = LDL^t$, with L and U respectively unitary lower and upper triangular matrices, and D positive diagonal matrix. Indeed and following the remarks made in [14], shaped like that, L will not exhibit any specific pattern resulting in a dense matrix. It is also worth mentioning that M^{-1}, U^{-1}, U^{-t} or D^{-1} should not be interpreted as matrix transpose or inverse, but rather as mathematical operators acting on vectors. This is precisely the motivation behind the now well-established rigid body dynamics algorithms introduce in [14], that we propose to extend in the context of constrained dynamics.

C. Sparse Cholesky factorization of K_μ

We highlight in this subsection the structure of the Cholesky decomposition of K_μ. For that purpose, we first decompose K_μ as the product of three related and invertible matrices as follows:

$$K_\mu = \begin{bmatrix} I_m & J_{f_c} M^{-1} \\ 0 & I_n \end{bmatrix} \begin{bmatrix} -\Lambda_\mu^{-1} & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} I_m & 0 \\ M^{-1} J_{f_c}^t & I_n \end{bmatrix}.$$

We denote by $(U_{\Lambda_\mu^{-1}}, D_{\Lambda_\mu^{-1}})$ the Cholesky decomposition associated to Λ_μ^{-1} and rewrite $(U_{\Lambda_M}, D_{\Lambda_M})$ the Cholesky decomposition associated to M, such that $\Lambda_\mu^{-1} = U_{\Lambda_\mu^{-1}} D_{\Lambda_\mu^{-1}} U_{\Lambda_\mu^{-1}}^t$ and $M = U_{\Lambda_M} D_{\Lambda_M} U_{\Lambda_M}^t$. It is now possible to write K_μ as the product of two triangular matrices with one diagonal matrix, following the expression:

$$K_\mu = \overbrace{\begin{bmatrix} U_{\Lambda_\mu^{-1}} & J_{f_c} U_{\Lambda_M}^{-t} D_{\Lambda_M}^{-1} \\ 0 & U_{\Lambda_M} \end{bmatrix}}^{U_{K_\mu}} \overbrace{\begin{bmatrix} -D_{\Lambda_\mu^{-1}} & 0 \\ 0 & D_{\Lambda_M} \end{bmatrix}}^{D_{K_\mu}} \underbrace{\begin{bmatrix} U_{\Lambda_\mu^{-1}}^t & 0 \\ D_{\Lambda_M}^{-1} U_{\Lambda_M}^{-1} J_{f_c}^t & U_{\Lambda_M}^t \end{bmatrix}}_{U_{K_\mu}^t}, \tag{24}$$

expression which corresponds to nothing more than the Cholesky decomposition of K_μ, where by construction, U_{K_μ} is an upper triangular matrix of dimension $N + m$ and D_{K_μ} is a diagonal matrix of dimension $N + m$. We recognize in D_{K_μ} the negative curvature in the left top block and the positive curvature in the right bottom block, both related to the aforementioned notion of saddle-point exposed in Sec. III. This also means that it would have not been possible to use a standard Cholesky decomposition of the form U^tU, due to the presence of negative values on the diagonal matrix D_{K_μ}. Importantly, the presence of the regularization term μ in the expression of Λ_μ makes the Cholesky decomposition always well defined.

In this work, we extend the sparse algorithms introduced in [14] to the more generic setting of constrained dynamics. We assume that each constraint $c_i(\boldsymbol{q})$ only depends on a subset σ_i of the joints composing the kinematic tree, which covers, but not limited, bilateral contact and closed-loop constraint types. This notably implies that the Jacobian of each constraint $c_i(\boldsymbol{q})$, denoted by J_i, has non-zero elements only for the columns which are related to the joints present in σ_i. This statement also reads as:

$$j \notin \sigma_i \Rightarrow J_i[:, j] = \mathbf{0}_{n_i}, \tag{25}$$

where n_i is the size of the constraint c_i and $J_i[:, j]$ denotes the j^{th} column of the Jacobian matrix J_i. In particular, the total size of the constraint vector is given by $m = \sum_i n_i$.

