

Department of Precision and Microsystems Engineering

Topology Optimization Algorithm For Synthesis of Dynamically Balanced Compliant Mechanisms

NoI Römer

Report no : 2023.094
Coach : Dr. Ir. V. van der Wijk
Professor : Prof. Dr. Ir. J.L. Herder
Specialisation : Mechatronic System Design
Type of report : MSc. Thesis
Date : November 30th 2023

Preface

When I started the bachelor of Mechanical Engineering at 3mE, I remember one of the first lectures where our lecturer showed us a mechanism that resulted from a topology optimization run. This mechanism was a compliant cutter, with an output direction which was rotated 90 degrees with respect to the input direction, and where the entire function of the mechanism resulted from calculated deformation of a random looking lump of material. I was both amazed that a computer could design such a beautiful mechanism, and inspired that a normal person, just like myself, could have made said computer do that. Even though topology optimization was never a part of the Mechanical Engineering study program, this impression never left me.

Compliant mechanisms are a well-known topic in the MSc program, however. These mechanisms intrigued me by their simplicity, while being so powerful in terms of predictability and effectiveness. At the same time I was getting acquainted with these mechanisms, the subjects of static and dynamic balancing came to light during one of the courses. Static balancing first grabbed my attention, but this still unknown principle of dynamic balancing appeared to be the holy grail. That, I wanted to get an expert on. However, topology optimization never left my thoughts as being one of the most powerful ways of designing mechanisms.

Topology optimization and Dynamic balancing are two very niche subjects, and before I started working on this project, I knew little about either of them. I'm therefore very grateful to Volkert for his coaching in terms of conducting research and his guidance in the subject of Dynamic Balancing. However, I never expected the field of Topology Optimization to be as difficult and unforgiving as I experienced throughout the duration of this project. My eternal gratitude goes to Dirk for his advice on the difficult TopOpt problems I encountered, the interesting discussions and his great support on the subject.

Before you lies the product of a long journey through the fields of dynamic balancing, topology optimization and mechanism design. I'm proud to present this thesis to you, and to have contributed my part to bringing these two engineering disciplines together.

Contents

1	Introduction	1
1.1	Dynamic Balancing	1
1.2	Compliant Mechanisms	2
1.3	Research Objective	3
2	State Of The Art	5
2.1	Introduction	5
2.2	Compliant Mechanisms	6
2.2.1	What is a compliant mechanism?	6
2.2.2	Commonly used compliant mechanisms	7
2.2.3	Modelling and synthesis techniques for compliant mechanisms	10
2.3	Balancing Methods	14
2.3.1	Balancing conditions	15
2.3.2	Available balancing methods	16
2.4	Comparison of available methods	18
2.4.1	Compliant mechanisms	19
2.4.2	Dynamic balancing	19
2.5	Conclusion	20
3	Topology Optimization Algorithm for Synthesis of Dynamically Balanced Compliant Mechanisms	23
3.1	Introduction	23
3.2	Method	24
3.2.1	Used model and assumptions	24
3.2.2	Balance Conditions	25
3.2.3	Optimization problem	26
3.2.4	Design problem, starting conditions and stopping criteria	27
3.3	Results	29
3.3.1	Geometries	29
3.3.2	Verification and dynamic analysis	30
3.4	Discussion	33
3.5	Conclusion	34
4	Discussion	35
4.1	Strengths of the proposed topology optimization algorithm	35
4.2	Weaknesses of the proposed topology optimization algorithm	36
4.3	Recommendations and further research	36

5	Conclusion	39
A	Sensitivity calculations	45
A.1	Shaking Force Balance Constraint with respect to ρ	45
A.2	Shaking Force Balance Constraint with respect to $\tilde{\alpha}$	46
A.3	Shaking Moment Balance Constraint with respect to ρ	48
A.4	Shaking Moment Balance Constraint with respect to $\tilde{\alpha}$	49
B	Verification of sensitivity functions	51
C	Comsol analysis	53
C.1	Transformation from MATLAB to SolidWorks	53
C.2	COMSOL model	55
C.3	Frequency domain analysis	56
C.3.1	Balanced displacement inverter	56
C.3.2	Unbalanced displacement inverter	56
D	Other mechanisms	59
D.1	Diagonal Corners Mechanism	59
D.1.1	x-input,x-output,R=3	61
D.1.2	x-input,x-output,R=5	62
D.1.3	x-input,y-output,R=3	63
D.1.4	x-input,y-output,R=5	64
D.1.5	y-input,y-output,R=3	65
D.1.6	y-input,y-output,R=5	66
E	Matlab Code	67
E.1	Code overview	67
E.2	Inputs and options	67
E.3	Inputs	67
E.4	Options	68
E.5	More info	69
E.6	Matlab Code	70
E.7	topmma_balanced	70
E.8	SFbal	79
E.9	SMbal	80
E.10	dEEdx	81
E.11	topmma_unbalanced	81
E.12	FE	91
E.13	Situations	101
E.14	Fibs04x	101
E.15	Fibs04y	102
E.16	Fibs025x	103
E.17	Fibs025y	104
E.18	Fibs075x	105
E.19	Fibs075y	106
E.20	Fibsx	107
E.21	Fibsy	108
E.22	FTlbrt_xx	109
E.23	FTlbrt_xy	110

E.24 FTlbrt_yy 110

Chapter 1

Introduction

Machines are all around us in the everyday world. We use them for transportation, to do our work, wash our clothes, refrigerate or cook our food and to produce the products we use every day. Many of these machines are in fact mechanisms. A mechanism is a mechanical device used to transfer or transform motion, force or energy [1].

In the precision industry, mechanisms are required to perform jobs increasingly fast, precise and repeatable. To do this, many methods are developed and techniques are used. Two examples of such techniques are Dynamic Balancing and Compliant Mechanisms.

1.1 Dynamic Balancing

When a body is translated, a force is required to accelerate its mass. When the translation stops, the same force is required to decelerate that mass. These forces are called inertial forces or shaking forces. When the body is rotated, a similar phenomenon happens, but is then called shaking moment.

Shaking forces and shaking moments in mechanisms can be a significant force of base vibrations. These vibrations can be eliminated by designing the manipulator to be shaking-force balanced and shaking-moment balanced [3]. If a mechanism is shaking-force balanced and shaking-moment balanced, the mechanism is called dynamically balanced.

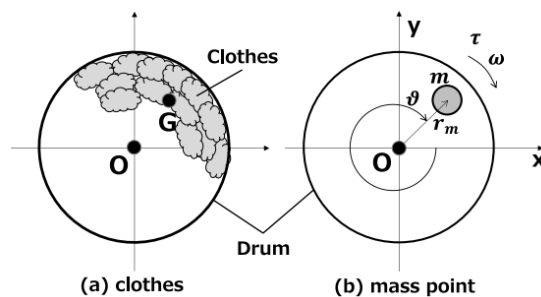


Figure 1.1: Single mass model of clothes rotating in a washing machine drum [2]

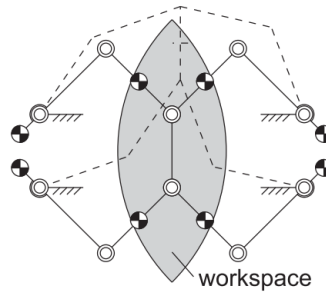


Figure 1.2: Schematic representation of the Dual-V manipulator [3]

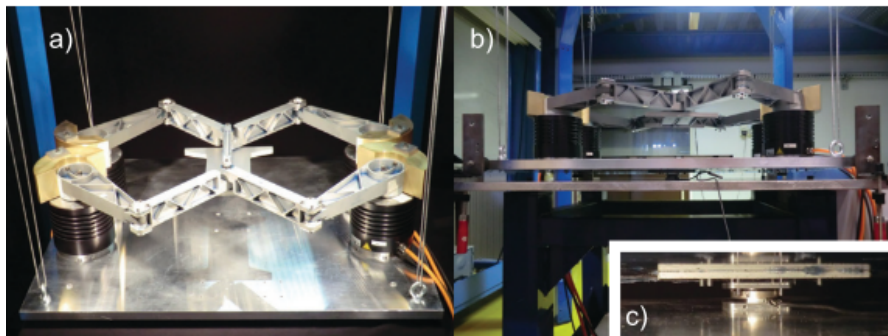


Figure 1.3: Prototype of the Dual-V manipulator [3]

An example of such base vibrations can be found in a common washing machine. When the washing machine starts its spin-dry process, the drum rotates at a high velocity. If the machine is filled with laundry, this causes the machine to shake heavily, causing loud noise. If the machine is empty, this phenomenon does not take place because the drum is dynamically balanced. When laundry is put into the machine, an imbalance is introduced, causing vibrations when the drum starts rotating. Mechanically, this can be modelled as shown in Figure 1.1. As shown in Figure 1.1b, the mass of the laundry is not at the center of rotation of the drum. This means the total center of mass of the system will move under rotation of the drum, causing shaking forces and thereby vibrations.

Another example of a dynamically balanced mechanism is the Dual V manipulator by van der Wijk [3]. This mechanism is designed to move a platform in x- and y-direction, and is shaking forced balanced and shaking moment balanced. The platform can also be rotated, but is in rotated case not as well-balanced anymore. In Figure 1.2, the design of the Dual-V manipulator is shown. In Figure 1.3, the prototype is shown.

1.2 Compliant Mechanisms

If something bends to do what it is meant to do, then it is compliant. If the flexibility that allows it to bend also helps it to accomplish something useful, then it is a compliant mechanism [4].

Compliant mechanisms have been around for a long time. An example of a compliant mechanism with a multi-millennia history is the bow and arrow [4]. Ancient bows used the flexibility of their

limbs to store energy to fire an arrow.

Also in present time, compliant mechanisms are all around us. The cap of a shampoo bottle is an example of a monolithic bi-stable hinge. In other words: a cap with a hinge and a cover, made of only one piece of material. A hairclip is another example so common people may not even recognise it as a compliant mechanism [4].

Compliant mechanisms are rapidly gaining importance [5]. They offer several advantages over rigid body mechanisms, such as fewer assembly process requirements, part-count reduction, increased precision, increased reliability and no need for lubrication [1], [6]. These can be considered into two main categories: cost reduction and increased performance.

This cost reduction explains why compliant mechanisms such as the shampoo bottle cap and the hairclip are quite common in everyday life. The increased precision and reliability, however, are reasons why compliant mechanisms are increasingly important in the engineering world.

1.3 Research Objective

Both compliant mechanism design and dynamic balancing are strong methodologies for achieving a high precision in mechanism. However, a comprehensive method for designing systems that are both dynamically balanced and compliant are not available yet. Most dynamic balancing methods consider rigid body linkages rather than compliant mechanisms. Some methods consider modal balancing, which is a technology for balancing harmonic vibrations in mechanisms[7], [8]. For large non-harmonically exited mechanisms, this method may however be insufficient.

Topology optimization is a strong design methodology which can be used to synthesize dynamically balanced compliant mechanisms. This methodology can already be used for designing unbalanced compliant mechanisms, but has in this thesis been extended with dynamic balance conditions.

The goal of this MSc. thesis project is formulated as follows:

“Develop a comprehensive method to synthesize dynamically balanced compliant mechanisms.”

Because many methods already exist to design dynamically balanced mechanisms or compliant mechanisms, the first step is to investigate the readily available methods and find a combination of such methods that may yield a comprehensive design methodology. The second step is to further investigate the most promising method and construct a design methodology for the synthesis of dynamically balanced compliant mechanisms. These two steps start with the following research question:

“Which currently existing methods seem most promising for designing dynamically balanced compliant mechanisms?”

A literature review is presented to answer this question in Chapter 2. The proposed methodology is a topology optimization algorithm, supplemented by the inherent balancing method by van der Wijk [9]. This algorithm is designed, explained, tested and discussed in Chapter 3. In Chapter 4 the approach is reviewed, and recommendations for further research and development of the algorithm are presented. The entire research project is summarized and wrapped up in Chapter 5.

Chapter 2

State Of The Art

2.1 Introduction

Machines have to fulfill increasingly high requirements every day. Machines are performing their tasks faster, more precise and more consistently than ever before, but the need for even higher performance is as high as ever. Especially in the precision industry the boundaries are extended rapidly, which requires new techniques and solutions.

In order to achieve a higher precision, the first goal is to reduce any noise and uncertainty. Two methods to do that are the use of compliant mechanisms and dynamic balancing. A mechanism is called compliant if its function is achieved due to the flexibility of the parts of the system, rather than relative movement of rigid parts with respect to each other. A system is called dynamically balanced when the inertial forces and moments of the system do not cause resulting forces on the base of the system. Both these methods are promising to achieve higher precision, but even though compliant mechanisms are increasingly common the combination of these two has not thoroughly been explored yet.

In this research paper, design methods and properties of both compliant mechanisms and dynamic balancing are investigated to look for similarities and possibilities to design dynamically balanced compliant mechanisms. The following question is to be answered:

Which currently existing methods seem most promising for designing dynamically balanced compliant mechanisms?

To answer this question, the following subquestions need to be answered first:

- Which compliant mechanisms are commonly used and which techniques of modelling or designing them are available?
- Which balancing methods are available?
- What are shortcomings of the available methods for balancing compliant mechanisms, and do they differ for various kinds of compliant mechanisms?

In section 2.2, 2.3 and 2.4 these subquestions will be answered respectively. In section 2.4, the comparison of these methods will also be summarized in order to answer the corresponding

subquestion. In chapter 2.5, the main research question is answered and a conclusion is presented. Also, recommendations for further research are given.

2.2 Compliant Mechanisms

Compliant mechanisms play an ever increasing role in the High-Tech industry. The combination of simplicity, predictability and absence of lubrication and play offer great advantages compared to general linkage systems[5], [10]. In this chapter, compliant mechanisms will be explained. After that, some common compliant mechanisms are shown for which dynamic balancing may be useful; for example compliant mechanisms that are used in the precision industry. Finally, methods of modelling and designing compliant mechanisms are treated.

2.2.1 What is a compliant mechanism?

Compliant mechanisms are defined as mechanisms that accomplish their function due to the deformation of one or more slender segments of their members [5]. This means they do not rely exclusively on the relative motion between joints and rigid links, but rather on the deformation of flexible components. In the precision industry, compliant mechanisms are often advantageous to rigid body linkages, for several reasons.

First, the absence of relative motion of two touching parts implies the absence of sliding friction. This means no lubrication is needed, no friction is present, and there will be less wear and noise during operation of the mechanism.

Secondly, due to the monolithic nature of compliant mechanisms, fewer parts are required. This simplifies assembly and reduces weight, thus reducing the production costs of a mechanism.

Finally, due to the absence of relative motion of connecting parts of the system, a compliant mechanism does not require as many production tolerances as a rigid link system and is no subject to mechanical play (and therefore uncertainty) in the joints. This means a compliant mechanism will generally produce more predictable and accurate movement compared to a rigid body mechanism, which is beneficial to machines in the precision industry.

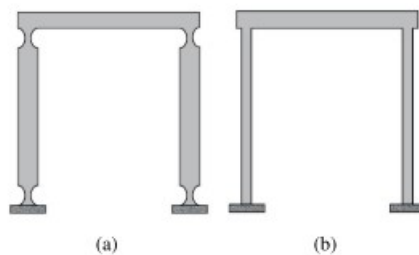


Figure 2.1: Lumped compliance (a) and distributed compliance (b)[10]

Compliant mechanisms can generally be divided into two types of compliance: lumped compliance and distributed compliance. Lumped compliance is a form of compliance where the bending is concentrated in a single point or a very short flexure. This type of compliance is comparable to a joint in a classic linkage system, and can sustain relatively high loads compared to distributed

compliance. Distributed compliance, however, is a form of compliance where the bending of the mechanism is distributed over a relatively long flexure. These flexures are prone to buckling but when the axial load of the flexures is low or absent, they are very low-resisting connections in a mechanism. The difference between lumped and distributed compliant guiders is shown in Figure 2.1.

2.2.2 Commonly used compliant mechanisms

The aim of this section is to give an overview of commonly used compliant mechanisms for which dynamic balancing may be useful. As the big advantage of dynamic balance is the elimination of inertial forces to the base, and thereby the reduction of noise in machines, the most likely field of application is the high precision industry. In the high precision industry, the most commonly used mechanisms are:

- Flexible compliant translators
- Notch Hinge compliant translators
- Cross-pivot flexures

Other examples of compliant mechanisms that allow for high precision purposes, which are therefore likely to be improved by dynamic balance, are:

- (Double) parallelogram flexure
- X-bob flexures
- Butterfly hinges

Most of these mechanisms make use of distributed compliance, so flexible beams are widely applied in these mechanisms. Therefore, dynamic balance of flexible beams may also prove useful, as has been investigated by Nijdam [8].

Compliant translators and parallelogram flexures

Compliant translators are flexure systems that allow a body to translate with certain degrees of freedom. Translations and rotations in other directions are constrained, with resistances generally several thousands times higher than the resistance in the freedom direction.

The compliant translator is shown in Figure 2.1, where 2.1(a) shows a notch joint compliant translator and 2.1(b) shows a completely flexible compliant translator. The flexible compliant translator of 2.1(b) makes use of distributed compliance and consists of two leaf flexures. These will bend when the rigid body is translated but will not invoke high resisting forces due to their low lateral stiffness. They are quite vulnerable to buckling though, as their axial stiffness is not very high either. The notch joint compliant translator makes use of lumped compliance and is highly comparable to a classical rigid body linkage mechanism. It is generally stiffer than a completely flexible compliant translator, both in its freedom direction and its constraint directions. It is also much more resistant to buckling due to its rigid middle part.

For small displacements, the movement in the constraint directions can be neglected. For larger displacements, this is often not the case. The parallelogram flexure, however, can allow larger displacements with even lower parasitic motions than a common compliant translator. An example of a double parallelogram flexure is shown in Figure 2.2. This consists of two parallelogram flexures that are symmetrically connected with the end-effector in between them.

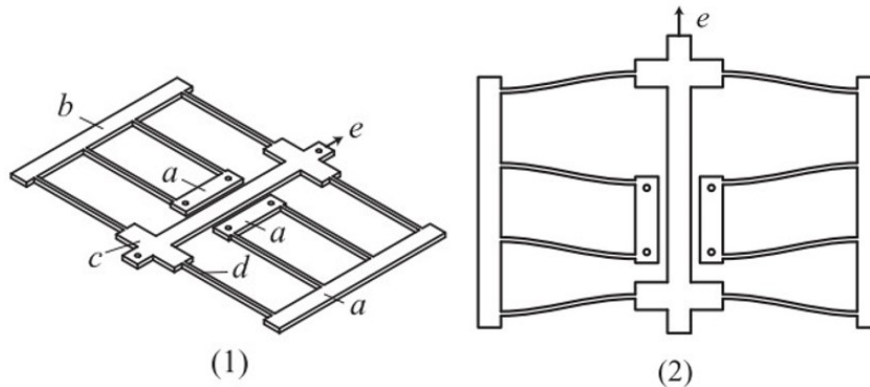


Figure 2.2: Double parallelogram flexure in spatial view (1) and planar view (2) [4]

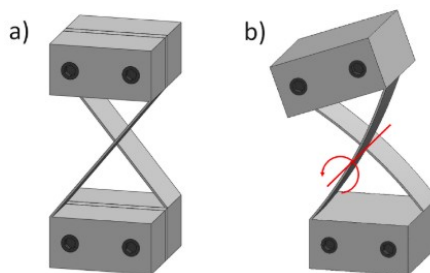


Figure 2.3: Cross-pivot flexure in undeformed (a) and deformed (b) state [11]

A parallelogram flexure consists of one normal compliant translator from the fixed world (a) to an intermediate stage (b), with another normal compliant translator from the intermediate stage to the end effector (c), parallel to the first stage.

Cross-pivot flexures and butterfly pivot

When a body is supported by one leaf spring, it is not constrained in its in-plane rotation. The center of rotation however, is not stationary. A cross-pivot flexure is a combination of two angled leaf-springs that together allow a rotation around a near-fixed center of rotation. Also, it constrains the body in its other directions, so the body has only one degree of freedom, which is a rotation approximately around the point where the flexures cross each other. This flexure is quite often used as a compliant hinge and is shown in Figure 2.3.

The butterfly pivot is a structure composed of four elementary pivots linked in series, which have their four centers of rotation collocated in the middle of the total structure [12]. This hinge is a rotational joint, quite comparable in its applications to a cross-pivot flexure. The butterfly pivot, however, has several advantages when compared to the cross-pivot flexure.

Due to its design, it allows larger stroke cycles than a cross-pivot flexure without fatigue failure. According to Henein et al. [12] their butterfly flexure allows for a $\pm 7.5^\circ$ stroke for 6 million cycles without failure, where an average pivot flexure would do only a $\pm 6^\circ$ stroke.

Other qualities of the butterfly pivot flexure are also discussed by Henein et al. but one particularly interesting one is the parasitic center shift of the butterfly pivot flexure. Due to the

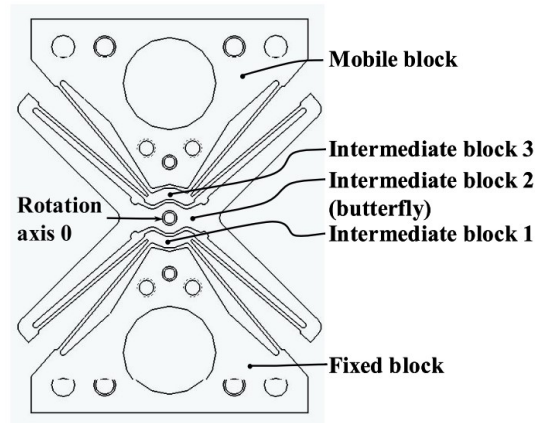


Figure 2.4: The Butterfly pivot flexure [12]

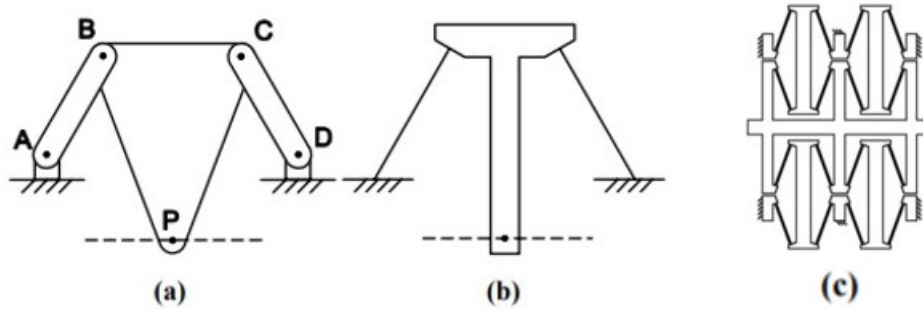


Figure 2.5: Roberts mechanism (a), compliant version of the Roberts mechanism (b) and the Xbob flexure (c) [13]

distribution of the total stroke of the flexure and the center shift compensation between the stages of the flexure, the parasitic center shift of the butterfly flexure is very low. This allows the butterfly pivot flexure to support large (pure) rotations with a high accuracy [12].

Xbob flexure

The Xbob flexure, as designed by Hubbard et al. [13] is a compliant mechanism, consisting of multiple compliant roberts mechanisms. The roberts mechanism, as shown in Figure 2.5 (a), is a four-bar linkage with a point P that will translate perfectly along a horizontal line. Figure 2.5 (b) shows a compliant version of this system, where the moving body is connected to the ground by two leaf flexures or notch hinges.

The Xbob mechanism is shown in Figure 2.5 (c) and consists of 8 compliant roberts mechanisms that all guide the same end-effector. In this figure, this end effector is thereby a perfectly horizontal translating body.

Methods	Advantages	Disadvantages	Applications
Elastic beam theory	<ul style="list-style-type: none"> • Analytical model • Easy to implement • Insightful deformation mechanism 	<ul style="list-style-type: none"> • Inapplicable for complicated configurations • Relatively low accuracy 	Static analysis
The matrix method	<ul style="list-style-type: none"> • Analytical model • Applicable for complicated configurations 	<ul style="list-style-type: none"> • Only the compliance of a flexure hinge is considered • Relatively low accuracy 	Static analysis
Castigliano's second theorem	<ul style="list-style-type: none"> • Analytical model • Easy to implement • Energy method 	<ul style="list-style-type: none"> • Inapplicable for complicated configurations • Relatively low accuracy 	Static analysis
Ryu's method	<ul style="list-style-type: none"> • Applicable for complicated configurations 	<ul style="list-style-type: none"> • Only the compliance of a flexure hinge is considered • Relatively low accuracy for statics and high order vibration modes 	Static and Dynamic analysis
PRBM + Lagrange' equation	<ul style="list-style-type: none"> • Easy to implement • Analytical model • Similar to rigid body dynamics 	<ul style="list-style-type: none"> • Only fundamental natural frequency is available • Inapplicable for distributed configurations 	Dynamic analysis
FEM	<ul style="list-style-type: none"> • Applicable for any configuration • Accurate results 	<ul style="list-style-type: none"> • Huge number of DOF • Inapplicable for control 	Static and dynamic analysis

Table 2.1: Summary of typical modelling methods for compliant mechanisms [14]

2.2.3 Modelling and synthesis techniques for compliant mechanisms

One of the main mechanics when considering compliant mechanism is the bending of flexible beams. Therefore, many of the modelling and design methods are based on an application of material mechanics. In the works of Ling et al. [14], a number of modelling methods and their advantages and disadvantages are discussed. The summary of their research is shown in Table 2.1. Other methods include the FACT method by Hopkins [15] and topology optimization methods [16], so these will also be treated in this section.

Basic material mechanics methods

Some methods in the works of Ling et al.[14] are simply based on the relation between force and displacement on a certain point of a system. This relation is determined due to known material properties or due to an already known stiffness of the beam or system. While other methods also make use of material mechanics, these methods are different because they only consider one or a few material mechanics calculations, rather than large scale analyses or methods where material mechanics are somewhere incorporated.

Elastic beam theory

Elastic beam theory makes use of mechanical derivations and material properties to determine the relation between force and displacement of elastic beams. Engineers often make use of simple outcomes of these mechanical derivations, for example the formulae in Table 2.2. This method is a linear model of the deflection of euler-bernoulli beams. Assumptions are that the material of the beams is homogeneous, Hookean, and the beam is long compared to its thickness and has a uniform cross-section.

The matrix method

As explained by Wang et al. [18], the matrix method is the formation of the compliance matrix (a 3x3 matrix containing the compliance of a structure in 2D: two translational and one rotational compliance). This compliance matrix can be used to determine the deflection of the material with respect to the applied force. A compliance matrix is the inverse of a stiffness matrix.

Castigliano's 2nd theorem

This method is also called "The Theorem of Least Work" and is based on Castigliano's second theorem. The 2nd theorem of Castigliano states that:


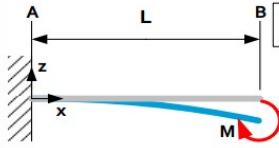
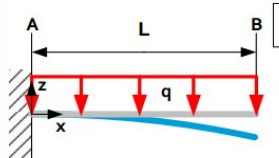
Load case	Curvature θ Sag δz	Reaction force R Shear force D Reaction moment M_R	Stress σ Stiffness C
	$\theta_A = 0$ $\theta_B = \frac{FL^2}{2EI}$ $\delta z_x = -\frac{(Fx^2)(3L-x)}{6EI}$ $\delta z_{max} = -\frac{FL^3}{3EI_y} @ x = L$	$R_A = F$ $R_B = N.A.$ $D_x = F$ $M_{Rx} = F(x-L)$ $M_{Rmax} = -FL @ x = 0$	$ \sigma_x = \frac{ M_{Rxs} }{I_y} u = -\frac{ F u(x-L)}{I_y}$ $ \sigma_{max} = \frac{ F Lu}{I_y} @ x = 0$ $ C_z = \left \frac{F}{\delta z_{max}} \right = \frac{3EI_y}{L^3} @ x = L$
	$\theta_A = 0$ $\theta_B = \frac{ML}{EI_y}$ $\delta z_x = -\frac{Mx^2}{2EI_y}$ $\delta z_{max} = -\frac{ML^2}{2EI_y} @ x = L$	$R_A = 0$ $R_B = N.A.$ $D_x = N.A.$ $M_{Rx} = M$ $M_{Rmax} = M @ x = const.$	$ \sigma_x = \frac{ M_{Rxs} }{I_y} u = \frac{ M u}{I_y}$ $ \sigma_{max} = \frac{ M u}{I_y} @ x = const.$ $ C_z = \left \frac{M}{\delta z_B} \right = \frac{EI_y}{L} @ x = L$
	$\theta_A = 0$ $\theta_B = \frac{qL^3}{6EI_y}$ $\delta z_x = -\frac{qx^2}{24EI_y} (6L^2 - 4Lx + x^2)$ $\delta z_{max} = -\frac{qL^4}{8EI_y} @ x = L$	$R_A = qL$ $R_B = N.A.$ $D_x = q(L-x)$ $M_{Rx} = -\frac{q(L-x)^2}{2}$ $M_{Rmax} = -\frac{qL^2}{2} @ x = 0$	$ \sigma_x = \frac{ M_{Rxs} }{I_y} u = \frac{ q u(L-x)^2}{2I_y}$ $ \sigma_{max} = \frac{ q L^2u}{2I_y} @ x = 0$ $ C_z = \left \frac{q}{\delta z_{max}} \right = \frac{8EI_y}{L^4} @ x = L$

Table 2.2: Several linear elastic beam theory equations [17]

"The first partial derivative of the total internal energy in a structure with respect to the force applied at any point is equal to the deflection at the point of application of that force in the direction of its line of action" [19].

This theory is applicable to linearly elastic (Hookean) material structures with constant temperature and unyielding supports. From Castigliano's 2nd theorem follows the theorem of least work, which states:

The redundant reaction components of a statically indeterminate structure are such that they make the internal work (strain energy) a minimum.

This theory thereby states that reaction forces are present within the material due to the deformation of said material, and thereby strongly resembles the matrix method and linear elastic beam theory.

Ryu's method

Ryu's method is the design process as executed by Ryu et al. [20] to optimally design a flexure hinge based XY θ wafer stage. This method consists of three steps:

- Preliminary design (a design of a compliant mechanism is made)
- Modelling the design physics
- Optimizing the model in order to optimize the design

The optimization parameters of Ryu were all dimensions of his design (all thicknesses in 3D). The optimization algorithm used by Ryu is the SQP algorithm.

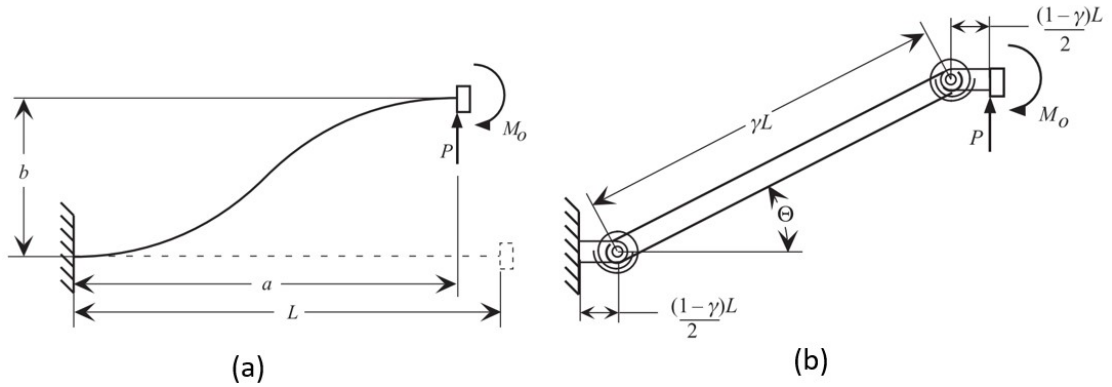


Figure 2.6: Flexible beam (a) and a PRBM approximation of that beam (b) [4]

Pseudo Rigid Body Model (PRBM)

A common tool to perform quick calculations on compliant mechanisms is the Pseudo Rigid Body Model (PRBM)[13]. This method approximates a flexible beam using a rigid beam with linear rotational springs at its joints. This yields simple linear approximations for the behaviour of the flexible beam. An example of this is shown in Figure 2.6. To gain a closer approximation to a distributed compliant flexure, more than one rigid body can be used with rotational springs between all bodies, which would yield a chain of alternating rigid bodies and springs. For notched flexures, an approximation with only one rigid body is most often used.

PRBM is often used for preliminary design choices in simple mechanisms, as the calculations are quick and easy. They generally do not take into account nonlinearities and can only be used for simple configurations, so often a FEM-analysis of the final mechanism is performed to ensure a working design.

Finite Element Methods

One of the most used methods for analysis of Compliant Mechanisms designs is analysis with Finite Element Methods (FEM). Finite element methods use numerical approximations of structures, by which the structure is divided in a finite number of small elements. To be able to analyze these elements, a global stiffness matrix is constructed. Then a computer is used to apply (mostly linear) mechanics to those small elements to form a full overview of stresses, displacements and reactions in structures (not limited to beams). FEM is often used in industry and allows to accurately analyze complex structures. Examples of commonly used FEM software are ANSYS, COMSOL and many CAD programmes such as AUTOCAD and SolidWorks, but numerical calculation programmes such as Matlab and Python can also be used.

