Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. xxx (xxxx) xxx
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Highlights

Deep learning for model order reduction of multibody systems to minimal coordinates

Andrea Angeli*, Wim Desmet, Frank Naets

Comput. Methods Appl. Mech. Engrg. xxx (xxxx) xxx

- An autoencoder neural network is proposed for multibody coordinate reduction.
- The approach enables a non-intrusive extraction of the minimal coordinate model.
- The training is based on a novel loss function factoring in the multibody dynamics.
- The methodology generalizes well until the motion remains within the training range.

Please cite this article as: A. Angeli, W. Desmet and F. Naets, Deep learning for model order reduction of multibody systems to minimal coordinates, Computer Methods in Applied Mechanics and Engineering (2020) 113517, https://doi.org/10.1016/j.cma.2020.113517.

CMA: 113517

ARTICLE IN PRESS

Available online at www.sciencedirect.com

ScienceDirect

Comput. Methods Appl. Mech. Engrg. xxx (xxxx) xxx

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Deep learning for model order reduction of multibody systems to minimal coordinates

Andrea Angeli*, Wim Desmet, Frank Naets

Department of mechanical engineering, KU Leuven, Celestijnenlaan 300, 3001 Leuven, Belgium
DMMS Lab, Flanders Make, Belgium

Received 28 November 2019; received in revised form 2 September 2020; accepted 15 October 2020
Available online xxxx

Abstract

Among the proposed formulations for rigid multibody dynamics, the minimal coordinates approach permits to parametrize the system motion with the minimal amount of degrees of freedom without the need of additional constraints equations. This leads to a system of ordinary differential equations to describe the motion which enables a straightforward combination of the model with control or estimation algorithms. However, an explicit relation between the model full coordinates and a minimal number of parameters is not always available or easily obtainable, especially for spatial closed-loop mechanisms. In this work, we therefore propose to deploy deep learning to find an approximation of such motion mappings. More specifically, an autoencoder neural network architecture is exploited for the nonlinear dimensionality reduction from full to minimal coordinates. A novel neural-network training scheme is introduced, which exploits the multibody model dynamics information to optimize the decoder-function derivatives so that they represent the tangent space and the curvature of the minimal coordinates manifold. This scheme leads to an effective description of the motion manifold which can be used to express the dynamics in minimal coordinates. The approach is validated on two reference rigid body mechanisms.

Keywords: Multibody dynamics; Minimal coordinates; Deep learning; Model order reduction

1. Introduction

Over the last years, the importance of numerical simulations has increased in order to reduce the cost of design and physical prototypes. In particular, multibody simulations are typically used to describe the system-level dynamics of complex mechanisms [1,2]. In literature, different formulations have been proposed to describe multibody models. The most common ones for rigid mechanisms can be classified by number and typology of Degrees Of Freedom (DOFs): the 'Minimal Coordinates (MC)' [3] and 'Joint Coordinates (JC)' [4] make use of a minimal amount of coordinates, the 'Cartesian Coordinates (CC)' [5] involve a combination of angular and position redundant coordinates, while the 'Natural Coordinates (NC)' [1,6] use position-only redundant coordinates.

The choice of a specific formulation typically depends on the purpose of the simulation model. The MC, in particular, offer the benefit of describing the model with the minimal amount of variables, not requiring additional constraints and allowing the use of Ordinary Differential Equations (ODEs) to describe the motion. Such structure

* Corresponding author at: Department of mechanical engineering, KU Leuven, Celestijnenlaan 300, 3001 Leuven, Belgium.
E-mail addresses: andrea.angeli@kuleuven.be (A. Angeli), wim.desmet@kuleuven.be (W. Desmet), frank.naets@kuleuven.be (F. Naets).

https://doi.org/10.1016/j.cma.2020.113517
0045-7825/

may be an added benefit to couple the models with control [7] or estimation [8] algorithms, as they are typically developed for ODE models. Examples of applications exploiting an ODE multibody model are [9] for the control of rigid mechanisms or [10] for state estimation of flexible systems.

The most straightforward approach to employ a minimal amount of parameters is through the use of the JC formulation: in fact, in the case of open-loop mechanisms, a minimal coordinate description is trivially obtained [4]. However, in other cases, obtaining an analytic mapping between the system coordinates and a minimal amount of parameters may not be straightforward. For example, closed-loop mechanisms are typically modelled through redundant coordinates with additional constraints [11] and it may not be possible to establish an explicit relation between such system DOFs and some minimal parameters.

In this work, a deep learning [12,13] approach to obtain the mapping from general to minimal coordinates in case of rigid mechanisms is proposed. Over the past years, several authors have explored the use of machine learning schemes to accelerate or improve dedicated mechanical analyses [14–16]. More specifically, we propose the use of a trained AutoEncoder (AE) neural network to approximate the mapping between the minimal and the system coordinates. An important benefit of the underlying neural network architecture is the applicability for multiple (coupled) minimal coordinates, which is e.g. not case for regular interpolation based schemes to approximate the kinematic map [9]. The aim is to obtain an encoding function that maps the full coordinates to a minimal number of parameters and a specular decoding function for the inverse transformation. However, contrary to common machine learning problems, the objective of the optimization is not only merely to minimize the error between the original and the reconstructed coordinate but also to obtain a sufficiently smooth function with continuous derivatives. For this purpose, we propose to embed the multibody model information in a recurrent neural-network scheme in order to minimize the overall simulation error together with the reconstruction error. This allows to train the decoder derivatives to properly describe the motion manifold with its tangent space and curvature, enabling an effective projection of the dynamics into the minimal coordinate space. Such optimization together with the automatic differentiation of the neural-network function permits an efficient way to reduce the model order on a nonlinear manifold.

It is moreover important to highlight that the data-driven nature of the methodology in this work depends on reference simulation(s) but it is non-intrusive with respect to the formulation or software used to evaluate the reference multibody model response. This is contrary to existing methods for general minimal coordinate modelling in multibody simulation like the Gröbner bases [11,17] which require explicit access to the analytical constraint equations, or the global modal parametrization [9,18] which requires a modified quasi-static analysis with linearization.

The paper is structured as follows: Section 2 refers to the related work and Section 3 recalls the multibody formulations used in the procedure. Section 4 introduces the neural network for the proposed model order reduction while Section 5 shows two numerical examples. Finally, Section 6 reports some concluding remarks.

2. Related work

A review of possible approaches for the constraint enforcement in multibody dynamics is reported by *Laulusa & Bauchau* [19], while the problems in simulating constrained systems are considered in [20].

In general, for open-loop mechanisms, relative joint coordinates directly provide a practical approach to obtain a minimal-coordinate constraint enforcement. However, this is not the case for closed-loop mechanisms and several methodologies have been proposed to circumvent the problem and locate equivalent minimal coordinates. For instance, it is possible to apply a model modification to substitute the closed-loop with an open-loop system plus superimposed force coupling [21]. Yet, the addition of such virtual springs may cause numerical ill-conditioning. Another possibility is the numerical conversion from a general to a MC formulation, where the mapping is locally recomputed at each configuration during the simulation [22]. However, this procedure might infer a (considerable) computational burden during the simulation process. An additional option for MC transformation based on Gröbner bases was presented by Uchida & McPhee in [11], which is a promising technique to obtain efficient rigid multibody descriptions. However, this approach requires access to the analytical constraint equations, which is not always available in the frame of commercial multibody software.

A limitation for general applicability of the above approaches is their high intrusiveness with the full multibody software stack. In [9], Brüls et al. propose to address the MC issue by considering a pre-computed range of configurations of the model. An interpolation between these pre-computed points then defines the map from the

minimal coordinates and the different initial model DOFs. The methodology has been successively extended to the case of flexible mechanisms in [18,23]. However, also this approach suffers some drawbacks. First of all, a dedicated solver needs to be set up to evaluate the different reference configurations. Secondly, this approach does not extend well to general spatial mechanisms with multiple minimal DOFs, as the multi-dimensional interpolation becomes highly non-trivial. Moreover, care has to be taken to guarantee sufficient continuity in the interpolation to avoid spurious energy dissipating or generating effects.

On the other hand, the challenge of converting general multibody models to minimal coordinates models may be seen in the frame of model order reduction, where the aim is to obtain a (close-to) equivalent lower order representation of a higher order system. Such problems have been recently tackled by exploiting neural networks and deep learning. For instance, in [24], neural networks and physical information are combined to identify the best choice of reduced order model.

Lately a particular neural network architecture, the autoencoder, has showed promising results in manifold learning and non-linear dimensionality reduction [25,26] and has been successfully applied to the learning of (low-dimensional) dynamical systems [27–29]. Its combination in a Model Order Reduction (MOR) scheme has been introduced in [30] while an application to multibody dynamics has been first proposed in [31].

In this work, we propose to combine the approximating capabilities of the autoencoder neural network with the available physical structure of a multibody problem in order to reduce a general multibody model to minimal coordinates and obtain an effective ODE description for rigid multibody systems.

3. Rigid multibody dynamics in natural and minimal coordinates

The Minimal Coordinate (MC) description permits to describe a rigid multibody system with a minimal amount of degrees of freedom without the necessity of additional constraints, thus leading to Ordinary Differential Equations (ODEs). However, in some cases, given a model expressed in a redundant-coordinates formalism, the corresponding MC may not be easily identifiable or the mapping with the full coordinates may not be available. Thus, while the system is defined by redundant coordinates and Differential Algebraic Equations (DAEs), the dynamics resides in a lower-dimension manifold and could be described by the (unknown) MC.

In this work, we exploit neural networks to approximate the function that relates the original coordinates to the MC. The function derivatives necessary to describe the dynamics on the resulting nonlinear manifold can be efficiently calculated thanks to automatic differentiation [32], commonly available in all the main neural network libraries as, for example, Tensorflow [33].

The methodology requires a consistent distance measure between two different system configurations, in order to effectively perform the optimization required to train the neural network. For multibody simulations, this implies that a combination of position and angular coordinates, e.g. a single point plus Euler angles, cannot be used without defining particular metrics for the angular distances. Instead, position-only DOFs enable a consistent Euclidean distance measure. While different non-angular parameters may be employed to define the body orientations, e.g. Euler parameters or basis vectors, here it is proposed to use Natural Coordinates (NC) in order to additionally take advantage of the constant mass matrix formulation.

