# Dynamics modelling of continuum parallel robots

Andrea Gotelli

COLLEGE SCIENCES
DOCTORAL DE L'INGENIERIE
PAYS DE LA LOIRE ET DES SYSTEMES

CENTRALE NANTES

# MÉMOIRE DE DOCTORAT DE

## L'ECOLE CENTRALE DE NANTES

ECOLE DOCTORALE N° 602
Sciences de l'Ingénierie et des Systèmes
Spécialité : Robotique

Par
**Andrea GOTELLI**

## Modélisation Dynamique des robots parallèles continus

**Projet de recherche doctoral présenté et soutenu à Nantes, le 9 décembre 2024**
**Unité de recherche: UMR 6004 Laboratoire des Sciences du Numérique de Nantes (LS2N)**

**Rapporteurs avant soutenance** :

| | | |
|---|---|---|
| Christian DURIEZ | Directeur de recherche | INRIA Lille |
| Kanty RABENOROSOA | Professeur des Universités | Université de Franche-Comté |

**Composition du Jury :**

| | | | |
|---|---|---|---|
| Président : | Fabien CANDELIER | Professeur des universités | Aix-Marseille Université |
| Examinateurs : | Jessica BURGNER-KAHRS | Associate Professor | University of Toronto, Canada |
| | Christian DURIEZ | Directeur de recherche | INRIA Lille |
| | Kanty RABENOROSOA | Professeur des Universités | Université de Franche-Comté |
| Directeur de recherches doctorales : | | | |
| | Sébastien BRIOT | Directeur de recherche CNRS | Ecole Centrale de Nantes |
| Co-dir. de recherches doctorales : | | | |
| | Frédéric BOYER | Enseignant-Chercheur HDR | Ecole Mines-Telecom Bretagne Pays de la Loire IMT Atlantique Nantes |
| Co-enc.de recherches doctorales : | | | |
| | Vincent LEBASTARD | Maître assistant HDR | Ecole Mines-Telecom Bretagne Pays de la Loire IMT Atlantique Nantes |

# Aknowledgements

First and foremost, I would like to express my gratitude to the jury members: Christian Duriez, Fabien Candelier, Jessica Burgner-Kahrs, and Kanty Rabenorosoa. Thank you for accepting to evaluate my thesis and for dedicating your time and expertise to this endeavor. Your valuable insights and constructive feedback, arising from your diverse fields of expertise, have provided me with crucial directions and research avenues for the next steps in my career.

My sincere thanks go to my supervisors: Frédéric Boyer, Sébastien Briot, and Vincent Lebastard, who guided me throughout these three years of research. This journey has been transformative, not only in shaping me as a researcher but also in fostering my personal growth. Each of you imparted invaluable lessons. Vincent, thank you for teaching me how to approach numerical problems and for demonstrating methodologies to break down complex challenges into manageable solutions. Frédéric, our many mathematical discussions were profoundly enriching, each brimming with essential knowledge and concepts that deepened my understanding. Sébastien, your advice on planning and organization significantly enhanced my rigor and efficiency, enabling me to become a more structured and methodical researcher. Together, your guidance and support have made this experience deeply enriching.

I am also deeply grateful to Frédéric Jourdan, Olivier Kermorgant, and Pascal André for the invaluable discussions we had on numerical implementations and methods. These exchanges were instrumental in refining my understanding of key principles for simple yet effective software development.

Special thanks are due to my thesis mentor, Benoît Delahaye, for the numerous meetings and thought-provoking conversations that illuminated the intricacies of research and provided me with a long-term perspective on academic and career decisions.

Finally, I would like to thank my parents for their unwavering support, especially during the early stages of this journey and in moments of doubt, which helped me reaching this milestone. To Amandine, thank you for your patience, support, and for encouraging me to trust my instincts. Your presence has been a source of strength throughout this journey.

# Contents

# List of Figures

# List of Tables

# Nomenclature

**Acronyms**

| | |
|---|---|
| w.r.t. | With respect to |
| DoF | Degrees of freedom |
| CR | Continuum robot |
| CPR | Continuum parallel robot |
| TD-CR | Tendon-driven continuum robot |
| TD-CPR | Tendon-driven continuum parallel robot |
| N/A | Not applicable |
| UML | Unified Modelling Language |
| IVP | Initial value problem |
| BVP | Boundary value problem |
| NE | Newton-Euler |
| RNEA | Recursive Newton-Euler Algorithms |
| FEM | Finite element modelling |
| GE-FEM | Geometrically-exact finite element modelling |
| DER | Discrete elastic rod |
| PCC | Piecewise constant curvature |
| PCS | Piecewise constant strain |
| PLS | Piecewise linear strain |
| VSA | Variable strain approach |
| $IDM$ | Inverse Dynamics Model |

$TIDM$ .... Tangent Inverse Dynamic Model

SMMS..... Soft Multi-Body Mechanical Systems

*i.e.* ..... *id est* Latin for "that is"

*e.g.* ..... *exempli gratia* Latin for "for example"

**Lie Algebra Notations**

$\mathcal{F}_e$ ..... The inertial frame

$\mathcal{F}_b$ ..... A body frame

$g_b$ ..... Pose of a (body) frame $\mathcal{F}_b$ in the base of $\mathcal{F}_e$

${}^a g_b$ ..... Pose of a (body) frame $\mathcal{F}_b$ in the base of $\mathcal{F}_a$

$\eta_b$ ..... Twist of $\mathcal{F}_b$ relative to $\mathcal{F}_e$ expressed in the base of $\mathcal{F}_b$

$\eta_{b/a}$ ..... Twist of $\mathcal{F}_b$ relative to $\mathcal{F}_a$ expressed in the base of $\mathcal{F}_b$

$F_b$ ..... Wrench exerted on $\mathcal{F}_b$ expressed in its local base

${}^a F_b$ ..... Resultant of the wrench $F_b$ transported to $\mathcal{F}_a$

**Mathematical Notations**

$\mathbb{1}_3$ ..... The $3 \times 3$ identity matrix

$\mathbb{1}_n$ ..... An $n \times n$ identity matrix

$\mathbb{0}_n$ ..... An $n \times n$ matrix of zeros

$\mathbb{0}_{n \times m}$ ..... An $n \times m$ matrix of zeros

**Physical Quantities**

$\mathcal{A}$ ..... Area

$\mathcal{H}$ ..... Stiffness matrix of a Cosserat rod cross-section

$\mathcal{M}$ ..... Inertia tensor for a rigid body

$\mu$ ..... Friction coefficient

$\rho$ ..... Specific mass

$E$ ..... Young modulus

$G$ ..... Shear modulus

$I_{xx}$ ..... Geometric moment of inertia along the $x$ axis

$I_{yy}$ ..... Geometric moment of inertia along the $y$ axis

$I_{zz}$ ..... Geometric moment of inertia along the $z$ axis

$m$ ..... Mass

**Variable Strain Approach Notations**

$q$ ..... Vector of generalized (elastic) coordinates

$Q$ ..... Vector of generalized forces

$Q_{act}$ ..... Vector of generalized actuated forces

$\Lambda$ ..... Internal stress-wrench

$\Lambda_{act}$ ..... Internal actuated stress-wrench

$\overline{F}_{\text{ext}}$ ..... External distributed wrench

$\xi$ ..... Rod strain field

$\xi^0$ ..... Rod reference strain field

$\epsilon$ ..... Rod strain

$\mathcal{K}_{\epsilon\epsilon}$ ..... Generalized stiffness matrix

$\mathcal{D}_{\epsilon\epsilon}$ ..... Generalized damping matrix

# List of Publications

The following publications are author's publications related on the topics of this thesis, and realized during the time of this Ph.D:

- **International Journals:**
  - F. Boyer, A. Gotelli, P. Tempel, V. Lebastard, F. Renda, and S. Briot, "Implicit time integration simulation of robots with rigid bodies and Cosserat rods based on a Newton-Euler recursive algorithm," IEEE Transactions on Robotics, pp. 1-20, 2023.
- **International Conferences:**
  - A. Gotelli, F. Zaccaria, O. Kermorgant, and S. Briot, "A gazebo simulator for continuum parallel robots," in Altuzarra, O., Kecskeméthy, A. (eds) Advances in Robot Kinematics 2022. ARK 2022. Springer Proceedings in Advanced Robotics, vol. 24, Springer, Cham, 2022, pp. 248–256

# Chapter 1

# Introduction

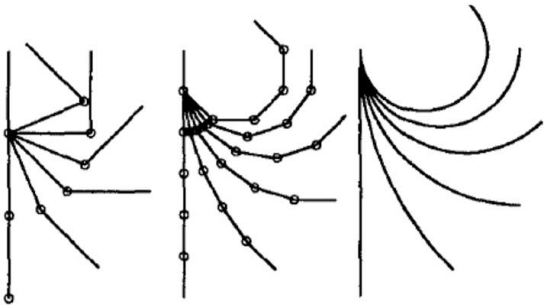
This thesis is situated within the field of continuum robotics which comprises robots that are composed of both rigid and flexible components. The latter enables these robots to undergo significant deformations. In this field we find Continuum Robots (CR) and Continuum Parallel Robots (CPR), the latter being the system of interests that will be the subject of this study. In order to fully realise their potential, it is necessary to gain an understanding of the most appropriate methodology for modelling them in a comprehensive manner.

In this introductory chapter we presents an overview on CRs and CPRs and we outline the contribution of this thesis.

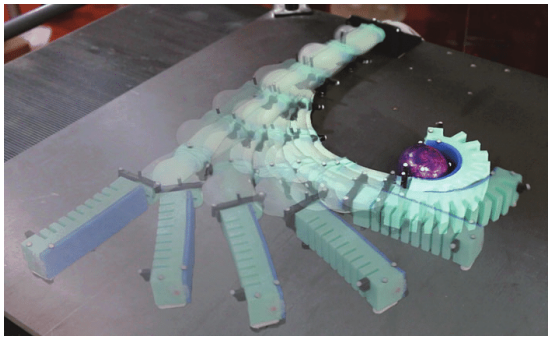
## 1.1 Introduction to Continuum Robotics

To provide a comprehensive overview of continuum robotics, this section begins with an overview of Continuum Robots (CR), the precursors to CPRs, which fuelled interest in the field starting in the early 1990s.

CRs are a specific type of robot that is formed with a thin elastic component, which undergoes significant displacements and deformations [1] (see Figure 1.1b). These robots are typically actuated by a special actuation system distributed along its length. The initial conceptualization of such robots dates back to the 1960s [2], while the late 1990s saw a significant rise in research on continuum robots [3], particularly in the fields of manipulation [4] and medical applications [5, 6]. Continuum robots are, essentially, manipulators with a design inspired by the bio-kingdom, where the actuation comes from muscles, tendons, fibers and joints. In comparison to rigid link serial robots, these robots exhibit greater flexibility and dexterity. The distinction between a robot with rigid links and continuum robots is exemplified in Figure 1.1a. An example of a continuum robot grasping an object is shown in Figure 1.1b, while other types of CRs are presented in Figure 1.2. As they undergo continuous deformation, they exhibit infinite Degrees of Freedom (DoF), *i.e.,* their shape may undergo changes without affecting the motion of their end. Consequently, they can be utilized in situations where the execution of intricate trajectories is required,



(a) Progress from rigid link robot to continuum robot [1]: from a rigid links robot (left) to a serpentine robot (center) and, finally, to a continuum robot (right).



(b) Trajectory of a continuum robot grasping an object [7].

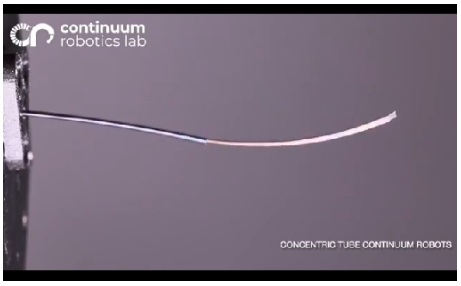
Figure 1.1: A continuum robot grasping an object (left) and a representation of rigid, serpentine and continuum robot (right).

such as navigating the shape of an object or traversing a curvilinear path. Furthermore, their flexibility has a considerable impact on safety when employed in collaborative tasks, although this does result in a reduction in payload capacity when compared to traditional rigid robotics.

In order to enhance the payload capacity while maintaining the benefits associated with a deformable structure, Continuum Parallel Robots (CPR) present a viable alternative. A CPR is a type of parallel robot having multiple slender bodies, typically made of thin elastic rods, connected in parallel. In order to define the structure of the robot, these rods are constrained with rigid bodies. Usually, the robot base constrains the proximal ends of the rods, while their distal ends are connected to a rigid platform (see Figure 1.3a). Other types of CPRs, with more complex structures, are studied in the literature (see Figures 1.3b– 1.4b). In general, this design allows to reduce the number of intermediate joints and the inertia of the robot. Compared to standard parallel robots, they exhibit greater compliance, larger work-space and a reduction of assembly and maintenance costs. However, CPRs are subject to certain inherent limitations associated with their definition and construction: *i*) their model incorporates non-linearities derived from the deformable bodies *ii*) their parallel architecture often precludes the derivation of an analytical solution and *iii*) in comparison to rigid robots, they exhibit a reduced payload capacity. Early research on these robot started in the early 2000s [16, 17, 18, 19]. In the late 2010s, the field attracted the attention of numerous researchers engaged in the investigation of statics modelling of CPRs and its applications [20]. Despite the growing interest in statics modelling, the dynamics modelling of CPRs remains an open problem in the current state of the art.

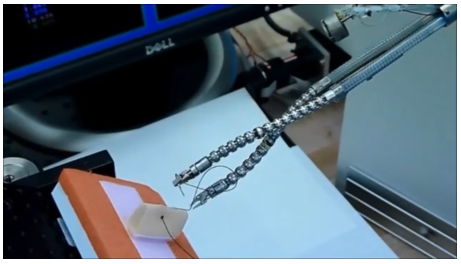
The aforementioned aspects of CPRs, particularly those concerning their non-linear model and the limited research on their dynamics modelling, provided the motivation for the research presented in this thesis. The following section presents a detailed examination

(a) TD-CR interacting with operator hand [8].




(b) Concentric tube robot[9].

(c) A swimming fish whose tail is a continuum robot [10].



(d) Two TD-CR used in surgery [11].

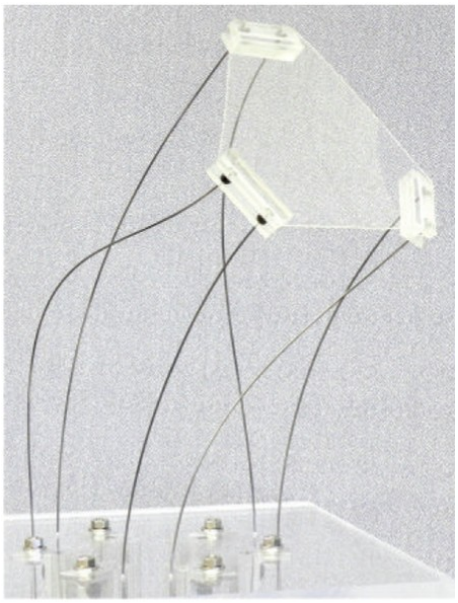
Figure 1.2: Different types of Continuum Robots (CR): a TD-CR (a), a concentric-tube robot (b), a bio inspired swimmer (tail as a CR c) and two TD-CR in a surgery (d).

of our contributions to the current state of the art, along with an overview of the structure of this document.

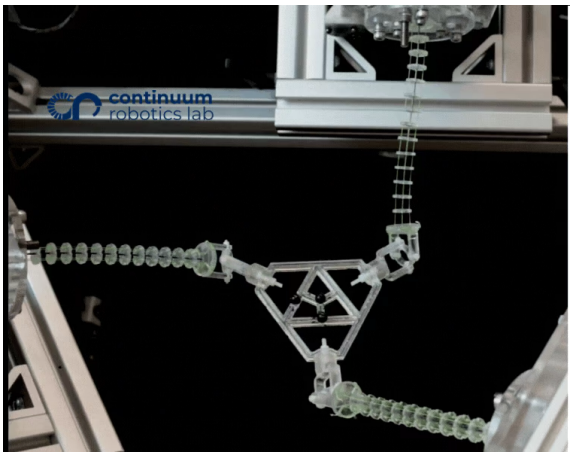
## 1.2 Contribution and Structure of this Thesis

This thesis is part of the COSSEROOTS ANR project, which aims to produce, for the first time, a comprehensive and unified set of methods and models dedicated to the modelling and control of slender robots with actuated large deformations. In this context, the primary objective of this thesis is to develop a dynamics model for CPRs grounded in the well-established methodologies of rigid-body robotics. To achieve this, the approach will utilize Newton-Euler (NE) recursive algorithms, which are not restricted to any specific CPR architecture. This flexibility allows the approach to be applied or extended to various architectural configurations of interest. The current state of the art and the findings of this study are outlined in the following chapters, for which a concise overview is provided.

In Chapter 2 we provide an overview of the current state of the art in modelling deformable bodies and CPRs. The chapter commences with an exposition of the Cosserat rod theory, which has become the prevailing approach for characterising slender deformable bodies in a considerable number of communities, including those of soft and continuum robotics, finite element modelling and computer graphics. This theory forms the basis for



(a) Stewart-Gough like CPR [12] designed with six slender bodies sliding through the base (sliding joints).



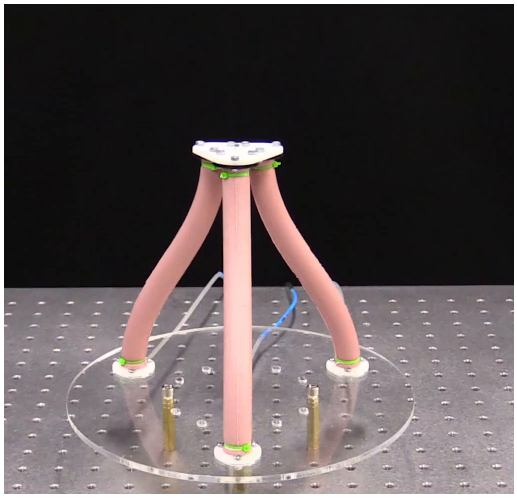

(b) Reconfigurable CPR having three actuated limbs placed on three actuated prismatic joints and connected to a platform with spherical joints [13].

Figure 1.3: Different types of CPRs in the literature: one with a localized actuation at the base (a) and one with a combination of localized actuation at the base and distributed tendon-driven actuation along the deformable bodies (b).

deriving a set of Partial Differential Equations (PDE) describing the kinematics of a slender deformable bodies. Subsequently, the PDE governing the dynamics of internal efforts will be identified through the application of Hamilton's principle. We will then provide an overview of the various models employed to resolve the PDEs, with a specific focus on the Variable Strain Approach (VSA) of [21], which has been selected as the foundation for this thesis. Having outlined the selected approach for modelling slender deformable bodies, we address a concise overview of the state of the art in modelling CPRs. This will include a summary of the numerous developments in the field of statics modelling, as well as an analysis of the limited progress made in the domain of their dynamics modelling, which motivates the research presented in this thesis.

In Chapter 3, we present our approach to the dynamics modelling of hybrid structures composed of both rigid and deformable bodies, arranged as either a serial or parallel manipulator, or a locomotor (*i.e.,* a system with a free-floating base). By utilizing the VSA to model deformable bodies as distributed articulations, we can incorporate the complexities of their dynamics within the classical Recursive Newton-Euler Algorithms (RNEA) used in rigid robotics. A standard structural dynamics formulation is then derived to address the dynamics of the whole structure. In the case of closed kinematics loops, the formulation consists of set of Differential Algebraic Equations (DAE) in the usual

(a) Delta-like CPR with intermediate constraints on the deformable rods between the base and the distal end [14].



(b) CPR with using pressure chambers to provide motion to its platform [15].

Figure 1.4: Different types of CPRs in the literature: one with intermediate constraints on the deformable rods (a) and one with pressure chambers to provide motion to its platform (b).

Lagrangian form, while the time integration is addressed with an implicit scheme. Finally, we will provide examples with different case studies: a flying rod, a soft manipulator actuated by drones as well as a bio-inspired tendon actuated swimmer.

Then, in Chapter 4 we specialize the approach of Chapter 3 to the case of CPRs. In this case the system of DAEs is time integrated with an implicit scheme that dampens the numerical noises affecting the dynamics of stiff systems, such as CPRs. The chapter then proceeds with the experimental validation of the approach using two planar CPR prototypes that differ in their actuation: one is distributed and the other is localised. The first prototype will be used to compare the simulation output with recorded data, demonstrating consistency with the predicted behaviour, while the goal of the second prototype is to show the model's ability to capture the robot's dynamics behaviour as it crosses a singularity.

In Chapter 5, we detail the proposed toolbox, which implements the approaches outlined in Chapters 3 and 4. The implementation, based on an Object-Oriented Programming (OOP) paradigm, is presented in terms of its software architecture and interfaces. Specifically, we detail how a system, either a hybrid structure as discussed in Chapter 3 or a CPR of Chapter 4, can be described as a generic tree of nodes. This tree is given as input to a generic dynamics simulator, which implements the Newton-Euler algorithm discussed in the previous chapters. Subsequently, we detail how the architecture can account for the different types of articulations (rigid joints and deformable bodies) and different implicit schemes. The chapter concludes with a case study in which we detail the various

components described in the chapter, as well as the differences between the various time implicit schemes, discussed in the previous chapters, and the spatial integrators required by the Cosserat rod PDEs.

Finally, in Chapter 6 we list our conclusions and the future perspectives.

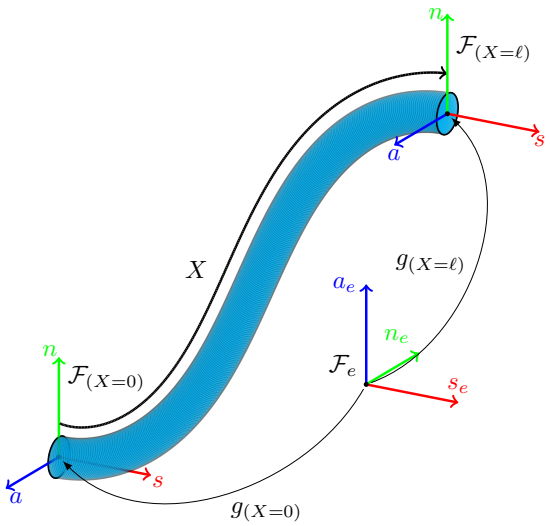
# Chapter 2

# State of The Art

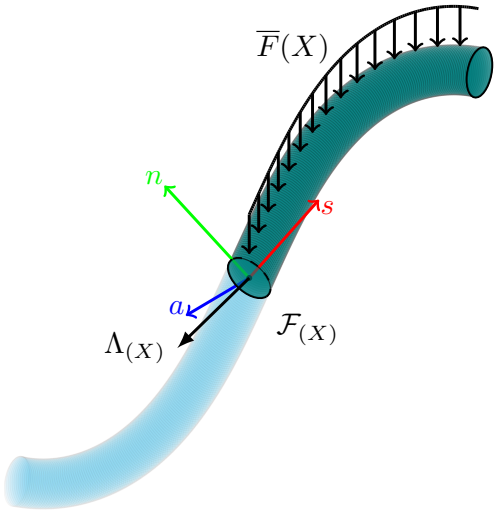
In order to provide an overview on the modelling of Continuum Parallel Robots (CPR), it is necessary first address the modelling of slender deformable bodies, a subject which is widely discussed in the literature. The presentation of the state of the art will be structured around the following three fundamental aspects inherent to any mechanical modelling [22]: *i*) a physical model (here the Cosserat rod theory), *ii*) a parametrization of the aforementioned model (*i.e.,* definition of its configuration space) and *iii*) a dynamics principle that offers a closed formulation of the system's dynamics within its configuration space. In accordance with the aforementioned order, we start with a concise overview of the Cosserat rod theory, in order to obtain a set of differential equations governing the dynamics of the slender deformable body. We will recall the rod configuration space, and, subsequently, we will obtain a closed formulation of the rod dynamics by application of the Hamilton principle. Then, we will provide a concise overview of some of the available methods for the numerical resolution of the rod dynamics. In particular, we provide a concise overview of solving the dynamics in the "strong-form" using the shooting method, and how alternative methods, based on reducing the rod's configuration space, are employed to solve the "weak-form" of the rod dynamics. The former consists of the governing equations of motion and their boundary conditions, while the latter is an integral form of these equations, which is need to formulate a problem of finite dimension [23]. Once an appropriate framework for modelling deformable bodies is established, we then provide a concise review of how the modelling of CPRs, both in terms of statics and dynamics, is typically addressed in the literature.

## 2.1 Cosserat Rod Theory

The Cosserat theory was proposed by the Cosserat brothers in 1909 [24] to model slender deformable bodies. Nowadays, with the emergence of continuous robotics, Cosserat rod theory has progressively imposed itself as one of the standards for modelling robots composed of rods actuated at their boundaries, or along them in a distributed way. In the first category, one finds the Concentric Tube Robots (CTR) [25, 26] and CPR [26], while the second includes tendon driven continuum robots (TD-CR) [27, 28], soft robots equipped with pressure chambers [29, 30] or hyper-redundant swimming robots inspired of slender




(a) Parametrisation of the rod as a one dimensional body, in the dimension of its arc length $X \in [0, \ell]$.




(b) Graphical representation of the internal stress-wrench $\Lambda$ acting from the segment $> X$ to the rest of the rod through the cross section at $X$.

Figure 2.1: Representation of the Cosserat rod with its local (body) frames (left) and its internal stress-wrench (right).

animals such as snakes and fish [31]. In this section, we recall the principle and provide a detailed account of the developments to obtain the set of PDEs that describe the rod dynamics [21, 32].

### 2.1.1 Geometrical Description

With the Cosserat rod theory, the rod is conceptualised as a slender body and virtually sectioned an infinite number of times, thereby creating a continuum of cross-sections stacked on top of each other. Consequently, a Cosserat rod can be defined as a one-dimensional body in the dimension of its arc length, denoted $X \in [1, \ell]$ with $\ell$ the length of the rod (Figure 2.1a). It is also assumed that the cross-sections remain planar and rigid, *i.e.,* there is no radial shear. For each cross-section, the centre of mass and the principal axis of inertia can be used to define a local frame. Denoting such a frame $\mathcal{F}(X)$ (see Appendix A), the origin is situated at the centre of the cross-section, with its directors $n$ and $a$ aligned with the principal axis of inertia, thus forming the cross-section plane, while $s$ is aligned with the plane's normal (Figure 2.1a). The pose of $\mathcal{F}(X)$ with respect to the inertial frame $\mathcal{F}_e$ is indicated by $g(X) \in SE(3)$. Hence, the rod configuration space is naturally defined as

$$\mathcal{C}_1 = \{g : X \in [0, \ell] \mapsto g(X) \in SE(3)\}. \tag{2.1}$$

The space-rate of variation of the pose along the rod domain is defined as:

$$g' = \frac{dg}{dX} = g\widehat{\xi}, \qquad \xi = \begin{bmatrix} K \\ \Gamma \end{bmatrix}, \tag{2.2}$$

where $\xi \in se(3) \cong \mathbb{R}^6$ is the cross-section space-twist, commonly denoted as strain field, with $K \in so(3) \cong \mathbb{R}^3$ and $\Gamma \in \mathbb{R}^3$ the space-rate of variation of $R$ and $r$, respectively. In detail, $K = (K_x, K_y, K_z)^T$ express the curvature of the rod in terms of torsion ($K_x$) and bending along the local $y$ and $z$ axis (with $K_y$ and $K_z$, respectively), while $\Gamma = (\Gamma_x, \Gamma_y, \Gamma_z)^T$ defines the rod stretching ($\Gamma_x$) and axial-shear ($\Gamma_y$ and $\Gamma_z$). When the rod is in a state of rest, $\xi^0 = ({K^0}^T, {\Gamma^0}^T)^T$ defines its reference strain field (*e.g.,* for a straight rod we have $K^0 = 0_{3\times1}$ and $\Gamma^0 = (1, 0, 0)^T$ where $\Gamma_x = 1$ means no stretching nor compression). We can express the strain of the rod, *i.e.,* the difference between the current strain field and the reference one, as $\epsilon = \xi - \xi^0$. With the introduction of $\xi$, we obtain a second definition of the Cosserat rod configuration space:

$$\mathcal{C}_2 = SE(3) \times \mathbb{S}, \qquad \mathbb{S} = \{\xi : X \in [0, \ell] \mapsto \xi(X) \in \mathbb{R}^6\}, \tag{2.3}$$

which states that the configuration space of a Cosserat rod is defined by the $SE(3)$ pose of its first cross-section in conjunction with the configuration space of its shapes, or "shape space", $\mathbb{S}$. As the rod is a continuous system, $\mathbb{S}$ is a functional space: an infinite dimensional space of $X$-parametrized curves in $\mathbb{R}^6$. We will see how this functional space can be reduced to a finite dimension vector space in Section 2.3.

### 2.1.2 Rod Kinematics

Once established a representation of the rod configuration, we now address the cross-sections velocities and accelerations. To that end, let us first define the twist $\eta = \left(g^{-1}\dot{g}\right)^\vee$ representing the velocity of $\mathcal{F}(X)$ expressed in its local coordinates (see Appendix A). Since the centreline coordinate $X$ is independent of time $t$ (see Remark 1 below), their derivatives are also independent and can be inverted as follows:

$$\frac{\partial g'}{\partial t} = \frac{\partial^2 g}{\partial X \partial t} = \frac{\partial \dot{g}}{\partial X}. \tag{2.4}$$

The left-most and right-most terms have the following definitions:

$$\frac{\partial g'}{\partial t} = \dot{g}\widehat{\xi} + g\widehat{\dot{\xi}}, \qquad \frac{\partial \dot{g}}{\partial X} = g'\widehat{\eta} + g\widehat{\eta}', \tag{2.5}$$

where the terms $\dot{\xi}$ and $\eta'$ respectively represent the time-rate of variation of the rod strain field and the space-rate of variation of the cross section twist. Combining Equation (2.5) into Equation (2.4) and isolating the terms containing the cross derivatives: $\dot{\xi}$ and $\eta'$, the latter has the following expression:

$$\widehat{\eta}' = \widehat{\eta}\widehat{\xi} - \widehat{\xi}\widehat{\eta} + \widehat{\dot{\xi}}, \tag{2.6}$$

Reminding now to the definition of the Lie commutator (Appendix A.3), we can simplify the formulation as follows:

$$\eta' = -\mathrm{ad}_\xi \eta + \dot{\xi}. \tag{2.7}$$

To complete the study on the rod kinematics, $\dot{\eta}$ is obtained by differentiating Equation (2.7) with respect to time:

$$\dot{\eta}' = -ad_{\dot{\xi}}\eta - \mathrm{ad}_\xi \dot{\eta} + \ddot{\xi}. \tag{2.8}$$

The combination of the aforementioned equations yields the PDE that describes the kinematics of a Cosserat rod.

$$\begin{cases} g' &= g\widehat{\xi}, \\ \eta' &= -\mathrm{ad}_\xi \eta + \dot{\xi}, \\ \dot{\eta}' &= -ad_{\dot{\xi}}\eta - \mathrm{ad}_\xi \dot{\eta} + \ddot{\xi}. \end{cases} \tag{2.9}$$

**Remark 1.** *The cross-section twist and pose are both dependent on $(X, t)$. Explicitly, they depend on $X$, while they depend on $t$ implicitly, as their time-rate of variation is obtained from the time integration of the rod dynamics model.* ∎

### 2.1.3 Material Properties

In accordance with the Cosserat rod theory, the following assumptions are made regarding the material properties: elastic behaviour, linear stress-strain characteristics and isotropy. Moreover, expressing the rod kinematics in the local cross-section frame allows the definition of material and geometric properties in the same frame, thereby ensuring their invariance with respect to rod configuration [33]. We then denote the area of the cross section with $\mathcal{A}$, the moments of inertia along the principal axes with $I_{yy}$, $I_{zz}$ and their resultant with $I_{xx} = I_{yy} + I_{zz}$, the Young modulus as $E$, the the Shear modulus as $G$ and the distributed mass with $\rho$. The resulting stiffness and inertia of the rod cross sections can be expressed as:

$$\mathcal{H} = \mathrm{diag}(GI_{xx}, EI_{yy}, EI_{zz}, E\mathcal{A}, G\mathcal{A}, G\mathcal{A}), \tag{2.10}$$

$$\mathcal{M} = \mathrm{diag}(\rho I_{xx}, \rho I_{yy}, \rho I_{zz}, \rho\mathcal{A}, \rho\mathcal{A}, \rho\mathcal{A}), \tag{2.11}$$

where $\mathcal{H} \in \mathbb{R}^{6\times6}$ is the matrix of the Hooke tensors and $\mathcal{M} \in \mathbb{R}^{6\times6}$ is the inertia matrix of the rod cross-sections.

**Remark 2.** *Note that we defined $\mathcal{H}$ and $\mathcal{M}$ as constant along the rod domain. This assumption can be relaxed by attributing to their elements an appropriate function defining the evolution of material and geometric properties along the rod domain.* ∎

### 2.1.4 Constitutive Laws

The material properties can be used to relate the internal strain ($\epsilon$) with the internal stress-wrench $\Lambda \in se^*(3) \cong \mathbb{R}^6$ with the internal constitutive laws. For a rod with with an internal actuation, it takes the following form:

$$\Lambda = \Lambda_{act} + \mathcal{H}\epsilon, \tag{2.12}$$

where $\Lambda$ indicates the efforts that is transmitted from one section of the rod to another (Figure 2.1b) and $\Lambda_{act}(X,t)$, which explicitly depends on time, is the internal wrench applied by an actuation system distributed along the rod body. The field of $\Lambda_{act}$ is to be defined in accordance with the distributed actuation acting along the slender body. Equation (2.12) can be interpreted as follows: the stress-wrench acting on the rod is balanced by the rod's internal elasticity and, if present, the effect of distributed internal actuation. To approximate the material friction between cross-sections, with a basic viscous damping model, the aforementioned constitutive law can be adapted as follows:

$$\Lambda = \Lambda_{act} + \mathcal{H}\epsilon + \mathcal{D}\dot{\epsilon}, \tag{2.13}$$

where $\mathcal{D}(X) \in \mathbb{R}^{6\times 6}$ is the damping matrix along the rod at $X$. Using the Kelvin-Voigt model, it is convenient to impose $\mathcal{D} = \mu\mathcal{H}$, with $\mu$ a scalar damping coefficient.

### 2.1.5 Hamilton Principle for Equation of Motion

In this section we apply the Hamilton principle to obtain a closed formulation of the internal stress-wrench in the rod configuration space. Other principles, such as Newton's laws and the principle of virtual works, would also yield comparable formulations [34]. However, the Hamilton principle is the most consistent with the geometry of the configuration space, which, in this context, is the Lie group [32]. Furthermore, it provides the PDEs and the boundary conditions in a systematic manner.

To obtain the rod equations of motion we need to start from its Lagrangian dynamics model, obtained from the rod kinetic and potential energy. Defining the cross-section potential energy as $\mathcal{U}(\xi) = \frac{1}{2}\epsilon^T\mathcal{H}\epsilon$ and the kinetic energy as $\mathcal{T}(\eta) = \frac{1}{2}\eta^T\mathcal{M}\eta$, the Lagrangian takes the following form:

$$\begin{aligned}\mathcal{L}(\eta,\xi) &= \mathcal{T}(\eta) &&- \mathcal{U}(\xi),\\ &= \frac{1}{2}\eta^T\mathcal{M}\eta &&- \frac{1}{2}\epsilon^T\mathcal{H}\epsilon.\end{aligned} \tag{2.14}$$

We introduce a set of wrenches (see Appendix A) expressed in the local coordinate frame: $\overline{F}_{\text{ext}}$ as the distributed wrench acting on the rod body, $F_{\text{ext},0}$ and $F_{\text{ext},\ell}$ as two wrenches applied to the rod proximal and distal cross-section, expressed in the base of $\mathcal{F}(X=0)$ and $\mathcal{F}(X=\ell)$, respectively. We can now define the virtual work of external wrenches $\delta\mathcal{W}$ as follows:

$$\delta\mathcal{W} = \int_0^\ell \delta\zeta^T\ \overline{F}_{\text{ext}}\ dX + \delta\zeta(0)^T F_{\text{ext},0} + \delta\zeta(\ell)^T F_{\text{ext},\ell}, \tag{2.15}$$

where the variation (virtual displacement) $\delta\zeta \in se(3)$, defined as $\delta\zeta = \left(g^{-1}\delta g\right)^\vee$, is applied on the set of poses of the Cosserat rod in such a way that $\delta\zeta(t_0) = \delta\zeta(t_1) = 0$. Applying the concept of action and stationary condition of the Hamilton principle [32], the virtual work of external wrenches can be related to the variation of cross-section Lagrangian as follows:

$$\int_{t_1}^{t_2}\int_0^\ell \delta\mathcal{L}\ dX\ dt = -\int_{t_1}^{t_2}\delta\mathcal{W}\ dt. \tag{2.16}$$

In order to obtain an expression for $\delta\mathcal{L}$ we use standard derivative of multivariable functions and similar relations to (2.7) with $\left(\frac{\partial}{\partial t}, \delta\right)$ $\left(\frac{\partial}{\partial X}, \delta\right)$ instead of $\left(\frac{\partial}{\partial X}, \frac{\partial}{\partial t}\right)$ in order to exchange variations and $(X, t)$ derivatives. Then, usual by parts integration can be applied to obtain:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \eta}\right) + \frac{d}{dX}\left(\frac{\partial \mathcal{L}}{\partial \xi}\right) - ad_{\eta}^{T}\frac{\partial \mathcal{L}}{\partial \eta} - ad_{\xi}^{T}\frac{\partial \mathcal{L}}{\partial \xi} = \overline{F}_{\text{ext}}, \tag{2.17}$$

which is the closed formulation of the rod dynamics. Equation (2.17) is provided with the boundary conditions:

$$\left(\frac{\partial \mathcal{L}}{\partial \xi}\right)(X=0) = F_{\text{ext},0}, \qquad \left(\frac{\partial \mathcal{L}}{\partial \xi}\right)(X=\ell) = -F_{\text{ext},\ell}. \tag{2.18}$$

Now from the definition of $\mathcal{L}$ (2.14), we can compute its partial derivatives with respect to $\eta$ and $\xi$ as:

$$\frac{\delta \mathcal{L}}{\delta \eta} = \mathcal{M}\eta, \qquad \frac{\delta \mathcal{L}}{\delta \xi} = -\mathcal{H}\left(\xi - \xi^{0}\right) = -\Lambda, \tag{2.19}$$

where in the right side of (2.19) we used the constitutive law (2.12) (with $\Lambda_{act} = 0$ for a passive rod). Introducing their definition in (2.17) and solving for $\Lambda$, we obtain the PDE of the internal stress-wrench of a Reissner-Simo rod [35]:

$$\begin{cases} \Lambda' & = \mathrm{ad}_{\xi}^{T}\Lambda + \mathcal{M}\dot{\eta} - \mathrm{ad}_{\eta}^{T}\mathcal{M}\eta - \overline{F}_{\text{ext}}, \\ \Lambda(X=0) & = -F_{\text{ext},0}, \\ \Lambda(X=\ell) & = -F_{\text{ext},\ell}, \end{cases} \tag{2.20}$$

where the bottom rows are the boundary conditions. In other words, Equation (2.20) consists in the rod equation of motion with its boundary conditions (*i.e.,* the strong form of the rod dynamics).

### 2.1.6 Conclusions on the Cosserat rod Theory

This section provided a concise overview of the Cosserat rod theory [24]. The cross section kinematics was defined along the rod arc length with a set of PDEs. The material properties and internal constitutive laws were outlined and applied to derive the PDE for the internal stress-wrench, using the Hamilton principle. The resulting set of PDEs can be applied to any sufficiently slender rod-like structure.

A number of techniques have been proposed in the literature for the modelling of deformable bodies. In what follows, we present a summary of these modelling strategies classified into two categories: the one related to the direct resolution of the continuous Cosserat model and those based on model reduction techniques. The former solves the rod dynamics in its "strong-form" while the latter group presents methodologies that reduce the rod configuration space to a finite-dimension vector space, thereby solving the rod dynamics as an integral on its reduced configuration space, *i.e.,* in its "weak-form".

**Remark 3.** *To obtain the dynamics of a system, two well know approaches in robotics are the Direct Dynamics Model (DDM) and the Invese Dynamics Model (IDM). The DDM allows to obtain the system acceleration (directly) from the current configuration, velocities and forces either coming from external factors (such as gravity) or applied by some actuators. On the other hand, the IDM uses the current kinematics state of the system (configuration, velocities and accelerations) to computes the external forces and the corresponding actuation efforts that are required to achieve such kinematics state. In what follows, we will remark how the Shooting method corresponds to the DDM applied to the Cosserat continuum while the model-reduction techniques actually corresponds to the IDM.*
■

## 2.2 Resolution of Continuous Cosserat Model

In this section we describe how the problem in Equation (2.9) and (2.20) can be solved directly with the Shooting method [26].

### 2.2.1 Applications

The shooting method is particularly appealing due to its intuitive nature and the simplicity of its implementation. This method is primarily used for the static modelling of slender deformable bodies and soft or continuum manipulators. Some of its application includes continuum robots [5, 36, 37] and CPRs as we will detail in Section 2.4.

### 2.2.2 Description

#### Statics Modelling

In order to detail the statics modelling of a Cosserat rod with the shooting method, we recall the PDEs that describe the rod's kinematics and dynamics, as presented in equations (2.9) and (2.20), which, for the statics modelling, simplifies to the following set of Ordinary Differential Equations (ODEs):

$$\begin{cases} g' & = g\widehat{\xi}, \\ \Lambda' & = ad_{\xi}\Lambda - \overline{F}_{\text{ext}}, \\ \xi & = \mathcal{H}^{-1}\left(\Lambda - \Lambda_{act}\right) + \xi^{0}, \\ g(X=0) = \mathbb{1}_4, & \quad \Lambda(X=\ell) \qquad = -F_{\text{ext},\ell}, \end{cases} \tag{2.21}$$

where $\Lambda_{act}$, in the case of a tendon actuation, can be computed as described in [38, 34]. This method is applied to the statics modelling of a Cosserat rod as a BVP, which in (2.21) act on the pose of the proximal cross section and the wrench at the rod distal end. Nevertheless, alternative boundary conditions may be specified. The solution of such BVP is found iteratively, refining the value of the initial condition in order to satisfy the one at the distal end. The application of this method at each loading step makes it particularly well suited to the quasi-static simulation of continuum and soft robots [27].

**Remark 4.** *It should be noted that Equation (2.12) necessitates the inversion of the stiffness matrix $\mathcal{H}$. Consequently, if used to model a deformable body with very low stiffness, this matrix will drastically become non-invertible and the approach fails.* ∎

### Dynamics Modelling

Following this success in the field of statics, the shooting methods was extended to the dynamics modelling of slender bodies [39]. This approach is equivalent to solving the following initial value problem (IVP):

$$\begin{cases} g' &= g\widehat{\xi}, \\ \eta' &= -ad_{\xi}\eta + \dot{\xi}, \\ \dot{\eta}' &= -ad_{\dot{\xi}}\eta - ad_{\xi}\dot{\eta} + \ddot{\xi}, \\ \Lambda' &= \mathrm{ad}_{\xi}^{T}\Lambda + \mathcal{M}\dot{\eta} - \mathrm{ad}_{\eta}^{T}\mathcal{M}\eta - \overline{F}_{\mathrm{ext}}, \end{cases} \tag{2.22}$$

which is initialized with the dynamics state $(g_0, \eta_0, \dot{\eta}_0, \Lambda_0)$ of the proximal cross-section. Inspired by the work of the ocean engineering community on towed submarine cables, at each time step of a simulation, the time derivatives of the kinematic fields are removed using an implicit integration scheme. This corresponds to obtain $\xi$ from (2.12) while $\dot{\xi}$ and $\ddot{\xi}$ are obtained from the constraints of the implicit scheme [40]. The BVP-IVP in space-time then changes to a simple BVP in space, which can be processed at each of these time steps by the shooting algorithm, in a similar way to the statics case, solving the rod dynamics in its "strong-form". Recently, it has been shown in [40] that: *i*) the forward dynamics BVP on which this approach is based derives from a singular optimal control problem, regularised by the implicit time integration scheme; *ii*) this regularisation works less and less well as the time step and/or beam stiffness decreases; *iii*) as a consequence, the approach fails in many practical circumstances encountered in soft robotics. More in details, the shooting method is much more difficult to apply in dynamics than in statics and requires good initial guesses to converge. Otherwise, the IVPs defined by (2.22) can blow up before reaching the distal end. To this aim, one can use the "multiple" or "modified shooting method" of [41]. Nevertheless, decreasing the time-step $\Delta t$ tends to degrade the numerical convergence of the iterative solver (Levenberg-Marquardt algorithm or Newton-Raphson). It has been proven in [40] that there always exists a critical time-step $\Delta t_c$ below which the method fails, for which an empirical law was identified with the form:

$$\Delta t_c \propto \psi\, l^2 \sqrt{\frac{\rho A}{EI}}, \tag{2.23}$$

where $\psi$ is a dimensionless pre-factor which seems to depend on other physical factors as the conditions of the test (imposed forces and motions), and also non-physical ones as the adopted numerical solver, and the machine accuracy, *e.g.*, $\psi = \frac{1}{100}$ in [40].

**Remark 5.** *Note that the Shooting method works on the $\mathcal{C}_2$ configuration space of the rod (2.3). It does not discretize the rod (except for the discretization required by the numeric integration) thus solving the rod dynamics in its "strong-form". The numerical integration*

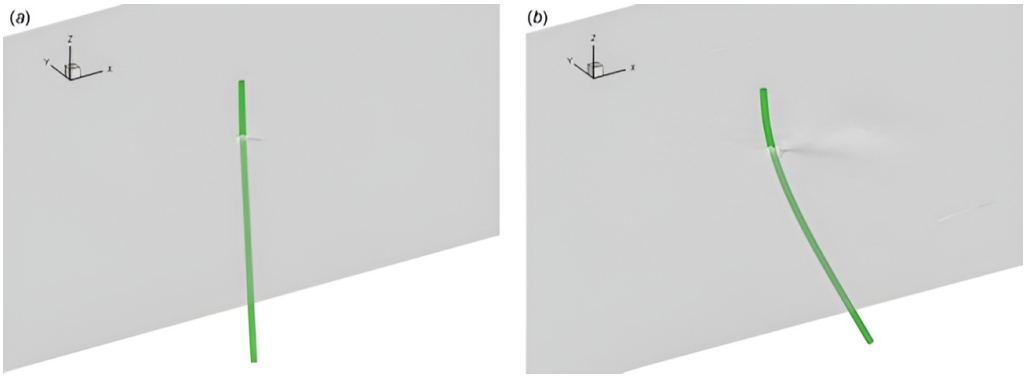



Figure 2.2: A cable fixed on its proximal end in a fluid with constant velocity: (a) initial configuration and (b) deformed cable at after 15 seconds [42].

*reconstructs the shape of the rod, as well as the field of internal stress-wrenches, while the configuration of the first cross section is imposed by design or by the iterative process.* ∎

### 2.2.3 Conclusions on Cosserat rod Direct Resolution

The shooting method represents one of the most frequently employed and well-understood techniques for analysing the statics of a Cosserat rod. Its success can be attributed to its intuitive nature and straightforward implementation. Nevertheless, this method, developed specifically for modelling slender deformable bodies, is limited by their flexibility. In particular, as the flexibility of the deformable body increases, the method becomes more numerically unstable. Moreover, despite its extensive use in the field of statics, its extension to dynamics modelling has proven to be a challenging and unstable process, as it is based on a singular optimal control problem [40].

These issues provides the motivation for further research into alternative approaches to obtain a dynamics model of a Cosserat rod with a reduction of its configuration space.

## 2.3 Model-Reduction Techniques

This section outlines a number of techniques for reducing the dimension of the configuration space of a Cosserat rod. In the first instance, we present the Geometrically-Exact Finite Element Modelling (GE-FEM), which directly discretises the weak form of the rod dynamics. Then, a different discretization, named Discrete Elastic Rods (DER), is presented. Subsequently, alternative methodologies are explored, with a particular focus on the discretisation of the field of strain. These include the Piecewise Constant Curvature (PCC), the Piecewise Constant Strain (PCS), the Piecewise Linear Strain (PLS) and the Variable Strain Approach (VSA).

### 2.3.1 Geometrically-Exact Finite Element Modelling

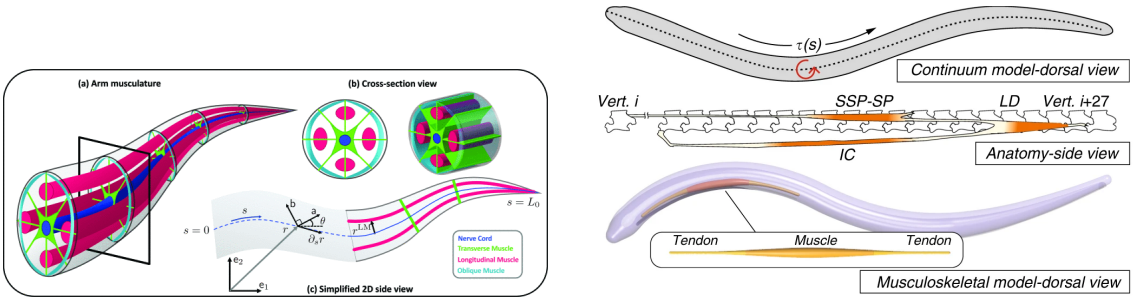
The Geometrically-Exact Finite Element Modelling (GE-FEM) approach has been introduced by J.C. Simo [35, 43, 44, 45, 46, 47]. In a single sentence it is an extension of the FEM method from classical to Cosserat media, *i.e.,* continuums constituted of particles that are not points in $\mathbb{R}^3$ but small rigid micro-structures or micro-solids to which we can attach material vectors named directors. In the case of rods, the micro-structures are their cross-sections with their orthonormed frame $\mathcal{F}(X)$ (Section 2.1.2). In works of Simo, the approach is applied to the full-Cosserat model of Reissner (with stretch and shear), and the natural configuration space of cross-sections is $SO(3)\times\mathbb{R}^3$. Based on this preliminaries, Simo extended all steps of the FEM, from a classical deformable system with configurations in $\mathbb{R}^3$ to Cosserat ones with configurations in the non-commutative Lie group $SO(3)$. The approach is said geometrically exact [44], since it exploits the geometry of the Lie group (a curved non-linear manifold) without resorting to any of the approximations of rotations used in the standard beam FEM elements, except the unavoidable ones required by the discretization in space (decomposition of the domain in finite elements), and in time (finite difference schemes of the time integration).

#### Application

Today, GE-FEM is at the foundation of the most advanced simulation software of complex deformable mechanism in finite deformations, as the multi body dynamic software MECANO, developed by the group of Leuven [48, 49, 50]. It has been used for large deployable space structures, submarine cables, large manipulators or other complex mechanical systems as gearboxes subject to vibrations. More recently is has been extended to sliding rods as those used to model hot extrusion, paper sheet winding or pulley and belt systems.

#### Description

As any FEM, the approach starts by the derivation of the weak-form of virtual works, obtained by the first variation of the action (the integral of the Lagrangian in Hamilton's principle), now performed in $\mathbb{R}^3 \times SO(3)$. In a second step, another variation (or second variation), is applied to this weak form, to build the tangent weak form. As claimed before, all the calculations required by these two variations are performed in an exact manner using differential geometry on $SO(3)$. Due to the non-commutative structure of $SO(3)$, these two variations can be performed on the left (spatial variations) or the right (material variations). If the two are equivalent, the second is much more better in terms of complexity (and numerical efficiency) since it exploits the inertial symmetry of solid mechanics (left invariance). Once the weak form and its linearization are performed, they are discretized both in space (with finite elements) and in time (with an implicit Newmark integrator extended to $SO(3)\times\mathbb{R}^3$). The approach, have been proved to be much more efficient than the usual FEM beam models both in terms of accuracy and computational efficiency. After the works of Simo, the GE-FEM has been refined and pushed forward in many directions [49, 51, 52, 53, 50]. Note that all these further works use the Reissner rod




(a) (b)

Figure 2.3: Different application of the DER approach to model (a) an octopus arm [54] and (b) a snake [55].

model of Simo, except in [53, 42], where the approach has been applied to Kirchhoff rods (Figure 2.2). In this later case, $SO(3)$ has been replaced by its fibration $S^2 \times SO(2)$ to tackle the no-shear constraints.

#### Advantages and Disadvantages

In spite of its successes, the GE-FEM has never been applied to continuum and soft robotics except as a reference to validate the strain-based parametrization that will be introduced later in this chapter. As explained in [21], this is probably due to two reasons. The first reason is that is conceptually and technically difficult to appropriate for other communities than structural dynamics. Moreover, even in the community of FEM, it is not canonical, since it needs to move from the usual vector calculus on $\mathbb{R}^3$ to the differential geometry of a non commutative Lie group, with many tedious implications. The second reason can be explained by the fact that the GE-FEM is based on an absolute parametrization of the rod configurations (as this is the case of the standard FEM in $\mathbb{R}^3$), which is not well suited to robotics uses and needs, especially for modelling the "intrinsic" distributed actuation technologies developed by robotics (e.g. based on tendons).

### 2.3.2 Discrete Elastic Rods

In the field of computer graphics, a method known as Discrete Elastic Rod (DER) has been developed as an efficient Lagrangian alternative to (GE)-FEM for the rapid interactive simulation of hair and other filament-like structures [56, 57], or otherwise used when computational speed is the predominant matter.

#### Applications

The DER approach has been used in numerous applications, including: the modelling of a soft locomotor [58], a limbed locomotor [59] and a frog-like soft robot [15]. The DER has recently been extended to Cosserat rods [60], which have been used to model an octopus arm [54] and a snake-like locomotor [55] (see Figures 2.3a and 2.3b, respectively). This

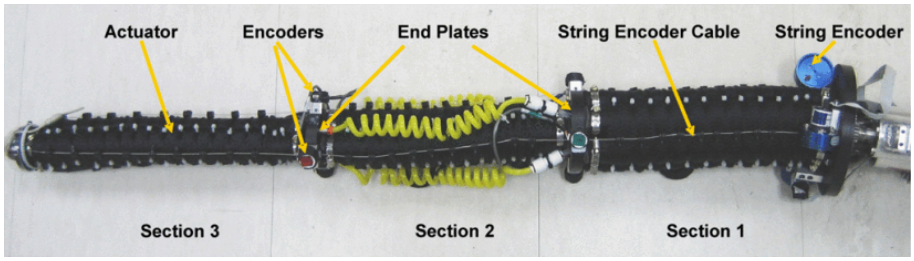



Figure 2.4: The OCTARM (ver. 5.2) continuum robot [66].

approach has been implemented in PyElastica, a free and open-source software package for the simulation of slender body assemblies in Python [61, 62]. To date, PyElastica represents the most complete implementation of the DER found in the literature.

#### Description

In DER, the discretization (reduction) process is applied from the outset to the rod itself, which is transformed into a discrete rod, *i.e.,* a set of edges separated by vertices [63]. By redefining all the geometric objects on this discrete manifold, and introducing them into the Lagrange equations, the reduced dynamics is obtained in form that resembles that provided by finite differences [64]. However, in contrast to usual finite differences, the approach preserves exactly all the intrinsic properties of the original continuous formulation [63]. First developed for Kirchhoff rods [64, 56], it was extended to handle connections to rigid bodies by the fast projection method (FPM) of [65].

#### Advantages and Disadvantages

The DER approach is computationally efficient and can model the three-dimensional dynamics of elastic and viscoelastic rods that are capable of bending, stretching, and twisting. Beyond these considerations, it is noteworthy that in all the references to DER that we have examined, including the few that have recently focused on robotics, the approach appears to have been applied to robots on an ad-hoc basis rather than in a more generic manner [58, 59, 15, 62]. Furthermore, even the most generic implementations of the approach [62] shows instabilities, as the approaches [39, 40]. To elaborate, an augmentation in the rigidity of the rod (Young's modulus and/or shear modulus) necessitates a substantial reduction in the time step $\Delta t$.Furthermore, as indicated in [60], in this case, the space step $\Delta X$ must verify the empirical relationship $\Delta X \approx 100\Delta t$ to avoid instability. In light of the aforementioned observations, it can be concluded that the DER is well suited to (very) soft robots, but less so to robots made of metals, such as Nitinol or spring steel, which is the case of CPRs.

### 2.3.3 Piecewise Constant Curvature

The Piecewise Constant Curvature (PCC) approach represents one of the earliest models for soft robots [67] and one of the most widely adopted models in this field [68]. The

simplicity and precision of its dynamic model in Lagrangian form are the primary sources of its fame.

#### Applications

The PCC approach is employed when the rapid resolution of the kinematic partial differential equations (PDEs) is necessary and when the soft robot can be regarded as a series of circular arcs. The PCC has been employed in the dynamics modelling of a variety of systems, including: an elephant trunk [69], the OCTARM of the Mechatronics Laboratory at Clemson University [70, 66] (see Figure 2.4), a continuum arm powered by pneumatic muscle actuators [71] and, finally, the Festo Bionic Handling Assistant [72].

#### Description

In this model, the slender deformable body (or the continuum manipulator) is represented as a sequence of circular arcs [67], the lengths of which can be extended or contracted. Each arc has constant curvature and is free from torsion and shear. In light of these assumptions, the configuration variables of the manipulator include the curvature and extension of all its sections. The Lagrangian form of the dynamics model is derived by combining the robot's kinetic and potential energies, which are characterised by a bending and an extension spring [66]. The assumption of constant curvature allows for the analytical solution of the Lagrangian dynamics model, which can be expressed in terms of the configuration variables.

#### Advantages and Disadvantages

This approach employs a method of serial sectioning, wherein a soft robot is divided into a series of sections with constant curvatures. The PCC enables the analytical reconstruction of the Lagrangian model of the system, thereby facilitating the control of the physical robot. Due to its inherent simplicity, the approach was employed in the development and dynamics modelling of numerous prototypes [70, 66, 71, 72]. Nevertheless, the assumption of constant curvatures restricts the scope of applicability of this method. Furthermore, this formulation is subject to numerous discontinuities and singularities. Despite the existence of an alternative state representation detailed in [73], the aforementioned drawbacks restrict the scope of applicability of the PCC approach.

### 2.3.4 Piecewise Constant Strain

The Piecewise Constant Strain (PCS) model, which can be seen as an extension of the PCC model, as it accounts for shear and torsional deformations, both of which are crucial for the analysis of out-of-plane external loads. In addition, this methodology permits the extension of the recursive composite-rigid-body and articulated-body algorithms to soft manipulators.

Figure 2.5: A multi-bending soft robot arm driven by cables [28].

### Applications

This approach is employed when the deformable body can be divided in different sections, for which torsion and axial-shear must be taken into account (*e.g.,* in the presence of an external force applied at an angle to the plane of analysis). The approach was tested on a number of prototypes, including: one fluidic actuated leg of the soft crawler FASTT [74], a multi-bending soft robot arm driven by cables [28, 75] (see Figure 2.5) and a Concentric Tube Robot (CTR) [76].

### Description

In this model, the soft robot is divided into multiple sections, each characterized by a constant strain field [75]. The collection of these strain fields defines the configuration variables of the manipulator. With a constant field of strain, it is possible to solve the PDEs of Equation (2.9) analytically [75]. The kinematics of each section is dependent on that of the preceding one, resulting in a recursive formulation for the overall manipulator. Subsequently, the dynamics state of the deformable body is obtained by a backward recursion on the sections. These two passages reminds the ones of the RNEA of rigid robotics. Finally, the Lagrangian dynamics model of the robot can be expressed in terms of the configuration variables, which is ideal for modelling and control.

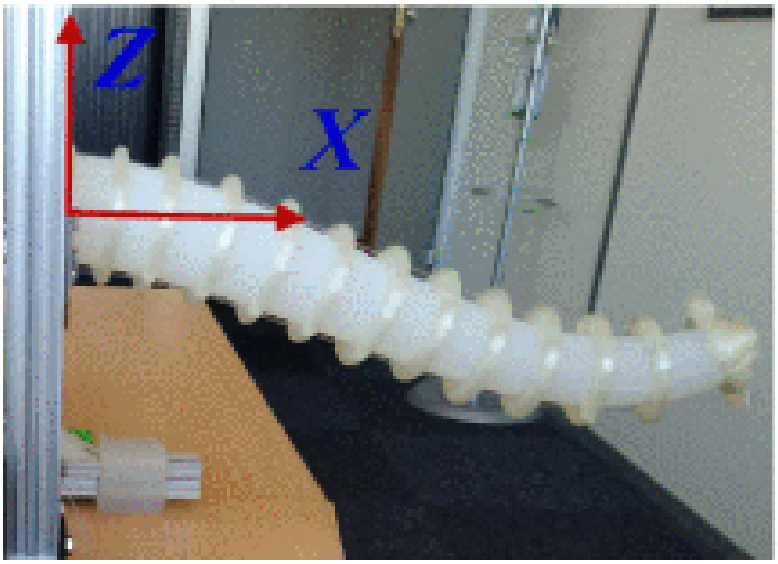



Figure 2.6: Soft robot actuated by cables [77].

