2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Exploiting sparsity in robot trajectory optimization with direct collocation and geometric algorithms

Daniel Cardona-Ortiz, Alvaro Paz and Gustavo Arechavaleta

Abstract— **This paper presents a robot trajectory optimization formulation that builds upon numerical optimal control and Lie group methods. In particular, the inherent sparsity of direct collocation is carefully analyzed to dramatically reduce the number of floating-point operations to get first-order information of the problem. We describe how sparsity exploitation is employed with both numerical and analytical differentiation. Furthermore, the use of geometric algorithms based on Lie groups and their associated Lie algebras allow to analytically evaluate the state equations and their derivatives with efficient recursive algorithms. We demonstrate the scalability of the proposed formulation with three different articulated robots, such as a finger, a mobile manipulator and a humanoid composed of five, eight and more than twenty degrees of freedom, respectively. The performance of our implementation in C++ is also validated and compared against a state-of-the-art general purpose numerical optimal control solver.**

I. Introduction

Direct transcription methods in numerical optimal control have evolved into powerful general purpose solvers capable to optimize the trajectories of complex dynamical systems subject to a wide variety of path constraints [1], [2], [3], [4]. The solvers [1] and [4] apply direct multiple shooting with dedicated sequential quadratic programs (SQP). On the other hand, the solvers [2] and [3] make use of pseudospectral collocation combined with interior-point methods or SQP.

Although part of the success is supported by the maturity of nonlinear programming (NLP), the role of transcription methods is of great importance. They are employed for converting the original optimal control problem (OCP) into a large-scale NLP [5], [6]. This process strongly impacts the size of the underlying NLP in terms of the number of decision variables, defect constraints to satisfy the system dynamics and the associated sparsity.

The aforementioned solvers have demonstrated good results to generate robotic movements [7], [8]. Recently, other specific solvers based on direct collocation have been suggested for robot motion optimization [9], [10], [11]. A shared aspect in all these works is the computational method that is used to obtain first-order information of the optimization problem. In particular, the gradient of the cost function and constraint Jacobians of the NLP are commonly computed with either finite differences or automatic differentiation. Otherwise, dedicated symbolic computation is employed as suggested in [11]. The main advantage of these methods is that the user does not have to provide how to compute first-order information as an input to the optimization problem. Thus, only the running cost and the system dynamics of the OCP are provided. However, finite differences methods severely affect the computational cost to obtain the solution while automatic differentiation turns to be very restrictive when the system dynamics implies the computation of the equations of motion of robotic systems with many degrees of freedom. In this work we evaluate both methods to highlight these inconveniences.

We propose to exploit as much as possible the sparse structure of the large-scale NLP, but also we introduce a key ingredient to dramatically reduce the computational time to obtain the solution regardless the complexity of robot's equations of motion. Notably, the use of geometric algorithms allows to evaluate the robot's equations of motion. In the same spirit of [12], where the robot's inverse dynamics and the partial derivatives with respect to collocation points with B-splines are computed with recursive algorithms, here we apply geometric operators to efficiently differentiate the robot's forward dynamics. Furthermore, we show how the proposed solver extends the applicability of numerical optimal control for robot trajectory optimization.

The organization of the paper is as follows. First, Section II formulates the OCP as well as its discretization to get the large-scale NLP. Then, Section III describes the recipe to exploit the sparse structure of the constraint Jacobian. Section IV introduces the analytical differentiation of the robot's forward dynamics with geometric algorithms. The numerical results are given in Section V with three different robots. Concluding remarks are given in Section VI.

II. Robot trajectory optimization problem

Let us formulate an optimal control problem (OCP) as follows: given an articulated rigid body system with n degrees of freedom (DoF) and a state vector

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix} \in \mathbb{R}^{2n}, \tag{1}$$

where $\{\boldsymbol{q}, \dot{\boldsymbol{q}}\} \in \mathbb{R}^n$ are the robot configuration, and joint velocities, respectively, find a control law (if exists) that leads the system from its initial state to a final one by solving the following OCP:

$$\underset{\boldsymbol{u}(t)\in\boldsymbol{U}}{\text{minimize}} \quad \int_0^T w(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \tag{2}$$

Authors are with Robotics and Advanced Manufacturing Group, Centro de Investigación y de Estudios Avanzados del IPN, Saltillo, Coah. México. {daniel.cardona,alvaro.paz,garechav}@cinvestav.mx

978-1-7281-7395-5/20/$31.00 ©2020 IEEE

$$\begin{aligned} \text{subject to} \quad & \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\ & \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \leq 0 \\ & \boldsymbol{x}_{lb} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{ub} \\ & \boldsymbol{x}(0) = \boldsymbol{x}_i \\ & \boldsymbol{x}(T) = \boldsymbol{x}_f \end{aligned} \tag{3}$$

where $\boldsymbol{U} \subset \mathbb{R}^n$ defines the admissible control domain, $w(\cdot)$ stands for the integrand of the Lagrangian, $g(\cdot) \in \mathbb{R}^p$ are the path constraints, and the dynamics of the robot is expressed in its state equation form as

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \begin{bmatrix} \dot{\boldsymbol{q}}(t) \\ \ddot{\boldsymbol{q}}(t) \end{bmatrix} \in \mathbb{R}^{2n}, \tag{4}$$

with

$$\ddot{\boldsymbol{q}}(t) = \boldsymbol{H}(\boldsymbol{q}(t))^{-1}(\boldsymbol{u}(t) - \boldsymbol{h}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) \tag{5}$$

where $\boldsymbol{H}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ denotes the inertia matrix, $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^n$ contains the non-linear terms, and $\boldsymbol{u}$ is the generalized input torques. The constraint vectors $\boldsymbol{x}_{lb} \in \mathbb{R}^{2n}$ and $\boldsymbol{x}_{ub} \in \mathbb{R}^{2n}$ are the lower and upper bounds of the states of the system, and $\{\boldsymbol{x}_i, \boldsymbol{x}_f\} \in \mathbb{R}^{2n}$ are the initial and final states.

