A Fully Compliant Pendulum Balancer with a Spherical Range of Motion

R.L.B. Barendse

TU Delft

A Fully Compliant Pendulum Balancer with a Spherical Range of Motion

by

R.L.B. Barendse

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday September 9, 2024 at 14:00.

Student number:	4579178	
Project Duration:	Sept. 4, 2023 – Sept. 9, 2024	
Thesis committee:	Prof.dr.ir J.L. Herder,	TU Delft, Chair
	Dr.ir. G. Radaelli,	TU Delft, Supervisor
	Dr. M.J. Mirzaali	TU Delft

This thesis is confidential and cannot be made public until September 9, 2025

An electronic version of this thesis is available at https://repository.tudelft.nl/.

TU Delft

Preface

After spending 8 full years at TU Delft, I can barely imagine my life beyond being a mechanical engineering student. Designing, building and testing mechanical projects is not only a major part of my education, but also my hobbies. In this thesis, I was able to build a test setup for a fully compliant pendulum balancer that I designed, making it almost feel like a massive hobby project.

Many setbacks were encountered over the past year, which extended the duration of the thesis further than I had hoped. Despite these challenges, I can say that I am truly proud of the work performed in this thesis and I owe gratitude to the people who helped me along the way.

First and foremost, I would like to express my gratitude to Giuseppe. The development of the TetraFEM tool would not have succeeded without his patience and trust in me, as at some points it seemed like it would never work. Our many meetings were always insightful and I cannot thank him enough for his supervision during my internship and my thesis. Furthermore I would like to thank the members of the ShellSkeletons group for their advise throughout the year. I also owe gratitude to Xander Burgerhout for helping me with setting up the optimizations. I would also like to thank Patrick van Holst and Gideon Emmaneel for their friendly support in the labs. I am also thankful towards my predecessors Jelle Rommers and Dion Hogervorst and I hope others will continue with this project as well. Finally, I would like to thank my family, friends and girlfriend for their love and support throughout my life. They make me the person I am proud of.

R.L.B. Barendse
Delft, September 2024

Contents

1

Paper

This chapter presents the main contribution of this thesis. The paper introduces a fully compliant pendulum balancer with a spherical range of motion, a novel concept not previously addressed in the available literature. Additonally, the paper introduces the TetraFEM tool, a computationally efficient algorithm developped for calculating and optimizing deflections of a compliant spherical joint. Both the performance of the prototype and the accuracy of the algorithm were tested, yielding satisfactory results.

A Fully Compliant Pendulum Balancer with a Spherical Range of Motion

Riley Barendse, Giuseppe Radaelli

***Abstract*—This paper introduces a fully compliant spherical joint with an optimized stiffness profile specifically for balancing a pendulum. The design builds on previous work that has successfully created a fully compliant spherical joint using tetrahedron-shaped elements connected in series. To efficiently compute and optimize the balancing behaviour of these compliant joints, a novel algorithm is developed and presented. Using this algorithm, optimizations are conducted to obtain simulated pendulum balancers under five different conditions. The performances of these results are analysed to assess the potential and limitations of the algorithm and these spherical joints. Based on one of the optimized results, a prototype is fabricated and expermentally validated, achieving a moment reduction of 90.5%. The deformation calculated by the TetraFEM tool closely matches the prototype's deformation with an accuracy of 89.6%, demonstrating its potential for application in the development of shoulder exoskeletons.**

I. Introduction

A pendulum is a device with a mass and a pivot point which is inherently unstable in the upright position due to gravitational forces. A compensating mechanism is required to prevent the mass from falling to the lower stable position. Such gravity compensation mechanisms can be classified based on energy utilization as active gravity compensation and passive gravity compensation [1]. An active gravity compensator requires a set of actuators and expends energy while balancing. The shoulder muscles used to keep a human arm in upright position exemplify an active way of compensating the gravitational forces of a pendulum (i.e. a human arm).

Passive gravity balancers do not expend energy while balancing a pendulum and most of them traditionally use counterweights, like in bascule bridges or cranes. A drawback of this method is a significant increase in the mass, inertia and volume of the system. An alternative method to balance pendulums, which does not suffer the aforementioned drawbacks, is the utilization of springs [2]. Linear springs have found usage as a passive gravity compensator for devices like desk lamps [3], robotic arms [4, 5] and exoskeletons [6, 7]. Although both linear springs and counterweights can effectively counteract the gravitational forces of a pendulum, they do not address friction forces within rigid linked systems. Overcoming this friction is still necessary before any movement in such systems can occur.

Compliant mechanisms on the other hand can move without friction as their motion is gained from the deflection of elastic members [8] rather than sliding or rolling contact. Other advantages of compliant mechanisms include less weight, less assembly parts and no backlash or wear [9]. Although compliant mechanisms experience no friction, they usually do have a non-zero stiffness which is often undesired. The deflection of the elastic members requires energy to gain motion and a continuous force on the compliant joint is required to remain in a certain deflected configuration.

A pendulum has potential energy and the gravitational forces acting on it are continuous. Some compliant joints can be designed such that the strain energy stored in the elastic members equals the potential energy of a pendulum, making the combined mechanism statically balanced. This is an elegant and energy efficient way to balance a pendulum over its rotational (1 dimensional) or spherical (2 dimensional) range of motion (ROM). Compliant joints with a spherical ROM and a remote center of rotation, meaning it rotates around a point which is not located within the physical joint, can have several practical applications, such as in the field of exoskeletons. Such a (passive) compliant pendulum balancer could be used as the functional component of an external shoulder joint, in case the wearer's shoulder muscles lack the strength for (active) gravity compensation.

Quite some research has already been done on pendulum balancers using compliant joints. Several have succeeded in developing a pendulum balancing joint which is fully compliant, meaning without the use of any rigid joints. Radaelli et al. optimized the shape of a single beam with constant thickness and both ends clamped such that it could successfully balance a mass over an almost circular arc [10]. Radaelli et al. also optimized the shape of a compliant shell mechanism such that it could balance its self-weight with an additional payload over one degree of freedom (DOF) [11]. Rijff et al. designed and optimized a pendulum balancing four-bar mechanism where he replaced the rigid joints with torsional springs [12]. Rommers et al. presented a design of a passive origami-like mechanism which could balance a pendulum [13]. Abouheidari et al. did research on the optimization and syntheses of a gravity-balancing torque-angle profile using a compliant helicoidal shell mechanism [14]. Although these mechanisms were fully compliant and could successfully balance a pendulum, the ROM was limited to a single rotational DOF.

Nobaveh et al. presented a passive wrist support using two optimized compliant spatial beams which significantly reduces the muscle force required for flexion and extension of the wrist, while remaining reletively compliant in supination and pronation direction [15]. This mechanism balances the weight of the hand, is fully compliant and has 2 DOF. However, the ROM is quite limited and is a pitch-roll motion instead of spherical.

Few have researched the possibilities of a compliant pendulum balancer with a spherical ROM. Tschiersky et al. explored the use of mechanisms that employ a flexible element which connects the supported arm to an attachment at the back, such that the necessary moment to lift the arm is significantly reduced [16]. Van der Kemp et al. designed a compact arm support using a compliant shell mechanism to reduce fatigue in the arms of surgeons during laparoscopic surgery [17]. Although both of these pendulum balancers have a spherical range of motion (ROM), they still rely on a rigid revolute joint for the medial/lateral rotation of the shoulder. Additionally, the compact arm support requires a linear guide and only balances a small portion of the arm's weight. The flexible elements from Tschiersky also protrude significantly from the body, making it aesthetically unappealing.

In their literature study on fully compliant remote center of motion mechanisms, Mak et al. [18] concluded that curved leaf flexures offer the most potential for use in exoskeletons, owing to their large range of motion and compact form factor. Some of these addressed mechanisms offer a spherical ROM [19–27], but none of them are optimized for balancing a pendulum. A fully compliant pendulum balancer with a spherical ROM is currently lacking in the literature, despite its great potential in fields like that of exoskeletons.

This paper aims to contribute in the development of shoulder exoskeletons by presenting the first fully compliant pendulum balancer with a spherical ROM, as well as explaining and validating the tool developed for predicting and optimizing its balancing behaviour.

The paper is structured as follows: Section II explains the design on which the novel pendulum balancer is based and elaborates on its geometry, the simulated model in the 'TetraFEM tool', the optimization and the experimental validation setup. Section III shows the results of the optimization and the experimental validation. After that the results, observations, limitations and future work are discussed in section IV and section V will summarize and conclude this paper.

II. Method

This section describes the methodology used to obtain the fully compliant pendulum balancer with a spherical range of motion. The Tetra I design from Rommers et al. [19] (see Figure 1) lays the foundation for the novel pendulum balancer and this section will therefore start by describing the design principles of this fully compliant spherical joint. Next, the geometry and parameters from which the novel pendulum balancer is built will be defined. Following this, the TetraFEM tool will be explained, which calculates and visualizes deformation based on the balancer's parameters and a certain load. Subsequently, the optimization process is described, which utilizes the TetraFEM tool to optimize the geometric parameters for balancing a given pendulum. Finally, the test setup and the measurement procedure for the produced prototype is described, which should experimentally verify the TetraFEM tool and the pendulum balancing properties of the prototype.

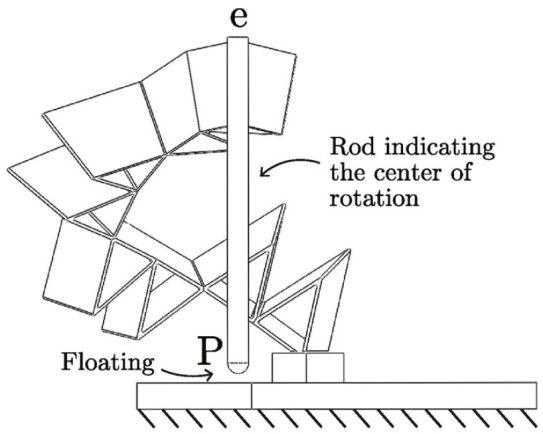

Fig. 1: The fully compliant spherical joint 'Tetra I', with its center of rotation at P. Retrieved from Rommers et al. [19].

A. Tetra I design principle

The Tetra I is one of the best performing curved leaf flexure mechanisms in the aforementioned literature study by Mak et al. [18]. It consists of a number of tetrahedron elements connected in series without intermediate bodies. Each tetrahedron is formed by three connected flexures with all inner and outer edges pointing towards its center of rotation P (see Figure 2). The tetrahedra are compliant for two rotational directions and stiff in the other four, creating a joint that is compliant for all rotations around point P, but very stiff for all translations. This design ensures a nearly constant center of rotation of the spherical joint. Although the Tetra I is not presented as a pendulum balancer, it contains many geometric parameters which can be optimized to obtain a desired moment profile. This is illustrated by Hogervorst et al. [28], who optimized the parameters of the Tetra I to obtain an axi-symmetrical energy field.

B. Geometry of the pendulum balancer

This subsection aims to define the generic geometry of the novel pendulum balancing joint and its tetrahedron elements. The novel joint consists of n tetrahedra connected in series. The first tetrahedron, T_1, is fixed on edge a to the environment and the last tetrahedron, T_n, is attached to a pendulum at edge c. Each tetrahedron consists of three blade flexures: AB, AC and BC, with AB and BC being equal in length. The independent geometric parameters which dictate the shape and dimensions of each tetrahedron are shown in Figure 3. The other independent geometric parameter γ determines the angle between each tetrahedron (T_i) and is illustrated in Figure 4. In this study, the parameters α and γ can have different corresponding values for each tetrahedron, but t_{min}, R_{in} and R_w remain the same value for all tetrahedra in this spherical joint.

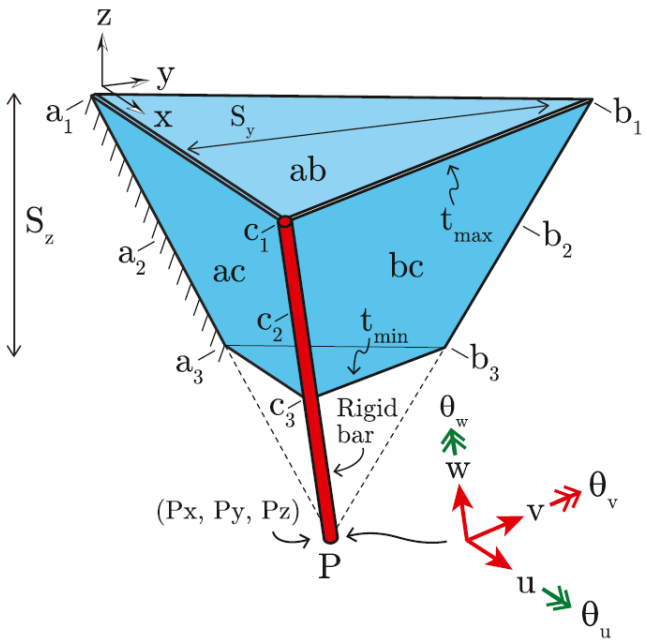

Fig. 2: A single tetrahedron element from the Tetra I design with its geometric parameters. It is compliant in the green coloured orientations θ_w and θ_u, but stiff in θ_v direction and all translations. The center of rotation is indicated by P. Retrieved from Rommers et al. [19].

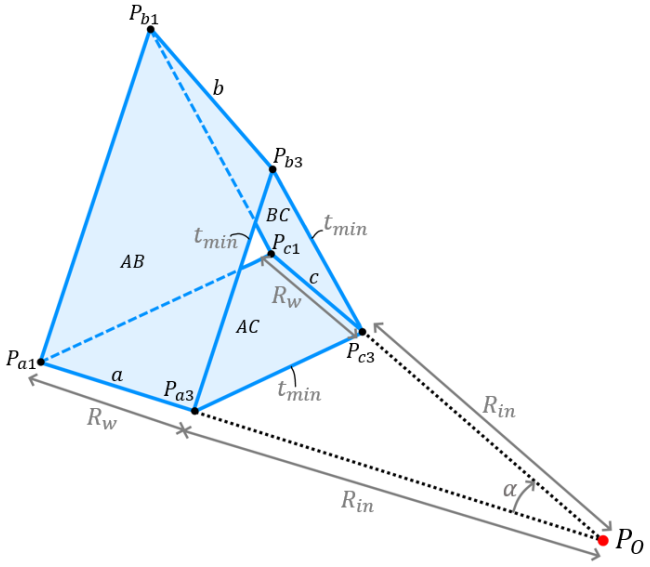

Fig. 3: The independent geometric parameters which form the tetrahedra.

Angular height: The angular height β (see Figure 5) of the tetrahedron is calculated using the aforementioned independent geometric parameters to minimize parasitic motion. The height of the tetrahedron as depicted in Figure 5 is determined using a formula from Rommers et al. [19] which minimizes parasitic motion if the shape of the tetrahedron is approximated

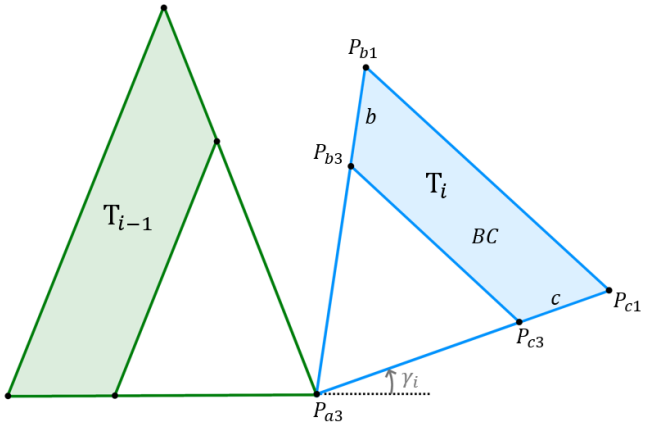

Fig. 4: A view perpendicular to edge a of T_i showing two tetrahedra and the independent geometric parameter γ_i, which determines the angle between the current tetrahedron (T_i) and the previous tetrahedron (T_{i-1}).

as a prism:

$$H = \frac{1}{2}\left[\sqrt{4L_{a2c2}^4 + \frac{48L_{a2c2}^2 R_{w\_m}^2(\nu+1)}{5}} + L_{a2c2}^2 + \frac{12R_{w\_m}^2(\nu+1)}{5}\right]^{\frac{1}{2}}, \tag{1}$$

where ν is Poisson's ratio $\nu = E/(2G) - 1$. The other terms

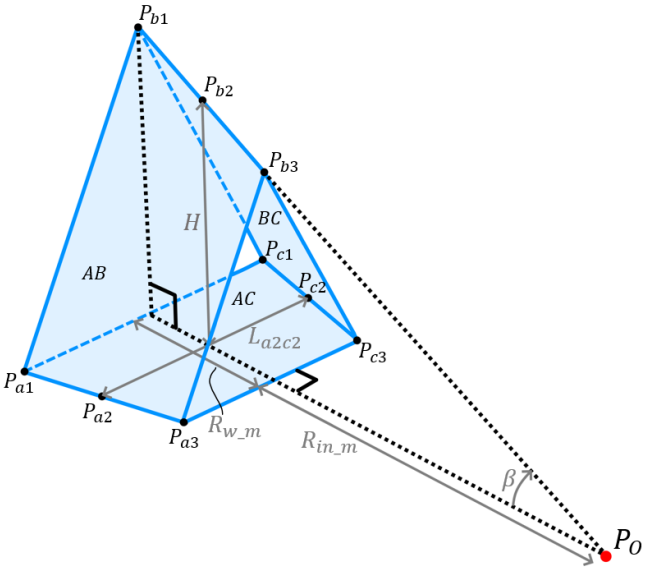

Fig. 5: A visualisation of the parameters necessary for calculating the (angular) height of the tetrahedron. Point P_{a2}, P_{b2} and P_{c2} are the midpoints of the edges on which they are located.

used in equation 1 are illustrated in Figure 5 as well and can be derived from the independent parameters:

$$L_{a2c2} = 2\sin\left(\frac{\alpha}{2}\right)\left(R_{in} + \frac{R_w}{2}\right), \tag{2}$$

$$R_{w\_m} = R_w \cos\left(\frac{\alpha}{2}\right). \tag{3}$$

The angular height β of the tetrahedron can be calculated using H from equation 1 and R_{in_m}:

$$R_{in\_m} = R_{in} \cos\left(\frac{\alpha}{2}\right), \tag{4}$$

$$\beta = \arctan\left(\frac{H}{R_{in\_m} + R_{w\_m}/2}\right). \tag{5}$$

Cross-section parameters: Similar to the Tetra I design, the flexures of the tetrahedra have a trapezoidal shaped cross-section, which is illustrated in Figure 6. On the edges between P_{a3}, P_{b3} and P_{c3} the thickness is the lowest with t_{min}, while on the edges between P_{a1}, P_{b1} and P_{c1} the thickness has the highest value with t_{max}. The equation for t_{max} is given below:

$$t_{max} = \frac{R_{in} + R_w}{R_{in}} \cdot t_{min}. \tag{6}$$

M_c is the centroid of the trapezoidal cross-section along edge c, as depicted in Figure 6. The distance R_{Mo} between M_c and the center of rotation P_O can be calculated using the following equation:

$$R_{Mo} = R_{in} + R_w - R_w \frac{t_{max} + 2t_{min}}{3(t_{max} + t_{min})}. \tag{7}$$

As edge a has the same cross-section and dimensions as edge c, the relative location of centroid M_a on edge a is the same as that of centroid M_c on edge c.

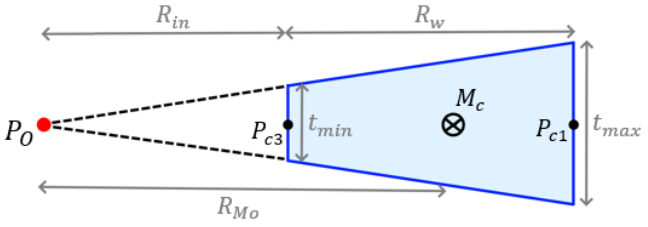

Fig. 6: The cross-section at edge c shown with a trapezoidal shape due to its variable thickness.

Flexure weight: The weight of the flexures cannot always be neglected and therefore the mass and Center of Gravity (COG) of each flexure need to be defined as well. Looking at Figure 7, it can be observed that flexure AC (similar to AB and BC) has the shape of a chopped off pyramid with a narrow rectangular base. The base has a width of t_{max} and a length of L_{a1c1}, which is the distance between P_{a1} and P_{c1}. The chopped off part of the pyramid has a base with a width of t_{min} and a length of L_{a3c3}, which is the distance between P_{a3} and P_{c3}. The relations between the independent geometric parameters and L_{a1c1} and L_{a3c3} respectively are defined as:

$$L_{a1c1} = 2\sin\left(\frac{\alpha}{2}\right)(R_{in} + R_w) \tag{8}$$

$$L_{a3c3} = 2\sin\left(\frac{\alpha}{2}\right)(R_{in}) \tag{9}$$

The mass of the flexure can be calculated by subtracting the volume of the small missing pyramid from the larger pyramid and multiplying the result by the density (ρ) of the flexure material. The equations for the volume of the small pyramid V_3, the volume of the large pyramid V_1 and the mass m_{AC} of flexure AC are defined as:

$$V_3 = \frac{t_{min} L_{a3c3} R_{in\_m}}{3} \tag{10}$$

$$V_1 = \frac{t_{max} L_{a1c1} (R_{in\_m} + R_{w\_m})}{3} \tag{11}$$

$$m_{AC} = \rho(V_1 - V_3). \tag{12}$$

The COG of a pyramid measured from the top (i.e. P_O) is 3/4 of the height of the pyramid. The COG of the large and small pyramids are required to calculate R_{COG_AC}, which is the distance between P_O and the COG of flexure AC (see Figure 7). This relation can be written as:

$$R_{COG\_3} = \frac{3R_{in\_m}}{4} \tag{13}$$

$$R_{COG\_1} = \frac{3(R_{in\_m} + R_{w\_m})}{4} \tag{14}$$

$$R_{COG\_AC} = \frac{R_{COG\_1} V_1 - R_{COG\_3} V_3}{V_1 - V_3} \tag{15}$$

where R_{COG_3} and R_{COG_1} are the distances between P_O and the COG of the small and the large pyramid respectively.

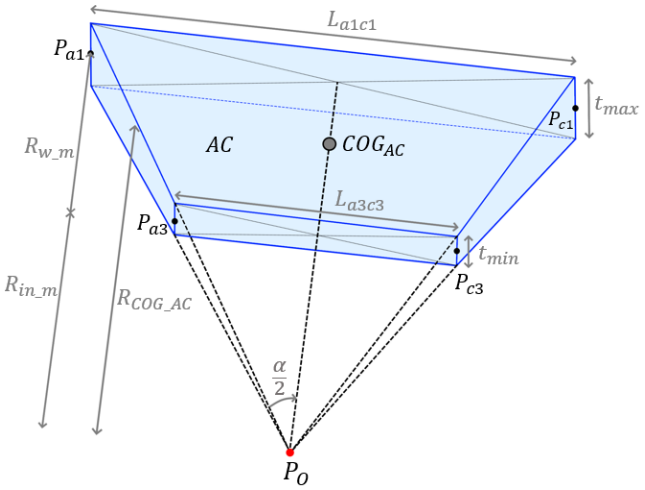

Fig. 7: Flexure AC depicted with its center of gravity at COG_{AC}.

C. TetraFEM tool

Calculating the balancing behavior of a simulated compliant joint can be quite computationally expensive. This is especially true when the balancing behavior needs to be optimized for a large number of independent geometric parameters, as is the case with the pendulum balancer discussed in this paper. The TetraFEM tool was developed as a computationally more efficient alternative compared to the currently available Finite Element Method (FEM) based simulation software. This FEM algorithm is written in Python and simulates the behaviour of a pendulum balancing spherical joint consisting of one or more tetrahedra connected in series. Efficient use of the chain algorithm [29] and several simplifications are applied to minimize computation time.

The main inputs of the TetraFEM tool are the independent geometric parameters of the joint, its relevant material properties, the mass and length of the pendulum and the intended range of motion. The main output of the TetraFEM tool is the calculated locations of deflected end-effector points corresponding to gravitational loads applied to the simulated model. The calculations done in the TetraFEM tool to obtain and visualize deflected end-effector points can be divided in four parts: The Flexure Compliance Analysis, the Tetrahedron Compliance Analysis, the Chain Algorithm and the Pendulum Balancing Test.

1) Flexure Compliance Analysis: First, the TetraFEM tool performs a flexure compliance analysis to obtain the compliance matrix of each flexure within the tetrahedra. This FEM analysis is heavily inspired by the work of F Pavari Rad et al. [21], who in return used the work of Zhang et al. [30] as a foundation to do a compliance analysis on a curved spherical flexure. The flexures in this flexure compliance analysis are assumed to be Timoshenko beams with one fixed end and one free end. This part describes how to calculate the compliance matrix of flexure AC, using this flexure compliance analysis.

Within each simulated tetrahedron, the TetraFEM tool considers flexure AC to be fixed on the edge of point M_a and subjected to an external load on its free end at point M_c. In addition, two global frames are defined as AC_a on point M_a and AC_c on point M_c (see Figure 8). A local frame AC_l is defined as well on local point M_{l_AC}, which is the centroid of a generic cross section of the flexure.

Stiffness matrix of *dl*: The generic external load acting on the free end (M_c) is balanced by a load acting on element dl. According to [21], dl as defined in Figure 8 has a stiffness matrix that can be written as:

$$^{AC\_l}\mathbf{K} = \mathrm{Diag}[EA_x, b_y GA_x, b_z GA_x, GJ, EI_y, EI_z], \quad (16)$$

where b_y and b_z are the shear coefficients and E and G are the Young's modulus and shear modulus. A_x, J, I_y and I_z are respectively the area, the torsional constant and the principal moments of inertia of the cross section of the flexure. As the

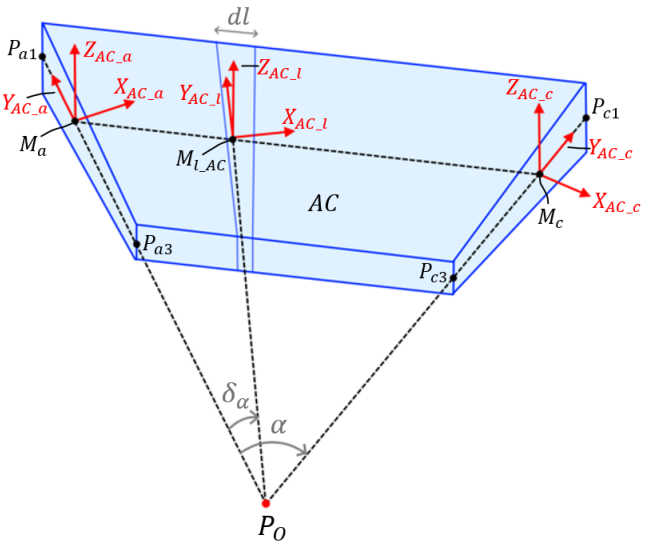

Fig. 8: The global and local coordinate systems used for the flexure compliance analysis of AC.

cross section of flexure AC is a trapezoid with a variable width (see Figure 9), these last four parameters are a function of δ_α:

$$R_{w\_M} = \frac{R_w}{\cos(\frac{\alpha}{2} - \delta_\alpha)}, \quad (17)$$

$$A_x = \frac{t_{max} + t_{min}}{2} \cdot R_{w\_M}, \quad (18)$$

$$I_y = \frac{R_{w\_M}(t_{min} + t_{max})(t_{min}^2 + t_{max}^2)}{48}, \quad (19)$$

$$I_z = \frac{R_{w\_M}^3(t_{min}^2 + 4t_{max}t_{min} + t_{max}^2)}{36(t_{max} + t_{min})}, \quad (20)$$

$$J = \frac{2}{3}\sin^3(\eta)(R_{out\_M}^4 - R_{in\_M}^4) - 16\sin^4(\eta)(V_L R_{out\_M}^4 + V_S R_{in\_M}^4), \quad (21)$$

where:

$$V_L = 0.10504 - 0.2\sin(\eta) + 0.3392\sin^2(\eta) - 0.53968\sin^3(\eta) + 0.82448\sin^4(\eta),$$

$$V_S = 0.10504 + 0.2\sin(\eta) + 0.3392\sin^2(\eta) + 0.53968\sin^3(\eta) + 0.82448\sin^4(\eta).$$

The other terms in equation 21 are defined as:

$$R_{in\_M} = \frac{R_{in}}{\cos(\frac{\alpha}{2} - \delta_\alpha)}, \quad (22)$$

$$R_{out\_M} = R_{in\_M} + R_{w\_M}, \quad (23)$$

$$\eta = \arctan\left(\frac{t_{min}}{2R_{in\_M}}\right) \quad (24)$$

Equation 21 is the formula for the torsional constant to account for the warping of annulus sector cross-sections [27]. The cross section of an annulus sector and a trapezoid are very similar

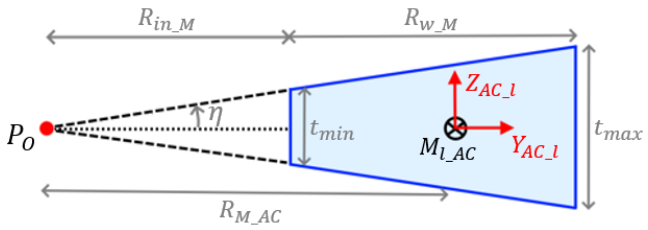

Fig. 9: The cross-section at local point M_{l_AC}. The dimensions η, R_{in_M}, R_{w_M}, and R_{M_AC} all depend on δ_α.

for $R_{w\_M} >> t_{max}$ and therefore this formula is used to calculate J.

Compliance matrix of flexure: The equation relating the stiffness matrix of element dl from equation 16 to the compliance matrix of the entire flexure is obtained from [21]:

$$^{AC\_c}\mathbf{C} = \int_l {}^{AC\_l}\mathbf{T}^T_{AC\_c} \cdot {}^{AC\_l}\mathbf{K}^{-1} \cdot {}^{AC\_l}\mathbf{T}_{AC\_c} \cdot dl. \quad (25)$$

Matrix $^{AC\_c}\mathbf{C}$ is the compliance matrix of flexure AC loaded at the free end M_c and expressed in frame AC_c. $^{AC\_l}\mathbf{T}_{AC\_c}$ in formula 25 is the adjoint transformation matrix relating local frame AC_l and global frame AC_c. The definition of this transformation matrix from [21] can be rewritten for flexure AC:

$$^{AC\_l}\mathbf{T}_{AC\_c} = \left[\begin{array}{c|c} {}^{AC\_l}\mathbf{R}_{AC\_c} & \mathbf{0} \\ \hline {}^{AC\_l}\tilde{\mathbf{r}}_{M\_c/M\_l\_AC} \cdot {}^{AC\_l}\mathbf{R}_{AC\_c} & {}^{AC\_l}\mathbf{R}_{AC\_c} \end{array}\right], \quad (26)$$

where $^{AC\_l}\mathbf{R}_{AC\_c}$ denotes the rotation matrix of frame AC_c with respect to AC_l. This rotation matrix can be derived by simply multiplying rotation matrices $^{AC\_l}\mathbf{R}_{AC\_a}$ with $^{AC\_a}\mathbf{R}_{AC\_c}$ according to [31]:

$$^{AC\_l}\mathbf{R}_{AC\_c} = {}^{AC\_l}\mathbf{R}_{AC\_a} \cdot {}^{AC\_a}\mathbf{R}_{AC\_c}, \quad (27)$$

where:

$$^{AC\_l}\mathbf{R}_{AC\_a} = \begin{bmatrix} \cos(\delta_\alpha) & -\sin(\delta_\alpha) & 0 \\ \sin(\delta_\alpha) & \cos(\delta_\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (28)$$

$$^{AC\_a}\mathbf{R}_{AC\_c} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (29)$$

Going back to equation 26, the term $^{AC\_l}\tilde{\mathbf{r}}_{M\_c/M\_l\_AC}$ denotes the skew symmetric matrix of $^{AC\_l}\mathbf{r}_{M\_c/M\_l\_AC}$, which is the position vector of local point M_{l_AC} relative to M_c with respect to local frame AC_l:

$$^{AC\_l}\tilde{\mathbf{r}}_{M\_c/M\_l\_AC} = \begin{bmatrix} 0 & -\mathbf{v}[2] & \mathbf{v}[1] \\ \mathbf{v}[2] & 0 & -\mathbf{v}[0] \\ -\mathbf{v}[1] & \mathbf{v}[0] & 0 \end{bmatrix}, \quad (30)$$

where $\mathbf{v} = {}^{AC\_l}\mathbf{r}_{M\_c/M\_l\_AC}$ and

$$^{AC\_l}\mathbf{r}_{M\_c/M\_l\_AC} = \begin{bmatrix} R_{Mo}\Big(-\mathrm{c}(\alpha)\mathrm{s}(\delta_\alpha) + \mathrm{s}(\alpha)\mathrm{c}(\delta_\alpha)\Big) \\ R_{Mo}\Big(\mathrm{s}(\alpha)\mathrm{s}(\delta_\alpha) + \mathrm{c}(\alpha)\mathrm{c}(\delta_\alpha) - \frac{\mathrm{c}(\alpha/2)}{\mathrm{c}(\delta_\alpha - \alpha/2)}\Big) \\ 0 \end{bmatrix}, \quad (31)$$

where $\mathrm{s}(q) = \sin(q)$ and $\mathrm{c}(q) = \cos(q)$ for any parameter q.

Solving compliance matrix numerically: As both the transformation matrix $^{AC\_l}\mathbf{T}_{AC\_c}$ and stiffness matrix $^{AC\_l}\mathbf{K}$ are a function of δ_α, the equation 25 should integrate over $d\delta_\alpha$ instead of dl in order to solve it. For small angles of $d\delta_\alpha$ the relation can be written as:

$$dl = R_{M\_AC} \cdot d\delta_\alpha, \quad (32)$$

where the length of R_{M_AC} (see Figure 9) is also a function of δ_α:

$$R_{M\_AC} = R_{Mo}\frac{\cos(\frac{\alpha}{2})}{\cos(\frac{\alpha}{2} - \delta_\alpha)}. \quad (33)$$

In order to save computation time, the integration from equation 25 is done numerically in the TetraFEM tool. The composite trapezoidal rule [32] is applied to numerically approximate the compliance matrix of flexure AC. If we define function f such that

$$f(\delta_\alpha) = {}^{AC\_l}\mathbf{T}^T_{AC\_c} \cdot {}^{AC\_l}\mathbf{K}^{-1} \cdot {}^{AC\_l}\mathbf{T}_{AC\_c} \cdot R_{M\_AC}$$

the compliance matrix of flexure AC can be numerically computed with the following equation:

$$^{AC\_c}\mathbf{C} \approx \frac{\alpha}{N}\Big(\frac{f(0)}{2} + \sum_{K=1}^{N-1}\Big(f\Big(\frac{K\cdot\alpha}{N}\Big) + \frac{f(\alpha)}{2}\Big)\Big), \quad (34)$$

where N is an integer determining the amount of dl elements to consider in this finite-element based method of calculating the compliance matrix. The flexure compliance analysis for the other flexures (AB and BC) are done with the same approach as the one explained for flexure AC.

2) Tetrahedron Compliance Analysis: After the compliance matrices of all flexures are calculated using the flexure compliance analysis, the compliance matrix of the tetrahedra can be obtained using a tetrahedron compliance analysis, which is explained in this part. This analysis is heavily inspired by the work of (again) F Pavari Rad et al. [21], who used it to obtain the compliance matrix of two flexures in series.

As illustrated in Figure 4, each tetrahedron is connected to the previous tetrahedron on edge a and connected to the next tetrahedron on edge c. As all tetrahedra are connected in series, the spherical joint can be seen as one large spring consisting of n smaller springs in series. Within a tetrahedron, flexure AB and BC are two flexure springs in series and the two combined are in parallel with flexure spring AC. In order to get the compliance matrix of the 3 flexures combined, the compliance matrices of the individual flexures first need to be

related to a common reference frame AC_c and a common point M_c. The compliance matrix of flexure AC is already related to AC_c and M_c after the flexure compliance analysis, but this is not the case for the other two flexures. The compliance matrix of flexure AB is related to point M_b and frame AB_b after the flexure compliance analysis and the compliance matrix of flexure BC is related to M_c and BC_c (see Figure 10).