The equations of finite element analysis can be expressed in matrix form as shown in equation 2.2.1, where $\{f\}$ is the force vector, $\{u\}$ is the displacement vector and $[K]$ is the global stiffness matrix.

$$[f] = [K] [u] \quad (2.2.1)$$

Another numerical method that's quite similar to FEM is the "Chain Algorithm" [1]. This method is applied to beams only, and divides the beam in one chain of elements. Then the system of equations as shown in equation 2.2.2 is solved in order to find a solution for the systems

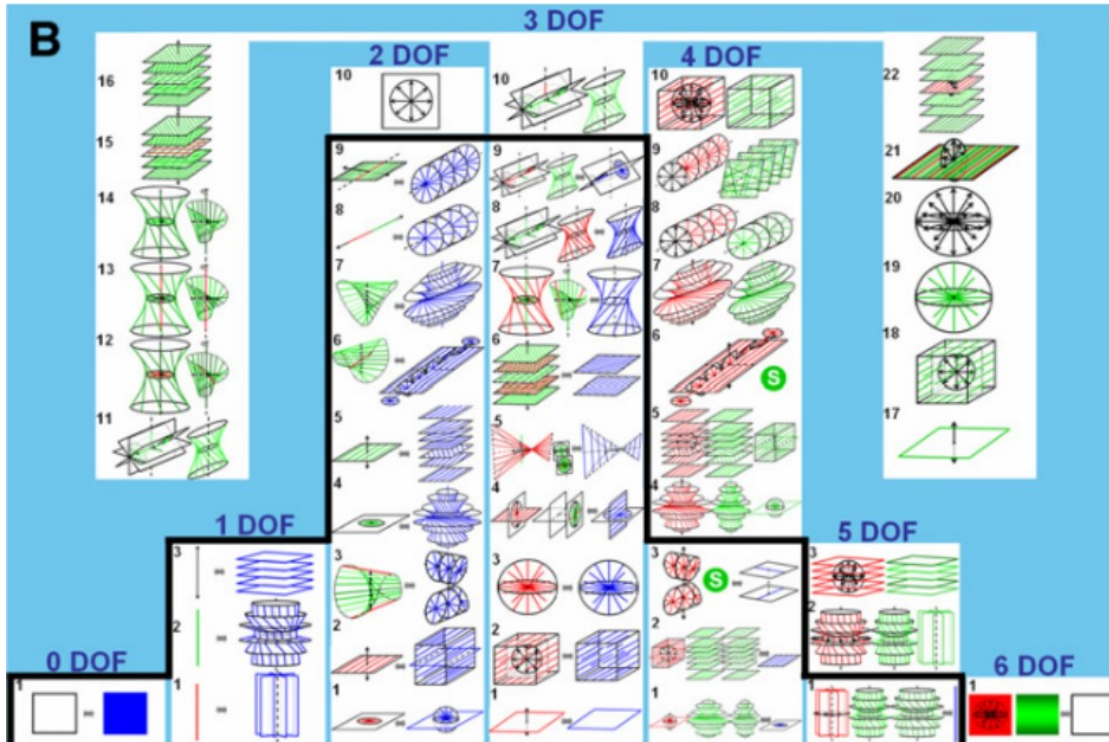


Figure 2.7: FACT table by Hopkins and Culpepper [15]

properties. In this equation, δ_{ax} and δ_{tr} are the axial and transverse displacement respectively, and θ the tip rotation. P is the axial or transverse force and M the resulting moment at the end of the element with respect to the begin of the element. $[K]$ is again the global stiffness matrix, but as its inverse is taken, it represents the compliance matrix. This system of equations is solved for every element of the chain.

$$\begin{bmatrix} \delta_{ax} \\ \delta_{tr} \\ \theta \end{bmatrix} = [K]^{-1} \begin{bmatrix} P_{ax} \\ P_{tr} \\ M \end{bmatrix} \quad (2.2.2)$$

FACT Method

FACT is an entirely different way of designing compliant mechanisms. The FACT method consists of an overview of constraint spaces, and their corresponding freedom spaces. This allows engineers to be more creative in the design of their optimal compliant mechanisms. The FACT method is based on the screw theory and therefore has a mathematical basis, but for design purposes, a chart with all possible outcomes of the method is available.

The FACT method is invented by Hopkins and Culpepper [15] and the FACT solution table is shown in Figure 2.7. This table contains all possible situations for the design of a compliant mechanism. When this table is used, the screw theory is not required to apply the FACT method.

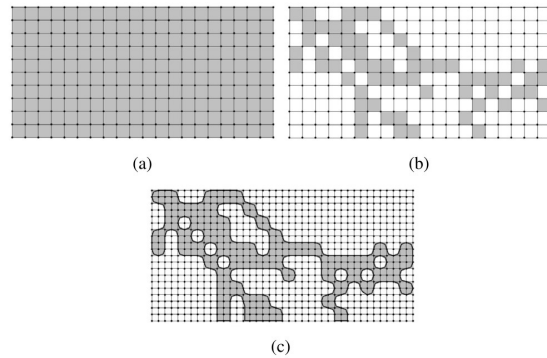


Figure 2.8: Three steps in topology optimization: (a) initial control meshes, (b) proposed control meshes arrangement, (c) proposed arrangement after subdivision. [5]

Topology Optimization

Optimization algorithms are often used to optimize certain parameters of a design. Size and shape optimization often change a few pre-determined variables to find the optimal dimensions for a set design. Topology optimization takes this to a higher level and changes the structural geometry, not only by changing dimensions but also by adding or removing holes, masses or parts. As described by Gallego: "Topology optimization methods search for the connectivities mainly as an existence/nonexistence problem of the constitutive elements inside a universal structure where all the possible and allowed connections are already defined [10]. A simple example is shown in Figure 2.8, where an initial rectangular workspace is shown where the algorithm subtracts material in certain squares of the mesh to obtain a mechanism or structure [5].

Topology optimization is the process of determining the optimal layout of material and connectivity inside a design domain [21]. This methodology can be used to synthesize monolithic designs, such as compliant mechanisms, based on a predefined set of boundary conditions and one or more desired functions.

In the research of Gallego [10] the static balancing of compliant mechanisms has been investigated. With a topology optimization approach, designs were synthesized with high performance, but perfect static balance has not been achieved.

Generally, topology optimization is used to determine optimal material distribution in static applications, such as the optimal material distribution in the wing of an aircraft [22] or a lightweight material distribution of a city bus [23]. Recently, mechanism design [6], [16] and design incorporating dynamic behaviour [24], [25] are becoming increasingly common.

2.3 Balancing Methods

A mechanism is a mechanical device used to transfer or transform motion, force or energy [1]. Traditionally, mechanisms consist of rigid bodies, connected with hinges, sliders or other types of joints. One of the most common mechanisms in this regard is the four-bar mechanism, as shown in Figure 2.9. A mechanism such as shown in Figure 2.9 can be simplified to a schematic drawing of only linkages and Centers of Mass of the bodies. An example of this is given in Figure 2.10 from the works of van der Wijk.

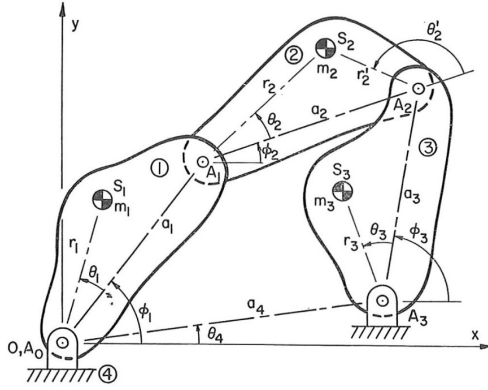


Figure 2.9: Four-bar linkage with arbitrary link mass distributions [26]

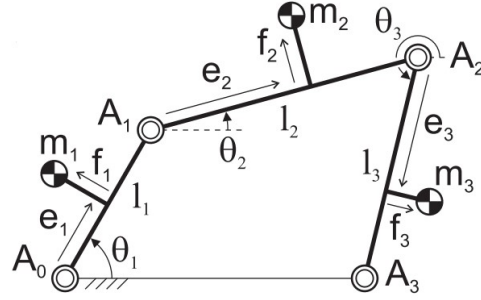


Figure 2.10: A simplified representation of a four-bar linkage with arbitrary link mass distributions [9]

By changing the angle θ_1 , the rest of the mechanism will move in a prescribed manner due to its geometric constraints at the links. The movement of the bodies of this mechanism, however, will incur inertial forces and moments. This can cause significant undesired vibrations in the system [9]. Especially in precision applications, these undesired vibrations can be a large source of uncertainty in the system. Therefore, the system should be designed such that these vibrations are sufficiently reduced or even eliminated.

A promising way of reducing or eliminating vibrations in mechanisms is the use of balancing techniques. A mechanism is balanced statically and dynamically if the resultant force vector and the resultant moment due to inertia forces are equal to 0 [27]. Balancing can be divided in two categories: static balancing and dynamic balancing. A system is statically balanced if the potential energy of the system is conserved upon movement of the system throughout a finite range of motion. A system is dynamically balanced if the shaking moments as well as the shaking forces are eliminated [28]. This means the forces and moments due to changes of inertial moments in the system cancel out, which means no net dynamic forces or moments are transmitted to the base of the system. When a system is shaking force balanced, this system is also statically balanced by definition [29]. This means a dynamically balanced system is always statically balanced.

2.3.1 Balancing conditions

In order to achieve dynamic balance, shaking forces and shaking moments should be eliminated. This means no residual forces or moments will act on the base of the system due to inertia forces of the system's movement. To achieve shaking force balance, the linear momentum of the system should always be zero, which means the common CoM of all elements of the mechanism should be stationary. The condition for force balance can be mathematically written by equation 2.3.1:

$$\mathbf{p} = \sum_i m_i \dot{\mathbf{r}}_i = 0 \quad (2.3.1)$$

In this equation \mathbf{p} is the total linear momentum vector of the system, i the link number, m_i the mass of link i and $\dot{\mathbf{r}}_i$ the velocity vector of link i .

To obtain shaking moment balance in the system, the angular momentum of all bodies combined

has to be constant as well. This is expressed by equation 2.3.2:

$$\mathbf{h}_{O,z} = \sum_i I_i \dot{\alpha}_i + \mathbf{e}(\mathbf{r}_i \times m_i \dot{\mathbf{r}}_i) = 0 \quad (2.3.2)$$

In this equation, I_i is the moment of inertia of member i , $\dot{\alpha}_i$ is the absolute angular velocity of member i , \mathbf{e} is the unit vector of rotation (in the z -direction for planar cases). The two terms in this equation represent the angular momentum of every individual body and the angular momentum that results from linear momentum at a distance from the base of the system for every individual body, respectively.

2.3.2 Available balancing methods

Throughout the years, many different techniques have been invented to design shaking force balanced mechanisms. Some of these techniques also provide shaking moment balance to these mechanisms, but this is not always the case. In this section, a number of methods to obtain (partial) dynamic balance is presented.

Counter-weighting

A simple method of obtaining shaking force balance is the addition of counterweights. When counterweights are correctly applied, the shaking forces of a system cancel out perfectly. However, counterweights may add a lot of mass to a system and cancelling out shaking moments may still be difficult. A lightweight solution to completely shaking force and shaking moment balance a system is through counter-rotary counterweights (CRCW's), as is shown by Herder and Gosselin [30]. These work by both applying counterweights (for shaking force balance) and making these counterweights rotate upon movement of the system (to achieve shaking moment balance).

Fischer's method of principal vectors

German physiologist, physicist and medical doctor Otto Fischer derived a method to derive kinetic energy in the human musculoskeletal system. This method has been used for years in the pre-computer era, and is also applicable to other mechanical systems. Especially Fischer's method of principal vectors has proven to be very useful in the analysis and synthesis of balanced linkage systems[31].

Fischer's method of principal vectors adds a series of binary structural elements in a parallelogram fashion to the original mechanism. One point in the resulting augmented mechanism will always coincide with the CoM of the system, which therefore gives insight in the dynamic balance of the system, as shown in Figure 2.11. From this method, conditions for shaking force balance can be derived.

Double Contour Transformation

V. A. Shchepetil'nikov proposed an alternate method called "double contour transformation" [32]. This method generates an attached proportional auxiliary mechanism which traces out the CoM of a plane mechanism. With an auxiliary mechanism and a counterweight, the movement of the CoM of the mechanism can be reduced to a stationary point, and the first harmonic of the shaking moment can be eliminated. This does not completely dynamically balance the mechanism, but it will already eliminate the shaking forces and reduce the shaking moments[29] [32].

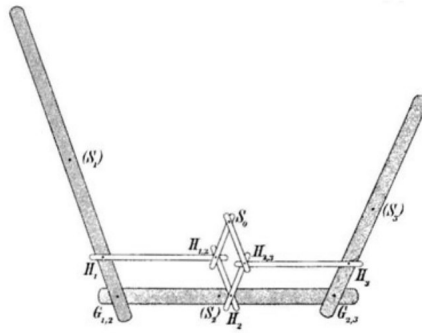


Figure 2.11: Mechanism by Fischer to trace the CoM of three links at S_0 by additional links [31]

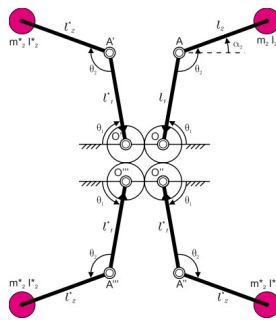


Figure 2.12: Balanced double pendulum by using axial and mirror symmetric mechanism duplicates [35]

Methods of duplicated mechanisms

An intuitive way of balancing a mechanism is by duplicating it. This technique achieves dynamic balance by letting two identical mechanisms perform identical yet opposite movements, which means the total reaction forces and moments on the base of the system will be zero and the system will be globally dynamically balanced. A common method for this is the method of "Counter-Rotary Counterweights" (CRCM)[33]–[36], which is shown in Figure 2.12.

In some cases in literature, systems can only be shaking force balanced but not shaking moment balanced. In those cases, adding just one oppositely moving system often proves to be enough to also shaking moment balance the system.

Linearly independent vectors

The method of linearly independent vectors determines the total CoM of the system by taking a mass-weighted summation of all CoM locations of the bodies of the system. This is done according to equation 2.3.3 [37].

$$\mathbf{R}_s = \frac{1}{M} \sum_{i=2}^n m_i \mathbf{R}_{ci} \quad (2.3.3)$$

In this equation \mathbf{R}_{ci} is the position vector of the CoM of body i , with m_i the mass of body i . M is the total mass of the system.

The equation of describing the position of the total mechanism center of mass is written in such a way that the coefficients of the time-dependent terms may be set equal to zero. In this way, the total center of mass can be made stationary and the shaking force vanishes [26].

The method of linearly independent vectors is extended by Bagci to the method of force balancing by idler loops [37]. This method allows complete shaking force balancing of irregular force transmission mechanisms whose force balancing failed due to the existence of one or more links that have connections to the fixed link and permit at least one linear freedom in each.

Screw Theory

Screw theory is a method of modelling motions or constraints of a system. It models the movement in a way that's comparable to the movement of a screw, with a rotation around an axis and a translation along that axis.

De Jong et al. made an attempt to design a dynamically balanced mechanism using screw theory [38]. They managed to create a instantaneously dynamically balanced 5-bar mechanism where the balanced paths had a shaking moment reduction of 95% compared to the non-balanced paths. The shaking forces were at least 96% lower than the internal bearing forces, indicating force balance. This means screw theory can be used to achieve at least instantaneous dynamic balance. Instantaneous balance means the system is only balanced along a certain trajectory. However, this is not the same as global balance, where the system is balanced throughout its entire range of motion. When speaking of dynamic balance, this generally implies global balance.

Linearly independent linear momentum

In the works of van der Wijk, linearly independent linear momentum is proposed as a straightforward and intuitive way to design dynamically balanced rigid body linkages [9]. This method has the capability to achieve both shaking force and shaking moment balance. In this method, the linear momentum of a closed chain linkage is expressed in a linearly independent form.

Modal balancing

A novel method of balancing mechanisms is modal balancing. This method approaches a mechanism as 100% elastic and balances it modal. In the research of Martinez, Meijaard and van der Wijk [39], the shaking force and shaking moment balance conditions have been found for flexible beams. This has been further investigated by the research of Nijdam [8]. Both researches proved modal balancing to be a useful tool in balancing of compliant mechanisms.

For flexible beams to be modally balanced, conditions such as equal and opposite dynamic forces are required. For this to be possible with flexible beams, the frequency of the movement of the beams (i.e. eigenfrequency of the beams) has to be equal.

In the research of Lisanne Nijdam [8], a rigid body balanced watch oscillator has been redesigned to a simpler mechanism with the same functionality. This redesign has been based on a modal balancing approach. Both mechanisms are shown in Figure 2.13.

2.4 Comparison of available methods

To design an inherently balanced compliant mechanism, a combination of compliant mechanism design methods and dynamic balancing methods is likely to be necessary. Combining these may

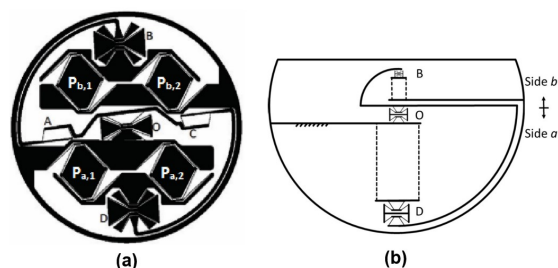


Figure 2.13: Original design by Weeke et al (a) and redesign by Nijdam (b) [8]

however prove to be a challenge. In this chapter, some limitations of both methods will be discussed, after which the challenges of combining them will be explored.

2.4.1 Compliant mechanisms

In industry, compliant mechanisms are often designed in a trial-and-error fashion. A design is made, dimensions are estimated using linear elastic material mechanics and the design is simulated using FEM. This process is then iterated until a good design is made.

Most methods for designing compliant mechanisms work this way, which leads to certain important limitations. These limitations are discussed below and explained per method in Table 2.3.

Firstly, most methods are used for analysis rather than synthesis of mechanisms. This can make them hard to incorporate in a more elaborate design method. These analysis techniques are only useful for determining or optimizing dimensions in a readily available design, rather than creating a new design. The FACT method and topology optimization, on the other hand, allow the engineer to invent new types of mechanisms.

Secondly, techniques such as FEM and topology optimization are computationally very expensive techniques rather than simpler methods such as elastic beam theory and PRBM. These therefore require computer programs to aid with the design, whereas simpler methods use calculations that can easily be solved by hand. The FACT method is especially different, as this design method focuses on constraining and allowing movement in specified degrees of freedom, rather than solving equations.

Thirdly, most dynamic balancing techniques concern planar rigid body linkages. This may cause problems when considering compliant mechanisms, as they by default use deformation and compliance instead of only rigid bodies and hinges. This can cause a mismatch when attempting to combine methods for designing dynamically balanced compliant mechanisms. The fact that most methods concern planar mechanisms will not automatically cause a mismatch, but is still a shortcoming of these methods because many mechanisms are 3-dimensional.

2.4.2 Dynamic balancing

When considering dynamic balancing, some properties are common among most of the design techniques. Most techniques are only considering rigid body linkage systems in 2D, and often additional masses or symmetric countersystems are used to balance the system. Furthermore, in the end, perfect balance can only be theoretically achieved, but is in practice not feasible due to

Method	Analysis or synthesis	Computational costs	Used for
Elastic beam theory	Analysis	Low	Slender beams
Matrix method	Analysis	Low	Solid structures
Castigliano's 2 nd theorem	Analysis	Low	Slender beams
Ryu's method	Analysis	Medium	Optimizing dimensions in any mechanism
PRBM + Lagrange	Analysis	Low	Compliant beams & notch hinges
FEM	Analysis	High	Any structure
FACT	Synthesis	None	Leaf flexure and wire flexure systems
Topology optimization	Synthesis	Very high	Any structure

Table 2.3: Compliant mechanism design techniques

manufacturing errors.

Another property of most methods for achieving dynamic balance is that most methods are focused on balancing existing mechanisms, rather than designing a system which is inherently balanced from the start. This is not necessarily a shortcoming, but likely limits the performance of the resulting mechanism in terms of weight and simplicity.

In Table 2.4 an overview is presented of the dynamic balancing methods that are treated in chapter 2.3. First, the balancing level is treated. This indicates the maximum level of dynamic balancing that's achievable with the method. Shaking force balance is the most common balancing level, but to achieve perfect dynamic balance, shaking moment balance is also required.

Another category by which the balancing methods can be distinguished is the type of mechanisms they can be applied to. In order to combine techniques, this could prove to be a very important property.

2.5 Conclusion

The available methods for designing dynamically balanced mechanisms and compliant mechanisms are quite far apart, but some of them do have similarities. The mentioned methods of this paper do present opportunities to create dynamically balanced compliant mechanisms, but they are not all straight-forward.

The research question of this paper was as follows:

Which currently existing methods seem most promising for dynamically balancing of compliant mechanisms?

The answer to this question consists of several opportunities of combinations of techniques. These are presented and briefly explained in this chapter.

Method	Balancing level	Balanced mechanism
Principal vectors	Shaking forces	Open and closed chain linkages
Double contour transformation	Shaking forces & Partially shaking moments	1 DOF closed chain linkages
Duplicate mechanisms	Shaking forces & shaking moments	At least parallel mechanisms, crank-slider mechanisms, 4-bar mechanisms
Linearly independent vectors	Shaking forces	1 DOF closed chain linkages
Screw Theory	Shaking forces & partially shaking moments	At least up to 2 DOF closed chain linkage
Inherent dynamic balancing	Shaking forces & shaking moments	Closed chain linkages
Modal balancing	Shaking forces	Flexible beams

Table 2.4: Dynamic balancing techniques

Topology optimization

Topology optimization is a computer aided design technique that allows an engineer to enter an objective and constraints and leave the design process to the algorithm. One of the constraints in this optimization could be dynamic balance of the system, but as this is quite a large and probably nonlinear constraint, this may result in infeasible design spaces. A first approach to incorporate dynamic balance would be the use of the basic principles of balance: constant linear and angular momentum. In the case of topology optimization, this would mean the mass-weighted displacement vectors of the elements of the system should add up to zero. The topology optimization approach to design dynamically balanced compliant mechanisms is very different from the existing methods and may therefore yield drastically different designs from the currently designed mechanisms. This may not only prove useful through the resulting new mechanisms, but also might bring new insight in dynamic balancing methods which were not found before.

Screw theory and the FACT method

In the paper of de Jong et al. [38] an example is presented of how screw theory can be used to design a dynamically balanced mechanism. The FACT method, one of the strong synthesis methods of compliant mechanisms, is also based on the screw theory. A combination of these methods is therefore a feasible way forward to design dynamically balanced compliant mechanisms. However, de Jong et al. showed that for their 5-bar linkage only instantaneous dynamic balance could be achieved. A 4-bar linkage may be globally balanced though, as it has fewer degrees of freedom than the designed 5-bar linkage and is therefore constraint to follow a specific path, such as the balanced path of the 5-bar linkage.

PRBM and any rigid body linkage system balancing method

PRBM is a compliant mechanism modelling method that simplifies a flexible beam to a rigid body with a torsional spring around its joint. As a rigid body linkage system consists of rigid bodies and joints, the many available linkage system balancing methods seem promising to design balanced compliant mechanisms. However, current balancing methods do not take internally vibrating

bodies into account. This means static balancing or modal balancing may be required to avoid unwanted vibration of the flexible members, so this method may not be sufficient.

Modal Balancing

As presented in the papers of Nijdam [8] and Martinez, Meijaard and van der Wijk [39], modal balancing is discussed as a method to balance flexible beams. This is the only method that has currently been used to balance compliant bodies and is still quite unexplored. Nijdam and Martinez showed multiple functioning balanced mechanisms, thereby proving the force of this method.

Chapter 3

Topology Optimization Algorithm for Synthesis of Dynamically Balanced Compliant Mechanisms

Abstract

This paper presents a new methodology for synthesizing dynamically balanced compliant mechanisms. A topology optimization algorithm is presented, which contains constraint functions to ensure dynamic balance in the resulting geometries. The dynamic balance is realized both in terms of shaking force balance and shaking moment balance using a constant linear and angular momentum approach. This approach does not take into account eigenmode, but focuses on the relative movement of bodies in the mechanism. The topology optimization algorithm is written in MATLAB and is based on the 99-line and the 88-line code. The method of moving asymptotes is used for updating the design throughout the iterations.

Two mechanisms are synthesized in MATLAB using the newly designed algorithm. One of these mechanisms is balanced, the other unbalanced. The resulting geometries are post-processed in SolidWorks. To ensure the roughness of the optimization algorithm and the transition through SolidWorks do not cause significant errors, the kinematic behaviour of the resulting geometry is verified through simulations in COMSOL. First, a static analysis is performed where inertial terms are not included, to compare the COMSOL results with the MATLAB code. After this validation, COMSOL is used to analyze the MATLAB geometries in a dynamic manner by analyzing eigenmodes, reaction forces, linear momentum and angular momentum. This analysis is done first in a quasistatic manner and after that with the inclusion of inertial terms. This shows a quasistatic approach is a good first step towards designing dynamically balanced compliant mechanisms with topology optimization, but eigenmodes may still disturb the dynamic balance of the resulting mechanism. Therefore, a dynamic analysis including

eigenfrequencies and modal balancing needs to be incorporated in the design process of dynamically balanced compliant mechanisms.

Nomenclature

Abbreviation	Meaning
COM	Center of Mass
SFB	Shaking Force Balance
SMB	Shaking Moment Balance
SIMP	Solid Isotropic Material with Penalization
SISO	Single Input Single Output
DOF	Degree of Freedom
MMA	Method of Moving Asymptotes

3.1 Introduction

For machines and mechanisms operating at high speeds and high precision, shaking forces and shaking moments are often undesired [35], as they cause vibrations, noise, wear, and fatigue problems, and therefore limit the full potential of many machines [26], [40]. These shaking forces and shaking moments can be mitigated or even completely removed by various techniques [9], [26], [29]–[32], [37], [38]. When there are no shaking forces nor shaking moments throughout the entire range of motion of a mechanism, that mechanism is Dynamically Balanced.

Dynamic balance is achieved in a mechanism if the linear and angular momenta are constant. If the Center of Mass (CoM) of a mechanism is stationary in all translational directions throughout the entire range of motion, the linear momentum is constant and the mechanism is Shaking Force Balanced (SFB). If the rotations of all masses in the mechanism around the CoM cancel out as well, the angular momentum is constant and the system is Shaking Moment Balanced (SMB) [9]. A system is dynamically balanced if both SFB and SMB are satisfied.

The first dynamic balancing techniques were only considering shaking forces, for example the method of principal

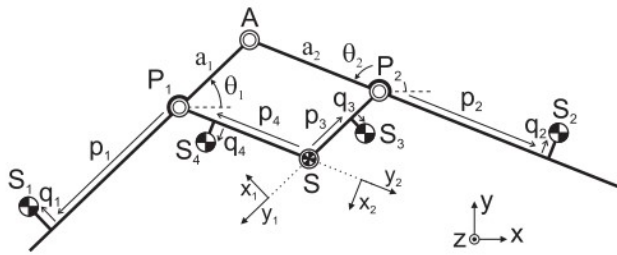


Figure 3.1: 2DOF pantograph ([9])

vectors by Fischer [31], double contour transformation [32], and the Method of Linearly Independent Vectors [26]. Later, shaking moments were also addressed, mostly by adding auxiliary mechanisms such as counter-rotary counterweights [30]. Often such auxiliary mechanisms are identical, yet opposite, mechanisms with the same kinematic properties [33], [35], [36]. These systems make use of symmetry to obtain shaking force and shaking moment balance. Van der Wijk proposed his method of inherent balancing [9] where mechanisms can be designed without the need for symmetry.

Using the inherent balancing method, systems such as the 2DOF pantograph in Figure 3.1, can be designed. In this method, a rigid body linkage with two input angles is designed, which is shaking force balanced. However, the representation of the system in Figure 3.1 is a simplified one, showing only the dimensions of the rigid bodies, linkages and their COM's, not the actual geometries of the bodies.

The available dynamic balancing methods generally consider rigid body linkages with prescribed CoMs, or symmetric auxiliary systems. This reduces the design freedom, and will change COM's by adding redundant mass with the sole purpose of balancing [41], rather than optimally redistributing mass.

In the meantime, compliant mechanisms are rapidly gaining importance, yet their design remains challenging[5]. One way of designing compliant mechanisms is Topology optimization [42]. Topology optimization has, however, not often been used for designing dynamically balanced mechanisms yet. A first attempt on designing a dynamically balanced four-bar mechanism using topology optimization is proposed by Ayala-Hernandez[25]. This approach shows a topology optimization algorithm with dynamic balancing constraints should be able to synthesize mechanisms which are inherently force- and moment-balanced, while distributing masses such that they contribute to the system's performance. This approach, however, only considered the balancing of an existing mechanism by optimizing two rigid bodies rather than inherently balanced design of the whole mechanism.

In this paper, a topology optimization algorithm is presented with a problem formulation which includes dynamic balance. In Section 3.2, the used model and the approach to dynamic balancing is explained. Several mech-

anisms are designed using this algorithm, which are then analyzed using COMSOL in Section 3.3, where they are compared to their unbalanced counterparts. The results are discussed and improvements and suggestions for further research are proposed in Section 3.4. Finally the conclusions are presented in Section 3.5.

3.2 Method

To synthesize dynamically balanced compliant mechanisms, a topology optimization problem formulation requires at least the following functions:

- An output function
- A volumetric constraint
- A shaking force balance (SFB) function
- A shaking moment balance (SMB) function

Depending on the desired outcome, either one of these functions or a combination of multiple functions is used as the objective function, with the others acting as constraints. Additionally, a finite element model is required to establish a relation between the forces and displacements of the elements in the design space. An optimization algorithm is then utilized to determine the change of the design variables at each iteration.

3.2.1 Used model and assumptions

Topology optimization algorithm

This topology optimization algorithm is based on the 99-line Matlab code of Bendsoe and Sigmund [42]. This algorithm is a well-documented This code employs a Solid Isotropic Material with Penalization (SIMP) model and progresses using the optimality criteria method. The optimality criteria method, however, is not easily extended to problems with multiple constraints, and is therefore changed to an MMA-algorithm (Method of Moving Asymptotes) by Svanberg [43], which is well-suited for topology optimization purposes [16] and quite standard in the treatment of advanced topology optimization problems. This algorithm utilizes non-conforming mesh, a linear finite element model and a volumetric constraint. This constraint assures that only a predetermined percentage of the design space can be filled with material. As a result of the penalization in the SIMP approach, the design space becomes a black-and-white mechanism, with the volumetric constraint determining the percentage of solid (black) elements.

The SIMP method adds a penalization factor on the design variables, which is favourable for a black-and-white distribution of the elements in the resulting geometry. This causes the original design variables to lose their physical representation, which means the penalized design variables represent the resulting physical geometry. Before the penalization, a filter is applied. The original design variables will be denoted as α , the filtered design variables as $\tilde{\alpha}$ and the penalised filtered design variables

(densities) will be denoted as ρ . Usually in literature, the original design variables are denoted as \mathbf{x} and the filtered design variables as $\tilde{\mathbf{x}}$, but to avoid confusion with positions and displacements in x-direction, $\boldsymbol{\alpha}$ is used in this article. In the code, however, \mathbf{x} is used as the vector of design variables.

Some parts of the 99-line code have been replaced by parts of the 88-line code by Andreassen et al. [44] for computational efficiency purposes. The 99-line code uses a loop over all elements to calculate the constraints, whereas the 88-line code uses only one matrix operation. This greatly increases performance, especially for large element-sized optimizations. In the 88-line code, a modified SIMP method is used, where a minimum stiffness is defined to prevent the stiffness matrix \mathbf{K} from becoming singular. For this optimization code, however, the classical SIMP method is used, because the MMA-algorithm already has a lower limit for the design variables, thereby ensuring a minimum stiffness.

Another assumption in the algorithm is to work in a 2-dimensional design space. The proposed approach should work equally well in a 3-dimensional design space, but a 2D-approach is sufficient for a proof of concept and highly preferable for computational efficiency.

Dynamic balancing model

Dynamic balance is only fully realized when both SFB and SMB are achieved. The linear momentum of the system's COM and the angular momentum around it need to remain constant throughout the entire range of motion, following the inherent balancing methodology by van der Wijk [9]. For most mechanisms, this means the linear and angular momenta need to be 0, as the system will likely be stationary on the start and/or finish of its movement. The balance conditions of the inherent balancing method are represented by equation 3.2.1 and 3.2.2.

$$\mathbf{L} = m\dot{\mathbf{r}} \quad (3.2.1)$$

$$\mathbf{H}_S = I\dot{\boldsymbol{\theta}} + m(\mathbf{r} \times \dot{\mathbf{r}}) \quad (3.2.2)$$

In these equations, \mathbf{L} is the linear momentum, m is the mass, \mathbf{r} is the position vector with $\dot{\mathbf{r}}$ the corresponding velocity vector. \mathbf{H}_S is the angular momentum around the total Center of Mass (CoM), I the moment of inertia, and $\dot{\boldsymbol{\theta}}$ the angular velocity.

The dynamics of the system are approached from a quasi-static perspective, omitting vibration analysis and solely focusing on linear and angular momentum. The approach to this linear and angular momentum is taken element-wise, as the topology optimization algorithm is based on elements rather than large rigid bodies.

