

UNIVERSITY OF WEST BOHEMIA

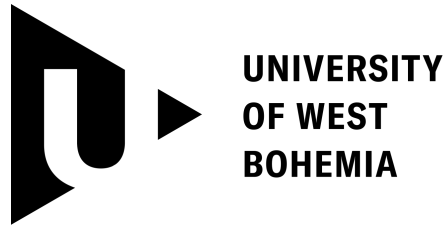
Faculty of Applied Sciences

Department of Mechanics

**DIPLOMA THESIS**

Pilsen, 2022

Martin Hrabačka



Faculty of Applied Sciences  
Department of Mechanics

# **Design and Dynamic Analysis of Active Tensegrity Structures**

Diploma thesis

Author: Martin Hrabačka  
Supervisor: Ing. Radek Bulín, Ph.D.

Pilsen, 2022

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd  
Akademický rok: 2021/2022

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Bc. Martin HRABAČKA**  
Osobní číslo: **A20N0015P**  
Studijní program: **N0715A270006 Aplikovaná mechanika**  
Specializace: **Dynamika konstrukcí a mechatronika**  
Téma práce: **Návrh a dynamická analýza aktivních tensegritických struktur**  
Zadávací katedra: **Katedra mechaniky**

## Zásady pro vypracování

1. Zpracování návrhu tensegrit s využitím form-finding metod.
2. Základní analýza stability a mechanických vlastností.
3. Začlenění aktivních prvků do tensegritické struktury.
4. Tvorba metodiky pro automatizované vytváření dynamických modelů.
5. Dynamická analýza aktivního modelu tensegritické struktury.



Rozsah diplomové práce: **30 – 50 stran A4 včetně příloh**  
Rozsah grafických prací:  
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

1. ZHANG, J.Y., OHSAKI, M.: Tensegrity Structures: Form, Stability, and Symmetry. Springer Japan, 2015. ISBN 978-4-431-54812-6.
2. SKELTON, R., de OLIVEIRA, M.: Tensegrity Systems. Springer-Verlag, 2009. ISBN 978-0-387-74241-0.
3. WROLDSEN, A.S.: Modelling and Control of Tensegrity Structures. Disertační práce, Norwegian University of Science and Technology, 2007.
4. SHABANA, A.A.: Dynamics of Multibody Systems. Fifth edition, Cambridge University Press, 2020. ISBN 978-1-108-48564-7.

Vedoucí diplomové práce: **Ing. Radek Bulín, Ph.D.**  
Nové technologie pro informační společnost

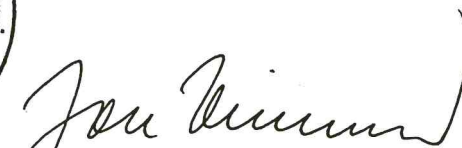
Konzultant diplomové práce: **Doc. Ing. Michal Hajžman, Ph.D.**  
Katedra mechaniky

Datum zadání diplomové práce: **18. října 2021**  
Termín odevzdání diplomové práce: **31. května 2022**



---

**Doc. Ing. Miloš Železný, Ph.D.**  
děkan



---

**Doc. Ing. Jan Vimmr, Ph.D.**  
vedoucí katedry

# Declaration

I hereby declare that this diploma thesis is completely my own work and I used only the cited sources.

Pilsen, 31th May 2022

Martin Hrabáčka

# Acknowledgement

I would like to thank Ing. Radek Bulín, Ph.D. for his exemplary guidance, expert advice, factual comments and patience.

I would also like to give special thanks to my partner and family for their unwavering support during my studies.

Martin Hračka

# Abstract

## Design and Dynamic Analysis of Active Tensegrity Structures

This diploma thesis first deals with the design of a general tensegrity structure using form-finding methods. Within path-planning, a specific actuation in form of adjustments of cables' rest lengths is assigned to the resulting structure so that it follows a prescribed complex trajectory. Since the path-planning problem is treated as static, an  $H_2$  controller is designed with the main objective of suppressing possible structural vibrations. With regard to the need of testing of dynamical properties and simulating motion of the structure, a methodology for automatic creation of tensegrity computational models in Simscape software is developed in the MATLAB scripting environment.

**Key words:** dynamics, computational modelling, active tensegrity structure, form-finding, path-planning,  $H_2$  controller

# Abstrakt

## Návrh a dynamická analýza aktivních tensegritických struktur

Tato diplomová práce se nejprve zabývá návrhem obecné tensegritické struktury pomocí form-finding metod. Výsledné strukturu je v rámci úlohy path-planningu přiřazena specifická aktuace ve formě změn volných délek lan tak, aby struktura opsala předepsanou komplexní trajektorii. Jelikož je k úloze path-planningu přistupováno jako ke statickému problému, pro tensegritickou strukturu je navíc navržen  $H_2$  regulátor s hlavním úkolem potlačit případné strukturální vibrace. S ohledem na nutnost testování dynamických vlastností a simulace pohybu struktury je v programovacím prostředí MATLAB vyvinuta metodika pro automatické sestavení výpočtového modelu tensegrity v programu Simscape.

**Klíčová slova:** dynamika, výpočtové modelování, aktivní tensegritická struktura, form-finding, path-planning,  $H_2$  regulátor

# Contents

<b>Assignment</b>	<b>3</b>
<b>Declaration</b>	<b>4</b>
<b>Acknowledgement</b>	<b>5</b>
<b>Abstract</b>	<b>6</b>
<b>1. Introduction</b>	<b>9</b>
1.1. Thesis content and goals . . . . .	11
<b>2. Form-finding</b>	<b>12</b>
2.1. Adaptive force density method . . . . .	13
2.1.1. Geometry matrix . . . . .	14
2.1.2. Force density matrix . . . . .	14
2.1.3. Super-stability . . . . .	16
2.1.4. First design stage . . . . .	17
2.1.5. Second design stage . . . . .	19
2.1.6. Examples . . . . .	20
2.2. Dynamic relaxation method . . . . .	21
2.2.1. Formulation of the DRM . . . . .	22
2.2.2. Ensuring numerical stability . . . . .	25
2.2.3. Reinitialization at kinetic energy peaks . . . . .	25
2.2.4. The DRM algorithm . . . . .	27
2.2.5. Examples . . . . .	29
2.3. Comparison of the AFDM and the DRM . . . . .	31
<b>3. Dynamics of tensegrity systems</b>	<b>32</b>
3.1. Multibody system dynamics . . . . .	32
3.2. Computational model of tensegrity structure . . . . .	34
3.2.1. Models of strut and cable . . . . .	35
3.2.2. Tensegrity model building . . . . .	37
3.2.3. Automatic model generation . . . . .	43
3.3. Modal analysis of computational model . . . . .	46
3.3.1. The eigenvalue problem . . . . .	46
3.3.2. Eigenfrequencies of the computational model . . . . .	47
3.3.3. Example . . . . .	48



<b>4. Active tensegrity structures</b>	<b>49</b>
4.1. Path-planning . . . . .	49
4.1.1. Optimization problem . . . . .	50
4.1.2. Algorithmization . . . . .	53
4.1.3. Examples . . . . .	54
4.2. Shape control . . . . .	62
4.2.1. Linearization . . . . .	62
4.2.2. Augmented plant . . . . .	65
4.2.3. $H_2$ controller . . . . .	65
4.2.4. Examples . . . . .	66
<b>5. Conclusion</b>	<b>70</b>
<b>Bibliography</b>	<b>71</b>
<b>A. Appendices</b>	<b>73</b>
A.1. Parameter table . . . . .	73
A.2. MATLAB function for automatic generation of Simscape tensegrity models	74

# 1. Introduction

The word *tensegrity* is a conjunction of two words, *tensile* and *integrity*, coined by B. Fuller [9]. In accordance with [14], it refers to the integrity of a stable self-equilibrated system that contains a discontinuous set of components in compression inside a network of components in tension. Fuller described tensegrities in quite an interesting way – small islands of compression in a sea of tension.



Figure 1.1.: Tensegrity in art: Kenneth Snelson's *Triple Crown* [12].

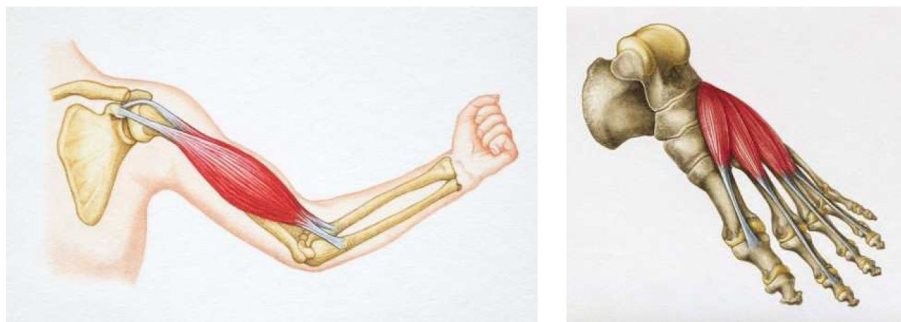


Figure 1.2.: Tensegrity in biology: human arm and foot [21].

Tensegrity structures were originally created and explored by artists fascinated by the magnificence of their appearance. Artistic people were rapidly followed by architects applying this new approach to structures such as geodesic domes [24]. A few decades after Fuller's invention in 1962, tensegrities started their expansion into other technical

fields, for instance space engineering with the development of a deployable antenna. In mentioned engineering areas, researchers have already revealed superior static features of tensegrities, e.g. deployability, very good strength to weight ratio, or energy efficiency, compared to traditional approaches [24]. However, tensegrity principles can be also applied to other areas including biology (bones and tendons in human body, cells, etc.). In Figure 1.1, there is an example of a tensegrity as an artistic work, and a biological example of tensegrity is depicted in Figure 1.2.

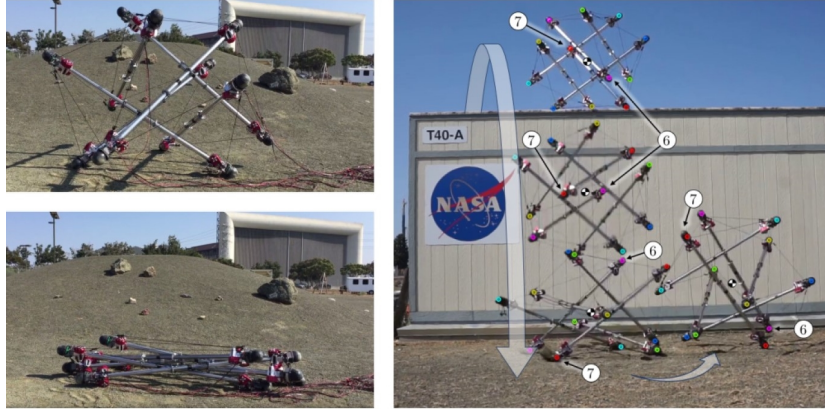


Figure 1.3.: Tensegrity in robotics: deployed structure, stowed structure, impact test [20].

Recently, there has been an increasing trend in publishing effort aiming on application of tensegrity structures in robotics, which is also the case of this work. Conventional robots are usually composed of rigid materials and structures bringing advantages of rigidity, high precision, and fast speed, but they also dispose of poor flexibility and adaptability. On the other hand, soft robots have excellent flexibility and adaptability, but the load capacity is limited and, moreover, there are still many difficulties in the design, manufacturing, sensing, modelling, and control [13]. Quite promising approach represents combining rigid and soft structures into tensegrities to gain advantages from both types and compensate their disadvantages. An example of a robotic tensegrity structure can be seen in Figure 1.3.

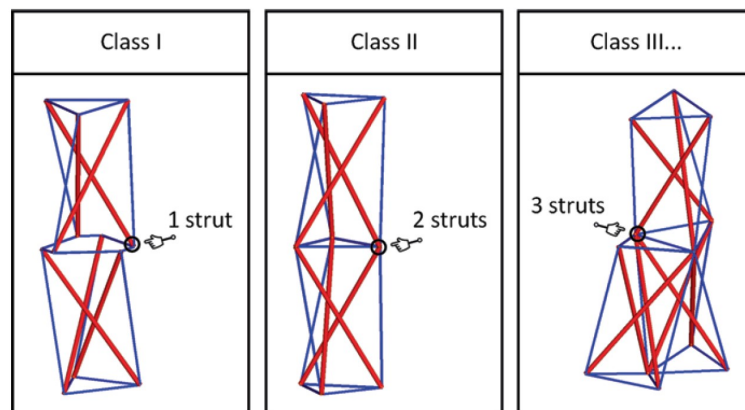


Figure 1.4.: Classification of a tensegrity structure [10].

There is a number of different body forms that can be utilized as basal members of tensegrity structures: bars, plates, or blocks as compressive parts, fibres, cables, nets, or membranes as tensile members. In this work, tensegrities are assumed to be composed of struts representing compressive members and tensile parts in form of deformable cables. Tensegrities of this kind can be sorted into classes determining maximal number of struts connected by one joint, e.g., structures of class 1 are the basic ones – they are not allowed to contain any joints connecting some struts together (an example of a class-1 tensegrity is the artwork in Figure 1.1). Classification of tensegrity structures is visualized in Figure 1.4.

### 1.1. Thesis content and goals

The main goal of this thesis is to present a comprehensive methodology for designing tensegrity robots. During the design process, several different problems are solved. Form-finding represents a problem of searching for a specific stable configuration associated with particular prestresses in all members. Two different form-finding methods, the Adaptive force density method and the Dynamic relaxation method, described in chapter 2 are utilized.

Dynamics of tensegrity systems is discussed in chapter 3. In its beginning, theoretical background of multibody system dynamics modelled by Lagrange's equations is briefly summarized. Main emphasis of the chapter is placed on the development of a methodology that enables building a computational model of tensegrity structure in Simscape software. Moreover, a robust software for automatic creation of Simscape models is developed in order to eliminate time spent on building the model manually. Modal analysis of created computational model is then performed.

Because this work is focused on tensegrity robots, robotic manipulators to be more specific, it is necessary to incorporate active members into the structure and design the actuation so that the tensegrity performs required operation. A standard task of robotic manipulators is to move something from one place to another while following a desired trajectory. In case of tensegrity structures, the problem of designing the needed actuation for following a specified trajectory is called path-planning. This problem is solved by an optimization process in chapter 4. Subsequently, the structure is enhanced by an  $H_2$  controller designed to reduce position errors mainly in form of vibrations.

#### Thesis goals

The content of the work complies with goals of the thesis summarized in following points:

- Processing the design of tensegrity structures using form-finding methods.
- Basic analysis of stability and mechanical properties.
- Incorporation of active parts into tensegrity structures.
- Development of a methodology for automatic creation of dynamic models.
- Dynamic analysis of active tensegrity models.

## 2. Form-finding

Form-finding represents a common design problem for tension structures (e.g. membrane structures, cable-nets) when configuration associated with prestresses in state of equilibrium is searched for [29]. Regarding tensegrity structures, solving this problem is difficult because there is high interdependency between configuration and prestresses so they cannot be determined separately. Moreover, tensegrity structures maintain their stability without any support.

Nowadays, there are various numerical approaches how to solve the problem of form-finding. In this work, two different methods are described and used, each of them advantageously applicable at a different stage of active tensegrity structure development. Their advantages are summarized at the end of this chapter in section 2.3.

Before both methods are presented, some general points have to be clarified. A general passive 3-D tensegrity structure with  $n$  nodes and  $m$  members is fully determined when, in addition to member and material properties (e.g. member's cross section or Young's modulus), its topology, nodal positions and prestresses in each member are specified. As already mentioned, form-finding methods should provide the answer about nodal positions and members' prestresses. This thesis is not focused on searching for a suitable topology or members' and material properties, it assumes that they are known apriori.

It is suitable to point out that both form-finding methods are presented as solutions for 3-D structures, but they can be used for 2-D structures too. (Methods should be also applicable to other dimensions, but it does not make much sense from the mechanical point of view.)

Topology of a structure can be expressed by the connectivity matrix<sup>1</sup>  $\mathbf{C} \in \mathbb{R}^{m \times n}$  representing all  $m$  connections between  $n$  nodes. It is supposed that the  $k$ -th member ( $k = 1, 2, \dots, m$ ) begins in the node  $i^{start}$  and ends in the node  $i^{end}$  ( $i^{start}, i^{end} = 1, 2, \dots, n$ ). Then the element  $C_{k,i}$  ( $i = 1, 2, \dots, n$ ) of the matrix  $\mathbf{C}$  is defined according to [29] as

$$C_{k,i} = \begin{cases} \operatorname{sgn}(i^{end} - i), & i = i^{start}, \\ \operatorname{sgn}(i^{start} - i), & i = i^{end}, \\ 0, & \text{otherwise.} \end{cases} \quad (2.1)$$

Coordinates of all nodes can be summarized into the configuration vector  $\mathbf{X} \in \mathbb{R}^{3n}$

$$\mathbf{X}^T = [\mathbf{x}^T, \mathbf{y}^T, \mathbf{z}^T], \quad (2.2)$$

---

<sup>1</sup>Connectivity matrix gives the information only about existence of nodal connections. It does not distinguish between struts and cables.

where vectors  $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$  of nodal coordinates in particular direction are assembled as

$$\begin{aligned}\mathbf{x} &= [x_{1,1}, x_{2,1}, \dots, x_{n,1}]^T, \\ \mathbf{y} &= [x_{1,2}, x_{2,2}, \dots, x_{n,2}]^T, \\ \mathbf{z} &= [x_{1,3}, x_{2,3}, \dots, x_{n,3}]^T,\end{aligned}\tag{2.3}$$

where the coordinate of  $i$ -th node in direction  $d$  is denoted as  $x_{i,d}$  ( $d = 1, 2, 3$ ). Alternatively, spatial position of the  $i$ -th node can be expressed by the vector

$$\boldsymbol{\xi}_i = [x_{i,1}, x_{i,2}, x_{i,3}]^T \in \mathbb{R}^3.\tag{2.4}$$

Analogously to definitions (2.2) and (2.3), the complete vector  $\mathbf{F} \in \mathbb{R}^{3n}$  of external nodal load and vectors  $\mathbf{f}_x, \mathbf{f}_y, \mathbf{f}_z \in \mathbb{R}^n$  of external load in particular direction can be introduced with  $f_{i,d}$  denoting the external force acting in the  $i$ -th node in direction  $d$ .

Members' internal forces are expressed by the vector  $\mathbf{s} \in \mathbb{R}^m$  of internal forces as

$$\mathbf{s} = [s_1, s_2, \dots, s_m]^T,\tag{2.5}$$

where  $s_k$  is the internal force in the  $k$ -th member, and, analogously, the vector  $\mathbf{l} \in \mathbb{R}^m$  of member lengths and the vector  $\mathbf{l}^0 \in \mathbb{R}^m$  of member rest lengths are defined as

$$\mathbf{l} = [l_1, l_2, \dots, l_m]^T\tag{2.6}$$

and

$$\mathbf{l}^0 = [l_1^0, l_2^0, \dots, l_m^0]^T.\tag{2.7}$$

In the following text, also the diagonal form  $\mathbf{L} = \text{diag}(\mathbf{l}) \in \mathbb{R}^{m \times m}$  of the vector  $\mathbf{l}$  is used. It is defined as

$$\mathbf{L} = \begin{bmatrix} l_1 & 0 & 0 & \dots & 0 \\ 0 & l_2 & 0 & \dots & 0 \\ 0 & 0 & l_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & l_m \end{bmatrix}.\tag{2.8}$$

## 2.1. Adaptive force density method

The essence of the original force density method lies in the introduction of force density concept. This concept allows a transformation of non-linear self-equilibrium<sup>2</sup> equations into linear ones. Originally, this approach was used for form-finding of net structures by [17]. These structures have many things in common with tensegrity structures, therefore, usage of the same concept is offered as a solution to tensegrity form-finding problem.

---

<sup>2</sup>Self-equilibrium = structure is equilibrated only by prestresses in the members (no external forces are applied).

But there is also a difference – net structures are attached to fixed points but tensegrity structures are free-standing. Thus, a new strategy is developed in comparison to the original method. This work is based on the adaptive force density method (AFDM) presented by [29] which is further modified.

The AFDM presented in this work generates a specific tensegrity based on two user inputs: topology of the structure and elevation of selected structural nodes.

The method is divided into two main parts. In the first stage, feasible force density vector is searched for and the second stage finds the self-equilibrated configuration in form of the configuration vector. These two outputs fully determine the tensegrity and the AFDM assures that it is a stable structure satisfying the constraints on elevation of selected nodes.

### 2.1.1. Geometry matrix

The geometry matrix represents an instrument involved in ensuring the stability of the designed structure. This short section only defines the matrix according to [29], all important facts connected to this matrix can be found in the mentioned paper [29].

Using the connectivity matrix and coordinate vectors defined in (2.1) and (2.3), coordinate difference vectors  $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^m$  can be defined as

$$\begin{aligned}\mathbf{u} &= \mathbf{C}\mathbf{x}, \\ \mathbf{v} &= \mathbf{C}\mathbf{y}, \\ \mathbf{w} &= \mathbf{C}\mathbf{z}.\end{aligned}\tag{2.9}$$

The matrix  $\mathbf{U} = \text{diag}(\mathbf{u}) \in \mathbb{R}^{m \times m}$  represents a diagonal form of the vector  $\mathbf{u}$  introduced similarly to the diagonal form  $\mathbf{L}$  in (2.8). Matrices  $\mathbf{V}, \mathbf{W} \in \mathbb{R}^{m \times m}$  are defined analogously.

The geometry matrix  $\mathbf{G} \in \mathbb{R}^{m \times \frac{D^2+D}{2}}$  is defined as

$$\mathbf{G} = [\mathbf{U}\mathbf{u}, \mathbf{V}\mathbf{v}, \mathbf{W}\mathbf{w}, \mathbf{U}\mathbf{v}, \mathbf{U}\mathbf{w}, \mathbf{V}\mathbf{w}],\tag{2.10}$$

where  $D$  is the spatial dimension of the structure ( $D = 3$  in this text).

### 2.1.2. Force density matrix

The heart of the AFDM represents the usage of the force density matrix  $\mathbf{E}$ , and this section is dedicated to its definition.

First, the quantity of the force density in the  $k$ -th member is introduced according to [17] as

$$q_k = \frac{s_k}{l_k}.\tag{2.11}$$

Force densities of all members are summarized in the force density vector  $\mathbf{q} \in \mathbb{R}^m$  as

$$\mathbf{q} = [q_1, q_2, \dots, q_m]^T.\tag{2.12}$$

It can be also derived from  $\mathbf{s}$  and  $\mathbf{L}$  defined in (2.5) and (2.8) as

$$\mathbf{q} = \mathbf{L}^{-1}\mathbf{s}. \quad (2.13)$$

The matrix  $\mathbf{Q} = \text{diag}(\mathbf{q}) \in \mathbb{R}^{m \times m}$  represents a diagonal version of the force density vector.

If the tensegrity structure is in static equilibrium, the equilibrium equation

$$\mathbf{D}\mathbf{s} = \mathbf{F} \quad (2.14)$$

derived in [29] must be satisfied. The load vector  $\mathbf{F}$  was introduced in the beginning of chapter 2. The equilibrium matrix  $\mathbf{D} \in \mathbb{R}^{3n \times m}$  is composed of sub-matrices  $\mathbf{D}_x, \mathbf{D}_y, \mathbf{D}_z \in \mathbb{R}^{n \times m}$  for each direction as

$$\mathbf{D}^T = [\mathbf{D}_x^T, \mathbf{D}_y^T, \mathbf{D}_z^T]. \quad (2.15)$$

For the matrix  $\mathbf{D}_x$ , it applies that

$$\mathbf{D}_x = \mathbf{C}^T \mathbf{U} \mathbf{L}^{-1}, \quad (2.16)$$

and analogously for  $\mathbf{D}_y$  and  $\mathbf{D}_z$  according to [29].

Equilibrium equation in the direction  $x$  (a sub-system of (2.14)) is obviously

$$\mathbf{D}_x \mathbf{s} = \mathbf{f}_x. \quad (2.17)$$

Using equations (2.16) and (2.13), the left side of (2.17) can be written as

$$\mathbf{D}_x \mathbf{s} = \mathbf{C}^T \mathbf{U} \mathbf{L}^{-1} \mathbf{s} = \mathbf{C}^T \mathbf{U} \mathbf{q}. \quad (2.18)$$

Applying general relation  $\text{diag}(\mathbf{a}) \cdot \mathbf{b} = \text{diag}(\mathbf{b}) \cdot \mathbf{a}$  between two vectors and their diagonal forms, (2.18) can be further modified as

$$\mathbf{C}^T \mathbf{U} \mathbf{q} = \mathbf{C}^T \mathbf{Q} \mathbf{u} = \mathbf{C}^T \mathbf{Q} \mathbf{C} \mathbf{x}, \quad (2.19)$$

where the definition (2.9) of the coordinate vector is also used.

Finally, the force density matrix  $\mathbf{E} \in \mathbb{R}^{n \times n}$  is defined as

$$\mathbf{E} = \mathbf{C}^T \mathbf{Q} \mathbf{C}, \quad (2.20)$$

so the equilibrium equation in the direction  $x$  can be written in form of

$$\mathbf{E} \mathbf{x} = \mathbf{f}_x. \quad (2.21)$$

The whole system of equilibrium equations is therefore

$$\begin{bmatrix} \mathbf{E} & 0 & 0 \\ 0 & \mathbf{E} & 0 \\ 0 & 0 & \mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \mathbf{H} \mathbf{X} = \mathbf{F}. \quad (2.22)$$



In addition to the definition (2.20), there is also a direct definition<sup>3</sup> of  $\mathbf{E}$  using particular force densities  $q_i$  of members. The element  $E_{i,j}$  ( $i, j = 1, 2, \dots, n$ ) is defined according to [29] as

$$E_{i,j} = \begin{cases} \sum_{k \in M_i} q_k, & i = j, \\ -q_k, & i \neq j \wedge \text{nodes } i, j \text{ are connected by member } k, \\ 0, & \text{otherwise,} \end{cases} \quad (2.23)$$

where  $M_i$  denotes the set of all members connected to  $i$ -th node.

### 2.1.3. Super-stability

The whole method relies on sufficient conditions of so-called super-stability, the term introduced in [6]. The criterion of super-stability is a stricter type of the general stability criterion (super-stable structures are a subset of stable structures, see Figure 2.1). In this paper, a stable structure is defined as a structure with total potential energy reaching its local minimum. The super-stable structure is a structure that is always stable in the state of self-equilibrium irrespective of material properties as well as level of prestresses [29].

According to [29], if following conditions are fulfilled then the structure is super-stable.

1. The force density matrix  $\mathbf{E}$  is positive semi-definite.
2. The force density matrix  $\mathbf{E}$  has the rank deficiency  $D + 1$ .
3. Rank of the geometry matrix  $\mathbf{G}$  is  $\frac{1}{2}(D^2 + D)$ .

First condition represents the necessary condition for super-stability in terms of force densities. The second one is the non-degeneracy condition for free-standing prestressed structures and the third one assures that the geometric realization of the structure is non-degenerate. Meaning and derivation of these conditions can be found in [29].

When these conditions are fulfilled, it is clear that the structure is stable with no need of inspecting the tangential stiffness matrix which might be difficult (or impossible) to discover, especially at the beginning of tensegrity design process.

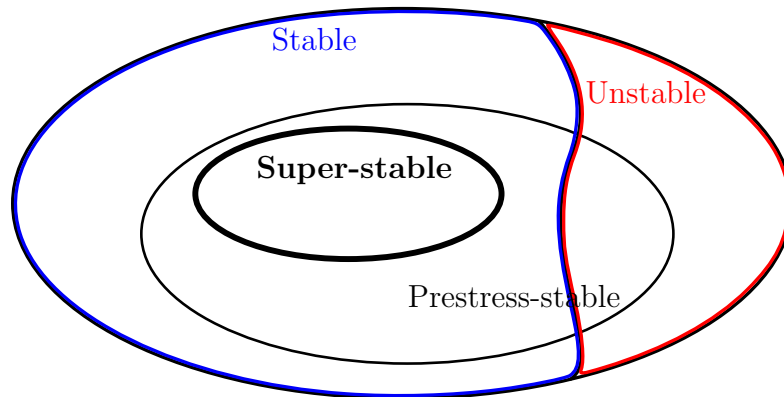


Figure 2.1.: Relationship among different criteria of stability.

<sup>3</sup>It is an analogous procedure to the stiffness matrix creation in case of discrete vibrating systems.

In Figure 2.1 inspired by [29], relationship among different stability criteria is visualized. Sets of stable and unstable structures are principally disjoint. The set of super-stable structures is a subset of stable structures which means that all super-stable sets are stable. Regarding pin-jointed structures, which the tensegrity structure truly is, there is another relevant stability criterion – prestress-stability. It assures stability only from the view of infinitesimal displacements from the self-equilibrium state and it does not imply stability as it can be seen in Figure 2.1. This work does not operate with prestress-stability.

#### 2.1.4. First design stage

As it was already mentioned, the first phase of the AFDM searches for feasible force densities. They can be discovered when first two conditions of super-stability are satisfied. These two conditions are possible to be merged in one requirement – force density matrix  $\mathbf{E}$  must have  $D + 1$  zero eigenvalues and other positive. The process described in the following text is proposed in [29] and [30].

If eigenvalue analysis of the matrix  $\mathbf{E}$  is performed, eigenvalues  $\lambda_\nu$  (sorted from the lowest to the highest value) and corresponding eigenvectors  $\mathbf{v}_\nu$  are obtained. They can be summarized into the spectral matrix

$$\mathbf{\Lambda} = \text{diag}([\lambda_1, \lambda_2, \dots, \lambda_n]), \quad (2.24)$$

a diagonal matrix with eigenvalues on its diagonal, and the matrix

$$\mathbf{\Phi} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] \quad (2.25)$$

composed of eigenvectors as its columns.

Using the fact that the force density matrix  $\mathbf{E}$  is real symmetric, spectral decomposition can be done as

$$\mathbf{E} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^T. \quad (2.26)$$

The modal matrix  $\mathbf{\Lambda}$  is required to have  $D + 1$  zero eigenvalues and other positive which is probably not satisfied without any external intervention. Thus, the modified modal matrix  $\bar{\mathbf{\Lambda}}$  is created with the modified set of eigenvalues  $\bar{\lambda}_\nu$ . This set is defined as

$$\bar{\lambda}_\nu = \begin{cases} 0, & \nu \leq D + 1, \\ \lambda_\nu, & \nu > D + 1 \end{cases} \quad (2.27)$$

and the modified force density matrix  $\bar{\mathbf{E}}$  using the spectral decomposition

$$\bar{\mathbf{E}} = \mathbf{\Phi}\bar{\mathbf{\Lambda}}\mathbf{\Phi}^T. \quad (2.28)$$

is created.

The task of the first design stage is not to find the feasible force density matrix  $\bar{\mathbf{E}}$ , but to find the feasible vector  $\mathbf{q}$  of force densities. If the direct definition (2.23) of  $\mathbf{E}$  is used,

it can be claimed according to [29] that  $i$ -th row  $\mathbf{E}_i \in \mathbb{R}^{1 \times n}$  of the force density matrix is possible to obtain by an equation

$$(\mathbf{E}_i)^T = {}_i\mathbf{B}\mathbf{q}, \quad (2.29)$$

where the component  ${}_iB_{j,k}$  of the matrix  ${}_i\mathbf{B} \in \mathbb{R}^{n \times m}$  is defined as

$${}_iB_{j,k} = \begin{cases} 1, & i = j \wedge \text{node } i \text{ contains member } k \in M_i, \\ -1, & i \neq j \wedge \text{nodes } i, j \text{ are connected by member } k, \\ 0, & \text{otherwise.} \end{cases} \quad (2.30)$$

By summarizing  ${}_i\mathbf{B}$  into a matrix  $\mathbf{B} \in \mathbb{R}^{n^2 \times m}$  in the manner of

$$\mathbf{B}^T = [{}_1\mathbf{B}^T, {}_2\mathbf{B}^T, \dots, {}_n\mathbf{B}^T] \quad (2.31)$$

and defining a vector  $\mathbf{g} \in \mathbb{R}^{n^2}$  as

$$\mathbf{g} = [\mathbf{E}_1, \mathbf{E}_2, \dots, \mathbf{E}_n]^T, \quad (2.32)$$

an equation

$$\mathbf{B}\mathbf{q} = \mathbf{g} \quad (2.33)$$

applies. Thus, using pseudo-inversion, the force density vector can be expressed as

$$\mathbf{q} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{g}. \quad (2.34)$$

Analogously, it applies that

$$\tilde{\mathbf{q}} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\tilde{\mathbf{g}}, \quad (2.35)$$

where the vector  $\tilde{\mathbf{g}}$  is assembled from  $\bar{\mathbf{E}}$  similarly to  $\mathbf{g}$ .