It is underlined that the original model may be defined in any general formulation: in fact, any simulated solution can be post-processed into natural coordinates through a kinematic projection (knowing the original formulation) or extracting the positions of four non-coplanar points per body. As for the mass matrix, it can be built from the 10 standard inertial parameters per body: mass, centre of gravity position and inertia moment. In this sense, the procedure is considered non-invasive with respect to the simulation software, as no direct access to the equations of motions is required. In the following, the NC are thus considered the starting point of the proposed approach.

In the two next subsections, the MC and NC formulations are respectively reviewed while the third one introduces the Model Order Reduction (MOR) scheme, given the mapping between the MC and the NC.

3.1. Minimal coordinates multibody formulation

Given the number n_m of rigid motion modes of a mechanism (in this work we only consider mechanisms with a constant n_m, so bifurcating systems are not considered), the dynamics can be expressed as function of the minimal coordinates $\boldsymbol{x}_m \in \mathbb{R}^{n_m}$ through ODEs:

$$\boldsymbol{f}_{\boldsymbol{m},m}(\boldsymbol{x}_m, \ddot{\boldsymbol{x}}_m) + \boldsymbol{f}_{\boldsymbol{g},m}(\boldsymbol{x}_m, \dot{\boldsymbol{x}}_m) = \boldsymbol{f}_{\boldsymbol{e},m}(\boldsymbol{x}_m) \tag{1}$$

where $\boldsymbol{f}_{\boldsymbol{m},m}, \boldsymbol{f}_{\boldsymbol{g},m} \in \mathbb{R}^{n_m}$ are, respectively, the acceleration and velocity dependent generalized inertial forces and $\boldsymbol{f}_{\boldsymbol{e},m} \in \mathbb{R}^{n_m}$ represents the vector of external generalized forces. Assuming that this model is not explicitly available, the proposed work aims to define it starting from a NC formulation once the mapping between the MC and NC is known.

3.2. Natural coordinates multibody formulation

The natural coordinates formulation [1] proposes the use of 12 redundant position-only DOFs per body (4 non-coplanar points) which permit to define a configuration independent mass matrix. For a system composed of n_n NC DOFs and n_c constraints (consisting both of joints constraints and constraints between the redundant coordinates), the dynamics can be expressed as:

$$\boldsymbol{M}_n\, \ddot{\boldsymbol{x}}_n + \left(\frac{\partial \boldsymbol{f}_c(\boldsymbol{x}_n)}{\partial \boldsymbol{x}_n}\right)^{\mathrm{T}} \boldsymbol{\lambda}_c = \boldsymbol{f}_{\boldsymbol{e},n} \tag{2}$$

$$\boldsymbol{f}_c(\boldsymbol{x}_n) = \boldsymbol{0} \tag{3}$$

where $\boldsymbol{x}_n \in \mathbb{R}^{n_n}$ is the vector of natural coordinates of the system, $\boldsymbol{M}_n \in \mathbb{R}^{n_n \times n_n}$ is the constant mass matrix, $\boldsymbol{f}_{\boldsymbol{e},n} \in \mathbb{R}^{n_n}$ is the vector of external forces, $\boldsymbol{\lambda}_c \in \mathbb{R}^{n_c}$ is the vector of Lagrange multipliers, $\boldsymbol{f}_c \in \mathbb{R}^{n_c}$ is the constraint vector function. Different force terms, e.g. reaction forces dependent on the Lagrange multipliers, are not considered in this work.

In the proposed approach, we will automatically infer the mapping between the minimal and natural coordinates from a set or reference simulations. The data of these reference simulations are collected as:

$$\mathcal{X}_n = \{\ldots, \boldsymbol{x}_n^{i,j}, \ldots\}, \mathcal{F}_{\mathcal{E},n} = \{\ldots, \boldsymbol{f}_{\boldsymbol{e},n}^{i,j}, \ldots\}, \mathcal{T} = \{\ldots, \Delta t^j, \ldots\}$$

where $\boldsymbol{x}_n^{i,j}$ and $\boldsymbol{f}_{\boldsymbol{e},n}^{i,j}$ respectively indicate the $\boldsymbol{x}_n$, $\boldsymbol{f}_{\boldsymbol{e},n}$ time-ordered sample $i = 1, \ldots, n_{t,j}$ of simulation $j = 1, \ldots, n_s$ with time step Δt^j. It is important to highlight that this time-ordering is key for the following procedure, in contrast to common data-driven model order reduction schemes where any collection of configuration snapshots is used. The sets $\mathcal{X}_n, \mathcal{F}_{\mathcal{E},n} \in \mathbb{R}^{n_n \times n_{t,j} \times n_s}$ and $\mathcal{T} \in \mathbb{R}^{n_s}$ are stored together with the matrix $\boldsymbol{M}_n$. In the following, the superscript j is dropped in order to simplify the notation, assuming we are referring to one of the training simulations.

In general, it is not necessary to perform the training simulations using a NC formulation for the approach proposed in this work. Any choice of coordinate description (available in commercial software) can be used to perform the actual simulations. The equivalent NC for setting up $\mathcal{X}_n$ can be obtained in post-processing of the training simulations by evaluating (any) four points on each body. This makes the presented approach particularly appealing from a practical perspective as it can operate non-intrusively with respect to the reference simulation scheme.

3.3. Minimal coordinate multibody dynamics through nonlinear model order reduction

In general, under the mentioned assumption of constant n_m (i.e. systems without singularities), we can define the following nonlinear mappings between the MC $\boldsymbol{x}_m$ and the NC $\boldsymbol{x}_n$:

$$\boldsymbol{h}_{\mapsto m}(\boldsymbol{x}_n) = \boldsymbol{x}_m \tag{4}$$

$$\boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m) = \boldsymbol{x}_n \tag{5}$$

The map $\boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m) = \boldsymbol{x}_n$ defines a nonlinear transformation which can be exploited to perform a model order reduction of a model expressed in NC. If these mappings are known, it is possible to the express the dynamics as a function of the MC which permits the implicit satisfaction of the constraints. Expressing the velocity in the tangent space of the manifold allows to define the system kinetic energy $\mathcal{K}$ as a function of the minimal coordinates:

$$\dot{\boldsymbol{x}}_n = \frac{\partial \boldsymbol{h}_{\mapsto n}}{\partial \boldsymbol{x}_m}\, \dot{\boldsymbol{x}}_m, \tag{6}$$

$$\mathcal{K} = \frac{1}{2}\, \dot{\boldsymbol{x}}_m^{\mathrm{T}} \left(\frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m}\right)^{\mathrm{T}} \boldsymbol{M}_n\, \frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m}\, \dot{\boldsymbol{x}}_m. \tag{7}$$

Through Hamilton's principle, this leads to the following inertial forces:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{K}}{\partial \dot{\boldsymbol{x}}_m}\right) + \frac{\partial \mathcal{K}}{\partial \boldsymbol{x}_m} = \boldsymbol{f}_{\boldsymbol{m},m} + \boldsymbol{f}_{\boldsymbol{g},m}, \tag{8}$$

$$\begin{aligned}\boldsymbol{f}_{\boldsymbol{m},m} &= \left(\frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m}\right)^{\mathrm{T}} \boldsymbol{M}_n \; \frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m} \; \ddot{\boldsymbol{x}}_m \\ &= \boldsymbol{M}_m \; \ddot{\boldsymbol{x}}_m, \end{aligned} \tag{9}$$

$$\begin{aligned}\boldsymbol{f}_{\boldsymbol{g},m} &= \left(\frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m}\right)^{\mathrm{T}} \boldsymbol{M}_n \; \frac{\partial^2 \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m \partial \boldsymbol{x}_m} \; \left[\dot{\boldsymbol{x}}_m \; \dot{\boldsymbol{x}}_m^{\mathrm{T}}\right] \\ &= \underline{\boldsymbol{G}}_m \; \left[\dot{\boldsymbol{x}}_m \; \dot{\boldsymbol{x}}_m^{\mathrm{T}}\right], \end{aligned} \tag{10}$$

where $\boldsymbol{M}_m \in \mathbb{R}^{n_m \times n_m}$ and $\underline{\boldsymbol{G}}_m \in \mathbb{R}^{n_m \times n_m \times n_m}$ are respectively defined as the MC mass matrix and the MC gyroscopic tensor.[1]

Finally, projecting the external forces:

$$\boldsymbol{f}_{\boldsymbol{e},m} = \left(\frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m}\right)^{\mathrm{T}} \boldsymbol{f}_{\boldsymbol{e},n} \tag{11}$$

allows to express the dynamics of the MC model as:

$$\boldsymbol{M}_m(\boldsymbol{x}_m) \; \ddot{\boldsymbol{x}}_m + \underline{\boldsymbol{G}}_m(\boldsymbol{x}_m) \; \left[\dot{\boldsymbol{x}}_m \; \dot{\boldsymbol{x}}_m^{\mathrm{T}}\right] = \boldsymbol{f}_{\boldsymbol{e},m}(\boldsymbol{x}_m) \tag{12}$$

As this is a set of ODEs, a relatively versatile selection of time-integration schemes is possible, in contrast to general constraint-based multibody formulations where a stiff solver is required to handle the DAEs. In particular, in this work we propose to use finite differences for the time integration, as presented in Algorithm 1, but different integration schemes are possible. At each time step, the transformation $\boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m)$ can project the MC $\boldsymbol{x}_m$ back into the NC $\hat{\boldsymbol{x}}_n$ for post-processing purposes.

Given the general MC dynamic simulation framework introduced above, the question remains how the non-linear mappings $\boldsymbol{h}_{\mapsto m}$ and $\boldsymbol{h}_{\mapsto n}$ can be constructed. In the next section, we propose to approximate these mappings using deep neural networks, by training them on a set of reference simulations for which the response is expressed in NC.

4. Deep learning of minimal coordinate manifold mapping

In this section, the function-approximation capabilities of neural networks are exploited to find the mappings between the natural coordinates and the minimal coordinates to enable an effective low order multibody dynamics description.

In Section 4.1, basic neural network architectures are briefly reviewed, and in Section 4.2 the application for model order reduction to minimal coordinates for multibody systems is presented.

4.1. Neural network architecture overview

For the purpose of generating the mapping between minimal and natural coordinates, we briefly review three key concepts: deep neural networks, recurrent neural networks, and autoencoders.