A. Direct collocation formulation

Among the existing collocation schemes to approximate the original OCP, where the system dynamics evolves continuously along the entire trajectory, we restrict our focus on a *Lobatto* method. In particular, the trapezoidal method is one of the simplest and computationally less expensive method. However, more sophisticated collocation schemes such as Runge Kutta [5] and orthogonal methods [13] can be applied. The use of the trapezoidal quadrature allows to convert the problem in (2) and (3) into a NLP of the form

$$\underset{\boldsymbol{z}}{\text{minimize}} \ \frac{1}{2} \sum_{k=0}^{N-1} h(w_k + w_{k+1}) \tag{6}$$

$$\begin{aligned} \text{subject to} \quad & \boldsymbol{c}_L \leq \boldsymbol{c}(\boldsymbol{z}) \leq \boldsymbol{c}_U \\ & \boldsymbol{x}_{lb} \leq \boldsymbol{x}_k \leq \boldsymbol{x}_{ub}, \quad \forall k \in \mathbb{N} \end{aligned} \tag{7}$$

where $h = \frac{t_f - t_0}{N-1}$, N is the number of collocation points and $\boldsymbol{z} \in \mathbb{R}^{3nN+2}$ contains the NLP decision variables

$$\boldsymbol{z} = [t_0 \ t_f \ \boldsymbol{x}_1^T \ \boldsymbol{u}_1^T \ \boldsymbol{x}_2^T \ \boldsymbol{u}_2^T \ \ldots \ \boldsymbol{x}_N^T \ \boldsymbol{u}_N^T]^T \tag{8}$$

The system dynamics is now expressed as a set of collocation constraints

$$\boldsymbol{\Phi} = [\boldsymbol{\varphi}_1^T \ \boldsymbol{\varphi}_2^T \ \ldots \ \boldsymbol{\varphi}_{N-1}^T]^T \in \mathbb{R}^{2n(N-1)} \tag{9}$$

with

$$\boldsymbol{\varphi}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k - \frac{1}{2} h(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k) \tag{10}$$

The path constraints have the form

$$\boldsymbol{\gamma} = [\boldsymbol{g}_1^T \ \boldsymbol{g}_2^T \ \ldots \ \boldsymbol{g}_N^T]^T \in \mathbb{R}^{pN} \tag{11}$$

and $\boldsymbol{x}(0)$ and $\boldsymbol{x}(T)$ are known as event constraints

$$\boldsymbol{e} = [\boldsymbol{x}_0^T \ \boldsymbol{x}_N^T]^T \in \mathbb{R}^{4n} \tag{12}$$

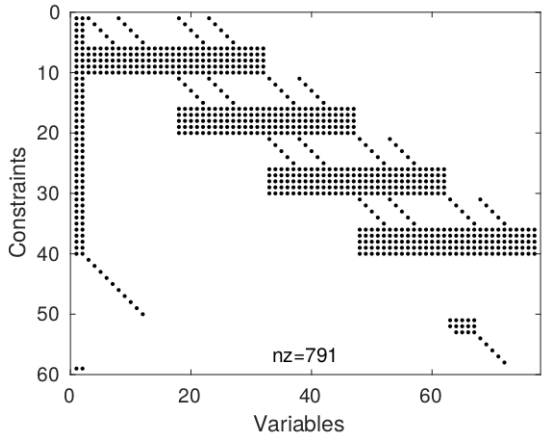


Fig. 1: Sparsity of full constraint Jacobian for a 5-DoF robot with $N = 5$ and nz is the number of non-zero elements.

The overall NLP constraint vector is defined as

$$\boldsymbol{c}(\boldsymbol{z}) = [\boldsymbol{\Phi}^T \ \boldsymbol{\gamma}^T \ \boldsymbol{e}^T]^T \in \mathbb{R}^{n_{cns}} \tag{13}$$

together with bound constraints

$$\boldsymbol{c}_L = [\mathbf{0} \ \mathbf{0} \ \ldots \ \mathbf{0} \ \boldsymbol{g}_L \ \boldsymbol{g}_L \ \ldots \ \boldsymbol{g}_L \ \boldsymbol{x}_i \ \boldsymbol{x}_f] \in \mathbb{R}^{n_{cns}} \tag{14}$$

$$\boldsymbol{c}_U = [\mathbf{0} \ \mathbf{0} \ \ldots \ \mathbf{0} \ \boldsymbol{g}_U \ \boldsymbol{g}_U \ \ldots \ \boldsymbol{g}_U \ \boldsymbol{x}_i \ \boldsymbol{x}_f] \in \mathbb{R}^{n_{cns}} \tag{15}$$

where $n_{cns} = (2n + p)N + 2n$.

B. Sparsity of NLP

In order to solve the problem given in (6) and (7), the use of numerical optimization strategies such as sequential quadratic programming or interior-point methods become essential. In any case, the Karush-Kuhn-Tucker (KKT) first-order necessary conditions for optimality must be satisfied [14]. Thus, the gradient of (6) and the Jacobian of (7) must be available. In numerical optimal control, it is well known that the size of the constraint Jacobian is large and its structure is sparse [5]. This is mainly due to the number of defect constraints (10) for satisfying the system dynamics. In particular, the Jacobian of (7) is defined as

$$\boldsymbol{J}(\boldsymbol{z}) = \frac{\partial \boldsymbol{c}(\boldsymbol{z})}{\partial \boldsymbol{z}} = \begin{bmatrix} \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{z}} \\ \\ \frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{z}} \\ \\ \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{z}} \end{bmatrix} \in \mathbb{R}^{n_z \times n_{cns}} \tag{16}$$

and the associated sparsity shaped by (10), (11) and (12) fulfills the following conditions

$$\frac{\partial \boldsymbol{\varphi}_i}{\partial \boldsymbol{z}_k} \in \mathbb{R}^{2n \times 3n} = \begin{cases} \frac{\partial \boldsymbol{\varphi}_i}{\partial \boldsymbol{z}_k} & \text{if } i = k \mid i = k+1 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{z}_k} \in \mathbb{R}^{p \times 3n} = \begin{cases} \frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{z}_k} & \text{if } i = k \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{x}_k} \in \mathbb{R}^{2n \times 2n} = \begin{cases} \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{x}_k} & \text{if } k = 0 \mid k = N \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{u}_k} \in \mathbb{R}^{2n \times 2n} = \mathbf{0}$$

where $\boldsymbol{z}_k = [\boldsymbol{x}_k^T \ \boldsymbol{u}_k^T]^T \in \mathbb{R}^{3n}$. Fig. 1 illustrates the structure of (16) by considering the state space representation of the equations of motion of a 5-DoF robotic finger with $N = 5$.

