Department of Precision and Microsystems Engineering

Machining cost estimation in topology optimization for 2.5-axis CNC milling

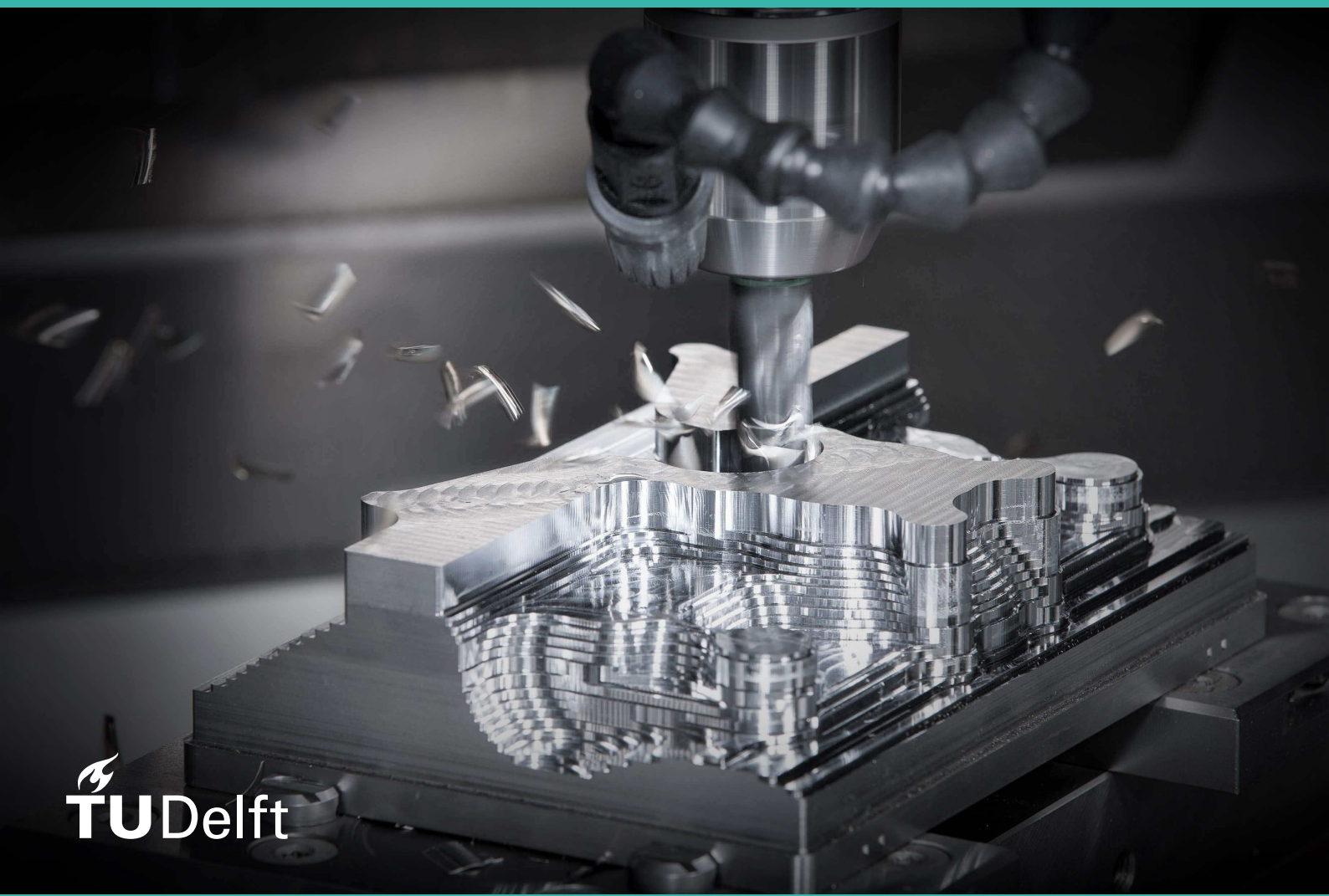
D.P. Treurniet

Report no : 2021.092
Coach : Dr.ir. J.F.L. Goosen
Professor : Dr.ir. M. Langelaar
Specialisation : SOM
Type of report : Master thesis
Date : 16-11-2021

Machining cost estimation in topology optimization for 2.5-axis CNC milling

D.P. Treurniet

November 16, 2021
Delft University of Technology
Carried out at Demcon Robotic Systems



TU Delft

Abstract

The work presented in this report is an attempt to allow density-based topology optimization algorithms to take into account machining costs when optimizing. To this end, and estimation model for the cost of 2.5-axis CNC milling was developed based on the principles of Design for Manufacturing.

This estimation model was implemented in a generic topology optimization algorithm in five steps of complexity. The behaviour of the model was evaluated at each step by means of a set of experiments, showing how the addition of the cost estimation influences the optimization results.

The most promising results from the experiments are validated with Autodesk Fusion 360, with which the designs were programmed to be machined on a CNC mill. The machining times from this software were used to calculate the actual machining costs and these were compared to the machining costs estimated by the optimizer.

Two formulations of the method were found to be useful for estimating the machining cost of designs. A simple formulation that uses the shape factor and the differentiation between internal and external pockets to estimate the cost resulted in a cost saving of 13% at the expensive of a compliance increase of 9%. The more complex formulation uses multiple pocket domains to evaluate pocket-specific properties. This allowed a more accurate estimation of the costs, but did not result in a better performing optimization. The cost saving was comparable at 12%, but the compliance increased by 16%.

It was concluded that there is a use for both methods. The simple formulation allows to find cheaper designs, but does not allow much control in the cost estimation. The complex formulation however can be used to fine-tune the method for a specific situation, which could enable it to perform better that the simple formulation.

Contents

Chapter 1
Introduction

Throughout the history of manufacturing, many different processes have been developed that are being used in all kinds of applications today. Some processes are highly optimized for mass production of simple parts, such as stamping and injection moulding. Other processes are better suited for smaller quantities with more challenging requirements, such as turning and milling. While additive manufacturing techniques are gaining popularity, especially in the prototyping industry, it is not expected that they will outperform conventional machining processes in the near future. Factors like speed, cost at large volumes and usable materials are holding back the adaptation of additive manufacturing in the current industry for the coming years [19].

In the category of conventional manufacturing processes, milling is an important process that is often used in many situations. When complex geometries are required with high geometrical accuracy, milling is one of the only processes that can achieve this efficiently. For this reason, milling is the manufacturing process of choice for many components in industries like aerospace, semiconductor and medical. The first milling machines were operated by manual control (e.g. the leftmost machine shown in Figure 1.1), which required a skilled operator for every machine. When Computer Numerical Control (CNC) technology was introduced in the 1960s, this process became even better suited for industrial manufacturing. The machine in the middle of Figure 1.1 is an example of such a CNC mill. Programming these machines only had to be done once and operating them became much easier to do [16]. In more recent years, robotics have been used to automate the milling process even more, replacing machine operators by robotic handlers that can extract finished components and load new materials in the machines. The rightmost machine shown in Figure 1.1 is an example of a mill that can be fully integrated with robotics to achieve this level of automation.

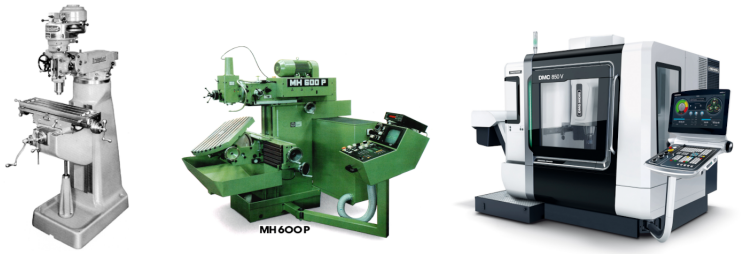


Figure 1.1: Examples of milling machines from different time periods. Left to right: Bridgeport (USA, 1966), Maho MH600P (Germany, 1983), DMG Mori DMC 850 V (Japan, 2020).

During the same time period that these advancements were made in the field of milling technology, a novel design method was being developed in universities as well. In the field of engineering, the concept of structural optimization was known for many years, but was impractical to use due to its complexity. This changed when high-speed computers became available to researchers in the middle of the 20th century. The development of new optimization methods and the finite element analysis resulted in the development of topology optimization techniques that are still actively being worked on today [23].

Topology optimization uses computational methods to find the optimal distribution of material in a domain to achieve a certain goal. Examples of such a goal are to create a structure that is as stiff as possible with a limited amount of material, has certain dynamic properties such as specific eigenfrequencies or allows the most efficient heat extraction. These optimal designs often show complex geometries that would not have been possible to come up with otherwise. This makes the method

especially useful when conventional design methodology is not sufficient to achieve the required performance of a design. However, the typical complexity of optimized designs makes them challenging to manufacture as well, hindering the use of topology optimization in real engineering applications. This motivated researchers to implement methods of incorporating manufacturing constraints for all kinds of processes, among which also milling. Examples of such constraints are presented by Vatanabe et al. [26] and Langelaar [14].

When considering milling as a manufacturing process, a few relevant constraints have been developed for topology optimization. Because the smallest tool diameter determines the minimum radius that can be cut, a minimum feature size on the void phase is useful to apply. This ensures that no features smaller than a set diameter can be present in the result. A minimum feature size for the solid phase is useful as well, because thin walls can cause unwanted vibrations, leading to poor quality and accuracy [8]. The two methods can be combined to ensure a minimum feature size in both phases [7]. Another important aspect of millable designs is tool accessibility. With the cross-sectional shape of the tool given, Langelaar showed how tool accessibility can be guaranteed for the optimized design [14].

The implementation of manufacturing constraints to topology optimization has improved its industrial usability by ensuring the machinability of generated designs. However, optimality for manufacturing can not be achieved by only taking into account the constraints of the machining process. Many factors determine how suitable a design is for a specific manufacturing process (geometry, tolerances, materials used), but a single metric that can be used to generalize these aspects is the manufacturing cost. From an industrial point of view, a design can be regarded as optimal for a manufacturing process when the cost of manufacturing has been minimized. Hence, to improve the usability of topology optimization in the industry, the cost of manufacturing should be included as a part of the optimization process.

How the cost of manufacturing is determined depends on the specific process that is considered. The scope of this research is therefore limited to milling, which was identified as an important manufacturing process in the industries that already employ topology optimization as an design method. To make the first steps in the development of such a method, the simplest type of millable designs is considered first, namely those suitable for 2.5-axis milling. Such a design can be described using a 2-dimensional image, which can be projected into the third dimension. Machines designed for this type of milling can translate the tool in all 3 directions relative to the workpiece, but often lack the ability to move all 3 axis simultaneously due to software or hardware limitations [17]. Instead, the tool is positioned at the right height and the cutting is done by moving the tool in the horizontal plane. By only considering this type of millable designs, a 2-dimensional topology optimization framework can be used, which is desirable for reasons of simplicity.

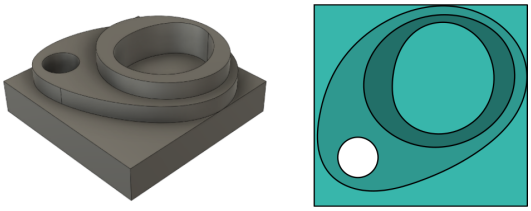


Figure 1.2: Example of a 2.5-axis millable part. Left: 3-dimensional view of design. Right: 2-dimensional description of the design in which the colour represents the height of that region.

Given the reasons why machining cost is an important aspect to consider when trying to optimize designs in the industry, the research question that is considered in this work can be clearly defined:

How can the cost of manufacturing for 2.5-axis CNC milling be implemented in existing topology optimization methods?

In order to answer this question, the principle of Design for Manufacturing is considered in Chapter 2 to understand what constitutes the costs for milling, and a cost estimation model is defined. Chapter 3 shows how the method was integrated in 5 steps by incrementally adding complexity and testing functionality with experiments. After the complete model has been implemented, Chapter 4 presents a validation of the results. Finally, the work is discussed and concluded in Chapter 5 and 6. Recommendations for future work are presented in the final Chapter 7.

Chapter 2
Design for Manufacturing

The importance of taking into account manufacturing when designing a part has already been highlighted in the introduction of this report. If this is not done, resulting designs might perform very well but are hard to manufacture, resulting in an unnecessarily expensive product. Sometimes this can not be avoided, especially when the requirements of a design are very challenging, but often a design can be modified to reduce manufacturing costs without sacrificing much of its functionality. Design for Manufacturing is the principle of taking these factors into account from the very start of a design cycle and was described by Poli [22].

The principle can be applied in all branches of manufacturing and some aspects are shared among many. Tight tolerances, exotic materials or manual labour are factors that increase cost in the majority of manufacturing processes. A closer look is taken at DfM for 2.5-axis CNC milling as this process is focussed on in this work. The goal of this chapter is to identify the properties of a design that determine its cost (and therefore suitability) of manufacturing using 2.5-axis CNC milling and to propose an estimation model that can be used in topology optimization.

2.1 DfM for 2.5-axis CNC milling

Two important terms are used when Design for Manufacturing is applied on 2.5-axis CNC milling, which will simply be called 'milling' from this point onwards. The first term is 'machinability' and this describes whether a design is possible to produce using this process [9], [13]. The limitations of milling can make it impossible for some designs to be machined and this term is used to indicate whether a design can be made or not.

Because this research is limited to 2.5-axis milling, any design that can be described using a 2-dimensional image is machinable. Internal voids, extreme overhangs or hidden geometry are examples of features that make a design non-machinable with milling, but these features can not be defined using a 2-dimensional image. So as long as a 2-dimensional description of the design is used in the optimization, the design will always be machinable. An example of such a 2-dimensional description of a 3-dimensional part is shown in the introduction of this report (Figure 1.2).

Previous work has mainly focussed on this aspect of manufacturing, ensuring manufacturability for all kinds of processes. Examples for casting, milling, turning and others are found in the work of Vatanabe et al. [26]. Langelaar presented a method to ensure manufacturability for milling when using a 3-dimensional description of the design [14]. While these efforts made optimization better suited to use in real applications, they only present machining constraints and do not explicitly provide a way of simplifying the manufacturing process in a way that will reduce the final cost of manufacturing.

The second term used in the context of DfM for milling is 'machinability-rating' [9], [13]. This term represents a non-binary quantity that indicates how challenging a design is to machine. When a design does not exceed the limitations of milling (the constraints are not violated), it does not make the design easy to machine immediately. Some relevant design factors that influence this machinability-rating are briefly reviewed:

- The internal radius of corners in a design determines the maximum tool radius that can be used to cut it. Having very small internal radii means that small tools are needed, which typically take more time to remove material. Larger internal radii allows the use of larger tools that can cut away more material in the same amount of time [17], [26].

- Another example of hard-to-machine features is thin walls. When such a wall is being machined, the lack of stiffness causes the wall to vibrate or deform under the tool pressure. This results in poor surface finish or even parts of the wall breaking out. Ensuring a minimum wall thickness

(dependent on the wall height) can ensure that these problems are avoided. If this is not ensured, special care has to be taken when machining these features by using specialized tools or reducing the cutting load [17], [26].

- The last example of a design feature that influences the machinability-rating is the depth and internal radius of pockets. When an internal radius is small, a smaller tool is required as described in the first point of this list. However, the depth of the pocket can add additional challenges when machining. When a deep pocket must be machined with tool of a small diameter, the rigidity of the tool itself becomes a concern. To avoid unwanted vibrations that influence the surface finish negatively, the cutting load must be reduced, which impacts the machining time and possibly tool life in a negative way [17].

These examples do not influence the machining in the same way. Small internal radii affect the required tools while thin walls or large depth/radius ratios affect the machining time of a part. In order to combine all these aspects into a single quantity that resembles the machinability-rating of a design, the cost of manufacturing is a suitable choice, as was mentioned in the introduction of this report as well. Needing more tools (and therefore more tool changes) or having to slow down the machining process will all translate to higher overall costs. Therefore, the cost is a suitable metric to generalize these effects and allows the integration of them into a single model. How the cost of machining can be determined is discussed in the following section.

2.2 Cost of machining process

The cost of machining is a summation of several factors. The most significant were identified by Dewhurst and Boothroyd as the machining time and the wear of tooling [6]. This first one is found by determining the machining time and multiplying it with a rate (usually expressed in euros per minute). The second term is determined by calculating the tool wear and taking into account the cost of replacing the tool when it reaches its end of life, which is a less significant factor in the total cost. For reasons of simplicity, the choice was therefore made to not include it in the cost estimation. Hence, in order to estimate the total cost, the machining time has to be determined.

CNC milling machines work by executing basic commands from a file which is generated by CAM (Computer-Aided Manufacturing) software. These commands can tell the machine to move the cutting tool to specific locations, set rotational speeds, change its tool to a different one, et cetera. With basic machine properties like maximum speed and tool change duration, the total machining time can be determined exactly by analysing the code without running the machine at all. Programming the design in CAM software is therefore the most accurate method of determining the machining time without actually producing the design [2]. A problem however is that this programming is done by a trained machine programmer and can not be included in an optimization algorithm for this reason. The solution is to develop a model that can estimate the machining time based on the design itself, without the need of an operator and to this end, it is important to first understand the machining process itself.

During CNC milling, the workpiece is clamped while the tool is rotating at high speed, cutting away unwanted material from the workpiece. This is achieved by letting the machine run programmed ‘operations’ using chosen ‘strategies’, where each strategy is used for a specific reason [17], [18]. Many strategies have been developed for different situations, but a small selection of them is used for the majority of work. This selection is listed below and illustrated in Figure 2.1.

Drilling: used for creating straight holes in a solid block by means of a drill (instead of a mill). This is one of the fastest methods of removing material, but drills wear down faster and can only by used when the full diameter is engaged with the material [17], [18].
Pocketing: used for creating cavities in a solid block by inserting the mill and removing material outwards by moving it in the horizontal plane. Modern CAM software can ensure constant tool engagement and cutting load, resulting in very high material removal rates and efficient cutting [18].
Contouring: used for creating vertical walls with a good surface finish. Often executed at lower cutting speeds to produce a high quality surface finish [18].
Slotting: used for creating narrow cavities by engaging the full width of the mill. Because this strategy engages the tool’s full width with the material, cutting depth and speed have to be adjusted in order to reduce cutting loads [18].

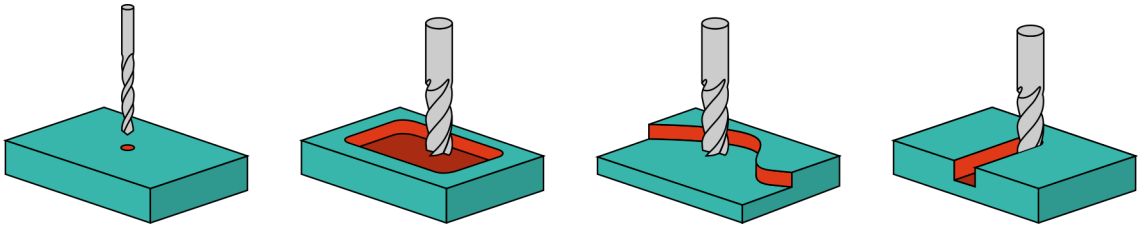


Figure 2.1: Illustrations of most used milling strategies. Left to right: drilling, pocketing, contouring and slotting. Areas being cut are marked in orange.