#### Advantages and Disadvantages

The discretization of the strain field enables a recursive and analytical computation of the manipulator's kinematics and dynamics. This enables the development of an efficient Lagrangian dynamics model, which can be used for the control of soft manipulators [74, 75, 76]. Furthermore, the parametrization of discrete strain fields represents the continuous counterpart of the joint variables parametrization observed in the context of multi-body robotics arms. Consequently, it can be integrated with efficient RNEA that are typically employed in rigid robotics. Nevertheless, the assumption of piecewise constant strain represents a limitation of this strategy.

### 2.3.5 Piecewise Linear Strain

This section presents a concise overview of the Piecewise Linear Strain (PLS) methodology [77], which may be viewed as an extension of the (PCS) and as a means of applying the FEM to the rod field of strain.

#### Applications

The PLS is used when the assumption of PCS needs to be relaxed while an analytical formulation of the Lagrangian dynamics model is still required [77]. This approach was employed to identify the material parameters of a soft manipulator (see Figure 2.6) to then evaluate the performances of the approach.

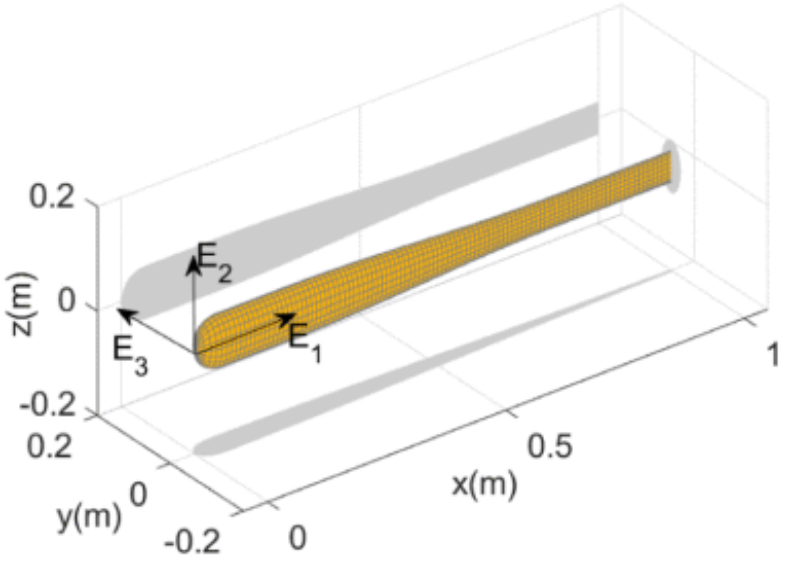



Figure 2.7: A bio-inspired locomotor (swimmer) mimicking a fish [40].

#### Description

As with the preceding models, the soft manipulator is divided into discrete sections. In this instance, however, the strain is not assumed to be constant, but rather to vary linearly along each section, as a linear interpolation of the strain assigned at both ends. With the strain so defined, the kinematics of the manipulator can be obtained in a sequential manner for each of its sections. As in the PCS, the PDEs of Equation (2.9) can be solved analytically [77].

#### Advantages and Disadvantages

The PLS approach represents an extension to the PCS, employing a finite element modelling approach on the rod field of strains. Similarly to the PCC and PCS, this approach requires the rod to be segmented. In contrast to the aforementioned approaches, the PLS does not assume a constant strain; rather, it allows for a linearly varying strain along each section. The extension of the PCS preserves an analytical solution for the deformable body dynamics, which in turn allows for the definition of their Lagrangian dynamics model. Nevertheless, this methodology is constrained by the limitations imposed on the rod strain field.

### 2.3.6 (Variable) Strain Parameterisation Approach

Finally, we present the strain-based parameterisation approach, or Variable Strain Approach (VSA) of [21], which serves as the fundamental methodology for this thesis study. This approach represents an extension of the PCS, whereby the assumption of constant strain field is no longer made. In the literature, the VSA has been used on soft or contin-

uum robots, including: a three-dimensional swimming eel-like robot [31] (Figure 2.7), soft robots as core of the SoroSim toolbox [78], for a flying beam [21] and other applications related to CPRs, which will be discussed in Section 2.4.

The following sections present the applications of this approach to reduce the configuration space of a Cosserat rod to a finite dimension vector space. Furthermore, we will apply the principle of virtual works to derivate an appropriate "weak-form" of the rod dynamics model.

### Reduction of the Configuration Space

First, we recall from [21] that we can select some among the six deformations: torsion, bending, stretching and shear, which we denote as the Degrees of Freedom (DoFs) of the rod. The selected deformations will be referred as "allowed" while the remaining as "constrained", for which we denote their number with $n_a$ and $\overline{n}_a = 6 - n_a$, respectively. Then, to model these different possibilities, we introduce two matrices: $A \in \mathbb{R}^{6\times n_a}$ and its complementary $\overline{A} \in \mathbb{R}^{\overline{n}_a\times 6}$ which define the allowed and the constrained DoFs, respectively. Both matrices are constructed from the identity matrix $\mathbb{1}_6$, for which we assign a DoF for every row and column, starting with the rotations followed by the translations, both ordered as $x$-$y$-$z$. Then, $A$ is constructed by selecting the columns of $\mathbb{1}_6$ corresponding to the allowed DoFs, while $\overline{A}$ is constructed by selecting the rows of $\mathbb{1}_6$ corresponding to the constrained DoFs. These two selection matrices have the following properties: $\overline{A}A = \mathbb{0}_{\overline{n}_a\times n_a}$, $A^TA = \mathbb{1}_{n_a}$ while $\overline{A}\overline{A}^T = \mathbb{1}_{\overline{n}_a}$. With these matrices, we can decompose the field of strain as follows:

$$\xi = A\xi_a + \overline{A}\xi_c, \tag{2.24}$$

where $\xi_a$ is the subset of allowed components of $\xi$ while $\xi_c$ the constrained ones. We give here an example for a Kirchhoff rod which deforms only due to bending and torsion:

$$\xi_a = \begin{bmatrix} K_x \\ K_y \\ K_z \end{bmatrix}, \quad \xi_c = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \overline{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.25}$$

Note that, in Equation (2.24), $\overline{A}\xi_c$ corresponds to $\xi^0$ and will be referred as such in the following. With this decomposition of the strain field, the shape space of the rod (2.3) is reduced to:

$$\mathbb{S} = \xi_a : X \in [0, \ell] \mapsto \xi_a(X) \in \mathbb{R}^{n_a}. \tag{2.26}$$

To reduce this functional space into a finite dimension vector space, we discretize each of the $n_a$ elements of $\xi_a$, denoted $\xi_{a,i}$ with $i = 1, \ldots, n_a$, using a basis and the corresponding

set of coefficients:

$$\xi_{a,i} = \begin{bmatrix} \phi_{i,1} & \phi_{i,2} & \dots & \phi_{i,n_{e,i}} \end{bmatrix} \begin{bmatrix} q_{i,1} \\ q_{i,2} \\ \vdots \\ q_{i,n_{e,i}} \end{bmatrix} = \phi_i q_i, \tag{2.27}$$

where $n_{e,i}$ is the number of modes used for the $i^{th}$ allowed deformation, ${\phi_i}^T \in \mathbb{R}^{n_{e,i}}$ a (polynomial) basis and $q_i \in \mathbb{R}^{n_{e,i}}$ its coefficients. We can repeat Equation (2.27) for every allowed deformation, and, by stacking all the required operation in a matrix form, we obtain:

$$\xi_a = \Phi q, \tag{2.28}$$

where $\Phi(X) \in \mathbb{R}^{n_e \times n_e}$ is a diagonal-block matrix containing all the $\phi_i$, with $n_e = \sum_{i=1}^{n_a} n_{e,i}$, and $q(t) \in \mathbb{R}^{n_e}$ is a stack of all their coefficients:

$$\Phi = \begin{bmatrix} \phi_1 & 0 & \dots & 0 \\ 0 & \phi_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \phi_{n_a} \end{bmatrix}, \qquad q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{n_a} \end{bmatrix}. \tag{2.29}$$

As a result, we reduced the configuration space of the rod to a finite dimension:

$$\mathcal{C}_3 = SE(3) \times \mathbb{R}^{n_e}, \tag{2.30}$$

which defines the configuration space of this approach. Finally, using (2.28) into (2.24) together with the definition of $\xi^0$ and differentiating with respect to time, we obtain:

$$\xi = A\Phi q + \xi^0, \qquad \dot{\xi} = A\Phi \dot{q}, \qquad \ddot{\xi} = A\Phi \ddot{q}, \tag{2.31}$$

### Formulation for a Distributed Actuation

We recall $\Lambda_{act}(X,t) \in se^*(3) \cong \mathbb{R}^6$ as the stress-wrench that models the effect of a distributed actuation (if any) across the $X$-cross section of the rod. Its formulation depends on how the distributed actuation interacts with the rod body. In the following, we give its definition for a rod actuated via a set of $n_t$ tendons of index ($\alpha = 1, 2, \dots, n_t$), with routings defined by their positions $D_\alpha(X) \in \mathbb{R}^3$ in the $X$-cross-sectional frames [21], $\Lambda_{act}$ can be detailed as:

$$\Lambda_{act} = \sum_{\alpha=1}^{n_t} \frac{1}{\|\Gamma_\alpha\|} \begin{pmatrix} D_\alpha \times \Gamma_\alpha \\ \Gamma_\alpha \end{pmatrix} T_\alpha(t), \tag{2.32}$$

where $T_\alpha$ is the tension exerted on the tendon $\alpha$, and $\Gamma_\alpha = \Gamma + K \times D_\alpha + D'_\alpha$, with $D'_\alpha$ the space-rate of variation of $D_\alpha$.

**Remark 6.** *In general, the model of $\Lambda_{act}$ depends on $\xi$ and needs to be derived on a case by case basis. It should be noted that alternative forms of distributed actuation are employed in the literature, including those utilising pneumatic chambers [79, 80], which will not be discussed in detail here. Nevertheless, provided that the forces exerted by the actuation along the rod body are known, a relationship analogous to that expressed in Equation (2.32) can be employed to map these effects within the stress-wrench field.* ∎

### Weak-Form of the Rod Dynamics

We now apply the principle of virtual works to the stress-wrench of the rod cross section:

$$\mathcal{W}_{int} = \int_0^\ell \delta\epsilon^T \Lambda \, dX, \tag{2.33}$$

where $\mathcal{W}_{int}$ is the virtual work of the internal stress-wrench with a variation of the strain denoted with $\delta\epsilon$. Now using the active constitutive law (2.13) into (2.33) we obtain the following:

$$\int_0^\ell \delta\epsilon^T \Lambda \, dX = \int_0^\ell \delta\epsilon^T \Lambda_{act} + \delta\epsilon^T \mathcal{H} \delta\epsilon^T + \delta\epsilon^T \mathcal{D}\dot{\epsilon} \, dX. \tag{2.34}$$

which can be made explicit with respect to the elastic coordinates by applying (2.31), which yields:

$$\begin{aligned} &\delta q^T \int_0^\ell \Phi^T A^T \Lambda \, dX = \\ &\delta q^T \int_0^\ell \Phi^T A^T \Lambda_{act} + \Phi^T A^T \mathcal{H} A \Phi q + \Phi^T A^T \mathcal{D} A \Phi \dot{q} \, dX. \end{aligned} \tag{2.35}$$

Being valid for all $\delta\epsilon$, and consequently all $\delta q$, we remove it from (2.35) obtaining, for the left hand side:

$$\int_0^\ell \Phi^T A^T \Lambda \, dX = Q_a, \tag{2.36}$$

where $Q_a$ denotes the projection of $\Lambda$ into a set of generalized forces. On the other hand, the right hand side of (2.35) has the following composition:

$$\int_0^\ell \Phi^T A^T \Lambda_{act} \, dX = Q_{act} \tag{2.37}$$

$$\int_0^\ell \Phi^T A^T \mathcal{H} A \Phi \, dX q = \mathcal{K}_{\epsilon\epsilon} q = Q_e \tag{2.38}$$

$$\int_0^\ell \Phi^T A^T \mathcal{D} A \Phi \, dX \dot{q} = \mathcal{D}_{\epsilon\epsilon} \dot{q} = C_e \tag{2.39}$$

where $Q_{act}$ is the set of generalized forces imposed by the actuation, $\mathcal{K}_{\epsilon\epsilon}$ is the generalized stiffness matrix, $Q_e$ is the set of generalized elastic forces, $\mathcal{D}_{\epsilon\epsilon}$ is the generalized damping matrix and $C_e$ is the set of generalized damping forces. Finally, using (2.36–2.39) into (2.34), gives the weak-form of the Cosserat rod dynamics:

$$Q_a = Q_{act} + Q_e + C_e \tag{2.40}$$

### Lagrangian Dynamics Model

By reminding that $\Lambda$, defined in (2.20), accounts for the inertia, centrifugal, Coriolis and configuration-dependent forces, the corresponding generalized forces can be rewritten as:

$$Q_a = M(q)\ddot{q} + Q_v(\dot{q}, q) + Q_c(q) \tag{2.41}$$

where $M \in \mathbb{R}^{n_e \times n_e}$ is the matrix of generalized inertia of the Cosserat rod, $Q_v \in \mathbb{R}^{n_e}$ the vector of Coriolis and centrifugal generalized forces and $Q_c \in \mathbb{R}^{n_e}$ the configuration-dependent forces. Using (2.41) and (2.38–2.39) in (2.40) gives the dynamics model of a Cosserat rod, parametrize with this variable strain approach, in its Lagrangian form:

$$Q_{act} = M(q)\ddot{q} + Q_v(\dot{q}, q) + Q_c(q) - \mathcal{K}_{\epsilon\epsilon} q - \mathcal{D}_{\epsilon\epsilon}\dot{q} \tag{2.42}$$

**Practical Implementation**

Once the methodology has been fully outlined, a concise overview on the practical implementation of the approach is provided. First $(\xi, \dot{\xi}, \ddot{\xi})$ is computed from $(q, \dot{q}, \ddot{q})$ (2.31), which then allows to reconstruct the kinematics of the Cosserat rod via a forward numerical integration (*i.e.,* from $X = 0$ to $X = \ell$) of the PDEs (2.9). Then, with the kinematics so defined, the field of internal stress-wrench is reconstructed by backward numerical integration (*i.e.,* from $X = \ell$ to $X = 0$) of (2.20), together with its projection in the generalized coordinates space (2.36), which, in this case, takes the form:

$$Q_a = -\int_{\ell}^{0} \Phi^T A^T \Lambda \, dX. \tag{2.43}$$

The two aforementioned numerical integrations define, respectively, the forward kinematics and backward dynamics of the RNEA, here applied to the continuum of a Cosserat rod. Once the dynamics state of the deformable body has been obtained, the equilibrium is evaluated using (2.40).

**Advantages and Disadvantages**

This approach assumes that the strain field varies along the arc length in some or all of its components (e.g., planar, Kirchhoff, or Reissner rod). The strain vector field can be decomposed using a basis and a set of generalized strain coordinates, reducing the rod's configuration space to $\mathcal{C}_3$. This transforms the Cosserat BVP into the "weak form" of its dynamics, which can be solved in two passes, similar to the standard Newton-Euler algorithms used in rigid robotics. However, the PDEs of Cosserat kinematics and dynamics (Equations (2.9) and (2.20)) lack an analytical solution. Consequently, numerical routines have been developed to address this issue, which, however, limit the performances of the approach, increasing the required computational time.

### 2.3.7 Conclusions on Model Reduction Techniques

The literature review on resolution of the Cosserat rod model addressed a range of approaches based on model reduction techniques, including those based on GE-FEM, DER, PCC, PCS, PLS, and VSA. In this context, GE-FEM has been the foundation of some of the most advanced simulation software for mechanisms with finite deformations. However, its conceptual complexity presents a significant learning curve, and its absolute parametrization is not ideal for the specific needs of continuum robotics. The other finite element approach of the DER allowed for a fast numerical resolution, although it was not prone

to the dynamics modelling of CPRs due to the instabilities that occur in the simulation of rather stiff slender bodies such as steel rods. Concerning to the strain-parametrization approaches, the PCC model offered an analytical Lagrangian dynamics model, with great advantages in terms of computational time, but remained limited by its assumption of constant curvature. The PCS and PLS relax the assumptions of the PCC, but still limit the range of variation of the strain field and require a slender body (or soft robot) to be sectioned multiple times. Finally, the VSA approach assumes the strain to be arbitrary variable and discretised on a (polynomial) base. This approach enables the computation of a Lagrangian dynamics model, though not with an analytical solution. However, it provides a dynamics model for slender deformable bodies using a continuous counterpart of the standard RNEA used in rigid robotics. In conclusion, the VSA was selected as foundation for this thesis, as the more comprehensive and adaptable approach among the alternatives.

## 2.4 Modelling of Continuum Parallel Robots

After having presented different modelling approaches for the dynamics of slender deformable bodies, in this section we provide a concise overview of the current state of the art in modelling CPRs.

### 2.4.1 Applications

The modelling of CPR has been a prominent topic of research over the past decade. A comprehensive overview of this subject has recently been published [20]. In this section, we present an account of some of the contributions in the statics and dynamics modelling of CPRs.

#### Statics

The statics modelling of CPR has been the subject of extensive research and analysis. The majority of authors employ the shooting method [38] (Section 2.2) for various applications, including the teleoperation of a Stewart-Gough CPR device [12] and a study on the effects of design and configuration on the manipulability and force-sensing potential of a CPR [81]. Other authors have employed the VSA approach for stability analysis [82], workspace analysis [83, 84] and experimental validation on a planar CPR prototype [85].

#### Dynamics

In contrast to statics modelling, the dynamics modelling of CPRs has been comparatively under explored. In this context, an extension of the Shooting method was applied to investigate the dynamics transitions of a Stewart-Gough-like CPR as it traversed unstable configurations [39].

### 2.4.2 Description

To create a statics or dynamics model for a CPR, it is essential to first describe each of its individual components. The base joints establish a connection between the CPR base and the proximal cross-section of the deformable bodies. These joints are typically modelled using standard rigid robotics approaches [86, 87]. In order to model the deformable bodies of the CPR limbs, it is possible to employ any of the previously discussed approaches outlined in Sections 2.2 and 2.3. Regardless of the selected methodology, once the deformable body has been reconstructed, their configuration is evaluated to ensure consistency with the CPR structure, defining a BVP. In particular, it is essential to guarantee equilibrium between the bodies and to ensure that the final configuration is in accordance with the architectural design of the CPR. In accordance with the standard procedure for any BVP, if the specified constraints are not met, the configuration variables of the CPR are adjusted iteratively until a configuration that satisfies the desired solution is achieved. Moreover, as represented in Figures (1.3–1.4) CPRs present different types of actuation which can be categorized in two groups: localized actuations typically at the CPR base consisting of rigid joint (prismatic 1.3b, revolute or changing the rod length 1.3a) and actuation mechanisms distributed along the rods (such as tendon driven actuation 1.3b and pneumatic 1.4b). These different actuations systems introduce further challenges to the development of a generic model for CPRs.

### 2.4.3 Conclusions on CPRs Modelling

The statics modelling of CPRs has gained increasing interest over the past decade, while, by contrast, their dynamics modelling remains largely unexplored. In both cases, CPRs modelling requires the combination of rigid bodies and joints modelling, the geometric constraints (imposed by the parallel architecture) and the modelling of deformable bodies. For the latter, the shooting method is the dominant approach in statics, but it is not a viable solution for dynamics modelling due to its dependence on a singular optimal control problem. Alternative methods, such as the Variable Strain Approach (VSA), offer more suitable solutions for the dynamics modelling of deformable bodies.

## 2.5 General Conclusions

In this chapter, we began by presenting the Cosserat rod theory, which allows to obtain the set of PDEs describing the kinematics and dynamics of a slender deformable body. These can be solved directly with the shooting method, *i.e.,* solving the strong-form of the rod dynamics. While promising results were obtained in statics, the model was proven to be unstable in dynamics. This motivated the investigation of alternative methodologies, based on the reduction of the configuration space of the rod to a finite dimension vector space, *i.e.,* solving the weak-form of the rod dynamics. After discussing the relevant approaches to modelling deformable bodies and outlining the advantages of the Variable Strain Approach (VSA), we proceeded to provide a comprehensive overview of the modelling of CPRs, which are composed by these deformable bodies. In this context, we provided a

concise account of the shooting method for statics modelling, along with a discussion of the few applications of the VSA. For the latter, despite the high computational time, the results in terms of stability, workspace analysis and robustness were promising. In terms of dynamics, there is a scarcity of available literature on the subject. Furthermore, the applications that do exist are based on the shooting method and are therefore prone to instability. Conversely, the VSA, which demonstrated considerable proficiency in the dynamics modelling of deformable bodies, may be an effective option for the dynamics modelling of CPRs, which provides the foundation for the present study.

# Chapter 3

# Dynamics Modelling of Hybrid Structures

In this chapter, we address the dynamics modelling and simulation of hybrid systems composed of Cosserat rods and rigid bodies in arbitrary topologies with the VSA, as in [78]. However, unlike all previous work based on this approach, the forward dynamics of soft systems are here integrated in time using an implicit "unconditionally" stable scheme. This choice strongly structures the simulation algorithm as it requires not only to compute the dynamics model of the robot, but also its tangent dynamics. The first is required to calculate the residual vector of the original DAEs constrained by the integration scheme, and the second to calculate the Jacobian of this vector. This extra effort, compared to explicit integration, has been accomplished by GE-FEM [35] and DER [56]. In the case of strain-parametrized models here considered, this extension is particularly challenging. Indeed, the use of relative variables (strains) instead of the absolute variables of GE-FEM or DER, increases the non-linearities of the model, which are then transferred from the stiffness matrix to the mass matrix.

To address this issue, the chapter refers to rigid robots dynamics, for which efficient algorithms working in relative (joint) coordinates have been developed since the 1980s in the well-known Newton-Euler framework [86]. To this end, the $IDM$ of [21] will be hybridized with that of rigid multibody-systems [88] in a single unified algorithm that generalizes the Lie group recursive formulation of rigid robot dynamics [89, 90]. Going one step further, applying an exact differentiation to this $IDM$ considered as an input-output map, provides a second algorithm named $TIDM$ for "Tangent Inverse Dynamic Model". By feeding these two new algorithms with inputs compatible with the implicit time integration scheme, in this case a one-step integrator of the Newmark class, both the residual vector and its Jacobian matrix can be computed in the Newton correction loop of a general predictor-corrector algorithm. Like in the GE-FEM of [49] and the FPM of [65], the case of closed loops is treated in its index-3 formulation[1], which, compared to Baumgart's

[1]The index of a system of DAEs is roughly defined as the number of times we need to differentiate the constraints in order to change the DAEs into ODEs (ordinary differential equations). For multi-body

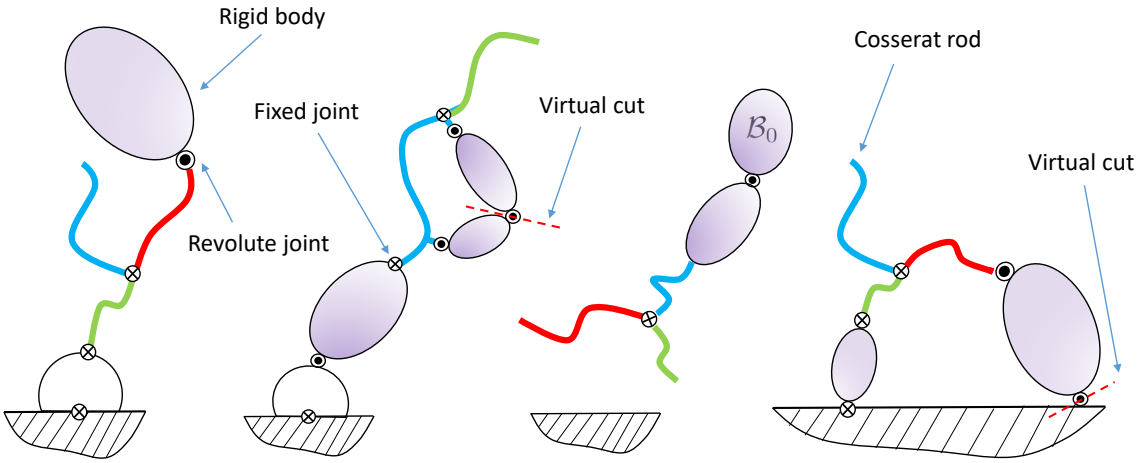



Figure 3.1: Different types of SMMS discussed in the chapter. The ellipsoids and thick lines represent rigid bodies and Cosserat rods, respectively. From left to right: (a) Tree-shaped manipulator. (b) Manipulator with a closed loop. (c) Locomotor in tree form. (d) Manipulator modelled as a locomotor connected to the ground (see Remark 7). The dashed lines represent the location of the virtual cuts.

index-1 method [92], used in [78] after [93], has the advantage of providing solutions that do not depend on the user's choice of some stabilisation parameters, while satisfying the constraints exactly [94]. At the end, the proposed dynamics simulation algorithm generalizes the quasi-static simulation algorithm of [82] and [34], recently proposed for CPRs and TD-CRs respectively. In particular, like in [34], the differential properties of the $IDM$ and $TIDM$ are exploited in order to achieve all the space integrations of generalized forces with spectral methods [95]. The general algorithm of the chapter is illustrated through numerical examples, both inspired from the GE-FEM literature of Cosserat rods and of some recent robotics applications.

## 3.1 Lagrangian model of a SMMS

### 3.1.1 Systems addressed

We consider a soft mobile multibody system (SMMS) (see Figure 3.1). Such a system consists of rigid bodies and Cosserat rods connected by fixed or revolute joints[2] which can be passive or torque actuated. The rods can be passive or internally actuated through distributed actuation systems as cables or pressurized fluids. The system can have a simple open, tree-like or closed topology, with kinematic loops. The system can be a manipulator

systems, differentiating once (resp. twice) the geometric constraints, changes the formulation from an index-3 to an index-2 (resp. index-1) [91].

[2]This is assumed for the sake of simplicity, but the approach can be extended with no difficulty to other joints as prismatic or universal ones.

or a locomotor depending whether it has one of its bodies (named the basis) connected to the ground, or not.

### 3.1.2 Lagrangian parametrization of a SMMS

If the considered SMMS contains some kinematic loops (see Figure 3.1 (b,d)), then we start by virtually cutting[3] each of these loops at the level of a passive joint (that can be fixed)[4]. Then, the configuration of the so defined tree-like SMMS is parametrized as follows. Any localized joint of revolute type is parametrized by one angle, and the set of all these angles is gathered in the $(n_r \times 1)$ vector denoted $q_r$. Regarding the rods, we use the VSA of Section 2.3. Gathering all the strain vectors $q_{\epsilon,j}$ defines an $(n_\epsilon \times 1)$ vector, noted $q_\epsilon$. When the SMMS is a manipulator, its configurations or "shapes", are entirely parametrized by the generalized coordinates vector $q = (q_r^T, q_\epsilon^T)^T$ of size $(n \times 1)$ with $n = n_r + n_\epsilon$. In contrast, if the SMMS is a locomotor, it is not only subject to time-variations of its internal DoFs parametrized by $q$, but also to some net rigid motions. These additional external DoFs require a further set of coordinates that parameterize the pose (position-orientation) of a frame $\mathcal{F}_0$ attached to one of the rigid bodies of the SMMS, that takes the meaning of a reference body $\mathcal{B}_0$. This rigid body can be a full (3D) rigid body, or a rigid cross sections at one of the tips of a Cosserat rod. The pose of this reference rigid body is parametrized by a homogeneous transformation $g_0 \in SE(3)$. Finally, in the most general case, the set of Lagrangian coordinates is defined as a pair $(g_0, q)$ *i.e.,* an element of the configuration space $SE(3) \times \mathbb{S}$ with $\mathbb{S} \cong \mathbb{R}^n$ the shape space of the SMMS. With this definition of the configuration space, the velocities of the SMMS are described at any time $t$, with a vector $(\eta_0^T, \dot{q}^T)^T$ of $\mathbb{R}^{6+n}$.

**Remark 7.** *Note that a same closed-loop SMMS can be modelled by different tree-like systems depending on the cuts chosen. To illustrate this, consider a planar CPR like the one in Figure 3.1 (d), where one of the upper joints is actuated, while all the others are passive. The system can be cut at the upper passive joint (shown in red in Figure 3.1 (d)) and the ground defined as the base $\mathcal{B}_0$ of a two-armed manipulator. Alternatively, all the lower joints can be cut off (shown in blue in Figure 3.1 (d)) and the upper rigid platform defined as the reference body $\mathcal{B}_0$ of a two legged locomotor.* ∎

### 3.1.3 Lagrangian dynamic model of a SMMS

Once the configuration space of a general SMMS is defined, applying one of the variational principles of dynamics as D'Alembert's principle of virtual works, or Hamilton's principle of least action, provides its dynamics as a set of differential-algebraic equations of the form

[3] A virtual cut is a purely mathematical process. Real cuts, such as the detachment of two bodies, are not covered.

[4] Once again, this condition is imposed for the sake of simplicity, but the approach can be extended to virtual cuts applied to active joints.

[49]:

$$\begin{cases} \begin{bmatrix} 0_{6\times 1} \\ Q_{act} \end{bmatrix} & = \begin{bmatrix} \mathcal{M}_{00} & M_{0q} \\ M_{q0} & M_{qq} \end{bmatrix} \begin{bmatrix} \dot{\eta}_0 \\ \ddot{q} \end{bmatrix} + \begin{bmatrix} F \\ Q \end{bmatrix} \\ \Upsilon(g_0, q) & = 0_{m\times 1}. \end{cases} \tag{3.1}$$

The bottom (algebraic) equations define a set of $m$ independent constraints imposed by the closure loops, which in the general case can depend on both $g_0$ and $q$ depending on the topology of the system after cuts and the choice of $\mathcal{B}_0$ (see Remark 7). The time-differential of these geometric constraints defines the kinematic form of constraints:

$$\dot{\Upsilon} = \begin{bmatrix} J_0 & J_q \end{bmatrix} \begin{bmatrix} \eta_0 \\ \dot{q} \end{bmatrix} = 0_{m\times 1}, \tag{3.2}$$

where $J_q(g_0, q)$ and $J_0(g_0, q)$ are some kinematic Jacobian matrices defined by the usual derivation in $\mathbb{R}^n$, for the first:

$$J_q\, \dot{q} = \left( \frac{\partial \Upsilon}{\partial q} \right) \dot{q}, \tag{3.3}$$

and by the directional derivative on $SE(3)$ for the second:

$$J_0\, \eta_0 = \left. \frac{d}{d\epsilon} \right|_{\epsilon=0} \Upsilon(g_0 \exp_{SE(3)}(\epsilon \hat{\eta}_0), q), \tag{3.4}$$

with "$\exp_{SE(3)}$", the exponential map of $SE(3)$ [50]. In the top (differential equations) of (3.1), from left to right, one finds: (1) The vector of generalized internal actuation forces $Q_{act} = (\tau_{act}^T, Q_{act\epsilon}^T)^T$ including the localized joint torques $\tau_{act}(t)$ as well as the distributed actuation generalized forces $Q_{act\epsilon}^T(q_\epsilon, t)$ of a set of cables for instance. (2) The $q$-dependent symmetric matrix of generalized inertia. (3) The vector of generalized accelerations. (4) The vector $(F^T, Q^T)^T = (F_v^T, Q_v^T)^T + (F_c^T, Q_c^T)^T$, with $(F_v^T, Q_v^T)^T(q, \eta_0, \dot{q})$ the velocity-dependent generalized forces including the effects of Coriolis and centrifugal accelerations, and $(F_c^T, Q_c^T)^T$, the vector of configuration-dependent forces. This vector can be detailed as:

$$\begin{bmatrix} F_c \\ Q_c \end{bmatrix} = \begin{bmatrix} F_g \\ Q_g + Q_e \end{bmatrix} - \begin{bmatrix} J_0^T \\ J_q^T \end{bmatrix} \lambda, \tag{3.5}$$

where $(F_g^T, Q_g^T)^T(g_0, q)$ is the vector of gravity forces while with $Q_e = (0_{1\times n_r}, Q_\epsilon^T)^T$ we express the internal restoring (elastic) forces, with $Q_\epsilon(q_\epsilon) = \mathcal{K}_{\epsilon\epsilon} q_\epsilon$, and $\mathcal{K}_{\epsilon\epsilon}$ the matrix of generalized (constant) stiffness. In (3.5), $J_0^T \lambda$ (respect. $J_q^T \lambda$) is the wrench in $se(3)^* \cong \mathbb{R}^6$ (respect. the vector of internal generalized forces in $\mathbb{R}^n$) induced by the reaction forces $\lambda$ imposed by the $m$ constraints of loops.

**Remark 8.** *Although we limited the approach to time-independent holonomic constraints of the form (3.1), all the results presented in this chapter can be extended without conceptual difficulty to constraints of the more general form [49]:*

$$\Upsilon(g_0, \eta_0, q, \dot{q}, t) = 0_{m\times 1}, \tag{3.6}$$

*where the components of $\Upsilon$ that depend on velocities $(\eta_0, \dot{q})$ can be non-integrable with respect to time i.e., non-holonomic [96].* ∎

**Remark 9.** *Using notation* $\chi = (g_0, \eta_0, \dot{\eta}_0, q, \dot{q}, \ddot{q}, \lambda)$*, the integration of the dynamics of the SMMS from a set of initial conditions at* $t = t_0$*, consists in finding a time-evolution:* $t \in [t_0, +\infty[ \mapsto \chi(t) \in \mathbb{R}^{3\times 6+3n+m}$*, solution of:*

$$\mathcal{R}(\chi(t), t) = 0_{(6+n+m)\times 1}, \tag{3.7}$$

*where* $\mathcal{R}$ *defines the dynamic residual vector of the SMMS, which can be deduced from* (3.1) *as:*

$$\mathcal{R} = \begin{bmatrix} \mathcal{R}_0 \\ \mathcal{R}_q \\ \mathcal{R}_\Upsilon \end{bmatrix} = \begin{bmatrix} \mathcal{M}_{00}\dot{\eta}_0 + M_{0q}\ddot{q} + F \\ M_{q0}\dot{\eta}_0 + M_{qq}\ddot{q} + Q - Q_{ad}(t) \\ \Upsilon \end{bmatrix}. \tag{3.8}$$

*Note that this residual vector is continuous in time and needs to be discretized for numerical simulation purpose. This discretization is based on an implicit integration approach as detailed in next section.* ∎

## 3.2 Time integration

In the context of simulation, a time-integrator allows approximating (3.7) along a sequence of time instants $\{t_k\}_{k\in\mathbb{N}^+}$, with $t_{k+1} = t_k + \Delta t$ and $\Delta t$ a time-step that we consider constant. For reasons of stability and simplicity, we adopt an implicit one-step scheme of the Newmark type [97]. Specifically designed for second order ODEs of Newtonian mechanics, this scheme has been refined over the years to meet the specific needs of structural dynamics [98]. Defined at the origin on a vector space, it can be directly applied to the generalized coordinates $q \in \mathbb{R}^n$ of the internal DoFs of a SMMS. In this case, the scheme requires any $(q, \dot{q}, \ddot{q})$ candidate to be a solution of (3.7) at a time $t_{n+1}$, to satisfy the usual implicit relations:

$$\dot{q} = a(q - q^{(n)}) + f_q^{(n)}, \qquad \ddot{q} = b(q - q^{(n)}) + h_q^{(n)}. \tag{3.9}$$

Here $q^{(n)}$ denotes the value of $q$ at $t_n$ (a notational convention that will be systematically used in what follows), $a$, $b$ are two scalars that depend on the time-step $\Delta t$, while $f_q^{(n)} = f(\dot{q}^{(n)}, \ddot{q}^{(n)})$, $h_q^{(n)} = h(\dot{q}^{(n)}, \ddot{q}^{(n)})$, with $f$ and $h$ two vector functions fixed by the scheme and reminded in Appendix B. As detailed in [40], this integrator can be extended to the external DoFs of $g_0 \in SE(3)$. To that end, we impose to any $(g_0, \eta_0, \dot{\eta}_0)$ candidate to be solution of (3.7) at $t_{n+1}$, to fulfil the Newmark scheme on $SO(3) \times \mathbb{R}^3$:

$$\begin{aligned} \begin{bmatrix} \Omega_0 \\ \dot{r}_0 \end{bmatrix} &= a \begin{bmatrix} \Theta_0 \\ d_0 \end{bmatrix} + \begin{bmatrix} f_\theta^{(n)} \\ f_r^{(n)} \end{bmatrix} \\ \begin{bmatrix} \dot{\Omega}_0 \\ \ddot{r}_0 \end{bmatrix} &= b \begin{bmatrix} \Theta_0 \\ d_0 \end{bmatrix} + \begin{bmatrix} h_\theta^{(n)} \\ r^{(n)} \end{bmatrix} \end{aligned} \tag{3.10}$$

where $g_0 = (R_0, r_0)$, $R_0 \in SO(3)$, $r_0 \in \mathbb{R}^3$, $\eta_0 = (\Omega_0^T, V_0^T)^T$, with $\Omega_0$ and $V_0 = R_0^T \dot{r}_0$ the angular and linear velocities of $\mathcal{B}_0$ in the body frame, while we introduced a new vector

$\nu_0 = (\Theta_0^T, d_0^T)^T \in so(3) \times \mathbb{R}^3 \cong \mathbb{R}^6$:

$$R_0 = R_0^{(n)} \exp_{SO(3)}(\widehat{\Theta}_0), \qquad r_0 = r_0^{(n)} + d_0, \tag{3.11}$$

with "$\exp_{SO(3)}$" here denoting the exponential map of $SO(3)$ [50]. Omitting the upper indices, in (3.10), $(f_\theta, h_\theta)$ and $(f_r, h_r)$ are defined similarly to $(f_q, h_q)$ of (3.9), but with $(\Omega_0, \dot{\Omega}_0)$ and $(\dot{r}_0, \ddot{r}_0)$ replacing $(\dot{q}, \ddot{q})$ respectively (see Appendix B). Introducing (3.10) and (3.11) into the kinematic definitions of $(g_0, \eta_0, \dot{\eta}_0)$ in terms of $R_0$ and $r_0$, allows expressing them at any time beyond $t_n$, in terms of $\nu_0$ only [40]:

$$\eta_0 = A(\nu_0), \qquad \dot{\eta}_0 = B(\nu_0), \qquad g_0 = C(\nu_0), \tag{3.12}$$

where the detailed expressions of $A$, $B$ and $C$ are reminded in Appendix B. Finally, thanks to the Newmark implicit scheme, the search for $(g_0, \eta_0, \dot{\eta}_0, q, \dot{q}, \ddot{q}, \lambda)$ at any time $t_{n+1}$, is reduced to that of $(\nu_0, q, \lambda)$.

**Remark 10.** *In the subsequent numerical resolution, one will need to use the increments of velocities and accelerations compatible with the Newmark integration. To get them, it suffices to apply the variation $\Delta$ to (3.9):*

$$\Delta\dot{q} = a\Delta q, \qquad \Delta\ddot{q} = b\Delta q. \tag{3.13}$$

*Similarly, defining the differential of $g_0$ as $\Delta\zeta_0 = (g_0^{-1}\Delta g_0)^\vee \in se(3)$, the $\Delta$-variation applied to (3.12) gives [40]:*

$$\Delta\eta_0 = \left(\frac{\partial A}{\partial \nu_0}\right)\Delta\nu_0, \quad \Delta\dot{\eta}_0 = \left(\frac{\partial B}{\partial \nu_0}\right)\Delta\nu_0, \quad \Delta\zeta_0 = \left(\frac{\partial C}{\partial \nu_0}\right)\Delta\nu_0, \tag{3.14}$$

*whose detailed expressions are reminded in Appendix B.* ∎

## 3.3 Numerical resolution

Based on this approximation, the numerical resolution is achieved by induction as follows. Assuming we know $\chi^{(n)}$ compatible with the constraints (3.9-3.12) and solution of (3.7) at $t_n$, we have to find a $\chi^{(n+1)}$ compatible with the same constraints, and solution of (3.7) at $t_{n+1}$. To that end, we first introduce (3.9-3.12) into (3.7). This changes the system of DAEs (3.7), into the algebraic system:

$$\overline{\mathcal{R}}^{(n)}(\overline{\chi}, t_{n+1}) = \begin{bmatrix} \overline{\mathcal{R}}_0^{(n)} \\ \overline{\mathcal{R}}_q^{(n)} \\ \overline{\mathcal{R}}_\Upsilon^{(n)} \end{bmatrix} = 0_{(6+n+m)\times 1}, \tag{3.15}$$

whose root $\overline{\chi} = (\nu_0, q, \lambda) \in \mathbb{R}^{6+n+m}$ defines $\overline{\chi}^{(n+1)}$, and the over-bar indicates that the residual vector is now constrained by the integrator, with upper index $(n)$ keeping trace of the dependence to $(g_0^{(n)}, \eta_0^{(n)}, \dot{\eta}_0^{(n)})$ and $(q^{(n)}, \dot{q}^{(n)}, \ddot{q}^{(n)})$ in (3.9) and (3.10). Now remark

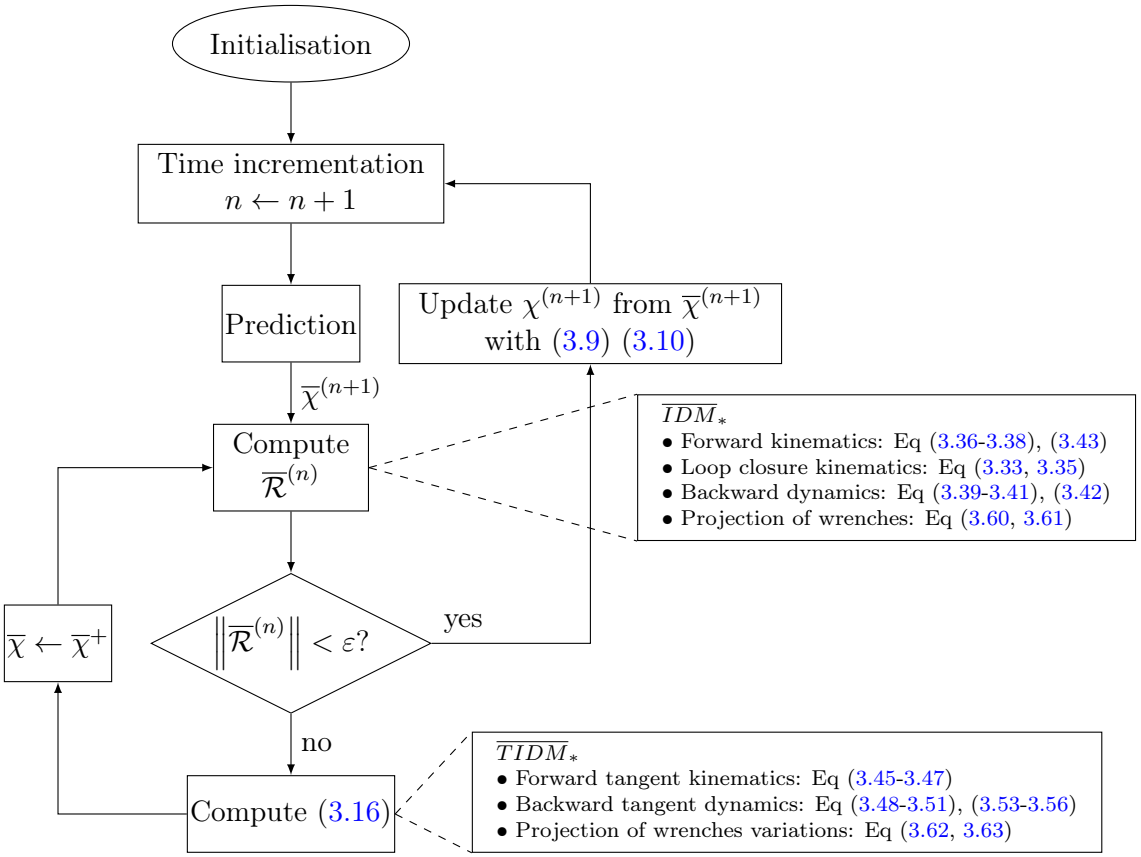



Figure 3.2: Flowchart of the "predictor-corrector" simulation algorithm. The calculation of the residual vector and its Jacobian are based on the $IDM$ and $TIDM$. The over-bar indicates that these algorithms are fed by inputs that fulfill the constraints (3.9–3.12) and (3.13–3.15), imposed by the integration scheme. The lower star defines some adaptations of these two algorithms allowing to calculate directly $\overline{\mathcal{R}}^{(n)}(\overline{\chi}, t_{n+1})$ and its differential, as detailed in Section 3.7.

that (3.15) is a square system of algebraic non-linear equations that can be addressed with an iterative root finder, as the Newton algorithm. In this latter case, the algorithm solves at each iteration the linearized system tangent to (3.15):

$$\overline{\chi}^+ = \overline{\chi} - \left(\frac{\partial \overline{\mathcal{R}}^{(n)}}{\partial \overline{\chi}}\right)^{-1} \overline{\mathcal{R}}^{(n)}(\overline{\chi}, t_{n+1}), \tag{3.16}$$

where the index "+", denotes the value of $\overline{\chi}$ corrected by the current Newton iteration. In this linear system, $(\partial \overline{\mathcal{R}}^{(n)}/\partial \overline{\chi})$ is the Jacobian matrix of the residual vector constrained

by the integrator, which details as:

$$\frac{\partial \overline{\mathcal{R}}^{(n)}}{\partial \overline{\chi}} = \begin{bmatrix} \frac{\partial \overline{\mathcal{R}}_0^{(n)}}{\partial \nu_0} & \frac{\partial \overline{\mathcal{R}}_0^{(n)}}{\partial q} & J_0^T \\ \frac{\partial \overline{\mathcal{R}}_q^{(n)}}{\partial \nu_0} & \frac{\partial \overline{\mathcal{R}}_q^{(n)}}{\partial q} & J_q^T \\ J_0 & J_q & 0 \end{bmatrix}. \tag{3.17}$$

Note that after convergence of the Newton loop, the full set $\chi$ is updated from the converged $\overline{\chi}$ by using (3.9-3.12). Furthermore, at each time step $t_n$, $(g_0, q)$ of $\overline{\chi} = (g_0, q, \lambda)$ can be initialized in the Newton algorithm, with one of the usual predictors of the Newmark scheme, as for instance, the "inertial" or "ballistic" predictor, which is defined by imposing $(\dot{\Omega}_0, \ddot{r}_0, \ddot{q}) = (0_{3\times1}, 0_{3\times1}, 0_{n\times1})$ in (3.9) and (3.10), while for $\lambda$, one can use its value at the previous step $t_{n-1}$. Finally, the dynamic simulator is structured by two nested loops, one global time-loop (the Newmark loop) and the other (Newton loop) that solves (3.15) (see Figure 3.2). Note that while this secondary loop runs, the time is frozen and incremented by the Newmark loop only after the Newton loop has converged.

**Remark 11.** *At each iteration of the Newton loop, the resolution of (3.16) requires the computation of $\overline{\mathcal{R}}^{(n)}$ and $(\partial \overline{\mathcal{R}}^{(n)}/\partial \overline{\chi})$. In order to avoid the heavy symbolic computations, we propose to compute $\overline{\mathcal{R}}^{(n)}$ implicitly, by applying the Newton-Euler (NE) approach of robotics [86]. To do so, we will use the inverse algorithm of SMMS, named $IDM$ for "Inverse Dynamics Model". Moreover, although the Jacobian $(\partial \overline{\mathcal{R}}^{(n)}/\partial \overline{\chi})$ can be evaluated numerically by finite difference approximations (e.g., as done, by default, by Matlab's* `fsolve` *function). Here, we propose here a more accurate computation of this matrix, based on another Newton-Euler algorithm named $TIDM$, for "Tangent Inverse Dynamics Model".* ∎

## 3.4 Calculation of $\mathcal{R}$ and its Jacobian matrix

To introduce the computation of the residual vector and its Jacobian matrix, let us reconsider the upper ODEs of (3.1), in which $Q_e$ is moved from the right to the left-hand side, and the vector $(0_{1\times6}, (Q_{act} - Q_e)^T)^T$, is replaced by a full vector of fictitious actuation forces $(F_a^T, Q_a^T)^T$. This defines an (inverse) Lagrangian model of the form:

$$\begin{bmatrix} F_a \\ Q_a \end{bmatrix} = \begin{bmatrix} \mathcal{M}_{00} & M_{0q} \\ M_{q0} & M_{qq} \end{bmatrix} \begin{bmatrix} \dot{\eta}_0 \\ \ddot{q} \end{bmatrix} + \begin{bmatrix} F_v + F_g - J_0^T \lambda \\ Q_v + Q_g - J_q^T \lambda \end{bmatrix}. \tag{3.18}$$

As this is the case of rigid manipulators [88], we will see soon (Section 3.6) that the Lagrangian model (3.18) can be realized alternatively by an inverse Newton-Euler algorithm in two pass, which formally reads:

$$\begin{bmatrix} F_a \\ Q_a \end{bmatrix} = IDM(\chi), \tag{3.19}$$

where remind that $\chi = (g_0, \eta_0, \dot{\eta}_0, q, \dot{q}, \ddot{q}, \lambda)$. Physically, this $IDM$ computes for any state $(g_0, \eta_0, q, \dot{q})$, the wrench $F_a$ and the full vector of internal forces $Q_a$ (applied on all the

entries of $q = (q_r^T, q_\epsilon^T)^T$), that would be required to ensure the SMMS, considered as fully actuated, to accelerate at $(\dot{\eta}_0, \ddot{q})$, while being subject to the external forces $\lambda$ applied at the tip of the cut branches. Now using the identity (3.19) in (3.15), the two top block-components of the residual vector of the SMMS constrained by the Newmark integrator read:

$$\begin{bmatrix} \overline{\mathcal{R}}_0^{(n)} \\ \overline{\mathcal{R}}_q^{(n)} \end{bmatrix} (\overline{\chi}, t_{n+1}) = \overline{IDM}\,(\overline{\chi}) - \begin{bmatrix} 0_{6\times 1} \\ \Xi(q, t_{n+1}) \end{bmatrix}, \tag{3.20}$$

where the overbar indicates that the $IDM$ is now fed with inputs compatible with the integrator constraints (3.9,3.10) *i.e.,* with inputs that depend on $\overline{\chi}$ only, while we used the further notation:

$$\Xi(q, t) = Q_{ad}(q_\epsilon, t) - Q_e(q_\epsilon). \tag{3.21}$$

In summary, one can compute $(\overline{\mathcal{R}}_0^{(n)}, \overline{\mathcal{R}}_q^{(n)})(\overline{\chi}, t_{n+1})$ with the $IDM$. To this end, it suffices to force its inputs to fulfill the constraints of the integrator (3.9,3.10), and to remove $(0_{1\times 6}, \Xi(q, t_{n+1})^T)^T$ from its vector of outputs. Remarkably, the same idea can be applied to the computation of the top two block-rows of $(\partial\overline{\mathcal{R}}^{(n)}/\partial\overline{\chi})$. Indeed, differentiating (3.20) provides the identity:

$$\begin{bmatrix} \Delta\overline{\mathcal{R}}_0^{(n)} \\ \Delta\overline{\mathcal{R}}_q^{(n)} \end{bmatrix} = \begin{bmatrix} \frac{\partial\overline{\mathcal{R}}_0^{(n)}}{\partial\overline{\chi}} \\ \frac{\partial\overline{\mathcal{R}}_q^{(n)}}{\partial\overline{\chi}} \end{bmatrix} \Delta\overline{\chi} = \overline{TIDM}\,(\overline{\chi}, \Delta\overline{\chi}) - \begin{bmatrix} 0_{6\times 1} \\ \Delta\Xi(q, \Delta q, t_{n+1}) \end{bmatrix} \tag{3.22}$$

where:

$$\Delta\Xi(q, \Delta q, t) = \Delta Q_{act}(q_\epsilon, \Delta q_\epsilon, t) - \Delta Q_e(q_\epsilon, \Delta q_\epsilon), \tag{3.23}$$

where, since the time is frozen in the correction Newton's loop, $\Delta Q_e = K_{\epsilon\epsilon}\Delta q_\epsilon$ and $\Delta Q_{act} = (\Delta\tau_{act}(t)^T, \Delta Q_{act\epsilon}^T)^T = (0_{1\times n_r}, \Delta Q_{act\epsilon}^T)^T$ with $Q_{act\epsilon}$ which may depend on $q_\epsilon$ [21]. In (3.22), $\overline{TIDM}$ denotes the tangent algorithm to the $IDM$ or $TIDM$, with the over-bar indicating that it is fed with inputs compatible with the constraints (3.9,3.10) and (3.13,3.14) imposed by the integrator. Formally, this further RNEA is defined by the differential of the input-output map (3.19):

$$\begin{bmatrix} \Delta F_a \\ \Delta Q_a \end{bmatrix} = TIDM\,(\chi, \Delta\chi)\,, \tag{3.24}$$

which for any $\chi = (g_0, \eta_0, \dot{\eta}_0, q, \dot{q}, \ddot{q}, \lambda)$, allows to compute the variations of outputs $\Delta F_a$ and $\Delta Q_a$ required by imposing variations $\Delta\chi = (\Delta\zeta_0, \Delta\eta_0, \Delta\dot{\eta}_0, \Delta q, \Delta\dot{q}, \Delta\ddot{q}, \Delta\lambda)$ as inputs. Now let us define $\delta_i \in \mathbb{R}^{6+n+m}$ as the unit vector, with zero entries, except the $i^{th}$, which is equal to one. Then, the identity (3.22) shows that feeding $\overline{TIDM}$ with $\overline{\chi}$ and $\Delta\overline{\chi} = \delta_i$, $i = 1, 2, \ldots, 6+n+m$, and removing from its outputs, $(0_{1\times 6}, \Delta\Xi^T)^T$ fed with the same inputs (see next remark), allows to compute column after column, the top two block-rows of $(\partial\overline{\mathcal{R}}^{(n)}/\partial\overline{\chi})$ in any $\overline{\chi}$.

**Remark 12.** *Before detailing the $IDM$ and $TIDM$ algorithms of a SMMS, it is worth mentioning that if we have at our disposal these two algorithms, exploiting the two identities (3.20) and (3.22) to compute their left-hand sides, requires to calculate $\Xi(t_{n+1}) =$*

$Q_{act}(t_{n+1}) - Q_e$ *and* $\Delta\Xi(t_{n+1}) = \Delta Q_{act}(t_{n+1}) - \Delta Q_e$ *(for the sake of concision, we will often indicate the time dependency, only). Moreover, using the identity* (3.20) *only provides* $\overline{\mathcal{R}}_0^{(n)}$ *and* $\overline{\mathcal{R}}_q^{(n)}$ *in* (3.8)*, and not* $\overline{\mathcal{R}}_\Upsilon^{(n)} = \Upsilon(g_0, q)$*. On the other hand, owing to the symmetries of the block matrix* (3.17)*, calculating its two top rows with the identity* (3.22) *is enough to reconstruct the full Jacobian* $(\partial\overline{\mathcal{R}}^{(n)}/\partial\overline{\chi})$*. As we will see later (see Section 3.7), the missing vectors* $\Xi(t_{n+1})$*,* $\Delta\Xi(t_{n+1})$*, and* $\overline{\mathcal{R}}_\Upsilon^{(n)} = \Upsilon(g_0, q)$ *can also be computed with some of the subparts of the* $IDM$ *and* $TIDM$*. Therefore, these two algorithms are the key of the computation of the residual vector and the Jacobian of* (3.16)*. Since they are both based on the Newton-Euler model of SMMS, we will now introduce this model.* ∎

## 3.5 Newton-Euler model of a SMMS

As per usually, a rigid link is modelled by a rigid 3D body of arbitrary shape (not necessarily rod-shaped). On the other hand, in order to easily hybridize the $IDM$ of rigid multi-body systems [88], with that of Cosserat rods [21] into a single generalized $IDM$, it is convenient to consider a Cosserat rod as two tip cross-sections (*i.e.,* two (fictitious) rigid bodies with no thickness and no inertia), connected by a distributed flexible, or continuum, joint. Therefore, the continuous model of poses, velocities, acceleration and stress of Cosserat rods is used in the model of joints and not bodies. As we will soon see, by adopting this choice, the resulting generalized $IDM$ will, at the highest level, take a form similar to that of the rigid case.

### 3.5.1 Segmentation of a SMMS

Using the above viewpoint, the tree-like SMMS of Section 3.1 is first segmented into a rigid body sequence consisting of the original 3D rigid bodies and the tip sections of the Cosserat rods considered as fictitious rigid bodies. All these rigid bodies are indexed following the usual Newton-Euler conventions *i.e.,* from the reference body $\mathcal{B}_0$, which is the fixed basis for a manipulator, or the free-floating one of a locomotor, to $\mathcal{B}_N$, with bodies $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_N$ numbered increasingly while descending the branches. As in any tree-like system, a body has only one antecedent and possibly several successors. In the subsequent developments, $j$ preferentially denotes the current body index of the algorithm, while $k = a(j)$ is its antecedent and $l$ is the index of one of its successors *i.e.,* $l \in s(j)$, where $s(j)$ is the set of the indexes of all the successors of $\mathcal{B}_j$. As for rigid systems, any two contiguous bodies are separated by a joint. Indexing each of these joints with the index of its succeeding body, one can form the set of joints $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_N$, some of them being rigid lumped joints (fixed or revolute) as those met in rigid systems, while others are continuum joints modelled by Cosserat rods. This segmentation process is illustrated in Figure 3.3. To handle a kinematic loop, we introduce a virtual cut at the concerned (passive) joint (see Section 3.1.2), thereby opening the loop and creating two separate branches. Each branch is provided with a fictitious rigid body connected via a fixed joint. This formulation facilitates the natural incorporation of virtual cuts into the NE formalism and supplies the frames required by the algorithm, which we discuss in the next section.

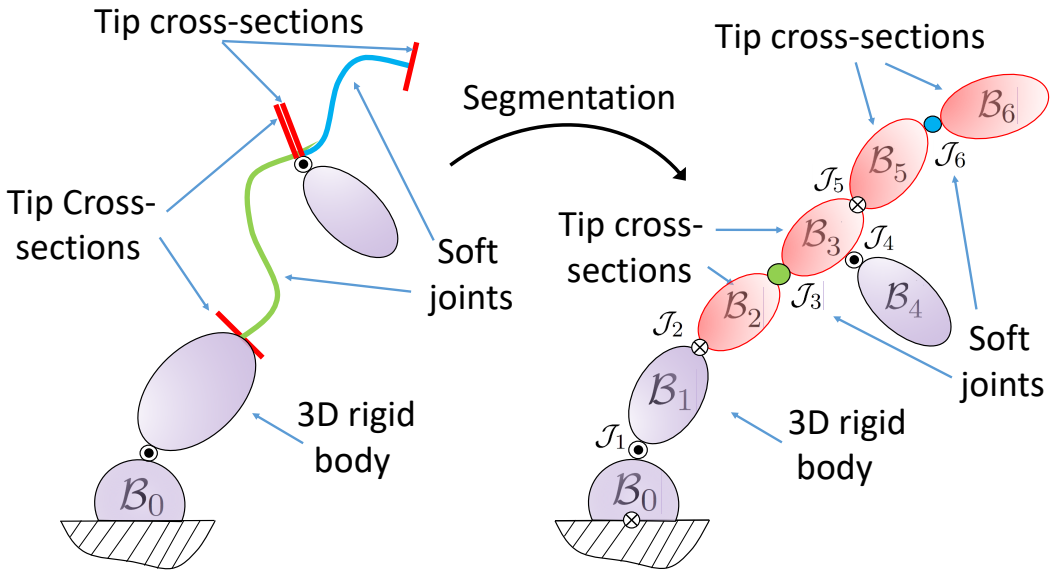



Figure 3.3: Segmentation of a SMMS for $IDM$ and $TIDM$. The two Cosserat rods are declared as joints $\mathcal{J}_3$ and $\mathcal{J}_6$. Their two tip-cross sections are declared as rigid bodies ($\mathcal{B}_2$,$\mathcal{B}_3$) and ($\mathcal{B}_5$,$\mathcal{B}_6$).