III. Sparse Jacobian exploitation

We first exploit the separability of (10) by rewriting it as

$$\varphi_k = [\boldsymbol{x}_{k+1} - \boldsymbol{x}_k] - \frac{1}{2}[h\boldsymbol{f}_{k+1}] - \frac{1}{2}[h\boldsymbol{f}_k] \tag{17}$$

It is then possible to express the NLP constraints (13) as

$$\boldsymbol{c}(\boldsymbol{z}) = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{y}(\boldsymbol{z}) \tag{18}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0}_{2n\times 2} & -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{I} & & & & \\ & & & -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{I} & & \\ \vdots & & & & & & \ddots & \\ \boldsymbol{0}_{2n\times 2} & & & & & & -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}$$

The matrix $\boldsymbol{A} \in \mathbb{R}^{n_{cns}\times(3nN+2)}$ contains copies of an identity matrix $\boldsymbol{I} \in \mathbb{R}^{2n\times 2n}$ and null matrices $\boldsymbol{0} \in \mathbb{R}^{2n\times n}$. The proposed shape of $\boldsymbol{B} \in \mathbb{R}^{n_{cns}\times N(2n+p)+4n}$ is

$$\boldsymbol{B} = -\frac{1}{2}\begin{bmatrix} \boldsymbol{I}_\varphi & \boldsymbol{I}_\varphi & & & & & \\ & \boldsymbol{I}_\varphi & \boldsymbol{I}_\varphi & & & & \\ & & & \ddots & & & \\ & & & & \boldsymbol{I}_\varphi & \boldsymbol{I}_\varphi & \\ & & & & & & \boldsymbol{I}_{\boldsymbol{g}} \end{bmatrix}$$

where $\boldsymbol{I}_\varphi \in \mathbb{R}^{n\times n}$ and $\boldsymbol{I}_{\boldsymbol{g}} \in \mathbb{R}^{(pN+4n)\times(pN+4n)}$ are identity matrices. The vector $\boldsymbol{y}(\boldsymbol{z}) \in \mathbb{R}^{N(2n+p)+4n}$ stacked the following terms

$$\boldsymbol{y}(\boldsymbol{z}) = \begin{bmatrix} \bar{\boldsymbol{f}} \\ \boldsymbol{\gamma} \\ \boldsymbol{e} \end{bmatrix} \tag{19}$$

where $\bar{\boldsymbol{f}} = \begin{bmatrix} h\boldsymbol{f}_1^T & h\boldsymbol{f}_2^T & h\boldsymbol{f}_3^T & \cdots & h\boldsymbol{f}_N^T \end{bmatrix}^T \in \mathbb{R}^{2nN}$

From (18), the constraint Jacobian (16) is rewritten as

$$\boldsymbol{J}(\boldsymbol{z}) = \boldsymbol{A} + \boldsymbol{B}\boldsymbol{D} \tag{20}$$

where $\boldsymbol{D} = \frac{\partial \boldsymbol{y}(\boldsymbol{z})}{\partial \boldsymbol{z}} \in \mathbb{R}^{(N(2n+p)+4n)\times(3nN+2)}$, and its structure is visualized in Fig. (2). Note that classical sparse matrix computation techniques can be applied to evaluate (18) and (20). However, in order to numerically calculate $\boldsymbol{D}$ we adopt the method suggested in [15] that performs sparse finite differences by efficiently grouping nonzero elements.

It is important to note that the sparse structure of $\boldsymbol{D}$ given by its main blocks corresponds to the partial derivatives of the system dynamics with respect to the initial and final time

$$\frac{\partial h\boldsymbol{f}_k}{\partial \boldsymbol{t}_0} = -\frac{\boldsymbol{f}_k}{N-1}\ , \ \frac{\partial h\boldsymbol{f}_k}{\partial \boldsymbol{t}_f} = \frac{\boldsymbol{f}_k}{N-1} \in \mathbb{R}^{2n} \tag{21}$$

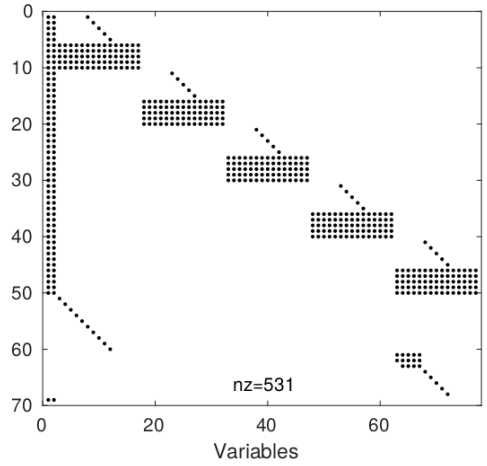


Fig. 2: Sparsity structure of $\boldsymbol{D}$ by considering the dynamics of a 5-DoF robotic finger with $N = 5$.

but, also with respect to the state and control variables. Thus, it can be deduce from (4) and (5) that

$$\frac{\partial h\boldsymbol{f}_k}{\partial \boldsymbol{x}_k} = h\begin{bmatrix} \frac{\partial \dot{\boldsymbol{q}}_k}{\partial \boldsymbol{q}_k} & \frac{\partial \dot{\boldsymbol{q}}_k}{\partial \dot{\boldsymbol{q}}_k} \\ \frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \boldsymbol{q}_k} & \frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \dot{\boldsymbol{q}}_k} \end{bmatrix} = h\begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{n\times n} \\ \frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \boldsymbol{q}_k} & \frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \dot{\boldsymbol{q}}_k} \end{bmatrix} \tag{22}$$

$$\frac{\partial h\boldsymbol{f}_k}{\partial \boldsymbol{u}_k} = h\begin{bmatrix} \frac{\partial \dot{\boldsymbol{q}}_k}{\partial \boldsymbol{u}_k} \\ \frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \boldsymbol{u}_k} \end{bmatrix} = h\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{H}(\boldsymbol{q}_k)^{-1} \end{bmatrix} \tag{23}$$

The computation of the derivatives $\frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \boldsymbol{q}_k}$, $\frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \dot{\boldsymbol{q}}_k}$ and $\frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \boldsymbol{u}_k}$ in (22) and (23) remains costly with sparse numerical differentiation. Instead, we apply geometric algorithms for multibody systems to analytically compute (5), (22) and (23) as it is explained in the next section.