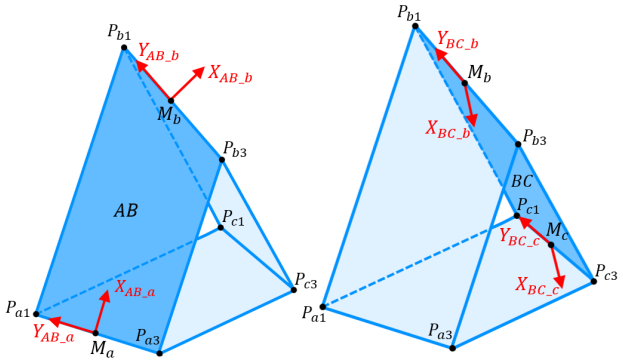

Fig. 10: The global coordinate systems used for the flexure compliance analysis of AB (left) and BC (right)

The compliance matrices of AB and BC can be related to point M_c and AC_c using transformation matrices, as explained in [21]. The transformation matrices for flexure AB and BC respectively are defined as:

$$^{AB\_b}\mathbf{T}_{AC\_c} = \left[\begin{array}{c|c} {}^{AB\_b}\mathbf{R}_{AC\_c} & \mathbf{0} \\ \hline {}^{AB\_b}\tilde{\mathbf{r}}_{M\_c/M\_b} \cdot {}^{AB\_b}\mathbf{R}_{AC\_c} & {}^{AB\_b}\mathbf{R}_{AC\_c} \end{array}\right] \quad (35)$$

$$^{BC\_c}\mathbf{T}_{AC\_c} = \left[\begin{array}{c|c} {}^{BC\_c}\mathbf{R}_{AC\_c} & \mathbf{0} \\ \hline \mathbf{0} & {}^{AC\_c}\mathbf{R}_{AC\_c} \end{array}\right] \quad (36)$$

where $^{AB\_b}\mathbf{R}_{AC\_c}$ and $^{BC\_c}\mathbf{R}_{AC\_c}$ denote the rotation matrix of frame AC_c with respect to AB_b and BC_c respectively. The position vector $^{AB\_b}\mathbf{r}_{M\_c/M\_b}$ in equation 35 locates point M_c from M_b with respect to frame AB_b. With the transformation matrices from 35 and 36, the numerically calculated compliance matrices $^{AB\_b}\mathbf{C}$ and $^{BC\_c}\mathbf{C}$ can be related to frame AC_c and point M_c:

$$^{AC\_c}\mathbf{C}_{AB\_T} = {}^{AB\_b}\mathbf{T}^T_{AC\_c} \cdot {}^{AB\_b}\mathbf{C} \cdot {}^{AB\_b}\mathbf{T}_{AC\_c}, \quad (37)$$

$$^{AC\_c}\mathbf{C}_{BC\_T} = {}^{BC\_c}\mathbf{T}^T_{AC\_c} \cdot {}^{BC\_c}\mathbf{C} \cdot {}^{BC\_c}\mathbf{T}_{AC\_c}, \quad (38)$$

where $^{AC\_c}\mathbf{C}_{AB\_T}$ and $^{AC\_c}\mathbf{C}_{BC\_T}$ are the transformed compliance matrices of flexures AB and BC. Flexures AB and BC are connected in series and therefore the compliance matrix of the flexures combined can simply be calculated as:

$$^{AC\_c}\mathbf{C}_{AB+BC} = {}^{AC\_c}\mathbf{C}_{AB\_T} + {}^{AC\_c}\mathbf{C}_{BC\_T}. \quad (39)$$

Flexure AC is connected in parallel with the combined flexures AB and BC, hence the compliance matrices need to be first inverted and then added to calculate the stiffness matrix of the total tetrahedron.

$$^{AC\_c}\mathbf{K}_{tetra} = {}^{AC\_c}\mathbf{C}^{-1}_{AB+BC} + {}^{AC\_c}\mathbf{C}^{-1}_{AC}, \quad (40)$$

where $^{AC\_c}\mathbf{C}_{AC}$ is the numerically calculated compliance matrix of flexure AC from equation 34. The inverse of stiffness matrix $^{AC\_c}\mathbf{K}_{tetra}$ equals the compliance matrix of the complete tetrahedron:

$$^{AC\_c}\mathbf{C}_{tetra} = {}^{AC\_c}\mathbf{K}^{-1}_{tetra}, \quad (41)$$

3) Chain Algorithm: The chain algorithm is a computationally efficient method to calculate nonlinear deformation in a compliant mechanism. Howell explained in [29] how the chain algorithm can be used to calculate the deflection of a flexible cantilever beam discretized into a number of beam elements. Each element is analysed in succession and subjected to an equivalent load. The elements are connected in series and considered fixed at the end of the previous element as shown in Figure 11. Although the deflection of each element is considered to be linear, the compliant mechanism as a whole may have a large nonlinear deflection as the deflections of the elements accumulate. This large deflection may alter the position, direction or magnitude of the load acting on the compliant mechanism and therefore the load and resulting deflection need to be calculated iteratively.

The chain algorithm is utilized in the TetraFEM tool as well to calculate the (nonlinear) deflection of the spherical joint when it is subjected to the load of a pendulum and the self-weight of the flexures. Conveniently, the joint in this paper is already discretized in n number of tetrahedron elements and its compliance matrices are calculated in the aforementioned tetrahedron compliance analysis.

Let $(M_{\_c\_u})_i$ in Figure 12 be the point M_c of T_i displaced due to the deflections of (only) the previous tetrahedra. The total equivalent load $(\mathbf{w}_{total})_i$ acts on the free end of T_i, which is where $(M_{\_c\_u})_i$ is located, and causes a deflection $(\mathbf{ds})_i$. This deflection depends on the equivalent loads and the compliance matrix of T_i and is assumed to be linear:

$$^{(AC\_c\_u)i}(\mathbf{ds})_i = {}^{AC\_c}\mathbf{C}_{(tetra)i} \cdot {}^{(AC\_c\_u)i}(\mathbf{w}_{total})_i. \quad (42)$$

where $(AC_{c\_u})_i$ denotes the changed frame AC_c of T_i, due to the deflections of only the previous tetrahedra.

Linking deflected frames: If the equivalent loads and compliance matrices are known for each tetrahedron, it is possible to link all the frames within the deflected joint. The deflection of T_i caused by $(\mathbf{w}_{total})_i$ changes the orientation of frame $(AC_{c\_u})_i$ to frame $(AC_{c\_d})_i$ as illustrated in the simplified Figure 12. To obtain frame $(AC_{c\_d})_i$ from the deflection of equation 42, three successive rotations about the unit vectors of $(AC_{c\_u})_i$ are required [33]. Thus the rotation

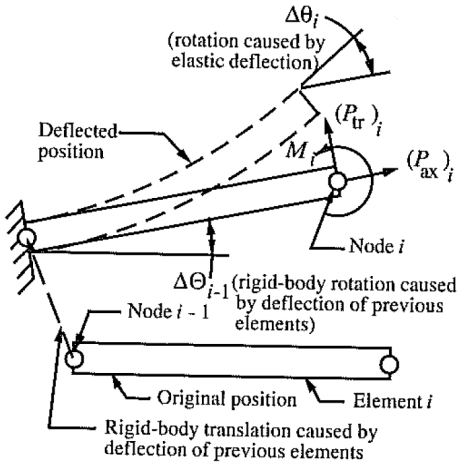

Fig. 11: A typical beam element as used by the chain algorithm. The position and orientation of Node i changes due to the accumulated deflections of previous elements and the deflection of Element i itself. Retrieved and modified from [29].

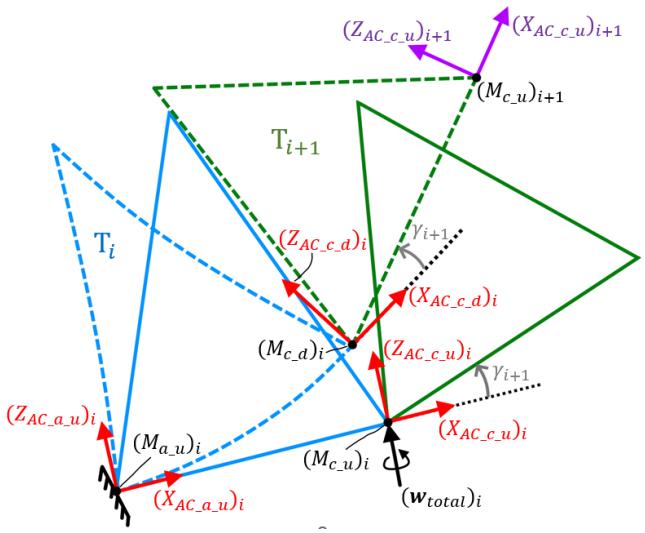

Fig. 12: A simplified 2D illustration of the deformation of T_i and the displacement of the subsequent tetrahedron caused by equivalent load $(\mathbf{w}_{total})_i$.

matrix of frame $(AC_{c\_d})_i$ with respect to $(AC_{c\_u})_i$ can be written as:

$$
{}^{(AC\_c\_d)i}\mathbf{R}_{(AC\_c\_u)i} = \begin{bmatrix} c(y)c(z) & s(x)s(y)c(z) - s(z)c(x) & s(x)s(z) + s(y)c(x)c(z) \\ s(z)c(y) & s(x)s(y)s(z) + c(x)c(z) & -s(x)c(z) + s(y)s(z)c(x) \\ -s(y) & s(x)c(y) & c(x)c(y) \end{bmatrix}, \quad (43)
$$

where $\mathrm{s}(q) = \sin(q)$ and $\mathrm{c}(q) = \cos(q)$ for any parameter q. The relation between terms x, y and z in equation 43 and ${}^{(AC\_c\_u)i}(\mathbf{ds})_i$ is defined as:

$$
\begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}^{(AC\_c\_u)i} \begin{bmatrix} ds_{\theta x} \\ ds_{\theta y} \\ ds_{\theta z} \end{bmatrix}_i .
$$

Frame $(AC_{a\_u})_{i+1}$ denotes the changed frame AC_a of T_{i+1}, due to deflections of previous tetrahedra (including T_i). The rotation matrix of $(AC_{a\_u})_{i+1}$ with respect to $(AC_{c\_d})_i$ is a function of the independent geometric parameter γ of T_{i+1}:

$$
{}^{(AC\_a\_u)i+1}\mathbf{R}_{(AC\_c\_d)i} = \begin{bmatrix} \cos(\gamma_{i+1}) & 0 & \sin(\gamma_{i+1}) \\ 0 & 1 & 0 \\ -\sin(\gamma_{i+1}) & 0 & \cos(\gamma_{i+1}) \end{bmatrix} \quad (44)
$$

The rotation matrix of frame $(AC_{c\_u})_{i+1}$ with respect to $(AC_{a\_u})_{i+1}$ is a function of the independent geometric parameter α of T_{i+1}:

$$
{}^{(AC\_c\_u)i+1}\mathbf{R}_{(AC\_a\_u)i+1} = \begin{bmatrix} \cos(\alpha_{i+1}) & -\sin(\alpha_{i+1}) & 0 \\ \sin(\alpha_{i+1}) & \cos(\alpha_{i+1}) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (45)
$$

Using a successive multiplication [31] of the rotation matrices from equations 43, 44 and 45, it is possible to determine the orientation of all relevant reference frames of the deflected joint within the TetraFEM simulation.

Linking deflected points: Similar to the reference frames, the locations of the points within the deflected joint can all be linked as well within the TetraFEM simulation. The deflection from equation 42 can be used to describe the position vector of point $(M_{c\_d})_i$ relative to $(M_{c\_u})_i$:

$$
{}^{(AC\_c\_u)i}\mathbf{r}_{(M\_c\_d)i/(M\_c\_u)i} = {}^{(AC\_c\_u)i} \begin{bmatrix} ds_x \\ ds_y \\ ds_z \end{bmatrix}_i , \quad (46)
$$

where point $(M_{c\_d})_i$ denotes point M_c of T_i, displaced due to the deflections of both the previous tetrahedra and T_i (see Figure 12). The tetrahedra are connected in series with edge c of T_i connected to edge a of T_{i+1}. The points $(M_{c\_d})_i$ and $(M_{a\_u})_{i+1}$ are therefore located in the exact same position and the position vector relating these two point can be written as:

$$
{}^{(AC\_c\_u)i}\mathbf{r}_{(M\_a\_u)i+1/(M\_c\_d)i} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \quad (47)
$$

The position vector of point $(M_{c\_u})_{i+1}$ (see Figure 11) relative to point $(M_{a\_u})_{i+1}$ can be calculated using the geometry of flexure AC from T_{i+1}:

$$
{}^{(AC\_a\_u)i+1}\mathbf{r}_{(M\_c\_u)i+1/(M\_a\_u)i+1} = \begin{bmatrix} R_{Mo} \cdot \sin(\alpha_{i+1}) \\ R_{Mo} \cdot (\cos(\alpha_{i+1}) - 1) \\ 0 \end{bmatrix}, \quad (48)
$$

where R_{Mo} is implemented from equation 7 and is equal for each tetrahedron. By summing up the relative position vectors from equation 46, 47 and 48 it is possible to determine

the position vector of $(M_{c\_u})_{i+1}$ relative to $(M_{c\_u})_i$. Note that the relative position vector of equation 48 is defined in frame $(AC_{c\_u})_{i+1}$ and therefore first needs to be rotated to the common frame $(AC_{c\_u})_i$:

$$
\begin{aligned}
{}^{(AC\_c\_u)i}\mathbf{r}_{(M\_c\_u)i+1/(M\_c\_u)i} = \\
{}^{(AC\_c\_u)i}\mathbf{r}_{(M\_c\_d)i/(M\_c\_u)i} + \\
{}^{(AC\_c\_u)i}\mathbf{R}_{(AC\_a\_u)i+1} \cdot {}^{(AC\_a\_u)i+1}\mathbf{r}_{(M\_c\_u)i+1/(M\_a\_u)i+1}
\end{aligned}
\tag{49}
$$

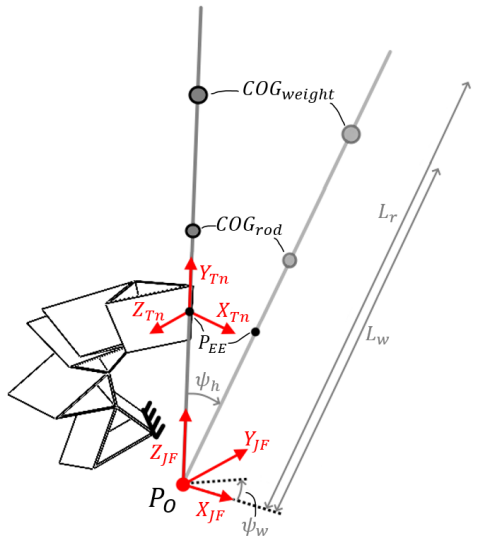

Fig. 13: A generic spherical joint composed of five tetrahedra with a pendulum shown in deformed (right) and undeformed (left) orientation. Coordinate system $X_{Tn}Y_{Tn}Z_{Tn}$ is associated with frame AC_c of T_n.

End effector point: Point $(M_{c\_d})_n$ is point M_c of the last tetrahedron, displaced due to the deflection of all n tetrahedra. The location of this point is an indicator for the total deflection of the spherical joint and will be further denoted as end effector point P_{EE} (see Figure 13). The position vector of P_{EE} relative to P_O can be calculated using the following equation:

$$
\begin{aligned}
{}^{JF}\mathbf{r}_{P_{EE}/P_O} = {}^{JF}\mathbf{R}_{(AC\_c)1} \cdot \begin{bmatrix} 0 \\ R_{Mo} \\ 0 \end{bmatrix} \\
+ \sum_{i=1}^{n-1} {}^{JF}\mathbf{R}_{(AC\_c)i} \cdot {}^{(AC\_c)i}\mathbf{r}_{(M\_c\_u)i+1/(M\_c\_u)i}
\end{aligned}
\tag{50}
$$

where joint frame JF is equal to the AC_c frame of T_n in undeformed state, rotated -90 degrees around its X-axis, as illustrated in the figure. If the joint is undeformed, hence $(\mathbf{ds})_i$ is a null vector for all i, the rod of the pendulum coincides with the Z-axis of JF and the rotation matrix of JF with respect to $(AC_c)_n$ can be written as:

$$
{}^{JF}\mathbf{R}_{(AC\_c)n} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{51}
$$

Pendulum load: The load of the pendulum to be balanced depends on its size and weight, which are inputs for the TetraFEM tool. As illustrated in Figure 13, the load acting on the joint also depends on the orientation of the pendulum. L_w is the distance between the center of rotation P_O and the COG of the weight on the pendulum with mass m_w. The mass m_r of the pendulum's rod has its COG at a distance of $L_r/2$ from P_O. If the pendulum is in equilibrium under an angle of ψ_h, the moment force M_{pend} at point P_O can be written as:

$$
M_{pend} = \sin(\psi_h)\Big(m_w g L_w + m_r g \frac{L_r}{2}\Big). \tag{52}
$$

where $g = 9.81 m/s^2$. The load of the pendulum $\mathbf{w}_{pend}$ at point P_O and expressed in reference frame JF equals:

$$
{}^{JF}\mathbf{w}_{pend} = \begin{bmatrix} 0 \\ 0 \\ -g(m_w + m_r) \\ -\sin(\psi_w) M_{pend} \\ \cos(\psi_w) M_{pend} \\ 0 \end{bmatrix}. \tag{53}
$$

The load of the pendulum needs to be related to frame $(AC_{c\_u})_i$ to obtain the equivalent force acting on T_i. The calculation of the transformation matrix for this process is very similar to that of equation 26 and 35 and can be written as:

$$
\begin{aligned}
&{}^{(AC\_c\_u)i}\mathbf{T}_{JF} = \\
&\left[\begin{array}{c|c} {}^{(AC\_c\_u)i}\mathbf{R}_{JF} & \mathbf{0} \\ \hline {}^{(AC\_c\_u)i}\tilde{\mathbf{r}}_{P_O/(M\_c\_u)i} \cdot {}^{(AC\_c\_u)i}\mathbf{R}_{JF} & {}^{(AC\_c\_u)i}\mathbf{R}_{JF} \end{array}\right]
\end{aligned}
\tag{54}
$$

where position vector ${}^{(AC\_c\_u)i}\mathbf{r}_{P_O/(M\_c\_u)i}$ locates P_O from $(M_{c\_u})_i$ with respect to frame $(AC_{c\_u})_i$. With the transformation matrix from 54, the equivalent load of the pendulum acting on T_i can be calculated:

$$
{}^{(AC\_c\_u)i}\mathbf{w}_{pend} = {}^{(AC\_c\_u)i}\mathbf{T}_{JF} \cdot {}^{JF}\mathbf{w}_{pend} \tag{55}
$$

Flexure weight load: The weights of the flexures cannot always be neglected and may need to be considered to accurately determine the total equivalent load $(\mathbf{w}_{total})_i$. Taking flexure AC as example, its gravitational load at COG_{AC} expressed in reference frame JF equals:

$$
{}^{JF}\mathbf{w}_{AC} = \begin{bmatrix} 0 \\ 0 \\ -g m_{AC} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{56}
$$

The weight of a flexure affects the deflection of T_i if the flexure is part of a tetrahedron T_k, where $i \leq k \leq n$. The transformation matrix relating the gravitational load of a flexure $(\mathrm{AC})_k$ to its equivalent load on T_i can be written as:

$$
\begin{aligned}
&{}^{(AC\_c\_u)i}\mathbf{T}_{JF} = \\
&\left[\begin{array}{c|c} {}^{(AC\_c\_u)i}\mathbf{R}_{JF} & \mathbf{0} \\ \hline {}^{(AC\_c\_u)i}\tilde{\mathbf{r}}_{Gk/(M\_c\_u)i} \cdot {}^{(AC\_c\_u)i}\mathbf{R}_{JF} & {}^{(AC\_c\_u)i}\mathbf{R}_{JF} \end{array}\right]
\end{aligned}
\tag{57}
$$

where Gk is an abbreviation for point $(COG_{AC})k$. It should be noted that the transformation matrix in equation 57 is not equal to that of equation 54. The position vector corresponding to the transformation matrix in 57 locates the COG of flexure $(AC)_k$ (instead of P_O) from point $(M_{c\_u})_i$ with respect to frame $(AC_{c\_u})_i$. This position vector can be calculated using the following equation by relating the reference frames and position vectors within the deflected joint:

$$\begin{aligned} {}^{(AC\_c\_u)i}\mathbf{r}_{(COG\_AC)k/(M\_c\_u)i} = \\ {}^{(AC\_c\_u)i}\mathbf{R}_{(AC\_c\_u)k} \cdot \begin{bmatrix} \sin(\alpha_k/2)R_{COG\_AC} \\ R_{Mo} - \cos(\alpha_k/2)R_{COG\_AC} \\ 0 \end{bmatrix} \\ +{}^{(AC\_c\_u)i}\mathbf{r}_{(M\_c\_u)k/(M\_c\_u)i} \end{aligned} \quad (58)$$

where R_{COG_AC} from equation 15 is used to relate point COG_{AC} to $(M_{c\_u})_k$. The equivalent gravitational load of flexure $(AC)_k$ acting on T_i can now be calculated by converting the gravitational load to frame $(AC_{c\_u})_i$ and point $(M_{c\_u})i$ with the transformation matrix of equation 57:

$${}^{(AC\_c\_u)i}\mathbf{w}_{(AC)k} = {}^{(AC\_c\_u)i}\mathbf{T}_{JF} \cdot {}^{JF}\mathbf{w}_{(AC)k} \quad (59)$$

Flexures AB and BC have corresponding masses m_{AB} and m_{BC} and equivalent loads ${}^{(AC\_c\_u)i}\mathbf{w}_{AB}$ and ${}^{(AC\_c\_u)i}\mathbf{w}_{BC}$ respectively. These parameters are computed in a very similar way as for flexure AC. The combined gravitational load ${}^{(AC\_c\_u)i}\mathbf{w}_k$ of the three flexures of T_k is defined as:

$$\begin{aligned} {}^{(AC\_c\_u)i}\mathbf{w}_k = {}^{(AC\_c\_u)i}\mathbf{w}_{(AC)k} \\ +{}^{(AC\_c\_u)i}\mathbf{w}_{(AB)k} + {}^{(AC\_c\_u)i}\mathbf{w}_{(BC)k} \end{aligned} \quad (60)$$

Total equivalent load: The total equivalent force causing the deflection in T_i (see equation 42) can now be calculated by combining the load of the pendulum and the weights of T_k:

$$\begin{aligned} {}^{(AC\_c\_u)i}(\mathbf{w}_{total})_i = \\ {}^{(AC\_c\_u)i}\mathbf{w}_{pend} + \sum_{k=i}^{n} {}^{(AC\_c\_u)i}\mathbf{w}_k \end{aligned} \quad (61)$$

4) Pendulum Balancing Test: This part will explain how the TetraFEM tool can quantify the pendulum balancing performance of its simulated spherical joint. The intended ROM of the simulated joint is defined by the range of parameters ψ_w and ψ_h (see Figure 13) and has the shape of a spherical segment. If the simulated joint would be a perfect spherical joint, meaning it only allows rotational movement around a constant center of rotation P_O, then point P_{EE} would be able to move over this spherical segment with a constant radius R_{Mo} (see equation 7). This ideal spherical segment is discretized in the TetraFEM tool by m evenly distributed points denoted as $(P_{goal})_j$. The position vector of $(P_{goal})_j$ relative to P_O is defined as:

$${}^{JF}\mathbf{r}_{(P\_goal)j/P_O} = \begin{bmatrix} \sin\big((\psi_h)_j\big)\cos\big((\psi_w)_j\big)R_{Mo} \\ \sin\big((\psi_h)_j\big)\sin\big((\psi_w)_j\big)R_{Mo} \\ R_{Mo}\Big(1 - \sin\big((\psi_h)_j\big)\Big) \end{bmatrix}, \quad (62)$$

where $(\psi_w)_j$ and $(\psi_h)_j$ are defined such that the $(P_{goal})_j$ points are evenly distributed over the spherical segment. A pendulum load related to $(P_{goal})_j$ can be calculated by implementing $(\psi_w)_j$ and $(\psi_h)_j$ in equations 52 and 53. The simulated spherical joint in the TetraFEM tool is subjected to this pendulum load $(\mathbf{w}_{pend})_j$ and the self-weight of the flexures to test its pendulum balancing performance.

It should be noted that this applied pendulum load $(\mathbf{w}_{pend})_j$ does not depend on the deflection of the joint and therefore does not need to be calculated iteratively, as explained in the chain algorithm II-C3. As a result, the computation time for deflections in the TetraFEM tool would be significantly reduced if the self-weight loads of the flexures, which do depend on the deflection of the joint, can be neglected.

The spherical joint is a perfect pendulum balancer if load $(\mathbf{w}_{total})_j$ makes point $(P_{EE})_j$ deflect exactly to the location of the corresponding $(P_{goal})_j$. The pendulum balancing performance can be quantified by averaging the magnitudes of the distances between $(P_{goal})_j$ and $(P_{EE})_j$ for all m points:

$$d_{avg} = \frac{1}{m}\sum_{j=1}^{m} \left|\left| {}^{JF}\mathbf{r}_{(P\_goal)j/P_O} - {}^{JF}\mathbf{r}_{(P_{EE})j/P_O} \right|\right|, \quad (63)$$

where d_{avg} represents a deviation that would be 0 meters for a perfect pendulum balancing joint. This deviation can be compared to s_{avg}, the average distance between $(P_{goal})_j$ and the undeformed location of P_{EE}, allowing the deformation resemblance η_{dr} between the simulated joint and the ideal pendulum balancing joint to be calculated:

$$\eta_{dr} = \left(1 - \frac{d_{avg}}{s_{avg}}\right) \cdot 100\%, \quad (64)$$

where

$$s_{avg} = \frac{1}{m}\sum_{j=1}^{m} \left\| {}^{JF}\mathbf{r}_{(P\_goal)j/EE\_undef} \right\| \quad (65)$$

and where EE_{undef} is point P_{EE} in undeformed state.

D. Optimization

This study conducted five separate optimizations on the DelftBlue supercomputer [34] to determine the independent geometric parameters of the best pendulum balancers for five different scenarios. The amount of dl elements per flexure, the R_{in} and R_w parameters, the material properties and the properties of the pendulum are equal and fixed for all five scenarios and are shown in Table I.

Fixed parameters	Value	Description
N	200	Amount of dl elements in a flexure
R_{in}	67 mm	Independent geometric parameter
R_w	25 mm	Independent geometric parameter
ρ	1010 kg/m^3	Density of flexure material
E	1700·10^6 Pa	Young's modulus
ν	0.38	Poisson's ratio
b_y	$\frac{5}{6}$	Shear coefficient
b_z	$\frac{5}{6}$	Shear coefficient
m_w	100 gram	Mass of weight on pendulum
m_r	23.5 gram	Mass of the rod
L_w	250 mm	Distance between weight and P_O
L_r	320 mm	Length of the rod

TABLE I: The fixed parameters used in the optimizations.

For each optimization the COBYLA [35] method is used to minimize the objective function d_{avg}, as defined in equation 63. The boundary values imposed on the optimizations are shown in Table II, where N_{iter} denotes the number of iterations.

Parameter	Boundary values
α_i [deg]	[5, 60]
γ_i [deg]	[-90, 35]
t_{min} [μm]	[1000, 1800]
N_{iter}	≤ 1000

TABLE II: The boundary values imposed on the optimizations.

Each separate optimization is repeated for 50 sets of different initial values for α_i, γ_i and t_{min}. These initial values are determined randomly, but within the boundary values from Table II.

E. Experimental validation

1) Experimental setup: A prototype is 3D-printed from PA-12 powder with Multi Jet Fusion printers to validate the TetraFEM tool and the results of the optimizations. This material should have the same properties as those depicted in Table I. The prototype is placed in an experimental setup as shown in Figure 14 and 15. The base of the prototype is fixed to a part with a fixed reference point and the other end of the prototype is attached to a pendulum with the same properties as in Table I. The weights on the pendulum rod can be shifted along the rod to increase or decrease parameter L_w and consequently change the load of the pendulum on the joint. The fixed reference point is placed such that its location is exactly 5 mm under the center of rotation, indicated by the P_O-indicator, if no loads are applied to the prototype.

Two phone holders, each supporting a mobile phone with a camera, are positioned at a 90-degree angle relative to each other and at an equal distance from the center of rotation. In the background of the prototype, rings are suspended freely on threads. These threads serve as a reference for the direction of gravity when capturing images of the prototype.

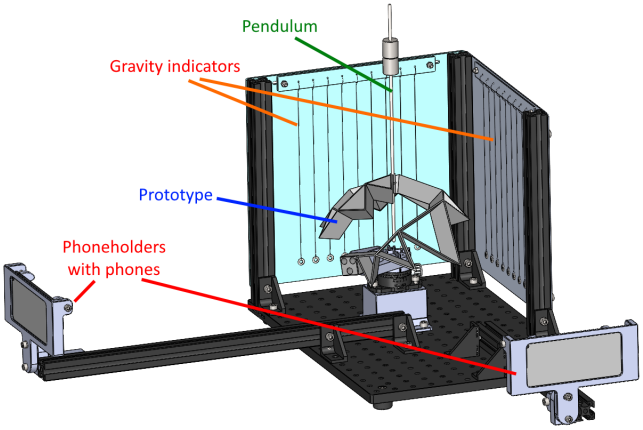

Fig. 14: A broad view of the experimental setup.

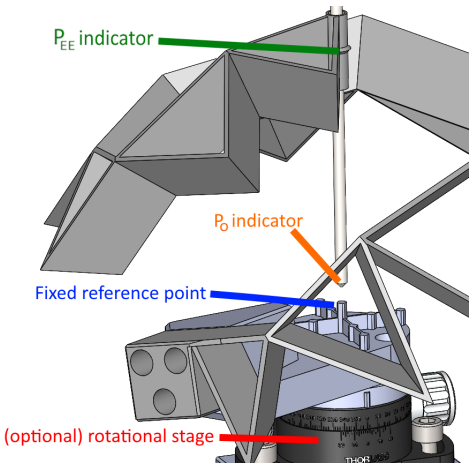

Fig. 15: A close-up view of the experimental setup.

2) Measurement procedure: With the pendulum attached to the prototype, the orientation and position of the pendulum rod for which the combined mechanism was in equilibrium were captured using two cameras. As the P_{EE}-indicator is located at a fixed distance R_{Mo} from the P_O-indicator, its position in equilibrium can be calculated from these images. This process was done for each stable and unstable equilibrium found for $\psi_h \leq 45$ degrees and was repeated for several values of L_w, the position of the weights on the pendulum rod, with increments of 2.5 mm until no additional equilibria were observed.

III. Results

A. Optimization results

1) Ten-tetrahedra optimization: The first optimization is performed on a spherical joint consisting of ten tetrahedra. The pendulum balancing area for which this joint is optimized is defined by the range of the polar and azimuth angles: $0 \leq \psi_h \leq 30$ and $0 \leq \psi_w < 360$. This spherical segment is discretized in the TetraFEM tool by $m = 11$ evenly distributed points, meaning the simulated pendulum balancer is tested for

TABLE III: Best optimization results

	10-tetrahedra			10-tetrahedra-NSW			8-tetrahedra			12-tetrahedra			10-tetrahedra-90x90		
i	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)
1	16.30	-	1290.61	33.84	-	1063.24	21.67	-	1173.48	48.65	-	1380.13	60.00	-	1512.45
2	27.74	-39.13	1290.61	24.30	-7.19	1063.24	35.42	-9.36	1173.48	20.23	13.39	1380.13	59.64	35	1512.45
3	56.64	7.83	1290.61	28.95	-10.39	1063.24	40.72	0.39	1173.48	17.76	-46.17	1380.13	60.00	34.32	1512.45
4	22.24	-22.22	1290.61	19.82	-46.91	1063.24	43.16	-54.16	1173.48	44.00	-10.09	1380.13	58.53	-20.51	1512.45
5	43.66	-44.36	1290.61	45.29	-29.06	1063.24	51.03	-55.14	1173.48	5.00	15.28	1380.13	37.27	-88.57	1512.45
6	43.02	-22.58	1290.61	35.44	-68.09	1063.24	24.32	-5.64	1173.48	7.57	-25.44	1380.13	31.56	35	1512.45
7	31.80	-38.04	1290.61	7.91	4.14	1063.24	27.49	-89.67	1173.48	30.88	3.39	1380.13	58.09	35	1512.45
8	15.71	-40.34	1290.61	12.62	-22.85	1063.24	41.88	-64.76	1173.48	59.91	-62.02	1380.13	46.87	27.55	1512.45
9	31.40	-83.87	1290.61	9.46	-37.30	1063.24				17.71	-1.54	1380.13	25.18	35	1512.45
10	24.23	-2.07	1290.61	37.62	-79.12	1063.24				18.20	-55.28	1380.13	47.26	-8.37	1512.45
11										20.43	-9.00	1380.13			
12										55.09	-85.95	1380.13			
Time	256 sec			18 sec			118 sec			464 sec			736 sec		
d_avg	1.31 mm			1.02 mm			1.22 mm			1.36 mm			10.77 mm		
η_dr	95.6%			96.6%			96.0%			95.5%			92.0%		

eleven different loads in each iteration of the optimization. The self-weight in this optimization is assumed to be significant and is recalculated and re-applied once in the chain algorithm due to the changed self-weight load in large deformations.

The optimization was run in parallel 50 times under these conditions, with a computation time of approximately 256 seconds per iteration. The optimization result with the lowest objective function d_{avg}, that does not self-intersect in undeformed state, is considered to be the best result. This optimized pendulum balancer was found after 201 iterations, yielding a d_{avg} of 1.31 mm and an η_{dr} of 95.6%. This result is illustrated in Figure 16 and its independent geometric parameters are listed in Table III under "10-tetrahedra".

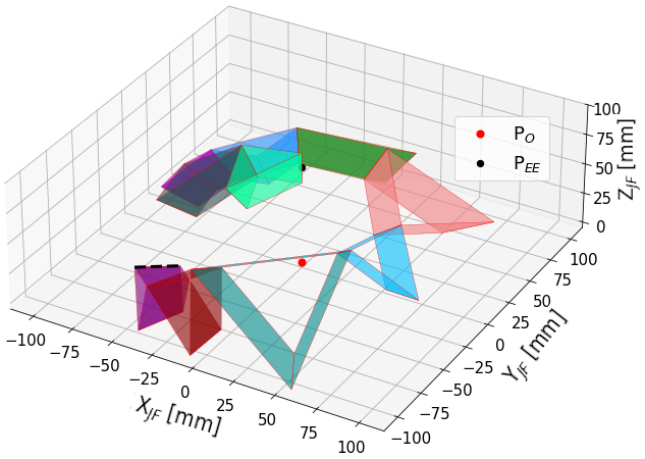

Fig. 16: A 3D view of the best performing pendulum balancer from the ten-tetrahedra optimization, depicted in its undeformed state.

As can be seen in Figure 16, the tetraFEM tool plots the flexures as semi-transparent planes instead of solids for simplicity. To make the tetrahedra easier to distinguish, each is given a different color. Furthermore, edge a of T_1 is depicted with a dashed line to indicate where the simulated joint is fixed to the environment and P_{EE} is the end effector point on edge c of T_n.

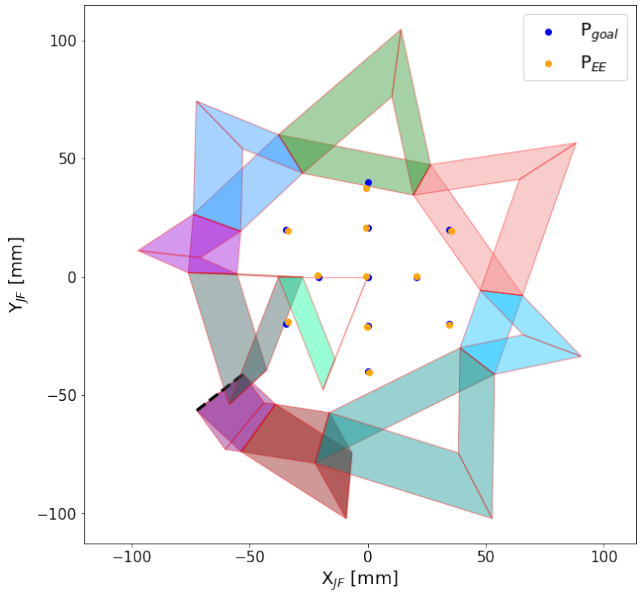

Fig. 17: A top-view of the best performing pendulum balancer from the ten-tetrahedra optimization with eleven P_{goal} and P_{EE} points.