The linear momentum of the system is represented by the sum of the masses of all elements, multiplied by their respective velocities. However, general topology optimization methods consider statics and therefore do not consider velocities but only displacements. To simplify this during the iteration process, a linear relation between the

displacements of the system's elements and the velocities is assumed, using element displacements (rather than velocities) for momentum analysis. After all, if all elements undergo a linearly increasing displacement in the same amount of time, their corresponding velocities will be proportional to their final displacements. Just like the actual linear and angular momentum, the linear and angular momentum approximations should be 0 for dynamic balance of the system.

3.2.2 Balance Conditions

As discussed earlier, SFB and SMB require the linear momentum and angular momentum of the COM to be 0, respectively. Assuming linearity, this condition is met when the displacements of the COM are 0 and the mass-weighted rotations of elements around the COM cancel out to 0. These two conditions are satisfied through equation 3.2.3 for SFB and 3.2.4 for SMB.

$$\mathbf{B}_{SF} = \frac{\sum (\rho_{el} \mathbf{u}_{el})}{\sum \rho_{el}} = \mathbf{0} \quad (3.2.3)$$

$$\mathbf{B}_{SM} = \frac{\sum (\rho_{el} (\mathbf{r}_{el} \times \mathbf{u}_{el}))}{\sum \rho_{el}} = \mathbf{0} \quad (3.2.4)$$

In these equations, \mathbf{u}_{el} is the displacement vector of an element, ρ_{el} is the density of that element, and \mathbf{r}_{el} is the position vector of the element with respect to the COM of the entire system. Both sums are taken over all elements.

The approach is limited to a 2-dimensional analysis, requiring two linear momentum equations and one angular momentum equation to determine the dynamic balance of the system.

The resulting balance conditions are given in Equation set 3.2.5:

$$u_x = \frac{\rho \mathbf{A}_x \mathbf{u}}{\sum \rho} = 0 \quad (3.2.5a)$$

$$u_y = \frac{\rho \mathbf{A}_y \mathbf{u}}{\sum \rho} = 0 \quad (3.2.5b)$$

$$R_z = \frac{\rho \mathbf{M} \mathbf{u}}{\sum \rho} = 0 \quad (3.2.5c)$$

$$\mathbf{M} = \mathbf{r}_x \mathbf{A}_y - \mathbf{r}_y \mathbf{A}_x \quad (3.2.5d)$$

In these equations, \mathbf{u} is the global displacement vector. The matrices \mathbf{A}_x and \mathbf{A}_y are multiplied with the global displacement vector to obtain element-averaged x- and y-displacements, rather than nodal x- and y-displacements. The balance equations are represented in their global form, computing the linear momenta in one linear algebraic equation.

The shaking moment balance constraint uses a matrix \mathbf{M} in the computation. This matrix takes into account both the node-to-element conversion and the computation of the angular momentum. In Equation 3.2.5d the computation of \mathbf{M} is shown, consisting of both linear momenta and their arms \mathbf{r}_x and \mathbf{r}_y with respect to the COM of the system in x- and y-direction, respectively. In this equation, \mathbf{r}_x and \mathbf{r}_y are diagonal matrices with the distance

between the COM and the corresponding element on the main diagonal. The representation in the code is a point-wise multiplication.

The balance equations, as displayed in equations 3.2.3, 3.2.4 and 3.2.5, differ from the general balance equations 3.2.1 and 3.2.2 as they do not consider the total system mass. In the case of perfect balance, this results in the same outcome because for the total linear momentum to be 0, either the mass or the displacement of the COM needs to be 0. As the mass cannot be 0, the displacement of the COM has to be. Nevertheless, mass is used to determine the individual contributions of the displacements of all elements and thereby the displacement of the COM.

Another difference between the method of inherent balancing and the topology optimization algorithm is the term $I\dot{\theta}$. During the iteration process, elements are considered pointmasses. Pointmasses do not have rotational degrees of freedom, so their angular momentum around their own COM is 0. If this assumption would not be made, the contribution of the angular momenta of the elements around their own COM would still be negligible with respect to their rotation around the system COM. Therefore, this term is ignored.

3.2.3 Optimization problem

The optimization problem, as proposed by Sigmund [42], aims to maximize the end-effector displacement of the system, being subject to a volume constraint. The stopping criteria and settings of the MMA-algorithm are applied as prescribed by Svanberg [43].

In the balanced case, SFB and SMB constraints must also be included, resulting in the optimization problem presented in Equation 3.2.6.

$$\begin{aligned} \min_{\rho} \quad & -u_{ee}(\rho) \\ \text{s.t.} \quad & V = \frac{\sum n_{el} \rho_{el}}{n_{el}} \leq V_{max} \\ & g_{SFB} = u_x^2 + u_y^2 \leq \epsilon_{SFB}^2 \\ & g_{SMB} = R_z^2 \leq \epsilon_{SMB}^2 \end{aligned} \quad (3.2.6)$$

Where u_{ee} is the displacement of the end-effector in the desired direction, V_{max} is the maximum design space density (percentage of elements that are allowed to be solid material), and R_z is an expression for the angular momentum of masses around the COM. Even though this is not the actual rotation or rotational velocity, this value does give an indication of the shaking moment balance of the system.

The values for ϵ are used as a relaxation of the SFB and SMB constraints. These constraints should be equality constraints set to 0, but instead are represented by inequality constraints with a small margin. This slightly enlarges the feasible domain during the iteration process (when compared to an epsilon value of 0) and increases the chance to find better local optima, at the cost of performance in terms of balance.

The value for ϵ_{SFB} physically represents the maximum allowed displacement of the COM of the system in mm, regardless of direction. The value ϵ_{SMB} represents the maximum allowed value of R_z , regardless of direction. In the optimization process, this value is very critical, as a high value will not yield an optimal solution in terms of balance, whereas a very low value may prevent the algorithm from converging. However, through the use of continuation, the constraints can be tightened throughout the optimization process to achieve better performance in terms of balance. This will be discussed later.

Balance functions and sensitivities

For the shaking force balance function, the desired outcome is a low absolute displacement of the center of mass, independent of the direction of that displacement. Therefore, the x- and y-displacements are squared.

For the optimization algorithm, the sensitivities of the objective function and constraints are required. For computational efficiency, it is preferable to determine these analytically. The calculations of these equations and their representation in the code can be found in appendices A and E, respectively.

For easier and faster computation of these gradients, adjoint functions are used for both constraints. The algorithm makes use of a linear finite element model, which is written in matrix representation as $\mathbf{f} = \mathbf{K}\mathbf{u}$. Here \mathbf{f} is the global force vector, \mathbf{K} is the global stiffness matrix and \mathbf{u} is the global displacement vector. Because of this, the function $\boldsymbol{\lambda}(\mathbf{f} - \mathbf{K}\mathbf{u}) = \mathbf{0}$ and can therefore be added to any other function. Note that the derivative $\frac{d\mathbf{f}}{d\rho_i} = \mathbf{0}$, as the vector \mathbf{f} contains the input forces which are independent of the design variables. This will be important when the sensitivities of the objective and constraint functions are calculated. The balance conditions of Equations 3.2.5 with the adjoint function incorporated are given in Equations 3.2.7. The adjoint variable vectors $\boldsymbol{\lambda}$ are different for all subequations shown there.

$$u_x = \frac{\rho \mathbf{A}_x \mathbf{u}}{\sum \rho} + \boldsymbol{\lambda}_x (\mathbf{f} - \mathbf{K}\mathbf{u}) \quad (3.2.7a)$$

$$u_y = \frac{\rho \mathbf{A}_y \mathbf{u}}{\sum \rho} + \boldsymbol{\lambda}_y (\mathbf{f} - \mathbf{K}\mathbf{u}) \quad (3.2.7b)$$

$$R_z = \frac{\rho \mathbf{M} \mathbf{u}}{\sum \rho} + \boldsymbol{\lambda}_z (\mathbf{f} - \mathbf{K}\mathbf{u}) \quad (3.2.7c)$$

The sensitivities derived from these equations are given in Equations 3.2.8. Due to the adjoint formulation one term with adjoint variables is present. In the sensitivity Equation 3.2.8a, the derivatives of u_x and u_y are used. For clarification, only the derivative of u_x is shown in Equation 3.2.8b, with the derivative for u_y being almost the same as the derivative of u_x , except it uses the matrix \mathbf{A}_y instead of \mathbf{A}_x . This term is dependent of λ_x , which is defined in Equation 3.2.8c. The derivatives in these equations are taken with respect to ρ_i , being the density of one of the elements in the design space. In the optimization

code, this calculation is performed for all element densities.

$$\frac{dg_{SFB}}{d\rho_i} = 2 \left(u_x \frac{du_x}{d\rho_i} + u_y \frac{du_y}{d\rho_i} \right) \quad (3.2.8a)$$

$$\frac{du_x}{d\rho_i} = \frac{[\mathbf{A}_x \mathbf{u}]_i}{\sum \rho} - \frac{\rho \mathbf{A}_x \mathbf{u}}{(\sum \rho)^2} - \lambda \frac{\partial \mathbf{K}}{\partial \rho_i} \mathbf{u} \quad (3.2.8b)$$

$$\lambda_x = \frac{\rho \mathbf{A}_x \mathbf{K}^{-1}}{\sum \rho} \quad (3.2.8c)$$

The sensitivities for the shaking moment balance constraint are shown in Equation set 3.2.9. Equation 3.2.9a shows the sensitivity equation of the actual SMB constraint, as presented in Equation 3.2.6. This equation is dependent of the derivative of R_z , which is shown in Equation 3.2.9b. This equation is then dependent on λ_z , which is shown in Equation 3.2.9c. Further elaboration on the derivation of these equations is found in Appendix A.

$$\frac{dg_{SMB}}{d\rho_i} = 2R_z \frac{dR_z}{d\rho_i} \quad (3.2.9a)$$

$$\frac{dR_z}{d\rho_i} = \frac{[\mathbf{M} \mathbf{u}]_i - 2R_z}{\sum \rho} - \lambda \frac{\partial \mathbf{K}}{\partial \rho_i} \mathbf{u} \quad (3.2.9b)$$

$$\lambda_z = \frac{\rho \mathbf{M} \mathbf{K}^{-1}}{\sum \rho} \quad (3.2.9c)$$

Filtering

For preventing mesh-like structures, a density-based filter is applied, as proposed by Andreassen [44]. The design variables are filtered as shown in Equation 3.2.10. The matrix H is defined once in the iteration loop. The calculation of this standard density filter is provided in the code in Appendix E and by Andreassen[44]. This approach is more robust compared to the original sensitivity filter of the 99-line code. The filter radius should be higher than 1 (generally starting at 1.2 or 1.5). Also, the filter radius be used to enforce minimum thicknesses of solid bodies and minimum thicknesses of cavities, and should be taken larger when a larger design space is used.

$$\tilde{\alpha}_e = \frac{1}{\sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei} \alpha_i \quad (3.2.10)$$

The density filter transforms the original set of design variables (α) to a new set of filtered design variables $\tilde{\alpha}$. These filtered design variables will be used for the calculation of the sensitivity functions of the objective function and the balance constraint functions. When this calculation is performed, the resulting sensitivity functions are reverse-filtered to obtain the sensitivity functions with respect to the original design variables, as shown in Equation 3.2.11, by application of the chain rule. These functions are then used in the MMA optimization to obtain new design variables for the next iteration.

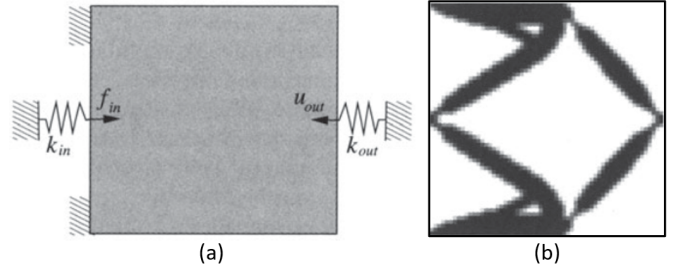


Figure 3.2: Original displacement inverter (a) boundary conditions and (b) resulting geometry [16]

$$\frac{\partial \psi}{\partial x_j} = \sum_{e \in N_j} \frac{\partial \psi}{\partial \tilde{x}_e} \frac{\partial \tilde{x}_e}{\partial x_j} = \sum_{e \in N_j} \frac{1}{\sum_{i \in N_e} H_{ei}} H_{je} \frac{\partial \psi}{\partial \tilde{x}_e} \quad (3.2.11)$$

3.2.4 Design problem, starting conditions and stopping criteria

The design problem which we study is a slightly modified version of the displacement-inverter design problem commonly studied in compliant mechanism topology optimization[6], [16]. This is single input, single output (SISO) mechanism with an output displacement opposite to the input displacement. This system is supported at the left upper and lower corner of the design domain, and results in the geometry as presented in Figure 3.2 (b). The boundary conditions are defined as shown in Figure 3.2 (a), with an input force at the node in the middle of the left boundary of the design domain and a desired output displacement at the middle node of the right boundary of the design domain. At these nodes, a fictional spring is attached to represent a force that would be required at the input and output to actually make the mechanism perform any work rather than just displacement.

For testing the dynamically balanced topology optimization code, the original displacement inverter is, due to its symmetry, inherently force-balanced in the vertical direction, and moment-balanced. This means this particular mechanism is not suitable for testing the SMB constraint, and only partly suitable for testing the SFB constraint.

To verify shaking force balance in both x- and y-direction, as well as shaking moment balance, an asymmetric system is more suitable. Therefore, an off-centered force inverter is designed, where the output displacement is defined slightly above the middle of the design domain at the right boundary. The kinematic boundary conditions are determined as translational constraints in the same corners of the design domain as the original design problem.

To allow the algorithm more design freedom, the design domain has been extended. Outside of the original design domain, which still contains the previously described constraints and input and output displacements, a free design domain is added. For this case, the design domain out-

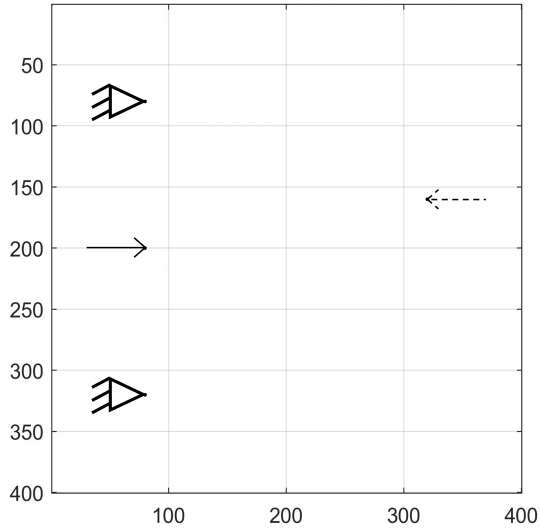


Figure 3.3: Boundary conditions of the design domain: two fixed boundary conditions, a horizontal input force (solid vector) and the desired output motion (dashed vector)

side of the original design domain has a thickness of 20% of the total design domain, at all sides. This value of 20% is chosen because this allows enough space to find a solution, without interfering with the boundaries of the total domain. This results in a design space as shown in Figure 3.3. This means the previously described constraints and input and output nodes are no longer defined at the boundaries of the design domain, but at 20% or 80%. The output displacement node is defined at 40% from the top of the total design domain, and 20% from the right boundary of the total design domain. The other variables used for this optimization are presented in Table 3.1.

The algorithm will generally continue until convergence. This means the maximum change of the design variables is below a prescribed value i.e. the geometry is not changing significantly anymore. This is the case when the objective function will no longer improve and the constraint functions are met. In the options, a minimum or maximum number of iterations can also be set, or a time limit can be applied. This generally yields designs that have not converged yet.

Additional options

Helpdesigner

Because the objective of the system to minimize the end-effector displacement (and in that way, maximize the end-effector displacement to the left) and the starting conditions yield a positive end-effector displacement, the algorithm will start converging to an end-effector displacement of 0. This is a local minimum where no material is present around the end-effector. To solve this, the “helpdesigner” option is included in the algorithm. If this is turned on, the algorithm will plot material along a line through the input and output nodes to enforce a connection to the rest of the material in the design domain. This helps the algorithm to escape the local minimum at

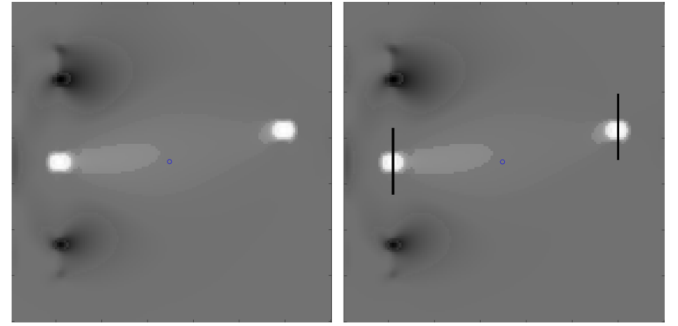


Figure 3.4: Local minimum at the start of the iteration cycle (left) and the helpdesigner option plotting two lines to escape that local minimum (right)

Parameter	Value
Young’s modulus	10 MPa
Poisson’s ratio	0.3
Springs stiffness	0.1 N/mm
Volume fraction	0.2
Penalty factor	3
Filter radius	3
# elements	400x400
Input force	5 N
boundary thickness	0.2
SF slack	0.2
SM slack	20

Table 3.1: Input parameters of the optimization

a displacement of 0. The situation of local minimum is shown in the left image of Figure 3.4. The next iteration with the “helpdesigner” active is shown in the right image of Figure 3.4.

Continuation

Another option in the optimization is continuation. This allows the user to start the optimization problem with relatively relaxed constraints, while tightening those constraints later. In the case of dynamic balance, this allows the algorithm a lot of design freedom in early iterations, while still achieving an increasingly well-balanced mechanism in later iterations.

Starting point

By default, the algorithm will have a “grey start” as starting point for the optimization. This means the design variables are all equal to the prescribed volume fraction. If necessary, another density field could be used as starting point. In the case of designing dynamically balanced mechanisms, an unbalanced mechanism can be a very good starting point for designing a balanced mechanism. If this is the case, the starting point will be a combination of the input density field and a grey start, with the design variables of white input elements having a low value and the black elements having a higher value, still all between 0 and 1.

3.3 Results

In this section the geometries of the optimization runs described in Section 3.2 are shown. These geometries are post-processed in SolidWorks to obtain mechanisms with smooth boundaries to eliminate obsolete material and stress concentrations due to the rough boundaries of the non-conforming mesh in the optimization algorithm. The post-processed geometries are then simulated using a conforming mesh in COMSOL to analyze the performance of the mechanisms, both in terms of their output displacement and their dynamic balance properties. This simulation also serves as a verification of the kinematic behaviour of the mechanism in the optimization algorithm.

3.3.1 Geometries

The resulting geometries for the unbalanced and balanced case are shown in Figures 3.5 and 3.6, respectively. In both figures, the geometry is shown in the left image. In the right image, the original geometry is shown in black with the deformed geometry plotted on top of it in red. A blue circle indicates the position of the COM of the mechanism in undeformed state, with the vector originating in the middle of the circle indicating the displacement of the COM in its deformed state. At the end effector, another vector is shown, indicating the displacement of the end-effector.

The only objective of the algorithm is to maximize the horizontal displacement pointing in the negative x-direction (left), regardless of vertical displacements. This results in a large parasitic vertical displacement of the end-effector and the COM in the unbalanced case, whereas the direction of the displacement of the end-effector is more horizontal in the balanced case.

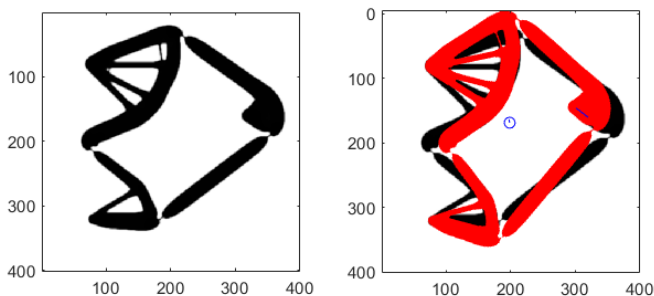


Figure 3.5: Unbalanced geometry in MATLAB, undeformed (left) and deformed (right)

The performance of the mechanism is evaluated in both the MATLAB model and the COMSOL simulation. The resulting values for the displacements of the COM, the input and the end-effector are shown in Table 3.2. Note that the positive y-direction in the MATLAB model is vertically downward, whereas the positive y-direction in the COMSOL simulations is vertically upward. The values for displacements are given in mm. Also, the position of the COM and a quantification for the angular momentum (R_z) are given, with the unit of R_z being mm^2 .

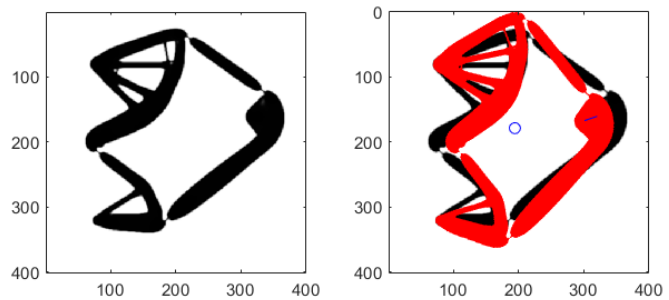


Figure 3.6: Balanced geometry in MATLAB, undeformed (left) and deformed (right)

Table 3.2: Calculated results in mm from MATLAB (R_z in mm^2)

Property	Unbalanced	Balanced	Difference
Ux_{in}	27.682	27.602	-0.3%
Uy_{in}	0.062584	0.49203	+686.2%
Ux_{out}	-21.258	-21.146	-0.5%
Uy_{out}	-16.126	7.3924	-54.2%
$UCOM_x$	-1.3976	-0.16116	-88.5%
$UCOM_y$	-5.9021	-0.11843	-98.0%
x_{COM}	198.65	194.35	-
y_{COM}	168.62	178.72	-
R_z	-556.76	-19.999	-96.6%
$B\&W_\alpha$	99.3%	99.4%	+0.07%
$B\&W_\rho$	93.7%	93.7%	-0.06%

The shown differences are in percentages of the absolute value of the balanced case compared to the absolute value of the unbalanced case. According to the MATLAB calculations, the shaking forces are reduced by 96.7% (the absolute value of the combined x- and y-components of the COM displacements from the table) and the shaking moments are reduced by 96.6%. The relative difference of the COM position is not shown as this is not relevant to the performance of the system.

Apart from displacements, R_z and the COM position, the so-called black-and-white fraction (B&W) of the design is reported. The two values indicate the B&W fractions of the design variables (α) and densities (ρ). This is shown as the percentage of design variables or densities being below 1% material or above 99% material, with respect to the total number of elements. Intermediate design variables or “gray” material is considered non-physical in this methodology. A low value of B&W indicates a large percentage of this non-physical material, and is therefore undesired. A little gray material is expected in the density (ρ) case, as the filtering will introduce more intermediate design variables at boundaries between solid and void elements.

The balanced mechanism has a slightly lower output displacement than the unbalanced mechanism. Also the input displacement is slightly lower when the same force is applied. This is explained by the balanced system storing more strain energy compared to the unbalanced case. The output displacement with respect to the input displacement

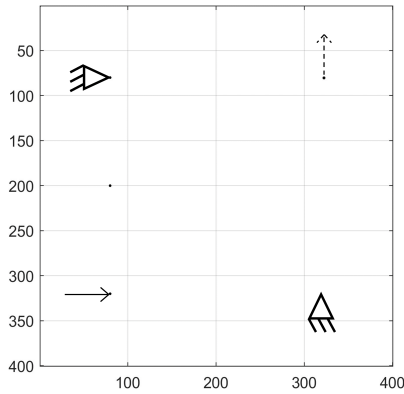


Figure 3.7: Design problem of an xy-manipulator

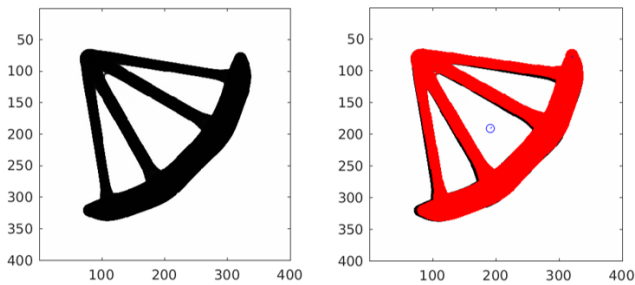


Figure 3.8: Unbalanced xy-manipulator

ment is slightly higher in the balanced case, however.

A second mechanism is also designed. The boundary conditions are shown in Figure 3.7. The design problem concerns a manipulator with an input force in x-direction at the bottom left corner, an output motion in the y-direction in the top right corner and fixed kinematic boundary conditions in the other two corners. Again, a 20% design space around this internal design space is applied, and fictional springs are applied at the input and output DOF to simulate actual work being done by the mechanism. This design problem yields an unbalanced geometry, as shown in Figure 3.8 and a balanced geometry in Figure 3.9. The unbalanced geometry is a simple lever, rotating around one of the fixed boundary conditions. The balanced geometry is slightly more complex. The mechanism shows an added mass, which is used to counteract the angular momentum of the mechanism.

In the case of the balanced geometry, the algorithm has

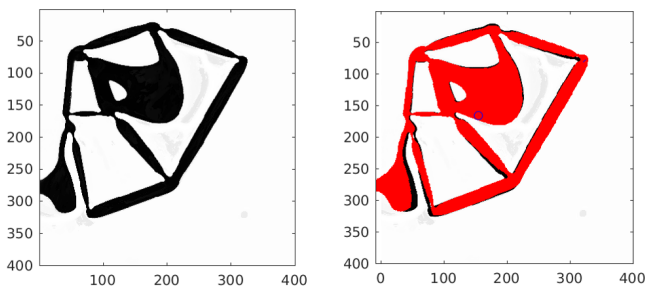


Figure 3.9: Balanced xy-manipulator

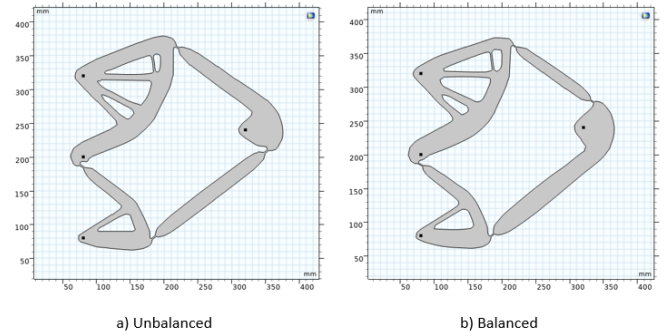


Figure 3.10: Unbalanced(a) and Balanced(b) systems with constraints and input/output nodes in COMSOL

not converged to a solution yet, but was stopped due to a computation time limit. The resulting geometry has a B&W value of only 12.2%, which clearly shows the convergence has not yet been reached. The geometry is shaking forced and shaking moment balanced, but due to the low B&W value, this cannot be guaranteed.

Due to the lack of convergence, this geometry will not be analyzed further. However, dynamic behaviour of the added mass will likely have a large impact on the balance of the mechanism when dynamic behaviour is taken into account. Other geometries like this one can be found in Appendix D.

3.3.2 Verification and dynamic analysis

The resulting geometry from the MATLAB optimization is post-processed in SolidWorks. This is done by hand, because automatic tools in SolidWorks were not able to precisely follow the optimized geometry. The resulting mechanisms are converted to 2-dimensional .dxf files which will be used as input geometries in the COMSOL analysis. The 400x400 elements design space is converted to a 400x400mm geometry.

The post-processed geometries are imported into the COMSOL simulation environment after which kinematic boundary conditions are added, analogous to the original design problem. The input and output nodes are also positioned accordingly. After importing, the geometry is slightly repositioned to ensure the geometry and the boundary conditions are correctly aligned. This is required to cope with possible errors in the conversion process, such as the scaling errors. This is done by inspecting the outer boundaries of the geometries in the MATLAB environment and the COMSOL environment. The simulation is performed in 2D, and similar to the optimization algorithm, plane stress is assumed and the out of plane thickness is set to 1mm.

To quantify the performance of the design methodology, the resulting geometry is simulated in several ways: A static study, a frequency-domain study and a time-dependent (transient) study. The first simulation is a static study to verify the MATLAB analysis and the COMSOL analysis are comparable. The applied force is identical to the ap-

Table 3.3: Results from COMSOL simulation in mm (R_z in mm^2)

Property	Unbalanced	Balanced	Difference
U_{xin}	27.877	27.643	-0.8%
U_{yin}	-0.069968	-0.34330	+390.7%
U_{xout}	-21.436	-21.447	+0.0%
U_{yout}	16.142	-7.6955	-52.3%
U_{COMx}	-1.3886	-0.12250	-91.2%
U_{COMy}	5.9485	0.10067	-98.3%
x_{com}	198.41	194.11	-
y_{com}	233.15	221.43	-
R_z	566.95	-2.8309	-99.5%

plied force in the MATLAB model, so the results in the COMSOL model should be approximately equal to the MATLAB results. This study can however only be used to determine the kinematic behaviour of the system and to verify the COM displacements; not to determine the actual shaking forces and moments.

The second simulation is a frequency-domain study. This is used to determine the natural frequencies and eigenmodes of the system. The main goals of this analysis are to gain insight in the behaviour of the mechanism, and to ensure the transient study will not be influenced by unexpected dynamical behaviour of the system.

The third simulation is a transient analysis, where the dynamic behaviour of the system will be taken into account. This simulation can be performed quasi-statically, or with inertial terms taken into account. Both simulations are performed, because both serve a different purpose. A quasi-static analysis can be used to verify the COM displacement approach as a guideline for SFB and SMB, as described in Section 3.2. A simulation with inertial terms will give more insight in the actual dynamic behaviour of the systems which would be designed with this method.

Static Study

The resulting values in the static study for the displacements of the input node, end effector and COM are shown in Table 3.3. The deformed mechanisms according to the static COMSOL simulation are shown in Figure 3.11. The color legend shows the displacement magnitude. These deformed systems show deformations for approximately the same system with the same boundary conditions and input forces as the systems in Figures 3.5 and 3.6.

The results of the static study in Table 3.3 should be similar to the results of the topology optimization algorithm in Table 3.2. In Appendix C and table C.1, a clearer comparison is made between the results of the static analysis on the geometry according to the topology optimization model and the COMSOL simulation. This shows the translation from the non-conforming mesh and analysis in the optimization model to the analysis in COMSOL is performed correctly.

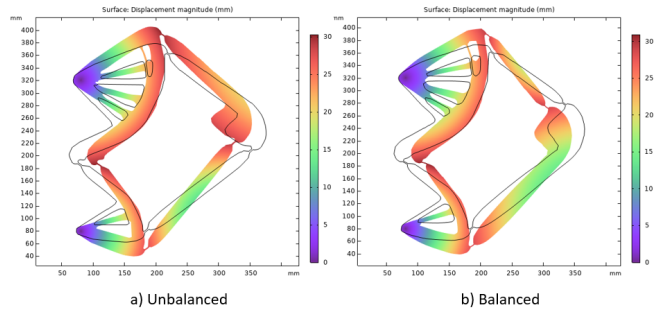


Figure 3.11: Deformed unbalanced (a) and balanced (b) force inverter mechanisms in COMSOL

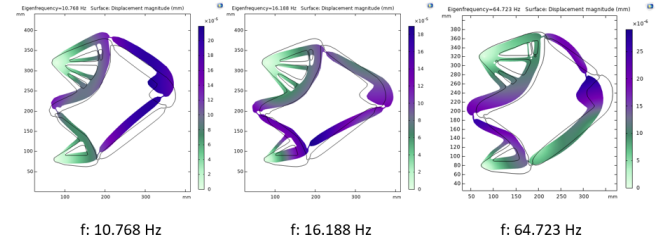


Figure 3.12: Eigenmodes of the 3 dominant natural frequencies of the balanced displacement inverter.

Frequency Domain Study

For the frequency domain study, the same input force of 5N is applied, and the mechanism is actuated harmonically at frequencies logarithmically ranging from 0.01Hz to 100Hz. Also, an eigenfrequency study is performed, which determines the natural frequencies and corresponding eigenmodes of the system. This shows a series of natural frequencies of the mechanisms, which are provided in Table 3.4.

The frequency domain study also shows an overview of the reaction forces in x- and y-direction for different frequencies. This gives a clear insight in the balance of the system at the natural frequencies. The most dominant eigenmodes in terms of balance are the second eigenmode at 16.176 Hz, and the sixth at 64.723 Hz. The contribution of the first eigenmode is, however, clearly visible in Figure 3.15 as well. The corresponding eigenmode is the vertical displacement of the entire mechanism, as it is not constrained in the vertical direction apart from the fixed boundary conditions. This eigenmode can cause significant noise in the mechanism as the optimization methodology does not take such behaviour into account.

