Abstract

In this work a large scale topology optimization tool in the Open Source framework KRATOS is presented. Topology Optimization Application was developed in 2015 and is written in the programming languages C++ and Python. The application was not active in the last years. Within this thesis the old version of the topology optimization code is reactivated and updated. To achieve this the application is linked to the Structural Mechanics Application in KRATOS and the current interface of Python and C++ is implemented. Based on this, the optimization tool is improved and extended with new implementations. The new implementations include existing topology optimization methods, such as density filtering and the optimization algorithm Method of Moving Asymptotes (MMA). The extensions and changes within the code are verified with a benchmark code in Matlab. This is done by using a MMB beam as a benchmark example. Moreover, the optimization tool in KRATOS is applied to a large scale problem using an aerial lift pylon as an example.

Keywords: Topology optimization, Open Source, MMA, Kratos, Python, C++

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations .

E_0	Young's modulus of the base material
$C(\underline{x})$	Objective function
$C(x_e)$	Compliance of the element
$\frac{dC}{d\underline{x}}$	Objective function sensitivity
$\frac{d\underline{C}}{d\underline{x}}$	Vector of objective function sensitivities
$\underline{g}$	Constraint function value
$\frac{d\underline{g}}{d\underline{x}}$	Vector of constraint function value sensitivities
η	Damping parameter
x_e	Design variable
$\underline{u}$	Global displacement vector
$x_{e,\text{phys}}$	Density of design variable
$\underline{u_e}$	Element displacement vector
E_e	Young's modulus of design varaible
$\frac{d\hat{C}}{dx_e}$	Filtered objective function sensitivity
$\hat{x}_e$	Filtered design variable
r_{min}	Filtering radius
$\underline{F}$	Force vector
$\underline{\underline{K}}$	Global stiffness matrix
q	Grey scale exponent
λ	Lagrangian multiplier
L	Lower asymptote
$E_{\min}$	Minimum Young's modulus
m	Move limit
N	Number of elements
j	Number of iterations
N_e	Number of surrounding elements
n	Number of equality constraints
a	Number of inequality constraints
B	Optimality condition
p	Penalty factor

x_0	Density of base material
$\underline{\underline{K_e}}$	Element stiffness matrix
$\underline{\underline{K_{e,0}}}$	Element stiffness matrix with Young's modulus = 1
U	Upper asymptote
$\underline{x}$	Vector with design variables
$\frac{dg}{dx_e}$	Constraint function sensitivity
v_{frac}	Volume fraction limit
g_{comp}	Volume fration of the model within the iteration
H_{ei}	Weighting factor
$x^{\max}$	Maximum allowable value of the design variable within MMA Algorithm
$x_{\max}$	Maximum allowable value of the design variable
$x^{\min}$	Minimum allowable value of the design variable within MMA Algorithm
$x_{\min}$	Minimum allowable value of the design variable

1 Introduction

1.1 Motivation

Lightweight structures and optimized designs are important issues in mechanical engineering. In general an optimization can improve the performance of a structure as well as reduce the resources needed to construct it. By optimizing a structure, the weight can be reduced and less weight results in lower forces working onto the structure. With the lower forces the structure can be optimized again. Within this “virtuous circle of lightweight engineering design” one of the steps to optimize the structure can be the topology optimization.

Topology optimization is an advanced structural design method, where by searching the optimal distributions of the material inside the defined design domain, the stiffness of the structure can be improved. This can be achieved by gradient based optimization algorithms. Such algorithms are often used in the field of structural mechanics (Baier et al. 1994).

Topology optimization is one of the research fields at the Institute of Aircraft Design at the Technical University Munich, especially within the project “MILAN - Morphing Wings for Sailplanes”. As a part of this project a new optimization code was developed for linear and non-linear finite element method as well as stress constraints (Reinisch 2017, 2019).

Following the example of REINISCH, a new version of Topology Optimization Application will be introduced. This is done as a part of this thesis at the Institute of Aircraft Design at TUM in collaboration with the Free University of Bozen-Bolzano. Topology Optimization Application is implemented in the open source framework KRATOS and was developed by GONZALES and MALFAVON (Gonzalez 2015, Malfavon 2016). Through the reanimation and extension of the Topology Optimization Application in KRATOS, a new tool for future optimization problems in projects like MILAN or in topics of further research is available.

Within the scope of this work the existing code was reactivated, old functions were improved and new implementations were applied. One of which is an additional optimization algorithm: the Method of Moving Asymptotes (MMA). Within this thesis the new implementations are presented and verified. To do this, the Matlab topology optimization code for 3D structures, called `top3d`, is taken as a benchmark code (Liu and Tovar 2014). To achieve a comparability between the codes, a benchmark example is introduced by using

a MBB beam.

To present a practical example, a structure from the industry is optimized. An aerial lift pylon from LEITNER is modeled and simulated, to show a possible application of the code on a structure in the industry environment.

The first chapter of the thesis introduces topology optimization and the mathematical background. Hereby included are the different filtering methods and optimization algorithms used by the application. This is followed by a presentation of the framework KRATOS and the existing Topology Optimization Application. Based on this, the new implementation is presented. Using the benchmark example, the new functions are compared with either the benchmark code or the existing Topology Optimization Application. With the example of the aerial lift pylon, the use of the code on a large structure is presented. The thesis is concluded with a short summary of the work and possible extensions as well as improvements of the application.

The final version of the Topology Optimization Application as a result of this thesis can be found in the following repository: `https://github.com/PhiHo-eng/Kratos`.

2 Fundamentals of Topology Optimization

Topology and shape optimization of existing problems and structures are topics with an increasing importance within the engineering community (Bletzinger 2020). Advancing technology and breakthroughs like 3D printing make applications as topology optimization or shape optimization more important. Structures which seem to be the best solution in either amount of material, occurring forces or even just design appearance, do not seem so hard to construct and build as they did a few years ago. Saving material within a given volume and application is also a major advantage of these optimization applications. Existing structures can be modified and improved, without losing the original functionality, given that the forces and surrounding conditions are implemented in the simulation accordingly.

This seems to be only possible for companies with a budget which is big enough to finance such advanced programs. The thought behind this project is to create a topology optimization code as part of the KRATOS MULTIPHYSICS open-software Large-scale computation software. In order to provide smaller companies or research facilities an opportunity to optimize their structures or at least improve them and save material and therefore costs in the construction.

In order to find an optimum, there are different approaches to get an optimized structure. These different approaches can be divided in two categories. The methods are based on either gradient based calculations or on non-gradient based calculations. Within this thesis topology optimization with gradient based algorithms is used. The goal of topology optimization is to find the right vector $\underline{x}$ with the densities so that the constraints are satisfied.

2.1 Introduction to Topology Optimization

To find the optimum material distribution in a given volume, topology optimization can be used. The optimization is constrained by the percentage by which the design domain is covered. The boundary conditions define where the structure is kept into place and where the force works onto the structure.

In figure 2.1 the design domain is shown with the given boundary conditions. The volume to be optimized is defined within Γ. This domain contains the space Ω and is connected with the surrounding at Γ_d. The domain is being affected from the outside by t at the

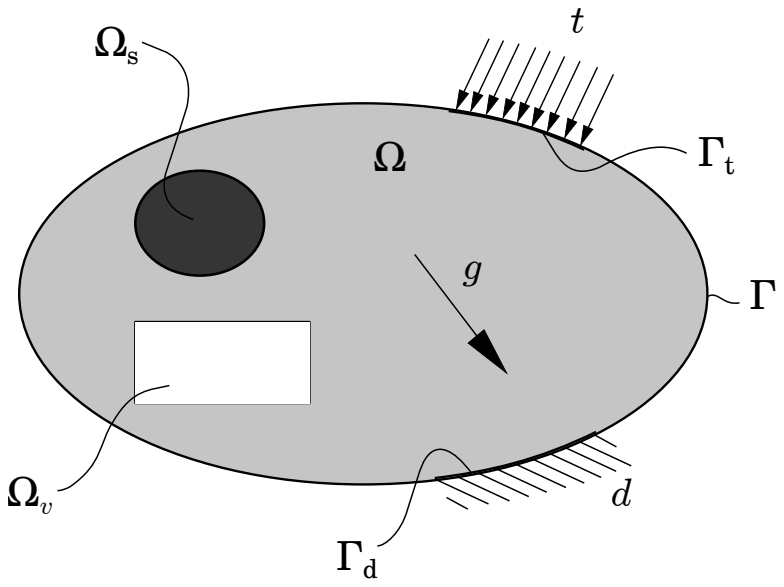


Figure 2.1: Parametrization of the design domain (Bendsoe 2004).

surface Γ_t and by the body forces g within itself. Important to know is that within the space Ω there can be spaces that are defined to be solid Ω_s or to be a void space Ω_v. With these definitions and the topology optimization an optimal material distribution is seeked (Bendsoe 2004).

To solve the given problem the original volume has to be discretized. This means that the volume is divided into smaller elements. The density of these elements is referred to as the design variable. The design variables can be written as a vector:

$$\underline{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}^T \tag{2.1}$$

By using the optimization, the design variables are changed in order to get the optimal material distribution, while remaining within the domain and the given constraints.

To achive the optimization, the minimum compliance approach (see section 2.3) is used. In this formulation the structure with the highest stiffness is seeked by varying the design variables.

The optimization process can be seen in figure 2.2. To begin with, a structural analysis is made with the discretized design volume. For this a finite element method (FEM) approach is used, which is the most common method for structural analysis (Wall 2020).

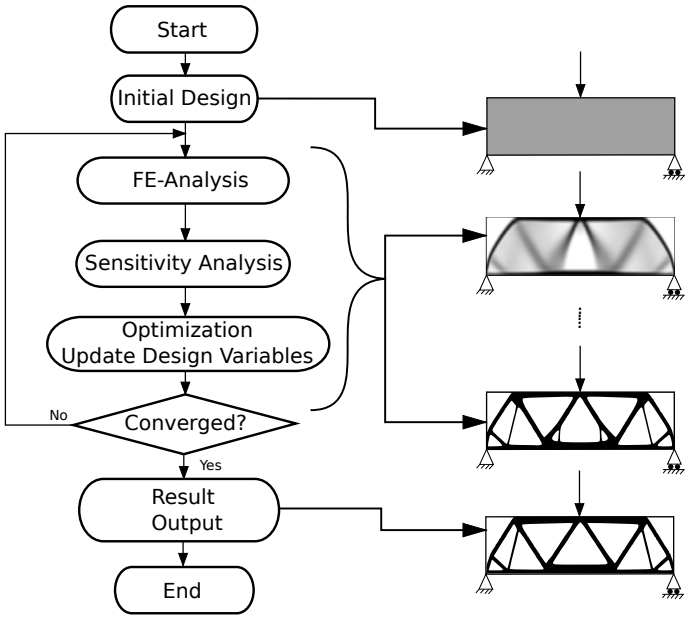


Figure 2.2: General topology optimization process flow (Bendsoe 2004).

With the results of the Finite Element Analysis (FEA), the compliance and the sensitivities are calculated, followed by the filtering of these sensitivities. The gained information is used in the next step to update the design variable vector $\underline{x}$ with an optimization algorithm (see section 2.7). In the final step the convergence criteria is checked. Should the optimization have reached the convergence criteria the process is stopped, otherwise the next iteration is initiated.

To avoid a trivial solution, where the beam is fully filled with material, the amount of volume covering the design domain has to be constrained. This is done by the volume fraction constraint, which is common for a topology optimization problem.

With these information it is possible to write the basic formula for the optimization problem:

$$\begin{aligned} & \min_{x \in X} \{C(\underline{x})\} \\ \text{subject to:} \quad & g_j(\underline{x}) \leq 0 \qquad j = 1, ..., a \\ & h_k(\underline{x}) = 0 \qquad k = 1, ..., n \end{aligned} \tag{2.2}$$

where the N design variables of $\underline{x}$ are varied in the range of X in order to minimize the objective function C. For the optimization to work, the inequality constraint g_j or the equality

constraint h_k have to be fulfilled. The given functions are connected with the so-called system equations, which define the physical behavior of the problem at hand. With topology optimization these are being solved with FEM. Therefore the behavior of the element has an impact on the stiffness matrices of the model. In this case solid elements use the given material properties, whereas the void elements use very low material property values. To do this, the design variables and the element stiffness matrices have to be linked mathematically.

With the material distribution being a discrete problem (either solid or void elements), also the design variables are discrete. Therefore the computational effort to solve it is very high (Zhu et al. 2016). To avoid this, discrete values of the design variable are replaced with continuous values (Bendsoe 2004). As a result, the mathematical relationship between design variables and element stiffness matrices is continuous too. This modification is called the relaxation of the design variables (Sigmund and Petersson 1998).

By using so-called interpolation methods this mathematical connection can be described. In this work the most frequently applied interpolation method is used: the SIMP approach. Within the following section this approach will be further discussed.

2.2 SIMP (Solid Isotropic Material Penalization)

Topology optimization, as it is shown in this work, uses the normalized densities x_e of the element as design variables. x_e is defined as:

$$x_e = \frac{x_{e,\text{phys}}}{x_0} \quad \text{with} \quad 0 < x_e < 1 \tag{2.3}$$

where $x_{e,\text{phys}}$ is the density of the given element and x_0 the density of the given material. By this definition the solid element is achieved with $x_e = 1$ and the void element with $x_e = 0$. To connect the design variables to the element stiffness matrices, the SIMP ("Solid Isotropic Material with Penalization" (Zhou and Rozvany 1991)) approach is introduced.

By changing the Young's modulus, the properties of the element can be altered and with the SIMP approach the properties are linked to the design variables. For solid elements the Young's modulus E_0 of the base material is used. For void elements on the other hand a very low Young's $E_{\min}$ modulus is applied. It has to be avoided to set the $E_{\min}$ to zero, otherwise singularities in the stiffness matrices could occur. Therefore following

conditions have to be applied:

$$\begin{aligned} E_e &= E_{\min} \quad \text{for} \quad x_e = 0 \quad , 0 < E_{\min} \leq E_e \leq E_0 \\ E_e &= E_0 \quad \text{for} \quad x_e = 1 \end{aligned} \tag{2.4}$$

Although the properties can be modified by the SIMP approach so that semi-dense elements can occur, this is not possible in reality. To avoid these semi-dense elements, the so-called grey elements, a penalty term p is introduced. With this modification the relation can be written as:

$$E_e\left(x_e\right) = E_0 \cdot x_e^p \quad \text{with} \quad x_{\min} < x_e < 1 \quad . \tag{2.5}$$

In order for the condition $E\left(x_{e,\min}\right) = E_{\min} > 0$ to be true, a lower boundary for the design variable is introduced: $x_{\min}$.

Building on this, the modified SIMP approach can be arranged. In this case the constraint of the minimum density is not needed. Only the minimum Young's modulus is integrated in the equation:

$$E_e\left(x_e\right) = E_{min} + \left(E_0 - E_{min}\right) x_e^p \quad \text{with} \quad 0 < x_e < 1 \tag{2.6}$$

where x_e has to be within the boundary values 0 and 1. With this formulation the application is well suited for filtering methods. Additionally, the penalty term does not affect the void properties (Sigmund 2007).

The penalty term embraces the design variables to develop towards 0 and 1 solutions in order to avoid grey elements. For two dimensional problems the penalty factor is mostly chosen as 2, with 3-dimensional problems commonly 3 (Bendsoe 2004).

Figure 2.3 shows the resulting modification of the SIMP approach from equation 2.6. With higher values for the penalty term p, elements with intermediate densities are punished. In these cases the value of the Young's modulus E_e decreases drastically towards $E_{\min}$ with densities below 1. With this modification grey elements are avoided and a black white solution is encouraged.

In the FEA every element stiffness matrix is arranged with the Young's modulus $E = 1$.

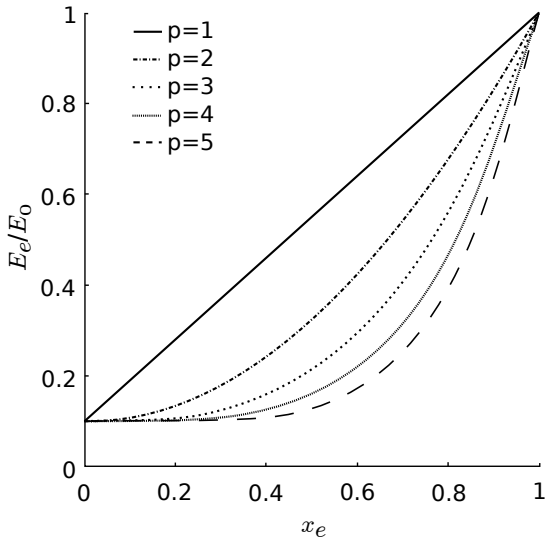


Figure 2.3: Penalization of the semi-dens elements within the SIMP approach (Reinisch 2019).