IV. Geometric Algorithms

The main feature of geometric algorithms is their simple way to differentiate exponential expressions [16]. It has been possible to differentiate the recursive Recursive Newton-Euler Algorithm (RNEA) to solve the inverse dynamics problem for the minimum-effort OCP [12]. Based on RNEA [17], the computation of $\frac{\partial \ddot{\boldsymbol{q}}}{\partial \boldsymbol{x}}$ has been proposed in [18] where the inverse of the inertia matrix $\boldsymbol{H}(\boldsymbol{q})^{-1}$ is used to map from $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}$ to $\frac{\partial \ddot{\boldsymbol{q}}}{\partial \boldsymbol{x}}$. It is also known that the forward dynamics problem can be solved very efficiently by means of the Articulated-Body Algorithm (ABA) with spatial vectors [17]. In [19], it is proposed a geometric version to differentiate the ABA with respect to (1). In much the same way, we provide a geometric differentiation of the ABA, but we explicitly describe how to compute each expression when tensor products are involved.

A. Lie Groups and algebras for robot motion

The geometric elements to compute the robot's equations of motion are detailed in [20]. Here we just point out that the configuration of a rigid-body and the elements of its tangent space are defined by $\boldsymbol{G} \in SE(3)$ and $\boldsymbol{\xi} \in se(3)$, respectively,

where the Lie group $SE(3)$ contains all possible rigid-body motions, and $se(3)$ is its associated Lie algebra that contains motion elements denoted by $\boldsymbol{\xi}$, *e.g.* screw $\boldsymbol{s}$, twist $\boldsymbol{\nu}$ and spatial acceleration $\dot{\boldsymbol{\nu}}$. On the other hand, $\boldsymbol{\xi}^* \in se^*(3)$ is an element of the cotangent space acting on the rigid-body, *e.g.* wrench $\boldsymbol{F}$ and momentum $\boldsymbol{\mu}$. The transformation between elements of the group and its algebras is expressed by the adjoint operators

$$\begin{aligned} \mathrm{Ad}_G(\boldsymbol{\xi}) &= \boldsymbol{G}[\boldsymbol{\xi}]\boldsymbol{G}^{-1} && (24)\\ \mathrm{Ad}^*_G(\boldsymbol{\xi}^*) &= \boldsymbol{G}[\boldsymbol{\xi}^*]\boldsymbol{G}^{-1} && (25)\\ \mathrm{ad}_{\xi_1}(\boldsymbol{\xi}_2) &= [\boldsymbol{\xi}_1][\boldsymbol{\xi}_2] - [\boldsymbol{\xi}_2][\boldsymbol{\xi}_1] && (26)\\ \mathrm{ad}^*_{\xi}(\boldsymbol{\xi}^*) &= [\boldsymbol{\xi}^*][\boldsymbol{\xi}] \; - \; [\boldsymbol{\xi}][\boldsymbol{\xi}^*] && (27) \end{aligned}$$

where $[\cdot]$ is the matrix form of such element [20].

For robots composed by articulated rigid bodies, the local transformation between their reference frames attached to them is computed as

$$\boldsymbol{G}^i_{i-1}(q_i) = \boldsymbol{G}^i_{i-1}(0)e^{[s_i]q_i} \quad (28)$$

where i is the body frame, $e^{[s_i]q_i}$ is the matrix exponential map, $\boldsymbol{s}_i \in se(3)$ is the i-th screw axis of motion expressed in its local reference frame, and $\boldsymbol{G}^i_{i-1}(0)$ is the local transformation at robot's home posture.

The wrench $\boldsymbol{F} \in se^*(3)$ acting on a rigid-body can be calculated from the controlled Euler-Poincaré equation [21]:

$$\boldsymbol{F} = \boldsymbol{M}\dot{\boldsymbol{\nu}} - \mathrm{ad}^*_{\nu}(\boldsymbol{\mu}) \quad (29)$$

where $\boldsymbol{\nu} \in se(3)$ and $\dot{\boldsymbol{\nu}} \in se(3)$ are the twist and its time derivative while $\boldsymbol{M} \in \mathbb{R}^{6\times6}$ and $\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{\nu} \in se^*(3)$ are the rigid-body inertia matrix and its momentum, respectively.

Also, the adjoint operators establish a transformation between the Lie group and its algebra via the exponentials

$$\mathrm{Ad}_{e^{[\xi]q}} = e^{\mathrm{ad}_{\xi q}} : se(3) \to SE(3) \quad (30)$$

where $e^{[\xi]q} \in SE(3)$, $\boldsymbol{\xi} \in se(3)$ and $q \in \mathbb{R}$.

B. Forward dynamics

Let us rewrite the ABA in terms of geometric elements. As the main objective is to obtain an equation of the form

$$\boldsymbol{F}_i = \boldsymbol{M}^A_i\dot{\boldsymbol{\nu}}_i + \boldsymbol{p}^A_i \quad (31)$$

where $\boldsymbol{M}^A_i$ and $\boldsymbol{p}^A_i$ are the cumulative inertia and wrench bias, respectively, the next auxiliary variables should be defined

$$\begin{aligned} \boldsymbol{\nu}_i &= \mathrm{Ad}_{G_i^{i-1}}\boldsymbol{\nu}_{i-1} + \boldsymbol{s}_i\dot{q}_i && (32)\\ \boldsymbol{c}_i &= \mathrm{ad}_{\nu_i}\boldsymbol{s}_i\dot{q}_i \quad \text{and} \quad \boldsymbol{p}_i = -\mathrm{ad}^*_{\nu_i}\boldsymbol{M}_i\boldsymbol{\nu}_i && (33) \end{aligned}$$

where $\mathrm{Ad}_{G_i^{i-1}}$ is the Adjoint map and $G_i^{i-1} = \left(G^i_{i-1}\right)^{-1}$.