Eigenmode analysis then showed that the second eigenmode is the horizontal desired deformation of the mechanism, which should cause less noise compared to the first eigenmode as it the mechanism is balanced for this motion. However, high peak is visible in the reaction forces plot in Figure 3.13. This is explained by the springs that are attached to the input and output node, which will exert high reaction forces when the nodes are heavily displaced. When resonance appears due to the excitation of the system and the eigenmode coinciding, this will be the

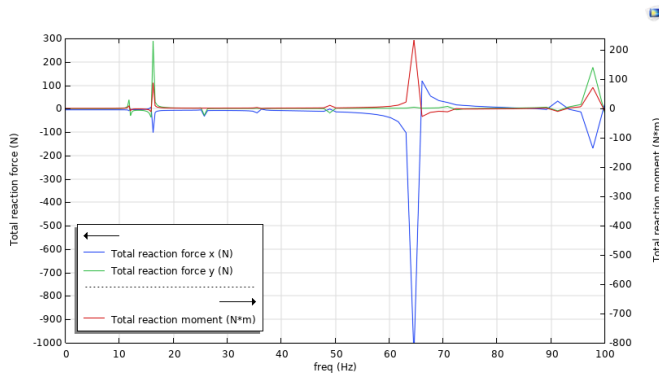


Figure 3.13: Frequency domain study reaction forces and moments (balanced mechanism)

Unbalanced	Balanced
10.768	11.872
16.188	16.176
23.798	25.698
24.692	35.783
54.334	49.029
63.005	64.723

Table 3.4: First eigenfrequencies of the force inverter[Hz]

case.

The third to fifth eigenmodes are relatively small, and the sixth eigenmode slightly above 60 Hz shows a peak on the frequency domain analysis. This eigenmode has an even higher amplitude, which is explained in the exact same way as the second eigenmode. However, now the springs are extended in the same direction, causing the spring forces not to partially cancel each other out but adding up. This causes much higher reaction forces, especially in x-direction. However, the system will be very stiff in this direction, which explains why the transient study will not show this behaviour. The eigenmodes of the balanced mechanism can be seen in Figure 3.12. More information on this and the form of the eigenmodes is available in Appendix C.

Time-dependent studies

Knowing the eigenfrequencies of the system, the system can be actuated at a frequency which is not a natural frequency to gain insight in common system behaviour. For example, a frequency of 2 Hz is chosen. In the quasistatic simulation, the system is analyzed without incorporating inertial forces. COM accelerations reach up to 0.023 m/s^2 for the balanced mechanism and 0.46 m/s^2 for the unbalanced mechanism. For the input force of 5N at a frequency of 2Hz, the resulting COM accelerations are shown in Figure 3.14. The accelerations are proportional to the previously determined displacements, and significantly lower in the balanced mechanism compared to the unbalanced mechanism.

The study with inertial terms shows more chaotic behaviour compared to the quasistatic study. Higher fre-

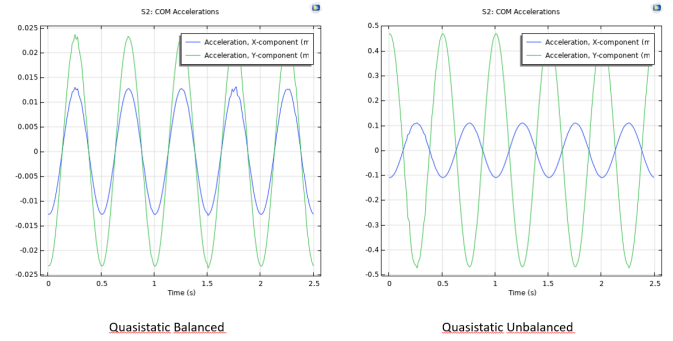


Figure 3.14: Quasistatic accelerations from a COMSOL simulation

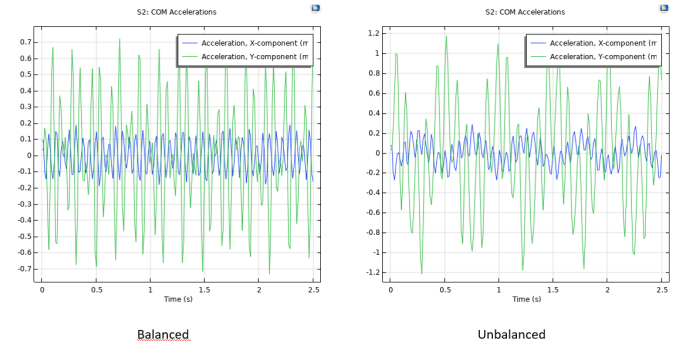


Figure 3.15: Accelerations from COMSOL simulation with inertial terms

quency terms are present, and both in the balanced and the unbalanced case dominant in magnitude over the accelerations in the quasistatic study. Relative to these vibrations, the quasistatic accelerations of the system COM are very small in the balanced case, whereas they share the same order of magnitude in the unbalanced case.

To gain more insight in the behaviour of the system, a step input is applied on the input node. An input force of 5N is applied in a timestep of 0.1s, after which the system can still vibrate. A quasistatic analysis is not performed here, as this does not show any new information. An analysis with a step input force with inertial terms results in the linear and angular momentum plots of the unbalanced and balanced mechanisms in Figures 3.16 and 3.17, respectively.

The linear momentum in the step response shows a few interesting things. As the system is not damped, the vibrations do not visibly decay. Furthermore, multiple natural frequencies are present in the response in x-direction, but in the linear momentum in y-direction and the angular momentum the first natural frequency appears dominant. This can be explained by the lack of balancing or constraint in this direction, and is visible in the figure through the relatively high linear momentum in y-direction compared to the x-direction.

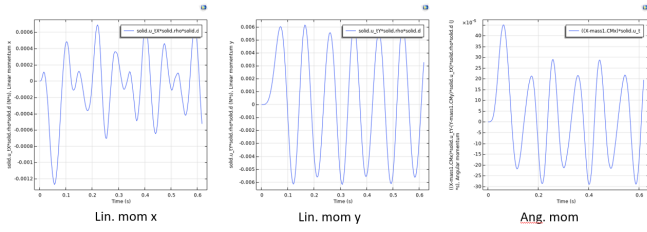


Figure 3.16: Linear and angular momentum of the unbalanced displacement inverter after a step input

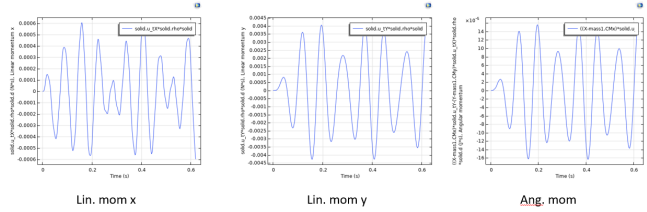


Figure 3.17: Linear and angular momentum of the balanced displacement inverter after a step input

3.4 Discussion

The topology optimization algorithm in MATLAB has been able to solve the provided design problem and synthesize a compliant mechanism with lower shaking forces and shaking moments. A quasistatic simulation of the mechanism with a harmonic input of 2 Hz proved the relation between shaking forces and COM displacements. A simulation with inertial terms proved dynamic behaviour of the system still largely contributes to the shaking forces and shaking moments in the system, however. In this section, these mechanisms and simulations are discussed and improvements on the approach are proposed.

Comparison optimization algorithm with simulation

As shown in the static analysis, the system is simulated in both MATLAB and COMSOL. The conversion from MATLAB to COMSOL is performed in SolidWorks. When comparing the results of Table 3.2 and Table 3.3, slight differences become apparent. Most values differ approximately 1% or less when comparing the COMSOL and MATLAB simulations. Three main reasons could be found for this.

A reason for small differences could be a difference in the finite element analysis through the approach or numerical errors. In matlab, a 400x400 element non-conforming mesh is used, whereas the COMSOL analysis uses a conforming (triangular element) mesh with significantly coarser elements in the large rigid bodies and significantly finer elements around smaller members such as the flexure joints. The MATLAB FEM analysis uses approximately 40000 elements for the analysis of the mechanism (approximately all resulting black and grey elements in a grid when the input values in Table 3.1 are used). The COMSOL analysis uses between 10000 and 15000 elements of varying

sizes. This will very likely result in slightly different values from MATLAB. More information on the COMSOL FEM analyses is presented in Appendix C.

Another reason could be the resulting geometry from optimization algorithm may not have a perfect black-and-white distribution of the densities. Especially when the optimization algorithm has not converged within the given timespan or number of iterations, large gray areas may be present. Masses that appear black may be gray (and therefore contributing less to the COM position and displacement), or lightgray masses may be present where no material is present in the transformed mechanism (therefore secretly contributing to the COM position and displacement). This may also slightly affect the stiffness of the system, and thereby the kinematic behaviour. To verify this, the black and white fractions are compared in Table 3.2, which shows the unbalanced and balanced mechanism both have a mostly black and white distribution. This therefore does not explain such large differences nor a change in kinematic behaviour.

Finally, the transformation of a filtered design domain to an actual mechanism without blurry boundaries may result in slight differences between the mechanism in MATLAB and the transformed mechanism in COMSOL. This operation is performed by hand because automatic techniques in SolidWorks did not result in good approximations of the designed mechanism. This is, however, still prone to errors. In Appendix C, Table C.1 shows the values from the analysis of an automatically converted geometry and a by hand converted geometry. Although both are not exactly the same as the results from the optimization algorithm, this table clearly shows the impact of the transformation of the mechanism. For more reliable results, a more accurate and repeatable approach should be applied.

Eigenmodes and dynamic behaviour

The static and quasistatic time-dependent COMSOL simulations show a clear relation between the quasistatic COM accelerations and the static COM displacements. However, when inertial terms are included, the COM accelerations grow significantly. Large vibrations with a higher frequency become apparent, with amplitudes exceeding the quasistatic COM Accelerations. The high amplitude noise has a frequency around 11 Hz, which shows the first eigenmode is excited. This eigenmode consists of a displacement of the entire mechanism in vertical direction, which also explains why the undesired vibrations are very large in the y-direction compared to the vibration in the x-direction.

The vibrations in the x-direction can be caused by the second eigenmode, which is the same as the desired deformation of the mechanism. This explains why the magnitude of this vibration is relatively low, and means a higher output displacement can be achieved when the mechanism is actuated at the corresponding natural frequency.

This clearly shows that for properly balanced compliant

mechanisms using this approach, eigenmode management is required in the design phase. Another option could be damping the system, but obviously this is not preferred. This eigenmode management can be performed in two ways. Low frequency eigenmodes can be avoided in the design process, thereby only allowing higher frequency eigenmodes which may never be excited. Another option would be to take the vibration and amplitude of eigenmodes into account, and balance those along with the quasistatic balancing. An example of modal balancing is found in the paper of Nijdam on balancing of flexible beams [8]. Here, some conditions for dynamically balancing vibrating flexures are shown. However, incorporating this in a topology optimization algorithm may prove difficult as topology optimization generally considers static analysis. Solutions for this could be found in the paper by Venini and Pingaro on static and dynamic topology optimization [24], where an approach for incorporating dynamics in the design of Multi-Input-Multi-Output mechanisms is discussed. Approaches on constraining eigenfrequencies can be found in the works of Pedersen [45], or Bendsøe and Sigmund [16].

Geometric nonlinearities

In the topology optimization algorithm, geometric nonlinearities are thus far not taken into account. For some cases, like the geometry of the offset force inverter, this yields small errors. In other cases, it may lead to infeasible solutions with a very small range of motion. For compliant mechanism design, it is therefore advised to incorporate geometric nonlinearities in the topology optimization algorithm, as proposed by Bendsøe and Sigmund [16]. Two problems need to be solved for large ranges of motion. The first is the kinematic behaviour of the mechanism which may yield desired results for small displacements but not for larger ranges of motion (for example the snap-through of a joint). A second issue could be the collision of two bodies in the mechanism, constricting further movement.

Other recommendations

Apart from the recommendation to include eigenfrequencies in the topology optimization problem, other recommendations for improvement of this code or future work also arise from this project. When design problems with a smaller feasible domain are attempted, the algorithm does not always converge. The use of other optimization algorithms, such as GCMMA [46], may increase the chance or speed of convergence of the optimization problem.

Also, this methodology is applicable for 3D problems as well, rather than just 2D problems. However, this can become computationally intensive, and may therefore require speed improvements to the rest of the code. This would however be an interesting step in further research into dynamically balanced compliant mechanisms.

3.5 Conclusion

The goal of this research was to design a topology optimization algorithm in MATLAB which is capable of synthesizing dynamically balanced compliant mechanisms. A methodology is proposed, based on the inherent balancing method by van der Wijk [9]. This method achieves dynamic balance by ensuring the linear momentum of the COM is stationary, as well as the angular momentum about the COM, throughout the entire range of motion.

An algorithm is designed with dynamic balancing capabilities, based on the 99-line code by Bendsøe and Sigmund [16], the 88-line code by Andreassen [44], and the MMA code by Svanberg [43]. This algorithm uses a linear finite element model to synthesize compliant mechanisms, for any given set of boundary conditions.

This algorithm is used to design two displacement-inverters with an output displacement at an offset with respect to the input displacement. One of these displacement inverters is dynamically balanced, the other is not. The kinematic behaviour of the geometries is validated using a conforming mesh instead of a non-conforming mesh in COMSOL, after which the performance of these mechanisms is analyzed in terms of the desired objective of the mechanism and the dynamic balance properties.

The resulting mechanisms satisfy the requirements to which the optimization algorithm should have synthesized them. However, the used approach did not take into account the dynamic responses of the mechanisms, which show unexpected vibrations when the systems are dynamically analyzed. Recommendations are presented to incorporate this behaviour in the design process to reduce these effects.

Chapter 4

Discussion

This MSc. thesis showed an overview of available design methods for the synthesis of dynamically balanced or compliant mechanisms. Several possible approaches are named, of which topology optimization was the most promising. This chapter will discuss the strengths and weaknesses of the topology optimization algorithm which is presented in Chapter 3. Finally, recommendations will be done for further research and development of the algorithm, on top of the recommendations in the chapter itself.

4.1 Strengths of the proposed topology optimization algorithm

The proposed topology optimization algorithm has proven to be able to synthesize mechanisms with better dynamic balance properties than without the dynamic balance constraints. The strengths of the topology optimization algorithm can be summarized as versatility and the ability to find simple solutions to otherwise difficult problems. This can more specifically be described by the following strengths:

- **No predefined geometry required:** General dynamic balancing methods consider dimensions of predetermined geometries and positions of COM's of the rigid bodies in those mechanisms. The proposed topology optimization algorithm does not require a predetermined geometry as a starting point, but is often able to generate a mechanism from a blank starting point.
- **Not limited to rigid body linkages:** Dynamic balancing methods mostly consider rigid body linkages. The proposed methodology is able to design rigid body linkages with lumped compliant connections (rather than hinges), but may also use distributed compliance and is therefore more versatile.
- **Simple solutions for SMB:** Most dynamic balancing methods require symmetry or large auxiliary mechanisms to realize SMB. The topology optimization algorithm is able to find solutions that are simpler and may require only little changes to the unbalanced solution to a design problem.
- **Mass redistribution instead of addition:** In several of the existing dynamic balancing methods, mechanisms are balanced by adding mass, rather than redistributing mass.

Because a topology optimization algorithm is redistributing mass instead of just adding it, leaving a lighter resulting geometry.

- **2D or 3D:** The proposed methodology is designed in a 2D environment for computational purposes. It would however require only little changes to perform a 3D optimization, as the way the functions are determined could easily be extended to 3 dimensions.

4.2 Weaknesses of the proposed topology optimization algorithm

The topology optimization algorithm is not a perfect solution to every problem, however. Some weaknesses consider convergence issues, high computation times or simply infeasible problems. Also, it should be taken into account that topology optimization results often need post-processing, which may introduce new disturbances.

- **Does not always yield solutions:** Some sets of boundary conditions yield no feasible results, either because there are none or because the algorithm cannot converge to a feasible local minimum. This could be caused by the constraints being too tight, but also by the highly nonlinear nature of the optimization problem and the extraordinary large amount of possible outcomes.
- **Does not incorporate natural frequencies and eigenmodes:** As far as dynamic balance is achievable with the inherent balancing method, the algorithm performs very well. However, when elastic mechanisms with free DOFS are concerned, such as the compliant displacement inverter, eigenmodes play an important role and could cause significant shaking forces and moments.
- **Computational intensity:** Topology optimization is a computationally intensive process. The geometries in this thesis took hours to converge to a solution, while the problem was relatively small (2D, only 400x400 elements). As computation time grows exponentially with more elements, a proper 3D optimization may be computationally very expensive.

4.3 Recommendations and further research

As discussed, some weaknesses of the topology optimization algorithm are based on long computation times and the lack of modal analysis in the design phase. These issues, among other, pose a great starting direction for further research.

- **Increase versatility:** Many mechanisms in the real world are 3-dimensional, so the topology optimization algorithm should be extended to yield 3D solutions. The methodology is suitable for extension to 3D, but serious effort has to be put into computational efficiency. Also, a multi-input-multi-output optimization problem would be an interesting extension of the current methodology. Finally, some optimization attempts did not yield solutions. Looking into other optimization algorithms such as GCMMA[46] may help with convergence issues to solve more difficult problems as well.
- **Incorporate modal balancing:** Modal balancing is an essential part of dynamically balanced compliant mechanism design. The basic principles are explored by Nijdam [8], but these principles may still be hard to incorporate in a topology optimization algorithm. A good starting point for this would be the dynamic topology optimization review by

Venini and Pingaro [24], as they describe ways take dynamic effects into account during the topology optimization process.

- **Controlling all DOF's:** One of the major causes of noise in the designed displacement inverter is the first eigenmode, which consists of a translation of the mechanism along the unconstrained and undesired vertical DOF. This should be prevented by for example adding more constraint functions, adding more boundary conditions to prevent such DOF's, or by defining a Multi-Input-Multi-Output optimization problem.
- **Nonlinearity:** Nonlinearities should be taken into account. Bodies interaction (especially common for large ranges of motion) or geometric limits to the range of motion of bodies should be considered in the design phase.

Chapter 5

Conclusion

In this paper, a comprehensive design method for dynamically balanced compliant mechanisms is developed.

An evaluation of existing methods for designing compliant mechanisms, as well as an evaluation of methods for designing dynamically balanced mechanisms is performed. This yielded 4 possible combinations of methods that could be used to develop one comprehensive design method. Out of these possible combinations, a topology optimization algorithm with the inherent balance method is selected as the most promising.

A topology optimization methodology is defined, and an algorithm is designed. This algorithm is based on the 99-line code by Bendsøe and Sigmund [16], the 88-line method by Andreassen [44] and the MMA algorithm by Svanberg [43]. The dynamic balancing approach is a constant linear and angular momentum approach, based on the inherent balancing method by van der Wijk [9].

The algorithm is tested and proven with the displacement inverter, which is a classic topology optimization problem. The original displacement inverter is however already shaking force balanced in y-direction and shaking moment balanced, so as such not suitable to prove the strength of the methodology. Therefore, this displacement inverter is slightly adjusted to introduce an imbalance.

Two geometries are synthesized which perform the desired function of which one is dynamically balanced both in terms of shaking forces and shaking moments. These are analyzed using finite element analysis, which showed the approach was effective. However, noise is still present due to natural frequencies and corresponding eigenmodes in the system, as this was not incorporated in the design process. Possible solutions for this are proposed and recommendations are done for further research and improvement of the algorithm.

The goal of this thesis, to develop a comprehensive method to synthesize dynamically balanced compliant mechanisms, has been fulfilled. A topology optimization algorithm is designed, tested and verified and shows potential for the synthesis of dynamically balanced compliant mechanisms. The algorithm can still be improved, but the potential has been proven and the desired mechanisms can successfully be designed.

Bibliography

- [1] L. L. Howell, *Compliant Mechanisms.pdf*. Wiley, 2001, ISBN: 0-471-38478-X.
- [2] T. Shimizu, H. Funakoshi, T. Kobayashi, and K. Sugimoto, “Reduction of noise and vibration in drum type washing machine using Q-learning,” *Control Engineering Practice*, vol. 122, no. February, p. 105 095, 2022, ISSN: 09670661. DOI: 10.1016/j.conengprac.2022.105095. [Online]. Available: <https://doi.org/10.1016/j.conengprac.2022.105095>.
- [3] V. van der Wijk, S. Krut, F. Pierrot, and J. L. Herder, “Design and experimental evaluation of a dynamically balanced redundant planar 4-RRR parallel manipulator,” *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 744–759, 2013, ISSN: 0278-3649. DOI: 10.1177/0278364913484183.
- [4] L. L. Howell, S. P. Magleby, and B. M. Olsen, *Handbook of Compliant Mecahnisms*. 2019, ISBN: 9781119953456.
- [5] J. A. Gallego and J. L. Herder, “Synthesis methods in compliant mechanisms: An overview,” in *Proceedings of the ASME Design Engineering Technical Conference*, vol. 7, 2009. DOI: 10.1115/DETC2009-86845.
- [6] X. Zhang and B. Zhu, *Topology Optimization of Compliant Mechanisms*. 2018, pp. 1–24, ISBN: 9789811304323. [Online]. Available: https://doi.org/10.1007/978-981-13-0432-3_1.
- [7] J. J. de Jong, B. E. Schaars, and D. M. Brouwer, “The influence of flexibility on the force balance quality: A frequency domain approach,” in *European Society for Precision Engineering and Nanotechnology, Conference Proceedings - 19th International Conference and Exhibition, EUSPEN 2019*, 2019, pp. 546–549, ISBN: 9780995775145.
- [8] L. Nijdam, “Dynamic balance principles based on a flexible beam for the synthesis of dynamically balanced compliant mechanisms,” TU Delft, Tech. Rep., 2021.
- [9] V. Van der Wijk, “Methodology for analysis and synthesis of inherently force and moment-balanced mechanisms - theory and applications,” Ph.D. dissertation, University of Twente, Enschede, 2014, p. 251, ISBN: 9789036536301. DOI: 10.3990/1.9789036536301. [Online]. Available: <https://doi.org/10.3990/1.9789036536301>.
- [10] J. A. Gallego, “Statically Balanced Compliant Mechanisms: Theory and Synthesis,” Ph.D. dissertation, TU Delft, 2013, pp. 1–219, ISBN: 9789461862150. [Online]. Available: <http://repository.tudelft.nl/view/ir/uuid:081e3d0e-173b-4a36-b8d6-c29c70eb7ae3/>.
- [11] R. M. Panas, F. Sun, L. Bekker, and J. B. Hopkins, “Combining cross-pivot flexures to generate improved kinematically equivalent flexure systems,” *Precision Engineering*, vol. 72, no. December 2020, pp. 237–249, 2021, ISSN: 01416359. DOI: 10.1016/j.precisioneng.2021.05.001. [Online]. Available: <https://doi.org/10.1016/j.precisioneng.2021.05.001>.

- [12] S. Henein, P. Spanoudakis, S. Droz, L. I. Myklebust, and E. Onillon, “Flexure pivot for aerospace mechanisms,” *European Space Agency, (Special Publication) ESA SP*, no. 524, pp. 285–288, 2003, ISSN: 03796566.
- [13] N. B. Hubbard, J. W. Wittwer, J. A. Kennedy, D. L. Wilcox, and L. L. Howell, “A novel fully compliant planar linear-motion mechanism 57008,” *Proceedings of the ASME Design Engineering Technical Conference*, vol. 2 A, pp. 1–5, 2004. DOI: 10.1115/detc2004-57008.
- [14] M. Ling, J. Cao, Z. Jiang, and J. Lin, “A semi-analytical modeling method for the static and dynamic analysis of complex compliant mechanism,” *Precision Engineering*, vol. 52, 2018, ISSN: 01416359. DOI: 10.1016/j.precisioneng.2017.11.008.
- [15] J. B. Hopkins and M. L. Culpepper, “Synthesis of precision serial flexure systems using freedom and constraint topologies (FACT),” *Precision Engineering*, vol. 35, no. 4, 2011, ISSN: 01416359. DOI: 10.1016/j.precisioneng.2011.04.006.
- [16] M. P. Bendsøe and O. Sigmund, *Topology Optimization: Theory, Methods and Applications*. 2004, p. 381, ISBN: 978-3-642-07698-5. DOI: 10.1007/978-3-662-05086-. [Online]. Available: <https://link.springer.com/book/10.1007/978-3-662-05086-6#toc>.
- [17] JPE, *BEAM THEORY: BENDING*. [Online]. Available: <https://www.jpe-innovations.com/precision-point/beam-theory-bending/>.
- [18] J. Wang, Y. Yang, R. Yang, P. Feng, and P. Guo, “On the validity of compliance-based matrix method in output compliance modeling of flexure-hinge mechanism,” *Precision Engineering*, vol. 56, no. February, pp. 485–495, 2019, ISSN: 01416359. DOI: 10.1016/j.precisioneng.2019.02.006. [Online]. Available: <https://doi.org/10.1016/j.precisioneng.2019.02.006>.
- [19] A. (U. Prakash, *The theorem of least work*, West Lafayette, Indiana, 2012. [Online]. Available: https://engineering.purdue.edu/~ce474/Docs/The%20Theorem%20of%20Least%20Work_2012.pdf.
- [20] J. W. Ryu, D. G. Gweon, and K. S. Moon, “Optimal design of a flexure hinge based XY θ wafer stage,” *Precision Engineering*, vol. 21, no. 1, pp. 18–28, 1997, ISSN: 01416359. DOI: 10.1016/s0141-6359(97)00064-0.
- [21] J. D. Deaton and R. V. Grandhi, “A survey of structural and multidisciplinary continuum topology optimization: Post 2000,” *Structural and Multidisciplinary Optimization*, vol. 49, no. 1, pp. 1–38, 2014, ISSN: 1615147X. DOI: 10.1007/s00158-013-0956-z.
- [22] D. Walker, D. Liu, and A. Jennings, “Topology optimization of an aircraft wing,” *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, no. January, pp. 1–8, 2015. DOI: 10.2514/6.2015-0976.
- [23] S. Lu, H. Ma, L. Xin, and W. Zuo, “Lightweight design of bus frames from multi-material topology optimization to cross-sectional size optimization,” *Engineering Optimization*, vol. 51, no. 6, pp. 961–977, 2019, ISSN: 10290273. DOI: 10.1080/0305215X.2018.1506770.
- [24] P. Venini and M. Pingaro, “Static and dynamic topology optimization: an innovative unifying approach,” *Structural and Multidisciplinary Optimization*, vol. 66, no. 4, pp. 1–27, 2023, ISSN: 16151488. DOI: 10.1007/s00158-023-03528-6. [Online]. Available: <https://doi.org/10.1007/s00158-023-03528-6>.
- [25] J. Emmanuel Ayala-Hernandez, S. Briot, and J. Jesús Cervantes-Sánchez, “Structural Topology Optimization of Reactionless Four-Bar Linkages,” *Journal of Mechanical Design*, vol. 144, no. 11, 2022, ISSN: 10500472. DOI: 10.1115/1.4054876.
- [26] R. S. Berkof and G. G. Lowen, “A new method for completely force balancing simple linkages,” *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, vol. 91, no. 1, 1969, ISSN: 15288935. DOI: 10.1115/1.3591524.

- [27] V. A. Kamenskii, “On the question of the balancing of plane linkages,” *Journal of Mechanisms*, vol. 3, no. 4, pp. 303–322, 1968, ISSN: 00222569. DOI: 10.1016/0022-2569(68)90006-2.
- [28] J. L. Herder, “Static and Dynamic Balancing of Robots,” in *Encyclopedia of Robotics*, 2020. DOI: 10.1007/978-3-642-41610-1_{156-1}.
- [29] V. Arakelian, M. Dahan, and M. Smith, “A Historical Review of the Evolution of the Theory on Balancing of Mechanisms,” in *International Symposium on History of Machines and Mechanisms Proceedings HMM 2000*, 2000, pp. 291–300. DOI: 10.1007/978-94-015-9554-4_{33}.
- [30] J. L. Herder and C. M. Gosselin, “A counter-rotary counterweight (CRCW) for light-weight dynamic balancing,” in *Proceedings of the ASME Design Engineering Technical Conference*, vol. 2 A, 2004. DOI: 10.1115/detc2004-57246.
- [31] V. van der Wijk and J. L. Herder, “The work of Otto Fischer and the historical development of his method of principal vectors for mechanism and machine science,” in *History of Mechanism and Machine Science*, vol. 15, 2012, pp. 521–534. DOI: 10.1007/978-94-007-4132-4_{36}.
- [32] V. A. Shchepetil’nikov, “The determination of the mass centers of mechanisms in connection with the problem of mechanism balancing,” *Journal of Mechanisms*, vol. 3, no. 4, 1968, ISSN: 00222569. DOI: 10.1016/0022-2569(68)90011-6.
- [33] V. Van Der Wijk and J. L. Herder, “Double pendulum balanced by counter-rotary counter-masses as useful element for synthesis of dynamically balanced mechanisms,” in *Proceedings of the ASME Design Engineering Technical Conference*, vol. 2, 2008. DOI: 10.1115/DETC2008-49402.
- [34] V. Van Der Wijk and J. L. Herder, “Synthesis of dynamically balanced mechanisms by using counter-rotary counter-mass balanced double pendula,” *Journal of Mechanical Design, Transactions of the ASME*, vol. 131, no. 11, 2009, ISSN: 10500472. DOI: 10.1115/1.3179150.
- [35] V. van der Wijk, J. L. Herder, and B. Demeulenaere, “Comparison of various dynamic balancing principles regarding additional mass and additional inertia,” *Journal of Mechanisms and Robotics*, vol. 1, no. 4, pp. 1–9, 2009, ISSN: 19424302. DOI: 10.1115/1.3211022.
- [36] V. Van der Wijk and J. L. Herder, “Dynamic Balancing of Mechanisms by using an Actively Driven Counter-Rotary Counter-Mass for Low Mass and Low Inertia,” in *Proceedings of the Second International Workshop on Fundamental Issues and Future Research Directions for Parallel Mechanisms and Manipulators*, 2008.
- [37] C. Bagci, “Shaking force balancing of planar linkages with force transmission irregularities using balancing idler loops,” *Mechanism and Machine Theory*, vol. 14, no. 4, pp. 267–284, Jan. 1979, ISSN: 0094114X. DOI: 10.1016/0094-114X(79)90013-2.
- [38] J. J. de Jong, J. van Dijk, and J. L. Herder, “A screw based methodology for instantaneous dynamic balance,” *Mechanism and Machine Theory*, vol. 141, pp. 267–282, 2019, ISSN: 0094114X. DOI: 10.1016/j.mechmachtheory.2019.07.014.
- [39] S. Martinez, J. P. Meijaard, and V. Van Der Wijk, “On the Shaking Force Balancing of Compliant Mechanisms,” in *2019 IEEE 7th International Conference on Control, Mechatronics and Automation, ICCMA 2019*, 2019, pp. 310–314, ISBN: 9781728137872. DOI: 10.1109/ICCMA46720.2019.8988681.
- [40] G. G. Lowen and R. S. Berkof, “Survey of investigations into the balancing of linkages,” *Journal of Mechanisms*, vol. 3, no. 4, 1968, ISSN: 00222569. DOI: 10.1016/0022-2569(68)90001-3.

- [41] V. van der Wijk, "On the grand 4R four-bar based inherently balanced linkage architecture," in *Mechanisms and Machine Science*, vol. 43, London, 2017, pp. 473–480. DOI: 10.1007/978-3-319-44156-6{_}48.
- [42] O. Sigmund, "A 99 line topology optimization code written in matlab," *Structural and Multidisciplinary Optimization*, vol. 21, no. 2, pp. 120–127, 2001, ISSN: 1615147X. DOI: 10.1007/s001580050176.
- [43] K. Svanberg, "The method of moving asymptotes - a new method for structural optimization," *International Journal for Numerical Methods in Engineering.*, vol. 24, no. October 1985, pp. 359–373, 1987.
- [44] E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund, "Efficient topology optimization in MATLAB using 88 lines of code," *Structural and Multidisciplinary Optimization*, vol. 43, no. 1, pp. 1–16, 2011, ISSN: 1615147X. DOI: 10.1007/s00158-010-0594-7.
- [45] N. L. Pedersen, "Maximization of eigenvalues using topology optimization," *Structural and Multidisciplinary Optimization*, vol. 20, no. 1, pp. 2–11, 2000, ISSN: 1615147X. DOI: 10.1007/s001580050130.
- [46] K. Svanberg, "MMA and GCMMA - two methods for nonlinear optimization, versions September 2007," *Technical report, Optimization and Systems Theory*, vol. 1, pp. 1–15, 2007.

Appendix A

Sensitivity calculations

In this appendix, the calculations of the sensitivities for the shaking force balance and shaking moment balance conditions are discussed. It is important to note that the shaking force balance equation consists of two nearly identical parts, where the only difference is the use of A_x or A_y (the matrix for the conversion of the nodal to element displacement in x- or y-direction respectively). Also note that, due to the way the algorithm of Sigmund [42] is organized, the positive y-direction is downwards, so the angular momentum (for shaking moment balance) is defined positive in clockwise direction. This, however, has no significant impact on the equations as the shaking moment balance requires the angular momentum to be as low as possible, regardless of its direction.

The calculations are performed in 3 ways. First, the sensitivity of a constraint is calculated with respect to ρ_i . Then the sensitivity constraint is calculated with respect to $\tilde{\alpha}_i$. This calculation is verified by the much simpler calculation that follows from applying the chain rule in differentiation, as shown in equation A.0.1.

$$\frac{df}{d\tilde{\alpha}_i} = \frac{df}{d\rho_i} \frac{d\rho_i}{d\tilde{\alpha}_i} \quad (\text{A.0.1})$$

In the optimization algorithm, all constraints are normalized. This is not shown in the calculations in this appendix for the purpose of clarity.