The result is $\underline{\underline{K_{e,0}}}$, which is in a second step modified with the SIMP approach:

$$\underline{\underline{K_e}} = \left(E_{min} + (E_0 - E_{min})\, x_e^p\right) \underline{\underline{K_{e,0}}} \quad . \tag{2.7}$$

By considering all stiffness matrices $\underline{\underline{K_e}}$ of the elements, the global stiffness matrix $\underline{\underline{K}}$ can be assembled.

2.3 Compliance Minimization

To find the structure with the highest stiffness with the given constraint, the minimum compliance formulation is used (Bendsoe 2004). Minimum compliance can also be described as the minimization of the strain energy. Therefore the energy that results from the force upon the structure and the displacement which is caused by it, is minimized. The objective function of the minimization problem is:

$$\min_{x \in X} \{C\,(\underline{x})\} \tag{2.8}$$

from the equation 2.2 and the corresponding constraints. The compliance can be written as:

$$C(\underline{x}) = \underline{F}^T \underline{u} \quad . \tag{2.9}$$

With $\underline{F}$ being the external force onto the structure and $\underline{u}$ the resulting displacement vector. By reforming the equations using the stiffness matrix $\underline{\underline{K}}$ of the structure, the force can also be written as:

$$\underline{F} = \underline{\underline{K}}\underline{u} \quad . \tag{2.10}$$

With this in mind the compliance $C(\underline{x})$ can also be written in the following manner:

$$\begin{aligned} C(\underline{x}) \quad &= \underline{F}^T \underline{u} \\ &= \underline{u}^T \underline{\underline{K}} \underline{u} \end{aligned} \quad . \tag{2.11}$$

The compliance can now be calculated for every single element:

$$C(x_e) = \underline{u_e}^T \underline{\underline{K_e}} \underline{u_e} \tag{2.12}$$

with $\underline{u_e}$ being the displacement vector of the element. Using this information the compliance of the structure can be written as a summation of the compliance of every element:

$$C(\underline{x}) = \sum_{e=1}^{N} \underline{u_e}^T \underline{\underline{K_e}} \underline{u_e} \quad . \tag{2.13}$$

Introducing the modified SIMP approach from equation 2.7, the compliance can be calculated as:

$$C(\underline{x}) = \sum_{e=1}^{N} \left(E_{min} + (E_0 - E_{min})\, x_e^p \right) \underline{u_e}^T \underline{\underline{K_{e,0}}} \underline{u_e} \quad . \tag{2.14}$$

To avoid a trivial solution, where the design domain is filled with solid elements, the constraint has to be applied to the minimization problem. The volume fraction v_{frac} limits the allowed amount of material within the design domain. The constraint function within this work is written as:

$$\sum_{e=1}^{N} x_e \leq v_{\text{frac}} \cdot N \tag{2.15}$$

with N being the number of elements. This means the summation of all element densities has to be smaller than the limited amount of material.

With the gained information the minimization problem can be written as:

$$\begin{array}{ll} \min_{x_e} & \left\{ C(\underline{x}) = \sum_{e=1}^{N} \left(E_{min} + (E_0 - E_{min})\, x_e^p \right) \underline{u_e}^T \underline{\underline{K_{e,0}}}\, \underline{u_e} \right\} \\ \text{subject to} & \sum_{e=1}^{N} x_e \leq v_{\text{frac}} \cdot N \\ & 0 \leq x_e \leq 1 \end{array} \tag{2.16}$$

2.4 Mathematical and Numerical Obstacles

In this section numerical and mathematical problems, that can occur within a topology optimization, are presented.

2.4.1 Mesh Dependency

Due to the fact that the design domain is discretized into a certain number of elements, also the result is dependent on that discretization. This can for one be improved by using smaller elements, but this may lead to a different structure as can be seen in the figure 2.4. Ideally the same structure should appear only with a more accurate structure of the optimization result.

There are different methods to avoid the mesh dependency. Although they all restrict the spacial distribution of the material in some way, the most commonly used methods are the filtering methods, where a filter is applied to either the sensitivities or the design variables within the optimization process.

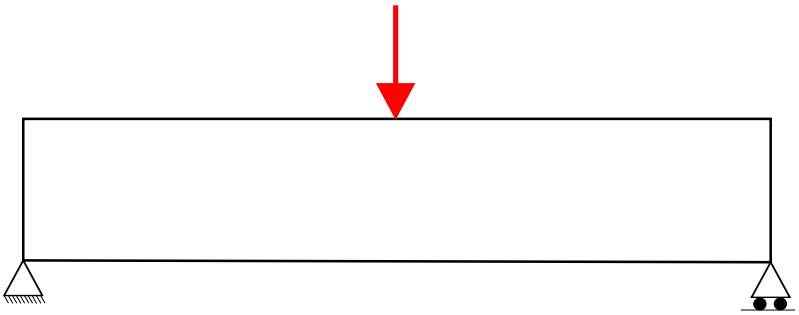
2.4.2 Checkerboard Patterns

The result of a topology optimization may include structure with evenly distributed solid and void elements. This is called the checkerboard problem. Bad numerical modeling by the finite element method, which overestimates the stiffness of the structure, causes the checkerboard effect (Bendsoe 2004). A so-called artificial stiffness is obtained and therefore the structure seems to have a high stiffness. In figure 2.5 a structure with several checkerboard pattern is shown. Their typical checkerboard like distribution of the solid and void elements can be seen.

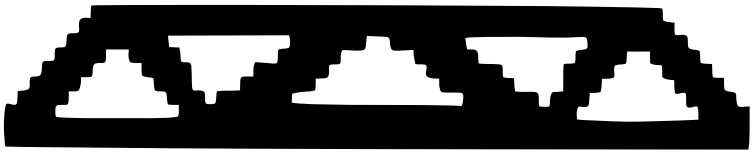
By using the filtering methods, which were mentioned in the previous section, this problem too can be avoided.

2.5 Sensitivity Analysis

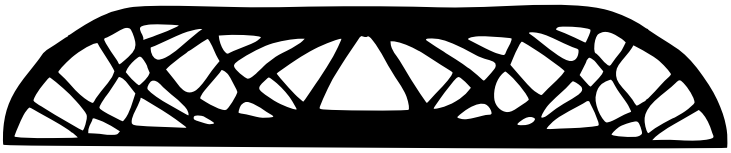
The topology optimization in this work is gradient based. This means that the gradients of the objective function as well as the constraint function are used to improve the structure



(a) Parametrization of the design room.



(b) Solution with discretization of 600 elements



(c) Solution with discretization of 5400 elements

Figure 2.4: Mesh dependency of the of the optimization problems (Sigmund and Petersson 1998).

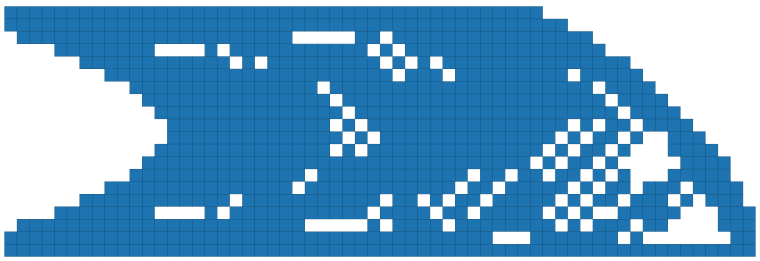


Figure 2.5: Checkerboard effect shown with the Cantilever beam. The optimization was made with OC and $v_{\text{frac}} = 0.5$

in each iteration and eventually reach an optimum. To do this, the sensitivities have to be calculated in every iteration. Since the optimization problem in this work only has one objective function with many design variables, the best way to determine the gradients is by using the adjoint method. The derivative of the objective function can be determined by using the chain rule:

$$\frac{dC}{d\underline{x}} = \frac{\delta C}{\delta \underline{x}} + \frac{\delta C}{\delta \underline{u}} \cdot \frac{\delta \underline{u}}{\delta \underline{x}} \quad . \tag{2.17}$$

In order to get the derivative of displacement $\frac{\delta \underline{u}}{\delta \underline{x}}$ the statement from equation 2.10 is used:

$$\underline{F} = \underline{\underline{K}}\underline{u} \quad . \tag{2.18}$$

The derivative of this equation is:

$$\frac{\delta \underline{F}}{\delta \underline{x}} = \frac{\delta \underline{\underline{K}}}{\delta \underline{x}}\underline{u} + \underline{\underline{K}}\frac{\delta \underline{u}}{\delta \underline{x}} \quad . \tag{2.19}$$

Thus, the derivative of the displacement results in:

$$\frac{\delta \underline{u}}{\delta \underline{x}} = \underline{\underline{K}}^{-1}\left(\frac{\delta \underline{F}}{\delta \underline{x}} - \frac{\delta \underline{\underline{K}}}{\delta \underline{x}}\underline{u}\right) \quad . \tag{2.20}$$

At this point it shall be noted that this is a self-adjoint system, which means the original problem can be solved with a different right-hand side (the so-called pseudo load vector). Normally the equation 2.20 would have to be solved in order to determine $\frac{\delta \underline{u}}{\delta \underline{x}}$. With this result the initial problem $\frac{dC}{d\underline{x}}$ can be solved. In this case a simplification can be made by implementing the equation 2.20 into the first problem, equation 2.17. This results into:

$$\frac{dC}{d\underline{x}} = \frac{\delta C}{\delta \underline{x}} + \frac{\delta C}{\delta \underline{u}}\underline{\underline{K}}^{-1}\left(\frac{\delta \underline{F}}{\delta \underline{x}} - \frac{\delta \underline{\underline{K}}}{\delta \underline{x}}\underline{u}\right) \quad . \tag{2.21}$$

From equation 2.9 it is known that:

$$C(\underline{x}) = \underline{F}^T\underline{u} \quad , \tag{2.22}$$

and additionally it shall be noted that the objective function as well as the force do not necessarily rely on the density of the elements. With this information the equation 2.21

can be reduced to:

$$\frac{dC}{d\underline{x}} = \underline{F}^T \underline{\underline{K}}^{-1} \left(-\frac{\delta \underline{\underline{K}}}{\delta \underline{x}} \underline{u} \right) \quad . \tag{2.23}$$

By using equation 2.10 this can be further simplified to:

$$\frac{dC}{d\underline{x}} = -\underline{u}^T \frac{\delta \underline{\underline{K}}}{\delta \underline{x}} \underline{u} \quad . \tag{2.24}$$

Introducing the elemental modification from equation 2.7 and 2.12 the sensitivity of every element can be obtained by:

$$\frac{dC}{dx_e} = -p \cdot x_e^p \left(E_{min} + (E_0 - E_{min}) \right) \underline{u_e}^T \underline{\underline{K_{e,0}}} \underline{u_e} \quad . \tag{2.25}$$

It can be seen from this equation, that the sensitivities have a negative value. So every additional element decreases the objective function and therefore increases the global stiffness of the system.

It has to be noted that these sensitivities are only local information on the element level. This is a very useful property because it allows a straight forward parallelization (Malfavon 2016).

2.6 Filtering Methods

To avoid the complications in chapter 2.4 filtering methods can be applied. The aim of these methods is to generate a structure without numerical and mathematical problems. This is achieved with mainly two types of filters: For one there is the sensitivity filter (Sigmund 1997, Sigmund 1994) and the density filter (Bruns and Tortorelli 2001). These two filters can be used together with projection methods so that semi-dens areas and the checkerboard patterns can be avoided in the solution (Guest et al. 2004) .

In this chapter three types of filters are presented, even though mainly sensitivity filtering is used in this work.

2.6.1 Sensitivity Filter

The main concept for the sensitivity filter is to evaluate the sensitivities around the given element. With this information the element sensitivity is changed so that elements with high surrounding sensitivities will be rewarded and those who have low gradients around them will be punished. This way it is possible to smooth the spatial density sensitivity distribution and therefore get a more black and white solution at the update of the densities.

This filtering is done by the weighted average of the surrounding gradients. An example of the filtering principle can be seen in the figure 2.6. Depending on the filtering-radius, which can be set by the user, a certain number of neighbours is considered. Mathematically the filtering can be written as:

$$\frac{\hat{dC}}{dx_e} = \left(\frac{\sum_{i \epsilon N_e} H_{ei} x_i \frac{dC}{dx_e}}{\max(\gamma, x_e) \sum_{i \epsilon N_e} H_{ei}} \right) \quad . \tag{2.26}$$

In this equation $\frac{\hat{dC}}{dx_e}$ is the filtered sensitivity. N_e represents the number of elements that are within the filtering-radius. The constant γ is set to a low value in order to avoid a division by zero. x_i is the density of each surrounding element. Commonly a linear decaying function is used for the weighting factor H_{ei} and can be calculated as follows (Sigmund 2007):

$$H_{ei} = \max\left(0, r_{\min} - \text{dist}(i, e)\right) \quad . \tag{2.27}$$

The radius r_{min} is the user defined filtering radius. This radius is subtracted by the value of the distance from the center coordinate of the given element and the neighbour element. This means that the value of the weighting factor H_{ei} decreases linearly with increasing distance between the elements. For elements outside the filtering radius, the weighting factor is equal to zero. In figure 2.6 a visualization of the weighting factor can be seen. One can observe, of how the weighting factor reduces with further distance to the given element.

2.6.2 Density Filter

The filtering of the densities follows the same principle as the sensitivity filter. By using the weighting factor, the surrounding elements are determined. With the information about the density of the neighbour elements the density of the given element can be altered:

$$\hat{x}_e = \left(\frac{\sum_{i \epsilon N_e} H_{ei} x_i}{\sum_{i \epsilon N_e} H_{ei}} \right) \quad . \tag{2.28}$$

The result is an updated design variable, which is affected by the variables of the surrounding elements. It should be noted that by applying the density filter the design variables lose their physical meaning (Sigmund 2007). The densities x_e are replaced by the filtered densities $\hat{x}_e$ and therefore also the modified SIMP approach in equation 2.7 is

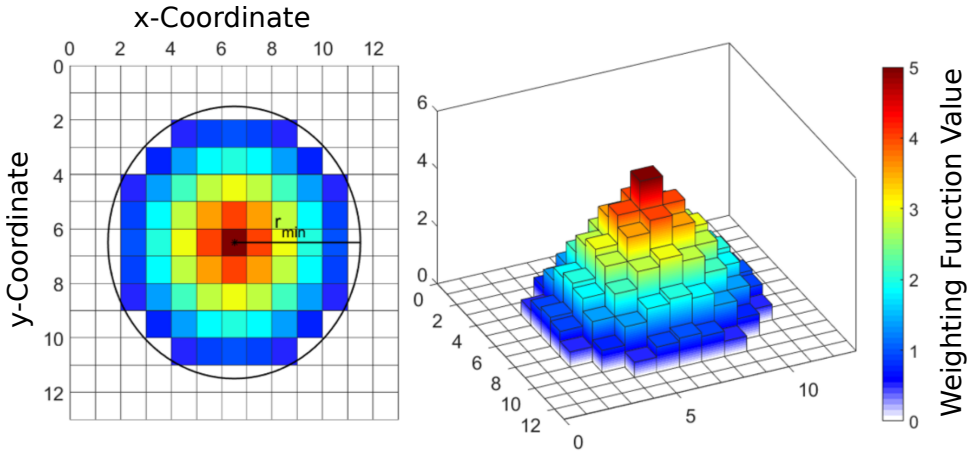


Figure 2.6: Graphical visualization of the weighting factor H_{ei} with the parameters $r_{\min} = 5$, $e = 85$ (Reinisch 2017).

changed with the filtered design variables.

2.6.3 Grey Scale Filter

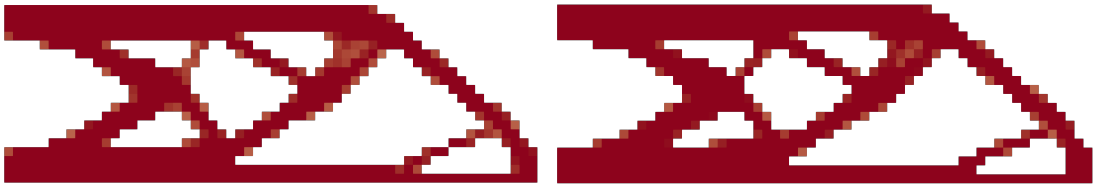
The grey scale filter is used for Optimization Criterion (OC). This non-linear approach by GROENWOLD and ETMAN (Groenwold and Etman 2009), that is implemented in the update procedure of the densities in the OC optimization algorithm, is used to get a discrete black and white solution. The filter varies the exponent q over the iterations j in the update process and can so penalize the semi-dens variables:

$$q = \begin{cases} q = 1 & \text{if} \quad j \leq 15 \\ \min\left(q^{max}, 1.01q\right) & \text{if} \quad j > 15 \end{cases} \tag{2.29}$$

With this optional implementation solid-void elements dominate the outcome of the optimization. Figure 2.7 shows an example, in which the filter was applied in the structure (b). It can be seen, that more elements with the design variable 1 (dark red) are within the structure.