We also define the variables $\boldsymbol{U}_i$, d_i and u_i for the assembly method [17] as follows

$$\boldsymbol{U}_i = \boldsymbol{M}^A_i\boldsymbol{s}_i; \quad d_i = \boldsymbol{s}_i^\top\boldsymbol{U}_i; \quad u_i = \tau_i - \boldsymbol{s}_i^\top\boldsymbol{p}^A_i \quad (34)$$

To encapsulate the predecessors inertial effects, two new variables are defined as

$$\begin{aligned} \boldsymbol{M}^a_i &= \boldsymbol{M}^A_i - \boldsymbol{U}_i d_i^{-1}\boldsymbol{U}_i^\top && (35)\\ \boldsymbol{p}^a_i &= \boldsymbol{p}^A_i + \boldsymbol{M}^a_i\boldsymbol{c}_i + \boldsymbol{U}_i d_i^{-1}u_i && (36) \end{aligned}$$

Thus, $\boldsymbol{M}^A_i$ and $\boldsymbol{p}^A_i$ in (31) are obtained with a back-projection as

$$\begin{aligned} \boldsymbol{M}^A_i &= \boldsymbol{M}_i + \mathrm{Ad}^*_{G^i_{i+1}}\boldsymbol{M}^a_{i+1}\mathrm{Ad}_{G^i_{i+1}} && (37)\\ \boldsymbol{p}^A_i &= \boldsymbol{p}_i + \mathrm{Ad}^*_{G^i_{i+1}}\boldsymbol{p}^a_{i+1} && (38) \end{aligned}$$

Finally, the forward dynamics problem is reformulated with the previous expressions as

$$\ddot{q}_i = d_i^{-1}\left(u_i - \boldsymbol{U}_i^\top\dot{\boldsymbol{\nu}}^a_i\right) \quad (39)$$

where $\dot{\boldsymbol{\nu}}^a_i$ is a partial expression for the acceleration, then

$$\dot{\boldsymbol{\nu}}^a_i = \mathrm{Ad}_{G_i^{i-1}}\dot{\boldsymbol{\nu}}_{i-1} + \boldsymbol{c}_i \quad (40)$$

Once (39) and (40) are computed, the acceleration $\dot{\boldsymbol{\nu}}_i$ should be updated by means of $\dot{\boldsymbol{\nu}}_i = \dot{\boldsymbol{\nu}}^a_i + \boldsymbol{s}_i\ddot{q}_i$.

C. Differentiating the forward dynamics

To obtain $\frac{\partial\ddot{\boldsymbol{q}}}{\partial\boldsymbol{x}}$, the derivatives of the Adjoint operators must be computed (the details are in Appendix A of [12]) as

$$\begin{aligned} \frac{\partial\mathrm{Ad}^{-1}_G\boldsymbol{\xi}}{\partial\boldsymbol{x}} &= \mathrm{Ad}^{-1}_G\frac{\partial\boldsymbol{\xi}}{\partial\boldsymbol{x}} - \mathrm{ad}_s\mathrm{Ad}^{-1}_G\boldsymbol{\xi}\frac{\partial q}{\partial\boldsymbol{x}} && (41)\\ \frac{\partial\mathrm{Ad}^*_{G^{-1}}\boldsymbol{\xi}^*}{\partial\boldsymbol{x}} &= \mathrm{Ad}^*_{G^{-1}}\frac{\partial\boldsymbol{\xi}^*}{\partial\boldsymbol{x}} + \mathrm{Ad}^*_{G^{-1}}\mathrm{ad}^*_{-s}\boldsymbol{\xi}^*\frac{\partial q}{\partial\boldsymbol{x}} && (42) \end{aligned}$$

First, lets differentiate the twist (32) as follows

$$\frac{\partial\boldsymbol{\nu}_i}{\partial\boldsymbol{x}} = \mathrm{Ad}_{G_i^{i-1}}\frac{\partial\boldsymbol{\nu}_{i-1}}{\partial\boldsymbol{x}} - \mathrm{ad}_{s_i}\boldsymbol{\nu}_i\frac{\partial q_i}{\partial\boldsymbol{x}} + \boldsymbol{s}_i\frac{\partial\dot{q}_i}{\partial\boldsymbol{x}} \quad (43)$$

since the screw $\boldsymbol{s}_i$ is constant and $\mathrm{ad}_\xi\boldsymbol{\xi} = 0$. Note that $\frac{\partial q_i}{\partial\boldsymbol{x}} \in \mathbb{R}^{1\times2n}$ and $\frac{\partial\dot{q}_i}{\partial\boldsymbol{x}} \in \mathbb{R}^{1\times2n}$ are the i-th rows of $\frac{\partial\boldsymbol{q}}{\partial\boldsymbol{x}} \in \mathbb{R}^{n\times2n}$ and $\frac{\partial\dot{\boldsymbol{q}}}{\partial\boldsymbol{x}} \in \mathbb{R}^{n\times2n}$, respectively.

Also, the expressions (33) are differentiated as

$$\begin{aligned} \frac{\partial\boldsymbol{c}_i}{\partial\boldsymbol{x}} &= -\mathrm{ad}_{s_i}\left(\frac{\partial\boldsymbol{\nu}_i}{\partial\boldsymbol{x}}\dot{q}_i + \boldsymbol{\nu}_i\frac{\partial\dot{q}_i}{\partial\boldsymbol{x}}\right) && (44)\\ \frac{\partial\boldsymbol{p}_i}{\partial\boldsymbol{x}} &= -\mathrm{ad}^*_{\frac{\partial\boldsymbol{\nu}_i}{\partial x}}\otimes\boldsymbol{M}_i\boldsymbol{\nu}_i - \mathrm{ad}^*_{\boldsymbol{\nu}_i}\boldsymbol{M}_i\frac{\partial\boldsymbol{\nu}_i}{\partial\boldsymbol{x}} && (45) \end{aligned}$$