[1] The notation of $\boldsymbol{f}_{\boldsymbol{g},m} \in \mathbb{R}^{n_m}$ and $\underline{\boldsymbol{G}}_m \in \mathbb{R}^{n_m \times n_m \times n_m}$, where the double partial derivative of a vector is assumed to return a 3-dimensional tensor and the matrix multiplication with a tensor is intended with respect to its two inner dimensions as indicated by the Einstein summation convention over the Greek-subscripts dimensions, is:

$$\begin{aligned}\boldsymbol{f}_{\boldsymbol{g},m} &= \sum_{k=1}^{n_m} \left(\frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m}\right)^{\mathrm{T}} \boldsymbol{M}_n \; \frac{\partial^2 \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m \partial \boldsymbol{x}_{m,k}} \; \dot{\boldsymbol{x}}_m \; \dot{\boldsymbol{x}}_{m,k} \;=\; \underline{\boldsymbol{G}}_{m_{\alpha,\beta,\gamma}} \; \dot{\boldsymbol{x}}_{m_\beta} \; \dot{\boldsymbol{x}}_{m_\gamma} \\ &= \left(\frac{\partial \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m}\right)^{\mathrm{T}} \boldsymbol{M}_n \; \frac{\partial^2 \boldsymbol{h}_{\mapsto m}}{\partial \boldsymbol{x}_m \partial \boldsymbol{x}_m} \; \left[\dot{\boldsymbol{x}}_m \; \dot{\boldsymbol{x}}_m^{\mathrm{T}}\right] \;=\; \underline{\boldsymbol{G}}_m \; \left[\dot{\boldsymbol{x}}_m \; \dot{\boldsymbol{x}}_m^{\mathrm{T}}\right]. \end{aligned}$$

Algorithm 1 time-domain integration

1: Recall:

$$\boldsymbol{x}_n^{i=1}, \qquad \boldsymbol{x}_n^{i=2}, \qquad \Delta t, \qquad \boldsymbol{M}_n$$

2: Define:

$$\boldsymbol{f}_{e,n}$$

3: Assign:

$$\boldsymbol{x}_m^{i=1} = \boldsymbol{h}_{\mapsto m}(\boldsymbol{x}_n^{i=1}), \qquad \boldsymbol{x}_m^{i=2} = \boldsymbol{h}_{\mapsto m}(\boldsymbol{x}_n^{i=2})$$
$$\hat{\boldsymbol{x}}_n^{i=1} = \boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m^{i=1}), \qquad \hat{\boldsymbol{x}}_n^{i=2} = \boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m^{i=2})$$

4: **for** $i = 2 : n_t - 1$ **do**

$$\dot{\boldsymbol{x}}_m^i \approx \dot{\boldsymbol{x}}_m^{i-\frac{1}{2}} = \frac{\boldsymbol{x}_m^i - \boldsymbol{x}_m^{i-1}}{\Delta t}$$
$$\boldsymbol{M}_m^i = \left(\frac{\partial \boldsymbol{h}_{\mapsto n}}{\partial \boldsymbol{x}_m^i}\right)^{\mathrm{T}} \boldsymbol{M}_n \frac{\partial \boldsymbol{h}_{\mapsto n}}{\partial \boldsymbol{x}_m^i}$$
$$\underline{\boldsymbol{G}}_m^i = \left(\frac{\partial \boldsymbol{h}_{\mapsto n}}{\partial \boldsymbol{x}_m^i}\right)^{\mathrm{T}} \boldsymbol{M}_n \frac{\partial^2 \boldsymbol{h}_{\mapsto n}}{\partial \boldsymbol{x}_m^i \partial \boldsymbol{x}_m^i}$$
$$\boldsymbol{f}_{e,m}^i = \left(\frac{\partial \boldsymbol{h}_{\mapsto n}}{\partial \boldsymbol{x}_m^i}\right)^{\mathrm{T}} \boldsymbol{f}_{e,n}^i$$
$$\boldsymbol{x}_m^{i+1} = \boldsymbol{x}_m^i + \Delta t\ \dot{\boldsymbol{x}}_m^{i-\frac{1}{2}} + \Delta t^2 \left(\boldsymbol{M}_m^i\right)^{-1} \left(\boldsymbol{f}_{e,m}^i - \underline{\boldsymbol{G}}_m^i \left[\dot{\boldsymbol{x}}_m^i\ \dot{\boldsymbol{x}}_m^{i\,\mathrm{T}}\right]\right)$$
$$\hat{\boldsymbol{x}}_n^{i+1} = \boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m^{i+1})$$

5: **end for**

In general, a (feedforward) neural network layer, conceptually shown in Fig. 1, can be described as:

$$\boldsymbol{x}_o = \boldsymbol{h}_{\boldsymbol{a}}(\boldsymbol{W}\ \boldsymbol{x}_i + \boldsymbol{b}) \tag{13}$$

where $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$ is the input vector to the layer, $\boldsymbol{h}_{\boldsymbol{a}} \in \mathbb{R}^{n_o}$ is the element-wise activation function evaluated on a vector, $\mathcal{P} = \{\boldsymbol{W} \in \mathbb{R}^{n_o \times n_i}, \boldsymbol{b} \in \mathbb{R}^{n_o}\}$ are the parameters to be identified in order to minimize a certain loss function, and $\boldsymbol{x}_o \in \mathbb{R}^{n_o}$ represents the output of the layer. The so-called activation function introduces the nonlinear behaviour of the network and a good choice of this function will depend on the application [13].

Typical activation functions are the Rectified Linear Unit (ReLU) and the sigmoid, respectively:

$$\boldsymbol{h}_{ReLU}(x_l) = \max(0, x_l) \tag{14}$$
$$\boldsymbol{h}_{sig}(x_l) = \frac{1}{1 + e^{-x_l}} \tag{15}$$

where $x_l \in \mathbb{R}$ is one of the n_o elements of the layer, which after the nonlinear transformation will form the output vector $\boldsymbol{x}_o$ as in Eq. (13).

After its introduction, ReLU has replaced the sigmoid as the standard choice for feedforward networks thanks to its computational efficiency and fast convergence. However, in this work we will employ sigmoid functions. In fact, as we will need to perform derivation with respect to the states of the neural network functions in order to generate the contributions for Algorithm 1, it is key to have continuous derivatives with respect to these states. This is not possible with the ReLU function, as for most applications only derivatives with respect to the parameters $\mathcal{P}$ are required, but not with respect to the states. On the other hand, the sigmoid guarantees smooth (second-order) derivatives, according to:

$$\frac{\partial \boldsymbol{h}_{sig}(x_l)}{\partial x_l} = \boldsymbol{h}_{sig}(x_l)\left(1 - \boldsymbol{h}_{sig}(x_l)\right) \tag{16}$$

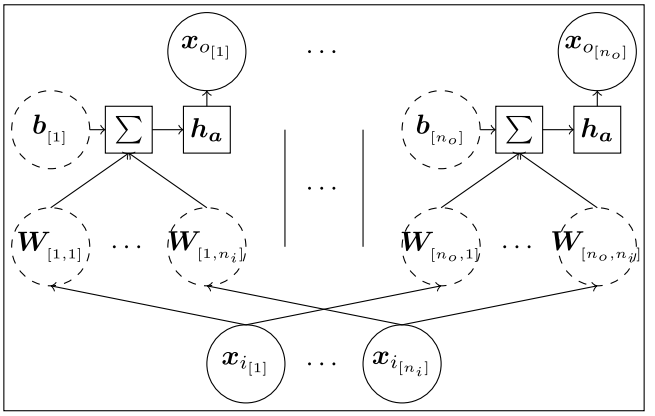

Fig. 1. A neural network layer. On the left, the weighted (by a row of the weight matrix $\boldsymbol{W}$) elements of the input $\boldsymbol{x}_i$ are summed together with the corresponding element of the bias $\boldsymbol{b}$ and "activate" the function $\boldsymbol{h_a}$ to obtain an element of the output $\boldsymbol{x}_o$; moving toward the right, the procedure is repeated for each row of $\boldsymbol{W}$, obtaining the full output vector. The dashed circles represent the "trainable" parameters.

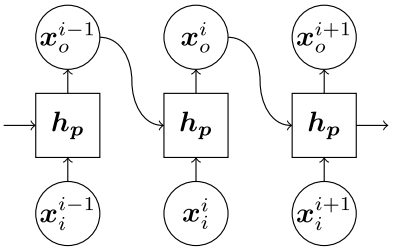

Fig. 2. A recurrent neural network, where each output $\boldsymbol{x}_o^i$ is influenced by the input $\boldsymbol{x}_i^i$ and the previous-step output $\boldsymbol{x}_o^{i-1}$.

In literature, different twice-differentiable functions are available [13] and may be used, specially for particular cases as, for instance, the sine function to approximate periodic coordinates [34].

In order to increase the neural network approximation capabilities, it is possible to concatenate several layers obtaining a "deep neural network" with the output of a layer used as input of the next one, for the n_l layers:

$$\boldsymbol{x}_{o,k} = \boldsymbol{h_a}_{,k}(\boldsymbol{W}_k\ \boldsymbol{x}_{o,k-1} + \boldsymbol{b}_k),\ \ \boldsymbol{x}_{o,k} \in \mathbb{R}^{n_k},\ \ k = 1 \ldots, n_l \tag{17}$$

where $\boldsymbol{x}_{o,0} = \boldsymbol{x}_i \in \mathbb{R}^{n_i}$ and $\boldsymbol{x}_{o,n_l} = \boldsymbol{x}_o \in \mathbb{R}^{n_o}$ are, respectively, the initial input and the final output of the neural network. Their relation can be summarized by the entire neural-network function $\boldsymbol{h_l}$ with parameters $\mathcal{P}_l = \{\boldsymbol{W}_k, \boldsymbol{b}_k \mid k = 1, \ldots, n_l\}$:

$$\boldsymbol{x}_o = \boldsymbol{h_l}(\boldsymbol{x}_i, \mathcal{P}_l) \tag{18}$$

A specific extension of this framework aimed to process sequence data is represented by recurrent neural networks. The relation between two consecutive states is expressed through additional trainable parameters. For example, in a single-layer network, at step i the current output depends also on the previous one as:

$$\boldsymbol{x}_o^i = \boldsymbol{h_a}(\boldsymbol{W}_1\ \boldsymbol{x}_i^i + \boldsymbol{b}_1) + \boldsymbol{h_a}(\boldsymbol{W}_2\ \boldsymbol{x}_o^{i-1} + \boldsymbol{b}_2) \tag{19}$$