2.7 Optimization Algorithm

Different optimization algorithms can be used in order to update the densities and therefore solve the problem at hand. In this chapter two important optimization algorithms are reviewed. Both of these algorithms are implemented into the code.



(a) Optimization without grey scale filter **(b)** Optimization with grey scale filter

Figure 2.7: Visualization of the grey scale filter. The Cantilever beam optimized without the grey scale filter (a) and with the grey scale filter (b).

2.7.1 Optimization Criterion (OC)

Optimality Criteria (OC) is commonly used for optimization problems with only one constraint (Aage et al. 2015).

As in all topology optimization methods, this algorithm searches iterativly for the optimum design variable value for each element. To do so, the Lagrangian multiplier λ is introduced and linked to the volume fraction of the problem at hand. Based on the criteria in equation 2.30 the design variable is then updated. Mathematically the update procedure can be written as (Bendsoe 1995):

$$x_e = \begin{cases} \max(0, x_e - m) & \text{if} \quad x_e B_e^{\eta} \leq \max(0, x_e - m) \\ \min(0, x_e + m) & \text{if} \quad x_e B_e^{\eta} \leq \min(1, x_e - m) \\ x_e B_e^{\eta} & \text{otherwise} \end{cases} . \tag{2.30}$$

The tuning parameters η and m are set to the values 0.5 and 0.2. The constant m is called move limit and limits the change of the design variable in one iteration. For numerical damping purposes of the optimality condition η is introduced. The optimality condition is given as:

$$B_e = \frac{-\frac{dC}{dx_e}}{\lambda \frac{dg}{dx_e}} , \tag{2.31}$$

where $-\frac{dC}{dx_e}$ is the compliance sensitivity and $\frac{dg}{dx_e}$ the constraint function sensitivity. The Lagrangian multiplier λ is updated every iteration, so that the volume constraint is satisfied. This is done by implementing a bisection algorithm. This means that an upper and a lower bound is set and the difference between the two bounds is divided by 2. This is

done every iteration until the constraint function of equation 2.15 is satisfied (Gonzalez 2015).

2.7.2 Method of Moving Asymptotes (MMA)

The second optimization algorithm is the Method of Moving Asymptotes (MMA). MMA is a so-called dual method, which allows to solve problems with more than just one constraint. In order to use the dual method, the optimization problem has to be separable (Harzheim 2014).

Separability can be achieved by using sequential approximate optimization algorithms. With the implementation of these algorithms the objective function and the constraints can be approximated by separable functions. The MMA approach introduced by SVANBERG is used in this work (Svanberg 1987).

Much like the OC algorithm, also MMA is a gradient-based optimization algorithm. The optimization problem at hand is approximated locally with a convex separable subproblem and subsequently solved with dual methods. The approximation is made based on the sensitivities of the given element i in the iteration j, but also additional parameters like the "moving asymptotes" U_i^j and L_i^j are updated with every iteration, based on information of previous optimizations.

The following implementation of the MMA algorithm is based on the work of SVANBERG (Svanberg 2007). In every iteration j the approximation of the objective function $C(\underline{x})$ can be written as:

$$C\left(\underline{x}\right) \approx C\left(\underline{x}^j\right) + \sum_{i=1}^{n} \left(\frac{r_i^j}{U_i^j - x_i} + \frac{s_i^j}{x_i - L_i^j} \right) \tag{2.32}$$

With x_i as the design variables. r_i and s_i can be determined with the gradient information of the given element as well as the upper and lower asypmtotes U_i^j and L_i^j:

$$r_i^j = \left(U_i^j - x_i^j\right)^2 \left(1.001 \left(\frac{dC}{dx_i}\left(\underline{x}^j\right)\right)^+ + 0.001 \left(\frac{dC}{dx_i}\left(\underline{x}^j\right)\right)^- + \frac{10^{-5}}{x_i^{\max} - x_i^{\min}} \right) \quad , \tag{2.33}$$

$$s_i^j = \left(x_i^j - L_i^j\right)^2 \left(0.001 \left(\frac{dC}{dx_i}\left(\underline{x}^j\right)\right)^+ + 1.001 \left(\frac{dC}{dx_i}\left(\underline{x}^j\right)\right)^- + \frac{10^{-5}}{x_i^{\max} - x_i^{\min}} \right) \quad . \tag{2.34}$$

In this declaration the choice of the first and second value within the brackets is deter-

mined by:

$$\begin{aligned} \left(\frac{dC}{dx_i}\left(\underline{x}^j\right)\right)^+ &= \max\left(\frac{dC}{dx_i}\left(\underline{x}^j\right), 0\right) \\ \left(\frac{dC}{dx_i}\left(\underline{x}^j\right)\right)^- &= \max\left(-\frac{dC}{dx_i}\left(\underline{x}^j\right), 0\right) \end{aligned} \quad . \tag{2.35}$$

The values $x_i^{\max}$ and $x_i^{\min}$ are determined in every iteration by:

$$\begin{aligned} x_i^{\max} &= \min\left(x_{\max}, x_i^j + m\right) \\ x_i^{\min} &= \max\left(x_{\min}, x_i^j - m\right) \end{aligned} \quad . \tag{2.36}$$

With $x_{\max}$ and $x_{\min}$ being the maximum and minimum value of the design variable, which is in the case of the topology optimization 1 and 0. The constant m represents the move limit, limiting the possible change of the design variable.

The values L and U represent the lower and upper asymptotes. These are the limits in which the variable can move in order to optimize the problem. Furthermore, the limits are moved too in order to reach a better convergence behavior. The modification of the asymptotes in the first two iterations, where there is no information about the previous design variables, are:

$$\begin{aligned} L_i^j &= x_i^j - 0.5\left(x_i^{\max} - x_i^{\min}\right) \quad , \\ U_i^j &= x_i^j + 0.5\left(x_i^{\max} - x_i^{\min}\right) \quad . \end{aligned} \tag{2.37}$$

In the iterations with $j > 2$ the asymptotes are varied by the following declaration:

$$\begin{aligned} L_i^j &= x_i^j - \gamma_i^j\left(x_i^{j-1} - L_i^{j-1}\right) \quad , \\ U_i^j &= x_i^j + \gamma_i^j\left(U_i^{j-1} - x_i^{j-1}\right) \quad , \end{aligned} \tag{2.38}$$

where

$$\gamma_i^j \quad \begin{cases} 0.7 & \text{if} \quad \left(x_i^j - x_i^{j-1}\right)\left(x_i^{j-1} - x_i^{j-2}\right) < 0 \\ 1.2 & \text{if} \quad \left(x_i^j - x_i^{j-1}\right)\left(x_i^{j-1} - x_i^{j-2}\right) > 0 \\ 1 & \text{if} \quad \left(x_i^j - x_i^{j-1}\right)\left(x_i^{j-1} - x_i^{j-2}\right) = 0 \end{cases} \quad . \tag{2.39}$$

With this approach the limits are either tightened or widened, depending on how the design variable changed from the previous iteration to the given iteration $x_i^j - x_i^{j-1}$ and the same is done for the iterations before that: $x_i^{j-1} - x_i^{j-2}$. Should the gradients of those two iterations be in the same direction, the asymptotes are widened, should they have

values that have a different sign the asymptotes are narrowed.

With this implementation the convergence can be stabilized if oscillations should occur, otherwise the solving process is accelerated.

3 Open-Source Topology Optimization Software

In the year 2001 SIGMUND published an open-source topology optimization code written in Matlab. The intention behind the project was of educational nature (Sigmund 2001). In order to get an inside and a simple introduction to topology optimization the code `top99`, an optimization tool with 99 lines of code, was published. It already had a few implementations which were discussed in chapter 2. The code consist of a simple compliance minimization with a volume constraint. By solving a simple Cantilever beam the optimal material distribution within the design domain is obtained. The code is still available for educational purposes.

Ten years later in 2011 an updated version of this code was released. The new code was named `top88`, because the application was reduced to 88 lines of code. New was, that a high efficient pre-allocating array function and vectorised loops were implemented (Andreassen et al. 2011). Furthermore, this code had a new filtering method included. The application was also available as a mobile version under the name `TopOpt`. Even though this code is able to calculate the equation using direct solvers and 4-noded elements, the code has several limitations. For instance setting boundary conditions or implementing multiple loads seem to cause significant problems. Since the code was again intended for educational purposes, the `top88` is not able to solve larger and more complex problems.

In 2014 TOVAR and LIU developed a Topology optimization application for 3D structures in Matlab called `top3d` (Liu and Tovar 2014). Also this code was intended for students and newcomers in the field of topology optimization. The 169 line code consists of a finite element analysis, sensitivity analysis, sensitivity and density filtering. The optimization is done by the OC algorithm. Very promising though is the fact that the authors promise an easy implementation of the MMA optimization method. Additionally, multiple loads and a variety of boundary conditions can be simulated. According to the author, the code is about 30 times faster than traditionally used direct solvers at that time.

These three codes are only a few examples of many open-source codes, which were developed over the years (Hunter 2009, Talischi et al. 2012, Aage et al. 2015).

3.1 KRATOS - Open-Source framework

KRATOS is an open-source framework for solving engineering problems with numerical methods. It is of great interest to KRATOS developers and the community to simplify the

development of new numerical methods. The main programming language is C++, which allows modularity and high performance. Various problem-types can be solved using KRATOS: structural analysis, fluid dynamics, thermal problems and more. The framework is structed as follows. The basic tools (database, linear algebra, search structures, etc.) are implemented in the so-called “core”. With the tools different “applications” can be formed, extended or changed. Therefore implementing new problem types and applications to solve them, can be done without further changes in the “core” of the software.

An additional advantage of KRATOS is the use of two different layers: Python and C++. This is realized by using Python as an user interface, where top-level process can be set and settings for the simulation are changed easily. Because Python does not need any compiling in order to work, process settings and parameters are more readable and most importantly the variations of such is less time consuming and less complicated.

Due to the fact that interpreted languages (needs no compiling) like Python are less efficient than compiled languages, the computational task is done by the second layer written in C++. In order to make this work, Python and C++ are linked by a corresponding interface.

KRATOS offers the opportunity to use HPC. Additionally a full parallelization is possible due to the OpenMP and MPI. With the straight forward parallelization, KRATOS is very attractive in the matter of implementing new applications.

3.2 Topology Optimization in KRATOS

In the year 2015 a large scale-open source topology optimization application was introduced by GONZALES in KRATOS (Gonzalez 2015). Due to the advantages mentioned in the previous section, an application for topology optimization seemed to be an interesting project. The TopOpt Application was designed and implemented in KRATOS.
Gonzales published a new code, which had several abilities that were not included in open-source topology optimization before. Some of those where:

- 3D analysis tool using arbitrary hexahedral elements.
- Active and passive elements.
- Introducing non-restricted geometries without dependency on domain, shape or size.

- Continuation strategy to avoid local minima in the early iterations (for more information about this see (Gonzalez 2015)).
- The opportunity to work with multiple load and boundary conditions.
- Implementation of filtering methods like sensitivity filter and grey scale filter.

Even though the application was efficient, minimizing the compliance of the given structure and simultaneously remaining inside the volume constraint, one major problem as within the code. As mentioned in the chapter 3.1, KRATOS consists of the core and the applications. The applications are using the needed tools from the core. The Topology Optimization Application (TopOpt) was depending on the Solid Mechanics Application of KRATOS. The code was structured by using functions and scripts from Solid Mechanics Application.Therefore the next goal was to make the TopOpt more independent from Solid Mechanics Application.

In 2016 the Topology Optimization Application was extended by MALFAVON (Malfavon 2016). The application was implemented as an independent application in KRATOS. TopOpt was still based on Solid Mechanics Application. But elements, utilities and functions were written for the application in order to get independence in KRATOS. The extensions of the code were:

- Implementation of tetrahedral elements.
- Parameters and equation were adapted to the new elements.
- Different strategy for the computation of sensitivities and to get the element information from the FE analysis.
- Utilities and functions written in order to get a more independent application.
- A restart file was introduced. In this case an additional .mdpa file is written by the code in order to get additional information in the process of optimization. The user can choose the number of iterations after which such a file is printed.
- The user interface was changed.

With the new elements the implementation of the SIMP approach had to be changed. This was done by introducing a local residual approach. By doing so, the implementation effort was reduced and the opportunity for parallelization was a result.

3.2.1 Topology Optimization Application

To understand the new implementations of the code made within this thesis, the old code is presented. This is done by introducing the different working layers and function scripts. This shall make it easier to understand the implementations and how they work in the code.

The Topology Optimization Application was developed according to the KRATOS framework. As mentioned before, the code is written in two programming languages: Python and C++. Python is used to provide an easier environment and to make the process of the Application more readable. C++ on the other hand is used to provide a higher performance, low-level memory manipulation and a sufficient flexibility in large design domains.

In order to understand the code and to follow the process flow, a step by step flow chart of the optimization process can be seen in figure 3.1. For the optimization process to start a initial design is provided. With this model a FEA is performed. Building on that the objective function and the volume fraction of the current model are determined for every element. This is followed by the calculation of the gradients for both design variable and constraint function. The resulting sensitivities are filtered by the sensitivity filter. Based on the gained information, the optimization algorithm is launched and the design variable is updated. The process is repeated until the objective function change in the iteration is below a certain value, which is set by the user. In the given figure the density filter is implemented too, as it shows the optimization process of the new Topology Optimization Application.

To initiate the optimization process the Python script `run_TopOpt.py` is called. This script acts as the main part of the simulation. The model is called by using a `.mdpa` file, the conditions of the model are saved in a `.json` file and are applied by calling the file. Additionally a Python-file with the optimization parameters is part of the process. In order to start a topology optimization following scripts are needed:

- `run_TopOpt.py`
- `OptimizationParameters.py`
- `Model.mdpa`

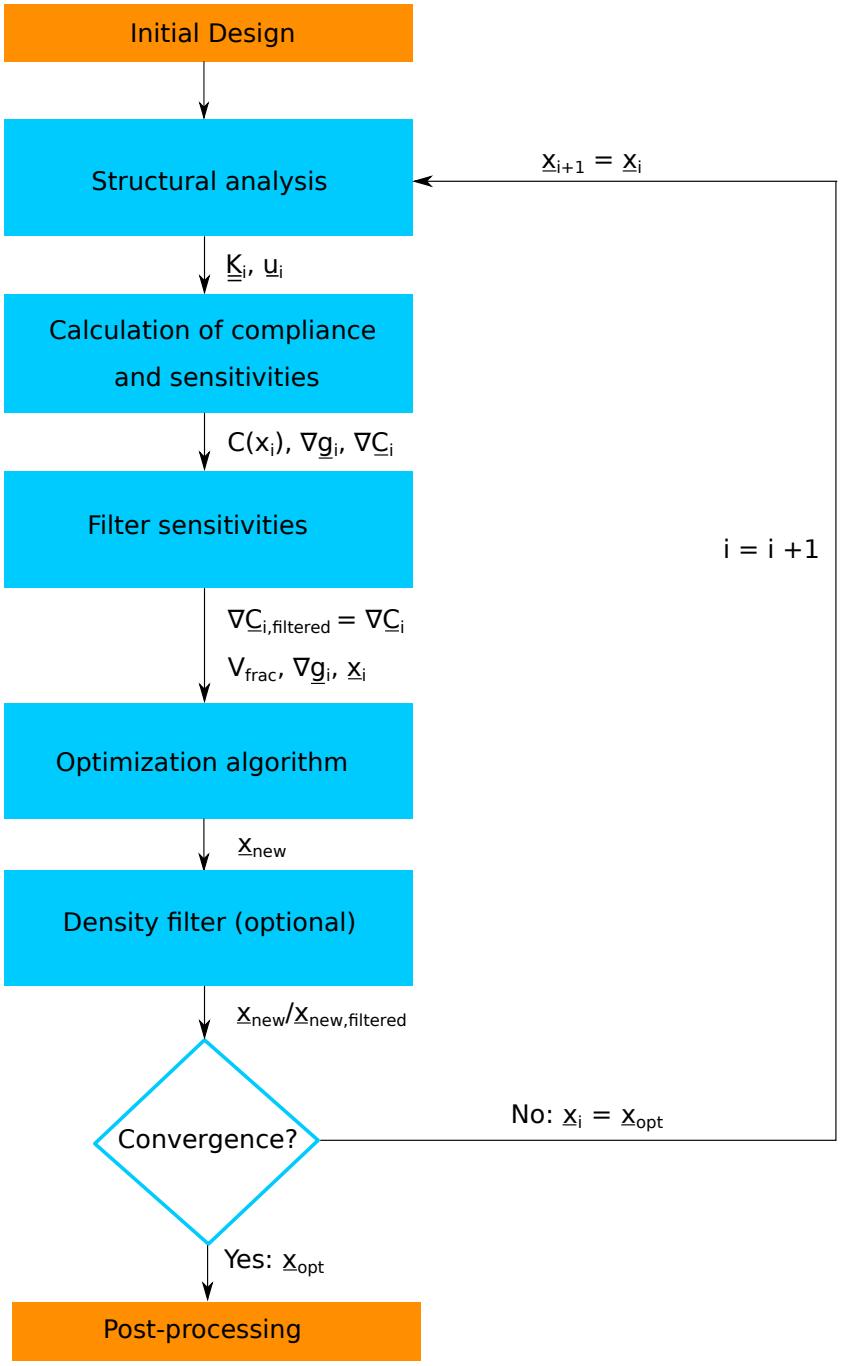


Figure 3.1: Flowchart of the optimization process.