Figure 17 depicts a top-view of the best simulated ten-tetrahedra pendulum balancer, featuring eleven blue dots, each partially covered by an orange dot. The blue dots represent the m evenly distributed P_{goal} points, while the orange dots represent the corresponding P_{EE} points. As explained in the Pendulum Balancing Test (II-C4), the P_{goal} points indicate the ψ_w and ψ_h angles of the pendulum load acting on the simulated joint, with the highest M_{pend} value at the outer ring, where ψ_h = 30 degrees. This pendulum load, along with the self-weight load, causes the P_{EE} point to deflect to the location indicated by an orange dot. The top-view plot provides a convenient visualisation of the simulated pendulum balancer's performance, as this directly correlates to the distance between

the P_{goal} and P_{EE} points. However, it should be noted that the distance in Z_{JF}-direction between P_{EE} and P_{goal}, which is not visible in Figure 17, may not always be negligible (see Figure 18).

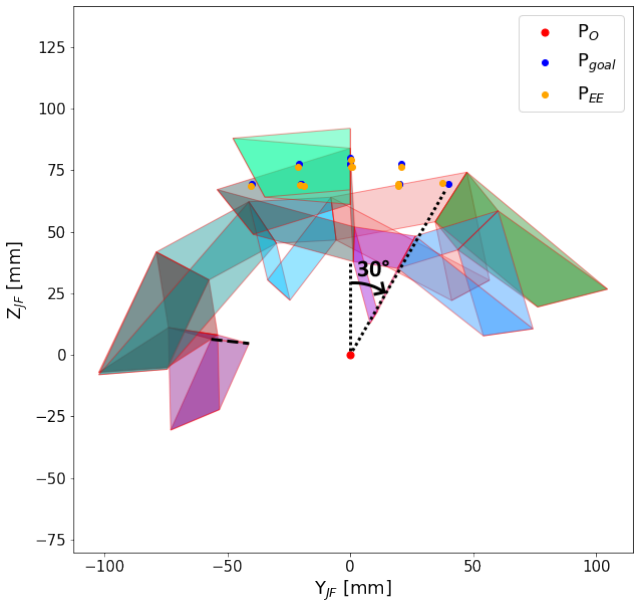

Fig. 18: A (Y-Z) side view of the best performing pendulum balancer from the ten-tetrahedra optimization.

The performance of the optimized pendulum balancer was also tested for m = 28 in the TetraFEM tool, and the resulting top-view plot is shown in Figure 19. The η_{dr} in this test was 95.7%.

2) Ten-tetrahedra-NSW optimization: The second optimization is performed under the same conditions as the ten-tetrahedra optimization, but with the self-weight assumed to be negligible and excluded from the calculations. The optimization was run in parallel 50 times under these conditions, with a computation time of approximately 18 seconds per iteration. The best performing result from the ten-tetrahedra-NSW (No Self Weight) optimization was found after 315 iterations, yielding a d_{avg} of 1.02 mm and an η_{dr} of 96.6%. The top-view of the best ten-tetrahedra-NSW optimization result is shown in Figure 20 and its independent geometric parameters are listed in Table III under "10-tetrahedra-NSW".

The performance of the optimized pendulum balancer is also tested with self-weight considered as a significant load, which is recalculated and re-applied in the TetraFEM tool, similar to the ten-tetrahedra optimization in III-A1. The resulting deformation resemblance η_{dr} in this test is 94.0% and the top-view plot is depicted in Figure 21.

3) Eight-tetrahedra optimization: The third optimization is performed under the same conditions as the ten-tetrahedra optimization, but with eight tetrahedra instead of ten. The optimization was run in parallel 50 times under these conditions, with a computation time of approximately 118 seconds per iteration. The best performing result from the eight-tetrahedra

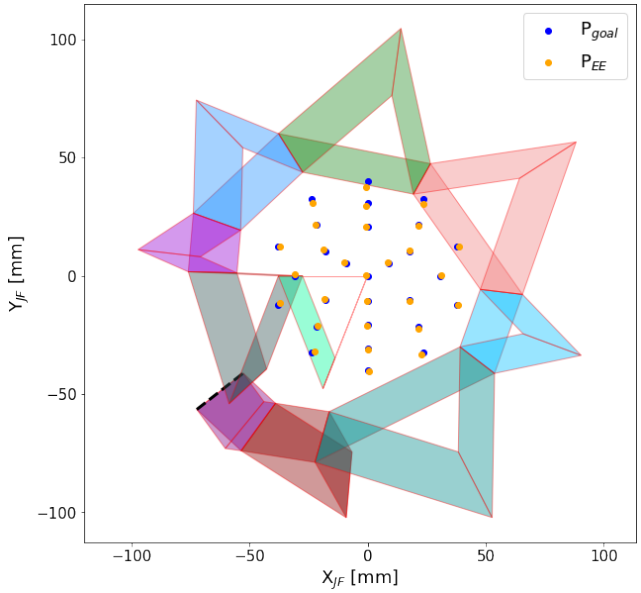

Fig. 19: A top-view of the best performing pendulum balancer from the ten-tetrahedra optimization with 28 P_{goal} and P_{EE} points.

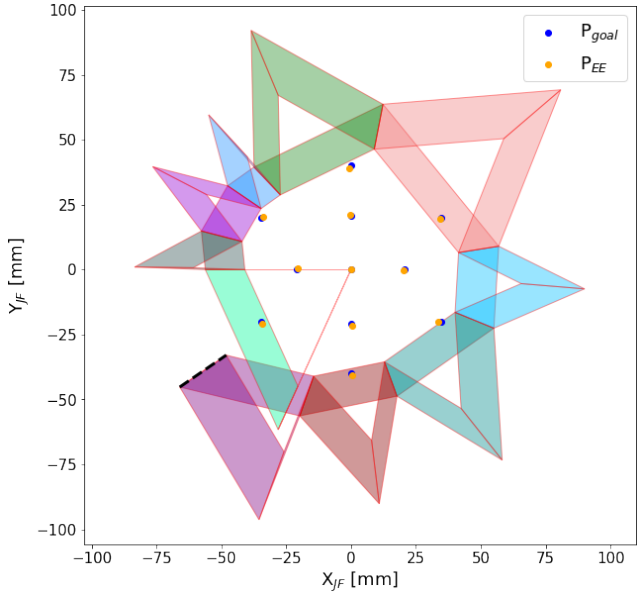

Fig. 20: A top-view of the best performing pendulum balancer from the ten-tetrahedra-NSW optimization with eleven P_{goal} and P_{EE} points.

optimization was found after 261 iteration, yielding a d_{avg} of 1.22 mm and an η_{dr} of 96.0%. The top-view of this optimization result is shown in Figure 22 and its independent geometric parameters are listed in Table III under "8-tetrahedra".

4) Twelve-tetrahedra optimization: The fourth optimization is performed under the same conditions as the ten-tetrahedra optimization, but with twelve tetrahedra instead of ten. The

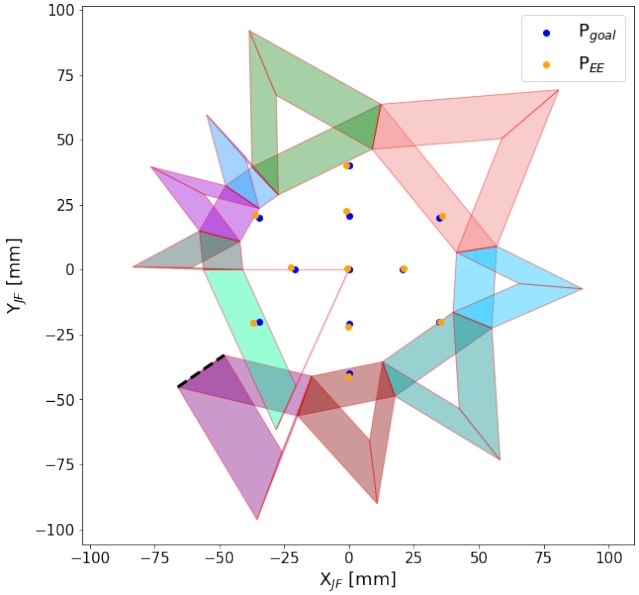

Fig. 21: A top-view of the best performing pendulum balancer from the ten-tetrahedra-NSW optimization with eleven P_{goal} and P_{EE} points.

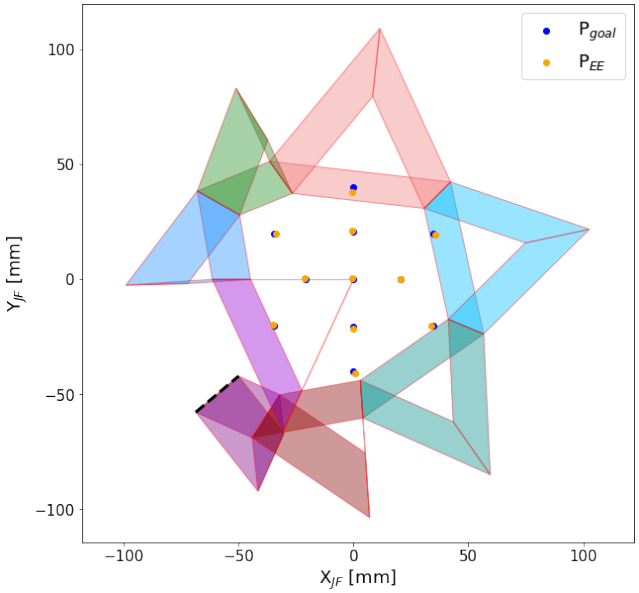

Fig. 22: A top-view of the best performing pendulum balancer from the eight-tetrahedra optimization with eleven P_{goal} and P_{EE} points.

optimization was run in parallel 50 times under these conditions, with a computation time of approximately 464 seconds per iteration. The best performing result from the twelve-tetrahedra optimization was found after 232 iterations, yielding a d_{avg} of 1.36 mm and an η_{dr} of 95.5%. The top-view of this optimization result is shown in Figure 23 and its independent geometric parameters are listed in Table III under "12-tetrahedra".

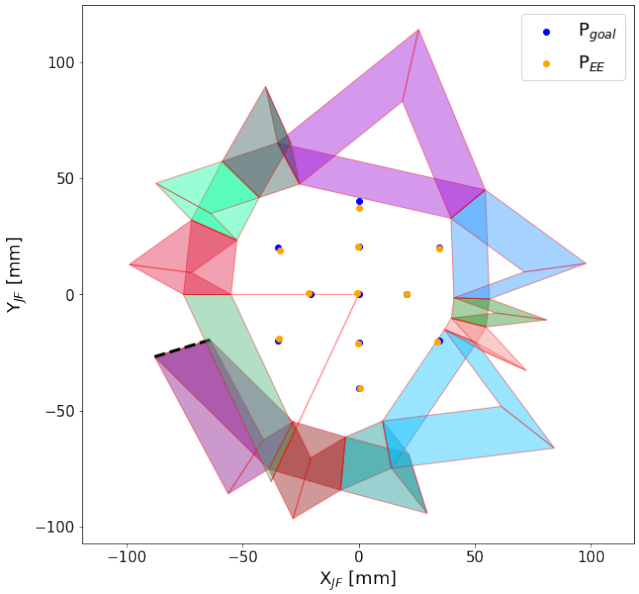

Fig. 23: A top-view of the best performing pendulum balancer from the twelve-tetrahedra optimization with eleven P_{goal} and P_{EE} points.

5) Ten-tetrahedra-90x90 optimization.: The fifth and final optimization is performed under the same conditions as the ten-tetrahedra optimization, but with m = 18 evenly distributed points and a range of $0 \leq \psi_h \leq 90$ and $0 \leq \psi_w \leq 90$ for the polar and azimuth angles. The optimization was run in parallel 50 times under these conditions, with a computation time of approximately 736 seconds per iteration. The best performing result from this ten-tetrahedra-90x90 optimization was found after 284 iterations, yielding a d_{avg} of 10.77 mm and an η_{dr} of 92.0%. The top-view of this optimization result is shown in Figure 24 and its independent geometric parameters are listed in Table III under "10-tetrahedra-90x90". While the distances between the P_{EE} and P_{goal} points in the Z_{JF} direction were relatively minor in the other optimization results, they are considerably more pronounced in this particular optimization result, as depicted in Figure 25.

B. Tetra I as pendulum balancer

The original Tetra I design as defined in [36] is simulated in the TetraFEM tool as well. Its performance as a pendulum balancer is tested for the same range, amount of points and self-weight calculations as described in the ten-tetrahedra optimization (III-A1). The resulting η_{dr} is 32.5% and the top-view is depicted in Figure 26.

C. Prototype validation

The tetrahedra of the 3D-printed prototype described in subsection II-E share the same geometric parameters as the best performing pendulum balancer of the ten-tetrahedra optimization. A top-view of the prototype is depicted in Figure 27.

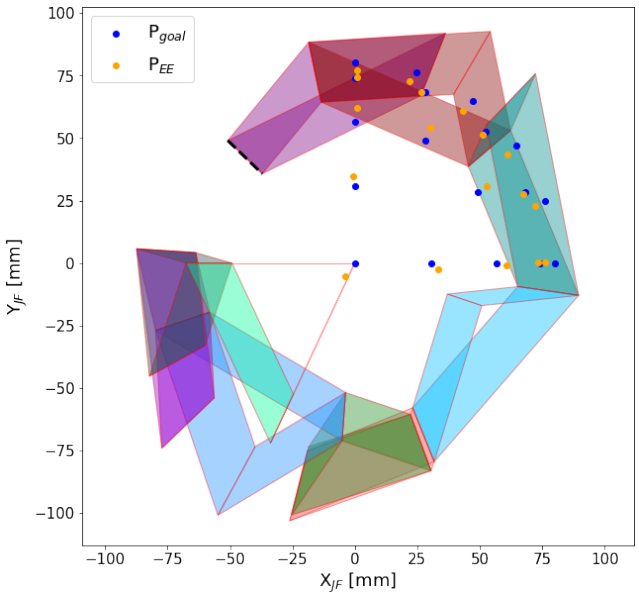

Fig. 24: A top-view of the best performing pendulum balancer from the ten-tetrahedra-90x90 optimization with 18 P_{goal} and P_{EE} points.

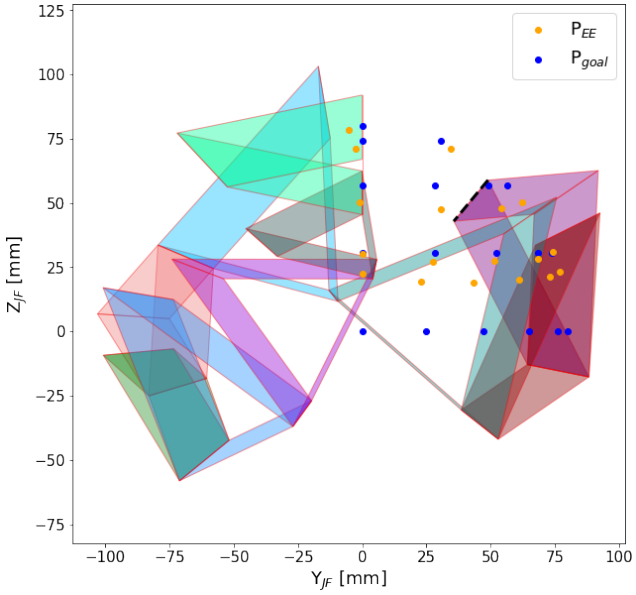

Fig. 25: A YZ side view of the best performing pendulum balancer from the ten-tetrahedra-90x90 optimization with 18 P_{goal} and P_{EE} points.

This view differs slightly from the top-view of the simulated pendulum balancer in Figure 17 due to deflection under its own weight and some creep experienced during shipment.

Figure 28 illustrates the results of the prototype's experimental validation test, using the top-view of the simulated joint as a reference. A total of 30 stable and 31 unstable equilibria were identified for $200 \leq L_w \leq 267.5$ mm, with 2.5 mm

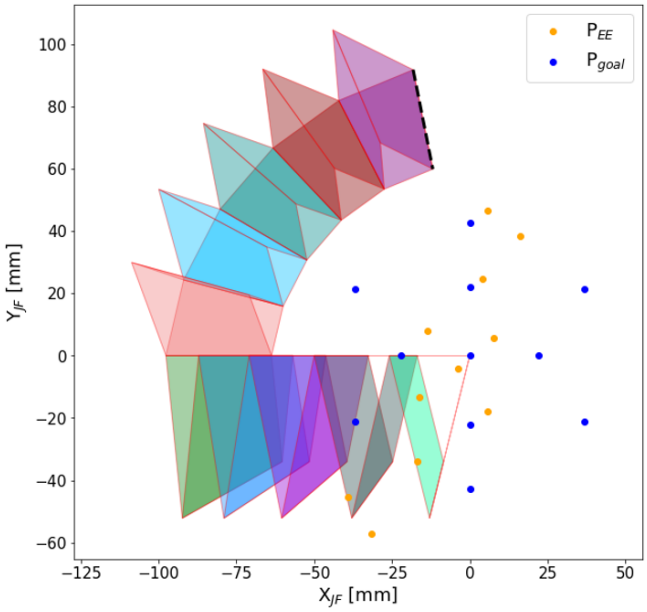

Fig. 26: A top-view of the Tetra I design, tested as a pendulum balancer for eleven P_{goal} and P_{EE} points.

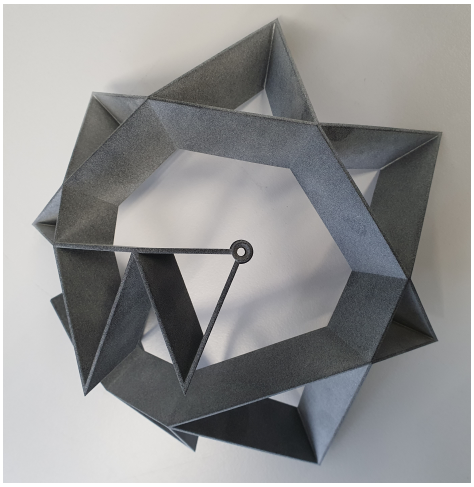

Fig. 27: top-view of the 3D-printed prototype.

increments. The location of the prototype's P_{EE}-indicator is shown for each equilibrium e, represented as a square for stable equilibria and as a diamond for unstable equilibria. The colors of these squares and diamonds represent the value of $(L_w)_e$ when the equilibrium was found. The disk in the figure depicts the top-view of the spherical segment checked for equilibria, which is defined by polar angle $\psi_h \leq 45$ degrees and radius R_{Mo}. The purple area within this spherical segment indicates where the prototype would self-intersect, and therefore these regions could not be reached during the experimental validation test.

The prototype is optimized for ψ_h up to 30 degrees, which is indicated by the yellow circle in Figure 28. For an ideal pendulum balancing spherical joint, the whole area

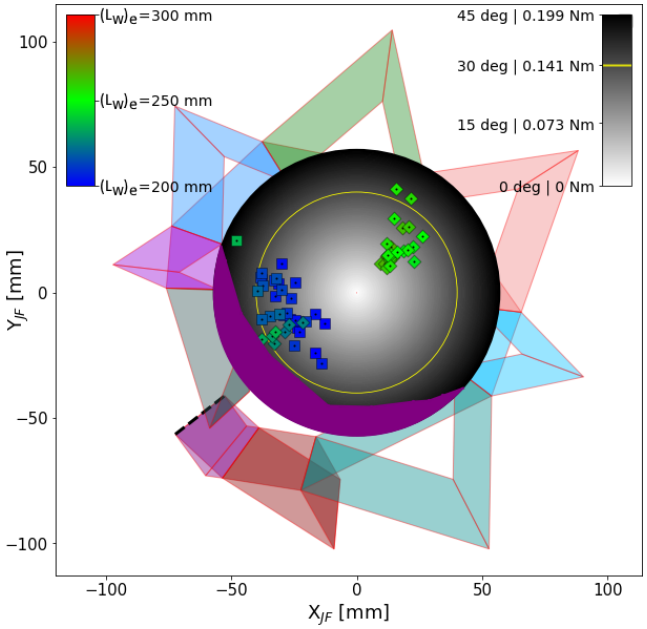

Fig. 28: A top-view of the P_{EE}-indicator points for 61 identified equilibria.

encapsulated by this yellow circle would be filled with green squares and diamonds. However, many of the squares and diamonds depicted in the figure have a non-green color, as their equilibrium e was found for $(L_w)_e \neq 250$ mm. The balancing performance of the prototype at equilibrium e for a pendulum with L_w = 250 can be quantified by defining its moment error at this orientation:

$$(M_{error})_e = m_w g \sin\big((\psi_h)_e\big)\big((L_w)_e - L_w\big). \tag{66}$$

The balancing performance of the prototype for its intended ROM can be approximated by taking the average of the absolute moment errors for all e with $\psi_h \leq 30$ degrees:

$$M_{avg\_error} = \frac{1}{v}\sum_{u=1}^{v} |(M_{error})_u| \tag{67}$$

where u indicates an equilibrium e with $\psi_h \leq 30$ degrees and v equals the total amount of u equilibria.

The prototype's validation test yields a M_{avg_error} of 0.00945 Nm. This value can be related to the total moment force of the pendulum, M_{pend} (see equation 52), which is depicted in the colorbar in the upper-right corner of Figure 28 for several values of ψ_h. For reference, this average moment error is equal to the moment force of the pendulum (with L_w = 250 mm) at an angle ψ_h = 1.9 degrees.

Finally, the average moment reduction of the prototype for a pendulum with L_w = 250 mm can be calculated with equation:

$$\eta_{red} = \left(1 - \frac{M_{avg\_error}}{M_{avg}}\right) \cdot 100\%, \tag{68}$$

where M_{avg} is the average moment force of the pendulum under angle $(\psi_h)_u$:

$$M_{avg} = \frac{1}{v}\sum_{u=1}^{v} \sin\big((\psi_h)_u\big)\left(m_w g L_w + m_r g \frac{L_r}{2}\right) \tag{69}$$

Based on the identified equilibria in the prototype's validation test, the moment reduction η_{red} equals 90.5%.

D. TetraFEM validation

The accuracy of the TetraFEM tool can be assessed by comparing the prototype's deformation to the simulated joint's deformation under equal conditions. The process and result of this TetraFEM validation test are explained in this subsection.

During the prototype's validation test, the P_O-indicator showed minimal shifting while identifying the equilibria and its distance to the P_{EE}-indicator is fixed at R_{Mo}. As a result, the P_{EE}-indicator points found for the prototype are located on a nearly ideal spherical segment, where each point can represent the polar and azimuth angles, respectively $(\psi_h)_e$ and $(\psi_w)_e$, of the pendulum in equilibrium e. The TetraFEM tool can be validated using these P_{EE}-indicator points as P_{goal} points in the simulation, similar to what is described in the Pendulum Balancing Test in II-C4.

The simulated joint in the TetraFEM tool is subjected to the force of a pendulum with angles $(\psi_h)_e$ and $(\psi_w)_e$, and with its weights at $(L_w)_e$. The prototype is based on the best-performing result from the ten-tetrahedra optimization and this simulated joint therefore shares the same independent geometric parameters (see Table III) and fixed parameters (see Table I). The self-weight of the simulated joint is considered to be significant and is applied in the same manner as in the ten-tetrahedra result. For an ideal TetraFEM tool, the simulated joint with these conditions would deflect all $(P_{EE})_e$ points precisely to their corresponding $(P_{goal})_e$ points, as the prototype is in equilibrium at each $(P_{goal})_e$ point for a pendulum with $L_w = (L_w)_e$.

Figure 29 depicts the deflected P_{EE} points for 17 P_{goal} points. These P_{goal} points correspond to the P_{EE}-indicator points identified for $(L_w)_e = 250 + 7.5n$ mm, where n equals the integers from -6 to 2. The selection of P_{goal} points was made to ensure clarity in Figure 29 and avoid overlapping points.

The error of the TetraFEM tool in calculating the prototype's deformation can be quantified by determining d_{avg}, the average distance between the $(P_{goal})_e$ and $(P_{EE})_e$ points, in a similar way as described in equation 63. The deformation resemblance η_{dr} between the simulated joint and the prototype can then be calculated in a similarly to equation 64, where m = 17 and s_{avg} represents the average deflected distance of the $(P_{goal})_e$ points in the figure. This η_{dr} value reflects the overall accuracy of the TetraFEM tool in calculating the prototype's nonlinear deformations in this validation test, which is 89.6%.

IV. Discussion

The eight-tetrahedra optimization, despite having the fewest variables to optimize, yielded unique independent geometric

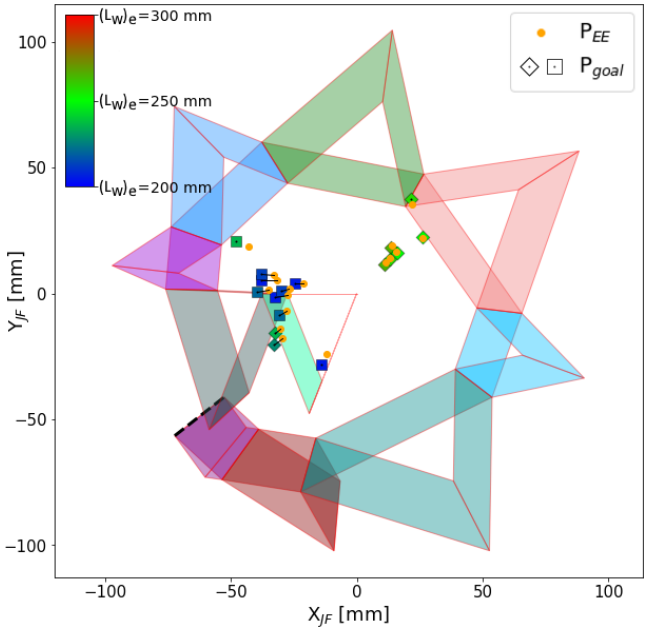

Fig. 29: A top-view of the simulated joint with 17 P_{goal} and P_{EE} points. Each P_{goal} point matches the position of a P_{EE}-indicator point in equilibrium.

parameters across all 50 optimization runs. This suggest that the lowest identified objective function is unlikely to represent the global minimum for any of the five seperate optimizations. Consequently, additional optimization runs from random starting points or tighter boundaries for the variables could potentially produce better performing simulated pendulum balancers than what identified in this research.

Nevertheless, the ten-tetrahedra optimization resulted in the independent geometric parameters of a well-performing simulated pendulum balancer, achieving a η_{dr} of 95.6% in the TetraFEM tool. For comparison, the simulated Tetra I design of Rommers et al. [19], which is not optimized for balancing a pendulum, yields a significantly lower deformation resemblance with a η_{dr} 32.5%. The optimized pendulum balancer was also tested for 28 evenly distributed P_{goal} points, which resulted in a very similar η_{dr} of 95.7%. This demonstrates that testing for only eleven evenly distributed P_{goal} points is sufficient to cover the intended ROM of the simulated joint.

The TetraFEM tool has demonstrated its computational efficiency in simulating the deformation of a compliant spherical joint consisting of tetrahedra connected in series. Eleven nonlinear deformations were calculated in each iteration of the ten-tetrahedra optimization process, with a computation time of approximately 256 seconds per iteration. The majority of this time is dedicated to calculating the center of gravity of the flexures and iteratively recalculating and reapplying the self-weight. The results show that if the self-weight of the joint is negligible, the computation time significantly decreases to just 18 seconds per iteration. The ten-tetrahedra-NSW optimization also yielded improved performance, with a η_{dr} of 96.6%. However, the results underscore that with the material properties and pendulum parameters outlined in Table I, the self-weight of the simulated joint should not be neglected for accurately calculating deflections. To reduce total computation time while still accounting for self-weight, optimization runs could be divided into two stages. The first stage would optimize random initial variables while neglecting self-weight, therefore achieving fast results. In the second stage, these optimized variables would be further refined with self-weight considerations, requiring few additional iterations to complete the optimization.

The results show that computation time can also be reduced by decreasing the number of tetrahedra in the simulated joint. This reduction lowers the computation time per iteration to 118 seconds and, on average, decreases the number of iterations required to complete the optimization run, as fewer variables need to be optimized. The eight-tetrahedra optimization yields a pendulum balancer which even slightly outperforms the ten-tetrahedra counterpart with a η_{dr} of 96.0%. This suggests that eight tetrahedra still provide sufficient optimizable variables to create effective pendulum balancers under the given conditions in the simulation. However, it is important to note that the deformation of each individual tetrahedron is assumed to be linear. If a simulated joint consists of few tetrahedra and is subjected to large deformations, this assumption might lose its validity. For relatively small deformations, such as $\psi_h \leq$ 30 degrees, using more than ten tetrahedra in a spherical joint is generally not recommended, as it does not seem to enhance performance and significantly increases computation time. This further supported by the twelve-tetrahedra optimization result, which performs worse than the ten-tetrahedra version with a η_{dr} of 95.5% and has a computation time of 464 seconds per iteration.

Of the five optimizations, the ten-tetrahedra-90x90 optimization was the least successful with its best performing result scoring a η_{dr} of 92.0%. A well performing pendulum balancer for this range of motion should have a rotational stiffness close to 0 Nm at $\psi_h = 90$ degrees, as M_{pend} is approximately constant at that point. However as the P_{EE} points in Figure 24 and 25 indicate, this stiffness profile could not be found in the ten-tetrahedra-90x90 optimization for the given conditions and assumptions in the TetraFEM tool. A better result for this optimization might be achieved by exploring the possibility of optimizing t_{min} for individual tetrahedra, or by investigating the effects of replacing some of the straight flexures in the simulation with curved flexures or flexures with a varying width along its length.

With the exception of the ten-tetrahedra-90x90 optimization, all best performing simulated pendulum balancers exhibit the shape of a spiral rotating counterclockwise from T_1 to T_n. However, this spiral shape is neither imposed by the boundaries in the TetraFEM simulation nor is it a prerequisite for well-performing pendulum balancers. Taking the ten-tetrahedra optimization as an example, the result with the lowest objective function yields a completely different shape, but self-intersects in undeformed state and is therefore

not considered to be the best result. The result with the third lowest scoring objective function for this optimization is not shaped like a spiral either and does not self-intersect in undeformed state. However, it's objective function d_{avg} is higher than the simulated pendulum balancer featured in Figure 16 and is therefore not mentioned in the Results section III.

The identified equilibria demonstrate optimal balancing behaviour of the prototype at those orientations for a pendulum with its weights at $(L_w)_e$. The prototype's average moment reduction for a pendulum with L_w = 250mm is 90.5%, based on the identified equilibria, indicating that the prototype is a fairly effective pendulum balancer for its intended pendulum and ROM. However, it is important to recognize that no equilibria were identified for large portions of the intended ROM, limiting the available data on the prototype's balancing performance in these areas. This limitation arises from the experimental setup used in this study. While a different setup involving force sensors could capture data across the entire ROM, as demonstrated in [28], it would also constrain the joint's movement and require meticulous calibration for each measurement. To avoid these complexities, a simpler, non-invasive measurement setup was selected for this study.

Figure 28 shows that the prototype with a pendulum has two distinct regions where the majority of equilibria were found: One region contains only unstable equilibria where $(L_w)_e \approx 250$ mm and another region with primarily stable equilibria where $200 \leq (L_w)_e \leq 250$ mm. The difference in the prototype's stiffness between the stable and unstable region can be attributed to creep, as the prototype was somewhat compressed during shipment. This compression has effectively reduced its rotational stiffness in orientations where $\psi_w \approx 225$ degrees (the stable region) and increased the stiffness in orientations where $\psi_w \approx 45$ degrees (the unstable region). Considering the effects of creep, the overall stiffness of the prototype is somewhat lower than ideal. This could be due to the effective Young's modulus of the prototype's material being slightly lower than its advertised tensile modulus of 1700 MPa [37].

The TetraFEM tool validation III-D confirms that the deflections of the simulated joint in the TetraFEM tool correspond pretty well to the deflections of the prototype. The η_{dr} of 89.6% would likely be even higher without the prototype's creep and presumably its lower Young's modulus, as the TetraFEM tool was the most accurate for equilibria where $(L_w)_e \approx 250$ mm.

For future work, the TetraFEM tool could be improved by incorparating an algorithm to detect self-intersections of the simulated joint, enabling the application of penalties during optimization runs. This feature should ideally identify self-intersections in both the undeformed and deformed states within the intended ROM, as optimizing a pendulum balancer for a certain ROM is pointless if it cannot be reached. A simpler method to prevent self-intersections could involve adjusting the fixed independent parameter R_{in} between tetrahedra, either gradually or in steps, allowing them to deflect past each other. Another potential avenue for future research could involve simulating contact mechanics and using it as a source of nonlinearity to improve the pendulum balancing performance of the simulated joint.

For large deflections, the optimizations could yield better results if t_{min} is allowed to be optimized for individual flexures. However, this approach would significantly increase the number of variables and iterations required per optimization run. An alternative approach might involve deliberately introducing nonlinear behaviour in some of the tetrahedra, which presents an interesting avenue for future research. This could result in zero rotational stiffness at $\psi_h \approx 90$ degrees or even negative rotational stiffness at ψ_h 90 by replacing some of the straight flexures with curved or varying-width flexures. While the TetraFEM tool could still be utilized for this research, the flexure compliance analysis would need substantial modifications to account for the changed flexures, as the current TetraFEM tool assumes the flexures to behave like Timoshenko beams and with linear deflections.

The prototype can be improved by manufacturing it from a metal, ideally one with a high Young's modulus-to-density ratio. Metals are generally less susceptible to creep and stress relaxation, which would increase the resemblance between the prototype and the simulated pendulum balancer, potentially improving the prototype's performance. Additionally, if the material has a high Young's modulus-to-density ratio, the self-weight of the prototype might be negligible, leading to increased computational efficiency in the TetraFEM tool. This efficiency gain would allow for more optimization runs or the inclusion of additional variables, likely resulting in better optimization outcomes and enhanced prototype performance.

Metal prototypes could be manufactured using 3D printing with titanium alloys or, as a more cost-effective alternative, by laser-cutting flexures that are manually assembled and fixed to small intermediate bodies to orient the flexures. In this latter approach, the flexures would have rectangular rather than trapezoidal cross-sections, necessitating adjustments to the flexure compliance analysis within the TetraFEM tool to accommodate this design change. Addtionally, the compliant spherical joint will likely exhibit slightly more parasitic motion of the center of rotation with these flexures [25].

Another approach to counteract the effects of self-weight and thus improve computational efficiency is the addition of a second pendulum balancing spherical joint on the lower half of the sphere. This second balancer should be attached to the pendulum as well and be a mirrored, but otherwise identical version of the upper balancer. This setup would cancel out the moments due to self-weight of the combined mechanism. Self-intersection can be avoided by selecting pendulum balancers that only occupy the upper half of the sphere, such as the ten-tetrahedron result shown in Figure 30. Additionally, adjusting the parameter R_{in}, as previously mentioned, could further help prevent self-intersections. The stiffness of the spherical joint would double, allowing it to balance a pendulum twice

as heavy. Although this double pendulum balancer might be challenging to implement in an exoskeleton due to its dimensions, it remains a promising area for future research.

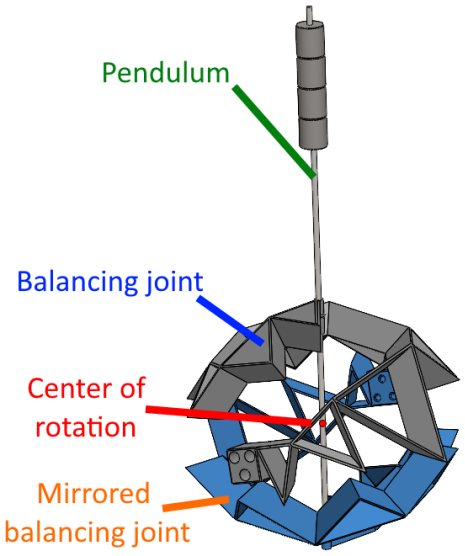

Fig. 30

V. Conclusion

This paper presents the first fully compliant pendulum balancer with a spherical range of motion. Its design is based on the Tetra I joint [19], with its geometry optimized for balancing a certain pendulum. The study includes optimizations under five different conditions to better understand the potential and limitations of these pendulum balancers. From one of these optimizations, the best performing result is 3D printed. This prototype achieved an average moment reduction of 90.5% in the regions where equilibria were identified.

