Int. J. Appl. Math. Comput. Sci., 2025, Vol. 35, No. 4, 577–589
DOI: 10.61822/amcs-2025-0041

TRAJECTORY OPTIMIZATION FOR HIGHLY ARTICULATED ROBOTS BASED ON SPARSITY–FREE LOCAL DIRECT COLLOCATION

DANIEL CARDONA-ORTIZ[a], GUSTAVO ARECHAVALETA[a,*]

[a] Robotics and Advanced Manufacturing Group
Centre for Research and Advanced Studies of IPN
Saltillo, Coahuila, México
e-mail: {daniel.cardona,gustavo.arechavaleta}@cinvestav.mx

In this paper, we introduce a numerical optimal control scheme (NOCS) for generating dynamically feasible robot motions under several constraints while optimizing a given performance criterion. In particular, the NOCS transforms continuous optimal control problems into large-scale sparsity-free nonlinear programs (NLPs) by means of a dedicated strategy called the block indexation procedure (BIP). As a result, the optimized open-loop control law is obtained fast under limited-memory allocation. The robot's equations of motion, and their partial derivatives with respect to the state of the robot and control inputs, are analytically evaluated. For this, state-of-the-art algorithms available in the Pinocchio and RBDL open-source libraries are used. Otherwise, the NOCS applies the BIP with numerical differentiation techniques. The effectiveness of the NOCS is numerically validated with different robots composed by many degrees of freedom. Also, we provide performance comparisons against CasADi, a popular general purpose optimal control framework that applies automatic differentiation.

Keywords: robot trajectory optimization, numerical optimal control, direct collocation.

1. Introduction

Numerical optimal control schemes (NOCSs) have become popular in the area of robot motion generation and control. In part, this is due to the maturity of nonlinear programming (NLP) given that direct transcription methods heavily rely on NLP solvers (Biegler and Zavala, 2009; Gill *et al*., 2005; Byrd *et al*., 2006). The aim of direct transcription is to transform a given continuous optimal control problem into its discrete version (Rao, 2009; Betts, 2010; Kelly, 2017).

Significant work has been done to develop sophisticated NOCSs based on direct transcription with either multiple shooting or collocation such as the ACADO toolkit (Houska *et al*., 2011), DirCol5i (Kelly, 2019), MUSCOD-II (Leineweber *et al*., 2003), PSOPT (Becerra, 2010), CasADi (Andersson *et al*., 2019) and CGPOPS (Agamawi and Rao, 2020). They are able to solve small to medium size optimal control problems in terms of the complexity of the system dynamics and associated constraints. In particular, the optimal control problems are mainly related to chemical (Leineweber *et al*., 2003) and aerospace (Betts and Erb, 2003) engineering. In addition, the ACADO toolkit is no longer supported while MUSCOD-II and CGPOPS represent sophisticated commercial options. Concerning the PSOPT and CasADi solvers, they are open-source projects that are capable to tackle a wide variety of trajectory optimization problems, but they cannot deal with highly articulated robots such as humanoids. An important limitation is the use of automatic differentiation to analytically evaluate the corresponding cost function, nonlinear constraints and their partial derivatives with respect to decision variables. Thus, the underlying data structures ask for a considerable amount of memory allocation.

In this article, we propose a specialized NOCS[1] for generating dynamically feasible robot motions by exploiting the structure of local direct collocation to cope with its inherent sparsity. It takes advantage of the properties of low order Gauss–Lobatto collocation methods (Herman and Conway, 1996), i.e., trapezoidal and Hermite–Simpson. Also, it performs an efficient analytical evaluation of the robot's equations of motion

*Corresponding author

[1] https://github.com/daniel-cardona/NOCS.

and their linearization to build the vector of defect constraints and, more relevant, the constraint Jacobian, which is a highly sparse matrix evaluated at will by the numerical optimization solver (Betts, 2010; Kelly, 2017; Cardona-Ortiz *et al*., 2020). A key component, named the block indexation procedure (BIP), is introduced to efficiently handle the sparsity generated by direct collocation. Different from available general purpose frameworks, the proposed NOCS is able to generate challenging motions for a wide variety of robots with many degrees of freedom (DoFs) such as mobile manipulators and humanoid robots. Also, our NOCS has been implemented in C++ programming language as an open-source project available at `https://github.com/daniel-cardona/NOCS`.

1.1. Problem formulation. Let us consider a given robotic system composed by a tree-like kinematic structure with n DoFs. Then, the state of the robot stands for

$$\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{q}(t) \\ \dot{\boldsymbol{q}}(t) \end{bmatrix} \in \mathbb{R}^{n_s}, \tag{1}$$

where $n_s = 2n$, $\{\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)\} \in \mathbb{R}^n$ represent the joint configuration and velocity profiles, respectively, along time t. The problem consists in finding a control law $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ (if it exists) to generate the sequence of robot motions from a robot's initial state to a final one by solving

$$\underset{\boldsymbol{x}(t),\boldsymbol{u}(t)}{\text{minimize}} \quad \int_{t_i}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t))\, \mathrm{d}t \tag{2}$$

$$\begin{aligned} \text{subject to} \quad & \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \\ & \boldsymbol{g}_L \leq \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \leq \boldsymbol{g}_U, \\ & \boldsymbol{x}_L \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_U, \\ & \boldsymbol{u}_L \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_U, \\ & \boldsymbol{x}(t_i) = \boldsymbol{x}_i, \\ & \boldsymbol{x}(t_f) = \boldsymbol{x}_f, \end{aligned}$$

where $L : \mathbb{R}^{n_s} \times \mathbb{R}^{n_u} \to \mathbb{R}$ stands for the criterion to be optimized along the trajectory from initial time t_i to final time t_f. The set of path constraints is given by $\boldsymbol{g} : \mathbb{R}^{n_s} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_p}$, $\{\boldsymbol{g}_L, \boldsymbol{g}_U\} \in \mathbb{R}^p$ are the lower and upper bounds of path constraints, $\{\boldsymbol{x}_L, \boldsymbol{x}_U\} \in \mathbb{R}^{n_s}$ are the lower and upper bounds for the state vector, $\{\boldsymbol{u}_L, \boldsymbol{u}_U\} \in \mathbb{R}^{n_u}$ are the lower and upper bounds of the control inputs, and $\{\boldsymbol{x}_i, \boldsymbol{x}_f\} \in \mathbb{R}^{n_s}$ are the initial and final states. The system dynamics of a robot have the following form:

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \begin{bmatrix} \dot{\boldsymbol{q}}(t) \\ \ddot{\boldsymbol{q}}(t) \end{bmatrix} \in \mathbb{R}^{n_s}, \tag{3}$$

with

$$\ddot{\boldsymbol{q}}(t) = \boldsymbol{H}(\boldsymbol{q}(t))^{-1} \left(\boldsymbol{u}(t) - \boldsymbol{h}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))\right), \tag{4}$$

where $\boldsymbol{H} : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ denotes the inertia matrix, $\boldsymbol{h}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) \triangleq \boldsymbol{C}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))\dot{\boldsymbol{q}} - \boldsymbol{N}(\boldsymbol{q}(t))$ is the vector of nonlinear effects such as Coriolis and centrifugal forces $\boldsymbol{C} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ and gravitational terms $\boldsymbol{N} : \mathbb{R}^n \to \mathbb{R}^n$.

It is difficult to obtain closed-form expressions for the optimal control laws, or knowledge of their form (if it exists), when nonlinear dynamics are considered in the problem (2). Realistically, then, the use of numerical techniques such as transcription methods should be employed to compute approximated optimal trajectories. In these terms, direct collocation transforms the problem (2) into a large-scale NLP of the form

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & \psi(\boldsymbol{z}) \\ \text{subject to} \quad & \boldsymbol{c}_L \leq \boldsymbol{c}(\boldsymbol{z}) \leq \boldsymbol{c}_U, \\ & \boldsymbol{z}_L \leq \boldsymbol{z} \leq \boldsymbol{z}_U, \end{aligned} \tag{5}$$

where $\boldsymbol{z}$ is the vector of decision variables. Commonly, it contains the state $\boldsymbol{x}(t_k)$ and control $\boldsymbol{u}(t_k)$ evaluated along the discrete time sequence t_k with $k = 1, 2, \ldots, N_c$, where N_c is the number of grid points. The cost function to be minimized is $\psi(\boldsymbol{z}) : \mathbb{R}^{n_z} \to \mathbb{R}$ and $\boldsymbol{c}(\boldsymbol{z}) : \mathbb{R}^{n_z} \to \mathbb{R}^{n_c}$ is the vector of constraints. The vectors $\{\boldsymbol{z}_L, \boldsymbol{z}_U\} \in \mathbb{R}^{n_z}$ stand for the lower and upper bounds associated to the vector of decision variables, and $\{\boldsymbol{c}_L, \boldsymbol{c}_U\} \in \mathbb{R}^{n_c}$ represent the lower and upper bounds of constraints.