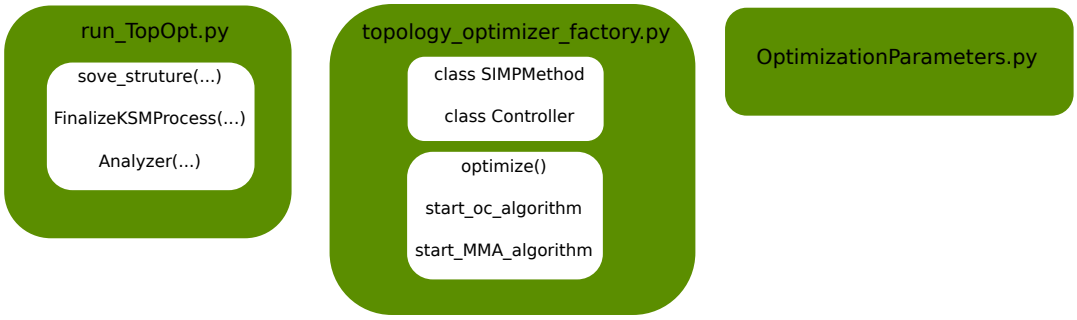


Figure 3.2: Python scripts and their main functions.

- `ProjectParameters.json`

In the figure 3.2 the needed Python scripts in order to run the simulation are shown. For the user only two of those are of importance: The `run_TopOpt.py` file, which calls the FE analysis and initializes the optimization procedure in every iteration. Also the optimizer is constructed in this file and the `topology_optimization_factory.py` is called by the code. The second user defined file, the `OptimizationParameters.py`, is included in the optimization by the `topology_optimization_factory.py`. By calling the constructed optimizer the optimization process in launched. This is followed by an adjusted FEA of the structure. The following steps are performed:

- Structural problem is solved to get the displacement vector $\underline{u}$.
- Objective function values are determined and saved, based on the displacement $\underline{u}$.
- The constraint value is calculated.
- The sensitivities of the objective function are called.
- The sensitivities of the constraint function are called.

The optimization runs through these steps in every iteration, and saves the sensitivities and objective function values inside the element.

`OptimizationParameters.py` is the script where the user can adjust the parameters for the topology optimization. The script contains: the properties of the design variables, the filtering options, which optimization algorithms is used and the convergence criteria as well as the maximum number of iterations. Additionally the response functions are defined by assigning objective function and constraint function. Furthermore the post-

processing settings are implemented in this Python script, which consist of the writing of the restart file as well as the output-configurations.

Within the `.mdpa` file the elements are defined. The type of elements used for the simulation and the coordinates where each element finds its place. Additionally the volumes, surfaces or points for the boundary conditions and loads are defined. The `ProjectParameters.json` on the other hand has the information about the boundary conditions and loads itself. Also the settings for the FE solver are saved within the `.json` file.

In order to understand the optimization process and the function within it, the C++ layer has to be observed. The code, while running through the Python code in `topology_optimization_factory.py`, calls the different functions which are defined as C++ scripts in the application. The most important scripts for the optimization process are:

- `residualbased_incrementalupdate_static_simp_scheme.h`:
  The script updates the global stiffness matrix of the model in order to calculate the new displacement vector $\underline{u}$ in the FEA.
- `sturctural_response_function_utilities.h`:
  This script calls every element to get its objective variable and the constraint variable. These get summed up in order to get the objective function and the volume fraction.
- `structure_adjoint_sensitivity_strategy.h`:
  This script calls every element and launches the calculation of the sensitivities for the objective variable as well as the constraint function.
- `small_displacement_simp_element.cpp`
  This script contains the actual calculation of the sensitivities, objective function and constraint function.
- `topology_filtering_utilities.h`
  Within this script the sensitivities are filtered.
- `topology_updating_utilities.h`
  To execute the update of the design variable this script is used.

The figure 3.3 shows the different scripts with the functions they contain (although the

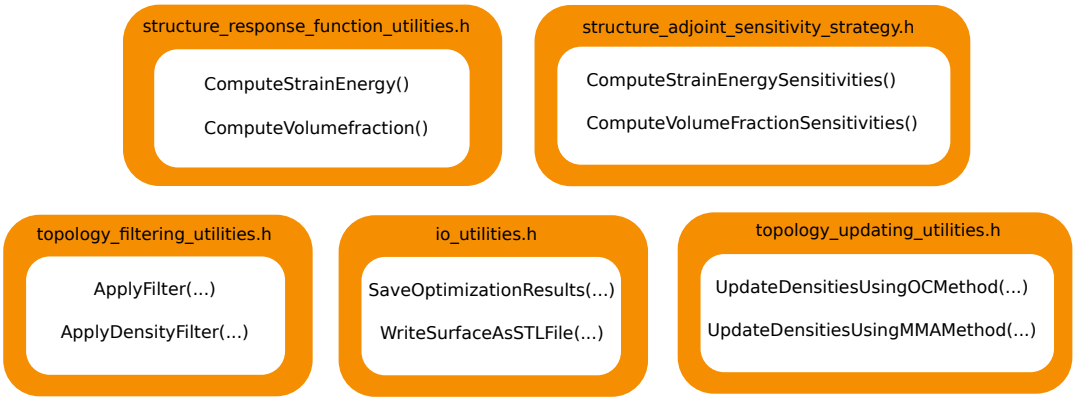


Figure 3.3: C++ scripts with their implemented functions.

new functions are already implemented).

With the information in the `ProjectParameters.json` and `OptimizationParameters.py` different output files are constructed, when a simulation was done successfully. Within this work the resulting files were written for the pre and post processing tool GiD. In these files, the distribution of the design variable, the displacement and the stress reaction within the structure are saved and can be observed in the post processing tool.

In addition, a post-processing tool is available in Topology Optimization Application. With this Python code, the design variables below a certain threshold (defined by the user) are deleted and a structure with the remaining elements is obtained. By reading the restart file, which is printed in the end of the simulation, and the information about the model within it, the Python file `run_TopOpt_PostProcessing.py` can be launched in the terminal. The resulting structure can be observed in the post processing tool GiD.

3.3 Introduction to the new Topology Optimization Application in KRATOS

The original code was stored in the KRATOS Legacy and therefore not up to date with the current version of KRATOS. To get the code even running new implementations were added and old ones had to be cut out. To do this, the KRATOS version of October 2020 was used to develop the updated version of the optimization tool.

The code from GONZALES and MALFAVON was based on the Solid Mechanics Application. The new version of the code would be based on the Structural Mechanics Application. To do so, all dependencies of the the Solid Mechanics Application were removed

and replaced by the equivalent functions from Structural Mechanics Application.

Due to the fact that the applications in KRATOS use a similar structure and similar elements only small parts had to be changed. Especially the includes of the scripts and functions needed to be updated to the equivalent ones of Structural Mechanics. One issue was to convert the used elements into elements readable for Structural Elements. Thus, the base of the used elements had to be changed. Up to this point the element base `small_displacement_element` from Solid Mechanics was used, now the elements constructed in the simulation and used by the Topology Optimization Application are based on the element type `small_displacement` from Structural Mechanics Application.

Additionally, the interface between Python and C++ had been changed since the code was last used. The original version used Boost as an interface between the two programming languages. But for C++ and Python to interact Pybind11 had to be implemented into the different scripts.

With these changes the code was able to be compiled and the opportunity was given to start improving the code as well as implementing new functions.

3.3.1 Dependency of the Filtering Process from the Young's modulus

In order to apply filtering methods, the elements surrounding the given element are considered. By using the filtering radius $r_{\min}$ the number of elements can be changed by the user. The elements inside the radius are called and their sensitivities or densities are determined. Additionally the distance from the given element to the neighbour element brings a weighting factor into the filtering (see equation 2.27 and figure 2.6). In other codes like `top99`, `top88` or `top3d` the filtering is done before the optimization process starts (Sigmund 2001, Andreassen et al. 2011, Liu and Tovar 2014). Due to the interaction of Python and C++ this is implemented differently inside the Topology Optimization Application: The filtering method is implemented directly inside the `topology_filtering_utilities.h`. In order to execute the filter function, every element is called inside the script and their neighbours are checked for their densities or sensitivities.

Although this solution seems to be very good, by running the process completely inside C++, which provides a faster computation of the task, there is a problem: The deformed model from the FEA was used for the filtering. This means that the elements are not in

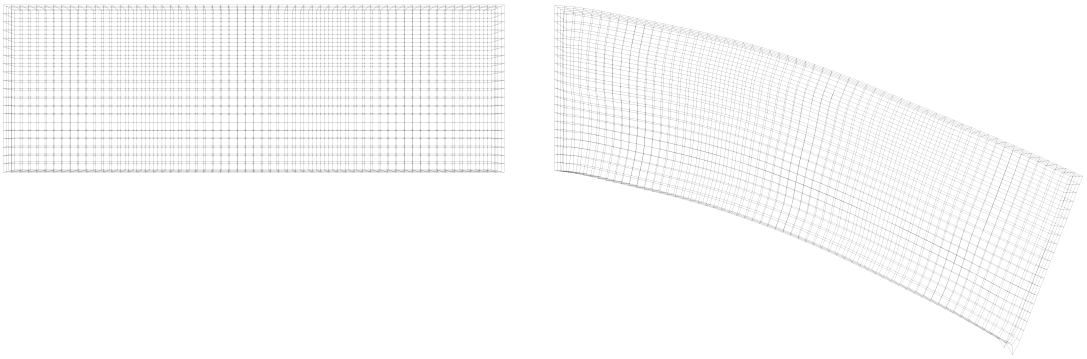


Figure 3.4: Comparing the undeformed and deformed model.

their original position. Therefore the filtering process is depending on the Young's modulus of the structure. While a large Young's modulus does not affect the filtering process, because the displacement of the structure is relatively small, a small Young's Modulus has a major impact on the the filter function. Due to the high displacement of the structure, the filtering radius $r_{\min}$ includes less elements. This means that less elements are considered and the resulting structure changes. Figure 3.4 is a representative visualization to show the difference between deformed and undeformed structure. It can bee seen, that with the use of the structure on the left hand side, the filter process is more accurate, especially when the displacement increases even further, which it does for models with a low Young's modulus.

This problem was solved by changing the center coordinates of the elements. The displacement of every element was added to their center coordinates, in order to get a undeformed model. This means the filter is applied to the non-deformed model and the surrounding elements can be considered as it is intended. By saving the center coordinates in a so-called tree and then adding the displacement inside the program, the model itself is not changed and can be used further. The tree is build and deleted in every iteration. This leaves room for improvements, in order to avoid building the tree in every iteration.

3.3.2 Implementation of the Density Filter

The density filter was implemented equivalent to the sensitivity filter. As can be seen in equation 2.28, the weighting function is also needed with the density filter. Therefore the updated filtering process, which was discussed in the previous section, was also included into this function. The addition to the code was included as follows: By calling the

function a tree (as described in section 3.3.1) is created and the filtering radius $r_{\min}$ the surrounding elements are examined. The densities of the neighbour elements are saved within the tree. With the gained information and the weighting factor, the new density of the given element is calculated. The resulting design variable is then saved within the element and is used for the next iteration. This is implemented inside the `topology_filtering_utilities.h` (see figure 3.3) with the function name `ApplyFilterDensity()`.

The use of this filter is optional and it is up to the user to decide whether it is used or not. Is the filter used though, a less branched structure is anticipated. Less branched means in this case that the result is more concrete, that there are less additional branches within the structure. An example of this can be seen in figure 4.6.

The function is launched inside the `topology_optimizer_factory.py` right after the updating of the design variables. This filter can be activated by the user with an additional line in the `OptimizationParameters.py`:

```
density_filter="density"
```

The filter can be activated in both optimization algorithms OC or MMA.

3.3.3 Implementation of MMA as Optimization Method

By introducing the Method of Moving Asymptotes (MMA) into the code, the user can now choose between two optimization algorithms. The MMA optimization algorithm is a well established method to optimize large-scale topology optimization problems (Bendsoe 2004). Due to the fact that OC and MMA require the same types of computations, the algorithm is fairly easy to implement (Bruns and Tortorelli 2001).

For this work a C++ MMA-script, written by Jérémie Dumas, was implemented. The code is available in the repository (`https://github.com/jdumas/mma`), free to use and is based on the work of SVANBERG (Svanberg 2007). It can also be extended to a GCMMA, which is the globally convergent version of MMA. In this work though only MMA was implemented.

The MMA was placed into the code as follows:

The new function UpdateDensitiesUsingMMAMethod() was created within `topology_updating_utilities.h`. Inside the C++ script the optimization is executed. Because the MMA script by Jérémie Dumas was written to run entirely inside a C++ environment, the

functions and the needed inputs had to be extended. In order to get the information of the previous iterations, which would be lost due to the change from C++ to Python within the optimization, several of the parameters are now saved within the element and provided to the function as additional information. With this change the updating of the design variable is calling the extended function:

$$\text{Update}\left(\underline{\mathrm{x}}, \underline{\frac{dC}{d\underline{x}}}, g, \underline{\frac{dg}{d\underline{x}}}, \underline{x}_j^{\min}, \underline{x}_j^{\max}, \underline{x}_{j-1}, \underline{x}_{j-2}, j, \underline{L}_j, \underline{U}_j\right) \quad . \tag{3.1}$$

In here some of the input values need to be in the form of a vector. Therefore vectors are created, by running through the elements, calling their properties and saving them inside the new vectors. The update is with $\underline{x}$ being the vector with the design variables, $\underline{\frac{dC}{d\underline{x}}}$ the vector with the corresponding sensitivities of the objective function.

g is the constraint function value of the given iteration and $\underline{\frac{dg}{d\underline{x}}}$ the sensitivity of the constraint function value. $\underline{\mathrm{x}}_{\min}$ and $\underline{\mathrm{x}}_{\max}$ are the limits for the design variables set in every iteration. Additionally for the new implementation the design variables of the previous iterations are needed: $\underline{x}_{j-1}$ and $\underline{x}_{j-2}$. In order for the optimization algorithm to know the number of iteration which are already done, the current number of iteration j is provided for the update. Finally the aysmptotes of the previous iteration are provided by $\underline{L}_j$ and $\underline{U}_j$.

The design variables and the objective function are saved inside the element and can therefore be called and saved within a vector. The constraint function value is given by:

$$g = g_{comp} - v_{\text{frac}} \quad , \tag{3.2}$$

with v_{frac} being the volume fraction limit set by the user. g_{comp} is the volume fraction of the given iteration and defined by:

$$g_{\text{comp}} = \sum_{i=1}^{n} x_e \quad . \tag{3.3}$$

x_e are the design variables and N the number of elements. As a result the sensitivity of the element of the constraint function value is defined as:

$$\begin{aligned} \frac{dg}{dx_e} &= \frac{d}{dx_e}(x_e) \\ &= 1 \end{aligned} \quad . \tag{3.4}$$

With this the vector $\frac{dg}{dx}$ can be formed.

The upper and lower limit $\underline{x}_j^{\max}$ and $\underline{x}_j^{\min}$ are determined according to the equation 2.36. The number of iterations j is used as an input variable in order for the updating function to properly calculate the asymptotes $\underline{L}_j$ and $\underline{U}_j$ (see equation 2.37 and 2.38).

After the updating is achieved, the new design variables are saved within the element. Additionally the design variables of the previous sections, as well as the asymptotes are saved within the element and can be called in the next iteration.

The introduced optimization algorithm uses the default parameters within the updating procedure, which are suggested by SVANBERG to create and solve the sub-problem (Svanberg 2007).

Whether the OC algorithm or the MMA algorithm is used can be chosen by the user in the `OptimizationParamters.py` by writing:

`optimization_algorithm="MMA_algorithm"` or `"oc_algoritm"`

Although the MMA algorithm was implemented properly, there is still a small problem, which was not solved within this work. The fact that the MMA script is initialized in every iteration could be a source of problem. This issue is further discussed in chapter 4.2.3.