Additionally, this paper presents and validates the TetraFEM tool, an algorithm developed as a computationally efficient alternative to the currently available FEM-based simulation software. The TetraFEM tool is used to simulate and optimize the behaviour of a pendulum balancing spherical joint consisting of tetrahedra connected in series. The simulated joint's deformation corresponds well to the prototype's deformation with an accuracy of 89.6%. In this study, the TetraFEM tool was extensively utilized to quantify the pendulum balancing performance during optimization runs and demonstrates potential for similar applications, such as calculating and optimizing the functional component of a shoulder exoskeleton.

References

[1] Y. R. Chheta, R. M. Joshi, K. K. Gotewal, and M. ManoahStephen, "A review on passive gravity compensation," in *2017 international conference of electronics, communication and aerospace technology (ICECA)*, vol. 1. IEEE, 2017, pp. 184–189.

[2] J. Herder, "Energy-free systems; theory, conception and design of statically balanced spring mechanisms," Ph.D. dissertation, Delft University of Technology, 2001.

[3] M. French and M. Widden, "The spring-and-lever balancing mechanism, george carwardine and the anglepoise lamp," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 214, no. 3, pp. 501–508, 2000.

[4] W.-B. Lee, S.-D. Lee, and J.-B. Song, "Design of a 6-dof collaborative robot arm with counterbalance mechanisms," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3696–3701.

[5] H.-S. Kim and J.-B. Song, "Low-cost robot arm with 3-dof counterbalance mechanism," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4183–4188.

[6] H.-C. Hsieh, L. Chien, and C.-C. Lan, "Mechanical design of a gravity-balancing wearable exoskeleton for the motion enhancement of human upper limb," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4992–4997.

[7] P. Bilancia and G. Berselli, "Conceptual design and virtual prototyping of a wearable upper limb exoskeleton for assisted operations," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 15, no. 4, pp. 525–539, 2021.

[8] L. L. Howell, "Compliant mechanisms," in *21st Century Kinematics: The 2012 NSF Workshop*. Springer, 2013, pp. 189–216.

[9] ——, *Introduction to Compliant Mechanisms*. John Wiley Sons, Ltd, 2013, ch. 1, pp. 1–13. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118516485.ch1

[10] G. Radaelli and J. Herder, "A monolithic compliant large-range gravity balancer," *Mechanism and Machine Theory*, vol. 102, 2015.

[11] ——, "Gravity balanced compliant shell mechanisms," *International Journal of Solids and Structures*, vol. 118, pp. 78–88, 2017.

[12] B. L. Rijff, J. L. Herder, and G. Radaelli, "An energy approach to the design of single degree of freedom gravity balancers with compliant joints," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 54839, 2011, pp. 137–148.

[13] J. Rommers, G. Radaelli, and J. L. Herder, "A design tool for a single vertex compliant-facet origami mechanism including torsional hinge lines," *Journal of Mechanisms and Robotics*, vol. 9, no. 6, p. 061015, 2017.

[14] S. Abouheidari, G. Radaelli, and J. Herder, "Synthesis of nonlinear torque-angle profile using compliant helicoidal shell joint," Master's thesis, Delft University of Technology, 2024, available at: http://resolver.tudelft.nl/uuid:71500cd9-d2e3-4c02-8aca-63edad0638b2.

[15] A. Amoozandeh Nobaveh, G. Radaelli, and J. L. Herder,

"A design tool for passive wrist support," in *Wearable Robotics: Challenges and Trends: Proceedings of the 5th International Symposium on Wearable Robotics, WeRob2020, and of WearRAcon Europe 2020, October 13–16, 2020*. Springer, 2022, pp. 13–17.

[16] M. Tschiersky, E. E. Hekman, J. L. Herder, and D. M. Brouwer, "Gravity balancing flexure spring mechanisms for shoulder support in assistive orthoses," *IEEE Transactions on Medical Robotics and Bionics*, vol. 4, no. 2, pp. 448–459, 2022.

[17] S. v. d. Kemp and J. Herder, "Design of a compact wearable arm support utilizing shape optimized shell mechanisms: To be worn underneath the surgical gown," 2018, available at: http://resolver.tudelft.nl/uuid:3664c6fd-da05-48b5-a503-a9e373d912a0.

[18] R. Mak, A. Amoozandeh Nobaveh, G. Radaelli, and J. L. Herder, "A Curved Compliant Differential Mechanism With Neutral Stability," *Journal of Mechanisms and Robotics*, vol. 16, no. 1, p. 011003, 03 2023. [Online]. Available: https://doi.org/10.1115/1.4056867

[19] J. Rommers, V. van der Wijk, and J. L. Herder, "A new type of spherical flexure joint based on tetrahedron elements," *Precision Engineering*, vol. 71, p. 130–140, 2021.

[20] Y. Jingjun, P. Xu, S. Minglei, Z. Shanshan, B. Shushing, and Z. Guanghua, "A new large-stroke compliant joint micro/nano positioner design based on compliant building blocks," in *2009 ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*, 2009, pp. 409–416.

[21] F. Parvari Rad, G. Berselli, R. Vertechy, and V. Parenti-Castelli, *Stiffness Analysis of a Fully Compliant Spherical Chain with Two Degrees of Freedom*. Cham: Springer International Publishing, 2014, pp. 273–284. [Online]. Available: https://doi.org/10.1007/978-3-319-06698-1_29

[22] F. P. Rad, "Design and characterization of curved and spherical flexure hinges for planar and spatial compliant mechanisms," 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:106658342

[23] F. P. Rad, G. Berselli, R. Vertechy, and V. P. Castelli, "Compliance based characterization of spherical flexure hinges for spatial compliant mechanisms," in *Advances on Theory and Practice of Robots and Manipulators*, M. Ceccarelli and V. A. Glazunov, Eds. Cham: Springer International Publishing, 2014, pp. 401–409.

[24] F. Parvari Rad, R. Vertechy, G. Berselli, and V. Parenti-Castelli, "Design and stiffness evaluation of a compliant joint with parallel architecture realizing an approximately spherical motion," *Actuators*, vol. 7, no. 2, 2018. [Online]. Available: https://www.mdpi.com/2076-0825/7/2/20

[25] F. Parvari Rad, R. Vertechy, G. Berselli, and V. Parenti-Castelli, "Analytical compliance analysis and finite element verification of spherical flexure hinges for spatial compliant mechanisms," *Mechanism and Machine Theory*, vol. 101, pp. 168–180, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094114X16000112

[26] G. Berselli, F. Parvari Rad, R. Vertechy, and V. Parenti Castelli, "Comparative evaluation of straight and curved beam flexures for selectively compliant mechanisms," in *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2013, pp. 1761–1766.

[27] F. Parvari Rad, R. Vertechy, G. Berselli, and V. Parenti-Castelli, *Compliant Serial 3R Chain with Spherical Flexures*. Cham: Springer International Publishing, 2018, pp. 11–21. [Online]. Available: https://doi.org/10.1007/978-3-319-56802-7_2

[28] D. Hogervorst, G. Radaelli, and J. Herder, "A neutrally stable quasi-compliant spherical joint with a remote center of rotation," 11 2022.

[29] L. L. Howell, *Compliant mechanisms*. John Wiley Sons, 2001, ch. 7.

[30] S. Zhang and E. Fasse, "A finite-element-based method to determine the spatial stiffness properties of a notch hinge," *Journal of Mechanical Design - J MECH DESIGN*, vol. 123, 03 2001.

[31] H. Vallery, A. Schwab, and T. U. (Delft), *Advanced Dynamics*. Delft University of Technology, 2017. [Online]. Available: https://books.google.nl/books?id=gUd8tgEACAAJ

[32] G. K. Smyth, "Numerical integration," in *Encyclopedia of Biostatistics*, P. Armitage and T. Colton, Eds. London: Wiley, 1998, pp. 3088–3095.

[33] S. D. Team, "Sympy: Physics vector classes api documentation," https://docs.sympy.org/latest/modules/physics/vector/api/classes.html, accessed: 2024-08-25.

[34] Delft High Performance Computing Centre (DHPC), "DelftBlue Supercomputer (Phase 2)," https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[35] S. D. Team, "Scipy: Cobyla optimization method in minimize function," https://docs.scipy.org/doc/scipy/reference/optimize.minimize-cobyla.html, accessed: 2024-08-25.

[36] J. Rommers, "New spherical flexure joint designs (compliant mechanisms)," YouTube, 2021, accessed: 2024-08-25. [Online]. Available: https://www.youtube.com/watch?v=DAngcygU7tc

[37] Materialise, "Pa12 mjf 3d printing material," https://www.materialise.com/en/industrial/3d-printing-materials/pa12-mjf, accessed: 2024-08-25.

A

Literature Report

This appendix presents the literature research performed before the start of the main thesis project. The review aims to collect and categorize papers in the field of compliant passive gravity balancers. One of the gaps found in the literature inspired the research performed in the main paper of this thesis.

Review on Passive Gravity Balancers using Compliant Mechanisms

Riley Barendse

***Abstract*—Passively compensating for the gravitational forces of a mass in a system is traditionally achieved using counterweights or linear springs. Recently, some designs emerged that utilize the strain energy of deflected compliant mechanisms for gravity balancing. Compliant mechanisms offer several advantages over their rigid linked counterparts, including lower costs, less weight, reduced bulkiness and the absence of friction, wear or backlash. Therefore, the utilization of compliant mechanisms can result in superior gravity balancers, but the literature on Compliant Passive Gravity Balancers (CPGBs) is currently limited. This literature review aims to collect and categorize papers concerning CPGB designs while identifying unexplored areas within the current literature for future research. The categorization is based on balancing type and dimensional configuration, which are respectively subdivided in level of compliance and range of motion. Gaps found in the literature are in the field of pendulum balancers with a spatial range of motion, fully compliant pendulum balancers with a spherical range of motion and mass balancers as a whole. Directions for future research in the latter two gaps are proposed.**

***Keywords*—Passive Gravity Balancing, Compliant Mechanisms, Pendulum Balancers, Mass Balancers**

I. Introduction

For centuries, humans have utilized passive methods to counteract gravitational loads. One of the simplest techniques involves using counterweights, a mechanism still widely used in structures like bascule bridges, elevators, and cranes. However, a drawback of this approach is the increased mass, inertia, and volume it introduces to the system [1]. An alternative method, which avoids the drawbacks associated with counterweights, is the use of springs to balance weight [2]. Linear springs have found applications in gravity balancing for various devices such as desk lamps [3], robot arms [4, 5] and exoskeletons [6, 7]. While linear springs effectively compensate for gravitational forces, they fall short in addressing friction forces within rigid linked systems. The operating force will need to overcome this friction before movement in these systems is possible.

Compliant mechanisms gain their motion from the deflection of elastic members [8] rather than sliding or rolling contact and can therefore experience zero friction or wear. Among other advantages, compliant mechanisms can also have less weight, less assembly parts and reduced or zero backlash [9]. However, the fact that strain energy is stored in a deflected compliant joint is often undesired as it requires energy to get the desired motion from the joint and a continuous force is required to remain in a certain configuration. This phenomenon can be negated by creating zero stiffness structures [10], but can also be used to compensate the gravitational forces acting on the compliant mechanism, creating a passive gravity balancer with the aforementioned advantages of compliant mechanisms, including being frictionless. This concept could be used to make cost-effective passive gravity balancers with superior performance compared to their traditional rigid-body counterparts.

Quite some research has been done on classifying compliant mechanisms: [11] presents a classification of compliant mechanisms based on deformation behaviour, [12] categorizes based on functionality, behaviour and structure, [13] categorizes based on flexure type and [14] classifies specifically zero stiffness compliant rotary joints based on type of compliant rotary joint and the zero stiffness working principle of the mechanism.

The literature on passive gravity balancing has been reviewed as well: [1] presents a classification of passive gravity compensators based on balancing mechanism and [15] classifies based on both the balancing mechanism and its application. However, utilizing compliant mechanisms in passive gravity ballancers is not recognized in either the literature reviews on compliant mechanisms or the reviews on passive gravity balancers. Despite its great potential, a literature review on the utilization of compliant mechanisms in passive gravity balancers is currently lacking. This paper aims to fill this gap by gathering existing CPGB (Compliant Passive Gravity Balancer) designs, categorizing them based on functionality, dimensional characteristics, level of compliance and range of motion and identifying the unexplored areas in the current literature for future research

Section II outlines the systematic approach for gathering relevant literature and explains the criteria for selecting the papers, followed by the categorization process. Section III will present the corresponding results and will briefly explain the contents of the found literature. The method and results will be interpreted and discussed in section IV and finally section V will conclude this review and give recommendations for future research.

II. Method

A. Literature search

The literature search was conducted in January 2023 using the Scopus, Web of Science and TU Delft repository databases. The author, currently pursuing a Master's degree at TU Delft,

TABLE I
SEARCH STRING USED IN SCOPUS

	AND			
	AND		W/10	
OR	Compliant	Passive*	Gravity	Balanc*
	Flexure*	Static*	Weight	Stabili*
	Shell	Assistive	Mass	Compensat*
	”Torsion bar*”		Pendulum	Equilibrator
	”Planar PRE/4 spring*”		”Constant force”	
			”Constant load”	
	Publication Year >2003			

includes the TU Delft repository as he acknowledges the relevant research conducted by former students, some of which may not be found on Scopus or Web of Science.

A search string for the title, keywords or abstract was developed to systematically find relevant literature concerning CPGB designs in these three databases. Table II displays the search string used for the Scopus database incorporating several operators to retrieve the most relevant literature as output. Terms with an asterisk represent any word that contains the preceding part of the term. The ’PRE/4’ Scopus operator in table II indicates that four words may be present between ’planar’ and ’spring’ because planar springs are relevant but may have words like ’compression’, ’extension’ or both separating the two terms. The ’W/10’ Scopus operator means that only ten words are allowed between the third and fourth column (or vice versa) to increase the relevance of the database results. Web of Science lacks a proximity operators similar to ’PRE/’ and therefore its search string has both ’PRE/’ and ’W/’ substituted with the ’NEAR/’ operator, which serves the same function as the ’W/’ operator of Scopus. The repository of TU Delft has no proximity operators and therefor has ’PRE/4’ and ’W/10’ replaced with the ’AND’ operator in its search string. Additionally, the quotation marks around ”Torsion bar*” and ”Planar PRE/4 spring*” were replaced with parentheses as the TU Delft repository does not seem to recognize the asterisk as a wildcard when placed between quotation marks. Finally, only literature published in the last 20 years was considered as CPGBs are a relatively new concept, and no relevant database results published prior to 2004 are expected.

B. Criteria

To be included in this review, a paper in the literature search must meet four conditions. Firstly, the full paper must be accessible for a TU Delft student, such as the author, without requiring any additional purchases. Secondly, the system described in the paper must have a balanced weight, resulting in the potential energy of the system being (nearly) constant for its intended (non-zero) range of motion. Thirdly, the gravity compensation mechanism of the system in the paper needs to be passive and may not require energy expenditure for balancing. The fourth and final criterion is that a compliant mechanism must be included in the gravity compensation mechanism of the paper. Mechanisms utilizing rubber bands and helical compression/tension springs can often not be made monolithically and these components are generally attached to the mechanism using contact based hinges. Consequently, rubber bands or helical compression/tension springs do not suffice as compliant mechanisms for this literature review. In contrast, flexures, torsion bars, torsion springs, compliant shells, and planar springs are deemed regarded as compliant mechanisms in this review.

C. Categorization

The CPGB designs in this review will be labeled either as a ’Pendulum balancer’ or as a ’Mass balancer’. Pendulum balancers can primarily rotate the balanced mass over a nearly constant radius. On the other hand, mass balancers can translate a balanced mass along a near-straight line parallel to the force of gravity. Both the pendulum balancers and mass ballancers are classified as either ”Fully compliant” if all relevant joints are compliant or ”Partially compliant” if some traditional rigid-body joints are utilized in the mechanism as well.

The CPGB designs will also be categorized based on their dimensional configuration and their (range of) motion. In this paper a design has a planar configuration if it can be easily represented with a single two dimensional image, for example (a planar combination of) planar lumped/distributed flexures or coil springs. On the other hand, designs with a spatial configurations extend into three dimensions requiring a three-dimensional perspective of the design, for example torsion bars, origami structures or spatial beams. CPGB designs with a spatial configuration will be further divided into ”Planar motion,” ”Spherical motion,” and ”Spatial motion” based on the range of motion of the gravity balancer.

III. RESULTS

The literature search as described in subsection II-A resulted in 19 papers which met the criteria from subsection II-B. Four additional papers were found by checking the references of the most relevant literature and one other paper

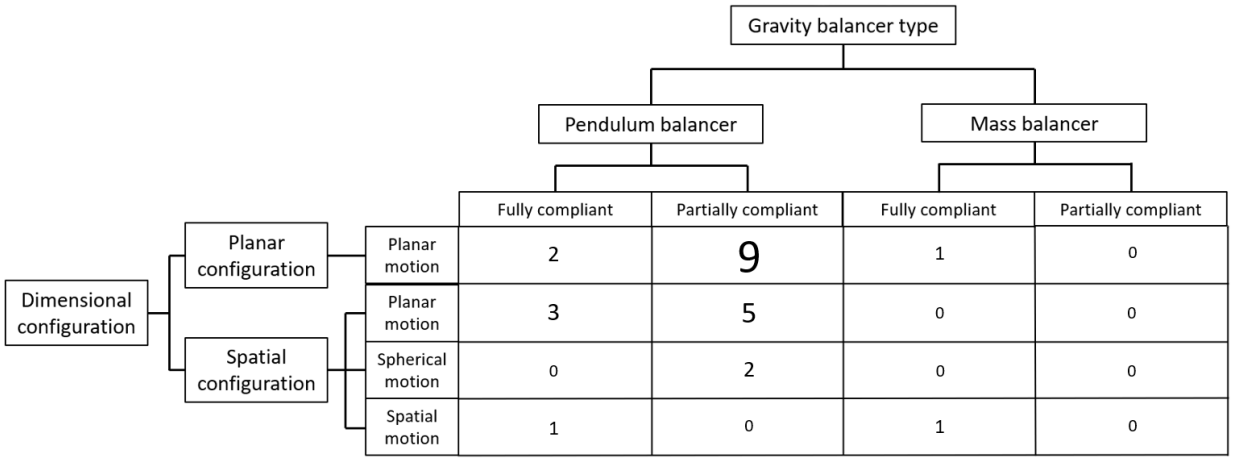

Fig. 1. Literature sorting table

is a master thesis received through private communication from a fellow student who recently graduated. The designs in the 24 papers are categorised in table 1 as described in subsection II-C and will be outlined in this chapter.

A. *Pendulum balancers - Fully compliant*

1) Planar configuration - Planar motion [16, 17]: Two fully compliant pendulum balancers with a planar configuration and a planar range of motion were found as a result of the literature search.

Radaelli designed a fully compliant gravity balancer consisting of one single complex-shaped beam with constant thickness [16]. The beam was modeled as a planar isogeometric Bernoulli beam, had an optimized shape and was clamped on both ends. The prototype made from a polycarbonate sheet could successfully balance a substantial weight over an almost circular arc, as can be seen in figure 2.

Rijff designed a single degree of freedom gravity balancer where all of the rigid joints were replaced with compliant joints (torsion springs) [17]. This CPGB uses a four-bar mechanism with multiple rigid links which are connected by four compliant torsion springs, as can be seen in figure 3. The parameters of the links and springs are optimized such that it maximizes the moment reduction over its range of motion.

2) Spatial configuration - Planar motion [18–20]: Three fully compliant pendulum balancers with a spatial configuration a and planar range of motion were found as a result of the literature search.

Radaelli has used a shape optimization procedure to create statically balanced mechanisms where the self-weight of a compliant shell mechanism and an additional payload can be balanced by the elastic forces of the deforming shell [18]. Two prototypes were made out of PETG thermoplastic sheet

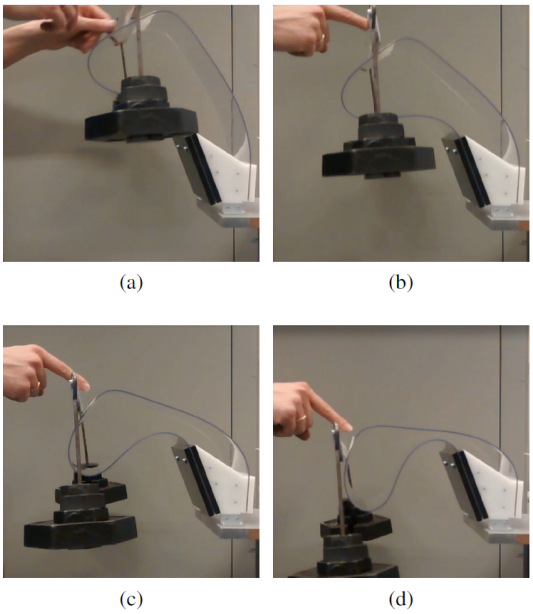

Fig. 2. Polycarbonate beam balancing a weight over its range of motion while clamped onto a support structure. Retrieved from [16].

material, which showed a good qualitative gravity balancing match, but not quantitatively. One of these prototypes can be seen in figure 4.

Rommers proposed a design tool in which hinge lines with torsional stiffness are used to design passive origami-like mechanisms [19]. Figure 5 shows the PRBM of the mechanism which has three hinge lines with each a torsional stiffness and a point C_m which is constrained in Y-direction. The angular rotations of the torsional hinge lines can be written as a function of θ_{joint} to construct the moment curve of the mechanism. The hinge lines are created by applying tape between two steel plates in an alternating pattern and

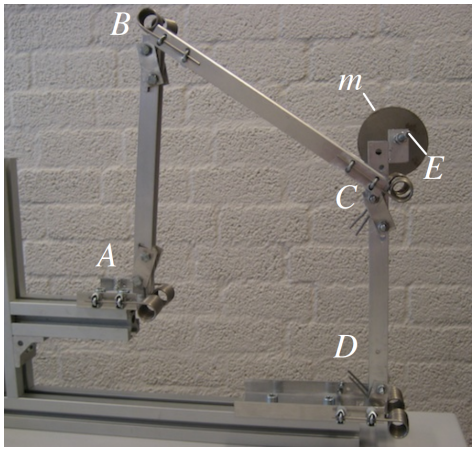

Fig. 3. The four-bar mechanism with rigid links, compliant joints (A,B,C and D), the end effector (E) and a balanced mass (m). Retrieved from [17].

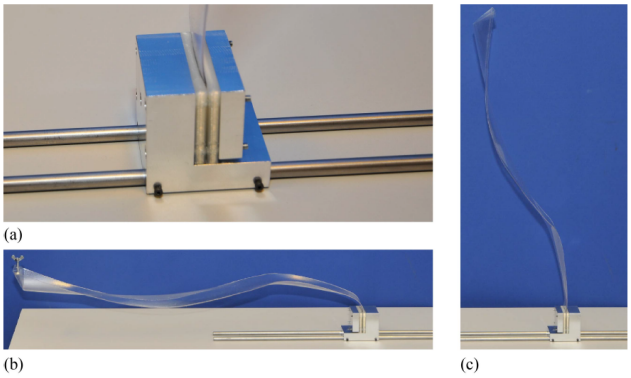

Fig. 4. One of the compliant shell prototypes. (a) The base during testing. (b) The loaded shell prototype in maximum deformation. (c) The unloaded shell. Retrieved from [18].

torsional stiffness is added by clamping torsion bars at both sides of the hinge lines. One of the example designs presented in the paper is a gravity compensating joint which could balance a mass of 0.4 kg between 40 and 120 degrees of rotation.

Abouheidari did research on the optimization and syntheses of a gravity-balancing torque-angle profile using a compliant helicoidal shell mechanism [20]. The mechanisms consist of several axisymmetric helicoidal shells with parameters such that its torque profiles match that of a sine function. One such mechanism is the L-shape design, see figure 6. Although the FEM shows excellent gravity balancing properties for the mechanisms, the experimental results illustrate a considerable difference, perhaps due to the unpredictable behaviour of the resin material used for production of the prototype.

3) Spatial configuration - Spherical motion: Zero fully compliant pendulum balancers with a spatial configuration and a spherical range of motion were found as a result of the literature search.

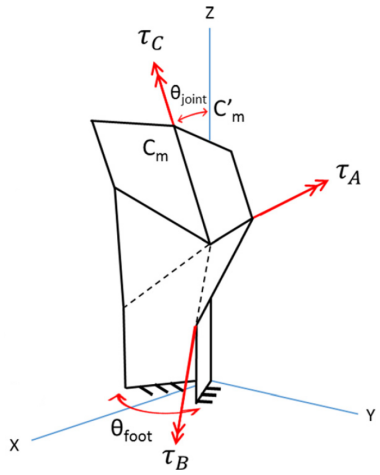

Fig. 5. PRBM of the passive origami-like mechanism. Retrieved from [19].

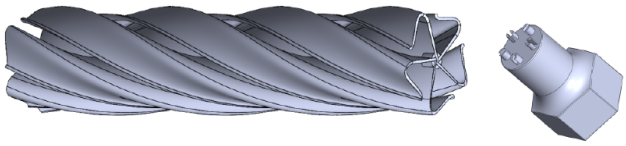

Fig. 6. The optimized L-shape design for a 90-degree gravity balancing profile including a tool to apply torque on the mechanism. Retrieved and modified from [20].

4) Spatial configuration - Spatial motion [21]: One fully compliant pendulum balancers with a spatial configuration and a spatial range of motion was found as a result of the literature search.

Nobaveh designed a passive wrist support using two compliant spatial beams as gravity balancer [21]. It attaches to the forearm, wrist and hand which are connected by the spatial beams as can be seen in figure 7. The overall shape and cross-section dimensions of the spatial beams are attained using an optimization technique and significantly reduces the muscle force needed for flexion and extension of the wrist, while remaining relatively compliant in supination and pronation direction as well. However, it should be noted that the range of motion with this wrist support is quite limited and still requires 30% of the usual muscle force for hand movement within that restricted range.

Fig. 7. The 3D printed prototype with forearm, wrist and hand support connected by the spatial beams. Retrieved from [21].

B. Pendulum balancers - Partially compliant

1) Planar configuration - Planar motion [22–30]: Nine partially compliant pendulum balancers with a planar configuration and a planar range of motion were found as a result of the literature search. In order to keep this paper concise, only two designs will be elaborated on.

Tschiersky designed three different planar flexure springs which can balance the weight of a forearm in an elbow orthosis [22]. Set A is monolithic and is optimized for a constant thickness, Set B is monolithic and is allowed to have a variable thickness and Set C is a concentric nested version of Set B, see figure 8. The nested spring design (set C) can store significantly more elastic energy, compared to Set A and B, by utilizing the otherwise unused space within the original spring envelop. All three designs show good gravity balancing behaviour for 180 degrees of rotation, but are not fully compliant as a revolute joint is required to counter parasitic motion when used for gravity balancing.

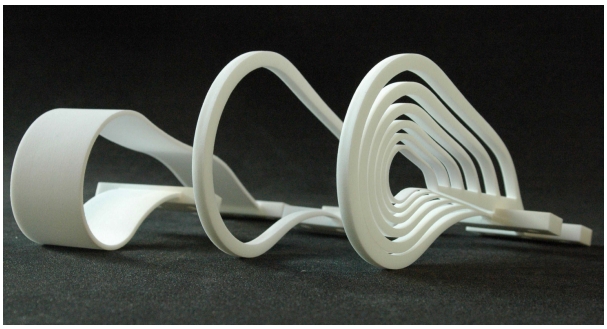

Fig. 8. From left to right: Set A, Set B, Set C. Retrieved from [22].

Cheng developed a planar two-DOF gravity compensator for both the upper arm and fore arm [26]. The gravity compensating effect is provided by two torsional compliant beams which have a torque profile optimized for gravity balancing. The design also includes a decoupling (parallelogram four-bar) mechanism, which can be seen in figure 9, for isolating the torsional effect between the linkages. The design shows good gravity balancing behaviour for 180 degrees in both joints, but is not fully compliant as several revolute joints are used for the decoupling mechanism and to counter parasitic motion.

2) Spatial configuration - Planar motion [31–35]: Five partially compliant pendulum balancers with a spatial configuration and a planar range of motion were found as a result of the literature search. In order to keep this paper concise, only two designs will be elaborated on.

Geerts designed a single degree of freedom arm support which can balance the weight of a forearm using an optimized compliant shell mechanism [31]. Thin McKibben muscles are used in the design additionally as artificial bicep muscles. The

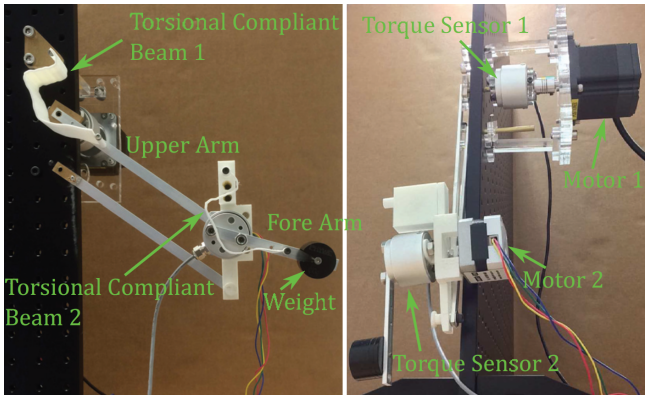

Fig. 9. Experimental setup of the gravity balanced upper arm and forearm using two torsional compliant beams and a decoupling mechanism. The motors and sensors in this picture are required for the measurements. Retrieved from [26].

compliant shell is placed below the elbow as can be seen in figure 10, and its (simulated) optimized torque profile matches the desired gravity balancing behaviour for a range of 120 degrees. This CPGB does require a revolute (elbow) joint to function and is therefore not considered fully compliant.

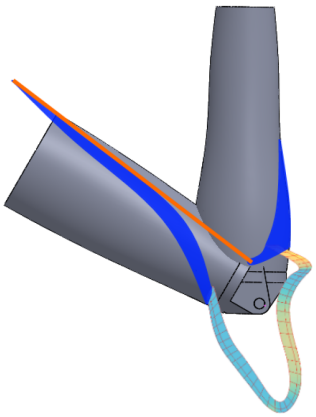

Fig. 10. The design of the compliant shell mechanism build around the elbow joint. Retrieved from [31].

Radaelli designed a statically balanced inverted pendulum using three parallel compliant torsion bars and mechanical stops [35]. The mechanical stops could engage and disengage the torsion bars, see figure 11, such that minimal work is required to rotate the pendulum in the CPGB over a range of about 90 degrees. The gravity balancing effect works exceptionally well, but the CPGB is not fully compliant due to the mechanical stops and revolute joint in the system.

3) Spatial configuration - Spherical motion [36, 37]: Two partially compliant pendulum balancers with a spatial configuration and a spherical range of motion were found as a result of the literature search.

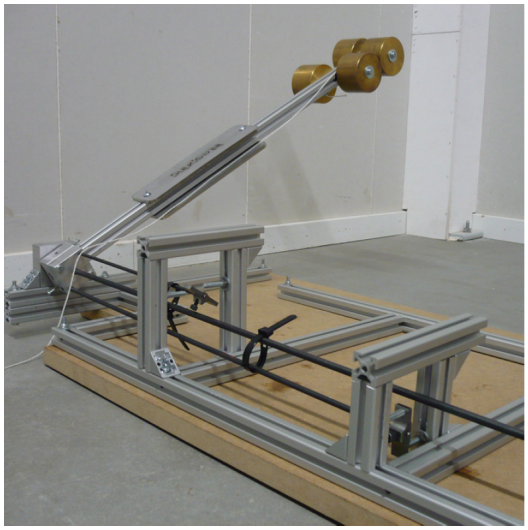

Fig. 11. Picture of the prototype with three (black) torsion bars, mechanical stops and a inverted pendulum. Retrieved from [35].

Tschiersky has done research on shoulder orthoses by exploring the use of mechanisms that employ a flexible element at the back that acts as energy storage, transmission and part of the load bearing structure [36]. He found the optimized shape and thickness contribution for the flexure in several kinematic support conditions. The two clamp-clamp designs, see figure 12 are the most compliant designs as they only require one revolute joint for the medial/lateral rotation of the shoulder.

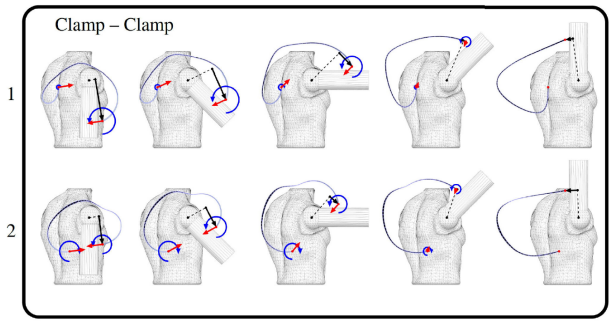

Fig. 12. Two designs of the clamp-clamp kinematic category positioned in five different shoulder flexion angles. Reaction moments (blue) and reaction forces (red) are present in both ends of the flexible element as both ends are fixed. Retrieved from [36].

Van der Kemp designed a compact arm support to reduce fatigue in the arms of surgeons during laparoscopic surgery [37]. This design uses a compliant shell mechanism attached to a frame on the back and a brace on the bicep, see figure 13. This CPGB reduces the forces required by shoulder muscles and is very compact and easy to make. However this arm support only balances for a small portion of the arms weight and is not fully compliant because of the linear guide and revolute joint.

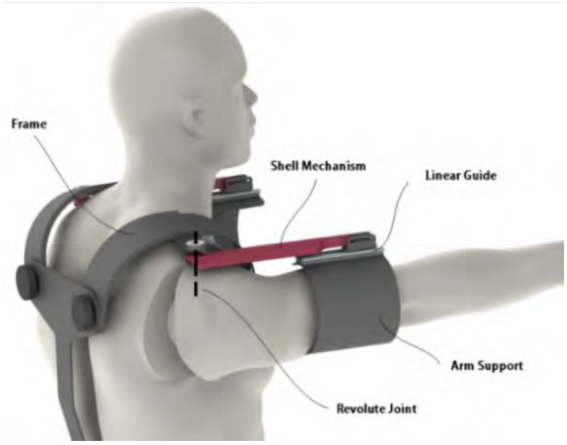

Fig. 13. Conceptual design of the arm support for surgeons during laparoscopic surgery. Retrieved from [37].

4) Spatial configuration - Spatial motion: Zero partially compliant pendulum balancers with a spatial configuration and a spatial range of motion were found as a result of the literature search.

C. Mass balancers - Fully compliant

1) Planar configuration - Planar motion [38]: One fully compliant mass balancer with a planar configuration and a planar range of motion was found as a result of the literature search.

Chen presented the concept of a weight compensator which employs a constant-force compliant mechanism [38]. It is proposed that the design of this CPGB can be used to compensate the weight of a humanoid robot when attached to its legs, see figure 14. This passive weight compensation could greatly improve the dynamic performance and energy efficiency of the humanoid robot. According to the calculations in the paper, the weight can be balanced very well for a large vertical range as long as the gravity force vector is in line with the base of the lowest flexure.

2) Spatial configuration - Planar motion: Zero fully compliant mass balancers with a spatial configuration and a planar range of motion were found as a result of the literature search.

3) Spatial configuration - Spherical motion: Zero fully compliant mass balancers with a spatial configuration and a spherical range of motion were found as a result of the literature search.

4) Spatial configuration - spatial motion [39]: One fully compliant mass balancer with a spatial configuration and a spatial range of motion was found as a result of the literature search.

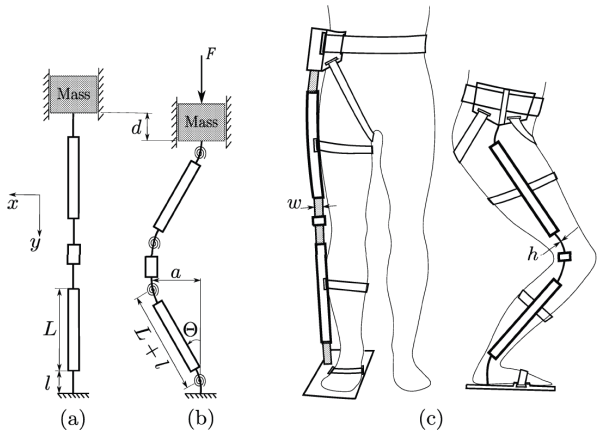

Fig. 14. (a) Schematic diagram of a constant-force mechanism balancing a mass, (b) the corresponding pseudo-rigid-body model (PRBM), (c) implementation example for a humanoid robot. Retrieved from [38].