and the variables $\boldsymbol{U}_i$, d_i and u_i from (34) as follows

$$\begin{aligned} \frac{\partial\boldsymbol{U}_i}{\partial\boldsymbol{x}} &= \frac{\partial\boldsymbol{M}^A_i}{\partial\boldsymbol{x}}\otimes\boldsymbol{s}_i && (46)\\ \frac{\partial d_i}{\partial\boldsymbol{x}} &= \boldsymbol{s}_i^\top\frac{\partial\boldsymbol{U}_i}{\partial\boldsymbol{x}} && (47)\\ \frac{\partial u_i}{\partial\boldsymbol{x}} &= \frac{\partial\tau_i}{\partial\boldsymbol{x}} - \boldsymbol{s}_i^\top\frac{\partial\boldsymbol{p}^A_i}{\partial\boldsymbol{x}} && (48) \end{aligned}$$

where $\otimes$ is the tensor product since $\frac{\partial\boldsymbol{M}^A_i}{\partial\boldsymbol{x}} \in \mathbb{R}^{6\times6\times2n}$.

The partial derivative of $d_i^{-1} \in \mathbb{R}$ is computed as

$$\frac{\partial d_i^{-1}}{\partial\boldsymbol{x}} = -d_i^{-2}\frac{\partial d_i}{\partial\boldsymbol{x}} = -d_i^{-2}\boldsymbol{s}_i^\top\frac{\partial\boldsymbol{U}_i}{\partial\boldsymbol{x}} \quad (49)$$

Now, the variables $\boldsymbol{M}^a_i$ from (35) and $\boldsymbol{p}^a_i$ from (36) can be differentiated with multidimensional calculus as

$$\begin{aligned} \frac{\partial\boldsymbol{M}^a_i}{\partial\boldsymbol{x}} &= \frac{\partial\boldsymbol{M}^A_i}{\partial\boldsymbol{x}} - \frac{\partial\boldsymbol{U}_i}{\partial\boldsymbol{x}}\otimes d_i^{-1}\boldsymbol{U}_i^\top \\ &\quad -\boldsymbol{U}_i\boldsymbol{U}_i^\top\otimes\frac{\partial d_i^{-1}}{\partial\boldsymbol{x}} - \boldsymbol{U}_i d_i^{-1}\otimes\frac{\partial\boldsymbol{U}_i^\top}{\partial\boldsymbol{x}} && (50)\\ \frac{\partial\boldsymbol{p}^a_i}{\partial\boldsymbol{x}} &= \frac{\partial\boldsymbol{p}^A_i}{\partial\boldsymbol{x}} + \frac{\partial\boldsymbol{M}^a_i}{\partial\boldsymbol{x}}\otimes\boldsymbol{c}_i + \boldsymbol{M}^a_i\frac{\partial\boldsymbol{c}_i}{\partial\boldsymbol{x}} + \boldsymbol{U}_i d_i^{-1}\frac{\partial u_i}{\partial\boldsymbol{x}} \\ &\quad + \frac{\partial\boldsymbol{U}_i}{\partial\boldsymbol{x}}d_i^{-1}u_i + \boldsymbol{U}_i u_i\frac{\partial d_i^{-1}}{\partial\boldsymbol{x}} && (51) \end{aligned}$$

By applying (41) and (42) into (37) and (38) we get

$$\frac{\partial \boldsymbol{M}_i^A}{\partial \boldsymbol{x}} = \mathrm{Ad}^*_{G_{i+1}^i} \otimes \frac{\partial \boldsymbol{M}_{i+1}^a}{\partial \boldsymbol{x}} \otimes \mathrm{Ad}_{G_{i+1}^i} \quad (52)$$
$$+\mathrm{Ad}^*_{G_{i+1}^i}\left(\mathrm{ad}^*_{-s_{i+1}}\boldsymbol{M}_{i+1}^a - \boldsymbol{M}_{i+1}^a \mathrm{ad}_{s_{i+1}}\right)\mathrm{Ad}_{G_{i+1}^i} \otimes \frac{\partial q_{i+1}}{\partial \boldsymbol{x}}$$
$$\frac{\partial \boldsymbol{p}_i^A}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{p}_i}{\partial \boldsymbol{x}} + \mathrm{Ad}^*_{G_{i+1}^i}\left(\mathrm{ad}^*_{-s_{i+1}}\boldsymbol{p}_{i+1}^a \frac{\partial q_{i+1}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{p}_{i+1}^a}{\partial \boldsymbol{x}}\right) \quad (53)$$

Our last objective is to differentiate the equation (39). For this, we first differentiate the variable $\dot{\boldsymbol{\nu}}_i^a$ from (40) as follows

$$\frac{\partial \dot{\boldsymbol{\nu}}_i^a}{\partial \boldsymbol{x}} = \mathrm{Ad}_{G_i^{i-1}}\frac{\partial \dot{\boldsymbol{\nu}}_{i-1}}{\partial \boldsymbol{x}} - \mathrm{ad}_{s_i}\mathrm{Ad}_{G_i^{i-1}}\dot{\boldsymbol{\nu}}_{i-1}\frac{\partial q_i}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{c}_i}{\partial \boldsymbol{x}} \quad (54)$$

At this point we have all the elements to obtain $\frac{\partial \ddot{q}_i}{\partial \boldsymbol{x}}$ by means of differentiating (39) as

$$\frac{\partial \ddot{q}_i}{\partial \boldsymbol{x}} = d_i^{-1}\left(\frac{\partial u_i}{\partial \boldsymbol{x}} - \dot{\boldsymbol{\nu}}_i^{a\top}\frac{\partial \boldsymbol{U}_i}{\partial \boldsymbol{x}} - \boldsymbol{U}_i^\top \frac{\partial \dot{\boldsymbol{\nu}}_i^a}{\partial \boldsymbol{x}}\right)$$
$$+\frac{\partial d_i^{-1}}{\partial \boldsymbol{x}}\left(u_i - \boldsymbol{U}_i^\top \dot{\boldsymbol{\nu}}_i^a\right) \quad (55)$$

The acceleration differentiation is also updated with

$$\frac{\partial \dot{\boldsymbol{\nu}}_i}{\partial \boldsymbol{x}} = \frac{\partial \dot{\boldsymbol{\nu}}_i^a}{\partial \boldsymbol{x}} + \boldsymbol{s}_i \frac{\partial \ddot{q}_i}{\partial \boldsymbol{x}} \quad (56)$$

Recursively solving the last expressions in three loops, similar to ABA, $\frac{\partial \ddot{\boldsymbol{q}}}{\partial \boldsymbol{x}}$ can be retrieved, and plugged in (22). Finally, the computation of (23) is performed as suggested in [22], but with geometric operators in our implementation.