Numerical optimization solvers (e.g., Ipopt, SNOPT, KNITRO) are capable of solving the problem (5). However, they require first order information on (5) to numerically converge to a local optimum. Thus, it is critical to efficiently compute the gradient of the cost function defined as

$$\nabla_z \psi(\boldsymbol{z}) \triangleq \frac{\partial \psi}{\partial \boldsymbol{z}} \in \mathbb{R}^{n_z}, \tag{6}$$

together with the constraint Jacobian defined as

$$\boldsymbol{J}(\boldsymbol{z}) \triangleq \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{z}} \in \mathbb{R}^{n_c \times n_z}, \tag{7}$$

which is a highly sparse matrix (Betts, 2010).

1.2. Related work. Trajectory planning for linear systems has been widely studied (Liu *et al*., 2022). However, the majority of robots turn out to be nonlinear. Thus, a popular direct shooting method known as differential dynamic programming (DDP), introduced by (Mayne, 1966), has attracted the attention of roboticists for generating dynamically feasible robot motions (Tassa *et al*., 2014; Budhiraja *et al*., 2018; Howell *et al*., 2019; Mastalli *et al*., 2020). Some of the reasons are that DDP consists of two compact algorithms and, contrary to direct collocation methods, it is sparsity-free by design. However, it has been initially formulated to

satisfy the system dynamics while optimizing a given performance criterion. In other words, path constraints together with simple state and control bounds cannot be added to the problem in a straightforward manner. As a consequence, several extensions have been proposed to consider additional constraints such as maintaining dynamic balance of robot locomotion, hybrid modes due to contact interactions between the robot and its environment, and obstacle avoidance, among others. Currently, there is no consensus on which variant of DDP should be applied to generate robot motions under nonlinear path constraints.

On the other hand, direct collocation has been applied to solve trajectory optimization problems subject to a wide variety of constraints that are naturally found in robotic tasks (Posa *et al*., 2014; Hereid *et al*., 2018; Cardona-Ortiz *et al*., 2020). Posa *et al*. (2014) proposed a direct collocation strategy to consider linear complementarity constraints that are useful to generate robot manipulation and locomotion behaviors under contact interactions with the environment. Although the optimal control problem is highly nonlinear and difficult to solve for planar robots, the number of decision variables and constraints remains relatively small. The direct collocation strategy proposed by Hereid *et al*. (2018), which is based on a modified Hermite–Simpson method, is able to generate 3D humanoid walking gait. The size of the problem clearly increased, which implies an intense computation to get the solution. The reason is mainly due to the number of defect constraints, which play an important role to satisfy the system dynamics along the optimized trajectory. Also, the evaluation of first order information of the problem is based on symbolic computation (Hereid *et al*., 2018). Thus, it is necessary to re-execute the symbolic computation each time the system dynamics and constraints suffer from modifications, which represents a hard limitation.

Different strategies have been suggested to compute exact or approximated large and sparse constraint Jacobians. Curtis *et al*. (1974) proposed a sparse finite differences (SFDs) method. It is based on group perturbation such that each group contains no constant elements of the sparse matrix. SFD has been integrated in some NOCSs (Becerra, 2010; Cardona-Ortiz *et al*., 2020). However, its approximate nature introduces truncation errors which often lead to poor convergence. Recently, geometric recursive algorithms have been proposed within the discrete mechanics and optimal control framework, known as DMOC (Villanueva-Piñon and Arechavaleta, 2025), to overcome sparsity while evaluating analytically (6) and (7).

Automatic differentiation (AD) (Weinstein and Rao, 2017; Walther *et al*., 2003) is a well known alternative to analytically evaluate (6) and (7). It uses complex data structures to perform scalar operations by following the basic rules of differentiation. General purpose optimal control solvers such as CasADi (Andersson *et al*., 2019), CGPOPS (Agamawi and Rao, 2020) and PSOPT (Becerra, 2010) take advantage of AD. However, its use comes at the cost of memory consumption and computational overhead, specially for robots with many DoFs. In contrast, there exist specialized and very efficient recursive algorithms to analytically compute the partial derivatives of the robot's equations of motion with respect to the state and control inputs, i.e., decision variables (Carpentier and Mansard, 2018; Paz and Arechavaleta, 2023; Singh *et al*., 2024). Such algorithms turn out to be less demanding than AD (Carpentier *et al*., 2019).

In our preliminary work (Cardona-Ortiz *et al*., 2020), dedicated recursive geometric algorithms were proposed to evaluate the equations of motion of robotic systems together with their analytical partial differentiation with respect to the state and control inputs. Also, we detected sparsity for trapezoidal collocation with a block insertion strategy. In this paper, we introduce a novel indexation procedure, the so-called BIP, to efficiently handle large and sparse matrices. In addition, the sparsity detection is extended to Hermite–Simpson collocation and mesh refinement is integrated to improve accuracy. The BIP works with either analytical or numerical differentiation methods. Consequently, the important bottleneck to evaluate (6) and (7) is mitigated. As a result, the proposed NOCS outperforms the state-of-the-art trajectory optimization solvers, and it facilitates the possibility to consider highly articulated mechanisms such as humanoid robots.

This paper is organized as follows. Section 2 briefly recalls the key elements of local direct collocation to transcribe an optimal control problem into an NLP. In Section 3, we introduce the proposed BIP to handle the involved sparse matrices to evaluate the constraint Jacobian. Section 4 details the proposed NOCS. Then, Section 5 shows the numerical evaluation of the BIP and NOCS. Some concluding remarks are given in Section 6.

2. Local direct collocation

Let us briefly recall the direct transcription process based on local collocation techniques, i.e., trapezoidal and Hermite–Simpson. These methods allow the continuous-time problem to be approximated by a finite number of decision variables. First, the discretization is performed by dividing the total duration in N_c collocation points as

$$t_i = t_1 < t_2 < \ldots < t_k < \ldots < t_{N_c} = t_f, \quad (8)$$

where $k \in \{1, 2, \ldots, N_c\}$ iterates along the discrete time intervals, and $\{t_i, t_f\} \in \mathbb{R}$ are the initial and final time,

respectively. Commonly, the time horizon is mapped to a fixed domain $[0, 1]$ as follows:

$$\tau_k = \frac{t_k - t_i}{\Delta T} \quad \text{for} \quad k = 1, \ldots, N_c, \tag{9}$$

where $\Delta T = t_f - t_i$. Now, the continuous-time state and control inputs, $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$, respectively, are discretized according to the collocation technique as described below.

2.1. Trapezoidal collocation method. Trapezoidal collocation implies that the performance index (2) and system dynamics (3) are both approximated as piecewise linear functions. First, the continuous state $\boldsymbol{x}(t)$ is discretized according to the sequence of collocation points in time, i.e., $\boldsymbol{x}_k = \boldsymbol{x}(t_k)$. Then, the discrete control inputs are discretized in a similar manner, i.e., $\boldsymbol{u}_k = \boldsymbol{u}(t_k)$. Thus, the vector of decision variables gathers the following data:

$$\boldsymbol{z} = [t_i \; t_f \; \boldsymbol{z}_1^T \; \ldots \; \boldsymbol{z}_{N_c}^T]^T \in \mathbb{R}^{n_z}, \tag{10}$$

where each stacked vector $\boldsymbol{z}_k = [\boldsymbol{x}_k^T \; \boldsymbol{u}_k^T]^T$ contains the corresponding discrete state and control inputs at the k-th collocation point.

The performance index (2), which represents the Lagrange term of the optimal control problem, is approximated by means of the trapezoidal integration rule as follows:

$$\psi(\boldsymbol{z}) = \sum_{k=1}^{N_c-1} \frac{h_k}{2}(L_{k+1} + L_k), \tag{11}$$

where $h_k = \Delta\tau_k \Delta T$, $\Delta\tau_k = \tau_{k+1} - \tau_k$ such that $\tau_k \in [0, 1]$. The evaluation of the Lagrange term at each collocation point is represented as $L_k = L(\boldsymbol{x}_k, \boldsymbol{u}_k)$. The evolution over time of the system dynamics (3) should be expressed as a set of collocation constraints as follows:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - \frac{h_k}{2}(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k) = \boldsymbol{0}, \tag{12}$$

where $\boldsymbol{f}_k = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ for $k = 1, \ldots, N_c$.