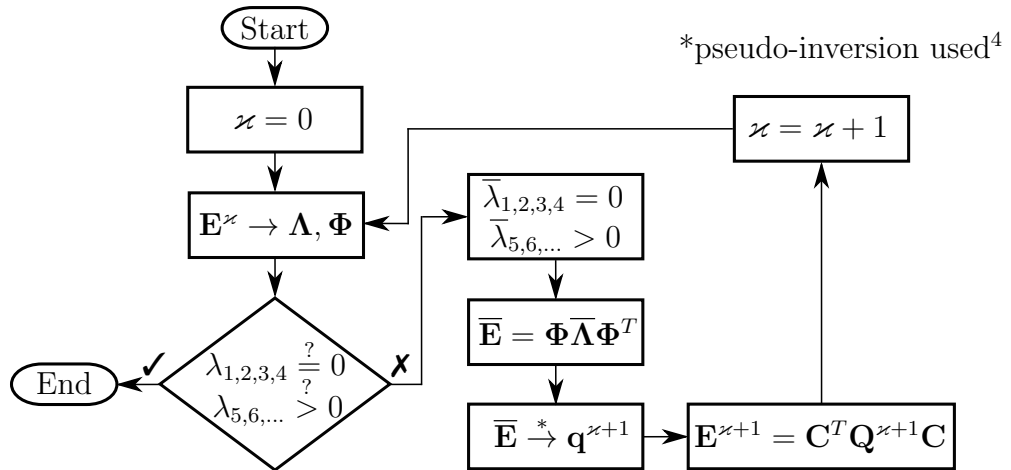


Figure 2.2.: Flow chart of the first stage of the AFDM.

<sup>4</sup>Mentioning, that pseudo-inversion is used during the process, is important. Matrices  $\mathbf{E}^{\kappa+1} \neq \bar{\mathbf{E}}$  which is the reason why an iterative process is needed. As claimed in [29], for  $\kappa \gg 1$ :  $\mathbf{E}^{\kappa+1} \approx \bar{\mathbf{E}}$ .

In summary, the new vector  $\tilde{\mathbf{q}}$  of force densities is derived from the modified force density matrix  $\tilde{\mathbf{E}}$  in terms of the least square method. Due to the usage of the least square method, the new matrix  $\tilde{\mathbf{E}}$  created from the vector  $\tilde{\mathbf{q}}$  using the equation (2.20) is probably not the same as the matrix  $\bar{\mathbf{E}}$ . Therefore, the matrix  $\tilde{\mathbf{E}}$  does not probably have the required eigenvalues  $\bar{\lambda}_\nu$ . As suggested in [29], the matrix  $\tilde{\mathbf{E}}$  attains eigenvalues  $\bar{\lambda}_\nu$  (with sufficient accuracy) after a sufficient number of iterations.

In Figure 2.2, there is a flow chart of the first design stage of the AFDM. To emphasize the iterative character of the algorithm, the notation  $\mathbf{q}^{\varkappa+1} = \tilde{\mathbf{q}}$  and  $\mathbf{E}^{\varkappa+1} = \tilde{\mathbf{E}}$  is used, where  $\varkappa$  denotes an iterator. The flow chart is only a symbolical illustration, not all necessary operations are visualized in it, it is meant to be only a supplementary material to the previous text.

### 2.1.5. Second design stage

The aim of the second design stage is to determine the self-equilibrated configuration of the structure represented by the vector  $\mathbf{X}$  of nodal coordinates. There are two requirements that have to be satisfied. First, rank of the geometry matrix  $\mathbf{G}$  has to be  $\frac{1}{2}(D^2 + D)$  to assure super-stability of the resulting structure. Second, as indicated in the introduction to the AFDM, elevation of preselected nodes has to meet predefined values.

Solution to this problem starts with mentioning already discussed equilibrium equations (2.22). They can be easily turned into self-equilibrium equations by omitting external forces on the right side as

$$\begin{bmatrix} \mathbf{E} & 0 & 0 \\ 0 & \mathbf{E} & 0 \\ 0 & 0 & \mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \mathbf{HX} = \mathbf{0}. \quad (2.36)$$

As pointed out in [30], equations (2.36) are actually non-linear with respect to the nodal coordinates  $\mathbf{x}, \mathbf{y}, \mathbf{z}$  – the force density matrix  $\mathbf{E}$  is dependent on member lengths and these lengths are non-linear functions of nodal coordinates. But if the force densities are already determined,  $\mathbf{E}$  is constant and self-equilibrium equations become linear.

Rank deficiency of the matrix  $\mathbf{E}$  is  $D + 1$ , therefore, rank deficiency of the matrix  $\mathbf{H}$  is  $3(D + 1)$ . This means that  $3(D + 1)$  linearly independent solutions (configurations of the structure) of the system (2.36) can be found when no geometrical constraint is applied.

Hence, there is space for adding constraints on specified altitude in specified nodes. Particular values  $b_v$  ( $v = 1, 2, \dots, n^b$ ) of altitudes are summarized into the vector  $\mathbf{b} \in \mathbb{R}^{n^b}$  as

$$\mathbf{b} = [b_1, b_2, \dots, b_{n^b}]^T, \quad (2.37)$$

where  $n^b$  is the number of nodes with specified altitude. Nodes with specified altitude are marked in the matrix  $\mathbf{A} \in \mathbb{R}^{n^b \times n}$  assembled as

$$A_{v,i} = \begin{cases} 1, & \text{node } i \text{ has specified altitude } b_v, \\ 0, & \text{otherwise.} \end{cases} \quad (2.38)$$

To meet specified nodal altitudes, the system of equations

$$\mathbf{A}\mathbf{z} = \mathbf{b} \quad (2.39)$$

has to be satisfied in addition to the system (2.36) too.

Systems (2.36) and (2.39) are possible to be merged together into a form of

$$\begin{bmatrix} \mathbf{E} & 0 & 0 \\ 0 & \mathbf{E} & 0 \\ 0 & 0 & \mathbf{E} \\ 0 & 0 & \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \widehat{\mathbf{H}}\mathbf{X} = \widehat{\mathbf{b}}, \quad (2.40)$$

where  $\widehat{\mathbf{b}} \in \mathbb{R}^{dn+n^b}$  and  $\widehat{\mathbf{H}} \in \mathbb{R}^{(dn+n^b) \times dn}$ .

The solution of this linear system can be achieved using common methods, e.g. pseudo-inversion. When an initial configuration  $\mathbf{X}_0$  is specified, it is possible to apply an iterative approach, which is used in this work in form of MATLAB function *fsolve* application.

When this linear system is solved, a geometrical non-degeneracy condition has to be checked. If the rank of the geometry matrix  $\mathbf{G}$  is  $\frac{1}{2}(D^2 + D)$ , the solution  $\mathbf{X}$  is declared as the resulting configuration. Otherwise, altitude constraints (or constraints for other coordinates) are added/modified/removed, modified system of equations is solved and the new matrix  $\mathbf{G}$  is checked again. This is repeated until the feasible configuration is achieved. However, it is good to point out that the requirement on the rank of the matrix  $\mathbf{G}$  is satisfied in most cases so adjustments of constraints are generally not needed.

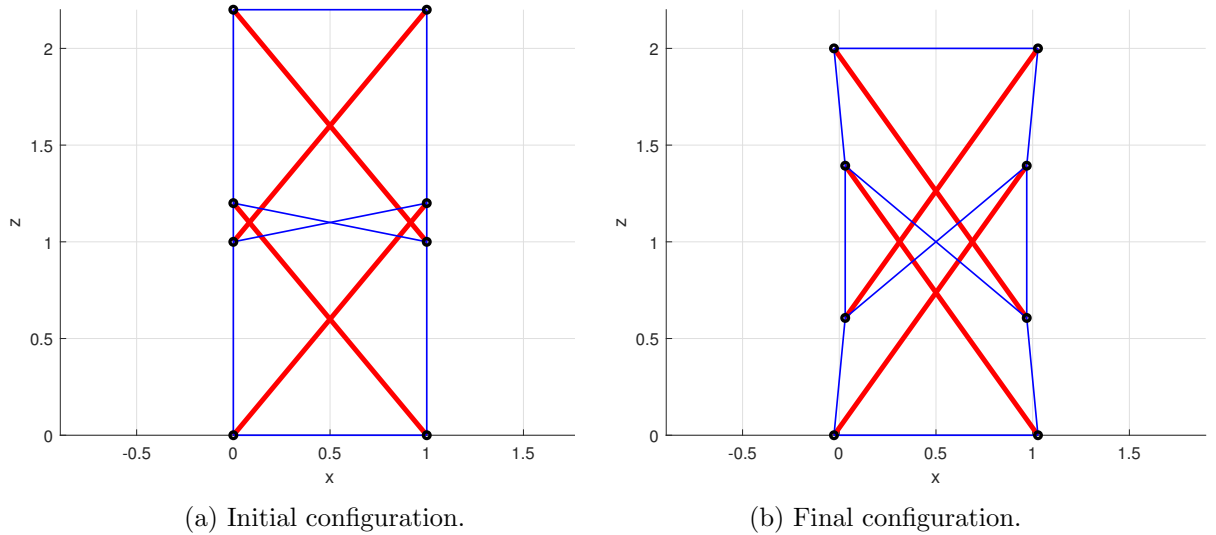


Figure 2.3.: A 2-D example structure subjected to the AFDM.

### 2.1.6. Examples

This section contains two structures, which initial configurations are subjected to the AFDM. Values of members' properties of both structures are available in Appendix A.1.

The first one is a 2-D tensegrity tower with two simple floors. The total height of initial structure is 2.2 [m], while it is required that the final structure has the total height of exactly 2 [m]. Both initial and final super-stable configuration are visualized in Figure 2.3. The symmetry of the resulting force density vector corresponds with the symmetry of the structure configuration, while particular values of force densities in particular members are not important. The required height is met accurately.

A 3-D tensegrity tower with two floors represents the second example. The initial height of the structure is 2 [m]. Let altitudes of all nodes be prescribed, specifically as 0 [m] for nodes in the lowest layer, 1 [m] for nodes in the middle layer, and 2 [m] for nodes in the top layer. Both initial and final configuration are visualized in Figure 2.4 (beware of different scaling). In this case, altitudes are not met precisely (expected due to definition of more than  $3(D + 1)$  constraints) – altitude of the bottom layer is 0 [m] in all nodes, altitude of the middle layer is 1.005 [m] in all nodes, and altitude of the top layer is 1.995 [m] in all nodes. Again, the symmetry of the resulting force density vector corresponds with the symmetry of the final configuration.

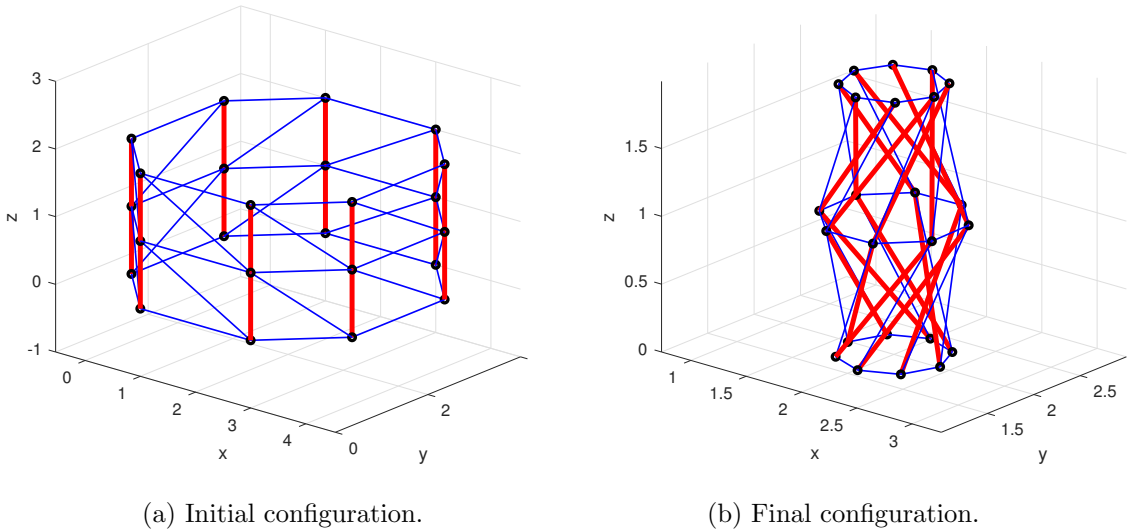


Figure 2.4.: A 3-D example structure subjected to the AFDM.

## 2.2. Dynamic relaxation method

Unlike the previous described form-finding method, the AFDM, which is based purely on mathematical analysis and operations with matrices, the dynamic relaxation method (DRM) is based on the analysis of mechanical system real behaviour.

The general concept of the DRM is to trace step-by-step the motion of the structure until the structure reaches a static equilibrium due to introduced damping. Originally, the method uses the viscous damping of nodal movements. According to [2], damping is formed proportionally to the product of nodal velocities and mass components in this case and most rapid convergence is obtained by critically damping the lowest frequency

mode. An alternative to viscous damping is the usage of kinetic damping procedure which authors of [3, 25] found very efficient regarding speed of convergence in case of very large disturbances from equilibrated configuration.

### 2.2.1. Formulation of the DRM

In this work, the kinetic damping process is applied. As described in [2], this approach represents a procedure when movements of an undamped structure are traced until a local peak (maximum) of total kinetic energy is reached. At this point, all nodal velocities are set to zero (no movements of the structure) and the process is restarted from the current configuration. These steps are repeated until the energy is completely dissipated and a static equilibrium is found. Following section introduces governing equations of this process presented in [2].

According to the Newton's second law, motion of  $i$ -th node in direction  $d$  at time  $t$  is governed by equation

$$r_{i,d}(t) = m_{i,d}\dot{v}_{i,d}(t), \quad i = 1, 2, \dots, n, \quad d = 1, 2, 3, \quad t \geq 0, \quad (2.41)$$

where  $r_{i,d}$  represents a residual force,  $\dot{v}_{i,d}$  is nodal acceleration and  $m_{i,d}$  is lumped mass.

#### Residual force

At any time, it is possible to express the residual force  $r_{i,d}$  according to [2] as

$$r_{i,d}(t) = f_{i,d}(t) + \sum_{k_i \in M_i} s_{k_i}(t) \cdot n_{k_i,d}(t), \quad (2.42)$$

where summation is done through the set  $M_i$  of all the members connected to the  $i$ -th node,  $s_{k_i}$  is the internal force in the  $k_i$ -th member,  $f_{i,d}$  is an external load applied to the  $i$ -th node and  $n_{k_i,d}$  is the  $d$ -th component of the  $k_i$ -th member's normalized direction vector  $\mathbf{n}_{k_i}$  expressed as

$$n_{k_i,d}(t) = \frac{x_{j,d}(t) - x_{i,d}(t)}{l_{k_i}(t)}, \quad (2.43)$$

where  $x_{i,d}$  the  $d$ -th coordinate of the node  $i$ ,  $x_{j,d}$  is the  $d$ -th coordinate of the  $k_i$ -th member's second node and  $l_{k_i}$  is the member's length. The product  $s_{k_i} \cdot \mathbf{n}_{k_i}$  expresses the spatial vector of the internal force  $\mathbf{s}_{k_i}$  acting on the  $i$ -th node, while the product  $s_{k_i} \cdot n_{k_i,d}$  in (2.42) specifies the component of this force in the direction  $d$ .

It should be explicitly noted that the direction vector  $\mathbf{n}_{k_i}$  varies for the same index  $k_i$  depending on which node is the main one (node with the index  $i$ ) and which node is the second one (node  $j$ ). This definition of  $\mathbf{n}_{k_i}$  by the eq. (2.43) ensures the law of action and reaction satisfaction in case of member forces.

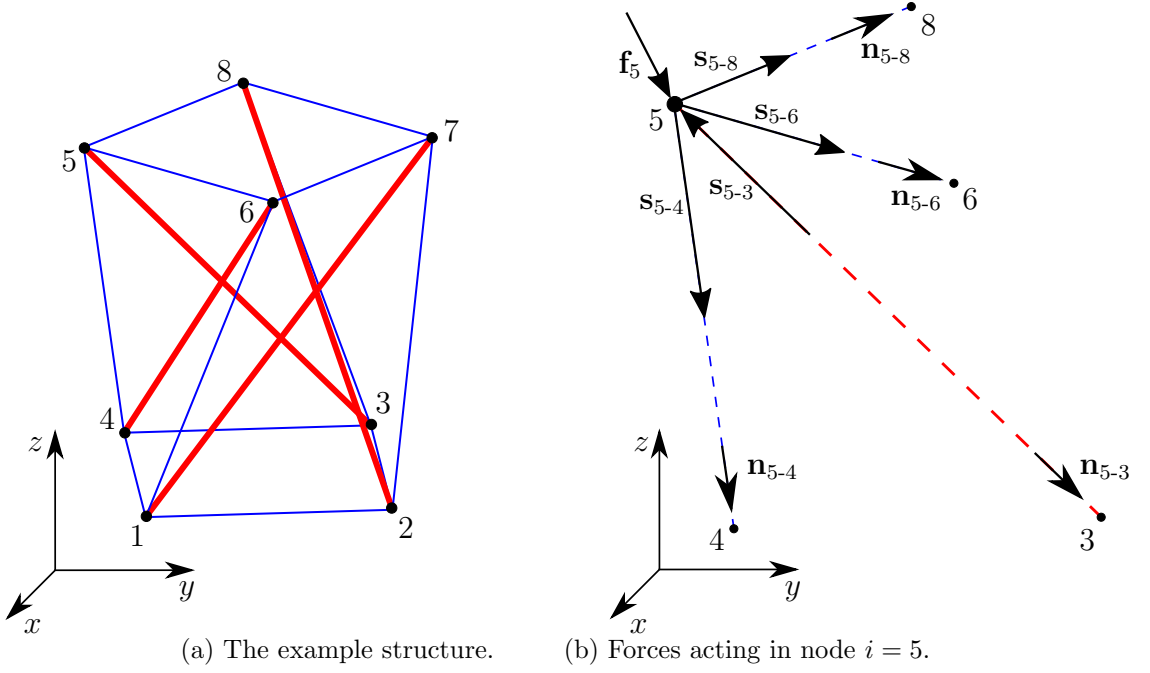


Figure 2.5.: The example structure and an analysis of nodal forces.

In Figure 2.5a, there is an example 3-D tensegrity, and forces acting on a chosen node are visualized in Figure 2.5b. Figures should clarify the issue of both acting nodal forces and indexing  $i, j, k_i$ . With the focus on the right figure, the node chosen for the analysis is the 5-th node ( $i = 5$ ). Four members are connected to the node, so there are four internal forces  $\mathbf{s}_{k_i} = s_{k_i} \cdot \mathbf{n}_{k_i}$  and direction vectors  $\mathbf{n}_{k_i}$ , where the index  $k_i$  is gradually  $k_i = 5-3, 5-4, 5-6, 5-8$  (the notation is only symbolical for better orientation). And, for instance, in case of the member  $k_i = 5-3$ , its first node is obviously  $i = 5$  and its second node is  $j = 3$ . There is also another force acting in the node 5, an external force  $\mathbf{f}_5$ .

Combining (2.42) and (2.43), it is possible to write the equation for the residual force as

$$r_{i,d}(t) = f_{i,d}(t) + \sum_{k_i \in M_i} s_{k_i}(t) \cdot \frac{x_{j,d}(t) - x_{i,d}(t)}{l_{k_i}(t)}. \quad (2.44)$$

Member's length (distance between two corresponding nodes) is calculated as the Euclidean norm

$$l_k(t) = \|\boldsymbol{\xi}_j(t) - \boldsymbol{\xi}_i(t)\| = \sqrt{\sum_d [x_{j,d}(t) - x_{i,d}(t)]^2}. \quad (2.45)$$

Calculation of the internal force in a member depends on the type of the member. For a strut, it is defined as a standard elastic force

$$s_k(t) = k_k^e \cdot [l_k(t) - l_k^0], \quad k = 1, 2, \dots, m, \quad (2.46)$$



where  $l_k^0$  is the rest length of the member and

$$k_k^e = \frac{E^s A^s}{l_k^0} \quad (2.47)$$

is its elastic stiffness, where the strut's cross section  $A^s$  and Young's modulus  $E^s$  appear.

For a cable, it is defined similarly but as a one-sided binding. Thus,

$$s_k(t) = \begin{cases} k_k^e \cdot [l_k(t) - l_k^0] & \text{for } l_k(t) - l_k^0 > 0, \\ 0 & \text{else,} \end{cases} \quad (2.48)$$

where the elastic stiffness  $k_k^e$  is now

$$k_k^e = \frac{E^c A^c}{l_k^0}, \quad (2.49)$$

where  $A^c$  is the cross section of a cable and  $E^c$  is Young's modulus of a cable.

If the weight of the structure is considered, it is necessary to add the gravity  $f_i^g$  in form of

$$f_i^g = g \cdot m_i^{real} \quad (2.50)$$

to the appropriate component of the external load  $f_{i,d}$  (it is  $d = 3$  in this work). The gravitational acceleration is denoted as  $g$  and  $m_i^{real}$  is the real lumped mass in the  $i$ -th node expressed as

$$m_i^{real} = \sum_{k_i \in M_i^s} \frac{m_{k_i}^s}{2}, \quad (2.51)$$

where summation is done through the set  $M_i^s$  of all struts connected to the  $i$ -th node, and  $m_{k_i}^s$  is the weight of the  $k_i$ -th strut. Mass of cables is not considered.

### Updating nodal position and velocity

Central difference formula can be used as an approximation of nodal acceleration  $\dot{v}_{i,d}(t)$  in form of

$$\dot{v}_{i,d}(t) = \frac{v_{i,d}(t + \frac{\Delta t}{2}) - v_{i,d}(t - \frac{\Delta t}{2})}{\Delta t}, \quad (2.52)$$

so nodal velocity  $v_{i,d}(t + \frac{\Delta t}{2})$  is obtained with respect to the eq. (2.41) in form of

$$v_{i,d}(t + \frac{\Delta t}{2}) = v_{i,d}(t - \frac{\Delta t}{2}) + \frac{\Delta t}{m_{i,d}} \cdot r_{i,d}(t), \quad (2.53)$$

where  $\Delta t$  is time step and the residual force  $r_{i,d}$  is expressed by eq. (2.44).

Updated nodal position  $x_{i,d}(t + \Delta t)$  can be derived using average velocity (at midpoint of time interval) as

$$x_{i,d}(t + \Delta t) = x_{i,d}(t) + \Delta t \cdot v_{i,d}(t + \frac{\Delta t}{2}). \quad (2.54)$$

When the new configuration is obtained, it is possible to express the updated residual force  $r_{i,d}(t + \Delta t)$  according to the eq. (2.44) and repeat the process  $r_{i,d}(t) \rightarrow v_{i,d}(t + \frac{\Delta t}{2}) \rightarrow x_{i,d}(t + \Delta t)$  for  $t = t + \Delta t$ .

### 2.2.2. Ensuring numerical stability

As pointed out in [1] or [2], when a static solution rather than tracing the real dynamic behaviour is searched for, there is no need for setting  $m_{i,d}$  to real values ( $m_{i,d} = m_i^{real}$ ), it is advantageous to adjust these masses to ensure convergence of the iterative process.

When searching for a suitable choice of  $m_{i,d}$ , it is possible to come from the idea of finding a critical value  $\Delta t^{cr}$  of the time increment. The author of [2] claims that the process is divergent when the velocity  $v_{i,d}(t + \frac{\Delta t}{2})$  of one node in some direction is higher and with opposite sign compared to the velocity  $v_{i,d}(t - \frac{\Delta t}{2})$  in the previous step. Therefore, the critical case is present when the velocity reaches the state

$$v_{i,d}(t + \frac{3\Delta t^{cr}}{2}) = -v_{i,d}(t + \frac{\Delta t^{cr}}{2}) = v_{i,d}(t - \frac{\Delta t^{cr}}{2}). \quad (2.55)$$

Using the idea (2.55), it is possible to express the critical time increment  $\Delta t^{cr}(m_{i,d})$  dependent on arbitrarily chosen fictitious mass  $m_{i,d}$ . Apparently, a reverse approach can be used – expressing a critical value  $m_{i,d}^{cr}(\Delta t)$  of mass dependent on arbitrary choice of time increment  $\Delta t$ . Finally, fictitious mass  $m_{i,d}$  used in the iterative process is chosen as  $m_{i,d} = m_{i,d}^{cr}$ , while its explicit expression is set out in [2].

During the implementation of this work, it turned out that it is better to use an alternative setting of  $m_{i,d}$  recommended by [1] in form of

$$m_{i,d} = m_{i,d}(t) = 2\Delta t^2 \cdot k_{i,d}(t). \quad (2.56)$$

The direct stiffness  $k_{i,d}$  is calculated by summing up partial stiffnesses supplied by the members connected to the  $i$ -th node. Each partial stiffness is composed of the elastic stiffness  $k_{k_i}^e$  and the geometric stiffness  $k_{k_i}^g$ , so  $k_{i,d}$  can be written as

$$k_{i,d}(t) = \sum_{k_i \in M_i} [k_{k_i}^e + k_{k_i}^g(t)] \left[ \frac{x_{j,d}(t) - x_{i,d}(t)}{l_{k_i}(t)} \right]^2, \quad (2.57)$$

where

$$k_{k_i}^g(t) = \frac{s_{k_i}(t)}{l_{k_i}(t)} \quad (2.58)$$

and  $k_{k_i}^e$  has been already defined in eq. (2.47) for struts, or in (2.49) for cables.

### 2.2.3. Reinitialization at kinetic energy peaks

As it was already mentioned in the introduction of section 2.2.1, kinetic damping is used. When the process is starting or restarting (at time  $t = t^*$ ), nodal velocity is set

$$v_{i,d}(t^*) = 0. \quad (2.59)$$

If this is combined with forward difference formula

$$\dot{v}_{i,d}(t^*) = \frac{v_{i,d}(t^* + \frac{\Delta t}{2}) - v_{i,d}(t^*)}{\frac{\Delta t}{2}} \quad (2.60)$$

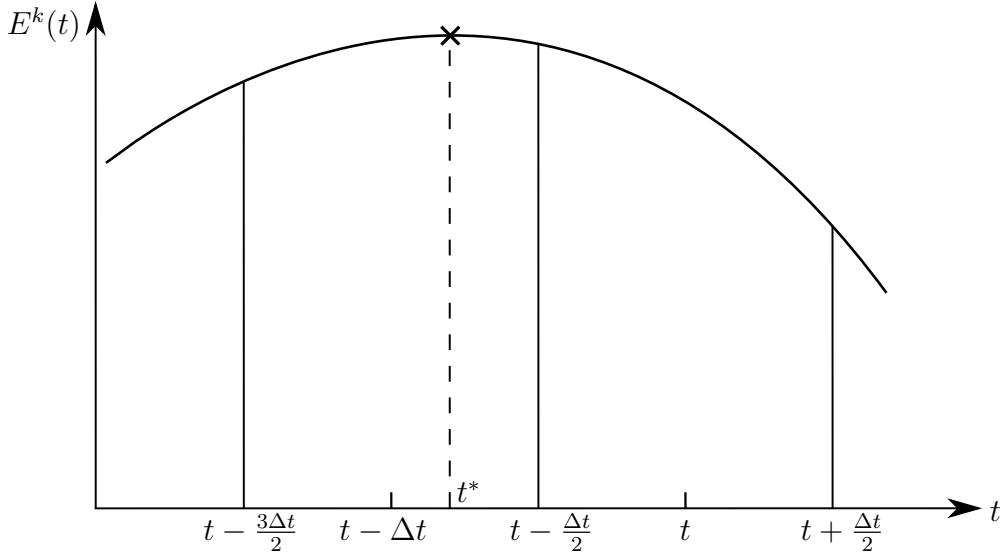


Figure 2.6.: Local peak of total kinetic energy.

and with the eq. (2.41), it is possible to express initial value of nodal velocity  $v_{i,d}(t^* + \frac{\Delta t}{2})$  after start or restart as

$$v_{i,d}(t^* + \frac{\Delta t}{2}) = \frac{\Delta t}{2m_{i,d}(t^*)} \cdot r_{i,d}(t^*) \quad (2.61)$$

which is used in the first iteration after start or restart.

It is also important to point out that the detection of the local peak of the kinetic energy  $E^k$ , which is calculated as

$$E^k(t) = \frac{1}{2} \sum_{i=1}^n \sum_{d=1}^3 m_{i,d}(t) v_{i,d}^2(t), \quad (2.62)$$

is done by monitoring the declination of the energy. Using an inequality

$$E^k(t - \frac{3\Delta t}{2}) < E^k(t - \frac{\Delta t}{2}) \geq E^k(t + \frac{\Delta t}{2}), \quad (2.63)$$

it is clear that the kinetic energy peak  $E^k(t^*)$  occurs in the interval  $t^* \in \langle t - \frac{3\Delta t}{2}, t + \frac{\Delta t}{2} \rangle$ , which is visualized in Figure 2.6, but it is not possible to specify the energy peak only from this information, so the function  $E^k(t)$  has to be approximated.

According to [2], a quadratic polynomial can be used to fit the current ( $t + \frac{\Delta t}{2}$ ) and two previous values of kinetic energy. Then the interval of possible peak appearance is due to the symmetry of the parabola limited to  $t^* \in \langle t - \Delta t, t \rangle$  and, moreover, it is possible to derive the exact time  $t^*$  of the peak of the approximated energy according to [2] as

$$t^* = t - \delta t^* = t - \Delta t \cdot c, \quad (2.64)$$

where

$$c = \frac{E^k(t - \frac{\Delta t}{2}) - E^k(t + \frac{\Delta t}{2})}{2E^k(t - \frac{\Delta t}{2}) - E^k(t + \frac{\Delta t}{2}) - E^k(t - \frac{3\Delta t}{2})}. \quad (2.65)$$

In the same manner as the eq. (2.54) was obtained, calculation of coordinates  $x_{i,d}(t^*)$  at peak time can be done:

$$x_{i,d}(t^*) = x_{i,d}(t + \Delta t) - \Delta t \cdot v_{i,d}(t + \frac{\Delta t}{2}) - \Delta t \cdot c \cdot v_{i,d}(t - \frac{\Delta t}{2}). \quad (2.66)$$

Using this equation and eq. (2.53), it applies that

$$x_{i,d}(t^*) = x_{i,d}(t + \Delta t) - \Delta t \cdot (1 + c) \cdot v_{i,d}(t + \frac{\Delta t}{2}) + \frac{\Delta t^2}{2} \cdot c \cdot \frac{r_{i,d}(t)}{m_{i,d}(t)}. \quad (2.67)$$

### 2.2.4. The DRM algorithm

In this section, the DRM algorithm is presented. First, it is suitable to introduce matrix forms of some related variables and define their dimensions.

Coordinates  $x_{i,d}$  of all nodes can be assembled again into vectors  $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$  defined in (2.3) forming the vector  $\mathbf{X} \in \mathbb{R}^{3n}$  of nodal coordinates defined in (2.2). This can be summarized into a symbolic notation  $x_{i,d} \rightarrow \mathbf{x}, \mathbf{y}, \mathbf{z} \rightarrow \mathbf{X}$ .