**Remark 13.** *The proximal and distal cross sections of a rod $j$, located at $X = 0$ and $X = \ell_j$, define the rigid bodies $\mathcal{B}_k$ and $\mathcal{B}_j$, respectively. The continuum joint $\mathcal{J}_j$ is then defined by the remaining portion of the rod, within the domain $X \in ]0, \ell_j[$, which precedes the distal cross section $\mathcal{B}_j$. It is important to note that this definition aligns with the application of the Cosserat PDEs introduced in (2.9). Specifically, the numerical integration of these equations yields the relative kinematics of $\mathcal{F}(X = \ell_j)$ w.r.t. $\mathcal{F}(X = 0)$ in a manner consistent with how a rigid joint $\mathcal{J}_j$ relates the kinematics of $\mathcal{B}_j$ to the one of $\mathcal{B}_k$.* ∎

### 3.5.2 Frames

According to the standard Newton-Euler conventions for rigid systems [87], each joint within the SMMS, denoted as $\mathcal{J}_j$, is associated with a frame $\mathcal{F}_j = (O_j, s_j, n_j, a_j)$. In the case of a rigid joint, the frame $\mathcal{F}_j$ is centred at the joint origin, and for a revolute joint, the axis $a_j$ aligns with the joint's axis of rotation. For a distributed, or continuum, joint, the frame $\mathcal{F}_j$ corresponds to $\mathcal{F}(X = \ell_j)$. If a body $\mathcal{B}_j$ is located at the end of a branch resulting from the virtual cut of a loop, the fixed joint $\mathcal{J}_j^+$, paired with the fictitious rigid body $\mathcal{B}_j^+$, is provided with the frame $\mathcal{F}_j^+ = (O_j^+, s_j^+, n_j^+, a_j^+)$. This frame has $O_j^+$ positioned at the (fixed) joint origin and $a_j^+$ aligned with the axis of the cut joint, assuming it is a revolute joint. Finally, the Euclidean space is equipped with an inertial frame $\mathcal{F}_e$.

**Remark 14.** *Note that the use of fixed joints and fictitious rigid bodies does not affect the operation of the algorithm. Indeed, fixed joints introduce no coordinates, while fictitious*

*rigid bodies do not contribute to the system's inertia.* ∎

### 3.5.3 Newton-Euler model of rigid bodies

In the Newton-Euler approach, the configuration of each rigid body $\mathcal{B}_j$ is defined by the inertial pose of its body frame *i.e.,* $g_j = (R_j, r_j) \in SE(3)$, with $R_j$ and $r_j$ the orientation matrix and origin's position of $\mathcal{F}_j$, in $\mathcal{F}_e$. Introducing the inertial velocity twist of $\mathcal{B}_j$ in the body frame $\eta_j = (g_j^{-1}\dot{g}_j)^\vee \in \mathbb{R}^6$, and applying Newton's laws and Euler's theorem to each body $\mathcal{B}_j$ isolated in the structure, provides the usual Newton-Euler equations that govern the time evolution of the $g_j$s on $SE(3)$:

- NE dynamics balance (ODEs) of rigid bodies:

$$\mathcal{M}_j\dot{\eta}_j - \mathrm{ad}^T_{\eta_j}\mathcal{M}_j\eta_j = F_{\mathrm{ext},j} + F_j - \sum_{l\in s(j)} {}^jF_l. \tag{3.25}$$

In these dynamic balances, $\mathcal{M}_j \in \mathbb{R}^{6\times6}$ is the (screw) inertia matrix of an arbitrary shaped rigid body $\mathcal{B}_j$ (equal to zero if $\mathcal{B}_j$ is a fictitious body), $F_{\mathrm{ext},j} \in \mathbb{R}^6$ is the wrench of external forces (e.g gravity, external contacts) exerted on it, $F_j \in \mathbb{R}^6$ is the wrench of internal contacts exerted by the antecedent body $\mathcal{B}_k$, $(k = a(j))$ onto $\mathcal{B}_j$. With these definitions and our notational conventions, the wrenches in the sum of (3.25) are contact wrenches transmitted by $B_j$ to all its successors $\mathcal{B}_l$ through the joints that connect them. Therefore they are first defined in the $\mathcal{F}_l$ frames by $F_l$, and then expressed in $\mathcal{F}_j$ through ${}^jF_l$, while action-reaction principle motivates the minus sign.

### 3.5.4 Newton-Euler model of continuum joints

In the Newton-Euler approach, the configuration of a Cosserat rod $\mathcal{J}_j$ is defined by the inertial pose of all its cross-sectional frames, as described in Section 2.1, and reminded here as $g_j(X) = (R_j, r_j)(X) \in SE(3)$, $\forall X \in ]0, \ell_j[$, where the domain has been restricted as the cross-sections at $X = 0$ and $X = \ell_j$ are considered as rigid bodies. As detailed in Section 2.1, we recall here that applying Newton's laws and Euler's theorem along such a rod, provides the following closed formulation that governs the time-evolution of $g_j(.)$:

- NE dynamic balance of $X$-cross sections (PDE of Section 2.1.5), $\forall X \in ]0, \ell_j[$:

$$\mathcal{M}_j\dot{\eta}_j - \mathrm{ad}^T_{\eta_j}\mathcal{M}_j\eta_j = \bar{F}_{ext,j} + \Lambda_j' - \mathrm{ad}^T_{\xi_j}\Lambda_j, \tag{3.26}$$

- Boundary conditions:

$$\Lambda_j(0) = -F_{a(j)}, \qquad \Lambda_j(\ell_j) = -F_j. \tag{3.27}$$

- Active constitutive law along $[0, \ell_j]$:

$$\Lambda_j = \Lambda_{act,j}(t) + \mathcal{H}\epsilon_j. \tag{3.28}$$

Where $\epsilon_j(X)$, $\xi_j^0(X)$, $\mathcal{M}_j$, $\mathcal{H}_j$, $\Lambda_j(X)$, $\Lambda_{act,j}$ were defined in Section 2.1 and the aforementioned constitutive law can be adapted to approximate internal damping.

### 3.5.5 Kinematic model of joint connections

Using the segmentation of Section 3.5.1, a joint connecting any two consecutive rigid bodies $\mathcal{B}_k$ and $\mathcal{B}_j$ imposes the relation between their poses:

$$g_j = g_k \, {}^k g_j. \tag{3.29}$$

In details, according to the standard uses of rigid robots, when $\mathcal{J}_j$ is a rigid lumped joint, the relative pose ${}^k g_j$ in (3.29) is a function of $q_{rj}$ or is fixed by design, depending whether the joint is revolute or fixed (see Remark 15 below) Furthermore, we use the $A_j$, matrix previously introduced in Section 2.3.6, to parametrize the DoFs of the joint. As an example, if $\mathcal{J}_j$ is a revolute joint, with $a_j$ as axis of rotation, then $A_j = (0,0,1,0,0,0)^T$. When $\mathcal{J}_j$ is a (distributed) soft joint, the internal rod kinematics (Reissner, Kirchhoff...) imposes to any two infinitely close cross-sections along the joint, the differential relation between their poses (Section 2.3.6):

$$g_j' = g_j \widehat{\xi} = g_j \left( \xi_j^0 + A_j \Phi q_{\epsilon j} \right)^\wedge, \tag{3.30}$$

in which we remind that the vector of strain coordinates of $\mathcal{J}_j$ $q_{\epsilon,j} \in \mathbb{R}^{n_{\epsilon,j}}$, defines a sub-part of $q_\epsilon$. In the following RNEA, (3.29) and (3.30) are used to relate the inertial poses $g_j$ and $g_j(.)$ to the SMMS generalized coordinates $q = (q_r^T, q_\epsilon^T)^T$ of the Lagrangian parametrization of Section 3.1. Finally, remark that the cut revolute joint of a closed loop will not introduce such kinematic parameters, but rather Lagrange multipliers in charge of forcing a set of closure kinematic constraints that we are now going to detail.

**Remark 15.** *The relative pose ${}^k g_j$ of a rigid joint can be detailed as:*

$${}^k g_j = {}^k g_{j,c} \, g_j(q_j), \tag{3.31}$$

*where ${}^k g_{j,c}$ is a constant rigid transformation form the antecedent (constant offset) and $g_j(q_{rj})$ is a pose which depends on the joint type:*

$$g_j(q_{rj}) = \begin{bmatrix} \exp_{SO(3)}(\Theta_j) & D_j \\ 0_{1\times 3} & 1 \end{bmatrix}, \tag{3.32}$$

*with $\nu_j = (\Theta_j^T, D_j^T) = A_j q_j$.* ∎

### 3.5.6 Kinematic model of loop closures

Let us consider two terminal bodies $\mathcal{B}_p$ and $\mathcal{B}_s$ connected by a revolute joint virtually cut when opening a loop. These are be provided with the fixed joints $\mathcal{J}_{p+}$ and $\mathcal{J}_{s+}$ supporting the frames $\mathcal{F}_{p+}$ and $\mathcal{F}_{s+}$, respectively (Section 3.5.2). Then, using the convention that when $s > p$, $\mathcal{F}_{p+}$ is the leader frame and $\mathcal{F}_{s+}$ is the follower one, a virtually cut revolute joint imposes to the relative pose ${}^{p+} g_{s+}$ to be compatible with the five scalar geometric constraints:

$$\Upsilon_{s|p} = \overline{A}_{p+} \begin{bmatrix} \log_{SO(3)}({}^{p+}R_{s+}) \\ {}^{p+}r_{s+} \end{bmatrix} = \overline{A}_{p+}^T \begin{bmatrix} 0_{3\times 1} \\ 0_{3\times 1} \end{bmatrix}, \tag{3.33}$$

where the $\log_{SO(3)}$-map stands for the inverse of the exponential in $SO(3)$ [50], while $\overline{A}_{p+}$ is the complementary matrix for the (virtually cut) revolute joint. Again, in this case, $\overline{A}_{p+}$ corresponds to the constrained DoFs matrix, introduced in Section 2.3.6, representing the constrained motion of the (virtually cut) joint. This context holds for any type of cut joint where it suffices to replace the 5 constraints above by any number of constraints depending on the nature of the joint. In particular, virtually cutting a fixed joint introduces six constraints instead of five (*i.e.*, $\overline{A}_{p+} = \mathbb{1}_6$). Time-differentiating (3.33) provides the kinematic form of the closure loop constraints:

$$\dot{\Upsilon}_{s|p} = J_{s|p}\,\eta_{s_+/p_+} = \overline{A}_{p+}0_{6\times 1}, \tag{3.34}$$

where $\eta_{s_+/p_+} = ({}^{s_+}g_{p_+}\,{}^{p_+}\dot{g}_{s_+})^\vee = \eta_{s_+} - \mathrm{Ad}_{{}^{s_+}g_{p_+}}\eta_{p_+}$ is the relative twist of the follower frame $\mathcal{F}_{s_+}$ with respect to the leader one $\mathcal{F}_{p_+}$, while $J_{s|p}$, is a kinematic Jacobian matrix defined by:

$$J_{s|p} = \overline{A}_{p+}\begin{bmatrix} T(\Theta_{s|p})^{-1} & \mathbb{0}_3 \\ \mathbb{0}_3 & {}^{p+}R_{s+} \end{bmatrix}, \tag{3.35}$$

which depends on $\Theta_{s|p} = \log({}^{p+}R_{s+}) \in so(3) \cong \mathbb{R}^3$. Finally, note that if the loop is connected to the ground, we simply have $s = 0$.

**Remark 16.** *Gathering all the equations from (3.25) to (3.35) defines the Newton-Euler formulation of the SMMS dynamics. It is here expressed on the Lie group $SE(3)$ as this was the case for rigid robots in [89, 90]. This formulation is entirely equivalent to the Lagrangian one defined by the set of DAEs (3.1). As announced in Section 3.4, we are now going to exploit this equivalence in order to calculate the residual of the Lagrangian formulation (3.15) and its Jacobian matrix (3.17). This is accomplished through the implementation of two RNEAs that are detailed in the next section.* ∎

## 3.6 Recursive $IDM$ and $TIDM$ of a SMMS

### 3.6.1 Newton-Euler $IDM$ of a SMMS

Like for a rigid manipulator [88], the $IDM$ proceeds in two passes. The first, descending (from the base to the ends of the branches), calculates the inertial poses, velocities and accelerations along the branches from the knowledge of $(g_0, \eta_0, \dot{\eta}_0, q, \dot{q}, \ddot{q})$. The second, ascending, calculates the interbody wrenches transmitted across the rigid and soft joints from the knowledge of $\lambda$ and the inertial kinematics calculated by the first pass. These interbody wrenches are then projected on the DoFs of joints to calculate $Q_a$ of (3.18), while $F_a$ is the last "interbody" wrench computed by the second pass. As introduced before, thanks to the declaration of rods as distributed, or continuum, joints, at a first (high) level, this algorithm takes the usual form of that of rigid systems, but here applied to the segmentation of Section 3.5.1. The only difference lies in a second (low) level, where the usual discrete forward and backward transfers on inter-body kinematics (poses, velocities, accelerations) and inter-body wrenches, respectively, are replaced by continuous transfers obtained by integrating forward (3.30) (and its first and second temporal derivatives), and backward (3.26), when the joint is a soft distributed one.

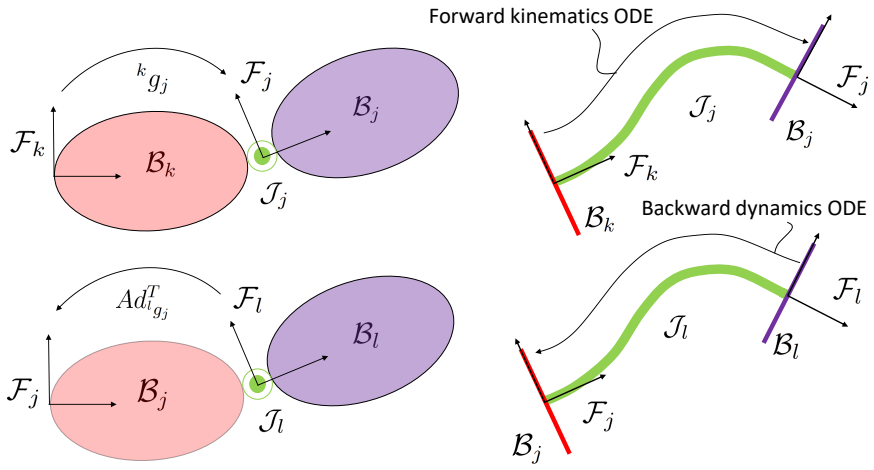



Figure 3.4: Forward transport of poses (top) and backward transport of wrenches (bottom) over a rigid (left) and a soft joint (right). The joints are coloured green, the rigid bodies red and purple.

### Forward kinematics

Time differentiating twice (3.29) provides the kinematics model that relates the pose, velocities and accelerations of all the rigid bodies $\mathcal{B}_j$ (including the tip cross sections of soft links) in the form of a recursion on body indexes, initialized by $(g_0, \eta_0, \dot{\eta}_0)$:

$$\begin{array}{l} \text{For} \quad j = 1, 2, \ldots, N, \text{ do :} \\ k = a(j), \\ \left\{ \begin{array}{ll} g_j & = g_k {}^k g_j, \\ \eta_j & = Ad_{{}^j g_k} \eta_k + \eta_{j/k}, \\ \dot{\eta}_j & = Ad_{{}^j g_k} \dot{\eta}_k + ad_{\eta_j} \eta_{j/k} + \dot{\eta}_{j/k}, \end{array} \right. \end{array} \tag{3.36}$$

where from now on, $\eta_{j/k} = \left( {}^k g_j^{-1} \, {}^k \dot{g}_j \right)^\vee \in \mathbb{R}^6$ denotes the twist of relative velocities between $\mathcal{B}_j$ and its antecedent $\mathcal{B}_k$ expressed in the mobile frame of $\mathcal{B}_j$. Now, when computing (3.36), we have two cases:

- If $\mathcal{J}_j$ is a lumped rigid one, one uses the usual relations:

$$ {}^k g_j = {}^k g_j(q_{rj}), \quad \eta_{j/k} = A_j \dot{q}_{rj}, \quad \dot{\eta}_{j/k} = A_j \ddot{q}_{rj}, \tag{3.37}$$

  while if the joint is fixed, ${}^k g_j$ is a constant pose that depends on the design.

- If $\mathcal{J}_j$ is a continuum joint, $({}^k g_j, \eta_{j/k}, \dot{\eta}_{j/k})$ is the terminal value of the forward integration from $X = 0$ to $\ell_j$ of the system of ODEs described in (2.9), which we recall

here as:

$$\frac{d}{dX}\begin{bmatrix} {}^k g_j \\ \eta_{j/k} \\ \dot{\eta}_{j/k} \end{bmatrix} = \begin{bmatrix} {}^k g_j \hat{\xi}_j \\ -\mathrm{ad}_{\xi_j}\eta_{j/k} + \dot{\xi}_j \\ -\mathrm{ad}_{\xi_j}\dot{\eta}_{j/k} - \mathrm{ad}_{\dot{\xi}_j}\eta_{j/k} + \ddot{\xi}_j \end{bmatrix}, \tag{3.38}$$

which is initialized with $\left({}^k g_j, \eta_{j/k}, \dot{\eta}_{j/k}\right)(0) = (\mathbb{1}_4, 0_{6\times 1}, 0_{6\times 1})$ while with the VSA: $(\xi_j, \dot{\xi}_j, \ddot{\xi}_j) = (A_j\Phi_j q_{\epsilon,j} + \xi_j^0, A_j\Phi_j \dot{q}_{\epsilon,j}, A_j\Phi_j \ddot{q}_{\epsilon,j})$.

To summarize, if $\mathcal{J}_j$ is rigid, the joint kinematics between $\mathcal{B}_k$ and $\mathcal{B}_j$ are directly given by the discrete (lumped) relations (3.37), while if it is continuum, they are defined by the continuum composition of poses, velocities and accelerations along the distributed joints, with (3.38). These discrete and continuous kinematics transfers along the joints are illustrated in the top two diagrams of the Figure 3.4.

### Backward dynamics

The model that relates the wrenches transmitted between the $\mathcal{B}_j$s (including the tip cross-sections of the rods) consists of the backward recursion on body indexes:

$$\begin{aligned} &\text{For} \quad j = 1, 2, \ldots, N, \ \text{do}: \\ &k = a(j), \\ &F_j = \mathcal{M}_j\dot{\eta}_j - \mathrm{ad}^T_{\eta_j}\mathcal{M}_j\eta_j - F_{\mathrm{ext},j} + \sum_{l\in s(j)} {}^j F_l, \end{aligned} \tag{3.39}$$

which are initialized by the wrenches applied to the tip of the branches as detailed in the Remark 17 below. Now when computing (3.39), we have two cases:

- If $\mathcal{J}_l$ is rigid, we use the usual relations:

$$ {}^j F_l = Ad^T_{{}^l g_j} F_l, \tag{3.40}$$

- If $\mathcal{J}_l$ is soft, ${}^j F_l = -\Lambda_l(0)$ is the result of the backward integration from $X = \ell_l$ to 0 of:

$$\Lambda_l' = ad^T_{\xi_l}\Lambda_l + \mathcal{M}_l\dot{\eta}_l - ad^T_{\eta_l}\mathcal{M}_l\eta_l - \overline{F}_{\mathrm{ext},l}, \tag{3.41}$$

  starting from initial conditions $\Lambda_l(\ell_l) = -F_l$.

In words, if the $\mathcal{J}_l$ joint is rigid, $F_l$ is directly transmitted to $\mathcal{B}_j$ through the rigid transport of (3.40), while if it is soft, the contact wrench $-F_l$ is transmitted to $\mathcal{B}_j$ through the continuum transport of the wrench along the rod that connects $\mathcal{B}_j$ to $\mathcal{B}_l$ according to its dynamic balance (3.41). This context is illustrated in Figure 3.4 (bottom).

**Remark 17.** *The initial conditions of the backward dynamics enter through the sum term in (3.39), when $j$ is the index, say $p$, of a terminal body of a branch. In particular, if such a terminal body does not support any cut joint, it does not contribute to the initial*

*conditions in* (3.39). *Otherwise, let us consider two terminal bodies* $\mathcal{B}_p$ *and* $\mathcal{B}_s$ *result of the virtual cut of a revolute joint (Section 3.1). Then, using the kinematic model of constraints* (3.35) *and the duality between twists and wrenches, we have:*

$$F_{s+} = J_{s|p}^T \lambda_{s|p}, \qquad F_{p_+} = -Ad^T_{s_+ g_{p_+}} F_{s_+}, \tag{3.42}$$

*where* $\lambda_{s|p} \in \mathbb{R}^5$ *is a vector of reaction generalized forces, dual of* $\dot{\Upsilon}_{s|p}$ *in* (3.35), *which defines five components of the Lagrangian vector* $\lambda$ *in* (3.5), *while* $\lambda_{s|p} \in \mathbb{R}^6$ *if the cut joint is fixed.* ∎

**Remark 18.** *Note that in* (3.41), *one needs to use the inertial fields, denoted* $(g_l, \eta_l, \dot{\eta}_l)(.)$, *that can be computed with the formulas:*

$$\begin{cases} g_l(X) & = g_j \, {}^j g_l(X), \\ \eta_l(X) & = Ad_{j_{g_l(X)}} \eta_j + \eta_{l/j}(X), \\ \dot{\eta}_l(X) & = Ad_{j_{g_l(X)}} (\dot{\eta}_j + ad_{\eta_{l/j}(X)} \eta_j) + \dot{\eta}_{l/j}(X), \end{cases} \tag{3.43}$$

*where the inter-body kinematic fields* $({}^k g_j, \eta_{j/k}, \dot{\eta}_{j/k})(.)$ *along the rods are known after the first pass through the integration of* (3.38). ∎

**Summary of the** $IDM$

The $IDM$ algorithm progresses as follows. First, a forward pass initialized by $(g_0, \eta_0, \dot{\eta}_0)$ and fed by $(q, \dot{q}, \ddot{q})$, computes from the basis $\mathcal{B}_0$ to all the tip branches, the inertial poses, velocities and accelerations through the composition of the kinematic model of rigid bodies and soft joints (3.36,3.37,3.38) and using (3.43). These kinematic variables are used to feed a second backward pass that computes the inter-body wrenches from the tip branches to the basis $\mathcal{B}_0$ through the composition of the dynamic model of rigid bodies and soft joints (3.39,3.40,3.41). According to (3.42), this second pass is initialized by the contact forces $\lambda$ transmitted by the cut joints, that are part of the inputs to the algorithm. While running, the second pass provides all the contact wrenches $F_j$ and $\Lambda_j(.)$ transmitted across the rigid bodies and continuum joints to which we apply the projective relations:

$$Q_{ar,j} = A_j^T F_j, \qquad Q_{a\epsilon,j} = -\int_0^{l_j} \Phi_j^T A_j^T \Lambda_j dX, \tag{3.44}$$

for all lumped revolute and continuum joints. Then all the values of (3.44) are used to fill $Q_a = (Q_{ar}^T, Q_{a\epsilon}^T)^T$. Moreover, if the system has a free floating basis, the algorithm also provides $F_0 = F_a$ which completes the outputs $(F_a^T, Q_a^T)^T$ of the input-output map (3.19).

### 3.6.2 Newton-Euler $TIDM$ of a SMMS

The $TIDM$ is derived by calculating the linear perturbations of the outputs of the $IDM$ produced by linear perturbations of its inputs. Practically, it is simply deduced from the $IDM$ by applying the variation $\Delta$ to all the above expressions and exploiting the

properties of the Ad and ad maps. Like the $IDM$, the $TIDM$ consists of two recursions on the body indexes, one (tangent) forward, computes the variations of kinematics, the second, (tangent) backward, computes those of the internal wrenches transmitted along the structure.

### Tangent forward kinematics

It is deduced from the variation $\Delta$ of (3.36):

$$\begin{aligned}
&\text{For} \quad j = 1, 2, \ldots, N, \ \text{do}: \\
&k = a(j), \\
&\begin{cases}
\Delta\zeta_j &= \mathrm{Ad}_{j g_k} \Delta\zeta_k + \Delta\zeta_{j/k}, \\
\Delta\eta_j &= \mathrm{Ad}_{j g_k} \Delta\eta_k + \mathrm{ad}_{\left(\mathrm{Ad}_{j g_k} \eta_k\right)} \Delta\zeta_{j/k} + \Delta\eta_{j/k}, \\
\Delta\dot{\eta}_j &= \mathrm{Ad}_{j g_k} \Delta\dot{\eta}_k + \mathrm{ad}_{\left(\mathrm{Ad}_{j g_k} \dot{\eta}_k\right)} \Delta\zeta_{j/k} - \mathrm{ad}_{\eta_{j/k}} \Delta\eta_j + \mathrm{ad}_{\eta_j} \Delta\eta_{j/k} + \Delta\dot{\eta}_{j/k},
\end{cases}
\end{aligned} \tag{3.45}$$

which is initialized by the basis kinematic variations $(\Delta\zeta_0, \Delta\eta_0, \Delta\dot{\eta}_0)$, and where:

- If $\mathcal{J}_j$ is a lumped rigid joint, one simply has:

$$\left(\Delta\zeta_{j/k}, \Delta\eta_{j/k}, \Delta\dot{\eta}_{j/k}\right) = A_j \left(\Delta q_{rj}, \Delta\dot{q}_{rj}, \Delta\ddot{q}_{rj}\right). \tag{3.46}$$

- If $\mathcal{J}_j$ is soft, $(\Delta\zeta_{j/k}, \Delta\eta_{j/k}, \Delta\dot{\eta}_{j/k})$ is the terminal value of the forward integration:

$$\frac{d}{dX} \begin{bmatrix} \Delta\zeta_{j/k} \\ \Delta\eta_{j/k} \\ \Delta\dot{\eta}_{j/k} \end{bmatrix} = \begin{bmatrix} -\mathrm{ad}_{\xi_j} \Delta\zeta_{j/k} \\ -\mathrm{ad}_{\xi_j} \Delta\eta_{j/k} + \mathrm{ad}_{\eta_{j/k}} \Delta\xi_j \\ -\mathrm{ad}_{\xi_j} \Delta\dot{\eta}_{j/k} - \mathrm{ad}_{\dot{\xi}_j} \Delta\eta_{j/k} + \mathrm{ad}_{\eta_{j/k}} \Delta\dot{\xi}_j + \mathrm{ad}_{\dot{\eta}_{j/k}} \Delta\xi_j \end{bmatrix} + \begin{bmatrix} \Delta\xi_j \\ \Delta\dot{\xi}_j \\ \Delta\ddot{\xi}_j \end{bmatrix} \tag{3.47}$$

  which is fed by the initial conditions $(\Delta\zeta_{j/k}, \Delta\eta_{j/k}, \Delta\dot{\eta}_{j/k})(0) = 0_{6\times 3}$, and the set $(\Delta\xi_j, \Delta\dot{\xi}_j, \Delta\ddot{\xi}_j) = A_j \Phi_j (\Delta q_{\epsilon,j}, \Delta\dot{q}_{\epsilon,j}, \Delta\ddot{q}_{\epsilon,j})$.

### Tangent backward dynamics

Applying the variation $\Delta$ to (3.39) gives:

$$\begin{aligned}
&\text{For} \quad j = 1, 2, \ldots, N, \ \text{do}: \\
&\Delta F_j = \mathcal{M}_j \Delta\dot{\eta}_j - \mathrm{ad}^T_{\eta_j} \mathcal{M}_j \Delta\eta_j - \mathrm{ad}^T_{\Delta\eta_j} \mathcal{M}_j \eta_j - \Delta F_{\mathrm{ext},j} + \sum_{l \in s(j)} \Delta^j F_l,
\end{aligned} \tag{3.48}$$

which is initialized by the variations of the wrenches applied onto the terminal bodies of the branches *i.e.,* through some $\Delta^j F_l$ when $\mathcal{B}_j$ is a terminal body. These variations take different expressions depending whether the terminal body supports a virtually cut joint or not (see section below). Once again, we have two cases:

- If $\mathcal{J}_l$ is a rigid joint, one has:

$$\Delta^j F_l = \mathrm{Ad}^T_{{}^l g_j} \Delta F_l + \Delta \mathrm{Ad}^T_{{}^l g_j} F_l, \tag{3.49}$$

  which can be detailed as:

$$\Delta^j F_l = \mathrm{Ad}^T_{{}^l g_j} \Delta F_l - \mathrm{ad}^T_{\Delta\zeta_{l/j}} \mathrm{Ad}^T_{{}^l g_j} F_l, \tag{3.50}$$

- If $\mathcal{J}_l$ is soft, $\Delta^j F_l = -\Delta\Lambda_l(0)$ is the result of the backward integration from $X = \ell_l$ to 0 of:

$$\Delta\Lambda_l' = \mathcal{M}_l \Delta\dot{\eta}_l - \mathrm{ad}^T_{\eta_l} \mathcal{M}_l \Delta\eta_l - \mathrm{ad}^T_{\Delta\eta_l} \mathcal{M}_l \eta_l + \mathrm{ad}^T_{\xi_l} \Delta\Lambda_l + \mathrm{ad}^T_{\Delta\xi_l} \Lambda_l - \Delta\overline{F}_{\mathrm{ext},l}, \tag{3.51}$$

  starting from initial conditions $\Delta\Lambda_l(\ell_l) = -\Delta F_l$.

In these ODEs, $\Delta\xi_l = A_l \Phi_l \, \Delta q_{\epsilon,l}$, while $\Delta\eta_l$, $\Delta\dot{\eta}_l$, and even $\Delta\zeta_l$ (which can be required by $\Delta\bar{F}_{\mathrm{ext},l}$), are all deduced by varying (3.43), that provides formulae similar to (3.47) with $(l, j)$ instead of $(j, k)$. Once all the $\Delta F_j$s and $\Delta\Lambda_j(.)$s known, one can calculate the outputs of the *TIDM i.e.,*:

$$\Delta Q_{ar,j} = A_j^T \Delta F_j, \qquad \Delta Q_{a\epsilon,j} = -\int_0^{l_j} \Phi_j^T A_j^T \Delta\Lambda_j dX, \tag{3.52}$$

depending whether it is lumped (rigid) or distributed (soft). Finally, if $\mathcal{B}_0$ is a free floating basis, $\Delta F_a = \Delta F_0$, and the algorithm feeds back all the outputs $(\Delta F_a^T, \Delta Q_{a,r}^T, \Delta Q_{a,\epsilon}^T)^T$ of the input-output map (3.24).

### Initialization of tangent backward dynamics

Using the notations of Remark 16, if a terminal body supports a cut, this body can be a leader $\mathcal{B}_p$ or a follower $\mathcal{B}_s$ *i.e.,* one can have $(j, l) = (p, p_+)$ or $(s, s_+)$ in (3.48). Then, since ${}^s g_{s+}$ and ${}^p g_{p+}$ are constant, the initial conditions of (3.48) are of the form $\Delta^p F_{p+} = Ad^T_{{}^{p+}g_p} \Delta F_{p+}$ and $\Delta^s F_{s+} = Ad^T_{{}^{s+}g_s} \Delta F_{s+}$. Thus, we need to calculate $\Delta F_{p+}$ and $\Delta F_{s+}$. Regarding $\Delta F_{s+}$, applying $\Delta$ to the left expression of (3.42), gives first:

$$\Delta F_{s+} = J^T_{s|p} \Delta\lambda_{s|p} + (\Delta J_{s|p})^T \lambda_{s|p}, \tag{3.53}$$

with $\Delta\lambda_{s|p}$ a sub-vector of $\Delta\lambda$, and where we used the variation of the Jacobian matrix (3.35):

$$\Delta J_{s|p} = \overline{A}_{p+} \begin{bmatrix} \Delta T(\Theta_{s|p})^{-1} & 0_{3\times3} \\ 0_{3\times3} & {}^{p+}R_{s+} T(\Theta_{s|p}) \end{bmatrix}, \tag{3.54}$$

with $\Delta T$ the second differential of the exponential map of $SO(3)$ [50]. Regarding $\Delta F_{p+}$, using the properties of Ad and ad, the variation $\Delta$ applied to the right expression of (3.42) gives:

$$\Delta F_{p+} = -\mathrm{Ad}^T_{{}^{s+}g_{p+}} (\Delta F_{s+} - \mathrm{ad}^T_{\Delta\zeta_{s+/p+}} F_{s+}), \tag{3.55}$$

which involves the relative variation:

$$\Delta\zeta_{s+/p+} = \Delta\zeta_{s+} - \mathrm{Ad}_{s+g_{p+}}\Delta\zeta_{p+}, \tag{3.56}$$

where $\Delta\zeta_{s+} = Ad_{s+g_s}\Delta\zeta_s$, $\Delta\zeta_{p+} = Ad_{p+g_p}\Delta\zeta_p$ are deduced from $\Delta\zeta_s$ and $\Delta\zeta_p$ computed by the forward tangent kinematics (3.45). Therefore, if $(j,l) = (s,s_+)$, (3.48) is initialized with (3.53,3.54), and if $(j,l) = (p,p_+)$, with (3.55,3.56), both being function of $(\lambda,\Delta\lambda)$. At last, if a terminal body of a branch does not support any virtual cut, we simply take $\Delta^j F_l = 0_{6\times 1}$ as initial condition.

## 3.7 Calculation of $\overline{\mathcal{R}}^{(n)}$ and $(\partial\overline{\mathcal{R}}^{(n)}/\partial\overline{\chi})$

Referring to Remark 11, the calculation of $\overline{\mathcal{R}}^{(n)}(\overline{\chi},t_{n+1})$ and its Jacobian matrix, *i.e.*, $(\partial\overline{\mathcal{R}}^{(n)}/\partial\overline{\chi})(\overline{\chi},t_{n+1})$, requires that of $\overline{\mathcal{R}}^{(n)}_\lambda = \Upsilon(g_0,q)$ along with the vectors $\Xi(q,t_{n+1})$ and $\Delta\Xi(q,\Delta q,t_{n+1})$ in (3.20) and (3.22). These additional computations are by-products of $IDM$ and $TIDM$ which we will now detail.

### 3.7.1 Calculation of $\overline{\mathcal{R}}^{(n)}_\lambda = \Upsilon(g_0,q)$

For each loop virtually cut between $\mathcal{B}_p$ and $\mathcal{B}_s$, one can compute a vector of the type $\Upsilon_{p|s}$ by feeding the general formula (3.33) with ${}^{p+}g_{s_+} = g_{p+}^{-1}g_{s+}$ numerically calculated by the forward kinematics (3.36) along each branch of the SMMS. Note that each of the $\Upsilon_{p|s}$-vectors defines a set of non-linear algebraic equations depending in general on $(g_0,q)$. Gathering all of these $\Upsilon_{p|s}$-vectors, provides the expected $\Upsilon$ of the original system of DAEs (3.1).

### 3.7.2 Calculation of $\Xi(t_{n+1})$ and $\Delta\Xi(t_{n+1})$

To finish the computation of the residual vector (3.20), it remains to compute $\Xi(t_{n+1}) = Q_{act}(t_{n+1}) - Q_e$. As regards $Q_e$, we have $Q_e = (0^T_{1\times n_r}, Q^T_\epsilon)^T$, with $Q_\epsilon$ obtained by gathering the vectors of $\mathbb{R}^{n_{\epsilon,j}}$:

$$Q_{\epsilon,j} = K_{\epsilon\epsilon,j}\, q_{\epsilon,j}, \tag{3.57}$$

for all soft $\mathcal{J}_j$s, with $K_{\epsilon\epsilon,j}$ defined in Section 2.3.6. Remarking that $K_{\epsilon\epsilon,j} \in \mathbb{R}^{n_{\epsilon,j}\times n_{\epsilon,j}}$ is constant, we have $Q_\epsilon = K_{\epsilon\epsilon}q_\epsilon$ with $K_{\epsilon\epsilon} = \mathrm{diag}(K_{\epsilon\epsilon,j})$ is the constant stiffness matrix of the entire structure. As regards $Q_{act}$, we have $Q_{act}(t_{n+1}) = (\tau_d^T, Q^T_{act\epsilon})^T(t_{n+1})$, where $Q_{act\epsilon}(t_{n+1})$ is obtained by gathering the vectors $Q_{act\epsilon,j} \in \mathbb{R}^{n_{\epsilon,j}}$, defined in Section 2.3.6 which we recall here as:

$$Q_{act\epsilon,j}(t_{n+1}) = -\int_0^{\ell_j} \Phi_j^T A_j^T \Lambda_{act,j}(t_{n+1})dX, \tag{3.58}$$

for all soft $\mathcal{J}_j$s. Similarly, one can compute $\Delta\Xi(t_{n+1}) = \Delta Q_{act}(t_{n+1}) - \Delta Q_e$ by differentiating (3.57) and (3.58). The former variation is merely $\Delta Q_\epsilon = K_{\epsilon\epsilon}\Delta q_\epsilon$, while the latter is $\Delta Q_{act} = (0_{1\times n_r}, \Delta Q^T_{act\epsilon}(t_{n+1}))^T$, with:

$$\Delta Q_{act\epsilon,j}(t_{n+1}) = -\int_0^{\ell_j} \Phi_j^T A_j^T \Delta\Lambda_{act,j}(t_{n+1})dX. \tag{3.59}$$

### 3.7.3 Adaptation of the $IDM$ and $TIDM$ to compute the residual and its Jacobian

Finally, to calculate the two top entries of $\overline{\mathcal{R}}^{(n)}(\overline{\chi}, t_{n+1})$, one can numerically integrate by quadrature (3.57) and (3.58), and remove $\Xi(t_{n+1})$ from $\overline{IDM}(\overline{\chi})$, or more directly, replace in the $IDM$, the left side equation of (3.44) by:

$$\mathcal{R}_j(\chi, t) = A_j^T F_j - \tau_{act,j}(t), \tag{3.60}$$

if $\mathcal{J}_j$ is rigid, and the right side one by:

$$\mathcal{R}_j(\chi, t) = \int_0^{\ell_j} \Phi_j^T A_j^T (\Lambda_{act,j}(t) + \mathcal{H}_j A_j \Phi_j q_{\epsilon,j} - \Lambda_j) dX, \tag{3.61}$$

if it is soft.

Similarly, one can calculate the two top rows of $(\partial\overline{\mathcal{R}}^{(n)}/\partial\overline{\chi})(\overline{\chi})$, by replacing in the $TIDM$, the left-side expression of (3.52) by:

$$\Delta\mathcal{R}_j(\chi, \Delta\chi, t) = A_j^T \Delta F_j, \tag{3.62}$$

if $\mathcal{J}_j$ is rigid, and the right-hand side one by:

$$\Delta\mathcal{R}_j = \int_0^{\ell_j} \Phi_j^T A_j^T (\Delta\Lambda_{act,j}(t) + \mathcal{H}_j A_j \Phi_j \Delta q_{\epsilon,j} - \Delta\Lambda_j) dX, \tag{3.63}$$

if $\mathcal{J}_j$ is soft, where we dropped the dependence on $\chi$, $\Delta\overline{\chi}$ and $t$ for conciseness. Finally, feeding the $IDM$ and $TIDM$, with inputs compatible with the Newmark scheme (3.9,3.12-3.14) and using as outputs (3.60,3.61) and (3.63,3.62) instead of (3.44) and (3.52) for every $\mathcal{J}_j$, allows to compute directly $\overline{\mathcal{R}}^{(n)}$ and $\Delta\overline{\mathcal{R}}^{(n)}$. This defines two further algorithms, noted $\overline{IDM}_\star$ and $\overline{TIDM}_\star$ in Figure 3.2, where the $\overline{IDM}_\star$ also includes the calculation of $\overline{\mathcal{R}}_\lambda^{(n)} = \Upsilon(g_0, q)$ of Section 3.7.1.

## 3.8 Space integration

The algorithm needs to forward (respect. backward) integrate a sequence of space initial value problems (IVPs) defined by (3.38) (respect. (3.41)), along the tree, where the initial conditions of one joint, are fixed by the final conditions imposed by its antecedent body (respect. by its successors). This can be achieved with standard finite-difference integrators as `ODE45` of Matlab [21]. Nevertheless, let us remark that $(\xi_j, \dot{\xi}_j, \ddot{\xi}_j)(.)$ being fixed by the inputs $(q_{r,j}, \dot{q}_{r,j}, \ddot{q}_{r,j})$ of the algorithm, they can be considered as constants in (3.38) which turn to be linear ODEs with respect to $(g_j, \eta_j, \dot{\eta}_j)(.)$. Similarly, once the first pass is achieved, $(g_j, \eta_j, \dot{\eta}_j)(.)$ can be considered as fixed in the backward ODEs (3.41), which become linear with respect to the $\Lambda_j(.)$s. Parameterizing the rotational component $R_j(.)$ of $g_j(.) = (R_j, r_j)(.)$, with a unit quaternion field[5] $H_j(.) : X \in [0, \ell_j] \mapsto H_j(X) \in \mathbb{R}^4$, this

[5] Alternatively, one could use quadrature integration based on Magnus expansion in order to remain in $SO(3)$ as this is done in [99, 100].

property of the $IDM$ allows us to apply efficient spectral methods to the integration of (3.38) and (3.41), as will be detailed in Chapter 5.

These methods can be optimized by exploiting a further property of the $IDM$ related to its differential structure. Indeed, if one details all the fields of (3.38,3.41) into their angular and linear components, with notations: $(R_j, r_j) = (H_j, r_j)$, $\xi_j = (K_j^T, \Gamma_j^T)^T$, $\eta_j = (\Omega_j^T, V_j^T)^T$, $\dot{\eta}_j = (\dot{\Omega}_j^T, \dot{V}_j^T)^T$ and $\Lambda_j = (C_j^T, N_j^T)^T$, it is straightforward to show that each of these two sets of ODEs enjoys a block triangular differential structure which can be solved in cascade, by integrating block after block in an ordered manner, a sequence of linear systems of the canonical state-form:

$$x' = A(X)x + b(X)\ ,\ X \in [0, l_j], \tag{3.64}$$

where for each soft $\mathcal{J}_j$, the state vector $x \in \mathbb{R}^3$ or $\mathbb{R}^4$, successively takes the meaning of $H_j$, $r_j$, $\Omega_j$, $V_j$, $\dot{\Omega}_j$, $\dot{V}_j$ for the forward ODEs (3.38), and of $N_j$, $C_j$ for the backward one (3.41). Respecting this order, at each step of the cascade resolution, one uses the result of the previous step to feed $A$ and $b$ in (3.64).Therefore, owing to this differential property of the $IDM$, spectral integration based on collocation over a Chebyshev grid, allows to change ODEs (3.38,3.41) into algebraic linear systems of triangular structure, that can be solved efficiently with respect to the successive finite dimensional vectors of the values of the fields $(H_j, r_j, \Omega_j, V_j, \dot{\Omega}_j, \dot{V}_j, C_j, N_j)(.)$ at the Chebyshev nodes of the grids. Finally, by adding to the linear backward ODE systems, the expressions (3.61) reformulated as ODEs of the form (3.64) with $A = 0$, one completes the spatial integrations required by the computation of the residual vector $\overline{\mathcal{R}}^{(n)}$, while a strictly similar process can be applied with the $TIDM$ in order to compute $\Delta\overline{\mathcal{R}}^{(n)}$, and finally the expected Jacobian of the residual vector. Before concluding this section, let us note that while all the formulas required by $IDM$ and $TIDM$ were obtained by symbolic calculations on the Lie group $SE(3)$, in their numerical implementation, these expressions are integrated on $SO(3) \times \mathbb{R}^3$ with $SO(3)$ parametrized by usual quaternions, as was done in [31, 27].

## 3.9 Illustrative examples

We now illustrate the above general algorithm on a set of numerical examples and refer the reader to Appendix C for a symbolic illustration. We start by two elementary benches well known from the literature on geometrically exact FEM where they are used to validate new numerical methods in the field [21]. These first examples consists of one single rod in different conditions of connection offered by the algorithm. They are here used to compare several methods recently proposed in soft robotics. Once this is achieved, the section proceeds with two examples more closely related to robotics. Note that, for these examples, the algorithm has been implemented in Matlab. The optimization of the algorithm and its efficient implementation will be detailed in Chapter 5.

Before introducing the example, it is important to point out that these examples are included to show the adaptability of the algorithm, however the ones in Sections 3.9.1 and

3.9.3 have been implemented by Vincent Lebastard while the example is Section 3.9.2 by Philipp Tempel.

### 3.9.1 Single rod benchmarks

We consider single rods in different conditions of connection with the ground. In the first case, a soft and a stiff rod is cantilevered *i.e.,* fixed at one end and free at the other [53]. In the second, another two soft and stiff rods are kinematically free at both ends according to the so-called flying rod bench [35]. Note that these tests are complementary since they illustrate the case of a fixed and a free base system (locomotor and manipulator), while closed loop systems will be illustrated later with a two rods system. These two first benches were used to validate the strain-based parametrization of Section 2.3.6 against GE-FEM in [21] and to study the relationships between the shooting based resolution and optimal control in [40]. We use them here to provide the reader with a very first idea of the robustness and time-consumption of the algorithm in comparison to the explicit time-integration methods of [62], [21] and [78], as well as to the shooting-based approach of [39] and [40] based on implicit integration. Recall that although all these approaches are based on the same model (Cosserat rods), [21] and [78] use the modal strain reduction of the VSA described in Section 2.3.6, while [39, 40] and [62] require no configuration space reduction beyond nodal discretization (based on finite differences in the first two cases, and on DER in the third).

#### Cantilever rod

The soft rod is that of [53] with $\ell = 10\,\mathrm{m}$, $\rho = 7800\,\mathrm{kg.m^{-3}}$, $E = 10^5\,\mathrm{MPa}$, and a circular cross section with 0.01 m diameter. The stiff rod is that of [40], with $l = 0.4\,\mathrm{m}$, $\rho = 8000\,\mathrm{kg.m^{-3}}$, $E = 2 \times 10^3\,\mathrm{MPa}$ and 0.002 m diameter. Each of them is first subject to a horizontal static force $(f, 0, 0)^T$ in the inertial frame, and released in the vertical gravity at $t = 0\,\mathrm{s}$. In Figure 3.5, we reported a set of snapshots simulated with $f = 50\,\mathrm{N}$ over 10 s, and $f = 10\,\mathrm{N}$ over 1 s, for the soft and stiff rods respectively. Simulations based on the strain parametrization are performed with the Kirchhoff model and 3 bending modes. Note that the results of all methods that converged are indistinguishable.

#### Flying rod

Each of the two rods floats in vacuum and without gravity. They are initially inclined and loaded in the inertial plane $(o, e_1, e_2)$ as indicated in Figure 3.6 (a) and (b). The soft beam is that of Simo [35] *i.e.,* $\ell = 10\,\mathrm{m}$, stiffness matrix $\mathcal{H} = \mathrm{diag}(5, 5, 5, 10^2, 10^2, 10^2)10^2$, and inertia-tensor $\mathcal{M} = \mathrm{diag}(10, 10, 10, 1, 1, 1)$ (all in IF units). The stiff rod parameters are those of [40], i.e $\ell = 1\,\mathrm{m}$, $\rho = 10^3\,\mathrm{kg.m^{-3}}$, $E = 1\,\mathrm{MPa}$, for a diameter of 0.1 m. Strain parametrization is performed with the Kirchhoff model with 3 modes for the two curvatures and for torsion. Figures 3.6 (b) and 3.6 (c) show a few snapshots of the two rods simulated over 10 s. Note that the results of all the methods that converged are indistinguishable. With the exception of PyElastica [62], which is programmed in Python, all other methods

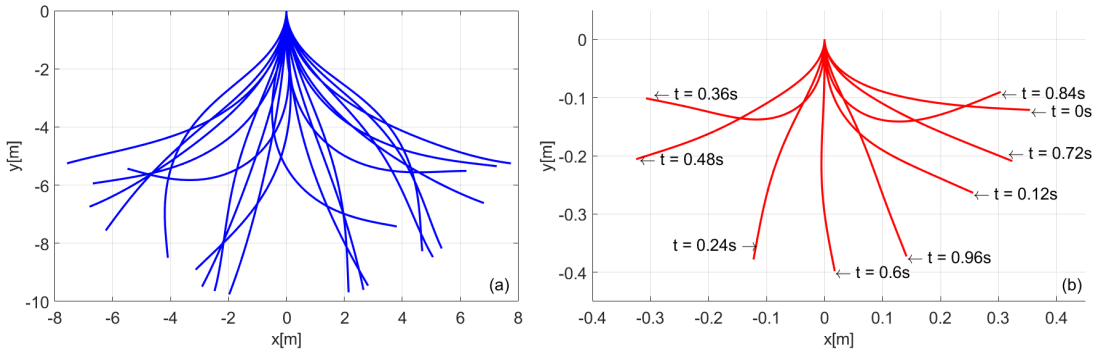



Figure 3.5: Clamped soft (a) (blue) and stiff (b) (red) beam released after bending with a horizontal tip force $f = 50\,\mathrm{N}$ and $10\,\mathrm{N}$ respectively. (a) snapshots every $0.5\,\mathrm{s}$ for $10\,\mathrm{s}$ of simulation. (b) selected snapshots for $1\,\mathrm{s}$ of simulation.

were implemented in Matlab. All simulations were run on a single computer[6]. In addition, the tuning parameters of each method (time and space-steps, Chebyshev grid, quadrature order, etc.) were chosen in such a way as to obtain an accuracy similar to that of the GE-FEM, without penalising any of them in terms of efficiency. In details $\Delta t = 0.01\,\mathrm{s}$ for the implicit integrators of [40] and that of the paper, tolerance of `ODE45` in the time-integration of [21] and [78] is $10^{-6}$, while the spatial grids of [78], PyElastica and the present algorithm have $20$ , $50$ and $20$ discretization points respectively. Finally, note that in the case of soft rods, the shooting approach of [39] as well as its extension to floating base systems of [40], fail for intrinsic reasons related to the singular character of the underlying optimal control problem from which the spatial BVP is derived [40].

| Test▼ \| Method ▶ | Shooting[39, 40] | PyElastica[62] | VSA[21] | SoroSim[78] | Ours |
|---|---|---|---|---|---|
| Stiff cantilever rod | 71.9 | 336 | 6173 | 4.5 | 7.1 |
| Soft cantilever rod | fails | 133 | 1675 | 1.9 | 9.1 |
| Stiff flying rod | 52.2 | 36 | 2806 | 7.2 | 8.5 |
| Soft flying rod | fails | 1.6 | 609 | 1.8 | 7.9 |

Table 3.1: "Simulation-time over real-time" ratio, for cantilever and flying rod benches obtained with the Shooting method [39, 40], PyElastica [62], the VSA [21], SoroSim [78] and the algorithm of the chapter.

As expected, the simulation times for methods based on explicit integration increase with stiffness since in this case, the time step must be reduced to preserve stability. In contrast, the chapter's implicit integrator can be run with a large time step in both cases, which explains why simulation time is independent of rod stiffness. By way of illustration, the critical time step beyond which PyElastica is no longer stable is $3.4 \times 10^{-6}$ s for the stiff cantilevered rod, and must be divided by $\sqrt{\alpha}$ when $E$ is multiplied by $\alpha \in [0.01, 100]$,

[6] A PC dell (DESKTOP-5F02EKM), Processeur Intel(R) Core(TM) i9-9880H CPU @ 2.30 GHz, RAM 32.0 Go)

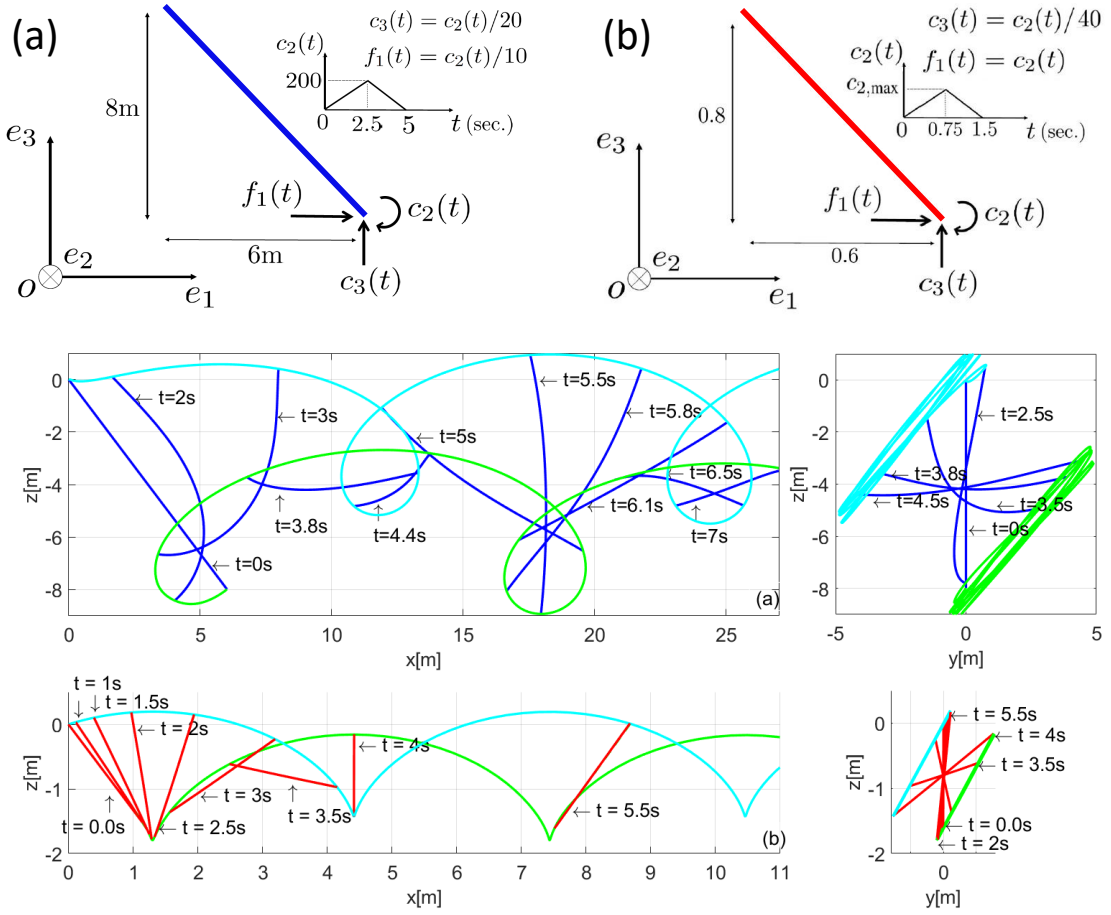



Figure 3.6: Top: Conditions of the test for the soft (a) (blue) and stiff (b) (red) rods. Snapshots of the soft (middle) and stiff (bottom) flying rod projected on $x - z$ (left), and $y - z$ (right) planes.

whereas our implicit integrator remains stable over this range of stiffness with a time step of 0.1 s. Finally, it should be remembered that these initial estimates of calculation times are far from definitive, and will be systematically studied in the future. For the moment, they tend to show that the extra computations of the Jacobian of the residual vector required by implicit integration are well compensated by the width of the time steps it allows, as it is often the case in non-linear structural dynamics [101].

### 3.9.2 Example 1: A mass attached to a rod moved by two flying drones at its two ends

We consider the case of a locomotor, namely a cuboid payload carried by two identical hexarotor flying drones through two identical rods. The ends of the two rods are connected by fixed joints to the drones (UAVs), and to the payload. They are modelled as 3D

rigid bodies subject to a lift force of controlled direction and strength. According to our general setting, this SMMS is segmented as indicated in Figure 3.7 *i.e.,* it contains seven rigid bodies $\mathcal{B}_{0,1,\dots,6}$, namely three 3D ones and the four tip cross-sections of the rods. These bodies are connected by six joints $\mathcal{J}_{1,2,\dots,6}$, namely four fixed lumped ones and two distributed soft ones. One of the two drones is considered as the reference body $\mathcal{B}_0$, and all the other rigid bodies are numbered increasingly along this open chain. The dynamics of the UAV rotors are assumed to be fast enough to impose instant velocity control of external wrenches:

$$j = 0, 6: \qquad F_{\text{ext},j} = k_P\, e_{\eta_j} + k_I \int e_{\eta_j}\, dt, \tag{3.65}$$

with $k_P$ and $k_I$ two diagonal $6 \times 6$ matrices of proportional and integral gains respectively, and $e_{\eta_j} \in \mathbb{R}^6$ the velocity error defined by:

$$e_{\eta_j} = \text{Ad}_{(g_j^{-1} g_j^d)} \eta_j^d - \eta_j, \tag{3.66}$$

with $t \mapsto \eta_j^d(t) = ((g_j^d)^{-1} \dot{g}_j^d)^\vee$ a desired velocity time-evolution. In figures 3.8, 3.9, we reported the results of a simulation of this system obtained with $\eta_j^d(t) = (0_{1\times 3}, V_j^{dT}(t))$ and $V_0^d(t) = (0, 0, \sin(\pi t))^T$, $V_6^d(t) = (0, 0, \sin\left(\frac{\pi}{2} t\right))^T$. The simulation is performed over 10s with a time-step $\Delta t = 0.01\,\text{s}$ and requires on average, 3 Newton's steps per time-step. The two rods $\mathcal{J}_2$ and $\mathcal{J}_5$, are Kirchhoff, straight at rest, and parametrized with 3 Chebyshev polynomials for each of the three components (two curvatures and torsion) of $\epsilon_j = K_j$, $j = 2, 5$. Their geometrical and physical parameters are $\ell = 1\,\text{m}$, $\rho = 950\,\text{kg.m}^{-3}$, diameter $= 0.07\,\text{m}$, $E = 0.25\,\text{MPa}$. For the sake of simplicity, gravity is neglected, while we did not seek to optimize the controllers but rather to reveal the richness of the dynamic interactions between the drones and the rods. To this end, taking $(k_P, k_I) = (\text{diag}(20, 20), \text{diag}(2.5, 2.5))$ in (3.65), defines a soft controller that keeps the velocity errors limited, while not reacting too much to the elastic restoring torques induced by the rods on the two drones. In Figure 3.8, snapshots and trajectories of the two drones and the payload $\mathcal{B}_3$ show how the rods move $\mathcal{B}_3$ on an almost cyclic circular path. The time evolution of the displacement of these 3 bodies is displayed in Figure 3.9.

### 3.9.3 Example 2: Tendon actuated bio-inspired swimmer

In this second example, we consider a bio-inspired locomotor, namely an undulatory swimmer of 1m length and non-uniform elliptical cross-sections along its rostro-caudal axis. Its geometry is that of [102] with maximum lateral width of 0.1 m. However, in contrast to [102], we here consider it as being constituted of three segments serially fixed to each other from the head to the tail (see Figure 3.10). The first segment (the head) is a rigid body, the second is a rod internally actuated by a pair of antagonistic tendons as those currently used for TACRs [27], the third is a passive elastic rod. The swimmer is neutrally buoyant *i.e.,* $\rho = 10^3\,\text{kg.m}^{-3}$, and the first and second rod are Kirchhoff with a bending stiffness $EI = 4.9\,\text{Pa.m}^4$ and $EI = 4.9 \times 10^{-2}\,\text{Pa.m}^4$ respectively. The tendons are parallel to the robot axis and located at a distance $D = 0.05\,\text{m}$ from it. According to our general setting, this system is segmented in 5 bodies $\mathcal{B}_{0,1,2,3,4}$, with $\mathcal{B}_0$, a 3D rigid body modelling the head,

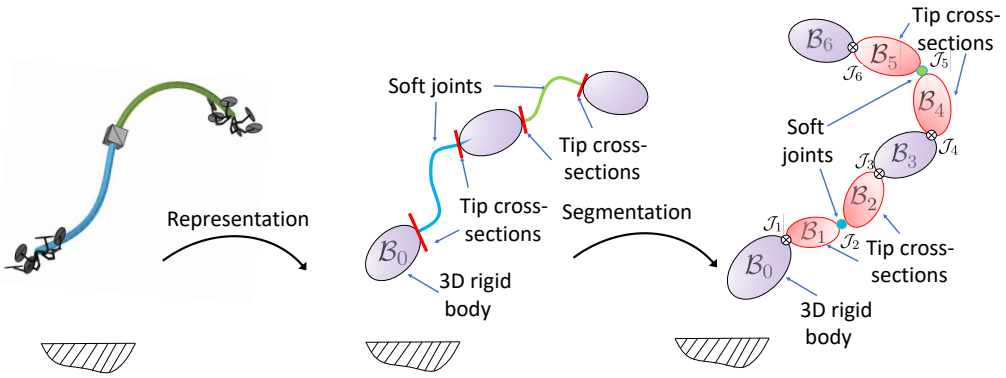



Figure 3.7: Segmentation of a cuboid mass attached to two rods moved by two flying drones. The two Cosserat rods are declared as joints $\mathcal{J}_2$ and $\mathcal{J}_5$, respectively. Their two tip-cross sections are declared as rigid bodies $(\mathcal{B}_1, \mathcal{B}_2)$ and $(\mathcal{B}_4, \mathcal{B}_5)$, respectively.