Dunning designed a low stiffness six degree of freedom compliant precision stage [39]. The spatial structure, see figure 15, is designed such that it cancels out the stiffness of all six degrees of freedom for a small range of motion. This CPGB can balance a weight placed on the moving platform by combining bi-stable buckling beams with v-shaped positive stiffness beams, but also for a very small range of motion.

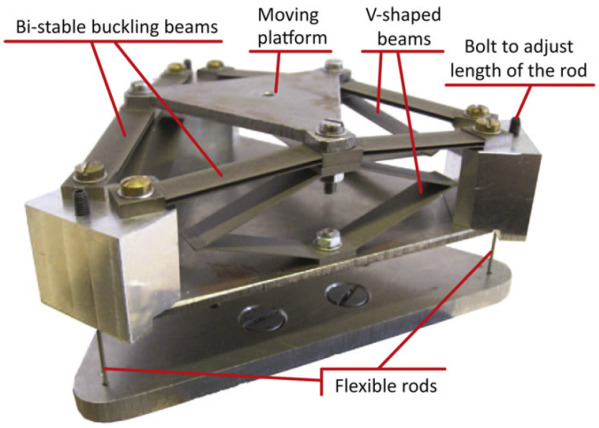

Fig. 15. The prototype of the low stiffness six degree of freedom compliant precision stage. Retrieved from [39].

D. Mass balancers - Partially compliant

No partially compliant mass balancers were found during this literature search.

IV. Discussion

Table II shows the search string which aims to find all relevant literature concerning Compliant Passive Gravity Balancer (CPGB) designs. Therefore it consists of synonyms or closely related terms for 'compliant', 'passive', 'gravity' and 'balancer'. These terms are combined with operators and resulted in 19 relevant papers. This leaves room for improvement as four other papers regarding CPGB designs [23, 27–29] were found by checking the references of the most relevant literature. It may be noted that these four papers all regard partially compliant pendulum balancers with a planar configuration and planar motion, but this observation is considered to be coincidental as the category in question also simply delivered the most results. Removing the words related to 'passive' from the search string would prevent overlooking three out of those four papers and was considered, but would lead to a copious amount of search results and was therefore discarded as an option.

From the results in the literature sorting table 1 it can be obtained that most compliant passive pendulum balancers have a planar range of motion. This observation is to be expected as pendulum balancers with a planar motion have only a single (rotational) degree of freedom over which to optimize the joints torque profile. Compliant pendulum balancers with a spherical or spatial range of motion, on the other hand, demand optimization for multiple degrees of freedom, resulting in a more intricate shape and a more complex design process. This complexity, coupled with the relatively recent emergence of compliant pendulum balancers as a research field, may account for the fewer CPGB designs observed in the literature with a spherical or spatial range of motion.

From table 1 it can also be obtained that generally more pendulum balancers are partially compliant than fully compliant. This observation is to be expected as well as rigid-link revolute joints can be a convenient way to eliminate parasitic (translating) motion. Fully compliant CPGB designs however require the compliant joint to have sufficient stiffness in undesired directions while remaining compliant in the desired directions. This increases the complexity of the design problem and can explain why less papers were found regarding fully compliant pendulum balancers compared to their partially compliant counterparts.

From table 1 it is evident that very few mass balancers were identified in this literature search. The absence of mass balancers can be attributed partially by the criteria in subsection II-B, as it excludes papers where compliant constant force mechanisms are used for other means than compensating a weight. In the majority of cases should a compliant constant force mechanism be able to balance a mass, given that the gravitational force acts as a constant force. This is demonstrated in sources [38, 39], both of which are consequently added to this literature review. However, other fully compliant constant force mechanisms [40–45] and partially compliant constant force mechanisms [46, 47], designed for purposed other than gravity balancing, have been excluded based on the specified criteria. Moreover, most of

the designs in these excluded sources only have a very small range of motion for the constant force mechanism, making them less interesting as potential mass balancers.

Source [48] requires an online purchase from the author for access and is therefore excluded from the literature review according to the criteria outlined in subsection II-B. The abstract suggests a pendulum balancer with a planar range of motion, but this cannot be confirmed without access to the full paper.

Furthermore it is noticeable that a substantial amount of pendulum balancers (10 out of 21 papers) are designed as a functional part of an orthoses or support of (part of) a human arm. This is motivated by people with reduced muscle strength, like Duchenne Muscular Disease patients, who could benefit from a weight compensating orthosis [49]. If integrated in an orthosis, a pendulum balancer can compensate the weight of a patients hand, lower arm or entire arm, reducing the muscle forces required for wrist, elbow or shoulder movements respectively. This support allows the wearer to regain mobility in the upper extremity limbs and improve their quality of life.

V. Conclusion

This literature review aimes to gather and categorize existing CPGB designs while identifying unexplored areas within the current literature for future research. To achieve this goal a literature search was conducted and the resulting relevant papers were categorized based on balancing type and dimensional configuration and subdivided in level of compliance and range of motion respectively. This categorization reveals several gaps in the current literature, offering potential directions for future research. One major gap in the literature concerns compliant mass balancers with only a few relevant papers falling within this whole category. Additionally, two other gaps can be found in the field of fully compliant pendulum balancers with a spherical range of motion, which yields zero results, and in the field of pendulum balancers with a spatial range of motion, which yields a single (partially compliant) result from the literature search.

Future research could be directed towards the development of more mass balancers using compliant constant force mechanisms. As gravitational force is constant, the conversion of existing compliant constant force mechanisms into compliant mass balancers should be relatively simple. However, currently this principle is not well explored yet and more research into this area could yield valuable novel designs.

Another interesting topic for future research is the design of a fully compliant pendulum balancer with a spherical range of motion. According to the literature search, such a device is yet to be designed. Two partially compliant counterparts are obtained from the literature search, both of which are designed to be the functional part of a shoulder orhtosis. Both spherical pendulum balancers allow a large range of motion and possess a remote center of rotation, which are prerequisites for inclusion in a shoulder orthosis. However, one of these designs protrudes very far from the body [36] and the other compensates only a fraction of the weight of the arm [37]. Additionally, both of these two designs are only partially compliant and still bear the disadvantages of rigid linked mechanisms. It seems that the design of a fully compliant pendulum balancer with a spherical range of motion could fill a gap in the literature and additionally be beneficial for people requiring a shoulder orthosis, assuming it has a sufficient range of motion and a remote centre of rotation.

A design strategy could be modifying the fully compliant spherical flexure joint presented by Jelle Rommers [50]. His design is formed by tetrahedron-shaped elements, each composed of three blade flexures with a trapezoidal shape, that are connected in series without intermediate bodies, see figure 16. This compliant joint has a remote center of rotation, is relatively compact, has a large spherical range of motion and a high translational stiffness, but is not optimized for gravity balancing. By tweaking the dimensions and angles between the tetrahedron-shaped elements a gravity balancing torque profile for this joint could be achievable, but this hypothesis will need to be validated in future research.

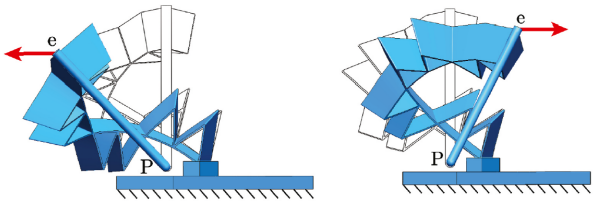

Fig. 16. A fully compliant spherical joint formed by tetrahedron-shaped elements connected in series. It is subjected to a horizontal load at e, showing that it rotates about point P with little parasitic motion. Retrieved from [50].

References

[1] Y. R. Chheta, R. M. Joshi, K. K. Gotewal, and M. ManoahStephen, "A review on passive gravity compensation," in *2017 international conference of electronics, communication and aerospace technology (ICECA)*, vol. 1. IEEE, 2017, pp. 184–189.

[2] J. Herder, "Energy-free systems; theory, conception and design of statically balanced spring mechanisms," Ph.D. dissertation, Delft University of Technology, 2001.

[3] M. French and M. Widden, "The spring-and-lever balancing mechanism, george carwardine and the anglepoise lamp," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 214, no. 3, pp. 501–508, 2000.

[4] W.-B. Lee, S.-D. Lee, and J.-B. Song, "Design of a 6-dof collaborative robot arm with counterbalance mechanisms," in *2017 IEEE International Conference on*

Robotics and Automation (ICRA). IEEE, 2017, pp. 3696–3701.

[5] H.-S. Kim and J.-B. Song, "Low-cost robot arm with 3-dof counterbalance mechanism," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4183–4188.

[6] H.-C. Hsieh, L. Chien, and C.-C. Lan, "Mechanical design of a gravity-balancing wearable exoskeleton for the motion enhancement of human upper limb," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4992–4997.

[7] P. Bilancia and G. Berselli, "Conceptual design and virtual prototyping of a wearable upper limb exoskeleton for assisted operations," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 15, no. 4, pp. 525–539, 2021.

[8] L. L. Howell, "Compliant mechanisms," in *21st Century Kinematics: The 2012 NSF Workshop*. Springer, 2013, pp. 189–216.

[9] ——, *Introduction to Compliant Mechanisms*. John Wiley Sons, Ltd, 2013, ch. 1, pp. 1–13. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118516485.ch1

[10] M. Schenk and S. D. Guest, "On zero stiffness," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 228, no. 10, pp. 1701–1714, 2014.

[11] L. Zentner and V. Böhm, "On the classification of compliant mechanisms," in *Proceedings of EUCOMES 08*, M. Ceccarelli, Ed. Springer Netherlands, 2009, pp. 431–438.

[12] D. Russo and A. Caputi, "How to classify compliant mechanisms," in *Design Tools and Methods in Industrial Engineering: Proceedings of the International Conference on Design Tools and Methods in Industrial Engineering, ADM 2019, September 9–10, 2019, Modena, Italy*. Springer, 2020, pp. 552–564.

[13] D. Farhadi Machekposhti, N. Tolou, and J. Herder, "A review on compliant joints and rigid-body constant velocity universal joints toward the design of compliant homokinetic couplings," *Journal of Mechanical Design*, vol. 137, no. 3, p. 032301, 2015.

[14] D. Hogervorst, "Classification and comparison of zero stiffness compliant rotary joints," Delft University of Technology, Literature report, 2022, part of the master's thesis *A Neutrally Stable Quasi-Compliant Spherical Joint With a Remote Center of Rotation*. Available at: http://resolver.tudelft.nl/uuid:121da43a-6321-4550-8b34-0452814eadb0.

[15] Q. Lu, C. Ortega, and O. Ma, "Passive gravity compensation mechanisms: technologies and applications," *Recent Patents on Engineering*, vol. 5, no. 1, pp. 32–44, 2011.

[16] G. Radaelli and J. Herder, "A monolithic compliant large-range gravity balancer," *Mechanism and Machine Theory*, vol. 102, 2015.

[17] B. L. Rijff, J. L. Herder, and G. Radaelli, "An energy approach to the design of single degree of freedom gravity balancers with compliant joints," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 54839, 2011, pp. 137–148.

[18] G. Radaelli and J. Herder, "Gravity balanced compliant shell mechanisms," *International Journal of Solids and Structures*, vol. 118, pp. 78–88, 2017.

[19] J. Rommers, G. Radaelli, and J. L. Herder, "A design tool for a single vertex compliant-facet origami mechanism including torsional hinge lines," *Journal of Mechanisms and Robotics*, vol. 9, no. 6, p. 061015, 2017.

[20] S. Abouheidari, G. Radaelli, and J. Herder, "Synthesis of nonlinear torque-angle profile using compliant helicoidal shell joint," Master's thesis, Delft University of Technology, 2024, available at: http://resolver.tudelft.nl/uuid:71500cd9-d2e3-4c02-8aca-63edad0638b2.

[21] A. Amoozandeh Nobaveh, G. Radaelli, and J. L. Herder, "A design tool for passive wrist support," in *Wearable Robotics: Challenges and Trends: Proceedings of the 5th International Symposium on Wearable Robotics, WeRob2020, and of WearRAcon Europe 2020, October 13–16, 2020*. Springer, 2022, pp. 13–17.

[22] M. Tschiersky, E. E. Hekman, D. M. Brouwer, and J. L. Herder, "Gravity balancing flexure springs for an assistive elbow orthosis," *IEEE Transactions on Medical Robotics and Bionics*, vol. 1, no. 3, pp. 177–188, 2019.

[23] G. Radaelli and J. L. Herder, "Isogeometric shape optimization for compliant mechanisms with prescribed load paths," in *International design engineering technical conferences and computers and information in engineering conference*, vol. 46360. American Society of Mechanical Engineers, 2014, p. V05AT08A046.

[24] ——, "Shape optimization and sensitivity of compliant beams for prescribed load-displacement response," *Mechanical Sciences*, vol. 7, no. 2, pp. 219–232, 2016.

[25] Z.-W. Yang and C.-C. Lan, "An adjustable gravity-balancing mechanism using planar extension and compression springs," *Mechanism and Machine Theory*, vol. 92, pp. 314–329, 2015.

[26] Z. Cheng, S. Foong, D. Sun, and U.-X. Tan, "Towards a multi-dof passive balancing mechanism for upper limbs," in *2015 IEEE International Conference on Rehabilitation Robotics (ICORR)*, 2015, pp. 508–513.

[27] Z. Shan, M. Endo, H. Nakamura, and S. Tanaka, *Irregular-Shaped Torsion Spring Design for Gravity Compensation in Linkage Systems: A Modified CPRBM Based Methodology*, 2023, pp. 269–278.

[28] S. Zhang and G. Chen, "Design of compliant bistable mechanism for rear trunk lid of cars," in *Intelligent Robotics and Applications: 4th International Conference, ICIRA 2011, Aachen, Germany, December 6-8, 2011, Proceedings, Part I 4*. Springer, 2011, pp. 291–299.

[29] J. Stroo, J. Herder, and A. Dunning, "Feasibility study of a balanced upper arm orthosis based on bending beams," Master's thesis, Delft University of Technol-

ogy, 2014, available at: http://resolver.tudelft.nl/uuid:86ee53cf-7fb4-4a9b-ac97-0aece6864af4.

[30] J. G. Kleinjan, A. G. Dunning, and J. L. Herder, "Design of a compact actuated compliant elbow joint," *International Journal of Structural Stability and Dynamics*, vol. 14, no. 08, p. 1440030, 2014.

[31] J. Geerts, G. Radaelli, and J. Herder, "Towards compliant and compact arm supports," Master's thesis, Delft University of Technology, 2017, available at: http://resolver.tudelft.nl/uuid:928c2ad9-cea1-4bd5-a5be-5a5571957225.

[32] B. G. Bijlsma, G. Radaelli, and J. L. Herder, "Design of a compact gravity equilibrator with an unlimited range of motion," *Journal of Mechanisms and Robotics*, vol. 9, p. 061003, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:118754010

[33] H.-C. Hsieh and C.-C. Lan, "A lightweight gravity-balanced exoskeleton for home rehabilitation of upper limbs," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2014, pp. 972–977.

[34] J. H. F. van Osch, F.C.T. Van der Helm, "Design of an adjustable gravity equilibrator using torsion bars," Master's thesis, Delft University of Technology, 2011, available at: http://resolver.tudelft.nl/uuid:6234f2a7-53a5-4b5b-8919-b3700e17da55.

[35] G. Radaelli, R. Buskermolen, R. Barents, and J. Herder, "Static balancing of an inverted pendulum with prestressed torsion bars," *Mechanism and Machine Theory*, vol. 108, pp. 14–26, 2017.

[36] M. Tschiersky, E. E. Hekman, J. L. Herder, and D. M. Brouwer, "Gravity balancing flexure spring mechanisms for shoulder support in assistive orthoses," *IEEE Transactions on Medical Robotics and Bionics*, vol. 4, no. 2, pp. 448–459, 2022.

[37] S. v. d. Kemp and J. Herder, "Design of a compact wearable arm support utilizing shape optimized shell mechanisms: To be worn underneath the surgical gown," 2018, available at: http://resolver.tudelft.nl/uuid:3664c6fd-da05-48b5-a503-a9e373d912a0.

[38] G. Chen and S. Zhang, "Fully-compliant statically-balanced mechanisms without prestressing assembly: Concepts and case studies," *Mechanical Sciences*, vol. 2, 2011.

[39] A. Dunning, N. Tolou, and J. Herder, "A compact low-stiffness six degrees of freedom compliant precision stage," *Precision Engineering*, vol. 37, p. 380–388, 2013.

[40] C.-H. Liu, T.-L. Chen, F.-M. Chung, and M.-C. Hsu, *A Topology Optimization Method to Synthesize Geometrically Nonlinear Compliant Constant-Force Mechanisms*, 2023, pp. 259–268.

[41] N. Tolou, P. Pluimers, B. D. Jensen, S. Magleby, L. Howell, and J. L. Herder, "Constant force micro mechanism out of carbon nanotube forest," in *Proceedings of the 12th euspen International Conference Stockholm–Jun*, 2011.

[42] Q. Xu, "Design of a large-stroke bistable mechanism for the application in constant-force micropositioning stage," *Journal of Mechanisms and Robotics*, vol. 9, no. 1, p. 011006, 2017.

[43] P. R. Kuppens, J. L. Herder, and N. Tolou, "Permanent stiffness reduction by thermal oxidation of silicon," *Journal of Microelectromechanical Systems*, vol. 28, no. 5, pp. 900–909, 2019.

[44] K. A. Tolman, E. G. Merriam, and L. L. Howell, "Compliant constant-force linear-motion mechanism," *Mechanism and Machine Theory*, vol. 106, pp. 68–79, 2016.

[45] P. Wang, "Design of a flexure-based constant-force xy precision positioning stage," *Mechanism and Machine Theory*, vol. 108, 2017.

[46] C. Boyle, L. Howell, S. Magleby, and M. Evans, "Dynamic modeling of compliant constant-force compression mechanisms," *Mechanism and Machine Theory*, vol. 38, pp. 1469–1487, 2003.

[47] J. Meaders and C. Mattson, "Robust design optimization of compliant constant force contacts with simulated pin joints," in *48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. Aerospace Research Central, 2007, pp. 1–13.

[48] N. Le Chau, H. Giang Le, and T.-P. Dao, "A gravity balance mechanism using compliant mechanism," in *Computational Intelligence Methods for Green Technology and Sustainable Development*. Springer International Publishing, 2021, pp. 431–439.

[49] J. L. Prat, "Control interfaces to actively support the arm function of men with duchenne muscular dystrophy," Ph.D. dissertation, University of Twente, 2016.

[50] J. Rommers, V. van der Wijk, and J. L. Herder, "A new type of spherical flexure joint based on tetrahedron elements," *Precision Engineering*, vol. 71, p. 130–140, 2021.

B

Top 5 Optimization Results

This appendix presents the relevant data of the five optimization results with the lowest objective function d_{avg} for all five seperate optimizations. Unlike the Results section of the main paper, this appendix includes the self-intersecting results that were previously excluded. Each section corresponds to one of the optimization conditions and features a table displaying the data of its top 5 simulated pendulum balancers. This data includes the independent geometric parameters, the number of iterations N_{iter} needed in the optimization run, the objective function d_{avg} and the deformation resemblance η_{dr} of the simulated pendulum balancers. The table will also indicate whether the shape of the optimization result is that of a counterclockwise (CCW) spiral and if it self-intersects in undeformed state.

Finally, it should be noted that equation 61 in the main paper contained a slight error back when the optimization runs were executed. The tables in this appendix reflect the original d_{avg} and η_{dr} values of the optimization results, while the values in the Results section of the main paper correspond to the corrected recalculations. As a result, minor differences may be observed between the data presented here and in the main paper.

B.1. Ten-tetrahedra optimization

	Result #1			Result #2			Result #3			Result #4			Result #5		
i	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)
1	46.18	-	1369.84	16.30	-	1290.61	41.61	-	1428.24	17.52	-	1507.09	60.00	-	1685.07
2	59.50	-71.99	1369.84	27.74	-39.13	1290.61	59.12	18.29	1428.24	49.80	16.88	1507.09	60.00	5.78	1685.07
3	51.26	-63.26	1369.84	56.64	7.83	1290.61	54.78	35.00	1428.24	53.66	31.26	1507.09	51.83	33.46	1685.07
4	47.89	-4.19	1369.84	22.24	-22.22	1290.61	51.72	35.00	1428.24	57.56	-48.48	1507.09	59.31	-6.03	1685.07
5	41.45	-47.44	1369.84	43.66	-44.36	1290.61	33.19	-2.64	1428.24	16.87	-41.85	1507.09	29.96	-20.29	1685.07
6	17.33	29.17	1369.84	43.02	-22.58	1290.61	60.00	-75.58	1428.24	59.99	33.42	1507.09	55.94	24.27	1685.07
7	42.06	-68.77	1369.84	31.80	-38.04	1290.61	59.99	-3.66	1428.24	23.97	-42.56	1507.09	43.17	-3.36	1685.07
8	5.00	-90.00	1369.84	15.71	-40.34	1290.61	40.88	-88.63	1428.24	54.05	-23.04	1507.09	60.00	-20.87	1685.07
9	34.39	-56.54	1369.84	31.40	-83.87	1290.61	53.30	-66.26	1428.24	49.34	-90.00	1507.09	60.00	-45.82	1685.07
10	29.08	-6.16	1369.84	24.23	-2.07	1290.61	5.00	57.01	1428.24	20.70	-24.60	1507.09	60.00	-54.07	1685.07
N_iter		436			201			1000			538			420	
d_avg		1.29 mm			1.31 mm			1.49 mm			1.51 mm			1.56 mm	
η_dr		95.7%			95.7%			95.1%			95.0%			94.8%	
CCW Spiral		No			Yes			No			Yes			Yes	
Intersecting		Yes			No			No			No			Yes	

Table B.1: Top 5 results of the ten-tetrahedra optimization

B.2. Ten-tetrahedra-NSW optimization

	Result #1			Result #2			Result #3			Result #4			Result #5		
i	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)
1	14.76	-	1116.23	59.99	-	1640.97	33.84	-	1063.24	60.00	-	1477.625	60.00	-	1650.94
2	23.83	-36.70	1116.23	37.63	-71.15	1640.97	24.30	-7.19	1063.24	41.56	-73.77	1477.625	45.96	-89.94	1650.94
3	30.01	-63.14	1116.23	37.17	34.70	1640.97	28.95	-10.39	1063.24	26.75	-16.54	1477.625	60.00	-69.99	1650.94
4	37.56	-53.14	1116.23	59.98	-27.17	1640.97	19.82	-46.91	1063.24	35.84	-16.01	1477.625	59.96	-19.87	1650.94
5	43.37	-25.45	1116.23	59.87	31.80	1640.97	45.29	-29.06	1063.24	59.71	-17.55	1477.625	59.73	32.49	1650.94
6	48.49	-68.31	1116.23	40.74	-69.91	1640.97	35.44	-68.09	1063.24	19.46	-27.66	1477.625	60.00	-54.71	1650.94
7	13.77	-0.41	1116.23	56.63	34.88	1640.97	7.91	4.14	1063.24	24.62	-37.49	1477.625	50.81	29.27	1650.94
8	35.93	-52.46	1116.23	18.85	34.87	1640.97	12.62	-22.85	1063.24	44.88	13.89	1477.625	40.34	35.00	1650.94
9	16.71	-71.77	1116.23	48.63	34.90	1640.97	9.46	-37.30	1063.24	49.90	34.28	1477.625	60.00	35.00	1650.94
10	32.33	-51.83	1116.23	46.12	14.96	1640.97	37.62	-79.12	1063.24	53.90	35.00	1477.625	11.68	-9.76	1650.94
N_iter		387			550			315			203			342	
d_avg		1.01 mm			1.01 mm			1.02 mm			1.05 mm			1.07 mm	
η_dr		96.7%			96.7%			96.6%			96.5%			96.5%	
CCW Spiral		Yes			No			Yes			No			No	
Intersecting		Yes			Yes			No			No			No	

Table B.2: Top 5 results of the ten-tetrahedra-NSW optimization

B.3. Eight-tetrahedra optimization

	Result #1			Result #2			Result #3			Result #4			Result #5		
i	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)
1	21.67	-	1173.48	34.61	-	1454.02	56.50	-	1505.91	52.86	-	1514.06	60.00	-	1616.91
2	35.42	-9.36	1173.48	33.66	-31.66	1454.02	58.90	-16.49	1505.91	59.61	-40.06	1514.06	46.53	32.97	1616.91
3	40.72	0.39	1173.48	54.38	16.57	1454.02	60.00	-59.80	1505.91	56.31	-40.47	1514.06	56.89	-66.51	1616.91
4	43.16	-54.16	1173.48	59.91	-7.84	1454.02	59.99	-27.87	1505.91	60.00	16.96	1514.06	59.63	34.96	1616.91
5	51.03	-55.14	1173.48	56.11	-35.69	1454.02	59.72	12.60	1505.91	60.00	-37.99	1514.06	60.00	-18.44	1616.91
6	24.32	-5.64	1173.48	59.38	-19.57	1454.02	60.00	-58.93	1505.91	59.98	-41.33	1514.06	60.00	-16.75	1616.91
7	27.49	-89.67	1173.48	39.17	-76.59	1454.02	49.02	-57.81	1505.91	37.53	-88.66	1514.06	60.00	-46.74	1616.91
8	41.88	-64.76	1173.48	48.47	-71.01	1454.02	51.36	-89.97	1505.91	32.85	-26.44	1514.06	60.00	-71.78	1616.91
N_iter		261			819			824			482			228	
d_avg		1.24 mm			1.29 mm			1.40 mm			1.42 mm			1.50 mm	
η_dr		95.9%			95.7%			95.4%			95.3%			92.0%	
CCW Spiral		Yes			Yes			Yes			Yes			Yes	
Intersecting		No			No			Yes			No			Yes	

Table B.3: Top 5 results of the eight-tetrahedra optimization

B.4. Twelve-tetrahedra optimization

	Result #1			Result #2			Result #3			Result #4			Result #5		
i	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)
1	12.60	-	1104.33	48.65	-	1380.13	18.11	-	1451.10	60.00	-	1644.96	26.90	-	1057.00
2	18.17	0.06	1104.33	20.23	13.39	1380.13	14.68	-21.12	1451.10	59.80	-3.02	1644.96	8.79	-21.33	1057.00
3	15.23	-6.76	1104.33	17.76	-46.17	1380.13	49.96	-45.38	1451.10	34.75	-78.85	1644.96	33.47	-49.38	1057.00
4	30.88	-26.59	1104.33	44.00	-10.09	1380.13	56.04	-70.66	1451.10	59.59	-89.74	1644.96	23.21	-55.92	1057.00
5	53.92	-89.37	1104.33	5.00	15.28	1380.13	60.00	-80.64	1451.10	43.49	8.96	1644.96	5.06	16.76	1057.00
6	17.24	-29.94	1104.33	7.57	-25.44	1380.13	41.35	-6.15	1451.10	54.10	-48.43	1644.96	8.36	-25.11	1057.00
7	7.18	-34.97	1104.33	30.88	3.39	1380.13	58.26	-43.49	1451.10	33.62	26.08	1644.96	42.00	-40.92	1057.00
8	15.34	-11.14	1104.33	59.91	-62.02	1380.13	26.69	-31.45	1451.10	33.72	-32.78	1644.96	25.73	-25.83	1057.00
9	36.96	-1.04	1104.33	17.71	-1.54	1380.13	24.27	21.49	1451.10	47.66	-44.84	1644.96	32.06	-88.27	1057.00
10	41.61	-71.67	1104.33	18.20	-55.28	1380.13	37.82	-70.63	1451.10	22.71	12.15	1644.96	26.68	35.00	1057.00
11	17.29	-55.15	1104.33	20.43	-9.00	1380.13	17.72	-23.54	1451.10	12.59	-62.91	1644.96	7.37	-70.07	1057.00
12	35.24	-74.30	1104.33	55.09	-85.95	1380.13	52.24	-90.00	1451.10	55.61	-88.84	1644.96	39.23	-84.09	1057.00
N_iter		433			232			354			439			237	
d_avg		1.26 mm			1.37 mm			1.45 mm			1.54 mm			1.61 mm	
η_dr		95.8%			95.5%			95.2%			94.9%			94.7%	
CCW Spiral		Yes			Yes			No			No			Yes	
Intersecting		Yes			No			Yes			Yes			Yes	

Table B.4: Top 5 results of the twelve-tetrahedra optimization

B.5. Ten-tetrahedra-90x90 optimization

	Result #1			Result #2			Result #3			Result #4			Result #5		
i	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)	α (deg)	γ (deg)	t_min (µm)
1	60.00	-	1512.45	60.00	-	1630.33	60.00	-	1634.48	60.00	-	1583.53	28.98	-	1341.03
2	59.64	35.00	1512.45	58.78	-50.69	1630.33	59.96	-7.88	1634.48	59.87	13.15	1583.53	28.47	-37.09	1341.03
3	60.00	34.32	1512.45	51.31	33.59	1630.33	52.95	-28.97	1634.48	56.06	-29.95	1583.53	41.61	-63.82	1341.03
4	58.53	-20.51	1512.45	59.88	-21.98	1630.33	59.62	27.73	1634.48	47.78	-66.04	1583.53	36.11	22.19	1341.03
5	37.27	-88.57	1512.45	59.58	-2.32	1630.33	59.91	-26.47	1634.48	60.00	25.09	1583.53	59.97	-40.39	1341.03
6	31.56	35.00	1512.45	59.51	-11.59	1630.33	59.91	35.00	1634.48	59.37	26.82	1583.53	34.77	-11.63	1341.03
7	58.09	35.00	1512.45	56.88	26.32	1630.33	59.81	-28.11	1634.48	56.93	-53.14	1583.53	52.18	-41.53	1341.03
8	46.87	27.55	1512.45	60.00	-47.11	1630.33	51.07	-19.29	1634.48	42.10	17.59	1583.53	59.79	-26.66	1341.03
9	25.18	35.00	1512.45	59.92	-17.84	1630.33	57.86	-5.92	1634.48	60.00	-49.90	1583.53	19.26	31.62	1341.03
10	47.26	-8.37	1512.45	60.00	-55.19	1630.33	60.00	-72.33	1634.48	60.00	-40.42	1583.53	48.50	31.95	1341.03
N_iter		284			347			679			569			295	
d_avg		10.78 mm			10.97 mm			10.99 mm			11.01 mm			11.02 mm	
η_dr		92.0%			91.8%			91.8%			91.8%			91.8%	
CCW Spiral		No			Yes			Yes			Yes			No	
Intersecting		No			Yes			Yes			Yes			Yes	

Table B.5: Top 5 results of the ten-tetrahedra-90x90 optimization

C

Global minimum analysis

This appendix showcases the lack of resemblance among the 50 optimization run results of the eight-tetrahedra optimization, suggesting that a global minimum is not likely reached. The eight-tetrahedra optimization is chosen for this analysis due to its relatively small number of variables, maximizing the likelihood of obtaining similar results.

Table C.1 lists the 50 optimization results. While all d_{avg} values in the table are unique, the objective functions of number 13, 20, 40 and 47 are fairly close to 14, 21, 41 and 48 respectively. The t_{min} values of these optimization results are listed in Table C.2, allowing for comparison with the results that have similar objective functions. The table reveals that all four comparisons exhibit significant differences in t_{min} values, indicating that they do not correspond to similar local minima. Consequently, it is unlikely that Number 1 represents the global minimum.

Number	d_{avg} (mm)	Number	d_{avg} (mm)
1	1.24	26	3.13
2	1.29	27	3.16
3	1.40	28	3.20
4	1.42	29	3.31
5	1.50	30	3.40
6	1.55	31	3.62
7	1.56	32	3.70
8	1.61	33	3.77
9	1.66	34	3.97
10	1.67	35	4.14
11	1.77	36	4.22
12	1.83	37	4.40
13	**1.94**	38	4.41
14	**1.95**	39	4.53
15	1.97	**40**	**4.64**
16	2.00	**41**	**4.65**
17	2.10	42	4.78
18	2.37	43	4.81
19	2.49	44	5.19
20	**2.60**	45	5.27
21	**2.61**	46	5.34
22	2.76	**47**	**5.41**
23	2.91	**48**	**5.42**
24	2.94	49	5.57
25	3.10	50	6.45

Table C.1: The results of the eight-tetrahedra optimization runs.

Number	t_{min} (μm)
13	1388.03
14	1550.49
20	1539.81
21	1392.33
40	1199.38
41	1204.19
47	1033.85
48	1173.60

Table C.2: The t_{min} values of the results with similar objective functions.

D

Sensitivity Analysis

This appendix presents a sensitivity analysis focused on the Young's modulus and the torsional constant of the simulated joint.
The results from the prototype's validation test suggest that the prototype is generally too compliant, as most equilibria were found for $(L_w)_e$ < 250 mm. A likely explanation for this discrepancy is a lower effective Young's modulus of the material than the advertised tensile modulus of 1700 MPa. Another possible factor could be a difference between the simulated torsional constants and the true torsional constants, as equation 21 in the main paper assumes the flexure's cross-section to be a spherical segment instead of a trapezoid. To investigate these two hypotheses, two sensitivity analyses are performed using the conditions and P_{goal} points from the TetraFEM validation test as described in subsection III-D.

D.1. Young's modulus

The deformation resemblance η_{dr} between the simulated joint and the prototype is calculated for 10 different Young's modulus values between 0.8·E and 1.2·E. The result of this sensitivity analysis is shown in figure D.1.

The figure shows a significant increase in deformation resemblance around E = 1.52 GPa, reinforcing the presumption of a lower effective Young's modulus of the prototype's material. Figure D.2 shows the deflected P_{EE} points of the simulated joint used in the TetraFEM validation, similar to Figure 29 of the main paper, but with a Young's modulus of 1520 MPa instead of 1700 MPa. The deformation resemblance here is 92.4% (instead of 89.6%) and this improvement is visible in the figure as well.

D.2. Torsional constant

The deformation resemblance η_{dr} between the simulated joint and the prototype is calculated for 10 different multipliers of the torsional constant (J_{mult}) with values between 0.5 and 1.5. The result of this sensitivity analysis is shown in figure D.3.

The figure shows a (almost) constant deformation resemblance, suggesting that the torsional constant of a single flexure has negligible impact. This is a logical conclusion as the torsional stiffness of a flexure in a tetrahedron element is mainly gained from the other two flexures.

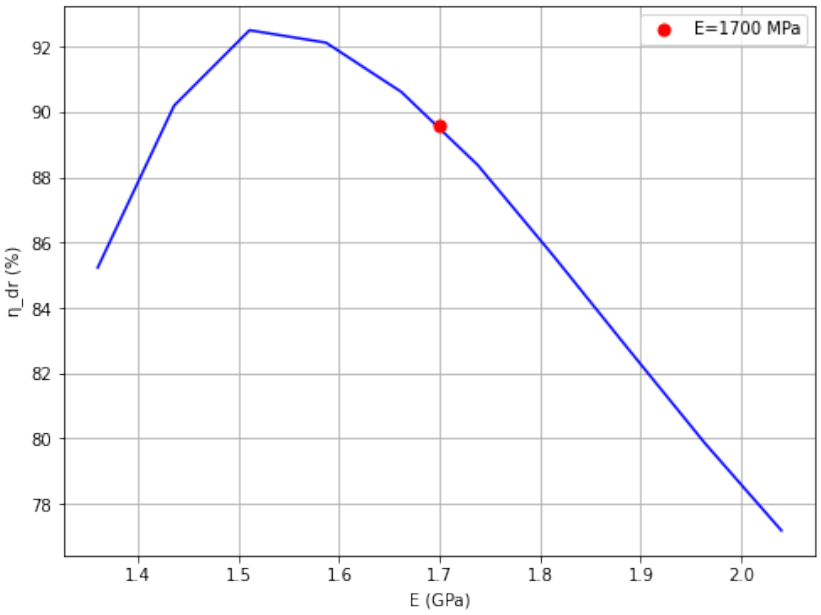

Figure D.1: E vs η_{dr} sensitivity analysis.