An important aspect when choosing a strategy is how the tool will be loaded. To maximize tool lifetime and reduce the chance of tool failure, tools should be used according to the specification of its manufacturer. The majority of mills are designed to be removing material along the full height of the cutting edges. This way of using tools is preferred by manufacturers because it wears the tool down in a predictable manner. Contouring and pocketing are both examples of strategies that allow this, while slotting does not. Because the full width of the mill is engaged with the material during a slotting operation, cutting loads on the mill are generally too high to engage the full height as well. Because of this, only the lower portion of the cutting edges are used and the mill wears down non-uniformly, which can cause unpredictable tool failures. Although slotting is considered one the most basic milling strategies, the unpredictability of it causes manufacturers to avoid the strategy whenever they can. The general machining process can therefore be described by the remaining three strategies: drilling, pocketing and contouring. The details of these strategies will be explored further in the next sections.

2.2.1 Drilling

While mills are designed to remove material while moving radially, drills are very efficient at removing material while moving axially. So-called centre-cutting mills can also cut while moving axially, but this wears the mill down in an unpredictable manner like the slotting strategy and is therefore avoided whenever possible. Drills wear down as well, but because their geometry is less complex, their cost is lower than the cost of a similarly sized mill. This makes drills a good choice for creating new holes that can serve as entry points for mills in following operations.

The cost of drilling can be estimated by the number of holes N_H multiplied by a constant machining rate R_D that includes moving the drill to the right location and drilling the hole:

$$C_D = N_H \cdot R_D. \tag{2.1}$$

A more accurate but complex estimation would also include the drilling diameter and hole depth, as both of these factors influence the drilling time of a hole. However, it is expected that the drilling time does not contribute much to the total machining time. Whether including this cost is useful to include is a question to be answered later, but using a more complex version is not expected to be necessary.

2.2.2 Pocketing

Most of the material can often be removed with an operation using the pocketing strategy. It allows to use the full cutting height of the mill, which increases process reliability and tool lifetime, while load on the tool is kept reasonable by decreasing the width of the cut. Initially inserting the mill into the material can create uneven wear of the tool, but this is avoided by pre-drilling the material in the right places. These entry points are only necessary when internal pockets are to be machined, because otherwise the mill can enter the material from the side. Toolpaths of pocketing operations often start in the centre of a pocket and spiral outwards, ensuring a constant load while cutting. A side-effect of these spiral-shaped toolpaths is however that vertical walls are not milled with a good surface finish, because the mill is cutting them piecewise instead of continuously. Contouring is therefore used to mill the vertical walls with a better surface finish.

The cost of pocketing can be determined by calculating the machining time and multiplying it with a constant machining rate R_P. A first order approximation of the total cutting time can be made by evaluating the total volume (or area, in case of a 2-dimensional design description) to be machined away:

$$C_P = R_P \cdot A_{\text{void}}. \tag{2.2}$$

This basic approximation assumes that the complete area can be machined away with a constant rate, but this is often not the case. Because the pocketing strategy starts cutting in the centre of the pocket and spirals outwards, only circular pockets can be cut at this perfect efficiency (as shown in the left pocket depicted in Figure 2.2). When the shape of the pocket is not circular, the spiralling motion has to be interrupted when the pocket wall is reached. From that point, the mill has to be moved to new position to start the next cut. This is known as 'air-milling' and during this time, no material is machined away and therefore less material is machined per unit time [18]. Figure 2.2 shows how a circular pocket will result in maximum cutting efficiency and a square pocket requires air-milling moves to mill its corners.

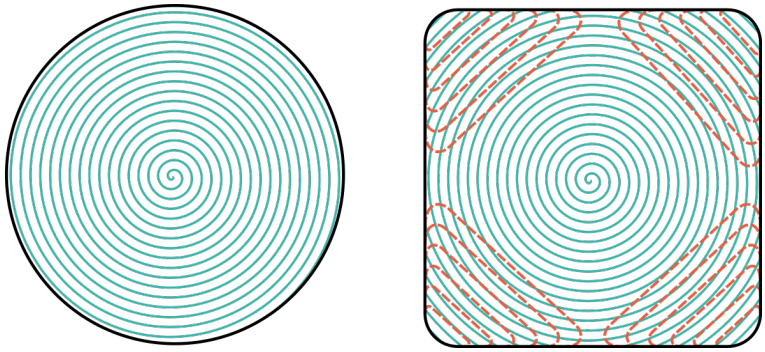


Figure 2.2: Illustrations of tool paths for a circular and square pocket. Blue lines indicate cutting moves, orange dotted lines indicate non-cutting moves.

In order to incorporate the effect of air-milling on the machining cost, the shape of each pocket should be taken into account. The goal of this addition is to make non-circular pockets more expensive to machine by increasing the machining time associated with them (to take air-milling into account). Additionally, a larger deviation from a circle should lead to a larger increase in machining time. A geometrical relationship between the perimeter and area of a shape called 'circularity' was identified as a potentially useful quantity. This circularity C of a shape is defined as the ratio between its perimeter P squared and its area A, as shown in Equation 2.3. However, to avoid ambiguity between costs and the circularity in mathematical notations, the circularity is renamed to 'shape factor' SF:

$$C = \mathrm{SF} = \frac{P^2}{4\pi A}. \tag{2.3}$$

By including the factor 4π in the denominator, the value of this quantity equals unity when a perfect circle is considered. The value increases as the shape deviates more from circular, which is the desired behaviour. With this quantity, the machining cost can be adjusted by multiplying it with the previous formulation of the pocketing cost (shown in Equation 2.2). This will produce the desired result of higher pocketing costs for non-circular pockets.

$$C_P = R_P \cdot C \cdot \mathrm{A_{void}} = R_P \cdot SF \cdot \mathrm{A_{void}}. \tag{2.4}$$

In order to validate whether the shape factor as defined above results in an accurate cost increase as the pocket shape deviates further from circular, the pocketing cost of a set of test pockets is determined and compared to the actual pocketing costs. These actual costs are found by programming the milling operation in CAM software. Because the shape factor results in an increase of costs that is relative to the cost determined based only on the void area, it is insensitive to machining parameters like cutting speed or material removal rate. Two sets of pocket shapes were designed, shown in Figure 2.3. The first set consists of square and triangular pockets with varying circularity. The second set consists of freely drawn pocket shapes. This set is used to check how the method behaves when highly non-circular shapes are considered.
To visualize the findings using the first set of pockets, the machining times found with CAM are normalized using the machining time of the perfectly circular pocket. The shape factor of this pocket equals unity and because the shape factor will by used as a scaling factor, the normalized machining times and the shape factors can be compared directly.

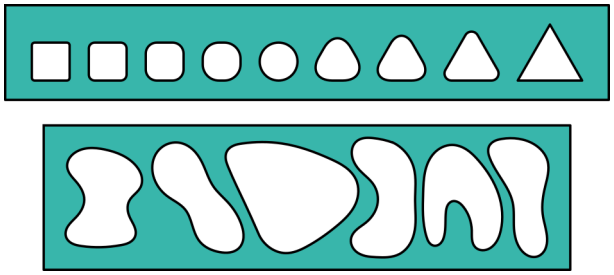


Figure 2.3: Pockets used for validating the shape factor as a method of increasing pocketing cost based on pocket shape.

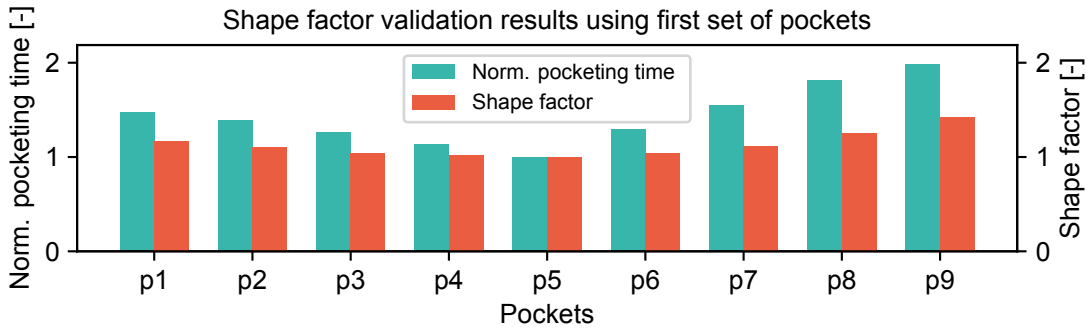


Figure 2.4: Shape factor validation results using first set of pockets. Pockets p1 to p9 correspond to the pockets shown in the top of Figure 2.3, counting from left to right.

To visualize the results for the second set of pockets, normalization of the machining time is not possible because no circular reference pocket is available. Instead, the pocketing time found with CAM is divided by the pocket area. If the estimation works correctly, this value should scale linearly with the shape factor of the pocket. The values are plotted in Figure 2.5, in which the values of the pocketing times divided by the areas are included in the caption as well for clarity.

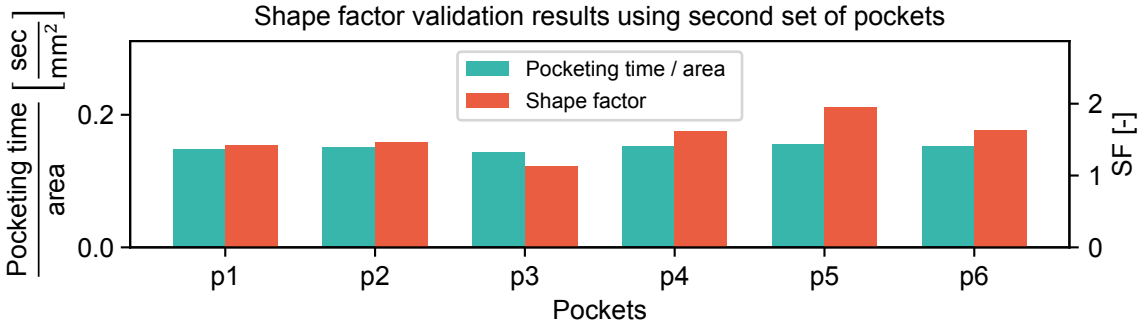


Figure 2.5: Shape factor validation results using second set of pockets. Pockets p1 to p6 correspond to the pockets shown in the bottom of Figure 2.3, counting from left to right. The values of the pocketing times divided by the areas are 0.149, 0.150, 0.144, 0.152, 0.156 and 0.152 for pockets p1 to p6 respectively.

When looking at the results plotted in Figure 2.4, some conclusions can be drawn. First of all, the machining times behave as expected. The circular pocket has the smallest pocketing time, which translates to the lowest cost. As the pocket shapes deviate further from circular, the pocketing time increases. The shape factor shows the same behaviour, but the values do not increase with the same amount as the normalized pocketing times. For example, while pocket 9 (rightmost triangular pocket) has a normalized pocketing time of 1.99, its shape factor is only 1.42.

The results of the second set of pockets (Figure 2.5) show a different behaviour. The results indicate that the shape factor is too low for pockets that are close to circular (which was also concluded based on the results of the first set of pockets), but too high for pockets with high shape factor (all pockets in the second set except for p3).

Summarizing, the shape factor manages to capture the trend of increasing pocketing times for pockets with shapes close to circular. However, the shape factor is generally too low for pockets close to circular which results in a cost estimation that is too low. When pockets are highly non-circular, the shape factor raises faster that the machining time, resulting in a cost estimation that is too high. In all cases, non-circular pockets result in higher costs, where increase of cost is dependent on the non-circularity, which was the original goal. Therefore, the decision was made to use the shape factor as described in Equation 2.4. A function might be developed to transform the shape factor and make the estimation more accurate, but for reasons of simplicity and time, this was not further investigated in this research.

2.2.3 Contouring

After the pocketing operation has finished removing the majority of material, the vertical walls have poor surface quality, which is usually not acceptable for the finished product. Contouring is used to improve this by cutting the walls in one continuous motion. It allows usage of the full cutting height, which makes it a suitable strategy for manufacturers. The cost of contouring can be estimated by evaluating the total length of the toolpath of the contouring operation. Because this operations is used as a finishing step, it can be assumed that walls are finished in a single pass. This means that the total length is equal to the total perimeter P of the design, when described in a 2-dimensional image. Multiplying this length by a machining rate for contouring R_C, again dependent on the machine and tool parameters, an estimation of the contouring cost can be made:

$$C_C = R_C \cdot P. \tag{2.5}$$

In this equation, it is assumed that the height of every wall is low enough such that it can indeed be cut in a single pass. In some components, the height of the wall to be finished might be larger than the height of the cutting edge, which means that multiple passes are required to finish the complete wall. This effect is not included in this estimation for reasons of simplicity, but an implementation could be rather straight-forward by adjusting contouring rate R_C. For example, when considering a pocket with a depth that requires two passes, the value of R_C could be doubled to account for this extra pass.

2.2.4 Total machining cost

With the cost of the three basic operation defined as shown above, the total machining cost of a part is estimated by summing the three operation costs together:

$$C_M = C_D + C_P + C_C. \tag{2.6}$$

The significance of each component is determined by the machining rates (R_D, R_P and R_C). The values for contouring and pocketing used throughout the rest of this work are determined based on a situation where both operations are performed with the same tool. To calculate the contouring and pocketing rate, a constant general machining rate R_M is chosen as 1 euro per minute. The pocketing rate can then be calculated by dividing this machining rate by the material removal rate expressed in a unit area per minute:

$$R_P = \frac{R_M}{\mathrm{MRR}_P} = \frac{R_M}{V_P \cdot W} = \frac{R_M}{V_P \cdot 0.1D}. \tag{2.7}$$

The material removal rate MRR_P is calculated by multiplying the cutting speed for pocketing V_P with the width of the cut W, which is typically 10% of the tool diameter D for pocketing [17], [18]. The same evaluation can be made for the contouring cost, where the material removal rate is simply the cutting speed V_C. Because contouring is a finishing operation, the cutting speed is typically half the speed used for pocketing [17], [18]. This results in the following evaluation:

$$R_C = \frac{R_M}{\mathrm{MRR}_C} = \frac{R_M}{V_C} = \frac{R_M}{0.5 \cdot V_P}. \tag{2.8}$$

The ratio between the pocketing and contouring rates can now be determined:

$$\frac{R_P}{R_C} = \frac{R_M}{V_P \cdot 0.1D} \Big/ \frac{R_M}{0.5 \cdot V_P} = \frac{0.5}{0.1D}. \tag{2.9}$$

It is clear that by using the assumptions stated above, the ratio is only dependent on the tool diameter. Because only a single tool is considered in this research, this diameter is set to be 1, which means that the pocketing rate is five times the contouring rate. Hence, the values 5.0 and 1.0 are used for R_P and R_C respectively throughout the rest of this work. If a future extension would include the use of multiple tools, these values should be determined for each tool, such that accurate significance is achieved. The drilling rate is finally set at 10.0. This value depends on the machine parameters such as maximum travel speed but also on the defined thickness of the design. Later validation will check whether this value is scaled correctly with respect to the other rates.

Chapter 3
Development of Cost Functions

With the basic machining operations as defined in Chapter 2, the cost of these operations can now be defined in a way that can be implemented in an optimization algorithm. This will be done in steps to ensure the proper function of each element before adding more complexity to the cost function. Each step will be verified with experiments which show the influence of what has been added to the method.

3.1 Baseline Procedure

Before adding any machining cost estimation to the optimization algorithm, the starting point should first be clearly defined. As discussed in Chapter 1, the choice was made to use a density-based model with the SIMP modification to prevent grey values in the final design. In order to combat the checkerboarding effect and mesh-dependency, a density filter is applied to the domain with design variables ($\mathbf{D}$) with a radius of 1.5 times the element size. Because the density filter ensures that no sharp transitions from void to solid can occur, the resulting design will have blurry edges. A final Heaviside projection is therefore applied to create the final density domain ($\boldsymbol{\rho}$) with less intermediate values, which will be used to determine the compliance and volume properties. Although compliance-only optimization does not rely on the Heaviside projection to yield usable results, it was found that including the projection step improved the results in the following experiments. For reference, the implementation of the density filter and the smooth Heaviside projection is included in Appendix A.

The procedure to create the density domain from the design domain are illustrated in Figure 3.1 and can be mathematically described as well:

$$\mathbf{F} = \mathrm{DF}(\mathbf{D}), \tag{3.1}$$

$$\boldsymbol{\rho} = \mathrm{HS}(\mathbf{F}). \tag{3.2}$$

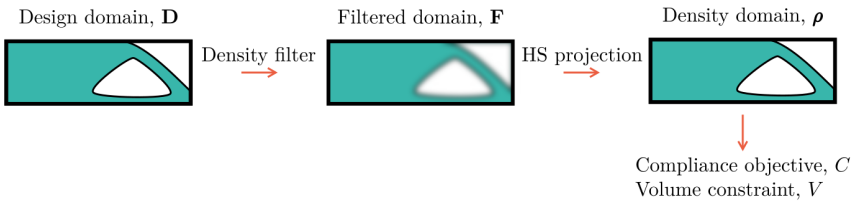


Figure 3.1: Illustration of baseline procedure for optimization. Design domain $\mathbf{D}$ is filtered to create filtered domain $\mathbf{F}$. A smooth Heaviside projection is used on the filtered domain to create the density domain $\boldsymbol{\rho}$, which is then used to evaluate the compliance and volume properties.

3.2 Basic Pocketing and Contouring

The first step to a complete implementation of a cost estimation as described in the previous chapter is to start with a basic implementation of pocketing and contouring costs. Only after this basic version of machining cost estimation has been implemented successfully, more complex features can be added to the total cost estimation. This section explains these basic implementations of pocketing and contouring and their effect on the resulting design is investigated with an experiment.

3.2.1 Pocketing

Chapter 2 describes how the shape of a pocket influences the machining time. However, a first-order approximation of this machining time can be made without including the pocket shape, as will be shown

in this first experiment. The cost is simply determined by multiplying the total area to remove by the pre-defined machining rate for pocketing (R_P). The total void area can be calculated by summing 1 minus the element's density, multiplied by the element's area. With a generic domain $\mathbf{x}$ as an input, the function A_{void} is defined as such:

$$\mathsf{A}_{\text{void}}(\mathbf{x}) = \sum_{i=0}^{N} [A_i \cdot (1 - x_i)] = A_{elem} \sum_{i=0}^{N} (1 - x_i)\,. \tag{3.3}$$

If a regular mesh is used, it means that the areas of all elements are equal and the element-specific quantity A_i can be generalized to the constant A_{elem} and taken out of the summation (also shown in Equation 3.3). This simplification is used for all coming experiments because only regular meshes were used.