A.1 Shaking Force Balance Constraint with respect to ρ

The shaking force balance constraint is defined in 3.2.6. This consists of two terms, being the sum of the squares of 3.2.5a and 3.2.5b. The resulting sensitivity equation is presented in 3.2.8a. To obtain the full expression of this equation 3.2.8a, the derivatives of equations 3.2.7a and 3.2.7a are required. As discussed before, they are nearly the same, so in this paragraph, the derivative of equation 3.2.7a is presented, along with the full derivative of the shaking force balance constraint in equation 3.2.6. The shaking force balance equation is defined as follows:

$$g_{SFB} = u_x^2 + u_y^2 \leq \epsilon_{SFB}$$

From this equation, equation 3.2.8a is the derivative with respect to ρ_i which is denoted as:

$$\frac{dg_{SFB}}{d\rho_i} = 2 \left(u_x \frac{du_x}{d\rho_i} + u_y \frac{du_y}{d\rho_i} \right)$$

Equation 3.2.5a, representing the displacement of the COM in x-direction, is as follows:

$$u_x = \frac{(\rho \mathbf{A}_x \mathbf{u})}{\sum \rho} + \lambda_x (\mathbf{f} - \mathbf{K} \mathbf{u}) = 0$$

Taking the derivative with respect to density ρ_i yields:

$$\frac{du_x}{d\rho_i} = \frac{(\sum \rho)([\mathbf{A}_x \mathbf{u}]_i + \rho \mathbf{A}_x \frac{d\mathbf{u}}{d\rho_i}) - \rho \mathbf{A}_x \mathbf{u}}{(\sum \rho)^2} - \lambda_x \mathbf{K} \frac{d\mathbf{u}}{d\rho_i} - \lambda_x \frac{d\mathbf{K}}{d\rho_i} \mathbf{u} \quad (\text{A.1.1})$$

This equation is simplified in order to eliminate the term $\frac{d\mathbf{u}}{d\rho_i}$. This yields:

$$\frac{du_x}{d\rho_i} = \frac{[\mathbf{A}_x \mathbf{u}]_i}{\sum \rho} - \frac{\rho \mathbf{A}_x \mathbf{u}}{(\sum \rho)^2} - \lambda_x \frac{d\mathbf{K}}{d\rho_i} \mathbf{u} + \left(\frac{\rho \mathbf{A}_x}{\sum \rho} - \lambda_x \mathbf{K} \right) \frac{d\mathbf{u}}{d\rho_i} \quad (\text{A.1.2})$$

The last term, consisting of an expression between brackets and the derivative $\frac{d\mathbf{u}}{d\rho_i}$, should be eliminated. This is the case if:

$$\left(\frac{\rho \mathbf{A}_x}{\sum \rho} - \lambda_x \mathbf{K} \right) = \mathbf{0} \quad (\text{A.1.3})$$

This yields an expression for λ_x :

$$\lambda_x = \frac{\rho \mathbf{A}_x \mathbf{K}^{-1}}{\sum \rho} \quad (\text{A.1.4})$$

Finally, this yields the shaking force balance constraint:

$$\frac{du_x}{d\rho_i} = \frac{[\mathbf{A}_x \mathbf{u}]_i}{\sum \rho} - \frac{\rho \mathbf{A}_x \mathbf{u}}{(\sum \rho)^2} - \lambda_x \frac{d\mathbf{K}}{d\rho_i} \mathbf{u} \quad (\text{A.1.5})$$

A.2 Shaking Force Balance Constraint with respect to $\tilde{\alpha}$

The same calculations can be performed to obtain the sensitivities with respect to $\tilde{\alpha}$ rather than ρ_i . Equation 3.2.5a, representing the displacement of the COM in x-direction, is as follows:

$$u_x = \frac{(\tilde{\alpha}^p \mathbf{A}_x \mathbf{u})}{\sum \tilde{\alpha}^p} + \lambda_x (\mathbf{f} - \mathbf{K} \mathbf{u}) = 0$$

Taking the derivative with respect to density $\tilde{\alpha}_i$ yields:

$$\frac{du_x}{d\tilde{\alpha}_i} = \frac{(\sum \tilde{\alpha}^p)([p\tilde{\alpha}^{(p-1)} \mathbf{A}_x \mathbf{u}]_i + \tilde{\alpha}^p \mathbf{A}_x \frac{d\mathbf{u}}{d\tilde{\alpha}_i}) - \tilde{\alpha}^p \mathbf{A}_x \mathbf{u} p \tilde{\alpha}_i^{(p-1)}}{(\sum \tilde{\alpha}^p)^2} - \lambda_x \mathbf{K} \frac{d\mathbf{u}}{d\tilde{\alpha}_i} - \lambda_x \frac{d\mathbf{K}}{d\tilde{\alpha}_i} \mathbf{u} \quad (\text{A.2.1})$$

This equation is simplified in order to eliminate the term $\frac{d\mathbf{u}}{d\tilde{\alpha}_i}$. This yields:

$$\frac{du_x}{d\tilde{\alpha}_i} = p \tilde{\alpha}_i^{(p-1)} \left(\frac{[\mathbf{A}_x \mathbf{u}]_i}{\sum \tilde{\alpha}^p} - \frac{\tilde{\alpha}^p \mathbf{A}_x \mathbf{u}}{(\sum \tilde{\alpha}^p)^2} \right) - \lambda_x \frac{d\mathbf{K}}{d\tilde{\alpha}_i} \mathbf{u} + \left(\frac{\tilde{\alpha}^p \mathbf{A}_x}{\sum \tilde{\alpha}^p} - \lambda_x \mathbf{K} \right) \frac{d\mathbf{u}}{d\tilde{\alpha}_i} \quad (\text{A.2.2})$$

The last term, consisting of an expression between brackets and the derivative $\frac{d\mathbf{u}}{d\tilde{\alpha}_i}$, should be eliminated. This is the case if:

$$\left(\frac{\tilde{\alpha}^p \mathbf{A}_x}{\sum \tilde{\alpha}^p} - \lambda_x \mathbf{K} \right) = \mathbf{0} \quad (\text{A.2.3})$$

This yields an expression for λ_x :

$$\lambda_x = \frac{\tilde{\alpha}^p \mathbf{A}_x \mathbf{K}^{-1}}{\sum \tilde{\alpha}^p} \quad (\text{A.2.4})$$

Finally, this yields the shaking force balance constraint:

$$\frac{d\mathbf{u}_x}{d\tilde{\alpha}_i} = p\tilde{\alpha}_i^{(p-1)} \left(\frac{[\mathbf{A}_x \mathbf{u}]_i}{\sum \tilde{\alpha}^p} - \frac{\tilde{\alpha}^p \mathbf{A}_x \mathbf{u}}{(\sum \tilde{\alpha}^p)^2} \right) - \lambda_x \frac{d\mathbf{K}}{d\tilde{\alpha}_i} \mathbf{u} \quad (\text{A.2.5})$$

A.3 Shaking Moment Balance Constraint with respect to ρ

The shaking moment balance constraint is defined in Equation 3.2.6, and looks as follows:

$$g_{SMB} = R_z^2 \leq \epsilon_{SMB}^2$$

The sensitivity of this equation is given in Equation 3.2.9a and looks as follows:

$$\frac{dg_{SMB}}{d\rho_i} = 2R_z \frac{dR_z}{d\rho_i}$$

To obtain the full expression of this equation, expressions for R_z and $\frac{dR_z}{d\rho_i}$ are required. The representation of R_z in linear algebraic form is shown in Equation 3.2.7c, and for clarification shown below:

$$R_z = \frac{\rho \mathbf{M} \mathbf{u}}{\sum \rho} + \lambda_z (\mathbf{f} - \mathbf{K} \mathbf{u})$$

In this equation, the matrix M is used which is given by Equation 3.2.5d, and for clarification shown below:

$$\mathbf{M} = \mathbf{r}_x \mathbf{A}_y - \mathbf{r}_y \mathbf{A}_x$$

\mathbf{r}_x and \mathbf{r}_y are the position vectors of the element with respect to the COM in x- and y-direction, respectively. They are found by equation A.3.1, where \mathbf{x} is the vector of x-positions of all elements.

$$\mathbf{r}_x = \mathbf{x} - x_{COM} \mathbf{e}_x \quad (\text{A.3.1})$$

The position of the COM is determined by equation A.3.2.

$$x_{COM} = \frac{\sum_i x_i \rho_i}{\sum \rho_i} \quad (\text{A.3.2})$$

In order to obtain the derivative of R_z with respect to ρ_i , the derivatives of the building blocks in these equations need to be known. This starts with the derivative of Equation A.3.2, which is given in Equation A.3.3.

$$\frac{dx_{COM}}{d\rho_i} = \frac{(\sum_i \rho_i) x_i - \sum x_i \rho_i}{(\sum_i \rho_i)^2} = \frac{x_i - x_{COM}}{\sum_i \rho_i} = \frac{\mathbf{r}_x}{\sum \rho} \quad (\text{A.3.3})$$

Knowing this, the derivative of \mathbf{r}_x can be found, as shown in Equation A.3.4.

$$\frac{d\mathbf{r}_x}{d\rho_i} = -\frac{dx_{COM}}{d\rho_i} \mathbf{e}_x = -\frac{\mathbf{r}_x}{\sum \rho} \quad (\text{A.3.4})$$

Knowing this, the derivative of \mathbf{M} can be found, as shown in Equation A.3.5.

$$\frac{d\mathbf{M}}{d\rho_i} = \frac{d\mathbf{r}_x}{d\rho_i} \mathbf{A}_y - \frac{d\mathbf{r}_y}{d\rho_i} \mathbf{A}_x = -\frac{\mathbf{M}}{\sum \rho} \quad (\text{A.3.5})$$

Taking the derivative of R_z with respect to ρ_i yields the following equation A.3.6.

$$\frac{dR_z}{d\rho_i} = \frac{(\sum \rho) ([\mathbf{M} \mathbf{u}]_i - \frac{\rho \mathbf{M} \mathbf{u}}{\sum \rho} + \rho \mathbf{M} \frac{d\mathbf{u}}{d\rho_i}) - \rho \mathbf{M} \mathbf{u}}{(\sum \rho)^2} - \lambda_z \mathbf{K} \frac{d\mathbf{u}}{d\rho_i} - \lambda_z \frac{d\mathbf{K}}{d\rho_i} \mathbf{u} \quad (\text{A.3.6})$$

Rewriting equation A.3.6 yields:

$$\frac{dR_z}{d\rho_i} = \frac{([\mathbf{M}\mathbf{u}]_i - 2R_z)}{\sum \rho} - \lambda_z \frac{d\mathbf{K}}{d\rho_i} \mathbf{u} + \left(\frac{\rho\mathbf{M}}{\sum \rho} - \lambda_z \mathbf{K} \right) \frac{d\mathbf{u}}{d\rho_i} \quad (\text{A.3.7})$$

A.4 Shaking Moment Balance Constraint with respect to $\tilde{\alpha}$

The shaking moment balance constraint is defined in Equation 3.2.6, and looks as follows:

$$g_{SMB} = R_z^2 \leq \epsilon_{SMB}^2$$

The sensitivity of this equation is given in Equation 3.2.9a and looks as follows:

$$\frac{dg_{SMB}}{d\tilde{\alpha}_i} = 2R_z \frac{dR_z}{d\tilde{\alpha}_i}$$

To obtain the full expression of this equation, expressions for R_z and $\frac{dR_z}{d\tilde{\alpha}_i}$ are required. The representation of R_z in linear algebraic form is shown in Equation 3.2.7c, and for clarification shown below:

$$R_z = \frac{\tilde{\alpha}^p \mathbf{M}\mathbf{u}}{\sum \tilde{\alpha}^p} + \lambda_z (\mathbf{f} - \mathbf{K}\mathbf{u})$$

In this equation, the matrix M is used which is given by Equation 3.2.5d, and for clarification shown below:

$$\mathbf{M} = \mathbf{r}_x \mathbf{A}_y - \mathbf{r}_y \mathbf{A}_x$$

\mathbf{r}_x and \mathbf{r}_y are the position vectors of the element with respect to the COM in x- and y-direction, respectively. They are found by equation A.4.1, where \mathbf{x} is the vector of x-positions of all elements.

$$\mathbf{r}_x = \mathbf{x} - x_{COM} \quad (\text{A.4.1})$$

The position of the COM is determined by equation A.4.2.

$$x_{COM} = \frac{\sum_i x_i \tilde{\alpha}_i^p}{\sum \tilde{\alpha}_i^p} \quad (\text{A.4.2})$$

In order to obtain the derivative of R_z with respect to α_i , the derivatives of the building blocks in these equations need to be known. This starts with the derivative of Equation A.3.2, which is given in Equation A.4.3.

$$\frac{dx_{COM}}{d\tilde{\alpha}_i} = p\tilde{\alpha}_i^{(p-1)} \frac{(\sum_i \tilde{\alpha}_i^p)x_i - \sum x_i \tilde{\alpha}_i}{(\sum_i \tilde{\alpha}_i^p)^2} = p\tilde{\alpha}_i^{(p-1)} \frac{x_i - x_{COM}}{\sum_i \tilde{\alpha}_i^p} = p\tilde{\alpha}_i^{(p-1)} \frac{\mathbf{r}_x}{\sum \tilde{\alpha}^p} \quad (\text{A.4.3})$$

Knowing this, the derivative of \mathbf{r}_x can be found, as shown in Equation A.4.4.

$$\frac{d\mathbf{r}_x}{d\tilde{\alpha}_i} = -\frac{dx_{COM}}{d\tilde{\alpha}_i} = -p\tilde{\alpha}_i^{(p-1)} \frac{\mathbf{r}_x}{\sum \tilde{\alpha}^p} \quad (\text{A.4.4})$$

Knowing this, the derivative of \mathbf{M} can be found, as shown in Equation A.3.5.

$$\frac{d\mathbf{M}}{d\tilde{\alpha}_i} = \frac{d\mathbf{r}_x}{d\tilde{\alpha}_i} \mathbf{A}_y - \frac{d\mathbf{r}_y}{d\tilde{\alpha}_i} \mathbf{A}_x = -p\tilde{\alpha}_i^{(p-1)} \frac{\mathbf{M}}{\sum \tilde{\alpha}^p} \quad (\text{A.4.5})$$

Taking the derivative of R_z with respect to α_i yields the following equation A.4.6.

$$\frac{dR_z}{d\tilde{\alpha}_i} = \frac{(\sum \tilde{\alpha}^p)([p\tilde{\alpha}_i^{(p-1)} \mathbf{M}\mathbf{u}]_i - \frac{\tilde{\alpha}^p p \tilde{\alpha}_i^{(p-1)} \mathbf{M}\mathbf{u}}{\sum \tilde{\alpha}^p} + \tilde{\alpha}^p \mathbf{M} \frac{d\mathbf{u}}{d\tilde{\alpha}_i}) - p\tilde{\alpha}_i^{(p-1)} \tilde{\alpha}^p \mathbf{M}\mathbf{u}}{(\sum \tilde{\alpha}^p)^2} - \lambda_z \mathbf{K} \frac{d\mathbf{u}}{d\tilde{\alpha}_i} - \lambda_z \frac{d\mathbf{K}}{d\tilde{\alpha}_i} \mathbf{u} \quad (\text{A.4.6})$$

Rewriting equation A.4.6 yields:

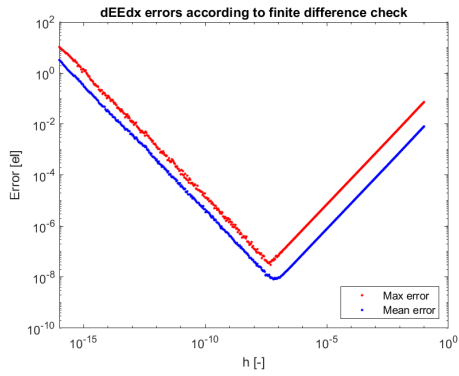
$$\frac{dR_z}{d\tilde{\alpha}_i} = p\tilde{\alpha}_i^{(p-1)} \frac{([\mathbf{M}\mathbf{u}]_i - 2R_z)}{\sum \tilde{\alpha}^p} - \lambda_z \frac{d\mathbf{K}}{d\tilde{\alpha}_i} \mathbf{u} + \left(\frac{\tilde{\alpha}^p \mathbf{M}}{\sum \tilde{\alpha}^p} - \lambda_z \mathbf{K} \right) \frac{d\mathbf{u}}{d\tilde{\alpha}_i} \quad (\text{A.4.7})$$

Appendix B

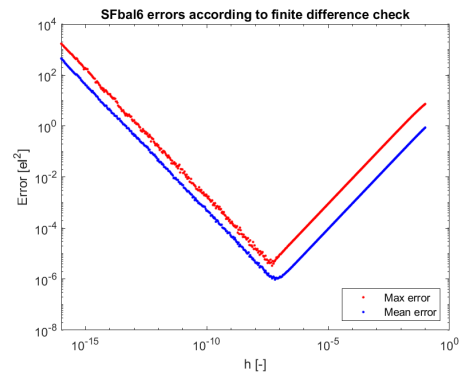
Verification of sensitivity functions

In this section, the sensitivity functions are verified using a finite difference analysis. This finite difference analysis numerically approaches the analytically determined derivatives of the objective and constraint functions. A forward difference scheme is used with a step size h which is added to the design variables. This step size is logarithmically increased from 10^{-16} to 10^{-1} . The resulting error is given in elements for the end-effector displacement, and squared elements for the shaking force balance (as this is a square of the x- and y-displacements) and the shaking moment balance (as this is calculated by multiplying the displacements of elements with their distance from the COM, both in elements).

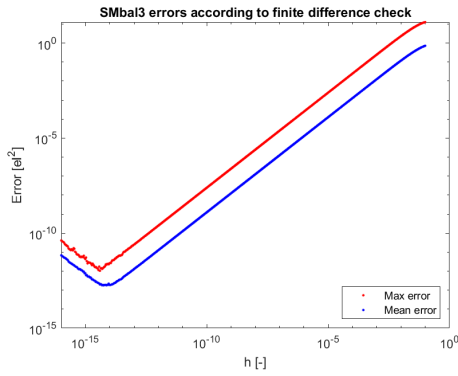
In figures B.1a to B.1c, the response functions are plotted of these analyses. In this analysis, the force inverter by Sigmund is used, in a 10×10 elements design space. The "mean error" and the "max error" are displayed, being the average error and the largest error of these 100 elements.



(a) End-effector displacement sensitivity function



(b) SFB sensitivity function



(c) SMB sensitivity function

Figure B.1: Finite difference verification of objective and constraint sensitivity functions

Appendix C

Comsol analysis

In this appendix, more information on the Comsol simulations will be shown. First, the conversion from MATLAB to SolidWorks geometries is presented, then the model in COMSOL is further explained, and finally the eigenmodes of the off-centered displacement inverter are shown.

C.1 Transformation from MATLAB to SolidWorks

The transformation from a matlab geometry to a SolidWorks drawing has first been done automatically with the 'Sketch Picture' option in the SolidWorks sketch toolbox. This did not yield the desired results, so some postprocessing (especially around flexure parts) was still required. The resulting unbalanced geometry was quite well-converted by SolidWorks, so only small changes were made. The result is shown in Figure C.1.

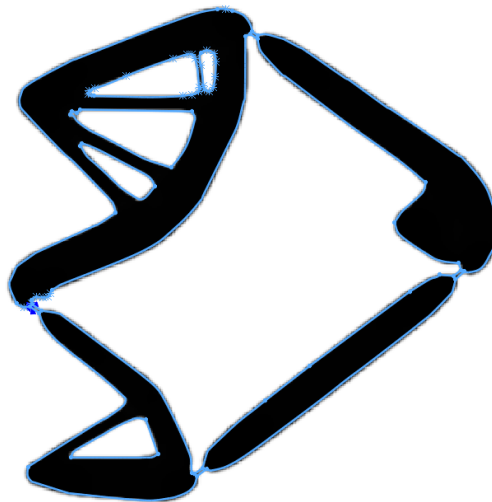


Figure C.1: Semi-automatically sketched unbalanced geometry in SolidWorks

The balanced geometry has been converted in the same way, as shown in Figure C.2. This



Figure C.2: Semi-automatically sketched balanced geometry in SolidWorks

however did not follow the geometry in MATLAB as well as desired. Therefore, the sketch is made by hand, as shown in Figure C.3. The differences between the resulting displacements in



Figure C.3: Hand-sketched balanced geometry in SolidWorks

the MATLAB and COMSOL simulations is shown in Table C.1. This table shows quite clear differences arise, and some values are closer to the values from the MATLAB simulation compared to Figure C.2 while some are closer compared to Figure C.3. This shows a better method for this transformation would improve the reliability of the results.

A probable source of uncertainty in the transformation may be the thickness of the hinges in the geometry. In Figure C.4 clear differences are visible between the unfiltered and filtered design variables, and the densities. The SolidWorks drawings and the MATLAB simulations are based

Table C.1: Results from balanced mechanism in COMSOL simulation in mm (R_z in mm^2) (MATLAB values converted to COMSOL coordinate system)

Property	Automatic	By hand	Matlab
U_{xin}	27.991	27.643	27.602
U_{yin}	-0.79447	-0.34330	-0.49203
U_{xout}	-21.219	-21.447	-21.146
U_{yout}	-8.0948	-7.6955	-7.3924
U_{COMx}	-0.16178	-0.12250	-0.16116
U_{COMy}	-0.29510	0.10067	0.11843
x_{com}	194.95	194.11	194.35
y_{com}	220.46	221.43	221.28
R_z	7.6061	-2.8309	19.999

on the densities. In the transition to a COMSOL geometry, scaling issues may occur. Therefore,

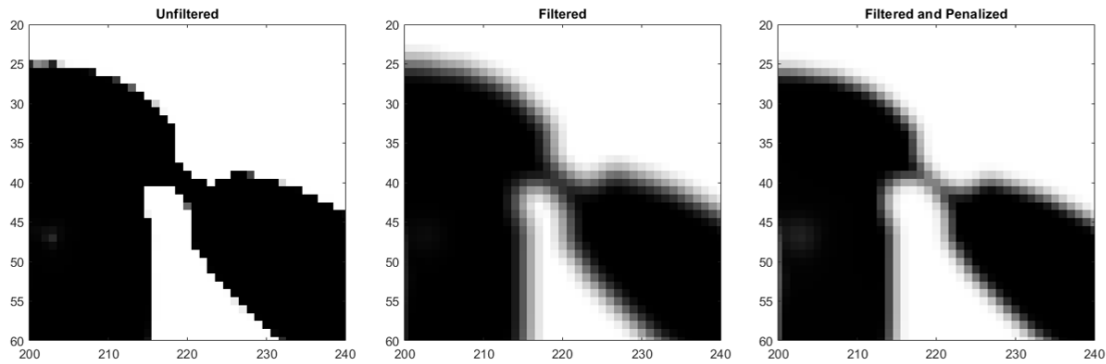


Figure C.4: Expanded view of the top notch joint in the balanced geometry in Matlab for the geometries of the unfiltered design variables, the filtered design variables and the densities.

the figure in SolidWorks has elements plotted in the corners. These are used to calibrate the image, by scaling to the original (400mm) design space size and the correct placement of the boundary conditions. This is incorporated in the COMSOL Model. The fact that the “colormap(gray)” command in MATLAB displaces all elements by 0.5 element in x- and y-direction is also solved with this method. In MATLAB plots, this 0.5 element displacement is taken into account for the COM and end-effector displacement plots.

C.2 COMSOL model

The geometry in COMSOL is implemented as described before. The material is set to the same values as the values in the MATLAB model. The solid mechanics toolbox is applied. The boundary conditions, input node and end-effector node are constructed according to the same coordinates as the MATLAB model. On the input node and end-effector node, springs are defined with the same stiffness as the springs in the MATLAB model, with a spring constant only in the x-direction. The rest of the model is described in Section 3.3. The studies performed in COMSOL are:

- Stationary study

- Time Dependent study
- Eigenfrequency study
- Frequency Domain study

The resulting values are derived as follows:

- Input and output displacements are taken from the nodal displacements.
- COM values (displacement, velocity and acceleration) are derived from the surface average of the entire domain.
- Linear momentum is derived using domain probes which take the integral of the expression $\text{solid.u.tX}*\text{solid.rho}*\text{solid.d}$ and the expression $\text{solid.u.tY}*\text{solid.rho}*\text{solid.d}$ for linear momentum in x- and y-direction, respectively
- Angular momentum is derived using a domain probe with the expression $((\text{X-mass1.CMx})*\text{solid.u.tY} - (\text{Y-mass1.CMy})*\text{solid.u.tX})*\text{solid.rho}*\text{solid.d}$
- Reaction forces and moments are derived from the sum of the kinematic boundary condition points, the input node and the output node.

C.3 Frequency domain analysis

In Figures C.5 and C.6, the eigenmodes of the balanced and unbalanced geometries of the displacement inverter are presented, respectively.

C.3.1 Balanced displacement inverter

C.3.2 Unbalanced displacement inverter

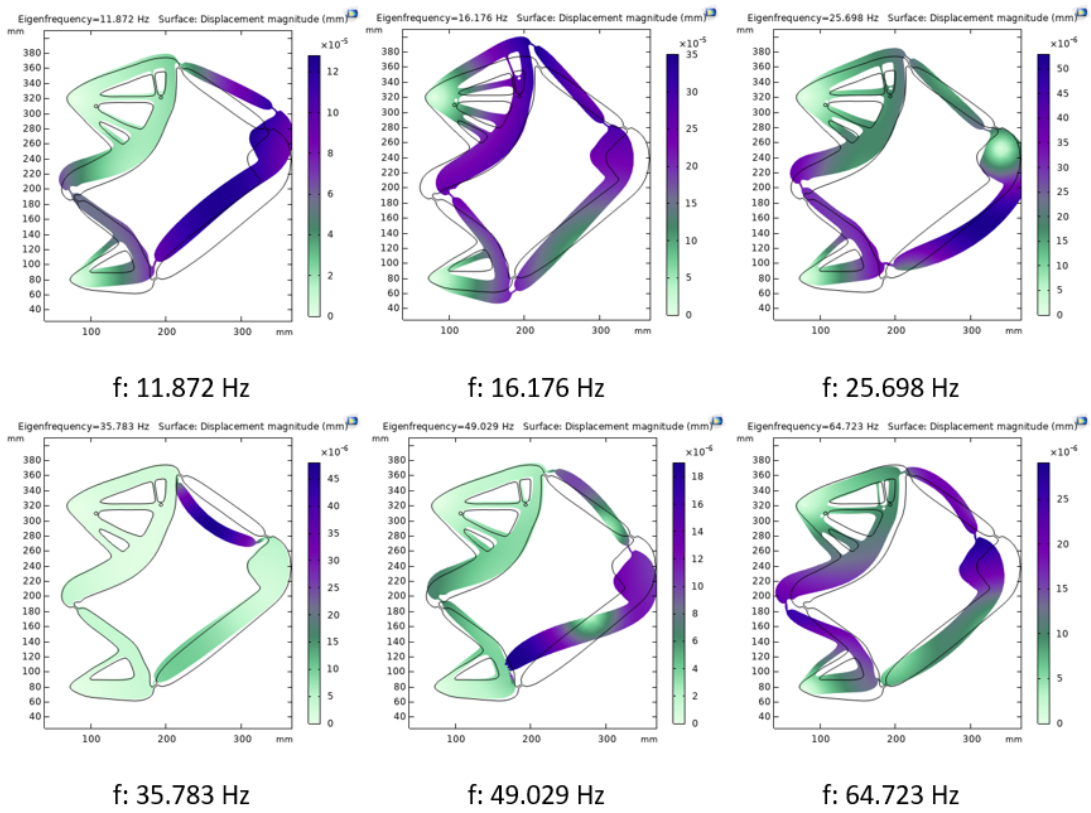


Figure C.5: Eigenmodes of balanced displacement inverter

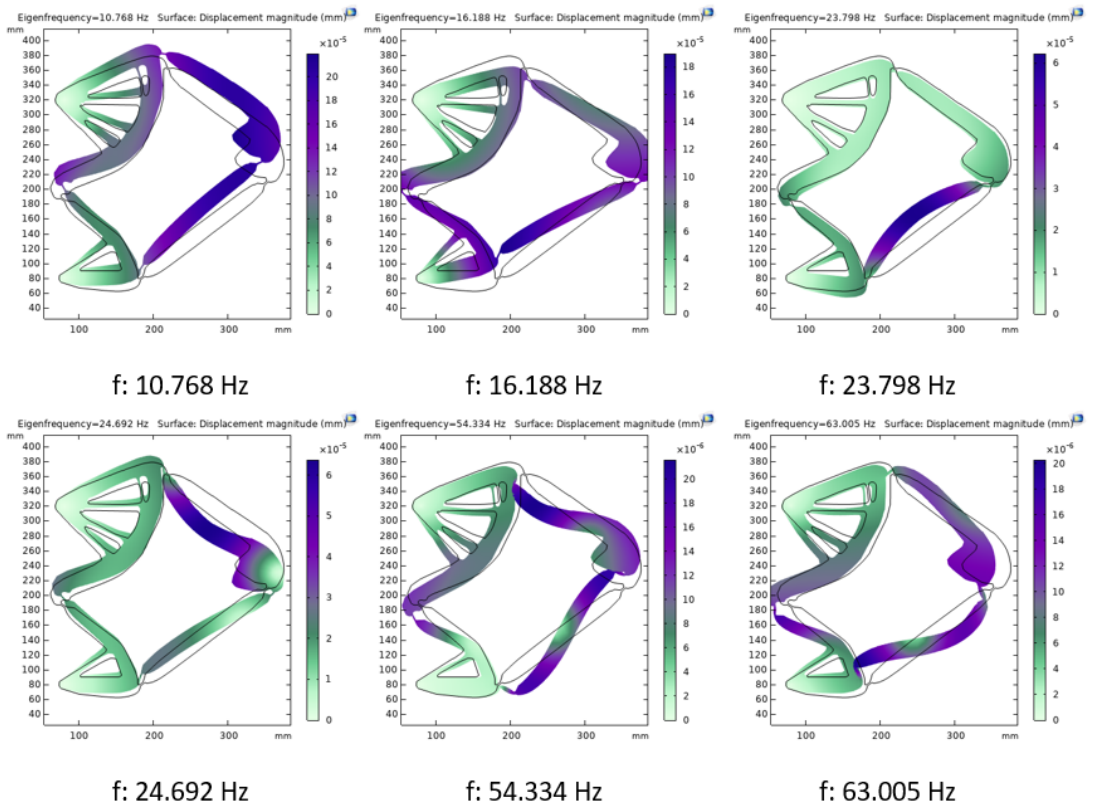


Figure C.6: Eigenmodes of unbalanced displacement inverter

Appendix D

Other mechanisms

In this Appendix, some other results from optimization attempts are shown. They are not further discussed.

D.1 Diagonal Corners Mechanism

The mechanisms in this section are results of the following set of boundary conditions, as depicted in Figure D.1:

- Two fixed boundary conditions at the top left and bottom right corner
- An input force at the node in the bottom left corner
- An output motion at the node in the top right corner
- A design domain around the previously described boundary conditions of a thickness of 20% of the entire domain.

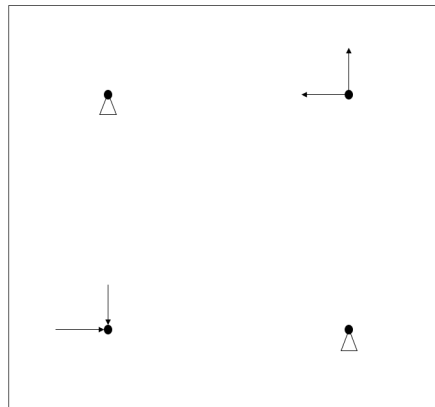


Figure D.1: Boundary conditions of the Diagonal Corners Mechanism, either an x- or y-directed force as input and an x- or y-directed displacement as output.

The parameters are presented in Table D.1. The filter radius and input and output directions are varied. Each design set is specified by the filter radius, input direction and output direction in the header.