2.2. Hermite–Simpson collocation method. Hermite–Simpson approximates the performance index and system dynamics as piecewise quadratic functions. Thus, it provides higher-order accuracy compared to trapezoidal collocation. However, more collocation points are involved. In addition to regular collocation points, interior points should be considered, which are located at $\bar{t}_k = \frac{t_{k+1} - t_k}{2}$. The vector of decision variables gathers more data due to midpoint evaluations of the state and control inputs as follows:

$$\boldsymbol{z} = [t_i \; t_f \; \boldsymbol{z}_1^T \; \bar{\boldsymbol{z}}_2^T \; \boldsymbol{z}_2^T \; \ldots \; \bar{\boldsymbol{z}}_{N_c}^T \; \boldsymbol{z}_{N_c}^T]^T, \tag{13}$$

where each stacked vector $\bar{\boldsymbol{z}}_k = [\bar{\boldsymbol{x}}_k^T \; \bar{\boldsymbol{u}}_k^T]^T$ contains the state and control inputs evaluated at $\bar{t}_k$. Then, the performance index is approximated by applying the Simpson quadrature as follows:

$$\psi(\boldsymbol{z}) = \sum_{k=1}^{N_c-1} \frac{h_k}{6}(L_{k+1} + 4\bar{L}_k + L_k), \tag{14}$$

where $\bar{L}_k = L(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$ stands for the Lagrange term evaluated at $\bar{t}_k$.

Unlike the trapezoidal collocation method, the Hermite–Simpson one requires two collocation constraints for discretizing the system dynamics over time. The first one applies the Simpson quadrature,

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - \frac{h_k}{6}(\boldsymbol{f}_{k+1} + 4\bar{\boldsymbol{f}}_k + \boldsymbol{f}_k) = \boldsymbol{0}, \tag{15}$$

where $\bar{\boldsymbol{f}}_k = \boldsymbol{f}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$. The second, named the Hermite interpolant, enforces the system dynamics at $\bar{t}_k$ as

$$\bar{\boldsymbol{x}}_{k+1} - \frac{1}{2}(\boldsymbol{x}_{k+1} + \boldsymbol{x}_k) - \frac{h_k}{8}(\boldsymbol{f}_k - \boldsymbol{f}_{k+1}) = \boldsymbol{0}. \tag{16}$$

We are using the separated form of Hermite–Simpson discretization. However, there exists an alternative known as compressed form, but the inherent sparsity of the problem is not fully exploited (Betts, 2010).

2.3. Cost function and its gradient. The transcription of the performance index (2), with either the trapezoidal (11) or the Hermite–Simpson (14), method takes the following form:

$$\psi(\boldsymbol{z}) = c\boldsymbol{w}^T \boldsymbol{L}(\boldsymbol{z}), \tag{17}$$

where $\boldsymbol{L}(\boldsymbol{z}) \triangleq [L_1 \; \cdots \; L_k \; \cdots \; L_{N_c}]^T$ is the vector containing the Lagrange term evaluations over collocation points, $\boldsymbol{w} \triangleq [w_1 \; \cdots \; w_k \; \cdots \; w_{N_c}]^T$ is a vector of weights, and c is a constant. For trapezoidal collocation, $c = \frac{\Delta T}{2}$, and $c = \frac{\Delta T}{6}$ for Hermite–Simpson one. The vector of weights has the following structure:

$$\begin{aligned} w_1 &= \Delta\tau_1, \\ w_k &= \Delta\tau_{k+1} + \Delta\tau_k \text{ for } k = 2, \ldots, N_c - 1, \\ w_{N_c} &= \Delta\tau_{N_c-1}. \end{aligned}$$

Notice that $\boldsymbol{w}$ should contain interior points for the Hermite–Simpson collection, i.e., $\bar{w}_k = 4\Delta\tau_k$.

The gradient of the cost function (17) stands for

$$\nabla_z \psi(\boldsymbol{z}) = \begin{bmatrix} \nabla_{t_i}\psi \\ \nabla_{t_f}\psi \\ \nabla_{z_1}\psi \\ \vdots \\ \nabla_{z_k}\psi \\ \vdots \\ \nabla_{z_{N_c}}\psi \end{bmatrix}, \tag{18}$$

where

$$\nabla_{t_f}\psi = \frac{1}{a}\boldsymbol{w}^T\boldsymbol{L}(\boldsymbol{z}), \tag{19}$$

$$\nabla_{t_i}\psi = -\frac{1}{a}\boldsymbol{w}^T\boldsymbol{L}(\boldsymbol{z}), \tag{20}$$

$$\nabla_{z_k}\psi = \frac{\Delta T w_k}{a}\left[\nabla_{x_k}\psi^T \;\; \nabla_{u_k}\psi^T\right]^T, \tag{21}$$

such that $a = 2$ for the trapezoidal method. For the Hermite–Simpson one, $a = 6$, and additional partial differentiation terms should be considered, i.e., $\nabla_{\bar{z}_k}\psi$ at interior collocation points $\bar{t}_k$.

2.4. Defect, path and boundary constraints. The vector of constraints $\boldsymbol{c}(\boldsymbol{z})$ of the NLP (5) can be expressed as (Betts, 2010)

$$\boldsymbol{c}(\boldsymbol{z}) = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{y}(\boldsymbol{z}), \tag{22}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are sparse constant matrices, while $\boldsymbol{y} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_c}$ is a vector composed by the following nonlinear functions:

$$\boldsymbol{y}(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{\Phi}(\boldsymbol{z}) \\ \boldsymbol{g}(\boldsymbol{z}) \\ \boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_{N_c}) \end{bmatrix}, \tag{23}$$

with $\boldsymbol{\Phi}(\boldsymbol{z})$ containing the set of defect constraints, which corresponds to the discretization of the system dynamics in terms of the trapezoidal or Hermite–Simpson collocation methods. The separability property can be applied to rewrite the trapezoidal collocation constraints (12) as

$$\boldsymbol{\Phi}_k = \boldsymbol{A}_k \begin{bmatrix} \boldsymbol{z}_k \\ \boldsymbol{z}_{k+1} \end{bmatrix} + \boldsymbol{B}_k \begin{bmatrix} \Delta T \boldsymbol{f}_k \\ \Delta T \boldsymbol{f}_{k+1} \end{bmatrix} = \boldsymbol{0}, \tag{24}$$

where

$$\boldsymbol{A}_k = \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0}_{n_s \times n_u} & \boldsymbol{I} & \boldsymbol{0}_{n_s \times n_u} \end{bmatrix} \in \mathbb{R}^{n_s \times 2(n_s + n_u)}, \tag{25}$$

$$\boldsymbol{B}_k = \begin{bmatrix} -\frac{\Delta\tau_k}{2}\boldsymbol{I} & -\frac{\Delta\tau_k}{2}\boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{n_s \times 2n_s}, \tag{26}$$

with $\boldsymbol{I} \in \mathbb{R}^{n_s \times n_s}$ as an identity matrix. The overall structure of sparse constant matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ for trapezoidal collocation is illustrated in Fig. 1.

Similarly, we employed the separated form of Hermite–Simpson collocation that imposes two defect constraints, which correspond to the Simpson quadrature (15) and the Hermite interpolant (16). Both constraints can be rewritten as

$$\boldsymbol{\Phi}_k = \boldsymbol{A}_k \begin{bmatrix} \boldsymbol{z}_k \\ \bar{\boldsymbol{z}}_k \\ \boldsymbol{z}_{k+1} \end{bmatrix} + \boldsymbol{B}_k \begin{bmatrix} \Delta T \boldsymbol{f}_k \\ \Delta T \bar{\boldsymbol{f}}_k \\ \Delta T \boldsymbol{f}_{k+1} \end{bmatrix} = \boldsymbol{0}, \tag{27}$$

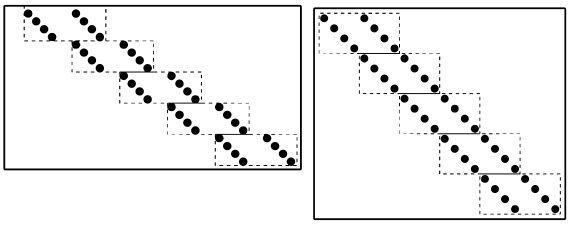

Fig. 1. Structure for matrices $\boldsymbol{A}$ (left) and $\boldsymbol{B}$ (right) for the trapezoidal scheme with $n_s = 4$ and $n_u = 2$. Each block defined by a dotted rectangle is known as $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ for $k = 1, \ldots, 5$, starting from top to bottom.