As reported in the schematic in Fig. 2, the structure can be extended to deep neural networks and, in general, can be represented by a recurrent neural network function $\boldsymbol{h_p}$ with parameters $\mathcal{P}_p$:

$$\boldsymbol{x}_o^i = \boldsymbol{h_p}(\boldsymbol{x}_i^i, \boldsymbol{x}_o^{i-1}, \mathcal{P}_p) \tag{20}$$

Finally, autoencoders (AE) are particular neural networks with a symmetrical structure: the encoder, represented by the function $\boldsymbol{h_e}$ and its parameters $\mathcal{P}_e$, transforms the high order input $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$ into the low order hidden state $\boldsymbol{x}_r \in \mathbb{R}^{n_r}$. Then, the decoder with the specular function $\boldsymbol{h_d}$ and parameters $\mathcal{P}_d$ aims to reconstruct the output

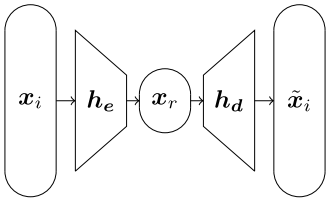

Fig. 3. The schematic of the undercomplete autoencoder, where the input $\boldsymbol{x}_i$ is reduced to $\boldsymbol{x}_r$ by the encoder $\boldsymbol{h_e}$ and backprojected to the output $\tilde{\boldsymbol{x}}_i$ by the decoder $\boldsymbol{h_d}$. In order to have $\tilde{\boldsymbol{x}}_i \approx \boldsymbol{x}_i$, $\boldsymbol{x}_r$ has to be a meaningful reduced parametrization.

$\boldsymbol{x}_o = \tilde{\boldsymbol{x}}_i \in \mathbb{R}^{n_i}$ as close as possible to the input $\boldsymbol{x}_i$:

$$\boldsymbol{x}_r = \boldsymbol{h_e}(\boldsymbol{x}_i, \mathcal{P}_e) \tag{21}$$

$$\boldsymbol{x}_i \approx \tilde{\boldsymbol{x}}_i = \boldsymbol{h_d}\left(\boldsymbol{x}_r(\boldsymbol{x}_i, \mathcal{P}_e), \mathcal{P}_d\right) \tag{22}$$

where $\boldsymbol{h_e}$ and $\boldsymbol{h_d}$ can be single-layer (perceptron) or deep neural networks. Different typologies of AE exist: if $n_i = n_r$ the networks may learn the identity function $\boldsymbol{h_e} = \boldsymbol{h_d} = \boldsymbol{I}$ and trivially copy the input, so different regularizations are added in order to obtain an useful parametrization $\boldsymbol{x}_r$. In particular, the undercomplete AE, showed in Fig. 3, has $n_r < n_i$ as regularization and aims to learn a reduced parametrization $\boldsymbol{x}_r$. The AE can be considered as a non-linear extension [25,26] of Principal Component Analysis (PCA) [35], which makes it naturally suitable for order reduction purposes.

Moreover, autoencoders can be extended to "recurrent autoencoders", where the ith hidden state $\boldsymbol{x}_r^i$ is influenced by the previous-step value $\boldsymbol{x}_r^{i-1}$. In the general case, such dependence on the previous step is expressed through additional trainable parameters as in Eq. (19). However, in this work it is proposed to couple a recurrent undercomplete autoencoder with the knowledge about the underlying multibody model. This allows to optimize also the derivatives of the decoder function as required by the model order reduction procedure to properly describe the dynamics in the reduced space. The procedure is showed in the next section.

4.2. Neural networks for model order reduction from natural to minimal coordinates in multibody simulation

Given a system defined or post-processed into position-only DOFs such as NC, three scenarios are possible:

1. The number of MC n_m is not known. This atypical situation for multibody systems falls into the category of manifold learning which is beyond the scope of this work. In literature, different methods have been proposed to estimate the dimension of a manifold [36,37] which can be used to find the number of MC and permit to relate to the second scenario, but we do not discuss this further in the current work.
2. The number of MC n_m is known but it is not possible to a-priori define them. Thus, the MC must be defined together with the mapping between such unknown MC and the NC. This case may be classified as unsupervised learning (neither MC or the mapping is known) and requires a full AE approach. This case is addressed in Section 4.2.1.
3. The MC are known or it is possible to choose some known parameters as MC. However, the analytic mapping between these MC and the NC is unknown or impractical to obtain. This case can be seen as a subcategory of the first scenario. In fact, knowing the MC and the direct mapping, the (supervised learning) problem consists in determining its inverse and does not require a full AE approach. This case is addressed in Section 4.2.2.

It is important to highlight that all approaches presented rely on the definition of a good loss function which allows to train the neural network parameters. It is for this reason that we proposed to employ a set of natural coordinates for the training data in Section 3. Mixed coordinate types, where both positional and rotational DOFs are combined do not allow to obtain a consistent Euclidean distance for the loss function. By contrast, the Euclidean distance for the NC is well defined.

The scheme for the model order reduction procedure is summarized in Algorithm 2.

Algorithm 2 model order reduction.

1: Run reference simulation(s) and collect NC data $\mathcal{X}_n, \mathcal{F}_{\mathcal{E},n}, \mathcal{T}, \boldsymbol{M}_n$ [cf. Section 3.2]
2: **if** n_m is unknown **then**
3: Estimate n_m [36,37]
4: **else if** $\boldsymbol{x}_m$ is known **then**
5: $\boldsymbol{h_e}(\boldsymbol{x}_n) = \boldsymbol{h}_{\mapsto m}(\boldsymbol{x}_n)$ [cf. Section 4.2.2]
6: **end if**
7: Train recurrent AE and find $\boldsymbol{x}_m, \boldsymbol{h_e}, \boldsymbol{h_d}$ [cf. Section 4.2.1]
8: Run reduced-order simulation [cf. Algorithm 1]

4.2.1. Recurrent autoencoder for minimal coordinates mapping

If the MC number n_m is available or has been estimated, an undercomplete AE neural network is used to approximate the mapping between the (unknown) MC and the NC:

$$\boldsymbol{x}_m = \boldsymbol{h}_{\mapsto m}(\boldsymbol{x}_n) \approx \tilde{\boldsymbol{x}}_m = \boldsymbol{h_e}(\boldsymbol{x}_n, \mathcal{P}_e) \tag{23}$$

$$\boldsymbol{x}_n = \boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m) \approx \tilde{\boldsymbol{x}}_n = \boldsymbol{h_d}\left(\tilde{\boldsymbol{x}}_m(\mathcal{P}_e), \mathcal{P}_d\right) \tag{24}$$

with the AE hidden dimension as $n_r = n_m$. In a straightforward approach the parameters $\{\mathcal{P}_e, \mathcal{P}_d\}$ can be optimized to minimize the Euclidean distance between the training data and the reconstruction, averaged over the n_t training samples as loss function $\mathcal{L}_{rec}$:

$$\mathcal{L}_{rec}(\boldsymbol{x}_n, \mathcal{P}_e, \mathcal{P}_d) = \frac{1}{n_t}\sum_{i=1}^{n_t}\left(\tilde{\boldsymbol{x}}_n^i(\boldsymbol{x}_n^i, \mathcal{P}_e, \mathcal{P}_d) - \boldsymbol{x}_n^i\right)^2 \tag{25}$$

However, this straightforward loss function $\mathcal{L}_{rec}$ definition suffers a critical drawback as it only accounts for the model kinematics while it neglects the time evolution or information about the function derivatives. These derivatives are however essential in Algorithm 1 to describe the manifold of the MC dynamics. Even slight overfitting on this purely kinematic data would lead to inaccurate or even unstable simulations.

Therefore, we propose to embed the known underlying multibody model of Section 3.3 in a recurrent AE architecture, as shown in Fig. 4, in order to minimize the squared simulation error and train also the decoder-function derivatives. This leads to a new loss function $\mathcal{L}_{sim}$:

$$\mathcal{L}_{sim}(\boldsymbol{x}_n, \mathcal{P}_d) = \frac{1}{n_t - 2}\sum_{i=2}^{n_t-1}\left(\hat{\boldsymbol{x}}_n^{i+1}(\boldsymbol{x}_n^{i-1}, \boldsymbol{x}_n^i, \mathcal{P}_d) - \boldsymbol{x}_n^{i+1}\right)^2 \tag{26}$$

where $\hat{\boldsymbol{x}}_n^{i+1}$ represents the predicted simulation value at the next time step, as in Algorithm 1:

$$\hat{\boldsymbol{x}}_n^{i+1} = \boldsymbol{h_d}\left(\tilde{\boldsymbol{x}}_m^i + \Delta t\ \dot{\tilde{\boldsymbol{x}}}_m^i + \Delta t^2\left(\tilde{\boldsymbol{M}}_m^i\right)^{-1}\left(\tilde{\boldsymbol{f}}_{\boldsymbol{e},m}^i - \underline{\tilde{\boldsymbol{G}}}_m^i\left[\dot{\tilde{\boldsymbol{x}}}_m^i\ \dot{\tilde{\boldsymbol{x}}}_m^{i\mathrm{T}}\right]\right)\right) \tag{27}$$

with:

$$\tilde{\boldsymbol{x}}_m^{i-1}(\boldsymbol{x}_n^{i-1}) = \boldsymbol{h_e}(\boldsymbol{x}_n^{i-1}) \tag{28}$$

$$\tilde{\boldsymbol{x}}_m^{i}(\boldsymbol{x}_n^{i}) = \boldsymbol{h_e}(\boldsymbol{x}_n^{i}) \tag{29}$$

$$\dot{\tilde{\boldsymbol{x}}}_m^i = \frac{\tilde{\boldsymbol{x}}_m^i - \tilde{\boldsymbol{x}}_m^{i-1}}{\Delta t} \tag{30}$$

$$\tilde{\boldsymbol{M}}_m^i(\mathcal{P}_d) = \left(\frac{\partial \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m^i}\right)^{\mathrm{T}} \boldsymbol{M}_n \frac{\partial \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m^i} \tag{31}$$

$$\underline{\tilde{\boldsymbol{G}}}_m^i(\mathcal{P}_d) = \left(\frac{\partial \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m^i}\right)^{\mathrm{T}} \boldsymbol{M}_n \frac{\partial^2 \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m^i \partial \tilde{\boldsymbol{x}}_m^i} \tag{32}$$

$$\tilde{\boldsymbol{f}}_{\boldsymbol{e},m}^i(\mathcal{P}_d) = \left(\frac{\partial \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m^i}\right)^{\mathrm{T}} \boldsymbol{f}_{\boldsymbol{e},n}^i \tag{33}$$

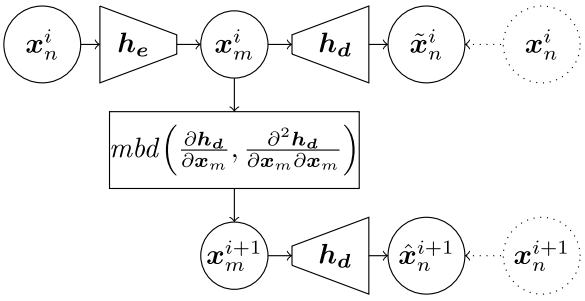

Fig. 4. The proposed recurrent autoencoder, where the encoder $\boldsymbol{h_e}$ and the decoder $\boldsymbol{h_d}$ respectively reduce and backproject the original coordinates, while the *mbd* block includes the multibody model reduced through (the derivatives of) $\boldsymbol{h_d}$ and permits to advance the reduced simulation in time. The trapezoids and the rectangle are the functions with the parameters to be optimized, the circles are the states and the dotted circles are the reference for the optimization.