Property	Value
nelx	400 els
nely	400 els
Volfrac	0.2
Penal	3
Young's modulus	10 MPa
Poisson's ratio	0.3
Springs stiffness	0.1 N/mm
Input force	1 N
boundary thickness	0.2
SF slack	0.01
SM slack	2

Table D.1: Input variables for following designs

D.1.1 x-input,x-output,R=3

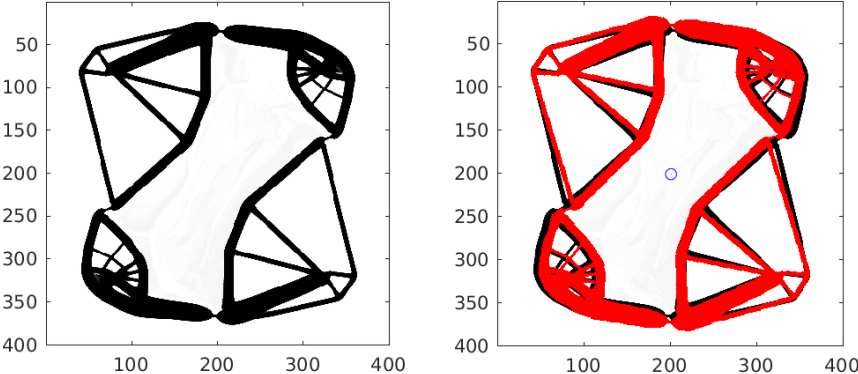


Figure D.2: Unbalanced

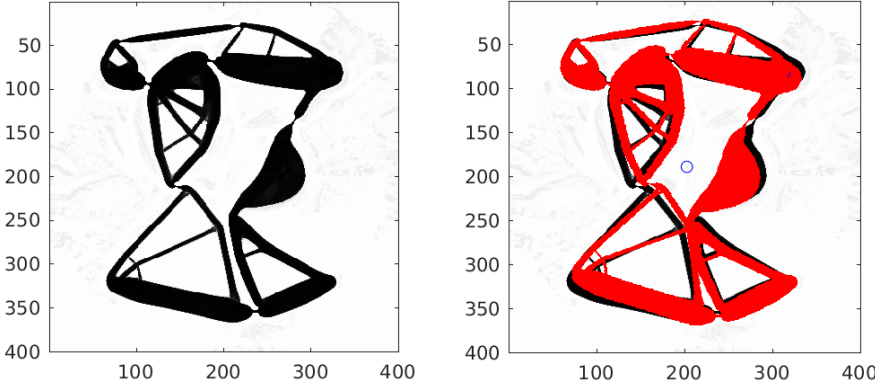


Figure D.3: Balanced

D.1.2 x -input, x -output, $R=5$

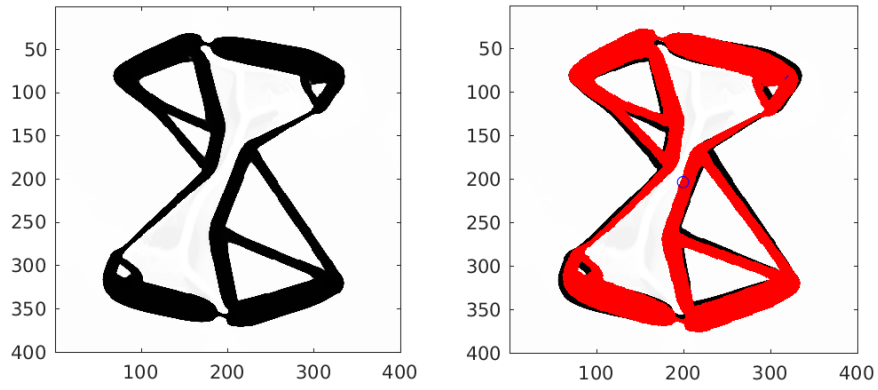


Figure D.4: Unbalanced

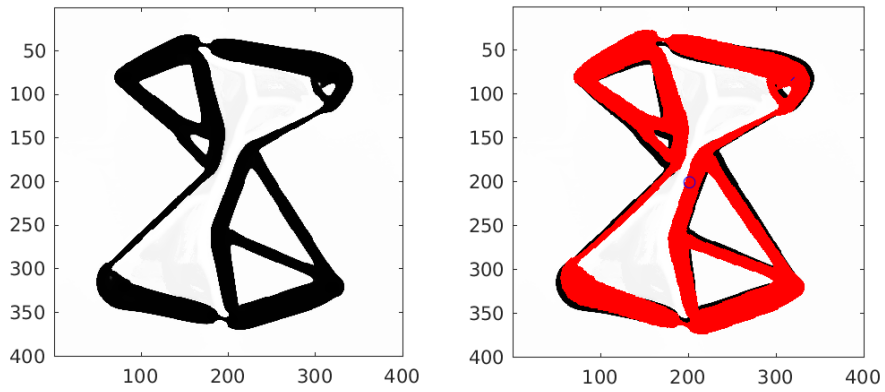


Figure D.5: Balanced

D.1.3 x-input,y-output,R=3

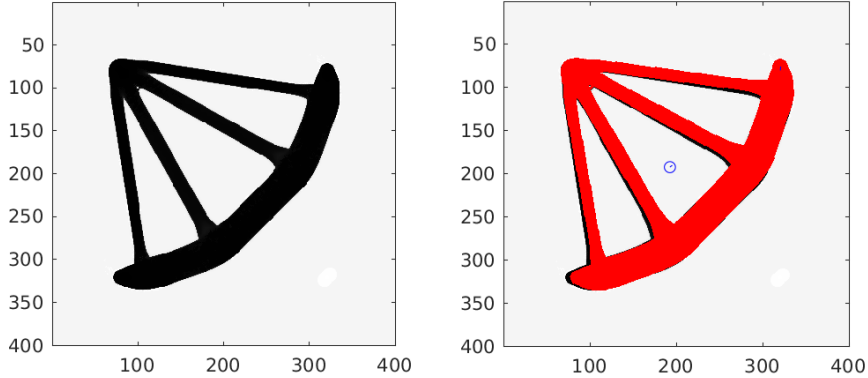


Figure D.6: Unbalanced

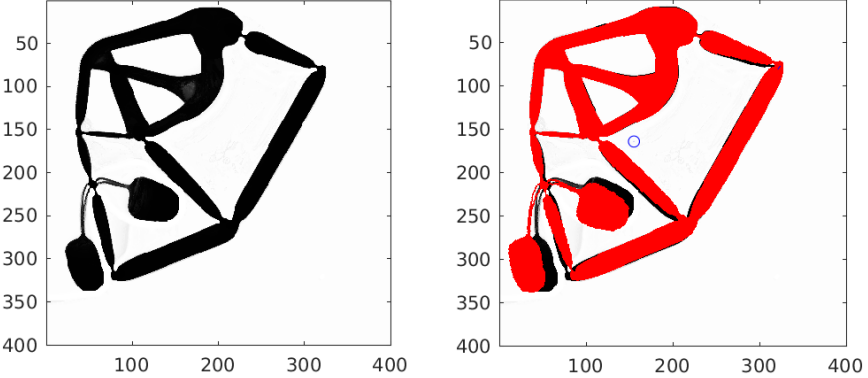


Figure D.7: Balanced

D.1.4 x-input,y-output, $R=5$

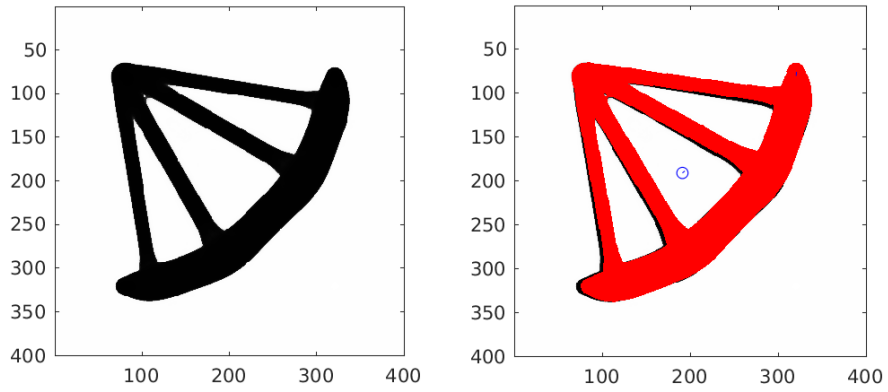


Figure D.8: Unbalanced

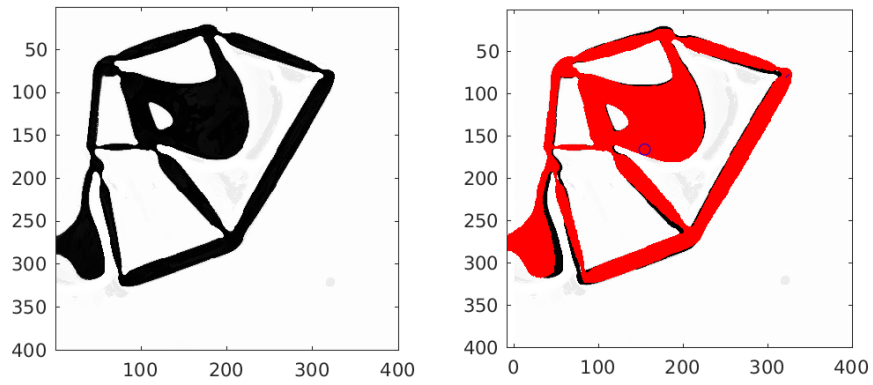


Figure D.9: Balanced

D.1.5 y -input, y -output, $R=3$

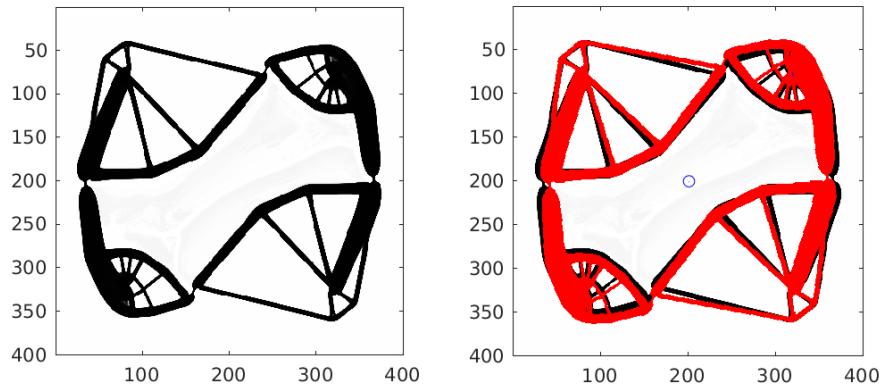


Figure D.10: Unbalanced

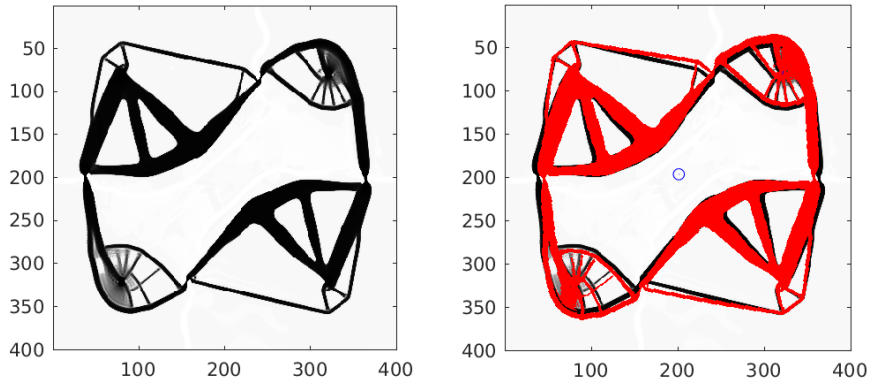


Figure D.11: Balanced

D.1.6 y -input, y -output, $R=5$

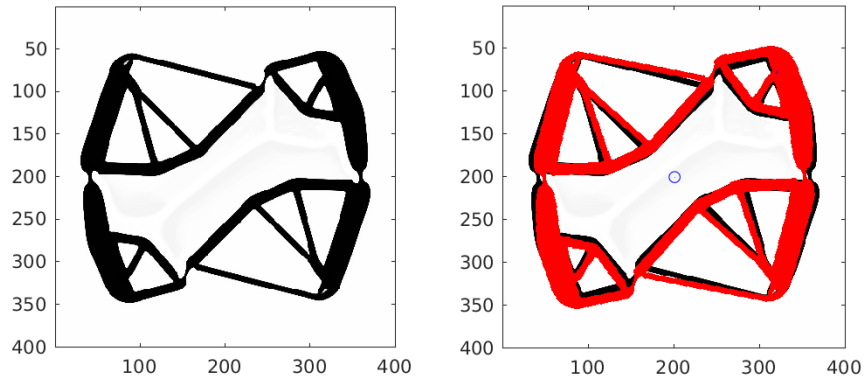


Figure D.12: Unbalanced

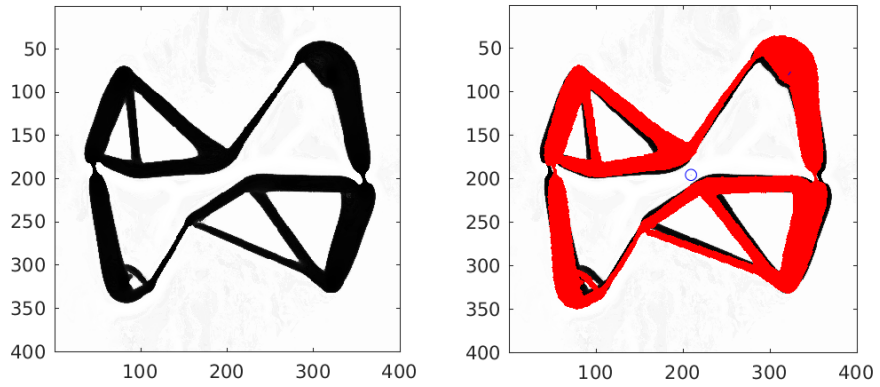


Figure D.13: Balanced

Appendix E

Matlab Code

The matlab code, based on the 99-line code by Sigmund [42] and the 88-line code by Andreassen [44] and the mma-code by Svanberg [43], consists of multiple functions. The building blocks are shown in section E.6. In section E.1, the code of the entire algorithm is explained. In section E.2, a manual is presented on how to use the code, which inputs are required and what options are available.

E.1 Code overview

The main algorithm is shown in the function `topmma.m`. The layout of this algorithm is shown in Figure E.1. Auxiliary functions are used, which allows for changing the balance conditions easily.

`topmma_balanced` is the

E.2 Inputs and options

The total matlab code has two sets of inputs. At the start of the optimization algorithm `topmma_balanced.m` or `topmma_unbalanced.m`, some inputs for the actual function and several internal optimization options are required. These options are elaborated in this section.

E.3 Inputs

The following input are put in the input line of the `topmma` function. **nelx** - Number of elements of the design domain in x-direction. This number should be possible to divide by 10 for most boundary conditions.

nely - Number of elements of the design domain in y-direction. This number should be possible to divide by 10 for most boundary conditions.

volfrac - Volumetric material fraction of the design variables.

penal0 - Penalty value to be applied in the SIMP method. If continuation is applied, this is the starting value.

rmin - Filter radius of the density filter.

s10 - Relaxation parameter value for the SFB constraint. If continuation is applied, this is the

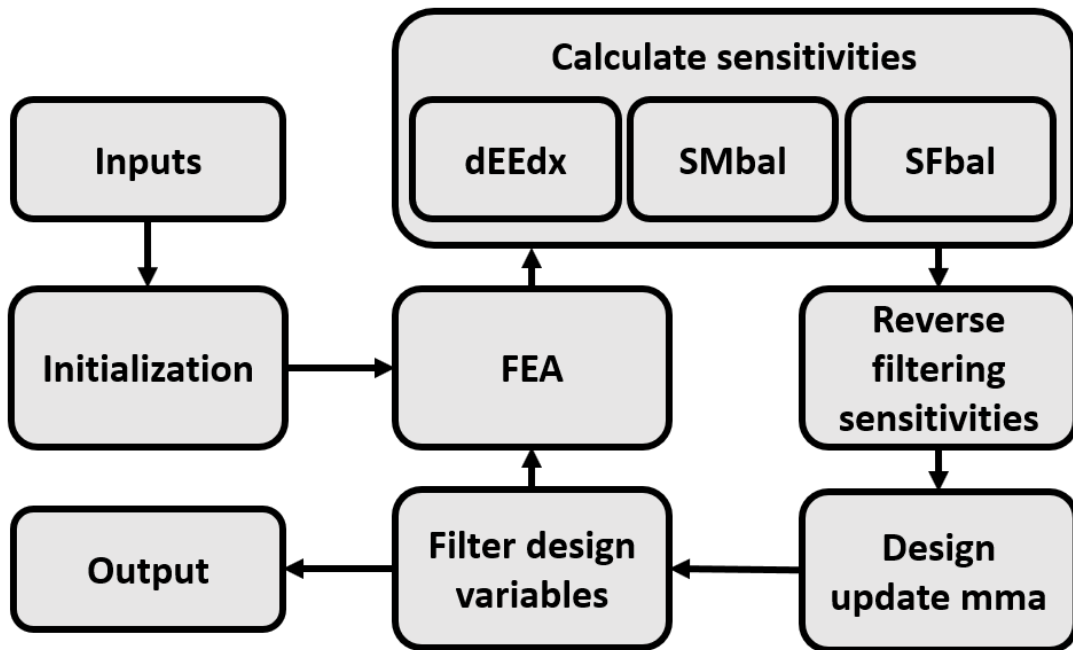


Figure E.1: Overview of the optimization algorithm

starting value.

slm0 - Relaxation parameter value for the SMB constraint. If continuation is applied, this is the starting value.

situation - Input set of boundary conditions. Has to be started with @.

startingconditions - Starting design variables. Should be a .mat file within apostrophes, for example: 'balanced.mat'. If a grey start is used, the input should be [].

E.4 Options

The following options are chosen within the `topmma` function. **plotdata** - Set 0 or 1 (off or on) to plot the deformed system and iteration data after the final iteration.

blocks - Set 0 or 1 (off or on) to apply solid or void blocks into the design domain with specified positions and sizes. These blocks will form a non-design domain. This value is only used to either apply or not apply such non-design domains.

record - Set 0 or 1 (off or on) to record and store videos of the optimization run (both design variables and densities)

maxiter - Maximum number of iterations until stopping, no matter whether the optimization has converged. If set to 0, no maximum is applied.

continuation - Set 0 or 1 (off or on) to apply continuation. From a predefined iteration number during a predefined number of iterations change some predefined parameters.

helpdesigner - Set 0 or 1 (off or on) to use the "Helpdesigner" function, which plots lines through the input and output node if a local minimum is preventing convergence.

miniter - Minimum number of iterations.

maxchange - Maximum change value. If the highest change in design variables in an iteration is below this value, the optimization run is assumed to have converged and will finish.

F_in - Scalar input force value in Newtons

bs - Boundary size as a fraction of the total design domain. This boundary size is a design domain around the original design domain. 0.5 means the entire domain is outside domain, so will not be possible. A value of 0 means no outside design domain is applied. The value for bs should not be below 0 and always below 0.5.

The following function will only be applied if blocks is turned on

blocksize - n*5 matrix containing the x- and y- coordinates of the blocks upper left and lower right corner (x_{top},y_{top},x_{bot},y_{bot}) of each block in each row, and the corresponding density in the 5th column.

The following functions will only be applied if continuation is turned on

slend - Final SFB slack parameter after continuation.

slmend - Final SMB slack parameter after continuation.

contloop - Iteration where continuation starts.

contduration - Amount of iterations over which the slowly increasing continuation will distribute.

E.5 More info

Before starting, always start the matlab stopwatch. If the file Runoptimization.m is used, this is already done.

Nelx and Nely have to be a multiple of 10 for most situations.

It should be noted that for compliant mechanism design, the volumetric constraint should be chosen lower than for static low compliance design. When the volume fraction is high, rigid bodies will just become thicker, which increases the change of them interfering with eachother. As collision between bodies is not taken into account in this algorithm, the volume fraction should be picked accordingly. This algorithm generally produces good results for a volume fraction of 0.2-0.3.

The penalty value is generally taken as 3. If a higher penalty value is used, the algorithm will converge to a black-and-white solution sooner, but maybe not reach the optimal solution. More info is available in the works of Bendsøe and Sigmund [16].

The filter radius Rmin is generally chosen higher than 1. For small problems (for example 50x50 elements) a value of 1.2 or 1.5 will be sufficient. For larger problems, the filter radius should also be increased. This will also increase minimum thicknesses of members and holes.

The sl0 and slm0 values determine the maximum values for the displacement of the COM or the angular momentum approximation, respectively. It is dependent on the design space size how high a value is to be strict or relaxed.

Input situations can be found in Appendix E.6.

E.6 Matlab Code

The following codes are presented in this appendix: **topmma_balanced** - Main optimization code, also containing all options and inputs.

SFbal - Shaking force balance constraint and sensitivity calculator

SMBal - Shaking moment balance constraint and sensitivity calculator

dEEdx - Objective function sensitivity calculator

topmma_unbalanced - Main optimization code, also containing all options and inputs, without balance constraints.

FE - Finite element analysis code.

lk - Element stiffness matrix.

mma - Method of moving asymptotes optimization algorithm by Svanberg[43].

Runoptimization - Example of a script that will run the optimization

E.7 topmma_balanced

```
1  %%%% A 99 LINE TOPOLOGY OPTIMIZATION CODE BY OLE SIGMUND, JANUARY
   2000 %%%
2  %%%% CODE MODIFIED FOR INCREASED SPEED, September 2002, BY OLE
   SIGMUND %%%
3  % This code has been modified by Nol R mer for his MSc. Thesis
   at the TU
4  % Delft. The goal of the modifications is to allow the algorithm
   to design
5  % Dynamically balanced compliant mechanisms and gain insight in
   the
6  % optimization process. This code generates balanced mechanisms.
7
8  function topmma_balanced(nelx,nely,volfrac,penal0,rmin,s10,slm0,
   situation,startingconditions)
9  %% Setting up the optimization loop
10 % OPTIONS
11 % If an option is set to 1, it is on. If it is set to 0, it is off
   . The
12 % situation parameter is a nested function. The chosen situation
   has to be
13 % inserted with an @ sign in front, so every function in the
   algorithm will
14 % have the correct input values.
15 plotdata = 1; % Plots deformed system after the final iteration
16 blocks = 0; % Add blocks of size "blocksize = [x,y]" to the input
   and output points. Along a design space boundary, this should
   be an even number.
17 record = 1; % Record iteration steps to obtain a video of the
   optimization run.
18 maxiter = 0; % Maximum number of iterations. If set to 0, the code
   will run until convergence.
```

```

19 maxtime = 19.5; % Maximum time in hours. If set to 0, the code
    will run until convergence.
20 continuation = 1; % Continuation. If set to 1, continuation is on.
21 helpdesigner = 1;
22 miniter = 50;
23 maxchange = 0.01;
24 F_in = 5; % Input force in Newtons
25 bs = 0.2; % Boundariespace
26
27 %%%%%%%%% Added blocks or designless domains %%%%%%%%%
28 blocksize = round([0.45*nelx,0.45*nely,0.55*nelx,0.55*nely,0;
    0.45*nelx 0.4*nely 0.55*nelx 0.45*nely 1]); % n*5 matrix
    containing the x- and y- coordinates of the blocks upper left
    and lower right corner (xtop,ytop,xbot,ybot) of each block in
    each row, and the corresponding density.
29
30 %%%%%%%%% Continuation %%%%%%%%%
31 penal = penal0;
32 sl = sl0; % Slack parameter: how many elements the COM is allowed
    to move at the start
33 slm = slm0; % Moment balance slack parameter at the start
34 slend = 0.02; % Slack parameter: how many elements the COM is
    allowed to move at the end
35 slmend = 2; % Moment balance slack parameter at the end
36 penalend = penal0;
37 contloop = 250; % Iteration when the continuation starts
38 contduration = 500; % Amount of iterations over which the slowly
    increasing continuation will distribute
39
40 %% INITIALIZATION
41 % Starting conditions implementation
42 if isempty(startingconditions) == 1
43     xmat = ones(nely,nelx)*volfrac; % Grey start
44 else
45     temp = struct2cell(load(startingconditions));
46     x0 = temp{1};
47     clear temp
48     x0 = reshape(x0,nely,nelx);
49     if size(x0)~= [nely,nelx]
50         error('startingconditions do not correspond to design
            space size')
51     end
52     xmat = volfrac*(1-x0)+(1-volfrac)*x0;
53 end
54
55 % Initialize values
56 [din,dout,fixeddofs,F,namesitu] = situation(nelx,nely,F_in,bs);
57 name = string(['V23_balanced(' sprintf('%4i',nelx) ',' sprintf('%4
    i',nely) ',' sprintf('%6.2f',volfrac) ',' sprintf('%2.1f',penal

```

```

    ) ',' sprintf('%2.1f',rmin) ',' namesitu ')]);
58 [KE] = lk;
59 loop = 0;
60 change = 1;
61 [Ax,Ay,x_disp,y_disp,x_undisp,y_undisp,nel,edofMat] = Initialize(
    nelx,nely);
62 [H,Hs] = Initialize_filter(nelx,nely,rmin);
63 x = reshape(xmat,nel,1);
64 opt = mma(nel);
65 U = zeros(2*(nely+1)*(nelx+1),2);
66
67 % Include blocks and nondesign domains
68 [block_el_nrs_zero,block_el_nrs_one] = blocksmat2vec(blocksize,
    nelx,nely);
69 if blocks == 1
70     x(block_el_nrs_zero) = 1E-6;
71     x(block_el_nrs_one) = 1;
72 end
73 xflt = x;
74 figure(1);
75
76 %% START ITERATION
77 while change > maxchange || loop < miniter+1
78     loop = loop + 1;
79     xfltmat = reshape(xflt,nely,nelx);
80 % Calculate COM
81 [COMX,COMY] = COM(xfltmat);
82 % FE-ANALYSIS
83 [U,c,~]=FE(nelx,nely,xfltmat,penal,F,KE,din,dout,fixeddofs);
84 lambdac = U(:,2);
85 U = U(:,1);
86 % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
87 [dc] = dEEdx(nelx,nely,U,lambdac,penal,xfltmat,KE);
88 [gx,dgxdx,Ux,Uy] = SFbal(xflt,nelx,nely,U,KE,penal,sl,Ax,Ay,din,
    dout,fixeddofs);
89 [gmom,dgdxmom,mom] = SMbal(xflt,nelx,nely,U,KE,penal,slm,Ax,Ay,
    COMX,COMY,din,dout,fixeddofs);
90
91 g(1) = c;
92 dgdx(1:nel,1) = reshape(dc,nel,1);
93 g(2) = sum(xflt)/(volfrac*nel) - 1;
94 dgdx(1:nel,2) = 1/volfrac/nel;
95 g(3) = gx - 1;
96 dgdx(1:nel,3) = dgxdx;
97 g(4) = gmom - 1;
98 dgdx(1:nel,4) = dgdxmom;
99
100 % REVERSE FILTERING OF SENSITIVITIES
101 for i = 1:length(g)

```

```

102 [dgdx(1:nel,i)] = H*(dgdx(1:nel,i)./Hs);
103 end
104
105 if blocks == 1
106     for i = 1:length(g)
107         dgdx(union(block_el_nrs_zero,block_el_nrs_one),i) = 0;
108     end
109 end
110 xold=x;
111 % DESIGN UPDATE BY THE METHOD OF MOVING ASYMPTOTES
112 [opt,x] = opt.update(xold,g,dgdx);
113 if loop >= 10
114     if helpdesigner == 1 && c >= -1E-1
115         elin = floor(din/2)-floor(din/(2*nely));
116         elout = floor(dout/2)-floor(dout/(2*nely));
117         x([elin-nely/10:elin+nely/10,elout-nely/10:elout+nely/10])
            = max(x);
118     end
119 end
120 xflt(:) = (H*x(:))./Hs;
121 if blocks == 1
122     x(block_el_nrs_zero) = 1E-6;
123     x(block_el_nrs_one) = 1;
124 end
125 xmat = reshape(x,nely,nelx);
126 xmatflt = reshape(xflt,nely,nelx);
127 % PRINT RESULTS
128 bwfrac = (sum(x>0.99)+sum(x<0.01))/nel;
129 change = max(max(abs(x-xold)));
130 disp([' It.: ' sprintf('%4i',loop) ' Endeff.: ' sprintf('%10.4f'
            ,full(c)) ...
131     ' Vol.: ' sprintf('%6.3f',sum(sum(xmat))/(nelx*nely)) ...
132     ' ch.: ' sprintf('%6.3f',change) ...
133     ' U: ' sprintf('%6.3f',sqrt(Ux^2+Uy^2)) ...
134     ' Mom: ' sprintf('%6.3f',mom)...
135     ' BWfrac: ' sprintf('%6.3f',bwfrac)...
136     ' Uin: ' sprintf('%6.3f',full(U(din)))...
137     ])
138 % PLOT DENSITIES AND STORE PROGRESS
139 % COM's are offset 0.5 elements because the grayscale command
            is offset
140 % 0.5 elements as well.
141 % Design values
142 figure(1)
143 colormap(gray); imagesc(-xmat); axis equal; axis tight; axis
            on;pause(1e-6);
144 hold on
145 title('Design values');
146 plot(COMX+0.5,COMY+0.5,'ob')

```



```

147     quiver(COMX+0.5,COMY+0.5,Ux,Uy,'b');
148     hold off
149     if record == 1
150     movie1(opt.iter) = getframe(gcf);
151     end
152     % Densities
153     figure(2)
154     colormap(gray); imagesc(-xmatflt.^3); axis equal; axis tight;
        axis on;pause(1e-6);
155     hold on
156     title('Densities');
157     plot(COMX+0.5,COMY+0.5,'ob')
158     quiver(COMX+0.5,COMY+0.5,Ux,Uy,'b');
159     hold off
160     if record == 1
161     movie2(opt.iter) = getframe(gcf);
162     end
163     % Other plots
164     Uxplot(opt.iter) = Ux;
165     Uyplot(opt.iter) = Uy;
166     Objplot(opt.iter) = c;
167     bwfracplot(opt.iter) = bwfrac;
168     Changeplot(opt.iter) = change;
169     xstore(:,loop) = xflt;
170
171     % Nonconvergent iteration limits (time or maxiter)
172     if opt.iter == maxiter
173         change = 0;
174         disp('Maximum number of iterations reached');
175     end
176     if maxtime ~= 0 && toc >= maxtime*3600
177         change = 0;
178         disp('Time limit reached')
179     end
180
181     % Continuation
182     if continuation == 1
183         if loop > contloop && loop < contloop+contduration
184             sl = sl-1/contduration*(sl0-slend);
185             slm = slm-1/contduration*(slm0-slmend);
186             penal = penal-1/contduration*(penal0-penalend);
187         end
188     end
189 end
190
191 %% After Final Iteration
192 x_disp((1:4),:) = x_undisp((1:4),:) + U(edofMat(:, [1 3 5 7]))';
193 y_disp((1:4),:) = y_undisp((1:4),:) + U(edofMat(:, [2 4 6 8]))';
194 save('balanced.mat','x')

```

```

195 save(strjoin(string([name, 'xstore.mat'])), 'xstore', '-mat')
196 if plotdata == 1 % Allows turning on and off in options
197     figure(3);
198     subplot(2,2,1), plot([1:1:opt.iter],Uxplot, 'r-'), xlabel('
        Iterations'), ylabel('Displacement [elements]'), title('
        COM-displacements');
199     hold on
200     plot([1:1:opt.iter],Uyplot, 'b-'), xlabel('Iterations'),
        ylabel('Displacement [elements]');
201     legend('x-displacement', 'y-displacement');
202     subplot(2,2,2), plot([1:1:opt.iter],bwfracplot*100, 'r-'),
        xlabel('Iterations'), ylabel('bwfrac [%]'), title('Black&
        white fraction');
203     subplot(2,2,3), plot([1:1:opt.iter],Objplot, 'r-'), xlabel('
        Iterations'), ylabel('Displacement [elements]'), title('
        End-effector Displacement');
204     subplot(2,2,4), plot([1:1:opt.iter],Changeplot, 'r-'),
        xlabel('Iterations'), ylabel('Density change'), title('
        Max change');
205     saveas(gcf, strjoin(string([name, 'iterationdata.fig'])));
206     figure(4);
207     subplot(1,2,1), colormap(gray); imagesc(-xmat); axis equal
        ; axis tight; axis on; pause(1e-6);
208     subplot(1,2,2), colormap(gray); imagesc(-xmat); axis equal
        ; axis tight; axis on; pause(1e-6);
209     figure(4)
210     xplot = sparse(round(x));
211     [elplot,~,~] = find(xplot);
212     patch(x_disp(:,elplot), y_disp(:,elplot), 'r', 'FaceAlpha', 1,
        'EdgeColor', 'none'); pause(1e-6);
213     hold on
214     plot(COMX+0.5, COMY+0.5, 'ob')
215     quiver(COMX+0.5, COMY+0.5, Ux, Uy, 'b');
216     dyout = 2*round(dout/2);
217     dxout = dyout - 1;
218     xout = floor(ceil(dout/2)/(nely+1))+0.5; % +0.5 to
        compensate for colormap(gray) command placing nodes at
        +-0.5 elements
219     yout = ceil(dout/2)-(xout-0.5)*(nely+1)-1+0.5;
220     quiver(xout+0.5, yout+0.5, U(dxout), U(dyout), 'b');
221     saveas(gcf, strjoin(string([name, 'Deformedplot.png'])));
222 end
223 %% Make videos
224 % Video design values
225     writerObj = VideoWriter(strjoin(string([name, 'Design
        values'])));
226     writerObj.FrameRate = 10; % set the fps
227     % open the video writer
228     open(writerObj);