Analogously, vectors summarizing other nodal properties are created. The summary (including already defined coordinate vector  $\mathbf{X}$  for better orientation) of these vectors follows:

$$x_{i,d}(t) \in \mathbb{R} \rightarrow \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t) \in \mathbb{R}^n \rightarrow \mathbf{X}(t) \in \mathbb{R}^{3n}, \quad (2.68)$$

$$v_{i,d}(t) \in \mathbb{R} \rightarrow \mathbf{v}_x(t), \mathbf{v}_y(t), \mathbf{v}_z(t) \in \mathbb{R}^n \rightarrow \mathbf{V}(t) \in \mathbb{R}^{3n}, \quad (2.69)$$

$$\dot{v}_{i,d}(t) \in \mathbb{R} \rightarrow \dot{\mathbf{v}}_x(t), \dot{\mathbf{v}}_y(t), \dot{\mathbf{v}}_z(t) \in \mathbb{R}^n \rightarrow \dot{\mathbf{V}}(t) \in \mathbb{R}^{3n}, \quad (2.70)$$

$$m_{i,d}(t) \in \mathbb{R} \rightarrow \mathbf{m}_x(t), \mathbf{m}_y(t), \mathbf{m}_z(t) \in \mathbb{R}^n \rightarrow \mathbf{M}(t) \in \mathbb{R}^{3n}, \quad (2.71)$$

$$k_{i,d}(t) \in \mathbb{R} \rightarrow \mathbf{k}_x(t), \mathbf{k}_y(t), \mathbf{k}_z(t) \in \mathbb{R}^n \rightarrow \mathbf{K}(t) \in \mathbb{R}^{3n}, \quad (2.72)$$

$$r_{i,d}(t) \in \mathbb{R} \rightarrow \mathbf{r}_x(t), \mathbf{r}_y(t), \mathbf{r}_z(t) \in \mathbb{R}^n \rightarrow \mathbf{R}(t) \in \mathbb{R}^{3n}, \quad (2.73)$$

$$f_{i,d}(t) \in \mathbb{R} \rightarrow \mathbf{f}_x(t), \mathbf{f}_y(t), \mathbf{f}_z(t) \in \mathbb{R}^n \rightarrow \mathbf{F}(t) \in \mathbb{R}^{3n}. \quad (2.74)$$

Properties, that belong to members, are summarized into vectors as follows:

$$\mathbf{l}(t) = [l_1(t), l_2(t), \dots, l_m(t)]^T \in \mathbb{R}^m, \quad (2.75)$$

$$\mathbf{s}(t) = [s_1(t), s_2(t), \dots, s_m(t)]^T \in \mathbb{R}^m, \quad (2.76)$$

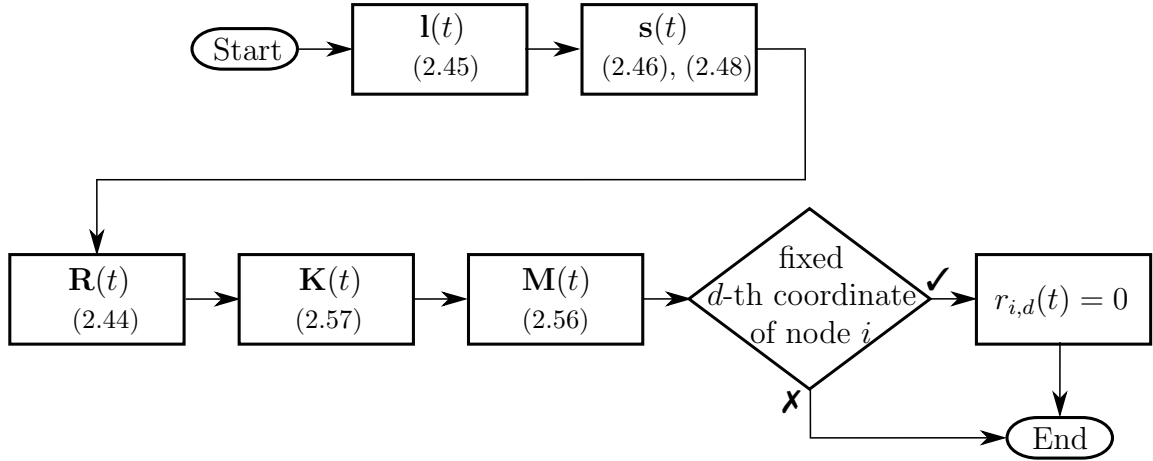
$$\mathbf{k}^e(t) = [k_1^e(t), k_2^e(t), \dots, k_m^e(t)]^T \in \mathbb{R}^m, \quad (2.77)$$

$$\mathbf{k}^g(t) = [k_1^g(t), k_2^g(t), \dots, k_m^g(t)]^T \in \mathbb{R}^m, \quad (2.78)$$

Before a global algorithm for the whole method is introduced, a sub-algorithm determining member lengths  $\mathbf{l}$ , member internal forces  $\mathbf{s}$ , nodal residuums  $\mathbf{R}$  and fictitious nodal masses  $\mathbf{M}$  is visualized in Figure 2.7. It is quite a straightforward algorithm with possibility of prescribing fixed position in some direction to preselected node(s)<sup>5</sup>.

---

<sup>5</sup>The same behaviour, prescription of specific altitude to some nodes, as in case of the AFDM can be achieved.


 Figure 2.7.: Flow chart of  $\mathbf{l}$ ,  $\mathbf{s}$ ,  $\mathbf{R}$ ,  $\mathbf{M}$  calculation.

In order to make the DRM algorithm output similar to the AFDM algorithm output, there is a requirement for the conversion of an internal force  $s_k(t)$  to a force density

$$q_k(t) = \frac{s_k(t)}{l_k(t)} \quad (2.79)$$

and particular force densities create the force density vector

$$\mathbf{q}(t) = [q_1, q_2, \dots, q_m]^T \quad (2.80)$$

together, which is the same definition as in (2.12).

In the introduction to section 2.2.1, it was mentioned that the process is repeated until a static equilibrium is found. This can be interpreted as a stopping condition in form of an acceptably low value of the total nodal residuum norm  $\|\mathbf{R}\|$ . In this work, the Euclidean definition of a norm is used. The stopping condition can be written as

$$\|\mathbf{R}(t)\| \leq R^{tol}, \quad (2.81)$$

where  $R^{tol}$  is a tolerance parameter set acceptably low.

The complete DRM algorithm presented in Figure 2.8 is with some modifications inspired by [1].

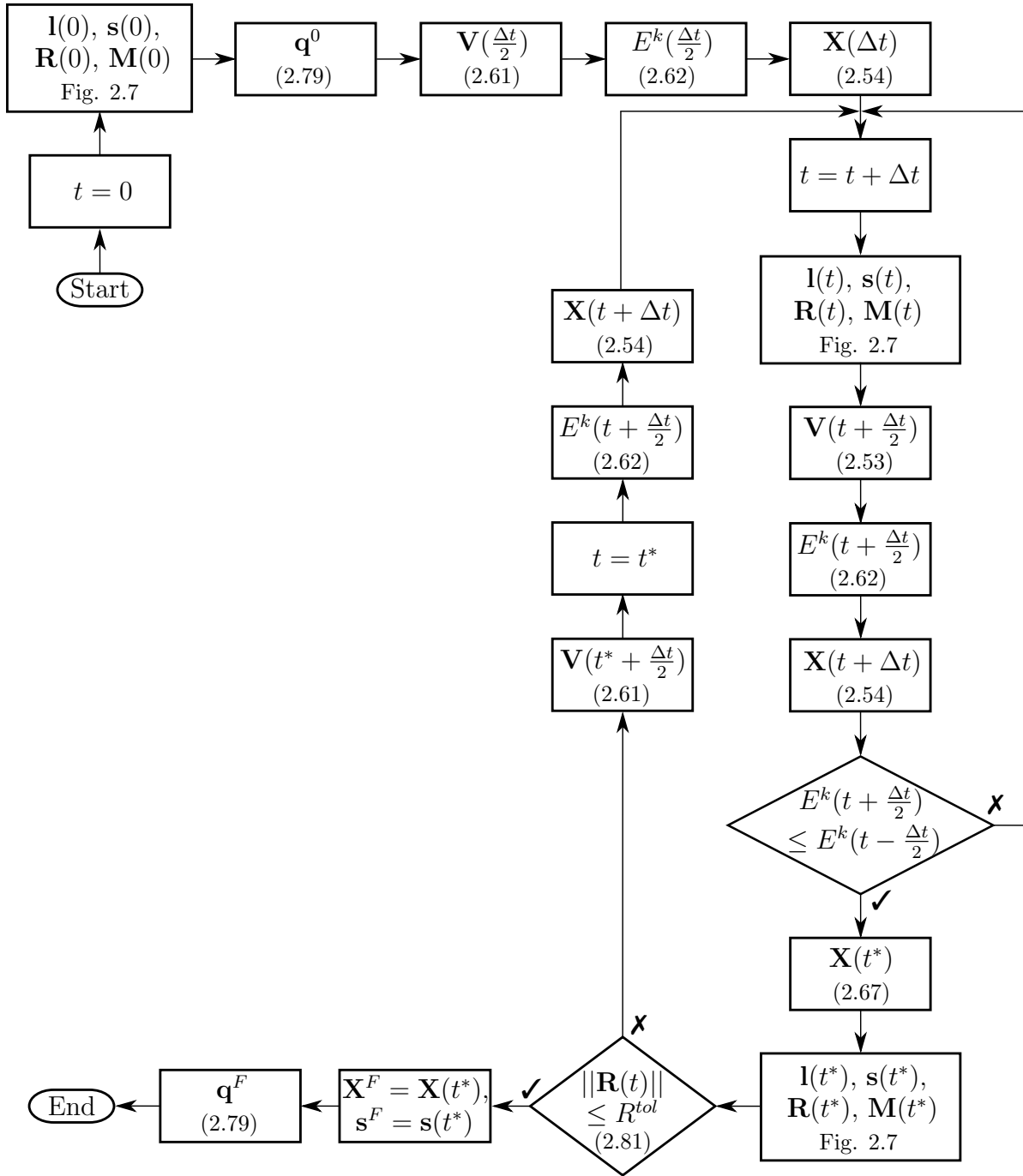


Figure 2.8.: Flow chart of the DRM process.

### 2.2.5. Examples

The same example structures as shown in case of the AFDM in section 2.1.6 are presented to illustrate the usage of the DRM. The final super-stable configuration calculated by the AFDM is now used as the initial configuration for the DRM, while cables' rest lengths

## 2. Form-finding

---

are calculated from particular force densities provided by the AFDM. Thus, the initial state of structures is now exactly the same as the final state in case of the AFDM. To demonstrate benefits of the the DRM, let all three cables on the right side of the 2-D tensegrity be shortened by 0.1 [m] and all three cables on the left side be extended by 0.1 [m]. In case of the second example, the 3-D tensegrity tower, all eight horizontal cables forming the middle ring are shortened by 0.2 [m]. Both examples with initial and final configuration are depicted in figures 2.9 and 2.10. Exploited values of members' properties and parameter  $R^{tol}$  are available in Appendix A.1.

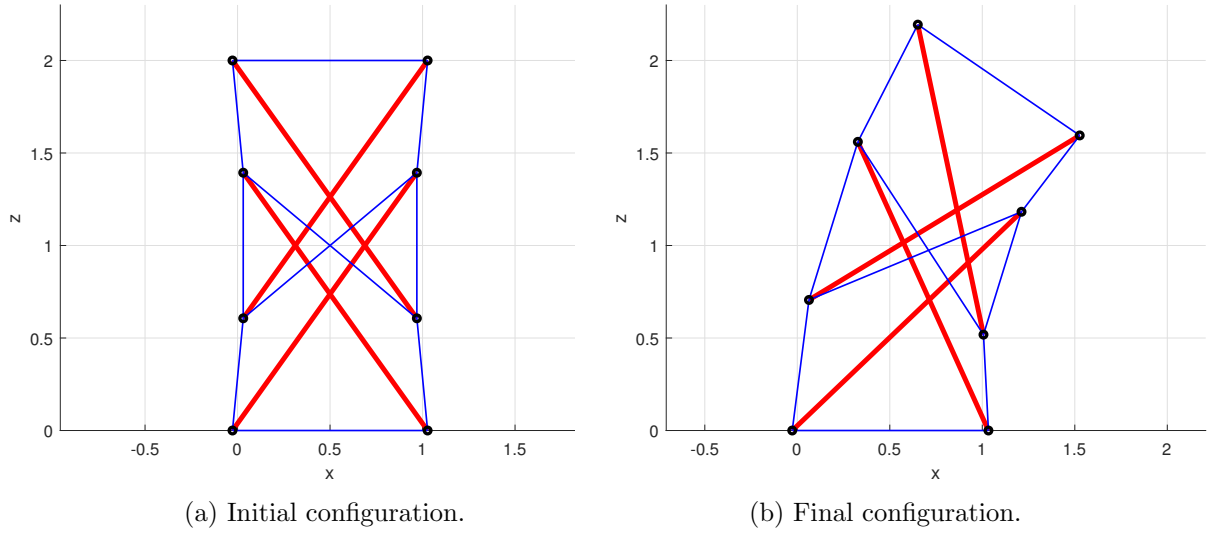


Figure 2.9.: A 2-D example structure subjected to the DRM.

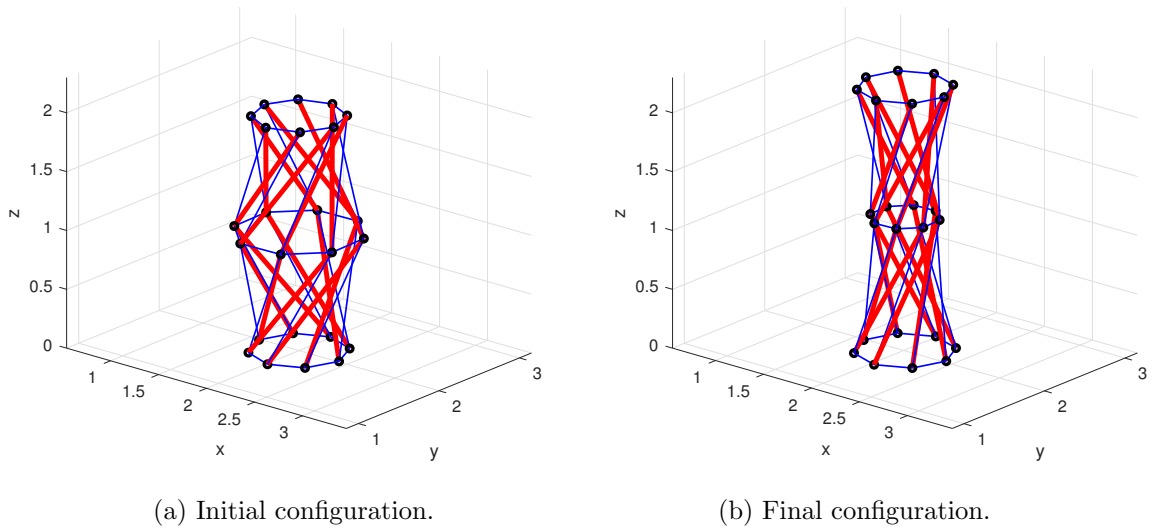


Figure 2.10.: A 3-D example structure subjected to the DRM.

### **2.3. Comparison of the AFDM and the DRM**

In this chapter, two form-finding methods were presented, each one based on a different approach. Their applicability in the active tensegrity design process is different, while both of them bring advantages when applied in an appropriate design stage.

The AFDM is very useful when only few inputs from a designer are available – it allows to find a super-stable form of a tensegrity only based on given topology and an initial idea of the structure appearance. This becomes very helpful especially in the beginning of the design process when the main aim is the exploration of various stable tensegrity configurations with the same topology without much effort. From the explored set, the designer is able to choose the one with advantageous shape and other features with respect to particular tensegrity application.

On the other hand, in further design steps, when a tensegrity structure is already fully defined, the usage of the AFDM becomes quite inappropriate, since the method can change the whole structure (see examples in section 2.1.6, extensive change especially in case of the 3-D structure). This is the right time for the usage of the DRM. The method provides the information about statically equilibrated configuration of a structure without any changes of structural properties. This is extremely helpful when the aim is to investigate static response on particular structure actuation in form of changes of members' rest lengths (this work allows to actuate only cables, struts are always passive). The DRM also allows to include external forces, such as gravity or external load. Considering these facts, the DRM method can be advantageously used during the path-planning process presented in chapter 4.1.



## 3. Dynamics of tensegrity systems

This chapter is focused on modelling of tensegrity structure dynamics. It contains a brief introduction to mathematical description of tensegrity dynamics and the main emphasis is placed on the automatic generation of computational models in the Simscape software.

### 3.1. Multibody system dynamics

The aim of this section is to provide a brief summary of equations describing the dynamics of multibody systems applied on tensegrity structures. The description is based on [8] and [19].

Basically, the presented summary is a set of findings achieved by the Hamilton's principle application. These findings are called the Lagrange's equations of the first kind (the use of  $p > n^{dof}$  generalized coordinates is considered). Dynamics of a tensegrity structure with  $n^{dof}$  degrees of freedom can be described by an algebraic-differential system of  $p + r$  equations in a matrix form

$$\begin{bmatrix} \mathbf{M} & \mathbf{A}(\mathbf{q}) \\ \mathbf{A}^T(\mathbf{q}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{a}(\mathbf{q}, \dot{\mathbf{q}}, t) \\ \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) \end{bmatrix}, \quad (3.1)$$

where the number  $p$  of generalized coordinates is equal to

$$p = 6 \cdot m^s \quad (3.2)$$

in case of a 3-D system assumed in this work, the number of rigid struts in the structure is denoted as  $m^s$ , and

$$r = p - n^{dof} \quad (3.3)$$

defines how many constraint equations are specified.

Unknown variables of the system (3.1) are the Lagrange multipliers summarized in the vector

$$\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \dots, \lambda_r]^T \in \mathbb{R}^r \quad (3.4)$$

and generalized coordinates summarized in the vector  $\mathbf{q}$  that can be established in various ways (e.g. coordinates of mass centres of struts in combination with the Euler's angles). Because this question is beyond the scope of this thesis, let just generalized coordinates of the  $v$ -th strut be defined by three coordinates  $\rho_{v,d}$  and three angles  $\varphi_{v,d}$ . Thus,

$$\boldsymbol{\rho}_v = [\rho_{v,1}, \rho_{v,2}, \rho_{v,3}]^T, \quad v = 1, 2, \dots, m^s, \quad (3.5)$$

$$\boldsymbol{\varphi}_v = [\varphi_{v,1}, \varphi_{v,2}, \varphi_{v,3}]^T, \quad v = 1, 2, \dots, m^s, \quad (3.6)$$

$$\mathbf{q} = [\boldsymbol{\rho}_1, \boldsymbol{\varphi}_1, \boldsymbol{\rho}_2, \boldsymbol{\varphi}_2, \dots, \boldsymbol{\rho}_{m^s}, \boldsymbol{\varphi}_{m^s}]^T \in \mathbb{R}^p. \quad (3.7)$$

The mass matrix  $\mathbf{M}$  can be expressed as

$$\mathbf{M} = \frac{\partial}{\partial \dot{\mathbf{q}}^T} \left( \frac{\partial E^k}{\partial \dot{\mathbf{q}}} \right) \in \mathbb{R}^{p \times p}, \quad (3.8)$$

where  $E^k$  is the kinetic energy.

Let only holonomic and scleronomic constraints among generalized coordinates be considered. Constraint equations can be written in form of

$$\mathbf{c}(\mathbf{q}) = \mathbf{0} \in \mathbb{R}^r. \quad (3.9)$$

Then, the constraint Jacobian matrix  $\mathbf{A}$  is defined as

$$\mathbf{A}(\mathbf{q}) = \frac{\partial \mathbf{c}^T(\mathbf{q})}{\partial \mathbf{q}} \in \mathbb{R}^{p \times r}. \quad (3.10)$$

Vectors  $\mathbf{a}$ ,  $\mathbf{b}$  are introduced as

$$\mathbf{a}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{f} + \frac{\partial E^k}{\partial \mathbf{q}} - \frac{\partial}{\partial \mathbf{q}^T} \left( \frac{\partial E^k}{\partial \dot{\mathbf{q}}} \right) \dot{\mathbf{q}} \in \mathbb{R}^p \quad (3.11)$$

and

$$\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) = -\dot{\mathbf{A}}^T(\mathbf{q}) \dot{\mathbf{q}} \in \mathbb{R}^r. \quad (3.12)$$

In  $\mathbf{a}$ , there is the vector  $\mathbf{f}$  of all generalized forces except inertial ones (they are included by  $E^k$ ) – external loads acting on the structure, gravity forces or elastic and viscous forces generated by cables incorporated into the structure. The last two forces mentioned are defined as

$$s_k^e(t) = \begin{cases} k_k^e \cdot [l_k(t) - l_k^0] & \text{for } l_k(t) - l_k^0 > 0, \\ 0 & \text{else} \end{cases} \quad (3.13)$$

and

$$s_k^v(t) = \begin{cases} d_k \cdot \dot{l}_k(t) & \text{for } l_k(t) - l_k^0 > 0, \\ 0 & \text{else} \end{cases} \quad (3.14)$$

for the  $k$ -th cable connecting nodes  $k_1$  and  $k_2$  with coordinates  $\boldsymbol{\xi}_{k_1}$  and  $\boldsymbol{\xi}_{k_2}$ . The elastic stiffness of the cable is defined as

$$k_k^e = \frac{E^c A^c}{l_k^0} \quad (3.15)$$

and its viscous damping can be defined proportionally to the cable stiffness as

$$d_k = \beta k_k^e. \quad (3.16)$$

The rest length is denoted as  $l_k^0$ , Young's modulus as  $E^c$ , cross section as  $A^c$  and proportional damping coefficient as  $\beta$ . Cable's current length is expressed as the Euclidean norm

$$l_k(t) = \|\boldsymbol{\xi}_{k_1}(t) - \boldsymbol{\xi}_{k_2}(t)\|. \quad (3.17)$$

Its time derivative  $\dot{l}_k(t)$  expresses the magnitude of relative velocity between corresponding nodes.

Other approaches to modelling cables can be found for instance in [4]. Incorporation of cable forces  $s_k^e$  and  $s_k^v$  into the generalized force vector  $\mathbf{f}$  is dependent on particular definition of generalized coordinates through nodal positions  $\boldsymbol{\xi}_i(\boldsymbol{\rho}_v, \boldsymbol{\varphi}_v)$ , thus, it is not further analysed.

This work is not focused on the precise mathematical description of multibody system dynamics since its objectives are different. Nevertheless, there is a number of publications dealing with this issue, for instance [21], [22] or [27]. In the first mentioned paper, a detailed dynamic analysis of tensegrity systems of the class 1 is performed.

### 3.2. Computational model of tensegrity structure

Lots of software has been developed in past years in order to model the dynamics of multibody systems such as Adams, Simpack, RecurDyn, etc. In this work, Simscape is used for this purpose. It is an extension package of MATLAB-Simulink, it uses the same block environment, and, moreover, both standard Simulink and special Simscape blocks can be combined together. Simscape enables modelling and simulating multi-domain systems, this thesis exploits only its subpackage *Multibody* with the focus on multibody systems from the field of solid mechanics. As an illustration of the development environment, there is an example of Simscape model in form of a mechanical oscillator in Figure 3.1.

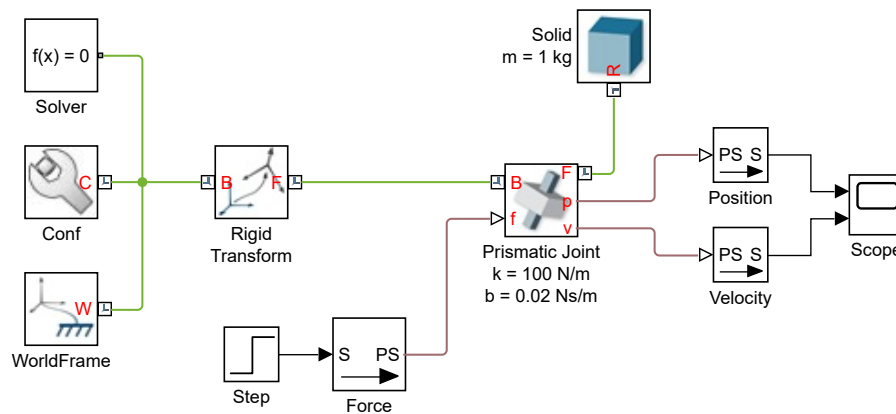


Figure 3.1.: Simscape model of a mechanical oscillator.

The principle of modelling in Simscape lies in quite intuitive connecting elementary blocks together imitating physical connections in the real modelled structure. Unlike

Simulink blocks, Simscape blocks have a definite physical meaning (bodies, joints, sensors and actuators, constraints and drivers, and force elements). Standard Simulink blocks have distinct input and output ports. The connections among those blocks (*signal lines*) represent inputs to and outputs from mathematical functions. This concept is not applicable to a mechanical system due to Newton's third law of action and reaction (if body  $A$  acts on a body  $B$  with force  $\mathbf{F}$ ,  $B$  also acts on  $A$  with a force  $-\mathbf{F}$ ) – there is no definite direction of signal flow. Special *connection lines* anchored at both ends to a *connection port* are introduced with Simscape [18].

As [18] pointed out, many commercial software packages for multibody dynamics use the formulation in absolute coordinates. Using this approach, each body is assigned 6 degrees of freedom. Then, depending on interaction of bodies due to joints, etc., constraint equations are formed. This results in a large number of configuration variables and relatively simple constraint equations, but also in a sparse mass matrix. Simscape uses a different strategy in form of relative coordinates. In this approach, a body is initially given zero degrees of freedom. Additional degrees of freedom are added by connecting joints to the body. Therefore, far fewer configuration variables and constraint equations are required. The drawback of this approach is the dense mass matrix containing the constraints implicitly, and more complex constraint equations.

Via Simscape, it is possible to create multi-layer models which improves clarity and user orientation. It also enables creation of own components and adding them into an own library which speeds up subsequent modelling. A great advantage for purposes of this work is its natural collaboration with “classic” MATLAB workspace. It is possible to rapidly exchange various data in form of variables, to execute MATLAB scripts within the model, or to edit the model in the scripting environment. Other features and benefits can be found in [7]. The last mentioned feature is fully utilized in this work in form of automatic generation of Simscape models of tensegrity structures introduced in section 3.2.3.

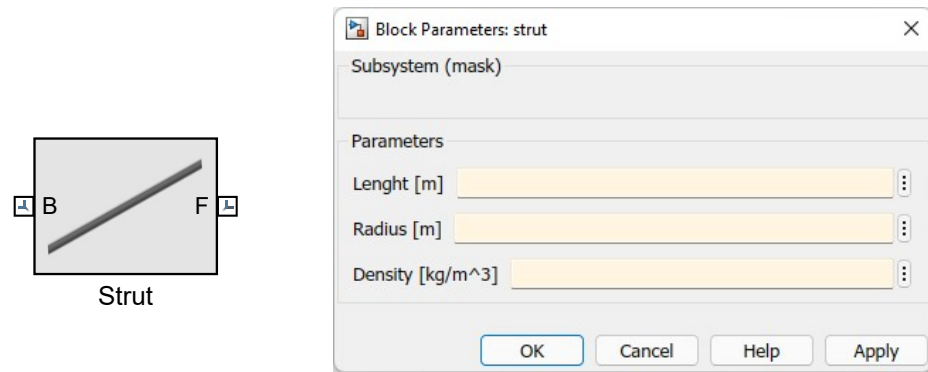
#### 3.2.1. Models of strut and cable

Two main components used in modelling tensegrities are naturally a strut and a cable. Their implementation and features are described in the following text.

##### **Strut**

The model of a strut is covered in a block *Strut*. There are two connection ports (imitating strut's nodes), through which the block can be connected to other elements. Strut's properties (length, radius, mass density) can be easily edited in a so-called mask (opening by clicking on the block). The block and the mask are visualized in Figure 3.2.

This component is modelled simply, see Figure 3.3. It is a homogeneous rigid cylinder represented by the block *Solid*. Strut's ends (two nodes of the tensegrity) are realized by *Connection Port* blocks.



(a) Block.

(b) Mask.

Figure 3.2.: The *Strut* block and its mask.

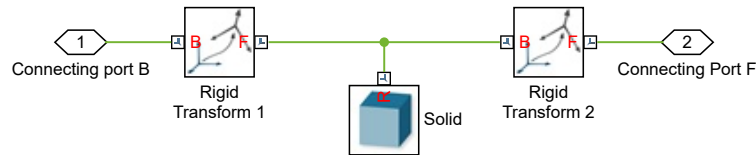
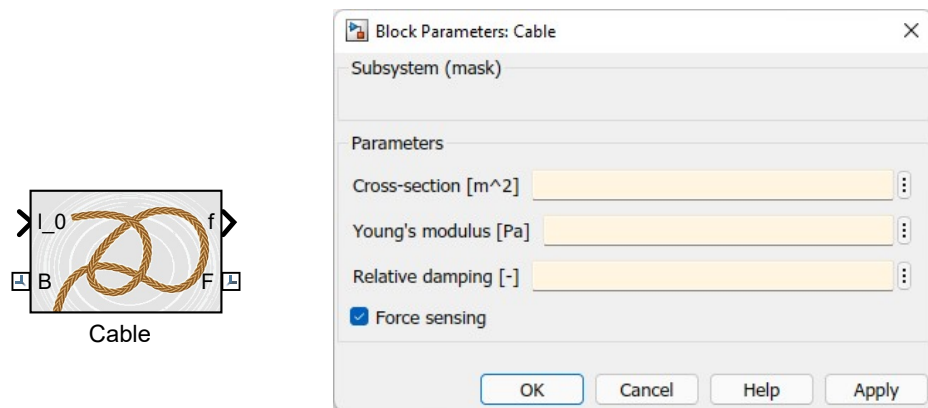


Figure 3.3.: Model of a strut.

### Cable

Similarly to the *Strut* component, a cable component has its own covering block with two connection ports, it is called the *Cable*. There is an *Outport*, through which it is possible to track current value of the cable's internal force, and an *Inport* enabling continuous changes of the cable's rest length which is an important feature for structure actuation. Cable's properties (cross section, Young's modulus, mass density, relative damping) and force sensing can be easily edited in a mask. The block and the mask are shown in Figure 3.4.



(a) Block.

(b) Mask.

Figure 3.4.: The *Cable* block and its mask.

As it was already mentioned in previous chapters, a cable represents a viscoelastic force between two structure nodes. This is realized in the model by a block *Internal Force* acting between two blocks *Connection Port*. The elastic component of the force is defined in (3.13) and the viscous component in (3.14). There is a necessary component *Transform Sensor* implemented to the model that provides information about relative velocity and distance between two nodes (*Connection Ports*). A force sensor continuously exporting the information about the current cable force is implemented by a block *Outputport* and the rest length is provided to the model via a block *Inport*. The block realization of mentioned points is depicted in Figure 3.5.

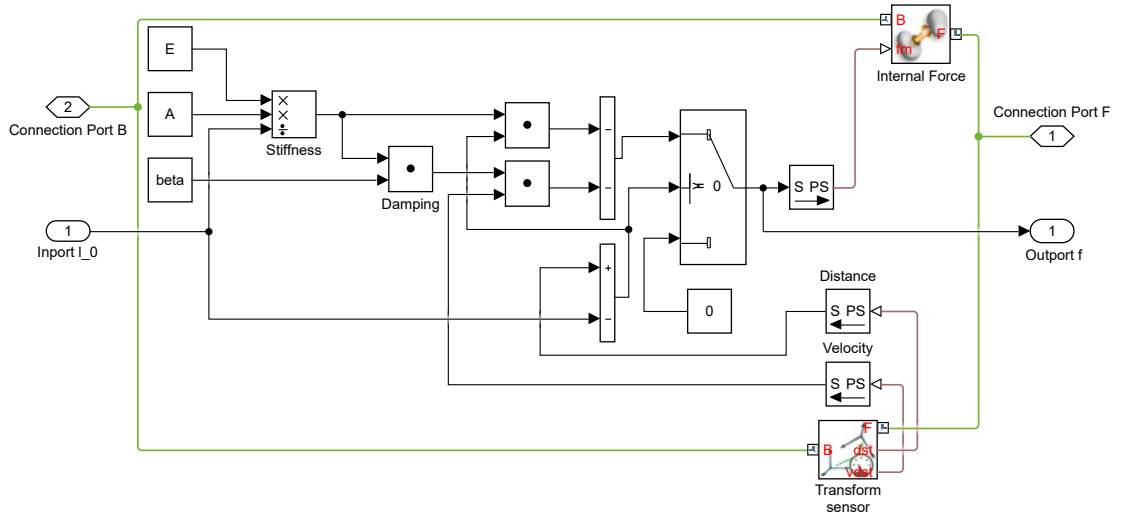


Figure 3.5.: Model of a cable.

### 3.2.2. Tensegrity model building

This part is dedicated to manual creation of 3-D tensegrity models (2-D models captured as well) of any class. The resulting model simulates the real structure lying on an immobile flat rigid surface.

#### Elementary blocks

There are three standard blocks that have to be contained in every Simscape model. The *Solver* defining numerical solver settings, the *Mechanism Configuration* used mainly for setting gravitational acceleration properties, and the *World Frame* representing the global coordinate system origin. These base blocks must be connected together and all physical components (struts and cables) must be connected to them (directly, or indirectly through other components). Regarding gravitational acceleration setting, it is assumed to act in the negative direction of the axis  $z$ .

The base of every Simscape multibody model is formed by solid bodies. Each body has to be rigidly connected to a frame (local/global) with the origin in the mass center.

Translation and/or rotation of one frame relative to another frame is done through the *Rigid Transform* block. In this moment, it is good to come back to the model of a strut depicted in Figure 3.3. The block *Solid* inherently contains a local frame with the origin in the mass centre and with the axis  $z$  oriented in the direction of the strut's symmetry axis. Because it is intended to connect other struts or cables to its ends (not to the mass centre), two *Rigid Transform* blocks have to be used realizing translation by half the strut's length on both sides.