Similarly to the Cholesky decomposition of the JSIM alone, the Cholesky decomposition of the KKT matrix K_μ exhibits a structured sparsity pattern associated to the JSIM. This reads:

$$i \notin s(j) \Rightarrow U_{K_\mu}[j+m, i+m] = 0, \tag{26}$$

where we need to add to the joint indexes the total size of the constraints to correctly capture the dimensions of K_μ. In addition, according to the properties (23) and (25), K_μ also exhibits a structure sparsity pattern associated to the constraints. If we denote by r_i the indexes of the rows associated to the constraint c_i in U_{K_μ}, this pattern follows the rule:

$$j \notin \sigma_i \Rightarrow U_{K_\mu}[r_i, j+m] = \mathbf{0}_{n_i}. \tag{27}$$

In other words, if a joint does not contribute to a constraint c_i, then the related block in U_{K_μ} will be zero. Finally, the only non-structured part of U_{K_μ} is its upper left triangular block of dimension $m \times m$, totally dense in most standard cases [1].

D. Rigid body algorithms associated to (U_{K_μ}, D_{K_μ})

Rooted on the previous analysis, Alg. 1 summarizes the main rigid body dynamic algorithm to compute the Cholesky factorization of the KKT matrix $K_\mu = U_{K_\mu} D_{K_\mu} U_{K_\mu}^t$. The proposed algorithm can be performed in-place, meaning that they will directly operate on the entries of the KKT matrix without requiring further vector or matrix copy operations. For clarity of presentation, the following notations are used in Alg. 1. We denote by A the matrix K_μ on which the Cholesky factorization operates. i_l stands for the index of the first row of the i^{th} constraint in K_μ. This pseudo-code is very similar in spirit to the ones presented in [14]. Yet, we chose to present the computations for a factorization exploiting an upper triangular matrix, while Featherstone uses a lower triangular matrix. The *pass over the joints* exactly corresponds to the Cholesky decomposition of the JSIM. The other code blocks are completely novel and related to the constraints.

Following the essence of this pseudo-code, it is possible to directly extend it to the matrix operations $U^t\boldsymbol{x}$, $U\boldsymbol{x}$, $U^{-t}\boldsymbol{x}$ or $U^{-1}\boldsymbol{x}$ associated to the Cholesky decomposition. We skip their presentations to avoid overloading the paper.

E. Cholesky factorization of Λ_μ^{-1}

Following the structure exhibited in (24), it appears that the Cholesky decomposition of Λ_μ (previously denoted by $(U_{\Lambda_\mu^{-1}}, D_{\Lambda_\mu^{-1}})$) can be extracted as a side product of Alg. 1,. $U_{\Lambda_\mu^{-1}}$ and $-D_{\Lambda_\mu^{-1}}$ correspond to the upper left block of dimension $m \times m$ of U_{K_μ} and D_{K_μ} respectively. To the best of the authors' knowledge, this is the first time that such a connection is made.

While this result is interesting for robot control in order to get an expression of the OSIM, it may also impact the way to solve complementarity problems [11] in robot simulations. The Delassus matrix has a central role in contact simulation.

Algorithm 1 Pseudo-code of the Cholesky factorization of A

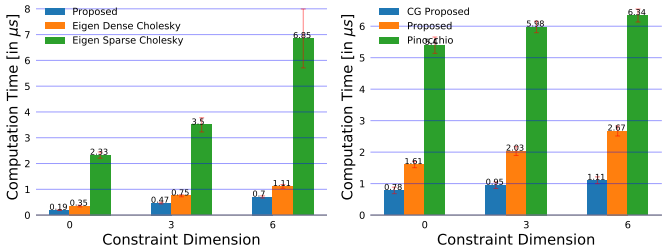
```
for k = N to 1 do
    i = p(k)
    while i > 0 do
        a = A_{i+m,k+m} / A_{k+m,k+m}
        j = i
        Pass over the joints
        while j > 0 do
            A_{j+m,i+m} = A_{j+m,i+m} − A_{j+m,k+m} a
            j = p(j)
        end
        Pass over the constraints
        for l = n_c to 1 do
            if i ∈ σ_l
              for j = n_i to 1 do
                  A_{j+i_l,i+m} = A_{j+i_l,i+m} − A_{j+i_l,k+m} a
              end
            end
        end
        A_{i+m,k+m} = a
        i = p(i)
    end
end
Dense factorization related to the OSIM
for l = n_c to 1 do
    for k̃ = n_i to 1 do
        k = i_l + k̃
        for i = k − 1 to 1 do
            a = A_{i,k} / A_{k,k}
            for j = i to 1 do
                A_{j,i} = A_{j,i} − A_{j,k} a
            end
            A_{i,k} = a
        end
    end
end
```

It corresponds to the Hessian of the linear or nonlinear complementarity problem [5]. Most of the current simulators use Gauss-Seidel like algorithms [22], which are pivoting methods and might be slow to converge, especially when Λ_μ^{-1} is badly conditioned. The formulation that we propose here would enable Newton-based methods, which should be then preferred to accelerate the converge rate of contact simulators.