This single-step prediction can be easily coupled with standard machine learning optimizations such as stochastic gradient descent [13], which iterates the optimization on a subset, called mini-batch, of the data in order to speed up the optimization and avoid to compute the gradient with respect to all the samples. In fact, the training algorithm can be fed with mini-batches of (groups of) three consecutive reference points $\boldsymbol{x}_n^{i-1}, \boldsymbol{x}_n^i, \boldsymbol{x}_n^{i+1}$; while common recurrent neural networks may require to be fed with all the n_t samples of the time evolution, leading to slow convergence, possible vanishing/exploding gradients [13] or the necessity to reshape the network architecture to perform the time simulation [38].

The derivatives of the decoder function with respect to the states are necessary for these operations. In this work, we exploit the automatic differentiation capabilities of common machine learning toolboxes in order to evaluate these derivatives efficiently. In this particular loss function, a twice differentiable activation function is necessary, like the sigmoid function of Eq. (15).

For the actual training of the AE we therefore propose to use a loss function that combines both the reconstruction loss from Eq. (25) and dynamic simulation loss from Eq. (26):

$$\underset{\mathcal{P}_e, \mathcal{P}_d}{\operatorname{argmin}} \left(\mathcal{L}_{rec}(\boldsymbol{x}_n, \mathcal{P}_e, \mathcal{P}_d) + \mathcal{L}_{sim}(\boldsymbol{x}_n, \mathcal{P}_d)\right) \tag{34}$$

As it will be demonstrated in Section 5, this choice of loss allows to effectively circumvent issues related to the poor properties of the derivatives of the decoder functions and lead to a reliable mapping from MC to NC to perform simulations.

Alternatively, a loss function based on velocity and acceleration information is discussed in Appendix.

4.2.2. Recurrent neural network for minimal coordinates mapping

The case in which the MC are known or it is possible to select some specific DOFs (e.g. a subset of the original NC or a known transformation) results in an available $\boldsymbol{h}_{\mapsto m}$ mapping. We can therefore consider the encoder function exactly known as defined by the user:

$$\boldsymbol{x}_m = \boldsymbol{h}_{\mapsto m}(\boldsymbol{x}_n) = \boldsymbol{h_e}(\boldsymbol{x}_n). \tag{35}$$

In many cases this will for example be a simple selection of some of the NC as MC, such that the encoder becomes a simple linear selection matrix:

$$\boldsymbol{x}_m = \boldsymbol{V}_{\mapsto m}\, \boldsymbol{x}_n, \tag{36}$$

where $\boldsymbol{V}_{\mapsto m} \in \mathbb{R}^{n_m \times n_n}$ is a sparse selection matrix.

The problem described in the previous section is then simplified to determine only the decoder neural network:

$$\boldsymbol{x}_n = \boldsymbol{h}_{\mapsto n}(\boldsymbol{x}_m) \approx \tilde{\boldsymbol{x}}_n = \boldsymbol{h_d}(\boldsymbol{x}_m, \mathcal{P}_d) \tag{37}$$

Thus, the loss function to be minimized is:

$$\underset{\mathcal{P}_d}{\operatorname{argmin}} \left(\mathcal{L}^*_{rec}(\boldsymbol{x}_n, \mathcal{P}_d) + \mathcal{L}^*_{sim}(\boldsymbol{x}_n, \mathcal{P}_d)\right) \tag{38}$$

where:

$$\mathcal{L}^*_{rec}(\boldsymbol{x}_n, \mathcal{P}_d) = \frac{1}{n_t}\sum_{i=1}^{n_t}\left(\tilde{\boldsymbol{x}}^i_n(\boldsymbol{x}^i_n, \mathcal{P}_d) - \boldsymbol{x}^i_n\right)^2 \tag{39}$$

$$\mathcal{L}^*_{sim}(\boldsymbol{x}_n, \mathcal{P}_d) = \frac{1}{n_t - 2}\sum_{i=2}^{n_t-1}\left(\hat{\boldsymbol{x}}^{i+1}_n(\boldsymbol{x}^{i-1}_n, \boldsymbol{x}^i_n, \mathcal{P}_d) - \boldsymbol{x}^{i+1}_n\right)^2 \tag{40}$$

with:

$$\boldsymbol{x}^{i-1}_m(\boldsymbol{x}^{i-1}_n) = \boldsymbol{h}_{\mapsto m}(\boldsymbol{x}^{i-1}_n) \tag{41}$$

$$\boldsymbol{x}^{i}_m(\boldsymbol{x}^{i}_n) = \boldsymbol{h}_{\mapsto m}(\boldsymbol{x}^{i}_n) \tag{42}$$

$$\dot{\tilde{\boldsymbol{x}}}^i_m = \frac{\boldsymbol{x}^i_m - \boldsymbol{x}^{i-1}_m}{\Delta t} \tag{43}$$

$$\tilde{\boldsymbol{M}}^i_m(\mathcal{P}_d) = \left(\frac{\partial \boldsymbol{h}_d}{\partial \tilde{\boldsymbol{x}}^i_m}\right)^{\mathrm{T}} \boldsymbol{M}_n \frac{\partial \boldsymbol{h}_d}{\partial \tilde{\boldsymbol{x}}^i_m} \tag{44}$$

$$\underline{\tilde{\boldsymbol{G}}}^i_m(\mathcal{P}_d) = \left(\frac{\partial \boldsymbol{h}_d}{\partial \tilde{\boldsymbol{x}}^i_m}\right)^{\mathrm{T}} \boldsymbol{M}_n \frac{\partial^2 \boldsymbol{h}_d}{\partial \tilde{\boldsymbol{x}}^i_m \partial \tilde{\boldsymbol{x}}^i_m} \tag{45}$$

$$\tilde{\boldsymbol{f}}^i_{e,m}(\mathcal{P}_d) = \left(\frac{\partial \boldsymbol{h}_d}{\partial \tilde{\boldsymbol{x}}^i_m}\right)^{\mathrm{T}} \boldsymbol{f}^i_{e,n} \tag{46}$$

5. Validation

In this section we demonstrate the proposed approach for converting rigid multibody models into reduced order minimal coordinate models on two numerical examples. In the first case in Section 5.1, it is assumed that only the number of MC is known and the procedure presented in Section 4.2.1 is applied. In the second example in Section 5.2, the choice of MC is known and the procedure explained in Section 4.2.2 is followed. As previously indicated, only mechanisms without singularities are analysed in this work.

The reference multibody simulations are performed in a general purpose multibody simulation toolbox in Matlab [39] using a natural coordinates formulation. For each example, three simulations consisting of 10 000 samples ($\Delta t = 0.001$ s, $t = 10$ s) are performed with different input forces: the first is used for the neural network training (95% of data for training, 5% for cross validation and early stopping [13]).

The remaining two datasets are used for validation. In fact, we aim to assess the generalization capabilities of the trained network with respect to novel inputs, as in case of control or estimation schemes the exact run-time excitation may be unknown. In particular, the first application shows the case of inputs with different amplitudes while the second application shows shape-varying inputs.

Furthermore, the fact that, thanks to the physical information embedding, the training requires the use of a single simulation (and in general as little data as possible) is considered an additional benefit.

The presented optimization and MC modelling approach are implemented in Tensorflow [33]. The neural network is built with sigmoid activation functions as in Eq. (15) to guarantee differentiability and its parameters are optimized using the Adam algorithm [40], a variant of stochastic gradient descent. 10 000 epochs (i.e. training iterations) are carried out, requiring less than 40 minutes for the optimization (per application example), with all the computations performed on a consumer laptop.

As indicated, the main advantage of the procedure is the obtained ODE structure, while the computation time is not the focus of this work. However, in the future, the procedure may possibly target real-time applications given that, from purely qualitative assessments, the simulation time is reduced from several minutes for the original models to few seconds for the reduced models.

5.1. Macpherson suspension mechanism

The first application case is a MacPherson suspension, shown in Fig. 5. The kinematic model is composed of six bodies: a lower control arm is linked to the chassis by two spherical joints and to the steering knuckle by a

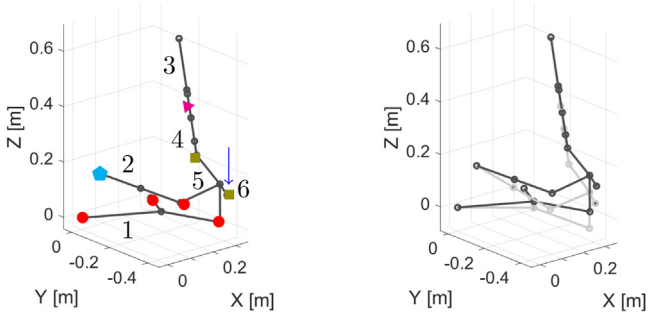

Fig. 5. The MacPherson suspension model. On the left, the system with the enumerated bodies (listed in Table 1), the external input as (blue) arrow and the joints as polygons — the (red) circles represent the spherical joints, the (magenta) triangle represents the prismatic joint, the (cyan) pentagon represents a Cardan joint and the (olive) squares represent rigid joints. On the right, the model in two different configurations. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
The enumerated list of bodies for the MacPherson suspension model with the respective mass m and inertia moments J.