If only *Rigid Transform* blocks are used for the transition between two frames, it means that these frames are fixed to each other. When it is intended to release some degrees of freedom, specific constraint blocks have to be used. There are lots of different constraints available in the Simscape library, but this work utilizes only two of them. The *Spherical Joint* enables relative rotation of the frames in all directions, and the connection between two frames can be completely interrupted by using the *6-DOF Joint*. This type of constraint also offers a possibility to define required motion in any direction which enables creation of a planar constraint by prescribing zero relative motion in some direction. Let this special type of the *6-DOF Joint* call the *Planar Joint* for better distinction in further text.

### Building process

The main task is to place all struts in correct positions and to attach them correctly to adjacent struts or to the *World Frame*. After that, adding cables is quite simple. In Figure 3.6, there is a 2-D example tensegrity structure of the class 2 (placed in the plane with  $y = 0$ ), on which the model building procedure is illustrated.

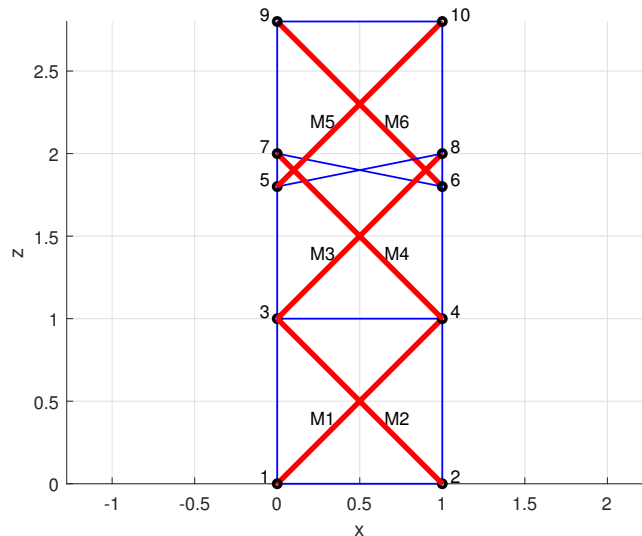


Figure 3.6.: A 2-D example structure.

Placement of struts always starts from the bottom of the structure. Struts, which nodes are located in the lowest  $z$ -level, are the ones touching the surface (represented by the *World Frame*). There is always one node located in the lowest  $z$ -level that is connected to

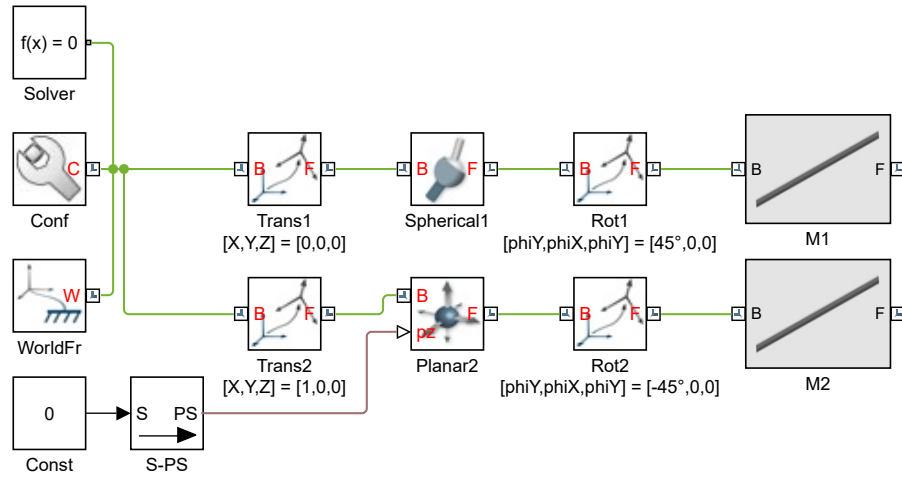


Figure 3.7.: The example structure model in progress (1).

the *World Frame* by a *Spherical Joint* and other nodes in the lowest  $z$ -level are connected to the *World Frame* by a *Planar Joint* fixing the node's motion in the direction  $z$ . This imitates the situation that the structure is lying on the surface with fixed position of one node. To place the struts with mentioned nodes in the correct position, it is necessary to use the *Rigid Transform* block, whereas the order of the individual operations must be taken into account.<sup>6</sup> First, the strut is shifted to the correct place defined by its node's position, then, an appropriate constraint is used, and, finally, rotation<sup>7</sup> with the center in the node is realized. In Figure 3.7, there is a particular realization of topics discussed in this paragraph.

In case of tensegrities of higher class, there is one or more nodes with more than one struts, therefore, there must be a settled procedure how to connect more struts together. Let the strut already existing in the model be labeled as the base and the strut being added to the model as the follower. The basic idea is to connect the base and the follower by a *Spherical Joint* and a *Rigid Transform* realizing the relative rotation. This procedure can become quite tricky in case of 3-D structures. The question is how to define relative angles of rotation of the follower when the base is already somehow rotated in the global frame. In this work, a simple solution in form of two separate rotations is used. First, the rotation neutralizing<sup>8</sup> the rotation of the base is realized. After that, it is possible to perform the second rotation realizing the absolute rotation between the *World Frame* and the follower. In case of more than two struts connected in one node, this procedure is repeated (the same base with a different follower). See the example structure model with two class-2 nodes 3 a 4 processed in Figure 3.8.

<sup>6</sup>The order is important only in case of the *Planar Joint* due to correct orientation of the fixing plane.

However, the procedure is the same for all types of constraints to maintain consistency.

<sup>7</sup>The rotation is realized using Euler angles – a sequence of three elementary rotations: precession around the axis  $y$ , nutation around the  $x$  axis and spin around the  $y$  axis. Since struts are axially symmetric, spin is not needed.

<sup>8</sup>The rotation is neutralized when the reverse sequence of elementary rotations is used together with opposite values of elementary angles.



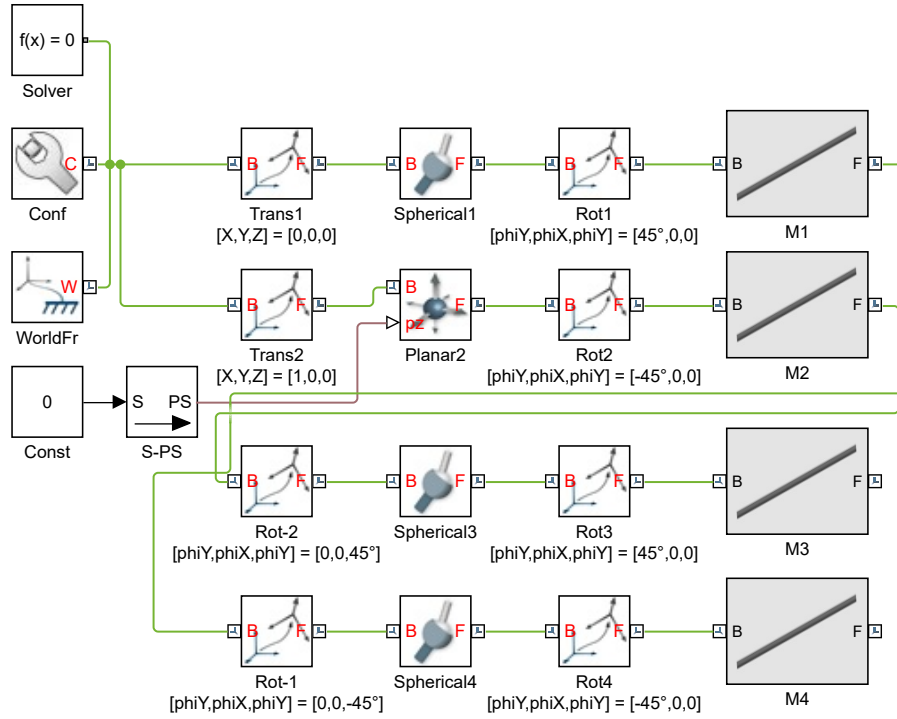


Figure 3.8.: The example structure model in progress (2).

Another topic, that has to be discussed, is the situation when a separate set<sup>9</sup> of struts needs to be added to the model. First, it has to be determined which node of which strut is going to be added to the model as the first one from the set. In this work, the node with the lowest  $z$ -coordinate from the set is chosen. The corresponding strut is then shifted to the correct place defined by chosen node's position, a *6-DOF Joint* is added between the node and the *World Frame*, and, finally, the strut is rotated with the rotation center in the node. After that, remaining struts from the set are added as described in the previous paragraph. In case of the example structure, there are two separate sets remaining (each one containing only one strut,  $M5$  and  $M6$ ). Processed Simscape model is shown in Figure 3.9.

The last discussed issue related to struts' placement is the problem of loops. Using procedures described in previous paragraphs, all struts can be added to the model (even those struts supposed to create a loop) but there is no procedure assuring kinematic closure of possible loops. The loop closure is done by addition of a *Spherical Joint* between ends of struts that are supposed to be connected but they have not been, yet. Generally in Simscape, joints of all types can be placed between two frames with the same position and orientation. Local frames of struts' ends are already in the same place, but they are probably rotated differently. Therefore, it is necessary to add another block of *Rigid Transform* connected to both struts' ends realizing the rotation to the same position. There are no struts creating a loop in the example structure in Figure 3.6, thus,

<sup>9</sup>A separate set of struts represent struts connected together only by *Spherical Joints* (e.g. a chain of 4 struts connected in 3 nodes) and none of these struts has been added to the model, yet.

### 3. Dynamics of tensegrity systems

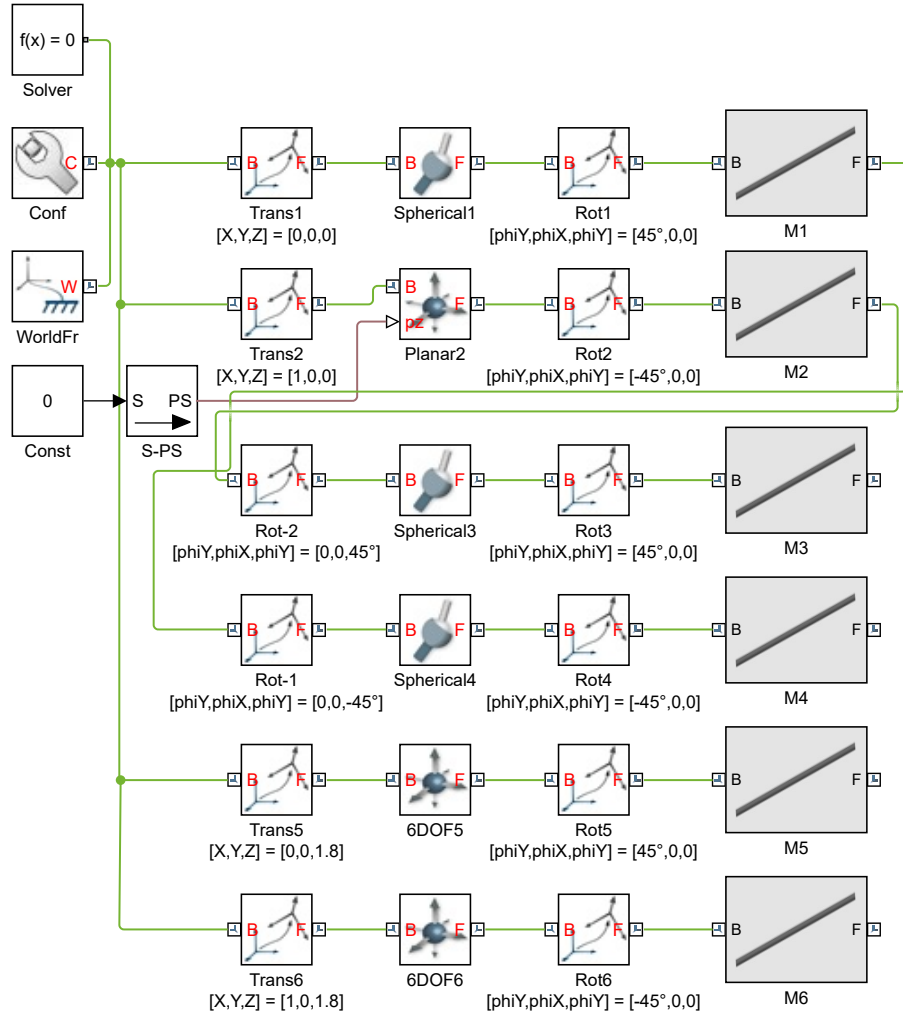


Figure 3.9.: The example structure model in progress (3).

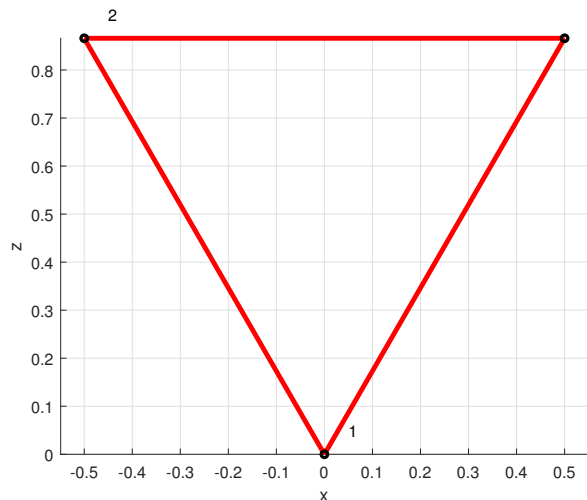


Figure 3.10.: A 2-D example structure with a loop.

a primitive structure (not a tensegrity) consisting of three struts forming one loop is shown in Figure 3.10. In Figure 3.11, there is its Simscape model, where the loop closure is performed in the node 3, while blocks realizing the closure are highlighted.

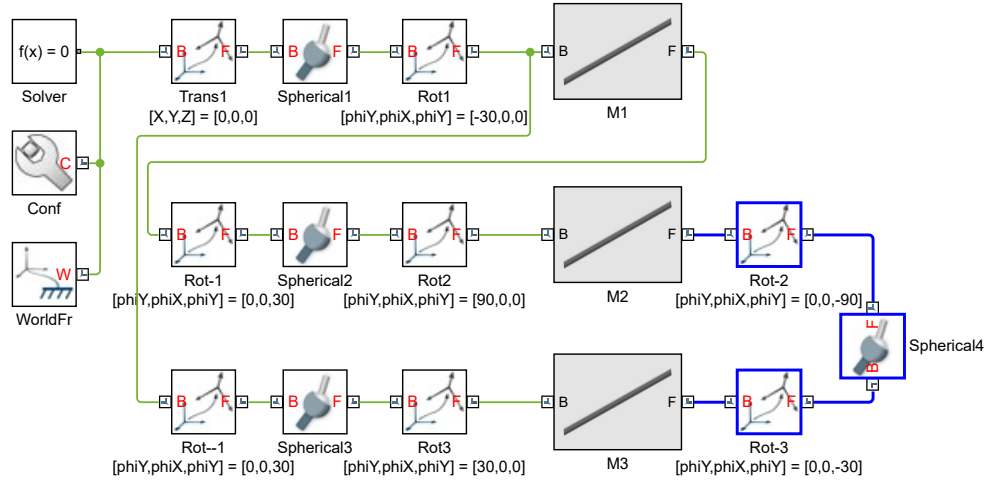


Figure 3.11.: The model of the example structure with a loop.

To finalize the model of a tensegrity, cables are put between appropriate nodes. No other blocks are needed. Complete model of the example tensegrity from Figure 3.6 is depicted in Figure 3.12, where connection lines connecting a cable with nodes 6 and 7 are highlighted for illustration.

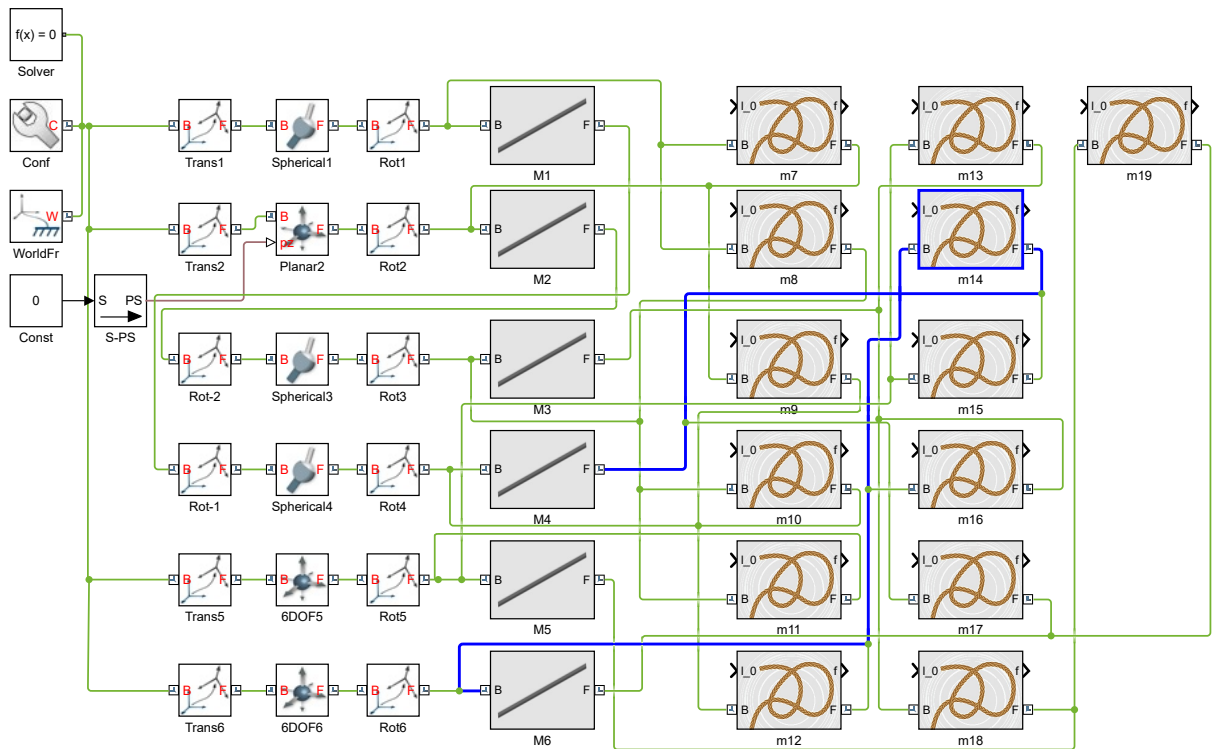


Figure 3.12.: The complete example structure model.

The tensegrity model itself is now ready. Last step necessary for running the simulation successfully is to provide inputs to all cables in form of rest lengths  $l_k^0$  (which can be constant or variable in time) and to set the numerical ODE (ordinary differential equations) solver properly. In Simulink settings, various ODE solvers are divided into two groups: solvers with fixed time step and with variable time step. Simulink offers a possibility to choose the solver automatically, however, two particular solvers turned out to be quite efficient – *ode15s* and *ode45*. The first mentioned one is also recommended by [7] as an efficient solver for stiff problems. Last but not least, extremely important is the time step setting. In default, its maximal size is selected automatically, but it is highly recommended to set it manually to  $10^{-3}$  s or less. Exploited settings are available in Appendix A.1.

Simscape also provides visualization of the model both in its initial state and during the simulation. Unfortunately, it is not possible to easily visualize cables since they are massless (and shapeless) and represented only by forces. Nevertheless, it is still possible to visualize them using the standard MATLAB scripting environment as a part of simulation post-processing. See Simscape visualization of the example tensegrity in Figure 3.13.

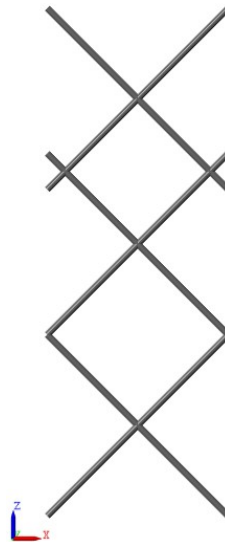


Figure 3.13.: Simscape visualization of the example structure.

### 3.2.3. Automatic model generation

Manual building process of a computational model can become quite time consuming in case of a complex structure, however, it still follows the same rules described in the previous section. Therefore, it is appropriate to use process automation. The aim is to implement a MATLAB function (in the scripting environment) that generates a Simscape model of a tensegrity structure based on provided inputs defined in the following text.

This section contains only the basic information about expected inputs and utilized MATLAB library functions. The text does not clarify the question of algorithmization,

which is quite complex. In case of the interest in particular implementation, see Appendix A.2 containing the developed function.

### Required inputs

Automatic generation of a computational tensegrity model is based on processing of the following data: the connectivity matrix  $\mathbf{C} \in \mathbb{R}^{m \times n}$ , the configuration vector  $\mathbf{X}^0 \in \mathbb{R}^{3n}$  with initial coordinates of all structure nodes, the force density vector  $\mathbf{q}^0 \in \mathbb{R}^m$  with initial values of members' force densities, the vector **type**  $\in \mathbb{R}^m$  containing boolean information about members' types (0 – strut, 1 – cable) and properties of particular struts (mass density  $\rho^s$ , cross-section  $A^s$ ) and cables (cross-section  $A^c$ , Young's modulus  $E^c$ , proportional damping coefficient  $\beta$ ).

The implementation is also ready for possible actuation of cables defined by the path-planning process, which is presented in chapter 4.1. Therefore, two additional inputs have to be specified: discrete sets  $\mathbf{I}^0(\varkappa T) \in \mathbb{R}^{m^c}$  of rest lengths and a time period  $T$  of transition between sets  $\mathbf{I}^0(\varkappa T)$  and  $\mathbf{I}^0(\varkappa T + T)$  together representing piecewise linear time functions of cables' rest lengths. The meaning of  $\varkappa$  and other details regarding the topic of tensegrity actuation can be found in chapter 4.

### Elementary functions

There are several functions in the MATLAB library that create, modify, save, or close a Simscape model. Here is the list of functions that are used in this work:

- *new\_system*(filename) – creates a new model file “filename.slx”
- *open\_system*(filename) – opens the model “filename.slx”
- *save\_system*(filename) – saves the model “filename.slx”
- *close\_system*(filename, saveflag) – closes the model “filename.slx” with or without saving depending on the boolean variable “saveflag”
- *add\_block*(source, dest) – adds a Simulink/Simscape block with the library path “source” to the destination path<sup>10</sup> “dest”.
- *delete\_block*(dest) – deletes a Simulink/Simscape block with the destination path “dest”.
- *add\_line*(sys,out,in) – creates a connection from the output “out” to the input “in” between blocks located in the system path<sup>11</sup> “sys”.
- *delete\_line*(sys,out,in) – deletes a connection from the output “out” to the input “in” between blocks located in the system path “sys”.

<sup>10</sup>The destination path contains information about the model and its specific layer (subsystem) where the block is going to be located. The destination path ends with the block's unique name.

<sup>11</sup>The system path is the same as the destination path except its ending – the block's name is missing. Blocks' names are specified by output/input.

### Resulting model features

Implemented function in Appendix A.2 is able to automatically create models of tensegrity structures of any class containing arbitrary number of struts and cables (their features have been already described in the previous text). The whole model is packed into a subsystem for better manipulation with the resulting model and the parametrization of the model is saved into a .mat file in order to be able to change tensegrity properties easily. The model also enables tracking of nodal positions and cables' forces. In default, positions are saved as array variables into a workspace after simulation, and forces can be watched during the simulation via a *Scope* block, but this default setting can be easily changed manually according to user's needs.

A generated model corresponding to the example structure from Figure 3.6 is depicted in Figure 3.14. The subsystem is unpacked. The whole model is not captured due to its size, but its visible part is sufficient as an illustration of appearance of resulting models.

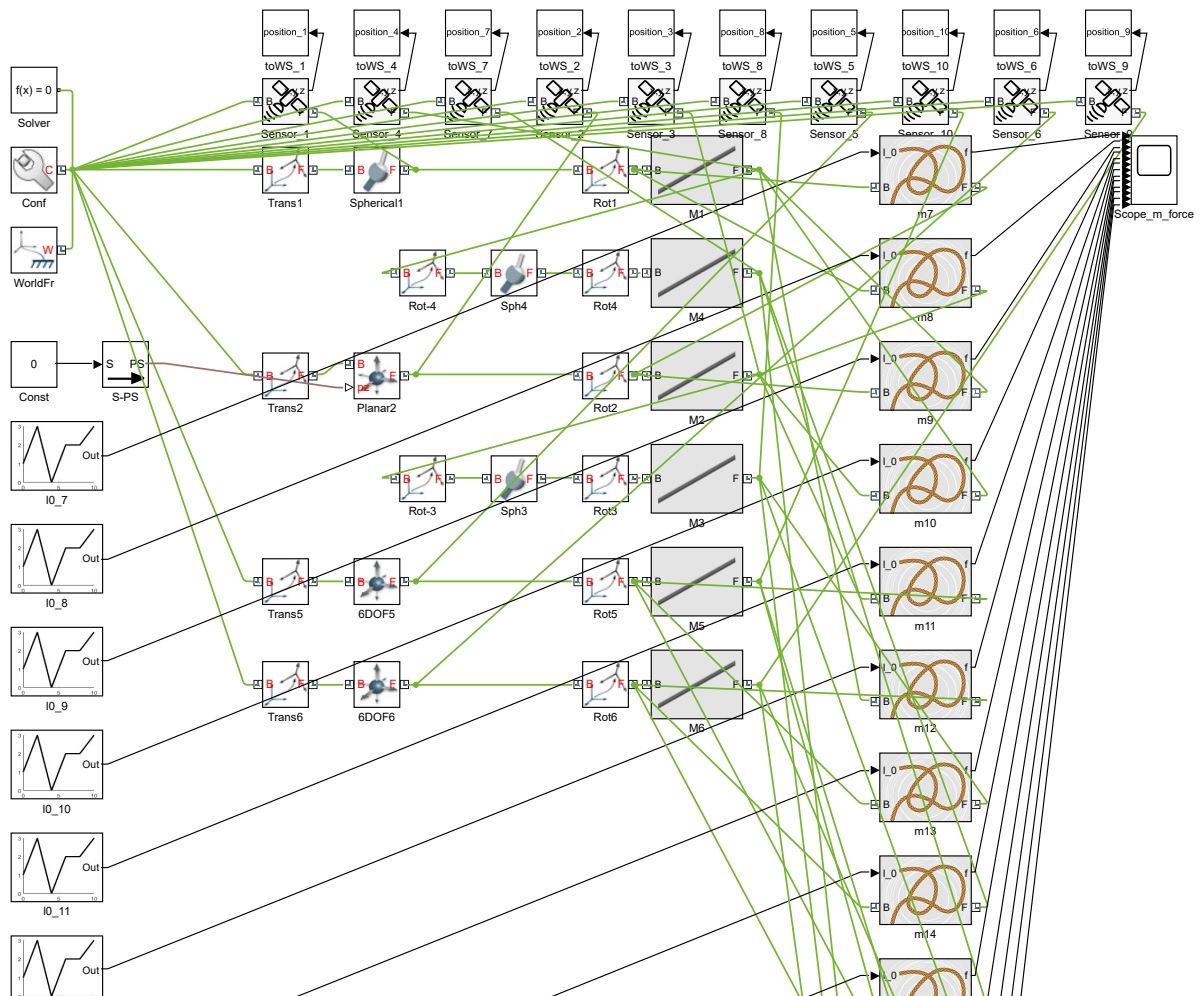


Figure 3.14.: Automatically generated model of the example structure.

### 3.3. Modal analysis of computational model

Modal analysis is one of standard tasks of mechanical structure dynamics. It is an operation leading to determination of eigenvalues and eigenvectors of examined linearized system. Based on this information, it is possible to obtain system's eigenfrequencies and corresponding mode shapes. Because a tensegrity structure is generally prone to vibrate, knowledge of its eigenfrequencies is essential mainly with respect to design of a controller regarding this work.

#### 3.3.1. The eigenvalue problem

A linear time-invariant discrete mechanical system can be expressed by a motion equation

$$\mathbf{M}\ddot{\mathbf{q}}(t) + \mathbf{B}\dot{\mathbf{q}}(t) + \mathbf{K}\mathbf{q}(t) = \mathbf{f}(t), \quad (3.18)$$

where  $t$  is time,  $\mathbf{q} \in \mathbb{R}^{n^{dof}}$  is the vector of generalized coordinates,  $\mathbf{f} \in \mathbb{R}^{n^{dof}}$  denotes the vector of generalized forces, and  $\mathbf{M}, \mathbf{B}, \mathbf{K} \in \mathbb{R}^{n^{dof} \times n^{dof}}$  are the mass, damping, stiffness matrix. Number of degrees of freedom is denoted as  $n^{dof}$ . In case of a weakly damped system, modal analysis is performed with associated conservative homogeneous system. However, modal analysis can be also performed with the original non-conservative system. According to [8], the homogeneous version of the equation (3.18) and a trivial identity

$$\mathbf{M}\dot{\mathbf{q}}(t) - \mathbf{M}\dot{\mathbf{q}}(t) = \mathbf{0} \quad (3.19)$$

can be written together in a matrix form

$$\begin{bmatrix} \mathbf{B} & \mathbf{M} \\ \mathbf{M} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}(t) \\ \ddot{\mathbf{q}}(t) \end{bmatrix} - \begin{bmatrix} -\mathbf{K} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \mathbf{q}(t) \\ \dot{\mathbf{q}}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \quad (3.20)$$

which can be expressed in a compact form

$$\mathbf{N}\dot{\mathbf{x}}(t) - \mathbf{P}\mathbf{x}(t) = \mathbf{0}, \quad (3.21)$$

where  $\mathbf{x} \in \mathbb{R}^{2n^{dof}}$  is the state vector. The state vector derivative can be expressed from equation (3.21) as

$$\dot{\mathbf{x}}(t) = \mathbf{N}^{-1}\mathbf{P}\mathbf{x}(t) \quad (3.22)$$

and then rewritten using the system matrix  $\mathbf{A} \in \mathbb{R}^{2n^{dof} \times 2n^{dof}}$  as

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t). \quad (3.23)$$

In accordance with [8], the solution to the equation (3.23) is expected in form of

$$\mathbf{x}(t) = \mathbf{u}_\nu e^{\lambda_\nu t}. \quad (3.24)$$

After substituting  $\mathbf{x}$  and  $\dot{\mathbf{x}}$  by (3.24) and its time derivative, equation (3.23) can be expressed as

$$(\lambda_\nu \mathbf{I} - \mathbf{A})\mathbf{u}_\nu = \mathbf{0}, \quad (3.25)$$

where  $\mathbf{I} = \text{diag}\{1, 1, \dots, 1\} \in \mathbb{R}^{2n^{dof} \times 2n^{dof}}$ . Equation (3.25) is a standard form of the eigenvalue problem, where  $\lambda_\nu \in \mathbb{C}$  represents the  $\nu$ -th eigenvalue and  $\mathbf{u}_\nu \in \mathbb{C}^{2n^{dof}}$  denotes the corresponding eigenvector.

The eigenvalue problem (3.25) is solved by  $2n^{dof}$  complex non-trivial eigenvectors and complex eigenvalues. Eigenvalues can be sorted into three groups:

$$\lambda_\nu = \alpha_\nu + i\beta_\nu, \quad \nu = 1, 2, \dots, k, \quad (3.26a)$$

$$\lambda_\nu = \alpha_\nu - i\beta_\nu, \quad \nu = k + 1, k + 2, \dots, 2k, \quad (3.26b)$$

$$\lambda_\nu = \alpha_\nu, \quad \nu = 2k + 1, 2k + 2, \dots, 2n^{dof}, \quad (3.26c)$$

where  $\alpha_\nu$  and  $\beta_\nu$  are the real and the imaginary part of the  $\nu$ -th eigenvalue. First, only eigenvalues from groups (3.26a),(3.26b) are focused. According to [23], the magnitude  $|\lambda_\nu|$  represents the eigenfrequency of the associated conservative system to (3.18)

$$\Omega_\nu = |\lambda_\nu|, \quad (3.27)$$

the real part  $\alpha_\nu$  defines the proportional modal damping parameter  $D_\nu$  as

$$D_\nu = -\frac{\alpha_\nu}{|\lambda_\nu|}, \quad (3.28)$$

and, finally, the imaginary part  $\beta_\nu$  stands for the non-zero eigenfrequency

$$\Omega_\nu^D = \beta_\nu \quad (3.29)$$

of the damped system (3.18). With real eigenvalues from the group (3.26c), it is possible to define overdamped and zero eigenfrequencies.