Its sensitivity with respect to the input variables x_i, which will be needed to implement this function in a gradient-based update scheme, can be found by differentiating the function to the input variables. If a regular mesh is used (like in all the following experiments), this gradient becomes independent of the input variable considered:

$$\frac{\partial \mathsf{A}_{\text{void}}(\mathbf{x})}{\partial x_i} = -A_{elem}. \tag{3.4}$$

With this function to determine the void area of a domain, the pocketing cost can be defined as this void area, multiplied with the machining rate for pocketing R_P:

$$C_P(\mathbf{x}) = R_P \cdot \mathsf{A}_{\text{void}}(\mathbf{x}). \tag{3.5}$$

Pocketing rate R_P is defined as a cost per unit area (i.e. $€/mm^2$) and is given as a constant based on machine parameters, tool specifications, et cetera. The sensitivity of the pocketing cost C_P is found by differentiating to the input variables again:

$$\frac{\partial C_P(\mathbf{x})}{\partial x_i} = R_P \frac{\partial \mathsf{A}_{\text{void}}(\mathbf{x})}{\partial x_i} = -R_P \cdot A_{elem}. \tag{3.6}$$

When looking at the sensitivity of the pocket cost, some interesting properties can be observed. Due to the $1-x_i$ which is used in the void area calculation, a minus sign appears in the sensitivity. Furthermore, because machining rate R_P and element area A_{elem} will always have positive values, the sign of the sensitivity will always be negative. This means that an increase of a value in input domain $\mathbf{x}$ will always lead to a decrease in pocketing cost, which makes intuitive sense (more solid elements equals less material to mill away). It should be noted that factors like minimum tool radii or wall thicknesses are not considered at this point. If more complex factors like these are included, an increasing value in the input domain could increase pocketing cost as well, as this change might force the use of a smaller tool or a lower cutting speed.

It can also be observed that the sensitivity is independent of the input variables themselves. This is an opportunity for minimizing the computational effort, as the sensitivity can be computed once and re-used for the rest of the optimization process. However, since the computation of this sensitivity is very simple, the savings are expected to be minimal and this possibility of time-saving was not quantified.

When the pocketing costs are determined based on the density domain $\boldsymbol{\rho}$, the sensitivity defined in Equation 3.6 can be used to find the sensitivity with respect to the density elements ρ_i. However, the optimizer needs the sensitivities with respect to the design variables D_i. These are found by including the sensitivities of the smooth Heaviside projection ($\mathsf{HS}(\mathbf{x})$) and the density filter ($\mathsf{DF}(\mathbf{x})$) as such:

$$\frac{\partial C_P(\boldsymbol{\rho})}{\partial D_i} = \frac{\partial C_P(\boldsymbol{\rho})}{\partial \rho_i} \frac{\partial \rho_i}{\partial F_i} \frac{\partial F_i}{\partial D_i} = \frac{\partial C_P(\boldsymbol{\rho})}{\partial \rho_i} \frac{\partial \mathsf{HS}(\mathbf{F})_i}{\partial F_i} \frac{\partial \mathsf{DF}(\mathbf{D})_i}{\partial D_i}. \tag{3.7}$$

With the pocketing cost and its sensitivity defined, it is ready to be implemented in an optimization algorithm.

3.2.2 Contouring

The cost of contouring can be implemented as described in Chapter 2 without any simplifications as the perimeter is the only quantity in this formulation that depends on input domain $\boldsymbol{\rho}$. The machining rate for contouring R_C is expressed as a unit cost per unit length, i.e. $€/mm$. To determine the perimeter of a discretized domain, the Total Variance function is used, which evaluates the perimeter by summing local gradients in 8 different directions (TV_8, defined in Equation C.4) of all elements in the domain. Why this function was chosen and how it is implemented is further detailed in Appendix C. With the contouring rate and perimeter function, the cost of contouring C_C can be defined:

$$mC_C(\boldsymbol{\rho}) = R_C \cdot \text{Perim}(\boldsymbol{\rho}) = R_C \cdot TV_8(\boldsymbol{\rho}). \tag{3.8}$$

When calculating the perimeter of a domain using the TV_8 method, a padding must be applied around the domain to allow the elements on the domain's boundary to be included in the calculation. The value assigned to these elements in the padding have an effect on the outcome of the perimeter calculation. Figure 3.2 shows the difference in calculated perimeters between a padding value of 0 and 1. Since the perimeter shown in the right figure is the perimeter which will actually be machined, a padding value of 0 is used in the TV_8 calculation.

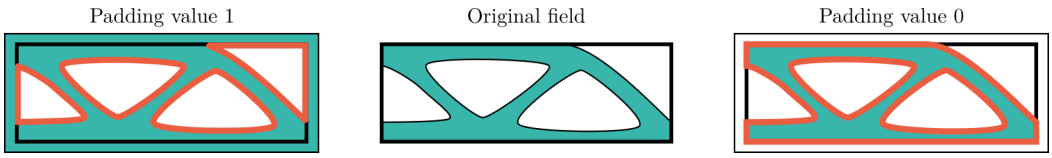


Figure 3.2: Input domain in the centre with two padded versions left and right of it. The left padding is filled with a value of 1 and the right padding is filled with value 0. The resulting perimeters are shown in orange, indicating the effect of the padding value on the calculated perimeter.

The sensitivity of the contouring cost with respect to the input variables x_i is equal to the sensitivity of the TV_8 function, multiplied by machining rate R_C:

$$\frac{\partial C_C(\mathbf{x})}{\partial x_i} = R_C \cdot \frac{\partial TV_8(\mathbf{x})}{\partial x_i}. \tag{3.9}$$

When used in topology optimization, the sensitivities with respect to the design variables are required, so the same steps are used as was done for the pocketing cost sensitivity to find these:

$$\frac{\partial C_C(\boldsymbol{\rho})}{\partial D_i} = \frac{\partial C_C(\boldsymbol{\rho})}{\partial \rho_i} \frac{\partial \text{HS}(\mathbf{F})_i}{\partial F_i} \frac{\partial \text{DF}(\mathbf{D})_i}{\partial D_i}. \tag{3.10}$$

3.2.3 Total machining cost

The total machining costs C_m are found by summing the pocketing and contouring costs together:

$$C_m = C_P + C_C \tag{3.11}$$

However, when they are to be combined with the compliance objective, both values should be in the same order of magnitude. To achieve this, reference values are defined for both the compliance and the machining costs by evaluating these quantities for the initial design. During the optimization, both objectives are normalized with these reference values before they are combined. A weight factor is also used to choose the significance of compliance and machining cost. The machining cost weight factor w_{mc} is multiplied with the machining cost C_m and $1 - w_{mc}$ is multiplied with the compliance C before adding them to the final objective F:

$$F = w_{mc} \cdot C_m + (1 - w_m c) \cdot C. \tag{3.12}$$

3.2.4 Experiments

To see the effects of the pocketing and contouring cost estimation on the optimized design, two experiments are performed. First, only contouring costs are taken into account in the machining cost estimation. In order to reduce the contouring costs and optimize this objective, the total perimeter of

the design should be minimized. This means that the outcome is expected to have its corners rounded more, be compacter and have less internal pockets compared to a design optimized for compliance only.

In the second experiment, the pocketing costs are added to the machining cost estimation. Because the pocketing costs are not dependent on the shape of the design, no topological changes are expected compared to the first experiment. However, because the addition of pocketing costs makes the total machining costs higher than before, experiments with the same relative weight factor as in the first experiments will likely not result in the same designs.

Experiment 1A: Only contouring cost estimation

The settings for this experiment are shown in Table 3.1. Six runs have been done with different weight factors for the machining cost estimation. Together with the reference design, these results are presented in Figure 3.3 with the values of the compliance and machining cost objectives shown in Figure 3.4.

Table 3.1: Optimization parameters for experiment 1A.

Optimization parameters for experiment 1A			
Domain dimensions:	20x30	SIMP factor:	3.0
Mesh size:	160x240	Contouring rate, R_C:	1.0
Load case:	MBB-beam		
Volume limit:	60%		

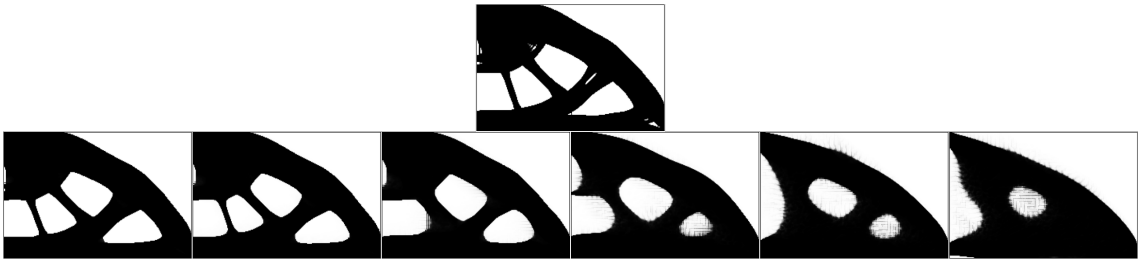


Figure 3.3: Results of experiment 1A using only basic contouring cost estimation. The top image is the reference result when no cost estimation is included (only compliance is minimized). Designs in the bottom row are generated with cost estimation included, with the relative weight for the cost objective increasing from left to right: 2%, 5%, 10%, 30%, 50% and 70%.

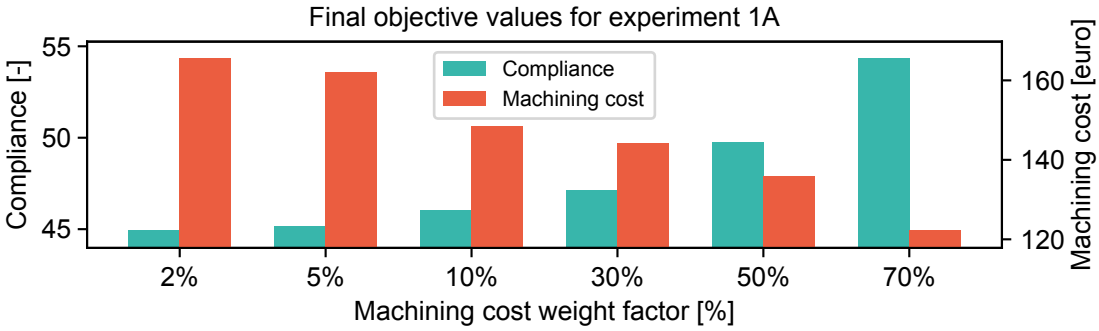


Figure 3.4: Values of the compliance and machining cost objectives after optimizations are finished for experiment 1A.

Experiment 1B: Contouring and pocketing cost estimation

The settings for this experiment are shown in Table 3.2. In this experiment, pocketing costs are added and the rates are set to the values found in Chapter 2. The results are presented in Figure 3.5 with the values of the compliance and machining cost objectives shown in Figure 3.6.

Table 3.2: Optimization parameters for experiment 1B.

Optimization parameters for experiment 1B			
Domain dimensions:	20x30	SIMP factor:	3.0
Mesh size:	160x240	Contouring rate, R_C:	1.0
Load case:	MBB-beam	Pocketing rate, R_P:	5.0
Volume limit:	60%		

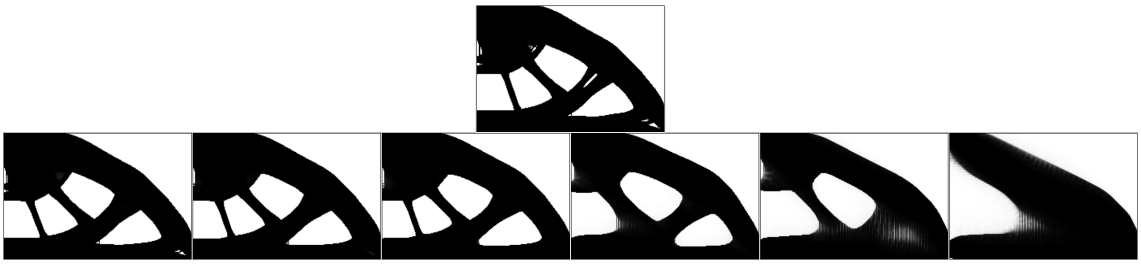


Figure 3.5: Results of experiment 1B using basic contouring and pocketing cost estimation. The top image is the reference result when no cost estimation is included (only compliance is minimized). Designs in the bottom row are generated with cost estimation included, with the relative weight for the cost objective increasing from left to right: 2%, 5%, 10%, 30%, 50% and 70%.

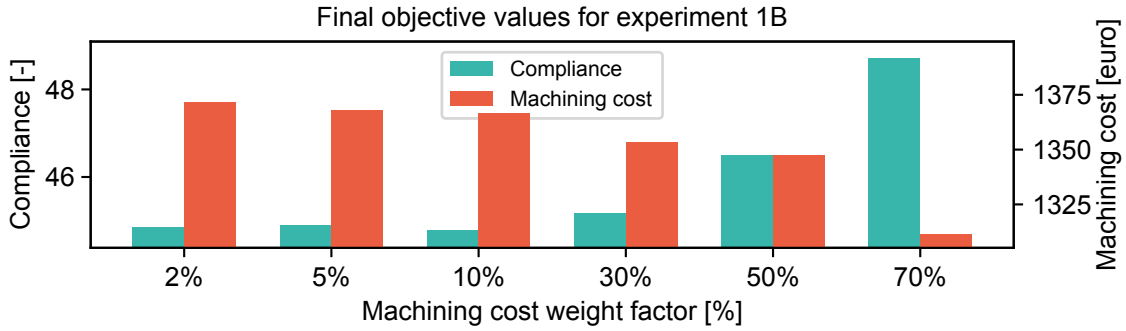


Figure 3.6: Values of the compliance and machining cost objectives after optimizations are finished for experiment 1B.

3.2.5 Conclusions

When looking at the generated designs, a couple of conclusions can be drawn. The most notable effect of the machining cost objective is that less internal pockets end up in the optimized design. Internal pockets cause the perimeter of the domain to be large, which causes high contouring costs. Hence, removing internal pockets is an efficient way of reducing these costs and thereby the total machining cost as well. When the machining cost weight is increased, more emphasis is put on minimizing the machining cost, so more internal pockets are removed. When the machining cost weight factor is pushed to 90%, experiment 1B reaches a solution without any internal pockets.

A second observation is that all corners, both internal and external, are getting rounded by the addition of the machining cost estimation. This was expected as well, because a rounded corner will always lead to a lower perimeter and therefore a lower machining cost.

When comparing experiment 1A with 1B, it can be seen that the addition of the pocketing cost does not change the results drastically. All the effects seen in experiment 1A can be seen in 1B as well. This was expected as the pocketing cost is not dependent on the geometry of the design, so it should also not influence it in the optimization. Because the pocketing cost does add to the total cost, the results at identical machining cost weight factors are not exactly the same.

When looking closely at the boundary of the designs of experiment 1A, especially at high machining cost weight factors, it appears to be jagged and with many intermediate values, which is not desired and unexpected. Having such jagged edges should result in a high perimeter and therefore high contouring

costs. This is explained by the fact that jagged edges with intermediate values result in artificially low perimeter calculations. The intermediate values are allowed because the weight factor for the compliance objective (which uses the SIMP method to avoid intermediate values) is very low. The same effect is visible in experiment 1B, albeit less pronounced as in 1A. A solution to this problem could be to introduce penalization in the machining cost estimation as well. This idea will be investigated in the next section.

3.3 Penalized Pocketing and Contouring

As explained in the previous section, when using the most basic cost estimation model, grey densities appear in the generated design as the cost importance is increased. The primary mechanism to prevent these grey values from appearing is integrated in the compliance cost objective in the form of the SIMP technique. It was concluded that if a similar mechanism could be integrated in the cost estimation model, this could help the optimizer to converge on a design without grey density values.

Since the machining cost objective consists of two elements (pocketing and contouring costs), both should penalize grey densities to maximize the effect. The proposed penalization method for pocketing is very similar to the SIMP technique in that it raises the density values to a power p before calculating the void area. After this step, values that were between 0 and 1 are lowered, which results in higher pocketing costs (because there is more void area to machine). To implement this penalization, a modified version of the void area function A_{void} is defined:

$$\mathsf{A}_{\text{void}}^{\text{penal}}(\mathbf{x}) = \sum_{i=0}^{N} \left[A_i \cdot \left(1 - x_i^p\right)\right] = A_{elem} \sum_{i=0}^{N} \left(1 - x_i^p\right). \tag{3.13}$$

This function is differentiated with respect to the input variables to find its sensitivity (with the assumption of a regular mesh where all element areas are equal):

$$\frac{\partial \mathsf{A}_{\text{void}}^{\text{penal}}(\mathbf{x})}{\partial x_i} = -A_{elem} \cdot p \cdot x_i^{p-1}. \tag{3.14}$$

To apply penalization to the pocketing cost calculation, the void area function A_{void} in Equation 3.5 can be replaced by the newly defined $\mathsf{A}_{\text{void}}^{\text{penal}}$:

$$C_P(\mathbf{x}) = R_P \cdot \mathsf{A}_{\text{void}}^{\text{penal}}(\mathbf{x}). \tag{3.15}$$

Note that by using the penalized version of the void area function, the sensitivity of the pocketing cost is now dependent on local density values (as opposed to the original formulation shown in Equation 3.5).

Penalization for contouring cost is not as straight-forward as for the pocketing cost. Where the value of a single element determines its contribution to the pocketing cost, contouring cost is driven by the difference between elements. Hence, in order to penalize grey values in contouring, the difference between elements with grey values should be increased. The smooth Heaviside projection can be used to achieve this, as it pushes grey values to either 0 or 1. The penalized version of the contouring cost function then looks as follows:

$$C_C(\mathbf{x}) = R_C \cdot TV_8(\mathsf{HS}(\mathbf{x})). \tag{3.16}$$

The smooth Heaviside projection uses a smoothness constant β which determines how much the grey values are pushed to 0 and 1. By increasing β, grey values are changed by a larger amount, which corresponds to a larger penalization. Therefore, β can be used as the penalization factor for the contouring cost. Note that the addition of the smooth Heaviside projection requires its sensitivity to be taken into account in the sensitivity of the penalized contouring cost as well.

3.3.1 Experiments

With the adjustment made to the pocketing cost estimation, a new experiment can be done to verify its functionality. To make a good comparison between the previous experiment and this one, all parameters such as the load case used, the number elements, SIMP factor and machining rates remain the

same. The newly introduced penalization factor on the pocketing cost is set to 3.0, which is the same value as the SIMP factor. It is expected that since this penalization works in a very similar way, the value of the standard SIMP factor of 3.0 is a good starting point. The penalization factor for contouring (β) is set to 4.0.

Experiment 2A: Penalized contouring
The parameters for this experiment are shown in Table 3.3, which are the same as the ones used in experiment 1A, but now the penalization factors are added. Figure 3.7 shows the results of this experiment.