F. Generality of the approach

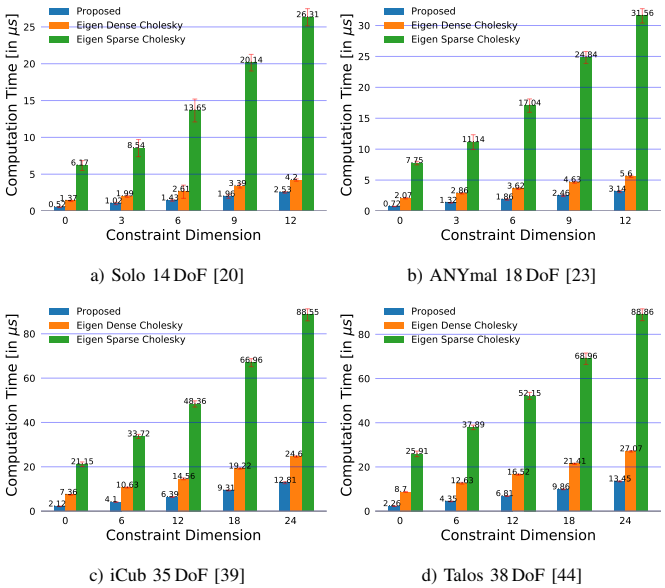
Until now, we have only considered the case where the matrix M at the bottom right corner of K_μ in (18) corresponds to the JSIM. Similarly to the factorization introduced in [14], it is worth mentioning that the proposed approach is not limited to this particular case but can be generalized to any symmetric positive-definite matrix satisfying the regular numbering scheme and the property defined by (22).

For example, this includes the particular case where M is a diagonal matrix made of positive elements, which occurs in



a) Cholesky Decomposition b) Contact Dynamics

Fig. 4. UR5 Benchmarks [27]: Our formulation is even able to improve the computational timings for a small dimension (6 DoF) manipulator.



a) Solo 14 DoF [20] b) ANYmal 18 DoF [23]

c) iCub 35 DoF [39] d) Talos 38 DoF [44]

Fig. 5. Proposed Cholesky decomposition (ours) vs Eigen sparse Cholesky (`Eigen::SimplicialLDLT`) vs Eigen dense Cholesky (`Eigen::LDLT`) solver: Our proposed sparse formulation exploits at best the sparsity of the kinematic structure of a robot to compute the Cholesky decomposition associated to the KKT matrix $K_\mu(\boldsymbol{q})$ (18). The benchmarks show the higher performance of our sparse formulation against the sparse (`Eigen::SimplicialLDLT`) and dense Cholesky decomposition (`Eigen::LDLT`) of the Eigen [21] library.

many optimization setups in robotics like inverse kinematics or inverse geometry. In that cases the optimization routines need to iteratively solve problems similar to (6), especially in the context of augmented Lagrangian or sequential quadratic programming methods. These setups should in turn benefit from the proposed generic decomposition to significantly accelerate their internal computations, as shown in the case of forward dynamics in the next section.

V. Results

We implemented in `C++` both the sparse Cholesky factorization from Sec. IV and the corresponding constrained dynamics formulation within the Pinocchio library [9, 10]. Comparison of our implementation with the state of the art highlights the benefits of the proposed proximal formulation of the constrained dynamics and the exploitation of the branch-induced sparsity. We benchmark our implementation for multiple robots with different dimensions, kinematic structures, and type of constraints. Additionally, we show the results of our constrained dynamics implementation to simulate the motion of different robots[3]. Additional results are also depicted in the companion video.

A. Benchmark specifications

We use two metrics to demonstrate the performance of our algorithm against state-of-the-art implementations currently available in the community: Time taken for the Cholesky decomposition of the relaxed KKT matrix $K_\mu(\boldsymbol{q})$, and the time taken for solution of the constrained dynamics (6).

All our benchmarking results were obtained on a single thread of `Intel(R) Core(TM) i7-7820HQ @ 2.90GHz CPU` running `Ubuntu 16.04`. The average and the standard deviation of our computation timings were obtained from a dataset of 10^5 trials. The benchmarking was done using `clock_gettime` function in `C++` with `CLOCK_MONOTONIC` as the time source. We disabled TurboBoost to reduce variability in benchmark timings. Publicly available `urdf` and `sdf` models were used to load the robots in our environment. All code was compiled with `Clang++ 10.0`, and Eigen [21] library `v3.4.0-rc1`.