### 3.3.2. Eigenfrequencies of the computational model

The model of a tensegrity created in Simscape is non-linear. Therefore, it must be linearized to be able to perform modal analysis. The topic of discrete model linearization is discussed in the next chapter in section 4.2.1. However, Simscape offers quite an interesting option of model linearization in a specified state, which usage is very simple. The product of Simscape linearization is in form of a linear time-invariant state-space model represented by matrices  $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$  (more about this form is mentioned in section 4.2.1).

For the purpose of modal analysis, only the system matrix  $\mathbf{A} \in \mathbb{R}^{2n^{dof} \times 2n^{dof}}$  is necessary. This matrix has a different form than the system matrix in (3.23) depending on the linearization performed by Simscape. This is due to different introduction of the state-space vector. Nevertheless, eigenfrequencies of the structure are independent from particular selection of the state-space vector. After using the MATLAB function  $\text{eig}(\mathbf{A})$  for calculation of eigenvalues (and eigenvectors) of  $\mathbf{A}$ , eigenfrequencies  $\Omega_\nu^D$  of the computational model are expressed according to the equation (3.29).



### 3.3.3. Example

Naturally, structural prestress has a significant effect on eigenfrequencies. Thanks to the simple procedure proposed in the previous section 3.3.2, this statement can be easily proved. The example structure is a 2-D tensegrity tower used as an example in previous form-finding sections extended by one floor and topped by a special termination, the top node of which represents an effector. The structure is depicted in Figure 3.15a.

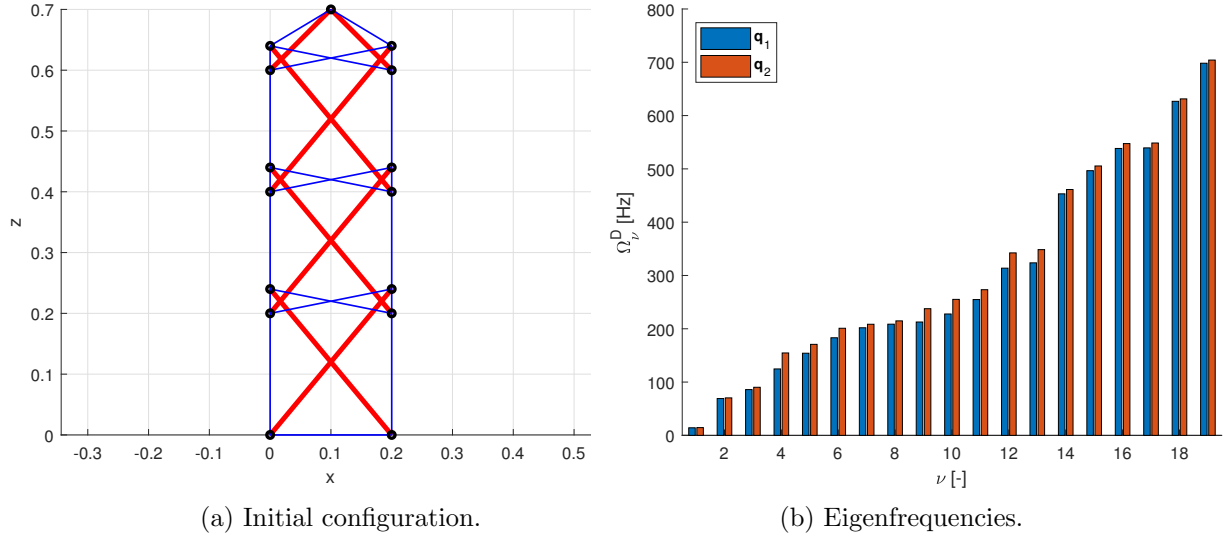


Figure 3.15.: A 2-D structure and its eigenfrequencies.

The structure showed in Figure 3.15a contains cables that are not pretensioned – their rest lengths are defined by the distance of two corresponding nodes. Cables are then shortened causing prestress in the whole structure. After that, the DRM described in section 2.2 is used to figure out the equilibrated configuration, in which the structure is then linearized. Finally, modal analysis is performed and eigenfrequencies are determined.

In the first case, each cable is shortened by 3% of its initial rest length which causes prestress represented by the force density vector  $\mathbf{q}_1$ . In the second case, cables are shortened by 6% causing force densities  $\mathbf{q}_2$ . Resulting eigenfrequencies<sup>12</sup> of both cases are compared in Figure 3.15b. The structure has 19 degrees of freedom (in  $xz$  plane) which corresponds with the number of eigenfrequencies. The comparison confirms a general mechanical rule – the more a structure is pretensioned, the higher eigenfrequencies the structure has.

<sup>12</sup>In fact, only a half of all eigenfrequencies is visualized. The second half of them does not bring any new information, since they are the same (complex conjugate eigenvalues).

## 4. Active tensegrity structures

Until this point, only passive tensegrity structures were examined. In this chapter, active tensegrities are introduced. There are various ways how to actuate tensegrity structures, e.g. changing rest lengths of elements via actuators. This work considers actuation in form of adjusting rest lengths of cables, all struts in the structure are still assumed passive.

First of all, it is necessary to determine the main goal of control. This work deals with tensegrity shape control. Let a set of desired trajectories be defined for one or more structure nodes. These nodes are supposed to follow desired trajectories with minimal deviations while the whole structure maintains its integrity during shape change. Usually, the desired trajectory is assigned only to some of the structure nodes, not to all of them. Therefore, one of control design crucial tasks is to find feasible trajectories of the remaining nodes to satisfy mentioned control goals (minimal deviations from desired trajectories, maintaining integrity). This problem is called path-planning.

### 4.1. Path-planning

In the beginning of this section, two terms are introduced to simplify the following text. Let master nodes be defined as structure nodes that are supposed to follow desired trajectories. On the other hand, slave nodes are the nodes without any assigned desired trajectory.

As mentioned in the introduction of this chapter, the term path-planning is used for the problem of searching for trajectories of slave nodes with the objective of minimizing the deviation of master nodes' trajectories from their desired trajectories and maintaining structural integrity. However, primary goal of path-planning is different. The goal is to design the actuation (adjustments of cables' rest lengths) that leads master nodes through desired trajectories (trajectories of slave nodes are only the result of designed actuation).

Evidently, an important role of path-planning represent positions  $\xi_i$  of master nodes specified by coordinates  $x_{i,d}$ . The index  $i$  denotes the  $i$ -th node from the set  $N^m$  of all  $n^m$  master nodes, and  $d$  is the particular direction. Positions of all master nodes can be summarized into the vector

$$\mathbf{Y} = [\dots, x_{i,1}, \dots, x_{i,2}, \dots, x_{i,3}, \dots]^T \in \mathbb{R}^{3n^m} \quad (4.1)$$

with analogous structure to the structure configuration vector  $\mathbf{X}$  introduced by (2.2) in chapter 2 but containing only master nodes. The vector  $\mathbf{Y}$  is called the configuration

vector of master nodes. If  $\mathbf{Y} = \mathbf{Y}(t)$ , then this vector function of time can be called the vector of master nodes' trajectories.

### Problem discretization

Significant simplification of the time-continuous problem of path-planning is realized by its discretization. Let a continuous desired trajectory be defined as

$$\boldsymbol{\xi}_i^{des} = \boldsymbol{\xi}_i^{des}(t) = \begin{bmatrix} x_{i,1}^{des}(t) \\ x_{i,2}^{des}(t) \\ x_{i,3}^{des}(t) \end{bmatrix} \in \mathbb{R}^3, \quad t \in \langle 0, t^f \rangle, \quad i \in N^m, \quad (4.2)$$

for the  $i$ -th node from the set  $N^m$  of master nodes, consisting of desired coordinates  $x_{i,d}^{des}$  for each direction. Independent variable is time  $t$  while initial time is assumed to be 0 and final time is denoted as  $t^f$ . All desired trajectories can be summarized into the time-variable vector

$$\mathbf{Y}^{des}(t) = [\dots, x_{i,1}^{des}(t), \dots, x_{i,2}^{des}(t), \dots, x_{i,3}^{des}(t), \dots]^T \in \mathbb{R}^{3n^m} \quad (4.3)$$

with the same structure as  $\mathbf{Y}(t)$  introduced in (4.1).

Each trajectory is sampled with a constant sampling period  $T$  into  $n^T + 1$  discrete desired positions

$$\boldsymbol{\xi}_i^{des}(\varkappa T) = \begin{bmatrix} x_{i,1}^{des}(\varkappa T) \\ x_{i,2}^{des}(\varkappa T) \\ x_{i,3}^{des}(\varkappa T) \end{bmatrix} \in \mathbb{R}^3, \quad \varkappa = 0, 1, \dots, n^T, \quad i \in N^m. \quad (4.4)$$

Similarly to (4.2) and (4.3), desired positions (4.4) for all master nodes are summarized into  $n^T + 1$  discrete desired configuration vectors

$$\mathbf{Y}^{des}(\varkappa T) \in \mathbb{R}^{3n^m}, \quad \varkappa = 0, 1, \dots, n^T. \quad (4.5)$$

Solution to the discretized problem of path-planning is represented by  $n^T + 1$  discrete sets of cables' rest lengths:

$$\mathbf{l}^0(\varkappa T) \in \mathbb{R}^{n^c}, \quad \varkappa = 0, 1, \dots, n^T, \quad (4.6)$$

where  $n^c$  is the number of cables in the structure.

#### 4.1.1. Optimization problem

One of possible ways how to deal with path-planning is to consider it as an optimization problem. This approach is utilized for instance in [26, 28, 5], however, each paper considers slightly different objectives of path-planning. Authors of [26] use nested optimization without applying the DRM method. Another paper, [5], deals with planning of a trajectory between a couple of relatively remote positions. It uses genetic algorithm

in combination with the DRM. Resulting optimized trajectories usually dispose of quite chaotic or random appearance.

In this work, discretized path-planning is solved by  $n^T + 1$  separate optimization iterations. Constrained optimization problem is formulated in each iteration ( $\varkappa = 0, 1, \dots, n^T$ ) as

$$\boldsymbol{\delta}^* = \arg \min_{\boldsymbol{\delta}} \Psi(\mathbf{X}, \mathbf{s}), \quad (4.7)$$

$$\text{s.t. } \mathbf{D}\mathbf{s} = \mathbf{F}, \quad (4.8)$$

$$\Delta_k^l \leq \delta_k \leq \Delta_k^u, \quad k \in K^c, \quad (4.9)$$

where  $\Psi$  denotes the objective function, optimization variables are adjustments  $\boldsymbol{\delta}$  of cables' initial rest lengths  $\overline{\mathbf{L}}^0$  ( $\delta_k$  and  $\overline{L}_k^0$  are values valid for the  $k$ -th cable from the set  $K^c$  of all  $m^c$  cables),  $\mathbf{D}$  is the equilibrium matrix defined in (2.15),  $\mathbf{s}$  is the vector of internal forces in structure elements and  $\mathbf{F}$  is the vector of external nodal loads. Variables  $\Delta_k^l \leq 0$ ,  $\Delta_k^u \geq 0$  denote lower and upper bounds of adjustments of cables' rest lengths summarized into vectors  $\boldsymbol{\Delta}^l$ ,  $\boldsymbol{\Delta}^u$ . Subsequently, the vector  $\mathbf{l}^{0,*}$  of optimized cables' rest lengths can be naturally calculated from optimized adjustments  $\boldsymbol{\delta}^*$  of cables' rest lengths as

$$\mathbf{l}^{0,*} = \overline{\mathbf{L}}^0 + \boldsymbol{\delta}^*. \quad (4.10)$$

As indicated in (4.7), the value of the objective function  $\Psi$  is dependent on the tensegrity configuration  $\mathbf{X}$ . Particular configuration is the result of specific selection of adjustments  $\boldsymbol{\delta}$ , or, more formally, the configuration must satisfy the static equilibrium condition expressed in (4.8) for chosen adjustments of rest lengths. At this point, the DRM, a very efficient tool described in chapter 2.2 providing statically equilibrated  $\mathbf{X}$  as the result of specific  $\boldsymbol{\delta}$ , is fully utilized<sup>13</sup>. Moreover, the DRM informs about internal forces  $\mathbf{s}$ , which are also used in the objective function. From the expression (4.9), it is clear that the adjustment  $\delta_k$  of the  $k$ -th cable's rest length must be from the corresponding interval  $\langle \Delta_k^l, \Delta_k^u \rangle$ .

The expression (4.7) is only a general notation of the minimization problem. The actual challenging part is determination of the objective function  $\Psi$  in combination with appropriate choice of the numerical optimization solver. The objective of the optimization is to minimize errors of master nodes' positions from desired positions in combination with maintaining the integrity of the structure. According to [26], structural integrity is guaranteed when all internal forces in cables are positive, i.e. all cables are in tension. Additionally, cables' internal forces should be in a feasible range due to both limited power of cables' actuators and preventing struts from buckling. Therefore, in order to reduce stress in the active structure and to prevent the structure from the integrity loss, undesirable increase or decrease in cables' internal forces between optimization iterations is minimized.

Based on requirements summarized in the previous paragraph, preliminary expectations of the objective function are that it must process the structure configuration  $\mathbf{X}$

---

<sup>13</sup>The usage of the DRM fully replaces the need for solving the system of equations (4.8).

(strictly speaking, only the master nodes' configuration  $\mathbf{Y}$ , which is contained in  $\mathbf{X}$ ) and cables' internal forces  $\mathbf{s}$ . This is already noted in (4.7). The appropriate selection of the objective function was the subject of considerable effort. The best results were achieved by specifying the problem as the multi-objective optimization in form of a *minimax* problem. The objective function is expressed as

$$\Psi = \max_{\gamma} \psi_{\gamma}, \quad (4.11)$$

where  $\psi_{\gamma}$  ( $\gamma = 1, 2$ ) are components of the vector function

$$\boldsymbol{\psi} = [\psi_1, \psi_2]^T \in \mathbb{R}^2. \quad (4.12)$$

Components of  $\boldsymbol{\psi}$  are specified as

$$\psi_1 = w^Y (\mathbf{Y} - \mathbf{Y}^{des})^T (\mathbf{Y} - \mathbf{Y}^{des}), \quad (4.13)$$

quantifying position errors of master nodes with the weight parameter  $w^Y$ , and

$$\psi_2 = w^P P + w^S \sum_{k \in K^c} |s_k - \bar{s}_k|, \quad (4.14)$$

where  $\bar{s}_k$  is the final value of the internal force in the  $k$ -th cable achieved in the previous optimization iteration. Regardless of  $P$  and  $w^P$ ,  $\psi_2$  quantifies differences in cables' internal forces with the weight parameter  $w^S$ . The penalty member  $P$  combined with the penalization weight  $w^P$  penalizes values of internal forces in cables lower than the threshold  $s^{min} > 0$  in form of

$$P = \sum_{k \in K^c} P_k, \quad (4.15)$$

where

$$P_k = \begin{cases} (s^{min} - s_k)^2 & s_k < s^{min}, \\ 0, & \text{otherwise.} \end{cases} \quad (4.16)$$

In many cases, multi-objective optimization process reaches the state that components  $\psi_{1,2}$  of objective function are contradictory – reducing one component inevitably results in increase of the other one. This is called the problem of Pareto optimality. Mutual settings of weight parameters  $w^Y$ ,  $w^S$  (partially  $w^P$ ) fundamentally affect which optimization objective is preferred. Theoretically, let a balanced setting be assumed for a case when  $w^Y$  is set to value  $W^Y$  and  $w^S$  to value  $W^S$ . This state results in balanced minimization of position errors and total difference of internal forces. If  $w^Y > W^Y$  and, simultaneously,  $w^S < W^S$ , it is clear that minimization of position errors is preferred. Analogously, the opposite case applies. When both weights are increased, i.e.  $w^Y > W^Y$ ,  $w^S > W^S$ , it could be intuitively said that more emphasis is placed on both objectives. Generally, it is true, but there is a danger of numerical process collapse, which grows with the growth of weights  $w^Y$ ,  $w^S$ . From given examples, it is evident that appropriate setting of weights represents an important stage of path-planning process tuning.

In this work, the function *fminimax* from the MATLAB *Optimization Toolbox* is applied as a numerical solver of the described optimization problem.

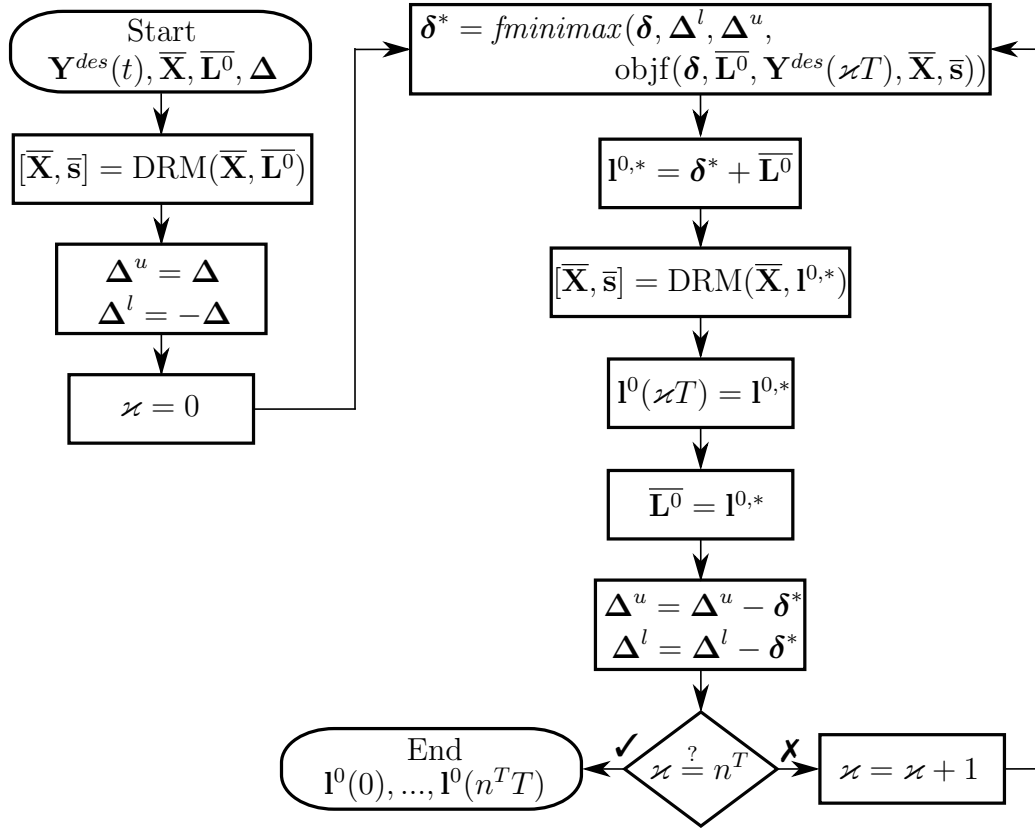


Figure 4.1.: Flow chart of the path-planning process.

### 4.1.2. Algorithmization

This section summarizes the numerical process of path-planning, which is visualized by a flow chart in Figure 4.1. Due to algorithm clarity improvement, there are marked only the most important variables from the algorithmization point of view (e.g. structure variables, process parameters, or solver settings are not depicted).

The first box of the flow chart includes input data of the process: desired trajectories  $\mathbf{Y}^{des}(t)$ , initial configuration  $\bar{\mathbf{X}}$  of the structure, initial values  $\bar{\mathbf{L}}^0$  of cables' rest lengths, and maximal allowed adjustments  $\Delta$  of cables' rest lengths (maximal shortening/lengthening).  $\bar{\mathbf{X}}$  and  $\bar{\mathbf{L}}^0$  can be advantageously determined by the AFDM described in chapter 2.1 – the output from the AFDM is in form of the super-stable configuration and force densities, which can be used to calculate cables' rest lengths. The first step is to calculate equilibrated configuration and equilibrated cables' internal forces from  $\bar{\mathbf{X}}$  and  $\bar{\mathbf{L}}^0$  applying the DRM. Equilibrated configuration overwrites the original initial configuration  $\bar{\mathbf{X}}$  because this configuration is used as the new initial configuration in following steps. Then, upper and lower bounds  $\Delta^u, \Delta^l$  of optimization parameters are set. The last step before entering the optimization cycle represents initialization of the iterator  $\varkappa$ .

The first step in the optimization cycle is the crucial one, optimization itself. The MATLAB function *fminimax* solving the optimization problem described in previous

section 4.1.1 is invoked<sup>14</sup>. Its first three inputs are the vector  $\boldsymbol{\delta}$  of optimization parameters, and upper and lower bounds  $\boldsymbol{\Delta}^u, \boldsymbol{\Delta}^l$  of optimization parameters. The following one is the reference “objf” to the objective function. There is a separate flow chart in Figure 4.2 dedicated to evaluation of the objective function. As the output, the function *fminimax* provides optimal values  $\boldsymbol{\delta}^*$  of adjustments of cables’ rest lengths  $\bar{\mathbf{L}}^0$ .

After the optimization, optimized adjustments  $\boldsymbol{\delta}^*$  are converted to optimized rest lengths  $\mathbf{l}^{0,*}$  of cables. Then, the DRM is applied in order to discover the equilibrated configuration and equilibrated cables’ internal forces achieved from the initial configuration  $\bar{\mathbf{X}}$  combined with optimized cables’ rest lengths  $\mathbf{l}^{0,*}$ . Original values of variables  $\bar{\mathbf{X}}, \bar{\mathbf{s}}$  are overwritten by new data again. In next two steps, optimized cables’ rest lengths  $\mathbf{l}^{0,*}$  are saved to  $\mathbf{l}^0(\varkappa T)$  representing output data, and they are also used as initial rest lengths  $\bar{\mathbf{L}}^0$  in the next optimization iteration. Finally, upper and lower bounds  $\boldsymbol{\Delta}^u, \boldsymbol{\Delta}^l$  are adjusted.

Optimization cycle is interrupted when all iterations are performed, otherwise, the path-planning process continues with increasing the iterator and performing another iteration. The result of path-planning represents the output in form of  $n^T + 1$  optimized sets of cables’ rest lengths:  $\mathbf{l}^0(0), \mathbf{l}^0(T), \dots, \mathbf{l}^0(n^T T)$ .

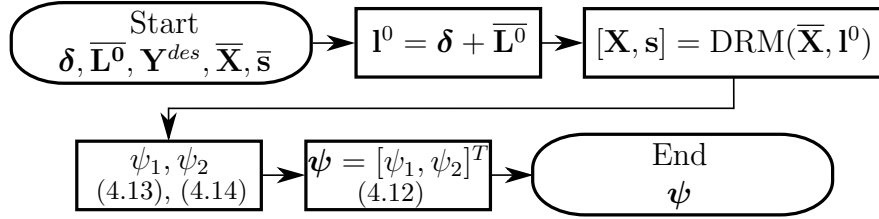


Figure 4.2.: Flow chart of the objective function  $\boldsymbol{\psi}$  evaluation.

As mentioned in the previous text, Figure 4.2 contains a flow chart visualizing evaluation of the objective function  $\boldsymbol{\psi}$ . Input data provided to the function “objf”, which is referenced during *fmincon* invocation (see Figure 4.1), are current values  $\boldsymbol{\delta}$  of optimization parameters, initial rest lengths  $\bar{\mathbf{L}}^0$  of cables, desired positions  $\mathbf{Y}^{des}$  of master nodes, the initial structure configuration  $\bar{\mathbf{X}}$ , and cables’ internal forces  $\bar{\mathbf{s}}$ .

First, adjustments  $\boldsymbol{\delta}$  are converted to rest lengths  $\mathbf{l}^0$  of cables. Then, the equilibrated configuration  $\mathbf{X}$  of the structure and equilibrated internal forces  $\mathbf{s}$  in cables are expressed by the DRM. After that, particular components  $\psi_{1,2}$  of the objective function are evaluated, and, finally, both components are returned as the output in form of the vector  $\boldsymbol{\psi}$ .

### 4.1.3. Examples

This section contains results of several calculations concerning two different structures (2-D and 3-D) showing functionality and generality of the proposed path-planning procedure. Values of path-planning parameters as well as members’ properties are available in Appendix A.1.

<sup>14</sup>The invocation of *fminimax* input parameters is not noted correctly according to MATLAB syntax – only a symbolical notation is used.

## 2-D structure

The first example structure is a 2-D tensegrity tower already used as an example in section 3.3 dedicated to modal analysis. The initial configuration of the structure is depicted in Figure 4.3.

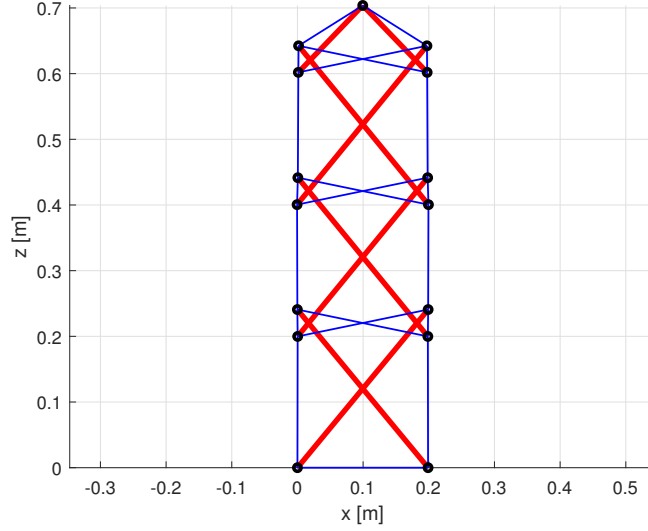


Figure 4.3.: A 2-D structure in the initial configuration.

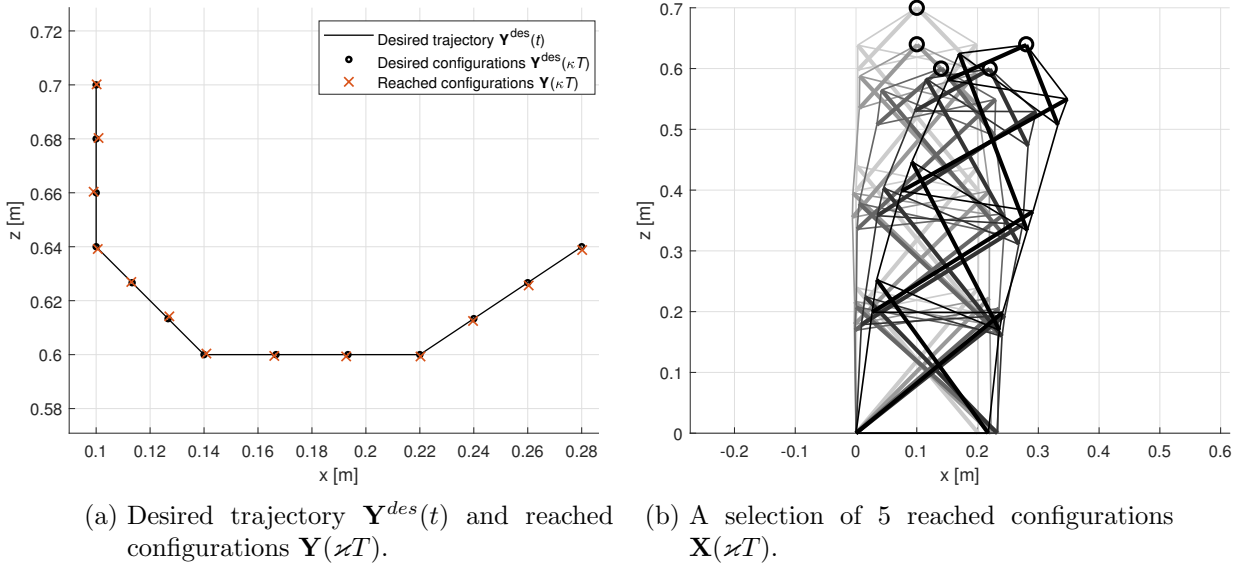


Figure 4.4.: The 2-D structure subjected to path-planning.

Two different desired trajectories are assigned to the top node (it is the only master node, thus,  $n^m = 1$ ). First, the focus is laid on a piecewise linear desired trajectory  $\mathbf{Y}^{des}(t)$  depicted in Figure 4.4a. For path-planning purposes, it is discretized into 13



( $n^T = 12$ ) separate points  $\mathbf{Y}^{des}(\varkappa T)$  that are also visualized in the figure. Resulting positions  $\mathbf{Y}(\varkappa T)$  of the top node for each desired configuration provided by the path-planning procedure are marked with red crosses in Figure 4.4a. Little deviations can be observed but the result can be definitely considered as sufficiently accurate. In the figure 4.4b, there are 5 resulting structure configurations  $\mathbf{X}(\varkappa T)$  visualized, specifically for  $\varkappa = 0, 3, 6, 9, 12$ , where  $\varkappa = 0, 12$  represent the first and the last path-planning iteration. Positions of the top node can be easily compared with desired positions represented by black circles.

The essential path-planning product is in form of discrete sets of cables' rest lengths  $\mathbf{l}^0(\varkappa T)$ . Based on them, a time-continuous actuation  $\mathbf{l}^0(t)$  can be produced by linear interpolation. It is clear that the sampling period  $T$  determines actuation speed and thus speed of resulting motion of the actuated structure. First, the parameter is set to  $T = 3$  s, corresponding actuation is denoted as  $\mathbf{l}^{0,A}(t), t \in \langle 0 \text{ s}, 36 \text{ s} \rangle$ , and it is visualized in Figure 4.5a.

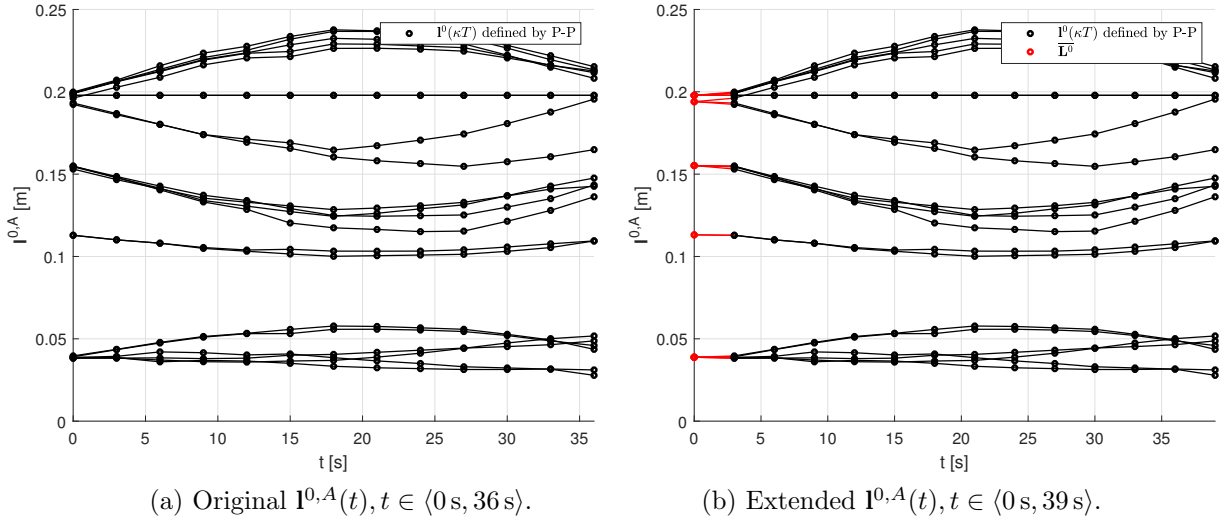


Figure 4.5.: Continuous actuation  $\mathbf{l}^{0,A}(t)$ .

In order to make the beginning of structure motion smooth, actuation  $\mathbf{l}^{0,A}(t)$  is shifted in time domain by one sampling period,  $t = t + T$ , and then extended with additional linear stage for  $t \in \langle 0, T \rangle$  defined by  $\mathbf{l}^{0,A}(0) = \overline{\mathbf{L}}^0$ , where  $\overline{\mathbf{L}}^0$  is the initial setting of cables' rest lengths in the beginning of the path-planning process (see Figure 4.1, box "Start"). Modified actuation  $\mathbf{l}^{0,A}(t), t \in \langle 0 \text{ s}, 39 \text{ s} \rangle$ , is depicted in Figure 4.5b.<sup>15</sup>

Figure 4.6 contains results obtained by a Simscape simulation of the discussed structure actuated by  $\mathbf{l}^{0,A}(t)$ . In Figure 4.6a, there is the trajectory  $\mathbf{Y}(t)$  of the top node compared to the desired trajectory  $\mathbf{Y}^{des}(t)$ . There are only minor observable deviations confirming discrete path-planning results from Figure 4.4. In Figure 4.6b, there are cables' internal forces  $\mathbf{s}(t)$  calculated by Simscape compared to linearly interpolated discrete sets of

<sup>15</sup>Actuation modification described in this paragraph is used in all following simulations without mentioning it again.