```

```

229         % write the frames to the video
230         for i=1:length(movie1)
231             % convert the image to a frame
232             frame = movie1(i) ;
233             writeVideo(writerObj, frame);
234         end
235         % close the writer object
236         close(writerObj);
237
238     % Video densities
239     writerObj = VideoWriter(strjoin(string([name, 'Densities'])
240         ));
241     writerObj.FrameRate = 10; % set the fps
242     % open the video writer
243     open(writerObj);
244     % write the frames to the video
245     for i=1:length(movie2)
246         % convert the image to a frame
247         frame = movie2(i) ;
248         writeVideo(writerObj, frame);
249     end
250     % close the writer object
251     close(writerObj);
252
253 %% Auxiliary functions
254
255 %%%%%%%%% COM calculation %%%%%%%%%
256 function [COMX, COMY] = COM(xPhys)
257 [nely, nelx] = size(xPhys);
258 xdist = ones(nely, 1)*[1:nelx];
259 ydist = [1:nely]'*ones(1, nelx);
260 COMX = sum(sum(xPhys.*xdist))/sum(sum(xPhys));
261 COMY = sum(sum(xPhys.*ydist))/sum(sum(xPhys));
262
263
264 %%%%%%%%% Initialize function %%%%%%%%%
265 function [Ax, Ay, x_disp, y_disp, x_undisp, y_undisp, nel, edofMat] =
266     Initialize(nelx, nely)
267 % Function Initialize
268 nel = nelx*nely;
269 ndof = 2*(nelx+1)*(nely+1);
270 nodenrs = reshape(1:(nelx+1)*(nely+1), nely+1, nelx+1);
271 edofVec = reshape(2*nodenrs(1:end-1, 1:end-1)+1, nel, 1);
272         % Used to make edofMat
273 edofMat = repmat(edofVec, 1, 8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],
274     nel, 1);
275 % Ax and Ay matrices

```

```

273 Ax = 0.25*sparse(kron([1:1:nel],ones(1,4)),reshape(edofMat(:,[1 3
5 7]'),4*nel,1),ones(1,4*nel),nel,ndof);
274 Ay = 0.25*sparse(kron([1:1:nel],ones(1,4)),reshape(edofMat(:,[2 4
6 8]'),4*nel,1),ones(1,4*nel),nel,ndof);
275 x_undisp = kron((0.5+ones(4,1)*[1:nelx]-[ones(1,nelx);zeros(2,nelx
);ones(1,nelx)]),ones(1,nely));
276 y_undisp = kron(ones(1,nelx),(0.5+ones(4,1)*[1:nely]-[zeros(2,nely
);ones(2,nely)]));
277 x_disp = x_undisp;
278 y_disp = y_undisp;
279
280 %%%%%% Initialize Filter %%%%%%
281 function [H,Hs] = Initialize_filter(nelx,nely,rmin)
282 iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
283 jH = ones(size(iH));
284 sH = zeros(size(iH));
285 k = 0;
286 for i1 = 1:nelx
287     for j1 = 1:nely
288         e1 = (i1-1)*nely+j1;
289         for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),
nelx)
290             for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)
-1),nely)
291                 e2 = (i2-1)*nely+j2;
292                 k = k+1;
293                 iH(k) = e1;
294                 jH(k) = e2;
295                 sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
296             end
297         end
298     end
299 end
300 H = sparse(iH,jH,sH);
301 Hs = sum(H,2);
302
303 function [block_el_nrs_zero,block_el_nrs_one] = blocksmat2vec(
blocksize,nelx,nely)
304 % This function is used to predefine rectangular non-design
domains in
305 % the total design domain, with a predefined density of 0 or 1.
306 %
307 % block_el_nrs_zero element numbers in the vector x with a
predefined
308 % value of 0 and sensitivity of 0.
309 % block_el_nrs_one element numbers in the vector x with a
predefined
310 % value of 1 and sensitivity of 0.
311

```

```

312 %   Blocksize           n*5 matrix containing the upper left and
      lower
313 %                       right x and y coordinates of n rectangular
      bodies
314 %                       of which the element densities should be 0
      or 1.
315 %                       The order of the columns is as follows:
316 %                       [xtop,ytop,xbot,ybot,density]
317 %                       with the density being 0 or 1.
318 %   nelx                # elements in the design domain in x-
      direction
319 %   nely                # elements in the design domain in y-
      direction
320
321 [n,~] = size(blocksize);
322 block_el_nrs_zero = [];
323 block_el_nrs_one = [];
324 for i = 1:n
325     blocksize(i,:);
326     numbers = reshape(1:nelx*nely,nely,nelx);
327     blocknrs_i = numbers([blocksize(i,2):blocksize(i,4)],[
      blocksize(i,1):blocksize(i,3)]);
328     if blocksize(i,5) == 1
329         block_el_nrs_one = union(blocknrs_i,block_el_nrs_one);
330     elseif blocksize(i,5) == 0
331         block_el_nrs_zero = union(blocknrs_i,block_el_nrs_zero);
332     else
333         disp('error: blocks in "blocksize" not defined as 0 or 1')
334     end
335 end
336
337 %
338 %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
339 % Original code was written by Ole Sigmund, Department of Solid
      %
340 % Mechanics, Technical University of Denmark, DK-2800 Lyngby,
      Denmark. %
341 % Please sent your comments to the author: sigmund@fam.dtu.dk
      %
342 %
      %
343 % The code is intended for educational purposes and theoretical
      details %
344 % are discussed in the paper
      %

```

```

345 % "A 99 line topology optimization code written in Matlab"
346 % by Ole Sigmund (2001), Structural and Multidisciplinary
347 % Optimization, Vol 21, pp. 120--127.
348 %
349 % The code as well as a postscript version of the paper can be
350 % downloaded from the web-site: http://www.topopt.dtu.dk
351 %
352 % Disclaimer:
353 % The author reserves all rights but does not guaranty that the
354 % code is free from errors. Furthermore, he shall not be liable in any
355 % event caused by the use of the program.
356 %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

E.8 SFbal

```

1 % This function is used to obtain balance conditions and
  % sensitivities
2 % using the normal input variables. The used balance equation is a
3 % quadratic shaking force balance equation, on penalized design
  % variables.
4
5 % Function:  $U_x^2 + U_y^2 + \text{adjoint} \leq s_1^2$ 
6
7 function [g,dgdx,Ux,Uy] = SFbal(x,nelx,nely,U,KE,penal,s1,Ax,Ay,
  % din,dout,fixeddofs)
8 % Initialize
9 xPhys = reshape(x,nely,nelx);
10 nel = nelx*nely;
11 nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
12 edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nel,1);
13 edofMat = repmat(edofVec,1,8)+repmat([-2 -1 2*nely+[0 1 2 3] 0 1],
  % nel,1);
14 % Adjoint functions calculations

```

```

15 Flambdax = Ax'*(x.^penal)/sum(x.^penal);
16 [lambdax,~,~]=FE(nelx,nely,xPhys,penal,Flambdax,KE,din,dout,
    fixeddofs);
17 lambdax = lambdax(:,1)';
18 Flambday = Ay'*(x.^penal)/sum(x.^penal);
19 [lambday,~,~]=FE(nelx,nely,xPhys,penal,Flambday,KE,din,dout,
    fixeddofs);
20 lambday = lambday(:,1)';
21 % COM displacement and sensitivity of COM displacement
22 Ux = (x.^penal)'*Ax*U(:,1)/sum(x.^penal);
23 Uy = (x.^penal)'*Ay*U(:,1)/sum(x.^penal);
24 duxdx = penal*x.^(penal-1).*(Ax*U(:,1))/sum(x.^penal) - penal*x.^(
    penal-1)*Ux/sum(x.^penal) - (penal*x'.^(penal-1).*sum(lambdax(
    edofMat)'.*(KE*U(edofMat))))';
25 duydx = penal*x.^(penal-1).*(Ay*U(:,1))/sum(x.^penal) - penal*x.^(
    penal-1)*Uy/sum(x.^penal) - (penal*x'.^(penal-1).*sum(lambday(
    edofMat)'.*(KE*U(edofMat))))';
26 % Definition of balance conditions and sensitivities
27 g = (Ux^2 + Uy^2)/sl^2;
28 dgdx = 2*(Ux*duxdx + Uy*duydx)/sl^2;
29 end

```

E.9 Smbal

```

1 % This function is used to obtain balance conditions and
    sensitivities
2 % using the normal input variables. The used balance equation is a
3 % quadratic shaking force balance equation.
4
5 % Function: (rho*Ux*ry - rho*Uy*rx)/sum(rho) + adjoint <= slm^2
6
7 function [g,dgdx,mom] = Smbal(x,nelx,nely,U,KE,penal,slm,Ax,Ay,
    COMX,COMY,din,dout,fixeddofs)
8 % Initialize
9 xmat = reshape(x,nely,nelx);
10 nel = nelx*nely;
11 nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
12 edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
13 edofMat = repmat(edofVec,1,8)+repmat([-2 -1 2*nely+[0 1 2 3] 0 1],
    nelx*nely,1);
14 xdist = reshape(ones(nely,1)*[1:nelx],nel,1);
15 ydist = reshape([1:nely]'*ones(1,nelx),nel,1);
16 rx = xdist-COMX;
17 ry = ydist-COMY;
18 iMat = kron([1:nel]',ones(4,1));
19 jxMat = reshape(edofMat(:,[2 4 6 8])',nel*4,1);
20 jyMat = reshape(edofMat(:,[1 3 5 7])',nel*4,1);
21 rxMat = kron(rx,ones(4,1));

```

```

22 ryMat = kron(ry,ones(4,1));
23 rxsparse = sparse(iMat,jxMat,rxMat);
24 rysparse = sparse(iMat,jyMat,ryMat);
25 rysparse(nel,(1+nelx)*(1+nely)*2) = 0;
26
27 % Calculations
28 M = rxsparse.*Ay - rysparse.*Ax;
29 Flambda = ((x.^penal)'*M/sum(x.^penal))';
30 [lambda,~,~]=FE(nelx,nely,xmat,penal,Flambda,KE,din,dout,fixeddofs
    );
31 lambda = lambda';
32
33 mom = ((x.^penal)'*M*U)/sum(x.^penal);
34 dmomdx = (penal/sum(x.^penal)*x.^(penal-1)) .* (M*U(:,1) - 2*mom)
    ...
    -(penal*x'.^(penal-1).*sum(U(edofMat)'.*(KE*lambda(
        edofMat)'))));
36
37 % Definition of balance conditions and sensitivities
38 g = mom^2/slm^2;
39 dgdx = 2*mom*dmomdx/slm^2;
40 end

```

E.10 dEEdx

```

1 % This function calculates the end effector sensitivity
2 function [dc] = dEEdx(nelx,nely,U,lambdac,penal,xmat,KE)
3 % Initialize
4 x = reshape(xmat,nelx*nely,1);
5 nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
6 edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
7 edofMat = repmat(edofVec,1,8)+repmat([-2 -1 2*nely+[0 1 2 3] 0 1],
    nelx*nely,1);
8 % Sensitivity calculation
9 Ue1 = U(edofMat);
10 Ue2 = lambdac(edofMat);
11 dc = (penal*x'.^(penal-1).*sum(Ue1'.*(KE*Ue2')));

```

E.11 topmma_unbalanced

```

1 %%% A 99 LINE TOPOLOGY OPTIMIZATION CODE BY OLE SIGMUND, JANUARY
    2000 %%%
2 %%% CODE MODIFIED FOR INCREASED SPEED, September 2002, BY OLE
    SIGMUND %%%
3 % This code has been modified by Nol R mer for his MSc. Thesis
    at the TU

```



```

4 | % Delft. The goal of the modifications is to allow the algorithm
   | to design
5 | % Dynamically balanced compliant mechanisms and gain insight in
   | the
6 | % optimization process. This code designs unbalanced mechanisms
   | and is
7 | % used for reference.
8 |
9 | function topmma_unbalanced(nelx,nely,volfrac,penal0,rmin,sl0,slm0,
   | situation,startingconditions)
10 | %% Setting up the optimization loop
11 | % OPTIONS
12 | % If an option is set to 1, it is on. If it is set to 0, it is off
   | . The
13 | % situation parameter is a nested function. The chosen situation
   | has to be
14 | % inserted with an @ sign in front, so every function in the
   | algorithm will
15 | % have the correct input values.
16 | plotdata = 1; % Plots deformed system after the final iteration
17 | blocks = 0; % Add blocks of size "blocksize = [x,y]" to the input
   | and output points. Along a design space boundary, this should
   | be an even number.
18 | record = 1; % Record iteration steps to obtain a video of the
   | optimization run.
19 | maxiter = 0; % Maximum number of iterations. If set to 0, the code
   | will run until convergence.
20 | maxtime = 19.5; % Maximum time in hours. If set to 0, the code
   | will run until convergence.
21 | continuation = 0; % Continuation. If set to 1, continuation is on.
22 | helpdesigner = 1;
23 | miniter = 50;
24 | maxchange = 0.01;
25 | F_in = 5; % Input force in Newtons
26 | bs = 0.2; % Boundariespace
27 |
28 | %%%%%%%%% Added blocks or designless domains %%%%%%%%%
29 | blocksize = round([0.45*nelx,0.45*nely,0.55*nelx,0.55*nely,0;
   | 0.45*nelx 0.4*nely 0.55*nelx 0.45*nely 1]); % n*5 matrix
   | containing the x- and y- coordinates of the blocks upper left
   | and lower right corner (xtop,ytop,xbot,ybot) of each block in
   | each row, and the corresponding density.
30 |
31 | %%%%%%%%% Continuation %%%%%%%%%
32 | penal = penal0;
33 | sl = sl0; % Slack parameter: how many elements the COM is allowed
   | to move at the start
34 | slm = slm0; % Moment balance slack parameter at the start

```

```

35 slend = 0.01; % Slack parameter: how many elements the COM is
    allowed to move at the end
36 slmend = 0.5; % Moment balance slack parameter at the end
37 penalend = penal0;
38 contloop = 250; % Iteration when the continuation starts
39 contduration = 500; % Amount of iterations over which the slowly
    increasing continuation will distribute
40
41 %% INITIALIZATION
42 % Starting conditions implementation
43 if isempty(startingconditions) == 1
44     xmat = ones(nely,nelx)*volfrac; % Grey start
45 else
46     temp = struct2cell(load(startingconditions));
47     x0 = temp{1};
48     clear temp
49     x0 = reshape(x0,nely,nelx);
50     if size(x0)~= [nely,nelx]
51         error('startingconditions do not correspond to design
                space size')
52     end
53     xmat = volfrac*(1-x0)+(1-volfrac)*x0;
54 end
55
56 % Initialize values
57 [din,dout,fixeddofs,F,namesitu] = situation(nelx,nely,F_in,bs);
58 name = string(['V23_unbalanced(' sprintf('%4i',nelx) ',' sprintf('
    %4i',nely) ',' sprintf('%6.2f',volfrac) ',' sprintf('%2.1f',
    penal) ',' sprintf('%2.1f',rmin) ',' namesitu ')']);
59 [KE] = lk;
60 loop = 0;
61 change = 1;
62 [Ax,Ay,x_disp,y_disp,x_undisp,y_undisp,nel,edofMat] = Initialize(
    nelx,nely);
63 [H,Hs] = Initialize_filter(nelx,nely,rmin);
64 x = reshape(xmat,nel,1);
65 opt = mma(nel);
66 U = zeros(2*(nely+1)*(nelx+1),2);
67
68 % Include blocks and nondesign domains
69 [block_el_nrs_zero,block_el_nrs_one] = blocksmat2vec(blocksize,
    nelx,nely);
70 if blocks == 1
71     x(block_el_nrs_zero) = 1E-6;
72     x(block_el_nrs_one) = 1;
73 end
74 xflt = x;
75 figure(1);
76

```

```

77 %% START ITERATION
78 while change > maxchange || loop < miniter+1
79 loop = loop + 1;
80 xfltmat = reshape(xflt,nely,nelx);
81 % Calculate COM
82 [COMX,COMY] = COM(xfltmat);
83 % FE-ANALYSIS
84 [U,c,~]=FE(nelx,nely,xfltmat,penal,F,KE,din,dout,fixeddofs);
85 lambdac = U(:,2);
86 U = U(:,1);
87 % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
88 [dc] = dEEdx(nelx,nely,U,lambdac,penal,xfltmat,KE);
89 [~,~,Ux,Uy] = SFbal(xflt,nelx,nely,U,KE,penal,sl,Ax,Ay,din,dout,
    fixeddofs);
90 [~,~,mom] = SMBal(xflt,nelx,nely,U,KE,penal,slm,Ax,Ay,COMX,COMY,
    din,dout,fixeddofs);
91
92 g(1) = c;
93 dgdx(1:nel,1) = reshape(dc,nel,1);
94 g(2) = sum(xflt)/(volfrac*nel) - 1;
95 dgdx(1:nel,2) = 1/volfrac/nel;
96
97 % REVERSE FILTERING OF SENSITIVITIES
98 for i = 1:length(g)
99     [dgdx(1:nel,i)] = H*(dgdx(1:nel,i)./Hs);
100 end
101
102 if blocks == 1
103     for i = 1:length(g)
104         dgdx(union(block_el_nrs_zero,block_el_nrs_one),i) = 0;
105     end
106 end
107 xold=x;
108 % DESIGN UPDATE BY THE METHOD OF MOVING ASYMPTOTES
109 [opt,x] = opt.update(xold,g,dgdx);
110 if loop >= 10
111     if helpdesigner == 1 && c >= -1E-1
112         elin = floor(din/2)-floor(din/(2*nely));
113         elout = floor(dout/2)-floor(dout/(2*nely));
114         x([elin-nely/10:elin+nely/10,elout-nely/10:elout+nely/10])
            = max(x);
115     end
116 end
117 xflt(:) = (H*x(:))./Hs;
118 if blocks == 1
119     x(block_el_nrs_zero) = 1E-6;
120     x(block_el_nrs_one) = 1;
121 end
122 xmat = reshape(x,nely,nelx);

```

```

123   xmatflt = reshape(xflt,nely,nelx);
124   % PRINT RESULTS
125   bwfrac = (sum(x>0.99)+sum(x<0.01))/nel;
126   change = max(max(abs(x-xold)));
127   disp([' It.: ' sprintf('%4i',loop) ' Endeff.: ' sprintf('%10.4f'
    ,full(c)) ...
128       ' Vol.: ' sprintf('%6.3f',sum(sum(xmat))/(nelx*nely)) ...
129       ' ch.: ' sprintf('%6.3f',change ) ...
130       ' U: ' sprintf('%6.3f',sqrt(Ux^2+Uy^2)) ...
131       ' Mom: ' sprintf('%6.3f',mom)...
132       ' BWfrac:' sprintf('%6.3f',bwfrac)...
133       ' Uin:' sprintf('%6.3f',full(U(din)))...
134   ])
135   % PLOT DENSITIES AND STORE PROGRESS
136   % COM's are offset 0.5 elements because the grayscale command
    is offset
137   % 0.5 elements as well.
138   % Design values
139   figure(1)
140   colormap(gray); imagesc(-xmat); axis equal; axis tight; axis
    on;pause(1e-6);
141   hold on
142   title('Design values');
143   plot(COMX+0.5,COMY+0.5,'ob')
144   quiver(COMX+0.5,COMY+0.5,Ux,Uy,'b');
145   hold off
146   if record == 1
147   movie1(opt.iter) = getframe(gcf);
148   end
149   % Densities
150   figure(2)
151   colormap(gray); imagesc(-xmatflt.^3); axis equal; axis tight;
    axis on;pause(1e-6);
152   hold on
153   title('Densities');
154   plot(COMX+0.5,COMY+0.5,'ob')
155   quiver(COMX+0.5,COMY+0.5,Ux,Uy,'b');
156   hold off
157   if record == 1
158   movie2(opt.iter) = getframe(gcf);
159   end
160   % Other plots
161   Uxplot(opt.iter) = Ux;
162   Uyplot(opt.iter) = Uy;
163   Objplot(opt.iter) = c;
164   bwfracplot(opt.iter) = bwfrac;
165   Changeplot(opt.iter) = change;
166   xstore(:,loop) = xflt;
167

```

```

168 % Nonconvergent iteration limits (time or maxiter)
169 if opt.iter == maxiter
170     change = 0;
171     disp('Maximum number of iterations reached');
172 end
173 if maxtime ~= 0 && toc >= maxtime*3600
174     change = 0;
175     disp('Time limit reached')
176 end
177
178 % Continuation
179 if continuation == 1
180     if loop > contloop && loop < contloop+contduration
181         sl = sl-1/contduration*(sl0-slend);
182         slm = slm-1/contduration*(slm0-slmend);
183         penal = penal-1/contduration*(penal0-penalend);
184     end
185 end
186 end
187
188 %% After Final Iteration
189 x_disp((1:4),:) = x_undisp((1:4),:) + U(edofMat(:,[1 3 5 7]));
190 y_disp((1:4),:) = y_undisp((1:4),:) + U(edofMat(:,[2 4 6 8]));
191 save('unbalanced.mat','x')
192 save(strjoin(string([name,'xstore.mat'])), 'xstore', '-mat')
193 if plotdata == 1 % Allows turning on and off in options
194     figure(3);
195     subplot(2,2,1), plot([1:1:opt.iter],Uxplot,'r-'), xlabel('
        Iterations'),ylabel('Displacement [elements]'),title('
        COM-displacements');
196     hold on
197     plot([1:1:opt.iter],Uyplot,'b-'), xlabel('Iterations'),
        ylabel('Displacement [elements]');
198     legend('x-displacement','y-displacement');
199     subplot(2,2,2), plot([1:1:opt.iter],bwfracplot*100,'r-'),
        xlabel('Iterations'),ylabel('bwfrac [%]'),title('Black&
        white fraction');
200     subplot(2,2,3), plot([1:1:opt.iter],Objplot,'r-'), xlabel('
        Iterations'),ylabel('Displacement [elements]'),title('
        End-effector Displacement');
201     subplot(2,2,4), plot([1:1:opt.iter],Changeplot,'r-'),
        xlabel('Iterations'),ylabel('Density change'),title('
        Max change');
202     saveas(gcf,strjoin(string([name,'iterationdata.fig'])));
203     figure(4);
204     subplot(1,2,1), colormap(gray); imagesc(-xmat); axis equal
        ; axis tight; axis on;pause(1e-6);
205     subplot(1,2,2), colormap(gray); imagesc(-xmat); axis equal
        ; axis tight; axis on;pause(1e-6);

```

```

206     figure(4)
207     xplot = sparse(round(x));
208     [elplot,~,~] = find(xplot);
209     patch(x_disp(:,elplot),y_disp(:,elplot),'r','FaceAlpha',1,
          'EdgeColor','none');pause(1e-6);
210     hold on
211     plot(COMX+0.5,COMY+0.5,'ob')
212     quiver(COMX+0.5,COMY+0.5,Ux,Uy,'b');
213     dyout = 2*round(dout/2);
214     dxout = dyout - 1;
215     xout = floor(ceil(dout/2)/(nely+1))+0.5; % +0.5 to
          compensate for colormap(gray) command placing nodes at
          +-0.5 elements
216     yout = ceil(dout/2)-(xout-0.5)*(nely+1)-1+0.5;
217     quiver(xout,yout,U(dxout),U(dyout),'b');
218     saveas(gcf,strjoin(string([name,'Deformedplot.png'])));
219 end
220 %% Make videos
221 % Video design values
222     writerObj = VideoWriter(strjoin(string([name,'Design
          values'])));
223     writerObj.FrameRate = 10; % set the fps
224     % open the video writer
225     open(writerObj);
226     % write the frames to the video
227     for i=1:length(movie1)
228     % convert the image to a frame
229     frame = movie1(i) ;
230     writeVideo(writerObj, frame);
231     end
232     % close the writer object
233     close(writerObj);
234
235 % Video densities
236     writerObj = VideoWriter(strjoin(string([name,'Densities']
          )));
237     writerObj.FrameRate = 10; % set the fps
238     % open the video writer
239     open(writerObj);
240     % write the frames to the video
241     for i=1:length(movie2)
242     % convert the image to a frame
243     frame = movie2(i) ;
244     writeVideo(writerObj, frame);
245     end
246     % close the writer object
247     close(writerObj);
248
249

```

```

250 %% Auxiliary functions
251
252 %%%%%%%%% COM calculation %%%%%%%%%
253 function [COMX,COMY] = COM(xPhys)
254 [nely,nelx] = size(xPhys);
255 xdist = ones(nely,1)*[1:nelx];
256 ydist = [1:nely]'*ones(1,nelx);
257 COMX = sum(sum(xPhys.*xdist))/sum(sum(xPhys));
258 COMY = sum(sum(xPhys.*ydist))/sum(sum(xPhys));
259
260
261 %%%%%%%%% Initialize function %%%%%%%%%
262 function [Ax,Ay,x_disp,y_disp,x_undisp,y_undisp,nel,edofMat] =
    Initialize(nelx,nely)
263 % Function Initialize
264 nel = nelx*nely;
265 ndof = 2*(nelx+1)*(nely+1);
266 nodenrs = reshape(1:(nelx+1)*(nely+1),nely+1,nelx+1);
267 edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nel,1);
    % Used to make edofMat
268 edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],
    nel,1);
269 % Ax and Ay matrices
270 Ax = 0.25*sparse(kron([1:1:nel],ones(1,4)),reshape(edofMat(:,[1 3
    5 7])',4*nel,1),ones(1,4*nel),nel,ndof);
271 Ay = 0.25*sparse(kron([1:1:nel],ones(1,4)),reshape(edofMat(:,[2 4
    6 8])',4*nel,1),ones(1,4*nel),nel,ndof);
272 x_undisp = kron((0.5+ones(4,1)*[1:nelx]-[ones(1,nelx);zeros(2,nelx
    );ones(1,nelx)]),ones(1,nely));
273 y_undisp = kron(ones(1,nelx),(0.5+ones(4,1)*[1:nely]-[zeros(2,nely
    );ones(2,nely)]));
274 x_disp = x_undisp;
275 y_disp = y_undisp;
276
277 %%%%%%%%% Initialize Filter %%%%%%%%%
278 function [H,Hs] = Initialize_filter(nelx,nely,rmin)
279 iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
280 jH = ones(size(iH));
281 sH = zeros(size(iH));
282 k = 0;
283 for i1 = 1:nelx
284     for j1 = 1:nely
285         e1 = (i1-1)*nely+j1;
286         for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),
            nelx)
287             for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)
                -1),nely)
288                 e2 = (i2-1)*nely+j2;
289                 k = k+1;

```

```

290         iH(k) = e1;
291         jH(k) = e2;
292         sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
293     end
294 end
295 end
296 end
297 H = sparse(iH,jH,sH);
298 Hs = sum(H,2);
299
300 function [block_el_nrs_zero,block_el_nrs_one] = blocksmat2vec(
    blocksize,nelx,nely)
301 % This function is used to predefine rectangular non-design
    domains in
302 % the total design domain, with a predefined density of 0 or 1.
303 %
304 % block_el_nrs_zero    element numbers in the vector x with a
    predefined
305 %                      value of 0 and sensitivity of 0.
306 % block_el_nrs_one    element numbers in the vector x with a
    predefined
307 %                      value of 1 and sensitivity of 0.
308 %
309 % Blocksize          n*5 matrix containing the upper left and
    lower
310 %                   right x and y coordinates of n rectangular
    bodies
311 %                   of which the element densities should be 0
    or 1.
312 %                   The order of the columns is as follows:
313 %                   [xtop,ytop,xbot,ybot,density]
314 %                   with the density being 0 or 1.
315 % nelx                # elements in the design domain in x-
    direction
316 % nely                # elements in the design domain in y-
    direction
317
318 [n,~] = size(blocksize);
319 block_el_nrs_zero = [];
320 block_el_nrs_one = [];
321 for i = 1:n
322     blocksize(i,:);
323     numbers = reshape(1:nelx*nely,nely,nelx);
324     blocknrs_i = numbers([blocksize(i,2):blocksize(i,4)],[
        blocksize(i,1):blocksize(i,3)]);
325     if blocksize(i,5) == 1
326         block_el_nrs_one = union(blocknrs_i,block_el_nrs_one);
327     elseif blocksize(i,5) == 0
328         block_el_nrs_zero = union(blocknrs_i,block_el_nrs_zero);

```



```

329     else
330         disp('error: blocks in "blocksize" not defined as 0 or 1')
331     end
332 end
333
334 %
335 %
336 % Original code was written by Ole Sigmund, Department of Solid
337 % Mechanics, Technical University of Denmark, DK-2800 Lyngby,
338 % Please sent your comments to the author: sigmund@fam.dtu.dk
339 %
340 % The code is intended for educational purposes and theoretical
341 % details are discussed in the paper
342 % "A 99 line topology optimization code written in Matlab"
343 % by Ole Sigmund (2001), Structural and Multidisciplinary
344 % Optimization, Vol 21, pp. 120--127.
345 %
346 % The code as well as a postscript version of the paper can be
347 % downloaded from the web-site: http://www.topopt.dtu.dk
348 %
349 % Disclaimer:
350 % The author reserves all rights but does not guaranty that the
351 % code is free from errors. Furthermore, he shall not be liable in any
352 % event caused by the use of the program.
353 %

```

E.12 FE

```

1  %%%%%%%%%%% FE-ANALYSIS
   %%%%%%%%%%%
2  function [U,Uout,K]=FE(nelx,nely,x,penal,F,KE,din,dout,fixeddofs)
3  nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
4  edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
5  edofMat = repmat(edofVec,1,8)+repmat([-2 -1 2*nely+[0 1 2 3] 0 1],
   nelx*nely,1);
6  iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
7  jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
8
9  % DEFINE LOADS AND SUPPORTS
10 sK = reshape(KE(:)*(x(:)'.^penal),64*nelx*nely,1);
11 K = sparse(iK,jK,sK); K = (K+K')/2;
12 K(din,din) = K(din,din) + 0.1;
13 K(dout,dout) = K(dout,dout) + 0.1;
14 alldofs      = [1:2*(nely+1)*(nelx+1)];
15 freedofs     = setdiff(alldofs,fixeddofs);
16
17 % SOLVING
18 U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
19 U(fixeddofs,:)= 0;
20 Uout = U(dout,1);

```

lk

```

1  %%%%%%%%%%% ELEMENT STIFFNESS MATRIX
   %%%%%%%%%%%
2  function [KE]=lk
3  E = 10;
4  nu = 0.3;
5  k=[ 1/2-nu/6    1/8+nu/8  -1/4-nu/12  -1/8+3*nu/8  ...
6     -1/4+nu/12  -1/8-nu/8    nu/6         1/8-3*nu/8];
7  KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
8                   k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
9                   k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
10                  k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
11                  k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
12                  k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
13                  k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
14                  k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];

```

mma

```

1  %
2  %   Written in May 1999 by

```

```

3 % Krister Svanberg <krille@math.kth.se>
4 % Department of Mathematics
5 % SE-10044 Stockholm, Sweden.
6 %
7 % Modified ("spdiags" instead of "diag") April 2002
8 %
9 %
10 % This function mmasub performs one MMA-iteration, aimed at
11 % solving the nonlinear programming problem:
12 %
13 % Minimize f_0(x) + a_0*z + sum( c_i*y_i + 0.5*d_i*(y_i)^2 )
14 % subject to f_i(x) - a_i*z - y_i <= 0, i = 1,...,m
15 %             xmin_j <= x_j <= xmax_j, j = 1,...,n
16 %             z >= 0, y_i >= 0, i = 1,...,m
17 %*** INPUT:
18 %
19 % m = The number of general constraints.
20 % n = The number of variables x_j.
21 % iter = Current iteration number ( =1 the first time mmasub is
22 % called).
23 % xval = Column vector with the current values of the variables
24 % x_j.
25 % xmin = Column vector with the lower bounds for the variables
26 % x_j.
27 % xmax = Column vector with the upper bounds for the variables
28 % x_j.
29 % xold1 = xval, one iteration ago (provided that iter>1).
30 % xold2 = xval, two iterations ago (provided that iter>2).
31 % f0val = The value of the objective function f_0 at xval.
32 % df0dx = Column vector with the derivatives of the objective
33 % function
34 % f_0 with respect to the variables x_j, calculated at
35 % xval.
36 % df0dx2 = Column vector with the non-mixed second derivatives of
37 % the
38 % objective function f_0 with respect to the variables
39 % x_j,
40 % calculated at xval. df0dx2(j) = the second derivative
41 % of f_0 with respect to x_j (twice).
42 % Important note: If second derivatives are not available
43 % ,
44 % simply let df0dx2 = 0*df0dx.
45 % fval = Column vector with the values of the constraint
46 % functions f_i,
47 % calculated at xval.
48 % dfdx = (m x n)-matrix with the derivatives of the constraint
49 % functions
50 % f_i with respect to the variables x_j, calculated at
51 % xval.

```

```

40 %           dfdx(i,j) = the derivative of f_i with respect to x_j.
41 % dfdx2 = (m x n)-matrix with the non-mixed second derivatives of
42 %           the
43 %           constraint functions f_i with respect to the variables
44 %           x_j,
45 %           calculated at xval. dfdx2(i,j) = the second derivative
46 %           of f_i with respect to x_j (twice).
47 %           Important note: If second derivatives are not available
48 %           ,
49 %           simply let dfdx2 = 0*dfdx.
50 % low  = Column vector with the lower asymptotes from the
51 %           previous
52 %           iteration (provided that iter>1).
53 % upp  = Column vector with the upper asymptotes from the
54 %           previous
55 %           iteration (provided that iter>1).
56 % a0   = The constants a_0 in the term a_0*z.
57 % a    = Column vector with the constants a_i in the terms a_i*z
58 %           .
59 % c    = Column vector with the constants c_i in the terms c_i*
60 %           y_i.
61 % d    = Column vector with the constants d_i in the terms 0.5*
62 %           d_i*(y_i)^2.
63 %
64 %*** OUTPUT:
65 %
66 % xmma = Column vector with the optimal values of the variables
67 %           x_j
68 %           in the current MMA subproblem.
69 % ymma = Column vector with the optimal values of the variables
70 %           y_i
71 %           in the current MMA subproblem.
72 % zmma = Scalar with the optimal value of the variable z
73 %           in the current MMA subproblem.
74 % lam  = Lagrange multipliers for the m general MMA constraints.
75 % xsi  = Lagrange multipliers for the n constraints  $\alpha_j - x_j$ 
76 %            $\leq 0$ .
77 % eta  = Lagrange multipliers for the n constraints  $x_j - \beta_j$ 
78 %            $\leq 0$ .
79 % mu   = Lagrange multipliers for the m constraints  $-y_i \leq 0$ .
80 % zet  = Lagrange multiplier for the single constraint  $-z \leq 0$ .
81 % s    = Slack variables for the m general MMA constraints.
82 % low  = Column vector with the lower asymptotes, calculated and
83 %           used
84 %           in the current MMA subproblem.
85 % upp  = Column vector with the upper asymptotes, calculated and
86 %           used
87 %           in the current MMA subproblem.
88 %

```