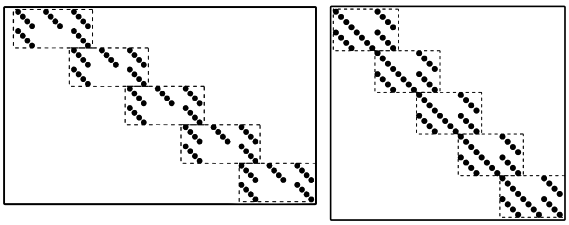

Fig. 2. Structure for matrices $\boldsymbol{A}$ (left) and $\boldsymbol{B}$ (right) for the Hermite–Simpson scheme with $n_s = 4$ and $n_u = 2$. Each block defined by a dotted rectangle is known as $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ for $k = 1, \ldots, 5$, starting from top to bottom.

where the constant block matrices $\boldsymbol{A}_k \in \mathbb{R}^{2n_s \times 3(n_s+n_u)}$ and $\boldsymbol{B}_k \in \mathbb{R}^{2n_s \times 3n_s}$ are

$$\boldsymbol{A}_k = \begin{bmatrix} -\frac{1}{2}\boldsymbol{I} & \boldsymbol{0}_{n_u} & \boldsymbol{I} & \boldsymbol{0}_{n_u} & -\frac{1}{2}\boldsymbol{I} & \boldsymbol{0}_{n_u} \\ -\boldsymbol{I} & \boldsymbol{0}_{n_u} & \boldsymbol{0}_{n_s \times n_s} & \boldsymbol{0}_{n_u} & -\boldsymbol{I} & \boldsymbol{0}_{n_u} \end{bmatrix},$$

$$\boldsymbol{B}_k = \begin{bmatrix} -\frac{\Delta\tau_k}{8}\boldsymbol{I} & \boldsymbol{0}_{n_s \times n_s} & \frac{\Delta\tau_k}{8}\boldsymbol{I} \\ -\frac{\Delta\tau_k}{6}\boldsymbol{I} & -\frac{2\Delta\tau_k}{3}\boldsymbol{I} & -\frac{\Delta\tau_k}{6}\boldsymbol{I} \end{bmatrix}.$$

In this case, the sparsity pattern of constant matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is shown in Fig. 2.

In addition, path constraints are also stacked in (23) as $\boldsymbol{g}(\boldsymbol{z})$, and the vector $\boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_{N_c})$ represents the boundary constraints.

2.5. Constraint Jacobian. The constraint Jacobian given in (7) is obtained by partially differentiating (22) with respect to the vector of decision variables such that

$$\boldsymbol{J}(\boldsymbol{z}) = \boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}(\boldsymbol{z}), \tag{28}$$

where $\boldsymbol{D}(\boldsymbol{z}) \triangleq \frac{\partial \boldsymbol{y}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ is the partial derivative of $\boldsymbol{y}(\boldsymbol{z})$ given in (23) with respect to the vector of decision variables. Its structure is as follows:

$$\boldsymbol{D}(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{D}_t & \boldsymbol{D}_f \\ \boldsymbol{D}_{e_t} & \boldsymbol{D}_{e_x} \\ \boldsymbol{0} & \boldsymbol{D}_g \end{bmatrix}, \tag{29}$$

where

$$\boldsymbol{D}_t = \begin{bmatrix} -\boldsymbol{f}_1 & \boldsymbol{f}_1 \\ -\boldsymbol{f}_2 & \boldsymbol{f}_2 \\ \vdots & \vdots \\ -\boldsymbol{f}_{N_c} & \boldsymbol{f}_{N_c} \end{bmatrix} \tag{30}$$

gathers the partial derivative of defect constraints $\boldsymbol{\Phi}(\boldsymbol{z})$ with respect to the initial and final time. The next block contains the partial derivative of event constraints $\boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_{N_c})$ with respect to the initial and final time, respectively,

$$\boldsymbol{D}_{e_t} = \begin{bmatrix} \dfrac{\partial \boldsymbol{e}}{\partial t_i} & \dfrac{\partial \boldsymbol{e}}{\partial t_f} \end{bmatrix}. \tag{31}$$

Then, the sparse block matrix $\boldsymbol{D}_f$ is the partial derivative of defect constraints $\boldsymbol{\Phi}(\boldsymbol{z})$ with respect to the state and control inputs at each collocation point. It has the following sparsity pattern:

$$\boldsymbol{D}_f = \begin{bmatrix} \dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}_1} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}_2} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}_{N_c}} \end{bmatrix}, \tag{32}$$

where

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}_k} = \Delta T \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{n\times n} & \boldsymbol{0} \\ \dfrac{\partial \ddot{\boldsymbol{q}}_k}{\partial \boldsymbol{q}_k} & \dfrac{\partial \ddot{\boldsymbol{q}}_k}{\partial \dot{\boldsymbol{q}}_k} & \boldsymbol{H}(\boldsymbol{q}_k)^{-1} \end{bmatrix}. \tag{33}$$

Therefore, the robot forward dynamics have to be differentiated with respect to the state and control inputs. The analytical evaluation of (4) and (33) can be efficiently performed with Pinocchio, which is an out-of-the-box robotics library (Carpentier *et al*., 2019). There exist other implementations such as the RBDL (Felis, 2017) to analytically evaluate (4), and more recently Paz and Arechavaleta (2023) as well as Singh *et al*. (2024) also computed analytically (4) and (33).

The remaining sparse block matrices in (29) correspond to the partial derivative of event and path constraints with respect to the state and control inputs as follows:

$$\boldsymbol{D}_{e_x} = \begin{bmatrix} \dfrac{\partial \boldsymbol{e}}{\partial \boldsymbol{x}_1} & \boldsymbol{0} & \dots & \dfrac{\partial \boldsymbol{e}}{\partial \boldsymbol{x}_{N_c}} & \boldsymbol{0} \end{bmatrix}, \tag{34}$$

$$\boldsymbol{D}_g = \begin{bmatrix} \dfrac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}_1} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \dfrac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}_2} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \dfrac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}_{N_c}} \end{bmatrix}. \tag{35}$$

It is then of great importance to compute the constraint Jacobian (28) very efficiently. Although the use of robotics packages to analytically obtain (4), $\partial \ddot{\boldsymbol{q}}_k / \partial \boldsymbol{q}_k$ and $\partial \ddot{\boldsymbol{q}}_k / \partial \dot{\boldsymbol{q}}_k$ helps to reduce the computational cost, the arithmetic complexity to compute the underlying sparse matrix addition and product in (28) can be dramatically reduced.

3. Sparsity-free constraint Jacobian

The aim here is to efficiently evaluate the constraint Jacobian (28). First, the involved matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{D}(\boldsymbol{z})$ should be rewritten as lists of triples to only gather the position of their nonzero entries, i.e., row index, column index and value. Then, the lists of triples allow performing sparsity-free matrix addition and multiplication to evaluate (28) by means of a numerical sparse linear algebra package available in the work of Guennebaud and Benoît (2023). Thus, the explicit computation of (28) is avoided.

We propose a block indexation procedure (BIP) to build the lists of triples. In particular, the BIP exploits the separability property of local collocation together with the known sparsity patterns, which are deduced from the number of collocation points N_c, state variables n_s, and control inputs n_u.

First, we define the list of triples of a given sparse matrix $\boldsymbol{A}$ as a matrix with the following structure:

$$\boldsymbol{T}_A \quad = [\boldsymbol{r},\ \boldsymbol{c},\ \boldsymbol{v}] \in \mathbb{R}^{m\times 3}, \tag{36}$$

where m is the number of nonzero entries of $\boldsymbol{A}$ while the column vectors $\{\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{v}\} \in \mathbb{R}^{m\times 1}$ store the row indexes, column indexes and the associated nonzero entries of $\boldsymbol{A}$, respectively. In addition, the use of the colon operator(:) is employed for indexing and the accessing the data of $\boldsymbol{T}_A \in \mathbb{R}^{m\times 3}$ similar to the Matlab language style. Thus, $\boldsymbol{T}_A[:, 2]$ refers to the second column of $\boldsymbol{T}_A$.

The aim of the BIP is to build the representation of a given sparse matrix $\boldsymbol{A}$ in terms of a list of triples as (36). As described in Section 2, the sparsity pattern of the constaint Jacobian in local collocation methods can be deduced in advance. Therefore, the BIP makes use of such prior knowledge to efficiently build the list of triples.

Let us consider the sparse constant matrix $\boldsymbol{A}$ in trapezoidal collocation with its sparsity pattern illustrated in Fig. 1 (left). It is observed that it contains a sequence of blocks, $\boldsymbol{A}_1, \dots, \boldsymbol{A}_k, \dots, \boldsymbol{A}_{N_c}$, with the same sparsity pattern. Thus, it is sufficient to generate only one list of triples for one block by applying a dedicated algorithm, available in the linear algebra library Eigen V.3.3.4 (Guennebaud and Benoît, 2023), that efficiently iterates over rows and columns of the block matrix to extract the nonzero entries and store them as triples. Then, N_c copies of the generated list of triples are stacked in lexicographical order $\{\boldsymbol{T}_{A_1}, \dots, \boldsymbol{T}_{A_k}, \dots, \boldsymbol{T}_{A_{N_c}}\}$. The

Algorithm 1. Block indexation procedure.