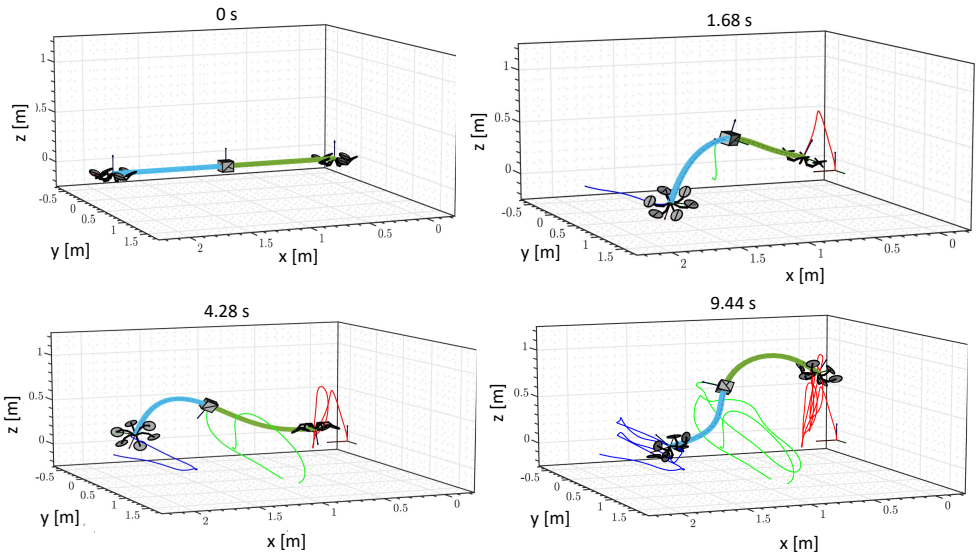



Figure 3.8: From left to right and top to bottom: Snapshots and trajectories of the two drones and the cuboid payload between $t = 0\,\mathrm{s}$ and $t = 1.68$, 4.22, and 9.44 s

and all the others standing for the rod's end cross-sections. All these bodies are connected by four joints $\mathcal{J}_{1,2,3,4}$, with $\mathcal{J}_{1,3}$ two lumped fixed joints, and $\mathcal{J}_{2,4}$ two distributed soft ones. The hydrodynamic forces are modelled with Lighthill's Large Amplitude Elongated Body Theory (LAEBT), here coupled to the Cosserat model as proposed in [102]. The planar swimming of this bio-inspired system can be achieved by applying to the two tendons, the tensions:

$$T_{\pm}(t) = T_0 + T_1 \sin\left(\omega t \pm \frac{\pi}{2}\right), \tag{3.67}$$

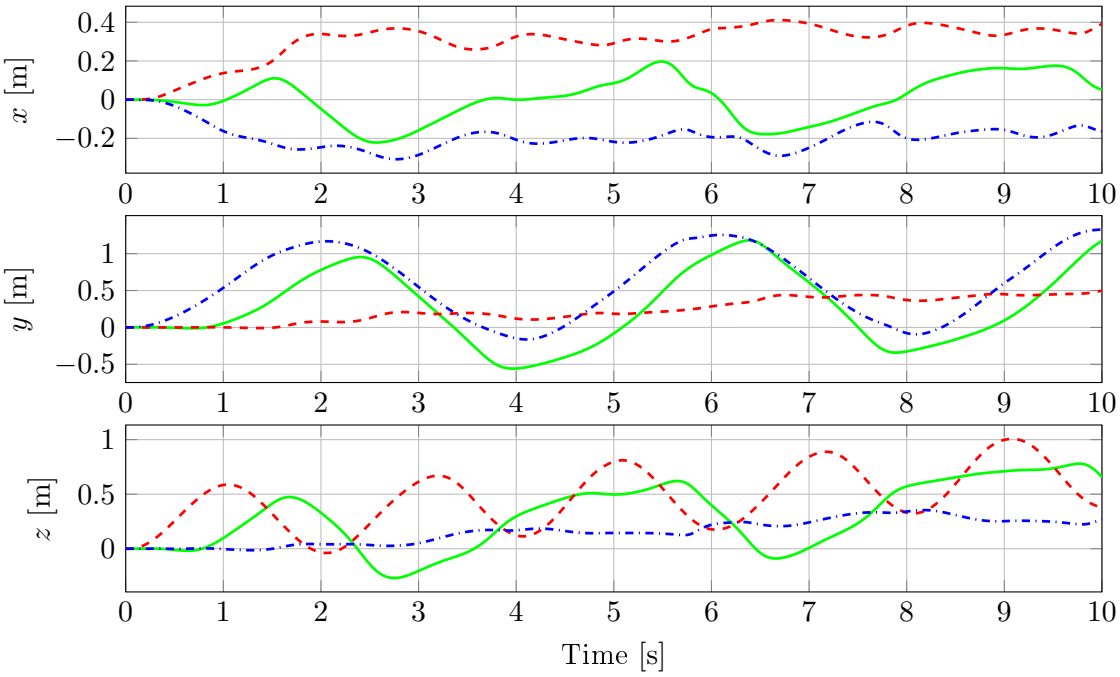



Figure 3.9: Displacements of $\mathcal{B}_0$ (solid green), $\mathcal{B}_3$ (dashed red) and $\mathcal{B}_6$ (dashdotted blue) vs time in the inertial frame, projected onto the $x$ (top), $y$ (middle) and $z$-axis (bottom).

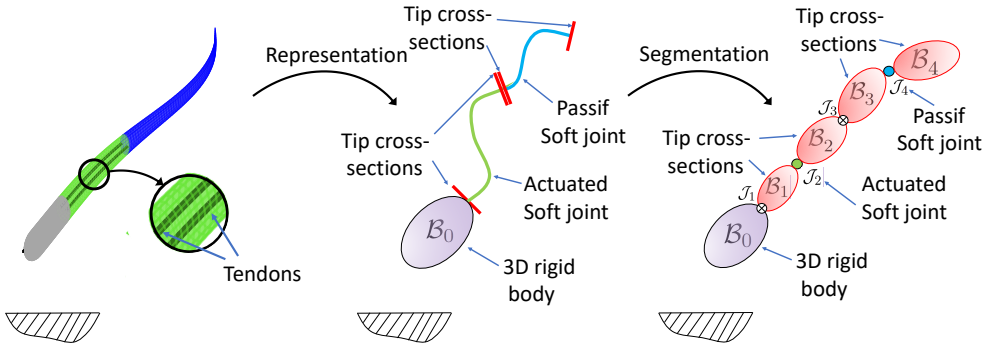



Figure 3.10: Segmentation of a bio-inspired swimmer actuated with tendons. The two Cosserat rods are declared as joints $\mathcal{J}_2$ and $\mathcal{J}_4$, respectively. Their two tip-cross sections are declared as rigid bodies $(\mathcal{B}_1, \mathcal{B}_2)$ and $(\mathcal{B}_3, \mathcal{B}_4)$, respectively.

where indexes $+$ and $-$ denote the left and right tendons respectively, $T_0 > 0$ is a constant antagonistic component ensuring that $T_\pm(t) > 0$ for all $t$, and $T_1 > 0$ is the amplitude of an harmonic component of pulsation $\omega$, responsible of the body undulations. In figures 3.11 and 3.12, we reported some simulation results carried out over 10 s with $\Delta t = 0.01$ s, while using 5 modes, *i.e.*, $n_{e,i} = 5$, for the Chebyshev polynomial prametrizing each rod curvature. The results are obtained by using the control law (3.67) in the model (2.32),

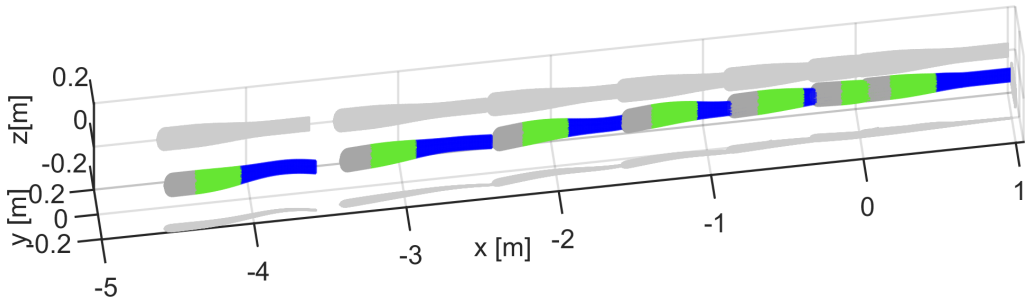



Figure 3.11: Snapshots every 1.65 s of a continuum undulatory swimmer internally controlled with a single pair of tendons.

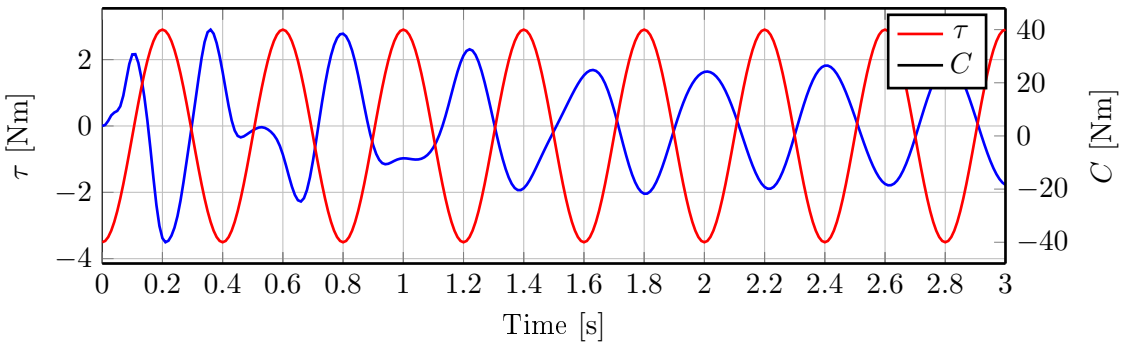



Figure 3.12: Time evolutions of control $\tau(t) = D(T_+ - T_-)(t)$ and elastic $C = (EI_3K_{Z,3})(X = 0)$ torques along the first 3 s of swim.

with $T_0 = T_1 = 20\,\mathrm{N}$, and $\omega = 5\pi$ RAD/s. In Figure 3.11, snapshots of the robot are drawn every 1.65 s. Figure 3.12 shows the time evolution (over the first 3 s) of the control torque $\tau(t) = D(T_+ - T_-)(t)$, and of the elastic torque $C = (EI_3K_{Z,3})(0)$ at the root of the third (elastic) body $\mathcal{B}_3$, where the subscript $Z$, indicates a component taken along the normal to the plane of the swim.

## 3.10 Conclusions

In this chapter, we have proposed a new algorithm for solving the forward dynamics of soft and continuum robots consisting of rigid bodies connected in arbitrary topologies by localised joints and/or soft links, possibly actuated or not. The soft, or continuum, joints are modelled as Cosserat rods parametrized with strain modes. Unlike the literature approaches for simulating these models, our approach is designed to work with an implicit time integration scheme. We choose the Newmark scheme which is known to be among the best suited for stiff problems of non-linear structural dynamics. Using such a scheme allows changing the DAEs of the robot into a system of non-linear algebraic equations. At each time-step, we solve this system with a Newton-Raphson scheme, requiring the residual and its Jacobian. To this aim we proposed a new recursive Newton-Euler algorithm,

which, at its highest level, works indifferently on rigid and continuum joints. Although the results are encouraging, the current implementation of the approach remains at the proof-of-concept stage. It has been tested on several classical soft robotics benchmarks, including clamped and flying rods, where it was compared against other software and GE-FEM, demonstrating both efficiency and accuracy. Further examples, more relevant to robotics, were also explored, such as the simulation of a drone-driven soft manipulator and a bio-inspired swimmer[7].

Moving forward, the next step will be to specialize this approach for CPRs and subsequently focus on its optimization and more generic implementation.

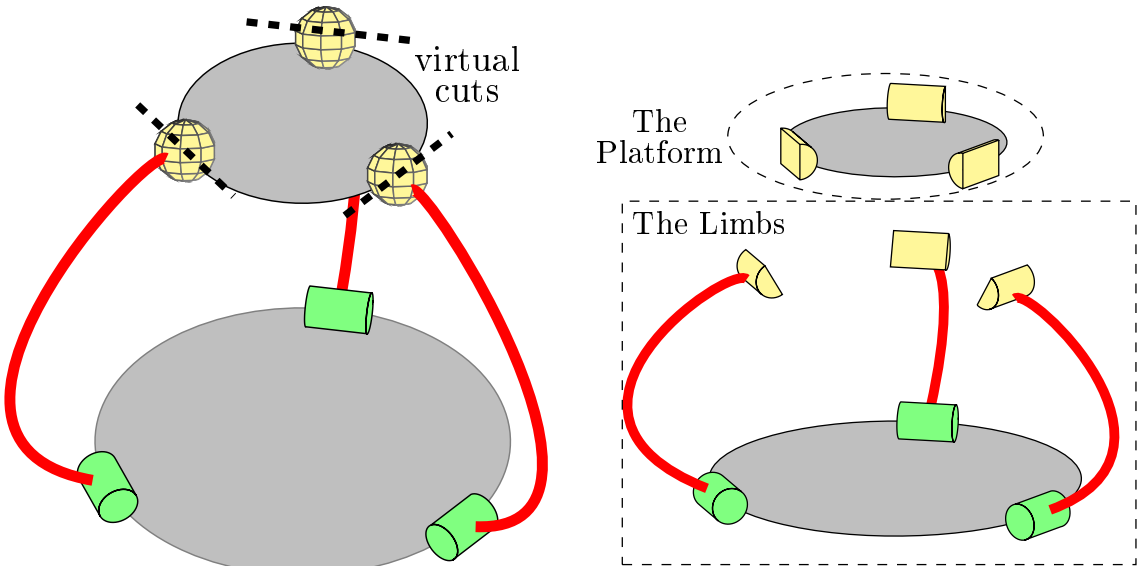
[7]These examples were implemented by Philipp Tempel and Vincent Lebastard, respectively.

# Chapter 4

# Dynamics Modelling and Experimental Validation of CPRs

As discussed in the Introduction (Chapter 1), different CPRs exist in the literature. Here we focus on those consisting of identical limbs attached to a fixed base and a rigid platform, with a set of proximal and distal joints, respectively (Figure 4.1a). The joints considered are clamped, revolute or spherical. Other types, such as prismatic, cylindrical[1], universal[2] and screw joints, are not considered here, but can be easily tackled by the approach. Moreover, it is assumed that the limbs are only connected together through the base and platform, *i.e.,* there is no mechanism linking them together. The approach from Chapter 3 used virtual cuts to open all the kinematics loops of a SMMS. In this case, this result in leaving one limb still attached to the platform (Figure 4.1a). Being all limbs identical, this leaves with the ambiguity of which limbs to use. Morover, this limb is connected to the platform through a spherical joint (Figure 4.1a), the corresponding generalized coordinates $q_r$ represent the three degrees of rotation incurring in possible singularities of representation. For these reasons, the approach from Chapter 3 will be properly adapted to the CPRs, emplyoing the virtual cuts to fully isolate the platform. In the terminology of multi-body systems, this corresponds to virtually cutting all the distal joints, effectively making the platform a floating rigid body and the limbs a tree-like system. These two subsystem will be denoted as *the platform* and *the limbs*. Given the system's symmetry, isolating the platform is a logical approach and is standard practice in rigid parallel robots [103]. Moreover, breaking this symmetry by attaching the platform to one of the limbs would be impractical, as it introduces ambiguity in selecting which of the identical limbs to attach it to. Then, the NE passes will run on each of the two sub-system to obtain the CPR dynamics model. The resulting set of DAEs will be time integrated with an implicit scheme that dampens numerical noises that typically affect this kind of stiff system. Finally, the chapter concludes with the experimental validation of the proposed approach on two planar CPR prototypes. The first is used to compare simulated behaviour with recorded data,

[1] Note that a cylindrical joint can be modelled by the concatenation of a prismatic and a revolute joint.
[2] An universal joint can be modelled by two revolute joints, *i.e.,* as a 2R joint.




(a) Generic representation of a CPR with it components: its rigid bodies (gray), the rigid joints (green), the deformable rods (blue) and the virtually cut joints (yellow).

(b) Representation of the limbs and platform subsystems with the virtual cuts of the distal joints.

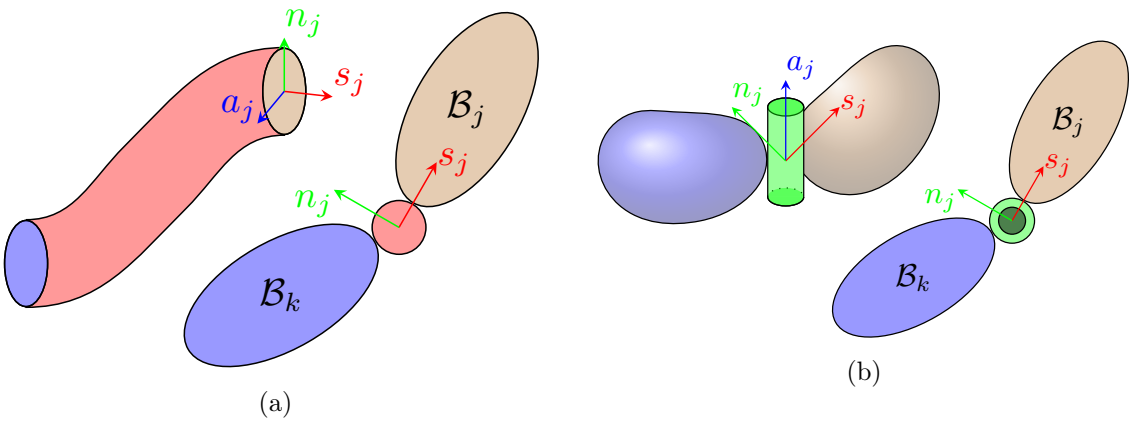
Figure 4.1: Representation of the CPR of interest (left) and illustration of the subsystem of the limbs and the platform (right).

demonstrating consistency in the results, while the second prototype is used to demonstrate the approach's ability to capture stable-unstable-stable transitions.

In the following, we will outline the modelling of the two subsystems: the platform and the limbs. The closure constraints that enforce the connection between these subsystems will also be explained. Next, the dynamics model of CPRs will be introduced, followed by its numerical resolution using a time implicit scheme and the NE passes. Finally, we will address the experimental validation on the two planar CPR prototypes.

## 4.1 Model of the limbs

The subsystem of the limbs is composed by $n_l$ identical limbs attached to the CPR base. Each limb, numbered by the index $i$, with $i = 1, \ldots, n_l$, is a serial kinematic chain attached to the CPR base. Descending from the base, it consists of two components only: a proximal rigid joint and a deformable body, as the distal joint is virtually cut. Each limb is also equipped with an actuation mechanism, which may act locally through the proximal joint, or/and in a distributed way, along the deformable body as in the case of tendon actuated continuum robots [13] and tendon actuated CPRs [104]. We now describe these components in more details and introduce the parametrization of their kinematics.




(a) (b)

Figure 4.2: Representation of a Cosserat rod with corresponding continuum joint, connecting the base (blue) and tip (brown) cross sections, (a) and a rigid revolute joint (b).

### 4.1.1 Limbs rigid Joints

With our choice of virtual cuts, the only rigid joints considered in our model of the CPR are the proximal ones, that belong to the limbs subsystem and connect the base to the rods. Following the conventions of Section 3.5.2, each joint denoted $\mathcal{J}_j$, is provided with a frame $\mathcal{F}_j$ positioned on the center of the joint $O_j$ with its unit vectors $(s_j, n_j, a_j)$ materially attached to the supported body, if the joint is revolute $a_j$ is supported by its rotation axis (Figure 4.2a). Any rigid joint $j$ has the matrix $A_j$ of Section 2.3.6 defined accordingly to its DoFs, which are parametrized by a vector of generalized coordinates denoted $q_j \in \mathbb{R}^{n_a}$ as described in Chapter 3 but here dropping the "$r$" suffix for conciseness.

### 4.1.2 Limbs deformable Bodies

Each of the the limbs deformable body is enough slender to be modelled as a Cosserat rod (Figure 2.1a). As described in Section 3.5.2, the configuration of a rod $j$ can be represented by the field of the inertial pose $g_j(X)$ of its cross-sectional frames $\mathcal{F}(X)$, while its kinematics is characterized by a set of coordinates $(q, \dot{q}, \ddot{q})_j$, which represent the generalized elastic coordinates, velocities, and accelerations of the rod. Note that, again, we dropped the suffix "$\epsilon$" used in Chapter 3.

### 4.1.3 Limbs dynamics

Finally, based on these choices, the configuration state of the CPR limbs subsystem is parametrized by the vector $q = (q_1^T, q_2^T, \ldots, q_{n_l}^T)^T$, which contains all the generalized (or configuration) coordinates of every rigid joint and deformable body of each limb, numbered with a depth-first approach. With this definition of its configuration space, the dynamics of the limbs subsystem takes the usual Lagrangian form:

$$Q_{act}(q,t) = M(q)\ddot{q} + Q_v(q,\dot{q}) + Q_{\text{ext}}(q) - Q_c(q), \tag{4.1}$$

where $M = \text{diag}(M_1, \ldots, M_{nl})$ is the inertia matrix of the limbs subsystem with $M_i$ that of the limb $i$. In the same manner, we identify $Q_{act} = (Q_{act,1}^T, Q_{act,2}^T, \ldots, Q_{act,nl}^T)^T$ and $Q_v = (Q_{v,1}^T, Q_{v,2}^T, \ldots, Q_{v,nl}^T)^T$ are the vectors of generalized actuation forces and that of Coriolis and centrifugal forces respectively, while $Q_{\text{ext}}(q) = (Q_{\text{ext},1}^T, Q_{\text{ext},2}^T, \ldots, Q_{\text{ext},nl}^T)^T$ is the vector of generalized forces from external actions such as gravity. Finally, we denote with $Q_c = (Q_{c,1}^T, Q_{c,2}^T, \ldots, Q_{c,nl}^T)^T$ the vector of generalized contact forces exerted by the platform onto the limbs through the (virtually cut) distal joints, which will be parametrized by a minimal set of independent Lagrange multipliers.

## 4.2 Model of the platform

With our choice of virtual cuts, the platform is considered as a floating rigid body provided with a body frame $\mathcal{F}_p$ with $O_p$ positioned at the mass center of the platform, and $s_p$, $n_p$ and $a_p$ aligned with its principal axis of inertia. The platform subsystem also includes the $n_l$ fixed joints and fictitious rigid bodies result of the virtual cuts. Numbering each of these fixed joints with the index $i$ of the limb they connect, they are provided with a frame $\mathcal{F}_{p_i}^+$ positioned at the geometric center of the joint, whose basis is rigidly attached to the platform. As a result, the pose of $\mathcal{F}_{p_i}^+$ w.r.t the platform frame $\mathcal{F}_p$ is defined by a constant transformation ${}^p g_{p_i+}$ fixed by the design (see Figure 4.1b).

### 4.2.1 Platform kinematics

The platform kinematic state is parametrized with $g_p$, $\eta_p$ and $\dot{\eta}_p$. As the platform is considered as a floating body, its kinematics state will be assigned by the implicit scheme directly, which will be detailed in Section 4.7.1.

### 4.2.2 Platform dynamics

The configuration space of the platform subsystem being $SE(3)$, the contact forces and torques are modelled by a set of $n_l$ wrenches. These wrenches are exerted on the platform by the limbs, through the distal joints. In details, each (virtually cut) distal joint $i$ transmits to the platform a wrench $F_{p_i}^+$, parametrized by a vector of Lagrange multipliers $\lambda_i$ (which corresponds to $F_p^+$ of Equation (3.42)). The resultant of these contact wrenches onto the platform, noted $F_{p,c}$ ($c$ for "contact" or "constraints"), is defined as:

$$F_{p,c} = \sum_{i=1}^{n_l} \text{Ad}_{{}^{p_i+}g_p}^T F_{p_i+} \tag{4.2}$$

where Ad is the usual adjoint map on $SE(3)$ defined in Appendix A and ${}^{p_i+}g_p = {}^p g_{p_i+}^{-1}$. Having defined the platform configuration space by $SE(3)$, its dynamics is naturally given by the usual Newton-Euler model of a rigid body, detailed in Section 3.5.3 which we recall here as:

$$\mathcal{M}_p \dot{\eta}_p - \text{ad}_{\eta_p}^T \mathcal{M}_p \eta_p - F_{\text{ext},p} + F_{p,c} = 0, \tag{4.3}$$

## 4.3 Model of Closure Constraints

As discussed in Chapter 3, when a system presents closed loops we require to open those loops with some virtual cuts, which introduce some geometric constraints and the contact wrenches enforcing those constraints. In this section, we specialized the subject to the case of a CPR.

### 4.3.1 Frames for virtual cuts

At the level of the virtually cut (distal) joints $\mathcal{J}_{p_i}$ $i = 1, \ldots, n_l$, two additional frames are introduced: one attached to the platform side, already introduced in Section 4.2 and noted $\mathcal{F}_{p_i}^+$, the other attached to the distal tip of the $i^{\text{th}}$ limb, simply noted $\mathcal{F}_i^+$ (see Figure 4.1b). In any CPR configuration compatible with the joint connections, the origins $O_{p_i}^+$ and $O_i^+$ of $\mathcal{F}_{p_i}^+$ and $\mathcal{F}_i^+$ coincide with the center of joint $\mathcal{J}_i$, with $a_{p_i}^+$ and $a_i^+$ supported by the joint axis, if it is revolute. If the distal joints are spherical, only the origins of $\mathcal{F}_{p_i}^+$ and $\mathcal{F}_i^+$ must coincide, while their respective base is that of the rigid body to which they are attached (base of the platform in one case, base of the tip section in the other).

### 4.3.2 Geometric form of Constraints

For the purpose of numerical simulation, the platform and limbs subsystems are reassembled by adding to their dynamics (4.1) and (4.3) a set of constraints imposed by the distal joints, which we recall here as:

$$\Upsilon_i(g_p, q) = \overline{A}_i \begin{bmatrix} \log\left(R_{p_i+}^T R_{i+}\right) \\ {}^{p_i+}r_{i+} \end{bmatrix} = 0_{\bar{n}_a \times 1}. \tag{4.4}$$

where $\overline{A}_i$ is the constrained DoF matrix defined in Section 3.5.6. Considering all the distal joints, $\Upsilon = (\Upsilon_1^T, \Upsilon_2^T, \ldots \Upsilon_{n_l}^T)^T \in \mathbb{R}^{\overline{n}_a n_l}$ represents the loop closure constraint equations of the entire CPR, in their geometric form as:

$$\Upsilon(g_p, q) = 0_{(\overline{n}_a n_l)\times 1}. \tag{4.5}$$

### 4.3.3 Kinematic form of Constraints

As discussed in Section 3.1 the two subsystem dynamics (4.1) and (4.3,4.2) will be reassembled by a set of Lagrange multipliers in charge of forcing the constraints (4.5). To achieve this, the mapping from the dual vector space of the constraint parametrization to the space of contact wrenches $F_{p_i+}$, as well as to the generalized contact forces exerted on the limbs, was defined in Equation (3.2), which is recalled as:

$$\dot{\Upsilon} = \begin{bmatrix} J_p & J_q \end{bmatrix} \begin{bmatrix} \eta_p \\ \dot{q} \end{bmatrix} \tag{4.6}$$

## 4.4 Dynamics Model of a CPR

We are now able to state the dynamics of our CPR as a subcase of the one of a hybrid (rigid-deformable) mobile multi-body system, defined in Section 3.1, which we recall here as follows:

$$\begin{cases} \begin{bmatrix} 0 \\ Q_{act} \end{bmatrix} & = \begin{bmatrix} \mathcal{M}_p & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} \dot{\eta}_p \\ \ddot{q} \end{bmatrix} + \begin{bmatrix} F_p \\ Q \end{bmatrix} - \begin{bmatrix} J_p^T \\ J_q^T \end{bmatrix} \lambda \\ \Upsilon\left(g_p, q\right) & = 0 \end{cases} \tag{4.7}$$

where from top to bottom, we recognize the dynamics of the platform (4.3), with $F_p = -\mathrm{ad}^T_{\eta_p}\mathcal{M}_p\eta_p - F_{\mathrm{ext},p}$, those of the limbs (4.1) with $Q = Q_v + Q_{\mathrm{ext}}$. As expected, these two dynamics are coupled by the contact wrenches and generalized forces of (4.3) and (4.1), which take the detailed form: $(F^T_{p,c}, Q^T_c)^T = (J_p,\ J_q)^T\,\lambda$, where $\lambda$ is a vector of Lagrange multipliers forcing the holonomic algebraic constraints $\Upsilon = 0$ imposed by the (ideal) distal joints, while $J_p$ and $J_q$ are the Jacobian matrices of $\Upsilon$ defined by (4.6).

The system (4.7) defines a closed formulation of the CPR dynamics parametrized by the set of state variables:

$$\chi = (g_p,\ \eta_p,\ \dot{\eta}_p,\ q,\ \dot{q},\ \ddot{q},\ \lambda) \tag{4.8}$$

The numerical resolution of (4.7) follows the approach described in Section 3.3, for which we briefly detail the application to the context of CPRs.

## 4.5 Numerical resolution

The time integration of the CPR dynamics is performed with a $\alpha$-generalized implicit scheme, which is a direct adaptation of the Newmark scheme [105] previously applied to a SMMS 3.3.

### 4.5.1 Generalized-$\alpha$ scheme

The simulation of constrained mechanical systems (such as a CPR), requires the time integration of a systems of DAEs with geometric constraints as the ones in (4.7). These geometric constraints induce high frequencies parasitic modes making any undamped schemes unconditionally unstable [98]. To overcome this issue, different integrators with internal damping are available in the literature such as: the Hilber, Hughes and Taylor (HHT) [106], the Wood, Bossak and Zienkiewicz (WBZ-$\alpha$) [107], the Generalized-$\alpha$ scheme [108] and others. The latter is a generalization of all the previously listed implicit schemes and presents an optimal damping ratio, *i.e.,* for a given value of high frequency dissipation it minimises the damping of low frequencies (which correspond to the actual dynamics response of the system). The schemes introduces two parameters $\alpha_m$ and $\alpha_f$ which are computed as:

$$\alpha_m = \frac{2\rho_\infty - 1}{\rho_\infty + 1}, \qquad \alpha_f = \frac{\rho_\infty}{\rho_\infty + 1}, \tag{4.9}$$

where $\rho_\infty \in [0,1]$ is the desired value of high frequency damping ($\rho_\infty = 0$ gives an asymptotic annihilation of high frequencies while $\rho_\infty = 1$ provides no dissipation). This scheme adapts the prediction and correction routines of the Newmark scheme, detailed in Section 3.3, to include a term of "pseudo-accelerations" using $\gamma$ and $\beta$ selected in order to have suitable accuracy and stability properties [108, 109]:

$$\gamma = \frac{1}{2} - \alpha_m + \alpha_f, \qquad \beta = \frac{1}{4}\left(\gamma + \frac{1}{2}\right)^2. \tag{4.10}$$

As regards the prediction routine of the state at $t_{n+1}$, it is still inertial (*i.e.*, s.t. $\ddot{q}_{n+1} = 0$), except that now the acceleration $\ddot{q}_n$ is replaced by a pseudo-acceleration that depends on $\alpha_m$ and $\alpha_f$ [109]:

$$a_{n+1} = \frac{\alpha_m}{1-\alpha_m} a_n + \frac{1-\alpha_f}{1-\alpha_m} \ddot{q}_{n+1} + \frac{\alpha_f}{1-\alpha_m} \ddot{q}_n \tag{4.11}$$

The correction routine is identical to that of Newmark, where $(a, b)$ are now computed as:

$$a = \frac{\gamma}{h\beta}, \qquad b = \frac{1-\alpha_m}{h^2\beta\,(1-\alpha_f)}. \tag{4.12}$$

### 4.5.2 Parametrization of the current state $\chi$ imposed by the implicit integrator

As detailed in Section 3.2, with an implicit integration all the velocities and accelerations in $\chi$ (4.8), can be parametrized with a vector of independent configuration variables that the simulation algorithm will need to compute at each time step. To derive this parametrization, first remark that using the usual integrator (3.9) applied to $(q, \dot{q}, \ddot{q})$ allows to express $(\dot{q}, \ddot{q})$ as a function of $d_q$ and the past state only. As regards the platform kinematics $(g_p, \eta_p, \dot{\eta}_p)$, they are first expressed in terms of $(r_p, \dot{r}_p, \ddot{r}_p)$ and $(R_p, \Omega_p, \dot{\Omega}_p)$ separately. Then, applying the usual scheme (3.9) to the first triplet, and its generalization (3.10,3.11) to the second triplet, $(g_p, \eta_p, \dot{\eta}_p)$ can be entirely determined by the vector of independent variables:

$$\nu_p = \begin{bmatrix} \Theta_p \\ d_p \end{bmatrix}, \tag{4.13}$$

through the kinematic relations $g_p = C(\nu_p)$, $\eta_p = A(\nu_p)$ and $\dot{\eta}_p = B(\nu_p)$ of Section 3.2. Finally, at any discrete time of the simulation, the CPR dynamics constrained by the integration scheme is entirely parametrized by the reduced state vector:

$$\overline{\chi} = \begin{bmatrix} \nu_p \\ q \\ \lambda \end{bmatrix}, \tag{4.14}$$

where from now on, the top bar $\overline{\ }$ designates any function that depends solely on $\overline{\chi}$.

### 4.5.3 Parametrization of $\Delta\chi$ in terms of $\Delta\bar{\chi}$

As detailed in Section 3.2, the simulation algorithm will also need the maps that relate the linear perturbations of $\chi$, noted $\Delta\chi$, to those of $\bar{\chi}$, noted $\Delta\bar{\chi}$, which we remind here, for the limbs generalized coordinates vector as:

$$\Delta\dot{q} = a\Delta q, \qquad \Delta\ddot{q} = b\Delta q, \tag{4.15}$$

and, for the platform kinematics, can be expressed in terms of the variation $\Delta\nu_p$ only, as:

$$\Delta\zeta_p = \frac{\partial C}{\partial \nu_p}\Delta\nu_p, \quad \Delta\eta_p = \frac{\partial A}{\partial \nu_p}\Delta\nu_p, \quad \Delta\dot{\eta}_p = \frac{\partial B}{\partial \nu_p}\Delta\nu_p, \tag{4.16}$$

where $\Delta\zeta_p = (g_p^{-1}\Delta g_p)^\vee$, and $\frac{\partial A}{\partial \nu_p}$, $\frac{\partial B}{\partial \nu_p}$ and $\frac{\partial C}{\partial \nu_p}$ are the Jacobian matrices defined in Section 3.2.

### 4.5.4 Prediction-correction simulation algorithm

Under these constraints, at each time step of a simulation the system of DAEs (4.7) can be changed into a residual vector, as defined in Section 3.3, which we recall here, dropping the $(n)$ superscript, as:

$$\overline{\mathcal{R}}(\overline{\chi}) = \begin{bmatrix} \overline{\mathcal{R}}_p \\ \overline{\mathcal{R}}_q \\ \overline{\mathcal{R}}_\Upsilon \end{bmatrix} = 0, \tag{4.17}$$

At each time step of the simulation, the solution of (4.17) can be found iteratively with a Newton-Raphson loop initialized by a prediction. To this aim, the Jacobian matrix $\overline{\mathcal{J}} = \frac{\partial \overline{\mathcal{R}}}{\partial \overline{\chi}}$ is required to iteratively correct the value of $\overline{\chi}$ according to:

$$\overline{\chi} \leftarrow \overline{\chi} - \overline{\mathcal{J}}^{-1}\overline{\mathcal{R}}. \tag{4.18}$$

As explained in Section 3.3, $\overline{\mathcal{R}}$ and $\overline{\mathcal{J}}$ are computed using two recursive Newton-Euler algorithms, respectively: the Inverse Dynamics Model ($IDM$) and its Tangent version (named $TIDM$), for which we recall the formulation applied on the CPRs of interest.

## 4.6 Newton-Euler model of a CPR

In this section we apply the Newton-Euler model of Section 3.5 to our CPR. To introduce this implementation, let us first remind that our choice of virtual cuts led us to consider two tree-like subsystem of the platform and the limbs. This contrasts with the choice proposed in Chapter 3 where, after virtual cuts, the original system is changed into a single arborescent one. As a result of this difference, we will apply the Newton-Euler modelling of [86, 87] to the limbs and platform, considered separately. To handle loop closures with a single algorithm, the platform will be declared as a rigid tree-like locomotor, with fixed joints and virtual bodies. The Newton-Euler limbs model, on the other hand, is more complex, and we will start with it in order to recall the approach in the most general

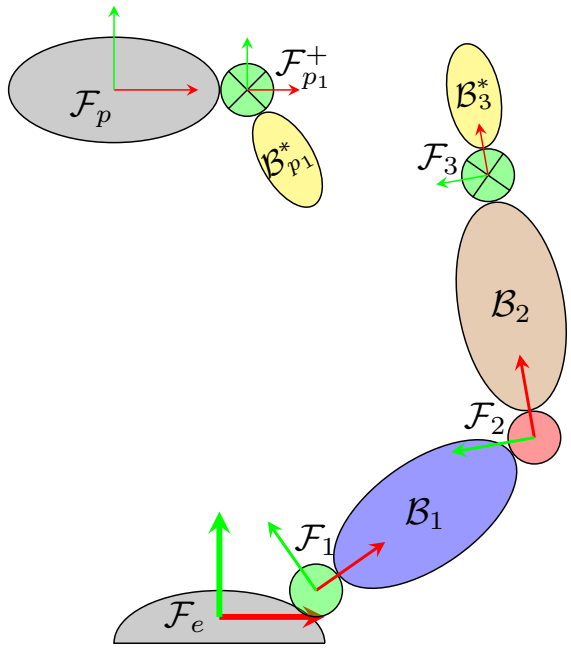



Figure 4.3: Newton-Euler segmentation of the first limb, *i.e.,* with $i = 1$, with the two fictitious rigid bodies and fixed joints created by the virtual cut of the distal joint $\mathcal{J}_{p_1}$.

context, which we will then degrade to apply to the platform. We start by recalling the segmentation of Section 3.5 applied to a limb to obtain the sequence of bodies and joints. We then recall the $IDM$ of its dynamics and the computation of the residual vector of (4.17) and its Jacobian.

### 4.6.1 Segmentation of limbs

Following the procedure of Section 3.5.2, each rod is segmented into two rigid bodies defined by its two tip cross-sections, while its internal domain defines a continuum (flexible) joint. Then, each virtual cut is replaced by a pair of virtual rigid bodies rigidly attached to the distal section of both (now serial) branches (here the platform and the limbs). Based on this segmentation,the subsystem of limbs is changed into a tree-like system of $3n_l + 1$ rigid bodies (2 cross-sections and one virtual body per limb + the base) noted $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_{3n_l+1}$, where $B_0$ is the base, and any $\mathcal{B}_j$ has only one antecedent $\mathcal{B}_k$ and one successor $\mathcal{B}_l$, *i.e.,* $k < j < l$, except the base, which has no antecedent, and the virtual bodies which have no successor. Each limb contains 3 joints, for a total of $3n_l$ joints noted $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_{3n_l}$. From the base to the distal end of each limb (Figure 4.3), we meet firstly a localized rigid joint connecting the base to the proximal section of its flexible rod), secondly, a continuum joint connecting the proximal section to the distal section of the same rod (Figure 4.2a), and thirdly, the fixed joint connecting the cross section to the virtual body. Remind that this last joint is the one used in the loop closure model.

### 4.6.2 *IDM* of CPR Limbs

Fed by the state variables $(q, \dot{q}, \ddot{q}, \lambda)$ of $\chi$, the limbs $IDM$ calculates the internal wrenches transmitted along the limbs, as discussed in Section 3.6, which we now recall.

#### Forward Kinematics

This model relates the kinematics of each pair of consecutive rigid bodies $\mathcal{B}_k$ and $\mathcal{B}_j$. It is given by the usual NE kinematics which are here fed by $(q, \dot{q}, \ddot{q})$, and initialized by a constant inertial pose, and zero velocity and acceleration (the base being fixed):

$$\begin{aligned} &\text{For } j = 1, 2, \ldots, 3n_l + 1 \ : \\ &\begin{cases} g_j = g_k \, {}^k g_j, \\ \eta_j = \mathrm{Ad}_{{}^j g_k} \eta_k + \eta_{j/k}, \\ \dot{\eta}_j = \mathrm{Ad}_{{}^j g_k} \dot{\eta}_k + \mathrm{ad}_{\eta_j} \eta_{j/k} + \dot{\eta}_{j/k}, \end{cases} \end{aligned} \tag{4.19}$$

where the set $\left({}^k g_j, \eta_{j/k}, \dot{\eta}_{j/k}\right)$ is computed as follows:

- If $\mathcal{J}_j$ is a rigid joint, one uses the standard relations:

$$ {}^k g_j = {}^k g_j(q_j), \quad \eta_{j/k} = A_j \dot{q}_j, \quad \dot{\eta}_{j/k} = A_j \ddot{q}_j \tag{4.20}$$

- If $\mathcal{J}_j$ is a continuum joint, one needs to integrate forward the Cosserat rod kinematics ODEs of Section 2.1.2:

$$\begin{cases} {}^k g_j' &= {}^k g_j \widehat{\xi}_j \\ \eta_{j/k}' &= -\mathrm{ad}_{\xi_j} \eta_{j/k} + \dot{\xi}_j \\ \dot{\eta}_{j/k}' &= -\mathrm{ad}_{\xi_j} \dot{\eta}_{j/k} - \mathrm{ad}_{\dot{\xi}_j} \eta_{j/k} + \ddot{\xi}_j \end{cases} \tag{4.21}$$

  with initial conditions $({}^k g_j, \eta_{j/k}, \dot{\eta}_{j/k})(0) = (\mathbb{1}_4, 0, 0)$.

Once these relative kinematics known for each of the limbs, one can compute the inertial poses, velocities and accelerations $(g_j, \eta_j, \dot{\eta}_j)(X)$ along the rods as discussed Remark 18, which are then used to feed the backward dynamics.

#### Backward Dynamics

For every rigid body $\mathcal{B}_j$, the algorithm computes the wrenches it receives from its successors $\mathcal{B}_l$, with $l \in s(j)$ the set of the indices of the successors of $\mathcal{B}_j$[3]. As this is usually done in NE formulation of rigid robotics, this computation is performed with the standard backward recursion:

$$\begin{aligned} &\text{For } j = 3n_l + 1, 3n_l, \ldots, 1 \ : \\ &F_j = \sum_{l \in s(j)} {}^j F_l + \mathcal{M}_j \dot{\eta}_j - ad^T_{\eta_j} \mathcal{M}_j \eta_j - F_{\mathrm{ext},j} \end{aligned} \tag{4.22}$$

[3]Note that even though, in the case of a limb, $s(j)$ is reduced to a single index, we consider this more general situation to be used for the platform subsystem.

where ${}^{j}F_l$ the wrench coming from $\mathcal{J}_l$. At the distal end of each limb, the backward dynamics is initialized by the contact wrench (Remark 17) which we recall here as:

$$ {}^{j}F_l = F_{i+} = \begin{bmatrix} T(\Theta_i)^{-T} & \mathbb{0}_3 \\ \mathbb{0}_3 & {}^{i+}R_{p_i+}^T \end{bmatrix} \overline{A}_i^T \lambda_l \triangleq J_{i+}^T \lambda_l, \tag{4.23} $$

where $T(\Theta_i)^{-1}$ is detailed in [50], ${}^{i+}R_{p_i+} = R_{i+}^T R_{p_i+}$. Equation (4.23) implies that the initial condition of the backward pass starting on the tip of the $i^{th}$ limb is conditioned by the relative pose of the two virtual bodies that replace the (virtually) cut joint $\mathcal{J}_{p_i}$. In the subsequent steps, one has:

- If $\mathcal{J}_l$ is a rigid joint:
$$ {}^{j}F_l = \mathrm{Ad}_{l_{g_j}}^T F_l, \tag{4.24} $$
- If $\mathcal{J}_l$ is a continuum joint, one takes $\Lambda_l(X=0) = -{}^{j}F_l$ and $\Lambda_l$ is deduced from the integration from $X = \ell_l$ to $X = 0$ of:
$$ \Lambda_l' = \mathrm{ad}_{\xi_l}^T \Lambda_l + \mathcal{M}_l \dot{\eta}_l - \mathrm{ad}_{\eta_l}^T \mathcal{M}_l \eta_l - \overline{F}_{\mathrm{ext},l} \tag{4.25} $$

### 4.6.3 Segmentation and $IDM$ of the platform

Fed by $(g_p, \eta_p, \dot{\eta}_p, \lambda)$ of $\chi$, the $IDM$ of the platform computes the left hand side of (4.3) with (4.2). It is obtained with the same RNEAs, but now applied to a tree-like system constituted by the floating platform $\mathcal{B}_0$ connected through $n_l$ fixed joints to the virtual (mass-less) rigid bodies $\mathcal{B}_{p_i}^{\star}$ numbered consistently with the limbs. Each of these fixed joints rigidly supports one of the distal joint frames $\mathcal{F}_{p_i}^{+}$ of Section 4.2. With this segmentation, the platform subsystem defines a rigid locomotor with $n_l + 1$ bodies whose floating base $\mathcal{B}_0$ is the CPR platform and its successors ($s(0) = \{1, 2, \ldots, n_l\}$) define the sequence of virtual rigid bodies. All the joints being rigid, only the upper level (4.19,4.22) of the previous $IDM$ applies. In details, the forward kinematic pass takes the form (4.19) but now for $j = 1, \ldots, n_l$, where $k = 0$ is the index of the platform, $j = 1, 2, \ldots, n_l$ that of limbs, and ${}^{0}g_j = {}^{p}g_{p_j+}$ are constant transformations. Initializing this first pass with $(g_0, \eta_0, \dot{\eta}_0) = (g_p, \eta_p, \dot{\eta}_p)$ allows to compute the kinematics $(g_j, \eta_j, \dot{\eta}_j)$ of all the distal joints frames $j = 1, \ldots, n_l$. Once, these kinematic variables known, the platform $IDM$ enters into its backward pass which takes the same form as (4.22), except that it now runs from $j = n_l$ to $j = 0$, since the base is floating:

$$ \begin{aligned} &\text{For } j = n_l, n_l - 1, \ldots, 0 : \\ &F_j = \sum_{l \in s(j)} {}^{j}F_l + \mathcal{M}_j \dot{\eta}_j - ad_{\eta_j}^T \mathcal{M}_j \eta_j - F_{\mathrm{ext},j} \end{aligned} \tag{4.26} $$

The virtual bodies having no inertia nor successors, we have $\mathcal{M}_j = 0$ and for $j = n_l, n_{l-1}, \ldots 1$, (4.26) reduces to:

$$ F_j = {}^{j}F_l = F_{p_i+} \tag{4.27} $$

where the virtual bodies are subject to the contact wrenches $F_{p_j+}$ transmitted by the distal joints of Section 4.3 and here considered as external wrenches for the platform subsystem. Referring to Section 3.6, these computations are initialized with the contact wrenches coming from the limbs:

$$F_{p_i+} = -Ad^T_{i+_{g_{p_i+}}} F_{i+} = -Ad^T_{i+_{g_{p_i+}}} J^T_{i+}\lambda_i, \tag{4.28}$$

where $J_{i+}$ is defined by (4.23). Finally, for $j = 0$, (4.26) and (4.24) gives:

$$F_0 = \sum_{l=1}^{n_l} Ad^T_{p_l+_{g_0}} F_{p_l+} + \mathcal{M}_0\dot{\eta}_0 - ad^T_{\eta_0}\mathcal{M}_0\eta_0 - F_{\text{ext},0}, \tag{4.29}$$

## 4.7 Computation of the Residual vector

The process for the computation of the residual vector, detailed in Section 3.4, is now recalled to be applied to the system of interest.

### 4.7.1 Computation of the platform Residual $\overline{\mathcal{R}}_p$

The platform $IDM$ can be directly used to compute $\overline{\mathcal{R}}_p$ as follows. Firstly, use the integration scheme with the kinematics transformations between $SE(3)$ and $SO(3) \times \mathbb{R}^3$, to express the initial conditions of the platform $IDM$ forward pass as:

$$(g_0, \eta_0, \dot{\eta}_0) = (C_p, A_p, B_p)(\nu_p). \tag{4.30}$$

Secondly, initialize the backward pass fed by these kinematics, with the $\lambda$-dependent initial conditions (4.28), directly gives the expected residual sub-vector $\overline{\mathcal{R}}_p$ as outputs:

$$\bar{\mathcal{R}}_p = F_0, \tag{4.31}$$

Indeed, $F_0$ is merely the right hand side of (4.29,4.2) constrained by the implicit integration scheme.

### 4.7.2 Computation of the Limbs Residual $\overline{\mathcal{R}}_q$

As detailed in Section 3.4, the components $\overline{\mathcal{R}}_j$ of the residual vector of limbs $\overline{\mathcal{R}}_q$ are defined as the dynamic balance along the DoFs of rigid or continuum joints $\mathcal{J}_j$, under the constraints of the integration scheme.

#### Residual of a Rigid Joint

For a rigid joint, the actuation torque $Q_{act,j}$ locally applied to $\mathcal{J}_j$ is balanced by $A_j^T F_j$ where $F_j$ is given by (4.22) with (4.24), and we have:

$$\overline{\mathcal{R}}_j = A_j^T F_j - Q_{act,j}. \tag{4.32}$$

**Residual of a Continuum Joint**

For a continuum joint, the internal balance of the rod is then given by the active constitutive law:

$$\Lambda_j - \mathcal{H}_j \left(\xi_j - \xi_j^o\right) - \Lambda_{act,j} = 0, \tag{4.33}$$

and by the weak-form of the rod dynamics:

$$\overline{\mathcal{R}}_j = \int_0^{\ell_j} \Phi_j^T \left(\Lambda_j - \Lambda_{act,j} - \mathcal{H}_j \Phi_j q_j\right) dX. \tag{4.34}$$

Finally, gathering the $\overline{\mathcal{R}}_j$ of rigid and continuum joints along all the limbs, provides the residual vector of limbs subsystem $\bar{\mathcal{R}}_q$.

### 4.7.3 Computation of the constraint Residual $\overline{\mathcal{R}}_\Upsilon$

This computation is a by-product of the $IDM$. In fact, the forward kinematics of limbs provides the inertial tip poses $g_{i+}$, while that of the platform gives $g_{p_i+}$. Therefore, $\overline{\mathcal{R}}_\Upsilon$ is computed by (4.5).

## 4.8 Computation of the Jacobian matrix for the residual

The Jacobian matrix of (4.18), denoted $\overline{\mathcal{J}}$, is a linear map that relates the variations of configuration variables $\overline{\chi}$ (4.8) to those of the residual vector $\overline{\mathcal{R}}$:

$$\overline{\mathcal{J}} = \frac{\partial \overline{\mathcal{R}}}{\partial \overline{\chi}} = \begin{bmatrix} \frac{\partial \overline{\mathcal{R}}_p}{\partial \nu_p} & \frac{\partial \overline{\mathcal{R}}_p}{\partial q} & J_p^T \\ \frac{\partial \overline{\mathcal{R}}_q}{\partial \nu_p} & \frac{\partial \overline{\mathcal{R}}_q}{\partial q} & J_q^T \\ J_p & J_q & \mathbb{0}_m \end{bmatrix}, \tag{4.35}$$

which, as detailed in Section 3.3, requires the computation of only:

$$\frac{\partial}{\partial \overline{\chi}} \begin{bmatrix} \overline{\mathcal{R}}_p \\ \overline{\mathcal{R}}_q \end{bmatrix} = \begin{bmatrix} \frac{\partial \overline{\mathcal{R}}_p}{\partial \nu_p} & \frac{\partial \overline{\mathcal{R}}_p}{\partial q} & J_p^T \\ \frac{\partial \overline{\mathcal{R}}_q}{\partial \nu_p} & \frac{\partial \overline{\mathcal{R}}_q}{\partial q} & J_q^T \end{bmatrix}. \tag{4.36}$$

The terms in (4.36) can be computed applying the recursive $TIDM$ routine of Section 3.6.2 and 3.7.3 to both tree-like structures of the limbs and the platform. For conciseness, we do not recall the $TIDM$ algorithm, but we limit to the definition of its initial conditions to obtain all the terms of (4.36). In detail, each of these terms can be computed column-wise from the perturbation $\Delta\chi$ consisting in sequentially feed the $TIDM$ with $\Delta\nu_p$, $\Delta q$ and $\Delta\lambda$.

### 4.8.1 Influence of $\Delta\nu_p$

The perturbation of $\Delta\nu_p$ is mapped to $(\Delta\zeta_p, \Delta\eta_p, \Delta\dot{\eta}_p)$ by the implicit relation (4.16), defining the tangent kinematics of the platform floating body. Then, the tangent kinematics of all the $\mathcal{B}^\star_{p_i}$ are computed directly as:

$$\left(\Delta\zeta_{p_i+}, \Delta\eta_{p_i+}, \Delta\dot{\eta}_{p_i+}\right) = \mathrm{Ad}_{p_i+ g_p} \left(\Delta\zeta_p, \Delta\eta_p, \Delta\dot{\eta}_p\right), \tag{4.37}$$

which is a subcase of (3.45). On the other hand, as $\Delta q = 0$, the limbs forward tangent kinematics results in $(\Delta\zeta_{i+}, \Delta\eta_{i+}, \Delta\dot{\eta}_{i+}) = (0,0,0)$, with $i = 1, \dots, n_l$. Then, the $\Delta\zeta_{p_i+}$ is used to compute the initial condition of the backward tangent dynamics, as detailed in (3.53–3.55), which we remind here as:

$$\Delta F_{p_i+} = -\mathrm{Ad}^T_{i+g_{p_i+}} \left( \Delta F_{i+} - \mathrm{ad}^T_{\Delta\zeta_{i+/p_i+}} F_{i+} \right), \tag{4.38}$$

with $\Delta\zeta_{i+/p_i+} = \Delta\zeta_{i+} - \mathrm{Ad}_{i+g_{p_i+}} \Delta\zeta_{p_i+}$ and $\Delta F_{i+}$ which details as:

$$\Delta F_{i+} = J^T_{i+} \Delta\lambda_i + \Delta J^T_{i+} \lambda_i \tag{4.39}$$

where $\Delta J_{i+}$ is the variation of $J_{i+}$, defined in (4.23), which we recall as:

$$\Delta J_{i+} = \overline{A}_{p_i+} \begin{bmatrix} \Delta T\left(\Theta_i\right)^{-1} & \mathbb{0}_3 \\ \mathbb{0}_3 & R_i T\left(\Theta_i\right) \end{bmatrix}. \tag{4.40}$$

Having, in this case, $\Delta\zeta_{i+} = 0 \in \mathbb{R}^6$, as no perturbation is applied on the limbs, as well as $\Delta\lambda = 0$. The computation of $\Delta F_{p_i+}$ and $\Delta F_{i+}$ provide the initial condition for the backward dynamics which runs on both subsystems and, as described in Section 3.6.2 and 3.7.3. On the platform subsystem, the backward pass of the $TIDM$ computes $\Delta\overline{\mathcal{R}}_p = \Delta F_p$, using Equation (3.48), here detailed as:

$$\Delta F_p = \mathcal{M}_p \Delta\dot{\eta}_p - \mathrm{ad}^T_{\eta_p} \mathcal{M}_p \Delta\eta_p - \mathrm{ad}^T_{\Delta\eta_p} \mathcal{M}_p \eta_p + \Delta F_{\mathrm{ext},p} + \sum_{l \in s(p)} \Delta^p F_{p_i+}, \tag{4.41}$$

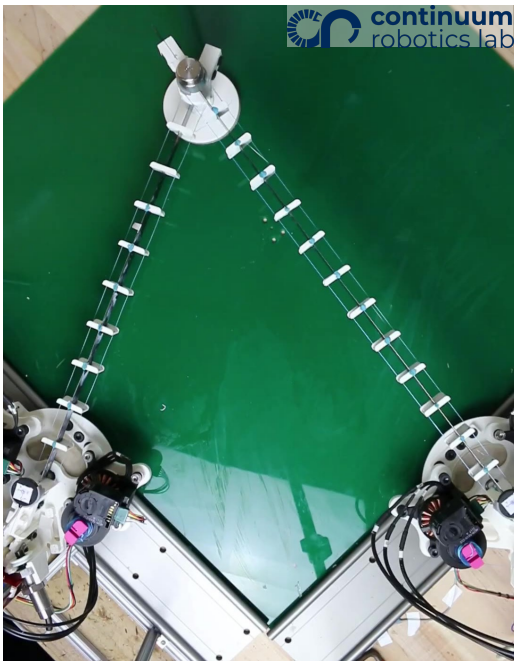
which corresponds to $\frac{\partial\overline{\mathcal{R}}_p}{\partial\nu_p}$ of (4.36). On the other hand, on the limbs subsystem, all the $\Delta\overline{\mathcal{R}}_j$ of the CPRs joints computed. These (tangent) balances are defined by (3.62–3.63), which details as:

$$\begin{aligned} \Delta\overline{\mathcal{R}}_j &= A^T_j \Delta F_j - \Delta Q_{act}, \\ \Delta\overline{\mathcal{R}}_j &= \int_0^{\ell_j} \Phi^T_j A^T_j \left(\Lambda_j - \Delta\Lambda_{act,j} - \mathcal{H}_j A_j \Delta q_j\right) dX, \end{aligned} \tag{4.42}$$

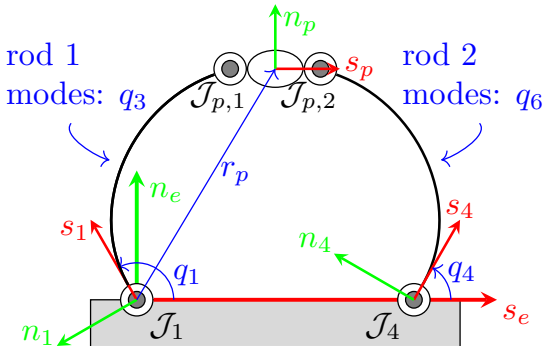
for a rigid and a continuum joint, respectively. The collection of all the $\Delta\overline{\mathcal{R}}_j$ defines $\frac{\partial\overline{\mathcal{R}}_q}{\partial\nu_p}$

### 4.8.2 Influence of $\Delta q$

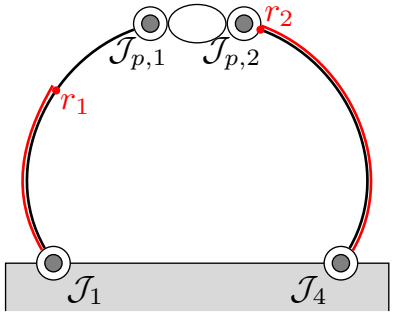
The unitary perturbations the limbs generalized coordinates $\Delta q$ is mapped to $\Delta\dot{q}$ and $\Delta\ddot{q}$ with the implicit relations (4.15). These are are used to compute the relative tangent kinematics of the limbs joints using Equations (3.46–3.47). Then, the forward tangent kinematics of the limbs subsystem is computed with (3.45), which provide the $\Delta\zeta_{i+}$ of all the virtual rigid bodies $\mathcal{B}^\star_{i+}$ attached at the tip of the rods. On the other hand, being the platform a separate subsystem, its tangent kinematics is not affected and $(\Delta\zeta_{p_i+}, \Delta\eta_{p_i+}, \Delta\dot{\eta}_{p_i+}) = (\Delta\zeta_p, \Delta\eta_p, \Delta\dot{\eta}_p) = (0,0,0)$. With the so defined (tangent) kinematics state, the backward dynamics is initialized computing $\Delta F_{p_i+}$ (4.38) and $\Delta F_{i+}$ (4.39) for the platform and the limbs subsystems, respectively. As detailed in Section 4.8.1, the backward pass runs on the platform subsystem to compute $\Delta\overline{\mathcal{R}}_p$ (4.41) (which in this case corresponds to $\frac{\partial\overline{\mathcal{R}}_p}{\partial q}$) and on the limbs subsystem to compute all the $\Delta\overline{\mathcal{R}}_j$s (4.42) corresponding to $\frac{\partial\overline{\mathcal{R}}_q}{\partial q}$.