4 Topology Optimization Results

To investigate the new functions inside the code and their implementations, this chapter provides a comparison with the `top3d` code from LIU and TOVAR (Liu and Tovar 2014). Different examples are presented and evaluated in order to determine the diversities in the results among the new functions. Therefore a Cantilever Beam is introduced. The beam can be seen in figure (4.1) and has the dimensions of 60x20x$4mm$. For the following examples the beam is discretized into elements of the form 1x1x$1mm$. This gives a total number of 4800 elements and 6408 nodes. The boundary conditions are as follows: The force is working onto the structure at the points in the front right corner. The beam itself is mounted onto a structure with the latter surface and thus no displacements in this area are allowed.

This beam is also implemented as an example into the Topology Optimization Application in KRATOS. Two versions of this beam are available in the examples: For one example the hexahedral elements are taken, for the second one the elements are implemented as tetrahedral.

4.1 Introduction of the Benchmark Code

In order to compare the Topology Optimization application with an existing code, the Matlab code `top3d` is introduced. The code can be freely downloaded on the website `https://www.top3d.app/`. By comparing the results of the code, the new implementations within Topology Optimization Application are verified.

To optimize a structure in Matlab, the model is defined by the user by simply setting the values of either coordinate axis. Although this provides an easy and fast use of the code, more complicate structures can not be optimized with this application. For more information about the code see the work of LIU and TOVAR (Liu and Tovar 2014). In the figure 4.2 a 3D structure can be seen, which was optimized with `top3d`.

To have a comparability, the model inside the Matlab-code has the same dimensions as the Cantilever beam from the examples in the Topology Optimization Application. The dimensions are: $60x20x4mm$. For both Matlab and Kratos the Young's modulus is set to 1. Additionally the boundary conditions in Matlab equivalent to the conditions of the example described in the introduction of this chapter.

For the simulations the parameters are set according to the values in table 4.1. It can

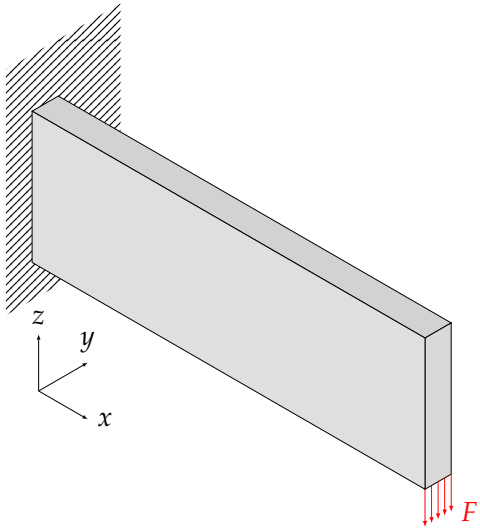


Figure 4.1: Cantilever beam as benchmark model.

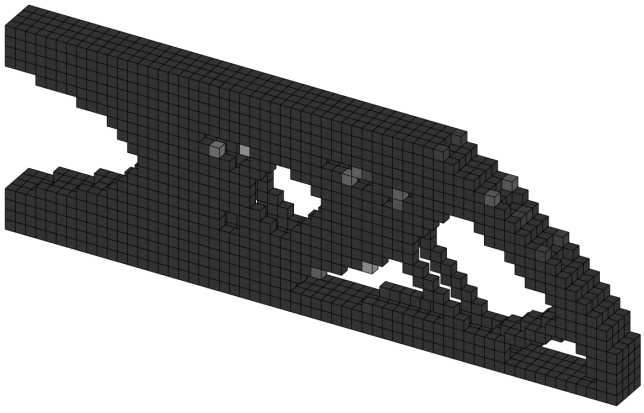


Figure 4.2: Optimization of the Cantilever beam with the `top3d` code in Matlab.

Tolerance: 1e-4 for KRATOS / 0.01 for Matlab	Penalty factor: 3
max. Iterations: 100/200	Filter radius: 1.5
Volume Fraction: 0.3	Grey Scale Filter: No
Young's Modulus: 1	Continuation Strategy: No
Optimization Algorithm: oc_algorithm	Density Filter: No

Table 4.1: Optimization parameters for the example of the Cantilever beam optimized with Topology Optimization Application and with Matlab.

be seen in this table that two different termination criteria are set in the codes. This is because the Matlab code uses the relative change of the design variable over the iterations as termination criterion. If the definition:

$$\max\left(\| \underline{x}_j - \underline{x}_{j-1} \|\right) \quad , \tag{4.1}$$

is below a certain threshold set by the user, the optimization process is stopped. In the Topology Optimization Application the relative change of the objective function is observed in order to determine whether the optimization should be stopped or continued.

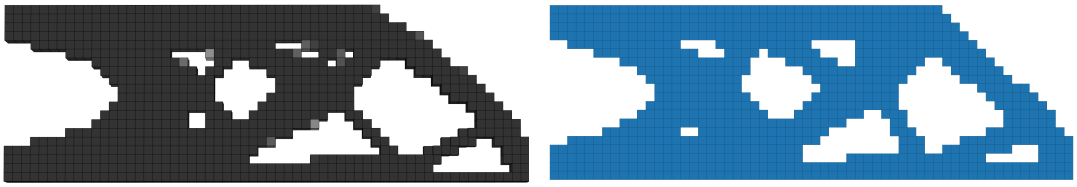
When comparing the results, it has to be considered that the codes are using filtering and therefore the weighting function differently. In Matlab the detection of the neighbour elements and the calculation of the weighting function is done before the optimization process starts, whereas in KRATOS this is done in every iteration step.

Also the maximum iterations in Matlab are recommended higher and therefore set to 200, whereas in Kratos this is set to 100.

4.2 Evaluation of the new Implementations

In this chapter new implemented functions are compared and evaluated. For the examples where the Matlab code is used, the parameters are set according to the values shown in table 4.1. Before the presentation of the results it has to be mentioned that the Matlab simulations were made on a different computer as the ones made with KRATOS. Additionally, it is important to know that the time consumption of the simulations with Topology Optimization Application are relatively high, because these were done with the KRATOS debug version.

Both simulations use the same element size and hexahedral element structure. The result and the compliance change over the iterations as well as the number of iterations



(a) Cantilever beam optimized in Matlab **(b)** Cantilever beam optimized in Kratos

Figure 4.3: Comparison of the optimization of a Cantilever beam using OC in Matlab (a) and Kratos (b).

	Matlab	KRATOS
Number of elements	4800	4800
Number of nodes	6408	6408
Number of iterations	176	23
Time of solution [s]	86	556
Compliance of optimized structure	1069	1144

Table 4.2: Comparison between existing Matlab code top3d and the Topology Optimization Application

can be seen in figure (4.3).

From the figure (4.3) it can be seen that the results from KRATOS and Matlab are very similar. The differences can be explained with the different implementation of the termination criterion. In the table 4.2 and in figure 4.4 can be seen that the Topology Optimization Application only needs 23 iterations for the optimization to stop. Whereas the Matlab code needs 176 iterations. Although with the higher number of iterations, the compliance of the structure optimized in Matlab is lower than the compliance of the structure optimized in KRATOS. When comparing the time, the Matlab code is about 8 times faster than the Kratos optimization. But this is due to the fact, that Kratos is still implemented as the debug version and therefore an optimization is expected to take significantly longer.

4.2.1 Improvement of the Filtering Process

With the implementation of the improved filtering procedure, better results are expected for structures with a low Young's modulus. In the figure 4.5 the result of the simulation with the old, original version of Topology Optimization Application can be seen. This simulation was made with the same parameters as shown in the previous section in table 4.1 and figure 4.3. As discussed the filtering of the deformed structure could lead to checkerboard like patterns. This can be seen very good in the figure. Due to the deformed model, the function can not find any or only a few elements within the filtering

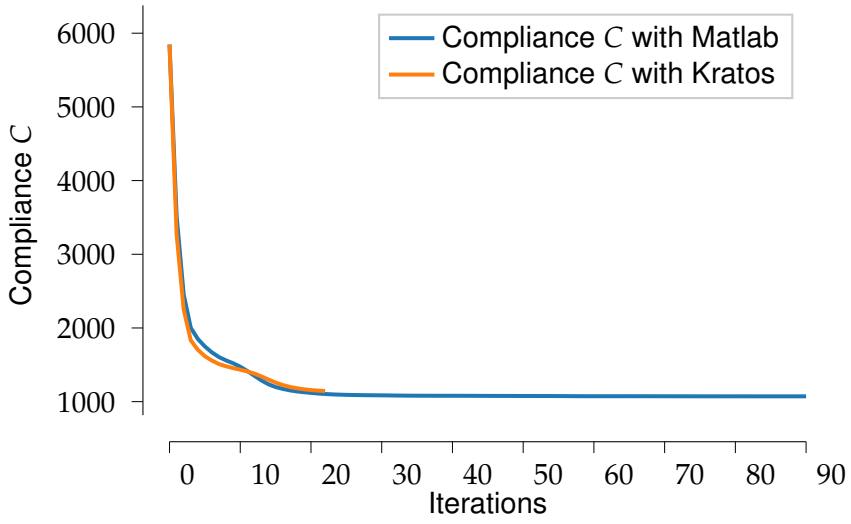


Figure 4.4: The change of the compliance over the iterations. The plot in blue shows the change of the simulation in Matlab, the orange one the optimization in KRATOS.

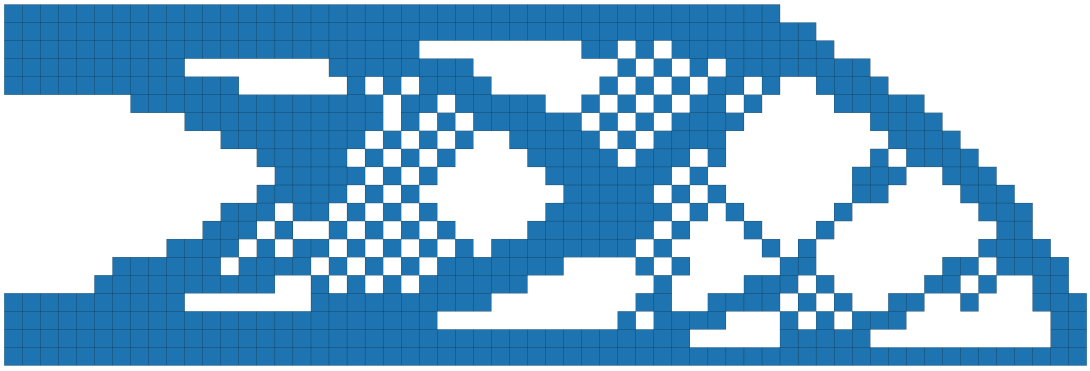


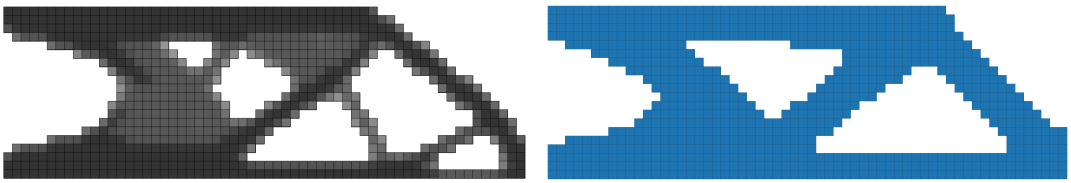
Figure 4.5: Result of the Topology Optimization Application, with a simulation of the cantilever beam and the filtering applied to the deformed model.

radius $r_{\min}$ and therefore a structure that seems to not have been filtered is obtained.

In table 4.3 the simulation results with the updated filter function and the simulation with the old implementation are compared. Comparing the results of the two filtering functions, the old one would seem better. The simulation with the deformed model needs less iterations in order for the objective function to converge. Therefore less time is needed for the optimization. Additionally the compliance of the structure is lower. But if the structural result in figure 4.5, which is the optimized structure of this simulation, is considered, the reason for these improvements is clear: Artificial stiffness due to checkerboard patterns. This leads to a virtually high stiffness in the optimization. The structure, as can be seen

	Old implementation	New implementation
Number of elements	4800	4800
Number of nodes	6408	6408
Number of iterations	21	23
Time of solution [s]	506	556
Compliance of optimized structure	1082	1144

Table 4.3: Comparison of Matlab and KRATOS using the density filter. The parameters form table 4.1 were used.



(a) Cantilever beam optimized in Matlab with density filter

(b) Cantilever beam optimized in Kratos with density filter

Figure 4.6: Optimization of the Cantilever beam with the density filter in Matlab (a) and in KRATOS (b).

in the figure 4.5, is physically not reproducible and has therefore to be avoided.

Although the applied approach works, the filtering could be further improved. For example by implementing the filter before the optimization process, as it is done in the benchmark code. This would not only save computational time but also this would be done before the structural analysis and therefore the model does not have any displacements yet.

As simple as it sounds, this proved to be a bigger problem than anticipated because of the interaction of the C++ code and Python. Therefore the solution found within this work can be considered a satisfying solution.

4.2.2 Evaluation of the Density Filter Implementation

The impact of the density filter onto the results will be over-viewed in this section. In order to activate the density filter the corresponding line in the `OptimizationParameters.py` is changed:

```
density_filter="density"
```

Should the character `"density"` be pronounced in a different way or incorrectly, the filtering function will be ignored during the optimization process.

The figure 4.6 compares the optimization within Matlab and with KRATOS using the den-

	Matlab	Kratos
Number of elements	4800	4800
Number of nodes	6408	6408
Number of iterations	141	28
Time of solution [s]	52	673
Compliance of optimized structure	1274	1247

Table 4.4: Comparison of the old filter function and the new filter function. The parameters form table 4.1 were used.

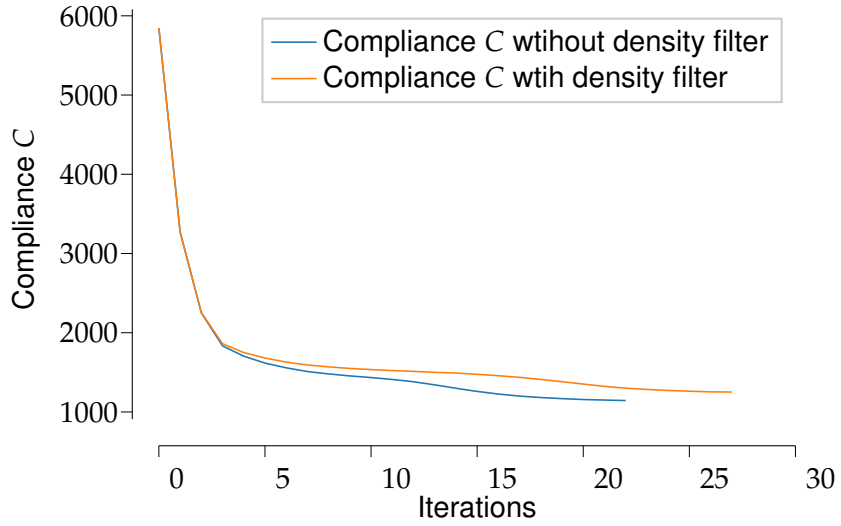


Figure 4.7: Change of compliance over the iterations using OC optimization algorithm. The optimization is done with and without density filter.

sity filter. In both results a less branched structure can be seen if they are compared with the structures in figure 4.3. Due to the the change of the structure, the compliance of both structures is higher than they are in the simulations without the density filter. Further it can be seen, that the compliance in the optimization with Kratos was reduced further than in the simulation with Matlab. By comparing the number of iterations, it can be observed, that KRATOS needs only 28 iterations while Matlab needs 141.

In the figure 4.7 the optimization by Topology Optimization Application with and without the density filter is shown. It can be seen, that the change of compliance is similar in the beginning. With further iterations the results drift apart. In the end the optimization with the density filter needs more iterations than the one without the filter.

Tolerance: 1e-4	Penalty factor: 3
max. Iterations: 100	Filter radius: 1.5
Volume Fraction: 0.3	Grey Scale Filter: No
Young's Modulus: 1	Continuation Strategy: No
Optimization Algorithm: MMA_algorithm	Density Filter: No

Table 4.5: Optimization parameters for the comparison of the MMA method and the OC method

	MMA algorithm	OC algorithm
Number of elements	4800	4800
Number of nodes	6408	6408
Number of iterations	33	23
Time of solution [s]	815	556
Compliance of optimized structure	1153	1144

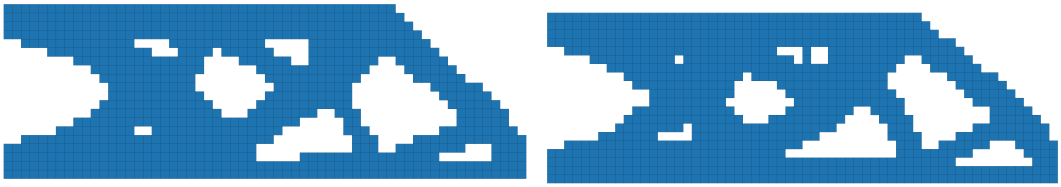
Table 4.6: Comparison of MMA and OC as optimization algorithm, by optimizing the Cantilever beam. The parameters form table 4.5 were used.