#### 4. Active tensegrity structures

cables' forces  $\mathbf{s}(\mathcal{z}T)$  from each iteration of path-planning. It can be observed that the linear approximation based only on path-planning data is pretty close to simulated values. Moreover, more important conclusion can be made: all forces are in a feasible range (maximal force around 600 N), integrity is maintained (minimal force around 80 N) and forces do not change during the process a lot.

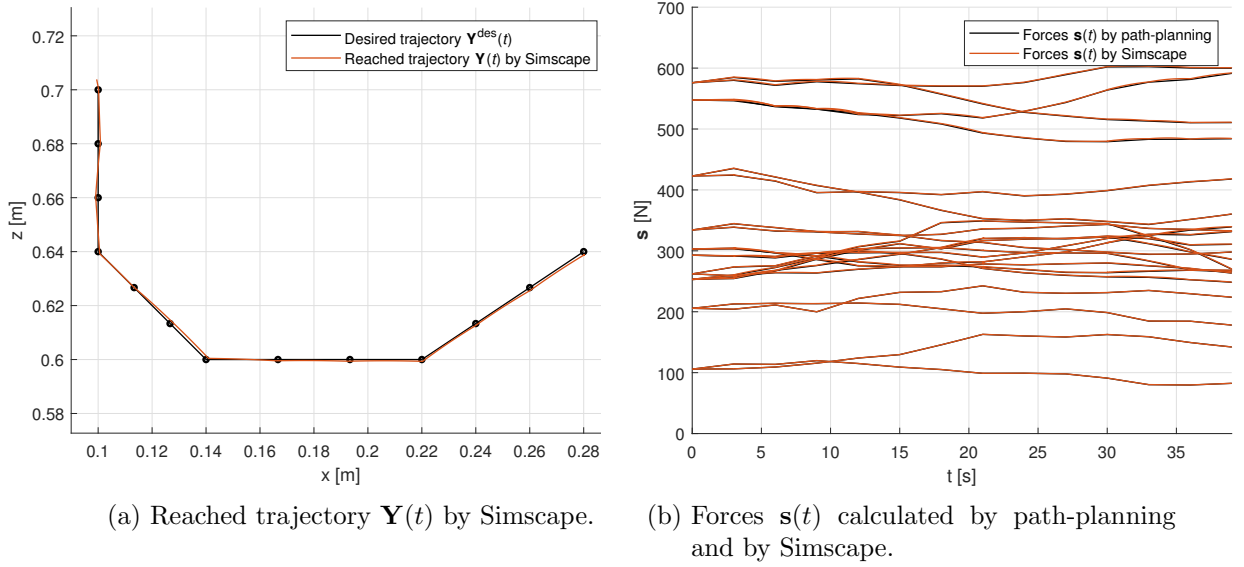


Figure 4.6.: A Simscape simulation of the 2-D structure with actuation  $\mathbf{I}^{0,A}(t)$  defined by path-planning.

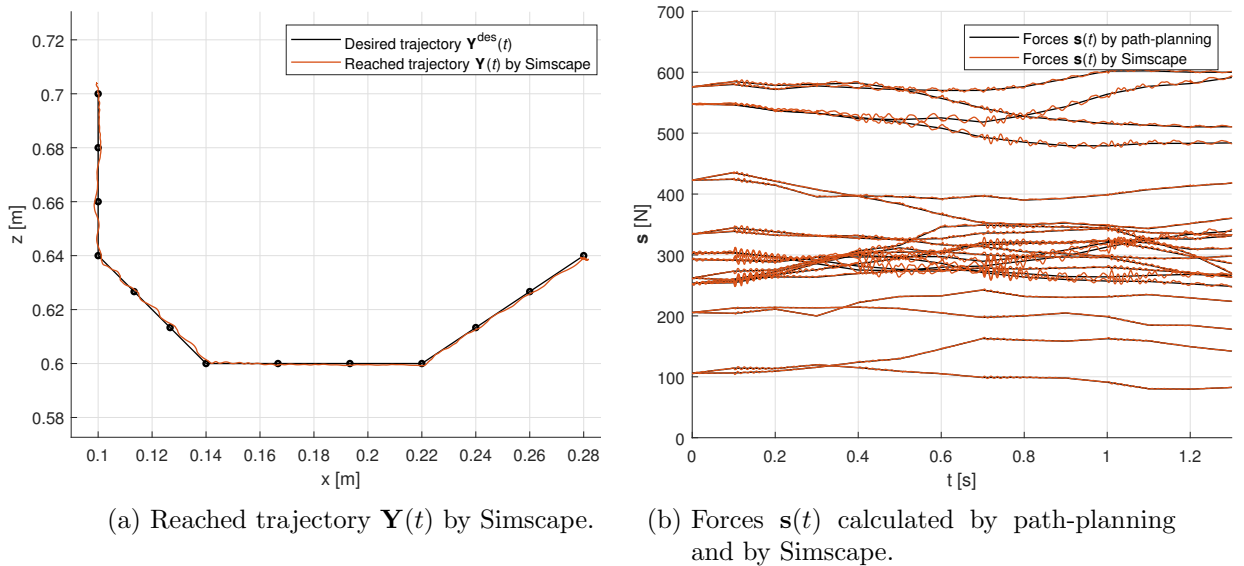


Figure 4.7.: A Simscape simulation of the 2-D structure with actuation  $\mathbf{I}^{0,B}(t)$  defined by path-planning.

The previous case with  $T = 3$  s represents a quasi-static process (the motion is relatively slow). Far more dynamic behaviour can be observed when the period is lowered to  $T = 0.1$  s defining the actuation  $l^{0,B}(t), t \in \langle 0 \text{ s}, 1.3 \text{ s} \rangle$ . Corresponding results are depicted in Figure 4.7. The calculated trajectory  $\mathbf{Y}(t)$  in Figure 4.7a still follows the desired trajectory  $\mathbf{Y}^{des}(t)$  but it is evident that the system is vibrating. This is confirmed by curves of cables' forces depicted in Figure 4.7b, which are oscillating too. The vibration is generated mainly by sharp changes in actuation, which are not smooth.

The second desired trajectory  $\mathbf{Y}^{des}(t)$  assigned to the top node is depicted in Figure 4.8a. It is composed of a linear part followed by a semicircle while the transition between these two curves is not smooth. The desired trajectory is discretized into 15 ( $n^T = 14$ ) isolated points  $\mathbf{Y}^{des}(\varkappa T)$ . Positions  $\mathbf{Y}(\varkappa T)$  of the top node reached in each iteration of the path-planning process are visualized in mentioned Figure 4.8a. As in the previous case, position errors are satisfying. Figure 4.8b shows 4 resulting structure configurations for  $\varkappa = 0, 2, 8, 14$ , while  $\varkappa = 0, 14$  represent the first and the last path-planning iteration.

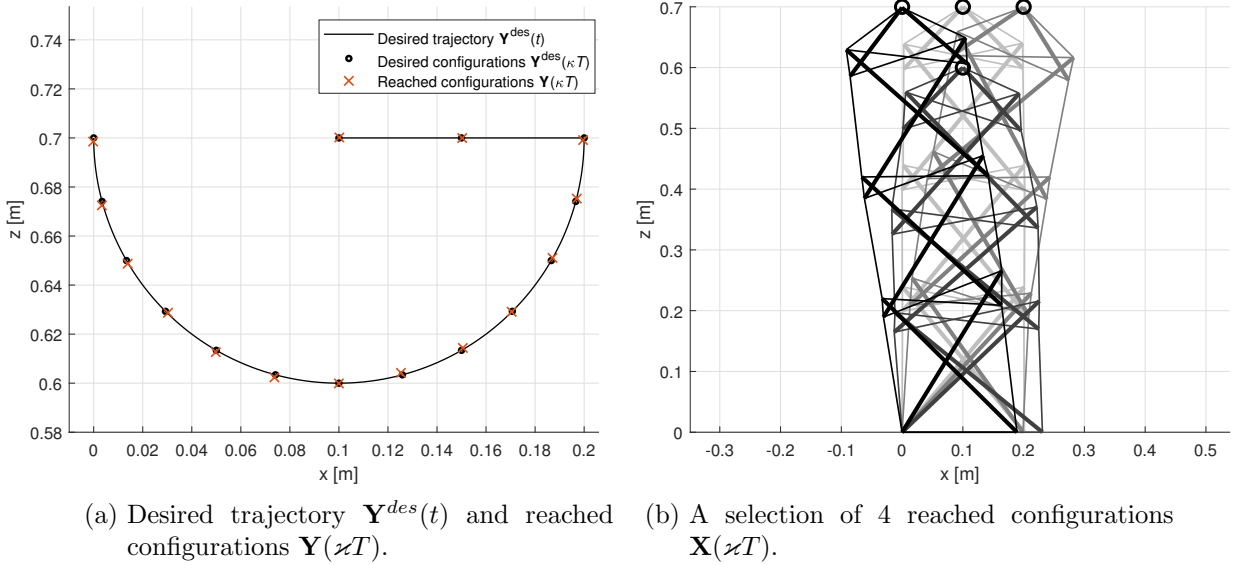


Figure 4.8.: The 2-D structure subjected to path-planning.

A dynamic simulation was performed for two cases again. First, with relatively high value of the sampling period  $T = 3$  s resulting in continuous actuation  $l^{0,A}(t), t \in \langle 0 \text{ s}, 45 \text{ s} \rangle$ . Corresponding results are shown in Figure 4.9. In the left figure, it can be seen that the calculated trajectory  $\mathbf{Y}(t)$  of the top node deviates from the desired trajectory  $\mathbf{Y}^{des}(t)$  mainly in the semicircle part between two adjacent desired path-planning configurations  $\mathbf{Y}^{des}(\varkappa T)$  and  $\mathbf{Y}^{des}(\varkappa T + T)$ . This is the result of linear interpolation of discrete sets of actuations  $l^0(\varkappa T)$  designed for depicted desired configurations. Right figure shows the comparison of cables' forces  $\mathbf{s}(t)$  calculated by Simscape and linearly interpolated sets of cables' forces  $\mathbf{s}(\varkappa T)$  calculated by path-planning. Quite a similar phenomenon as in case of Figure 4.6b applies – all forces are in acceptable range of values with maximal force

#### 4. Active tensegrity structures

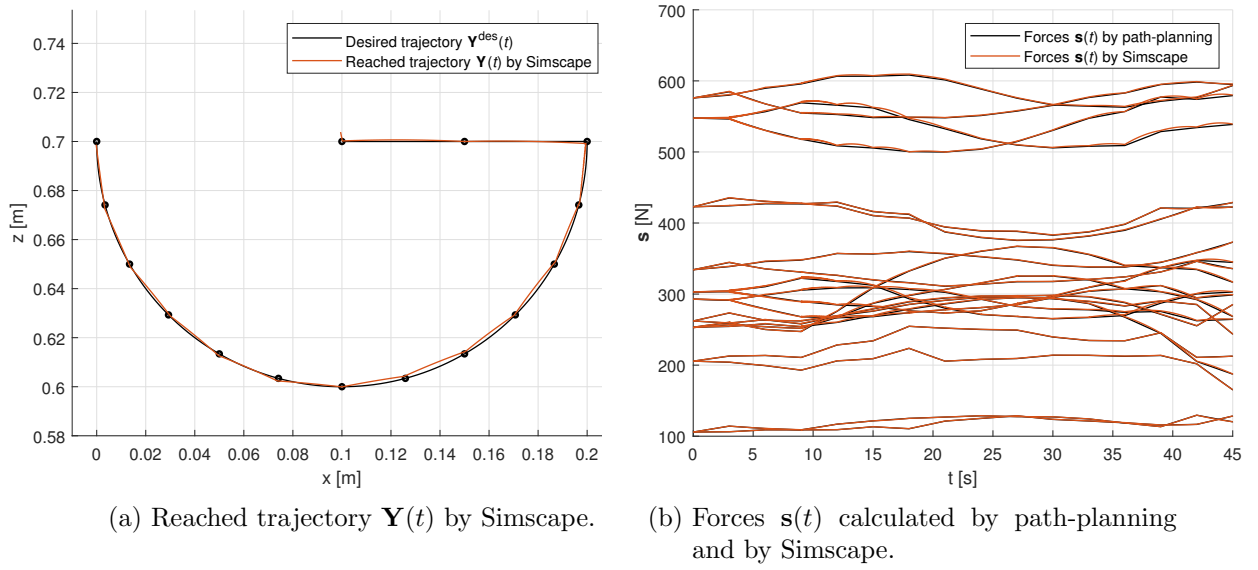


Figure 4.9.: A Simscape simulation of the 2-D structure with actuation  $\mathbf{I}^{0,A}(t)$  defined by path-planning.

about 610 N and minimal force about 100 N.

As promised, the second case of the dynamic simulation is presented too, specifically with the sampling period  $T = 0.3\text{s}$  and corresponding actuation  $\mathbf{I}^{0,B}(t), t \in \langle 0\text{s}, 4.5\text{s} \rangle$ . This actuation results in fast motion with occurrence of significant structural vibrations. From visual results in Figure 4.10a, it is clear that oscillations are excited mainly by the sharp transition between the linear and the semicircular part of the desired trajectory.

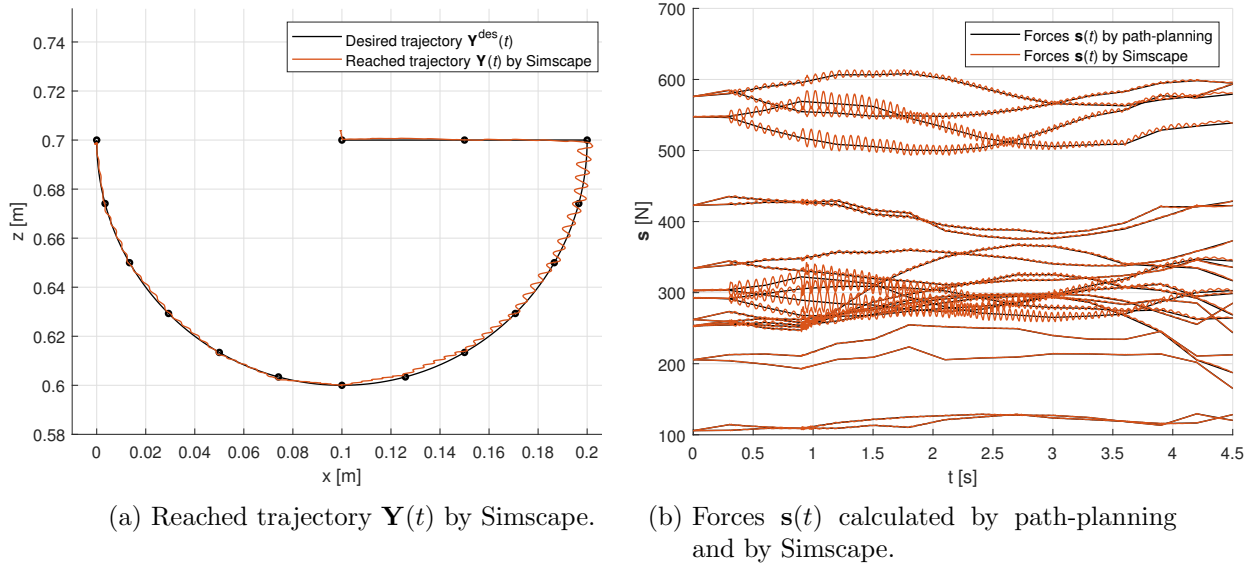


Figure 4.10.: A Simscape simulation of the 2-D structure with actuation  $\mathbf{I}^{0,B}(t)$  defined by path-planning.

### 3-D structure

The second example structure is in form of a 3-D tensegrity tower consisting of 3 floors. In Figure 4.11, the structure is visualized from two views. It is a symmetric tensegrity, each floor consists of 4 struts and adjacent floors are rotated 45° relative to each other.

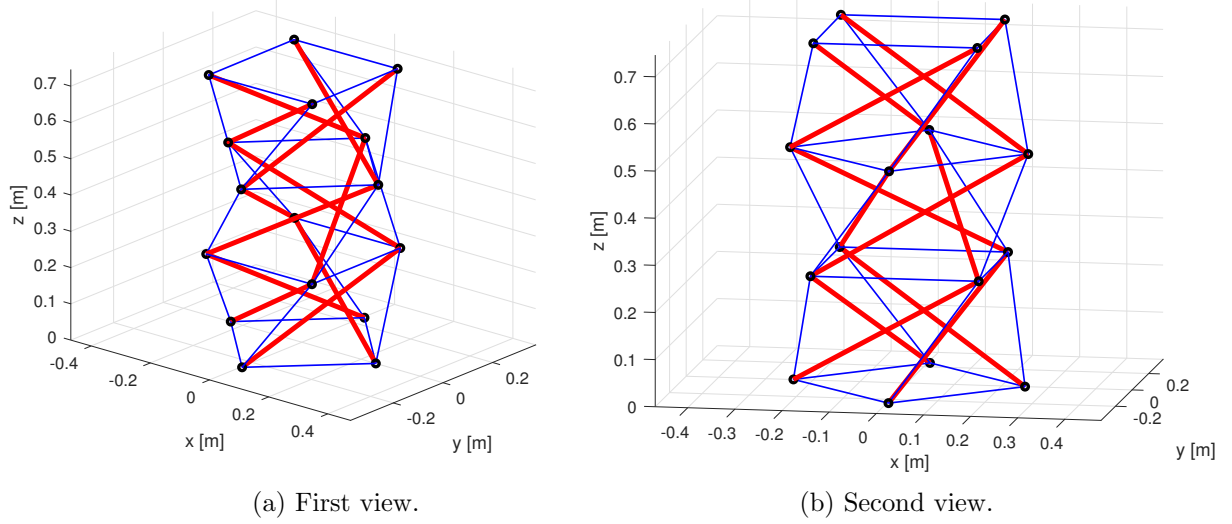


Figure 4.11.: A 3-D structure in the initial configuration.

Regarding this 3-D structure, it is required that the top behaves like a horizontal platform. Therefore, any tilting of the plane formed by 4 top nodes relative to the  $xy$  plane is undesirable during the tensegrity motion. Thus, all 4 top-layer nodes are selected as master nodes ( $n^m = 4$ ) and 4 parallel desired trajectories are designed – each one assigned

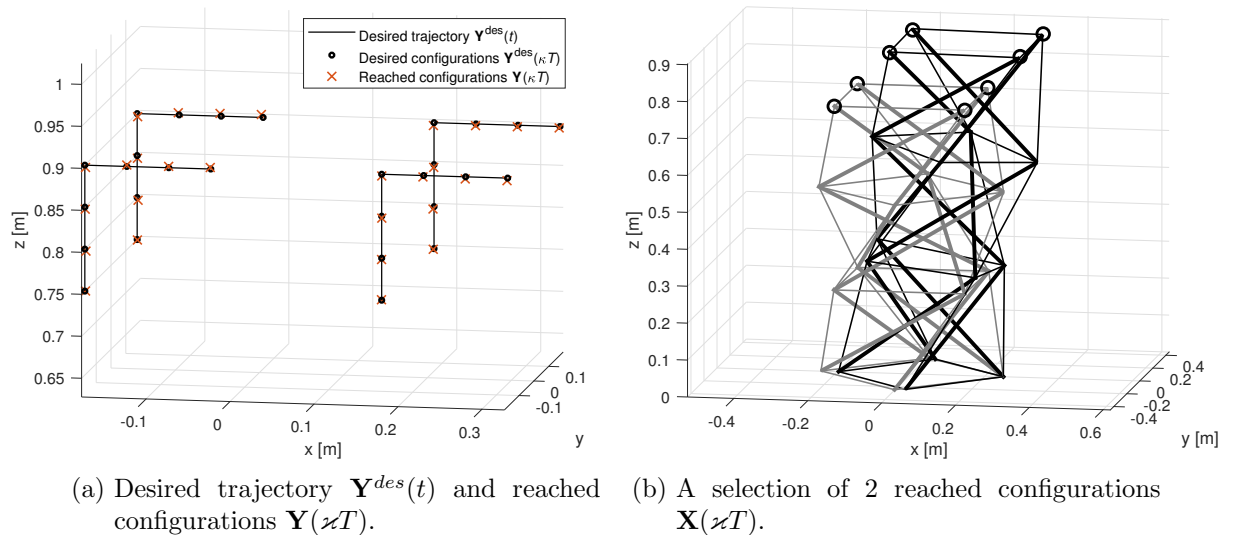


Figure 4.12.: The 3-D structure subjected to path-planning.

#### 4. Active tensegrity structures

to one of master nodes. Desired trajectories  $\mathbf{Y}^{des}(t)$  are depicted in Figure 4.12a. These trajectories are discretized into 7 ( $n^T = 6$ ) desired configurations  $\mathbf{Y}^{des}(\varkappa T)$ , each one consisting of 4 spatial positions. The structure is then subjected to path-planning and resulting configurations  $\mathbf{Y}(\varkappa T)$  are marked in Figure 4.12a. They seem accurate at the first sight. In Figure 4.12b, there are 2 resulting structure configurations  $\mathbf{X}(\varkappa T)$  visualized (more configurations would make the figure confusing). The light one is for the first path-planning iteration and the black one is for the last iteration. Black circles have the same meaning as in case of the 2-D structure, they represent desired positions  $\xi_i(\varkappa T)$  of each master node in the first and the last iteration.

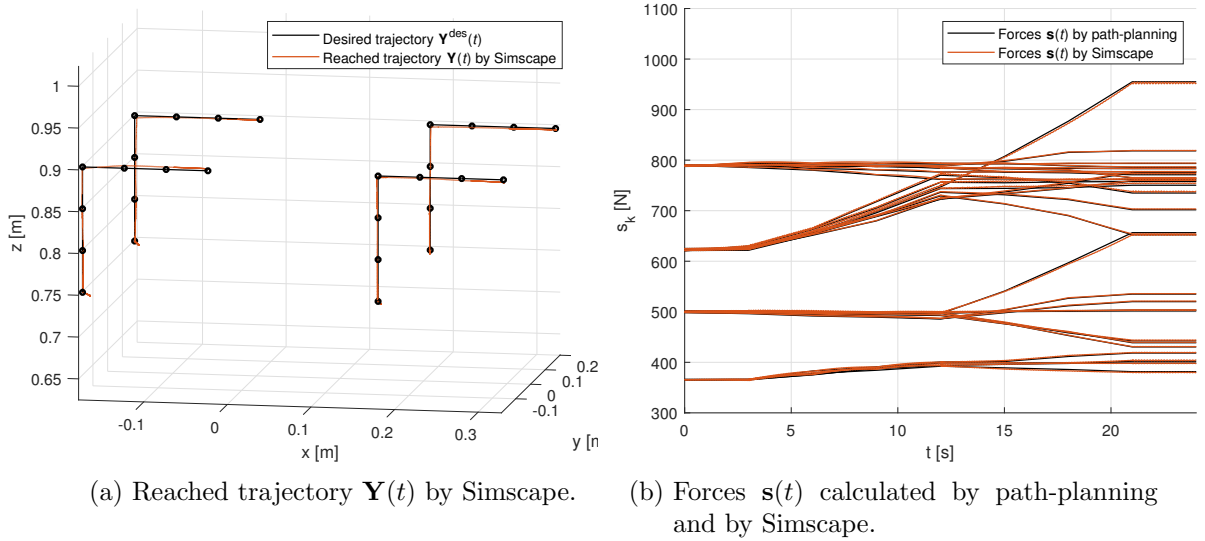


Figure 4.13.: A Simscape simulation of the 3-D structure with actuation  $\mathbf{I}^{0,A}(t)$  defined by path-planning.

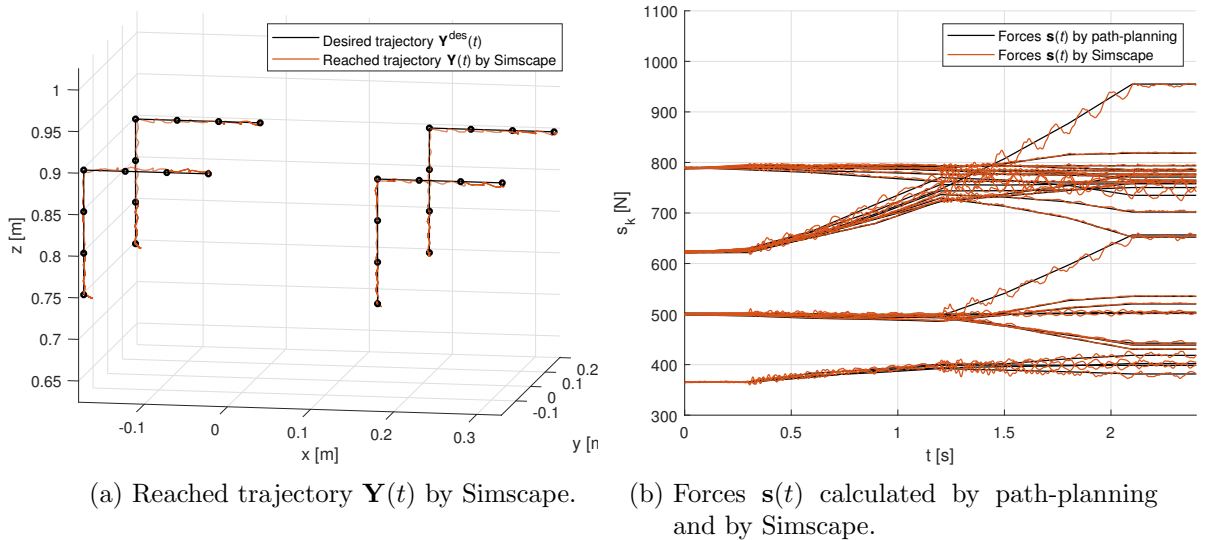


Figure 4.14.: A Simscape simulation of the 3-D structure with actuation  $\mathbf{I}^{0,B}(t)$  defined by path-planning.

In Figures 4.13 and 4.14, there are results of two Simscape simulations, the first one with actuation  $\mathbf{l}^{0,A}(t), t \in \langle 0\text{ s}, 24\text{ s} \rangle$ , as a result of the sampling period  $T = 3\text{ s}$ , and the second one with actuation  $\mathbf{l}^{0,B}(t), t \in \langle 0\text{ s}, 2.4\text{ s} \rangle$ , with the sampling period  $T = 0.3\text{ s}$ . The results are analogous to the previous 2-D example, therefore, no additional commentary is needed.

## 4.2. Shape control

From results presented in the previous section, it is clear that the first stage of control design presented in this work, path-planning, fulfils its role – it generates primary actuation enabling the structure to follow the desired trajectory. Eventually, in cases of slow motion, relatively straight desired paths, and precisely set path-planning process not allowing considerable position errors, it is possible to use the designed actuation as the final one. In opposite cases, mainly in cases of fast movements, or with desired paths that cannot be considered as almost linear, significant position errors occur mainly in form of vibrations. Therefore, subsequent actuation has to be performed to suppress position errors. A feedback  $H_2$  controller reducing position errors (both structural vibrations and non-oscillating deviations) is designed in the second stage of tensegrity control design in this work.

### 4.2.1. Linearization

Tensegrity model created in Simscape is non-linear, therefore, it has to be linearized to be able to design an  $H_2$  controller. Non-linear dynamics of the MIMO (multiple input, multiple output) system can be conventionally summarized as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{2n^{dof}}, \quad \mathbf{x}(t) \in \mathbb{R}^{2n^{dof}}, \mathbf{u}(t) \in \mathbb{R}^{m^c}, \quad (4.17)$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n^m}, \quad (4.18)$$

where  $\mathbf{x}$  is the vector of state variables,  $\mathbf{u}$  is the input vector (cables' rest lengths),  $\mathbf{y}$  is the output vector (positions of master nodes), and  $\mathbf{f}, \mathbf{g}$  are non-linear functions. The number of structure's degrees of freedom is denoted as  $n^{dof}$ , number of inputs (number of cables) as  $m^c$  and number of outputs (number of master nodes) as  $n^m$ .

Let the point  $\mathbf{x}^*$  be considered as an equilibrium point with corresponding equilibrium input  $\mathbf{u}^*$  and equilibrium output  $\mathbf{y}^*$ . For  $\mathbf{x}^*, \mathbf{u}^*$ , it applies that

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0} \quad (4.19)$$

according to [15]. If system starts from the initial state  $\mathbf{x}(0) = \mathbf{x}^*$  and the input  $\mathbf{u}(t) = \mathbf{u}^*$  is applied for all  $t \geq 0$ , the resulting state and the system output is in form of

$$\mathbf{x}(t) = \mathbf{x}^*, \quad t \geq 0, \quad (4.20)$$

$$\mathbf{y}(t) = \mathbf{y}^*. \quad (4.21)$$

Deviation variables  $\hat{\mathbf{x}}, \hat{\mathbf{u}}, \hat{\mathbf{y}}$  can be introduced as

$$\hat{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}^* \quad \rightarrow \quad \mathbf{x}(t) = \mathbf{x}^* + \hat{\mathbf{x}}(t), \quad (4.22)$$

$$\hat{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}^* \quad \rightarrow \quad \mathbf{u}(t) = \mathbf{u}^* + \hat{\mathbf{u}}(t), \quad (4.23)$$

$$\hat{\mathbf{y}}(t) = \mathbf{y}(t) - \mathbf{y}^* \quad \rightarrow \quad \mathbf{y}(t) = \mathbf{y}^* + \hat{\mathbf{y}}(t). \quad (4.24)$$

According to [11], the non-linear system defined in (4.17) and (4.18) combined with (4.22)–(4.24) can be expanded to Taylor series of the first order with higher orders omitted as

$$\begin{aligned} \dot{\hat{\mathbf{x}}} &= \dot{\mathbf{x}} + \dot{\hat{\mathbf{x}}} = \mathbf{f}(\mathbf{x}^* + \hat{\mathbf{x}}, \mathbf{u}^* + \hat{\mathbf{u}}) = \\ &= \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) + \frac{\partial \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{x}} (\mathbf{x} - \mathbf{x}^*) + \frac{\partial \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{u}} (\mathbf{u} - \mathbf{u}^*), \end{aligned} \quad (4.25)$$

$$\begin{aligned} \mathbf{y} &= \mathbf{y}^* + \hat{\mathbf{y}} = \mathbf{g}(\mathbf{x}^* + \hat{\mathbf{x}}, \mathbf{u}^* + \hat{\mathbf{u}}) = \\ &= \mathbf{g}(\mathbf{x}^*, \mathbf{u}^*) + \frac{\partial \mathbf{g}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{x}} (\mathbf{x} - \mathbf{x}^*) + \frac{\partial \mathbf{g}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{u}} (\mathbf{u} - \mathbf{u}^*). \end{aligned} \quad (4.26)$$

Because equations (4.19) and (4.21) apply, it can be written that

$$\dot{\hat{\mathbf{x}}} = \frac{\partial \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{x}} \hat{\mathbf{x}} + \frac{\partial \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{u}} \hat{\mathbf{u}}, \quad (4.27)$$

$$\hat{\mathbf{y}} = \frac{\partial \mathbf{g}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{x}} \hat{\mathbf{x}} + \frac{\partial \mathbf{g}(\mathbf{x}^*, \mathbf{u}^*)}{\partial \mathbf{u}} \hat{\mathbf{u}}. \quad (4.28)$$

Equations (4.27) and (4.28) represent linear time-invariant system valid in the neighbourhood of the equilibrium point  $\mathbf{x}^*$  with corresponding  $\mathbf{u}^*$  and  $\mathbf{y}^*$ , in other words, for small values of deviation variables  $\hat{\mathbf{x}}, \hat{\mathbf{u}}, \hat{\mathbf{y}}$ . Linearized state-space model can be expressed in a compact conventional matrix form

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\hat{\mathbf{u}}, \quad (4.29)$$

$$\hat{\mathbf{y}} = \mathbf{C}\hat{\mathbf{x}} + \mathbf{D}\hat{\mathbf{u}}, \quad (4.30)$$

where

$$\mathbf{A} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_{n^{dof}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n^{dof}}(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_1} & \cdots & \frac{\partial f_{n^{dof}}(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_{n^{dof}}} \end{bmatrix} \in \mathbb{R}^{n^{dof} \times n^{dof}}, \quad (4.31)$$

$$\mathbf{B} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_1} & \cdots & \frac{\partial f_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_{m^c}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n^{dof}}(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_1} & \cdots & \frac{\partial f_{n^{dof}}(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_{m^c}} \end{bmatrix} \in \mathbb{R}^{n^{dof} \times m^c}, \quad (4.32)$$

$$\mathbf{C} = \begin{bmatrix} \frac{\partial g_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_1} & \cdots & \frac{\partial g_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_{n^{dof}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{n^m}(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_1} & \cdots & \frac{\partial g_{n^m}(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_{n^{dof}}} \end{bmatrix} \in \mathbb{R}^{n^m \times n^{dof}}, \quad (4.33)$$



$$\mathbf{D} = \begin{bmatrix} \frac{\partial g_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_1} & \cdots & \frac{\partial g_1(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_{m^c}} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{n^m}(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_1} & \cdots & \frac{\partial g_{n^m}(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_{m^c}} \end{bmatrix} \in \mathbb{R}^{n^m \times m^c}. \quad (4.34)$$

As mentioned in the beginning of section 4.2.1, the input  $\mathbf{u}$  represents cables' rest lengths  $\mathbf{l}^0$  and the output  $\mathbf{y}$  is in form of positions  $\mathbf{Y}$  of master nodes. An appropriate equilibrium input  $\mathbf{u}^*$  with the corresponding response in form of the equilibrium output  $\mathbf{y}^*$  has to be determined. These two variables define the state, in which the system is linearized. One of suitable options is to choose cables' rest lengths  $\bar{\mathbf{L}}^0$  used as the input to the path-planning process (see Figure 4.1, box "Start"). The corresponding equilibrium output is also available – positions  $\bar{\mathbf{Y}}$  of master nodes calculated by the DRM in the beginning of the path-planning process (see Figure 4.1, first box after "Start", where  $\mathbf{X}$  naturally contains  $\mathbf{Y}$ ). Therefore, the input  $\hat{\mathbf{u}}$  represents deviations  $\delta$  of cables' rest lengths from the their initial setting  $\bar{\mathbf{L}}^0$ . Deviations of cables' rest lengths were already introduced in section 4.1 in form of adjustments of cables' rest lengths<sup>16</sup>. Thus, input deviations are defined analogously to (4.10) as

$$\delta(t) = \mathbf{l}^0(t) - \bar{\mathbf{L}}^0. \quad (4.35)$$

The output  $\hat{\mathbf{y}}$  represents deviations  $\hat{\mathbf{Y}}$  of master nodes' positions expressed as

$$\hat{\mathbf{Y}}(t) = \mathbf{Y}(t) - \bar{\mathbf{Y}}. \quad (4.36)$$

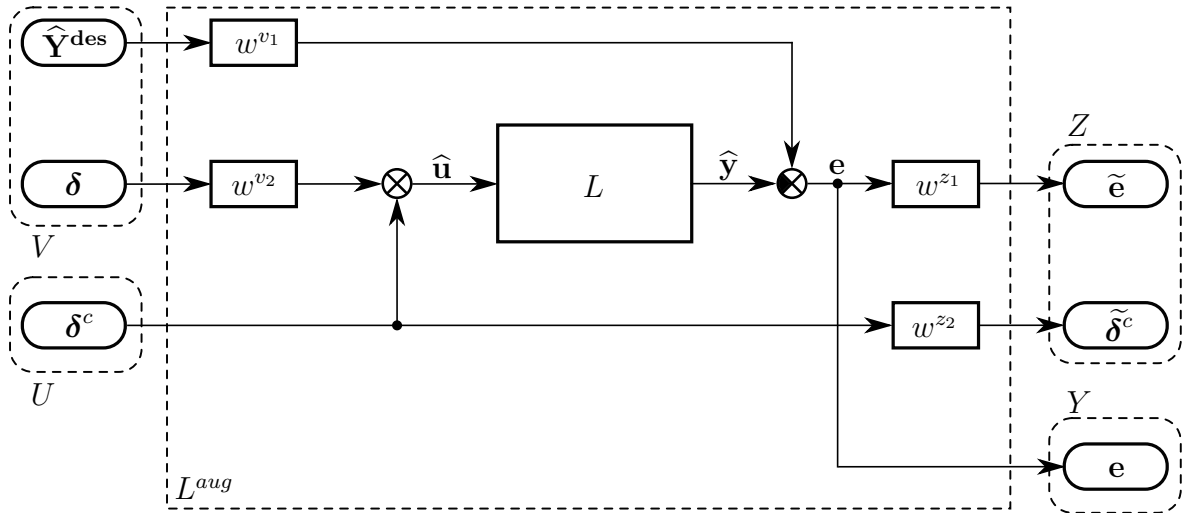


Figure 4.15.: Augmented plant.