Body	m [kg]	J_{xx} [kg m^2]	J_{yy} [kg m^2]	J_{zz} [kg m^2]
Control arm (1)	3.573	0.069	0.037	0.102
Tie rod (2)	1.10	0.011	0.00003	0.011
Upper damper (3)	0.75	0.009	0.009	0.00005
Lower damper (4)	2.85	0.032	0.032	0.002
Knuckle (5)	5.65	0.041	0.051	0.035
Spindle (6)	0.2	0.0006	0.001	0.0006

spherical joint; the knuckle is linked to the tie-rod by a spherical joint and to the strut by a prismatic joint; finally the wheel spindle is rigidly attached to the knuckle. A list of the bodies with their inertial properties is reported in Table 1. The system is excited on the wheel spindle through three time-varying vertical forces with different amplitudes, as shown in Fig. 6.

The model is initially available in NC and, given the suspension application the number of MC is known as $n_m = 1$. However, we do not predefine a mapping from the NC to the MC. Thus, a parametrization for the unknown MC and the related mappings are identified according to the procedure proposed in Section 4.2.1.

The simulation with the medium amplitude excitation "sim1" is used to train the autoencoder. The comparison between this training simulation and the proposed methodology for the positions of the body centres of gravity is showed in Fig. 8. A close-up of the vertical motion of the wheel-attachment together with its relative error is showed in Fig. 7. These figures both demonstrate good accuracy for the training case, as it can be expected. Fig. 7 shows that over time the error with respect to the training simulation increase. This drift is to be expected as small deviation on the MC inertia will amount to an increasing model difference.

The MC model is then used for the two non-trained excitation cases: the case with the force of lower amplitude "sim2" is shown in Fig. 9, while the case with the force of higher amplitude "sim3" is shown in Fig. 10. It can be noticed that the proposed neural-network based MC description generalizes well as long as the model response remains inside the trained motion range, as is the case in Fig. 9. However, if the resulting motion falls outside of the displacement range of the training dataset, e.g. as a result of higher excitation in Fig. 10, the proposed scheme becomes unreliable as the neural network does not allow to extrapolate accurately.

It is thus important to include in the training dataset a proper representation of the foreseen motion range, in order to properly optimize the neural network parameters.

5.2. Two-bar actuator mechanism

The second example is a two-bar actuator mechanism which can move in the xy plane. It consists of two bars linked together by a revolute joint and each bar has at the base a sliding joint with the y-axis as the allowed direction of motion. The model is shown in Fig. 11 with the list of the bodies and their inertias reported in Table 2.

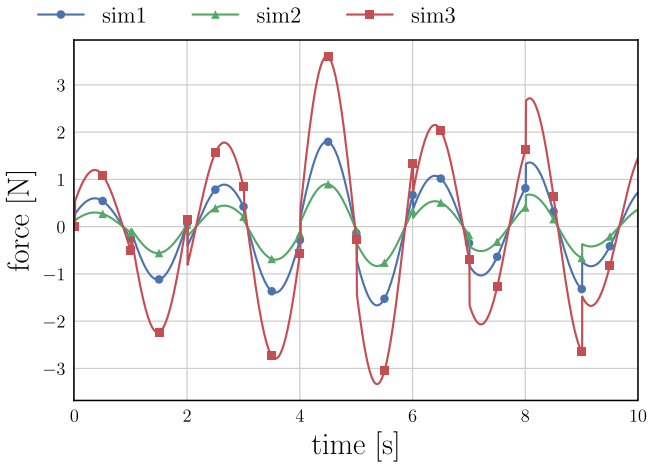

Fig. 6. The external forces applied to the wheel-attachment along the z axis, in the three different simulations. Only the simulation data resulting from the external forces $sim1$ are used for the neural network training.

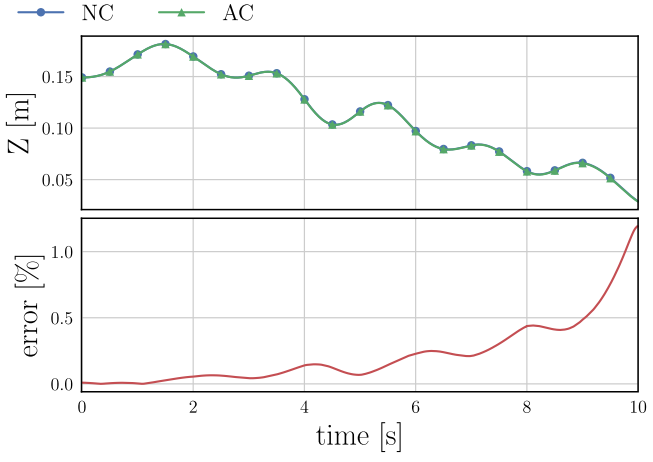

Fig. 7. Motion along the z axis of the wheel-attachment in 'Natural Coordinates (NC)' $\boldsymbol{x}_n$ and 'Autoencoder Coordinates (AC)' $\hat{\boldsymbol{x}}_n$. Above, comparison of the simulation; below, the corresponding relative error.

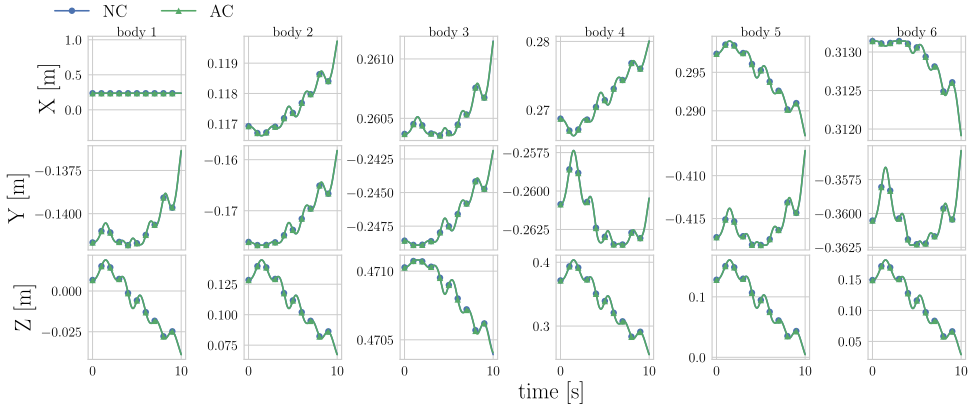

Fig. 8. Comparison of the simulation in 'Natural Coordinates (NC)' $\boldsymbol{x}_n$ and 'Autoencoder Coordinates (AC)' $\hat{\boldsymbol{x}}_n$; each column of plots represents the centre-of-gravity coordinates of one of the bodies.

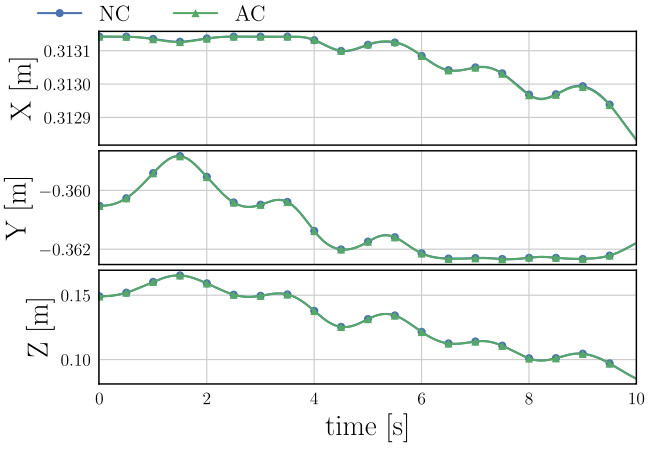

Fig. 9. Comparison of the simulation for the position of the wheel-attachment centre of gravity in 'Natural Coordinates (NC)' $\boldsymbol{x}_n$ and 'Autoencoder Coordinates (AC)' $\hat{\boldsymbol{x}}_n$, for a non-trained force. As it can be noticed, the method generalizes well.

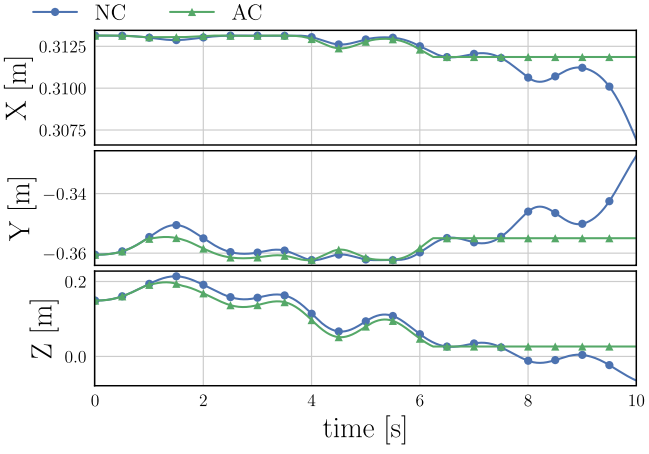

Fig. 10. Comparison of the simulation for the position of the wheel-attachment centre of gravity in 'Natural Coordinates (NC)' $\boldsymbol{x}_n$ and 'Autoencoder Coordinates (AC)' $\hat{\boldsymbol{x}}_n$ with a force leading to out-of-trained-motion range. The non-negligible out-of-range motion causes the AE simulation to fail due to the incapacity of extrapolating such values.

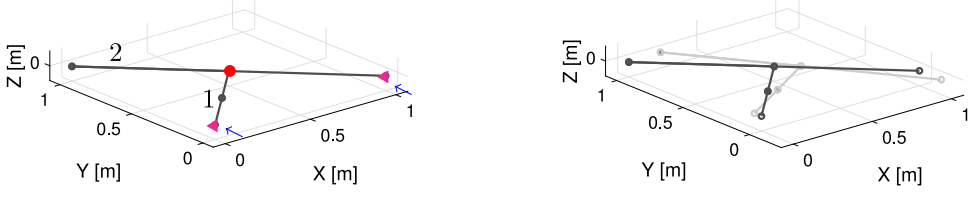

Fig. 11. The multibody model of the two-bars mechanism. On the left, the system with the enumerated bodies (listed in Table 2), the external inputs as (blue) arrows and the joints as polygons — the (red) circle represents the revolute joint and the (magenta) triangles represent the sliding joints. On the right, the model in two different configurations. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In this case, the y translation of the two-bar base points are used to control the system and are used as minimal coordinates. However, it is assumed that the analytic mapping to the NC coordinates is not explicitly available. Thus, it is obtained with the procedure described in Section 4.2.2. Three different types of excitation forces are applied, as shown in Fig. 12: a time-varying force "sim1", a lower sinusoidal force "sim2" and the first force scaled to

Table 2
The enumerated list of bodies for the two-bars mechanism with the respective mass m and inertia moments J.