(a) TD-CPR prototype at the Continuum Robotics Laboratory [110].




(b) Schema of the TD-CPR with its joints names and coordinates.




(c) Schema of the TD-CPR with a representation of the measured shape by the FBGSs (in red).

Figure 4.4: The Tendon-Driven planar CPR (TD-CPR) at the Continuum Robotics Laboratory (left), a detailed representation of the distributed actuation (right).

### 4.8.3 Influence of $\Delta\lambda$

Finally, for the last column of (4.36), the forward pass of the *TIDM* is not required as the unitary variations is applied on the Lagrange multipliers only. The initial condition of the tangent backward dynamics can be computed with (4.38–4.39), for which the expression of $\Delta F_{p_i+}$ simply becomes:

$$\Delta F_{p_i+} = -\mathrm{Ad}^T_{i+_{g_{p_i+}}} \Delta F_{i+}. \tag{4.43}$$

The, the *TIDM* backward pass on the two subsystems computes $\Delta\overline{\mathcal{R}}_p$ and $\Delta\overline{\mathcal{R}}_q$ corresponding to $J_p^T$ and $J_q^T$ for the platform and limbs subsystem, respectively.

Once established how the algorithm presented in Chapter 3 can be specialized to the context of CPR, we can proceed detailing its performances when compared to recorded behaviours.

## 4.9 Experimental Results

The objective of this section is to present the application of our model to two planar CPRs for experimental validation: a TD-CPR and a planar CPR actuated by rigid joints.

### 4.9.1 Tendon-Driven Planar CPR

This TD-CPR was designed by the Continuum Robotics Laboratory [110] and it presents two limbs connected together to a platform (Figure 4.4a). Each limb is composed by a proximal joint and a tendon-driven rod. The former is a passive revolute joint mounted on a rail, allowing for position adjustment. It includes bearings and anti-friction fabric to support a rigid body. This rigid body is designed to provide the required structural support for the hardware. It also accommodates the deformable rod and the actuators for the tendon-driven actuation [110], along with the corresponding force sensors and electronics. For the distributed actuation, each deformable rod is equipped with a series of plastic disks, equispaced along its body, guiding two cables parallel to its centreline. The cables slide into the disks hole while being attached at the rod tip and pulled at the rod base. For this robot, the platform consists of a rigid body coaxial with the distal revolute joints (Figure 4.4a). The platform slides on a plastic plate using another anti-friction fabric.

Once defined the robot structure, we can proceed detailing its dynamics simulation.

#### Simulation

We begin this section by listing the elements of $\overline{\chi}$ that define the configuration of the prototype. Then, the dynamics model of (4.7) is provided with an additional formulation for the friction at the proximal joints and platform. Finally, we will detail the parameters of the dynamics simulation.

**Configuration Variables** With the Newton-Euler segmentation of Section 4.6.1, numbering the limbs joints in a depth-first approach, the configuration variables of the CPR are:

$$\overline{\chi} = \begin{bmatrix} r_p^T & q_1 & q_3^T & q_4 & q_6^T & \lambda_1^T & \lambda_2^T \end{bmatrix}^T \tag{4.44}$$

where $q_1$ and $q_4$ are the two proximal revolute joints angles, $q_3$ and $q_6$ the two sets of generalized coordinates for the deformable bodies and $\lambda_1$ and $\lambda_2$ the two sets of Lagrange multipliers (Figure 4.4b). Note that, to connect the rod to the rigid bodies, two fixed joints are used. These joints do not require configuration variables, as they define two constant rigid transformations.

**Accounting the Joints and Platform Friction** For this system, the Lagrangian dynamics model (4.7) needs to be provided with the friction at the proximal joints and the platform. The former is taken into account modifying Equation (4.32) as follows:

$$\overline{\mathcal{R}}_j = A_j^T \left( F_j - F_{act,j} + F_{f,j} \right), \quad j = 1, 4 \tag{4.45}$$

where $F_{f,j} = -A_j \dot{q}_j \mu$ with $\mu$ is a frictional coefficient and $F_{act,j} = 0$ as the proximal joints are passive. On the other hand, the friction on the platform body can be seen as a force $N_{f,p}$ defined as:

$$N_{f,p} = \frac{V_p}{\|V_p\|} \mu_p \tag{4.46}$$

where $V_p$ is the linear velocity of the platform and $\mu_p$ its friction coefficient. Equation (4.46) express a force in the base of $\mathcal{F}_p$ which is added to (4.3) with the wrench $F_{f,p} = (N_{f,p}^T, 0^T)^T$.

**Simulation Parameters** The parameters used in the dynamics simulation are detailed in the following. For the proximal joints, their relative distance is $d = 0.435$ m and for their anti-friction fabric, we estimated a viscosity coefficient $\mu = 0.005$. The weight and moment of inertia of the two rigid bodies are: $m_{b,1} = 0.897$ kg, $m_{b,4} = 0.663$ kg and $I_{zz,b,1} = 4.268e^{-3}$ kgm$^2$ and $I_{zz,b,4} = 3.122e^{-3}$ kgm$^2$ whose differences in weight and inertia come from the different weight of one of the installed sensors. The two deformable rods have the following dimensions: the first rod measures $\ell_3 = 0.394$ m, the second rod measures $\ell_6 = 0.48$ m while both have a rectangular cross-section with a width of 0.51 mm and height of 19.05 mm, which extend along the $n$ and $a$ directors, respectively. The cables of both tendon-actuations have an offset from the rod center of $d_c = 11$ mm. The Young modulus and distributed weight were estimated as $E = 293$ GPa and $\rho = 9538\ \frac{kg}{m^3}$, respectively (see Appendix D). Both deformable rods were parametrized with 2 modes for the bending along their $a$ director, and the forward and backward ODEs, respectively (4.21) and (4.25), were integrated using a spectral numerical integration with 31 Chebyshev points [40]. The platform has a mass $m_p = 0.24\ kg$ and its anti-friction fabric has an estimated damping coefficient of $\mu_p = 0.01$. Finally, for the implicit time integrator we set the numerical damping using $\rho_\infty = 0.6$ and $\Delta t = 40$ ms.

Seeking experimental validation, simulations will be compared to a dataset of measurements on the CPR motion. We now detail the setup of the sensors on the robot which are used to record such dataset.

### Experimental Setup

The robot was actuated using a distributed (tendon-driven) system and equipped with various sensors to measure the revolute joint angles, the shape of the deformable bodies, and the tension in the distributed actuation. The following sections will provide a detailed overview of these components.

**Tendon Actuation** The motion of the TD-CPR is achieved applying a tension to the cables with the DC actuators described in [110]. Before starting the motion, we recorded the reference configuration of the robot for 10 $s$. Then, we sent a current to every actuator $c = 1, \ldots, 4$ with the sinusoidal law:

$$\tau_c = 0.1 + \sigma_c \left(1 - \cos\left(\theta(t)\right)\right), \quad t \geq t_{0,c} \tag{4.47}$$

| Cable: | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $\sigma_c$ [A] | 2.0 | 2.0 | 2.5 | 2.2 |
| $k_c$ [N] | 0.517 | 0.436 | 0.786 | 1.340 |
| $f_0$ [N] | -0.442 | 0.283 | 0.276 | 0.372 |

Table 4.1: Representation, for every cable, of the values of the scaling factor and zero-offset (for the force sensors) and the actuators gain.

where $t_{0,c} = 10 + \phi_c$ s where $\phi_c = 0.5$ s for $c = 2, 4$ and zero otherwise, $\theta = \omega\,(t - t_{0,c})$ RAD with $\omega = 1$ RAD/s and $\sigma_c$ is a gain defined in Table 4.1. The actuators are controlled in open loop without retroaction between the measured and desired current nor w.r.t. the desired cable tension. As a result, the applied tension may vary among the actuators.

**Cables Tensions** Force sensors are employed to measure the cable tension for both tendon-driven actuations. To this end, each cable ($c = 1, \ldots, 4$) is routed around a pulley undergoing a 90 deg turn. With the pulley mounted on the force sensor, a reaction force $f_m$ is measured and converted to the corresponding cable tensions in accordance with the following relation:

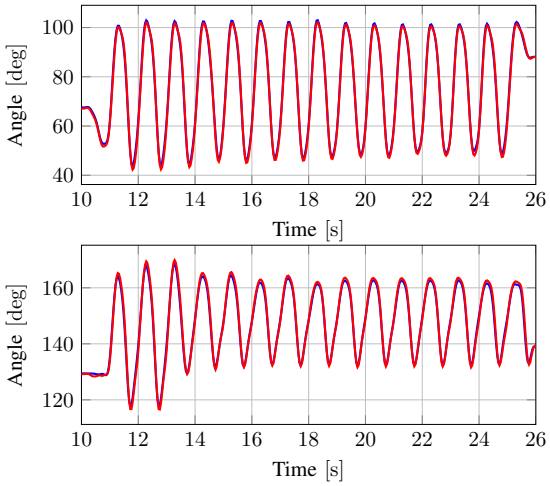
$$T_c = k_c\,(f_m - f_0) \tag{4.48}$$

where $k_c$ and $f_0$ are respectively the scaling factor and zero-offset of the force sensors (Table 4.1). All the four force sensors measurements were available at a frequency of 110 Hz.

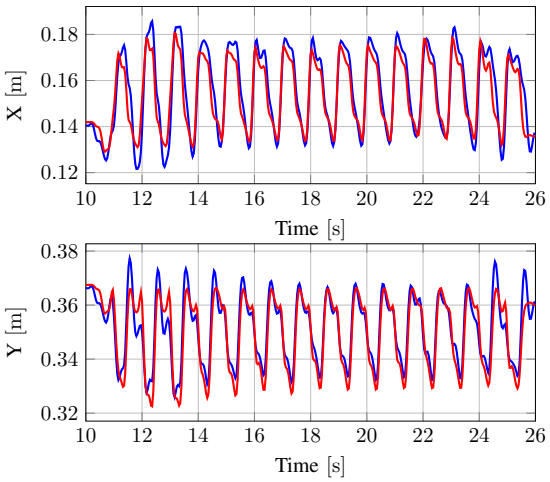
**Proximal Joints Angles** The Aurora electromagnetic tracking system, from Northern Digital Incorporated (NDI), was used to measure the pose of the frames $\mathcal{F}_1$ and $\mathcal{F}_4$ of the two proximal joints. This device is capable of detecting the $SE(3)$ position of magnetic coils placed in its workspace at a frequency of $f_s = 40$ Hz. We placed one coil on both proximal joints, at their center, allowing to compute their rotation angles ($q_1$ and $q_4$). However, due to the limitations of Aurora workspace, we could not place a coil on the platform body.

**Shapes of Deformable Bodies** Fibre Bragg Grating Sensor (FBGS) were applied to both rods to capture their shapes, with a sampling frequency $f_s = 25$ Hz. We used one sensors on each rod: the first one with a measuring length of 25 cm covering the 63% of the rod arc length while the second measures the integrality of the rod. In Figure 4.4c, the points $r_1$ and $r_2$ refer to the last measuring point of the sensor, placed on rod 1 and rod 2, respectively. Note that, for the second rod, $r_2$ corresponds to $r_p$ as the rod tip, the distal joint origin and the platform center are coaxial. These sensors were used to compute the generalized elastic coordinates of the rods ($q_3$ and $q_6$) from the measured shapes iteratively, finding their values that best reconstruct the rod shapes with the first relation of both (2.20) and (4.21).

**Data Filtering and Derivatives** All data is filtered with Butterworth filter of order 4 with cut-off frequency $f_c = 4$ Hz, the first and second derivatives are computed with the




(a) Rotation angles of the proximal joints $q_1$ (top) and $q_4$ (bottom).




(b) Trajectory of the $x$ (top) and $y$ (bottom) coordinate of the platform.

Figure 4.5: Comparison of the simulated (red) and measured (blue) values for $q_1$ and $q_4$ (left) and $r_p$ (right) obtained imposing the kinematics of the deformable bodies.

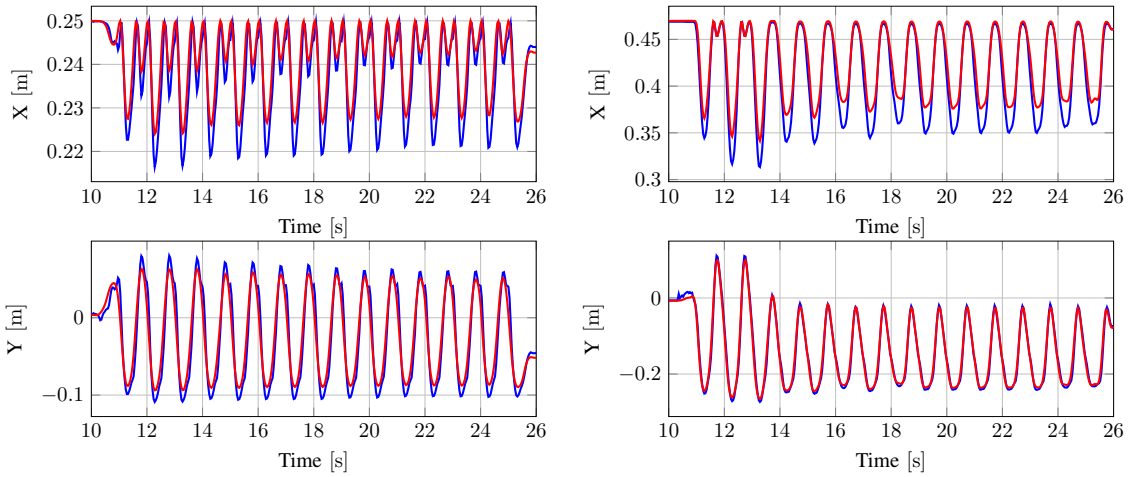



Figure 4.6: Comparison of the simulated (red) and measured (blue) values of $r_1$ (left) and $r_2$ (right) obtained imposing the kinematics of the proximal joints.

central difference $f'(t) = \frac{f(t+\Delta t)-f(t-\Delta t)}{2\Delta t}$, using $\Delta t$ as the biggest time-step of the sensors: the one of the FBGS as $\Delta t = 40$ ms. The other data is then interpolated to have reading with the same timestamp.

We can now compare the simulations with the dataset to validate the proposed approach.

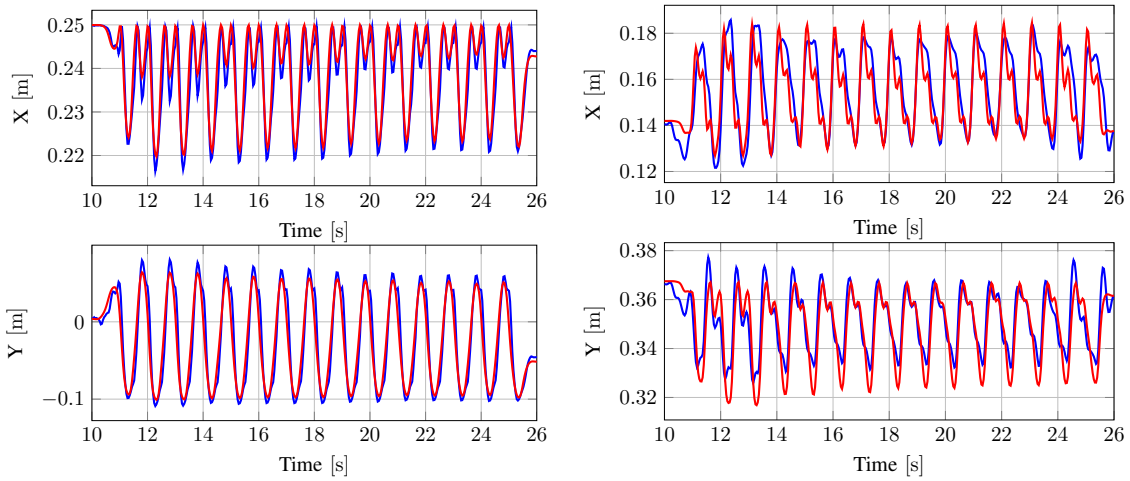



Figure 4.7: Comparison of the simulated (red) and measured (blue) values of $r_1$ (left) and $r_p$ (right) obtained imposing the kinematics of the first proximal joints and the shape of the second deformable rod.

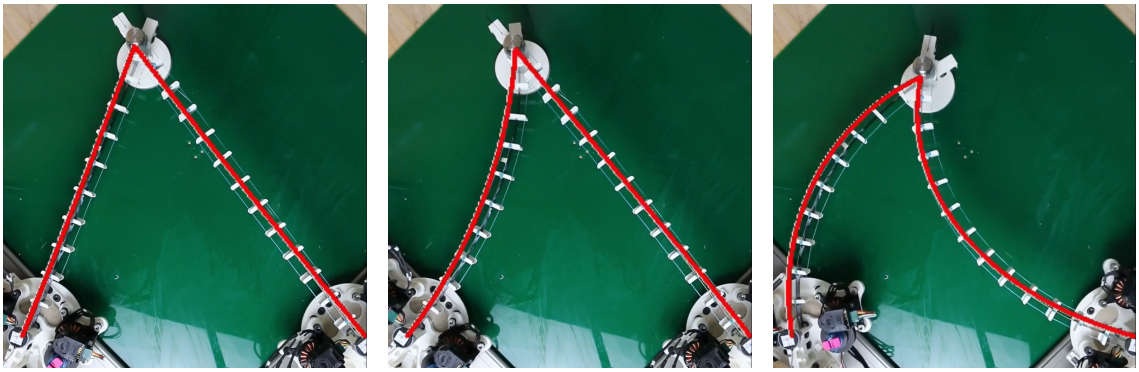
| values | units | max | mean | std | % |
|---|---|---|---|---|---|
| $q_{r,1}$ | deg | 2.6 | $4.4 \cdot 10^{-1}$ | $7.4 \cdot 10^{-1}$ | 0.79 |
| $q_{r,4}$ | deg | 3.0 | $1.3 \cdot 10^{-1}$ | $8 \cdot 10^{-1}$ | 0.33 |
| $r_{1,x}$ | m | $7.4 \cdot 10^{-3}$ | $5.9 \cdot 10^{-5}$ | $1.1 \cdot 10^{-3}$ | 0.02 |
| $r_{1,y}$ | m | $1 \cdot 10^{-2}$ | $3.3 \cdot 10^{-4}$ | $2.2 \cdot 10^{-3}$ | 0.09 |
| $r_{2,x}$ | m | $8.5 \cdot 10^{-3}$ | $6.4 \cdot 10^{-4}$ | $1.3 \cdot 10^{-3}$ | 0.10 |
| $r_{2,y}$ | m | $6.5 \cdot 10^{-3}$ | $1.4 \cdot 10^{-4}$ | $2.7 \cdot 10^{-3}$ | 0.02 |
| $r_{p,x}$ | m | $1.9 \cdot 10^{-2}$ | $7.5 \cdot 10^{-4}$ | $4.9 \cdot 10^{-3}$ | 1.39 |
| $r_{p,y}$ | m | $1.5 \cdot 10^{-2}$ | $5.2 \cdot 10^{-4}$ | $3.9 \cdot 10^{-3}$ | 1.29 |

Table 4.2: Errors between the simulated and measured configuration variables of (4.44), using $q_3$ and $q_6$ as input. The errors are expressed in terms of their maximum and mean value, with standard deviation (std) and the mean error w.r.t. the amplitude of motion (%).

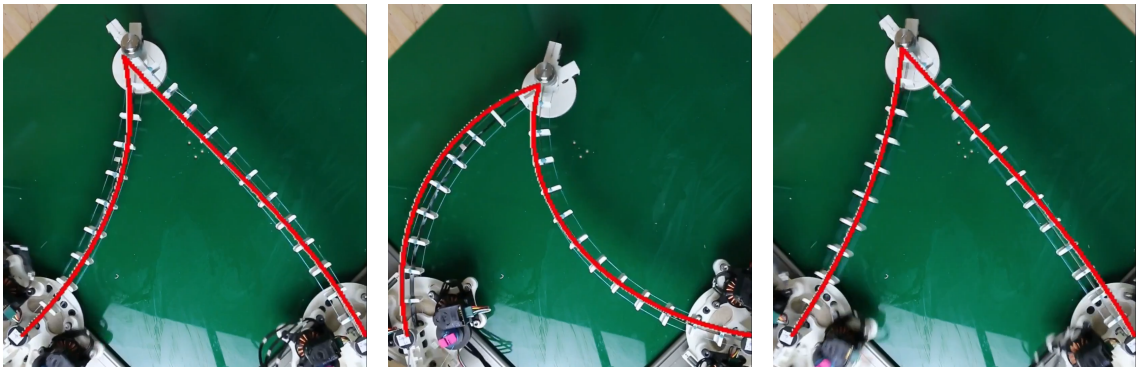
### Comparison of Simulated Behaviours

The objective of this section is to describe the steps we endured to validate our model by comparison with the available experimental data.

**Procedure for Validation** At first, we wanted to compute $Q_{act}$ from the measured cables tensions and fix its value in (4.7). As a result, the corresponding kinematics of proximal joints, deformable bodies and platform could have been compared with our measures. However, during the robot's movement, the behaviour of some actuators was not optimal, as multiple solicitations of the actuators created backlash and friction on their



(a) $t = 0$ to $t = 10$ s (b) $t = 10.19$ s (c) $t = 11.06$ s



(d) $t = 11.18$ s (e) $t = 12.04$ s (f) $t = 12.15$ s

Figure 4.8: Snapshots of the recorded dynamics motion with the superposed simulated shapes of the deformable rods (red) obtained by imposing the kinematics of the proximal joint $\mathcal{J}_1$ and the shape of rod 2.

shaft, significantly reducing their performances. As we operated the actuators in open loop, we could not counteract those effects. As a result, the measured tensions lacked the required smoothness to be used as input for the dynamics model (as they affect accelerations, which are integrated twice, those measures must be sufficiently smooth). Another possibility is to fix one or more of the kinematics values, using some of the measurements, simulate the corresponding dynamics behaviour (4.7) of the CPR and compare the output with the remaining measurements. In the following we detail three examples, with the purpose of validating different blocks of (4.7). In the first example, we fix the kinematics of the deformable bodies to validate the blocks modelling the dynamics of rigid bodies, proximal joints and platform as well as the one for the geometric constraints. Secondly, in another example we fix the kinematics of the proximal joints to validate the dynamics modelling of the deformable bodies. Finally, in one last example we show how the precision can be improved by combining the kinematics of one of the proximal joints and one of the deformable bodies.

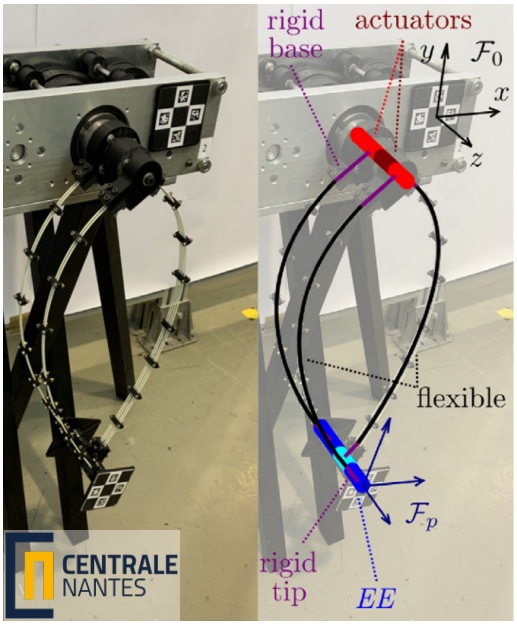
| values | units | max | mean | std | % |
|---|---|---|---|---|---|
| $q_{r,1}$ | deg | 0.0 | 0.0 | 0.0 | 0.00 |
| $q_{r,4}$ | deg | 0.0 | 0.0 | 0.0 | 0.00 |
| $r_{1,x}$ | m | $3.9 \cdot 10^{-2}$ | $3.9 \cdot 10^{-5}$ | $6.1 \cdot 10^{-3}$ | 0.01 |
| $r_{1,y}$ | m | $4.1 \cdot 10^{-2}$ | $2.4 \cdot 10^{-3}$ | $1.1 \cdot 10^{-2}$ | 0.68 |
| $r_{2,x}$ | m | $3.1 \cdot 10^{-2}$ | $6.1 \cdot 10^{-4}$ | $5.9 \cdot 10^{-3}$ | 0.09 |
| $r_{2,y}$ | m | $3 \cdot 10^{-2}$ | $7.3 \cdot 10^{-3}$ | $1.1 \cdot 10^{-2}$ | 1.02 |
| $r_{p,x}$ | m | $3.5 \cdot 10^{-2}$ | $6.3 \cdot 10^{-3}$ | $1.1 \cdot 10^{-2}$ | 11.62 |
| $r_{p,y}$ | m | $2.8 \cdot 10^{-2}$ | $2.1 \cdot 10^{-3}$ | $6.5 \cdot 10^{-3}$ | 5.27 |

Table 4.3: Errors between the simulated and measured configuration variables of (4.44), using $q_1$ and $q_4$ as input. The errors are expressed in terms of their maximum and mean value, with standard deviation (std) and the mean error w.r.t. the amplitude of motion (%)

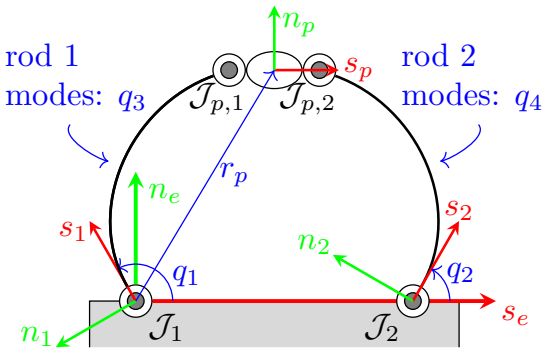
| values | units | max | mean | std | % |
|---|---|---|---|---|---|
| $q_{r,1}$ | deg | 0.0 | 0.0 | 0.0 | 0.00 |
| $q_{r,4}$ | deg | 3.3 | $2.1 \cdot 10^{-1}$ | 1.1 | 0.55 |
| $r_{1,x}$ | m | $4.7 \cdot 10^{-2}$ | $4.1 \cdot 10^{-4}$ | $6.5 \cdot 10^{-3}$ | 0.13 |
| $r_{1,y}$ | m | $5.3 \cdot 10^{-2}$ | $1.3 \cdot 10^{-4}$ | $1.3 \cdot 10^{-2}$ | 0.04 |
| $r_{2,x}$ | m | $8.5 \cdot 10^{-3}$ | $6.4 \cdot 10^{-4}$ | $1.3 \cdot 10^{-3}$ | 0.10 |
| $r_{2,y}$ | m | $6.5 \cdot 10^{-3}$ | $1.4 \cdot 10^{-4}$ | $2.7 \cdot 10^{-3}$ | 0.02 |
| $r_{p,x}$ | m | $2.1 \cdot 10^{-2}$ | $1.3 \cdot 10^{-3}$ | $6.8 \cdot 10^{-3}$ | 2.33 |
| $r_{p,y}$ | m | $1.6 \cdot 10^{-2}$ | $9.7 \cdot 10^{-4}$ | $5.4 \cdot 10^{-3}$ | 2.40 |

Table 4.4: Errors between the simulated and measured configuration variables of (4.44), using $q_1$ and $q_6$ as input. The errors are expressed in terms of their maximum and mean value, with standard deviation (std) and the mean error w.r.t. the amplitude of motion (%).

**Using the Deformable Bodies kinematics** In this first case, the kinematics of the deformable bodies are fixed into (4.7), specifically $(q_j, \dot{q}_j, \ddot{q}_j)$ with $j = 3, 6$, in order to find the corresponding behaviour of the platform and rigid joints, $r_p$, $q_1$ and $q_4$ respectively. In this scenario, the values of the revolute joint are found to be in close alignment with the measured values, with negligible discrepancies (Figure 4.5a). In contrast, discrepancies emerge when the position of the distal end is considered (Figure 4.5b). This measurement is particularly sensible to estimation errors, as it is derived from a combination of $q_4$ and the shape of the second rod. In Table 4.2 we reported the relative errors between the simulated and measured values. The table shows that on average the error between the joints angles remains of the order of a fraction of degree, which is less than 1% of their range of motion. We point out that, although the shape of the deformable bodies are imposed, there is an error between the simulated and measured values of $r_1$ and $r_2$. These values




(a) The RFRFR planar CPR prototype at LS2N [85].



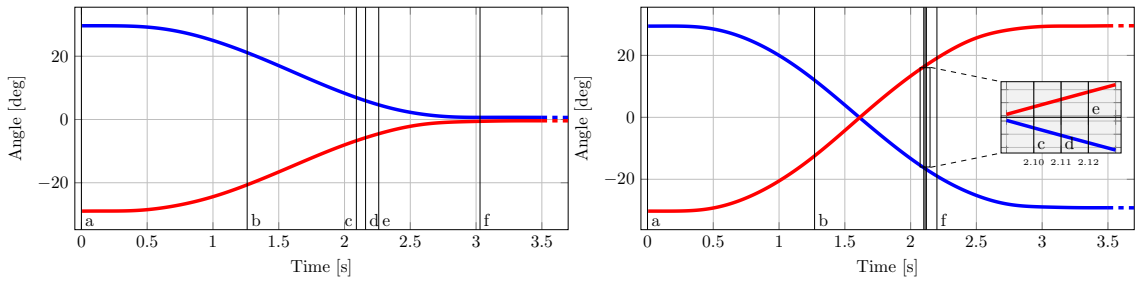

(b) Schema of the RFRFR with its proximal and distal joints.

Figure 4.9: The RFRFR Robot (left) with its generic schema (right).

are computed from the set of $q_3$ and $q_6$ which are reconstructed from the measured rod shape. The residual error of this reconstruction affects the comparison of the simulation w.r.t. the measured data. Finally, the error in the platform position $r_p$ is a combination of errors from the proximal joint, rod shape reconstruction, and data synchronization. This error remains slightly greater than 1% of the range of motion.

**Using the Proximal Joints kinematics** Fixing the the kinematics of the proximal joints: $q_1$, $q_3$, and respective derivatives, allows us to verify the model of deformable bodies ($q_3$ and $q_6$) as well as the platform ($r_p$). In Figure 4.6 we find the comparison of the simulated and measured position of $r_1$ and $r_2$ while a more detailed presentation of the relative differences is given in Table 4.3. First, we observe that the first two rows are zeros as the rotation angles are imposed and thus corresponds to the measured ones. On the other hand, we observe an increment in the relative difference of the position of $r_1$ and $r_2$ which now account for the error in the estimation of $q_3$ and $q_6$ together with the simulation errors. Finally, we see that the differences in $r_p$ have significantly increased due to the error in the reconstructed shape of the second rod and its synchronization with $q_4$.

**Combining Different Kinematics** In this last example we show how that we can select the kinematics of different parts as input of (4.7). Here, we choose to use the kinematics of the first proximal joint and the second rod, thus imposing $q_1$ and $q_6$ of (4.44). The results, in terms of the shape of position of $r_1$ and $r_p$ are shown in Figure 4.7. A more detailed




(a) Time evolution of $q_1$ (blue) and $q_2$ (red) for the first trajectory.

(b) Time evolution of $q_1$ (blue) and $q_2$ (red) for the second trajectory.

Figure 4.10: The trajectory of the proximal joint rotation angles ($q_1$ in blue and $q_2$ in red) is shown for the two stable-unstable-stable transitions: (a) for the first trajectory (b) for the second trajectory. The vertical lines indicate the time points at which snapshots were taken, corresponding to the sub-figures in Figure 4.11 (for plot a) and Figure 4.12 (for plot b).

representation of the relative errors is given in Table 4.4 where we see that the relative errors of $q_4$ and $r_1$ are of negligible order compared to the amplitude of their motions. Moreover, we see that the error on $r_1$ remains proportional to the one of $r_2$ meaning that it is mainly influenced by the reconstruction errors. Finally, the simulated position of the platform demonstrated improved accuracy, thanks to the full measurement of the shape of rod 2. In summary, this comparison demonstrates an enhancement in the precision for the the simulated platform position. Finally, Figure 4.8 presents a series of images illustrating the simulated shapes of the deformable bodies superimposed upon the recorded motion.

**Comment on the Results**

In conclusion, our model has been demonstrated to be capable of accurately reproducing the behaviour of all the constituent parts of the planar TD-CPR, although there remains room for improvement. The discrepancies observed in the qualitative comparison may be attributed to the presence of friction within the system and the rod stiffness, which were estimated but not identified, and the inertia of the various components, which could only be calculated geometrically and not verified.

### 4.9.2 Planar RFRFR CPR

The Planar RFRFR CPR, designed at LS2N [85], having two actuated proximal revolute joints and a passive revolute joint at the tip, connected by deformable, or flexible, rods (Figure 4.9a). The individual limb components can be identified using the conventional joint-type notation, represented by the sequence RFR, where R indicates a revolute joint, the underlined element denotes the actuated joint and the letter F indicates that the joint is flexible, or continuum. This CPR will be employed to illustrate the transition phases stable-unstable-stable, as made possible by the robot [85]. This robot differs from the

previous prototype in that it has the actuations localised at its proximal joints and is affected by gravity.

In this section we will show the behaviour of the CPR when it crosses a singularity, thereby illustrating that our model is capable of capturing such behaviours. We will start detailing the parameters for the dynamics simulation, then, we describe the setup of sensors and actuators installed on the prototype and, finally, we discuss the comparison of the simulated behaviours w.r.t. the recorded motions.

### Simulation

For this prototype, we will specify only the configuration variables and simulation parameters, as no changes are required to the Lagrangian dynamics model of (4.7).

**Configuration Variables** With the Newton-Euler segmentation of Section 4.6.1, and numbering the limbs joints in a breadth-first approach (Figure 4.9b), the configuration variables of the $\underline{\text{R}}\text{FRF}\underline{\text{R}}$ are:

$$\overline{\chi} = \begin{bmatrix} r_p^T & q_1 & q_2 & q_3^T & q_4^T & \lambda_1^T & \lambda_2^T \end{bmatrix}^T \tag{4.49}$$

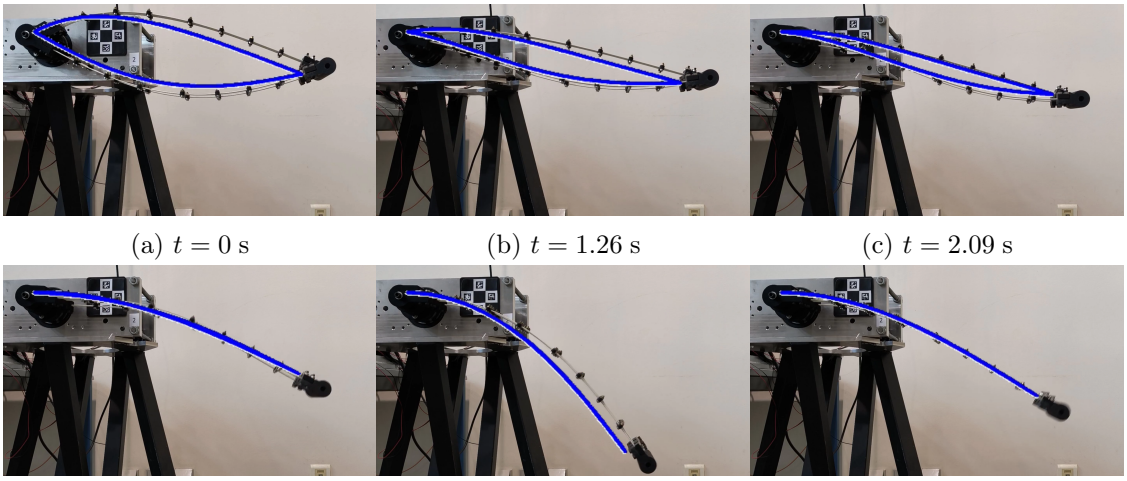
where $q_1$ and $q_2$ are the two proximal revolute joints angles, $q_3$ and $q_4$ the two sets of generalized coordinates for the deformable bodies and $\lambda_1$ and $\lambda_2$ the two sets of Lagrange multipliers.

**Remark 19.** *In this prototype, a different NE indexing method was used. Instead of the depth-first approach discussed earlier, a breadth-first method was applied to show that this alternative is both feasible and effective.* ∎

**Simulation Parameters** In terms of simulation parameters, each of the two deformable bodies is modelled as a Cosserat rod of length $\ell = 0.47$ m with width $w = 2.5$ mm and height $h = 10$ mm, along the rod $n$ and $a$ directors, respectively. Their material parameters are $E = 36$ GPA for the Young modulus and $\rho = 1900 \frac{\text{kg}}{\text{m}^3}$ for specific weight. For their strain based parametrization we used three modes, for the bending along their $a$ director, while for the numerical integration of their kinematics and dynamics ODEs, respectively (4.21) and (4.25), we used 31 Chebyshev points [40]. Finally, for the time integration, we used a spectral radius $\rho_\infty = 0.5$ and $\Delta t = 5$ ms.

### Experimental Setup

The robot actuation is composed of two DC Maxon motors DCX32L for which we imposed a joint-angles trajectory, while recording their rotation angles with the build-in encoders, at a frequency $f_s = 200$ Hz. We then proceed to filter those angles using a Butterworth filter of order 4 with cut-off frequency $f_c = 2$ Hz. The derivatives were computed with the central difference using a time step of $\Delta t = 5$ ms.



(a) $t = 0$ s (b) $t = 1.26$ s (c) $t = 2.09$ s

(d) $t = 2.16$ s (e) $t = 2.26$ s (f) $t = 3.03$ s

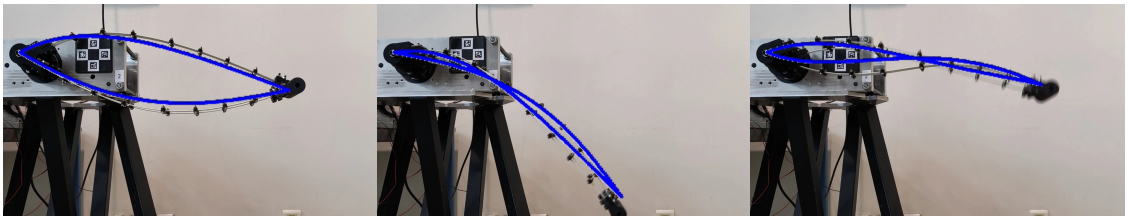
Figure 4.11: Snapshots of the motions of the RFRFR CPR bouncing when stopping at the singularity configuration $q_1 = q_2 = 0$ showing the simulated behaviour (blue lines) superposed to the recorded motion. The snapshots were taken at the time indicated by the vertical lines of Figure 4.10a, where the letters refers to the sub-figures (a–f) here presented.

### Comparison of Simulated Behaviours

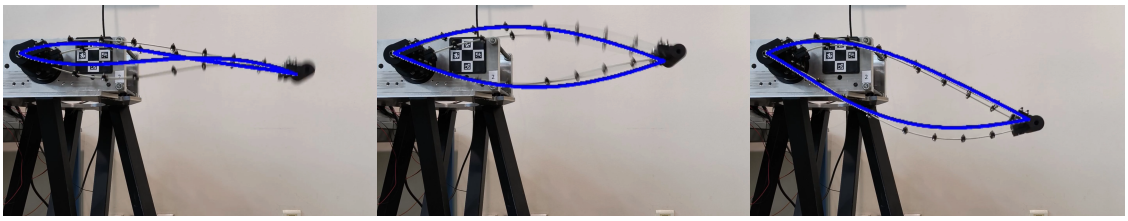
This section presents a qualitative comparison of the model with the recorded motion on the RFRFR CPR prototype. In order to achieve this, the robot is controlled by imposing a fifth-order polynomial trajectory on $q_1$ and $q_2$. Two distinct trajectories will be employed to illustrate the robot's behaviour in those scenarios, both of which involve the interaction with a transition stable-unstable-stable when the two proximal revolute joints are aligned. In both cases, we begin with the values of $q_1 = \frac{1}{6}\pi$ and $q_2 = -\frac{1}{6}\pi$, which define the reference configuration of the CPR. Then we proceed as follows for the two motions.

**First Trajectory** In this first case, the joint trajectory aligns the two proximal joints reducing their relative angle of rotation (Figure 4.10a). Starting from the reference configuration (Figure 4.11a) the robot start moving decreasing the relative angle between the proximal revolute joints (Figures 4.11b and 4.11c). Upon reaching this singular position (Figure 4.11d), the robot loses its stiffness and the platform drops under the influence of gravity (Figure 4.11e). Thanks to the elasticity of the deformable bodies, the limbs and platform start bouncing back (Figure 4.11f), which makes the robot to behave as a tilted pendulum (Figures 4.11d– 4.11f). In summary, from Figures (4.11a–4.11f), we can state that our model successfully captured the stable-unstable-stable transition which was the focus of this experience.

**Second Trajectory** In the second case, the same singular configuration is employed to stimulate the robot in a different manner. This time, the singularity is crossed instead of



(a) $t = 0$ s (b) $t = 1.27$ s (c) $t = 2.10$ s



(d) $t = 2.11$ s (e) $t = 2.12$ s (f) $t = 2.20$ s

Figure 4.12: Snapshots of the motions of the RFRFR CPR when crossing the singularity configuration $q_1 = q_2 = 0$ showing the simulated behaviour (blue lines) superposed to the recorded motion. The snapshots were taken at the time indicated by the vertical lines of Figure 4.10b, where the letters refers to the sub-figures (a–f) here presented.

being stopped at. In order to achieve this, the proximal joints are gradually rotated until their values are reversed, resulting in $q_1 = -\frac{1}{6}\pi$ and $q_2 = \frac{1}{6}\pi$ (Figure 4.10b). As in the previous case, the robot starts in the reference configuration (Figure 4.12a), upon reaching the singularity condition, the robot loses its stiffness, resulting in the distal plate falling under the influence of gravity (Figure 4.12b). As the proximal joints keep rotating, their relative angle increases together with the robot stiffness and the limbs, with the platform, start to rise (Figure 4.12c). However, the robot is now in an unstable configuration, characterised by the "S" shape of its deformable bodies (Figures 4.12b and 4.12c). Increasing even further the proximal joints relative angle, the unstable configuration disappears (Figure 4.12d) and the robot snaps into the stable configuration similar to the reference one (Figure 4.12e). The abrupt movement releases the internal elastic energy, thereby accelerating the platform and initiating oscillatory motion while the proximal joints reach the desired angles (Figures 4.12e and 4.12f). Figures (4.12a–4.12f) illustrate the efficacy of our model in capturing the motion depicted.

### Comment on the Results

In conclusion, our model has been demonstrated to be capable to capture transition stable-unstable-stable in two different scenarios. In both cases, the proposed approach was able to capture the moment of each transition and the corresponding behaviour.

### 4.9.3 Conclusions on Experimental Results

In summary, we used two planar CPRs to validate the proposed approach. We studied two distinct cases: a prototype with distributed actuation and non negligible friction and another with localized actuation presenting instabilities. In both cases, our simulations were able to reproduce the recorded behaviour. For the first prototype, we were able to measure different quantities during the robot motion, such as its joint angles and rods shapes. We used some of those measures as input for the dynamics model while we compared the output with the rest of the dataset. With this approach, we were able to validate different parts of our approach. On the other hand, with the second prototype, we were able to demonstrate that our approach is able to reconstruct the behaviour of a CPR even when crossing a singularity. To this aim, two cases were studied: one where the robot stays in the singular configuration and the other where it crosses that singularity, with an appreciable dynamics response.

## 4.10 Conclusion

In this chapter, we specialized the approach from Chapter 3 for the dynamics modelling of CPRs. In this context, the virtual cuts were used to isolate the CPR platform from the limbs, creating two subsystems. NE passes were used to compute their dynamics states, which are coupled by the geometric constraints and corresponding contact wrenches. For time integration of the resulting system of DAEs, the Generalized-$\alpha$ scheme was employed due to its optimal damping ratio (i.e., minimizing the damping of system dynamics while controlling the damping of numerical noise). This model was applied to two planar CPRs: one with distributed (tendon-driven) actuation of the deformable rods, and the other with localized actuation at the proximal joints. The first prototype, developed in collaboration with the Continuum Robotics Laboratory, was used to isolate and validate different components of the Lagrangian dynamics model based on recorded data during the robot's motion. The second prototype was used to validate the model with recorded motion of the robot bouncing and snapping under gravity, successfully capturing its dynamic behaviour. The similarity to the general approach in Chapter 3, along with its recursive nature, led to the development of the approach as a generic toolbox, which will be discussed in the next chapter.

# Chapter 5

# NEHA: A Newton-Euler Hybrid Algorithm Toolbox

The field of continuum robotics has been experiencing a growing interest, which began over the past decade. For modelling slender deformable bodies, various solutions, with different theoretical and implementation complexities, were discussed in Chapter 2. As the understanding of these deformable bodies advances, more complex systems, such as Continuum Robots (CRs) and Continuum Parallel Robots (CPRs), are being explored. For the control and analysis of such systems, numerical simulations are essential. This drives the need for computational tools, or toolboxes, that offer ready-to-use functionalities, thus relieving users from the burden of developing custom solutions from scratch.

In this context, the SOFA framework [111] stands out as a comprehensive solution, offering a broad range of algorithms, including lumped models, FEM, PCC, and advanced collision detection methods, all within a unified environment. It supports the use of constraints and interaction forces as well as both implicit and multi-step explicit time integrators. Although FEM is not ideally suited to the specific requirements of robotics, SOFA has made efforts to adapt FEM to better meet those needs. Additionally, within the COSSEROOTS ANR project, SOFA expanded its capabilities to include Cosserat rod theory, under the geometrically exact assumptions of the VSA approach. Despite these efforts, its generality and completeness still require users to have a solid understanding of the toolkit, along with familiarity with FEM modelling. Though tutorials and documentation are available, SOFA presents a significant learning curve for those unfamiliar with these concepts.

Another widely used simulator in the literature is SoRoSim [78], which is based on the VSA (Section 2.3.6). In this case, the VSA is employed to compute the numerical Jacobians that relate the kinematics and dynamics to the configuration variables (generalized elastic coordinates). This toolbox supports modelling, simulation, control, and optimization of various robotic systems, including: single-branched open and closed-chain configurations composed of soft and/or rigid components, with different actuator setups and external loading conditions. The system dynamics is integrated over time using explicit integration

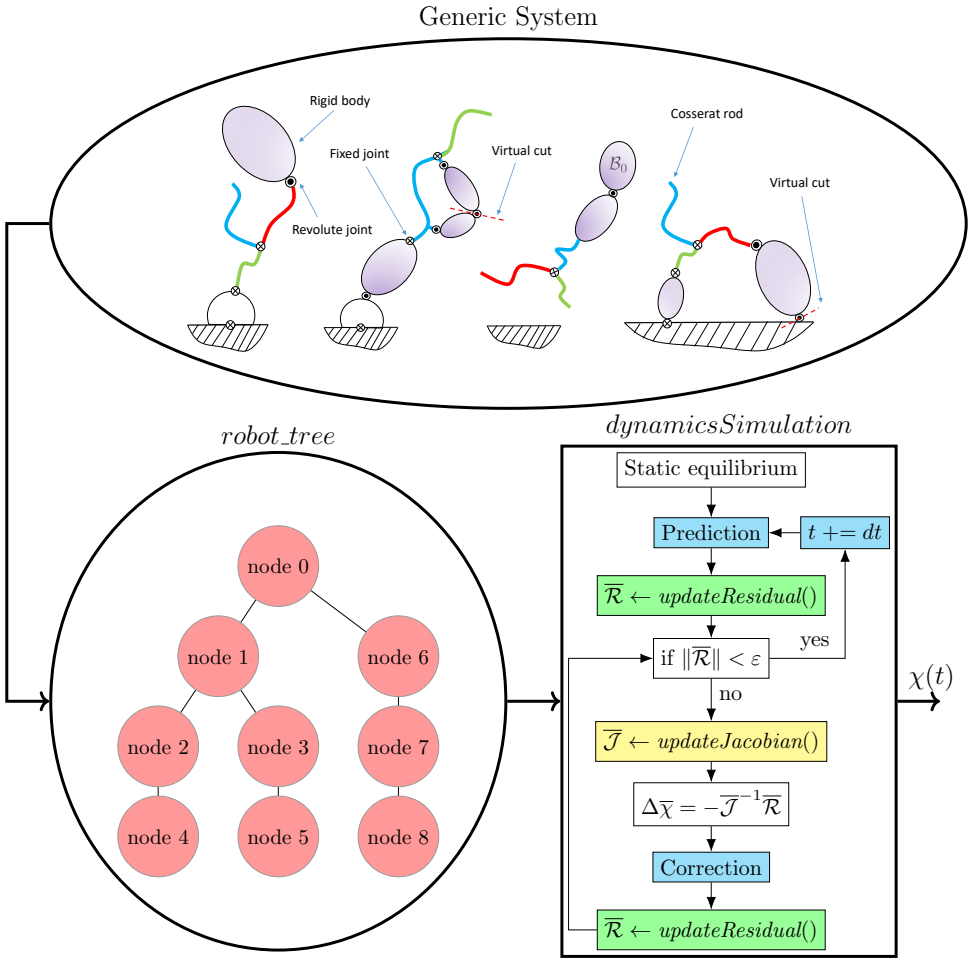



Figure 5.1: Different types of SMMS discussed in Chapter 3, which can be converted into a generic tree of nodes used for the dynamics simulation. The simulation is based on a time implicit scheme and gives as output the state of the system $\chi$ over time.

schemes. However, these schemes are not well-suited for the dynamics of stiff systems like CPRs [101]. Additionally, the use of NE passes to compute kinematics and dynamics Jacobians diverges from the initial objective of utilizing the standard $IDM$ and $TIDM$ routines from rigid robotics.

Finally, we highlight the PyElastica toolbox [61, 64], an open-source framework based on Cosserat rod theory. It is designed for simulating slender deformable bodies and structures composed of rod assemblies, enabling dynamics simulations of both natural and artificial systems. This toolkit employs the DER approach (Section 2.3.2) alongside an explicit time integration scheme. However, PyElastica does not allow for the suppression of shear or stretch, which act as high-penalty factors, making the dynamics stiff and difficult to integrate. This, combined with the use of an explicit scheme, heavily restricts the time step degrading numerical performances. Furthermore, PyElastica lacks traditional

actuation systems commonly used in robotics, such as actuated revolute joints, and the distributed actuation methods of continuum robotics (*e.g.,* tendon-driven actuations). It also does not provide statics modelling, which is crucial in continuum robotics, as many problems are best addressed in the static regime before moving to dynamic analysis.

Our toolbox should provide several key features: *i*) the ability to assemble the system and position it in a desired configuration before starting dynamics simulations, *ii*) support for both standard rigid and continuum robotics actuation systems, with the flexibility to implement new ones, *iii*) reliance on the numerical computation of the dynamics model ($IDM$ and $TIDM$), *iv*) use of a local parametrization approach (VSA) to simplify usage and maintain a relatively small problem size, *v*) an implicit time integration scheme to allow numerical stability and flexibility in the choice of time-step and *vi*) be implemented in C++ to enhance numerical performances.

More in detail, the implementation leverages the RNEAs to numerically compute the dynamics model, utilizing forward and backward passes, as is standard in rigid robotics [86, 87]. The objective is to develop $IDM$ and $TIDM$ routines that are adaptable to various system types (locomotor, manipulator, or parallel robot), body types (rigid or cross-section), and joint types (rigid or continuum joint), requiring a flexible system representation. To achieve this, the system is modeled as a dynamic tree, a widely-used method in rigid robotics [86, 112]. With "dynamic tree" we mean a sequence of elements, referred as nodes, which is designed to accommodate structural variations during simulation. This tree is used as input to the simulator, which computes the system's dynamics state at each time step through an implicit scheme. Figure 5.1 illustrates this concept, with different colors: red for system structure, blue for the time implicit scheme, green for residual vector computation, and yellow for the computation of its Jacobian matrix.

In order to implement such toolbox, we need to encapsulate the different aspects of the dynamics simulation (implicit scheme, bodies, joints) in some components able to interact and be assembled together. This kind of problems is typically addressed using an Object Oriented Programming paradigm which is the subject of the next section.

## 5.1 Object Oriented Programming

Object Oriented Programming (OOP) is a paradigm where the user defines a component (or object) as an autonomous unit having some predefined properties [113]. Specifically, an object allows to encapsulate some related attributes, or members, and provides some interfaces with the inputs and outputs of its functions, or routines. OOP allows for the definition of hierarchies among objects. This functionality is useful when multiple objects share the same interfaces (inputs and outputs) but need to modify, or override, the existing implementation. Routines that can be overridden are referred to as virtual, and an object containing virtual routines is called abstract, or base. On the other hand, the objects with function(s) override are referred as children, or specializations. The interest in this approach resides in using the base and child(ren) objects indistinctly, as they pro-

vides identical interfaces, while the actual implementation of the functions is automatically selected during the program execution.

In summary, an object represents a modular part of a system that encapsulates its contents. Its manifestation can be replaced within its environment at design time or run-time by another object, based on the compatibility of their interfaces. The Newton-Euler recursive algorithms of Chapters 3 and 4 are well suited for OOP with their recursive and sequential steps. However, before diving into the implementation of such approaches, we first define the notations and representations of object using the standard Unified Modelling Language UML (version 2.5.1) [114].

## 5.2 Unified Modelling Language and Conventions

We represent an object with an UML component [114], which can be used to define software systems of arbitrary size and complexity. A number of UML standard stereotypes exist that apply to a component. In this chapter, it is represented as a rectangle with a «keyword» indicating the type, either «component» or «abstract», placed above the object name, which follows the *UpperCamelCase* naming convention. In the upper right-hand corner, an icon, consisting of a rectangle with two smaller rectangles protruding from its left side, may be displayed. If the icon symbol is shown, the keyword «component» is omitted. In terms of attributes, an object member is designed by an ellipse and denoted using the *snake_case* convention while the routines are designed by blocks and denoted using the *camelCase* convention (followed by a pair of parentheses () when used in the Algorithms). The latter may have other information, such as the required ad provided interfaces (if any) and possibly how it interacts with the members. As standard UML notation, a provided interface is shown using the "lollipop" notation while a required interface is shown using the "fork" notation. Internally to the object, if a function makes use of another one it is indicated by an arrow while an interaction with a member is indicated by a simple line, for which the meaning could be reading, writing or both.

Once introduced the notation for the components, we can give a global overview of the software architecture of our toolbox.

## 5.3 Software Architecture of NEHA

This section presents the software architecture of our toolbox: NEHA. To this aim, we will use a dependency graph, which demonstrates the various layers of a software application with a representation from the highest level of abstraction (top of the graph) to the low level implementation (bottom of the graph). Arrows are employed to indicate the usage, or dependence, of an object with respect to another. A dashed arrow is used to express inheritance, whereby the pointed object represents the specialization. At each level, the components are are grouped in subsystems. In our case, the dependencies of the various components are shown in Figure 5.2. At the top we find the *neha* group, which stands for Newton-Euler Hybrid Algorithm, with the *DynamicsSimulator* object. This object uses an

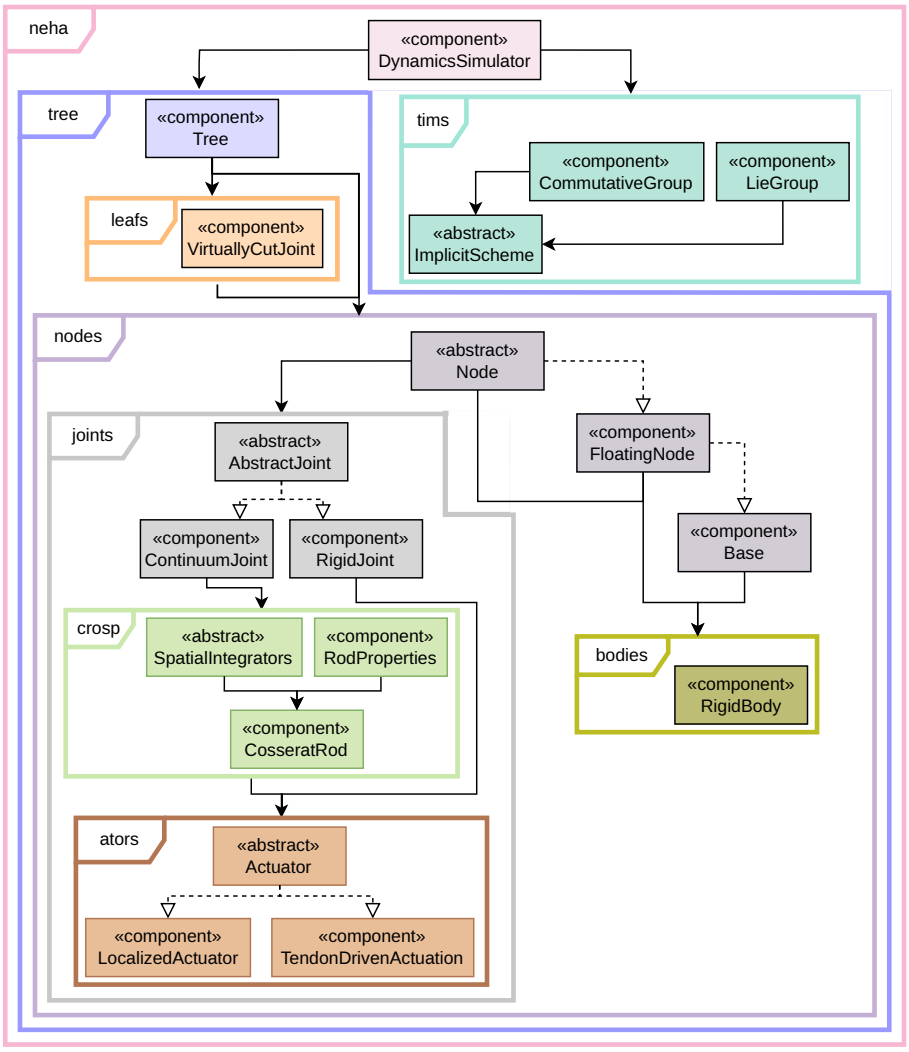



Figure 5.2: Dependency graph detailing the different groups, object and hierarchy used to define the NEHA toolbox.

implicit scheme to work on the generic dynamic tree. The former belongs to the *tims* group, which stands for "time implicit schemes". This group contains the abstract *ImplicitScheme* object, meant to define the interfaces that all the implicit schemes shares, while the two component of *CommutativeGroup* and *LieGroup* defines the integration steps required by a commutative group (*i.e.,* a vector space) and for the Lie Group, which requires a sightly different approach as discussed in Chapter 3 and 4. The other group, namely *tree*, contains the *Tree* object, which is responsible to represent the system as a generic tree of nodes, and two subgroups of components: the *leafs* and the *nodes*. The former group contains the *VirtuallyCutJoint* object, used by the *Tree* in case there are one, or more, closed kinematics loop to open, while the latter group contains the *Node*, *FloatingNode* and *BaseNode* objects. All types of nodes contains a body while only the *Node* object contains

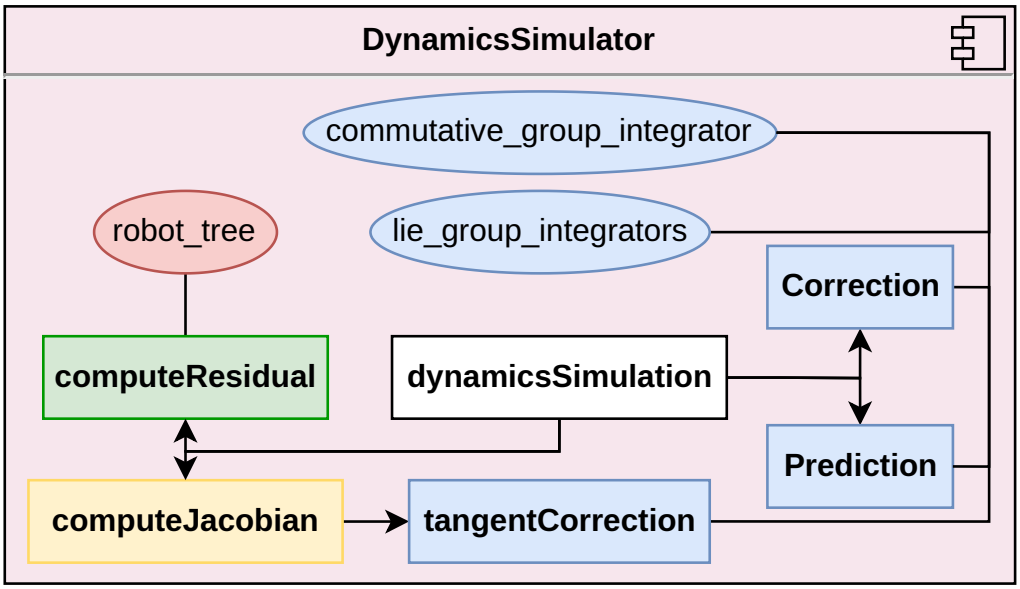



Figure 5.3: Component diagram for the *DynamicsSimulator* object.

a joint. Both rigid bodies and joints are described in their respective groups, denoted *bodies* and *joints*. These two groups, located at a lower level of abstraction, provide the actual implementation of the forward and backward passes depending on their type, as detailed in Chapter 3. Joints can be of type *RigidJoint* or *ContinuumJoint*, the latter, introduced in Section 3.5.4, consist of an encapsulation of a Cosserat rod to work in the NE formalism. The modelling of Cosserat rods is implemented within the *crosp* group, which stands for Cosserat Rod Strain Parametrized, and contains the *CosseratRod* object. This object links the *RodProperties* to the *SpatialIntegrators* to define the dynamics modelling of a rod (Section 2.1). Finally, a joint, either rigid or continuum, can be provided with some sort of actuation, which is implemented in the *atros* group, short for actuators. In this group we find the *Actuator* abstract object and the two specialization of a *LocalizedActuator* and *TendonDrivenActuation*.