For all benchmarks, we first compute the performance of our formulation without any constraints ($m = 0$). Then we sequentially add kinematic constraints on various joints and bench the performance each time. For quadruped robots (Solo [20], ANYmal [23]), 3D point contact constraints are sequentially added to each foot, while for Bipeds (iCub [39], Talos [44]), 6D surface contacts are added to the limbs. For the manipulator UR5 [27], we measure the performance of our algorithm with a) no constraints, b) one position constraint on gripper ($m = 3$), and c) one position and orientation constraint on gripper ($m = 6$).

B. Benchmark of the Cholesky decomposition

Fig. 4 and 5 show the mean and standard deviation of the computation timings for our proposed sparse Cholesky factorization against state-of-the-art implementations in Eigen `C++` library [21]. Overall, we can note a reduction in computation time by more than 50% for almost all robotic platforms when compared to Eigen's Dense Cholesky implementation. Since we exploit the sparsity of the robot kinematics to improve the performance, the improvement is dependent on the dimensions of the robot and the constraints. For small dimensions, like the serial chain UR5 manipulator [27], the improvement is smaller. On the other hand, non-serial kinematic chains and high dimension robotic platforms such as quadruped or biped robots introduce more sparsity in the dynamic equations which

[3]The open-source `C++` implementation, the exhaustive benchmarks and the full code are the main new features in Pinocchio 3. The pre-release of Pinocchio 3 is available under the tag 2.9.x on https://github.com/stack-of-tasks/pinocchio

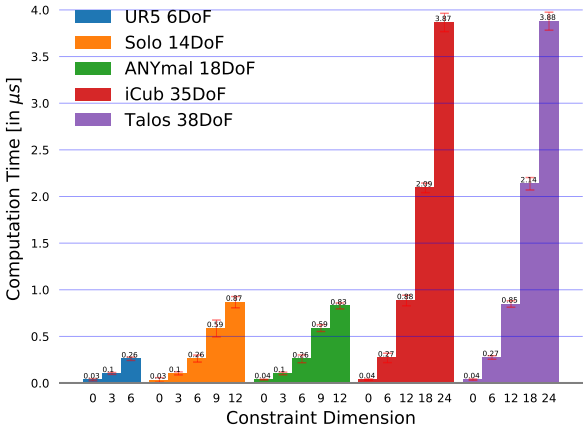


Fig. 6. Benchmarking of the inverse of the OSIM matrix: $\Lambda_\mu^{-1}(\boldsymbol{q})$ is directly available as an output from the sparse decomposition of the KKT Matrix $K_\mu(\boldsymbol{q})$ in Sec. IV

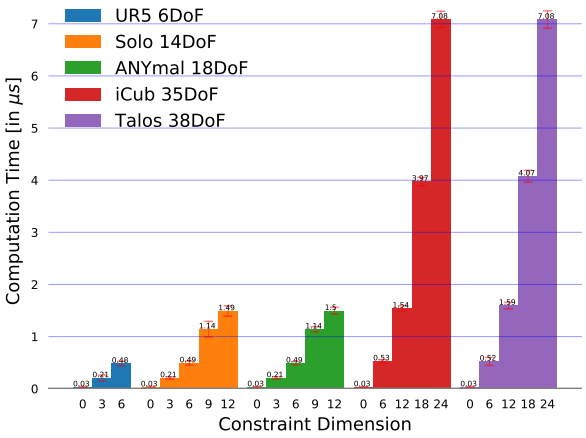


Fig. 7. Benchmarking of the OSIM matrix: $\Lambda_\mu(\boldsymbol{q})$ is easily computed by using the inverse of the sparse Cholesky decomposition of the KKT Matrix $K_\mu(\boldsymbol{q})$ in Sec. IV