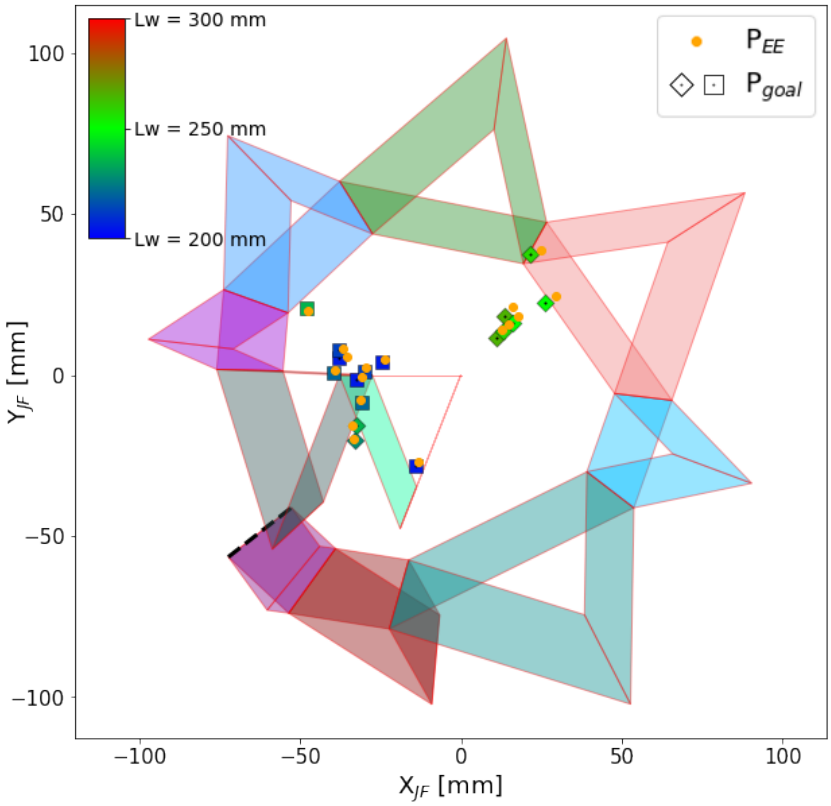

Figure D.2: The TetraFEM validation result for E=1.52GPa.

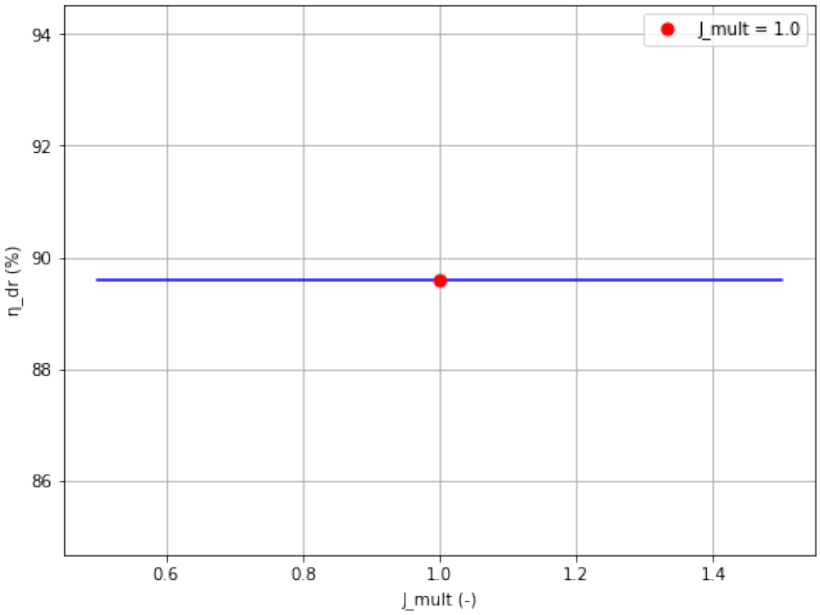

Figure D.3: J_{mult} vs η_{dr} sensitivity analysis

E

Calculations Flexure AB

The calculations related to the AB and BC flexures are largely left out of the main paper to prevent redundancy and to keep the Method section as concise as possible. Since flexures AB and AC are identical in shape and size, differing only in orientation, this appendix focuses exclusively on flexure AB. Here, the relevant equations for calculating the geometry, self-weight properties and compliance matrices of flexure AB are listed and briefly explained.

E.1. Geometry of flexure AB

The geometry of the AB flexure is fairly similar to that of the AC flexure, rotated counterclockwise by ϕ (see Figure E.1). The main differences are in the length of the flexure and AC is symmetric in two planes, while the AB flexure is only symmetric in one plane.

Defining orientation of frame AB_a:
Angle ϕ can be calculated using the formula below:

$$\phi = \arctan\left(\frac{\tan(\beta)}{\sin(\alpha/2)}\right)$$

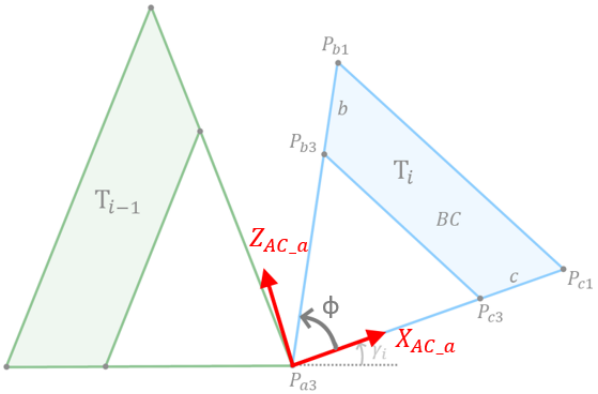

Figure E.1: A view perpendicular to edge a of T_i, showing the definition of parameter ϕ.

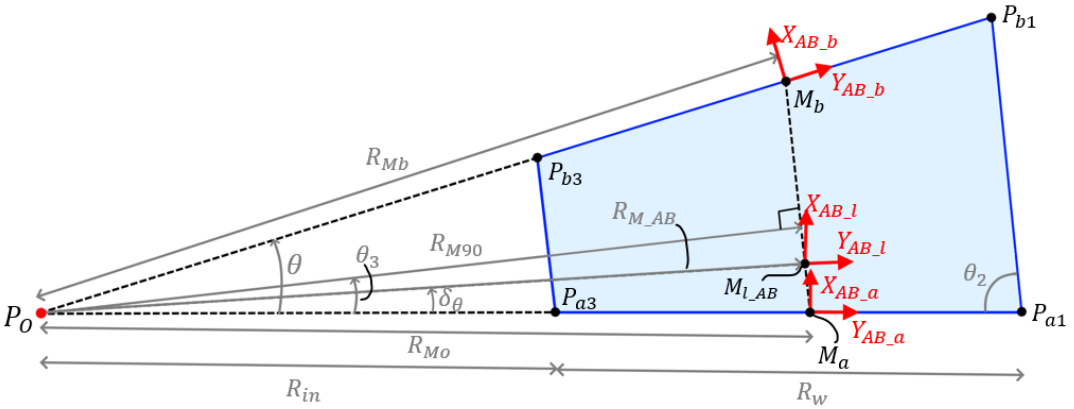

Figure E.2: A view perpendicular to Z_{AB_a}, showing the geometry and relevant coordinate systems of flexure AB.

The rotation matrix defining frame AB_a can be determined with angle ϕ:

$$^{AB\_a}\boldsymbol{R}_{AC\_a} = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix}$$

Defining orientation of frame AB_l:
AB_l is the local coordinate system of flexure AB and can be related to AB_a with the rotation matrix below:

$$^{AB\_l}\boldsymbol{R}_{AB\_a} = \begin{bmatrix} \cos(\delta_\theta) & -\sin(\delta_\theta) & 0 \\ \sin(\delta_\theta) & \cos(\delta_\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

Defining orientation of frame AB_b:
With frame AB_a and θ (see Figure E.2), the rotation matrix defining frame AB_b can be determined:

$$^{AB\_b}\boldsymbol{R}_{AB\_a} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where θ is the angle between the position vectors $\boldsymbol{r}_{P_O/P\_a3}$ and $\boldsymbol{r}_{P_O/P\_b3}$. These two position vectors are defined in frame AC_a as:

$$^{AC\_a}\boldsymbol{r}_{P_O/P\_a3} = \begin{bmatrix} 0 \\ R_{in} \\ 0 \end{bmatrix},$$

$$^{AC\_a}\boldsymbol{r}_{P_O/P\_b3} = \begin{bmatrix} 0 \\ R_{in\_m}\cos(\alpha) \\ R_{in\_m}\tan(\beta) \end{bmatrix}.$$

With these two position vectors, θ can be calculated using the dot product formula:

$$\theta = \arccos\left(\frac{{}^{AC\_a}\boldsymbol{r}_{P_O/P\_a3} \cdot {}^{AC\_a}\boldsymbol{r}_{P_O/P\_b3}}{\|{}^{AC\_a}\boldsymbol{r}_{P_O/P\_a3}\|\|{}^{AC\_a}\boldsymbol{r}_{P_O/P\_b3}\|}\right)$$

Determine position of point M_b:
Parameter R_{Mb} needs to be calculated to determine the position of M_b (see Figure E.2):

$$R_{Mb} = \|\boldsymbol{r}_{P_O/P\_b1} - \boldsymbol{r}_{P\_b1/P\_b3} \cdot \frac{t_{max} + 2t_{min}}{3(t_{max} + t_{min})}\|,$$

where

$${}^{AC\_a}\boldsymbol{r}_{P_O/P\_b1} = \begin{bmatrix} 0 \\ (R_{in\_m} + R_{w\_m})\cos(\alpha) \\ (R_{in\_m} + R_{w\_m})\tan(\beta) \end{bmatrix}$$

$${}^{AC\_a}\boldsymbol{r}_{P\_b1/P\_b3} = {}^{AC\_a}\boldsymbol{r}_{P_O/P\_b3} - {}^{AC\_a}\boldsymbol{r}_{P_O/P\_b1}$$

Finally, the position vector of M_b from point P_O is:

$${}^{AB\_b}\boldsymbol{r}_{P_O/M\_b} = \begin{bmatrix} 0 \\ R_{Mb} \\ 0 \end{bmatrix}.$$

Determine position of point M_{l_AB}: Parameter R_{M_AB} needs to be calculated to determine the position of local point M_{l_AB} (see Figure E.2):

$$R_{M\_AB} = \frac{R_{M90}}{\cos(\theta_3 - \delta_\theta)},$$

where

$$R_{M90} = \cos(\theta_3) R_{Mo}$$
$$\theta_3 = \frac{\pi}{2} - \theta_2$$

where θ_2 (see Figure E.2) is the angle between position vectors $\boldsymbol{r}_{P\_a3/P_O}$ and $\boldsymbol{r}_{P\_a3/P\_b3}$, which can be defined using the equations below:

$$\boldsymbol{r}_{P\_a3/P_O} = -\boldsymbol{r}_{P_O/P\_a3}$$
$$\boldsymbol{r}_{P\_a3/P\_b3} = \boldsymbol{r}_{P_O/P\_b3} - \boldsymbol{r}_{P_O/P\_a3}$$

With these two position vectors, θ_2 can be calculated using the dot product formula:

$$\theta_2 = \arccos\left(\frac{{}^{AC\_a}\boldsymbol{r}_{P\_a3/P_O} \cdot {}^{AC\_a}\boldsymbol{r}_{P\_a3/P\_b3}}{\|{}^{AC\_a}\boldsymbol{r}_{P\_a3/P_O}\|\|{}^{AC\_a}\boldsymbol{r}_{P\_a3/P\_b3}\|}\right)$$

Finally, the position vector of M_{l_AB} from point P_O is:

$${}^{AB\_l}\boldsymbol{r}_{P_O/M\_l\_AB} = \begin{bmatrix} 0 \\ R_{M\_AB} \\ 0 \end{bmatrix}.$$

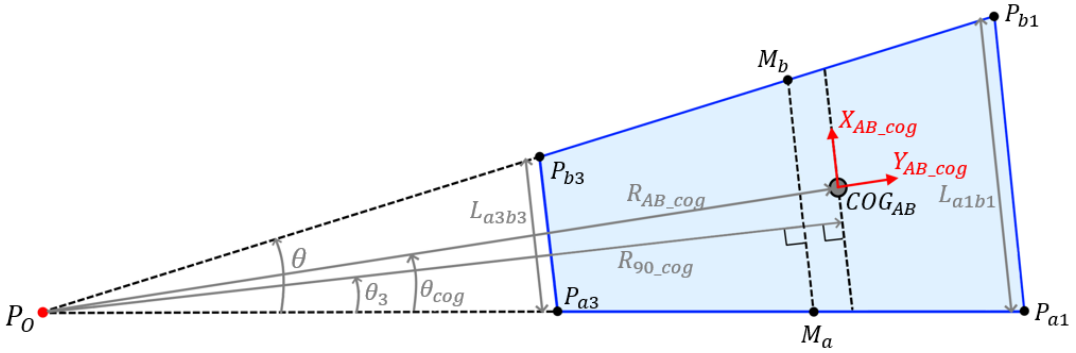

Figure E.3: A view perpendicular to Z_{AB_a}, highlighting the parameters and coordinate system related to the self-weight of flexure AB.

E.2. Mass parameters of flexure AB

Flexure AB has the shape of a chopped-off pyramid, similar to flexure AC. The calculations for flexure AB's mas and center of gravity (COG) therefore show many similarities to those in subsection II-B.

Calculating mass m_{AB}:
The volume of the large pyramid V_{1_AB} and the small chopped-off part of the pyramid V_{3_AB} of flexure AB can be calculated with the equations below:

$$V_{1\_AB} = \frac{t_{max} L_{a1b1} \cos(\theta_3)(R_{in} + R_w)}{3}$$

$$V_{3\_AB} = \frac{t_{min} L_{a3b3} \cos(\theta_3) R_{in}}{3}$$

where

$$L_{a3b3} = \|\boldsymbol{r}_{P_O/P\_b3} - \boldsymbol{r}_{P_O/P\_a3}\|$$
$$L_{a1b1} = \|\boldsymbol{r}_{P_O/P\_b1} - \boldsymbol{r}_{P_O/P\_a1}\|$$
$${}^{AB_a}\boldsymbol{r}_{P_O/P\_a1} = \begin{bmatrix} 0 \\ R_{in} + R_w \\ 0 \end{bmatrix}.$$

The mass of flexure AB can be calculated with these two volumes and the density of the material:

$$m_{AB} = \rho(V_{1\_AB} - V_{3\_AB})$$

Determine position of point COG_{AB}:
The orientation of frame AB_{cog} and parameter R_{AB_cog} (see Figure E.3) need to be calculated to determine the position of COG_{AB}. The rotation matrix relating frame AB_{cog} to AB_a equals:

$${}^{AB\_cog}\boldsymbol{R}_{AB\_a} = \begin{bmatrix} \cos(\theta_{cog}) & -\sin(\theta_{cog}) & 0 \\ \sin(\theta_{cog}) & \cos(\theta_{cog}) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where it is assumed that $\theta_{cog} \approx \theta/2$. Parameter R_{AB_cog} is calculated using the formula below:

$$R_{AB\_cog} = \frac{R_{90\_cog}}{\cos(\theta_{cog} - \theta_3)},$$

where R_{90_cog} is the distance between P_O and a line that intersects COG_{AB} and is parallel to vector $\boldsymbol{r}_{P\_a3/P\_b3}$ (see Figure E.3). R_{90_cog} is calculated with the equation below:

$$R_{90\_cog} = \frac{3(V_{1\_AB}(R_{in} + R_w)\cos(\theta_3) - V_{3\_AB}R_{in}\cos(\theta_3))}{4(V_{1\_AB} - V_{3\_AB})}$$

Finally, the position vector of COG_{AB} from point P_O is:

$$^{AB\_cog}\boldsymbol{r}_{P_O/COG\_AB} = \begin{bmatrix} 0 \\ R_{AB\_cog} \\ 0 \end{bmatrix}.$$

E.3. Flexure Compliance Analysis of flexure AB

The flexure compliance analysis for flexure AB is very similar to that of flexure AC, which is described in section II-C1 of the main paper. The same general formulas apply, only with some different parameters, which are calculated in the two previous sections of this appendix. The formulas below are the equations from section II-C1, but rewritten for flexure AB, resulting in its compliance matrix.

$$^{AB\_l}\boldsymbol{K} = \mathrm{Diag}[EA_x, b_yGA_x, b_zGA_x, GJ, EI_y, EI_z]$$

$$R_{w\_M\_AB} = \frac{R_w}{\cos(\theta_3 - \delta_\theta)}$$

$$A_x = \frac{t_{max} + t_{min}}{2} \cdot R_{w\_M\_AB}$$

$$I_y = \frac{R_{w\_M\_AB}(t_{min} + t_{max})(t_{min}^2 + t_{max}^2)}{48}$$

$$I_z = \frac{R_{w\_M\_AB}^3(t_{min}^2 + 4t_{max}t_{min} + t_{max}^2)}{36(t_{max} + t_{min})}$$

$$J = \frac{2}{3}\sin^3(\eta_{AB})(R_{out\_M\_AB}^4 - R_{in\_M\_AB}^4) - 16\sin^4(\eta_{AB})(V_{L\_AB}R_{out\_M\_AB}^4 + V_{S\_AB}R_{in\_M\_AB}^4)$$

$$V_{L\_AB} = 0.10504 - 0.2\sin(\eta_{AB}) + 0.3392\sin^2(\eta_{AB}) - 0.53968\sin^3(\eta_{AB}) + 0.82448\sin^4(\eta_{AB}),$$

$$V_{S\_AB} = 0.10504 + 0.2\sin(\eta_{AB}) + 0.3392\sin^2(\eta_{AB}) + 0.53968\sin^3(\eta_{AB}) + 0.82448\sin^4(\eta_{AB}).$$

$$R_{in\_M\_AB} = \frac{R_{in}}{\cos(\theta_3 - \delta_\theta)}$$

$$R_{out\_M\_AB} = R_{in\_M\_AB} + R_{w\_M\_AB}$$

$$\eta_{AB} = \arctan\left(\frac{t_{min}}{2R_{in\_M\_AB}}\right)$$

$$^{AB\_b}\boldsymbol{C} = \int_l {}^{AB\_l}\boldsymbol{T}_{AB\_b}^T \cdot {}^{AB\_l}\boldsymbol{K}^{-1} \cdot {}^{AB\_l}\boldsymbol{T}_{AB\_b} \cdot dl.$$

$$^{AB\_l}\boldsymbol{T}_{AB\_b} = \left[\begin{array}{c|c} {}^{AB\_l}\boldsymbol{R}_{AB\_b} & \boldsymbol{0} \\ \hline {}^{AB\_l}\widetilde{\boldsymbol{r}}_{M\_b/M\_l\_AB} \cdot {}^{AB\_l}\boldsymbol{R}_{AB\_b} & {}^{AB\_l}\boldsymbol{R}_{AB\_b} \end{array}\right]$$

$$^{AB\_l}\boldsymbol{R}_{AB\_b} = {}^{AB\_l}\boldsymbol{R}_{AB\_a} \cdot {}^{AB\_a}\boldsymbol{R}_{AB\_b},$$

$$^{AB\_l}\boldsymbol{R}_{AB\_a} = \begin{bmatrix} \cos(\delta_\theta) & -\sin(\delta_\theta) & 0 \\ \sin(\delta_\theta) & \cos(\delta_\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$^{AB\_a}\boldsymbol{R}_{AB\_b} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$^{AB\_l}\widetilde{\boldsymbol{r}}_{M\_b/M\_l\_AB} = \begin{bmatrix} 0 & -\boldsymbol{v}_{AB}[2] & \boldsymbol{v}_{AB}[1] \\ \boldsymbol{v}_{AB}[2] & 0 & -\boldsymbol{v}_{AB}[0] \\ -\boldsymbol{v}_{AB}[1] & \boldsymbol{v}_{AB}[0] & 0 \end{bmatrix}$$

$$\boldsymbol{v}_{AB} = ^{AB\_l}\boldsymbol{r}_{M\_b/M\_l\_AB}$$

$$^{AB\_l}\boldsymbol{r}_{M\_b/M\_l\_AB} = ^{AB\_l}\boldsymbol{r}_{P_O/M\_l\_AB} - ^{AB\_l}\boldsymbol{r}_{P_O/M\_b}$$

$$dl = R_{M\_AB} \cdot d\delta_\theta$$

$$R_{M\_AB} = R_{Mo}\frac{\cos(\theta_3)}{\cos(\theta_3 - \delta_\theta)}$$

$$f(\delta_\theta) = ^{AB\_l}\boldsymbol{T}^T_{AB\_b} \cdot ^{AB\_l}\boldsymbol{K}^{-1} \cdot ^{AB\_l}\boldsymbol{T}_{AB\_b} \cdot R_{M\_AB}$$

$$^{AB\_b}\boldsymbol{C} \approx \frac{\theta}{N}\Big(\frac{f(0)}{2} + \sum_{K=1}^{N-1}\Big(f\Big(\frac{K\cdot\theta}{N}\Big) + \frac{f(\theta)}{2}\Big)\Big)$$

F

Processing Data of the Experiment

This appendix elaborates on how the positions and orientations of the pendulum in equilibria were determined. Although it is not the focus of this appendix, it may be noted that the data on the limits in range of motion due to self-intersection (illustrated in Figure 28 of the main paper) is captured and processed in the same way.
As described in subsection II-E, two phones are fixed to the experimental setup to capture the position and orientation of the pendulum rod. The phones were left recording during the experiment and an assistant would swing her hands in front of both cameras if an equilibrium was found. As 61 equilibria were identified and captured in about two hours, this method was quite efficient, but requires some processing to obtain useful data. Processing the data can be divided in four steps:

- Find and screenshot equilibria
- Marking pixels
- Obtaining the translation and orientation of the pendulum rod
- Calculating the position of the P_{EE}-indicator

F.1. Find and screenshot equilibria

The first step is achieved by playing the recorded videos on two separate screens, one for the camera facing a steel plate in the background and one for the camera facing a transparent PMMA plate in the background. The videos are paused just before the assistant's hand would cover the cameras lens. After that, a screenshot of both screens is taken, resulting in an image such as Figure F.1. This single image shows two perpendicular angles of the position and orientation of the balanced pendulum.

F.2. Marking pixels

The second step involves importing the screenshot in paint.net and marking three pixels along the central axis of both pendulum rods. The location of these pixels will be used in the next step to calculated the orientation of the pendulum. The lowest of the three marked pixels should be on the P_O-indicator, such that it can be used to quantify the parasitic translation of the pendulum as well.
In the first screenshot of a video, the position of the fixed reference point and the direction of gravity need to be calibrated. Therefore, the top and bottom of the fixed reference point are marked, as well as three points along one of the gravity indicators (see Figure F.2). In the next screenshots these calibration markings are not required as long as the cameras remain fixed in the setup.

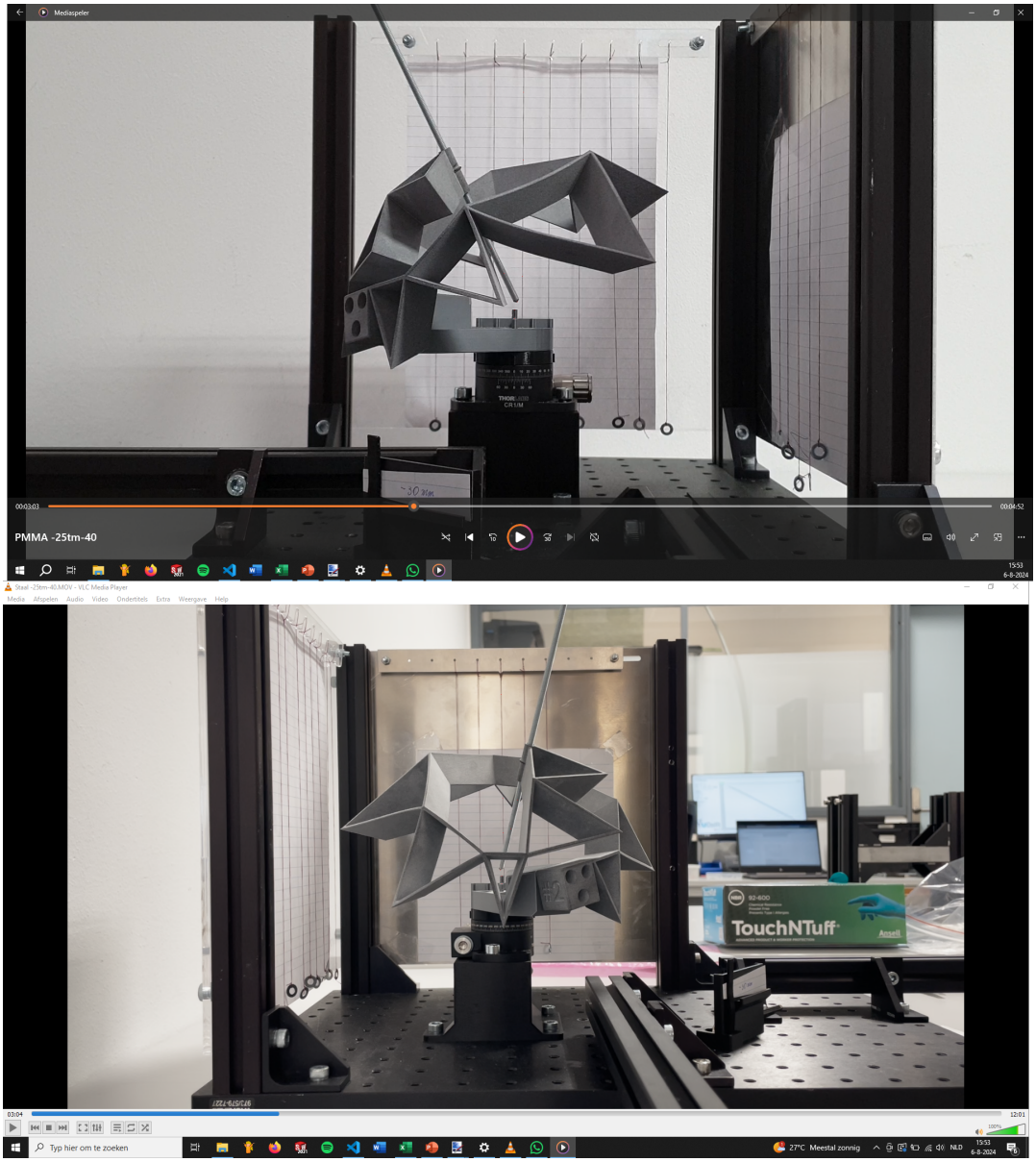

Figure F.1: A screenshot of both paused videos, showing two perpendicular projected views of the balanced pendulum. The upper screen has a transparent PMMA plate in the background and the lower screen a steel plate.

Figure F.2: A zoomed-in view of the upper (PMMA) screen showing two red marked pixels on the fixed reference point, three on the pendulum's central axis and one on a gravity indicator (two markings on the gravity indicator are located outside of this view).

F.3. Obtaining the translation and orientation of the pendulum rod

The three pixels on the gravity indicator are used to determine the direction of gravity in both projected views captured in the screenshot. This information quantifies the tilt angle under which the cameras recorded the video which is then factored into subsequent calculations to correct for this imperfection. Similarly, the three pixels on the central axis of the pendulum are used to determine the pendulum's angle in both projected views. These captured angles are then used to calculate the ψ_h and ψ_w angles. The parasitic motion is assessed using the marked P_O-indicator and the fixed reference point. The fixed reference point has a height of exactly 5mm and, in undeformed state, the P_O-indicator should be exactly 5mm above the top of the fixed reference point. By counting the pixels between the top and bottom markings, the size of a pixel in millimeters can be determined, allowing the translation of the P_O-indicator to be calculated.

F.4. Calculating the position of the P_{EE}-indicator

The P_{EE}-indicator is not always visible in the screenshots and is positioned relatively far away from the fixed reference point. Calculating its position directly using the known pixel-size in millimeters is therefore either not possible or prone to large errors. However, the location of the P_{EE}-indicator can still be calculated with the previously obtained translation of the P_O-indicator and the calculated ψ_h and ψ_w angles of the pendulum. Since the distance between the P_{EE} and P_O indicators is fixed and known as R_{Mo}, this information allows for an accurate calculation of the P_{EE} indicator's position.

G

Shell Scrips

This appendix presents one of the shell scripts used to submit a parallel job to the PME cluster of the DelftBlue supercomputer. The script instructs the 'BBO3_tien_tetras.py' file (the TetraFEM code specifically for the ten_tetrahedra optimization) to run simultaneously on 25 cores of a single node, with the results being saved to a text file named 'Results_BBO3_tien_tetras.log'. Since most nodes on the supercomputer have fewer than 50 cores, each parallel job was submitted twice to achieve the desired 50 optimization runs.

The shell scripts for the other optimizations differ only in walltime (expected maximum computation time), TetraFEM configuration, directory name and job name. As these differences are minor, they are not included in this appendix.

```
#!/bin/sh
#
#PBS -N BBO3_tien_tetras2
#PBS -l nodes=1:ppn=25,walltime=80:00:00
#PBS -q pme
#PBS -
#PBS -m bea

seq 25 | parallel -j 25 python ~/BBO3/Tien_tetras/BBO3_tien_tetras.py >> ~/BBO3/
    Tien_tetras/Result_BBO3_tien_tetras.log
```

H

TetraFEM Code

This appendix contains the TetraFEM algorithm written in Python. The code can be divided in 3 parts: Importing Libraries, Defining Functions and Control Panel. As the code itself contains few comments, a brief description of these 3 parts will be provided in this appendix.

H.1. Importing Libraries

The first and shortest part of the code is responsible for importing several libraries. These libraries are mainly required for mathematical calculations (numpy and sympy), plotting (matplotlib and mpl_toolkits) and optimizing (scipy).

H.2. Defining Funtions

The second and largest part of the code is responsible for defining the 23 functions of the TetraFEM tool. Table H.1 may help the reader understand the TetraFEM tool by providing a short description of each function and relating them to a section, figure or equation of the main paper or appendix.

H.3. Control Panel

The third part of the code lists the input of the TetraFEM algorithm. It contains:

- The amount of tetrahedra in the simulated joint
- All parameters from Table I
- Amount of self-weight iterations
- Parameters determining the evenly distributed P_{goal} points
- Data on the prototype's identified equilibria
- All boundaries from Table II
- The simulated joint's independent geometric parameters

Additionally, the Control Panel allows the user to choose between testing the simulated joint, validating the prototype, validating the TetraFEM tool itself or performing an optimization run.

Name function	Short description	(Mainly) relates to:
Geometric_calculations1	Calculates the weight and geometric parameters of flexure AC + calculates β.	Subsection II-B
Ref_frames_and_points1	Defines frames and points of flexure AC and points related to the tetrahedron.	Subsection II-B
Geometric_calculations2	Calculates the weight and geometric parameters of flexures AB and BC.	Appendix E
Ref_frames_and_points2	Defines frames and points of flexures AB and BC.	Appendix E
T_matrix_calculator	Calculates the transformation matrix given two points and two frames.	Eq. 26, 35, 36, 54 and 57
C_matrix_calculator	Calculates the compliance matrix of a flexure.	Subsubsection II-C1
C_matrix_ref_transformer	Transforms a compliance matrix to a different frame and point.	Eq. 37 and 38
sym_to_num_nonscalar	Changes a parameter from symbolic to numerical.	Eq. 34
Tetra_calculations	Uses the previous functions to calculate the tetrahedron's compliance matrix.	Subsubsection II-C2
Joint_calculations_slow	Loops Tetra_calculations n times and captures all relevant data.	Subsubsection II-C2
calculate_point_coords	Returns the coordinates of a point.	Fig. 16-26, 28, 29
tetra_plotter	Plots the flexures of a tetrahedron as (projected) planes.	Fig. 16-26, 28, 29
colour_point	Returns the color of a P_{EE}-indicator point based on its moment error.	Fig. 28, 29
prototype_val_plotter	Plots the points, ROM, colorbars and disk in Fig. 28 of the main paper.	Fig. 28
TetraFEM_val_plotter	Plots the P_{goal} points and the colorbar in Fig. 29 of the main paper.	Fig. 29
Joint_Sphere_plotter	Plots the figures, P_{goal} points (except for Fig. 29) and P_{EE} points.	Fig. 16-26, 28, 29
radius_calc	Calculates R_{Mo}.	Eq. 7
P_goal_calculator	Calculates the coordinates of the evenly distributed P_{goal} points.	Subsubsection II-C4
deflection_function	Calculates the deflection of the simulated joint.	Subsubsection II-C3
deflection_function_nsw	Calculates the deflection of the simulated joint, neglecting its self-weight.	Subsubsection II-C3
Def_EE_calc	Executes deflection_function for evenly spaced P_{goal} points + calculates s_{avg}.	Subsubsection II-C4
Def_EE_calc_TetraFEM_val	Executes deflection_function for the P_{EE}-indicator points + calculates s_{avg}.	Subsubsection II-C4
Pendulum_test	Uses the previous functions to calculate d_{avg} + the total calculation time.	Subsubsection II-C4

Table H.1: A short description of the 23 functions in the TetraFEM tool.