V. Simulation Results

Here, we validate the performance of our sparse trapezoidal optimization (STOPT). First, a comparison with an state-of-the-art numerical optimal control solver is performed. Then, a comparison between the sparse finite difference method versus geometric algorithms for computing the analytical Jacobian of the constraints.

The numerical simulations were performed on a laptop computer with an Intel Core i7-6500U processor and 8GB RAM. The implementation of STOPT has been written in C++ language together with the Eigen library and its sparse algebra module. We use Ipopt to solve the large-scale NLP.

A. Comparison with PSOPT [2]

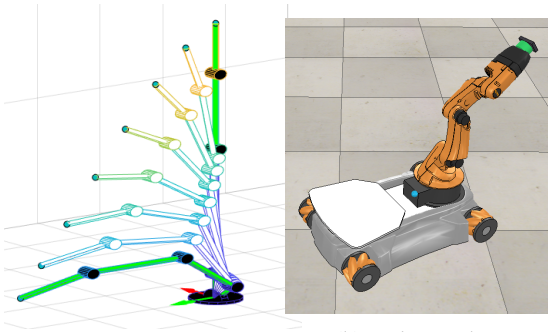
The problem to be solved is given by minimun-effort OCP with

$$w(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k) = \|\boldsymbol{u}_k\|^2, \quad (57)$$

for a finger-like robot with 5 DoF (Figure 3a). It turns out that the automatic differentiation method used in PSOPT does not support the description of complex equations of motion in a single analytical expression. Thus, the comparison was performed between the numerical differentiation method in PSOPT with the numerical and analytical differentiation methods in STOPT. In particular, the results of every evaluation are shown in TABLE I. It can be observed that PSOPT could not solve the OCP with 60 and 90 collocation points.

	Numerical				Analytical	
	PSOPT		STOPT		STOPT	
N	Iter	Time [s]	Iter	Time [s]	Iter	Time [s]
30	23	1587.5	70	77.611	65	20.573
60	-	-	74	246.046	76	77.306
90	-	-	78	522.022	66	135.07

TABLE I: PSOPT comparison



(a) 5-DoF finger-like robot (b) Kuka Youbot

Fig. 3: Robotic platforms used for the numerical simulations

B. Numerical versus analytical Jacobian computation

In order to demonstrate the performance and scalability of our proposal, a set of trajectory optimization problems with different number of collocation points were solved using geometric algorithms for evaluating the analytical Jacobian and sparse finite differences for estimating the numerical Jacobian. For all the cases, the OCP was solved with

$$w(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k) = \|\boldsymbol{q}_{k+1} - \boldsymbol{q}_k\|^2, \quad (58)$$

and an initial solution defined by the linear interpolation between the initial and final states.

1) Finger-like robot with 5 DoF: : The state equations of the first trajectory optimization problem corresponded to the robot in Figure 3a). This robot has 10 elements in the state, *i.e.*, 5 for joint positions and 5 for joint velocities, and 5 control inputs.

As a first test, single evaluations of the numerical and analytical Jacobians were performed to show the impact in the computational time when the number of collocation points increased.

N	n_{var}	n_{cns}	nz	Numerical	Analytical
30	452	311	4952	0.84 s	0.094 s
60	902	611	10052	2.95 s	0.2 s
90	1352	911	15152	5.357 s	0.28 s

TABLE II: Computational time to evaluate a single Jacobian with the finger-like robot. n_{var}, n_{cns} and nz are the number of decision variables, constraints and non-zero elements in the sparse constraint Jacobian, respectively.

The advantages to compute the analytical Jacobian with recursive geometric algorithms became evident by observing the results presented in TABLE III.

	Numerical Jacobian			Analytical Jacobian		
N	Iters	Eval	Time [s]	Iters	Eval	Time[s]
30	55	56	63.646	65	66	23.153
60	40	41	143.456	40	41	47.936
90	29	30	215.492	29	30	71.037

TABLE III: Trajectory optimization with the finger-like robot

2) Mobile manipulator with 8 DoF: The state equations were now evaluated with the KUKA youBot (see Figure 3b). The OCP considered 8 control inputs for each joint, and 16 elements for the state. As before the results of the time required of a single evaluation of the Jacobian with different number of collocation points are presented in TABLE IV. It is remarkable how the computational time to evaluate the numerical derivatives increased, while the analytical derivatives with geometric algorithms still required less than one second regardless the dimension of the problem. As it can be seen in TABLE V more NLP iterations and evaluations of the Jacobian were required which caused that the time needed for the Jacobian computation determined the time for solving the whole optimization problem.

N	n_{var}	n_{cns}	nz	Numerical	Analytical
30	722	497	11924	2.479	0.02
60	1442	977	24224	7.698	0.431
90	2162	1457	36524	16.09	0.624

TABLE IV: Computational time to evaluate a single Jacobian with KUKA youBot

	Numerical Jacobian			Analytical Jacobian		
N	Iters	Eval	Time [s]	Iters	Eval	Time[s]
30	368	369	1160.837	391	392	272.948
60	243	244	2236.503	148	149	318.436
90	150	151	2752.326	133	134	567.598

TABLE V: Trajectory optimization with KUKA youBot

3) Humanoid robot with 24 DoF: We asked STOPT to solve the trajectory optimization problem with a humanoid robot NAO. Clearly, the OCP became harder to be solved and required more time. In order to demonstrate the performance of our proposal, the state equations considered 48 elements in the state vector and 24 control inputs.

As it is seen in TABLE VI, the evaluation of the numerical Jacobian with sparse finite differences was expensive. However, the evaluation of the analytical Jacobian with geometric algorithms just required a few seconds.