Input: Ordered set of triple lists $\{\boldsymbol{T}_{A_1}, \ldots, \boldsymbol{T}_{A_{N_c}}\}$, row shift r_s and location Δ_r, column shift c_s and location Δ_c

1 Get number of triples m from $\boldsymbol{T}_{A_1}$
2 $\boldsymbol{T}_A = \{\}$
3 $\boldsymbol{e}_s = [1\ 1\ \ldots\ 1]^T \in \mathbb{R}^{m\times 1}$
4 **for** $k = 1$ **to** $k = N_c$ **do**
5 $\quad \boldsymbol{r}_s \leftarrow \boldsymbol{e}_s\,((k-1)r_s + 1) + \Delta_r$
6 $\quad \boldsymbol{c}_s \leftarrow \boldsymbol{e}_s\,((k-1)c_s + 1) + \Delta_c$
7 $\quad \boldsymbol{T}_{A_k}[:,1] \leftarrow \boldsymbol{T}_{A_k}[:,1] + \boldsymbol{r}_s$
8 $\quad \boldsymbol{T}_{A_k}[:,2] \leftarrow \boldsymbol{T}_{A_k}[:,2] + \boldsymbol{c}_s$
9 $\quad \boldsymbol{T}_A \leftarrow$ queued $(\boldsymbol{T}_{A_k})$
/* Return list of triples of sparse matrix $\boldsymbol{A}$ */
10 **return** $\boldsymbol{T}_A$

Table 1. BIP inputs for $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{D}_f$, $\boldsymbol{D}_g$, $\boldsymbol{D}_t$, $\boldsymbol{D}_{e_t}$ and $\boldsymbol{D}_{e_x}$.

Matrix	r_s	c_s	Δ_r	Δ_c
$\boldsymbol{A}$	cn_s	$c(n_s+n_u)$	0	0
$\boldsymbol{B}$	cn_s	cn_s	0	0
$\boldsymbol{D}_f$	n_s	n_s+n_u	0	2
$\boldsymbol{D}_g$	n_p	n_s+n_u	$n_sN_c+n_e+1$	2
$\boldsymbol{D}_t$	0	0	0	0
$\boldsymbol{D}_{e_t}$	0	0	n_sN_c+1	0
$\boldsymbol{D}_{e_x}$	0	0	n_sN_c+1	2

next step consists of shifting the row index and column index of each $\boldsymbol{T}_{A_k}$ to fit with the sparsity pattern of the associated container matrix $\boldsymbol{A}$.

Algorithm 1 details the block indexation procedure. The inputs of Algorithm 1 are the row shift r_s and column shift c_s. These input parameters for sparse constant matrix $\boldsymbol{A}$, illustrated in Fig. 1(left), correspond to the first row of Table 1, where n_s is the number of state variables, n_u is the number of control inputs, $c = 1$ for the trapezoidal methods or $c = 2$ for the Hermite–Simpson one.

In a similar manner, the BIP is applied to stack $\{\boldsymbol{T}_{B_1}, \ldots, \boldsymbol{T}_{B_k}, \ldots, \boldsymbol{T}_{B_{N_c}}\}$, i.e., the N_c copies of a list of triples corresponding to one block of constant matrix $\boldsymbol{B}$ as illustrated in Fig. 1 (right). The input parameters for this case correspond to the second row of Table 1. Although the nonzero entries of matrices $\boldsymbol{D}_f$ and $\boldsymbol{D}_g$ given in (32) and (35), respectively, are not constant, their sparsity patterns do not change. Thus, the BIP also builds their representations as a stack of copies of a list of triples for a given block of the container matrix, i.e., $\boldsymbol{D}_f$ and $\boldsymbol{D}_g$. The third and fourth rows of Table 1 show the inputs for block diagonal matrices of defect and path constraints, $\boldsymbol{D}_f$ and $\boldsymbol{D}_g$, respectively, where n_p is the number of path constraints and n_e is the number of event constraints. The last two columns of Table 1, represented as Δ_r and Δ_c, stand for the location of a sparse block within matrix $\boldsymbol{D}$ defined in (29).

Notice that the BIP also generates the stack of lists of triples of matrices $\boldsymbol{D}_t$, $\boldsymbol{D}_{e_t}$ and $\boldsymbol{D}_{e_x}$ according to the input parameters given in Table 1.

The generation of $\boldsymbol{T}_A, \boldsymbol{T}_B$ and $\boldsymbol{T}_D$ by means of the BIP allows performing the sparse matrix addition and multiplication operations to evaluate the constraint Jacobian:

$$\boldsymbol{T}_J = \boldsymbol{T}_A \oplus \boldsymbol{T}_B \otimes \boldsymbol{T}_D, \tag{37}$$

where $\oplus$ and $\otimes$ refer to sparse-sparse matrix addition and multiplication, respectively. Both operators are available in the linear algebra library Eigen V.3.3.4 (Guennebaud and Benoît, 2023).

4. Optimization strategy

Algorithm 2 outlines the main steps of the proposed sparsity-free optimization strategy based on local direct collocation.

In line 1 of Algorithm 2, the optimal control problem (2) is discretized according to the trapezoidal or the Hermite–Simpson method for obtaining the constrained nonlinear program (5). To determine the sparsity pattern of the constraint Jacobian in line 3, the nonzero entries of lists of triples $\boldsymbol{T}_A[:,3], \boldsymbol{T}_B[:,3]$ and $\boldsymbol{T}_D[:,3]$ are set to 1. Then, the computation of (37) gives the list of triples $\boldsymbol{T}_J$. It is important to notice that BIP Algorithm 1 is called only once to compute $\boldsymbol{T}_A, \boldsymbol{T}_B$ due to the fact their nonzero entries are constant. Concerning the generation of the list of triples $\boldsymbol{T}_D$ by BIP Algorithm 1, it can be observed that it is only needed to update the nonzero entries $\boldsymbol{T}_D[:,3]$ as many times as the optimization solver asks. Since the sparsity pattern of the container matrix $\boldsymbol{D}(\boldsymbol{z})$ does not change, $\boldsymbol{T}_D[:,1]$ and $\boldsymbol{T}_D[:,2]$ remain the same.

5. Numerical results

In this section, we validate the performance of the proposed sparsity-free optimization solver based on local direct collocation. First, we compare the proposed BIP algorithm against sparse finite differences (Curtis *et al.*, 1974; Becerra, 2010) and automatic differentiation (Andersson *et al.*, 2019). Then, we compare the computational cost for optimizing robot movements against CasADi (Andersson *et al.*, 2019), which is a popular general purpose numerical optimal control solver. All evaluations are focused on synthesizing dynamically feasible robot motions. Different robotic platforms were considered for evaluating the scalability of the proposed NOCS when the size of the underlying NLP increased. In particular, we discussed a humanoid robot NAO with 24 degrees of freedom.

Algorithm 2. Sparsity-free optimization strategy.

Input: $\boldsymbol{x}_i$ and $\boldsymbol{x}_f$, initial and final states, respectively. Initial and final time t_i, t_f and number of grid points N_c

/* From OCP to NLP */

1 Transcribe problem (2) into (5) according to either trapezoidal or Hermite–Simpson (H–S) collocation method

/* Compute randomly initial guess */

2 Set initial solution $\boldsymbol{z}_0$: trapezoidal (10) or H–S (13)

/* Generate lists of triples based on sparsity patterns of contraint Jacobian matrices */

3 Call Algorithm 1 to build $\boldsymbol{T}_A$, $\boldsymbol{T}_B$ and $\boldsymbol{T}_D$

4 Compute $\boldsymbol{T}_J$ with (37)

/* Solve NLP */

5 $\boldsymbol{z}^* \leftarrow$ numOptSolver$(\boldsymbol{z}_0, \boldsymbol{T}_J, costFcn, cnsFcn)$

6 **return** $\boldsymbol{z}^*$

7

8 **Function** $costFcn(\boldsymbol{w}, \boldsymbol{z})$**:**

/* Compute cost function (17) */

9 $\psi(\boldsymbol{z}) \leftarrow c\boldsymbol{w}^T \boldsymbol{L}(\boldsymbol{z})$

/* Compute gradient of $\psi(\boldsymbol{z})$ (18) */

10 $\nabla_z \psi \leftarrow$ queued $(\nabla_{t_i} \psi)$

11 $\nabla_z \psi \leftarrow$ queued $(\nabla_{t_f} \psi)$

12 **for** $k = 1$ **to** $k = N_c$ **do**

13 Extract $\boldsymbol{z}_k$ from $\boldsymbol{z}$

/* Compute $\nabla_{z_k} \psi$ with (21) */

14 $\nabla_z \psi \leftarrow$ queued $(\nabla_{z_k} \psi)$

15 return $\psi, \nabla_z \psi$

16

17 **Function** $cnsFcn(\boldsymbol{T}_A, \boldsymbol{T}_B, \boldsymbol{z})$**:**

18 Evaluate vector of non-linear functions $\boldsymbol{y}(\boldsymbol{z})$

/* Sparsity-free evaluation of non-linear constraints by means of sparse matrix-vector addition and multiplication operators */

19 $\boldsymbol{c} \leftarrow \boldsymbol{T}_A \oplus \boldsymbol{z} + \boldsymbol{T}_B \otimes \boldsymbol{y}(\boldsymbol{z})$

/* Sparsity-free evaluation of constraint Jacobian (29) */

20 Update $\boldsymbol{T}_D[:, 3]$ with partial differentiation (30)–(35)

21 Compute $\boldsymbol{T}_J$ with (37)

/* Return constraints vector $\boldsymbol{c}$ and triples list values of Jacobian matrix $\boldsymbol{T}_J[:, 3]$ */

22 return $\boldsymbol{c}, \boldsymbol{T}_J[:, 3]$

The experiments were implemented in the C++ programming language. The NLP solver was Ipopt V3.14.11 (Biegler and Zavala, 2009), the numerical linear algebra was solved by means of Eigen V.3.3.4 (Guennebaud and Benoît, 2023), and the computation of the robot's equations of motion and their partial differentiation with respect to the state and control inputs was performed with the very efficient library known as Pinocchio (Carpentier *et al*., 2019). All experiments were executed on a standard laptop with 16 GB of RAM and an Intel i7 processor running at 2.60 GHz. The source code associated with the proposed NOCS is freely available at `https://github.com/daniel-cardona/NOCS`.