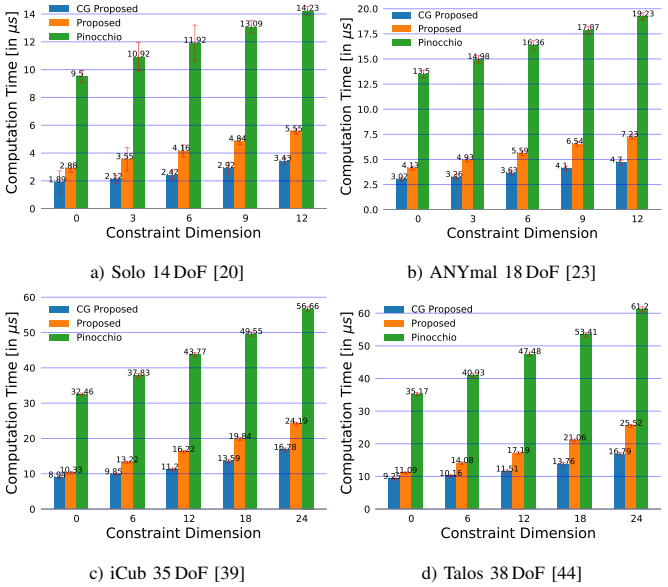
we can exploit. For example, Fig. 4 shows the Cholesky decomposition timings in UR5 (6 DoF) without constraints. The timings are improved by $\sim 45\%$ with our sparse formulation when compared with the Dense formulation from Eigen libray [21][4]. Fig. 5 shows the timings in popular quadruped (Solo [20], ANYmal [23]) and biped(iCub [39], Talos [44]) robots. We see much higher improvements in Talos (38 DoF) robot without constraints ($\sim 74\%$). Overall, all the robots show significant improvements in computation timings when the sparse formulation introduced in Sec. IV is exploited.

C. Extracting the OSIM matrix

The extraction of the OSIM is a dense operation that does not benefit from the sparsity of the robot kinematics. However, the computation of these quantities becomes straight-forward once decomposition of the KKT matrix $K_\mu(\boldsymbol{q})$ is available. Fig. 7 and 6 show the computation timings for extracting the OSIM and inverse of OSIM matrices after Cholesky decomposition. We see exceedingly tiny computation times ($\Lambda_\mu(\boldsymbol{q})$ OSIM Inverse) of $\sim 0.03\,\mu s$ for UR5 manipulator, and $\sim 7\,\mu s$ for higher dimensional Talos robot with 24 constraints.

Computation of the inverse OSIM $\Lambda_\mu^{-1}(\boldsymbol{q})$ is even cheaper, since the Cholesky decomposition can be directly used as discussed in Sec. IV (Talos with 24 constraints needs only $4\,\mu s$ to compute $\Lambda_\mu^{-1}(\boldsymbol{q})$). Overall, after the factorization is done, the timings depend only on the dimension of the constraints (since it is a dense operation). As a result, the vectorization provided by modern CPUs and supported by Eigen library helps to improve performance for higher dimensions.

[4]Curiously, serial robots such as robotic arms have a full dense mass matrix M. However, our rigid body dynamics C++ implementation still outperforms the off-the-shelf Cholesky solvers present in Eigen. This *a priori* unexpected gain is in part due to the fact that Eigen also performs a pivoting strategy when computing the Cholesky decomposition, mostly to improve the numerical stability of the overall factorization, which might be discarded in most robotic applications.



a) Solo 14 DoF [20]

b) ANYmal 18 DoF [23]

c) iCub 35 DoF [39]

d) Talos 38 DoF [44]

Fig. 8. Proposed Contact Dynamics: The benchmarks show performance of the proposed constrained dynamics algorithm against the state-of-the-art Pinocchio [10] C++ library for different robotic platforms.

D. Benchmarking the constrained forward dynamics

In Fig. 8, we see significant reduction with respect to the state-of-the-art in the timings for the solution of constrained dynamics (6), which involves the computation of the inverse of the KKT matrix K_μ. We benchmark our algorithm against the `forwardDynamics` function available in the Pinocchio [10] library. Moreover, we use the source-code generation tools (CppADCodegen [32]) available in the Pinocchio library to compile binary code for sparse decomposition of all our (robot, constraints) combinations. We see significant reduction in the timings for constrained dynamics because of the sparse solutions, with our formulation reducing timings by $> 50\%$ in all robots. In addition, we see a small improvement because