Table 3.3: Optimization parameters for experiment 2A.

Optimization parameters for experiment 2A			
Domain dimensions:	20x30	Contouring rate, R_C:	1.0
Mesh size:	160x240	Contouring penalization, β:	4.0
Load case:	MBB-beam		
SIMP factor:	3.0		
Volume limit:	60%		

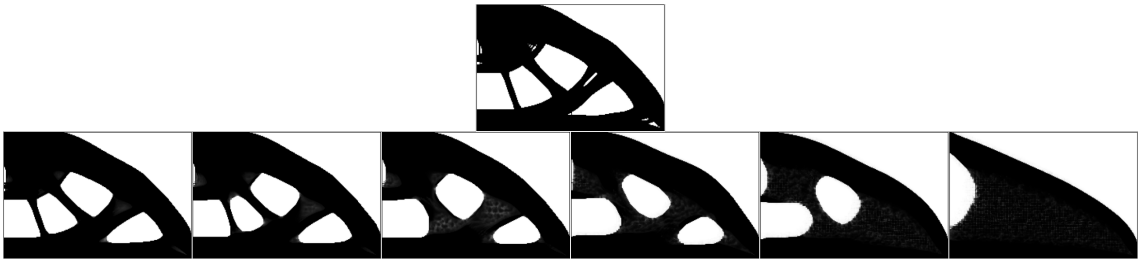


Figure 3.7: Results of experiment 2A. The top image shows the reference design which has only been optimized for compliance. The bottom row shows the results with machining cost included in the optimization. From left to right, the relative weights for the machining cost are: 2%, 5%, 10%, 30%, 50% and 70%.

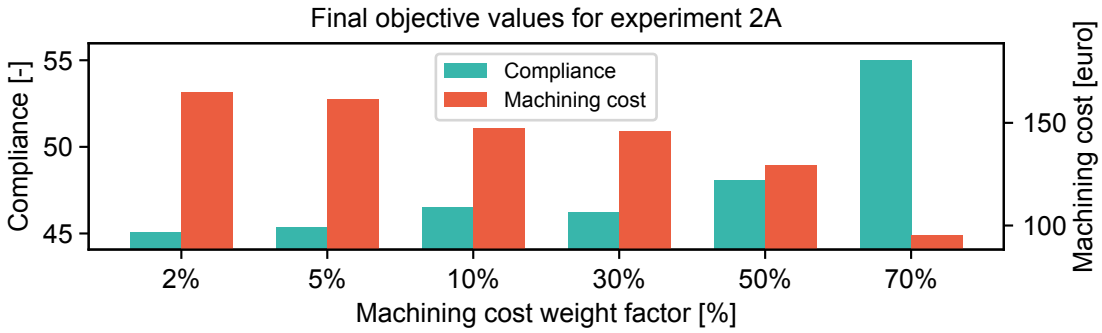


Figure 3.8: Values of the compliance and machining cost objectives after optimizations are finished for experiment 2A.

Experiment 2B: Penalized contouring and pocketing
The second experiment uses the same parameters as Experiment 1B, with the addition of the penalization factors. All parameters are shown in Table 3.4 with the results of the optimization runs presented in Figure 3.9.

3.3.2 Conclusions
When looking at experiment 2A and comparing it to the results of experiment 1A, the jagged edges with intermediate values have not completely disappeared, but they are much less pronounced. This

Table 3.4: Optimization parameters for experiment 2B.

Optimization parameters for experiment 2B			
Domain dimensions:	2x3	Contouring rate, R_C:	1.0
Mesh size:	160x240	Contouring penalization, β:	4.0
Load case:	MBB-beam	Pocketing rate, R_P:	5.0
SIMP factor:	3.0	Pocketing penalization, p:	3.0
Volume limit:	60%		

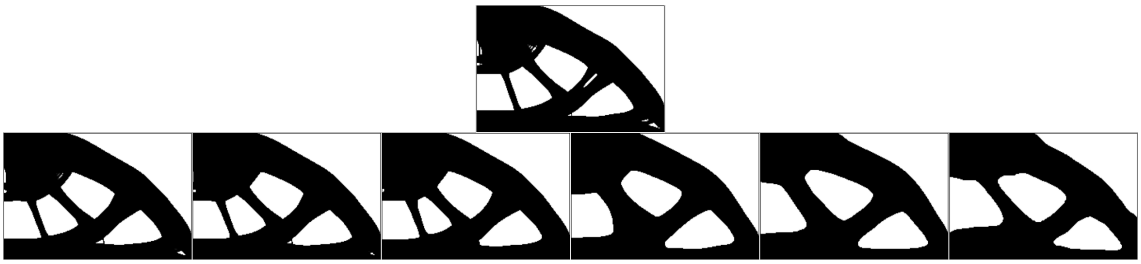


Figure 3.9: Results of experiment 2B. The top image shows the reference design which has only been optimized for compliance. The bottom row shows the results with machining cost included in the optimization. From left to right, the relative weights for the machining cost are: 2%, 5%, 10%, 30%, 50% and 70%.

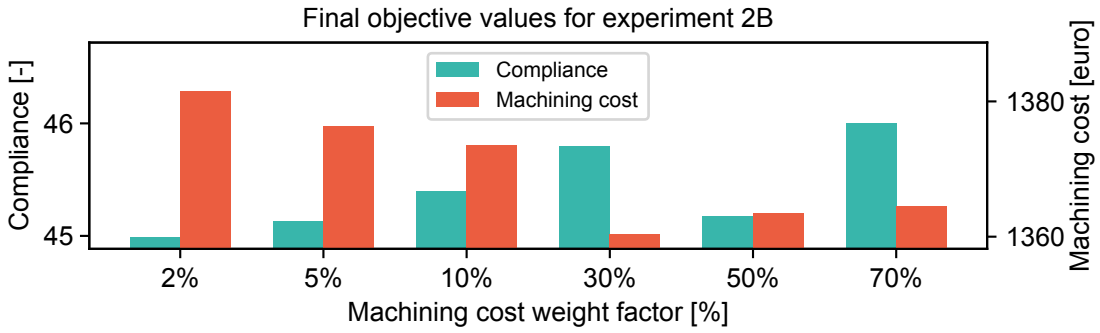


Figure 3.10: Values of the compliance and machining cost objectives after optimizations are finished for experiment 2B.

indicates that the penalization method helps to reduce this effect and improves the results. When comparing the results with the highest machining cost weight factor, the penalization has caused the ill-defined internal pocket from experiment 1A to disappear and result in a solid design without any internal pockets.

However, when looking closely at the solid regions in experiment 2A, a negative effect can be seen as well. Where the values in the solid regions of experiment 1A en 1B were equal to 1, in these results lower values can be found. This can be explained by the smooth Heaviside projection that is used to penalize intermediate values. While the penalization mechanism causes higher perimeters to be calculated for intermediate values around 0.5, the same projection also pushes values which were already close to 0 and 1 even further to those limits. In other words, by using the values slightly smaller than 1 in the density domain, the optimizer can save material that can be used in other locations. Because the Heaviside projection is applied before the contouring cost is determined, these regions of intermediate values barely contribute any contouring costs. This effect is weaker at low machining cost weight factors, because the SIMP mechanism is making intermediate densities inefficient. When the machining cost weight factor increases, the SIMP mechanism becomes less significant and the intermediate values start appearing in the solutions. When pocketing is included in the machining cost (in experiment 2B), the intermediate values disappear. This can be explained by the penalization mechanism of the pocketing cost calculation, which makes the use of intermediate values inefficient, just like the SIMP method.

Experiment 2B shows other unexpected results as well. When the machining cost weight factor is relatively low, the method behaves well and produces nice results. However, when the machining cost weight factor is increased to 70%, unexpected shapes appear in the results. Why these wavy shapes appear becomes clear when looking at how the design evolves throughout the iterations. In the first iteration, all design variables are equal to the volume limit of 0.6. During the first few iterations, when this domain full of intermediate values starts to crystallize into regions of void and solid phase, the optimizer very rapidly converges to the design as shown in Figure 3.9. With machining cost weight factors this high, it appears that the penalization creates a local minimum in the objective that the optimizer gets stuck in. This problem could be solved in numerous ways. A lower penalization factor could solve the issue, but this could also result in the return of intermediate values in solid and void regions. A different initialization could also avoid the optimizer to get stuck in such a local minimum and this method will be used in later experiments as well. A final solution could be to first let the optimizer iterate the design without penalization enabled. After reaching a certain level of convergence, the penalization factor could be increased to get rid of intermediate values without getting stuck in an undesired local minimum. This last option is used for the next experiment to reduce this unwanted effect. In addition, lower penalization factors are also used.

3.4 Shape factor

In Chapter 2, it was discussed how the cost of external pockets is different from the cost of internal ones. Internal pockets can be cut more efficiently when their shape is as round as possible because this minimizes the amount of air-milling. The compactness property (or 'shape factor') was introduces as a way to scale the costs according to the shape of the pocket. This section explains how the shape factor is implemented in the machining cost estimation model and its effects are shown using an experiment.

Because the scaling should only apply to internal pockets, the first step is to isolate the internal pocket from the input domain. A suitable method to this end is found in the 2019 paper by Langelaar, 'Topology optimization for multi-axis machining' [14]. In this paper, a milling filter (MF($\mathbf{x}$)) is defined which can be used to eliminate internal voids from a domain by combining cumulative sums in different directions. How this filter exactly works is further explained in Appendix D.

In order to use the milling filter to isolate the internal pockets in a domain, a couple of steps are required, shown in Figure 3.11. First, the input domain $\mathbf{P}$ is filtered with the milling filter, which yields a version of the input domain with its internal pockets removed, $\mathbf{P}^{\text{ext}}$:

$$\mathbf{P}^{\text{ext}}(\mathbf{P}) = \text{MF}(\mathbf{P}). \tag{3.17}$$

The next step is to create the internal pocket domain $\mathbf{P}^{\text{int}}$. Pockets that belong to this domain are the ones present in the original pocket domain $\mathbf{P}$, but that disappeared in the external pocket domain $\mathbf{P}^{\text{ext}}$ by means of the milling filter. Mathematically, this can be done by starting with a completely solid domain, adding $\mathbf{P}$ and subtracting $\mathbf{P}^{\text{int}}$ from it and this operation will be referred to as the 'inverted difference':

$$\mathbf{P}^{\text{int}}(\mathbf{P}) = 1 + \mathbf{P} - \mathbf{P}^{\text{ext}}(\mathbf{P}). \tag{3.18}$$

When the internal pocket domain is used to compute objective values later, the sensitivity of the internal pocket domain elements with respect to the original pocket domain elements is needed. This is found by differentiation:

$$\frac{\partial \mathbf{P}^{\text{int}}(\mathbf{P})}{\partial P_i} = 1 - \frac{\partial \mathbf{P}^{\text{ext}}(\mathbf{P})}{\partial P_i} = 1 - \frac{\partial \text{MF}(\mathbf{P})}{\partial P_i}. \tag{3.19}$$

The complete procedure to create the external and internal pocket domains from the original pocket domain is illustrated in Figure 3.11, which shows how the domains would look like with a simple geometry as pocket domain input.

With this procedure, pocketing costs can now be split into two separate costs: external and internal pocketing costs ($C_{\text{P,ext}}$ and $C_{\text{P,int}}$ respectively). The cost of pocketing external pockets and its sensitivity are the same as defined in the section (Equation 3.5 and 3.6), namely the machining rate R_P multiplied with penalized void area, but this is now calculated based on the external pocket domain which is obtained using the milling filter:

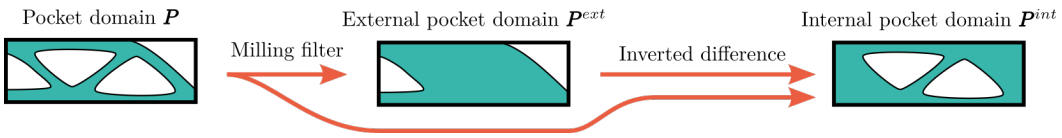


Figure 3.11: Procedure for separating internal pockets from internal pockets. The milling filter is used to generate $\mathbf{P}^{\text{ext}}$ from $\mathbf{P}$. The inverted difference operator is then used to generate $\mathbf{P}^{\text{int}}$.

$$C_{\text{P,ext}}(\mathbf{P}^{\text{ext}}) = R_P \cdot \text{A}_{\text{void}}^{\text{penal}}(\mathbf{P}^{\text{ext}}) = R_P \cdot \text{A}_{\text{void}}^{\text{penal}}(\text{MF}(\mathbf{P})). \tag{3.20}$$

The sensitivity of the external pocketing cost with respect to the pocket domain is found by first differentiating to the external pocket domain elements and multiplying this with the sensitivity of the external pocket domain with respect to the original pocket domain elements:

$$\frac{\partial C_{\text{P,ext}}(\mathbf{P}^{\text{ext}})}{\partial P_i} = R_P \frac{\partial \text{A}_{\text{void}}^{\text{penal}}(\mathbf{P}^{\text{ext}})}{\partial P_i^{\text{ext}}} \frac{\partial \mathbf{P}^{\text{ext}}(\mathbf{P})_i}{\partial P_i}. \tag{3.21}$$

Internal pockets have an additional factor in the pocketing cost estimation, namely the shape of the pocket. The shape factor SF has already been defined in Chapter 2 (Equation 2.3), but to make the function usable in the optimization, the perimeter function is replaced with the TV_8 function and the void area function is used to determine the area of the pocket with $\mathbf{P}^{\text{int}}$ as its input. The shape factor as a function of $\mathbf{P}^{\text{int}}$ and its sensitivity with respect to the domain elements P_i^{int} can be defined as follows (using the quotient rule):

$$\text{SF}(\mathbf{P}^{\text{int}}) = \frac{\text{TV}_8(\mathbf{P}^{\text{int}})^2}{4\pi \text{A}_{\text{void}}(\mathbf{P}^{\text{int}})}, \tag{3.22}$$

$$\frac{\partial \text{SF}(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}} = \text{TV}_8(\mathbf{P}^{\text{int}}) \cdot \frac{2\text{A}_{\text{void}}(\mathbf{P}^{\text{int}}) \frac{\partial \text{TV}_8(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}} - \text{TV}_8(\mathbf{P}^{\text{int}}) \frac{\partial \text{A}_{\text{void}}(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}}}{4\pi \text{A}_{\text{void}}(\mathbf{P}^{\text{int}})^2}. \tag{3.23}$$

Note that this sensitivity is with respect to the elements of the internal pocket domain. To get the sensitivities w.r.t. the original pocket domain, the sensitivities of the internal pocket domain w.r.t. the original pocket domain have to be included. With the shape factor and its sensitivity defined, the pocketing cost for internal pockets and its sensitivity can now be defined completely (using the product rule):

$$C_{\text{P,int}}(\mathbf{P}^{\text{int}}) = R_P \cdot \text{SF}(\mathbf{P}^{\text{int}}) \cdot \text{A}_{\text{void}}^{\text{penal}}(\mathbf{P}^{\text{int}}), \tag{3.24}$$

$$\frac{\partial C_{\text{P,int}}(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}} = R_P \left[\text{SF}(\mathbf{P}^{\text{int}}) \frac{\partial \text{A}_{\text{void}}^{\text{penal}}(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}} + \text{A}_{\text{void}}^{\text{penal}}(\mathbf{P}^{\text{int}}) \frac{\partial \text{SF}(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}} \right]. \tag{3.25}$$

It should be noted that both pocketing costs include the penalized version of the void area function, as was described in Section 3.3. The total pocketing cost is determined by summing the two together:

$$C_P = C_{\text{P,ext}} + C_{\text{P,int}}. \tag{3.26}$$

To find the sensitivity of the total pocketing cost to the pocket domain elements P_i, all previously defined sensitivities in this Section are used:

$$\frac{\partial C_P(\mathbf{P})}{\partial P_i} = \frac{\partial C_{\text{P,ext}}(\mathbf{P}^{\text{ext}})}{\partial P_i^{\text{ext}}} \frac{\partial \mathbf{P}^{\text{ext}}(\mathbf{P})_i}{\partial P_i} + \frac{\partial C_{\text{P,int}}(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}} \frac{\partial \mathbf{P}^{\text{int}}(\mathbf{P})_i}{\partial P_i}. \tag{3.27}$$

With the pocketing cost function and its sensitivity defined, it can be implemented in the topology optimization routine. One of the conclusions of the previous experiment was that penalization of the machining cost might result in the optimizer getting stuck in an undesired local minimum. To avoid this problem, the machining cost weight is set to 0 for the first few iterations of these experiments. This allows the optimizer to crystallize the initial intermediate values into regions of void and solid. After these first iterations, the machining cost weight factor is increased to the desired target. The number of iterations used in these and the following experiments was 15, which was found by trial and error.

3.4.1 Experiments

To see how the addition of the shape factor for internal pockets affects the outcome of the optimization, two experiments were designed. The first experiment uses domain dimensions and load case that, when optimized for compliance only, yields a design with three primary internal pockets and two external pockets. The second experiment is designed to have more internal pockets in the reference case.

A couple of expectations can be made based on the formulation of the method as described in this section. First of all, the introduction of the shape factor for internal pockets should result in them changing shape as the machining cost weight factor is increased. Because the shape factor does not act on external pockets, these are not expected to change shape. Finally, the presence of multiple internal pockets in the pocket domain will result in large values for the shape factor (due to the increasing perimeter). This means that when the machining cost weight factor is increased, the number of internal pockets is expected to go down in order to reduce the shape factor and thereby the internal pocketing costs.

Experiment 3A: small number of primary internal pockets

The parameters used for this experiment are listed in Table 3.5. Note that the penalization factors for contouring and pocketing are 2.0 and 3.0 respectively. The relative machining weight factors uses were 5%, 10%, 30%, 50%, 70% and 90%, shown in the bottom row of Figure 3.12. The reference design which was optimized for compliance only is shown in the top of the figure again. The resulting objective values of the results are shown in Figure 3.13.

Table 3.5: Optimization parameters for experiment 3A.

Optimization parameters for experiment 3A			
Domain dimensions:	20x30	Contouring rate, R_C:	1.0
Mesh size:	160x240	Contouring penal, β:	2.0
Load case:	MBB-beam	Pocketing rate, R_P:	5.0
Volume limit:	60%	Pocketing penal, p:	3.0
SIMP factor:	3.0		

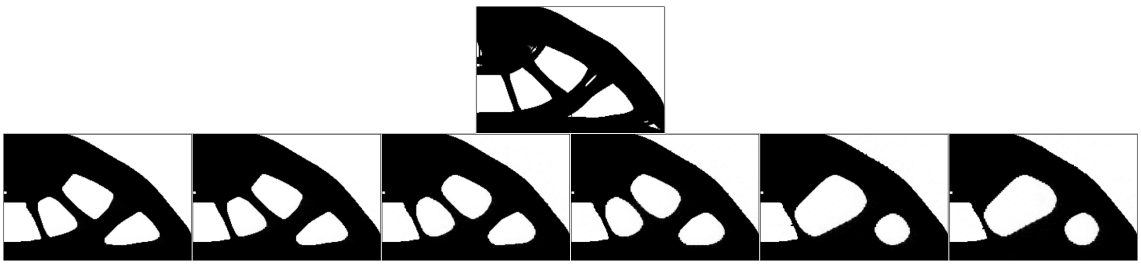


Figure 3.12: Results of experiment where the shape factor of internal pockets is included in the estimation of pocketing costs. The top design is the reference where no machining cost estimation is included. The bottom row of designs is generated using a weight factor of 5%, 10%, 30%, 50%, 70% and 90% from left to right.