Finally, a color is associated to every group of components which will be maintained thought the chapter together with the ones defined in Figure 5.1. In the following, we describe all the groups of Figure 5.2 from the highest to the lowest level of abstraction, thus starting with the *DynamicsSimulator* object. We then define how the different time implicit schemes and integrators are generically implemented. Once the constraints of the implicit schemes are accounted for, we proceed describing how the NE segmentation of a system can be converted in a generic dynamic tree of nodes. Finally, we will illustrate the usage of our toolbox with a simulation of a case study planar CPR. This simulation will also be used to address the comparison of the different time implicit schemes and spatial integrators for the Cosserat ODEs.

**Inputs:** $t$
**Function** `dynamicsSimulation()`:
  **for** ***every*** *step* ***in*** $[1, steps]$ **do**
    $t \mathrel{+}= dt$
    $robot\_tree \rightarrow updateActuations()$

    $Prediction()$

    $computeResidual()$

    **while** $\left\|\overline{\mathcal{R}}\right\| > \varepsilon$ **do**
      $computeJacobian()$
      $\Delta\overline{\chi} = -\overline{\mathcal{J}}^{-1}\overline{\mathcal{R}}$
      $Correction()$
      $computeResidual()$
    **end**
    $updateImplicitSchemes()$
  **end**

**end**

**Algorithm 1:** The Algorithm for the layout of the *dynamicsSimulation*

**Function** `Prediction()`:
  **for** ***every*** *floating_node* ***and*** *lie_group_integrator* ***in***
      $robot\_tree \rightarrow getFloatingNodes()$ ***and*** *lie_group_integrators* **do**
    $lie\_group\_integrator \rightarrow computePrediction()$
    $floating\_node \rightarrow assignKinematics()$
  **end**

  $commutative\_group\_integrator \rightarrow computePrediction()$
  $robot\_tree \rightarrow assignCoordinates()$
**end**

**Algorithm 2:** The Algorithm for the layout of the *Prediction*

## 5.4 Dynamics Simulator

The *DynamicsSimulator* object works at the highest level of abstraction (Figure 5.2) and mostly provides the functionalities to use the tree representation of the system with the constraints of an implicit scheme. This object, represented in Figure 5.3, contains the members *robot_tree*, *commutative_group_integrator* and *lie_group_integrators*. The former integrator is used for the generalized coordinates of the system, while the latter is a list of integrators for all the floating bodies of the system, if any. The routines of this object are detailed in the following.

**Inputs:** $\Delta\overline{\chi}$
**Function** `Correction()`:
  **for** ***every*** *floating_node* ***and*** *lie_group_integrator* ***in***
      *robot_tree* $\rightarrow$ *getFloatingNodes*() ***and*** *lie_group_integrators* **do**
    *floating_node* $\rightarrow$ *getKinematics*()
    *floating_node* $\rightarrow$ *getSOR*()
    *lie_group_integrator* $\rightarrow$ *computeCorrection*()

$$g_0^{k+1} = g_0^k \widehat{\Delta\zeta}_0, \qquad \eta_0^{k+1} = \eta_0^k + \Delta\eta_0, \qquad \dot{\eta}_0^{k+1} = \dot{\eta}_0^k + \Delta\dot{\eta}_0$$

    *floating_node* $\rightarrow$ *assignKinematics*()
  **end**

  *robot_tree* $\rightarrow$ *getCoordinates*()
  *commutative_group_integrator* $\rightarrow$ *computeCorrection* ()
  *robot_tree* $\rightarrow$ *assignCoordinates*()
**end**

**Algorithm 3:** The Algorithm for the layout of the *Correction*

### 5.4.1 The *dynamicsSimulation*

The *dynamicsSimulation* implements the flowchart of Figure 5.1, and is detailed in Algorithm 1. This routine takes as input $t$ specifying the amount of time to simulate the system which is used to compute the number of steps of the implicit schemes. Then, from a suitable statics configuration, it starts the time loop of the implicit schemes which is implemented by the *Prediction* and *Correction* routines. The latter will ve used in a Newton-Raphson loop with the *computeResidual* and *computeJacobian* routines. Additionally, a dedicated routine is used to update the actuation law of the system actuators.

### 5.4.2 The *Prediction*

The *Prediction* routine computes the parametrization of the kinematics state of the system in a generic way using the dynamic tree and the time integrators (Algorithm 2). Specifically, from the *robot_tree* it accesses the list of floating nodes, through the *getFloatingNodes* function, which is then paired with the *lie_group_integrator*. Looping through these coupled lists allows to work with the pair of a *FloatingNode* and its corresponding *LieGroup* time integrator. For each of this pair, the prediction is computed by the *lie_group_integrator* with *computePrediction*. The obtained kinematics is then assigned to the *floating_node* using *assignKinematics*. Once updated the kinematics of all floating nodes, the *commutative_group_integrator* is used to compute the prediction on the generalized coordinates of the system which are assigned to every joint with the *assignCoordinates* routine of the *robot_tree*.

**Function** `computeResidual()`:
  *robot_tree* → *forwardKinematics*()

  *robot_tree* → *backwardDynamics*()

**end**

**Algorithm 4:** The Algorithm for function *computeResidual*

### 5.4.3 The *Correction*

The *Correction* routine is used to apply a change of $\Delta\overline{\chi}$ into $\chi$ (Algorithm 3). Again, this routine works on the pair of a floating node and the corresponding time integrator. However, in this case, it first retrieves the current kinematics state of the floating node, $(g, \eta, \dot{\eta})^k$, and $\Delta\nu^{k+1}$ using the routines *getKinematics* and *getSOR*, respectively. The latter is used to map the section of $\Delta\overline{\chi}$ corresponding to the floating node to its $SO(3)\times\mathbb{R}^3$ parametrization of $\Delta\nu$, as will be detailed in Section 5.8. Afterward, the correction is computed with the *computeCorrection* of the implicit scheme taking as input $\Delta\nu^{k+1}$ and $(g, \eta, \dot{\eta})^k$ and computing the $(\Delta\zeta, \Delta\eta, \Delta\dot{\eta})^{k+1}$ which is then mapped into $(g, \eta, \dot{\eta})^{k+1}$ using the expression in the Algorithm. Finally, the obtained kinematics state is given to the node with the *assignKinematics* routine. A similar approach is used for the generalized coordinates of the system. First, the current coordinates are obtained with the *getCoordinates* of the *Tree* object, while $\Delta q$ is extracted from $\Delta\overline{\chi}$. Then, the correction is computed with the *computeCorrection* routine and, finally, the updated coordinates are assigned to the joints with the *Tree*'s *assignCoordinates* routine.

### 5.4.4 The *tangentCorrection*

In order to compute $\overline{\mathcal{J}}$ we need to map the unitary perturbation of $\Delta\overline{\chi}$ into $\Delta\chi$. This step is achieved by the *tangentCorrection* routine, which uses all the steps of the *Correction* but computing $\left(\Delta\zeta_0^{k+1}, \Delta\eta_0^{k+1}, \Delta\dot{\eta}_0^{k+1}\right)$. The computation of the unitary perturbations, and their propagation to the first and second derivatives, is obtained by calling *computeCorrection* routine using $(\mathbb{1}_4, 0, 0)$ or $(0, 0, 0)$ for a floating node or the generalized coordinates, respectively. This function then returns the tangent kinematics state which is assigned to the floating nodes with their *assignTangentKinematics* routine and to the joints with the *Tree*'s *assignTangentCoordinates* routine.

### 5.4.5 Computation of $\overline{\mathcal{R}}$ with *computeResidual*

The Lagrangian dynamics model, in the form of a residual vector, can be computed for any type of system, with the *computeResidual* routine (Algorithm 4). This routine consists of a calling the *forwardKinematics* of the *robot_tree* which defines the kinematics state of the whole system. Then, calling the *backwardDynamics* on the *robot_tree* allows to evaluate the geometric constrains, computing all the $\Upsilon_i$ which compose $\overline{\overline{\mathcal{R}}}_\Upsilon$, and to compute the contact wrenches, defining the initial condition for this backward pass which then defines

**Function** `computeJacobian()`**:**

**for** $i = 1, \ldots, dim(\overline{\chi})$ **do**
$\Delta\overline{\chi} \leftarrow 0$
$\Delta\overline{\chi}[i] = 1$

*TangentCorrection*()

*robot_tree* $\rightarrow$ *forwardTangentKinematics*()

*robot_tree* $\rightarrow$ *backwardTangentDynamics*()
**end**
**end**

**Algorithm 5:** The Algorithm for function *computeJacobian*

the kinematics state of the system and the balance at all of its joints, completing global residual vector $\overline{\mathcal{R}}$.

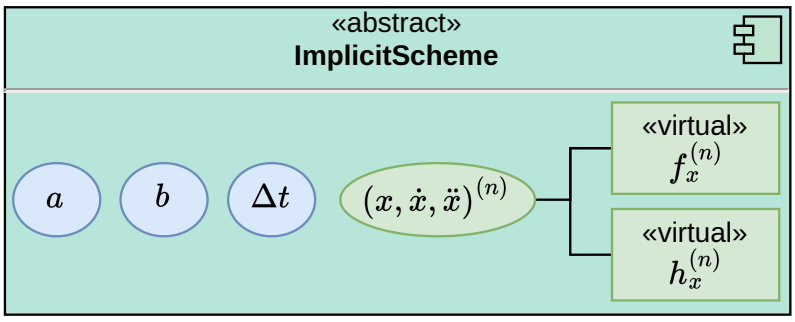
### 5.4.6 Computation of $\overline{\mathcal{J}}$ with *computeJacobian*

The Jacobian matrix is computed iteratively using the $TIDM$ of Chapter 3 fed by a series of sequential unitary changes of all the entries in $\overline{\chi}$ (Algorithm 5). In details, the function loops in the vector $\Delta\overline{\chi}$ to set an unitary change for each of its entries, and the *TangentCorrection* is used to propagate the variation $\Delta\overline{\chi}$ to $\Delta\chi$. Then, the *forwardTangentKinematics* and, subsequently, the *backwardTangentDynamics* routines are called. The latter, defines its initial condition before computing the tangent dynamics balance at all the joints, which sequentially fills the vector $\Delta\overline{\mathcal{R}}$. This vector will then be used to fill the Jacobian column-wise as detailed in Chapter 4.

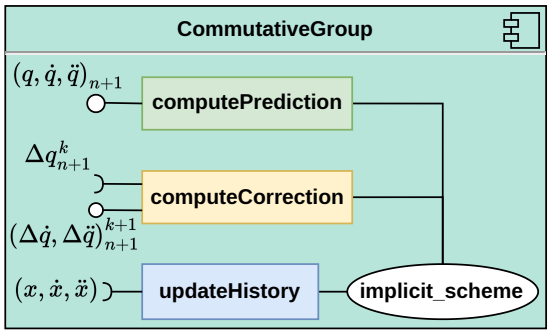
### 5.4.7 Update the Implicit Schemes

Finally, before proceeding to the next time step, the implicit schemes are updated with the *updateImplicitSchemes* routine which assigns the $(g, \eta, \dot{\eta})$ of every floating nodes and $(q, \dot{q}, \ddot{q})$ to the respective implicit scheme with their *updateHistory* routine which will be detailed in Section 5.5.

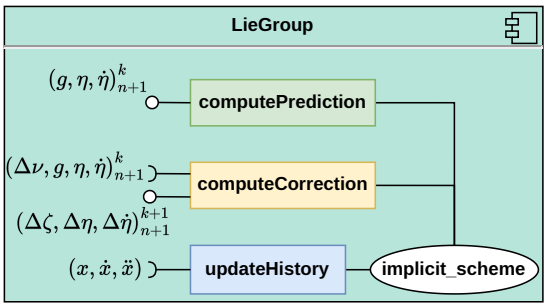
The implementation of the different time implicit schemes, which form the outer layer in the dynamics simulation, is the subject of the next section.




(a) Representation of the *implicit_ scheme* abstract object.




(b) Representation of the *CommutativeGroutp* implementation.




(c) Representation of the *LieGroup* implementation.

Figure 5.4: UML component diagrams for the different objects used to implement the implicit schemes.

## 5.5 Implicit Schemes for Time Integration

In Chapter 3 and 4 we have established that, for a hybrid (rigid-deformable) multi-body system, the dynamics model has the following Lagrangian form:

$$\begin{cases} \begin{bmatrix} 0 \\ Q_{act} \end{bmatrix} & = \begin{bmatrix} \mathcal{M}_0 & M_{0q} \\ M_{q0} & M_{qq} \end{bmatrix} \begin{bmatrix} \dot{\eta}_0 \\ \ddot{q} \end{bmatrix} + \begin{bmatrix} F_0 \\ Q \end{bmatrix} - \begin{bmatrix} J_0^T \\ J_q^T \end{bmatrix} \lambda, \\ \Upsilon\left(g_0, q\right) & = 0, \end{cases} \tag{5.1}$$

which defines a closed formulation of the system dynamics model, that, we remind, is parametrized by the set of state variables:

$$\chi = \left(g_0, \eta_0, \dot{\eta}_0, q, \dot{q}, \ddot{q}, \lambda\right), \tag{5.2}$$

where the suffix 0 is replaced by $p$ in the case of a CPR platform. The implicit schemes use their constraints to relate the first and second derivatives to the current configuration and the previous kinematics state(s), reducing (5.2) to the following set of configuration variables:

$$\overline{\chi} = \left(\nu_0, q, \lambda\right), \tag{5.3}$$

where $\nu_0 = \left(\Theta_0^T, d_0^T\right)^T$. Here, for a generic group $G$ we recall the formulation for the prediction for one of its element $x \in G$:

$$x = x^{(n)} \exp_G (d_x), \qquad \dot{x} = a \cdot d_x + f_x^{(n)}, \qquad \ddot{x} = b \cdot d_x + h_x^{(n)}, \tag{5.4}$$

where $d_x$ is a change of $x$, $f_x^{(n)}$, $h_x^{(n)}$, $a$ and $b$ are respectively two functions and two scalar coefficients that depend on the implicit scheme while $\exp_G (d_x)$ is the exponential of the Group $G$. If the group is commutative (*i.e.,* a vector space) it corresponds to the identity map, and the first relation of (5.4) becomes $x = x^{(n)} + d_x$. For the non commutative Lie-Group of the $SO(3)$ orientation, the $\exp_{SO(3)}$ is the one defined by [50]. On the other hand, the correction maps a perturbation of $\Delta x$ on $(x, \dot{x}, \ddot{x})^k$ to obtain $(x, \dot{x}, \ddot{x})^{k+1}$ using the following expressions for a vector space (commutative group):

$$x^{k+1} = x^k + \Delta x, \qquad \dot{x}^{k+1} = \dot{x}^k + a\Delta x, \qquad \ddot{x}^{k+1} = \ddot{x}^k + b\Delta x, \tag{5.5}$$

while, for the Lie group, we recall the formulation of Chapter 3:

$$\Delta\zeta_0 = \frac{\partial C}{\partial \nu}\Delta\nu_0, \quad \Delta\eta_0 = \frac{\partial A}{\partial \nu}\Delta\nu_0, \quad \Delta\dot{\eta}_0 = \frac{\partial B}{\partial \nu}\Delta\nu_0, \tag{5.6}$$

### 5.5.1 A Generic Implicit Scheme

Many implicit integration exist in the literature, including: the Newmark [105], the Hilber-Hughes-Taylor (HHT) [106], the Wood-Bossak-Zienkiewicz (WBZ-$\alpha$) [107] and, finally, the Generalized-$\alpha$ [108, 109], which all share these input-output interfaces for their computations. We can then define an abstract, or base, object *ImplicitScheme* (Figure 5.4a). This object contains the time-step $\Delta t$ and the $a$ and $b$ coefficients used in the prediction and correction and $(x, \dot{x}, \ddot{x})^{(n)}$ as the history of the kinematics state of the previous steps, containing as many element as required by the implicit scheme (*e.g.,* 1 for Newmark integrator and 2 for the implicit Euler). It also provides the interface for the implementation of $f_x^{(n)}$ and $h_x^{(n)}$, to be defined accordingly to the chosen scheme.

### 5.5.2 Commutative and Lie Group Integrators

The *ImplicitScheme* object is used as a member of both *CommutativeGroup* and *LieGroup* integrators (Figures 5.4b and 5.4c). These objects implement (5.4) in *computePrediction* using the identity map or the $\exp_{SO(3)}$, and, similarly, the correction using either (5.5) or (5.6) in *computeCorrection*, for the *CommutativeGroup* and *LieGroup* respectively. In this case, inheritance cannot be used, as the functions take different types and number of parameters. Finally, the routine *updateHistory* that takes as input the kinematics state $(x, \dot{x}, \ddot{x})$, solution of (5.1) at time $t_{n+1}$, to be stored in the history $(x, \dot{x}, \ddot{x})^{(n)}$ of the *implicit_scheme*.

The outer time integration loop is so defined for any kind of system. In the following, we describe the tree of nodes which is used to generically compute the residual and its Jacobian matrix.

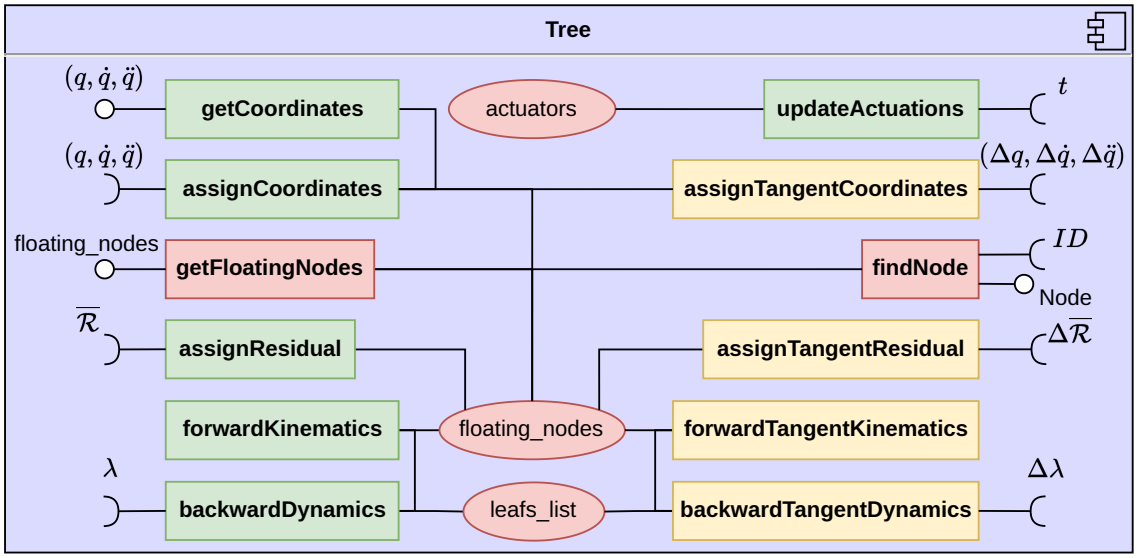



Figure 5.5: UML component diagram for the *Tree* object.

**Function** `forwardKinematics()`:
  **for** ***every*** *floating_node* ***in*** *floating_nodes* **do**
    *floating_node* $\rightarrow$ *forwardKinematics*()
  **end**
**end**

**Algorithm 6:** The Algorithm for the Tree *forwardKinematics*

## 5.6 Dynamic Tree

In this section, we detail the *Tree* object, which is designed to handle the requirements of the prediction-correction algorithm of the implicit schemes. Specifically, this object is responsible to provide a generic representation of the system and the functionalities required to compute the residual vector $\overline{\mathcal{R}}$, and its Jacobian matrix $\overline{\mathcal{J}}$. A tree is a structure made of nodes interconnected with a single parent-multiple children layout (Figure 5.1). As discussed in Chapters 3 and 4, if the tree has some kinematics loops, these are removed with some virtual cuts, until only open serial branches are left. In this context, each of these virtual cuts creates two "leafs", which correspond to a virtual rigid body connected through a fixed joint to the parent node. In terms of members, the tree has three lists, namely: *leafs_list*, *actuators* and *floating_nodes*. These lists are used to have direct access on: the virtually cut joints, the actuators and the floating nodes, respectively. The primary goal of this object is to provide access to the elements of the tree for NE passes. In the following, we describe the *forwardKinematics* and *backwardDynamics* routines. For conciseness, we do not cover other routines, which are used to retrieve or assign the generalized coordinates, residuals, and their variations.

**Inputs:** $\lambda$
**Function** `backwardDynamics()`:
  **for** ***every*** *cut_joint* ***in*** *leafs_list* **do**
    *cut_joint* $\rightarrow$ *computeGeometricConstraints*()
    *cut_joint* $\rightarrow$ *computeContactWrenches*()
  **end**

  **for** ***every*** *floating_node* ***in*** *floating_nodes* **do**
    *floating_node* $\rightarrow$ *backwardDynamics*()
  **end**
**end**

**Algorithm 7:** The Algorithm for the Tree *backwardDynamics*

### 5.6.1 The Forward Kinematics

The *forwardKinematics* routine, used to compute the kinematics state of the whole system, is described in Algorithm 6. It requires no input and simply calls the *forwardKinematics* routine on every member of the *floating_nodes* list, which, in turn, starts a recursive step on all the nodes, as will be detailed in Section 5.8.

### 5.6.2 The Backward Dynamics

The *backwardDynamics* routine (Algorithm 7) takes as input the set of Lagrange multipliers $\lambda$, and starts by defining the geometrical constraints and its initial condition (*computeGeometricConstraints* and *computeContactWrenches* routines). Then, it calls the *backwardDynamics* of every element of the *floating_nodes* list which computes the dynamics state of the system and the dynamics balance of all the joints, as will be detailed in Section 5.8.

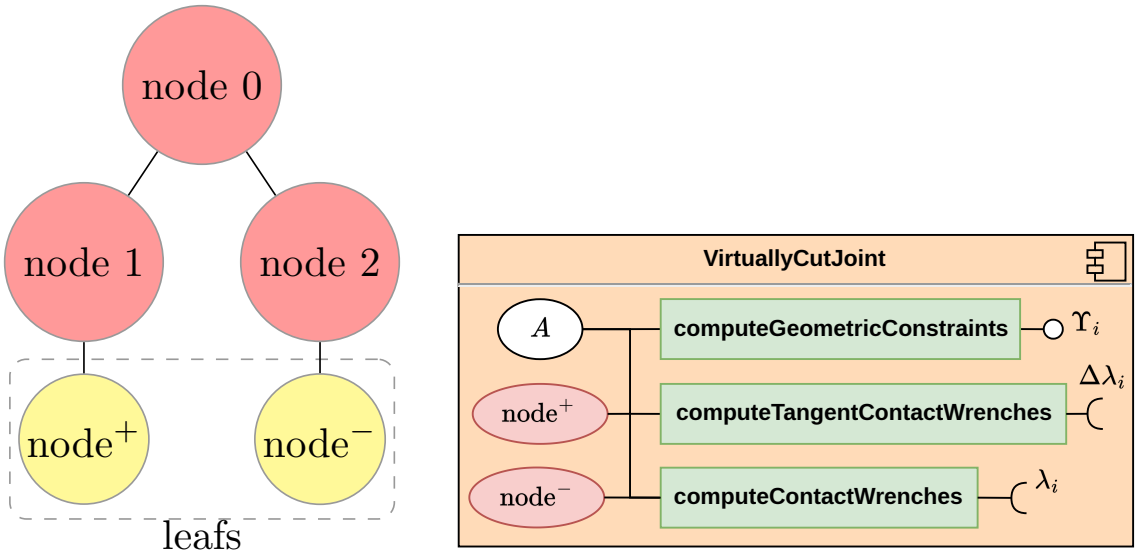
The forward and backward passes work sequentially on nodes and leafs of the tree. In the following, we detail these two elements specifying how they contribute in encapsulating the relative sequential steps of the NE passes.

## 5.7 Leafs

In this section we describe how the geometric (holonomic) conbstraints, and the corresponding contact wrenches, can be generically defined.

### 5.7.1 Representation of a Virtual Cut

A virtual cut is used to remove one of the system joints, breaking a closed loop into two serial branches. The virtually cut joint is replaced by two pairs of fixed joints and virtual rigid bodies, both attached to the distal ends of the aforementioned branches. Each fixed joint-virtual body pair forms a fictitious node, collectively referred to as "leafs" (Figure 5.6a). To handle this case, we created the *VirtuallyCutJoint* object shown in Figure 5.6b.




(a) Representation of the leafs created by a virtual cut.

(b) UML component diagram for the virtually cut joint object.

Figure 5.6: Representation of a dynamic tree (left) and object of the virtually cut joint (right).

This object has three members to store the joint constrained DoFs matrix, denoted $\overline{A}$, and the references to the two nodes that were created to replace the (virtually cut) joint connection, denoted node$^+$ and node$^-$.

### 5.7.2 Geometric Constraints and Contact Wrenches

This objects provides the interfaces to compute the geometric constraints and the contact wrenches. The former is implemented by the routine *computeGeometricConstraints* which accesses the kinematics of the referenced nodes to compute $\Upsilon_i$, returned as output. On the other hand, the routines *computeContactWrenches* and *computeTangentContactWrenches* require as input $\lambda_j$ (resp. $\Delta\lambda_j$) and use the relative (resp. tangent) kinematics of the referenced nodes to compute $F_\lambda^+$ and $F_\lambda^-$ (resp. $\Delta F_\lambda^+$ and $\Delta F_\lambda^-$). Then, the computed wrenches are directly assigned to the respective node body with its *addContactWrench* (resp. *addTangentContactWrench*) routine, which will be defined in Section 5.9.

With the so defined object, every kinematics loop is opened and the algorithm can work automatically assigning the geometric constraints and the contact wrenches. We now address the *nodes* groups which composes the lower level of the *tree* group.

## 5.8 The *nodes* group

In this section, we discuss the elements of the nodes group of Figure 5.2. Each node should encapsulate one of the elements involved in every step of the recursive forward

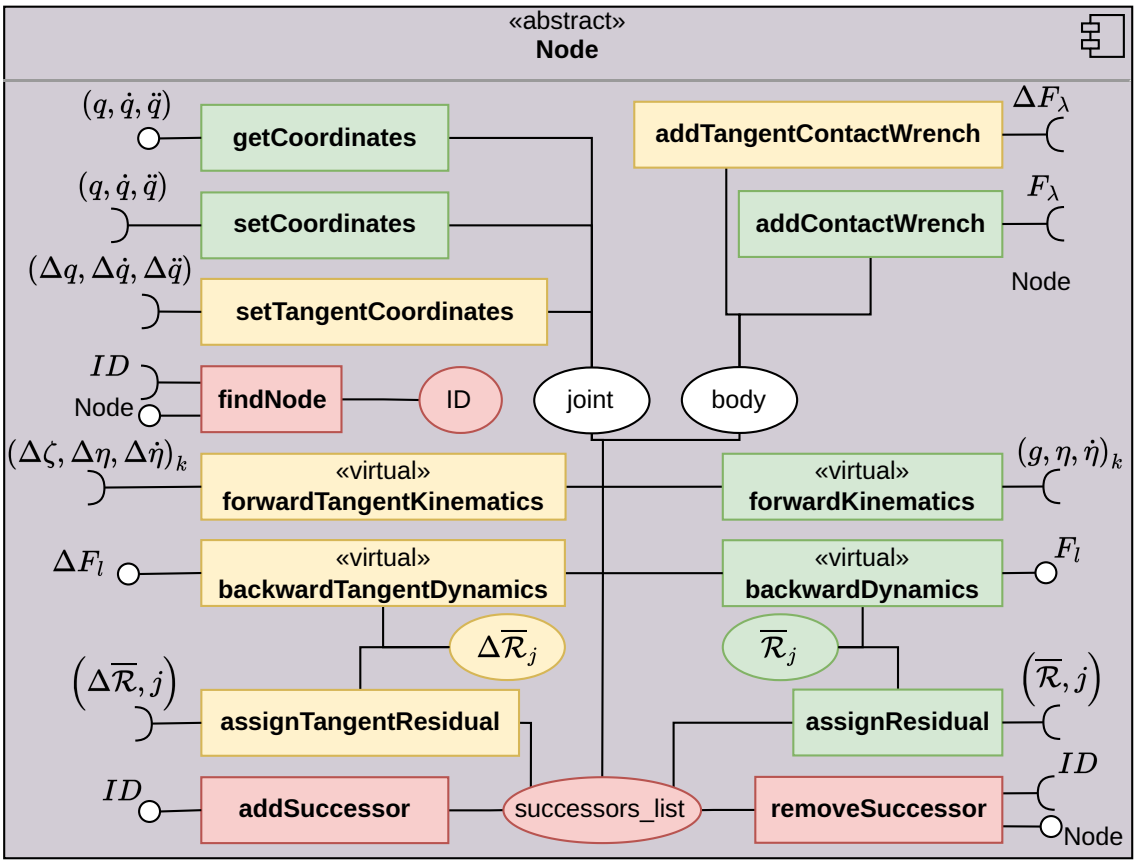



Figure 5.7: UML component diagram for the *Node* abstract object.

and backward passes. This, in most cases, correspond to the pair of a joint and a body. However, in Chapter 3 and 4, we have some special cases being the base of a locomotor or a CPR base and its floating platform (the base of a CPR is seen as a floating body which does not move). In these cases, the kinematics of the body is directly imposed by the time implicit scheme while the forward and backward passes should work without the instance of a joint. As different implementations are required, we created objects of: the *Node*, the *FloatingNode* and the *BaseNode*, which we now describe.

### 5.8.1 The Node Object

The *Node* object represents the standard case of a joint-body pair, whose concatenation forms the dynamic tree. The component diagram for the *Node* object is presented in Figure 5.7.

The members of *body* and *joint* will be addressed later in Sections 5.9 and 5.10. Each node is assigned a distinctive ID for identification purposes. The *successors_list* contains all the children and can be modified during simulation (dynamic tree). Specifically, a child can be added using the *addSuccessor* routine, while a specific successor can be removed

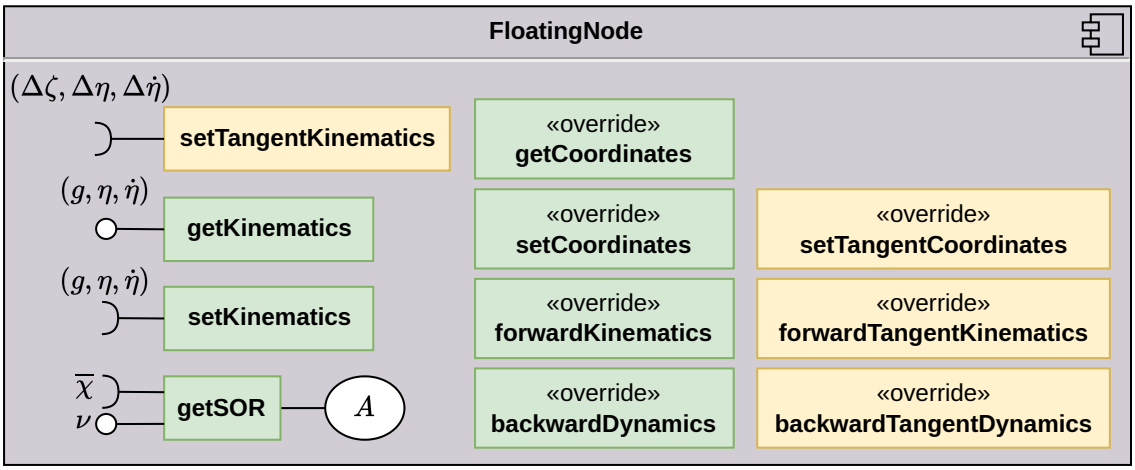



Figure 5.8: UML component diagram for the *FloatingNode* object.

**Inputs:** $(g_k, \eta_k, \dot{\eta}_k)$
**Function** `forwardKinematics()`:

| **Node** | **FloatingNode** |
|---|---|
| $joint \rightarrow getRelativeKinematics()$ | |
| $\begin{cases} g_j &= g_k {}^k g_j \\ \eta_j &= \mathrm{Ad}_{{}^j g_k} \eta_k + \eta_{j/k} \\ \dot{\eta}_j &= \mathrm{Ad}_{{}^j g_k} \dot{\eta}_k + \mathrm{ad}_{\eta_j} \eta_{j/k} + \dot{\eta}_{j/k} \end{cases}$ | $body \rightarrow getKinematics()$ |
| $joint \rightarrow setAbsoluteKinematics()$ | |
| $body \rightarrow setKinematics()$ | |

**for** ***every*** *successor* ***in*** *successors_list* **do**
  $successor \rightarrow forwardKinematics()$
**end**

**end**

**Algorithm 8:** The Algorithm for the Node *forwardKinematics*

by indicating its unique identifier to the *removeSuccessor* routine. Finally, to handle the requirement of the backward passes, the members $\overline{\mathcal{R}}_j$ and $\Delta\overline{\mathcal{R}}_j$ are reference to their section in $\overline{\mathcal{R}}$ and $\Delta\overline{\mathcal{R}}$. These references are recursively assigned with the *assignResidual* and *assignTangentResidual* routines. Concerning the routines, in the following we detail how the two NE passes are implemented.

### Forward Kinematics

The implementation of the *forwardKinematics* is shown in Algorithm 8. This routine takes the kinematics of the antecedent node as input, while $\left({}^k g_j, \eta_{j/k}, \dot{\eta}_{j/k}\right)$ is obtained from the

**Output:** ${}^{k}F_j$
**Function** `backwardDynamics()`:
  **for** ***every*** *successor* ***in*** *successors_list* **do**
    $successor \rightarrow backwardDynamics()$
  **end**

| **Node** | **FloatingNode** |
|---|---|
| $body \rightarrow localWrench()$ | |
| $joint \rightarrow transmittedWrench()$ | $body \rightarrow localWrench()$ |
| $joint \rightarrow computeBalance()$ | |

**end**

**Algorithm 9:** The Algorithm for the *backwardDynamics*

joint. The values $(g_j, \eta_j, \dot{\eta}_j)$ are then computed using Equation (3.36), which is reminded in the algorithm. Afterwards, it updates the joint relative kinematics to its absolute value, preparing it for the backward pass (Remark 18) and, for the same reason, it sets the kinematics of its rigid body. Finally, it gives $(g, \eta, \dot{\eta})_j$ as input for the *forwardKinematics* of each successor.

#### Backward Dynamics

The *backwardDynamics* routine (Algorithm 9) requires no inputs. It begins by iterating over the successors, descending the tree until the last node is reached. From there, it computes the dynamics, returning the wrench to the antecedent node, effectively running the tree backward. While iterating through the list of successors, it accumulates their transmitted wrenches in ${}^{j}F_l$. The cumulative wrench is then passed to the body's *localWrench* function, which calculates $F_j$. This wrench is then used by the joint's *transmittedWrench* function to compute ${}^{k}F_j$, analogous to ${}^{j}F_l$. Finally, the *computeBalance* routine is invoked to determine the joint balance $\overline{\mathcal{R}}_j$.

#### Access a Node of the Tree

In order to provide access to the nodes of the tree, the *findNode* routine can be used to for searching the *successors_list* for a node with the required ID. If no candidate is found, the analogous function is then invoked on each of the successors in order to continue the search on the remaining tree.

### 5.8.2 FloatingNode

The *FloatingNode* inherits all the members of *Node* while it overrides some routines, the ones marked with «override» (Figure 5.8). Additionally, we added a member and three routines to handle its particular kinematics. The member, denoted with $A$, accounts for

**Input:** ${}^{j}F_{l}$
**Output:** $F_j$
**Function** `localWrench()`:

$$F_j = \mathcal{M}_j\dot{\eta}_j - \mathrm{ad}^T_{\eta_j}\mathcal{M}_j\eta_j + F_{ext} + {}^{j}F_l + F_\lambda$$

**end**

**Algorithm 10:** The Algorithm for the *localWrench*

the allowed DoFs of the floating body (*e.g.,* for the locomotor of 3.9.3 $A = \mathbb{1}_6$ while for the floating platform of the planar CPRs of Section 4.9 $A = (\mathbb{1}_2, \mathbb{0}_{4\times2})$).

#### The Forward Kinematics

In this case, for the forward kinematics (Algorithm 8), it is only necessary to retrieve $(g, \eta, \dot{\eta})_j$ from the body, imposed by its time integrator, discarding the one passed as input. This kinematics state is then used as input for each successor, while the input one is ignored.

#### The Backward Dynamics

For a *FloatingNode*, the *backwardDynamics* routine (Algorithm 9) changes only after having computed ${}^{j}F_l$. In this case, this wrench is directly passed to the body to compute $F_j$ which corresponds to $\overline{\mathcal{R}}_j$.

#### $SO(3)\times\mathbb{R}^3$ Parametrization

The *getSOR* routine takes as input the corresponding part of $\overline{\chi}$, denoted $x$, and returns the vector $\nu = Ax$ accordingly to the DoFs defined by $A$.

### 5.8.3 BaseNode

The *BaseNode* object inherits from *FloatingNode* (Figure 5.2) and it is used to encapsulate the singular case of a manipulator base, which has no joints, as a floating body, but does not move. Specifically, the only routine that is overridden is *getSOR* which will always return $\nu = 0_{6\times1}$, as there are not DoFs so $A = []$.

In summary, each node interfaces its body and, if applicable, a joint with its antecedents and successors, using their interfaces regardless of type. The specific implementations for rigid or continuum joints and rigid bodies are detailed in the following sections.

## 5.9 The *RigidBody* Object

A rigid body has different roles in the NE passes: in the forward passes its (tangent) kinematics is assigned by the supporting joint, while in the backward passes it computes the (tangent) wrench applied at the supporting joint. The implementation of these steps

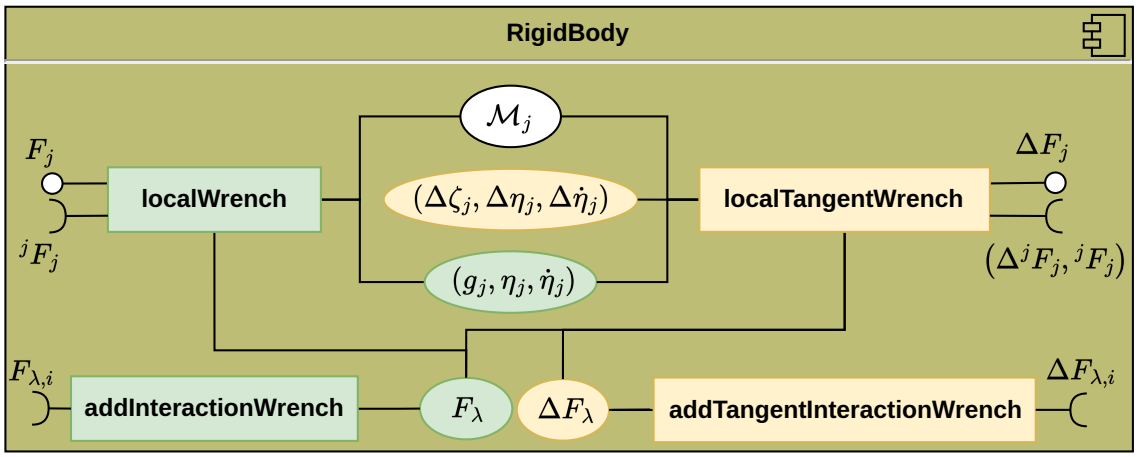



Figure 5.9: UML component diagram for the *RigidBody* object.

are provided in the *RigidBody* object, which we now detail in terms of members and routines.

### 5.9.1 Members

The object has the following members: $\mathcal{M}_j$ to store the inertia matrix of the body, $(g_j, \eta_j, \dot{\eta}_j)$ and $(\Delta\zeta_j, \Delta\eta_j, \Delta\dot{\eta}_j)$ to store its (resp. tangent) kinematics state and $F_\lambda$ and $\Delta F_\lambda$ used to store the cumulative sum of the (resp. tangent) contact wrenches coming from the *VirtuallyCutJoint*s. Note that, by setting $\mathcal{M} = \mathbb{0}_6$ the *RigidBody* becomes a fictitious body (*e.g.,* one of the extrema cross section of a Cosserat rod or the one introduced by a virtual cut).

### 5.9.2 Routines

In terms of routines, the object computes the cumulative sums of $F_\lambda$ (resp. $\Delta F_\lambda$) with the *addContactWrenches* (resp. *addTangentContactWrenches*) routine. On the other hand, *localWrench* and *localTangentWrench* compute the local (resp. tangent) wrench of the body at the joint frame. The *localWrench* routine is detailed in Algorithm 10, it takes as input the cumulative sum of the wrench from the successors, denoted ${}^jF_l$, while it return as output is the wrench at the joint $F_j$.

## 5.10 The *joints* group

As presented in Chapter 3 a joint can be rigid or continuum, resulting in two different sets of equations to compute the relative kinematics $\left({}^kg_j, \eta_{j/k}, \dot{\eta}_{j/k}\right)$ and the transmitted wrench ${}^jF_l$. We created the abstract object for a joint, denoted *AbstractJoint* (Figure 5.10), which has most of its routines as *virtual*, with their implementation given by the inheriting objects of *RigidJoint* and *ContinuumJoint* (Figure 5.2). The latter is just an

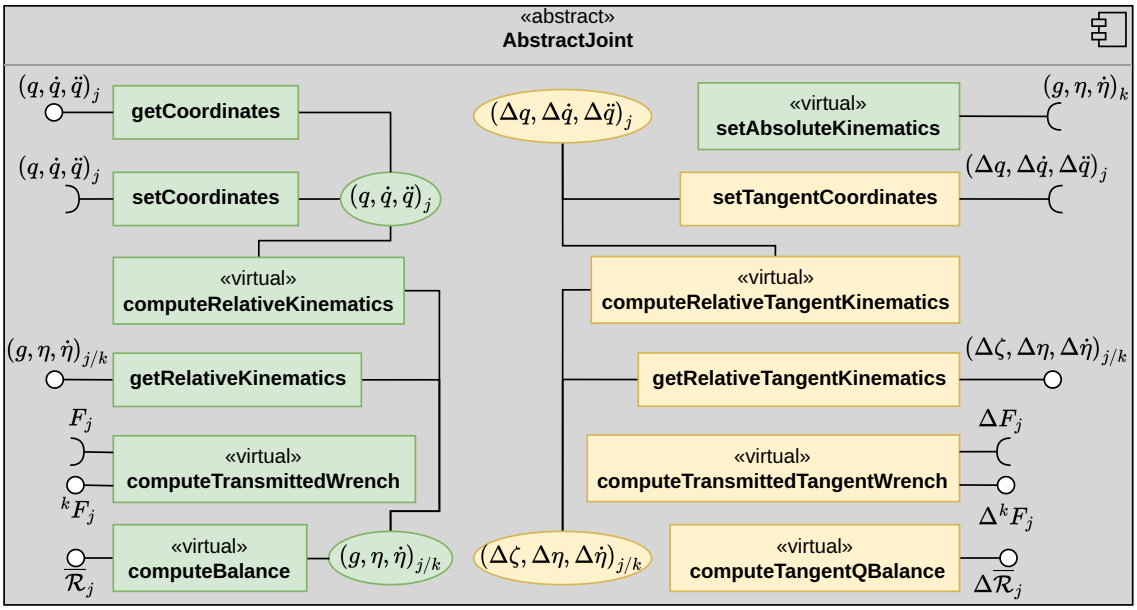



Figure 5.10: UML component diagram for the *AbstractJoint* object.

**Input:**
**Output:** $\left({}^kg_j, \eta_{j/k}, \dot{\eta}_{j/k}\right)$
**Function** `computeRelativeKinematics()`:

| **RigidJoint** | **ContinuumJoint** |
|---|---|
| | $\xi = A_j\Phi_j q_j + \xi^0,\ \dot{\xi} = A_j\Phi_j\dot{q}_j,\ \ddot{\xi} = A_j\Phi_j\ddot{q}_j,$ |
| $\begin{cases} {}^kg_j &= {}^kg_j(q_j) \\ \eta_{j/k} &= A_j\dot{q}_j \\ \dot{\eta}_{j/k} &= A_j\ddot{q}_j \end{cases}$ | $\begin{cases} {}^kg'_j &= {}^kg_j\widehat{\xi}_j \\ \eta'_{j/k} &= -\mathrm{ad}_{\xi_j}\eta_{j/k} + \dot{\xi}_j \\ \dot{\eta}'_{j/k} &= -\mathrm{ad}_{\xi_j}\dot{\eta}_{j/k} - \mathrm{ad}_{\dot{\xi}_j}\eta_{j/k} + \ddot{\xi}_j \end{cases}$ |
| | $\left({}^kg_j, \eta_{j/k}, \dot{\eta}_{j/k}\right)(X = 0) = (\mathbb{1}, 0, 0)$ |

**end**

**Algorithm 11:** The Algorithm for the *computeRelativeKinematics*

adapter to the object of Cosserat rod, such that it can format the input and output of the equations presented in Section 2.3.6.

### 5.10.1 The Joints Coordinates

During execution, the algorithm requires to read and change the joint coordinates every time the *Prediction*, *Correction* and *tangentCorrection* routines of are called (Section 5.4). To enable such access, the *setCoordinates* and *getCoordinates* routines allow the assignment (or retrieval) of the joint's generalized coordinates when called from the corresponding

**Input:** $F_l$
**Output:** ${}^{j}F_l$
**Function** `transmittedWrench()`:

| **RigidJoint** | **ContinuumJoint** |
|---|---|
| | $\Lambda(X=1) = -F_l$ |
| ${}^{j}F_l = \mathrm{Ad}^T_{l_{g_j}} F_l$ | $\Lambda'_l = \mathrm{ad}^T_{\xi_l}\Lambda_l + \mathcal{M}_l\dot{\eta}_l - \mathrm{ad}^T_{\eta_l}\mathcal{M}_l\eta_l - \overline{F}_{ext,l}$ |
| | ${}^{j}F_l = -\Lambda(X=0)$ |

**end**

**Algorithm 12:** The Algorithm for the *transmittedWrench*

**Input:**
**Output:** $\overline{\mathcal{R}}_j$
**Function** `computeBalance()`:

| **RigidJoint** | **ContinuumJoint** |
|---|---|
| $\overline{\mathcal{R}}_j = A_j^T F_j - Q_{act,j}$ | $\overline{\mathcal{R}}_j = Q_{a,j} - Q_{act,j} - Q_{e,j} - C_{e,j}$ |

**end**

**Algorithm 13:** The Algorithm for the *computeBalance*

routine of the *Node* object.

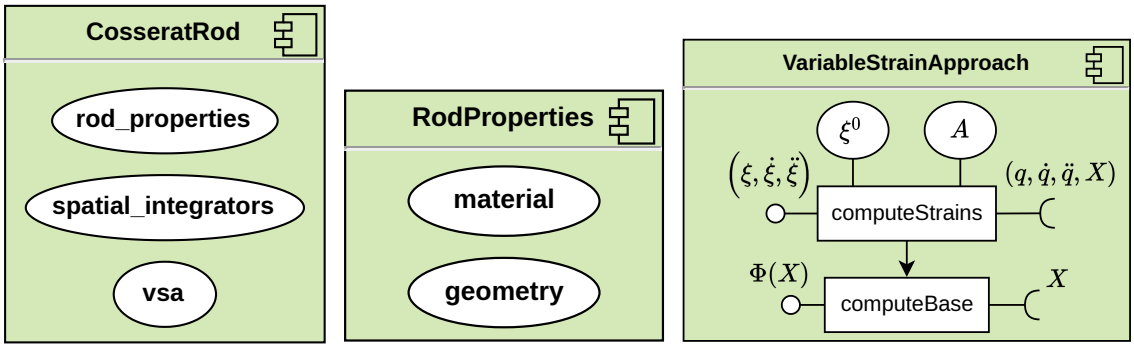
### 5.10.2 The Joints Relative Kinematics

The routine *computeRelativeKinematics* (Algorithm 11) requires no input, as it access the member $(q, \dot{q}, \ddot{q})$, and uses either Equation (3.37) or (3.38) depending on the joint type (for the latter, it computes the field of strain prior finding the solution of the PDEs with the given initial conditions). Then, the computed relative kinematics is stored as a member of the object.

### 5.10.3 The Joints Backward Dynamics

The backward dynamics requires to compute the wrench transmitted from a joint to its antecedent together with the computation of its internal balance. To this aim, we describe the *transmittedWrench* and *computeBalance* routines. The former routine, detailed in Algorithm 12, takes as input the wrench at the body $F_j$ and gives ${}^{j}F_l$ using Equation (3.40) or (3.41) depending on the type of joint. Finally, during the backward pass, *computeBalance* gives $\overline{\mathcal{R}}_j$ using one of the relation of (3.44) (Algorithm 13).

A joint, whether rigid or continuous, can be subject to some form of actuation. In the case of a *RigidJoint*, the actuator is a dedicated member, whereas in the case of a *ContinuumJoint*, it is directly assigned to the spatial integrators, as will be described in the following sections.




(a) (b) (c)

Figure 5.11: Representation of the *CosseratRod* object (a) which implements the modelling of a Cosserat rod containing the *RodProperties* object (b) and the *VariableStrainApproach* object (c).

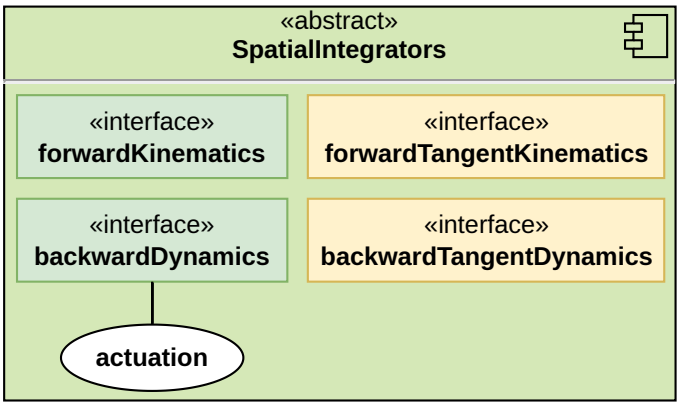



Figure 5.12: The *SpatialIntegrators* object responsible for the numerical integrations in the forward and backward passes.

**Remark 20.** *Note that the computation of the relative (resp. tangent) kinematics is split into three distinct parts to be removed from the NE forward passes. Consequently, the sequential assignment of coordinates to (resp. tangent) joints (which constitutes a serial set of tasks) can be followed by the parallel computation of the relative (resp. tangent) kinematics. On the other hand, the backward (resp. tangent) dynamics is a serial sequence of tasks which requires the knowledge of the state of the successors. As a result, the parallelization of these routines requires running the tree with a breadth-first approach.* ∎

## 5.11 The *CosseratRod* Object

The *CosseratRod* object serves to implement the dynamics modelling of a deformable body, treated as a Cosserat rod (Section 2.1), using the VSA (Section 2.3.6). We described the object in Figure 5.11a with its main members *rod_properties*, *spatial_integrators* and *vsa*.

### 5.11.1 The *RodProperties*

The *RodProperties* has two members used to specify the material and the geometry of the rod, denoted *material* and *geometry*, respectively (Figure 5.11b). The *material* member is an instance of the *MaterialProperties* object which is simply a collection of the material properties of the rod, *i.e.,* $E$, $G$, $\rho$ and $\mu$, while the *geometry* member specifies the shape of the cross section and is of the abstract object *CrossSection*. Different specialization for the cross sections exists, including: *CircularCrossSection* and *RectangularCrossSection*, which we do not detail here for conciseness. These object support the implementation of axial-varying geometry as the one of the bio-inspired swimmer of Section 3.9.3.

### 5.11.2 The *VariableStrainApproach*

The strain-based parametrization of the rod is contained in the *vsa* member which is of type *VariableStrainApproach*. This object contains the instance of the strain at rest $\xi^0$ and the routine *computeBase*, which computes $\Phi$ at the given input $X$, that is used by *computeStrains*, which, given the input $(q, \dot{q}, \ddot{q})$ and $X$ computes the set $(\xi, \dot{\xi}, \ddot{\xi})$.

### 5.11.3 The *SpatialIntegrators*

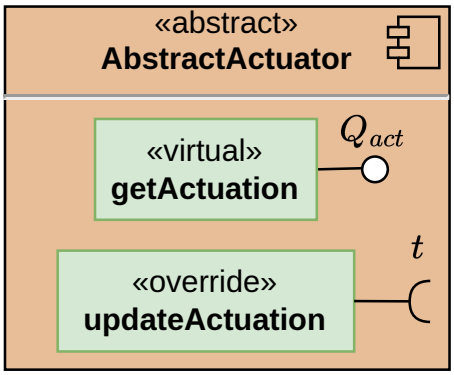
In order to reconstruct the rod kinematics and dynamics state, it is necessary to employ numerical integration techniques. This aspect is addressed by the *spatial_ integrators* member (*SpatialIntegrators* object Figure 5.12). This object implements the integrations required by the forward (resp. tangent) kinematics and backward (resp. tangent) dynamics, depending on the selected integrator type. A variety of integrators, which can be applied to this context, are discussed in the literature and a comparison of these will be addressed in Section 5.13.4. Ultimately, as the action of distributed actuation must be integrated along the rod domain, the integrator is provided with the instance of the employed distributed actuation as a member, in the case of an actuated rod.

## 5.12 Actuators

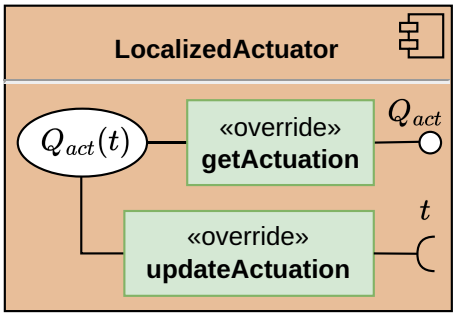
**Input:** $\xi, T_\alpha$
**Output:** $\Lambda_{act}$
**Function** `getDistributedWrench()`:
$\Gamma_\alpha = \Gamma + K \times D_\alpha + D'_\alpha$
$\Lambda_{act} = \sum_{\alpha=1}^{n_t} \frac{1}{\|\Gamma_\alpha\|} \begin{bmatrix} D_\alpha \times \Gamma_\alpha \\ \Gamma_\alpha \end{bmatrix} T_\alpha$
**end**

**Algorithm 14:** The Algorithm for the *getDistributedWrench*

In the context of our application, an actuator may be employed to provide a localised action, such as a revolute joint, or alternatively, it can be utilised to apply a distributed effort along a deformable body. In this manner, an abstract object, denoted *AbstractActuator*,

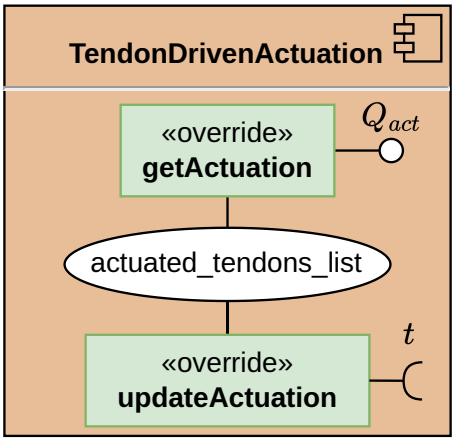



(a) The *Actuator* abstract object.

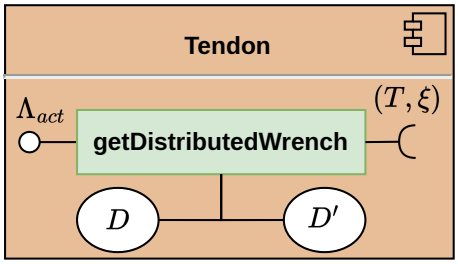



(b) The *LocalizedActuator* object.

Figure 5.13: UML component diagrams for the *Actuator* abstract object (left) and the inheriting *LocalizedActuator* object (right).




(a) *TendonDrivenActuation* component diagram.




(b) Component diagram for the *Tendon* object.

Figure 5.14: Representation of the objects required to implement a tendon-driven actuation.

was implemented (Figure 5.13a). This object provides the virtual routine *updateActuation*, which takes the current time as its input to update the current actuation law computing $Q_{act}$, which is returned by *getActuation* when required.

### 5.12.1 The *LocalizedActuator*

The *LocalizedActuator* object provides the necessary routines for implementing this specific type of actuation (Figure 5.13b) by overriding the *updateActuation*. Specifically, a custom function can be assigned to the object to implement the desired evolution of the actuation (e.g., the torque of a revolute joint).

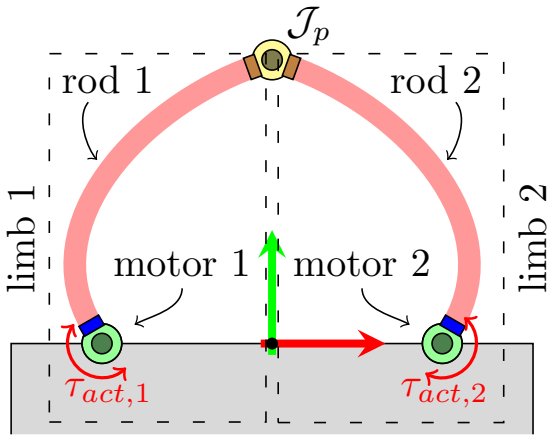



Figure 5.15: Representation of the planar RFRFR CPR made with two proximal revolute joints, actuated by motor 1 and motor 2, and two elastic thin rods, namely rod 1 and rod 2, connected by the passive revolute joint $\mathcal{J}_p$.

### 5.12.2 The *TendonDrivenActuation*

For the case of distributed actuation, we implemented a tendon-driven actuation using the *TendonDrivenActuation* object, which overrides both the *updateActuation* and *getActuation* methods. For this distributed actuation, the *actuated_tendons_list* contains a set of tendons paired with a motor, *i.e.,* a *LocalizedActuator*. The latter is used to compute $T_\alpha(t)$ based on a custom actuation law. The *Tendon* object contains the geometry of the placement of the tendon along the rod, with the members $D$ and $D'$, allowing to compute $\Lambda_{act}$ at any $X$ once provided the cable tension $T$ and $\xi(X)$ (Algorithm 14). Then, the *TendonDrivenActuation* simply loops through the list of paired *Tendon* and *LocalizedActuator* to compute the sum in Algorithm 14. Once the field of stress-wrench is reconstucted, its action needs to be projected into the generalized coordinate space by (2.37), which we recall here as:

$$Q_{act} = \int_0^\ell \Phi^T A^T \Lambda_{act} dX \tag{5.7}$$

For this reason, this kind of actuation is given to the *CosseratRod* object.

**Remark 21.** *If alternative types of distributed actuators are needed, one could simply define a mapping law that converts the applied effort into an actuated stress-wrench. This mapping can then be implemented in a new specialization of the AbstractActuator object.* ■

With the so defined structure, the dynamics simulator can handle a locomotor and manipulators with or without closed kinematics chains. In what follows, we provide a minimal example to show a practical implementation and usage of the NEHA toolbox.