4.2.3 Evaluation of the MMA Optimization Algorithm

By introducing the MMA optimization method, the code offers more opportunities for further extensions. This chapter overviews and compares the MMA method with the existing OC method of the TopOpt Application. This is done by using the Cantilever Beam model, which was described in the introduction of this chapter. The parameters for both simulations can be seen in the table 4.5.

The results of the simulations can be seen in figure 4.8 and in table 4.6. Additionally the change of the compliance of the two optimizations can be seen in figure 4.9.

The structures from the optimizations have a high similarity. Even though the MMA algorithm needs more iterations, the compliance values of the optimizations are very close to each other. If the time is considered, it can be seen that optimization of the structure with the OC algorithm is faster than the optimization with MMA. Additionally the plot of the compliance over the time shows, that OC as an optimization algorithm converges faster compared to the MMA algorithm. This for one is because the design value gets updated in the first iterations without further information (see equation 2.36). Only in the third iteration the updating process is improved with every iteration and therefore the code converges faster after that.



(a) Cantilever beam optimized with OC in KRATOS **(b)** Cantilever beam optimized with MMA in KRATOS

Figure 4.8: Optimization of the Cantilever beam with OC (a) and MMA (b) as optimization algorithms in KRATOS.

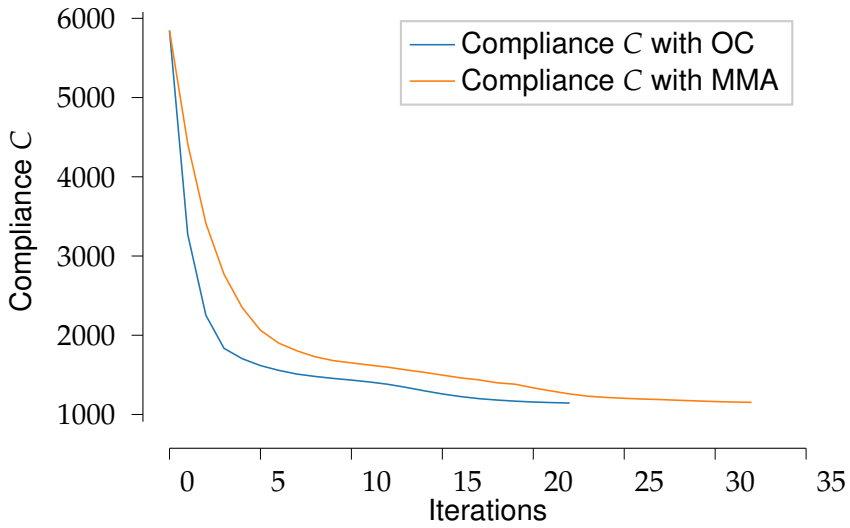


Figure 4.9: Change of compliance over the iterations with OC and MMA as optimization algorithms for the benchmark example.

4.3 Description of the Topology Optimization on an Aerial Lift Pylon

To give an example of how the code could be used in the industry an aerial lift pylon will be optimized. In this section the pylon and its boundary conditions as well as the design domain of the pylon are defined.

Aerial lifts are considered one of the oldest means of transport (Chair of Materials Handling 2020). In steep and harsh environment the cable car is still one of the most common used transportation utility. With over 22000 installed machines all over the world, this mode of transportation is still of importance. They are being used for transporting passengers as well as goods. With the cable car being a sustainable way of transport, their importance for urban transportation has increased in the last years (Chair of Materials Handling 2020). Some of the reasons why a cable car can be used as an urban way of transportation are the following:

- cable cars can be powered by electrical energy and can therefore reduce the production of carbon dioxide (given that the electricity is provided by renewable methods).
- Traffic inside cities could be reduced by using aerial lifts as a way of public transport.
- They are quite independent of the terrain where they are installed.
- There is no interaction between the cable car and other modes of transportation, which means no accidents with other transportation vehicles can occur.

Due to some of these reasons the idea of using cable cars as a public transportation system is present in different cities all over the world. In some cities this was already done, sometimes partly (Alshalalfah et al. 2012).

Although this seems like a good solution for public transport system, the construction of an aerial lift is a challenge and comes with high costs. Therefore the improvement of the different parts of the cable car as a project is an interesting subject.

Cable car systems normally consist of the following mechanical parts:

- Upper and lower station.
- One or more gondolas, depending on the type of aerial lift.

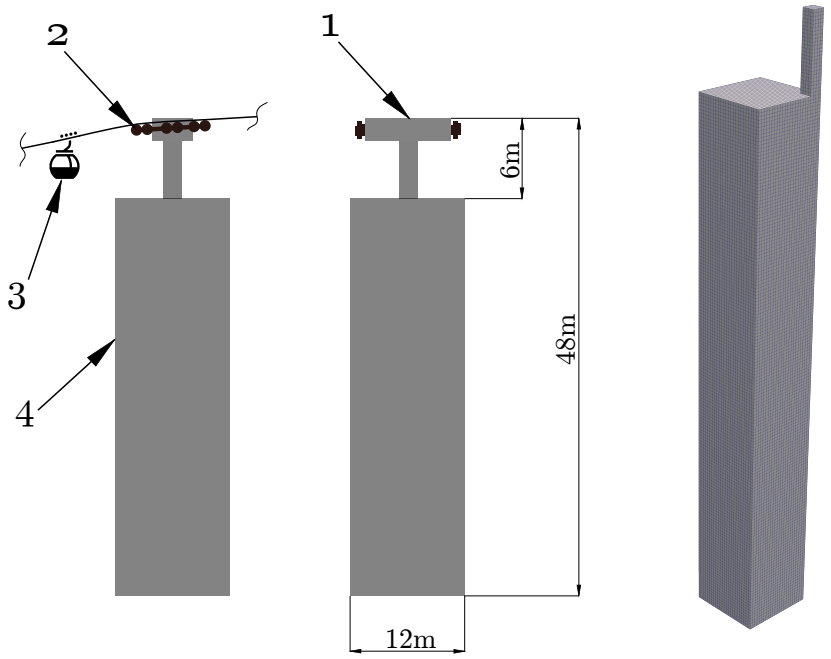
Figure 4.10: Pylon of a aerial lift as a tubular steel structure by LEITNER (Chair of Materials Handling 2020).

- One or more ropes, depending on the type of aerial lift.
- pylons along the line, if needed.

As an example from the industry and an example of how the code could be implemented in the industry a pylon is optimized. To do so the boundary conditions of the pylon have to be taken into account. Due to a collaboration with LEITNER the occurring loads where given for a mono-cable gondola (Leitner 2021). With the given conditions a model was constructed.

The forces and moments onto the structure are given on the top of the pylon (see figure 4.12), where an additional structure is mounted (see figure 4.11). This additional structure is the horizontal steel structure, the so-called yoke, where the shoes or sheave assembly are mounted (see figure 4.11). Onto the shoe or the assembly lies or runs the rope, depending on the type of aerial lift. The forces and momentum are given on the point, where the yoke is connected to the pylon. Therefore the constructed model only considers the pylon up to the point where the yoke is mounted.

In figure 4.11 it is visualized how the pylon was modeled and how the model is connected to the reality. The height of the pylon was given from LEITNER as $47,8m$. For the model a height of $48m$ was chosen. For the foundation of the pylon an assumption was made of



(a) Model of the aerial lift pylon.

(b) Meshed quarter model of the pylon.

Figure 4.11: The design domain (4) is shown in (a) with additional structures and parts of the pylon. The yoke (1), the shoes or sheave assembly (2) and the gondola (3) can be seen. In (b) the meshed design domain for the optimization is shown (mesh size = $0.5m$).

the foundation surface to be $12\text{x}12m$. This assumption was made for two reasons: The first reason was that the pylon is quite high for a mono-cable gondola system, therefore to provide enough stability, these measurements where chosen. The second reason was to give the optimization algorithm more space to develop the optimized structure. One might wonder, why the pylon is narrowed on the top. This was done, in order for the gondola to pass the pylon without any problems (see figure 4.11).

LEITNER provided the different load cases of the pylon. For the simulation in this work a fictional load-case was taken. For this the highest forces and highest moments are considered. The moments and forces result from:

- load of the cable car
- snow
- wind in the direction of the rope-way / perpendicular to the direction of the rope-way

Directions	Force	Moment
x-direction	$60kN$	$210kNm$
y-direction	$115kN$	$-18kNm$
z-direction	$200kN$	$-100kNm$

Table 4.7: Forces and moments working on the top of the pylon. This is an assumed load-case.

- ice on the ropes
- dynamic effects
- tension and weight of the ropes

The moments and forces are applied on the top of the pylon (see figure 4.12). The loads that are used can be seen in table 4.7.

In this case only one load case is considered, which would result in a structure, that is stronger on one side. Also this load case does not consider all the cases that can occur on the pylon. Depending on the way the aerial lift is used one of the rope sides can have higher loads. It has also a major affect on loads, considering which side the cable cars drive uphill and on which side downhill. Additionally, it is not known if the majority of the weight is transported downhill or uphill. Due to this uncertainties an assumption was made: In this work the pylon will be seen as a symmetrical building. Therefore only one quarter of the model can be taken for the simulation and with the symmetry the pylon can be constructed by mirroring the quarter model.

For the simulation the model had to be discretized. This was done by hexahedral elements with the size of $0,5m$. With this element size, the model was divided into 97152 elements and 106225 nodes (see figure 4.11).

4.4 Validation of the Implementations considering the Aerial Lift Pylon

To see a possible use of the Topology Optimization Application, the model of the aerial lift pylon was simulated in this section.

To not only show the Topology Optimization Application, but also the extension by the MMA algorithm, the simulation was done twice. The first time the pylon model was optimized using OC, whereas for the second optimization the MMA algorithm was used. In table 4.8 the used optimization parameter can be seen. The assumption was made, that the pylon would fill 0.15% of the developed model.

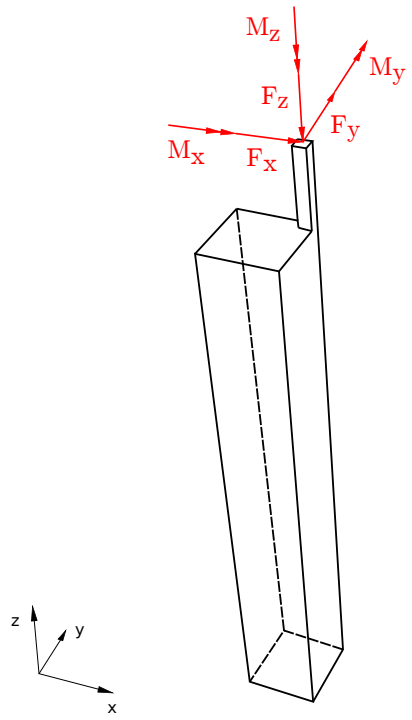


Figure 4.12: Forces and moments applied on the structure.

Tolerance: 1e-6	Penalty factor: 3
max. Iterations: 100	Filter radius: 0.75
Volume Fraction: 0.15	Grey Scale Filter: No
Young's Modulus: 210e+9	Continuation Strategy: No
Optimization Algorithm: MMA_algorithm	Density Filter: No

Table 4.8: Optimization parameters for the aerial lift pylon model.

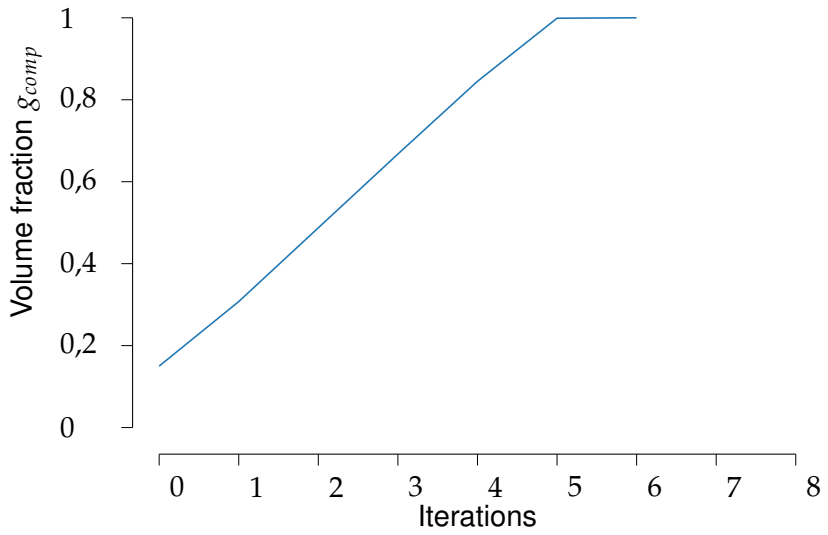


Figure 4.13: Change of volume fraction over the iterations with the implementation of the MMA algorithm in the Topology Optimization Application.

Before the results of the optimization are shown, it has to be mentioned, that with the optimization using MMA a problem occurred. Within the optimization algorithm the constraint function value, as implemented by equation 3.2, was not able to keep the volume fraction of every iteration below the set volume fraction limit. Therefore the volume fraction increased with every iteration (see figure 4.13). For this reason the constraint function value was set to a high number (1e+11) in order to keep the volume fraction in the allowed area. With this alternation of the code, the optimization of the pylon with MMA was possible.

The results of the optimizations can be seen in the table 4.9. As expected from the previous section, the OC algorithm converged within less iterations that the MMA algorithm did. Therefore also the time for optimizing the structure with OC was lower than the time needed with MMA.

Even though the optimization with MMA had more iterations, the resulting compliance of the two structures are not far apart. Especially if one considers, that the initial compliance was over 700000. Due to the fact that KRATOS was installed as debug version the time consumption of both simulations was very high.

In the figure 4.14 the resulting structures of the simulations can be seen. In this figure the obtained quarter models are shown. The model optimized with OC (on the left side) has a more regular structure when compared to the structure optimized with the MMA

	Optimality Criteria	Method of Moving Asymptotes
Number of elements	97152	97152
Number of nodes	106225	106225
Initial compliance	714199	714199
Number of iterations	84	100
Time for simulation $[s]$	30243	82049
Compliance of the optimized structure	4404	4546

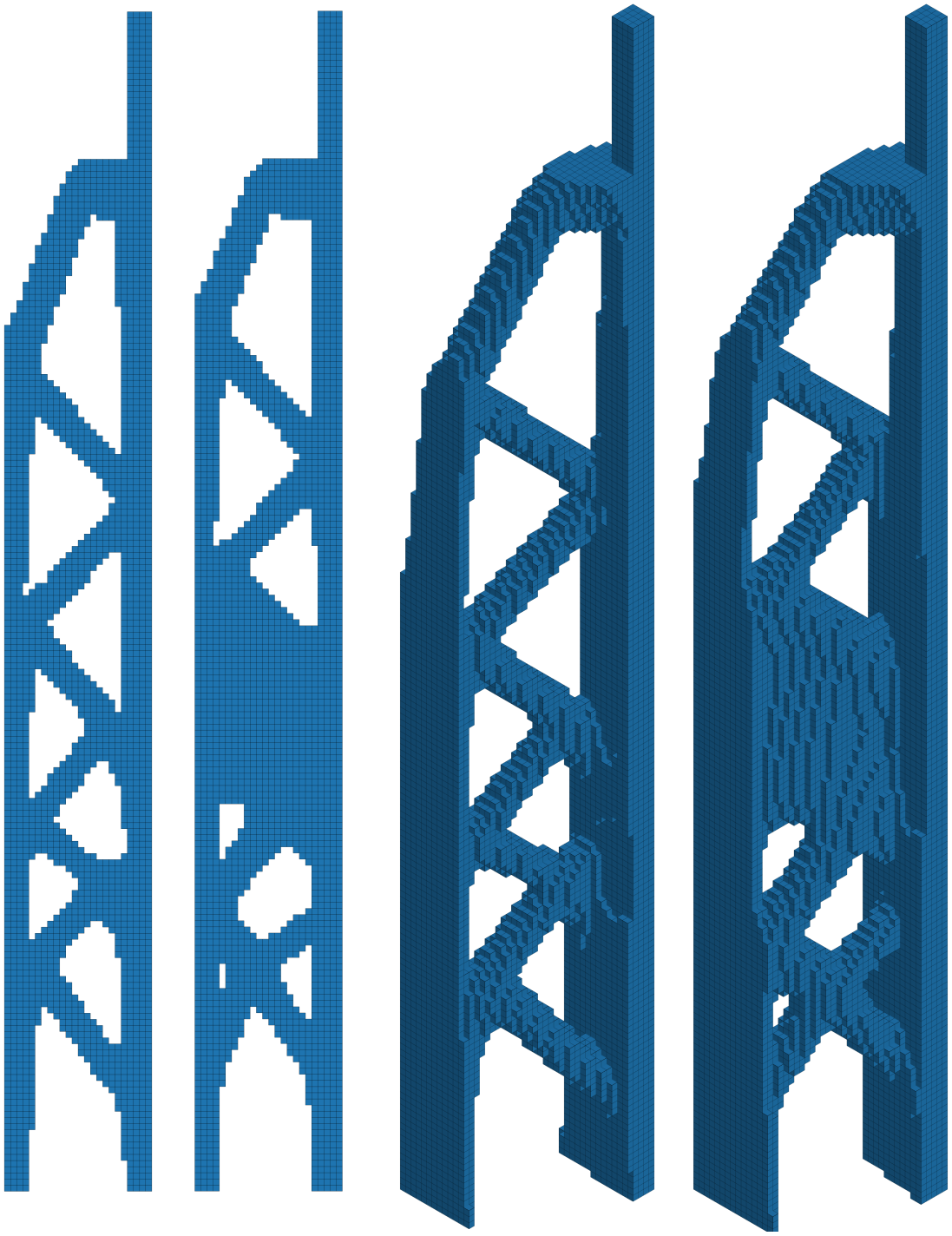
Table 4.9: Direct comparison of the simulations of the aerial lift pylon using the quarter model.

algorithm. This for one can be due to the problem with the constraint function, which was circumvented by setting the a very high constraint function value. On the other hand, this obtained structure could also be a result from the optimization algorithm reaching the 100 iterations, which were set as limit. By reaching the limit the optimization was stopped and therefore this could be the reason for the resulting structure.