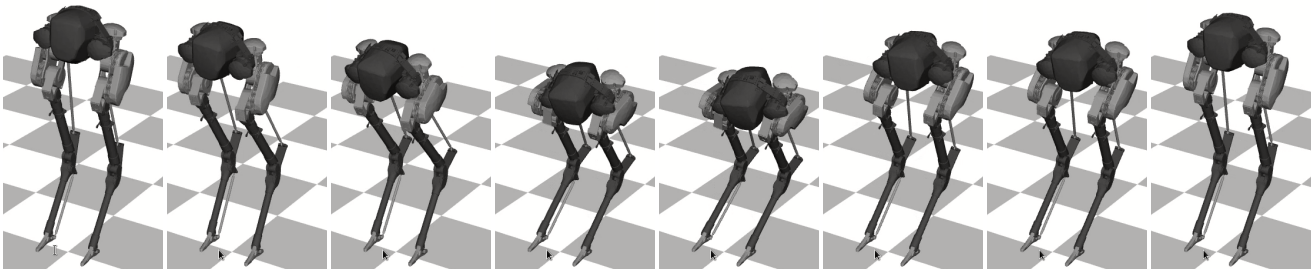


Fig. 9. Cassie robot [19] simulation: Each leg of the robot has two 3D kinematic loop constraints, and line contact with the ground (implemented as 3D constraints at heel and toe). $m = 24$

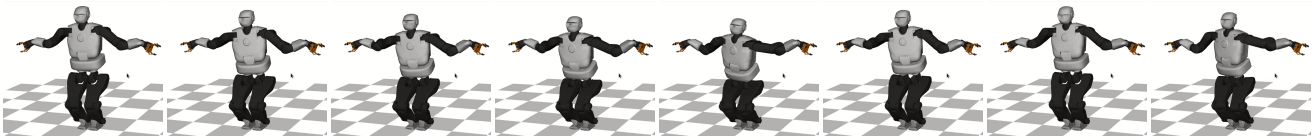


Fig. 10. Talos robot [44] simulation: Each limb of the robot has one 6 DoF kinematic constraint (surface contact). $m = 24$

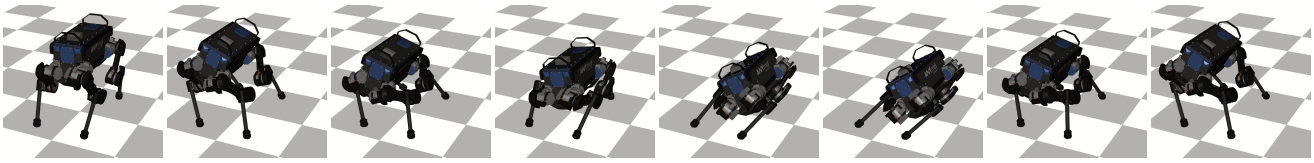


Fig. 11. ANYmal robot [23] simulation: Each limb of the robot has one 3 DoF kinematic constraint (point contact). $m = 12$

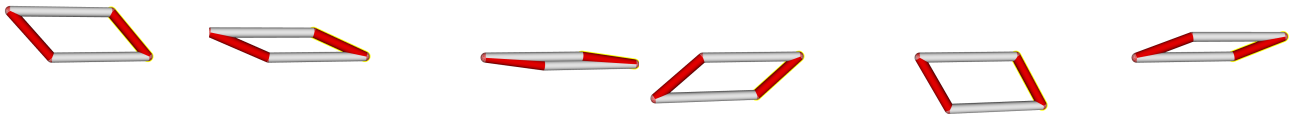


Fig. 12. 4-bar linkage system simulation: The multibody system has one 3 DoF kinematic loop constraint. $m = 3$

of the binary source-code, which further optimizes over redundant computations. However, this additional performance benefit is capped because of the lack of vectorization in binary compiled functions. In total, our sparse method and source-code generation vastly outperform the state-of-the-art implementation.

E. Simulation of constrained dynamics

Fig. 9, 10, 11 and 12 show the simulation of robots with different kinematic structures and different kinematic constraints in motion. We do task-space inverse dynamics while accounting for the a) kinematic constraints, and b) reference center of mass, to compute a reference torque and joint trajectory. For simplicity of demonstration of our contribution, we do not add any other constraints in our simulation.

The Cassie robot (Fig.9) has loop closure constraints at the hip and toe joints of each leg (parallelogram mechanisms). Each foot creates a line contact with the ground, which is implemented as two 3D constraints located at the ends of the line segment. The redundancy of this implementation is handled by the proximal formulation of our contact dynamics.

Each foot of the Talos robot (Fig.10) creates a surface contact with the ground, which is represented as a 6D constraint (spatial velocity). In addition, we add a 6 DoF constraint on each gripper of the robot to fix its placement. Similarly, there are point contact on each foot of the ANYmal robot (Fig. 11).

In addition to the robotic platforms, we add the pendulum motion of a 4-bar linkage (Fig. 12) with one 3 DoF loop closure constraint, which is very often used as a pedagogic example of a closed-loop mechanism.

All these motions are available in the companion video [5].