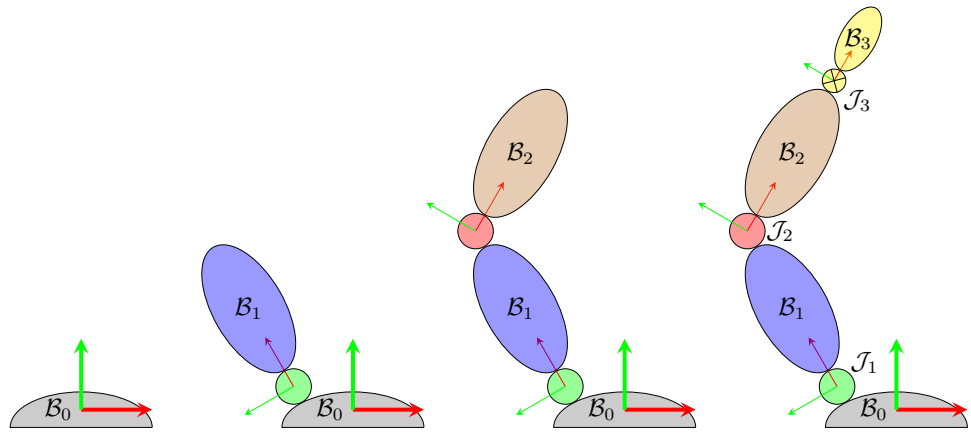



Figure 5.16: Sequential Newton-Euler segmentation for the limb 1 of the planar RFRFR CPR.

## 5.13 Case-Study: a planar RFRFR CPR

In this section, we provide an application of out toolbox using a planar RFRFR CPR. To this end, we will utilize simplified code snippets while simultaneously recalling the group definition for each object, as illustrated in Figure 5.2.

### 5.13.1 Description

This CPR has two identical limbs, namely limb 1 and limb 2, each formed by a proximal actuated revolute joint and a slender passive rod, connected together by a distal passive revolute joint, denoted as $\mathcal{J}_p$ (Figure 5.15). The two proximal joints are placed at a distance $d = 0.4$ m between their center, and are actuated with two motors applying the torques $\tau_{act,1}(t)$ and $\tau_{act,2}(t)$ which, for conciseness, their actuation law be detailed later (Section 5.13.4). the two rods have identical geometry, with length $\ell = 1$ m and circular cross section with radius $r = 1$ mm, and are made of standard spring steel, having Young modulus $E = 210$ GPa, shear modulus $G = 80$ GPa and volume mass $\rho = 7800$ kg.m$^{-3}$. For their modelling we allowed only the bending along their local $z$ axis (*i.e.*, $n_a = 1$) using a Legendre polynomial of order 2 (*i.e.*, $n_e = n_{e_1} = 3$) modes. For the spatial integrations, we used a spectral method with $N_c = 31$ discretization points. Finally, the for the time integration we used a step $\Delta t = 0.01$ s.

### 5.13.2 Step-by-Step Segmentation

To create the tree representation of this system, the user must apply the NE segmentation methodology described in Section 3.5.1 to define the bodies, joints, and nodes. This section provides a detailed analysis of this procedure applied to the case study, with a focus on limb 1, as limb 2 is identical.

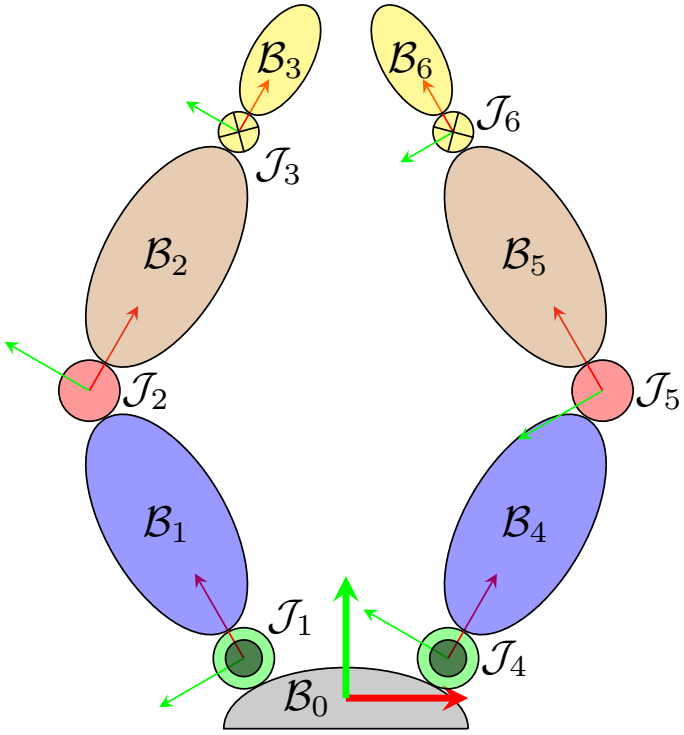



Figure 5.17: Newton-Euler segmentation for the planar RFRFR CPR.

### The base

The base of the robot corresponds to the fictitious body $\mathcal{B}_0$ (see Figure 5.16a), for which $\mathcal{M} = \mathbb{0}_6$ and $r_{cdm} = 0_{3\times1}$, and thus define the first node of the tree, denoted as node 0 of type *BaseNode*.

### The Proximal Revolute Joint

The first proximal joint is designated as the *RigidJoint* $\mathcal{J}_1$. Since the joint is revolute, with axis of rotation $a_1$, its DoFs matrix is defined as $A_1 = (0, 0, 1, 0, 0, 0)^T$ and $q_1$ represents its rotation angle. The pose relative to the previous node is given by ${}^0g_1(q_1)$ (3.31) which is provided with ${}^0g_{1,c} = (\mathbb{1}_3, {}^0r_1)$, with ${}^0r_1 = -(d/2, 0, 0)^T$ (Remark 15). The joint supports the proximal cross section of rod 1, which is modelled as the fictitious *RigidBody* $\mathcal{B}_1$ (see Figure 5.16b). Together, joint $\mathcal{J}_1$ and body $\mathcal{B}_1$ form node 1, which is a successor of node 0.

### The Rod

We represent the body of rod 1 as the passive *ContinuumJoint* $\mathcal{J}_2$, which supports the rod distal cross section, again modelled as the fictitious *RigidBody* $\mathcal{B}_2$ (see Figure 5.16c). This joint-body pair forms node 2, which is the successor of node 1.

| joint ID | type | Actuated | relative kinematics |
|---|---|---|---|
| 0 | N/A | N/A | N/A |
| 1 | RigidJoint | yes | $A_1$, $^0g_{1,c}$ |
| 2 | ContinuumJoint | no | rod 1 |
| 3 | RigidJoint | no | $A_3$, $\mathbb{1}_4$ |
| 4 | RigidJoint | yes | $A_4$, $^0g_{4,c}$ |
| 5 | ContinuumJoint | no | rod 2 |
| 6 | RigidJoint | no | $A_6$, $\mathbb{1}_4$ |

Table 5.1: Parameters used to define the *RigidJoint* and *ContinuumJoint* objects

| node ID | type | successors ID | body | joint |
|---|---|---|---|---|
| 0 | Base Node | $\{1, 4\}$ | $\mathcal{B}_0$ | N/A |
| 1 | Node | $\{2\}$ | $\mathcal{B}_1$ | $\mathcal{J}_1$ |
| 2 | Node | $\{3\}$ | $\mathcal{B}_2$ | $\mathcal{J}_2$ |
| 3 | Node | $\{\}$ | $\mathcal{B}_3$ | $\mathcal{J}_3$ |
| 4 | Node | $\{5\}$ | $\mathcal{B}_4$ | $\mathcal{J}_4$ |
| 5 | Node | $\{6\}$ | $\mathcal{B}_5$ | $\mathcal{J}_5$ |
| 6 | Node | $\{\}$ | $\mathcal{B}_6$ | $\mathcal{J}_6$ |

Table 5.2: Parameters used to define the *Node* and *BaseNode* objects.

### The Virtual Cut

Finally, at the end of the branch, the virtually cut joint $\mathcal{J}_p$ is replaced by two leaf nodes, each consisting of a fixed joint and a corresponding fictitious rigid body. On the first limb, this corresponds to adding a *RigidJoint* $\mathcal{J}_3$ and another fictitious *RigidBody* $\mathcal{B}_3$ (see Figure 5.16d). The rigid joint $\mathcal{J}_3$ is passive, with $A_3 = [\,]$, and does not introduce any offset from the previous node, so $^2g_{3,c} = \mathbb{1}_4$. Together with $\mathcal{B}_3$, this forms the last node of the limb, denoted as node 3, which is the successor of node 2 and, being the result of a virtual cut, has no further successors.

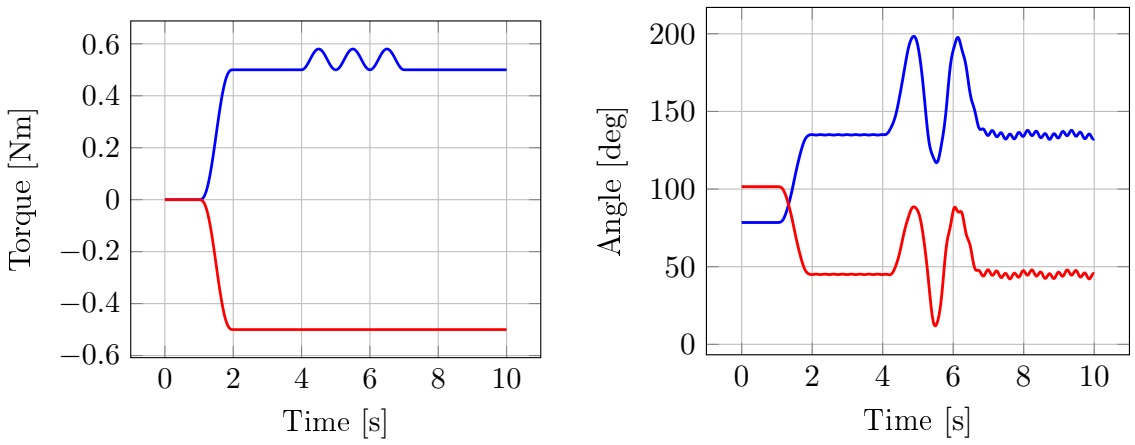
### Summary of the Newton-Euler Segmentation

This procedure can be applied to second limb as well, obtaining the NE segmentation for the RFRFR robot, represented in Figure 5.17. This allows to defined Table 5.1 for the joints and Table 5.2 for the nodes, which are prerequisites for the subsequent implementation. Regarding the latter, it is crucial that, for each node, the successor IDs are either empty or greater than the current node ID. Additionally, no successor should be assigned to a node that was created as a result of a virtual cut.

### 5.13.3 The Code Explained

```
1
2       ators::LocalizedActuator motor_1(τ_act,1(t))
3       ators::LocalizedActuator motor_2(τ_act,2(t))
4
5       crosp::CircularCrossSection cross_section(r)
6       crosp::MaterialProperties material
7       crosp::RodProperties rod_properties(material, cross_section, ℓ)
8
9       deformation_modes = (0,0,n_e,0,0,0)
10      crosp::VariableStrainParametrization vsa(deformation_modes)
11
12      crosp::CosseratRod rod_1(rod_properties, vsa)
13      crosp::CosseratRod rod_2(rod_1)
14
15      joints::RigidJoint joint_1(A_1, actuator_1, ⁰g_1,c)
16      joints::RigidJoint joint_4(A_4, actuator_2, ⁰g_4,c)
17
18      joints::RigidJoint joint_3, joint_6
19
20      joints::ContinuumJoint joint_2(rod_1)
21      joints::ContinuumJoint joint_5(rod_2)
22
23      bodies::RigidBody body_0, body1, ..., body_6
24
25      nodes::Node node_6(body_6, joint_6)
26      nodes::Node node_5(body_5, joint_5, {node_6})
27      nodes::Node node_4(body_4, joint_4, {node_5})
28      nodes::Node node_3(body_3, joint_3)
29      nodes::Node node_2(body_2, joint_2, {node_3})
30      nodes::Node node_1(body_1, joint_1, {node_2})
31
32      nodes::BaseNode node_0(body_0, {node_1, node_4})
33
34      leafs::VirtuallyCutJoint leaf_1(node_3, node_6, Ā_p)
35
36      tree::Tree robot_tree({node_0}, {motor_1, motor_2}, {leaf_1})
37
38      tims::CommutativeGroup commutative_integrator
39      tims::LieGroup lie_integrator
40
41
42      robot_tree→findNode(1)→setCoordinates(π/2)
43      robot_tree→findNode(4)→setCoordinates(π/2)
44
45      neha::DynamicSimulator simulator(robot_tree, {lie_integrator},
            commutative_integrator)
46
47      simulator→dynamicsSimulation(t)
```

The simulation of the RFRFR planar CPR can be setup by defining the actuators, the geometry of the rod, the various bodies, joints and nodes (Tables 5.1 and 5.2) and the implicit schemes. In detail, the actuators are defined in lines (2–3) by providing the functions $\tau_{act,1}(t)$ and $\tau_{act,2}(t)$ containing the actuation laws. Then we define the rods cross-sections geometric and material properties (lines 5–7) as well as they strain parametrization (lines 9–10) used to declare the two *CosseratRod* objects (lines 12–13). Subsequently, the NE segmentation is implemented defining the CPR joints (lines 15–21) and bodies (line




(a) (b)

Figure 5.18: Plot (a): time evolution of the actuation torques $\tau_{d,1}$ (blue) and $\tau_{d,2}$ (red). Plot (b) time evolution of the revolute joint angles $q_{r,1}$ (blue) and $q_{r,2}$ (red).

23). These objects are used to define the nodes (lines 25–32) which, together with the leafs (line 34), define the tree (line 36). The time integrators are then defined (lines 38–39) completing the requirements for the dynamics simulation. In order to provide initial condition to the statics assembly of the CPR, some initial condition can be provided by addressing the specific nodes of the tree (line 42–43). Finally, the *DynamicsSimulator* is initialized (line 45) and the simulation is launched for a duration of $t$ seconds (line 47). A more detailed explanation of the code is provided in Appendix E, while the result of the dynamics simulation are discussed in the following.

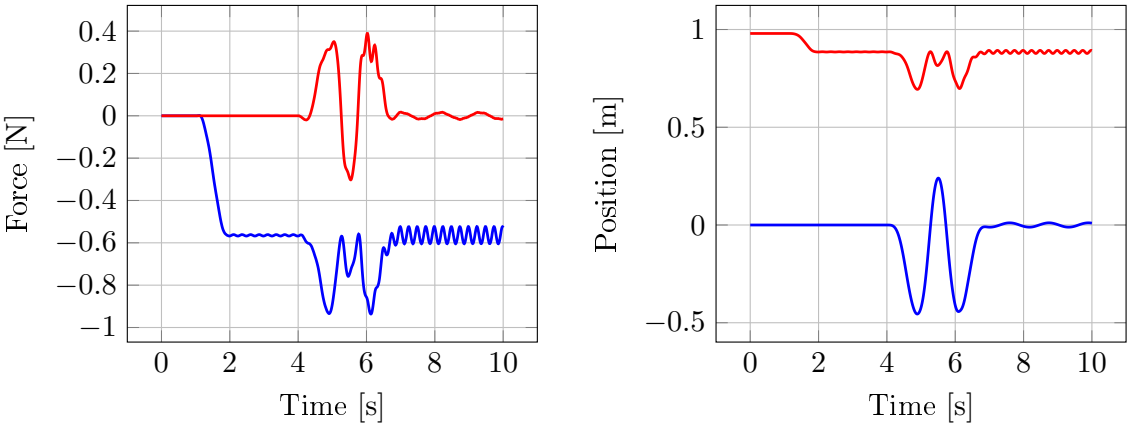
### 5.13.4 Dynamics Simulation Results

We now present the results of the dynamics simulation for the RFRFR CPR using the defined implementation. We begin by defining the actuation laws applied to the proximal joint torques, followed by a detailed analysis of the system's dynamic response. Next, the simulation layout will be used to compare the time implicit schemes discussed in Chapters 3 and 4. Finally, a comparison of different spatial integrators for the numerical integration of the Cosserat ODEs will be provided.

#### Actuation of the Proximal Revolute Joints

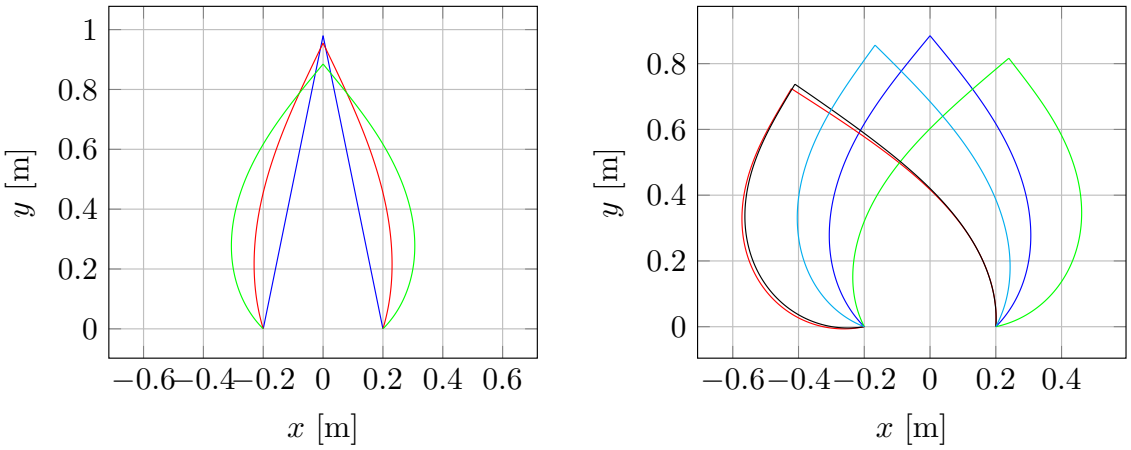
Once a suitable static configuration is achieved, the dynamics is started by imposing a prescribed time-evolution of the joint torques $\tau_{act,1}(t)$ and $\tau_{act,2}(t)$ plotted in Figure 5.18a. The two torques start with zero values between $t = 0$ s and $t = 1$ s. Their value is then progressively changed using a polynomial of order 5:

$$\tau_{act,2}(t) = -\tau_{act,1}(t) = k_r \left(6(t-1)^5 - 15(t-1)^4 + 10(t-1)^3\right) \tag{5.8}$$




(a) (b)

Figure 5.19: Plot (a): time evolution of the two components of $\lambda$ in the inertial frame (*i.e.*, ($\lambda_x$ (blue) and $\lambda_y$ (red))). Plot (b): time evolution of the joint $\mathcal{J}_p$ inertial position ($r_{p,x}$ in blue and $r_{p,y}$ in red).




(a) Snapshots at $t = 0$ s (blue) $t = 1.5$ s (red) and $t = 2$ s (green).

(b) Snapshots at $t = 4$ s (blue) $t = 5$ s (red), $t = 5.5$ s (green), $t = 6$ s (black) and $t = 6.5$ s (cyan).

Figure 5.20: Snapshots of the RFRFR planar CPR during the symmetric increase of torque (left) and during the sinusoidal phase (right).

where $t \in ]1, 2]$ s and $k_r = 0.5$ Nm/s are respectively the duration and the gain. Once the values at $t = 2$ s are reached, denoted $\tau_{hold,1}$ and $\tau_{hold,2}$, the torque $\tau_{act,2}(t)$ remains constants while $\tau_{act,1}(t)$ is subject to a sinusoidal variation between $t = 4$ s and $t = 7$ s, defined as:

$$\tau_{act,1}(t) = \tau_{hold,1} + k_s \left(1 - \cos(\omega(t - 4))\right) \tag{5.9}$$

where $k_s = 0.04\,\text{Nm.s}$, $\omega = 2\pi$ RAD/s and $t \in [4, 7]$. Note that the sinusoidal addendum to the torque is not symmetric but ranges from 0 to $2k_s$.

### Comments on the Simulated Behaviour

Figures 5.18b, 5.19a and 5.19b respectively show the time evolution of the joint angles, the Lagrange multipliers (in the inertial frame), and the inertial position of the tip (the passive joint $\mathcal{J}_p$). Figure 5.20 (a, b) display a sequence of snapshots over $[1, 2]$ s and $[4, 7]$ s, showing that the CPR first bends symmetrically with its tip getting closer to the base, then because of the oscillating component added to $\tau_{act,1}$, its distal end start oscillating. The robot undergoes three cycles of torque, whereas the movement of its upper and lower sections results in a twofold cycle. This discrepancy is attributed to the influence of inertial effects. When we control the robot by imposing torques, the inertia of the robot can affect the joint angles, as shown in Figure 5.18b. Note that as expected, one can observe structural (harmonic) oscillations superimposed with overall deformations imposed by the torque time variations.

Given that the system is undamped, its motion should persist over time without disruption. In the following section, we will examine how, depending on the implicit scheme, the simulation may diverge.

### Effects of Different Time Implicit Schemes

In this section we want to highlight the importance in the choice of the correct time implicit scheme for the overall performances of a dynamics simulation. To this aim, we extended the simulation previously described to let the CPR continue its oscillatory (undamped) motion. We will observe the effect of the Newark (Chapter 3) and Generalized-$\alpha$ (Chapter 4) schemes on the tip position, the Newton-Raphson iteration required at every time-step and the Lagrange multipliers. Even if the latter is not of common interest in robotics applications, it allows to show the high-frequency spurious noises introduced by the geometric constraints [98]. Even with this simple example, we see that the geometric constrains affects the stability of the Newamrk scheme which diverges at $t = 34.76$ s (Figure 5.21). However, before this critical point, there are no perceivable differences between the tip position computed by the Newmark and the Generalized-$\alpha$ schemes, which remains of the order of the threshold of the residual norm. The cause of this instability if observable in Figure 5.22, where for the Newmark scheme the Lagrange multipliers are subject of increasing numerical noises (Figure 5.22a). On the other hand, for the Generalized-$\alpha$ scheme, we can appreciate how the motion remains stable and undamped (Figure 5.21) and the numerical noises on the Lagrange multipliers have been completely eliminated (Figure 5.22b). Furthermore, concerning the number of iterations of the Newton-Raphson scheme (Figure 5.23), the numerical noises increase the required iteration of the Newmark scheme, while the Generalized-$\alpha$ maintains a limited number of iterations.

In summary, we began our simulations with the Newmark scheme of [105] (Chapter 3) and we experienced the numerical instabilities that typically affects the dynamics of

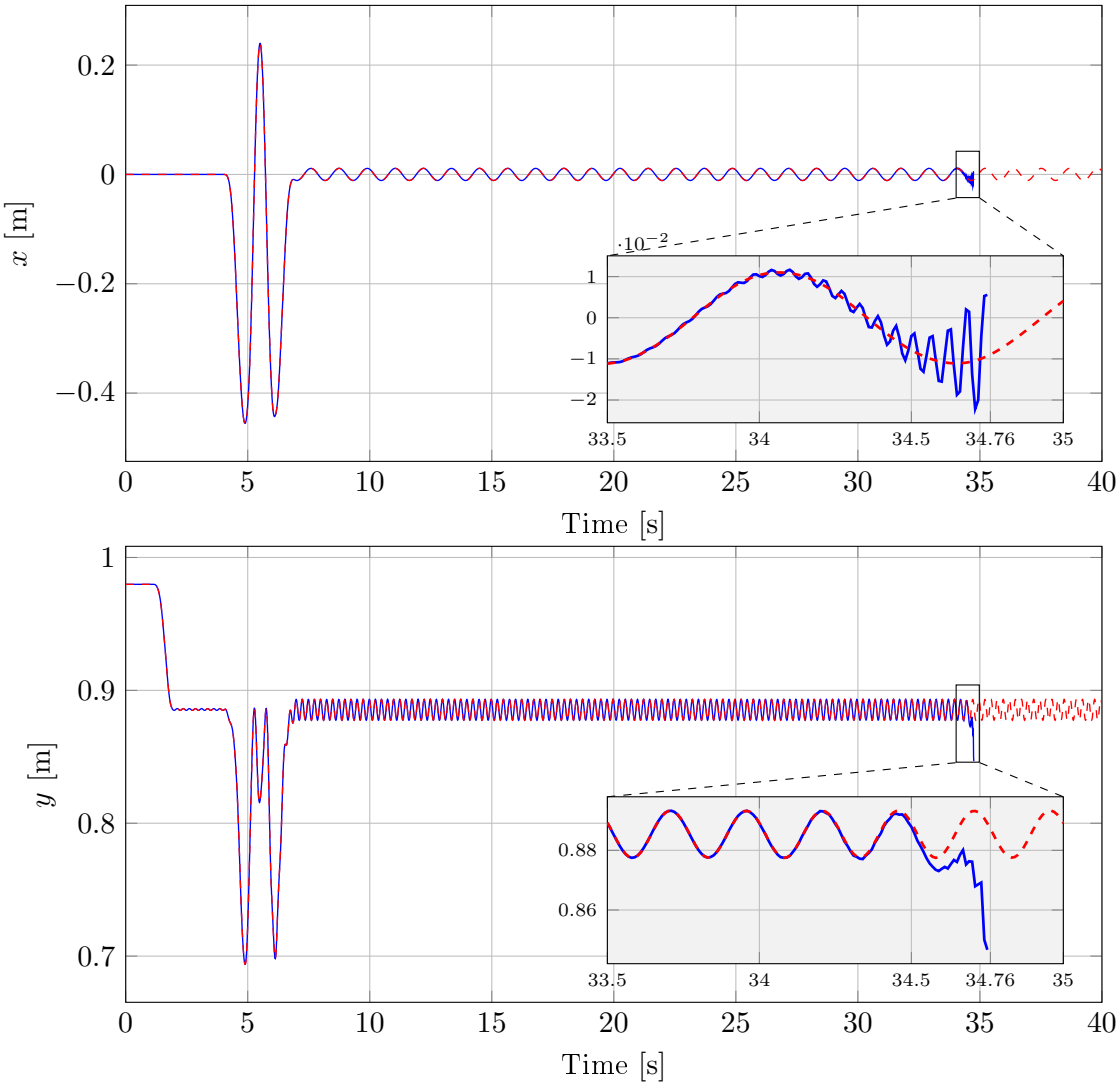
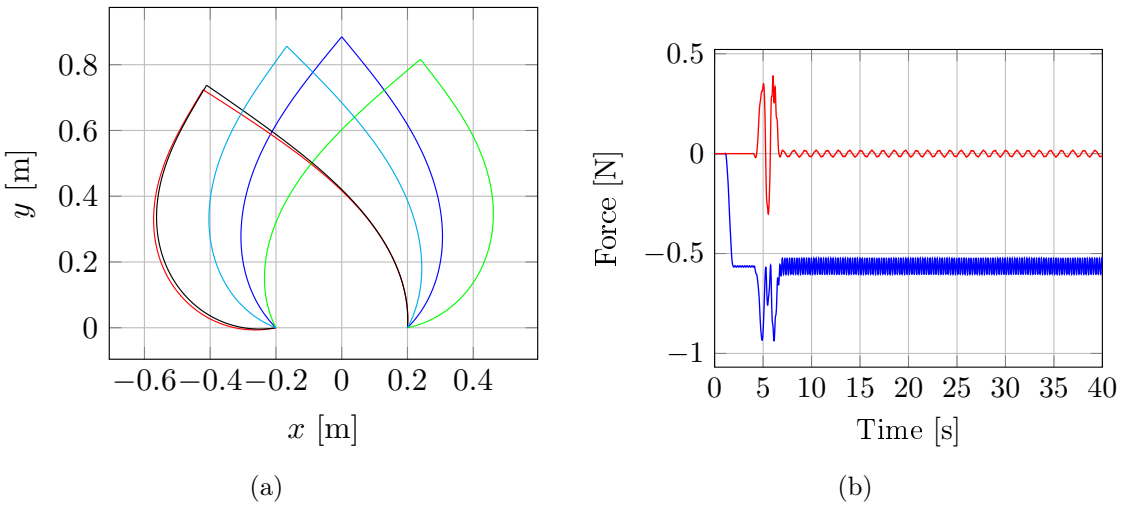



Figure 5.21: Comparison of the computed $x$ (above) and $y$ (below) position of the tip (joint $\mathcal{J}_p$) with the Newmark (blue) and Generalized-$\alpha$ (red) time implicit schemes.

stiff (DAEs) system [98]. Nevertheless, this scheme remains a valid option to integrate the dynamics of a tree-like locomotor or (serial) manipulator, or for a relatively short simulation. Different values of $\Delta t$ can also affect stability. Reducing $\Delta t$ to 1 ms increased the instabilities, causing the simulation to abort at $t = 30.02$ s. Conversely, increasing the time step to $\Delta t = 100$ ms led to instability even sooner, at $t = 15$ s, while $\Delta t = 50$ ms allowed the simulation to complete naturally, delaying the onset of instability. In order to damp the spurious numerical noises, one could add some damping to the internal constitutive law of the rods (Section 2.1.4). However, this also affects the dynamics of the system and the damping parameters have to be finely tuned to limit their effects on the




(a) (b)

Figure 5.22: Time evolution of the two components of $\lambda$ in the inertial frame (*i.e.*, $\lambda_x$ (blue) and $\lambda_y$ (red)) using the Newmark integrator (left) and the Generalized-$\alpha$ (right).

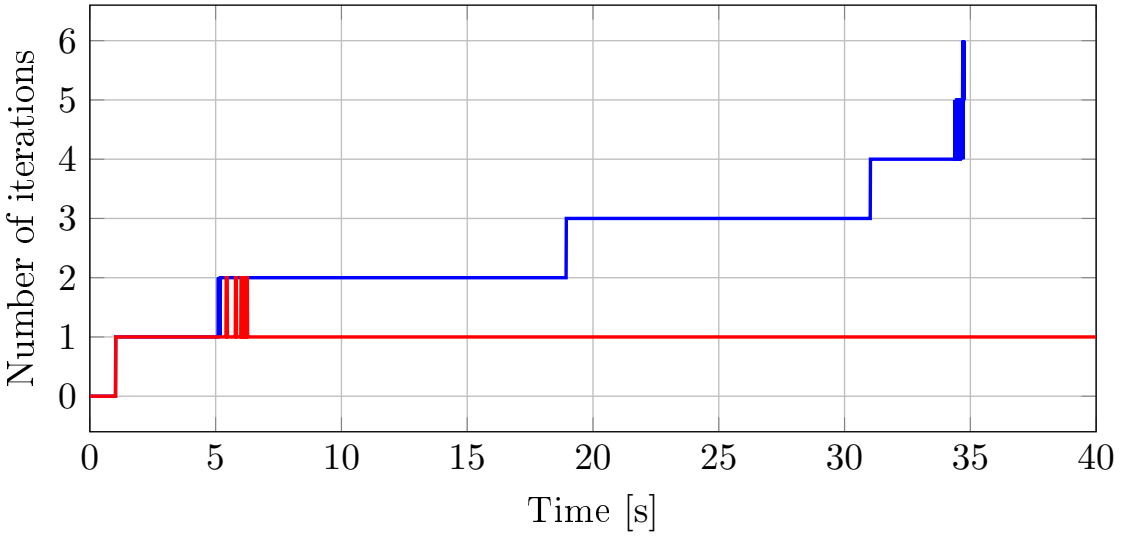



Figure 5.23: Number of Newton-Raphson iterations using the Newmark integrator (blue) and the Generalized-$\alpha$ (red).

system dynamics. The Generalized-$\alpha$ of [108, 115] (Chapter 4) successfully dampens the spurious numerical noises without affecting the system dynamics. In this case the robots maintains its motion and the numerical noises do not interfere with the simulation. In this case, the different time steps ($\Delta t = 1$, $\Delta t = 50$ and $\Delta t = 100$ ms) did not affect the stability of the simulation. As this implicit scheme works for both CPRs and tree-like structures, it has been chosen as the default for the implementation of the time integrators.

Once described the influence of the implicit schemes, we now proceed detailing how

| Runge-Kutta[116] | | | | Spectral [95] | | | | Magnus [117, 118] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n_e$ | FK | BD | $IDM$ | $N_c$ | FK | BD | $IDM$ | $N_c$ | FK | BK | $IDM$ |
| 5 | 380 | 960 | 1,340 | 11 | 53 | 20 | 73 | 11 | 28 | | |
| 10 | 570 | 1,280 | 1,850 | 21 | 360 | 140 | 500 | 21 | 65 | | |
| 15 | 780 | 1,900 | 3,680 | 31 | 860 | 280 | 1,140 | 31 | 85 | | |

Table 5.3: Computational time, in micro seconds $\mu s$, required by the different numerical integration methods to compute the forward kinematics (FK), the backward dynamics (BD) and the corresponding $IDM$, with standard deviations negligible.

different spatial integrators affect the numerical performances of our implementation.

### Performances of the Spatial Integrators

The spatial integration is required by the $IDM$ and $TIDM$ forward and backward passes, integrated respectively in the domains $X \in [0, \ell]$ and $X \in [\ell, 0]$. A variety of spatial integration methods have been proposed in the literature. In this section, we provide a concise overview of some of spatial integrator and we compare their computational time for the numerical integration of the forward and backward passes[1] using the Google benchmark library[119]. For the linear algrabra and matrix operations, we used the Eigen library [120].

**Runge-Kutta** The Runge-Kutta method is a classical approach for solving IVPs, often chosen when no other method is known or when the problem is simple and computational efficiency is not a primary concern [121]. Due to their adaptive control of step size $dX$, Runge-Kutta methods allow achieving a predetermined accuracy, though this comes at the expense of efficiency. In detail, no control over the step size require to evaluate the complete state of each pass, which grows sequentially between the passes of the $IDM$ and for the $TIDM$. This dimensional growth significantly increases the computational time required for the algorithm, thereby affecting overall performance. In Table 5.3 we reported the computational time required by the Runge-Kutta method, implemented by the Boost C++ library [116].

**Spectral Integration Method** On the other hand, the spectral integration method [95] discretizes the integration domain with a grid of $N_c$ elements. The integration is then interpreted as the sequential reconstruction of the state at each point of the grid. This method requires each of the system ODEs to be expressed in the canonical form:

$$y' = A(X)y + b(X), \tag{5.10}$$

which is the case for (2.9), (2.20) and their tangent version. This approach effectively reduces the required computational time (Table 5.3) without affecting precision (relative error w.r.t. the Runge-Kutta integrator proportional to $10^{-12}$ with $N_c > 21$).

[1]Results obtained with a PC Dell (Precision-7560), Processor Inter(R) Core(TM) i7-11850H @ 2.5 GHz, RAM 32 Go

**Integration via Magnus Expansion** Finally, we detail another spatial integration method based on the Magnus expansion [117, 118]. In comparison to the spectral integration method, this approach results in a further reduction in computational time (Table 5.3), again without penalties on the numerical precision. Furthermore, this methodology enables the direct integration of $R$ on the $SO(3)$ Group, thereby preserving its orthogonality.

## 5.14 Conclusions

In the literature, different toolboxes exist for the dynamics modelling of hybrid structures (*i.e.,* structures composed by rigid and slender deformable bodies). In this context we find the PyElastica and SoroSim toolboxes, which are ideal for the simulation of soft robots. However, PyElastica lacks the ability to implement typical actuation systems found in both rigid and soft robotics, and both toolboxes rely on explicit schemes, which are unsuitable for the time integration of stiff systems. Another widely used framework is SOFA, which provides a broad range of algorithms, including FEM, PCC, and, more recently, a geometrically exact implementation of the Cosserat rod theory using the VSA. However, the toolbox's completeness and generality come with the drawback of a steep learning curve for users. These problems prompted the proposal of our toolbox: NEHA[2] (which stands for Newton-Euler Hybrid Algorithm). For the time integration, the toolbox uses time implicit schemes, ensuring numerical stability. It is built on the NE passes from rigid robotics to compute the dynamics system, which is represented as a dynamic tree of nodes. Each of these nodes comprises a rigid body and, when applicable, a joint, which can be the result of a virtual cut (fictitious body and fixed joint). Each node has only one parent but can have multiple children, creating the hierarchical structure of the tree. This layout ensures a high level of generality. The user defines system composition while the constraints and requirements of the time implicit scheme-based dynamics simulation are managed automatically. The approach was then applied to a case study of a planar RFRFR CPR prototype to illustrate the required code. A discussion of the results from a dynamics simulation highlighted the difference between the time implicit schemes proposed in Chapters 3 and 4. We also presented a comparison of the computational times achieved using different spatial integrators for solving the Cosserat ODEs.

[2]The toolbox can be found at the GitHub page.

# Chapter 6

# Conclusion and Perspectives

## 6.1 General Conclusion and Contribution of this Thesis

This thesis is situated within the field of continuum robotics, specifically addressing the topic of dynamics modelling of Continuum Parallel Robots (CPR). In the Introduction (Chapter 1) we provided a concise overview of the field, describing CPRs with different architectures and properties. We explained that these robots typically consist of a parallel assembly of several, often identical, slender deformable bodies or rods. It was highlighted that, while significant progress has been made in the statics modelling of these robots, advances in dynamics have been limited. This comparative lack of focus on dynamics can be attributed to several factors: *i*) dynamics modelling of deformable bodies is complex, and existing approaches in the literature are not always suited for robotics applications, *ii*) the parallel architecture (or closed kinematics chains) introduces challenges similar to those of rigid parallel robotics and *iii*) the absence of a unified framework for simulating these robots makes it difficult for new researchers to grasp the fundamentals of the field. In this thesis, we addressed these challenges, providing a unified framework for modelling slender deformable bodies with or without a closed, or parallel, architecture. Additionally, we demonstrated the ability to detect CPR instabilities as part of our proof of concept.

To address the modelling of CPRs, in Chapter 2 we provided a concise overview in the state of the art of modelling deformable bodies. We started recalling the Cosserat rod theory to obtain the set of Partial Differential Equations (PDE) governing their kinematics, with their dynamics derived using the Hamilton principle. Various approaches for solving the dynamics of a Cosserat rod were described and categorized into two groups, based on whether the dynamics model is solved in the strong or weak form. The latter group comprises, among other methods, the Variable Strain Approach (VSA). The VSA was successfully applied to the statics of planar CPRs, leading to significant achievements in modelling, workspace analysis, and the identification of unstable configurations. Given these results, the VSA was selected as the appropriate methodology for addressing the CPRs dynamics modelling thought the thesis. The chapter concluded with a summary of CPR modelling, highlighting the limited focus on their dynamics in existing applications.

In the subsequent Chapter 3, we addressed the dynamics modelling of hybrid structures. With hybrid structure we mean a combination of standard joints (*e.g.,* fixed, revolute, spherical) with rigid and deformable bodies, assembled to form a manipulator or a locomotor (*i.e.,* a system with a free floating base), with or without closed kinematics chains. This approach relied on an implicit scheme for the time integration, which constrained the system kinematics to be parametrised by a set of configuration variables. For any closed kinematic chain, applying a virtual cut created two serial branches, with their geometric constraints and contact wrenches accounted for separately. Under these assumptions, the system dynamics, expressed in its Lagrangian form as a system of Differential Algebraic Equations (DAE), was structured as a residual vector. To compute this vector, the standard Newton-Euler (NE) routines were extended to include deformable slender bodies, treated as special articulations called "continuum joints". A similar structure was proposed to compute the Jacobian matrix of the residual vector. This matrix was then used to iteratively refine the set of configuration variables to solve the dynamics model at each time step. The proposed approach was evaluated in comparison with existing methods from the literature, utilising standard benchmarks, including a clamped rod and a free-floating beam. Furthermore, two simulated case studies, both focused on locomotors, were presented for illustration.

In Chapter 4, the approach was tailored to CPRs, which typically feature multiple identical limbs, composed of slender deformable bodies, connected in parallel by a rigid body known as the platform. Virtual cuts are now used to fully isolate the platform, turning it into a floating rigid body while the limbs form a tree-like system, denoted as *the platform* and *the limbs*. This approach, common in rigid parallel robots, avoids breaking symmetry and eliminates ambiguity in selecting which limb to attach the platform to. NE recursive passes were applied to both subsystems to compute the CPR dynamics state. The time integration of the resulting system DAEs was addressed with an alternative implicit scheme which avoids numerical instabilities. In order to evaluate the approach, two distinct case studies were proposed. In the first prototype, sensors were employed to measure the configuration of the proximal revolute joints and the shapes of the deformable (actuated) rods. Subsequently, the aforementioned measurements were compared with the simulated behaviour, demonstrating a good level of consistency and highlighting the potential of the approach, though there remains room for further refinement. The second prototype was utilised for the evaluation of the approach's efficacy in capturing transitions between stable and unstable states. In this instance, the approach demonstrated its capacity to accurately simulate the behaviours and transition phases associated with instabilities.

In Chapter 5 we proposed a toolbox, namely NEHA, addressing the dynamics modelling of CPRs as well as the systems addressed in Chapter 3. Since our approach is based on NE, it is well-suited for object-oriented programming, enabling the development of a generic dynamics simulator. The approach was then broken down into its various components, illustrating their interactions to create a generic representation of the system as a dynamic tree. A set of implicit schemes was employed to manage the time integration of the tree's different elements, including floating bodies and joints. Thanks to this generalization, the RNEAs automatically operate on the dynamic tree, making the computation

of the residual and its Jacobian straightforward. An application of the proposed toolbox was then demonstrated using a case-study RFRFR planar CPR. The presented example demonstrated a possible implementation and emphasised the diverse functionality that the toolbox is capable of supporting. Additionally, the dynamics simulation of this case study was used to illustrate the differences among the time implicit schemes and spatial integrators available in the literature, in terms of stability and numerical performances.

## 6.2 Perspectives

In terms of future perspectives, this thesis opens two main research directions. The first focuses on enhancing the numerical stability and robustness of the approach. Since it uses an iterative prediction-correction algorithm, optimizing each step is key. Additionally, improving the numerical integration for reconstructing the kinematics state and internal stress-wrench of the deformable bodies could greatly boost performance. The second research direction focuses on leveraging the efficiency and NE foundation of our approach to address areas such as control, identification, and state estimation. Specifically, it would be valuable to explore how various NE-based methods, developed for rigid robotics, can be adapted and applied to CPRs, or more broadly, to continuum robotics. The specifics of these research directions are outlined in the following sections.

### 6.2.1 Perspectives for Numerical Efficiency

Due to the intrinsic predictive-corrective structure of the algorithm, the simulation time is directly proportional to the number of iterations of the Newton loop. The computational efficiency of the algorithm can therefore be improved in two ways. Firstly, by reducing the number of Newton iterations (*i.e.,* by improving the convergence of the loop), and secondly, by optimising the calculation of the residual and, above all, of its Jacobian and its inversion. Regarding the last point, thanks to the strong reduction power of the VSA [21], the Jacobian matrix remains moderate in size for typical continuum and soft robotics designs, and its inversion remains reasonable in terms of computation time.

#### Reducing the number of Newton iterations

In the case of closed-loop systems, we can add penalties to the augmented Lagrangian in order to increase the convexity of the original variational formulation and improve the convergence of the Newton loop. Technically, these penalties simply add restoring forces and torques to the Lagrange multipliers that initialise the $IDM$. Finally, several other additional improvements based on the scaling of the original DAE system and its tangent linearization, will be applied in order to optimize the conditioning of the residual vector and its Jacobian matrix [94], [122].

### Improving spatial integration

A detailed analysis on the computation times showed that the spatial integration of the forward and backward ODEs of the Cosserat rods takes more than 90% of the total computation time, of which 70% is spent on the forward kinematics, and 30% on the backward dynamics. Although numerical integration by spectral methods reduces the time required, compared to the usual finite difference integration [21], we are considering other options to further reduce this bottleneck, such as the integration methods based on quadrature and Magnus expansions [100, 117, 118]. As partially addressed in Chapter 5, their implementation in C++ brought drastic improvements on the overall computational time. In this context, an hybrid Magnus-spectral spatial integrator could significantly reduce the computational time of the NE passes while ensuring numerical precision and adherence to Lie Group orthogonality.

### Parallelization

Newton-Euler algorithms are naturally suited to parallelization [86]. As a first example, we can define the relative kinematics of every joint in a parallel loop before entering the forward kinematics routine. In addition, the updates of Remark 18, which are necessary before entering the backward dynamics, can be executed in a parallel loop too. Moreover, depending on the structure of the robot, if we proceed with a breadth-first algorithm (see Remark 19) in the backward passes, these can also be parallelized on every layer of the tree. Beyond this parallelization based on NE formulation, the computation of the Jacobian matrix is the main source of slowdown, since it requires the $TIDM$ to be called several times. As this matrix is calculated column-wise, the parallelization of its calculation should considerably reduce the overall calculation time.

### Efficient Jacobian Computation

An algorithm can be designed to automatically identify the structure of the Jacobian matrix, reducing the number of iterations needed for its computation. As outlined in Chapter 4, not all NE passes are required to compute every column of the Jacobian. By eliminating unnecessary passes, especially the ones requiring numerical integration, the computational time for calculating the Jacobian matrix can be significantly reduced.

### Automatization and Functionalities

The toolbox detailed in Chapter 5 remains at the proof-of-concept stage. However, being based on RNEA, several aspects can be automated, and additional functionalities can be integrated. Specifically, the NE segmentation of a system could be systematically performed by implementing an appropriate function. To achieve this, the system could be described, in terms of components and structure, using Extensible Markup Language (XML) or a Unified Robotics Description Format (URDF) file from the ROS framework [123, 124]. Additionally, the initial assembly of a system, required in the case of a CPR,

and its positioning in a desired configuration can be simplified and automated by exploring the surrounding solution neighbourhood using metrics for conditioning and stability.

### 6.2.2 Toward Dynamic Control of CPRs

This work paved the way for the application of model-based controllers to CPRs and hybrid structures, as well as for the identification of dynamics parameters and state estimation.

#### Model-based controllers

The proposed approach offers an efficient model for CPRs and SMMS. This efficiency, and inherent generality, provide the advantage of enabling the application of model-based controllers in the field of continuum robotics. Specifically, schemes like impedance control can leverage the intrinsic safety of CPRs, enabling controlled interactions with the environment.

#### Reliable state estimation

Being the approach based on a relative parametrization, it uses few parameters ($q_r$ , $q_e$, ...) requiring a reduced number of measurement for the state estimation. Moreover, combining different sensors, such as visual servoing and Fiber Bragg Grating Sensors (FBGS), could enhance reliability. In this context, artificial intelligence could play a role in advancing state estimation for the deformable slender bodies.

#### Identification of Dynamics Parameters

A CPR has a complex architecture with various parameters for its components, including the inertia of rigid bodies, the material properties of slender deformable bodies, and the friction of the rigid joints. Some of these parameters can be estimated in advance, such as the rod stiffness for the TD-CPR of Section 4.9.1 (Appendix D). However, as discussed in Chapter 4, uncertainties in material properties and geometry can affect the precision of the simulation. Improved results could be achieved through dynamics parameter identification using a white-box or grey-box approach (*i.e.,* a purely mathematical model or a combination of mathematical models with neural networks).

#### Application to non-planar CPRs

In Chapter 4, the approach was applied specifically to CPRs, with experimental validation on two planar prototypes. While the modelling was developed without assuming planar motion, it did not fully demonstrate the potential of the approach. However, a simulation with CPR having more DoFs to be controlled was not implemented due to time constraints. Conducting such a simulation could better demonstrate the approach's applicability to more complex structures. In particular, with this kind of CPRs, the $SO(3) \times \mathbb{R}^3$ integrator could be fully utilized to integrate the kinematics of the free-floating platform, experiencing motions in all directions.

# Appendix A

# Lie Algebra

In this document we used the standard rigid mechanics notations of the Lie algebra. Here, we recall the basic principles, while a more detailed account is provided by Murray [125].

## A.1 Frames and Poses

Space is provided with an inertial (fixed) frame $\mathcal{F}_e = (O_e, s_e, n_e, a_e)$ (typically attached at the robot base), where $O_e$ is the origin and the directors $(s_e, n_e, a_e) = \mathbb{1}_3$. We define a body-frame as $\mathcal{F}_b = (O_b, s_b, n_b, a_b)$ where $s_b, n_b, a_b \in \mathbb{R}^3$ are its three mutually-orthogonal unit basis vectors. The pose of this frame can be parametrized w.r.t. $\mathcal{F}_e$ (or any other reference frame) by a transformation ${}^e g_b \in SE(3)$ defined as:

$$ {}^e g_b = \begin{bmatrix} {}^e R_b & {}^e r_b \\ 0 & 1 \end{bmatrix}, \tag{A.1} $$

where ${}^e R_b \in SO(3)$ is the rotation matrix associated to the frame orientation w.r.t. $\mathcal{F}_e$, *i.e.,* ${}^e R_b = (s_b, n_b, a_b)$, and ${}^e r_b$ is the vector of its origin $O_b$ in $\mathcal{F}_e$. A pose w.r.t. $\mathcal{F}_e$ is also referred as "inertial" pose. In the case the pose is expressed with respect to another frame, say $\mathcal{F}_a$, this is expressed as ${}^a g_b$. The inverse of a pose can be expressed by inverting the subscript with the superscript ${}^a g_b^{-1} = {}^b g_a$ now expressing the pose of $\mathcal{F}_a$ w.r.t. $\mathcal{F}_b$, which details as:

$$ {}^a g_b^{-1} = \begin{bmatrix} {}^a R_b^T & -{}^a R_b^T {}^a r_b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^b R_a & {}^b r_a \\ 0 & 1 \end{bmatrix} \tag{A.2} $$

## A.2 Twists and Wrenches

The differentiation w.r.t. the time, denoted $t$, is indicated with a dot. When differentiating the pose of a body frame $\mathcal{F}_b$ w.r.t. the inertial frame $\mathcal{F}_e$, one can pull it back into the local frame to obtain a twist $\eta_{b/e} \in se(3) \cong \mathbb{R}^6$, which in our notations is expressed as:

$$ {}^b \eta_{b/e} = \begin{bmatrix} {}^b \Omega_{b/e} \\ {}^b V_{b/e} \end{bmatrix} = \left( {}^e g_b^{-1} {}^e \dot{g}_b \right)^\vee, \tag{A.3} $$

where ${}^b \Omega_{b/e}, {}^b V_{b/e} \in \mathbb{R}^3$ are respectively the angular and linear velocity of $\mathcal{F}_b$ in its local coordinates, with ${}^b \Omega_{b/e} = \left( {}^e R_b^T {}^e \dot{R}_b \right)^\vee$, and $\cdot^\vee$ is the inverse of $\hat{\cdot}$ as map $\cdot^\vee : se(3) \mapsto \mathbb{R}^6$ defined later in Section A.3. Should the frame be computed w.r.t. the frame $\mathcal{F}_a$, but still expressed in $\mathcal{F}_b$, it is denoted as ${}^b \eta_{b/a}$. The dual of a twist is the wrench $F \in se^*(3) \cong \mathbb{R}^6$, generically denoted by $F = (C^T, N^T)^T$ with $C, N \in \mathbb{R}^3$ as the vector of couples and forces respectively. A wrench acting on $\mathcal{F}_b$ in its coordinates is denoted $F_b$ while if its equivalent wrench in $\mathcal{F}_a$ is denoted ${}^a F_b = \mathrm{Ad}_{{}^b g_a}^T F_b$, where the operator Ad is described in the following section.

**Remark 22.** *For clarity, we have included all superscripts and subscripts that indicate the frame of expression and the reference frame, respectively. However, throughout this*

*document, the left superscript is omitted if it matches the frame to which the twist or wrench applies (e.g., ${}^{b}\eta_{b/e} \to \eta_{b/e}$). Similarly, the reference frame is omitted when it refers to the inertial frame (e.g., $\eta_{b/e} \to \eta_{b}$).* ■

## A.3 Adjoint Maps

We define $\widehat{\cdot}$ as a map acting on vector spaces whose definition depends on the dimension of the vector space. Specifically, applying it on $\xi = \left(K^T, \Gamma^T\right)^T$ then:

$$\widehat{\cdot} : \mathbb{R}^6 \mapsto se(3) \quad | \quad \widehat{\xi} = \begin{bmatrix} \widehat{K} & \Gamma \\ 0 & 0 \end{bmatrix}. \tag{A.4}$$

On the other hand, applying this map on a vector of $\mathbb{R}^3$, *e.g.*, $K = (K_x, K_y, K_z)^T$, it behaves as:

$$\widehat{\cdot} : \mathbb{R}^3 \mapsto so(3) \quad | \quad \widehat{K} = \begin{bmatrix} 0 & -K_z & K_y \\ K_z & 0 & -K_x \\ -k_y & k_x & 0 \end{bmatrix}. \tag{A.5}$$

The adjoint action of $SE(3)$ on its Lie algebra is given by:

$$\mathrm{Ad} : SE(3) \times se(3) \mapsto se(3) \quad | \quad \mathrm{Ad}_g = \begin{bmatrix} R & 0 \\ \hat{r}R & R \end{bmatrix}. \tag{A.6}$$

On the other hand, the adjoint action of $se(3)$ on itself is given by:

$$\mathrm{ad} : se(3) \times se(3) \mapsto se(3) \quad | \quad \mathrm{ad}_\eta = \begin{bmatrix} \widehat{\Omega} & 0 \\ \widehat{V} & \widehat{\Omega} \end{bmatrix}, \tag{A.7}$$

which is also referred as the Lie bracket [125], which, for two twists $\eta_a$ and $\eta_b$ details as:

$$[\eta_a, \eta_b] = \mathrm{ad}_{\eta_a} \eta_b \tag{A.8}$$

# Appendix B

# Newmark Scheme Expressions

The implicit Newmark scheme takes the generic form (3.9), where $a$, $b$ and the two functions $f$ and $h$ are defined for any pair of vectors of same dimension $(x, y)$, by:

$$a = \frac{\gamma}{\beta \Delta t}, \qquad b = \frac{1}{\beta \Delta t^2}, \tag{B.1}$$

$$f(x, y) = \left(1 - \frac{\gamma}{\beta}\right) x + \Delta t \left(1 - \frac{\gamma}{2\beta}\right) y, \tag{B.2}$$

$$h(x, y) = -\frac{1}{\beta \Delta t} x + \left(1 - \frac{1}{2\beta}\right) y, \tag{B.3}$$

where $(\beta, \gamma)$ are two constant parameters such that $(\beta, \gamma) = (\frac{1}{4}, \frac{1}{2})$ ensures second order accuracy with no damping, a choice that is systematically adopted in the numerical examples. Taking $(x, y) = (\dot{q}^{(n)}, \ddot{q}^{(n)})$, $(\dot{r}_0^{(n)}, \ddot{r}_0^{(n)})$, $(\Omega_0^{(n)}, \dot{\Omega}_0^{(n)})$ in (B.2) and (B.3) defines $f_q^{(n)}, f_r^{(n)}, f_\theta^{(n)}$ and $h_q^{(n)}, h_r^{(n)}, h_\theta^{(n)}$ of (3.9) and (3.10) respectively. With these notations, the matrices $C$, $A$, $B$ of (3.12), can be detailed as [40]:

$$\begin{aligned}
C(\nu_0) &= \begin{bmatrix} R_0^{(n)} \exp(\hat{\Theta}_0) & r_0^{(n)} + d_0 \\ 0_{1\times 3} & 1 \end{bmatrix}, \\
A(\nu_0) &= \begin{bmatrix} \Omega_0 \\ V_0 \end{bmatrix} = \begin{bmatrix} a\Theta_0 + f_\theta^{(n)} \\ R_0^T (a\, d_0 + f_r^{(n)}) \end{bmatrix}, \\
B(\nu_0) &= \begin{bmatrix} \dot{\Omega}_0 \\ \dot{V}_0 \end{bmatrix} = \begin{bmatrix} b\Theta_0 + h_\theta^{(n)} \\ R_0^T (b\, d_0 + h_r^{(n)}) + V_0 \times \Omega_0 \end{bmatrix},
\end{aligned} \tag{B.4}$$

Differentiating the above expressions, one can show that:

$$\begin{aligned}
\frac{\partial C}{\partial \nu_0} &= \begin{bmatrix} T(\Theta_0) & \mathbb{0}_3 \\ \mathbb{0}_3 & R_0^T \end{bmatrix}, & \frac{\partial A}{\partial \nu_0} &= \begin{bmatrix} a\mathbb{1}_3 & \mathbb{0}_3 \\ \hat{V}_0 T(\Theta_0) & a R_0^T \end{bmatrix}, \\
\frac{\partial B}{\partial \nu_0} &= \begin{bmatrix} b1_{3\times 3} & \mathbb{0}_3 \\ E_0 T(\Theta_0) + a\hat{V}_0 & L_0 R_0^T \end{bmatrix},
\end{aligned} \tag{B.5}$$

where $E_0 = (\hat{A}_0 - \hat{\Omega}_0 \hat{V}_0)$, $L_0 = (b1_{3\times 3} - a\hat{\Omega}_0)$, $A_0 = \dot{V}_0 + \Omega_0 \times V_0$, and $(R_0^T \Delta R_0)^\vee = T(\Theta_0)\Delta\Theta_0$, is the differential of the exponential of $SO(3)$ [35].

# Appendix C

# Illustrative Example

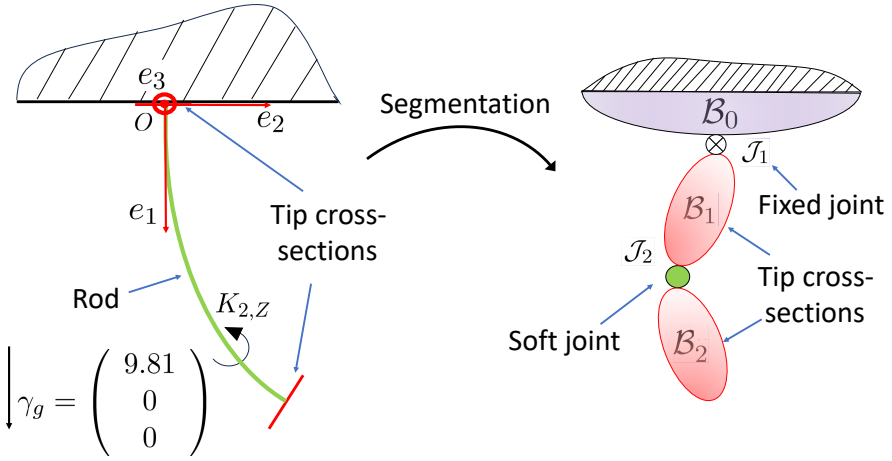



Figure C.1: Segmentation and parametrization of a cantilevered rod. The rod is fixed to the ground through $\mathcal{J}_1$. Its tip-cross sections define rigid bodies $(\mathcal{B}_1, \mathcal{B}_2)$, its internal domain is a soft joint $\mathcal{J}_2$. The basis $\mathcal{B}_0$ is fixed. $\mathcal{F}_e = \mathcal{F}_0 = (O, e_1, e_2, e_3)$ is the inertial frame. $K_{2,Z}(.)$ is the curvature field of the rod deformed in the plane $(e_1, e_2)$. $\gamma_g = (9.81, 0, 0)^T$ is the gravity acceleration field in $\mathcal{F}_e$.