Experiment 3B: more internal pockets

This experiment uses a domain shape that generates a larger number of internal pockets when optimized for compliance only. All parameters used are shown in Table 3.6. The results and their objective values are shown in Figure 3.14 and 3.15 respectively.

3.4.2 Conclusions

Interesting observations can be made when looking at the results of these experiments. Firstly, as the machining cost weight factor is increased, the number of internal pockets gets smaller. This already happened in the previous experiment due to the contouring cost, but the internal pockets that remain are rounded more than in the previous experiments. This is explained by the additional cost that is caused by the pocketing of pockets with non-circular shapes. It was unexpected that this effect would

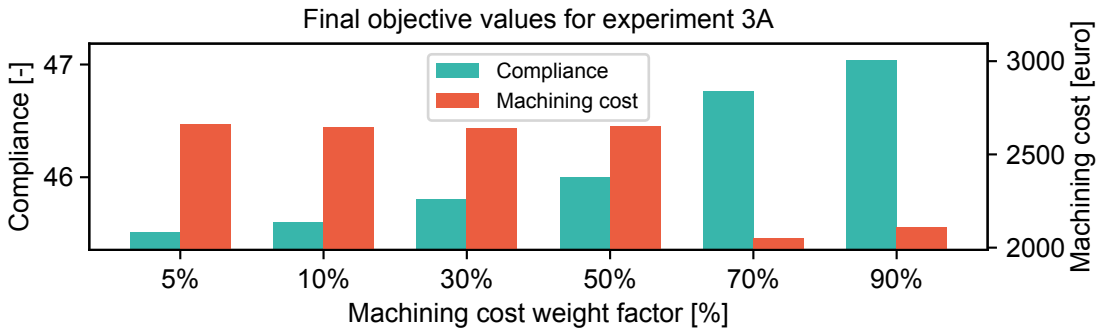


Figure 3.13: Values of the compliance and machining cost objectives after optimizations are finished for experiment 3A.

Table 3.6: Optimization parameters for experiment 3B.

Optimization parameters for experiment 3B			
Domain dimensions:	20x60	Contouring rate, R_C:	1.0
Mesh size:	110x330	Contouring penal, β:	2.0
Load case:	MBB-beam	Pocketing rate, R_P:	5.0
Volume limit:	60%	Pocketing penal, p:	3.0
SIMP factor:	3.0		

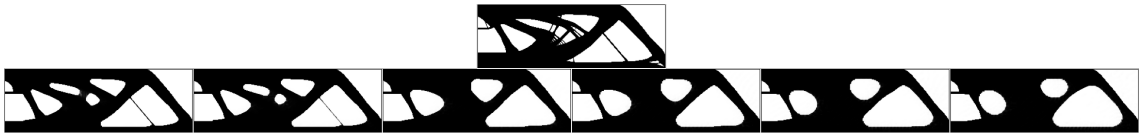


Figure 3.14: Results of experiment where the shape factor of internal pockets is included in the estimation of pocketing costs. The top design is the reference where no machining cost estimation is included. The bottom row of designs is generated using a weight factor of 5%, 10%, 30%, 50%, 70% and 90% from left to right.

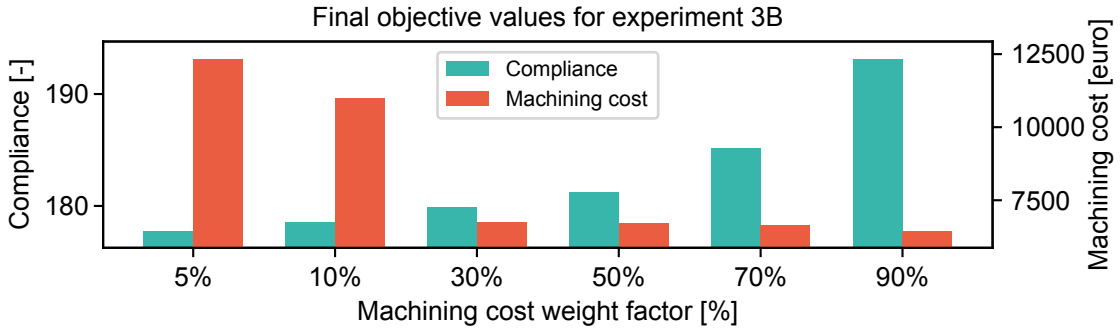


Figure 3.15: Values of the compliance and machining cost objectives after optimizations are finished for experiment 3B.

be visible when more pockets are present in the domain, because the definition of the shape factor is designed to be applied on a single pocket. When the area and perimeter of the domain is evaluated in these experiments however, they include multiple pockets, which results in a non-representative shape factor.

Having the machining cost weight factor be set to 0 during the first few iterations, in combination with a lower penalization factor for pocketing also solved the problem of getting stuck in undesired local minima during the first few iterations. Also when qualitatively looking at the results, they show the desired behaviour of the complete method, namely a reduction of internal pockets and rounding

of corners. This makes the formulation as it is presented here a potentially viable method to use for machining cost estimation and the results presented here will be validated in Chapter 4 as well.

An undesired effect can be seen as well when comparing the results using 70% and 90% machining cost weight factor of experiment 3A. The result of the 90% optimization has a higher machining cost than the result of the 70% optimization. This is unexpected and means that the 70% result is a better design when considering machining cost. When looking at these two designs, they appear very similar to each other and no clear differences in the geometry can be seen. By looking more closely at the machining cost objective, it was found that the contouring cost, as well as the pocketing cost is larger than the 70% result. This indicates that the perimeter of the 90% result is slightly larger which directly influences the contouring cost and the pocketing cost via the shape factor. An explanation for this behaviour might be that at a machining cost weight factor of 90%, the optimizer still gets stuck in a local minimum and does not manage to converge to a better result.

3.5 Multiple internal pockets

The previous experiment has showed how the shape factor can be taken into account when a single internal pocket is present in the input domain, but also showed that multiple internal pockets do not result in undesired results. The next logical step however, is to extend the method so the shape factor can properly evaluated for each internal pocket when multiple are present in the domain. In the current implementation, having multiple internal pockets in the pocket domain results in values for the shape factor that do not represent the shapes of those pockets.

When considering how the perimeter and void area functions operate, a method should be developed which ensures that a separate domain is created for each internal pocket. When this is done, the perimeter and void area functions can operate on these domains to find the properties of each of the internal pockets. Two strategies can be thought of to achieve this goal. The first being a top-down approach in which a domain containing several internal pockets is taken as input and a set of domains is returned, each containing a single internal pocket. The second approach can be considered bottom-up, as it takes a set of domains as input and compiles them into a resulting domain. The viability of both strategies is investigated such that the best one can be used in the method.

3.5.1 Top-down approach

How a top-down approach should work is illustrated in Figure 3.16. A domain containing (potentially) several internal pockets is given as an input. The method should recognize the pockets and copy them to their separate domains. It is clear that the number of output domains is dependent on the number of pockets in the input domain. Note that only internal pockets are required to be split into separate domains, because the shape factor is not calculated for external pockets. The external pockets can therefore all be grouped into a single output domain as shown in the figure as well (the leftmost output domain contains the external pockets).

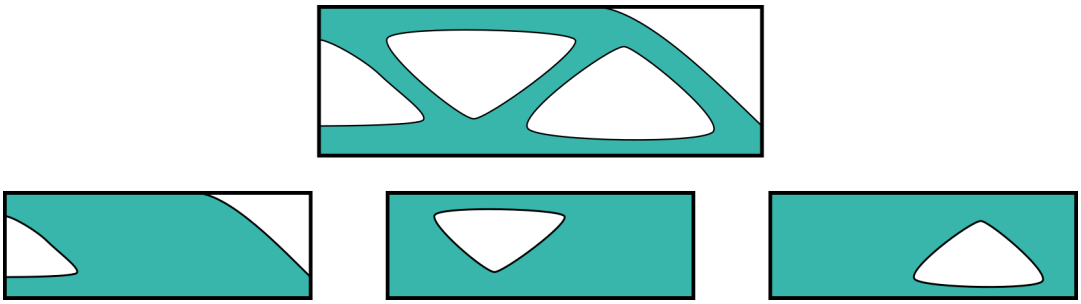


Figure 3.16: Illustration of top-down approach. Given the input domain shown in the top, this method should yield a set of domains (shown below the input domain) which all describe a single internal pocket. All external pockets can be combined in a single output domain.

The challenge of extracting features from an input domain is common in the of computer vision. An example in automation of biology experiments is using a camera to detect the number of bacterial colonies in a petri dish. A group of algorithms is often used, known as Connected Component Labelling algorithms [11]. These work by assigning a label to each pixel of the image and making sure that

connected pixels end up with the same label [5]. After the labelling process, which can be done in many ways, single features (pockets, in this case) can be extracted from the input domain. However, the binary and discrete nature of these algorithms present problems when implemented in gradient-based optimization schemes. These problems are explained in further detail in Appendix E.

3.5.2 Bottom-up approach

A bottom-up approach starts with the pockets defined in separate input domains and creates the resulting domain from those. It can be regarded as the reverse process illustrated in Figure 3.16, as the domains shown in the bottom row are the method's input and the top domain is its output. Each of the input domains contains a single pocket, which allows the evaluation of properties per pocket (area and perimeter) with the functions that have been discussed in the preceding sections. The output domain is found by taking all the pockets and compiling them to a single domain. How the input domains are initialized and how they are compiled using one of two methods (multiplication or a local minimum function) is discussed next.

Domain initialization

When only a single pocket domain is used to generate the final design, like in all the previous experiments, initialization of this pocket domain was straight-forward. By initializing all design variables to be equal to the allowed volume fraction, a domain was created that already fulfils the volume constraint in the first iteration of optimization. This method is also repeatable, which is important when comparing methods and doing experiments. However, now that multiple pocket domains are used, a different strategy is required.

If the method of setting all design variables to the allowed volume fraction is used with multiple pocket domains, a problem arises when the optimization starts. When the compilation function combines the pocket domains into the final design, the inputs to the smooth minimum function will all be equal. This will result in equal sensitivities of all the inputs. When the design is updated for the next iteration, because of the equal sensitivities, these updated domains will again be equal. In other words, there is no mechanism that motivates individual pockets to be formed in single pocket domains.

To prevent these equal sensitivities, either the inputs to the compilation function should not be equal or the objectives should be different for each pocket domain. The latter could be achieved by using machining rates in the machining cost estimation that change depending on the pocket domain considered. For example, if the machining rates are increased with 10% when calculating the machining cost of the next pocket domain, the sensitivities of the machining cost objective to each pocket domain will be different. This allows the updating scheme to generate new pocket domains that are not equal any more. However, changing process parameters like machining rates with arbitrary amounts is an artificial way of generating non-equal domains and is therefore not preferred.

The initialization of the design variables can have a great effect on the outcome of the optimization. This is why setting all design variables to the allowed volume fraction is standard when not dealing with multiple pocket domains, because it does not push the optimizer to a certain outcome a priori. But because this does not work when dealing with multiple pocket domains, the choice was therefore made to initialize the pocket domains by first executing an optimization without including the machining cost estimation, as illustrated in Figure 3.17. The starting point is a single pocket domain (1), initialized with the allowed volume fraction, as was used in previous experiments. This pocket domain is optimized for compliance only, resulting in design (2). Using the a CCL algorithm, design (2) is post-processed to create the set of pocket domains (3), which can then by used as initial pocket domains to start the optimization with machining cost estimation included, which will result in design (4).

A significant advantage of this method is that every pocket domain contains a single, well-defined pocket already in the first iteration. This should reduce the number of iterations needed to converge to the resulting design. Because the shape factor of a domain increases rapidly when more than one pocket is defined, the expectation is that no additional pockets will appear in the pocket domains after initialization. From a machining cost point of view, this is also not expected because additional pockets increase contouring cost.

It could be argued that by using this method, it is not likely for the optimizer to find designs with drastically different topologies that might perform better, due to the topology that is imposed on the

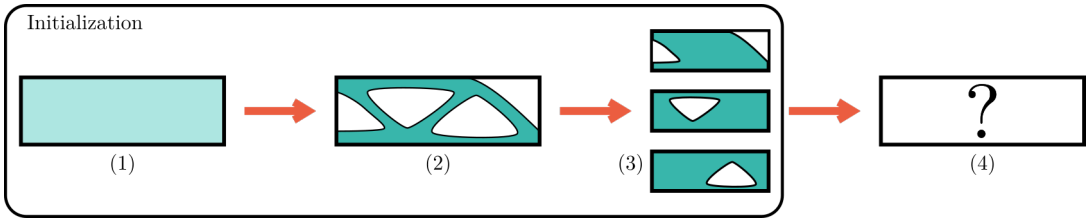


Figure 3.17: Illustration of the initialization process (outlined). Starting with a single pocket domain (1) which is optimized for compliance only. The resulting design (2) is post-processed to create a set of pocket domains (3), which is used to do further optimization (4).

first iteration. However, as explained above, expected topology changes are the deforming and disappearance of pockets, which are both not hindered by the initialization. Hence, using a compliance-only optimization to initialize the pocket domains for the main optimization that includes machining cost is the preferred way of initializing.

Domain compilation

After initialization, the method described in Section 3.4 is applied on each domain to calculate the machining cost of each of them. However, to determine the compliance of the resulting domain, the pocket domains must first be compiled into a single domain. The goal of this compilation is to end up with a domain that includes all pockets defined in all pocket domains. Multiplication or a smooth minimum function are both methods that can achieve this result, so these are investigated further.

Since elements belonging to pockets have low density values (close to 0) and solid elements have high density values (close to 1), one simple method of combining the input domains is to multiply them element-wise. When considering the elements in all input domains in a particular location, if their values are all close to 1, their product will also be close to 1. In other words, if an element represents the solid phase in all input domain, the element in the output domain will also represent the solid phase. On the other hand, if the void phase is represented in one or more of the input domains (these density values are close to 0), the element will also represent void phase in the output domain. This is the desired behaviour.

Since multiplication is an operation that can be differentiated without problems using the product rule, this method poses no issues when calculating the sensitivities with respect to the design variables. However, some negative effects can also be expected when using multiplication as the method for domain compilation. In the ideal case, pockets are fully defined by a single input domain and should only be sensitive to changes in that input domain. In the case of multiplication however, this is not the case.

When using the standard SIMP formulation, a minimum density value $\rho_{min} > 0$ is used to avoid singularities in the stiffness matrix. However, when using multiplication, this value will be multiplied with values from other input domains which can be smaller than 1. In that case, the compiled value can become smaller than ρ_{min}, which is beneficial due to the penalization of the SIMP formulation. Figure 3.18 shows how this effect will manifest in an optimization. Pockets in the input domains have density values of $\rho_{min} > 0$, but by giving other elements in the input domains density values smaller than 1, the compiled domain will have density values in the pockets smaller than ρ_{min}. This is undesirable behaviour because these values smaller than 1 in the input domains affect the machining cost estimation per pocket.

This problem can be avoided by implementing the extended SIMP method, which allows the assignment of density values of 0 without causing singularities in the input domain. The relation between density value and Young's modulus is modified as shown in Equation 3.28, where a minimum Young's modulus $E_{min} > 0$ is used instead of a minimum density value. Note that the density is still raised to a power p to penalize intermediate values. When an element reaches a density of 0, which is allowed with this formulation, elements from other input domains have no effect on the compiled value any more (0 times anything remains 0).

$$E_i(\rho_i) = E_{min} + (E_{max} - E_{min}) \cdot \rho_i^p \tag{3.28}$$

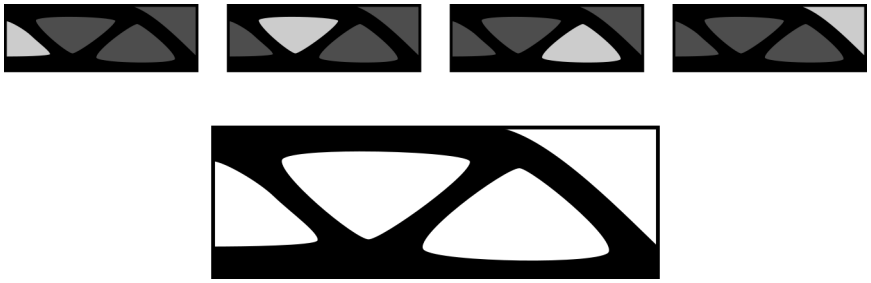


Figure 3.18: Illustration showing how compilation using multiplication leads to the coupling of all input domains to every pocket in the compiled domain. The greyscale represents density values where black elements are solid and white elements are void.

Instead of using multiplication to transfer pockets in the input domains to the compiled domain, another possible method is to use a minimum function. This function evaluates the minimum of all input domains element-wise and copies it to the output domain. An exact minimum function can not be used because this is not differentiable, hence a smooth variant is used. The LogExpSum and FracSumExp functions are both usable, but the FracSumExp function (shown in Equation 3.29) was deemed the better choice for reasons of linearity. The comparison between these two smooth minimum functions with more details is included in Appendix F.

$$\rho_j = \frac{\sum_{i=1}^{m} P_j^{(i)} e^{C_I P_j^{(i)}}}{\sum_{i=1}^{m} e^{C_I P_j^{(i)}}}. \tag{3.29}$$

Using smooth minimum functions does not completely remove the unwanted coupling between pocket domains, but it does change the behaviour. While the coupling depended in the element values when using multiplication, the coupling of this method depends on an external smoothing factor that can be freely chosen. It is expected that this will result in better convergence, because the amount of coupling is not depending on the input variables. For this reason, the smooth minimum function FracSumExp was used to compile the pocket domains into the resulting domain.

3.5.3 Experiments

The previous section explained how to create a bottom-up approach using a compliance-only optimization to initialize and a smooth minimum function to compile the pocket domains. This method is tested by means of two experiments. The experiments use reference designs with different domain geometries (20x40 and 20x60), in order to see how the method functions with different reference designs.

The penalization factors in these experiments (and the ones following these) were set to 1.0 to disable their effect on the optimization. It was found that they prevent the optimizer from changing the topology and shape of the design when a they are introduced in a black-and-white initial design.