Body	m [kg]	J_{xx} [kg m^2]	J_{yy} [kg m^2]	J_{zz} [kg m^2]
Short bar (1)	2.9	0.1208	0.1208	0
Long bar (2)	1.3	0.2167	0.2167	0

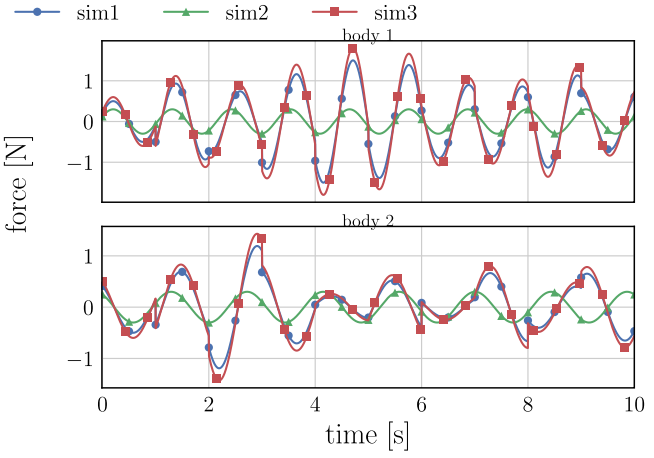

Fig. 12. The external forces applied to the two-bars base points along the y axis, in the three different simulations. Only the simulation data resulting from the external forces $sim1$ are used for the neural network training.

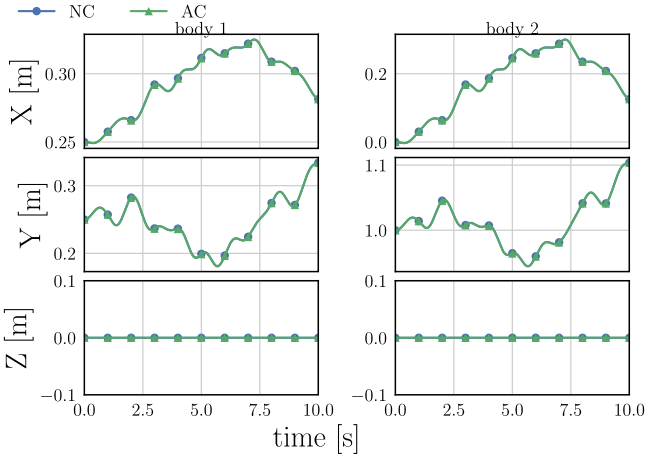

Fig. 13. Comparison of the simulation for the position of the bar centres of gravity in 'Natural Coordinates (NC)' $\boldsymbol{x}_n$ and 'Autoencoder Coordinates (AC)' $\hat{\boldsymbol{x}}_n$.

lightly increase its amplitude "sim3". The response for "sim1" is used to train the neural network. The comparison between the reference NC model and the proposed methodology for the centre of gravity positions of the two bars is shown in Fig. 13. Again, we see correspondence with respect to the training simulation.

In order to demonstrate the importance of including the dynamics in the loss function for the neural network training, Fig. 14 shows the response if only $\mathcal{L}_{rec}$ from Eq. (25) is employed for the neural network training. It shows the comparison of the centres of gravity for the two bars between the first reference simulation (NC) and:

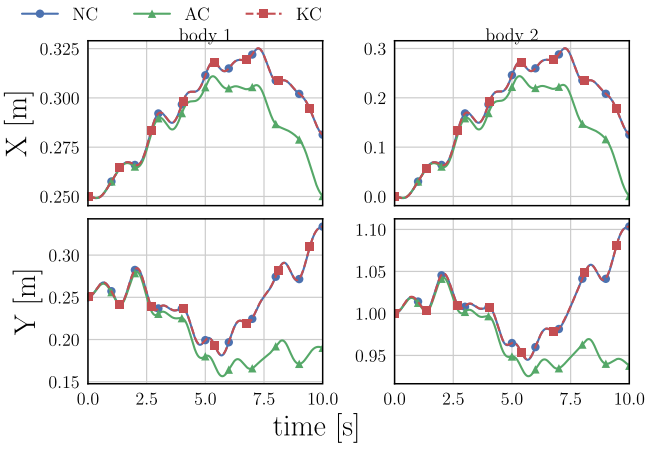

Fig. 14. Centres of gravity motion for the reference simulation $\boldsymbol{x}_n$ (NC), the reconstructed coordinates from the MC simulation $\hat{\boldsymbol{x}}_n$ with training only on reconstruction loss (AC) and the kinematically reconstructed coordinates $\tilde{\boldsymbol{x}}_n$ through the autoencoder application on the reference response (KC).

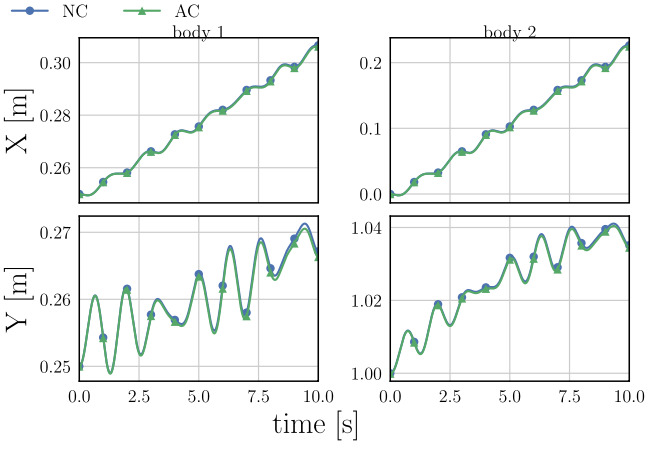

Fig. 15. Comparison of the simulation for the position of the bar centres of gravity in 'Natural Coordinates (NC)' $\boldsymbol{x}_n$ and 'Autoencoder Coordinates (AC)' $\hat{\boldsymbol{x}}_n$ in case of (lower) non-trained forces.

- the reconstructed coordinates from the minimal coordinates simulation, with training only on the reconstruction error (AC) – thus optimizing the neural network according to Eq. (25) and performing the simulation according to Algorithm 1 – to show that such dynamics prediction would fail;
- the reconstructed coordinates from a pure kinematic transformation through the encoder and decoder functions from the same optimization as above (KC) – thus $\tilde{\boldsymbol{x}}_n = \boldsymbol{h}_d(\boldsymbol{h}_e(\boldsymbol{x}_n))$ – to show that the training was successful but such mapping learned from the reconstruction loss cannot be used for model order reduction and dynamics predictions.

From this figure we clearly see that the reconstructive properties of the decoder are very accurate in the response of KC, and hence we can conclude that the decoder parameter training was well performed. However, the simulation response obtained with this decoder in minimal coordinates shows strong divergence from the training simulation. This demonstrates the high importance of the proposed loss function which also accounts for the dynamics.

Finally the procedure, using the proposed full loss function, is applied to the two non-trained simulations and the results are shown in Figs. 15 and 16. As no severe extrapolation occurs from the trained configurations, we again observe good accuracy for the proposed approach with a slightly increasing error over time.

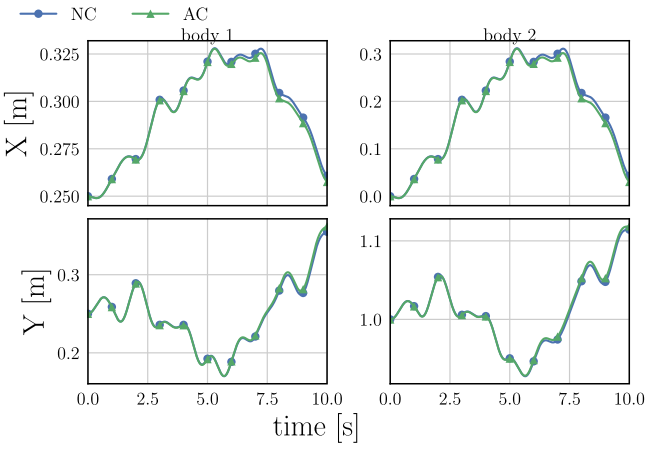

Fig. 16. Comparison of the simulation for the position of the bar centres of gravity in 'Natural Coordinates (NC)' $\boldsymbol{x}_n$ and 'Autoencoder Coordinates (AC)' $\hat{\boldsymbol{x}}_n$ in case of (marginally higher) non-trained forces.

6. Conclusion

The current work proposes a nonlinear model order reduction scheme which enables the reduction of a redundant coordinate rigid multibody model into a minimal coordinate model. In order to enable this reduction, an autoencoder framework is proposed. This approach has as main benefit that it can be employed non-intrusively with respect to the simulation software on existing rigid multibody models as it does not require direct access to the underlying model equations, and more in particular the constraint equations. The resulting model is a set of ordinary differential equations, rather than the common differential algebraic equations encountered in multibody simulation.

The autoencoder is employed to set up a continuous (nonlinear) mapping between a set of natural coordinates of the system and a set of minimal coordinates. A key point in setting up this map is to ensure sufficiently smooth derivatives with respect to the states in order to employ this map for projecting the dynamic equations in a Lagrangian framework. A dedicated loss function is proposed which allows to effectively take the dynamics into account at a feasible computational cost. This improved loss function is shown to be key for deploying the obtained neural networks in a dynamic simulation.

Two case studies are presented which demonstrate the good accuracy obtained with the presented approach as long as the range of motion lies within the training range. It is also demonstrated that the presented approach generalizes poorly outside of the training range and a sufficiently excited training simulation is therefore required.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The Research Fund KU Leuven, the Flanders Innovation & Entrepreneurship Agency within the ENDURANCE project, the AI impulse program of the Flemish Government, and the Research Foundation-Flanders (FWO) are gratefully acknowledged for their support.