<sup>16</sup>Adjustments  $\delta$  in section 4.1 have exactly the same meaning as deviations used in this section. The only difference is that  $\bar{\mathbf{L}}^0$  is changed after each iteration of the path-planning process. Here,  $\bar{\mathbf{L}}^0$  remains the same of course.

### 4.2.2. Augmented plant

An augmented plant is an instrument, with which it is possible to express control objectives. A specific augmented plant, which is slightly inspired by [26] and [16], is used in this work for  $H_2$  controller design. It is schematically depicted in Figure 4.15.

The linearized model of tensegrity expressed in (4.29)–(4.34) is denoted as  $L$  and the resulting augmented plant model as  $L^{aug}$ . Inputs of  $L^{aug}$  are divided into two separate sections. Section  $V$  includes external inputs, namely the deviation  $\widehat{\mathbf{Y}}^{des}$  of the current desired configuration from the configuration  $\overline{\mathbf{Y}}$  (same as in (4.36)) defined as

$$\widehat{\mathbf{Y}}^{des}(t) = \mathbf{Y}^{des}(t) - \overline{\mathbf{Y}}, \quad (4.37)$$

and deviations  $\boldsymbol{\delta}$  of cables' rest lengths. Section  $U$  includes controller effort  $\boldsymbol{\delta}^c$  (output from the controller). Outputs are also divided into two sections. Section  $Z$  contains signals that are desired to be minimized (control objectives), namely the weighted position error  $\tilde{\mathbf{e}}$  of master nodes and the weighted controller effort  $\tilde{\boldsymbol{\delta}}^c$ . Section  $Y$  representing controller inputs consists only of the position error  $\mathbf{e}$  of master nodes.

As mentioned, all external inputs and control objectives are subjected to filtering. In accordance with [26], weights  $w^{v1}$ ,  $w^{v2}$  represent input filters approximating spectrum of input signals, while weights  $w^{z1}$ ,  $w^{z2}$  are designed to emphasize relevant frequencies of the objectives.

### 4.2.3. $H_2$ controller

This section is dedicated to a few comments regarding  $H_2$  controller synthesis. The problem is defined according to [31] as searching for the controller  $C$ , which stabilizes the system in form of the augmented plant  $L^{aug}$  and minimizes the  $H_2$ -norm of transfer function from  $V$  to  $Z$ . A block diagram illustrating the closed loop consisting of controller and the augmented plant is depicted in Figure 4.16. For an overview, the  $H_2$ -norm of a general continuous system with transfer function  $T(s)$  is defined according to [7] as

$$\|T\|_2 = \sqrt{\frac{1}{2\pi} \int_{-\infty}^{\infty} \text{trace}\{T(i\omega)^H T(i\omega)\} d\omega}. \quad (4.38)$$

Further details about principles of  $H_2$  controller synthesis can be found for instance in [31]. This topic is beyond the scope of this work, thus, it is not further discussed.

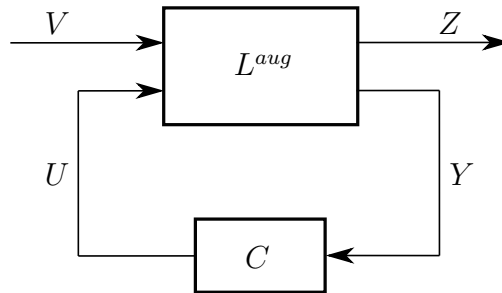


Figure 4.16.: Closed loop system.

In this work,  $H_2$  controller synthesis is performed by the MATLAB function *h2syn* from the *Robust Control Toolbox*.

The application of a robust controller is justifiable. The original tensegrity system, which is going to be controlled, is strongly non-linear. Also, working conditions can be quite far from the equilibrium point, in which the system is linearized. Furthermore, this work aims especially on active tensegrity structures in form of robotic manipulators, and, therefore, structures should be able to carry loads in form of additional mass.

#### 4.2.4. Examples

$H_2$  control is demonstrated on the 2-D example structure from section 4.1.3 depicted in Figure 4.3 with desired trajectory consisting of a line followed by a semicircle assigned to the top node. Primary actuation defined by path-planning with sampling period  $T = 0.3\text{s}$ , i.e.  $\mathbf{1}^{0,B}(t)$ , is considered. Apparently from the previous section, quality of the resulting  $H_2$  controller is derived from particular setting of weighting filters in the augmented plant. For purposes of this work, two sets of weighting filters (differing in the weight  $w^{z_1}$  designed as a low-pass filter in both cases) are selected, therefore, two different controllers labelled as *A* and *B* are designed. Filters' transfer functions are available in Appendix A.1. Benefits of both controllers are evident from following figures.

Figure 4.17 contains a comparison of the desired trajectory  $\mathbf{Y}^{des}(t)$  and calculated trajectories  $\mathbf{Y}(t)$  of the structure without control (actuation defined by path-planning is final), with controller *A*, and with controller *B*.

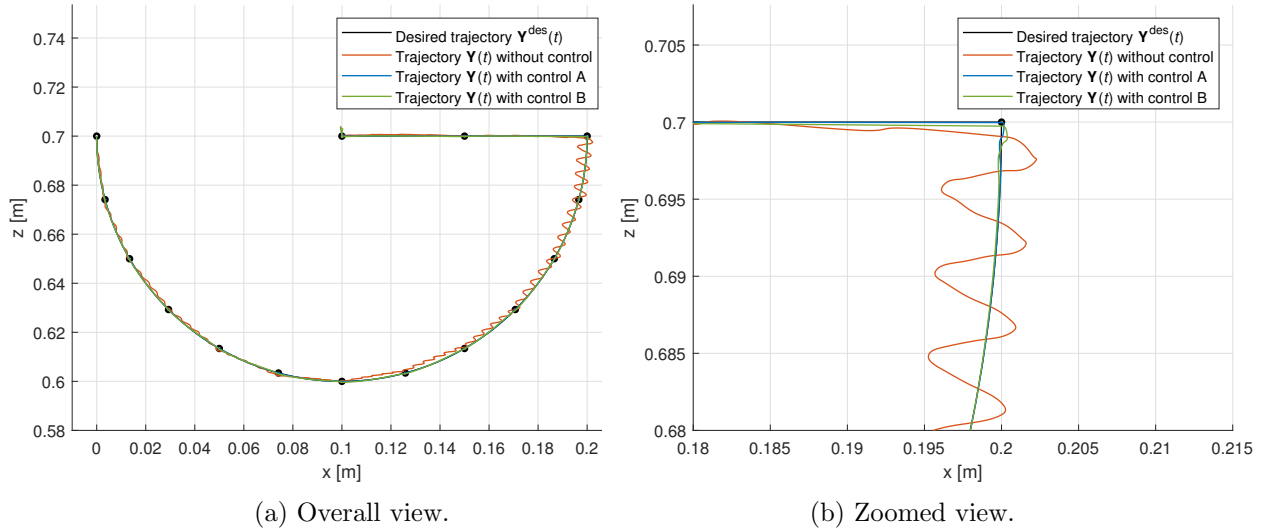


Figure 4.17.: Reached trajectory  $\mathbf{Y}(t)$  of uncontrolled and controlled 2-D structure.

At the first sight, the control quality is might not very clear from Figure 4.17. Thus, the total error  $\|\mathbf{e}(t)\|$  of the configuration  $\mathbf{Y}(t)$  from the desired configuration  $\mathbf{Y}^{des}(t)$  is defined as an Euclidean norm

$$\|\mathbf{e}(t)\| = \|\mathbf{Y}(t) - \mathbf{Y}^{des}(t)\|. \quad (4.39)$$

This quantity is visualized in Figure 4.18. From both mentioned Figures 4.17 and 4.18, it is obvious that both controllers satisfy the requirement of position error reduction. Both static error caused by linear approximation of primary actuation defined by path-planning and dynamic error in form of vibrations caused by sharp corner in the desired trajectory are suppressed. Moreover, Figure 4.18 shows that the initial position error  $\|\mathbf{e}(0)\|$  is reduced very fast. The controller *A* is much stronger than *B* in minimizing static errors. On the other hand, the controller *B* is designed to quickly suppress structural vibrations.

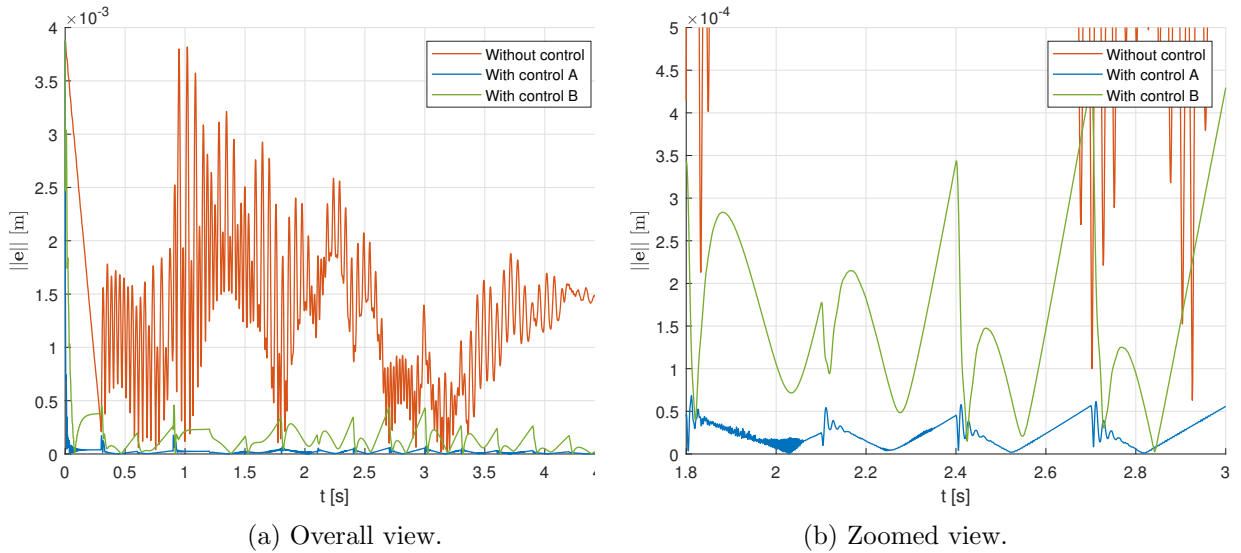


Figure 4.18.: Position error  $\|\mathbf{e}(t)\|$  of uncontrolled and controlled 2-D structure.

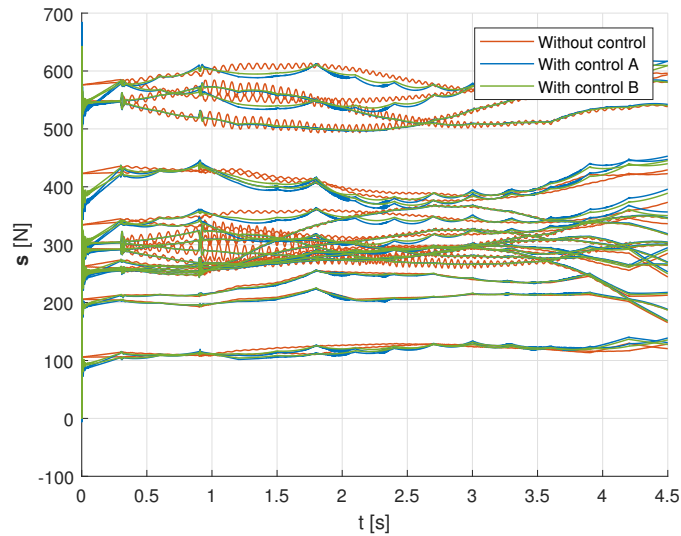


Figure 4.19.: Cables' forces  $\mathbf{s}(t)$  in uncontrolled and controlled 2-D structure.

Figure 4.19, where internal forces  $\mathbf{s}(t)$  are depicted, serves as a proof that the controlled structure is maintaining its integrity during the whole simulation. Moreover, magnitudes

of forces are pretty similar to the ones in the uncontrolled structure (tension in the uncontrolled and controlled structure is at the same level).

Input deviation  $\delta(t)$  defined in (4.35) represents only a different form of the primary actuation  $\mathbf{l}^0(t)$  designed by path-planning. This quantity is shown in Figure 4.20a. It can be used as an effective measure of control effort magnitudes. The control effort  $\delta^c(t)$  visualized in Figure 4.20b represents the output of the controller added to the actuation  $\mathbf{l}^0(t)$  delivered as the input to the controlled system. One of controller design criteria is the control effort minimization. This objective seems to be satisfied because the control effort  $\delta^c(t)$  is relatively small compared to the input deviation  $\delta(t)$ .

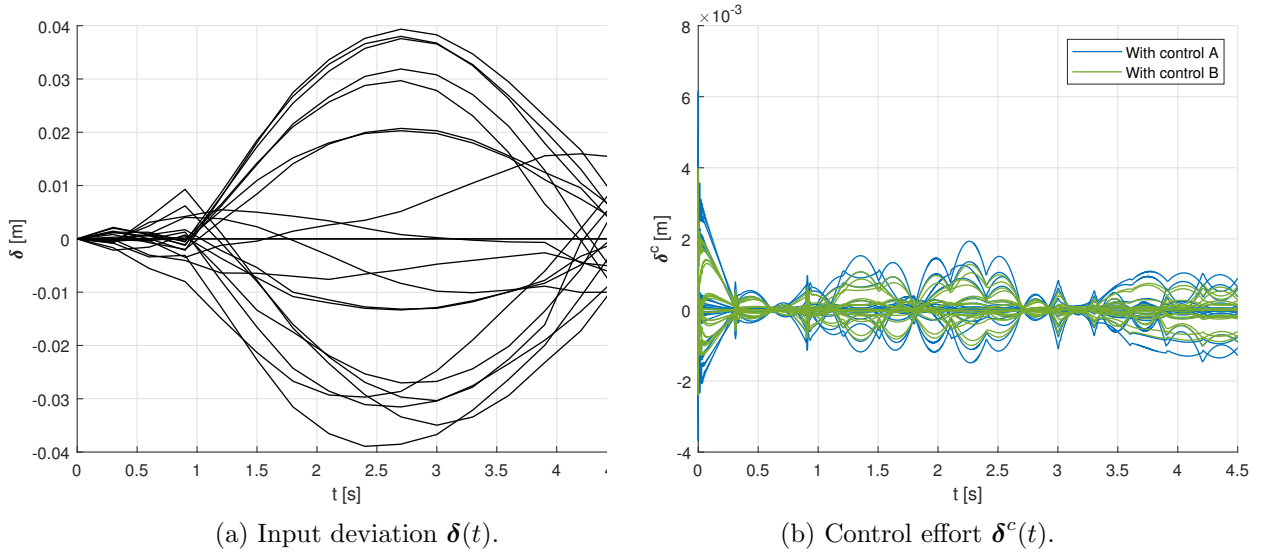


Figure 4.20.: Input deviation  $\delta(t)$  defined by path-planning and control effort  $\delta^c(t)$  of the controlled 2-D structure.

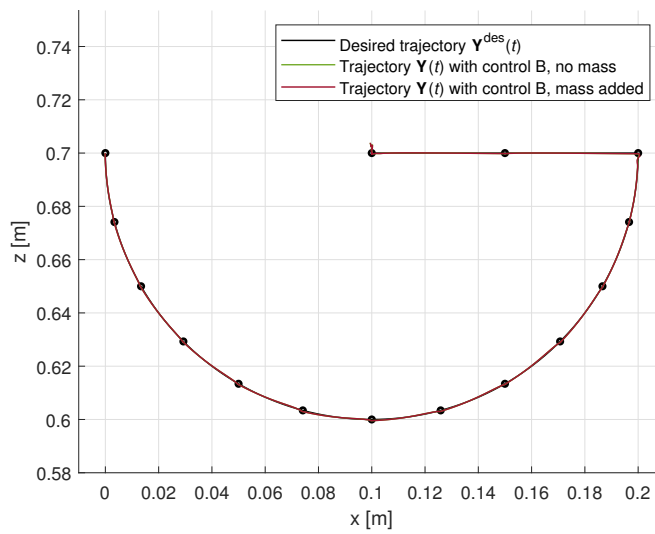


Figure 4.21.: Reached trajectory  $\mathbf{Y}(t)$  of controlled 2-D unloaded and loaded structure.

### Structure with additional mass

As the headline suggests, the same 2-D structure discussed in the previous text is loaded by additional point mass  $m^{add}$  placed to the top node. This imitates the situation that the robotic manipulator transfers some load by its effector. For purposes of this example,  $m^{add} \approx 0.2 \cdot m^{real}$ , where  $m^{real}$  denotes the weight of the whole structure, which is a considerable burden. Particular value of  $m^{add}$  is provided in Appendix A.1. The controller  $B$  designed for controlling of the unloaded structure is utilized for controlling this new system and original outputs from path-planning considering the unloaded structure are used as well.

In Figure 4.21, desired trajectory and calculated trajectories of unloaded and loaded controlled structures are compared. No problems in controlling the loaded structure are visible. Also the position error  $\|\mathbf{e}(t)\|$  and the control effort  $\delta^c(t)$  visualized in Figure 4.22 do not report any control problems. When comparing the simulation of unloaded and loaded structure, the only significant differences in these quantities are observed around  $t = 0.3\text{s}$  and  $t = 0.9\text{s}$  representing the start time of following the desired trajectory and time of encountering the sharp corner in the desired trajectory. It is definitely caused by higher inertia of the whole structure. Static effects of additional mass are suppressed very well.

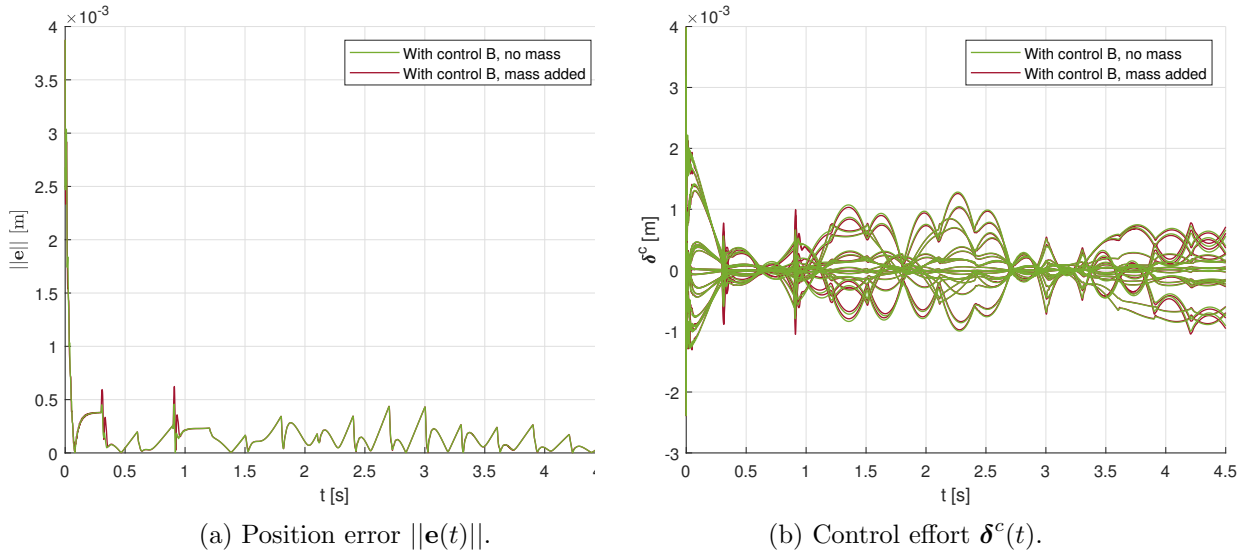


Figure 4.22.: Position error  $\|\mathbf{e}(t)\|$  and control effort  $\delta^c(t)$  of controlled 2-D unloaded and loaded structure.

## 5. Conclusion

The submitted thesis deals with the design of an active tensegrity structure. In chapter 2, the problem of form-finding (searching for a stable equilibrated configuration associated with prestresses) is treated with two different approaches: the Adaptive force density method (AFDM) and the Dynamic relaxation method (DRM). However, both methods solve the same problem, their applicability is different. Usage of the first mentioned one is advantageous mainly in the beginning of the design process, when almost no data are available and the designer's aim is to explore various stable configurations with the same topology without much effort. On the other hand, the second form-finding method requires a fully specified structure. It calculates the static response of the structure to a specific actuation, which is necessary to know during the design stage called path-planning.

The principal aim of path-planning is to design the actuation that leads the structure to follow a desired trajectory in form of trajectories of selected structure nodes. Path-planning is approached as an iterative optimization process utilizing the DRM when evaluating the objective function. All details regarding this topic can be found in section 4.1. If the motion of a tensegrity following a desired trajectory is relatively slow, it can be considered as quasi-static, and the proposed process of actuation design is sufficient. But in many robotic applications, motion can be quite fast, and thus, undesirable vibrations can occur. Because the presented path-planning process is based on static analysis only, it cannot suppress them. Therefore, an  $H_2$  controller is designed in section 4.2 to reduce vibrations.

Chapter 3 of the thesis dedicated to modelling the dynamics of tensegrity structures is started with Lagrange's equations. The main aim of this chapter is to present a developed methodology of tensegrity computational model creation in Simscape software. In order to make generation of Simscape dynamic models of tensegrities fast and smooth, a software automatically generating the Simscape model is developed in the MATLAB scripting environment as an alternative to manual creation.

The whole methodology consisting of individual fragments mentioned above form quite a robust instrument enabling the design of various active tensegrity structures of arbitrary class. Each part of the design process is tested with different structures. As the topic of the thesis represents pretty large area, there is plenty of potential for future augmentation mainly in the field of control. The proposed control approach does not ensure internal stability of the whole structure, therefore the designed  $H_2$  controller can destabilize the structure in some cases. This has to be treated before applying the controller into a real structure. Another possible development of the work can be performed by allowing the usage of other compressive members in form of at least 2-D bodies.

# Bibliography

- [1] Ali, N. B. H., Rhode-Barbarigos, L., Smith, I.: Analysis of clustered tensegrity structures using a modified dynamic relaxation algorithm, *International Journal of Solids and Structures*, vol. 48, no. 5, pp. 637-347, 2011.
- [2] Barnes, M.: Form finding and analysis of tension structures by dynamic relaxation, *International Journal of Space Structures*, vol. 14, no. 2, pp. 89-104, 1999.
- [3] Barnes, M.: *Form Finding and Analysis of Tension Structures by Dynamic Relaxation*, Ph.D. thesis, The City University, London, UK, 1977.
- [4] Bulín, R.: *Advanced computational methods for the dynamical analysis of multibody systems with flexible beams and ropes*, Ph.D. thesis, University of West Bohemia, Pilsen, Czech Republic, 2019.
- [5] Cai, H., Wang, M., Xu, X., Luo, Y.: A general model for both shape control and locomotion control of tensegrity systems, *Frontiers in Built Environment*, vol. 6, 2020.
- [6] Connelly, R., Whiteley, W.: Second-order rigidity and prestress stability for tensegrity frameworks, *SIAM Journal on Discrete Mathematics*, vol. 9, no. 3, pp. 453-491, 1996.
- [7] *Documentation, manuals and user's guide to MATLAB* [online], MathWorks, available: <https://www.mathworks.com/support.html>. [Accessed: 19-May-2022].
- [8] Dupal, J.: *Mechanics 3*, Script for lectures, University of West Bohemia, Pilsen, Czech Republic, 2012.
- [9] Fuller, R. B.: *Tensile-integrity structures*, United States Patent no. 3063521, November, 1962.
- [10] Gan, B. S.: *Computational modeling of tensegrity structures*, Springer, Cham, Switzerland, 2020, ISBN 978-3-030-17835-2.
- [11] Goubelj, M., Melichar, J.: *Linear systems 1*, Script for lectures, University of West Bohemia, Pilsen, Czech Republic, 2017.
- [12] Heartney, E.: *Kenneth Snelson: Art and ideas*, Kenneth Snelson's web, 2013.
- [13] Liu, Y., Bi, Q., Yue, X., Wu, J., Yang, B., Li, Y.: A review of tensegrity structures-based robots, *Mechanism and Machine Theory*, vol. 168, 2022.
- [14] Morto, R.: Tensegrity systems: state of the art, *International Journal of Space Structures*, vol. 7, no. 2, pp. 75-83, 1992.



- [15] Packard, A., Poolla, K., Horowitz, R.: *Dynamic systems and feedback*, Script for lectures, University of California, Berkeley, USA, 2002.
- [16] *Robust control of an active suspension* [online], MathWorks, available: <https://www.mathworks.com/help/robust/gs/active-suspension-control-design.html>. [Accessed: 19-May-2022].
- [17] Schek, H.-J.: The force density method for form finding and computation of general networks, *Computer Methods in Applied Mechanics and Engineering*, vol. 3, pp. 115-134, 1974.
- [18] Schlotter, M.: *Multibody system simulation with SimMechanics*, TU Darmstadt, Darmstadt, Germany, 2003.
- [19] Shabana, A. A.: *Dynamics of multibody systems*, Fifth edition, Cambridge University Press, Cambridge, UK, 2020, ISBN 978-1-108-48564-7.
- [20] Shah, D. S., Booth, J. W., Baines, R. L., Wang, K., Vespignani, M., Bekris, K., Kramer-Bottiglio, R.: Tensegrity robotics, *Soft Robotics*, vol. 10, 2021.
- [21] Skelton, R., de Oliveira, M.: *Tensegrity systems*, Springer, Berlin, Germany, 2009, ISBN 978-0-387-74241-0.
- [22] Skelton, R.: Dynamics of tensegrity systems: Compact forms, *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 2276-2281, 2006.
- [23] Smolík, L.: *Dynamic behavior analysis of turbocharger rotors*, Diploma thesis, University of West Bohemia, Pilsen, Czech Republic, 2013.
- [24] Tur, J. M. M., Juan, S. H.: Tensegrity frameworks: Dynamic analysis review and open problems, *Mechanism and Machine Theory*, vol. 44, pp. 1-18, 2009.
- [25] Wakefield, D. S.: *Pretensioned Networks Supported by Compression Arches*, Ph.D. thesis, The City University, London, UK, 1980.
- [26] van de Wijdeven, J., de Jager, B.: Shape change of tensegrity structures: Design and control, *American Control Conference*, pp. 2522-2527, 2005.
- [27] Wroldsen, A. S.: *Modelling and control of tensegrity structures*, Ph.D. thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2007.
- [28] Xu, X., Sun, F., Luo, Y., Xu, Y.: Collision-free planning of tensegrity structures, *Journal of Structural Engineering*, vol. 140, no. 4, 2005.
- [29] Zhang, J. Y., Ohsaki, M.: *Tensegrity Structures: Form, Stability, and Symmetry*, Springer, Tokyo, Japan, 2015, ISBN 978-4-431-54812-6.
- [30] Zhang, J. Y., Ohsaki, M.: Adaptive force density method for form-finding problem of tensegrity structures, *International Journal of Solids and Structures*, vol. 43, pp. 5658-5673, 2006.
- [31] Zhou, K., Doyle, J. C., Glover, K.: *Robust and optimal control*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1996.

# A. Appendices

## A.1. Parameter table

Type	Parameter	Value	Unit
Strut	Young's modulus, $E^s$	$210 \cdot 10^9$	Pa
	Cross-section, $A^s$	$5 \cdot 10^{-5}$	m
	Mass density, $\rho^s$	$7.8 \cdot 10^3$	kgm <sup>3</sup>
Cable	Young's modulus, $E^c$	$1 \cdot 10^9$	Pa
	Cross-section, $A^c$	$1 \cdot 10^{-5}$	m
	Proportional damping coef., $\beta$	$1 \cdot 10^{-4}$	—
DRM	Residuum tolerance, $R^{tol}$	0.1	N
Path-planning and optimization solver	<sup>17</sup> Maximal allowed adjustments, $\Delta_k$	0.2 / 0.5	m
	<sup>17</sup> Internal force threshold, $s^{min}$	$1 \cdot 10^3$ / $1 \cdot 10^4$	N
	<sup>17</sup> Position error weight, $w^S$	$1 \cdot 10^{-8}$ / $1 \cdot 10^{-7}$	—
	Penalty weight, $w^P$	$1 \cdot 10^3$	—
	Solver algorithm	<i>fminimax</i>	—
	<i>FiniteDifferenceType</i>	<i>forward</i>	—
	<i>OptimalityTolerance</i>	$1 \cdot 10^{-6}$	—
ODE solver	<sup>17</sup> Solver algorithm	<i>ode15s</i> / <i>ode45</i>	—
	<i>InitialStepSize</i>	$1 \cdot 10^{-10}$	s
	<i>MaxStepSize</i>	$1 \cdot 10^{-3}$	s
	<i>RelativeTolerance</i>	$1 \cdot 10^{-3}$	—
Augmented plant weighting filters and control	Input filter, $w^{v1}$	0.1	—
	Input filter, $w^{v2}$	0.05	—
	<sup>18</sup> Output filter, $w^{z1}$	$\frac{s+300}{s+0.1}$ / $\frac{s+30}{s+0.01}$	—
	Output filter, $w^{z2}$	0.01	—
	Additional mass, $m^{add}$	0.15	kg
General	Gravitational acceleration, $g$	-9.81	m/s <sup>2</sup>

Table A.1.: Summary of parameters and their values.