Experiment 4A: 20x40 domain

The parameters used in this experiment are listed in Table 3.7. The FracSumExp function is used to compile the pocket domains into the resulting domain with a smoothing factor of 5.0. This factor was chosen based on the results of the comparison in Appendix F, where this value resulted in a reasonable error of approximately 3%. The results of this experiment are shown in Figure 3.19, with the reference design shown at the top of the figure and the results with machining time estimation included below it. The resulting objective values are plotted in Figure 3.20.

Experiment 4B: 20x60 domain

This experiment uses a different domain geometry that results in more internal pockets in the reference design. With this, the behaviour of the machining time estimation model with a larger number of pockets (and therefore also more pocket domains) can be tested. The MBB-beam load case was applied on a (20x60) domain, discretized with a mesh of (110x330) elements, as can be seen in the parameter table of this experiment (Table 3.8).

Table 3.7: Optimization parameters for experiment 4A.

Optimization parameters			
Domain dimensions:	20x40	Compilation method:	FracSumExp
Mesh size:	140x280	Compilation factor, C_I:	5.0
Load case:	MBB-beam	Contouring rate, R_C:	1.0
Volume limit:	60%	Contouring penal, β:	1.0
Simp factor:	3.0	Pocketing rate, R_P:	5.0
		Pocketing penal, p:	1.0

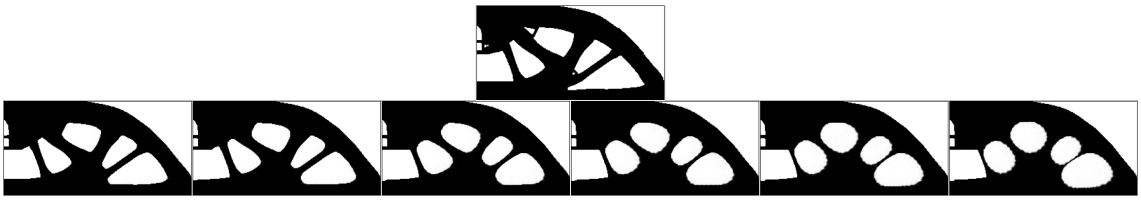


Figure 3.19: Results of experiment 4A, with the reference design show at the top of the figure. Below the reference are the results with machining time estimation included. The relative weights from left to right are 5%, 10%, 30%, 50%, 70% and 90%.

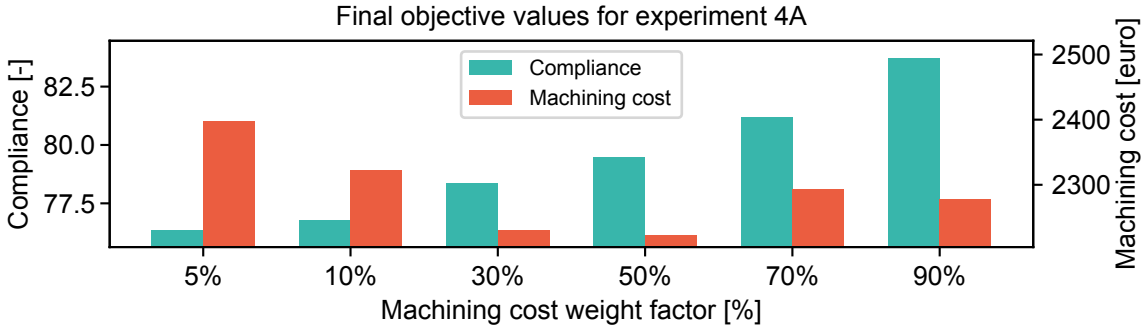


Figure 3.20: Values of the compliance and machining cost objectives after optimizations are finished for experiment 4A.

Table 3.8: Optimization parameters for experiment 4B.

Optimization parameters for experiment 4B			
Domain dimensions:	1x3	Compilation method	FracSumExp
Mesh size:	110x330	Compilation factor, C_I:	5.0
Load case:	MBB-beam	Contouring rate, R_C:	1.0
Volume limit:	60%	Contouring penal, β:	1.0
Filter radius:	1.5 elem.	Pocketing rate, R_P:	5.0
Simp factor:	3.0	Pocketing penal, p:	1.0

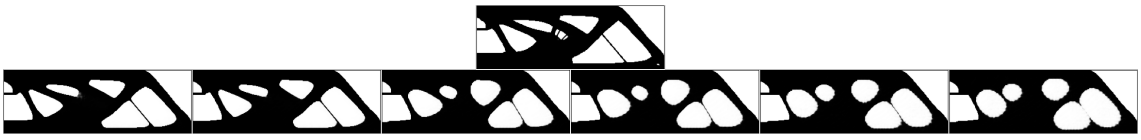


Figure 3.21: Results of experiment 4B, with the reference design show at the top of the figure. Below the reference are the results with machining time estimation included. The relative weights from left to right are 5%, 10%, 30%, 50%, 70% and 90%.

3.5.4 Conclusions

When comparing the results of these experiments to the ones from experiment 3A and 3B, a couple of observations can be made. The first is that while the number of internal pockets was reduced in

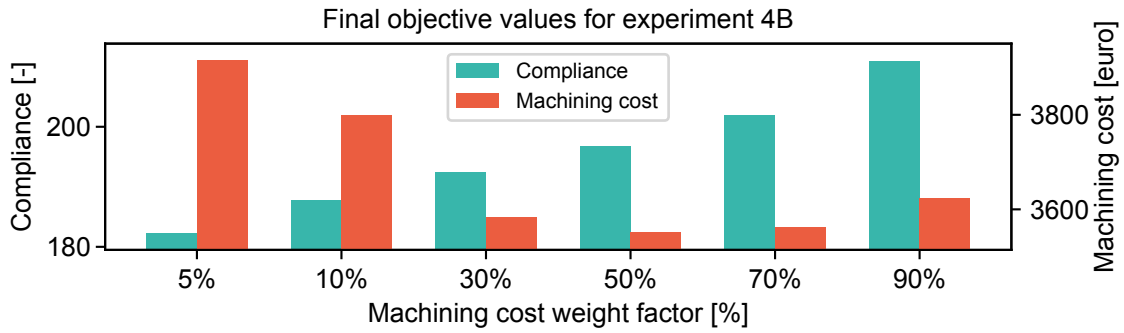


Figure 3.22: Values of the compliance and machining cost objectives after optimizations are finished for experiment 4B.

experiment 3A and 3B, this does not happen as quickly in these experiments. This can be explained by the fact that the shape factor is now evaluated for each internal pocket separately. Before this added complexity, a single shape factor of all internal pockets was evaluated, which was very high due to the large number of internal pockets. The fastest way to lower this shape factor is to reduce the number of pockets, but that mechanism does not work in this formulation any more. Instead, the shape factors are minimized by reshaping the internal pockets individually to resemble circles. Only the contouring cost is a driving factor to reduce the number of pockets. With the realistic contouring rate of 1.0, this effect is not strong enough to reduce the number of pockets effectively. This explanation is validated in the next experiment as well.

Another interesting phenomenon can be seen in the bottom-right corner of the results of experiments 4B. Two pockets are put extremely close to each other, to the point that the wall separating them can not contribute much to the stiffness of the structure. This indicates that these two pockets might as well be combined into a single triangular-shaped pocket. However, such a pocket would have a larger shape factor than the two pockets currently shown. In other words, the optimizer has found that machining two separate pockets with these shapes is more cost-effective than machining it as a single pocket. In reality, having another pocket increases cost because it required the drilling of another hole, but since drilling costs are not included in this formulation yet, the solution shown will be cheaper than the single triangular pocket.

3.6 Drilling cost

With the changes explained in the previous section, the method is now able to compute the machining cost of domains having multiple internal pockets. Contouring and pocketing costs are determined and their sum is the taken as the total machining cost. Adding drilling cost to the equation is the last step to complete the machining cost estimation model as it was described in Chapter 2.

A drilled hole is necessary for each internal pocket, as this allows a mill to enter the material without wearing the cutting edges irregularly. Hence, in order to determine if a drilled hole is necessary in a pocket domain, the presence of an internal pocket should be checked. In Section 3.4, the method of isolating internal pockets from an input domain has been discussed and this method can be used to this end as well. If any void area is detected in the internal pocket domain, the drilling cost of that domain should equal the pre-set (constant) drilling cost. If there are no void areas in the internal pocket domain, the drilling cost should be 0.

Because the process should remain differentiable, it is not possible to simply increase the drilling cost from 0 to the pre-set cost when an internal pocket is detected. Hence, a quantity is introduced to be used as a scaling factor, called the Internal Pocket Factor (IPF). This factor should be 0 when no internal pockets are detected and should be 1 when there is an internal pocket present. The drilling cost of the domain can then be determined by multiplying the IPF with the pre-set drilling cost.

The value of the IPF is calculated by transforming the normalized void area $\widetilde{\mathrm{A}}_{\text{void}}$ with a smooth Heaviside function. The normalized void area is the void area of a domain, expressed as a fraction of the total area (so its value is between 0 and 1):

$$\widetilde{\mathrm{A}}_{\text{void}}(\boldsymbol{\rho}) = \frac{N - \sum_{i=0}^{N} \rho_i^p}{N}. \tag{3.30}$$

The smooth step function is used to make sure the IPF is 0 when the normalized void area is 0, but increases rapidly to 1 when the normalized void area is larger than 0 [27]:

$$\text{IPF}(\mathbf{P}^{\text{int}}) = 1 - e^{-\beta \widetilde{\mathrm{A}}_{\text{void}}(\mathbf{P}^{\text{int}})} + \widetilde{\mathrm{A}}_{\text{void}}(\mathbf{P}^{\text{int}}) e^{-\beta}. \tag{3.31}$$

The β in this equation is the smoothness parameter of the smooth step function. By increasing this factor, the smooth version approaches an exact step function. However, a large β also means that the sensitivities of elements with high density values are lost. Hence, a value for β should be found that balances the approximation of an exact step function with the conservation of accurate sensitivities. The resulting IPF can then be multiplied with the pre-set drilling cost to find the drilling cost of this pocket domain:

$$C_D(\mathbf{P}^{\text{int}}) = R_D \cdot \text{IPF}(\mathbf{P}^{\text{int}}). \tag{3.32}$$

Note that $\mathbf{P}^{\text{int}}$ itself is a function of the pocket domain $\mathbf{P}$. The sensitivity of the drilling cost to the elements in the internal pocket domain can be found by differentiating the IPF to its input $\mathbf{P}^{\text{int}}$:

$$\frac{\partial \text{IPF}}{\partial P_i^{int}} = \beta e^{-\beta \widetilde{\mathrm{A}}_{\text{void}}(\mathbf{P}^{\text{int}})} + e^{-\beta}. \tag{3.33}$$

The sensitivity of the drilling cost C_D to the design variables can now be determined. Previously determined derivatives are used to transform the sensitivity with respect to the elements in the internal pocket domain to the design variables:

$$\frac{\partial C_D}{\partial D_i} = R_D \frac{\partial \text{IPF}(\mathbf{P}^{\text{int}})}{\partial P_i^{\text{int}}} \frac{\partial \mathbf{P}^{\text{int}}(\mathbf{P})_i}{\partial P_i} \frac{\partial P_i}{\partial D_i}. \tag{3.34}$$

Finally, with the drilling cost and its sensitivity with respect to the design variables defined, it can be added to the machining cost estimation model. The costs of all operations is simply added to find the total machining cost C_M, which means their sensitivities can also be added together to find the total sensitivity of the machining cost with respect to the design variables:

$$C_M = C_P + C_C + C_D, \tag{3.35}$$

$$\frac{\partial C_M}{\partial D_i} = \frac{\partial C_P}{\partial D_i} + \frac{\partial C_C}{\partial D_i} + \frac{\partial C_D}{\partial D_i}. \tag{3.36}$$

3.6.1 Experiment

Two experiments are done to check how the addition of drilling costs influences the resulting optimized designs. The same references were used as in the previous experiment, so the effects of the drilling cost can be clearly seen. The drilling cost adds another motivation for the optimizer to reduce the number of internal pockets. Hence, it is expected that the optimizer yields designs with less internal pockets than in the previous experiment.

Experiment 5A: 4 internal pockets with drilling cost

The parameters used in this experiment are listed in Table 3.9. The value of the drilling cost rate is set at 10, as described in Chapter 2. The results of the optimization are shown in Figure 3.23, with the values of the compliance and machining cost objective shown in Figure 3.24.

Experiment 5B: 7 internal pockets with drilling cost

Table 3.10 shows the parameters used for this experiment, which resulted in the designs shown in Figure 3.25. The objective values are shown in Figure 3.26.

Table 3.9: Optimization parameters for experiment 5A.

Optimization parameters			
Domain dimensions:	20x40	Contouring rate:	1.0
Mesh size:	140x280	Contouring penalization:	1.0
Load case:	MBB-beam	Pocketing rate:	5.0
Volume limit:	60%	Pocketing penalization:	1.0
SIMP factor:	3.0	Drilling rate:	10.0

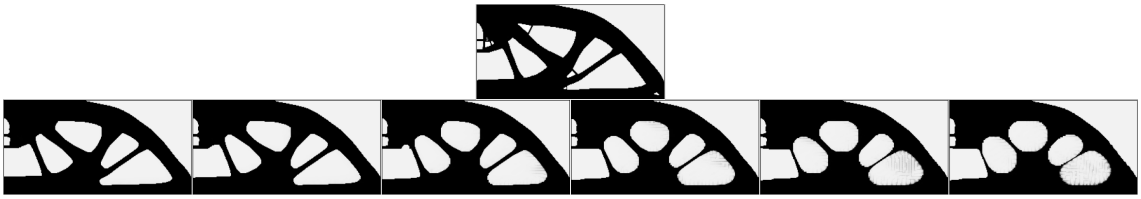


Figure 3.23: Results of experiment 5A using the complete machining cost estimation model. The top image is the reference result when no cost estimation is included (only compliance is minimized). Designs in the bottom row are generated with cost estimation included, with the relative weight for the cost objective increasing from left to right: 10%, 30%, 50%, 70%, 90% and 95%.

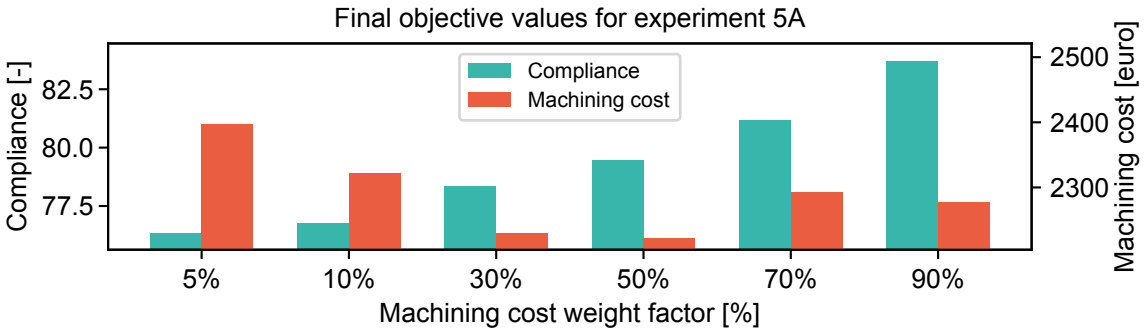


Figure 3.24: Values of the compliance and machining cost objectives after optimizations are finished for experiment 5A.

Table 3.10: Optimization parameters for experiment 5B.

Optimization parameters			
Domain dimensions:	20x60	Contouring rate:	1.0
Mesh size:	110x330	Contouring penalization:	1.0
Load case:	MBB-beam	Pocketing rate:	5.0
Volume limit:	60%	Pocketing penalization:	1.0
SIMP factor:	3.0	Drilling rate:	10.0

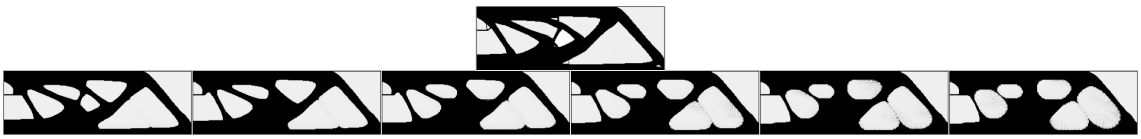


Figure 3.25: Results of experiment 5B using the complete machining cost estimation model. The top image is the reference result when no cost estimation is included (only compliance is minimized). Designs in the bottom row are generated with cost estimation included, with the relative weight for the cost objective increasing from left to right: 10%, 30%, 50%, 70%, 90% and 95%.

3.6.2 Conclusions

When considering how the addition of drilling costs has effected the results compared to the previous experiments, it is clear that the results are very similar. This indicates that the drilling cost is not a

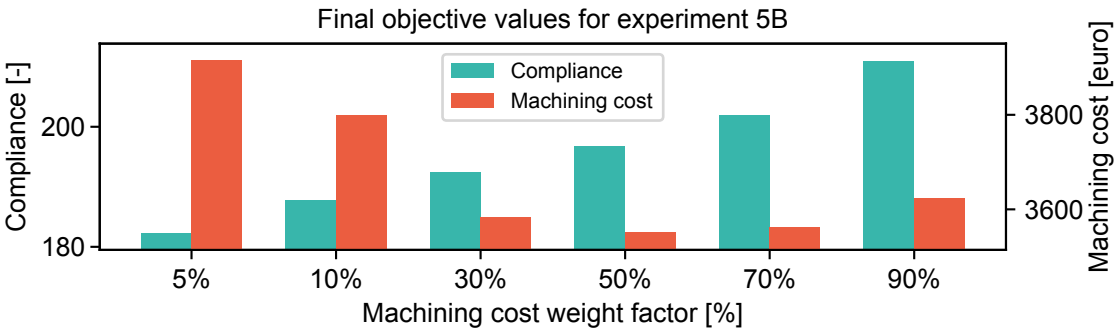


Figure 3.26: Values of the compliance and machining cost objectives after optimizations are finished for experiment 5B.

significant factor in the total machining cost. To verify this, a breakdown of the total machining cost for each result from experiment 5B is presented in Figure 3.27. This figure shows how the majority of the machining cost is determined by the pocketing cost, which explains why the optimizer does not remove pockets as well, as this does not yield a great cost saving. Whether this cost breakdown resembles the actual cost of machining (in other words, if the chosen rates are accurate) is checked Chapter 4.

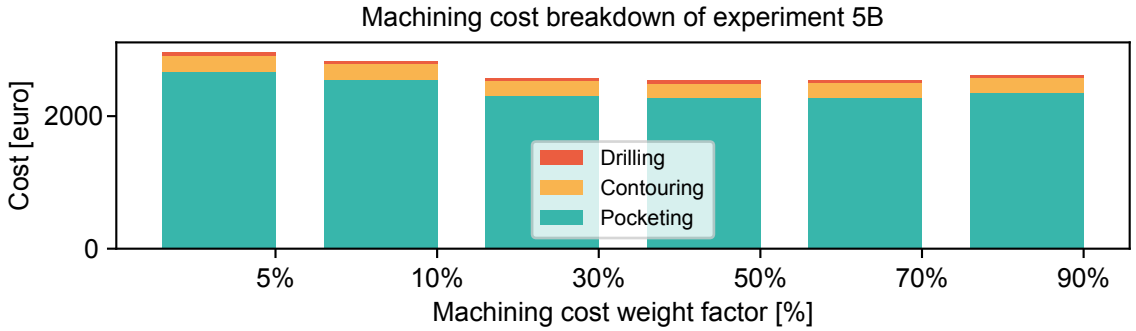


Figure 3.27: Cost breakdown of the resulting designs from experiment 5B.