N	n_{var}	n_{cns}	nz	Numerical	Analytical
30	2162	1489	105890	38.624	1.148
60	4332	2929	215330	131.532	3.02
90	6482	4369	324770	277.247	4.8

TABLE VI: Computational time to evaluate a single Jacobian with a NAO robot

The problem consists of reaching two different postures where dynamically feasible whole body motions were performed. These postures are shown in Figure 4.

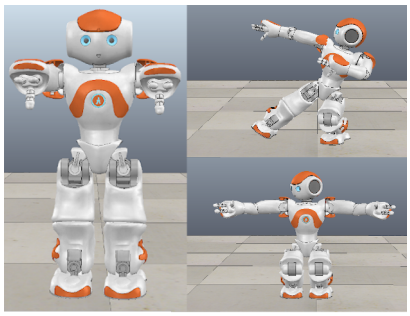


Fig. 4: Trajectory optimization problem to generate NAO postures

	Numerical Jacobian			Analytic Jacobian		
N	Iters	Eval	Time [s]	Iters	Eval	Time[s]
30	161	162	6314.451	232	233	826.657
60	164	165	21805.806	161	162	1719.301
90	92	93	25783.971	91	92	1957.938

TABLE VII: Numerical results of the first trajectory optimization problem with the NAO robot

	Numerical Jacobian			Analytic Jacobian		
N	Iters	Eval	Time [s]	Iters	Eval	Time[s]
30	229	230	8910.613	207	208	756.602
60	212	213	28215.852	225	226	2395.359
90	85	86	23934.295	73	74	1553.155

TABLE VIII: Numerical results of the second trajectory optimization problem with the NAO robot

As it is observed in TABLES VII and VIII, the use of a sparse finite difference method for estimating the constraint Jacobian was inefficient, which had a direct consequence in the overall time to solve the large-scale NLP. However, the use of the analytic Jacobian with the proposed geometric differentiation of ABA and the sparsity exploitation of the constraint Jacobian with trapezoidal collocation, saved huge amounts of time in finding solutions.

VI. Conclusion

We presented an efficient method that reduces the time for the computation of the Jacobian of the constraints required for solving a trajectory optimization problem, this reduction impacts in a large scale the computation time required to find a solution. In this proposal we take advantage of the separability and the sparsity of the trapezoidal collocation method in order to reach a sparse block structure where each part of the Jacobian can be easily identified and computed in an analytical way, in this case we suggest the use of the geometric algorithms in order to compute the most complex derivatives i.e. the equation of motions w.r.t to the states and controls. In order to probe the efficiency and scalabilty of our method a set of experiments with different robotic platforms are perform and different number of collocation point are performed. For seek of comparison a Here is easy to see that the use of the suggested analytical Jacobian computation saves a lot of computation time, allowing to find a solution of a complex humanoid robot in less than 30 minutes.

REFERENCES

[1] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkitan open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[2] Victor M Becerra. Solving complex optimal control problems at no cost with psopt. In *2010 IEEE International Symposium on Computer-Aided Control System Design*, pages 1391–1396. IEEE, 2010.

[3] Yunus M Agamawi and Anil V Rao. Cgpops: A c++ software for solving multiple-phase optimal control problems using adaptive gaussian quadrature collocation and sparse nonlinear programming. *arXiv preprint arXiv:1905.11898*, 2019.

[4] Daniel B Leineweber, Andreas Schäfer, Hans Georg Bock, and Johannes P Schlöder. An efficient multiple shooting based reduced sqp strategy for large-scale dynamic process optimization: Part ii: software aspects and applications. *Computers & chemical engineering*, 27(2):167–174, 2003.

[5] J.-T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Advances in Design and Control. SIAM, 2nd edition, 2010.

[6] Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.

[7] Mathieu Geisert and Nicolas Mansard. Trajectory generation for quadrotor based systems using numerical optimal control. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 2958–2964. IEEE, 2016.

[8] Anna Lena Emonds Née Kleesattel and Katja Mombaur. Inverse optimal control based enhancement of sprinting motion analysis with and without running-specific prostheses. In *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*, pages 556–562. IEEE, 2018.

[9] Matthew Kelly. An introduction to trajectory optimization: how to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[10] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[11] Ayonga Hereid, Christian M Hubicki, Eric A Cousineau, and Aaron D Ames. Dynamic humanoid locomotion: A scalable formulation for hzd gait optimization. *IEEE Transactions on Robotics*, 34(2):370–387, 2018.

[12] Alvaro Paz and Gustavo Arechavaleta. Practical guide to solve the minimum-effort problem with geometric algorithms and b-splines. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6720–6726. IEEE, 2019.

[13] Michael A Patterson and Anil V Rao. Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems. *Journal of Spacecraft and Rockets*, 49(2):354–377, 2012.

[14] J. Nocedal and S.-J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[15] Alan R Curtis, Michael JD Powell, and John K Reid. On the estimation of sparse jacobian matrices. *J. Inst. Math. Appl*, 13(1):117–120, 1974.

[16] Frank C Park, Beobkyoon Kim, Cheongjae Jang, and Jisoo Hong. Geometric algorithms for robot dynamics: A tutorial review. *Applied Mechanics Reviews*, 70(1):010803, 2018.

[17] R. Featherstone. *Rigid Body Dynamics Algorigthms*. Springer-Verlag, New York, 2008.

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

[19] Garett A Sohl and James E Bobrow. A recursive multibody dynamics and sensitivity algorithm for branched kinematic chains. *Journal of Dynamic Systems, Measurement, and Control*, 123(3):391–399, 2001.

[20] F.-C. Park, J.-E. Bobrow, and S.-R. Ploen. A lie group formulation of robot dynamics. *The International Journal of Robotics Research*, 14(6):609–618, 1995.

[21] Jerrold E Marsden and Tudor S Ratiu. *Introduction to mechanics and symmetry: a basic exposition of classical mechanical systems*, volume 17. Springer Science & Business Media, 2013.

[22] Justin Carpentier. Analytical inverse of the joint space inertia matrix. *Technical report, Laboratoire dAnalyse et dArchitecture des Systemes, URL https://hal.laas.fr/hal-01790934*, 2018.