Additionally in the figure 4.15 the optimized pylons can be seen from above. In this perspective it can be seen, that the material was mainly distributed in the middle of the model. This is due to the resulting force vector, which was applied to the structure. The forces were applied in the figure in order to show how they act on the structure. As stated in table 4.7 the force in y-direction is higher than the one working in x-direction. Therefore the distribution of the material is not in a 45° angle, it is shifted clockwise (considering the figure). This is to give more stability in the y-direction. In addition it can be observed that within the design domain of the structure optimized with MMA more material can be found. This too is due to the fact that the simulation with MMA was non fully converged yet.

Within the plots of the figure 4.16 the change of the objective function as well as the change of the volume fraction with the OC optimization algorithm can be seen. It can be observed that the compliance is minimized very fast in the beginning of the optimization, and than slowly converges within the 84 iterations. One can see also, that the volume fraction limit was also violated within the optimization. As can be seen in the figure, the volume fraction increased in the beginning and was then forced back to the allowed value.

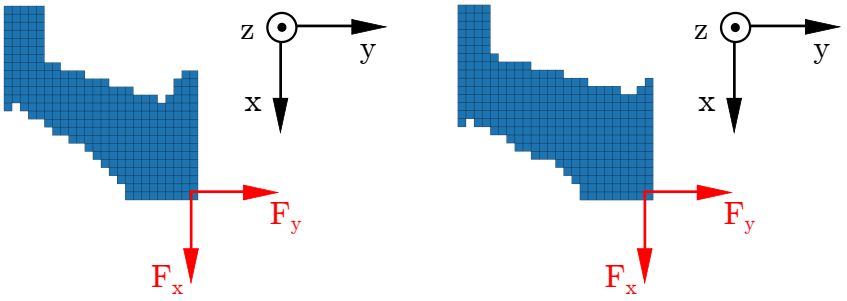
Figure 4.17 shows the change of the compliance and the volume fraction over the iterations using MMA as the optimization algorithm. It can be seen, that the compliance,



(a) View from the front: Left OC right MMA

(b) 3D view: left OC right MMA

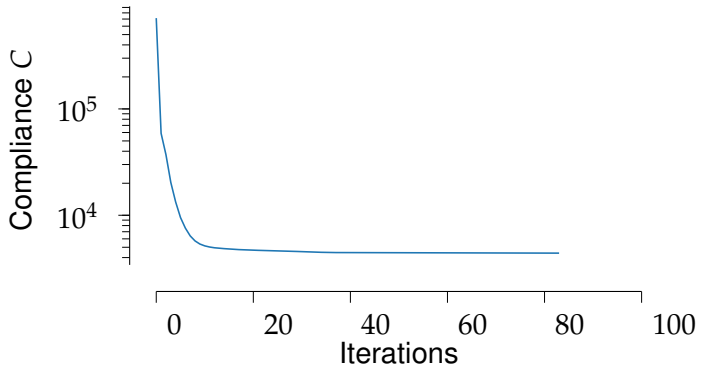
Figure 4.14: Resulting structures from the optimization of the aerial lift pylon model. The figure (a) shows the results in a 2D view, (b) shows the results in a 3D view.



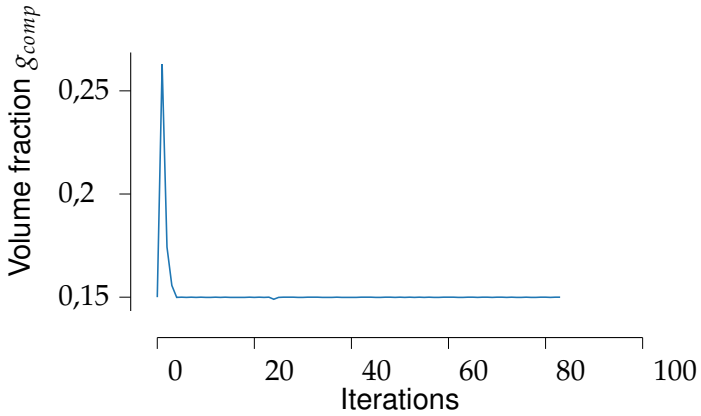
(a) Pylon seen from the top (OC)

(b) Pylon seen from the top (MMA)

Figure 4.15: The optimized pylons seen from above. In figure (a) the optimization was done with OC, in (b) with MMA.

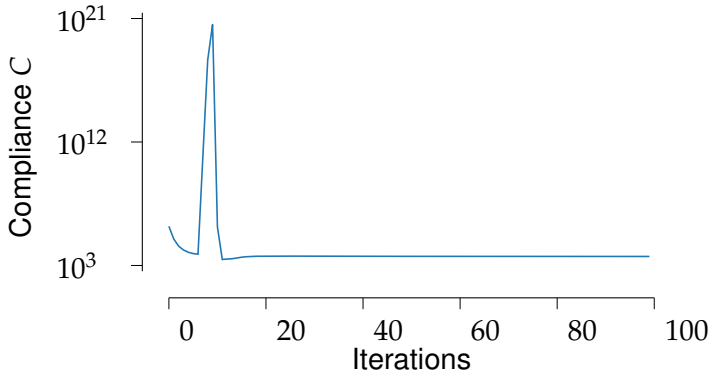


(a) Change of compliance in the optimization of the aerial lift pylon, using OC as optimization algorithm

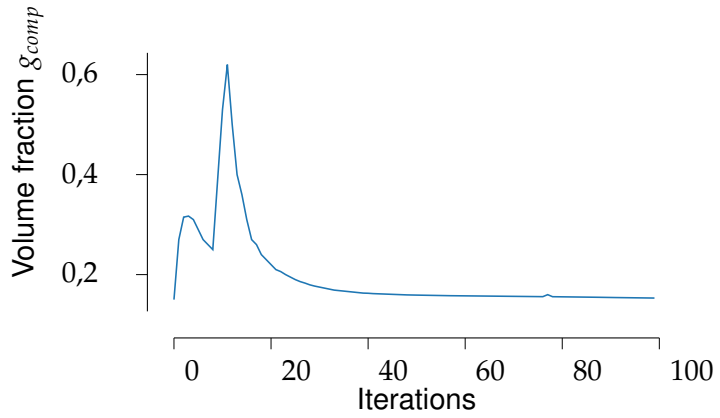


(b) Change of the volume fraction in the optimization of the aerial lift pylon, using OC as optimization algorithm

Figure 4.16: Change of the objective function (a) and the volume fraction (b) within the optimization of the aerial lift pylon, using OC as optimization algorithm.



(a) Change of compliance in the optimization of the aerial lift pylon, using MMA as optimization algorithm



(b) Change of the volume fraction in the optimization of the aerial lift pylon, using MMA as optimization algorithm

Figure 4.17: Change of the objective function (a) and the volume fraction (b) within the optimization of the aerial lift pylon, using MMA as optimization algorithm.

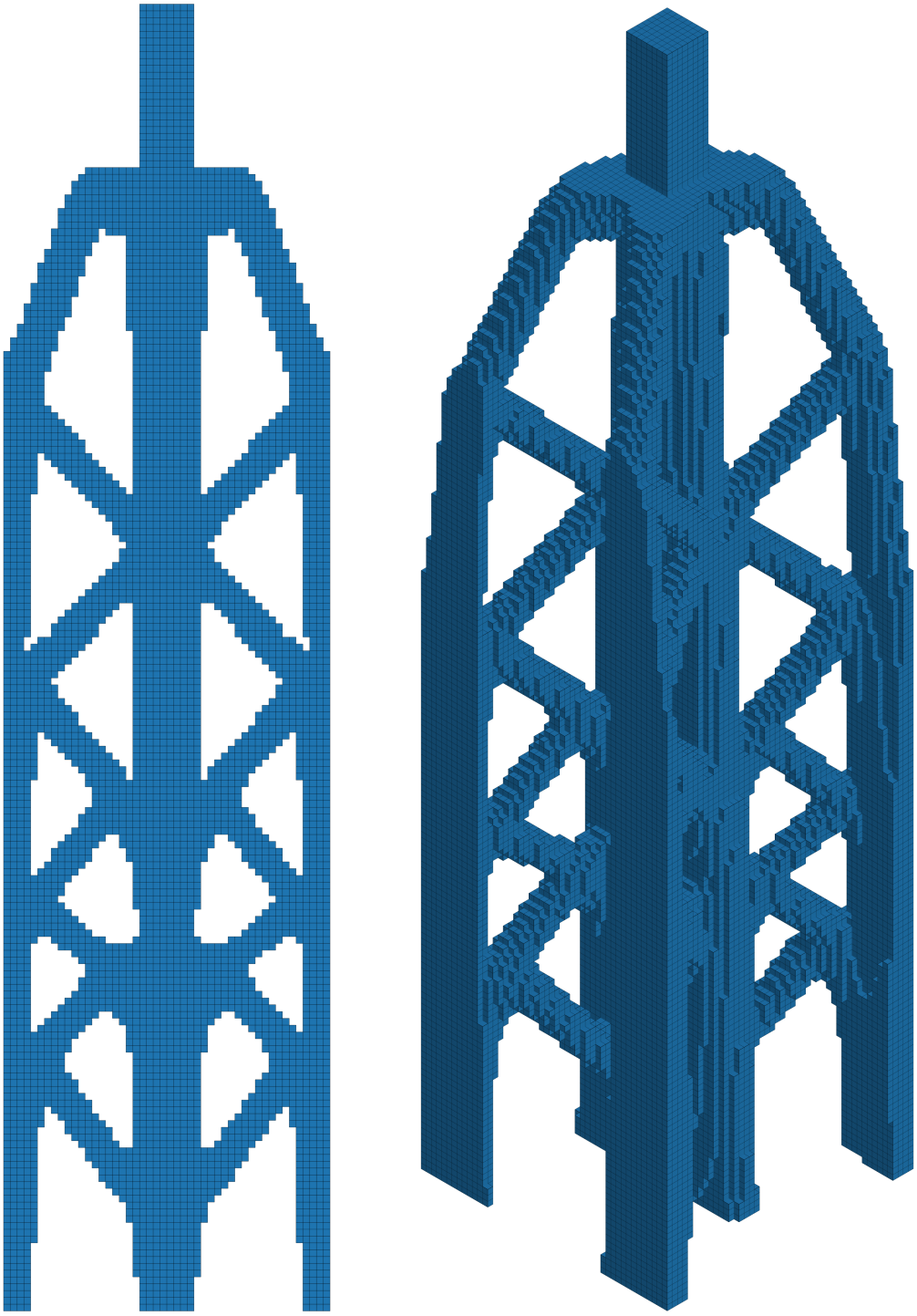
equivalent to the OC optimization, converges very fast in the beginning. But at the iteration number 8 the objective function increases rapidly until it falls back down 4 iterations later. This is a problem that should not occur. This could be caused by the alternation, which was done to circumvent the increasing volume fraction in every iteration. After this the objective function converges constantly and results in a compliance similar to the one of the simulation done with the OC algorithm (see table 4.9). The volume fraction increases also in the beginning and a second time in the period where also the objective function increases. After that the volume fraction decreases slowly to the point where it reaches the allowed value.

The problem with the increasing volume fraction could be caused due to the fact, that the MMA script is initiated in every iteration. As mentioned before, the script was intended

to be implemented in a C++ code, where it is initiated once and the optimization is done, while the script is active at all times. Although several implementations were made in order to provide the MMA updating procedure with the needed information, this could be the source of the problem and needs further investigation in the future.

In order to present the resulting aerial lift pylon as one, the quarter models were mirrored with paraview. The resulting structures can be seen in figure 4.18 and 4.19. From this structures it can be observed, that the pylon optimized with OC has a more regular structure, whereas the one optimized with MMA includes more material in the middle part (see figure 4.19).

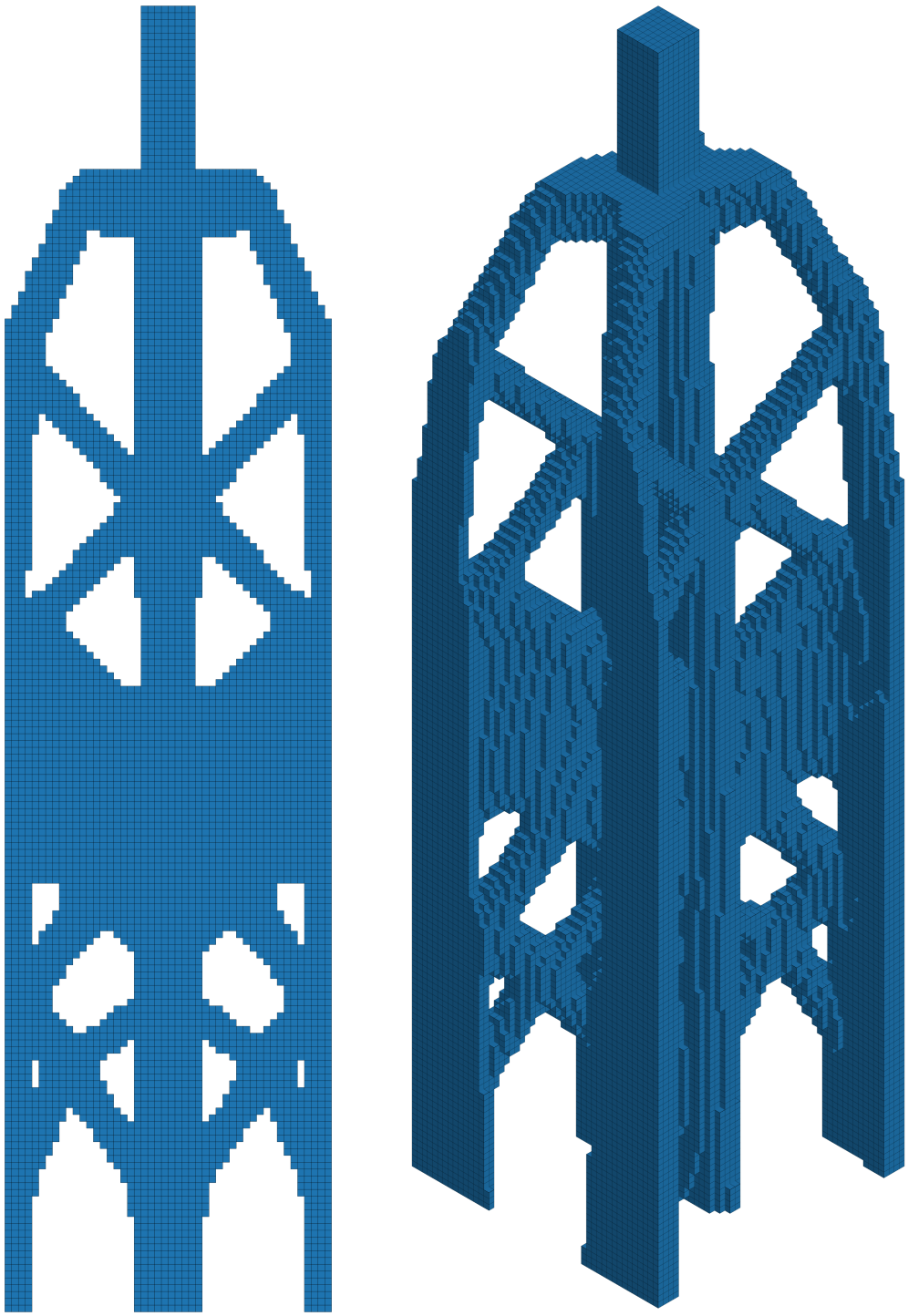
The obtained structures are oversized. To much material would be needed in order to realize this structures. But with this simulation it has been shown that examples from the industry can be modeled and optimized with Topology Optimization Application.