F. Robustness of the Proximal Algorithm

We also try to investigate the behaviour of the proximal algorithm in case of a highly redundant set of constraints. In a Talos robot, we put 6D surface constraints on the Right Foot, Left Foot, and the Left Hand, and then duplicate these constraints (thus, six 6 DoF constraints, $m = 36$).

The setup with $\mu = 0$ fails, as is expected in such a scenario, while the proximal algorithm with $\mu > 0$ is able to find a solution. This comparison is shown in the companion video.

VI. Discussions and Conclusion

In this paper, we have proposed an original formulation to compute the forward dynamics of a robotic system subject to constraints. The method uses generalized coordinates and implicit constraints, which leads both to an efficient and generic formulation. It is able to handle various kinds of mechanical constraints, in particular closed kinematic chains

[5] https://peertube.laas.fr/videos/watch/88ceab1e-2cc1-4bd1-9c9e-2be0c2c31890

and contact constraints. The method is built on several contributions. First we proposed an extension of the classical Cholesky decomposition of the joint space inertial matrix, which takes the constraints into account. This decomposition, based on the simple yet key reordering of the blocks of the KKT matrix, also reveals an elegant and efficient way of obtaining the decomposition of the operational-space inertia matrix, as a side effect of the main computations. It allows us to take into account a regularization factor, that we used to build a proximal resolution algorithm for the constrained dynamics. The resulting algorithm then computes a robust solution, despite ill-conditioning or near singularity, without sacrificing computational efficiency and numerical accuracy.

We have implemented our algorithm in an efficient framework, and benchmarked our contribution on several different systems, in the presence of both contact constraints and kinematic closed loop. The source code is available in the library Pinocchio along with the benchmarks to enable full reproduction of our work.

The proposed algorithm only handles bilateral constraints. Based on this work, we are working on a complete contact simulator able to handle unilateral constraints (i.e. friction, sliding, unbinding) as well. We also want to propose an efficient way of differentiating our algorithm and compute the derivatives of the constrained dynamics, going beyond classic derivatives of unconstrained forward or inverse dynamics [8]. This necessary for efficient use in an optimization problem, for example for optimal control or model predictive control of complex legged robots [6, 26, 31, 33, 36].

ACKNOWLEDGMENT

This work was supported in part by the HPC resources from GENCI-IDRIS (Grant AD011011342), the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute) and ANR-19-P3IA-000 (ANITI 3IA Institute), Louis Vuitton ENS Chair on Artificial Intelligence, and the European project MEMMO (Grant 780684).

REFERENCES

[1] David Baraff. Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 137–146, 1996.

[2] Jan Bender, Kenny Erleben, and Jeff Trinkle. Interactive simulation of rigid body dynamics in computer graphics. *Computer Graphics Forum*, 33(1):246–270, 2014.

[3] Michele Benzi, Gene H Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta numerica*, 14:1–137, 2005.

[4] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[5] Bernard Brogliato. *Nonsmooth mechanics*. Springer, 1999.

[6] Rohan Budhiraja, Justin Carpentier, Carlos Mastalli, and Nicolas Mansard. Differential dynamic programming for multi-phase rigid contact dynamics. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2018.

[7] Justin Carpentier. Analytical inverse of the joint space inertia matrix. Technical report, LAAS-CNRS, 2018.

[8] Justin Carpentier and Nicolas Mansard. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and Systems (RSS 2018)*, 2018.

[9] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. Pinocchio: fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio, 2015–2021.

[10] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE/SICE International Symposium on System Integration (SII)*, 2019.

[11] Richard W Cottle, Jong-Shi Pang, and Richard E Stone. *The linear complementarity problem*. SIAM, 2009.

[12] Étienne Delassus. Mémoire sur la théorie des liaisons finies unilatérales. In *Annales scientifiques de l'École Normale Supérieure*, volume 34, pages 95–179, 1917.

[13] Roy Featherstone. The calculation of robot dynamics using articulated-body inertias. *The international journal of robotics research*, 2(1):13–30, 1983.

[14] Roy Featherstone. Efficient factorization of the joint-space inertia matrix for branched kinematic trees. *The International Journal of Robotics Research*, 24(6):487–500, 2005.

[15] Roy Featherstone. Exploiting sparsity in operational-space dynamics. *The International Journal of Robotics Research*, 29(10):1353–1368, 2010.

[16] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[17] Martin L Felis. RBDL: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 41(2):495–511, 2017.