H.4. Python Code

```
1
2  globals().clear()
3
4  ### IMPORTING LIBRARIES ###
5
6  import sympy as sm
7  import sympy.physics.mechanics as me
8  import matplotlib.pyplot as plt
9  from mpl_toolkits.mplot3d import Axes3D, art3d
10 from mpl_toolkits.mplot3d.art3d import Poly3DCollection
11 from sympy.matrices import block_collapse
12 import numpy as np
13 from collections import namedtuple
14 import time
15 from scipy import optimize
16 from matplotlib import colors as mcolors
17 from mpl_toolkits.axes_grid1.inset_locator import inset_axes
18
19
20 ### DEFINING FUNCTIONS ###
21
22 def Geometric_calculations1(alpha_d, gamma_d, t_min, R_in, R_w, n_samples, E, G,
       density):
23
24         #From degrees to radians:
25         alpha = alpha_d/180*np.pi
26         gamma = gamma_d/180*np.pi
27
28         #Calculate maximum thickness
29         t_max = ((R_in+R_w)/R_in)*t_min
30
31         #AC flexure parameters
32         R_out = R_in+R_w
33         R_Mo = R_out - R_w*(t_max+2*t_min)/(3*(t_max+t_min))
34         R_Mm_AC = R_Mo*np.cos(alpha/2)
35         R_in_m_AC = R_in*np.cos(alpha/2)
36         R_w_m_AC = R_w*np.cos(alpha/2)
37         R_out_m_AC = R_out*np.cos(alpha/2)
38
39         L_a1c1 = 2*np.sin(0.5*alpha)*(R_out)
40         L_a2c2 = 2*np.sin(0.5*alpha)*(R_in+0.5*R_w)
41         L_a3c3 = 2*np.sin(0.5*alpha)*(R_in)
42
43         #Calculate mass stuff
44         Mass_AC = density*(L_a1c1*t_max*R_out_m_AC - L_a3c3*t_min*R_in_m_AC)/3
45         V_AC_1 = t_max*L_a1c1*R_out_m_AC/3
46         V_AC_3 = t_min*L_a3c3*R_in_m_AC/3
47         R_AC_1 = 3*R_out_m_AC/4
48         R_AC_3 = 3*R_in_m_AC/4
49         R_AC_com = (R_AC_1*V_AC_1-R_AC_3*V_AC_3)/(V_AC_1-V_AC_3)
50
51         #Calculating optimal beta:
52         poisson = E/(2*G)-1
53         L_a2c2 = 2*np.sin(0.5*alpha)*(R_in+0.5*R_w)
54         H = 0.5*((4*L_a2c2**4 + (48*L_a2c2**2*R_w_m_AC**2*(poisson+1))/5)**0.5 +
               L_a2c2**2 + (12*R_w_m_AC**2*(poisson+1))/5)**0.5
55         beta = np.arctan(H/(R_in_m_AC+R_w_m_AC/2))
56
57         #AB-BC flexure paramater
```

```
58          phi = np.arctan(np.tan(beta)/np.sin(alpha/2)) #Formule bepaald aan hand
                van reverse engineering
59
60          #Delta_alpha numerical values:
61          delta_alpha = np.linspace(0, alpha,n_samples,endpoint=True).transpose()
62
63          return alpha, beta, gamma, phi, t_min, t_max, R_in, R_w, R_out, R_Mo,
                R_Mm_AC, R_in_m_AC, R_w_m_AC, R_out_m_AC, delta_alpha, Mass_AC,
                R_AC_com
64
65
66  def Ref_frames_and_points1(T0, P_0, alpha, delta_alpha, beta, gamma, phi, R_Mo,
        R_Mm_AC, R_in, R_out, R_in_m_AC, R_out_m_AC, R_AC_com):
67      #Introduce the Reference frames
68      AC_a = me.ReferenceFrame('AC_a')
69      AC_c = me.ReferenceFrame('AC_c')
70      AC_l = me.ReferenceFrame('AC_l')
71      AC_m = me.ReferenceFrame('AC_m')
72      AB_a = me.ReferenceFrame('AB_a')
73
74      #Orient the Reference frames
75      AC_a.orient_axis(T0, -gamma, T0.y)
76      AC_c.orient_axis(AC_a, -alpha, AC_a.z)
77      AC_l.orient_axis(AC_a, -delta_alpha, AC_a.z)
78      AC_m.orient_axis(AC_a, -alpha/2, AC_a.z)
79      AB_a.orient_axis(AC_a, -phi, AC_a.y)
80
81      #Introduce the centroid Points
82      M_a = me.Point('M_a')
83      M_c = me.Point('M_c')
84      M_l_AC = me.Point('M_l_AC')
85
86      #Introduce other important Points
87      P_a1 = me.Point('P_a1')
88      P_a3 = me.Point('P_a3')
89      P_b1 = me.Point('P_b1')
90      P_b3 = me.Point('P_b3')
91      P_c1 = me.Point('P_c1')
92      P_c3 = me.Point('P_c3')
93      P_AC_com = me.Point('P_AC_com')
94
95      #Calculate Centroid radius of the AC flexure
96      R_M_AC = R_Mm_AC/sm.cos(alpha/2 - delta_alpha)
97
98      #Determine the position of the centroid Points
99      M_a.set_pos(P_0, R_Mo*AC_a.y)
100     M_c.set_pos(P_0, R_Mo*AC_c.y)
101     M_l_AC.set_pos(P_0, R_M_AC*AC_l.y)
102
103     #Determine the position of the other Points
104     P_a1.set_pos(P_0, R_out*AC_a.y)
105     P_a3.set_pos(P_0, R_in*AC_a.y)
106     P_b1.set_pos(P_0, R_out_m_AC*AC_m.y + (R_out_m_AC*np.tan(beta))*AC_m.z)
107     P_b3.set_pos(P_0, R_in_m_AC*AC_m.y + (R_in_m_AC*np.tan(beta))*AC_m.z)
108     P_c1.set_pos(P_0, R_out*AC_c.y)
109     P_c3.set_pos(P_0, R_in*AC_c.y)
110     P_AC_com.set_pos(P_0, R_AC_com*AC_m.y)
111
112     return AC_a, AC_c, AC_l, AB_a, M_a, M_c, M_l_AC, P_a1, P_a3, P_b1, P_b3, P_c1,
            P_c3, P_AC_com, R_M_AC
113
```

```
114
115 def Geometric_calculations2(P_0, AB_a, P_a3, P_a1, P_b3, P_b1, t_min, t_max, R_in,
        R_w, n_samples):
116     r_a3 = P_a3.pos_from(P_0).to_matrix(AB_a)
117     r_a1 = P_a1.pos_from(P_0).to_matrix(AB_a)
118     r_b3 = P_b3.pos_from(P_0).to_matrix(AB_a)
119     r_b1 = P_b1.pos_from(P_0).to_matrix(AB_a)
120     r_a3_b3 = P_b3.pos_from(P_a3).to_matrix(AB_a)
121
122     r_w_a = P_a1.pos_from(P_a3).to_matrix(AB_a)
123     r_w_b = P_b1.pos_from(P_b3).to_matrix(AB_a)
124
125     theta = sm.acos(r_a3.dot(r_b3)/( r_a3.norm() * r_b3.norm() ) )
126     theta2 = np.pi - sm.acos(-r_a3.dot(r_a3_b3)/( -r_a3.norm() * r_a3_b3.norm() )
          )
127     theta3 = np.pi/2 - theta2
128     theta4 = theta-theta3
129
130     r_Ma = r_a1 - r_w_a*(t_max+2*t_min)/(3*(t_max+t_min))
131     r_Mb = r_b1 - r_w_b*(t_max+2*t_min)/(3*(t_max+t_min))
132
133     R_Ma = r_Ma.norm()
134     R_Mb = r_Mb.norm()
135     R_w_a = r_w_a.norm()
136
137     R_M90_df = (sm.cos(theta3)*R_Ma)
138     R_w_90_df = (sm.cos(theta3)*R_w_a)
139
140     R_out = R_in+R_w
141     R_in_90_df = sm.cos(theta3)*R_in
142     R_out_90_df = sm.cos(theta3)*R_out
143     L_a3b3 = R_in_90_df*(sm.tan(theta3)+sm.tan(theta4))
144     L_a1b1 = R_out_90_df*(sm.tan(theta3)+sm.tan(theta4))
145
146     #Mass stuff
147     Mass_df = density/3*(L_a1b1*t_max*R_out_90_df - L_a3b3*t_min*R_in_90_df)
148     V_df_1 = t_max*L_a1b1*R_out_90_df/3
149     V_df_3 = t_min*L_a3b3*R_in_90_df/3
150     R_df_1 = 3*R_out_90_df/4
151     R_df_3 = 3*R_in_90_df/4
152     R_90_com = (R_df_1*V_df_1-R_df_3*V_df_3)/(V_df_1-V_df_3)
153     angle_df_com = theta/2 - theta3
154     R_df_com = R_90_com/sm.cos(angle_df_com)
155
156
157     delta_theta = np.linspace(0, float(theta), n_samples, endpoint=True).transpose
          ()
158
159     return theta, theta3, theta4, delta_theta, R_Mb, R_M90_df, R_w_90_df, Mass_df,
          R_df_com
160
161
162 def Ref_frames_and_points2(P_0, AB_a, AC_c, theta, theta3, theta4, delta_theta,
      phi, R_M90_df, R_Mb, R_df_com):
163     R_M_AB = R_M90_df/sm.cos(theta3-delta_theta)
164     R_M_BC = R_M90_df/sm.cos(theta4-delta_theta)
165
166     AB_b = me.ReferenceFrame('AB_b')
167     AB_l = me.ReferenceFrame('AB_l')
168     AB_com = me.ReferenceFrame('AB_com')
169     BC_c = me.ReferenceFrame('BC_c')
```

```
    BC_b = me.ReferenceFrame('BC_b')
    BC_l = me.ReferenceFrame('BC_l')
    BC_com = me.ReferenceFrame('BC_com')

    AB_b.orient_axis(AB_a, -theta, AB_a.z)
    AB_l.orient_axis(AB_a, -delta_theta, AB_a.z)
    AB_com.orient_axis(AB_a, -theta/2, AB_a.z)
    BC_c.orient_axis(AC_c, phi, AC_c.y)
    BC_b.orient_axis(BC_c, theta, BC_c.z)
    BC_l.orient_axis(BC_b, -delta_theta, BC_b.z)
    BC_com.orient_axis(BC_c, theta/2, BC_c.z)


    M_b = me.Point('M_b')
    M_l_AB = me.Point('M_l_AB')
    M_l_BC = me.Point('M_l_BC')

    P_AB_com = me.Point('P_AB_com')
    P_BC_com = me.Point('P_BC_com')

    M_b.set_pos(P_O, R_Mb*AB_b.y)
    M_l_AB.set_pos(P_O, R_M_AB*AB_l.y)
    M_l_BC.set_pos(P_O, R_M_BC*BC_l.y)

    P_AB_com.set_pos(P_O, R_df_com*AB_com.y)
    P_BC_com.set_pos(P_O, R_df_com*BC_com.y)

    return AB_b, AB_l, M_b, M_l_AB, R_M_AB, BC_c, BC_l, M_l_BC, R_M_BC, P_BC_com,
        P_BC_com


def T_matrix_calculator(P_begin, P_end, Ref_frame_begin, Ref_frame_end):
    r_end_begin = P_end.pos_from(P_begin)
    r_end_begin_mat = r_end_begin.to_matrix(Ref_frame_begin)                    #
        Convert to a matrix type
    begin_rr_end_begin = np.matrix([[0, -r_end_begin_mat[2], r_end_begin_mat[1]],
                        [r_end_begin_mat[2], 0, -r_end_begin_mat[0]],
                        [-r_end_begin_mat[1], r_end_begin_mat[0], 0]])

    begin_R_end = Ref_frame_begin.dcm(Ref_frame_end)
    begin_T_end = sm.Matrix.vstack(
        sm.Matrix.hstack(begin_R_end, sm.zeros(3)),
        sm.Matrix.hstack(begin_rr_end_begin*begin_R_end, begin_R_end)
    )
    return begin_T_end


def C_matrix_calculator(T_matrix, main_angle, sec_angle, delta_angle, R_M, R_w,
    R_in, t_min, t_max, G, E, c_shear, p_total, vals, n_samples):
    b_y = c_shear
    b_z = c_shear

    R_w_M = R_w/sm.cos(sec_angle-delta_angle)
    A_x = (t_max+t_min)*R_w_M/2
    I_y = R_w_M*(t_min+t_max)*(t_min**2+t_max**2)/(48)
    I_z = R_w_M**3*(t_min**2+4*t_max*t_min+t_max**2)/(36*(t_max+t_min))

    #t_avg = (t_max+t_min)/2
    #J = R_w_M*t_avg**3 * (1/3-0.21*(t_avg/R_w_M)*(1-t_avg**4/(12*R_w_M**4)))
                                        #[m^4] Torsional constant if approximated
        as rectangular cross-section
```

```
226     R_in_M = R_in/sm.cos(sec_angle-delta_angle)
227     R_out_M = R_in_M + R_w_M
228     eta = sm.atan(t_min/(2*R_in_M))
229     V_L = 0.10504 - 0.2*sm.sin(eta) + 0.3392*sm.sin(eta)**2 - 0.53968*sm.sin(eta)
            **3 + 0.82448*sm.sin(eta)**4
230     V_S = 0.10504 + 0.2*sm.sin(eta) + 0.3392*sm.sin(eta)**2 + 0.53968*sm.sin(eta)
            **3 + 0.82448*sm.sin(eta)**4
231     J = (2/3)*(sm.sin(eta)**3)*(R_out_M**4-R_in_M**4)-16*sm.sin(V_L*R_out_M**4+V_S
            *R_in_M**4)**4
232
233     l_K = sm.diag(E*A_x, b_y*G*A_x, b_z*G*A_x, G*J, E*I_y, E*I_z)
234
235     l_K_num = sym_to_num_nonscalar(l_K, delta_angle, p_total, [], vals, n_samples)
236     T_matrix_num = sym_to_num_nonscalar(T_matrix, delta_angle, p_total, [], vals,
            n_samples)
237     R_M_num = sym_to_num_nonscalar(R_M, delta_angle, p_total, [], vals, n_samples)
238
239     l_K_num_inv = np.linalg.inv(l_K_num)
240     T_matrix_num_trans = np.transpose(T_matrix_num, axes=(0,2,1))
241
242
243     C_function_num = np.zeros((n_samples,6, 6))
244
245     for i in range(n_samples):
246         C_function_num[i] = np.matmul((np.matmul(T_matrix_num_trans[i],
                l_K_num_inv[i])), T_matrix_num[i])*R_M_num[i]
247
248     C_matrix_num = (main_angle/n_samples)*(np.sum(C_function_num, axis=0)-(
            C_function_num[0]+C_function_num[n_samples-1])/2)
249
250     return C_matrix_num
251
252
253 def C_matrix_ref_transformer(C_matrix, P_begin, P_end, Ref_frame_begin,
        Ref_frame_end):
254
255     begin_T_end = T_matrix_calculator(P_begin, P_end, Ref_frame_begin,
            Ref_frame_end)
256     C_matrix_t = np.matmul((np.matmul(begin_T_end.transpose(), C_matrix)),
            begin_T_end)
257
258     return C_matrix_t
259
260
261 def sym_to_num_nonscalar(expr, delta_angle, p_total, p_scalar, values, n_samples):
262     expr_lamb = sm.lambdify(p_total, expr, 'numpy')
263
264     if isinstance(expr, sm.Mul):
265         expr_num = np.zeros(n_samples)
266     else:
267         expr_num = np.zeros((n_samples, 6, 6))
268
269     for i in range(n_samples):
270         args = [values[param] for param in p_scalar]
271         args.extend([values[delta_angle][i]])
272
273         expr_num[i] = expr_lamb(*args)
274
275     return expr_num
276
277
```

```
278 def Tetra_calculations(alpha_d, gamma_d, t_min, R_in, R_w, n_samples, E, G,
        density, c_shear_val, T0, P_0):
279
280     #Extracting values from the Geometric_calculations1 function
281     alpha, beta, gamma, phi, t_min, t_max, R_in, R_w, R_out, R_Mo, R_Mm_AC,
            R_in_m_AC, R_w_m_AC, R_out_m_AC, delta_alpha_val, Mass_AC, R_AC_com =
            Geometric_calculations1(alpha_d, gamma_d, t_min, R_in, R_w, n_samples, E, G
            , density)
282
283     #Introducing AC flexure symbol
284     delta_alpha = sm.symbols('delta_alpha')
285
286     #Extracting parameters from the Ref_frames_and_points1 function:
287     AC_a, AC_c, AC_l, AB_a, M_a, M_c, M_l_AC, P_a1, P_a3, P_b1, P_b3, P_c1, P_c3,
            P_AC_com, R_M_AC = Ref_frames_and_points1(T0, P_0, alpha, delta_alpha, beta
            , gamma, phi, R_Mo, R_Mm_AC, R_in, R_out, R_in_m_AC, R_out_m_AC, R_AC_com)
288
289     p_total1 = sm.Matrix([delta_alpha])
290     vals1 = {delta_alpha:delta_alpha_val}
291
292     #Extracting values from the Geometric_calculations2 function
293     theta, theta3, theta4, delta_theta_val, R_Mb, R_M90_df, R_w_90_df, Mass_df,
            R_df_com = Geometric_calculations2(P_0, AB_a, P_a3, P_a1, P_b3, P_b1, t_min
            , t_max, R_in, R_w, n_samples)
294
295     #Introducing AB-BC symbols
296     delta_theta = sm.symbols('delta_theta')
297
298     #Extracting parameters from the Ref_frames_and_points2 function:
299     AB_b, AB_l, M_b, M_l_AB, R_M_AB, BC_c, BC_l, M_l_BC, R_M_BC, P_AB_com,
            P_BC_com = Ref_frames_and_points2(P_0, AB_a, AC_c, theta, theta3, theta4,
            delta_theta, phi, R_M90_df, R_Mb, R_df_com)
300
301     p_total2 = sm.Matrix([delta_theta])
302     vals2 = {delta_theta:delta_theta_val}
303
304     #Extraction Transformation matrices
305     ACl_T_ACc = T_matrix_calculator(M_l_AC, M_c, AC_l, AC_c)
306     ABl_T_ABb = T_matrix_calculator(M_l_AB, M_b, AB_l, AB_b)
307     BCl_T_BCc = T_matrix_calculator(M_l_BC, M_c, BC_l, BC_c)
308
309     ACc_C_num = C_matrix_calculator(ACl_T_ACc, alpha, alpha/2, delta_alpha, R_M_AC
            , R_w_m_AC, R_in, t_min, t_max, G, E, c_shear, p_total1, vals1, n_samples)
310
311     ABb_C_num = C_matrix_calculator(ABl_T_ABb, theta, theta3, delta_theta, R_M_AB,
             R_w_90_df, R_in, t_min, t_max, G, E, c_shear, p_total2, vals2, n_samples)
312     BCc_C_num = C_matrix_calculator(BCl_T_BCc, theta, theta4, delta_theta, R_M_BC,
             R_w_90_df, R_in, t_min, t_max, G, E, c_shear, p_total2, vals2, n_samples)
313
314
315     ##########################################)
316
317     ABb_C_t = C_matrix_ref_transformer(ABb_C_num, M_b, M_c, AB_b, AC_c)
318     BCc_C_t = C_matrix_ref_transformer(BCc_C_num, M_c, M_c, BC_c, AC_c)
319
320     ABBCc_C_t = ABb_C_t+BCc_C_t
321
322     ABBCc_C_t = ABBCc_C_t.astype(float)
323     ACc_C_num = ACc_C_num.astype(float)
324
325     #Adding stiffness matrices
```

```
326     ABBCc_K_num = np.linalg.inv(ABBCc_C_t)
327     ACc_K_num = np.linalg.inv(ACc_C_num)
328     a_K_c = ABBCc_K_num+ACc_K_num
329     a_C_c = np.linalg.inv(a_K_c)
330 
331     Tetra_output = namedtuple(
332     'Tetra_output',
333     ['a_C_c', 'P_a1', 'P_a3', 'P_b1', 'P_b3', 'P_c1', 'P_c3',
334      'M_a', 'M_b', 'M_c', 'AC_c', 'AC_a', 'P_AC_com', 'P_AB_com', 'P_BC_com', '
             Mass_AC', 'Mass_df', 'phi', 'theta', 'R_AC_com', 'R_df_com']
335     )
336 
337     return  Tetra_output(a_C_c, P_a1, P_a3, P_b1, P_b3, P_c1, P_c3, M_a, M_b, M_c,
             AC_c, AC_a, P_AC_com, P_AB_com, P_BC_com, Mass_AC, Mass_df, phi, theta,
            R_AC_com, R_df_com)
338 
339 
340 def Joint_calculations_slow(alpha_d, gamma_d, t_min, R_in, R_w, n_samples, E, G,
    density, c_shear):
341 
342     T0 = me.ReferenceFrame('T0')            #Reference frame of the world
343     P_0 = me.Point('P_0')                   #The origin and center of rotation
344 
345     n_tetra = np.size(alpha_d)
346 
347     T_start = [T0]
348     Tetra_outputs = []
349 
350     P_a1 = []
351     P_a3 = []
352     P_b1 = []
353     P_b3 = []
354     P_c1 = []
355     P_c3 = []
356     M_a = []
357     M_b = []
358     M_c = []
359 
360     for i in range(n_tetra):
361 
362         #Tetra_outputs verkrijgen
363         Tetra_output = Tetra_calculations(alpha_d[i], gamma_d[i], t_min[i], R_in[i
                ], R_w[i], n_samples, E, G, density, c_shear, T_start[i], P_0)
364 
365         T_start.append(Tetra_output.AC_c)
366         Tetra_outputs.append(Tetra_output)
367 
368         #Points voor plotten verkrijgen
369         P_a1.append(Tetra_output.P_a1)
370         P_a3.append(Tetra_output.P_a3)
371         P_b1.append(Tetra_output.P_b1)
372         P_b3.append(Tetra_output.P_b3)
373         P_c1.append(Tetra_output.P_c1)
374         P_c3.append(Tetra_output.P_c3)
375         M_a.append(Tetra_output.M_a)
376         M_b.append(Tetra_output.M_b)
377         M_c.append(Tetra_output.M_c)
378 
379         if i == n_tetra-1:
380             T_joint = (Tetra_output.AC_c).orientnew('T_joint', 'Axis', (-np.pi/2,
                    (Tetra_output.AC_c).x))
```

```
381
382     return Tetra_outputs, P_a1, P_a3, P_b1, P_b3, P_c1, P_c3, M_c, T_joint, T0,
            P_0
383
384
385 def calculate_point_coords(point, origin, world_frame):
386     point_coords = point.pos_from(origin).to_matrix(world_frame)
387     return [float(element) for element in point_coords]
388
389
390 def tetra_plotter(P_a1, P_a3, P_b1, P_b3, P_c1, P_c3, M_c, origin, world_frame,
        ax_3d, ax_xz, ax_yz, ax_xy, color, i):
391     # Get the coordinates of the points in your reference frames
392
393     P_a1_coords = calculate_point_coords(P_a1[i], origin, world_frame)
394     P_a3_coords = calculate_point_coords(P_a3[i], origin, world_frame)
395     P_b1_coords = calculate_point_coords(P_b1[i], origin, world_frame)
396     P_b3_coords = calculate_point_coords(P_b3[i], origin, world_frame)
397     P_c1_coords = calculate_point_coords(P_c1[i], origin, world_frame)
398     P_c3_coords = calculate_point_coords(P_c3[i], origin, world_frame)
399
400     P_a1_coords_mm = np.array(P_a1_coords) * 1000
401     P_a3_coords_mm = np.array(P_a3_coords) * 1000
402     P_b1_coords_mm = np.array(P_b1_coords) * 1000
403     P_b3_coords_mm = np.array(P_b3_coords) * 1000
404     P_c1_coords_mm = np.array(P_c1_coords) * 1000
405     P_c3_coords_mm = np.array(P_c3_coords) * 1000
406
407     opvulling = 0.4
408
409     #Plotting the flexures as planes:
410     #In 3D:
411     surfaces = [[P_a1_coords_mm, P_a3_coords_mm, P_c3_coords_mm, P_c1_coords_mm],
412                 [P_a1_coords_mm, P_a3_coords_mm, P_b3_coords_mm, P_b1_coords_mm],
413                 [P_b1_coords_mm, P_b3_coords_mm, P_c3_coords_mm, P_c1_coords_mm]]
414     ax_3d.add_collection3d(Poly3DCollection(surfaces, facecolors=color[i],
            linewidths=0.3, edgecolors='r', alpha=opvulling))
415
416     surfaces_yz = [surface[:4] for surface in surfaces]
417
418     # Plot the projected surfaces
419     for surface in surfaces_yz:
420         surface = np.array(surface)  # Convert the surface to a NumPy array
421         ax_yz.add_patch(plt.Polygon(surface[:, [1, 2]], closed=True, facecolor=
                color[i], edgecolor='r', alpha=opvulling))
422
423     surfaces_xz = [surface[:4] for surface in surfaces]
424     for surface in surfaces_xz:
425         surface = np.array(surface) # Convert the surface to a NumPy array
426         ax_xz.add_patch(plt.Polygon(surface[:, [0, 2]], closed=True, facecolor=
                color[i], edgecolor='r', alpha=opvulling))
427
428     surfaces_xy = [surface[:4] for surface in surfaces]
429     for surface in surfaces_xy:
430         surface = np.array(surface)  # Convert the surface to a NumPy array
431         ax_xy.add_patch(plt.Polygon(surface[:, [0, 1]], closed=True, facecolor=
                color[i], edgecolor='r', alpha=opvulling))
432
433     if i == 0:
434         x = [P_a1_coords_mm[0], P_a3_coords_mm[0]]
435         y = [P_a1_coords_mm[1], P_a3_coords_mm[1]]
```

```
436         z = [P_a1_coords_mm[2], P_a3_coords_mm[2]]
437         ax_xy.plot(x, y, linewidth=3, c='k', linestyle='dashed', zorder=1)
438         ax_yz.plot(y, z, linewidth=3, c='k', linestyle='dashed')
439         ax_xz.plot(x, z, linewidth=3, c='k', linestyle='dashed')
440         ax_3d.plot(x, y, z, linewidth=3, c='k', linestyle='dashed', zorder=10)
441
442     if i == len(P_a1)-1:
443         EE_coords = calculate_point_coords(M_c[i], origin, world_frame)
444         #Plotting Points
445         #3D:
446         ax_3d.scatter(EE_coords[0]*1000, EE_coords[1]*1000, EE_coords[2]*1000, c='
                black', marker='o', label=r'P$_{EE}$', s=40)
447         #2D:
448         #ax_xz.scatter(EE_coords[0], EE_coords[2], c='black', marker='o', label='
                EE', s=40)
449         #ax_xy.scatter(EE_coords[0], EE_coords[1], c='black', marker='o', label='
                EE', s=30)
450         #ax_yz.scatter(EE_coords[1], EE_coords[2], c='black', marker='o', label='
                EE', s=40)
451
452     return
453
454
455 def colour_point(point, L_w, M_w, M_r, max_moment, indicator_multiplier):
456     L_w = L_w*1000 #in mm
457     #print(L_w, point[4])
458     if point[4]>L_w:
459         colour_func_point = ((indicator_multiplier*(point[4]-L_w))/L_w, 1-(
                indicator_multiplier*(point[4]-L_w))/L_w, 0, 1)
460     if point[4]==L_w:
461         colour_func_point = (0, 1, 0, 1)
462     if point[4]<L_w:
463         colour_func_point = (0, (L_w-indicator_multiplier*(L_w-point[4]))/L_w, 1-(
                L_w-indicator_multiplier*(L_w-point[4]))/L_w, 1)
464     #display("colour_func_point", colour_func_point)
465     if point[3] > 30:
466         Moment_error = "nvt"
467         Moment = "nvt"
468     else:
469         Moment_error = (M_w*g*(point[4]/1000-L_w/1000))*np.sin(point[3]/180*np.pi)
470         Moment = max_moment*np.sin(point[3]/180*np.pi)
471
472     return colour_func_point, Moment_error, Moment
473
474
475 def prototype_val_plotter(ax_xy, fig_xy, radius, H_p_range_d, L_w, L_r, M_w, M_r,
        g):
476     radius = radius *1000
477
478     ROM_points = [[28.8, -42.8, 59.8, 42.7], [22.3, -43.2, 62.5, 39.6], [5.7,
            -45.3, 65.1, 36.5], [-11.4, -45.1, 64.6, 32.9], [-23.7, -34.4, 67.5, 32.9],
             [-34.9, -24.8, 66.8, 34.0], [-46.2, -8.1, 64.9, 37.2], [-52.5, 12.9, 58.2,
             44.4], [34.3, -40.6, 58.8, 44.0]]
479
480     angle_lim = 45
481     radius_limxy = np.sin(angle_lim/180*np.pi)*radius
482     resolution = 60
483     radii_xy = np.linspace(0, radius_limxy, resolution)
484
485     max_moment = M_w * g * L_w + M_r * g * L_r / 2
486     color_value_max = max_moment*np.sin(angle_lim/180*np.pi)
```

```
    color_value_30 = max_moment*np.sin(30/180*np.pi)
    color_value_15 = max_moment*np.sin(15/180*np.pi)
    color_value_min = 0

    indicator_multiplier = 5
    L_w_min = L_w*(1-1/indicator_multiplier)*1000
    L_w_max = L_w*(1+1/indicator_multiplier)*1000

    for rad in radii_xy:
        colour_func_circle = (1-rad/radius_limxy, 1-rad/radius_limxy, 1-rad/
            radius_limxy, 1)
        coloured_circle = plt.Circle((0, 0), rad, edgecolor=colour_func_circle,
            facecolor='none', linewidth=3)
        ax_xy.add_patch(coloured_circle)# Create a circle

    #Add circle 30 degrees
    radius_30xy = np.sin(H_p_range_d[1]/180*np.pi)*radius
    circle = plt.Circle((0, 0), radius_30xy, edgecolor='yellow', facecolor='none',
         alpha=1, linewidth=0.8)
    ax_xy.add_patch(circle)# Create a circle

    abs_moment_error_coll = []
    Moment_coll = []

    for point in stable_points:
        colour_func_point, Moment_error, Moment = colour_point(point, L_w, M_w,
            M_r, max_moment, indicator_multiplier)
        if isinstance(Moment_error, (int, float)):
            abs_moment_error_coll.append(abs(Moment_error))
            Moment_coll.append(Moment)
        ax_xy.scatter(point[0], point[1], c=[colour_func_point], marker="s", s
            =100, edgecolor=(0,0,0,0.4), linewidth=1)
        ax_xy.scatter(point[0], point[1], c='k', marker=".", s=4)

    for point in unstable_points:
        colour_func_point, Moment_error, Moment = colour_point(point, L_w, M_w,
            M_r, max_moment, indicator_multiplier)
        if isinstance(Moment_error, (int, float)):
            abs_moment_error_coll.append(abs(Moment_error))
            Moment_coll.append(Moment)
        ax_xy.scatter(point[0], point[1], color=colour_func_point, marker="D", s
            =80, edgecolor=(0,0,0,0.4), linewidth=1)
        ax_xy.scatter(point[0], point[1], c='k', marker=".", s=4)

    for point in ROM_points:
        ax_xy.scatter(point[0], point[1], c='purple', s=5)

    avg_moment_error = sum(abs_moment_error_coll)/len(abs_moment_error_coll)
    perc_moment_red = (1 - sum(abs_moment_error_coll)/sum(Moment_coll))*100

    print("The average moment error = ", avg_moment_error, " Nm")
    print("The moment reduction = ", perc_moment_red, "%")

    # Create a colormap from white to black
    cmap = mcolors.LinearSegmentedColormap.from_list('WhiteBlack', [(1, 1, 1), (0,
         0, 0)])
    norm = mcolors.Normalize(vmin=color_value_min, vmax=color_value_max)
    # Add the colorbar
    sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
    sm.set_array([])
```

```
539     axins = inset_axes(ax_xy, width="5%", height="30%", loc='upper right',
            borderpad=1)
540     cbar = plt.colorbar(sm, cax=axins)
541     #cbar.set_label('Value')
542     cbar.ax.yaxis.set_ticks_position('left')
543     cbar.ax.yaxis.set_label_position('left')
544     cbar.set_ticks([color_value_min, color_value_15, color_value_30,
            color_value_max])
545     cbar.set_ticklabels(['0 deg | 0 Nm', f'15 deg | {color_value_15:.3f} Nm', f'30
             deg | {color_value_30:.3f} Nm', f'45 deg | {color_value_max:.3f} Nm'],
            fontsize=14)
546
547     # Create a colormap from blue to green to red
548     cmap2 = mcolors.LinearSegmentedColormap.from_list('BlueGreenRed', [(0, 0, 1),
            (0, 1, 0), (1, 0, 0)])
549     norm2 = mcolors.TwoSlopeNorm(vmin=L_w_min, vcenter=L_w*1000, vmax=L_w_max)
550     # Add the colorbar
551     sm2 = plt.cm.ScalarMappable(cmap=cmap2, norm=norm2)
552     sm2.set_array([])
553     axins2 = inset_axes(ax_xy, width="5%", height="30%", loc='upper left',
            borderpad=1)
554     cbar2 = plt.colorbar(sm2, cax=axins2)
555     #cbar.set_label('Value')
556     cbar2.ax.yaxis.set_ticks_position('right')
557     cbar2.ax.yaxis.set_label_position('right')
558     cbar2.set_ticks([L_w_min, L_w*1000, L_w_max])
559     cbar2.set_ticklabels([f'Lw = {L_w_min:.0f} mm', f'Lw = {L_w*1000:.0f} mm', f'
            Lw = {L_w_max:.0f} mm'], fontsize=14)
560
561     return
562
563
564 def TetraFEM_val_plotter(ax_xy, fig_xy, radius, H_p_range_d, L_w, L_r, M_w, M_r, g
        ):
565     radius = radius *1000
566
567     max_moment = M_w * g * L_w + M_r * g * L_r / 2
568
569     indicator_multiplier = 5
570     L_w_min = L_w*(1-1/indicator_multiplier)*1000
571     L_w_max = L_w*(1+1/indicator_multiplier)*1000
572
573
574     for point in stable_points:
575         colour_func_point, Moment_error, Moment = colour_point(point, L_w, M_w,
                M_r, max_moment, indicator_multiplier)
576         ax_xy.scatter(point[0], point[1], c=[colour_func_point], marker="s", s
                =100, edgecolor=(0,0,0,0.4), linewidth=1)
577         ax_xy.scatter(point[0], point[1], c='k', marker=".", s=4)
578
579     for point in unstable_points:
580         colour_func_point, Moment_error, Moment = colour_point(point, L_w, M_w,
                M_r, max_moment, indicator_multiplier)
581         ax_xy.scatter(point[0], point[1], color=colour_func_point, marker="D", s
                =80, edgecolor=(0,0,0,0.4), linewidth=1)
582         ax_xy.scatter(point[0], point[1], c='k', marker=".", s=4)
583
584     # Create a colormap from blue to green to red
585     cmap2 = mcolors.LinearSegmentedColormap.from_list('BlueGreenRed', [(0, 0, 1),
            (0, 1, 0), (1, 0, 0)])
586     norm2 = mcolors.TwoSlopeNorm(vmin=L_w_min, vcenter=L_w*1000, vmax=L_w_max)
```

```
587     # Add the colorbar
588     sm2 = plt.cm.ScalarMappable(cmap=cmap2, norm=norm2)
589     sm2.set_array([])
590     axins2 = inset_axes(ax_xy, width="5%", height="30%", loc='upper left',
            borderpad=1)
591     cbar2 = plt.colorbar(sm2, cax=axins2)
592     #cbar.set_label('Value')
593     cbar2.ax.yaxis.set_ticks_position('right')
594     cbar2.ax.yaxis.set_label_position('right')
595     cbar2.set_ticks([L_w_min, L_w*1000, L_w_max])
596     cbar2.set_ticklabels([f'Lw = {L_w_min:.0f} mm', f'Lw = {L_w*1000:.0f} mm', f'
            Lw = {L_w_max:.0f} mm'], fontsize=14)
597
598     return
599
600
601 def Joint_Sphere_plotter(DataPoints_def_EE, P_a1, P_a3, P_b1, P_b3, P_c1, P_c3,
        M_c, P_0, T_joint, radius, H_p_range_d, L_w, L_r, M_w, M_r, g):
602
603     P_goal_x = np.array([DataPoints_def_EE[nr].P_goal_loc[0] for nr in range(len(
            DataPoints_def_EE))])*1000    # Accessing the x-coordinate of each point
            and making it plottable by arraying it
604     P_goal_y = np.array([DataPoints_def_EE[nr].P_goal_loc[1] for nr in range(len(
            DataPoints_def_EE))])*1000    # Accessing the y-coordinate of each point
605     P_goal_z = np.array([DataPoints_def_EE[nr].P_goal_loc[2] for nr in range(len(
            DataPoints_def_EE))])*1000    # Accessing the z-coordinate of each point
606
607     P_EE_x = np.array([DataPoints_def_EE[nr].def_EE_location[0] for nr in range(
            len(DataPoints_def_EE))])*1000    # Accessing the x-coordinate of each
            point and making it plottable by arraying it
608     P_EE_y = np.array([DataPoints_def_EE[nr].def_EE_location[1] for nr in range(
            len(DataPoints_def_EE))])*1000    # Accessing the y-coordinate of each
            point
609     P_EE_z = np.array([DataPoints_def_EE[nr].def_EE_location[2] for nr in range(
            len(DataPoints_def_EE))])*1000    # Accessing the z-coordinate of each
            point
610
611     colour_tetra = ['purple', 'darkred', 'darkcyan', 'deepskyblue', 'lightcoral',
            'forestgreen', 'dodgerblue', 'darkviolet', 'darkslategray', '
            mediumspringgreen', 'crimson', 'mediumseagreen', 'darkturquoise', '
            mediumorchid' , 'royalblue', 'darkolivegreen', 'tomato', 'mediumblue', '
            mediumaquamarine']
612     min_value = -0.11*1000 #in mm
613     max_value = 0.11*1000 #in mm
614
615     # Create a 3D plot
616     fig_xy = plt.figure(figsize=(10, 10))
617     fig_3d = plt.figure(figsize=(12, 12))
618     fig_xz = plt.figure(figsize=(10, 10))
619     fig_yz = plt.figure(figsize=(10, 10))
620
621     ax_xy = fig_xy.add_subplot(111)  # X-Y plane (2D)
622     ax_3d = fig_3d.add_subplot(111, projection='3d')
623     ax_xz = fig_xz.add_subplot(111)  # X-Z plane (2D)
624     ax_yz = fig_yz.add_subplot(111)  # Y-Z plane (2D)
625
626     P_0_coords = (0, 0, 0)
627     ax_3d.scatter(P_0_coords[0], P_0_coords[1], P_0_coords[2], c='red', marker='o'
            , label=r'P$_{0}$', s = 50)
628     ax_xz.scatter(P_0_coords[0], P_0_coords[2], c='red', marker='o', label=r'P$_{0
            }$', s = 50)
```

```
629     ax_yz.scatter(P_0_coords[0], P_0_coords[2], c='red', marker='o', label=r'P$_{0
            }$', s = 50)
630     #ax_xy.scatter(P_0_coords[0], P_0_coords[2], c='red', marker='o', label=r'P$_{
            0}$', s = 50)
631
632     # Function to compute the coordinates of a point in a reference frame
633     for i in range(len(P_a1)):
634         tetra_plotter(P_a1, P_a3, P_b1, P_b3, P_c1, P_c3, M_c, P_0, T_joint, ax_3d
                , ax_xz, ax_yz, ax_xy, colour_tetra, i)