<sup>17</sup>Two different values are used: for both 2-D structures / for the 3-D structure.

<sup>18</sup>Two different values are used: for the controller *A* / for the controller *B*.

## A.2. MATLAB function for automatic generation of Simscape tensegrity models

```

1 % Martin Hrabacka, 31.5.2022
2 % Function for automatic generation of Simscape tensegrity models
3 %
4 % q      - force density vector
5 % C      - connectivity matrix
6 % X      - configuration vector
7 % type   - type of member
8 % fname  - name of resulting slx model
9 % beta   - cable's damping coefficient
10 % E_c    - cable's Young's modulus
11 % A_s    - strut's cross-section
12 % A_c    - cable's cross-section
13 % rho_s  - strut's mass density
14 % l_0    - discrete sets of cables' rest lengths
15 % T      - sampling period
16 %
17 function slx_auto_create(q,C,X,type,fname,beta,E_c,A_s,A_c,rho_s,l_0,T)
18
19 n = length(C(1,:));           % nr of nodes
20 m = length(C(:,1));          % nr of members
21 m_c = length(find(type));     % nr of cables
22 m_s = m - m_c;               % nr of struts
23
24 cable = zeros(10,m_c);       % cable parameter matrix (each column for each cable)
25                                     % [j;i_1;x_1;y_1;z_1;i_2;x_2;y_2;z_2;nodes_dist]
26                                     % j - member index, i - node index, x/y/z - node coordinates
27 strut = zeros(10,m_s);       % strut parameter matrix (each column for each strut)
28                                     % [j;i_1;x_1;y_1;z_1;i_2;x_2;y_2;z_2;l_0]
29                                     % j - member index, i - node index, x/y/z - node coordinates
30 N = [X(1:n)';X(n+1:2*n)';X(2*n+1:3*n)']; % modification of config. vector to matrix [3 x n]
31
32 %% Various calculations
33
34 t_stop = T*(length(l_0(1,:))) + T; % simulation final time
35 r_s = sqrt(A_s/pi);               % strut's radius
36
37 c_i = 0;                           % cable index in cable parameter matrix
38 s_i = 0;                           % strut index in strut parameter matrix
39 fl = 1;                             % flag for switching between first-second node
40
41 for j = 1:m                          % member index
42     for i = 1:n                       % node index
43         if C(j,i) ~= 0
44             if fl == 1
45                 first = [i;N(:,i)]; % node i = 1st node of member m
46                 fl = 2;
47             else
48                 second = [i;N(:,i)]; % node i = 2nd node of member m
49                 fl = 1;
50                 break;
51             end
52         end
53     end
54
55     if (first(4) - second(4)) > 0 % 1st node must be in lower z-level than 2nd node
56         aux = first;
57         first = second;
58         second = aux;
59     end
60
61     l = norm(first(2:4)-second(2:4)); % length of member m
62

```

## A. Appendices

---

```

63     if type(j) == 1                                % member m == cable
64         c_i = c_i + 1;
65         cable(1,c_i) = j;
66         cable(2:5,c_i) = first;
67         cable(6:9,c_i) = second;
68         cable(10,c_i) = 1;
69     else                                           % member m == strut
70         s_i = s_i + 1;
71         strut(1,s_i) = j;
72         strut(2:5,s_i) = first;
73         strut(6:9,s_i) = second;
74         strut(10,s_i) = 1;
75     end
76 end
77
78 l_0_seq = zeros(m_c,length(l_0(1,:))+1); % modified sequence of l_0 for actuation
79 c_i = 0;
80
81 for j = 1:m                                       % through all members
82     if type(j) == 1                               % type of member == cable
83         c_i = c_i + 1;
84         l_0_init = E_c*A_c*cable(10,c_i)/(cable(10,c_i)*q(j)+E_c*A_c); % calculation of l_0 from q
85         l_0_seq(c_i,:) = [l_0_init,l_0(c_i,:)];
86     end
87 end
88
89 end_nodes = zeros(n,1); % indices of nodes at the end of chains
90
91 for s_i = 1:m_s                                   % through all struts
92     end_nodes(strut(2,s_i),1) = end_nodes(strut(2,s_i),1) + 1; % beginning of s_i-th strut
93     end_nodes(strut(6,s_i),1) = end_nodes(strut(6,s_i),1) + 1; % end of s_i-th strut
94 end
95 end_nodes = find(end_nodes == 1);
96
97 %% Counting joints in the lowest z-level (spherical/planar)
98
99 joints_planar = zeros(0,1); % vector of nodes representing planar joints
100 z_min = min(strut(5,:)); % find the lowest z-level
101
102 nr = 0;
103 for n_i = 1:n                                     % through all nodes
104     if (N(3,n_i) - z_min) <= 1e-3                % different z-levels, resolution == 1e-3
105         nr = nr + 1;
106         if nr == 1
107             joint_sph = n_i;                     % node representing spherical joint
108         else
109             joints_planar = [joints_planar;n_i];
110         end
111     end
112 end
113
114 s_laying = zeros(0,1); % members laying in the lowest z-level
115
116 for s_i = 1:m_s                                   % through all struts
117     elev_s_i = strut([5,9],s_i);                 % z-coord. of both nodes of strut s_i
118     bool_laying = (elev_s_i == [z_min;z_min]);    % is node laying in the lowest z-level?
119     if (bool_laying(1) == 1) && (bool_laying(2) == 1) % member lies in the lowest z-level
120         s_laying = [s_laying;s_i];
121     end
122 end
123
124 %% File generation, "world" configuration, standard blocks
125
126 if exist(fname,'file') == 4
127     if bdIsLoaded(fname)
128         close_system(fname,0);

```

## A. Appendices

---

```

129     end
130     delete([fname, '.mdl']);
131 end
132 new_system(fname);
133 open_system(fname);
134
135 add_block('nesl_utility/Solver Configuration', [gcs, '/Solver'],...
136         'Position', [10 40 50 80], 'Orientation', 0);
137 add_block('sm_lib/Utilities/Mechanism Configuration', [gcs, '/Conf'],...
138         'Position', [10 110 50 150], 'Orientation', 0, 'GravityVector', '[0 0 -9.81]');
139 add_block('sm_lib/Frames and Transforms/World Frame', [gcs, '/WorldFr'],...
140         'Position', [10 180 50 220], 'Orientation', 0);
141 add_block('built-in/Constant', [gcs, '/Const'],...
142         'Position', [10 280 50 320], 'Orientation', 0, 'Value', '0');
143 add_block('nesl_utility/Simulink-PS Converter', [gcs, '/S-PS'],...
144         'Position', [90 280 130 320], 'Orientation', 0);
145 add_line(gcs, 'Solver/RConn1', 'Conf/RConn1');
146 add_line(gcs, 'Solver/RConn1', 'WorldFr/RConn1');
147 add_line(gcs, 'Const/1', 'S-PS/1');
148
149 set_param(gcs, 'Solver', 'ode15s', 'MaxStep', '1e-3', 'StopTime', num2str(t_stop));
150
151 % Generation of links between frame and struts in lowest z-level, part 1 (1x spherical)
152
153 POSITION = struct;    % structure for positioning blocks in the model
154 POSITION.size_half_small = [40/2, 40/2];    % x/y-dimensions of small blocks
155 POSITION.size_half_large = [80/2, 60/2];    % x/y-dimensions of large blocks
156 POSITION.y_gap = 90;    % y-gap between blocks
157 POSITION.x_gap_small = 80;    % small x-gap between blocks
158 POSITION.x_gap_large = 200;    % large x-gap between blocks
159 POSITION.x_level = 330;    % current x-position
160 POSITION.y_level = 130;    % current y-position
161 POSITION.x_level_sensor = 250;    % current x-position for sensors
162
163 members_used = zeros(0,1);    % already added members
164 nodes_used = zeros(0,1);    % already added nodes
165 strut_orient = zeros(m_s,1);    % orientation of struts compared to strut param. matrix
166 strut_rot = zeros(m_s,2);    % rotation of struts
167
168 missing_sph = -1*ones(n,1);    % index = node with missing sph joint
169                                % value = nr of the last strut added to the node
170
171 lib_path = 'sm_lib/Joints/Spherical Joint';
172 name = ['/Spherical', num2str(joint_sph)];
173 add_joint(name, POSITION, lib_path, 0);    % add spherical joint
174
175 POSITION.x_level = POSITION.x_level - POSITION.x_gap_small;
176 trans = "[" + N(1, joint_sph) + ", " + N(2, joint_sph) + ", " + N(3, joint_sph) + "];"
177 name = ['/Trans', num2str(joint_sph)];
178 add_trans(name, POSITION, trans);    % realize translation
179
180 add_line(gcs, 'Conf/RConn1',...
181         ['Trans', num2str(joint_sph), '/LConn1']);
182 add_line(gcs, ['Trans', num2str(joint_sph), '/RConn1'],...
183         ['Spherical', num2str(joint_sph), '/LConn1']);
184
185 POSITION.x_level = POSITION.x_level + POSITION.x_gap_small + POSITION.x_gap_large;
186 nodes_used = [nodes_used; joint_sph];
187 parent_name = ['Spherical', num2str(joint_sph)];    % name of the base
188 parent_rot_1 = 0;    % precession angle of the base
189 parent_rot_2 = 0;    % nutation angle of the base
190
191 recurs_layer_nr = 0;    % number of recurrences of the fcn 'add_node'
192
193 % add the node representing spherical joint
194 [members_used, nodes_used, end_nodes, ...

```

## A. Appendices

---

```

195 strut_orient,missing_sph,strut_rot,POSITION] = ...
196     add_node(joint_sph,joints_planar,members_used,nodes_used,...
197             recurs_layer_nr,parent_name,parent_rot_1,parent_rot_2,POSITION,...
198             missing_sph,strut_rot,s_laying,end_nodes,strut_orient,strut,type,N,C,r_s,rho_s);
199
200 %% Generation of links between frame and struts in lowest z-level, part 2 (planar)
201
202 POSITION.x_level = POSITION.x_level - POSITION.x_gap_large;
203
204 if isempty(joints_planar(:,1)) % only one node in lowest z-level -> no planar joints
205     for i = 1:length(joints_planar(:,1)) % through all nodes representing planar joint
206         if isempty(find(nodes_used == joints_planar(i))) % node 'joints_planar(i)' not added yet
207
208             lib_path = 'sm_lib/Joints/6-DOF Joint';
209             name = ['/Planar', num2str(joints_planar(i))];
210             add_joint(name,POSITION,lib_path,1); % add planar joint
211
212             POSITION.x_level = POSITION.x_level - POSITION.x_gap_small;
213             trans = "[" + N(1,joints_planar(i)) + ", "...
214                   + N(2,joints_planar(i)) + ", " + N(3,joints_planar(i)) + "];"
215             name = ['/Trans', num2str(joints_planar(i))];
216             add_trans(name,POSITION,trans); % realize translation
217
218             add_line(gcs, 'Conf/RConn1',...
219                     ['Trans', num2str(joints_planar(i)), '/LConn1']);
220             add_line(gcs, ['Trans', num2str(joints_planar(i)), '/RConn1'],...
221                     ['Planar', num2str(joints_planar(i)), '/LConn1']);
222
223             POSITION.x_level = POSITION.x_level + POSITION.x_gap_small + POSITION.x_gap_large;
224             nodes_used = [nodes_used;joints_planar(i)];
225             parent_name = ['Planar', num2str(joints_planar(i))]; % name of the base
226             parent_rot_1 = 0; % precession angle of the base
227             parent_rot_2 = 0; % nutation angle of the base
228
229             % add current node representing planar joint
230             [members_used,nodes_used,end_nodes,...
231             strut_orient,missing_sph,strut_rot,POSITION] = ...
232             add_node(joints_planar(i),joints_planar,members_used,nodes_used,...
233                     recurs_layer_nr,parent_name,parent_rot_1,parent_rot_2,POSITION,...
234                     missing_sph,strut_rot,s_laying,end_nodes,strut_orient,strut,type,N,C,r_s,rho_s);
235
236         end
237     POSITION.x_level = POSITION.x_level - POSITION.x_gap_large;
238     end
239 end
240
241
242 %% Generation of links between frame and struts in other z-levels (6DOF)
243
244 if length(nodes_used) ~= n % only one node in lowest z-level -> no planar joints
245     for i = 1:n % through all nodes
246         if isempty(find(nodes_used == i)) % i-th node not added yet
247
248             lib_path = 'sm_lib/Joints/6-DOF Joint';
249             name = ['/6DOF', num2str(i)];
250             add_joint(name,POSITION,lib_path,0);
251
252             POSITION.x_level = POSITION.x_level - POSITION.x_gap_small;
253             trans = "[" + N(1,i) + ", " + N(2,i) + ", " + N(3,i) + "];"
254             name = ['/Trans', num2str(i)];
255
256             add_trans(name,POSITION,trans);
257             add_line(gcs, 'Conf/RConn1',...
258                     ['Trans', num2str(i), '/LConn1']);
259             add_line(gcs, ['Trans', num2str(i), '/RConn1'],...
260                     ['6DOF', num2str(i), '/LConn1']);

```

## A. Appendices

---

```

261
262 POSITION.x_level = POSITION.x_level + POSITION.x_gap_small + POSITION.x_gap_large;
263 nodes_used = [nodes_used;i];
264 parent_name = ['6DOF', num2str(i)];
265 parent_rot_1 = 0;
266 parent_rot_2 = 0;
267
268 [members_used,nodes_used,end_nodes,...
269  strut_orient,missing_sph,strut_rot,POSITION] = ...
270  add_node(i, joints_planar,members_used,nodes_used,...
271          recurs_layer_nr,parent_name,parent_rot_1,parent_rot_2,POSITION,...
272          missing_sph,strut_rot,s_laying,end_nodes,strut_orient,strut,type,N,C,r_s,rho_s);
273
274 POSITION.x_level = POSITION.x_level - POSITION.x_gap_large;
275 end
276 end
277 end
278
279 %% Cables and their connection with struts
280
281 POSITION.y_level = POSITION.y_level - m_s*POSITION.y_gap;
282 POSITION.x_level = POSITION.x_level + POSITION.x_gap_small + 2*POSITION.x_gap_large;
283
284 for i = 1:m_c % through all cables
285
286 name = ['/m', num2str(cable(1,i))];
287 add_cable(name,POSITION,num2str(A_c),num2str(E_c),num2str(beta)); % add cable
288 add_actuation(num2str(cable(1,i)),num2str(l_0_seq(i,:)),num2str(T),POSITION); % add actuator
289
290 for side = 1:2 % through both ends of the cable
291 [lr,strut_index] = find(strut([2,6],:) == cable(side*4-2,i));
292 % lr - port is on the left side or right side?
293 % strut_index - strut index in strut parameter matrix
294 % cable(side*4-2,i) - nr of first/second node
295 if length(lr) > 1
296 lr = lr(1);
297 strut_index = strut_index(1);
298 end
299 if lr == 1
300 if strut_orient(strut_index) == 1
301 LR = 'L';
302 else
303 LR = 'R';
304 end
305 else
306 if strut_orient(strut_index) == 1
307 LR = 'R';
308 else
309 LR = 'L';
310 end
311 end
312 if side == 1 % connection of appropriate end of appropriate strut to cable's end
313 add_line(gcs,['m', num2str(cable(1,i)), '/LConn1'],...
314          ['M', num2str(strut(1,strut_index)), '//',num2str(LR),'Conn1']);
315 else
316 add_line(gcs,['m', num2str(cable(1,i)), '/RConn1'],...
317          ['M', num2str(strut(1,strut_index)), '//',num2str(LR),'Conn1']);
318 end
319 end
320 POSITION.y_level = POSITION.y_level + POSITION.y_gap;
321 end
322
323 %% Add 1 scope for all cables (force)
324
325 POSITION.y_level = POSITION.y_level - m_c*POSITION.y_gap;
326 POSITION.x_level = POSITION.x_level + POSITION.x_gap_large;

```

## A. Appendices

---

```

327
328 if m_c ~= 0 % some cable must exist
329     add_block('built-in/Scope', [gcs,'/Scope_m_force'],...
330         'Position', [POSITION.x_level-POSITION.size_half_small(1),...
331             POSITION.y_level-POSITION.size_half_small(2),...
332             POSITION.x_level+POSITION.size_half_small(1),...
333             POSITION.y_level+POSITION.size_half_small(2)],...
334         'Orientation',0,'NumInputPorts',num2str(m_c));
335 end
336
337 for i = 1:m_c % through all cables
338     add_line(gcs,['m', num2str(cable(1,i)), '/1'],...
339         ['Scope_m_force', '/', num2str(i)]);
340 end
341
342 %% Add missing constraints to close cycles
343
344 POSITION.x_level = POSITION.x_level + POSITION.x_gap_large;
345
346 missing_sph_nodes = find(missing_sph); % mark indices of nodes with missing spherical joint
347 for i = 1:length(missing_sph_nodes) % through all nodes with missing sph. joints
348
349     struts_in_node = find(C(:,missing_sph_nodes(i))); % all struts connected to the node
350     struts_in_node = struts_in_node.*(-type(struts_in_node)+ones(length(struts_in_node),1));
351     struts_in_node = struts_in_node(find(struts_in_node));
352
353     first_member = missing_sph(missing_sph_nodes(i)); % always on the right (F) side
354     if struts_in_node(1) == first_member
355         second_member = struts_in_node(2);
356     else
357         second_member = struts_in_node(1);
358     end
359
360     [lr,strut_index] = find(strut([2,6],:) == missing_sph_nodes(i));
361     % lr - port to node is on left (B) side or right (F) side?
362     % strut_index - strut index in strut parameter matrix
363     second_member_side = lr(find(strut_index == second_member));
364
365     if second_member_side == 1
366         if strut_orient(second_member) == 1
367             LR = 'L';
368         else
369             LR = 'R';
370         end
371     else
372         if strut_orient(second_member) == 1
373             LR = 'R';
374         else
375             LR = 'L';
376         end
377     end
378
379     name = ['/Rot*-', num2str(first_member)];
380     add_rot_2(name,POSITION,... % realize reverse rotation relative to first_member
381         "" + (-strut_rot(first_member,2)),"" + (-strut_rot(first_member,1)));
382
383     POSITION.x_level = POSITION.x_level + POSITION.x_gap_small;
384     add_line(gcs,['Rot*-', num2str(first_member), '/LConn1'],...
385         ['M', num2str(first_member), '/RConn1']);
386
387     lib_path = 'sm_lib/Joints/Spherical Joint';
388     name = ['/Spherical*', num2str(missing_sph_nodes(i))];
389     add_joint(name,POSITION,lib_path,0); % add missing joint
390     POSITION.x_level = POSITION.x_level + POSITION.x_gap_small;
391     add_line(gcs,['Spherical*', num2str(missing_sph_nodes(i)), '/LConn1'],...
392         ['/Rot*-', num2str(first_member), '/RConn1']);

```



## A. Appendices

---

```

393
394 name = ['/Rot*-', num2str(second_member)];
395 add_rot_2(name,POSITION,... % realize reverse rotation relative to first_member
396     "" + (-strut_rot(second_member,2)),"" + (-strut_rot(second_member,1)));
397
398 POSITION.x_level = POSITION.x_level + POSITION.x_gap_small;
399 add_line(gcs,['Rot*-', num2str(second_member), '/RConn1'],...
400     ['Spherical*', num2str(missing_sph_nodes(i)), '/RConn1']);
401 add_line(gcs,['Rot*-', num2str(second_member), '/LConn1'],...
402     ['M', num2str(second_member), '/', num2str(LR), 'Conn1']);
403
404 POSITION.y_level = POSITION.y_level + POSITION.y_gap;
405 POSITION.x_level = POSITION.x_level - 3*POSITION.x_gap_small;
406 end
407
408 %% End of model generation
409
410 save_system(fname);
411 open_system(fname);
412
413 end
414
415
416 %% Support functions
417
418 function add_joint(name,POSITION,lib_path,planar)
419     add_block(lib_path, [gcs,name],...
420         'Position', [POSITION.x_level-POSITION.size_half_small(1),...
421             POSITION.y_level-POSITION.size_half_small(2),...
422             POSITION.x_level+POSITION.size_half_small(1),...
423             POSITION.y_level+POSITION.size_half_small(2)],...
424         'Orientation',0);
425     if planar == 1
426         set_param([gcs,name],...
427             'PzTorqueActuationMode','ComputedTorque','PzMotionActuationMode','InputMotion');
428         name2 = name(2:length(name)); % without '/' at the beginning
429         add_line(gcs,'S-PS/RConn1',[name2, '/LConn2']);
430     end
431 end
432
433 function add_trans(name,POSITION,trans)
434     add_block('sm_lib/Frames and Transforms/Rigid Transform', [gcs,name],...
435         'Position', [POSITION.x_level-POSITION.size_half_small(1),...
436             POSITION.y_level-POSITION.size_half_small(2),...
437             POSITION.x_level+POSITION.size_half_small(1),...
438             POSITION.y_level+POSITION.size_half_small(2)],...
439         'Orientation',0,'TranslationMethod','Cartesian',...
440         'TranslationLengthUnit','m','TranslationCartesianOffset',trans);
441 end
442
443 function add_rot(name,POSITION,rot_1,rot_2)
444     add_block('sm_lib/Frames and Transforms/Rigid Transform', [gcs,name],...
445         'Position', [POSITION.x_level-POSITION.size_half_small(1),...
446             POSITION.y_level-POSITION.size_half_small(2),...
447             POSITION.x_level+POSITION.size_half_small(1),...
448             POSITION.y_level+POSITION.size_half_small(2)],...
449         'Orientation',0,'RotationMethod','RotationSequence','RotationSequence','YXY',...
450         'RotationSequenceAnglesUnits','deg',...
451         'RotationSequenceAngles','[' + rot_1 + ', ' + rot_2 + ', 0]');
452 end
453
454 function add_rot_2(name,POSITION,rot_1,rot_2)
455     add_block('sm_lib/Frames and Transforms/Rigid Transform', [gcs,name],...
456         'Position', [POSITION.x_level-POSITION.size_half_small(1),...
457             POSITION.y_level-POSITION.size_half_small(2),...
458             POSITION.x_level+POSITION.size_half_small(1),...

```

## A. Appendices

---

```

459         POSITION.y_level+POSITION.size_half_small(2)],...
460     'Orientation',0,'RotationMethod','RotationSequence','RotationSequence','YXY',...
461     'RotationSequenceAnglesUnits','deg',...
462     'RotationSequenceAngles',[0,' + rot_1 + ',' + rot_2 + '']);
463 end
464
465 function add_strut(name,POSITION,l,r,rho)
466     add_block('my_lib/Strut', [gcs,name],...
467         'Position', [POSITION.x_level-POSITION.size_half_large(1),...
468             POSITION.y_level-POSITION.size_half_large(2)],...
469             POSITION.x_level+POSITION.size_half_large(1),...
470             POSITION.y_level+POSITION.size_half_large(2)],...
471         'Orientation',0,'L',num2str(l),'r',num2str(r),'rho',num2str(rho));
472 end
473
474 function add_cable(name,POSITION,A,E,beta) %add_cable(name,POSITION,k,b)
475     add_block('my_lib/Cable with actuation', [gcs,name],...
476         'Position', [POSITION.x_level-POSITION.size_half_large(1),...
477             POSITION.y_level-POSITION.size_half_large(2)],...
478             POSITION.x_level+POSITION.size_half_large(1),...
479             POSITION.y_level+POSITION.size_half_large(2)],...
480         'Orientation',0,'A',A,'E',E,'beta',beta,'f_s',1);
481 end
482
483 function add_actuation(num_cable,cable_l_0_seq,T,POSITION)
484     add_block('my_lib/Linear interpolation of periodically sampled values',...
485         [gcs,['/10_', num_cable]],...
486         'Position', [10,...
487             250 + POSITION.y_level-POSITION.size_half_large(2),...
488             10 + 2*POSITION.size_half_large(1),...
489             250 + POSITION.y_level+POSITION.size_half_large(2)],...
490         'Orientation',0,'vector', ['[', cable_l_0_seq, ']', 'period',T];
491     add_line(gcs,['/10_', num_cable, '/1'], ['m', num_cable, '/1']);
492 end
493
494 function bool_added = add_nodal_sensor(node,POSITION,parent_name)
495     bool_added = 0;
496     sensor_exist = find_system(gcs,'Name',['Sensor_', num2str(node)]);
497     if isempty(sensor_exist)
498         add_block('my_lib/Position sensor', [gcs,['/Sensor_', num2str(node)],...
499             'Position', [POSITION.x_level_sensor-POSITION.size_half_small(1),...
500                 70-POSITION.size_half_small(2)],...
501                 POSITION.x_level_sensor+POSITION.size_half_small(1),...
502                 70+POSITION.size_half_small(2)],...
503             'Orientation',0);
504         add_line(gcs,['Sensor_', num2str(node), '/LConn1'],'Conf/RConn1');
505         add_line(gcs,['Sensor_', num2str(node), '/RConn1'],[parent_name,'/RConn1']);
506
507         add_block('simulink/Sinks/To Workspace', [gcs,['toWS_', num2str(node)],...
508             'Position', [POSITION.x_level_sensor-POSITION.size_half_small(1),...
509                 10-POSITION.size_half_small(2)],...
510                 POSITION.x_level_sensor+POSITION.size_half_small(1),...
511                 10+POSITION.size_half_small(2)],...
512             'Orientation',2,'VariableName',['position_', num2str(node)],'SaveFormat','Array');
513         add_line(gcs,['Sensor_', num2str(node), '/1'], ['toWS_', num2str(node), '/1']);
514         bool_added = 1;
515     end
516 end
517
518 function [members_used,nodes_used,end_nodes,...
519     strut_orient,missing_sph,strut_rot,POSITION] = ...
520     add_node(node,joints_planar,members_used,nodes_used,...
521         recurs_layer_nr,parent_name,parent_rot_1,parent_rot_2,POSITION,...
522         missing_sph,strut_rot,s_laying,end_nodes,strut_orient,strut,type,N,C,r,rho)
523
524     recurs_layer_nr = recurs_layer_nr + 1; % another recurrence of the fcn 'add_node'

```

## A. Appendices

---

```

525
526 struts_in_node = find(C(:,node));
527 struts_in_node = struts_in_node.*(-type(struts_in_node)+ones(length(struts_in_node),1));
528 struts_in_node = struts_in_node(find(struts_in_node)); % struts connected to the node
529 length_nodes_used = length(nodes_used(:,1)); % length of the vector nodes_used
530
531 bool_added = add_nodal_sensor(node,POSITION,parent_name); % try to add position sensor
532 if bool_added == 1
533     POSITION.x_level_sensor = POSITION.x_level_sensor + 4*POSITION.size_half_small(1);
534 end
535
536 if length(struts_in_node) >= 2 % more than 2 struts in the node
537     parent_member = str2double(strrep(parent_name,'M',''));
538     missing_sph(node) = parent_member;
539 else
540     missing_sph(node) = 0; % no sph. joints are missing in the node
541 end
542
543 for j = 1:length(struts_in_node) % through all struts in the node
544
545     current_member = struts_in_node(j); % current strut index
546     current_member_nodes = strut([2,6],current_member); % indices of strut's nodes
547
548     if isempty(find(members_used == current_member)) % strut not added to model yet
549
550         if (recurs_layer_nr == 1)...
551             || ( isempty(find(joints_planar == current_member_nodes(1))) &&...
552                 (isempty(find(joints_planar == current_member_nodes(2)))) )
553
554             missing_sph(node) = 0;
555
556             members_used = [members_used;current_member];
557
558             if recurs_layer_nr ~= 1 % not 1st recurrence of fcn 'add_node'
559
560                 POSITION.x_level = POSITION.x_level - 2*POSITION.x_gap_small;
561                 name = ['/Rot-', num2str(current_member)];
562                 add_rot_2(name,POSITION,"" + (-parent_rot_2)," " + (-parent_rot_1)); % realize reverse rot.
563                 POSITION.x_level = POSITION.x_level + 2*POSITION.x_gap_small;
564
565                 POSITION.x_level = POSITION.x_level - POSITION.x_gap_small;
566                 lib_path = 'sm_lib/Joints/Spherical Joint';
567                 name = ['/Sph', num2str(current_member)];
568                 add_joint(name,POSITION,lib_path,0); % add spherical joint
569                 POSITION.x_level = POSITION.x_level + POSITION.x_gap_small;
570
571             end
572
573             if node == strut(2,current_member) % the node is start node of the member
574                 first = N(:,current_member_nodes(1));
575                 second = N(:,current_member_nodes(2));
576                 strut_orient(current_member) = 1;
577             else
578                 first = N(:,current_member_nodes(2));
579                 second = N(:,current_member_nodes(1));
580                 strut_orient(current_member) = -1;
581             end
582             rot_1 = atan2((second(1) - first(1)),(second(3) - first(3)))*180/pi; % precession angle
583             rot_2 = -asin((second(2) - first(2))/norm(first - second))*180/pi; % nutation angle
584
585             name = ['/Rot', num2str(current_member)];
586             add_rot(name,POSITION,"" + rot_1," " + rot_2); % realize absolute rotation
587             strut_rot(current_member,1) = rot_1; % save the value of precession angle
588             strut_rot(current_member,2) = rot_2; % save the value of nutation angle
589
590             POSITION.x_level = POSITION.x_level + POSITION.x_gap_small;

```

## A. Appendices

---

```

591     name = ['/M', num2str(current_member)];
592     add_strut(name, POSITION, strut(10, current_member), r, rho); % add strut
593
594     if recurs_layer_nr ~= 1
595         add_line(gcs, ['Rot', num2str(current_member), '/RConn1'], ...
596                     ['M', num2str(current_member), '/LConn1']);
597         add_line(gcs, ['Rot', num2str(current_member), '/LConn1'], ...
598                     ['Sph', num2str(current_member), '/RConn1']);
599         add_line(gcs, ['Sph', num2str(current_member), '/LConn1'], ...
600                     ['Rot-', num2str(current_member), '/RConn1']);
601         add_line(gcs, ['Rot-', num2str(current_member), '/LConn1'], ...
602                     [parent_name, '/RConn1']);
603     else
604         add_line(gcs, ['Rot', num2str(current_member), '/RConn1'], ...
605                     ['M', num2str(current_member), '/LConn1']);
606         add_line(gcs, ['Rot', num2str(current_member), '/LConn1'], ...
607                     [parent_name, '/RConn1']);
608     end
609
610     POSITION.x_level = POSITION.x_level - POSITION.x_gap_small;
611     POSITION.y_level = POSITION.y_level + POSITION.y_gap;
612
613     parent_name_inner = ['M', num2str(current_member)]; % save strut's name for its followers
614
615     % first or second node of strut has been added now?
616     if current_member_nodes(1) == nodes_used(length_nodes_used)
617         if isempty(find(s_laying == current_member))
618             nodes_used = [nodes_used; current_member_nodes(2)];
619         end
620         % add second node of strut
621         [members_used, nodes_used, end_nodes, ...
622          strut_orient, missing_sph, strut_rot, POSITION] = ...
623          add_node(current_member_nodes(2), joints_planar, members_used, nodes_used, ...
624                  recurs_layer_nr, parent_name_inner, rot_1, rot_2, POSITION, ...
625                  missing_sph, strut_rot, s_laying, end_nodes, strut_orient, strut, type, N, C, r, rho);
626     else
627         if isempty(find(s_laying == current_member))
628             nodes_used = [nodes_used; current_member_nodes(1)];
629         end
630         % add first node of strut
631         [members_used, nodes_used, end_nodes, ...
632          strut_orient, missing_sph, strut_rot, POSITION] = ...
633          add_node(current_member_nodes(1), joints_planar, members_used, nodes_used, ...
634                  recurs_layer_nr, parent_name_inner, rot_1, rot_2, POSITION, ...
635                  missing_sph, strut_rot, s_laying, end_nodes, strut_orient, strut, type, N, C, r, rho);
636     end
637 end
638 end
639 end
640
641 end

```