```

75
76 classdef mma
77     properties
78         n;
79         m;
80         iter = 0;
81         x_min = 1e-3; % was 1e-3
82         x_max = 1;
83         xold1;
84         xold2;
85         a0 = 1.0;
86         a;
87         c;
88         d;
89         move = 0.2; %was 0.2
90         low;
91         upp;
92         dx;
93         change = 1;
94         xmin;
95         xmax;
96     end
97
98     methods
99         function obj = mma(n)
100             obj.n = n;
101             obj.xmin = obj.x_min * ones(obj.n, 1);
102             obj.xmax = obj.x_max * ones(obj.n, 1);
103             obj.dx = obj.xmax - obj.xmin;
104         end
105         function [obj, xmma] = update(obj,xval,g,dg)
106             obj.iter = obj.iter + 1;
107             df0dx = dg(:,1);
108             dfdx = dg(:,2:end)';
109             fval = g(2:end)';
110
111             if obj.iter == 1
112                 obj.m = size(df0dx,1);
113                 obj.a = zeros(obj.m, 1);
114                 obj.c = 1e3 * ones(obj.m, 1);
115                 obj.d = ones(obj.m, 1);
116             end
117
118             epsimin = sqrt(obj.m+obj.n)*10^(-9);
119             feps = 0.00001;
120             asyinit = 0.5;
121             asyincr = 1.2; %was 1.2
122             asydecr = 0.7; %was 0.7
123             albefa = 0.1;

```

```

124     een = ones(obj.n,1);
125     zeron = zeros(obj.n,1);
126
127     % Calculation of the asymptotes low and upp :
128     if obj.iter < 2.5
129         obj.low = xval - asyinit*obj.dx;
130         obj.upp = xval + asyinit*obj.dx;
131     else
132         zzz = (xval-obj.xold1).*(obj.xold1-obj.xold2);
133         factor = een;
134         factor(zzz > 0) = asyincr;
135         factor(zzz < 0) = asydecr;
136         lowmin = xval - 10.0 * obj.dx;
137         lowmax = xval - 0.01 * obj.dx;
138         uppmmin = xval + 0.01 * obj.dx;
139         uppmmax = xval + 10.0 * obj.dx;
140         obj.low = xval - factor.*(obj.xold1 - obj.low);
141         obj.upp = xval + factor.*(obj.upp - obj.xold1);
142         obj.low = max(obj.low, lowmin);
143         obj.low = min(obj.low, lowmax);
144         obj.upp = min(obj.upp, uppmmax);
145         obj.upp = max(obj.upp, uppmmin);
146     end
147
148     % Calculation of the bounds alfa and beta :
149     chmax = obj.move *obj.dx;
150     zzz = obj.low + albefa*(xval-obj.low);
151     alfa = max(zzz,obj.xmin);
152     alfa = max(alfa,xval - chmax);
153     zzz = obj.upp - albefa*(obj.upp-xval);
154     beta = min(zzz,obj.xmax);
155     beta = min(beta,xval + chmax);
156
157     % Calculations of p0, q0, P, Q and b.
158     ux1 = obj.upp-xval;
159     ux2 = ux1.*ux1;
160     xl1 = xval-obj.low;
161     xl2 = xl1.*xl1;
162     ul1 = obj.upp - obj.low;
163     ulinv1 = een./ul1;
164     uxinv1 = een./ux1;
165     xlinv1 = een./xl1;
166     p0 = zeron;
167     p0(df0dx > 0) = df0dx(df0dx > 0);
168     p0 = p0 + 0.001*abs(df0dx) + feps*ulinv1;
169     p0 = p0.*ux2;
170     q0 = zeron;
171     q0(df0dx < 0) = -df0dx(df0dx < 0);
172     q0 = q0 + 0.001*abs(df0dx) + feps*ulinv1;

```

```

173     q0 = q0.*xl2;
174     P = zeros(obj.m,obj.n);
175     P(dfdx > 0) = dfdx(find(dfdx > 0));
176     P = P * spdiags(ux2,0,obj.n,obj.n);
177     Q = zeros(obj.m,obj.n);
178     Q(dfdx < 0) = -dfdx(dfdx < 0);
179     Q = Q * spdiags(xl2,0,obj.n,obj.n);
180     b = P*uxinv1 + Q*xlinv1 - fval ;
181
182     %% Solving the subproblem by a primal-dual Newton
183     method
184     [xmma,~,~,~,~,~,~,~] = ...
185     subsolv(obj.m,obj.n,epsimin,obj.low,obj.upp,alfa,beta,
186     p0,q0,P,Q,obj.a0,obj.a,b,obj.c,obj.d);
187
188     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
189     obj.change = mean(abs(xmma(:) - xval(:)));
190     obj.xold2 = obj.xold1;
191     obj.xold1 = xval;
192
193     end
194 end
195
196 % This is the file subsolv.m
197 %
198 function [xmma,ymma,zmma,lamma,xsimma,etamma,mumma,zetmma,smma] =
199     ...
200     subsolv(m,n,epsimin,low,upp,alfa,beta,p0,q0,P,Q,a0,a,b,c,d)
201 %
202 % Written in May 1999 by
203 % Krister Svanberg <krille@math.kth.se>
204 % Department of Mathematics
205 % SE-10044 Stockholm, Sweden.
206 % This function subsolv solves the MMA subproblem:
207 %
208 % minimize    SUM[ p0j/(uppj-xj) + q0j/(xj-lowj) ] + a0*z +
209 %             + SUM[ ci*yi + 0.5*di*(yi)^2 ],
210 %
211 % subject to SUM[ pij/(uppj-xj) + qij/(xj-lowj) ] - ai*z - yi <=
212 %             bi,
213 %             alfaj <= xj <= betaj, yi >= 0, z >= 0.
214 % Input:  m, n, low, upp, alfa, beta, p0, q0, P, Q, a0, a, b, c, d
215 % Output: xmma,ymma,zmma, slack variables and Lagrange multiplers.
216 %

```

```

217 een = ones(n,1);
218 eem = ones(m,1);
219 epsi = 1;
220 epsvecn = epsi*een;
221 epsvecm = epsi*eem;
222 x = 0.5*(alfa+beta);
223 y = eem;
224 z = 1;
225 lam = eem;
226 xsi = een./(x-alfa);
227 xsi = max(xsi,een);
228 eta = een./(beta-x);
229 eta = max(eta,een);
230 mu = max(eem,0.5*c);
231 zet = 1;
232 s = eem;
233 itera = 0;
234
235 while epsi > epsimin
236     epsvecn = epsi*een;
237     epsvecm = epsi*eem;
238     ux1 = upp-x;
239     xl1 = x-low;
240     ux2 = ux1.*ux1;
241     xl2 = xl1.*xl1;
242     uxinv1 = een./ux1;
243     xlinv1 = een./xl1;
244
245     plam = p0 + P'*lam ;
246     qlam = q0 + Q'*lam ;
247     gvec = P*uxinv1 + Q*xlinv1;
248     dpsidx = plam./ux2 - qlam./xl2 ;
249
250     rex = dpsidx - xsi + eta;
251     rey = c + d.*y - mu - lam;
252     rez = a0 - zet - a'*lam;
253     relam = gvec - a*z - y + s - b;
254     rehsi = xsi.*(x-alfa) - epsvecn;
255     reeta = eta.*(beta-x) - epsvecn;
256     remu = mu.*y - epsvecm;
257     rezet = zet*z - epsi;
258     res = lam.*s - epsvecm;
259
260     residu1 = [rex' rey' rez]';
261     residu2 = [relam' rehsi' reeta' remu' rezet res]';
262     residu = [residu1' residu2]';
263     residunorm = sqrt(residu'*residu);
264     residumax = max(abs(residu));
265

```



```

266 ittt = 0;
267 while residumax > 0.9*epsi && ittt < 100
268     ittt=ittt + 1;
269     itera=itera + 1;
270
271     ux1 = upp-x;
272     x11 = x-low;
273     ux2 = ux1.*ux1;
274     x12 = x11.*x11;
275     ux3 = ux1.*ux2;
276     x13 = x11.*x12;
277     uxinv1 = een./ux1;
278     xlinv1 = een./x11;
279     uxinv2 = een./ux2;
280     xlinv2 = een./x12;
281     plam = p0 + P'*lam ;
282     qlam = q0 + Q'*lam ;
283     gvec = P*uxinv1 + Q*xlinv1;
284     GG = P*spdiags(uxinv2,0,n,n) - Q*spdiags(xlinv2,0,n,n);
285     dpsidx = plam./ux2 - qlam./x12 ;
286     delx = dpsidx - epsvecn./(x-alfa) + epsvecn./(beta-x);
287     dely = c + d.*y - lam - epsvecm./y;
288     delz = a0 - a'*lam - epsi/z;
289     dellam = gvec - a*z - y - b + epsvecm./lam;
290     diagx = plam./ux3 + qlam./x13;
291     diagx = 2*diagx + xsi./(x-alfa) + eta./(beta-x);
292     diagxinv = een./diagx;
293     diagy = d + mu./y;
294     diagyinv = eem./diagy;
295     diaglam = s./lam;
296     diaglami = diaglam+diagyinv;
297
298     if m < n
299         blam = dellam + dely./diagy - GG*(delx./diagx);
300         bb = [blam' delz]';
301         Alam = spdiags(diaglami,0,m,m) + GG*spdiags(diagxinv,0,n,n)
302             *GG';
303         AA = [Alam      a
304              a'      -zet/z ];
305         solut = AA\bb;
306         dlam = solut(1:m);
307         dz = solut(m+1);
308         dx = -delx./diagx - (GG'*dlam)./diagx;
309     else
310         diaglamiinv = eem./diaglami;
311         dellami = dellam + dely./diagy;
312         Axx = spdiags(diagx,0,n,n) + GG'*spdiags(diaglamiinv,0,m,m)
313             *GG;
314         azz = zet/z + a'*(a./diaglami);

```

```

313     axz = -GG'*(a./diaglamyi);
314     bx = delx + GG'*(dellamyi./diaglamyi);
315     bz = delz - a'*(dellamyi./diaglamyi);
316     AA = [Axx    axz
317           axz'   azz ];
318     bb = [-bx' -bz]';
319     solut = AA\bb;
320     dx = solut(1:n);
321     dz = solut(n+1);
322     dlam = (GG*dx)./diaglamyi - dz*(a./diaglamyi) + dellamyi./
           diaglamyi;
323     end
324
325     dy = -dely./diagy + dlam./diagy;
326     dxsi = -xsi + epsvecn./(x-alfa) - (xsi.*dx)/(x-alfa);
327     deta = -eta + epsvecn./(beta-x) + (eta.*dx)/(beta-x);
328     dmu = -mu + epsvecm./y - (mu.*dy)./y;
329     dzet = -zet + epsi/z - zet*dz/z;
330     ds = -s + epsvecm./lam - (s.*dlam)./lam;
331     xx = [ y' z lam' xsi' eta' mu' zet s']';
332     dxx = [dy' dz dlam' dxsi' deta' dmu' dzet ds']';
333
334     stepxx = -1.01*dxx./xx;
335     stmxx = max(stepxx);
336     stepalfa = -1.01*dx./(x-alfa);
337     stmalfa = max(stepalfa);
338     stepbeta = 1.01*dx./(beta-x);
339     stmbeta = max(stepbeta);
340     stmambe = max(stmalfa, stmbeta);
341     stmalbexx = max(stmambe, stmxx);
342     stminv = max(stmalbexx, 1);
343     steg = 1/stminv;
344
345     xold = x;
346     yold = y;
347     zold = z;
348     lamold = lam;
349     xsiold = xsi;
350     etaold = eta;
351     muold = mu;
352     zetold = zet;
353     sold = s;
354
355     itto = 0;
356     resinew = 2*residunorm;
357     while resinew > residunorm && itto < 50
358     itto = itto+1;
359
360     x = xold + steg*dx;

```

```

361     y = yold + steg*dy;
362     z = zold + steg*dz;
363     lam = lamold + steg*dlam;
364     xsi = xsiold + steg*dxsi;
365     eta = etaold + steg*deta;
366     mu = muold + steg*dmu;
367     zet = zetold + steg*dzet;
368     s = sold + steg*ds;
369     ux1 = upp-x;
370     xl1 = x-low;
371     ux2 = ux1.*ux1;
372     xl2 = xl1.*xl1;
373     uxinv1 = een./ux1;
374     xlinv1 = een./xl1;
375     plam = p0 + P'*lam ;
376     qlam = q0 + Q'*lam ;
377     gvec = P*uxinv1 + Q*xlinv1;
378     dpsidx = plam./ux2 - qlam./xl2 ;
379
380     rex = dpsidx - xsi + eta;
381     rey = c + d.*y - mu - lam;
382     rez = a0 - zet - a'*lam;
383     relam = gvec - a*z - y + s - b;
384     rexsi = xsi.*(x-alfa) - epsvecn;
385     reeta = eta.*(beta-x) - epsvecn;
386     remu = mu.*y - epsvecm;
387     rezet = zet*z - epsi;
388     res = lam.*s - epsvecm;
389
390     residu1 = [rex' rey' rez]';
391     residu2 = [relam' rexsi' reeta' remu' rezet res]';
392     residu = [residu1' residu2]';
393     resinew = sqrt(residu'*residu);
394     steg = steg/2;
395     end
396     residunorm=resinew;
397     residumax = max(abs(residu));
398     steg = 2*steg;
399     end
400     epsi = 0.1*epsi;
401     end
402
403     xmma = x;
404     ymma = y;
405     zmma = z;
406     lamma = lam;
407     xsimma = xsi;
408     etamma = eta;
409     mumma = mu;

```

```

410 zetmma = zet;
411 smma = s;
412
413 end

```

Runoptimization

```

1  %% This file is used to run the optimization of a balanced
   compliant mechanism
2  tic
3  clear all
4  close all
5
6  % Parameters
7  nelx = 400;
8  nely = 400;
9  volfrac = 0.3;
10 penal = 3;
11 rmin = 8;
12 sl = 0.5;
13 slm = 20;
14 situation = @FIbs;
15
16 % Optimization runs
17 topmma_unbalanced(nelx,nely,volfrac,penal,rmin,sl,slm,situation
   ,[])
18 % close all
19 topmma_balanced(nelx,nely,volfrac,penal,rmin,sl,slm,situation,'
   unbalanced.m')

```

E.13 Situations

Some situations are presented here. They are divided into two categories: FIbs (Force Inverter with Boundary Space) and FTlbrt (Force Transmitter Left Bottom to Right Top). The x or y behind FIbs means the output is in the x- or y-direction. If a value is also added, this denotes the height in the domain where the output node is located. The x or y behind FTlbrt means first the input direction, then the output direction.

E.14 FIbs04x

```

1  function [din,dout,fixeddofs,F,namesitu] = FIbs04x(nelx,nely,F_in,
   bs)
2  % In this situation, the force inverter is placed in the middle
   of a
3  % design domain with free design domain all around. The output
   is shifted

```

```

4 %   upward to 0.4 of the total design domain in y-direction.
5
6
7 %   din           Input DOF number
8 %   dout          Output DOF number
9 %   fixeddofs     Constrained DOF numbers
10 %  F             Force vector on all DOFs
11 %  namesitu      Situation name, used for naming datasets
12
13 %  nelx           Number of elements in x-direction
14 %  nely           Number of elements in y-direction
15 %  F_in          Magnitude of the input force
16 %  bs            Boundary domain size as part of design domain. 0
17 %               is no
18 %               boundary, 0.5 is no interior design domain and
19 %               therefore
20 %               infeasible. bs always should be between 0 and 0.5.
21
22 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
23     error('Error 1. Boundary design domain wrongly defined. Check
24           input situation.');
```

```

25 elseif round(0.1*nely) ~= 0.1*nely
26     error('Error 2. Design domain too small for boundary
27           conditions to be implemented correctly. Nelx and Nely have
28           to be a factor of 10.');
```

```

29 end
30 namesitu = 'FIbs04x';
31 din = 2*(bs)*nelx*(nely+1)+nely+1;
32 dout = 2*(1-bs)*nelx*(nely+1)+nely*0.8+1;
33 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2], 2*(bs*nelx+1)
34                 *(nely+1)-2*bs*nely-[0,1]);
35 F = sparse(2*(nely+1)*(nelx+1), 2);
36 F(din,1) = F_in;
37 F(dout,2) = -F_in;
38 end

```

E.15 FIbs04y

```

1 function [din,dout,fixeddofs,F,namesitu] = FIbs04y(nelx,nely,F_in,
2           bs)
3 %   In this situation, the force inverter is placed in the middle
4 %   of a
5 %   design domain with free design domain all around. The output
6 %   is shifted
7 %   upward to 0.4 of the total design domain in y-direction.
8
9
10
11 %  din           Input DOF number

```

```

8 % dout      Output DOF number
9 % fixeddofs Constrained DOF numbers
10 % F        Force vector on all DOFs
11 % namesitu Situation name, used for naming datasets
12
13 % nelx      Number of elements in x-direction
14 % nely      Number of elements in y-direction
15 % F_in     Magnitude of the input force
16 % bs       Boundary domain size as part of design domain. 0
17 %          is no
18 %          boundary, 0.5 is no interior design domain and
19 %          therefore
20 %          infeasible. bs always should be between 0 and 0.5.
21
22 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
23     error('Error 1. Boundary design domain wrongly defined. Check
24           input situation.');
```

```

25 elseif round(0.1*nely) ~= 0.1*nely
26     error('Error 2. Design domain too small for boundary
27           conditions to be implemented correctly. Nelx and Nely have
28           to be a factor of 10.');
```

```

29 end
30 namesitu = 'FIbs04y';
31 din = 2*(bs)*nelx*(nely+1)+nely+1;
32 dout = 2*(1-bs)*nelx*(nely+1)+nely*0.8+2;
33 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2], 2*(bs*nelx+1)
34                 *(nely+1)-2*bs*nely-[0,1]);
35 F = sparse(2*(nely+1)*(nelx+1), 2);
36 F(din,1) = F_in;
37 F(dout,2) = -F_in;
38 end

```

E.16 FIbs025x

```

1 function [din,dout,fixeddofs,F,namesitu] = FIbs025x(nelx,nely,F_in
2 ,bs)
3 % In this situation, the force inverter is placed in the middle
4 % of a
5 % design domain with free design domain all around. The output
6 % is shifted
7 % upward to 0.25 of the total design domain in y-direction.
8
9 % din      Input DOF number
10 % dout     Output DOF number
11 % fixeddofs Constrained DOF numbers
12 % F        Force vector on all DOFs
13 % namesitu Situation name, used for naming datasets

```

```

12
13 % nelx      Number of elements in x-direction
14 % nely      Number of elements in y-direction
15 % F_in      Magnitude of the input force
16 % bs        Boundary domain size as part of design domain. 0
    is no
17 %           boundary, 0.5 is no interior design domain and
    therefore
18 %           infeasible. bs always should be between 0 and 0.5.
19
20 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
21     error('Error 1. Boundary design domain wrongly defined. Check
    input situation.');
```

```

22 elseif round(0.1*nely) ~= 0.1*nely
23     error('Error 2. Design domain too small for boundary
    conditions to be implemented correctly. Nelx and Nely have
    to be a factor of 10.');
```

```

24 end
25 namesitu = 'FIbs025x';
26 din =2*(bs)*nelx*(nely+1)+nely+1;
27 dout= 2*(1-bs)*nelx*(nely+1)+nely*0.5+1;
28 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(bs*nelx+1)
    *(nely+1)-2*bs*nely-[0,1]);
29 F = sparse(2*(nely+1)*(nelx+1),2);
30 F(din,1) = F_in;
31 F(dout,2) = -F_in;
32 end
```

E.17 FIbs025y

```

1 function [din,dout,fixeddofs,F,namesitu] = FIbs025y(nelx,nely,F_in
    ,bs)
2 % In this situation, the force inverter is placed in the middle
    of a
3 % design domain with free design domain all around. The output
    is shifted
4 % upward to 0.25 of the total design domain in y-direction.
5
6
7 % din        Input DOF number
8 % dout       Output DOF number
9 % fixeddofs  Constrained DOF numbers
10 % F          Force vector on all DOFs
11 % namesitu   Situation name, used for naming datasets
12
13 % nelx      Number of elements in x-direction
14 % nely      Number of elements in y-direction
15 % F_in      Magnitude of the input force
```

```

16 % bs          Boundary domain size as part of design domain. 0
    is no
17 %           boundary, 0.5 is no interior design domain and
    therefore
18 %           infeasible. bs always should be between 0 and 0.5.
19
20 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
21     error('Error 1. Boundary design domain wrongly defined. Check
    input situation.');
```

```

22 elseif round(0.1*nely) ~= 0.1*nely
23     error('Error 2. Design domain too small for boundary
    conditions to be implemented correctly. Nelx and Nely have
    to be a factor of 10.');
```

```

24 end
25 namesitu = 'Fibs025y';
26 din =2*(bs)*nelx*(nely+1)+nely+1;
27 dout= 2*(1-bs)*nelx*(nely+1)+nely*0.5+2;
28 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(bs*nelx+1)
    *(nely+1)-2*bs*nely-[0,1]);
29 F = sparse(2*(nely+1)*(nelx+1),2);
30 F(din,1) = F_in;
31 F(dout,2) = -F_in;
32 end
```

E.18 Fibs075x

```

1 function [din,dout,fixeddofs,F,namesitu] = Fibs075x(nelx,nely,F_in
    ,bs)
2 %   In this situation, the force inverter is placed in the middle
    of a
3 %   design domain with free design domain all around. The output
    is shifted
4 %   upward to 0.75 of the total design domain in y-direction.
5
6
7 %   din          Input DOF number
8 %   dout         Output DOF number
9 %   fixeddofs    Constrained DOF numbers
10 %   F            Force vector on all DOFs
11 %   namesitu     Situation name, used for naming datasets
12
13 %   nelx         Number of elements in x-direction
14 %   nely         Number of elements in y-direction
15 %   F_in        Magnitude of the input force
16 %   bs          Boundary domain size as part of design domain. 0
    is no
17 %           boundary, 0.5 is no interior design domain and
    therefore
```



```

18 %             infeasible. bs always should be between 0 and 0.5.
19
20 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
21     error('Error 1. Boundary design domain wrongly defined. Check
           input situation. ');
22 elseif round(0.1*nely) ~= 0.1*nely
23     error('Error 2. Design domain too small for boundary
           conditions to be implemented correctly. Nelx and Nely have
           to be a factor of 10. ');
24 end
25 namesitu = 'Fibs075x';
26 din =2*(bs)*nelx*(nely+1)+nely+1;
27 dout= 2*(1-bs)*nelx*(nely+1)+nely*1.5+1;
28 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(bs*nelx+1)
           *(nely+1)-2*bs*nely-[0,1]);
29 F = sparse(2*(nely+1)*(nelx+1),2);
30 F(din,1) = F_in;
31 F(dout,2) = -F_in;
32 end

```

E.19 Fibs075y

```

1 function [din,dout,fixeddofs,F,namesitu] = Fibs075y(nelx,nely,F_in
,bs)
2 %   In this situation, the force inverter is placed in the middle
of a
3 %   design domain with free design domain all around. The output
is shifted
4 %   upward to 0.75 of the total design domain in y-direction.
5
6
7 %   din           Input DOF number
8 %   dout          Output DOF number
9 %   fixeddofs     Constrained DOF numbers
10 %   F             Force vector on all DOFs
11 %   namesitu     Situation name, used for naming datasets
12
13 %   nelx          Number of elements in x-direction
14 %   nely          Number of elements in y-direction
15 %   F_in         Magnitude of the input force
16 %   bs           Boundary domain size as part of design domain. 0
is no
17 %               boundary, 0.5 is no interior design domain and
therefore
18 %               infeasible. bs always should be between 0 and 0.5.
19
20 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely

```

```

21     error('Error 1. Boundary design domain wrongly defined. Check
        input situation. ');
22 elseif round(0.1*nely) ~= 0.1*nely
23     error('Error 2. Design domain too small for boundary
        conditions to be implemented correctly. Nelx and Nely have
        to be a factor of 10. ');
24 end
25 namesitu = 'Fibs075y';
26 din =2*(bs)*nelx*(nely+1)+nely+1;
27 dout= 2*(1-bs)*nelx*(nely+1)+nely*1.5+2;
28 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(bs*nelx+1)
        *(nely+1)-2*bs*nely-[0,1]);
29 F = sparse(2*(nely+1)*(nelx+1),2);
30 F(din,1) = F_in;
31 F(dout,2) = -F_in;
32 end

```

E.20 Fibsx

```

1 function [din,dout,fixeddofs,F,namesitu] = Fibsx(nelx,nely,F_in,bs
)
2 % In this situation, the force inverter is placed in the middle
of a
3 % design domain with free design domain all around.
4
5
6 % din      Input DOF number
7 % dout     Output DOF number
8 % fixeddofs Constrained DOF numbers
9 % F        Force vector on all DOFs
10 % namesitu Situation name, used for naming datasets
11
12 % nelx     Number of elements in x-direction
13 % nely     Number of elements in y-direction
14 % F_in    Magnitude of the input force
15 % bs      Boundary domain size as part of design domain. 0
is no
16 %         boundary, 0.5 is no interior design domain and
therefore
17 %         infeasible. bs always should be between 0 and 0.5.
18
19
20 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
21     error('Error 1. Boundary design domain wrongly defined. Check
        input situation. ');
22 elseif round(0.1*nely) ~= 0.1*nely
23     error('Error 2. Design domain too small for boundary
        conditions to be implemented correctly. Nelx and Nely have

```

```

                to be a factor of 10.');
```

24 end

25 namesitu = 'Fibsx';

26 din =2*(bs)*nelx*(nely+1)+nely+1;

27 dout= 2*(1-bs)*nelx*(nely+1)+nely+1;

28 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(bs*nelx+1)
 *(nely+1)-2*bs*nely-[0,1]);

29 F = sparse(2*(nely+1)*(nelx+1),2);

30 F(din,1) = F_in;

31 F(dout,2) = -F_in;

32 end

E.21 Fibsy

```

1 function [din,dout,fixeddofs,F,namesitu] = Fibsy(nelx,nely,F_in,bs
  )
2 %   In this situation, the force inverter is placed in the middle
  of a
3 %   design domain with free design domain all around.
4
5
6 %   din           Input DOF number
7 %   dout          Output DOF number
8 %   fixeddofs     Constrained DOF numbers
9 %   F             Force vector on all DOFs
10 %  namesitu      Situation name, used for naming datasets
11
12 %  nelx          Number of elements in x-direction
13 %  nely          Number of elements in y-direction
14 %  F_in         Magnitude of the input force
15 %  bs           Boundary domain size as part of design domain. 0
  is no
16 %              boundary, 0.5 is no interior design domain and
  therefore
17 %              infeasible. bs always should be between 0 and 0.5.
18
19
20 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
21     error('Error 1. Boundary design domain wrongly defined. Check
  input situation.');
```

22 elseif round(0.1*nely) ~= 0.1*nely

23 error('Error 2. Design domain too small for boundary
 conditions to be implemented correctly. Nelx and Nely have
 to be a factor of 10.');

24 end

25 namesitu = 'Fibsy';

26 din =2*(bs)*nelx*(nely+1)+nely+1;

27 dout= 2*(1-bs)*nelx*(nely+1)+nely+2;

```

28 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(bs*nelx+1)
    *(nely+1)-2*bs*nely-[0,1]);
29 F = sparse(2*(nely+1)*(nelx+1),2);
30 F(din,1) = F_in;
31 F(dout,2) = -F_in;
32 end

```

E.22 FTlbrt_xx

```

1 function [din,dout,fixeddofs,F,namesitu] = FTlbrt_xx(nelx,nely,
    F_in,bs)
2 % Force Transmitter left bottom x-direction to right top neg x-
    direction
3 % din and dout are the input and output DOFs, respectively.
4 % fixeddofs contains all constrained DOFs.
5 % F is the vector of applied forces on certain DOFs.
6 % namesitu is the name of the situation, to be used in the
    naming of the
7 % resulting data.
8
9 % In this situation, force is transmitted from the bottom left
    corner of
10 % the internal design domain to the top right corner of the
    internal
11 % design domain. The other two corners of the internal design
    domain are
12 % fixed joints, and a free design domain is allowed around this.
13
14 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
15     error('Error 1. Boundary design domain wrongly defined. Check
        input situation.');
```

```

16 elseif round(0.1*nely) ~= 0.1*nely
17     error('Error 2. Design domain too small for boundary
        conditions to be implemented correctly. Nelx and Nely have
        to be a factor of 10.');
```

```

18 end
19 namesitu = 'FTlbrt_xx';
20 din = 2*bs*nelx*(nely+1)+2*(1-bs)*nely+1;
21 dout= 2*(1-bs)*nelx*(nely+1)+2*bs*nely+1;
22 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(1-bs)*nelx
    *(nely+1)+2*(1-bs)*nely+[1,2]);
23 F = sparse(2*(nely+1)*(nelx+1),2);
24 F(din,1) = F_in;
25 F(dout,2) = -F_in;
26 end

```

E.23 FTlbrt_xy

```
1 function [din,dout,fixeddofs,F,namesitu] = FTlbrt_xy(nelx,nely,  
2 F_in,bs)  
3 % Force Transmitter left bottom x-direction to right top y-  
4 % direction  
5 % din and dout are the input and output DOFs, respectively.  
6 % fixeddofs contains all constrained DOFs.  
7 % F is the vector of applied forces on certain DOFs.  
8 % namesitu is the name of the situation, to be used in the  
9 % naming of the  
10 % resulting data.  
11 %  
12 % In this situation, force is transmitted from the bottom left  
13 % corner of  
14 % the internal design domain to the top right corner of the  
15 % internal  
16 % design domain. The other two corners of the internal design  
17 % domain are  
18 % fixed joints, and a free design domain is allowed around this.  
19  
20 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely  
21     error('Error 1. Boundary design domain wrongly defined. Check  
22     input situation.');
```

```
23 elseif round(0.1*nely) ~= 0.1*nely  
24     error('Error 2. Design domain too small for boundary  
25     conditions to be implemented correctly. Nelx and Nely have  
26     to be a factor of 10.');
```

```
27 end  
28 namesitu = 'FTlbrt_xy';  
29 din = 2*bs*nelx*(nely+1)+2*(1-bs)*nely+1;  
30 dout= 2*(1-bs)*nelx*(nely+1)+2*bs*nely+2;  
31 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(1-bs)*nelx  
32     *(nely+1)+2*(1-bs)*nely+[1,2]);  
33 F = sparse(2*(nely+1)*(nelx+1),2);  
34 F(din,1) = F_in;  
35 F(dout,2) = -F_in;  
36 end
```

E.24 FTlbrt_yy

```
1 function [din,dout,fixeddofs,F,namesitu] = FTlbrt_yy(nelx,nely,  
2 F_in,bs)  
3 % Force Transmitter left bottom y-direction to right top y-  
4 % direction  
5 % din and dout are the input and output DOFs, respectively.  
6 % fixeddofs contains all constrained DOFs.  
7 % F is the vector of applied forces on certain DOFs.
```

```

6 % namesitu is the name of the situation, to be used in the
  naming of the
7 % resulting data.
8
9 % In this situation, force is transmitted from the bottom left
  corner of
10 % the internal design domain to the top right corner of the
  internal
11 % design domain. The other two corners of the internal design
  domain are
12 % fixed joints, and a free design domain is allowed around this.
13
14 if round(bs*nelx) ~= bs*nelx || round(bs*nely) ~= bs*nely
15     error('Error 1. Boundary design domain wrongly defined. Check
  input situation.');
```

```

16 elseif round(0.1*nely) ~= 0.1*nely
17     error('Error 2. Design domain too small for boundary
  conditions to be implemented correctly. Nelx and Nely have
  to be a factor of 10.');
```

```

18 end
19 namesitu = 'FTlbrt_yy';
20 din = 2*bs*nelx*(nely+1)+2*(1-bs)*nely+2;
21 dout= 2*(1-bs)*nelx*(nely+1)+2*bs*nely+2;
22 fixeddofs = union(2*bs*nelx*(nely+1)+2*bs*nely+[1,2],2*(1-bs)*nelx
  *(nely+1)+2*(1-bs)*nely+[1,2]);
23 F = sparse(2*(nely+1)*(nelx+1),2);
24 F(din,1) = F_in;
25 F(dout,2) = -F_in;
26 end

```