To illustrate the simulation algorithm, we apply it to the simple case of the cantilevered rod of Section 3.9 (see Figure C.1). The rod has a length $\ell$, and a circular cross-section of area and axial moment of inertia $a$ and $I$ respectively. Its Young's modulus is $E$, and its density is $\rho$. According to the segmentation of Section 3.5.1, the rod corresponds to two cross-sections $\mathcal{B}_1$ and $\mathcal{B}_2$ connected together through a soft joint $\mathcal{J}_2$. This soft joint is modeled as a Kirchhoff rod moving in the $(e_1, e_2)$ plane and straight at rest, *i.e.*, $A_2 = (0, 0, 1, 0, 0, 0)^T$ and $\xi_2^0 = (0, 0, 0, 1, 0, 0)^T$, where $A_2$ selects the curvature field $K_{2,Z}(.)$ as the system's unique degrees of freedom. Once reduced on the basis $\Phi_2$ of the first $n_\epsilon$ Chebyshev polynomials, this field is replaced by the vector $q_{\epsilon,2} = q_2$ of the $n_\epsilon$ modal curvature coordinates of the rod. Referring to the general context of Section 3.3, we thus have $\chi = (q_2, \dot{q}_2, \ddot{q}_2)$ and $\overline{\chi} = q_2$, while the residual vector (constrained by the scheme) and its Jacobian are $\overline{\mathcal{R}} = \overline{\mathcal{R}}_2$ and $(\partial\overline{\mathcal{R}}/\partial\overline{\chi}) = (\partial\overline{\mathcal{R}}_2/\partial q_2)$ respectively. According to the flowchart 3.2, the predictor-corrector algorithm consists of an inner Newton correction loop included in an outer time-loop. At each new time step $t_{n+1}$, the state $(q_2, \dot{q}_2)$ is first predicted by forcing $\ddot{q}_2 = 0_{n_\epsilon \times 1}$ in the Newmark scheme (3.9) (inertial predictor), before entering the Newton loop, whose correction equation (3.16) requires computing both $\overline{\mathcal{R}}_2$ and $(\partial\overline{\mathcal{R}}_2/\partial q_2)$. The computation of $\overline{\mathcal{R}}_2$ is performed with the $\overline{IDM}_\star$ algorithm of Section 3.7.3. Since $\mathcal{J}_2$ is soft, $\overline{\mathcal{R}}_2$ is defined by (3.61), where $j = 2$, $\Lambda_{d,2} = 0_{6\times 1}$ (no internal actuation), $\int_0^{\ell_2} \Phi_2^T A_2^T \mathcal{H}_2 A_2 \Phi_2 \, dX = EI \, \mathbb{1}_{n_\epsilon}$ ($\Phi_2$ is orthonormal), and $\Lambda_2(.)$ is calculated by backward integrating (3.41) from $\Lambda_2(\ell) = 0_{6\times 1}$ (no tip load), with $\mathcal{M}_2 = \rho \, \mathrm{diag}(2I, I, I, a, a, a)$ and $\overline{F}_{\mathrm{ext},2} = (0_{1\times 3}, (R_2^T \gamma_g)^T)^T$ (gravity). The integration of (3.41) requires first to compute the inertial kinematic fields along the rod $(g_2, \eta_2, \dot{\eta}_2)(.)$. Since

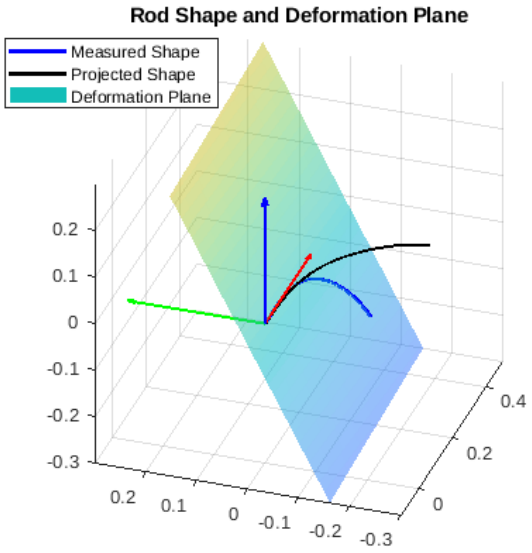
$\mathcal{J}_1$ is fixed, one can take $\mathcal{F}_1 = \mathcal{F}_e$, and $(g_2, \eta_2, \dot{\eta}_2)(.) = ({}^1g_2, \eta_{2/1}, \dot{\eta}_{2/1})(.)$ is obtained by forward integrating (3.38), starting from $({}^1g_2, \eta_{2/1}, \dot{\eta}_{2/1})(0) = (\mathbb{1}_4, 0_{6\times1}, 0_{6\times1})$, with $(\xi_2, \dot{\xi}_2, \ddot{\xi}_2) = (A_2\Phi_2 q_2 + \xi_2^0, A_2\Phi_2\dot{q}_2, A_2\Phi_2\ddot{q}_2)$ and $(\dot{q}_2, \ddot{q}_2)$ function of $q_2$, and $(q_2^{(n)}, \dot{q}_2^{(n)}, \ddot{q}_2^{(n)})$ through the Newmark scheme (3.9).
The computation of $(\partial\overline{\mathcal{R}}_2/\partial q_2)$ is performed with the $\overline{TIDM}_\star$ algorithm of Section 3.7.3. This algorithm calculates the variation $\Delta\overline{\mathcal{R}}_2$ of the outputs of the algorithm $\overline{IDM}_\star$, due to a variation $\Delta q_2$ of its inputs. Exploiting the identity $\Delta\overline{\mathcal{R}}_2 = (\partial\overline{\mathcal{R}}_2/\partial q_2)\Delta q_2$ shows that by imposing $\Delta q_2 = \delta_\alpha$, $\alpha = 1, 2...n_\epsilon$, we can calculate all the columns of $(\partial\overline{\mathcal{R}}_2/\partial q_2)$ one after the other. This algorithm proceeds as the previous one except that (3.63) replaces (3.61) and the backward and forward ODEs (3.41) and (3.38) are replaced by (3.51) and (3.47), the first being initialized with $\Delta\Lambda_2(\ell) = 0_{6\times1}$, and the second by $\Delta\zeta_{2/1}(0) = \Delta\eta_{2/1}(0) = \Delta\dot{\eta}_{2/1}(0) = 0_{6\times1}$. In (3.51) $\Delta\bar{F}_{\text{ext},2} = (0_{1\times3}, (\Delta R_2^T\gamma_g)^T)^T$ with $\Delta R_2^T\gamma_g = ((R_2^T\gamma_g)^\wedge, 0_{3\times3})\Delta\zeta_2$. In (3.47), $(\Delta\dot{q}_2, \Delta\ddot{q}_2) = (a\Delta q_2, b\Delta q_2)$. This simulation algorithm is summarized by the pseudo-code in Algorithm 15, where all space-integrations of ODEs are achieved in cascade with a standard spectral method as indicated in Section 3.8. Note also that in this particular case, $\mathcal{J}_1$ being fixed and the cross-sections of the rod ends $\mathcal{B}_1$ and $\mathcal{B}_2$ being massless, their model, used in the boundary conditions, introduces trivial twist and wrench transfers.

Initial conditions:
$q_2 = q_{2\,\text{init}}$, $\dot{q}_2 = \ddot{q}_2 = 0_{n_\epsilon \times 1}$
Boundary conditions: ${}^1g_2(0) = \mathbb{1}_4$, $\eta_{2/1}(0) = \dot{\eta}_{2/1}(0) = \Lambda_2(\ell) = 0_{6\times 1}$
$\Delta\zeta_{2/1}(0) = \Delta\eta_{2/1}(0) = \Delta\dot{\eta}_{2/1}(0) = \Delta\Lambda_2(\ell) = 0_{6\times 1}$
**for** $n = 0 \dots (t_f - t_0)/\Delta t$ **do**
  $q_2^{(n)} := q_2$, $\dot{q}_2^{(n)} := \dot{q}_2$, $\ddot{q}_2^{(n)} := \ddot{q}_2$
  *Prediction (Newmark scheme s.t. $\ddot{q}_2 = 0_{n_\epsilon\times 1}$):*
  $q_2 = q_2^{(n)} - (1/b)h_q^{(n)}$, $\dot{q}_2 = f_q^{(n)} - (a/b)h_q^{(n)}$, $\ddot{q}_2 = 0_{n_\epsilon\times 1}$
  *Correction:* $\overline{\mathcal{R}}_2 = 1$
  **while** $\overline{\mathcal{R}}_2 > \epsilon$ **do**
    *Computation of* $\overline{\mathcal{R}}_2$ with $\overline{IDM}_\star$:
    $\xi_2 := A_2\Phi_2 q_2 + \xi_2^0$, $\dot{\xi}_2 := A_2\Phi_2\dot{q}_2$, $\ddot{\xi}_2 := A_2\Phi_2\ddot{q}_2$
    Integrate forward from $X = 0$ to $l$ Eq. (3.38) to get:
    $g_2 := {}^1g_2$, $\eta_2 := \eta_{2/1}$, $\dot{\eta}_2 := \dot{\eta}_{2/1}$
    Integrate backward from $X = \ell$ to 0 Eq. (3.41) to get: $\Lambda_2$
    Compute $\overline{\mathcal{R}}_2$ using (64)
    *Computation of* $(\partial\overline{\mathcal{R}}_2/\partial q_2)$ with $\overline{TIDM}_\star$:
    **for** $\alpha = 1 \dots n_\epsilon$ **do**
      $\Delta\xi_2 := A_2\Phi_2\delta_\alpha$, $\Delta\dot{\xi}_2 := a\Delta\xi_2$, $\Delta\ddot{\xi}_2 := b\Delta\xi_2$
      Integrate forward from $X = 0$ to $l$ Eq. (3.47) to get: $\Delta\zeta_2 := \Delta\zeta_{2/1}$,
      $\Delta\eta_2 := \Delta\eta_{2/1}$, $\Delta\dot{\eta}_2 := \Delta\dot{\eta}_{2/1}$
      Integrate backward from $X = \ell$ to 0 Eq. (3.51) to get $\Delta\Lambda_2$
      Compute $\Delta\overline{\mathcal{R}}_2$ using (3.63)
      $\alpha^{\text{th}}$ column of $(\partial\overline{\mathcal{R}}_2/\partial q_2) := \Delta\overline{\mathcal{R}}_2$;
    **end**
    Update $q_2 := q_2 - (\partial\overline{\mathcal{R}}_2/\partial q_2)^{-1}\overline{\mathcal{R}}_2$ and $\dot{q}_2, \ddot{q}_2$ with (3.9)
  **end**
**end**

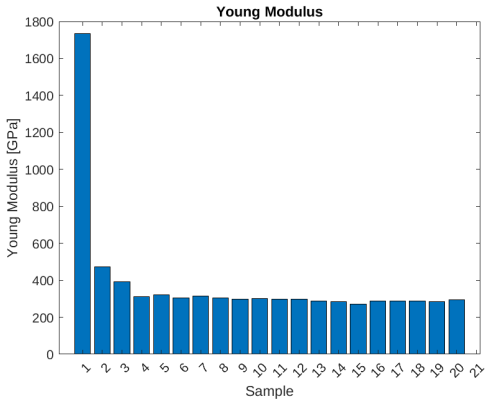
**Algorithm 15:** Simulation of the cantilevered rod.

# Appendix D

# Estimation of Young Modulus Using FBGS




(a) Measured shape of the bended rod (blue), belonging to its deformation plane, and its planar projection (black).




(b) Young modulus (in $GPa$) obtained from recorded bending and applied weight.

Figure D.1: Measured rod shape and respective deformation plane (left) and the computed Young modulus for each tension applied on the tendon-driven rod.

FBGS sensors, like those installed on the TD-CPR in Section 4.9.1, can be used to estimate the material properties of two slender rods. To achieve this, a tendon-driven rod of the same type as that on the TD-CPR was subjected to a set of known tensions $T_s$. Uncertainties in sensor placement can affect the measured shape and must be corrected before processing. To address this, the shapes are first projected onto their deformation plane, which may differ from the $xy$ plane (Figure D.1a). The plane coordinates are calculated using SVD decomposition, where the upper $3 \times 3$ matrix in V represents the rotation matrix of the containing plane. The rod shapes are then projected into these plane coordinates (black shape in Figure D.1a), while ensuring that all rotation matrices are right-handed and the projected shapes start correctly tangent to the $x$-axis. It is then possible to calculate the corresponding curvatures $K_s$, which are constant for a tendon-driven rod. Using the constitutive law (2.12) in the context of a rod deforming in a plane with constant curvature and with $\Lambda = 0$, it becomes possible to solve for the Young's modulus $E$:

$$E_s = \frac{\ell}{I_{zz} K_s} \int_0^1 \frac{\Phi A^T \Lambda_{act}}{\Phi^T \Phi} dX \tag{D.1}$$

with $I_{zz}$ the geometrical moment of inertia of the rod cross-sections along their $z$ axis. The results of this estimation are shown in Figure D.1b, where the obtained values of $E$ are consistent at higher loads. However, at lower loads, uncertainties due to sensor placement tolerances and residual plastic deformation of the rod resulted in non-negligible errors. As a result, the first three samples were discarded, and the average was computed using the remaining samples, yielding $E = 293\ GPa$ with a negligible standard deviation.

# Appendix E

# Application of the NEHA toolbox

```
48
49      ators::LocalizedActuator motor_1(τ_act,1(t))
50      ators::LocalizedActuator motor_2(τ_act,2(t))
51
52      crosp::CircularCrossSection cross_section(r)
53      crosp::MaterialProperties material
54      crosp::RodProperties rod_properties(material, cross_section, ℓ)
55
56      deformation_modes = (0,0,n_e,0,0,0)
57      crosp::VariableStrainParametrization vsa(deformation_modes)
58
59      crosp::CosseratRod rod_1(rod_properties, vsa)
60      crosp::CosseratRod rod_2(rod_1)
61
62      joints::RigidJoint joint_1(A_1, actuator_1, ⁰g_1,c)
63      joints::RigidJoint joint_4(A_4, actuator_2, ⁰g_4,c)
64
65      joints::RigidJoint joint_3, joint_6
66
67      joints::ContinuumJoint joint_2(rod_1)
68      joints::ContinuumJoint joint_5(rod_2)
69
70      bodies::RigidBody body_0, body1, ..., body_6
71
72      nodes::Node node_6(body_6, joint_6)
73      nodes::Node node_5(body_5, joint_5, {node_6})
74      nodes::Node node_4(body_4, joint_4, {node_5})
75      nodes::Node node_3(body_3, joint_3)
76      nodes::Node node_2(body_2, joint_2, {node_3})
77      nodes::Node node_1(body_1, joint_1, {node_2})
78
79      nodes::BaseNode node_0(body_0, {node_1, node_4})
80
81      leafs::VirtuallyCutJoint leaf_1(node_3, node_6, Ā_p)
82
83      tree::Tree robot_tree({node_0}, {motor_1, motor_2}, {leaf_1})
84
85      tims::CommutativeGroup commutative_integrator
86      tims::LieGroup lie_integrator
87
88      robot_tree→findNode(1)→setCoordinates(π/2)
89      robot_tree→findNode(4)→setCoordinates(π/2)
90
91      neha::DynamicSimulator simulator(robot_tree, {lie_integrator},
            commutative_integrator)
92
93      simulator→dynamicsSimulation(t)
```

The code presented above shows the implementation of the RFRFR planar CPR of Section 5.13. In the following, we proceed to detail the declaration of the required object and some of the functionalities of the NEHA toolbox.

## E.1 Implementing the Actuators

```
1
2       ators::LocalizedActuator motor_1(τ_act,1(t))
3       ators::LocalizedActuator motor_2(τ_act,2(t))
```

We initialize the two actuators, denoted motor 1 and motor 2, with the two functions $\tau_{act,1}(t)$ and $\tau_{act,2}(t)$ implementing their actuation laws (lines 2–3). The two actuators were declared as *LocalizedActuator* objects, defined in Section 5.12. In the event that distributed actuations are to be employed, one can follows the procedure of for a *TendonDrivenActuation* of Remark 23.

**Remark 23.** *For the TD-CPR of Section 4.9.1, each of the two rods has a pair of antagonist tendons. To implement the TendonDrivenActuation of Section 5.12.2 one needs to first define the actuation law for the tendons tension, denoted $T_1(t)$ and $T_2(t)$, and implemented by two LocalizedActuators.*

```
ators::LocalizedActuator motor_1(T_1(t))
ators::LocalizedActuator motor_2(T_2(t))
```

*Then the routing of the tendon has to be defined by $D(X)$ and $D'(X)$ as follows:*

```
ators::Tendon tendon_1(D_1, D'_1)
ators::Tendon tendon_2(D_2, D'_2)
```

*In this case they simplify to: $D_2 = -D_1 = (0, d_c, 0)^T$ and $D'_1 = D'_0 = (0, 0, 0)^T$, where we remind $d_c = 11{\cdot}10^{-3}$ m is the cable offset from the rod centreline. Finally the tendon-driven actuation is the collection of the tendons with the associated actuators (Section 5.12)*

```
ators::TendonDrivenActuation td_atuation_1({{motor_1, tendon_1}, {
    motor_2, tendon_2}};
```

*where the pairs* `{motor_1, tendon 1}` *and* `{motor 2, tendon 2}` *are passed to the TendonDrivenActuation object defining the distributed actuation of the first rod. The actuation of the other rod can be defined in a similar manner. Should a rod have more actuated tendons, the list of pairs can be extended as required.* ∎

## E.2 Defining the Rod Properties

```
4
5       crosp::CircularCrossSection cross_section(r)
6       crosp::MaterialProperties material
7       crosp::RodProperties rod_properties(material, cross_section, ℓ)
```

The rod geometry is specified by `crosp::CircularCrossSection cross_section(r)` (line 5) creating a *CircularCrossSection* object. This object computes the relative geometric quantities such as area and moments of inertia using the constant radius indicated by the scalar $r$.

Different shape of the cross-section can be defined, as described in Remark 24. The material properties of the rod are defined on line 6 by the *MaterialProperties* object (Section 5.11) which, by default, sets $E$, $G$ and $\rho$ as standard spring steel (the material used by our case study). Should a different material be required, the material properties can be changed accordingly (Remark 25). Then, the *RodProperties* object collects the aforementioned properties passed as parameters (line 7). By default, this object has the properties defined in lines (5–6). Therefore, the same outcome could have been achieved by using `crosp::RodProperties rod_properties`.

**Remark 24.** *Different types of cross sections can be created by selecting the appropriate object. For example, the rectangular cross-section of the TD-CPR was defined as:*

```
crosp::RectangularCrossSection cross_section(w, h)
```

*where the scalars $w$ and $h$ defines the the cross-section width and height. Furthermore, passing a function, instead of a scalar, such as $r(X)$ results in a $X$-parametrized cross section shape (e.g., the one of the bio-inspired swimmer of Section 3.9.3).* ∎

**Remark 25.** *For a material with specific properties, $E$, $G$, $\rho$ and $\mu$ can be given to the MaterialProperties object as follows:*

```
crosp::MaterialProperties material(E, G, ρ, μ)
```

*Alternatively, if only a few of these properties differ from the default values, they can be individually modified. For instance, if for a material slightly heavier than standard spring steel ($\rho = 8000\ kg.m^{-3}$), this parameter can be updated as follows:*

```
crosp::MaterialProperties material
material→ρ=8000
```

*As any other material property.* ∎

Once the material and geometric properties of the rod have been implemented, the next step is to define its strain-based parametrization.

## E.3 Rod Strain Parametrization

```
8
9       deformation_modes = (0,0,n_e,0,0,0)
10      crosp::VariableStrainParametrization vsa(deformation_modes)
```

We parametrize the strain field of the rod by selecting the allowed deformations and the corresponding number of modes. In this example, the rod deforms within a plane (bending along its $z$-axis), which we parametrized using $n_e = 3$ modes, defined through the *deformation_modes* vector (line 9). For the parametrization of Kirchhoff or Reissner rod, refer to Remark 26. This vector is given as input to the *VariableStrainParametrization* object

of Section 5.11 (line 10), which, by default, has a reference strain field $\xi^0 = (0, 0, 0, 1, 0, 0)^T$ and uses Legendre polynomials for the basis $\Phi$. The reference strain field can be changed by adding it to the construction list (Remark 26) while, the basis can be changed using `crosp::VariableStrainParametrization<crosp::polynomials::TYPE>` in line 10, where `TYPE` is to be replaced by `Legendre` or `Chebyshev`.

**Remark 26.** *Having to implement a Kirchhoff's, i.e., a rod with torsion and bending, requires $n_a = 3$ DoFs of the rod and one would need to define the deformation modes as:*

$$deformation\_modes = (n_{e,1}, n_{e,2}, n_{e,3}, 0, 0, 0)$$

*where $n_{e,1}$ $n_{e,2}$ and $n_{e,3}$ defines the number of modes parametrizing the torsion and bending along the rod $a$ and $n$ directors, respectively. If stretching and/or axial shear are to be accounted for (Reissner rod), one can add them into deformation_modes in a similar manner:*

$$deformation\_modes = (n_{e,1}, n_{e,2}, n_{e,3}, n_{e,4}, n_{e,5}, n_{e,6})$$

*where $n_{e,4}$ $n_{e,5}$ and $n_{e,6}$ define the number of modes parametrizing the stretching (or compression) and the two axial shear, along $a$ and $n$, respectively. Finally, if a rod has a predefined shape different from the straight configuration, such as a constant curvature, this information can be specified as follows:*

$$xi\_0 = (0, K_y^0, 0, 1, 0, 0)$$

$$crosp::VariableStrainParametrization\ vsa(deformation\_modes, xi\_0)$$

*where the value $K_y^0$ express the constant curvature of the rod.* ∎

## E.4 Creating the CosseratRod objects

The *CosseratRod* object can be defined by passing the *rod_properties* and the *vsa* as parameters.

```
11
12          crosp::CosseratRod rod_1(rod_properties, vsa)
13          crosp::CosseratRod rod_2(rod_1)
```

If a rod needs to be replicated, such as the second rod in our case study, which is identical to the first, this can be accomplished by passing the rod to be copied as a parameter, as shown in line 13. This will clone all of its properties to create a new rod.

The *CosseratRod* object utilises a spatial integration of the rod ODEs embedded within the $IDM$ (2.9 and 2.20) and $TIDM$ (Section 3.6.2). Different spatial integrators could be used to integrate the kinematics and dynamics PDEs. They can be chosen by changing the declaration of the *CosseratRod* object by `crosp::CosseratRod<crosp::integrators::TYPE>` where `TYPE` is to be replaced by `ODE45` for the Runge-Kutta integrator, `Spectral<`$N_c$`>` for the

spectral method and `Magnus<`$N_c$`>` for the integration via Magnus expansions, where $N_c$ indicates the dimension of the discretization grid. For conciseness, we detail the differences among these integrators later, in Section 5.13.4.

**Remark 27.** *In the case of an internally actuated rod the actuation system is provided as a parameter to the CosseratRod object, which is then stored in the spatial integrators. For instance, in the case of the TD-CPR (Section 4.9.1), the two rods are provided with a TendonDrivenActuations. For the first rod, the distributed actuation was described in Remark 23 and can be used as:*

```
crosp::CosseratRod rod_1(rod_properties, vsa, td_atuation_1)
```

*While the second rod is initialized in a similar manner.* ∎

Once introduced these physical parameters, the user must define the NE segmentation of the system (Section 5.13.2. On the basis of this segmentation, we can now proceed to the implementation of the CPR joints.

## E.5 Implementation of Continuum and Rigid Joints

```
14
15      joints::RigidJoint joint_1(A_1, actuator_1, ^0g_{1,c})
16      joints::RigidJoint joint_4(A_4, actuator_2, ^0g_{4,c})
17
18      joints::RigidJoint joint_3, joint_6
19
20      joints::ContinuumJoint joint_2(rod_1)
21      joints::ContinuumJoint joint_5(rod_2)
```

We start defining the *LocalizedActuator* objects by passing as parameters their DoFs matrix, actuation and relative pose w.r.t. the antecedent node (lines 15 and 16). For the other localized joints, we use the default constructor of *LocalizedActuator* defining a passive fixed joint with ${}^k g_{j,c} = \mathbb{1}_4$ (line 18). For what concerns a *ContinuumJoint*, it always requires the respective *CosseratRod* (lines 20–21).

**Remark 28.** *As long as the definition of rigid and continuum joints respects the properties listed in Table 5.1, they can be defined in any order, i.e., one could choose to define each joint in accordance with the indexation of the aforementioned NE segmentation.* ∎

## E.6 Implementation of Rigid Bodies

```
22
23      bodies::RigidBody body_0, body1, ..., body_6
```

Subsequently, all the *RigidBody* objects are defined, which, by default, are constructed as fictitious rigid bodies. If a body has a specific inertia and/or dimensions, it can be assigned to its object in construction (see Remark 29 below).

**Remark 29.** *For a real rigid body, such as the floating platform of a CPR, the corresponding RigidBody is initialized as follows:*

```
bodies::RigidBody platform_body(M_p)
```

*where $\mathcal{M}_p$ is the body inertia tensor.* ∎

Once the bodies and joints have been defined, the subsequent step is to group them in accordance with their relationship to the tree structure.

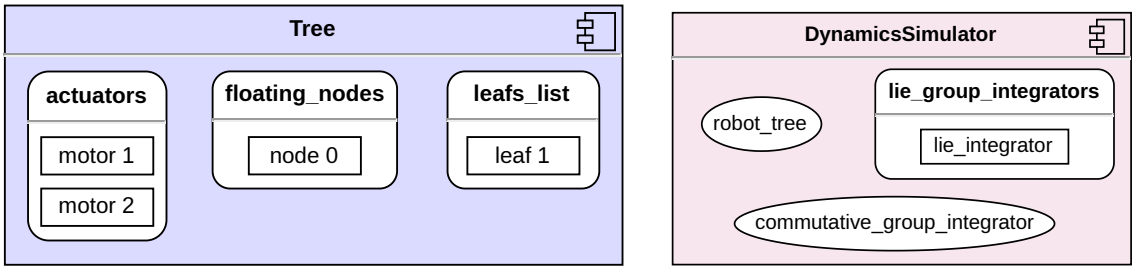
## E.7 Implementation of Nodes

The *Node* object, along with its specialised variants, is designed to address the interactions between each rigid body and the associated joint, if any, with both successors and antecedent ones (Section 5.8).

```
24
25          nodes::Node node_6(body_6, joint_6)
26          nodes::Node node_5(body_5, joint_5, {node_6})
27          nodes::Node node_4(body_4, joint_4, {node_5})
28          nodes::Node node_3(body_3, joint_3)
29          nodes::Node node_2(body_2, joint_2, {node_3})
30          nodes::Node node_1(body_1, joint_1, {node_2})
31
32          nodes::BaseNode node_0(body_0, {node_1, node_4})
```

In detail, the nodes are defined following the properties reported in Table 5.2 thus giving as parameter the respective body and, if applicable, the joint and the successors (indicated in between the brackets { }). If the nodes are created from their highest ID number, then each *Node* can be directly assigned with its successor(s), as detailed in lines 26–27 and 29–32. On the other hand, by default a *Node* object can be initialized without successors (see lines 25, 28). Should a successors be added during simulation (dynamic tree) one can follow the procedure described in Remark 30. Ultimately, the first node of our case study is of the *BaseNode* type (line 32), which implies that the node is immobile (see Section 5.8), and is the sole instance that does not necessitate a joint as a parameter.

**Remark 30.** *To describe how successors can be added to a node during simulation, we propose another approach to define the nodes of Table 5.2.*




(a) Members of the *Tree* for the RFRFR CPR

(b) Members of *DynamicsSimulator* object for the RFRFR CPR

Figure E.1: Representation of the actual members of the *Tree* and *DynamicsSimulator* for the RFRFR CPR

```
nodes::BaseNode node_0(body_0)
nodes::Node node_1(body_1, joint_1)
nodes::Node node_2(body_2, joint_2)
nodes::Node node_3(body_3, joint_3)
nodes::Node node_4(body_4, joint_4)
nodes::Node node_5(body_5, joint_5)
nodes::Node node_6(body_6, joint_6)

node_0->addSuccessor({node_1, node_4})
node_1->addSuccessor(node_2)
node_2->addSuccessor(node_3)
node_4->addSuccessor(node_5)
node_5->addSuccessor(node_6)
```

*Where we use the Node addSuccessors function (Section 5.8), a successor can be added to a node by passing the is as a parameter.* ∎

**Remark 31.** *It is important to highlight that the defining feature of this system that classifies it as a manipulator is the implementation of node 0 with a BaseNode object. Should this node be implemented as a FloatingNode, the system would then be classified as a locomotor.* ∎

## E.8 Defining the Leafs

```
33
34      leafs::VirtuallyCutJoint leaf_1(node_3, node_6, $\overline{A}_p$)
```

Once the nodes have been defined, the *VirtuallyCutJoint* object is then declared, specifying the nodes that are affected by the virtual cut and the constrained DoF matrix. In this example, the virtual cut of $\mathcal{J}_p$ resulted in the generation of two nodes, designated as 3 and 6 (Table 5.2). These nodes are passed as parameter with an arbitrary order. The associated constrained matrix $\overline{A}_p$ is that which would typically be associated with a revolute joint

with supporting axis $a_p$:

$$\overline{A}_p = \begin{bmatrix} \mathbb{1}_2 & 0_{2\times 4} \\ \mathbb{0}_3 & \mathbb{1}_3 \end{bmatrix}. \tag{E.1}$$

Finally, the *Tree* object can be constructed by giving the first *Node*, the actuators list and the list of *Leafs*.

## E.9 Composing the Tree

```
35
36          tree::Tree robot_tree({node_0}, {motor_1, motor_2}, {leaf_1})
```

In this case, `{motor_1, motor_2}` defined the list of actuators and `{leaf_1}` the list of leafs. The *robot_tree* so defined is represented in Figure E.1a, with attention of the members and the elements of each list. The *Tree* object is also responsible to define $\chi$, which for this study case has the following definition:

$$\chi = (q, \dot{q}, \ddot{q}, \lambda_p), \tag{E.2}$$

where $q = \left(q_1, q_2^T, q_4, q_5^T\right)^T$ with $q_1$ and $q_4$ representing the rotation angles for the revolute joints ($\mathcal{J}_1$ and $\mathcal{J}_4$), $q_2$ and $q_5$, both in $\mathbb{R}^{n_e}$, the elastic generalized coordinates for the two rods ($\mathcal{J}_2$ and $\mathcal{J}_5$) while $\lambda_p$ is the set of Lagrange multipliers associated to the virtually cut joint $\mathcal{J}_p$. As discussed in Section 3.3, the set $\chi$ is reduced to $\overline{\chi}$ by the constraints of an time implicit scheme, which implementation is the subject of the following section.

**Remark 32.** *For a CPR segmented as described in Chapter 4, the subsystem of the platform would have required its $SO(3)\times\mathbb{R}^3$ parametrization, changing (E.2) to:*

$$\chi = (g_p, \eta_p, \dot{\eta}_p, q, \dot{q}, \ddot{q}, \lambda), \tag{E.3}$$

*with $\lambda$ now containing all the required Lagrange multipliers.* ∎

## E.10 Time Integration with Implicit Schemes

Having defined the tree, we can now provide a detailed account of the implicit schemes (Section 5.5). Within the context of this study case, the implicit schemes allows to obtain $\overline{\chi} = (q, \lambda)$ from (E.2).

```
37
38          tims::CommutativeGroup commutative_integrator
39          tims::LieGroup lie_integrator
```

The *CommutativeGroup* integrator (line 38) is required for the generalized coordinate of the system, parametrizing $\dot{q}$ and $\ddot{q}$ w.r.t. $q$. In this case, this is sufficient to obtain $\overline{\chi} = (q, \lambda)$ from (E.2). Subsequently, the initialization of the *LieGroup* integrator is always required as it is associated with the first node of tree which is of type *FloatingNode* or the *BaseNode* (see Remark 33 below). Additionally, for a CPR segmented as described in

**Function** `staticInitialization()`:
  $robot\_tree \rightarrow updateActuations()$

  $computeResidual()$

  **while** $\left\|\overline{\mathcal{R}}\right\| > \varepsilon$ **do**
    **for** $i = 1, \ldots, dim\,(\overline{\chi})$ **do**
      $\Delta\overline{\chi} \leftarrow 0$
      $\Delta\overline{\chi}\,[i] = 1$

      $robot\_tree \rightarrow forwardTangentKinematics()$

      $robot\_tree \rightarrow backwardTangentDynamics()$
    **end**
    $\Delta\overline{\chi} = -\overline{\mathcal{J}}^{-1}\overline{\mathcal{R}}$
    $computeResidual()$
  **end**
**end**

**Algorithm 16:** The Algorithm for the layout of the statics initialization phase.

Chapter 4, the platform subsystem would require a second integrator of type *LieGroup*, enabling the computation of $\overline{\chi} = (\nu_p, q, \lambda)$ from (E.3) in Remark 32.

By default, the integrators use the Generalized-$\alpha$ implicit scheme described in Section 4.5.1, but other implicit schemes can be selected. For example, to replace the time implicit scheme of the *LieGroup* integrator, `implicit_schemes::LieGroup<implicit_schemes::TYPE>` is to be used, where `TYPE` is to be replaced by `Newmark`, `HHT<`$\rho_\infty$`>` or `GeneralizedAlpha<`$\rho_\infty$`>`, where $\rho_\infty$ specifies the rate of numerical damping [108]. The impact of these implicit schemes and the influence of the spectral radius will be addressed later in Section 5.13.4. All implicit integrators are initialized with the time-step $\Delta t$, if a desired length is not given, the default value of 0.01 s is assumed (which is the one used in our example as well).

**Remark 33.** *The first time integrator being of type LieGroup, it will either serve the kinematics of a locomotor base $(g_0, \eta_0, \dot{\eta}_0)$ or, in the case of a manipulator (fixed) base, have no effect, i.e., we consider $(g_0, \eta_0, \dot{\eta}_0) = (\mathbb{1}_4, 0, 0)$ constant over time. The selected layout allows for a general approach, though it involves some superfluous computations. However, these are relatively inexpensive in terms of computational time compared to the numerical integration (spatial integrators).* ∎

## E.11 Constructing the *DynamicsSimulator*

With the time integrators defined, the final step is to assemble the *DynamicSimulator* (Section 5.4). This is done by providing the *robot_tree* and the time integrators as parameters,

resulting in the object depicted in Figure E.1b.

```
40
41          robot_tree->findNode(1)->setCoordinates(π/2)
42          robot_tree->findNode(4)->setCoordinates(π/2)
43
44          neha::DynamicSimulator simulator(robot_tree, {lie_integrator},
                commutative_integrator)
```

As mentioned in Section 5.4, before starting the dynamics simulation, it is necessary to first find a suitable static configuration where the system is in internal equilibrium. Additionally, in the case of virtual cuts, the system must be correctly reassembled in a suitable configuration. Such configuration is found automatically by the *DynamicsSimulator* object, upon construction (line 44), using a the resolution of the statics model detailed in Algorithm 16. The initial condition can be specified, prior definition the *DynamicsSimulator* object, by assigning the desired values for some of the system joints (lines 41–42).

## E.12 Launch the System Dynamics

Once the static configuration has been found, the dynamics simulation can be started.

```
45
46          simulator->dynamicsSimulation(t)
```

In line 46 the simulation is started for a time of $t$ seconds. At the end of the simulation, a *.csv* file is returned containing the value of $\chi$ at every time step.

# Bibliography


[1] G. Robinson and J. Davies, "Continuum robots - a state of the art," *International Conference on Robotics and Automation*, vol. 4, pp. 2849–2854, 1999.

[2] V. Anderson, R. Horn, and A. S. of Mechanical Engineers, *Tensor Arm Manipulator Design*. American Society of Mechanical Engineers. Papers, American Society of Mechanical Engineers, 1967.

[3] M. Russo, S. M. H. Sadati, X. Dong, A. Mohammad, I. D. Walker, C. Bergeles, K. Xu, and D. A. Axinte, "Continuum robots: An overview," *Advanced Intelligent Systems*, vol. 5, Mar. 2023.

[4] S. Kolachalama and S. Lakshmanan, "Continuum robots for manipulation applications: A survey," *Journal of Robotics*, vol. 2020, pp. 1–19, 2020.

[5] J. Burgner-Kahrs, D. C. Rucker, and H. Choset, "Continuum robots for medical applications: A survey," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1261–1280, 2015.

[6] P. E. Dupont, N. Simaan, H. Choset, and C. Rucker, "Continuum robots for medical interventions," *Proceedings of the IEEE*, vol. 110, pp. 847–870, July 2022.

[7] R. K. Katzschmann, A. D. Marchese, and D. Rus, "Autonomous object manipulation using a soft planar grasping manipulator," *Soft Robotics*, vol. 2, pp. 155–164, Dec. 2015.

[8] T.-D. Nguyen and J. Burgner-Kahrs, "A tendon-driven continuum robot with extensible sections," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, sep 2015.

[9] J. Burgner, P. J. Swaney, R. A. Lathrop, K. D. Weaver, and R. J. Webster, "Debulking from within: A robotic steerable cannula for intracerebral hemorrhage evacuation," *IEEE Transactions on Biomedical Engineering*, vol. 60, pp. 2567–2575, Sept. 2013.

[10] R. K. Katzschmann, J. DelPreto, R. MacCurdy, and D. Rus, "Exploration of underwater life with an acoustically controlled soft robotic fish," *Science Robotics*, vol. 3, Mar. 2018.

[11] J. Ding, R. E. Goldman, K. Xu, P. K. Allen, D. L. Fowler, and N. Simaan, "Design and coordination kinematics of an insertable robotic effectors platform for single-port access surgery," *IEEE/ASME Transactions on Mechatronics*, vol. 18, pp. 1612–1624, Oct. 2013.

[12] J. Till, C. E. Bryson, S. Chung, A. Orekhov, and D. C. Rucker, "Efficient computation of multiple coupled cosserat rod models for real-time simulation and control of parallel continuum manipulators," may 2015.

[13] G. Boettcher, S. Lilge, and J. Burgner-Kahrs, "Design of a reconfigurable parallel continuum robot with tendon-actuated kinematic chains," *IEEE Robotics and Automation Letters*, vol. 6, pp. 1272–1279, Apr. 2021.

[14] Z. Yang, X. Zhu, and K. Xu, "Continuum delta robot: a novel translational parallel robot with continuum joints," July 2018.

[15] X. Huang, Z. J. Patterson, A. P. Sabelhaus, W. Huang, K. Chin, Z. Ren, M. K. Jawed, and C. Majidi, "Design and closed-loop motion planning of an untethered swimming soft robot using 2d discrete elastic rods simulations," *Advanced Intelligent Systems*, vol. 4, Sept. 2022.

[16] B. Kang, B. Yeung, and J. K. Mills, "Two-time scale controller design for a high speed planar parallel manipulator with structural flexibility," *Robotica*, vol. 20, pp. 519–528, Sept. 2002.

[17] G. Piras, W. Cleghorn, and J. Mills, "Dynamic finite-element analysis of a planar high-speed, high-precision parallel manipulator with flexible links," *Mechanism and Machine Theory*, vol. 40, pp. 849–862, July 2005.

[18] X. Zhang, J. K. Mills, and W. L. Cleghorn, "Dynamic modeling and experimental validation of a 3-prr parallel manipulator with flexible intermediate links," *Journal of Intelligent and Robotic Systems*, vol. 50, pp. 323–340, July 2007.

[19] S. Briot and W. Khalil, "Recursive symbolic calculation of the dynamic model of flexible parallel robots," in *2013 IEEE International Conference on Robotics and Automation*, pp. 5433–5438, IEEE, May 2013.

[20] S. Lilge, K. Nuelle, J. A. Childs, K. Wen, D. C. Rucker, and J. Burgner-Kahrs, "Parallel-continuum robots: A survey," *IEEE Transactions on Robotics*, pp. 1–20, 2024.

[21] F. Boyer, V. Lebastard, F. Candelier, and F. Renda, "Dynamics of continuum and soft robots: A strain parameterization based approach," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 847–863, 2021.

[22] J. C. Simo, J. E. Marsden, and P. Krishnaprasad, "The hamiltonian structure of nonlinear elasticity: the material and convective representations of solids, rods, and

plates," *Archive for Rational Mechanics and Analysis*, vol. 104, no. 2, pp. 125–183, 1988.

[23] J. Fish and T. Belytschko, *A first course in finite elements*, vol. 1. Wiley New York, 2007.

[24] E. Cosserat and F. Cosserat, *Theorie des corps dédormables.* A. Hermann et fils, 1909.

[25] P. E. Dupont, J. Lock, B. Itkowitz, and E. Butler, "Design and control of concentric-tube robots," *IEEE Trans. Robot.*, vol. 26, pp. 209–225, apr 2010.

[26] D. C. Rucker, B. A. Jones, and R. J. W. III, "A geometrically exact model for externally loaded concentric-tube continuum robots," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 769–780, 2010.

[27] D. C. Rucker and R. J. Webster, "Computing jacobians and compliance matrices for externally loaded continuum robots," *2011 IEEE International Conference on Robotics and Automation Shanghai International Conference Center May 9-13, 2011, Shanghai, China*, pp. 945–950, 2011.

[28] F. Renda, M. Giorelli, M. Calisti, M. Cianchetti, and C. Laschi, "Dynamic model of a multibending soft robot arm driven by cables," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1109–1122, 2014.

[29] A. D. Marchese and D. Rus, "Design, kinematics, and control of a soft spatial fluidic elastomer manipulator," *The International Journal of Robotics Research*, vol. 35, pp. 840–869, Oct. 2015.

[30] D. Trivedi, A. Lotfi, and C. Rahn, "Geometrically exact models for soft robotic manipulators," *IEEE Transactions on Robotics*, vol. 24, pp. 773–780, Aug. 2008.

[31] F. Boyer, M. Porez, and W. Khalil, "Macro-continuous computed torque algorithm for a three-dimensional eel-like robot," *IEEE Transactions on Robotics*, vol. 22, pp. 763–775, Aug. 2006.

[32] F. Boyer, V. Lebastard, F. Candelier, and F. Renda, "Extended hamilton's principle applied to geometrically exact kirchhoff sliding rods," *Journal of Sound and Vibration*, vol. 516, p. 116511, 2022.

[33] F. Boyer and F. Renda, "Poincare's equations for cosserat media: Application to shells," *Journal of Nonlinear Science*, vol. 27, pp. 1–44, July 2016.

[34] M. Tummers, V. Lebastard, F. Boyer, J. Troccaz, B. Rosa, and M. T. Chikhaoui, "Cosserat rod modeling of continuum robots from newtonian and lagrangian perspectives," *IEEE Transactions on Robotics*, vol. 39, pp. 2360–2378, June 2023.

[35] J. Simo, "A finite strain beam formulation. the three-dimensional dynamic problem. part i," *Computer Methods in Applied Mechanics and Engineering*, vol. 49, pp. 55–70, May 1985.

[36] P. Rao, Q. Peyron, S. Lilge, and J. Burgner-Kahrs, "How to model tendon-driven continuum robots and benchmark modelling performance," *Frontiers in Robotics and AI*, vol. 7, Feb. 2021.

[37] S. Lilge, T. D. Barfoot, and J. Burgner-Kahrs, "Continuum robot state estimation using gaussian process regression on se(3)," *The International Journal of Robotics Research*, vol. 41, pp. 1099–1120, Oct. 2022.

[38] D. C. Rucker and R. J. Webster III, "Statics and dynamics of continuum robots with general tendon routing and external loading," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1033–1044, 2011.

[39] J. Till, V. Aloi, and C. Rucker, "Real-time dynamics of soft and continuum robots based on cosserat rod models," *The International Journal of Robotics Research*, vol. 38, pp. 723–746, apr 2019.

[40] F. Boyer, V. Lebastard, F. Candelier, F. Renda, and M. Alamir, "Statics and dynamics of continuum robots based on cosserat rods and optimal control theories," *IEEE Transactions on Robotics*, vol. 39, pp. 1544–1562, apr 2023.

[41] R. Holsapple, R. Iyer, and D. Doman, "A modified simple shooting method for solving two-point boundary-value problems," *NASA STI/Recon Technical Report N*, vol. 6, pp. 6_2783– 6_2790, 02 2003.

[42] F. Boyer, G. De Nayer, A. Leroyer, and M. Visonneau, "Geometrically exact kirchhoff beam theory: Application to cable dynamics," *Journal of Computational and Nonlinear Dynamics*, vol. 6, Apr. 2011.

[43] J. Simo and L. Vu-Quoc, "A three-dimensional finite-strain rod model. part ii: Computational aspects," *Computer Methods in Applied Mechanics and Engineering*, vol. 58, no. 1, pp. 79–116, 1986.

[44] J. Simo and L. Vu-Quoc, "On the dynamics in space of rods undergoing large motions — a geometrically exact approach," *Computer Methods in Applied Mechanics and Engineering*, vol. 66, pp. 125–161, feb 1988.

[45] J. Simo and D. Fox, "On a stress resultant geometrically exact shell model. part i: Formulation and optimal parametrization," *Computer Methods in Applied Mechanics and Engineering*, vol. 72, pp. 267–304, Mar. 1989.

[46] J. Simo, D. Fox, and M. Rifai, "On a stress resultant geometrically exact shell model. part ii: The linear theory; computational aspects," *Computer Methods in Applied Mechanics and Engineering*, vol. 73, pp. 53–92, Apr. 1989.

[47] J. C. Simo, M. S. Rifai, and D. D. Fox, "On a stress resultant geometrically exact shell model. part vi: Conserving algorithms for non-linear dynamics," *International Journal for Numerical Methods in Engineering*, vol. 34, pp. 117–164, Mar. 1992.

[48] A. Cardona and M. Geradin, "A beam finite element non-linear theory with finite rotations," *International Journal for Numerical Methods in Engineering*, vol. 26, pp. 2403–2438, Nov. 1988.

[49] A. Cardona, M. Geradin, and D. Doan, "Rigid and flexible joint modelling in multibody dynamics using finite elements," *Computer Methods in Applied Mechanics and Engineering*, vol. 89, pp. 395–418, aug 1991.

[50] V. Sonneville, A. Cardona, and O. Bruls, "Geometrically exact beam finite element formulated on the special euclidean group se(3)," *Comput. Methods Appl. Mech. Engrg.*, vol. 268, pp. 451–474, 2014.

[51] A. Ibrahimbegović and M. A. Mikdad, "Finite rotations in dynamics of beams and implicit time-stepping schemes," *International Journal for Numerical Methods in Engineering*, vol. 41, no. 5, pp. 781–814, 1998.

[52] M. A. Crisfield and G. Jelenić;, "Objectivity of strain measures in the geometrically exact three-dimensional beam theory and its finite-element implementation," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 455, pp. 1125–1147, Mar. 1999.

[53] F. Boyer and D. Primault, "Finite element of slender beams in finite transformations: A geometrically exact approach„" *International Journal for Numerical Methods in Engineering*, vol. 59, no. 5, pp. 669–702, 2004.

[54] H.-S. Chang, U. Halder, E. Gribkova, A. Tekinalp, N. Naughton, M. Gazzola, and P. G. Mehta, "Controlling a cyberoctopus soft arm with muscle-like actuation," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 1383–1390, IEEE, Dec. 2021.

[55] X. Zhang, F. K. Chan, T. Parthasarathy, and M. Gazzola, "Modeling and simulation of complex dynamic musculoskeletal architectures," *Nature Communications*, vol. 10, Oct. 2019.

[56] M. Bergou, B. Audoly, E. Vouga, M. Wardetzky, and E. Grinspun, "Discrete viscous threads," *ACM Transactions on Graphics*, vol. 29, pp. 1–10, July 2010.

[57] B. Audoly, N. Clauvelin, P.-T. Brun, M. Bergou, E. Grinspun, and M. Wardetzky, "A discrete geometric approach for simulating the dynamics of thin viscous threads," *Journal of Computational Physics*, vol. 253, pp. 18–49, Nov. 2013.

[58] N. N. Goldberg, X. Huang, C. Majidi, A. Novelia, O. M. O'Reilly, D. A. Paley, and W. L. Scott, "On planar discrete elastic rod models for the locomotion of soft robots," *Soft Robotics*, vol. 6, pp. 595–610, Oct. 2019.

[59] W. Huang, X. Huang, C. Majidi, and M. K. Jawed, "Dynamic simulation of articulated soft robots," *Nature Communications*, vol. 11, May 2020.

[60] M. Gazzola, L. H. Dudte, A. G. McCormick, and L. Mahadevan, "Forward and inverse problems in the mechanics of soft filaments," *Royal Society Open Science*, vol. 5, no. 171628, 2017.

[61] M. Gazzola, "Pyelastica," *https://www.cosseratrods.org/*, vol. 32, no. 6, pp. 1371–1386, 1989.

[62] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola, "Elastica: A compliant mechanics environment for soft robotic control," *IEEE Robotics and Automation Letters*, vol. 6, pp. 3389–3396, Apr. 2021.

[63] E. Grinspun and A. Secord, "Discret differential geometry: An applied introduction," pp. 1–4, 2006.

[64] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun, "Discrete elastic rods," *ACM transactions on graphics (SIGGRAPH)*, pp. 63:1–63:12, 2008.

[65] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun, "Efficient simulation of inextensible cloth," in *ACM SIGGRAPH 2007 papers*, SIGGRAPH07, p. 49, ACM, July 2007.

[66] E. Tatlicioglu, I. D. Walker, and D. M. Dawson, "New dynamic models for planar extensible continuum robot manipulators," pp. 1485–1490, 2007.

[67] H. Mochiyama and T. Suzuki, "Dynamical modelling of a hyper-flexible manipulator," 2002.

[68] I. Robert J. Webster and B. A. Jones, "Design and kinematic modeling of constant curvature continuum robots: A review," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1661–1683, 2010.

[69] M. W. Hannan and I. D. Walker, "Kinematics and the implementation of an elephant's trunk manipulator and other continuum style robots," *Journal of Robotic Systems*, vol. 20, pp. 45–63, Feb. 2003.

[70] E. Tatlicioglu, I. D. Walker, and D. M. Dawson, "Dynamic modelling for planar extensible continuum robot manipulators," pp. 1357–1362, 2007.

[71] I. S. Godage, G. A. Medrano-Cerda, D. T. Branson, E. Guglielmino, and D. G. Caldwell, "Dynamics for variable length multisection continuum arms," *The International Journal of Robotics Research*, vol. 35, no. 6, pp. 695–722, 2016.

[72] V. Falkenhahn, T. Mahl, A. Hildebrandt, R. Neumann, and O. Sawodny, "Dynamic modeling of bellows-actuated continuum robots using the euler–lagrange formalism," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1483–1496, 2015.

[73] C. Della Santina, A. Bicchi, and D. Rus, "On an improved state parametrization for soft robots with piecewise constant curvature and its use in model based control," *IEEE Robotics and Automation Letters*, vol. 5, pp. 1001–1008, Apr. 2020.

[74] F. Renda, V. Cacucciolo, J. Dias, and L. Seneviratne, "Discrete cosserat approach for soft robot dynamics: A new piece-wise constant strain model with torsion and shears," pp. 5495–5502, 2016.

[75] F. Renda, F. Boyer, J. Dias, and L. Seneviratne, "Discrete cosserat approach for multisection soft manipulator dynamics," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1518–1533, 2018.

[76] M. Pourafzal, H. A. Talebi, and K. Rabenorosoa, "Piecewise constant strain kinematic model of externally loaded concentric tube robots," *Mechatronics*, vol. 74, p. 102502, Apr. 2021.

[77] H. Li, L. Xun, and G. Zheng, "Piecewise linear strain cosserat model for soft slender manipulator," *IEEE Transactions on Robotics*, vol. 39, pp. 2342–2359, June 2023.

[78] A. T. Mathew, I. B. Hmida, C. Armanini, F. Boyer, and F. Renda, "Sorosim: A matlab toolbox for hybrid rigid–soft robots based on the geometric variable-strain approach," *IEEE Robotics & Automation Magazine*, vol. 30, pp. 106–122, Sept. 2022.

[79] J. D. Greer, T. K. Morimoto, A. M. Okamura, and E. W. Hawkes, "Series pneumatic artificial muscles (spams) and application to a soft continuum robot," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5503–5510, IEEE, May 2017.

[80] M. E. Giannaccini, C. Xiang, A. Atyabi, T. Theodoridis, S. Nefti-Meziani, and S. Davis, "Novel design of a soft lightweight pneumatic continuum robot arm with decoupled variable stiffness and positioning," *Soft Robotics*, vol. 5, pp. 54–70, Feb. 2018.

[81] C. B. Black, J. Till, and D. C. Rucker, "Parallel continuum robots: Modeling, analysis, and actuation-based force sensing," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 29–47, 2018.

[82] S. Briot and F. Boyer, "A geometrically exact assumed strain modes approach for the geometrico- and kinemato-static modelings of continuum parallel robots," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1527–1543, 2023.

[83] F. Zaccaria, E. Ida, S. Briot, and M. Carricato, "Workspace computation of planar continuum parallel robots," *IEEE Robotics and Automation Letters*, vol. 7, pp. 2700–2707, Apr. 2022.

[84] F. Zaccaria, E. Idá, and S. Briot, "A boundary computation algorithm for the workspace evaluation of continuum parallel robots," *Journal of Mechanisms and Robotics*, vol. 16, June 2023.

[85] F. Zaccaria, E. Idà, and S. Briot, "Design and experimental equilibrium stability assessment of a rfrfr continuum parallel robot," *Mechatronics*, vol. 95, p. 103064, Nov. 2023.

[86] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.

[87] W. Khalil and E. Dombre, *Modeling identification and control of robots*. CRC Press, 2002.

[88] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, *On-Line Computational Scheme for Mechanical Manipulators*, vol. 102. ASME International, jun 1980.

[89] F. C. Park, J. E. Bobrow, and S. R. Ploen, "A lie group formulation of robot dynamics," *The International Journal of Robotics Research*, vol. 14, pp. 609–618, Dec. 1995.

[90] J. Kim, "Lie group formulation of articulated rigid body dynamics," tech. rep., School of Computer Science, Carnegie Mellon University.

[91] D. Pogorelov, "Differential–algebraic equations in multibody system modeling," *Numerical Algorithms*, vol. 19, pp. 183–194, Dec. 1998.

[92] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 1, pp. 1–16, June 1972.

[93] C. Armanini, I. Hussain, M. Z. Iqbal, D. Gan, D. Prattichizzo, and F. Renda, "Discrete cosserat approach for closed-chain soft robots: Application to the fin-ray finger," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2083–2098, 2021.

[94] A. Cardona and M. Géradin, "Numerical integration of second order differential—algebraic systems in flexible mechanism dynamics," *Computer-Aided Analysis of Rigid and Flexible Mechanical Systems*, pp. 501–529, 1994.

[95] L. N. Trefethen, *Spectral methods in MATLAB*. SIAM, 2000.

[96] F. Boyer, M. Porez, and J. Mauny, "Reduced dynamics of the non-holonomic whipple bicycle," *Journal of Nonlinear Science*, vol. 28, pp. 943–983, 2018.

[97] T. J. R. Hughes, *Finite Element Method-Linear Static and Dynamic Finite Element Analysis*. Englewood Cliffs, prentice-hall ed., 1987.

[98] A. Cardona and M. Geradin, "Time integration of the equations of motion in mechanism analysis," *Computers & Structures*, vol. 33, no. 3, pp. 801–820, 1989.

[99] F. Renda, C. Armanini, V. Lebastard, F. Candelier, and F. Boyer, "A geometric variable-strain approach for static modeling of soft manipulators with tendon and fluidic actuation," *IEEE Robotics and Automation Letters*, vol. 5, pp. 4006–4013, July 2020.

[100] A. L. Orekhov and N. Simaan, "Solving cosserat rod models via collocation and the magnus expansion," 2020.

[101] K. Subbaraj and M. Dokainish, "A survey of direct time-integration methods in computational structural dynamics—II. implicit methods," *Computers & Structures*, vol. 32, pp. 1387–1401, jan 1989.

[102] F. Boyer, M. Porez, A. Leroyer, and M. Visonneau, "Fast dynamics of an eel-like robot—comparisons with navier–stokes simulations," *IEEE Transactions on Robotics*, vol. 24, pp. 1274–1288, Dec. 2008.

[103] W. Khalil, "Dynamic modeling of robots using recursive newton-euler techniques," 2010.

[104] K. Nuelle, T. Sterneck, S. Lilge, D. Xiong, J. Burgner-Kahrs, and T. Ortmaier, "Modeling, calibration, and evaluation of a tendon-actuated planar parallel continuum robot," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5811–5818, 2020.

[105] N. M. Newmark, "A method of computation for structural dynamics," *Journal of the Engineering Mechanics Division*, vol. 85, pp. 67–94, jul 1959.

[106] H. M. Hilber, T. J. R. Hughes, and R. L. Taylor, "Improved numerical dissipation for time integration algorithms in structural dynamics," *Earthquake Engineering & Structural Dynamics*, vol. 5, pp. 283–292, jul 1977.

[107] W. L. Wood, M. Bossak, and O. C. Zienkiewicz, "An alpha modification of newmark's method," *International Journal for Numerical Methods in Engineering*, vol. 15, pp. 1562–1566, Oct. 1980.

[108] J. Chung and G. M. Hulbert, "A time integration algorithm for structural dynamics with improved numerical dissipation: The generalized-$\alpha$ method," *Journal of Applied Mechanics*, vol. 60, pp. 371–375, jun 1993.

[109] O. Brüls, A. Cardona, and M. Arnold, "Lie group generalized-$\alpha$ time integration of constrained flexible multibody systems," *Mechanism and Machine Theory*, vol. 48, pp. 121–137, feb 2012.

[110] R. M. Grassmann, C. Shentu, T. Hamoda, P. T. Dewi, and J. Burgner-Kahrs, "Open continuum robotics–one actuation module to create them all," *Frontiers in Robotics and AI*, vol. 11, Jan. 2024.

[111] J. Allard, S. Cotin, F. Faure, P. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni, "Sofa–an open source framework for medical simulation.," vol. 125, pp. 13–18, 2007.

[112] W. Khalil and E. Dombre, *Modeling, Identification & Control of Robots*. Electronics & Electrical, Kogan Page Science, 2002.

[113] P. Wegner, "Concepts and paradigms of object-oriented programming," *ACM SIGPLAN OOPS Messenger*, vol. 1, pp. 7–87, Aug. 1990.

[114] Object Management Group, "Unified modeling language." https://www.omg.org/spec/UML/2.5.1/About-UML, Dec. 2017. UML 2.5.1 Specification.

[115] M. Arnold and O. Brüls, "Convergence of the generalized-$\alpha$ scheme for constrained mechanical systems," *Multibody System Dynamics*, vol. 18, pp. 185–202, jul 2007.

[116] Boost, "Boost C++ Libraries." http://www.boost.org/, 2015. Last accessed 2015-06-30.

[117] S. Blanes, F. Casas, J. Oteo, and J. Ros, "The magnus expansion and some of its applications," *Physics Reports*, vol. 470, pp. 151–238, jan 2009.

[118] S. Blanes and E. Ponsoda, "Time-averaging and exponential integrators for non-homogeneous linear IVPs and BVPs," *Applied Numerical Mathematics*, vol. 62, pp. 875–894, aug 2012.

[119] Google, "Google benchmark." https://github.com/google/benchmark, 2024.

[120] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[121] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[122] C. L. Bottasso, D. Dopico, and L. Trainelli, "On the optimal scaling of index three daes in multibody dynamics," *Multibody System Dynamics*, vol. 19, pp. 3–20, 2008.

[123] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, 2009.

[124] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, May 2022.

[125] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

COLLEGE SCIENCES
DOCTORAL DE L'INGENIERIE
PAYS DE LA LOIRE ET DES SYSTEMES

CENTRALE NANTES

**Titre :** Modélisation Dynamique des robots paralleles continus

**Mots clés :** Cosserat , Robots parallèles continus, Algorithmes de Newton-Euler Récursifs

**Résumé :** Les robots parallèles continus (CPR) sont des manipulateurs composés de plusieurs corps minces et déformables, connectés en parallèle. Leur flexibilité permet de réduire les coûts de maintenance tout en diminuant les risques lors d'interactions avec les humains. Les CPR subissent de grandes déformations pour réaliser les mouvements souhaités. Alors que la majorité des études se sont concentrées sur la modélisation statique, peu de travaux ont abordé leur comportement dynamique.
Cette thèse se concentre sur la modélisation dynamique des CPR, avec pour objectif de développer une approche générique capable de traiter différents types de CPR. Un algorithme basé sur un schéma d'intégration temporelle implicite a été proposé pour caractériser la dynamique des systèmes hybrides rigides-déformables. Il s'applique à des manipulateurs constitués de corps minces et déformables, avec ou sans architectures parallèles, ainsi qu'à des locomoteurs, où la base est un corps flottant. L'algorithme adapte les passes standard de Newton-Euler utilisées en robotique rigide pour traiter les corps déformables et les architectures parallèles.
L'approche a été spécialisée pour les CPR et validée expérimentalement à travers deux expériences : la première a servi pour valider le calcul de la réponse cinématique du robot, tandis que la seconde a comparé le comportement simulé du robot lors du passage par une singularité. Enfin, l'implémentation d'un simulateur dynamique générique a été détaillée, en mettant en avant ses composants et leurs interfaces. Ce simulateur se présente comme une alternative stable et polyvalente aux autres outils disponibles dans la littérature.

**Title :** Dynamics modelling of continuum parallel robots

**Keywords :** Cosserat, Continuum parallel robots, Recursive Newton-Euler Algorithms

**Abstract :** Continuum Parallel Robots (CPRs) are manipulators consisting of multiple slender, deformable bodies connected in parallel. Their flexibility offers lower maintenance costs and reduces risks in human-robot interactions. CPRs undergo large deformations to achieve desired motions. While most studies have focused on statics modeling, few have addressed their dynamic behavior.
This thesis focuses on the dynamics modeling of CPRs, aiming to develop a generic approach capable of handling various CPR types. An algorithm based on an implicit time integration scheme was proposed to characterize the dynamics of hybrid rigid-deformable systems. This framework includes manipulators with slender deformable bodies, with or without parallel architectures, and locomotors, where the base is a floating body. The algorithm adapts the standard Newton-Euler passes used in rigid robotics to handle deformable bodies and parallel architectures. The approach was specialized for CPRs and validated experimentally through two experiments: the first validated the computed kinematics, while the second compared the simulated behavior of the robot as it crossed a singularity. Finally, the implementation of a generic dynamics simulator was detailed, highlighting its components and interfaces. This approach offers a stable and versatile alternative to other tools available in the literature.