3.7 Complete procedure

In the preceding sections, the method for determining machining cost of a design has been incrementally developed, adding complexity with every step. The result is a method that includes the cost of pocketing, contouring and drilling, which are the three basic operations defined in Chapter 2 that can be used to estimate the actual machining cost. The first step was a basic implementation of pocketing and contouring costs, which was improved by adding penalization in the following step. A method of differentiating between internal and external pockets was developed in the third step, which allowed the addition of shape factors for internal pocketing costs, but limited to a single internal pocket. This limitation was removed by defining the resulting design by intersecting a set of pocket domains and using those pocket domains to evaluate pocket-specific properties. Finally the drilling costs were added to complete the method.

The complete procedure, starting with the design variable vector and ending up with the objectives and constraints is illustrated in Figures 3.28 and 3.29. Figure 3.28 shows how the pocket domains are created from the design variables and Figure 3.29 shows how these pocket domains are used to find the machining cost, compliance and volume of the resulting design.

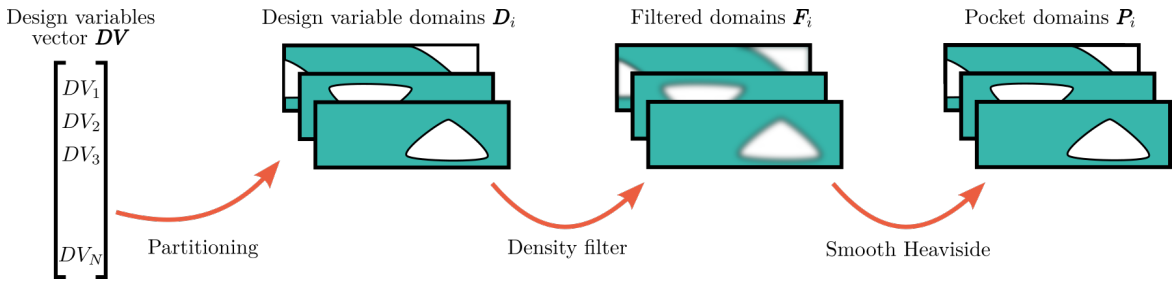


Figure 3.28: Illustration of the complete procedure to create the pocket domains $\mathbf{P}_i$, starting from the design variables vector **DV**. The design variables are grouped to create the design variable domains $\mathbf{D}_i$. These are filtered with the density filter to create $\mathbf{F}_i$. A smooth Heaviside is finally applied to create pocket domains $\mathbf{P}_i$.

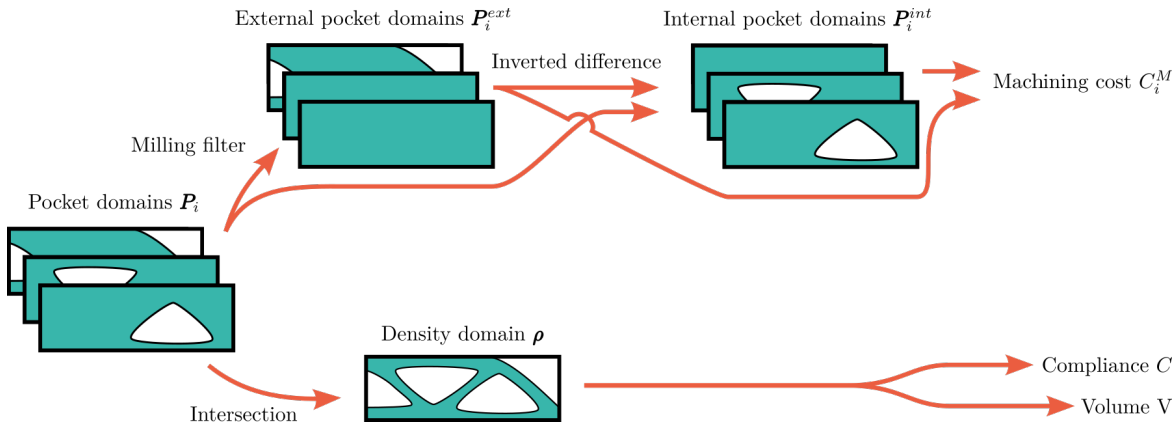


Figure 3.29: Illustration of the procedure to get the objectives and volume constraints from the pocket domains. The domains are filtered with the milling filter, which yields the internal pocket domains $\mathbf{P}_i^{\text{int}}$. Using the inverted difference operator, the internal pocket domains are found, which are used to evaluate the machining costs per domain C_i^M. To find the compliance and volume of the design, an intersection (smooth minimum function) is used to compile the pocket domains into the resulting density domain $\boldsymbol{\rho}$.

While Figures 3.28 and 3.29 show the total method visually, the same can be done mathematically.

$$C^M = \sum_i \left[C_C(\mathbf{P}_i) + C_P(\mathbf{P}_i) + C_D(\mathbf{P}_i)\right], \text{ with} \tag{3.37}$$

$$C_C(\mathbf{P}) = R_C \cdot \mathrm{TV}_8(\mathbf{P}), \tag{3.38}$$

$$C_P(\mathbf{P}) = R_P \cdot \mathrm{A}_{\text{void}}^{\text{penal}}(\mathrm{MF}(\mathbf{P})) + R_P \cdot \mathrm{SF}(1-(\mathrm{MF}(\mathbf{P})-\mathbf{P})) \cdot \mathrm{A}_{\text{void}}^{\text{penal}}(1-(\mathrm{MF}(\mathbf{P})-\mathbf{P})), \tag{3.39}$$

$$C_D(\mathbf{P}) = R_D \cdot \mathrm{IPF}(1-(\mathrm{MF}(\mathbf{P})-\mathbf{P})). \tag{3.40}$$

Note that when the machining cost is used in the optimization algorithm, a reference cost is used to normalize the value, such that it can be combined with the normalized compliance objective. For a detailed explanation of how the compliance objective and volume constraint are implemented, the reader is referred to previous work [3], [24].

Chapter 4
Validation

With the complete machining cost estimation model implemented as it was defined in Chapter 2, the obtained results can be validated to check whether the designs are actually cheaper when programmed for real CNC milling. The results from experiment 3B and 5B (shown again in Figure 4.1 and 4.2 respectively) were chosen for this validation. Experiment 5B was chosen because it uses the complete formulation as it was defined in Chapter 2. Experiment 3B was chosen because it showed the desired effects without the added complexity of using a set of pocket domains.

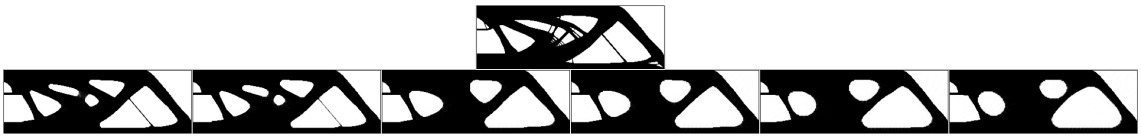


Figure 4.1: Results of experiment where the shape factor of internal pockets is included in the estimation of pocketing costs. The top design is the reference where no machining cost estimation is included. The bottom row of designs is generated using a weight factor of 5%, 10%, 30%, 50%, 70% and 90% from left to right.

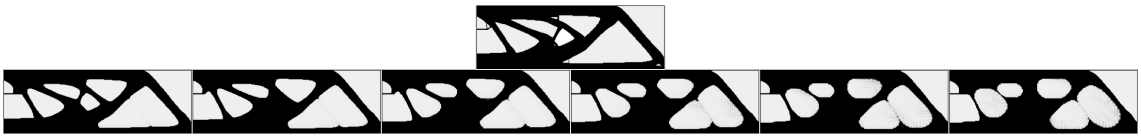


Figure 4.2: Results of experiment 5B using the complete machining cost estimation model. The top image is the reference result when no cost estimation is included (only compliance is minimized). Designs in the bottom row are generated with cost estimation included, with the relative weight for the cost objective increasing from left to right: 5%, 10%, 30%, 50%, 70% and 90%.

4.1 Method

In order to obtain actual machining times without milling the designs, CAM software is used to program the designs for a real milling machine. The software used was Autodesk Fusion 360, as it is free to use for students and supports the basic milling operations used in this research.

The designs were traced by hand as closely as possible and the 2-dimensional sketches were extruded to a constant thickness to transform them into 3-dimensional objects. This thickness is small enough such that all operations can be executed at the full depth in a single pass. Figure 4.3 shows how the result from experiment 5B with 50% machining cost included looks after extrusion in the CAM software.

The parts were programmed according to the machining steps defined in Chapter 2, meaning that each internal pocket was first pre-drilled. The drilling locations are determined by the software by finding the centre of the largest inscribed circle of each internal pocket. This allows the pocketing step to work most efficiently by reducing air-milling to a minimum. The pockets were then formed with a pocketing operation and finally the vertical walls were finished with a contouring operation.

The experiments were performed with pocketing and contouring rates of 5.0 and 1.0 respectively, which were chosen based on a mill of diameter 1 (explained in Section 2.2.4). Hence, a mill with the same diameter is used for the pocketing and contouring operations in this validation. Since the optimization does not impose a minimum length scale on the solid and void phases, it is possible that corners with smaller radii are present in the designs. These are not completely milled because the tool is too large, which would make the validated machining cost lower than the optimizer estimated it to be.

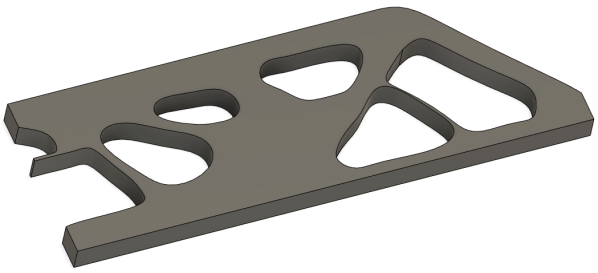


Figure 4.3: Result from experiment 5B with 50% machining cost included in the objective extruded to a 3D object in Autodesk Fusion 360.

However, it is not expected that this effect will have a significant impact on the total machining cost, so this effect is not further taken into account.

The tracing of the designs was done by hand, which introduces an error in the comparison as well. In order to quantify how accurate the designs were traced, their volume fraction was calculated. Because the optimized designs use 60% of the available volume, the extruded designs should also use 60% of their domain volume. Table 4.1 shows the volumes of the extruded designs taken from Autodesk Fusion. It can be seen that the extruded designs based on the results from experiment 3B are all within 1% of the target volume.

The extruded designs based on experiment 5B are not as close to the target volume. It can be seen that all experiments use about 6-7% less volume than then the allowed amount. This can be caused by the error in the pocket domain compilation, which results in values in the void phase that are slightly higher than 0. By analysing the results from experiment 5B, it was found that the elements in the void phase have a value of 0.063 or 6.3%, which suggests that this effect is indeed the cause of the discrepancy. Because some of the available volume was spent in these void elements, less than 60% of the volume was available to the solid phase in the optimization.

Table 4.1: Volume fractions of the extruded designs in Autodesk Fusion 360.

Volume of extruded designs						
M.C. weight factor	5%	10%	30%	50%	70%	90%
Volume fractions 3B	59.4%	59.8%	59.8%	59.4%	59.3%	59.1%
Volume fractions 5B	53.5%	53.9%	53.9%	53.8%	53.3%	53.0%

4.2 Results

After the programming in Autodesk Fusion 360 is done, the machining times can be evaluated and multiplied with the machining rates used in the optimization as well. These costs can then be compared to the ones found by the optimizer in the experiments. In order to compare the numbers easily, all machining costs are divided by the machining cost of the result with the smallest machining cost weight factor. This creates a relative machining cost that allows to quickly see how much cost is saved when the weight factor is increased.

When looking at the validation results of experiment 3B, it can be seen that the optimized designs get cheaper to machine as the machining cost weight factor is increased. The optimizer predicted that the savings would be larger, but this was to be expected because the shape factor implementation using multiple pockets in a single domain does not give accurate results. However, the optimizer still managed to save 13% of the machining cost compared to the result using only 5% machining cost in the objective. This was done at the expensive of 9% compliance.

The same analysis can be made with the validation results of experiment 5B. Because the shape factor is now determined with only a single pocket per domain, the optimizer did not overestimate the savings as much as it did in experiment 3B. However, it did suffer from a problem causing the estimated machining cost to increase from 50% weight factor to 90%. The actual machining cost did decrease monotonically with increasing weight factor, resulting in a saving of 12% machining cost at the expensive of 16% compliance.

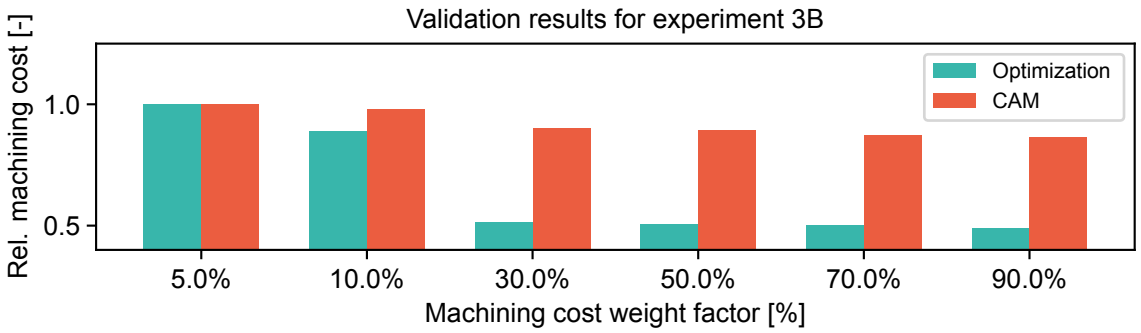


Figure 4.4: Validation results for experiment 3B.

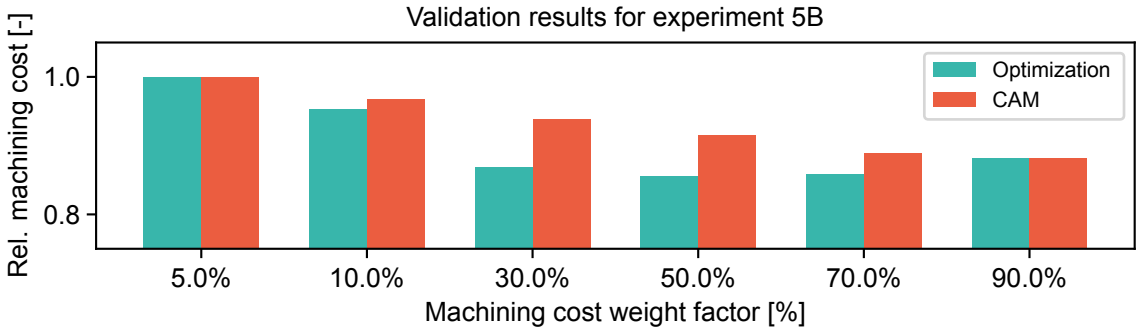


Figure 4.5: Validation results for experiment 5B.

Concluding, when comparing the saved machining costs with the sacrificed compliance, the formulation used in experiment 3B performs better (1.45% cost savings per 1% compliance loss) than the one used in experiment 5B (0.75% cost savings per 1% compliance loss). However, several improvements could be made in the formulation of experiment 5B that might increase its performance because the separation of internal and external pockets into pocket domains allows more tuning than the simple formulation of experiment 3B. At this point however, the added complexity of doing this does not pay off in more cost savings with less compliance loss.

4.3 Machining rates

At the end of Chapter 2, the machining rates for pocketing, contouring and drilling were chosen as 5.0, 1.0 and 10.0 respectively. This was based on relations between cutter dimensions, cutting speed and width of the cut. However, with some designs programmed in Autodesk Fusion 360 as well, the accuracy of these rates can be evaluated. Since the chosen rates were based on a constant rate for running the machine, the machining times for each operation in the results of experiment 5B should be proportional to the cost breakdown shown in Figure 3.27.

The breakdown of machining time for the results of experiment 5B is presented in Figure 4.6. When compared to the cost breakdown shown in Figure 3.27, it can be seen that the proportions in both plots are very similar, indicating that the rates chosen in Chapter 2 resulted in an accurate estimation. It should be noted that this only confirms the scaling of the rates with respect to each other, but not their absolute value. However, since the goal of this research was not to create an accurate time estimation, but rather to give the optimizer a metric to optimize for, the absolute values of the rates are not relevant.

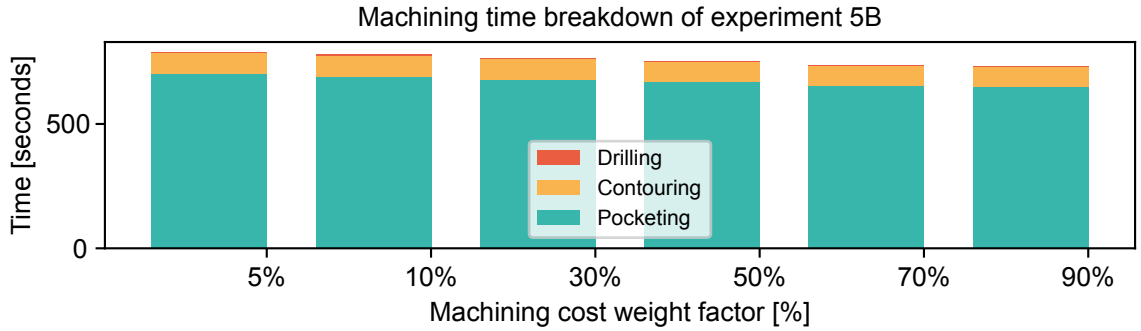


Figure 4.6: Machining time breakdown of the results from experiment 5B.

Chapter 5
Discussion

During the development of the cost functions presented in Chapter 3, several interesting aspects of the method have been noted. Some of these aspects are related to the results of the experiments and how the method is implemented. Other aspects are related to certain assumptions made at the beginning of the development. Finally, in the process of analysing the results, several potential improvements were identified which could not be tested due to time constraints.

5.1 Results

Based on the results shown in Chapter 3 and their validation, a few points of discussion were identified.

5.1.1 Sensitivity of smooth absolute function

As explained in Appendix C, in order to evaluate the perimeter of a domain, a smooth absolute function is required to determine the difference between the values of two elements. A fundamental problem of this approximation of the exact absolute function, is that when the input to the function is 0, its derivative is also 0. In practice, this means that whenever two elements are compared that have the same value, a difference of 0 is given as input to the smooth absolute function and the sensitivity calculated is 0 as well. A result of this is that whenever a region is present in the domain with all equal values, the computed sensitivity of the perimeter with respect to these elements is 0, which is not correct. This happens for all elements that are part of large regions of void or solid material, which is usually the majority of elements.