Appendix. Loss function

In the work, it is proposed to perform the optimization in Eq. (34):

$$\underset{\mathcal{P}_e,\mathcal{P}_d}{\operatorname{argmin}} \left(\mathcal{L}_{rec}(\boldsymbol{x}_n, \mathcal{P}_e, \mathcal{P}_d) + \mathcal{L}_{sim}(\boldsymbol{x}_n, \mathcal{P}_d)\right)$$

It is worth mentioning that a similar result may have been obtained optimizing the reconstructed NC velocity and acceleration:

$$\mathcal{L}_{vel}(\boldsymbol{x}_n, \dot{\boldsymbol{x}}_n, \mathcal{P}_e, \mathcal{P}_d) = \frac{1}{n_t}\sum_{i=1}^{n_t}\left(\dot{\tilde{\boldsymbol{x}}}_n^i(\boldsymbol{x}_n^i, \dot{\boldsymbol{x}}_n^i, \mathcal{P}_e, \mathcal{P}_d) - \dot{\boldsymbol{x}}_n^i\right)^2 \tag{A.1}$$

$$\mathcal{L}_{acc}(\boldsymbol{x}_n, \dot{\boldsymbol{x}}_n, \ddot{\boldsymbol{x}}_n, \mathcal{P}_e, \mathcal{P}_d) = \frac{1}{n_t}\sum_{i=1}^{n_t}\left(\ddot{\tilde{\boldsymbol{x}}}_n^i(\boldsymbol{x}_n^i, \dot{\boldsymbol{x}}_n^i, \ddot{\boldsymbol{x}}_n^i, \mathcal{P}_e, \mathcal{P}_d) - \ddot{\boldsymbol{x}}_n^i\right)^2 \tag{A.2}$$

with:

$$\dot{\tilde{\boldsymbol{x}}}_n = \frac{\partial \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m}\,\dot{\tilde{\boldsymbol{x}}}_m \tag{A.3}$$

$$\ddot{\tilde{\boldsymbol{x}}}_n = \frac{\partial \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m}\,\ddot{\tilde{\boldsymbol{x}}}_m + \frac{\partial^2 \boldsymbol{h_d}}{\partial \tilde{\boldsymbol{x}}_m \partial \tilde{\boldsymbol{x}}_m}\,(\dot{\tilde{\boldsymbol{x}}}_m\,\dot{\tilde{\boldsymbol{x}}}_m^{\mathrm{T}}) \tag{A.4}$$

$$\dot{\tilde{\boldsymbol{x}}}_m = \frac{\partial \boldsymbol{h_e}}{\partial \boldsymbol{x}_n}\,\dot{\boldsymbol{x}}_n \tag{A.5}$$

$$\ddot{\tilde{\boldsymbol{x}}}_m = \frac{\partial \boldsymbol{h_e}}{\partial \boldsymbol{x}_n}\,\ddot{\boldsymbol{x}}_n + \frac{\partial^2 \boldsymbol{h_e}}{\partial \boldsymbol{x}_n \partial \boldsymbol{x}_n}\,(\dot{\boldsymbol{x}}_n\,\dot{\boldsymbol{x}}_n^{\mathrm{T}}) \tag{A.6}$$

leading to the optimization problem:

$$\underset{\mathcal{P}_e, \mathcal{P}_d}{\operatorname{argmin}}\,(\mathcal{L}_{rec} + \lambda_{vel}\,\mathcal{L}_{vel} + \lambda_{acc}\,\mathcal{L}_{acc}) \tag{A.7}$$

where the dependencies as in Eqs. (25), (A.1), (A.2) have been omitted for brevity and $\lambda_{vel},\ \lambda_{acc} \in \mathbb{R}$ are hyperparameters to weight the different position, velocity and acceleration scales.

In addition to the knowledge of the reference velocity and acceleration information, it can be noticed that such optimization requires the tuning of the two hyperparameters and calculating the derivatives of both the encoder and decoder functions, leading to higher computational cost and training time.

On the other hand, Eq. (34) does not require encoder-function derivatives or scaling hyperparameters, since it involves only position data. For such reasons, it was preferred.

References

[1] J.G. de Jalón, E. Bayo, Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge, Springer Science & Business Media, 2012.
[2] W. Schiehlen, Advanced Multibody System Dynamics: simulation and Software Tools, Vol. 20, Springer Science & Business Media, 2013.
[3] M. Hiller, A. Kecskemethy, Dynamics of multibody systems with minimal coordinates, in: Computer-Aided Analysis of Rigid and Flexible Mechanical Systems, Springer, 1994, pp. 61–100.
[4] S.M. Issa, K.P. Arczewski, Kinematics and dynamics of multibody system based on natural and joint coordinates using velocity transformations, J. Theoret. Appl. Mech. 36 (4) (1998) 905–918.
[5] E.J. Haug, Computer Aided Kinematics and Dynamics of Mechanical Systems, Vol. 1, Allyn and Bacon Boston, 1989.
[6] J. Cuadrado, D. Dopico, M.A. Naya, M. Gonzalez, Real-time multibody dynamics and applications, in: Simulation Techniques for Applied Dynamics, Springer, 2008, pp. 247–311.
[7] D.E. Kirk, Optimal Control Theory: An Introduction, Courier Corporation, 2004.
[8] D. Simon, Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches, John Wiley & Sons, 2006.
[9] O. Brüls, P. Duysinx, J.-C. Golinval, A model reduction method for the control of rigid mechanisms, Multibody Syst. Dyn. 15 (3) (2006) 213–227.
[10] F. Naets, R. Pastorino, J. Cuadrado, W. Desmet, Online state and input force estimation for multibody models employing extended Kalman filtering, Multibody Syst. Dyn. 32 (3) (2014) 317–336.
[11] T. Uchida, J. McPhee, Using Gröbner bases to generate efficient kinematic solutions for the dynamic simulation of multi-loop mechanisms, Mech. Mach. Theory 52 (2012) 144–157.
[12] Y. Bengio, et al., Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.
[13] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016,.
[14] J.W. Suk, J.H. Kim, Y.H. Kim, A predictor algorithm for fast geometrically-nonlinear dynamic analysis, Comput. Methods Appl. Mech. Engrg. 192 (22–24) (2003) 2521–2538.
[15] A. Oishi, G. Yagawa, Computational mechanics enhanced by deep learning, Comput. Methods Appl. Mech. Engrg. 327 (2017) 327–351.
[16] F. Meister, T. Passerini, V. Mihalef, A. Tuysuzoglu, A. Maier, T. Mansi, Deep learning acceleration of total Lagrangian explicit dynamics for soft tissue mechanics, Comput. Methods Appl. Mech. Engrg. 358 (2020) 112628.

[17] T. Uchida, A. Callejo, J.G. de Jalón, J. McPhee, On the Gröbner basis triangularization of constraint equations in natural coordinates, Multibody Syst. Dyn. 31 (3) (2014) 371–392.
[18] O. Brüls, P. Duysinx, J.-C. Golinval, The global modal parameterization for non-linear model-order reduction in flexible multibody dynamics, Internat. J. Numer. Methods Engrg. 69 (5) (2007) 948–977.
[19] A. Laulusa, O.A. Bauchau, Review of classical approaches for constraint enforcement in multibody systems, J. Comput. Nonlinear Dyn. 3 (1) (2008) 011004.
[20] J. Kövecses, J.-C. Piedboeuf, A novel approach for the dynamic analysis and simulation of constrained mechanical systems, in: ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers Digital Collection, 2003, pp. 143–152.
[21] W. Schiehlen, A. Rükgauer, T. Schirle, Force coupling versus differential algebraic description of constrained multibody systems, Multibody Syst. Dyn. 4 (4) (2000) 317–340.
[22] R. Wehage, E. Haug, Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic systems, J. Mech. Des. 104 (1) (1982) 247–255.
[23] F. Naets, T. Tamarozzi, G.H. Heirman, W. Desmet, Real-time flexible multibody simulation with global modal parameterization, Multibody Syst. Dyn. 27 (3) (2012) 267–284.
[24] F. Nguyen, S.M. Barhli, D.P. Muñoz, D. Ryckelynck, Computer vision with error estimation for reduced order modeling of macroscopic mechanical tests, Complexity 2018 (2018).
[25] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[26] L. Van Der Maaten, E. Postma, J. Van den Herik, Dimensionality reduction: a comparative, J. Mach. Learn. Res. 10 (2009) 66–71.
[27] F.J. Gonzalez, M. Balajewicz, Learning low-dimensional feature dynamics using deep convolutional recurrent autoencoders, 2018, arXiv preprint arXiv:1808.01346.
[28] B. Lusch, J. Kutz, S.L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nature Commun. 9 (1) (2018) 4950.
[29] S.E. Otto, C.W. Rowley, Linearly recurrent autoencoder networks for learning dynamics, SIAM J. Appl. Dyn. Syst. 18 (1) (2019) 558–593.
[30] K. Lee, K. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, 2018, arXiv preprint arXiv:1812.08373.
[31] A. Angeli, F. Naets, W. Desmet, A machine learning approach for minimal coordinate multibody simulation, in: Multibody Dynamics, Springer, 2019, pp. 417–424.
[32] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (153) (2018).
[33] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, in: 12th {USENIX} Symposium on Operating Systems Design and Implementation, {OSDI} 16, 2016, pp. 265–283.
[34] A. Angeli, F. Naets, W. Desmet, Deep learning of (periodic) minimal coordinates for multibody simulations, In: 16th International Conference on Multibody Systems, Nonlinear Dynamics, and Control (MSNDC), Presented At-2020 ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, IDETC/CIE2020, 2020.
[35] I.T. Jolliffe, Principal Component Analysis, Springer, 1986.
[36] M. Brand, Charting a manifold, in: Advances in Neural Information Processing Systems, 2003, pp. 985–992.
[37] E. Facco, M. d'Errico, A. Rodriguez, A. Laio, Estimating the intrinsic dimension of datasets by a minimal neighborhood information, Sci. Rep. 7 (1) (2017) 12140.
[38] W. De Groote, E. Kikken, E. Hostens, S. Van Hoecke, G. Crevecoeur, Neural network augmented physics models for systems with partially unknown dynamics: Application to slider-crank mechanism, 2019, arXiv preprint arXiv:1910.12212.
[39] M. Vermaut, F. Naets, W. Desmet, A flexible natural coordinates formulation (FNCF) for the efficient simulation of small-deformation multibody systems, Internat. J. Numer. Methods Engrg. 115 (11) (2018) 1353–1370.
[40] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.