5.1. BIP performance. To evaluate the performance of the BIP, we executed Algorithm 2 with some modifications. In particular, Table 2 shows four methods employed to compute the constraint Jacobian in the function $cnsFcn$ of Algorithm 2. The method named BIP-N uses central finite differences to update $\boldsymbol{T}_D[:, 3]$ in line 20 of Algorithm 2. In contrast, BIP-A made use of the Pinocchio library to update $\boldsymbol{T}_D[:, 3]$ with analytical differentiation.

For all cases, we considered the kinematic structure of a snake-like robot composed by 3–36 DoFs. The criterion to be minimized was related to minimum-length trajectories: $||\boldsymbol{q}(t + \Delta t) - \boldsymbol{q}(t)||^2$. Only defect constraints related to the system dynamics were considered. Hermite–Simpson collocation was applied to transcribe the OCP into an NLP with 15, 30 and 60 collocation points. Also, it is important to mention that the average time was obtained by executing 100 times the same problem. Figure 3 reports the execution time.

It can be observed that BIP-N outperformed SFD as the number of DoFs increased. For a robot with 36 DoFs and 60 collocation points, BIP-N took an average of 374 ms while SFD employed 7 s. In addition, ADA was consistently faster than BIP-N, by around 17%. However, ADA cannot handle robots with more than 36 DoFs due to a demanding memory consumption. In contrast, BIP-N successfully handled a large-scale NLP with snake-like robots composed by 100 DoFs. As expected, the more efficient method was BIP-A, which consistently required less computation time than ADA. For problems considering the same robots with 60 collocation points, BIP-A took an average of 50 ms to compute the constraint Jacobian, whereas ADA took 212 ms. Overall, BIP-A was approximately four times faster than ADA.

The performance of the BIP was evaluated with more challenging problems in terms of the robotic platform and the number of collocation points. A humanoid robot NAO (see Fig. 4) with 24 DoFs was considered, along 120 collocation points. Four trajectory optimization problems were solved. All of them assume that the root

Table 2. Methods used to compute the constraint Jacobian for Algorithm 2: BIP-N refers to block indexation procedure with central finite differences to compute the values of the list of triples $\boldsymbol{T}_D$, BIP-A refers to block indexation procedure with analytical differentiation based on the Pinocchio library to compute the values of the list of triples $\boldsymbol{T}_D$.

Abbreviation	Algorithm
SFD	Sparse finite differences (Curtis *et al.*, 1974; Becerra, 2010)
ADA	CasADi's automatic differentiation (Andersson *et al.*, 2019)
BIP-N	Numerical block indexation
BIP-A	Analytical block indexation

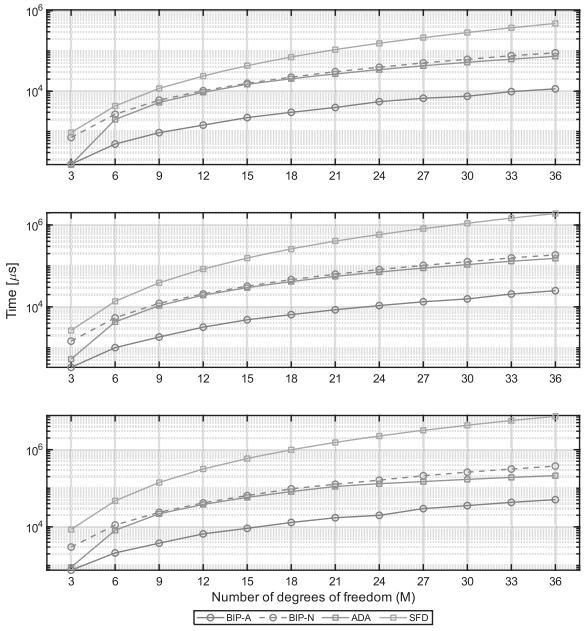

Fig. 3. Performance comparison to get the constraint Jacobian. Each graph shows the computational time for snake-like robots with increased DoFs. The top graph corresponds to 15 collocation points, the middle one to 30 collocation points and the bottom one was obtained with 60 collocation points. BIP-A outperformed the other alternatives to evaluate the constraint Jacobian.

reference frame of NAO is fixed, and it is attached to one foot. Contact dynamics are activated in the simulation environment where the humanoid model is not fixed to the ground. Thus, the resulting optimized motions are directly sending a signal to the humanoid in the simulation environment for validating that it is balanced. Otherwise, the humanoid falls and hits the ground. To prevent this, the robot's centroidal momentum is considered as a path constraint in the optimization problem to maintain the dynamic equilibrium of the humanoid while it is performing the motions (cf. Paz and Arechavaleta, 2024). Snapshots for two of them are depicted in Fig. 4.

We compared the performance of the BIP with two state-of-the-art rigid-body dynamics libraries: Pinocchio (Carpentier *et al.*, 2019) and the RBDL (Felis, 2017). The results are reported in Table 3. The first row implies

Table 3. BIP with Pinocchio and the RBDL. Each problem was solved 100 times to obtain the displayed average time.

Algorithm	Time (s)			
	Pose 1	Pose 2	Pose 3	Pose 4
Pinnochio (BIP-A)	33.138	50.701	45.716	58.394
Pinocchio (BIP-N)	76.427	137.69	77.915	103.549
RBDL (BIP-N)	77.143	136.177	98.949	117.095

the use of Pinocchio to compute both the set of defect constraints and to update the sparsity-free constraint Jacobian in lines 18 and 20 of Algorithm 2, respectively. The second row also applied Pinocchio to evaluate line 18, but central finite differences were used to compute the values of the list of triples $\boldsymbol{T}_D$ in line 20. The third row applied the RBDL to evaluate line 18 and numerical partial differentiation in line 20, similar to the second row. As expected, the PIB, together with Pinocchio, got the lowest computational time.

5.2. NOCS for robot trajectory optimization. We evaluated the performance of the proposed Algorithm 2 with three articulated robots. In addition, we compared the computational time to get the solution against the optimal control framework CasADi (Andersson *et al.*, 2019). The optimization problems were formulated as

$$\min_{\boldsymbol{x}(t),\boldsymbol{u}(t)} \quad \int_0^{t_f} \phi(\boldsymbol{x}(t),\boldsymbol{u}(t))\,\mathrm{d}t, \tag{38}$$

subject to system dynamics

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{q}}(t) \\ \boldsymbol{H}(\boldsymbol{q}(t))^{-1}(\boldsymbol{u}(t) - \boldsymbol{h}(\boldsymbol{q}(t),\dot{\boldsymbol{q}}(t)) \end{bmatrix} \in \mathbb{R}^{n_s} \tag{39}$$

as well as boundary constraints $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and $\boldsymbol{x}(t_f) = \boldsymbol{x}_f$. Three performance criteria were used (see Table 4). As is observed, we selected a standard robot manipulator UR5, a dual-arm robot and a humanoid robot NAO to demonstrate the versatility of the proposed NOCS for generating optimized motions with different articulated rigid-body systems.