(a) 2D view of optimized pylon using OC as optimization algorithm

(b) 3D view of optimized pylon using OC as optimization algorithm

Figure 4.18: Aerial lift pylon optimized with Topology Optimization application using OC as optimization algorithm. Figure (a) shows pylon in 2D view, (b) shows the pylon in 3D view.



(a) 2D view of optimized pylon using MMA as optimization algorithm

(b) 3D view of optimized pylon using MMA as optimization algorithm

Figure 4.19: Aerial lift pylon optimized with Topology Optimization application using MMA (modified) as optimization algorithm. Figure (a) shows pylon in 2D view, (b) shows the pylon in 3D view.

5 Outlook and Conclusion

Topology optimization and lightweight design remain an important factor in the industry as well as the research. Not only can structures be optimized by using less material, but the distribution of the material can lead to improved performances regarding stiffness or design. In addition, the increasing importance of additive manufacturing has a positive impact on the use of optimization methods. Designs that seemed to need a high effort to build in the past, are now being realized by this design method. Also the increasing performance of computers reduces the computational time and so large models can be simulated in a reasonable amount of time.

Topology Optimization Application, as an open source optimization tool in KRATOS, can be used for the presented purposes of optimization problems. 3D models can be crated and optimized within the this KRATOS tool. With it being open source, the opportunity for companies and research facilities stands to use this application in order to solve given problems or to introduce it to research topics. Provided, that the tool is further improved and extended, this could be the case for the Institute of Aircraft Design at the Technical University Munich as well as the Faculty of Science and Technology at the Free University of Bozen-Bolzano. The optimization tool could be introduced to projects like MILAN and extended accordingly.

The main interest of this thesis was to reactivate the existing (but not functional) Topology Optimization Application by linking it to the Structural Mechanics Application in KRATOS. By doing this the large scale topology optimization tool is now available in the KRATOS framework. In fact the thesis not only reactivated, but also extended the existing application.

5.1 Summary

This section offers a short summary of the new functions and improvements inside the code. The data, figures and results, which are summed up in this section, can be found on the following repository: `https://github.com/PhiHo-eng/Semester_Thesis_Hofer_Philipp_Data`.

Topology Optimization Application was linked to the Structural Mechanics Application in KRATOS. This was done by including the functions and scripts of Structural Mechanics Application into the code. Additionally the new PyBind11 as the interface between

Python and C++ was included in order for the code to communicate within and with the applications in KRATOS.

The improvement of the filtering method, where the filtering is now done on the undeformed structure, was the first step. This allows the software to optimize also structures with a very low Young's modulus. This is also important if the Young's modulus is normalized, which is common in topology optimization. Therefore this step was of great interest and as it can be seen in the results, the Young's modulus dependency of the filtering process was removed.

With the implementation of the density filter, not only can less branched structure be realized, also the opportunity for the extension of the code with the Heaviside projection is given. The more concrete structure, as it can be seen with the density filter, can be of importance if conventional construction techniques are used. Although the compliance is not as low as it would be without the filter, but the obtained model has a "easier" structure.

Introducing MMA to the topology optimization tool, does not only add a second optimization method, but also the opportunity to further extend the code. Due to the fact that MMA can optimize a problem with more than just one constraint, extensions as for example stress restrictions can be implemented into the code.

Comparing MMA with the OC optimization method, by using the benchmark example of the Cantilever beam, shows good results. Although OC achieves the simulation within less iterations, the resulting structure and compliance are very similar. With an additional change within the code, also the results of the optimized pylon show similarities. Taking the occurring problem into account, it has to be said, that the implementation of the MMA script within the optimization process needs further examination and possible alternations, in order for larger examples to be optimized without additional changes in the code.

5.2 Outlook

As proposed before, the Topology Optimization Application can and should be improved and extended, in order to be included within projects and further research. Interesting topics for possible extensions and therefore interesting for further research with this code are:

- Stress constraints by taking advantage of the new implementation of MMA within

the code.

- Implementation of non-linear elements
- Implementation of an-isotropic material
- multiple load-cases and multiple boundary conditions

In addition the post processing is not fully reactivated in this version. Only the volume extraction of the optimized design is possible. By re-implementing the surface mesh and the surface mesh smoothing, an STL file could be obtained. This would offer an additional advantage when using the Topology Optimization Application in combination with additive manufacturing.

5.3 Conclusion

Based on the shown results and the data within this thesis, it can be stated, that the reactivation of the Topology Optimization Application was done successfully. In the introduction the “virtuous circle of lightweight engineering design” was mentioned, were an optimization results in lower forces within the design and therefore in an lighter structure. In a way this principle can also be applied to the work done within this thesis: With additional functions better solutions were obtained and computational improvements were achieved. It is now up to further research and additional work to optimize the Topology Optimization Application.

Bibliography

Niels Aage, Erik Andreassen, and Boyan Stefanov Lazarov. Topology optimization using petsc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization*, 51(3):565–572, March 2015. ISSN 1615-1488. URL https://doi.org/10.1007/s00158-014-1157-0.

B. Alshalalfah, Amer Shalaby, Steven Dale, and Fadel Othman. Aerial ropeway transportation systems in the urban environment: State of the art. *Journal of Transportation Engineering*, 138:253–262, March 2012.

Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S. Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, January 2011. ISSN 1615-1488. URL https://doi.org/10.1007/s00158-010-0594-7.

Horst Baier, Christoph Seeßelberg, and Bernhard Specht. *Optimierung in der Strukturmechanik*. Vieweg+Teubner Verlag, 1994.

Martin P. Bendsoe. *Optimization of Structural Topology, Shape, and Material*. Springer, Berlin, Heidelberg, 1995.

Ole Bendsoe, Martin Philip; Sigmund. *Topology Optimization Theory, Methods, and Applications*. Springer, 2004. doi: 10.1007/978-3-662-05086-6.

K. Bletzinger. *Structural Optimization*, 2020.

Tyler E. Bruns and Daniel A. Tortorelli. Topology optimization of non-linear elastic structures and compliant mechanisms. *Computer Methods in Applied Mechanics and Engineering*, 190(26):3443–3459, March 2001. ISSN 0045-7825. URL https://www.sciencedirect.com/science/article/pii/S0045782500002784.

Logistics TÜV Süd Industrie Service GmbH WPK Austria GmbH Chair of Materials Handling, Material Flow. *Seilbahntechnik*. Lehrstuhl fMür Fördertechnik Materialfluss Logistik, Technische Universität München, Fakultät für Maschinenwesen, 2020.

Eric Ricardo Gonzalez. Three-dimensional topology optimization of arbitrary geometries using open-source general-purpose finite element software. Master's thesis, Technische Universität München, 2015.

Albert A. Groenwold and L. F. P. Etman. A simple heuristic for gray-scale suppression in optimality criterion-based topology optimization. *Structural and Multidisciplinary Optimization*, 39(2):217–225, August 2009. ISSN 1615-1488. URL https://doi.org/10.1007/s00158-008-0337-1.

J. K. Guest, J. H. Prévost, and T. Belytschko. Achieving minimum length scale in topology optimization using nodal design variables and projection functions. *Int. J. Numer. Meth. Engng.*, 61(2):238–254, September 2004. ISSN 0029-5981. URL https://doi.org/10.1002/nme.1064.

Lothar Harzheim. *Strukturoptimierung: Grundlagen und Anwendungen, Edition Harry Deutsch*. Europa Lehrmittel Verlag, 2014.

William Hunter. Predominantly solid-void three-dimensional topology optimisation using open source software. Master's thesis, University of Stellenbosch, Department of Mechanical and Mechatronic Engineering, 2009.

Leitner. Loadcases for aerial lift pylon. Technical report, Leitner, 2021.

Kai Liu and Andrés Tovar. An efficient 3d topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 50(6):1175–1196, December 2014. ISSN 1615-1488. URL https://doi.org/10.1007/s00158-014-1107-x.

Octaviano Farias Malfavon. Development of an open-source tool for 3d topology optimization and its extension towards integrated design and analysis. Master's thesis, Technische Universität München, 2016.

Joseph Reinisch. Topologieoptimierung von compliant mechanisms mit nichtlinearer fem und spannungsrestriktionen. Technical report, Technische Universität München, 2017.

Joseph Reinisch. Synthesis of compliant mechanisms for morphing wings with nonlinear

topology optimization. Master's thesis, Technische Universität München, 2019.

O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, April 2001. ISSN 1615-1488. URL https://doi.org/10.1007/s001580050176.

O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural optimization*, 16(1):68–75, August 1998. ISSN 1615-1488. URL https://doi.org/10.1007/BF01214002.

Ole Sigmund. *Design of Material Structures Using Topology Optimization*. PhD thesis, January 1994.

Ole Sigmund. On the design of compliant mechanisms using topology optimization. *null*, 25(4):493–524, January 1997. ISSN 0890-5452. doi: 10.1080/08905459708945415. URL https://doi.org/10.1080/08905459708945415.

Ole Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33(4):401–424, April 2007. ISSN 1615-1488. URL https://doi.org/10.1007/s00158-006-0087-x.

Krister Svanberg. The method of moving asymptotes–a new method for structural optimization. *Int. J. Numer. Meth. Engng.*, 24(2):359–373, February 1987. ISSN 0029-5981. URL https://doi.org/10.1002/nme.1620240207.

Krister Svanberg. *MMA and GCMMA â two methods for nonlinear optimization*. KTH, Stockholm, Sweden, 2007.

Cameron Talischi, Glaucio H. Paulino, Anderson Pereira, and Ivan F. M. Menezes. Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Structural and Multidisciplinary Optimization*, 45(3):329–357, March 2012. ISSN 1615-1488. URL https://doi.org/10.1007/s00158-011-0696-x.

Prof. Dr.-Ing. W. A. Wall. *Finite Elemente*. Lehrstuhl fur Numerische Mechanik, 2020.

M. Zhou and G. I. N. Rozvany. The coc algorithm, part ii: Topological, geometrical and generalized shape optimization. *Second World Congress on Computational Mechanics*, 89(1):309–336, August 1991. ISSN 0045-7825. URL https://www.sciencedirect.com/science/article/pii/0045782591900469.

Ji-Hong Zhu, Wei-Hong Zhang, and Liang Xia. Topology optimization in aircraft and aerospace structures design. *Archives of Computational Methods in Engineering*, 23 (4):595–622, December 2016. ISSN 1886-1784. URL https://doi.org/10.1007/s11831-015-9151-2.

6 Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ausschließlich unter Zuhilfenahme der im Literaturverzeichnis angegebenen Quellen angefertigt habe. Aus anderen Publikationen und Veröffentlichungen entnommenen Ideen, Abbildungen und Textstellen sind als solche direkt kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt worden.

Garching, den 17. Mai 2021

Philipp Hofer

A Appendix

A.1 Users Instruction

This chapter offers a quick guidance through the newly implemented code and how to use the new implementations. Beginning with the files which need to be changed or altered in order to perform a topology optimization (see figure 3.2).

The file descriptions inside `run_topOpt.py` have to be changed in order for the program to work and calculate the defined model. Inside the .mdpa file the right elements have to be set (SmallDisplacementSIMPElement). `ProjectParameter.json` contains the boundary conditions and solver settings, which normally don't need to be changed. Inside the `OptimizationParaeters.py` parameters for the code are adjusted in order for the optimization to work the way it is intended. For example the settings could be as in table A.1.

The simulation can be launched inside the terminal. By navigating to the folder with the needed files and writing the following command:

`python3run_TopOpt.py`,

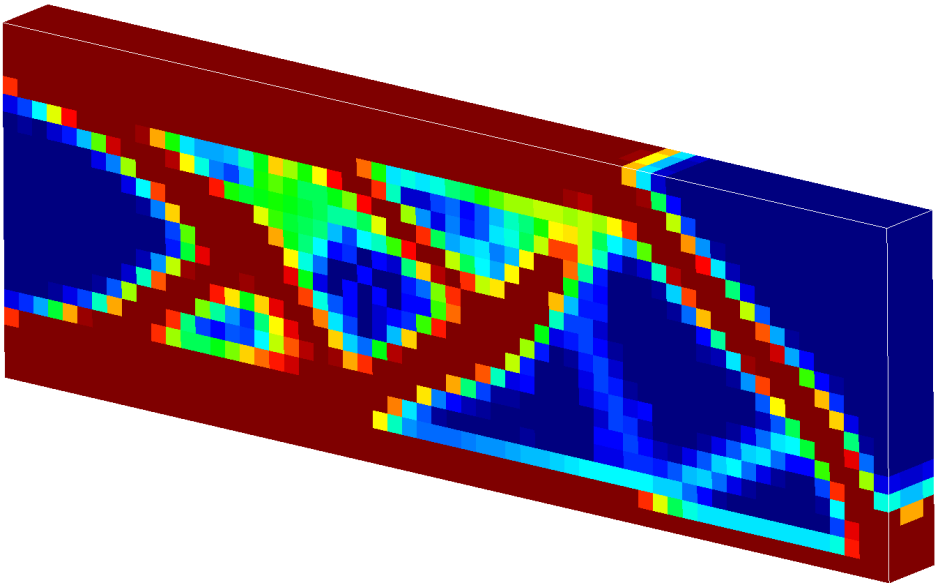
the simulation is initiated.

The KRATOS logo appears as well as the Structural Mechanics logo and the Topology Optimization logo. Given the case that all parameters were set accordingly, the optimization will be executed. While running the scripts in figure (3.3) their functions are called. The optimization follows the work flow as structured in the figure (A.1).

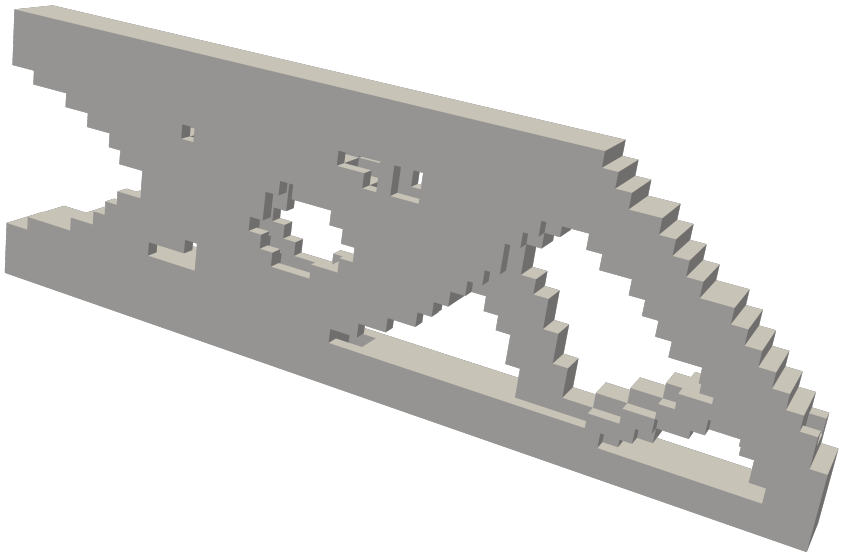
If the convergence criteria or the maximum number of iterations is reached, an output-file is produced, which can be observed with a post-processing software. Additionally the objective function, the constraint function value as well as the absolute and relative percentage change of the objective function can be seen on the screen.

Tolerance: 1e-4	Penalty factor: 3
max. Iterations: 100	Filter radius: 1.5
Volume Fraction: 0.5	Grey Scale Filter: No
Young's Modulus: 1	Continuation Strategy: No
Optimization Algorithm: MMA_algorithm	Density Filter: No

Table A.1: Optimization parameters



(a) Output file of the simulation viewed in GiD.



(b) Structure after TopOpt psot processing in paraview

Figure A.1: Output file of the simulation viewed in GiD and paraview.

In the case of this work the post-processing software GiD is used. Opening the output-file with a post-processor shows a structure as in figure A.1 (a). In the last iteration a `restart_file` is printed. By running the `run_TopOpt_PostProcessing.py` file with the filename of the last restart file, elements below a certain threshold (defined by the user) can be deleted and the structure in figure A.1 (b) can be obtained.