635
636     if prototype_val_job == True:
637         prototype_val_plotter(ax_xy, fig_xy, radius, H_p_range_d, L_w, L_r, M_w,
                M_r, g)
638
639     if TetraFEM_val_job == True:
640         TetraFEM_val_plotter(ax_xy, fig_xy, radius, H_p_range_d, L_w, L_r, M_w,
                M_r, g)
641
642     ax_3d.set_xlim([min_value, max_value])
643     ax_3d.set_ylim([min_value, max_value])
644     ax_3d.set_zlim([0, 2*max_value])
645     ax_3d.scatter(P_EE_x, P_EE_y, P_EE_z, c='orange', marker='o')
646     ax_3d.scatter(P_goal_x, P_goal_y, P_goal_z, c='blue', marker='o')
647
648     fontsize_labels = 18
649     fontsize_ticks = 15
650     fontsize_legend = 18
651
652     # Set labels for the axes
653     #ax_3d.set_xlabel(r'X$_{JF}$ [mm]', fontsize=fontsize_labels)
654     #ax_3d.set_ylabel(r'Y$_{JF}$ [mm]', fontsize=fontsize_labels)
655     #ax_3d.set_zlabel(r'Z$_{JF}$ [mm]', fontsize=fontsize_labels)
656
657     ax_xz.scatter(P_goal_x, P_goal_z, c='blue', marker='o', label = r'P$_{goal}$')
658     ax_xz.scatter(P_EE_x, P_EE_z, c='orange', marker='o', label = r'P$_{EE}$')
659     ax_xz.set_xlabel(r'X$_{JF}$ [mm]', fontsize=fontsize_labels)
660     ax_xz.set_ylabel(r'Z$_{JF}$ [mm]', fontsize=fontsize_labels)
661     ax_xz.set_ylim([0, max_value])
662     ax_xz.axis('equal')
663
664     ax_yz.scatter(P_goal_y, P_goal_z, c='blue', marker='o', label = r'P$_{goal}$')
665     ax_yz.scatter(P_EE_y, P_EE_z, c='orange', marker='o', label = r'P$_{EE}$')
666
667     ax_yz.set_xlabel(r'Y$_{JF}$ [mm]', fontsize=fontsize_labels)
668     ax_yz.set_ylabel(r'Z$_{JF}$ [mm]', fontsize=fontsize_labels)
669     ax_yz.set_ylim([min_value, max_value])
670     ax_yz.axis('equal')
671
672     if prototype_val_job == False:
673         if TetraFEM_val_job == False:
674             ax_xy.scatter(P_goal_x, P_goal_y, c='blue', marker='o', label = r'P$_{
                    goal}$')
675         ax_xy.scatter(P_EE_x, P_EE_y, c='orange', marker='o', label = r'P$_{EE}$')
676
677     ax_xy.set_xlabel(r'X$_{JF}$ [mm]', fontsize=fontsize_labels)
678     ax_xy.set_ylabel(r'Y$_{JF}$ [mm]', fontsize=fontsize_labels)
679     ax_xy.set_ylim([min_value, max_value])
680     ax_xy.axis('equal')
681
682     ax_3d.tick_params(axis='both', which='major', labelsize=fontsize_ticks)
683     ax_3d.tick_params(axis='z', which='major', labelsize=fontsize_ticks)
```

```
684
685     ax_xz.tick_params(axis='both', which='major', labelsize=fontsize_ticks)
686     ax_yz.tick_params(axis='both', which='major', labelsize=fontsize_ticks)
687     ax_xy.tick_params(axis='both', which='major', labelsize=fontsize_ticks)
688
689
690     #Add a legend
691     ax_3d.legend(bbox_to_anchor=(0.9, 0.55), prop={'size': fontsize_legend})
692     ax_xz.legend(prop={'size': fontsize_legend})
693
694     #ax_xy.legend(prop={'size': fontsize_legend})
695     ax_yz.legend(prop={'size': fontsize_legend})
696
697     # Show the 3D plot
698     plt.show()
699     return
700
701
702 def radius_calc(R_in, R_w, t_min):
703     t_max = ((R_in[-1]+R_w[-1])/R_in[-1])*t_min[-1]
704     R_out = R_in[-1]+R_w[-1]
705     radius = R_out - R_w[-1]*(t_max+2*t_min[-1])/(3*(t_max+t_min[-1]))
706     return radius
707
708
709 def P_goal_calculator(radius, H_polar_val, W_polar_val, T_joint, P_0):
710
711     P_goal = me.Point('P_goal')
712     x = radius*np.sin(H_polar_val)*np.cos(W_polar_val-np.pi/2)
713     y = radius*np.sin(H_polar_val)*np.sin(W_polar_val-np.pi/2)
714     z = radius*np.cos(H_polar_val)
715     P_goal.set_pos(P_0, x*T_joint.x + y*T_joint.y + z*T_joint.z)
716     P_goal_loc = P_goal.pos_from(P_0).to_matrix(T_joint)
717     return P_goal_loc
718
719
720 def deflection_function(Tetra_outputs, M_w, M_r, g, Mx, My, P_0, T0, T_joint,
    gamma_d, alpha_d, radius, P_AC_prev_com, P_AB_prev_com, P_BC_prev_com):
721
722     n_tetra = np.size(alpha_d)
723
724     #self-weight
725     W_grav_P_AC = []
726     W_grav_P_df = []
727
728     P_AC_next_com = [P_AC_prev_com[0]]
729     P_AB_next_com = [P_AB_prev_com[0]]
730     P_BC_next_com = [P_BC_prev_com[0]]
731
732     for i in range(n_tetra):
733         W_grav_P_AC.append(np.array([[0], [0], [-Tetra_outputs[i].Mass_AC *g],
                [0], [0], [0]]))
734         W_grav_P_df.append(np.array([[0], [0], [-Tetra_outputs[i].Mass_df *g],
                [0], [0], [0]]))
735     #self-weight
736
737     W_pendulum = np.array([[0], [0], [-(M_w+M_r)*g], [Mx], [My], [0]])
738     AC_a_start = me.ReferenceFrame('AC_a_start')
739     AC_c_start = me.ReferenceFrame('AC_c')
740     AC_a_start.orient_axis(T0, (-gamma_d[0]*np.pi/180), T0.y)
741     AC_c_start.orient_axis(AC_a_start, (-alpha_d[0]*np.pi/180), AC_a_start.z)
```

```
742     AC_c_next = AC_c_start
743
744     M_c_start = me.Point('M_c_start')
745
746     R_Mo = radius
747     M_c_start.set_pos(P_0, R_Mo*AC_c_start.y)
748     M_c_next = M_c_start
749
750     for i in range(n_tetra):                                #For-loop for calculating
            deflection in every tetra
751
752         M_c_undef = M_c_next                            #The undeformed M_c of tetra
                [i] is equal to the M_c_next of tetra[i-1]
753         AC_c_undef = AC_c_next                          #The undeformed AC_c of
                tetra[i] is equal to the AC_c_next of tetra[i-1]
754         W_i_mass = 0
755         #self-weight
756         for k in range(n_tetra-i):
757             T_mat_mass_AC = T_matrix_calculator(M_c_undef, P_AC_prev_com[k+i],
                    AC_c_undef, T_joint)
758             T_mat_mass_AB = T_matrix_calculator(M_c_undef, P_AB_prev_com[k+i],
                    AC_c_undef, T_joint)
759             T_mat_mass_BC = T_matrix_calculator(M_c_undef, P_BC_prev_com[k+i],
                    AC_c_undef, T_joint)
760             W_i_mass += np.dot(T_mat_mass_AC, W_grav_P_AC[i+k]) + np.dot(
                    T_mat_mass_AB, W_grav_P_df[i+k]) + np.dot(T_mat_mass_BC,
                    W_grav_P_df[i+k])
761         #self-weight
762
763         T_matrix_force_sym = T_matrix_calculator(M_c_undef, P_0, AC_c_undef,
                T_joint)
764         W_i_pendulum = np.dot(T_matrix_force_sym, W_pendulum)

            #Calculating the force/moment acting on M_c in ref-frame AC_c of tetra[
            i]
765         ds = np.dot(Tetra_outputs[i].a_C_c, W_i_pendulum + W_i_mass)

            #The local displacement of the specific tetra[i] in reference frame
            AC_c of tetra[i]
766
767         M_c_def = me.Point('M_c_def')
768         M_c_def.set_pos(M_c_undef, float(ds[0])*AC_c_undef.x + float(ds[1])*
                AC_c_undef.y + float(ds[2])*AC_c_undef.z)
                                                    #The location of M_c in the deformed
                 tetra[i]
769
770         AC_c_def = me.ReferenceFrame('AC_c_def')
771         AC_c_def.orient_space_fixed(AC_c_undef, (float(ds[3]), float(ds[4]), float
                (ds[5])), '123')
            #The orientation of AC_c in the deformed tetra[i]
772
773         if i < (n_tetra-1):

            #For every tetra applicable, except the last one:
774             M_a_next = M_c_def
775
776             AC_a_next = me.ReferenceFrame('AC_a_next')
777             AC_a_next.orient_axis(AC_c_def, (-gamma_d[i+1]*np.pi/180), AC_c_def.y)
                                                                    #Orientation of
                    AC_a_next is AC_c but rotated by gamma[i+1]
778
```

```
779             AC_c_next = me.ReferenceFrame('AC_c_next_frame')
780             AC_c_next.orient_axis(AC_a_next, (-alpha_d[i+1]*np.pi/180), AC_a_next.
                    z)                                                  #Orientation of
                    AC_c_next is AC_a_next but rotated by alpha[i+1]
781
782             r_num = (Tetra_outputs[i+1].M_c).pos_from(Tetra_outputs[i+1].M_a).
                    to_matrix(Tetra_outputs[i+1].AC_a)
                                                    #All from the undeformed joint:
                    The distance between M_a and M_c, expressed in AC_c.
783             M_c_next = me.Point('M_c_next')
784             M_c_next.set_pos(M_a_next, float(r_num[0])*AC_a_next.x + float(r_num
                    [1])*AC_a_next.y + float(r_num[2])*AC_a_next.z)     #The distance
                    is same as tetra[i+1] is not yet deformed, but needs to be
                    expressed in AC_c_next to get M_c_next.
785
786             #self-weight
787             AC_m_next = me.ReferenceFrame('AC_m_next')
788             AC_m_next.orient_axis(AC_a_next, (-alpha_d[i+1]/2*np.pi/180),
                    AC_a_next.z)
789             P_AC_next_com_point = me.Point('P_AC_next_com_point')
790             P_AC_next_com_point.set_pos(P_0, Tetra_outputs[i+1].R_AC_com*AC_m_next
                    .y)
791             P_AC_next_com.append(P_AC_next_com_point)
792
793             AB_a_next = me.ReferenceFrame('AB_a_next')
794             AB_a_next.orient_axis(AC_a_next, -Tetra_outputs[i+1].phi, AC_a_next.y)
795             AB_com_next = me.ReferenceFrame('AB_com_next')
796             AB_com_next.orient_axis(AB_a_next, -Tetra_outputs[i+1].theta/2,
                    AB_a_next.z)
797             P_AB_next_com_point = me.Point('P_AB_next_com_point')
798             P_AB_next_com_point.set_pos(P_0, Tetra_outputs[i+1].R_df_com*
                    AB_com_next.y)
799             P_AB_next_com.append(P_AB_next_com_point)
800
801             BC_c_next = me.ReferenceFrame('BC_c_next')
802             BC_c_next.orient_axis(AC_c_next, Tetra_outputs[i+1].phi, AC_c_next.y)
803             BC_com_next = me.ReferenceFrame('BC_com_next')
804             BC_com_next.orient_axis(BC_c_next, Tetra_outputs[i+1].theta/2,
                    BC_c_next.z)
805             P_BC_next_com_point = me.Point('P_BC_next_com_point')
806             P_BC_next_com_point.set_pos(P_0, Tetra_outputs[i+1].R_df_com*
                    BC_com_next.y)
807             P_BC_next_com.append(P_BC_next_com_point)
808             #self-weight
809
810
811
812     EE_deflection = M_c_def.pos_from(Tetra_outputs[n_tetra-1].M_c).to_matrix(
            T_joint)
813     def_EE_frame = AC_c_def
814     def_EE_point = M_c_def
815     #Point_def_EE = M_c_def
816     return def_EE_point, def_EE_frame, W_pendulum, EE_deflection, P_AC_next_com,
            P_AB_next_com, P_BC_next_com
817
818
819 def deflection_function_nsw(Tetra_outputs, M_w, M_r, g, Mx, My, P_0, T0, T_joint,
        gamma_d, alpha_d, radius, P_AC_prev_com, P_AB_prev_com, P_BC_prev_com):
820
821     n_tetra = np.size(alpha_d)
822
```

```
823     P_AC_next_com = [P_AC_prev_com[0]]
824     P_AB_next_com = [P_AB_prev_com[0]]
825     P_BC_next_com = [P_BC_prev_com[0]]
826 
827     W_pendulum = np.array([[0], [0], [-(M_w+M_r)*g], [Mx], [My], [0]])
828     AC_a_start = me.ReferenceFrame('AC_a_start')
829     AC_c_start = me.ReferenceFrame('AC_c')
830     AC_a_start.orient_axis(T0, (-gamma_d[0]*np.pi/180), T0.y)
831     AC_c_start.orient_axis(AC_a_start, (-alpha_d[0]*np.pi/180), AC_a_start.z)
832     AC_c_next = AC_c_start
833 
834     M_c_start = me.Point('M_c_start')
835 
836     R_Mo = radius
837     M_c_start.set_pos(P_0, R_Mo*AC_c_start.y)
838     M_c_next = M_c_start
839 
840     for i in range(n_tetra):                                #For-loop for calculating
            deflection in every tetra
841 
842         M_c_undef = M_c_next                            #The undeformed M_c of tetra
                [i] is equal to the M_c_next of tetra[i-1]
843         AC_c_undef = AC_c_next                          #The undeformed AC_c of
                tetra[i] is equal to the AC_c_next of tetra[i-1]
844 
845         T_matrix_force_sym = T_matrix_calculator(M_c_undef, P_0, AC_c_undef,
                T_joint) #Dit zou goed moeten zijn!
846         W_i_pendulum = np.dot(T_matrix_force_sym, W_pendulum)
847 
                #Calculating the force/moment acting on M_c in ref-frame AC_c of tetra[
                i]
            ds = np.dot(Tetra_outputs[i].a_C_c, W_i_pendulum)# + W_i_mass)
848 
                #The local displacement of the specific tetra[i] in reference frame
                AC_c of tetra[i]
849         M_c_def = me.Point('M_c_def')
850         M_c_def.set_pos(M_c_undef, float(ds[0])*AC_c_undef.x + float(ds[1])*
                AC_c_undef.y + float(ds[2])*AC_c_undef.z)
                                                #The location of M_c in the deformed
                 tetra[i]
851 
852         AC_c_def = me.ReferenceFrame('AC_c_def')
853         AC_c_def.orient_space_fixed(AC_c_undef, (float(ds[3]), float(ds[4]), float
                (ds[5])), '123')
                #The orientation of AC_c in the deformed tetra[i]
854 
855         if i < (n_tetra-1):
856 
                #For every tetra applicable, except the last one:
                M_a_next = M_c_def
857 
858             AC_a_next = me.ReferenceFrame('AC_a_next')
859             AC_a_next.orient_axis(AC_c_def, (-gamma_d[i+1]*np.pi/180), AC_c_def.y)
                                                                    #Orientation of
                    AC_a_next is AC_c but rotated by gamma[i+1]
860 
861             AC_c_next = me.ReferenceFrame('AC_c_next_frame')
862             AC_c_next.orient_axis(AC_a_next, (-alpha_d[i+1]*np.pi/180), AC_a_next.
                    z)                                          #Orientation of
                    AC_c_next is AC_a_next but rotated by alpha[i+1]
```

```
863
864             r_num = (Tetra_outputs[i+1].M_c).pos_from(Tetra_outputs[i+1].M_a).
                    to_matrix(Tetra_outputs[i+1].AC_a)
                                                    #All from the undeformed joint:
                    The distance between M_a and M_c, expressed in AC_c.
865             M_c_next = me.Point('M_c_next')
866             M_c_next.set_pos(M_a_next, float(r_num[0])*AC_a_next.x + float(r_num
                    [1])*AC_a_next.y + float(r_num[2])*AC_a_next.z)     #The distance
                    is same as tetra[i+1] is not yet deformed, but needs to be
                    expressed in AC_c_next to get M_c_next.
867
868     EE_deflection = M_c_def.pos_from(Tetra_outputs[n_tetra-1].M_c).to_matrix(
            T_joint)
869     def_EE_frame = AC_c_def
870     def_EE_point = M_c_def
871     #Point_def_EE = M_c_def
872     return def_EE_point, def_EE_frame, W_pendulum, EE_deflection, P_AC_next_com,
            P_AB_next_com, P_BC_next_com
873
874
875 class def_EE_data:
876     def __init__(self, def_EE_location, W_pendulum, P_goal_loc, distance_norm):
877         self.def_EE_location = def_EE_location
878         self.W_pendulum = W_pendulum
879         self.P_goal_loc = P_goal_loc
880         self.distance_norm = distance_norm
881
882
883 def Def_EE_calc(H_p_range_d, W_p_range_d, sample_density, self_weight_iterations,
        g, M_w, L_w, M_r, L_r, Tetra_outputs, P_0, T0, T_joint, gamma_d, alpha_d,
        radius):
884
885     DataPoints_def_EE = []
886     point_counter = 0
887
888     P_AC_prev_com = []
889     P_AB_prev_com = []
890     P_BC_prev_com = []
891
892     d_def_coll = []
893
894     M_max = g*M_w*L_w + g*M_r*(L_r/2)
895
896     H_p_range = [H_p_range_d[0]*np.pi/180, H_p_range_d[1]*np.pi/180]
897     W_p_range = [W_p_range_d[0]*np.pi/180, W_p_range_d[1]*np.pi/180]
898
899     Mass_total = 0
900     for i in range(len(alpha_d)):
901         Mass_total += Tetra_outputs[i].Mass_AC + 2*Tetra_outputs[i].Mass_df
902
903
904     for H_polar_val in np.linspace(H_p_range[0], H_p_range[1], sample_density,
            endpoint=True):
905         W_polar_samples = 1+int(sample_density*(np.sin(H_polar_val) * abs((
                W_p_range[1]-W_p_range[0])/(np.pi/2))))
906
907         if W_p_range[1]-W_p_range[0] == 2*np.pi:
908             end = False
909         else:
910             end = True
911
```

```
912         for W_polar_val in np.linspace(W_p_range[0], W_p_range[1], W_polar_samples
                , endpoint=end):
913             point_counter +=1
914             d_def = 2*np.sin(H_polar_val/2)*radius*1000
915             d_def_coll.append(d_def)
916 
917             Mx = np.cos(W_polar_val)*np.sin(H_polar_val)*M_max
918             My = np.sin(W_polar_val)*np.sin(H_polar_val)*M_max
919 
920             for i in range(len(alpha_d)):
921                 P_AC_prev_com.append(Tetra_outputs[i].P_AC_com)
922                 P_AB_prev_com.append(Tetra_outputs[i].P_AB_com)
923                 P_BC_prev_com.append(Tetra_outputs[i].P_BC_com)
924 
925             if self_weight_iterations>0:
926                 for it in range(self_weight_iterations-1):
927                     def_EE_point, def_EE_frame, W_pendulum, EE_deflection,
                            P_AC_prev_com, P_AB_prev_com, P_BC_prev_com =
                            deflection_function(Tetra_outputs, M_w, M_r, g, Mx, My, P_0
                            , T0, T_joint, gamma_d, alpha_d, radius, P_AC_prev_com,
                            P_AB_prev_com, P_BC_prev_com)
928 
929                 def_EE_point, def_EE_frame, W_pendulum, EE_deflection,
                        P_AC_next_com, P_AB_next_com, P_BC_next_com =
                        deflection_function(Tetra_outputs, M_w, M_r, g, Mx, My, P_0, T0
                        , T_joint, gamma_d, alpha_d, radius, P_AC_prev_com,
                        P_AB_prev_com, P_BC_prev_com)
930 
931             if self_weight_iterations==0:
932                 def_EE_point, def_EE_frame, W_pendulum, EE_deflection,
                        P_AC_next_com, P_AB_next_com, P_BC_next_com =
                        deflection_function_nsw(Tetra_outputs, M_w, M_r, g, Mx, My, P_0
                        , T0, T_joint, gamma_d, alpha_d, radius, P_AC_prev_com,
                        P_AB_prev_com, P_BC_prev_com)
933 
934             def_EE_location = def_EE_point.pos_from(P_0).to_matrix(T_joint)
935 
936             P_goal_loc = P_goal_calculator(radius, H_polar_val, W_polar_val,
                    T_joint, P_0)
937 
938             distance_vector = def_EE_location-P_goal_loc
939             distance_norm = (distance_vector[0]**2 + distance_vector[1]**2 +
                    distance_vector[2]**2)**0.5
940 
941             info_def_EE = def_EE_data(def_EE_location, W_pendulum, P_goal_loc,
                    distance_norm)
942             DataPoints_def_EE.append(info_def_EE)
943 
944 
945     d_def_avg = sum(d_def_coll)/len(d_def_coll)
946     print(f"The average deflected distance of the {len(d_def_coll)} P_goal points
            = {d_def_avg} mm")
947 
948     return DataPoints_def_EE, point_counter, Mass_total
949 
950 
951 def Def_EE_calc_TetraFEM_val(self_weight_iterations, g, M_w, M_r, L_r,
        Tetra_outputs, P_0, T0, T_joint, gamma_d, alpha_d, radius):
952     W_polar_range = []
953     H_polar_range = []
954     L_w_range = []
```

```
955
956     d_def_coll = []
957
958     for point in unstable_points:
959         W_polar_val = np.arctan(point[1]/point[0])
960         H_polar_val = np.arcsin(point[0]/(np.cos(W_polar_val)*(radius*1000)))
961         W_polar_range.append(W_polar_val+np.pi/2)
962         H_polar_range.append(H_polar_val)
963         L_w_range.append(point[4]/1000)
964
965         d_def = (point[0]**2+point[1]**2+(point[2]-radius*1000)**2)**0.5
966         d_def_coll.append(d_def)
967
968     for point in stable_points:
969         W_polar_val = np.arctan(point[1]/point[0])
970         H_polar_val = np.arcsin(point[0]/(np.cos(W_polar_val)*(radius*1000)))
971         W_polar_range.append(W_polar_val+np.pi/2)
972         H_polar_range.append(H_polar_val)
973
974         L_w_range.append(point[4]/1000)
975         d_def = (point[0]**2+point[1]**2+(point[2]-radius*1000)**2)**0.5
976         d_def_coll.append(d_def)
977
978
979     d_def_avg = sum(d_def_coll)/len(d_def_coll)
980     print(f"The average deflected distance of the {len(d_def_coll)} P_goal points
        = {d_def_avg} mm")
981
982     DataPoints_def_EE = []
983     point_counter = 0
984
985     P_AC_prev_com = []
986     P_AB_prev_com = []
987     P_BC_prev_com = []
988
989     Mass_total = 0
990     for i in range(len(alpha_d)):
991         Mass_total += Tetra_outputs[i].Mass_AC + 2*Tetra_outputs[i].Mass_df
992
993     for k in range(len(H_polar_range)):
994         H_polar_val = H_polar_range[k]
995         W_polar_val = W_polar_range[k]
996         L_w = L_w_range[k]
997
998         M_max = g*M_w*L_w + g*M_r*(L_r/2)
999
1000        point_counter +=1
1001        Mx = np.cos(W_polar_val)*np.sin(H_polar_val)*M_max
1002        My = np.sin(W_polar_val)*np.sin(H_polar_val)*M_max
1003
1004        for i in range(len(alpha_d)):
1005            P_AC_prev_com.append(Tetra_outputs[i].P_AC_com)
1006            P_AB_prev_com.append(Tetra_outputs[i].P_AB_com)
1007            P_BC_prev_com.append(Tetra_outputs[i].P_BC_com)
1008
1009        if self_weight_iterations>0:
1010            for it in range(self_weight_iterations-1):
1011                def_EE_point, def_EE_frame, W_pendulum, EE_deflection,
                        P_AC_prev_com, P_AB_prev_com, P_BC_prev_com =
                        deflection_function(Tetra_outputs, M_w, M_r, g, Mx, My, P_0, T0
                        , T_joint, gamma_d, alpha_d, radius, P_AC_prev_com,
```

```
                    P_AB_prev_com, P_BC_prev_com)
1012
1013            def_EE_point, def_EE_frame, W_pendulum, EE_deflection, P_AC_next_com,
                    P_AB_next_com, P_BC_next_com = deflection_function(Tetra_outputs,
                    M_w, M_r, g, Mx, My, P_0, T0, T_joint, gamma_d, alpha_d, radius,
                    P_AC_prev_com, P_AB_prev_com, P_BC_prev_com)
1014
1015        if self_weight_iterations==0:
1016            def_EE_point, def_EE_frame, W_pendulum, EE_deflection, P_AC_next_com,
                    P_AB_next_com, P_BC_next_com = deflection_function_nsw(
                    Tetra_outputs, M_w, M_r, g, Mx, My, P_0, T0, T_joint, gamma_d,
                    alpha_d, radius, P_AC_prev_com, P_AB_prev_com, P_BC_prev_com)
1017
1018        def_EE_location = def_EE_point.pos_from(P_0).to_matrix(T_joint)
1019
1020        P_goal_loc = P_goal_calculator(radius, H_polar_val, W_polar_val, T_joint,
                P_0)
1021        distance_vector = def_EE_location-P_goal_loc
1022        distance_norm = (distance_vector[0]**2 + distance_vector[1]**2 +
                distance_vector[2]**2)**0.5
1023        info_def_EE = def_EE_data(def_EE_location, W_pendulum, P_goal_loc,
                distance_norm)
1024        DataPoints_def_EE.append(info_def_EE)
1025
1026    return DataPoints_def_EE, point_counter, Mass_total
1027
1028
1029 def Pendulum_test(var_geo_vals, R_in, R_w, n_samples, E, G, density, c_shear,
        H_p_range_d, W_p_range_d, sample_density, self_weight_iterations, g, M_w, L_w,
        M_r, L_r):
1030    start_time = time.time()
1031
1032    global iteration  # Accessing the global iteration variable
1033    global starting_point
1034    iteration += 1
1035    display(" ")
1036    print(f"Iteration {iteration}")
1037    print(f"starting_point {starting_point}")
1038
1039    alpha_d = var_geo_vals[1:len(R_in)+1]
1040    gamma_d = var_geo_vals[len(R_in)+1:2*len(R_in)+1]
1041    t = var_geo_vals[0]
1042
1043    t_min = [t/(10**6)] *len(R_in)
1044
1045    Tetra_outputs, P_a1, P_a3, P_b1, P_b3, P_c1, P_c3, M_c, T_joint, T0, P_0 =
            Joint_calculations_slow(alpha_d, gamma_d, t_min, R_in, R_w, n_samples, E, G
            , density, c_shear)
1046
1047    radius = radius_calc(R_in, R_w, t_min)
1048
1049    if TetraFEM_val_job == True:
1050        DataPoints_def_EE, point_counter, Mass_total = Def_EE_calc_TetraFEM_val(
                self_weight_iterations, g, M_w, M_r, L_r, Tetra_outputs, P_0, T0,
                T_joint, gamma_d, alpha_d, radius)
1051    if TetraFEM_val_job == False:
1052        DataPoints_def_EE, point_counter, Mass_total = Def_EE_calc(H_p_range_d,
                W_p_range_d, sample_density, self_weight_iterations, g, M_w, L_w, M_r,
                L_r, Tetra_outputs, P_0, T0, T_joint, gamma_d, alpha_d, radius)
1053
1054    if Optimization_job == False:
```

```
        Joint_Sphere_plotter(DataPoints_def_EE, P_a1, P_a3, P_b1, P_b3, P_c1, P_c3
            , M_c, P_0, T_joint, radius, H_p_range_d, L_w, L_r, M_w, M_r, g)

    distance_norm_sum = 0

    if sample_density > 0:
        for i in range(point_counter):
            distance_norm_sum += DataPoints_def_EE[i].distance_norm
        distance_norm_avg = distance_norm_sum/point_counter
        print("The average distance between P_goal and P_EE is ",
            distance_norm_avg *1000, " mm")
    else:
        distance_norm_avg = 0

    #print("Total mass = ", Mass_total)

    print("var_geo_vals = ", var_geo_vals)

    opt_comp_time = time.time()-start_time
    print("opt_comp_time = ", opt_comp_time)

    return distance_norm_avg



### CONTROL PANEL ###

n_tetra = 10

#Fixed geometric parameters
R_in = [6.7e-2] *n_tetra            #[m] Inner radius
R_w = [2.5e-2] *n_tetra           #[m] Difference inner and outer radius

#Material parameters
E = 1700*10**6              #[Pa] Youngs modulus
poisson = 0.38
G = E/(2*(1+poisson))             #[Pa] Shear modulus
density = 1010 #kg/m^3
c_shear = 5/6               #[-] Shear coefficient

#Determines the accuracy of the numerical calculations
n_samples = 200

#Amount of self-weight iterations
self_weight_iterations = 0        #zero for no self-weight, two for accurate self-
    weight

#Forces parameters
g = 9.81
M_w = 0.1 #kg
L_w = 0.25 #m
M_r = 0.0235
L_r = 0.320

#P_goal points parameters
sample_density = 3 #higher is more points
H_p_range_d = [0, 30]               #polar height range
W_p_range_d = [0, 360]              #polar width, also range around JF.z

#TetraFEM_validation_job
TetraFEM_val_job = False
```

```
1113
1114 #prototype_validation job
1115 prototype_val_job = False
1116
1117 stable_points = [
1118     #[-12.6, -12.2, 77.0, 13.3, 200], [-23.7, -13.9, 94.8, 18.9, 200], [-16.4,
            -24.1, 73.3, 33.7, 200], [-26.8, -10.8, 73.6, 22.1, 200],
1119     #[-16.5, -8.7, 76.8, 13.9, 202.5], [-31.2, 3.6, 72.6, 23.7, 202.5], [-29.9,
            11.5, 72.0, 24.2, 202.5], [-24.7, -11.0, 74.2, 20.5, 202.5], [-22.8, -15.8,
             74.0, 21.3, 202.5],
1120     [-24.4, 4.1, 75.1, 18.4, 205], [-14.2, -28.3, 72.3, 24.7, 205], [-32.3, -1.4,
            72.1, 24.8, 205], [-37.9, 5.4, 69.2, 29.4, 205],  #TetraFEM_val_job
1121     #[-26.0, -2.3, 74.7, 19.5, 207.5], [-29.3, -9.1, 72.9, 23.4, 207.5],
1122     #[-20.3, -11.5, 75.8, 17.5, 210], [-33.0, 4.8, 71.9, 25.2, 210], [-24.9,
            -21.1, 72.3, 25.1, 210], [-27.9, -8.1, 73.6, 22.2, 210], [-38.2, 6.2, 68.9,
             29.8, 210],
1123     [-30.1, 1.3, 73.3, 22.7, 212.5], [-38.1, 7.9, 68.8, 29.9, 212.5], #
            TetraFEM_val_job
1124     #[-37.7, 2.6, 68.9, 29.3, 215], [-32.2, 5.8, 72.0, 24.9, 215],
1125     #[-34.6, -9.3, 70.2, 27.6, 217.5], [-38.0, -10.6, 68.2, 30.8, 217.5],
1126     [-30.9, -8.5, 72.0, 24.7, 220], [-39.5, 0.8, 67.9, 31.0, 220], #
            TetraFEM_val_job
1127     [-48.0, 20.6, 58.7, 42.9, 235] #TetraFEM_val_job
1128 ]
1129
1130 unstable_points = [
1131     #[-28.5, -15.7, 72.3, 25.0, 222.5],
1132     #[-21.7, -11.7, 75.0, 18.8, 225], [-26.9, -12.7, 73.0, 22.8, 225],
1133     [-32.9, -20.2, 69.0, 30.3, 227.5], #TetraFEM_val_job
1134     #[-33.7, -17.4, 69.1, 29.8, 230],
1135     #[-37.8, -18.8, 66.6, 33.6, 232.5],
1136     [-32.3, -15.5, 70.2, 28.0, 235], #TetraFEM_val_job
1137     #[23.0, 12.3, 74.4, 19.6, 245],
1138     #[18.6, 16.6, 74.8, 18.6, 247.5], [22.3, 18.4, 73.4, 21.6, 247.5],
1139     #[11.8, 14.4, 76.9, 13.7, 267.5],
1140     [11.3, 11.5, 77.5, 11.8, 265], [13.8, 18.2, 75.7, 16.9, 265], #
            TetraFEM_val_job
1141     #[14.3, 14.2, 76.5, 14.9, 262.5], [12.5, 15.1, 76.6, 14.5, 262.5], [9.5, 11.6,
             77.7, 11.1, 262.5], [18.3, 25.7, 72.3, 23.9, 262.5],
1142     #[13.1, 12.9, 76.9, 13.6, 260], [11.8, 9.9, 77.6, 11.4, 260], [20.8, 26.4,
            71.2, 25.8, 260],
1143     [13.2, 13.5, 76.8, 13.7, 257.5], [21.6, 37.4, 65.8, 33.6, 257.5], #
            TetraFEM_val_job
1144     #[11.9, 19.5, 75.7, 16.8, 255], [13.6, 15.2, 76.6, 15.0, 255], [15.7, 41.4,
            65.2, 34.3, 255], [13.1, 10.8, 77.4, 12.6, 255],
1145     #[20.2, 17.2, 74.5, 19.9, 252.5], [12.5, 15.1, 76.8, 14.5, 252.5], [14.8,
            29.6, 71.7, 24.9, 252.5],
1146     [26.0, 22.6, 71.0, 26.3, 250], [16.1, 16.3, 75.8, 17.1, 250] #TetraFEM_val_job
1147 ]
1148
1149 #Optimization parameters
1150 iteration = 0
1151 starting_point = 0
1152 max_iterations = 1000
1153 tol_var_geo_vals = 0.5
1154 min_geo_vals = [1000, 5, -90]
1155 max_geo_vals = [1800, 60, 35]
1156 step_geo_vals = 5
1157 Optimization_job = False
1158
1159
```

```
if Optimization_job == False:

    #Result BBO3V2_tien_tetras2
    alpha_d = [16.30, 27.74, 56.64, 22.24, 43.66, 43.02, 31.80, 15.71, 31.40,
        24.23]
    gamma_d = [0, -39.13, 7.83, -22.22, -44.36, -22.58, -38.04, -40.34, -83.87,
        -2.07]
    t = [1290.61]

    var_geo_vals = t + alpha_d + gamma_d
    distance_norm_avg = Pendulum_test(var_geo_vals, R_in, R_w, n_samples, E, G,
        density, c_shear, H_p_range_d, W_p_range_d, sample_density,
        self_weight_iterations, g, M_w, L_w, M_r, L_r)


if Optimization_job == True:
    constraints = [
        {'type': 'ineq', 'fun': lambda x: x[0] - min_geo_vals[0]},   # Lower bound
            for t
        {'type': 'ineq', 'fun': lambda x: max_geo_vals[0] - x[0]},   # Upper bound
            for t
        {'type': 'ineq', 'fun': lambda x: x[1:n_tetra+1] - min_geo_vals[1]},   #
            Lower bound for alpha_d
        {'type': 'ineq', 'fun': lambda x: max_geo_vals[1] - x[1:n_tetra+1]},   #
            Upper bound for alpha_d
        {'type': 'ineq', 'fun': lambda x: x[n_tetra+1:2*n_tetra+1] - min_geo_vals
            [2]},  # Lower bound for gamma_d
        {'type': 'ineq', 'fun': lambda x: max_geo_vals[2] - x[n_tetra+1:2*n_tetra
            +1]},  # Upper bound for gamma_d
    ]

    n_starting_points = 1

    for n in range(n_starting_points):
        iteration = 0
        t = np.random.uniform(min_geo_vals[0], max_geo_vals[0], size=(1,))
        alpha_d = np.random.uniform(min_geo_vals[1], max_geo_vals[1], size=(
            n_tetra,))
        gamma_d = np.random.uniform(min_geo_vals[2], max_geo_vals[2], size=(
            n_tetra,))
        var_geo_vals = np.concatenate((t, alpha_d, gamma_d))
        print(var_geo_vals)


        starting_point +=1
        # Perform optimization
        Optimized_geo_vals = optimize.minimize(
            Pendulum_test, var_geo_vals,
            args=(R_in, R_w, n_samples, E, G, density, c_shear, H_p_range_d,
                W_p_range_d, sample_density, self_weight_iterations, g, M_w, L_w,
                M_r, L_r),
            method='COBYLA',
            tol = tol_var_geo_vals,
            options={'maxiter': max_iterations, 'rhobeg': step_geo_vals},
            constraints=constraints
        )
        print(Optimized_geo_vals)
```