All optimization problems applied Hermite–Simpson collocation, and the resulting NLP is solved by means

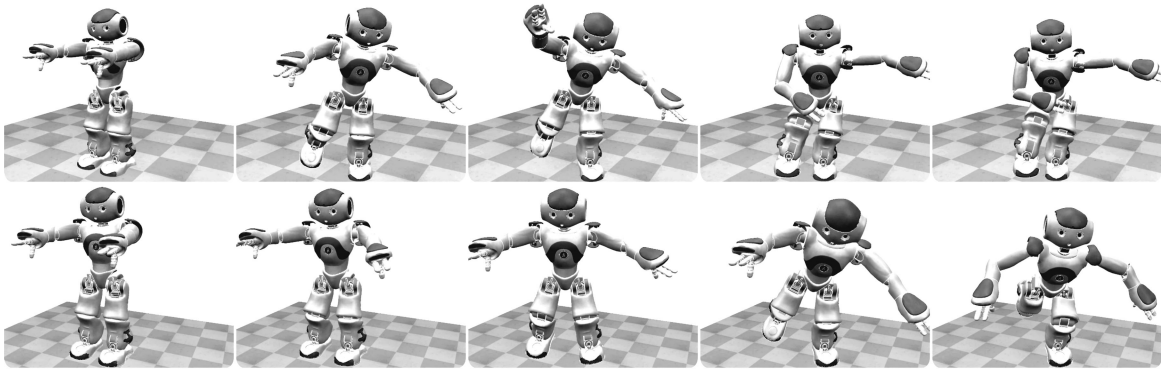

Fig. 4. NAO robot performed equilibrium tasks. Top-row: the robot was asked to move its arms while standing on one leg. Bottom-row: the robot was asked to reach an airplane-like posture.

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{q}}(t) \\ \boldsymbol{H}(\boldsymbol{q}(t))^{-1}\left(\boldsymbol{u}(t) + F_c(t)\boldsymbol{J}_g(\boldsymbol{q}(t))^T - \boldsymbol{h}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))\right) \\ \dot{x}_d(t) \\ \dot{y}_d(t) \\ \frac{F_c(t)}{m_1} - c\dot{x}_d(t) \\ \frac{F_c(t)}{m_1} - c\dot{y}_d(t) \end{bmatrix}, \tag{40}$$

$$\begin{aligned} g(x(\boldsymbol{q}(t)), y(\boldsymbol{q}(t)), x_d(t), y_d(t)) &\geq 0, \\ F_c(t) &\geq 0, \\ g(x(\boldsymbol{q}(t)), y(\boldsymbol{q}(t)), x_d(t), y_d(t))F_c(t) &= 0. \end{aligned}$$

of Ipopt with default parameters. The initial guess is generated via linear interpolation between the initial and desired robot states.

In Fig. 5, the computational time is reported. As is observed, the size of the NLP increased in terms of the robot's DoFs and the number of collocation points: 30, 45 and 60. The label NOCS implies the use of Pinocchio (Carpentier *et al.*, 2019) in line 20 of Algorithm 2. The label NOCS-Numerical applied central finite differences to compute the partial differentiation of the robot's equations of motion. As expected, the NOCS outperformed the CasADi library. It took at least 50% less time for all cases.

In addition, Table 5 shows the reached cost, number of iterations, constraint violation and required time to get optimized trajectories with 30, 60 and 120 collocation points. For this test, UR5 was considered, i.e., the first row of Table 4. The first section of Table 5 corresponds to the NOCS with BIP-A while the second one to the CasADi library. It can be observed that the main difference was the required time to get the solutions.

5.3. Complementarity constraints in the NOCS. Complementarity constraints were considered in the NOCS for solving simple contact interactions between a robot and its environment as proposed by Posa *et al.* (2014). In particular, we asked a 3-DoF planar manipulator to interact with a small disc for displacing it from its current position to a desired one. The cost functional (38) corresponds to the minimum effort criterion in terms of the L_2 norm of the control $||\boldsymbol{u}(t)||^2$ subject to the constrained dynamics expressed as (40), where the state vector $\boldsymbol{x}(t)$ contains the joint coordinates of the manipulator $\boldsymbol{q}(t) = [q_1(t)\ q_2(t)\ q_3(t)]^T \in \mathcal{T}^3$, its time derivative $\dot{\boldsymbol{q}}(t)$, and the coordinates of the disc together with its time derivative, $(x_d(t), y_d(t))$ and $(\dot{x}_d(t), \dot{y}_d(t))$, respectively. The magnitude of the contact force in the normal direction with respect to the circumference of the disc is represented by $F_c(t)$, $\boldsymbol{J}_g(\boldsymbol{q}(t))$ is the contact constraint gradient, c stands for the friction constant, and m_1 is the mass of the

Table 4. Different performance criteria.

Robot model	DoF(n_s)	$\phi(\boldsymbol{x}(t), \boldsymbol{u}(t))$
UR5	6 (12)	$\|\|\boldsymbol{q}(t+1) - \boldsymbol{q}(t)\|\|^2$
ABB YuMi	14 (28)	$\|\|\boldsymbol{u}(t)\|\|^2$
NAO	24 (48)	$\|\|\dot{\boldsymbol{q}}(t)\|\|^2$

Table 5. Performance comparison for the UR5 robot.

	NOCS with BIP-A and Ipopt			
N_c	Cost	Number of iterations	Constraint violation	Required time (s)
30	4.54×10^2	105	8.36×10^{-10}	2.3
60	4.89×10^2	125	1.78×10^{-10}	4.2
120	3.55×10^2	200	3.43×10^{-9}	6.38
	CasADi library with Ipopt			
N_c	Cost	Number of iterations	Constraint violation	Required time (s)
30	4.70×10^2	100	3.01×10^{-10}	2.93
60	5.33×10^2	125	1.9×10^{-11}	7.42
120	4.3×10^2	213	5.97×10^{-9}	24.86

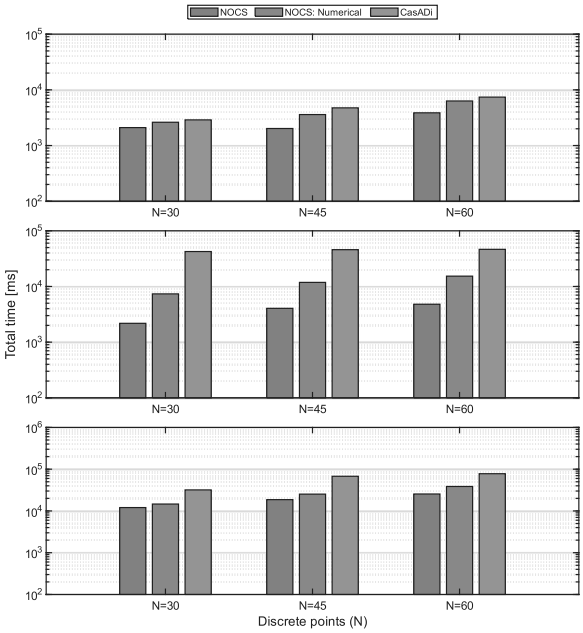

Fig. 5. Computational time comparison between the NOCS and the CasADi library. The selected models are: the UR5 robot (top), the YuMi robot (middle) and the NAO robot (bottom).

disc. The subsequent constraints are the complementarity conditions between the manipulator's end-effector and the circumference of the disc:

$$g(x(\boldsymbol{q}), y(\boldsymbol{q}), x_d, y_d) = (x(\boldsymbol{q}) - x_d)^2 + (y(\boldsymbol{q}) - y_d)^2 - r^2,$$

where $(x(\boldsymbol{q}), y(\boldsymbol{q}))$ is the position of the end-effector and r is the radius of the disc. Figure 6 illustrates the behavior.

5.4. Mesh refinement in the NOCS. An important consideration when solving trajectory optimization problems for robotic systems is to guarantee accurate solutions for ensuring the feasibility of system dynamics along the optimized trajectory. Thus, the proposed NOCS incorporates classical algorithms of mesh refinement (Betts, 2010; Patterson *et al.*, 2015). This works as follows. The accuracy of the initial optimized trajectory obtained with Algorithm 2 is evaluated. This is done by calculating an error of the approximated and the ideal system dynamics between collocation points. If the error is greater than a predefined threshold (e.g., 1×10^{-4}), then new collocation points are added to the problem. As a consequence, the size of the NLP increases. Next, Algorithm 2 takes the previous optimized trajectory as the initial solution to solve the new NLP. This process is repeated until the accuracy threshold is reached.

Table 6. Mesh refinement iterations applied to the ABB YuMi robot.

N_c	Collocation method	Error	Time [s]
15	H–S	0.01957	0.808
29	H–S	$2.91\times^{-3}$	1.16
57	H–S	$2.33\times^{-4}$	2.2
66	H–S	$9.26\times^{-5}$	2.6
Total time			6.78

Table 6 illustrates the mesh refinement in the NOCS for the ABB YuMi robot. The refinement algorithm terminated when the error was below 1×10^{-4}. Note that it took less than 7 s, and only required 66 collocation points.