[18] Paulo Flores, Margarida Machado, Eurico Seabra, and Miguel Tavares da Silva. A parametric study on the Baumgarte stabilization method for forward dynamics of constrained multibody systems. *Journal of computational and nonlinear dynamics*, 6(1), 2011.

[19] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J. Huang, and J. Grizzle. Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway. In *2019 American Control Conference (ACC)*, pages 4559–4566, 2019. doi: 10.23919/ACC.2019.8814833.

[20] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, T. Flayols, J. Fiene, A. Badri-Spröwitz, and L. Righetti. An open torque-controlled modular robot architecture for legged locomotion research. *IEEE Robotics*

and Automation Letters, 5(2):3650–3657, 2020. doi: 10.1109/LRA.2020.2976639.

[21] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[22] Peter C Horak and Jeff C Trinkle. On the similarities and differences among contact models in robot simulation. *IEEE Robotics and Automation Letters*, 4(2):493–499, 2019.

[23] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger. Anymal - a highly mobile and dynamic quadrupedal robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016. doi: 10.1109/IROS.2016.7758092.

[24] K. Ishihara, T. D. Itoh, and J. Morimoto. Full-body optimal control toward versatile and agile behaviors in a humanoid robot. *IEEE Robotics and Automation Letters*, 5(1):119–126, 2020. doi: 10.1109/LRA.2019.2947001.

[25] Abhinandan Jain. *Robot and multibody dynamics: analysis and algorithms*. Springer Science & Business Media, 2010.

[26] Sarah Kazdadi, Justin Carpentier, and Jean Ponce. Equality constrained differential dynamic programming. In *2021-IEEE International Conference on Robotics and Automation*, 2021.

[27] P. M. Kebria, S. Al-wais, H. Abdi, and S. Nahavandi. Kinematic and dynamic modelling of UR5 manipulator. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016. doi: 10.1109/SMC.2016.7844896.

[28] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[29] Twan Koolen and Robin Deits. Julia for robotics: Simulation and real-time control in a high-level programming language. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[30] Joseph-Louis Lagrange. *Mécanique analytique*. Académie Royale des Sciences, 1788.

[31] Quentin Le Lidec, Igor Kalevatykh, Ivan Laptev, Cordelia Schmid, and Justin Carpentier. Differentiable simulation for physical system identification. *IEEE Robotics and Automation Letters*, 6(2):3413–3420, 2021.

[32] João Rui Leal. Cppadcodegen. https://github.com/joaoleal/CppADCodeGen, 2020.

[33] H. Li and P. M. Wensing. Hybrid systems differential dynamic programming for whole-body motion planning of legged robots. *IEEE Robotics and Automation Letters*, 5(4):5448–5455, 2020. doi: 10.1109/LRA.2020.3007475.

[34] Kathryn Lilly and David Orin. Efficient O(n) recursive computation of the operational space inertia matrix. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(5), 1993.

[35] J. Luh, M. Walker, and R. Paul. On-line computational scheme for mechanical manipulators. *J. Dyn. Sys., Meas., Control.*, 102(2):69–76, June 1980.

[36] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2536–2542. IEEE, 2020.

[37] Jean-Pierre Merlet. *Parallel robots*, volume 128. Springer Science & Business Media, 2005.

[38] Michael Mistry and Ludovic Righetti. Operational space control of constrained and underactuated systems. *Robotics: Science and systems (RSS)*, 2012.

[39] Francesco Nori, Silvio Traversaro, Jorhabib Eljaik, Francesco Romano, Andrea Del Prete, and Daniele Pucci. iCub whole-body control through force regulation on rigid non-coplanar contacts. *Frontiers in Robotics and AI*, 2:6, 2015.

[40] David E Orin, RB McGhee, M Vukobratović, and G Hartoch. Kinematic and kinetic analysis of open-chain linkages utilizing newton-euler methods. *Mathematical Biosciences*, 43(1-2):107–130, 1979.

[41] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

[42] Jaeheung Park and Oussama Khatib. Contact consistent control framework for humanoid robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[43] R Tyrrell Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of operations research*, 1(2): 97–116, 1976.

[44] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux, J-P. Laumond, L. Marchionni, H. Tome, and F. Ferro. Talos: A new humanoid research platform targeted for industrial applications. In *IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, Nov 2017. doi: 10.1109/HUMANOIDS.2017.8246947.

[45] Patrick Wensing, Roy Featherstone, and David E Orin. A reduced-order recursive algorithm for the computation of the operational-space inertia matrix. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.