6. Conclusions

We proposed a complete numerical strategy to solve robot trajectory optimization problems. The robot motion problem was formulated as a classical optimal control one where a given performance index should be optimized while satisfying the system dynamics and additional constraints. In contrast to available solvers, we were able to generate motions of highly articulated robots with many degrees of freedom by both exploiting the inherent sparsity of local collocation and evaluating analytically the defect constraints along

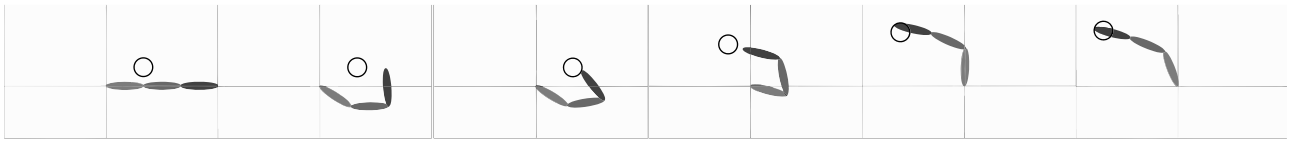

Fig. 6. Snapshots of the optimized motions of a robot that makes contact with a disc for moving it to a desired position.

with their partial differentiation with respect to decision variables. One of the key elements was the proposed block propagation procedure. The numerical evaluation and comparison of the proposed NOCS with state-of-the-art solvers demonstrate the contributions of the paper. The source-code is also available to reproduce the reported results and to be used by the community. Currently, we are extending the NOCS to global collocation methods.

Acknowledgment

Acknowledges the financial support of the National Council of Humanities, Science and Technology (CONACHyT) under the scholarship no. 943100.

References

Agamawi, Y.M. and Rao, A.V. (2020). CGPOPS: A C++ software for solving multiple-phase optimal control problems using adaptive Gaussian quadrature collocation and sparse nonlinear programming, *ACM Transactions on Mathematical Software* **46**(3): 1–38.

Andersson, J.A., Gillis, J., Horn, G., Rawlings, J.B. and Diehl, M. (2019). CASADI: A software framework for nonlinear optimization and optimal control, *Mathematical Programming Computation* **11**(1): 1–36.

Becerra, V.M. (2010). Solving complex optimal control problems at no cost with PSOPT, *2010 IEEE International Symposium on Computer-Aided Control System Design, Yokohama, Japan*, pp. 1391–1396.

Betts, J.T. (2010). *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM, Philadelphia.

Betts, J.T. and Erb, S.O. (2003). Optimal low thrust trajectories to the moon, *SIAM Journal on Applied Dynamical Systems* **2**(2): 144–170.

Biegler, L.T. and Zavala, V.M. (2009). Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization, *Computers & Chemical Engineering* **33**(3): 575–582.

Budhiraja, R., Carpentier, J., Mastalli, C. and Mansard, N. (2018). Differential dynamic programming for multi-phase rigid contact dynamics, *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids), Beijing, China*, pp. 1–9.

Byrd, R.H., Nocedal, J. and Waltz, R.A. (2006). KNITRO: An integrated package for nonlinear optimization, *in* G. Di Pillo and M. Roma (Eds), *Large-Scale Nonlinear Optimization*, Springer, Boston, pp. 35–59.

Cardona-Ortiz, D., Paz, A. and Arechavaleta, G. (2020). Exploiting sparsity in robot trajectory optimization with direct collocation and geometric algorithms, *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 469–475, (Virtual Event).

Carpentier, J. and Mansard, N. (2018). Analytical derivatives of rigid body dynamics algorithms, *Robotics: Science and Systems (RSS 2018), Pittsburgh, USA*.

Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O. and Mansard, N. (2019). The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives, *2019 IEEE/SICE International Symposium on System Integration (SII), Paris, France*, pp. 614–619.

Curtis, A.R., Powell, M.J. and Reid, J.K. (1974). On the estimation of sparse Jacobian matrices, *IMA Journal of Applied Mathematics* **13**(1): 117–120.

Felis, M.L. (2017). RBDL: An efficient rigid-body dynamics library using recursive algorithms, *Autonomous Robots* **41**(2): 495–511.

Gill, P.E., Murray, W. and Saunders, M.A. (2005). SNOPT: An SQP algorithm for large-scale constrained optimization, *SIAM Review* **47**(1): 99–131.

Guennebaud, G. and Benoît, J. (2023). Eigen v3.3.4, `https://gitlab.com/libeigen/eigen/$-$/tree/3.3`.

Hereid, A., Hubicki, C.M., Cousineau, E.A. and Ames, A.D. (2018). Dynamic humanoid locomotion: A scalable formulation for HZD gait optimization, *IEEE Transactions on Robotics* **34**(2): 370–387.

Herman, A.L. and Conway, B.A. (1996). Direct optimization using collocation based on high-order Gauss–Lobatto quadrature rules, *Journal of Guidance, Control, and Dynamics* **19**(3): 592–599.

Houska, B., Ferreau, H.J. and Diehl, M. (2011). ACADO Toolkit—An open-source framework for automatic control and dynamic optimization, *Optimal Control Applications and Methods* **32**(3): 298–312.

Howell, T.A., Jackson, B.E. and Manchester, Z. (2019). ALTRO: A fast solver for constrained trajectory optimization, *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China*, pp. 7674–7679.

Kelly, M. (2017). An introduction to trajectory optimization: How to do your own direct collocation, *SIAM Review* **59**(4): 849–904.

Kelly, M.P. (2019). DIRCOL5I: Trajectory optimization for problems with high-order derivatives, *Journal of Dynamic Systems, Measurement, and Control* **141**(3): 034502.

Leineweber, D.B., Schäfer, A., Bock, H.G. and Schlöder, J.P. (2003). 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.

Liu, G., Wu, S., Zhu, L., Wang, J. and Lv, Q. (2022). Fast and smooth trajectory planning for a class of linear systems based on parameter and constraint reduction, *International Journal of Applied Mathematics and Computer Science* **32**(1): 11–21, DOI: 10.34768/amcs-2022-0002.

Mastalli, C., Budhiraja, R., Merkt, W., Saurel, G., Hammoud, B., Naveau, M., Carpentier, J., Righetti, L., Vijayakumar, S. and Mansard, N. (2020). Crocoddyl: An efficient and versatile framework for multi-contact optimal control, *2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France*, pp. 2536–2542.

Mayne, D. (1966). A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems, *International Journal of Control* **3**(1): 85–95.

Patterson, M.A., Hager, W.W. and Rao, A.V. (2015). A PH mesh refinement method for optimal control, *Optimal Control Applications and Methods* **36**(4): 398–421.

Paz, A. and Arechavaleta, G. (2023). Analytical differentiation of the articulated-body algorithm: A geometric multilinear approach, *Multibody System Dynamics* **60**: 347–373.

Paz, A. and Arechavaleta, G. (2024). Humanoid trajectory optimization with b-splines and analytical centroidal momentum derivatives, *International Journal of Humanoid Robotics* **21**(02): 2350020.

Posa, M., Cantu, C. and Tedrake, R. (2014). A direct method for trajectory optimization of rigid bodies through contact, *International Journal of Robotics Research* **33**(1): 69–81.

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

Singh, S., Russell, R.P. and Wensing, P.M. (2024). On second-order derivatives of rigid-body dynamics: Theory & implementation, *IEEE Transactions on Robotics* **40**: 2233–2253.

Tassa, Y., Mansard, N. and Todorov, E. (2014). Control-limited differential dynamic programming, *2014 IEEE International Conference on Robotics and Automation (ICRA), Hong-Kong, China*, pp. 1168–1175.

Villanueva-Piñon, C. and Arechavaleta, G. (2025). DMOC-based robot trajectory optimization with analytical first-order information, *International Journal of Applied Mathematics and Computer Science* **35**(1): 83–96, DOI: 10.61822/amcs-2025-0007.

Walther, A., Griewank, A. and Vogel, O. (2003). ADOL-C: Automatic differentiation using operator overloading in C++, *Proceedings in Applied Mathematics and Mechanics*, **2**: 41–44.

Weinstein, M.J. and Rao, A.V. (2017). Algorithm 984: ADiGator, a toolbox for the algorithmic differentiation of mathematical functions in Matlab using source transformation via operator overloading, *ACM Transactions on Mathematical Software* **44**(2): 1–25.

Daniel Cardona-Ortiz received his MS degree in robotics and advanced manufacturing from CINVESTAV, Saltillo, Mexico, in 2019. Currently, he is a PhD candidate working on numerical optimal control algorithms for generating humanoid robot motions. Special attention in his research is given to software design, efficiency and scalability of numerical optimization methods applied to robotic and artificial intelligence systems.

Gustavo Arechavaleta received his MS degree from Tecnológico de Monterrey (ITESM), Campus Estado de México, in 2003, and his PhD degree from Institut National des Sciences Appliquees de Toulouse, France, for his work on the computational principles of human walking via optimal control within the GEPETTO Group, Laboratoire d'Analyse et d'Architecture des Systemes, CNRS, Toulouse, France, in 2007. In 2008 he joined the Robotics and Advanced Manufacturing Group at CINVESTAV, Saltillo, Mexico, where he is a researcher. In 2015 he spent a year in the Coordinated Science Laboratory at the University of Illinois at Urbana–Champaign as a visiting researcher. His research interests are mainly focused on robot trajectory optimization and vision-based robot navigation.

Received: 6 February 2025
Revised: 7 June 2025
Accepted: 15 July 2025