As a result, the optimizer does not know that changing an element within a region of void or solid material will increase the perimeter of the design (because this change introduces a new hole or forms an island of solid material). If other factors like the pocketing cost would not make the sensitivity of these elements non-zero, an oscillation might occur in the optimization where elements are changed back and forth. Whether this effect has a significant influence on the performance of the method remains to be investigated, as it was not directly observed in the experiments shown in this report.

Solving this problem means that either a different method of perimeter calculation should be used, which does not use an absolute function as a part of it, or the smooth absolute function should be modified to ensure it does not produce sensitivities of 0. However, due to time constraints, these options were not further investigated in this research.

5.1.2 Central differencing and oscillating domains

In the implementation of the perimeter evaluation, central differencing is used to evaluate the local gradient in the domain. However, a risk when using central differencing is that when a part of the domain oscillates between two values, no perimeter is found in these regions [15]. While the density filter prevents a true checkerboard-pattern from appearing in the solutions, the results of experiment 2B show alternating streaks of small and large density values, which can be explained by this phenomenon. In order to prevent this, forward of backwards differencing might be a better choice for this method.

5.1.3 Penalization hindering changes in geometry

The formulation defined in Section 3.5 introduced the idea of defining the density domain with a set of pocket domains, which get compiled into a single domain. It was explained why initialization with intermediate values was not possible in this situation and the choice was made to use a compliance-only optimization to initialize the pocket domains. However, after the initialization, the introduction of machining cost estimation should lead to new geometries. To accomplish this, values initialized as 0 might have to change to 1 and vice versa. The method of updating the design however limits the maximum change of each value, so intermediate values have to be introduced in order to accomplish

the change of geometry.

Intermediate values are unwanted in the final result, so penalization techniques have been introduced to demotivate the optimizer using them. While the penalization methods have proved useful at doing this (in experiment 2), they also hinder the optimizer in changing the initial geometry. This is why experiment 3 used lower values for the penalization factors and experiment 4 and 5 completely disabled their effect.

A solution to this might be to relax all penalization factors in the beginning of the optimization. This will lead to more intermediate values in the design, but depending on the convergence of the optimization, the penalization factors could be raised again, which should make the intermediate values disappear. This way, the optimizer is not hindered in changing the geometry to reduce the machining cost.

5.1.4 Complexity versus results

After the last experiment, a comparison can be made between the final formulation and earlier, less complex formulations. The conclusion has to be drawn that the very first formulation, defined and tested in Section 3.2, already showed very promising results. Corners were rounded and the number of internal pockets reduced as the machining cost weight factor was increased. These effects are the ingredients for designs of lower machining costs. The following formulations add complexity in the form of multiple pocket domains, which does not only complicate the method, but it also increases the computational cost of optimization. The number of design variables scales linearly with the number of pocket domains used, which can get quite high when complex geometries are considered.

Concluding, a trade-off can be made between complexity and method performance. Considering that the final formulation as presented in this paper does not perform better than the formulation used in experiment 3, it does allow more tuning and tweaking, which could enable it to perform better. However, whether the added computational cost is worth this potential performance increase is left to be determined by the end-user.

5.2 Assumptions

In the development of the method, some assumptions were made. For the sake of completeness, it is important to mention these assumptions and how they affect the results of optimization.

5.2.1 Shape of external pockets

In Chapter 2, the shape factor was introduced as a scaling factor for the machining time when considering internal pockets. The effect of this shape factor was seen in the experiments as a rounding of corners, reshaping the internal pockets to make them more circular. However, the shape of external pockets was assumed to not have a significant effect on their machining time. The reason for this was that when external pockets are machined, the mill does not start from the centre and circle outwards, but it enters the material from the side. After each pass, the mill has to move through the air back to the start of the next pass. Cutting in both directions would avoid this air-milling, but mills are designed to cut best in a single direction. Manufacturers will therefore prefer to cut only in that direction to maximize tool life and keep the wear predictable. In the validation of the method, this effect was ignored by enabling the mill to cut in both directions when cutting external pockets, which almost completely avoids any air-milling.

A more accurate estimation of the pocketing cost would take this effect into account by using a different machining rate R_P for external pockets. Because every external pocket will cause air-milling, regardless of its shape, this would make the estimated cost more accurate. The magnitude of this effect does however depend on the pocket shape, which means that some kind of model based on the pocket geometry is required to implement this (different from the model using the shape factor that is used for internal pockets). More research is needed to develop such a method and implement it in an optimization algorithm.

5.2.2 External pocket identification

In the method, the milling filter is used to differentiate between internal and external pockets. In order to successfully separate the internal pockets from the external ones, the milling filter checks accessibility of the void phases from multiple angles. For reasons of simplicity, the experiments shown in this

research use 4 principle directions to perform this check. However, when external pockets have complex geometry that 'hides' part of its void phase from all directions, this area will be misidentified as an internal pocket. An example is shown in Figure 5.1, where a external pocket is defined that is not fully recognized as such by the filter. Part of the solid phase hides the centre section of the external pocket from milling filter, so this area will be misidentified as an internal pocket by the algorithm. It should be noted that by increasing the number of considered machining directions, it is possible to reduce the area that is misidentified. When a part of the pocket is however hidden from all angles (of which Figure 5.1 is an example), some misidentification can not be avoided.

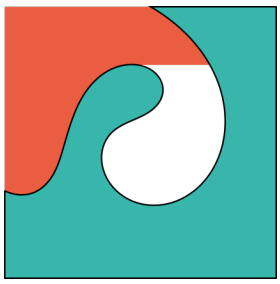


Figure 5.1: Example of external pocket that is not fully recognized as such by the milling filter. The orange area indicates the area that is found to be accessible by the filter, leaving the centre area undetected.

5.2.3 Other machining constraints

Chapter 2 explained how certain design features like internal radii and wall thicknesses affect the machinability of the design. The choice was made to not include functions like a minimum feature size in the method for reasons of simplicity. When new ideas are implemented for the first time and results are analysed, it is important to understand what is causing certain changes. In this work, the focus of the method was to use several techniques to model the machining cost of a design. A new method was presented to isolate internal pockets from external ones by means of the milling filter. The shape factor was introduced as a new kind of geometric property to evaluate and finally several domains were used to define a single output domain. All these steps were analysed to understand their effect on the outcome. To make this analysis as clear as possible, other functions that could change the outcome should be removed. For this reason, the choice was made to not add a minimum feature size function to the method at this point.

However, adding such a function would be a logical next step, so the implementation of such a function is briefly mentioned here. Guest proposed a method which uses a so-called multi-phase projection to ensure a minimum length scale on both the solid and the void phase [7]. However, implementing such a scheme in the machining cost estimation method is not without challenges. When the multi-phase projection is applied on the resulting density domain $\boldsymbol{\rho}$, the result will be a design that complies to the minimum feature size limits. However, the machining cost is calculated using the pocket domains $\mathbf{P}_i$, so any change in the designs caused by the multi-phase projection is not taken into account by the machining cost estimation. Experiments are needed to understand whether this causes significant problems, but it is expected that the multi-phase projection only changes the design slightly and will therefore not change the actual machining costs significantly.

5.3 Potential improvements

During the development of the cost functions, some aspects of the method caused undesired effects which had to be solved. While the resulting method is capable of generating designs better suited for milling, there are still quite some improvements that could be made.

5.3.1 Negative effects of penalization

As already discussed in Section 5.1.3, the addition of penalization hinders the optimizer in changing elements from void to solid and vice versa. If the penalization is only added after sufficient convergence has been achieved, the optimizer is not hindered in the first iterations, which allows it to change elements more easily. After this point, the added penalization helps in removing intermediate values

and creates a more well-defined result.

5.3.2 Necessity of smooth Heaviside projection

When the baseline procedure was defined in Section 3.1, a smooth Heaviside projection was used to create a density domain with less intermediate values than the filtered domain contains. However, when considering the final formulations of the machining cost components, there should be no problem when their inputs contain intermediate values. Hence, it can be argued that this projection step is not necessary and only causes sensitivity information to get lost. Running optimizations without this filter enabled was tried, but this revealed why it actually is necessary.

The milling filter used to create the external pocket domain uses cumulative sums to determine which void regions are external and which are internal. An effect of these cumulative sums is that a region of values slightly higher than 0 can be summed together to create a solid region. For example, if 10 elements with a density value of 0.1 lined up with each other, the cumulative sum will result in a solid element with value 1 at the position of the tenth element. By this effect, a few intermediate values can create external and thereby internal pocket domains that are not as expected, resulting in undesired behaviour. Figure 5.2 shows an example of this effect.

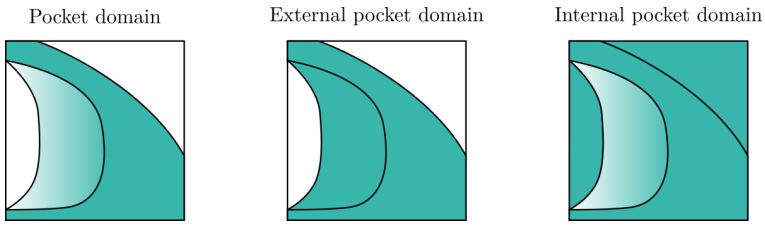


Figure 5.2: Illustration of how the cumulative sum causes the definition of internal pockets while they should not be recognized as such when no smooth Heaviside projection is used to project the pocket domain first.

A possible solution to this problem might be to not use cumulative sums, but a progressive maximum function. Starting from one side of the domain, the value of each element is replaced by the maximum between its own value and the value of the element before it. The first element is not changed, which results in the following formulation:

$$P_0^{\text{int}} = P_0, \tag{5.1}$$

$$P_i^{\text{int}} = \max[P_i^{\text{int}},\ P_{i-1}^{\text{int}}]. \tag{5.2}$$

The cumulative sum can be written as a matrix multiplied with the input domain in vector form. The same applies to this progressive maximum method, but this matrix is now dependent on the input domain. This makes the computation more expensive, but should still be possible.

If this progressive maximum is implemented, intermediate values can still show up in the pocket domains. However, intermediate values that are connected to an external pocket now show up in the external pocket domain instead of the internal pocket domain. Figure 5.3 shows the internal and external pocket domains created using the same input pocket domain as in Figure 5.2, but now using the progressive maximum. Whether this change could result in better results without the need of the Heaviside projection remains to be investigated.

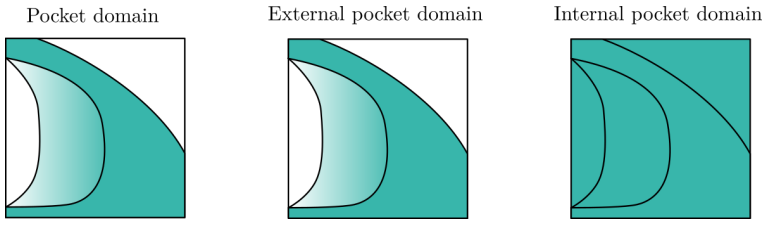


Figure 5.3: Illustration of how the progressive maximum prevents the definition of internal pockets but does allow intermediate values to show up in the external pocket domain.

Chapter 6
Conclusion

With all the results presented in the preceding chapters, some final conclusions can be drawn. Starting with the proposed machining cost estimation model defined in Chapter 2, it was found that the three basic operations allow a useful estimation of actual machining cost. The significance of the drilling cost was found to be very small, to the point where it contributes a negligible amount in the final machining cost objective. Furthermore, the pocketing costs were found to be most significant, making up around 90% of the total costs, leaving around 10% of the costs which represent the contouring operation. In practice, this means that the most important goal for minimizing machining cost is to maximize the pocket circularity. Reducing the number of pockets only affects the contouring and drilling costs (assuming a volume constraint is present), which are not as significant.

Zooming in to the pocketing costs, it was found that the shape factor is a suitable geometrical property to estimate the pocketing costs. When it is applied on a domain containing a single pocket, it is effective at motivating the optimizer to circularize the pocket, reducing its shape factor and thereby its costs. Even when it is used with a domain containing multiple pockets, it results in a reduction of the number of pockets, as well as a rounding of corners. These are both desired effects when optimizing for machining cost.

The contouring cost estimation is based on the perimeter evaluation by means of the TV_8 function. While this method gives accurate perimeter evaluations for geometries with complex shapes, it also presented some challenges for the method. Its dependency on an absolute operator and its use of central differencing both introduce sources of undesired behaviour. However, these effects were not found to be significant in the experiments performed in this research.

The implementation of the drilling cost showed to be effective at determining the actual drilling costs. However, its significance to the total machining cost was found to be negligible and its addition is not visible in the outcome of the optimizations.

The validation using Autodesk Fusion 360 showed that the machining costs found by the optimizer are proportional to the actual machining cost, which indicates that the method works as expected.

The method of distinguishing between internal and external pockets by means of the milling filter was found to be effective and useful. It allowed a different evaluation of machining costs for each type of pocket, as was proposed in Chapter 2. By increasing the number of considered machining directions, more complex geometries for external pockets can be correctly identified as being external, but the examples in this research did not require this additional complexity.

Finally, the idea of using a set of pocket domains to compile into a single density domain allowed the evaluation of pocket-specific properties such as their shape factor. However, it also requires a more complex initialization method to avoid all pocket domains from being identical. If a compliance-only optimization is used to this end (as was done in this research), the optimizer might be hindered by local minima. Other methods of initialization might avoid this problem, but this was not investigated further.

Concluding, the machining cost estimation model proved to allow optimization for machining cost with different levels of complexity. While the complex formulation using pocket domain compilation and pocket-specific property evaluation allowed a more accurate estimation of the machining cost, the simpler formulation achieved more cost savings at the expense of less compliance loss. This simpler formulation uses a single pocket domain and the milling filter to distinguish between internal and external pockets. At this point, this simple formulation gives better performance, but the complex formulation has more room for tuning and modifications, which might allow it to perform better.

Chapter 7
Recommendations

7.1 Adding other machining constraints

As discussed in the introduction of this work, many machining constraints have been implemented in topology optimization that apply to 2.5-axis CNC milling as well. Therefore it would make sense to add these constraints to this formulation to generate designs that take into account these constraints. The addition of minimum feature size in the void and solid phase especially should improve the results and prevent thin walls from appearing (like the ones observed in experiment 5B).

A different but relevant addition to the method would be to consider the use of multiple tools. In the current formulation, the tool diameter is not taken into account and the assumption is made that every operation is done with a single tool. A more realistic strategy however would be to start milling with large tools to take out the majority of the material that needs to be removed. Because this large tool is not able to get into the small internal radii, a smaller tool is used after the first to remove the remaining material. Depending on the geometry, more steps might even be used to minimize machining time with small tools.

Whether the implementation of this multi-tool machining is feasible is a question that deserves further investigation. Some existing machining constraints might be used to filter the domain for a specific tool shape and determine which parts of the domain are not reachable, but how this could be further integrated remains to be determined.

7.2 Extending the method to 3 dimensions

While the goal of this research was to focus on 2.5-axis CNC milling, the results presented in this report use 2-dimensional domains to describe the designs. In the formulation of the cost functions, it was assumed that the design has a unit thickness and that pockets go through the entire thickness of the design. While these assumptions result in designs that can be milled, more complex geometries are possible with 2.5-axis milling. A logical next step in the development of this method would therefore be to implement the possibility for pockets to have specific depths.

One possible implementation to achieve this, would be to add one design variable to each pocket domain that represents the depth of the pocket defined in that domain. For example, when the pocket depth design variable would have a value of 1, it means that the pocket in that domain is as deep as the thickness of the design (this is equal to the implementation presented in this work). However, a value of 0.5 would mean that only half of the thickness is pocketed away. An example of how such a design would look like in 3 dimensions is shown in Figure **??**. This example uses the an optimization result from experiment 5B, but instead of cutting all pockets completely through, each pocket is assigned a specific depth. The external and some internal pockets still cut through the entire thickness in this example, but it is possible for pockets to only cut to a specific depth.

If this extension is implemented, the finite element method used to evaluate the compliance also has to be extended to use 3-dimensional meshes. In order to save computational time, a regular mesh spanning the entire domain (in 3 dimensions) could be defined at the start of the optimization. In order to determine the density of each element in this mesh, the previously used smooth Heaviside function can be used again. When a depth variable is defined for every pocket domain between 0 and 1, this value can be used as the threshold value η in the smooth Heaviside function. This provides a convenient method to map the 2-dimensional pocket domain to a 3-dimensional domain. An example of how such a design could look like is included in Figure 7.1. The 2-dimensional pocket domains can still be used in the same way as presented in this work to determine the machining costs, assuming the depth of the pockets is small enough such that all milling operations can be done in a single pass (considering the maximum depth of the mill).

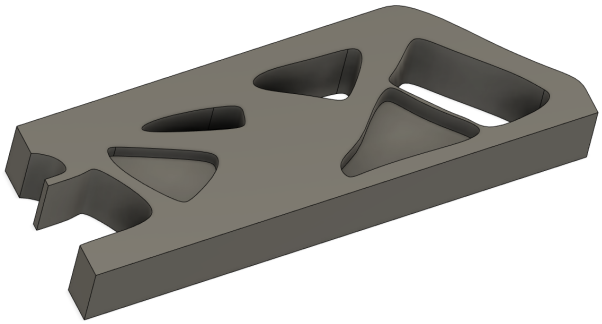


Figure 7.1: Example of a 3-dimensional design based on a 2-dimensional description with specifics depths per pocket.

The compliance evaluation will produce a total compliance as well as its sensitivity with respect to each of the elements in the 3-dimensional mesh. Because the optimizer still uses 2-dimensional design variable domains, some transformation of the compliance sensitivities will be required. If the smooth Heaviside function is used as a mapping between 2- and 3-dimensional pocket domains, this poses no problems and the sensitivities with respect to the design variables can be evaluated without problems.

7.3 Using other types of objectives

In this work, the choice was made to use compliance as the primary objective to optimize for. However, other objectives could also be used for which the milling cost estimation might be a valuable addition in real applications. For example, designs optimized for cooling performance can often lead to complex geometries in the form of coolant channels or heat conduction paths, as shown in Figure 7.2. Because designs like these are often milled, it would be very interesting for a designer to investigate how this design could be made more suitable for that manufacturing process. Cooling performance will decrease, but a different balance between cost and the cooling performance might be a better design in the end.

Figure 7.2: Example of optimized cooling plate for battery units. (Source: `https://www.diabatix.com/applications/liquid-cooling`)

7.4 Level-set method

One of the fundamental choices made in the development of the machining cost estimation model is to use a density-based description of the design. This choice was made to make the method align best with the most-used topology optimization techniques. The model uses basic properties of the geometry such as areas and perimeters to estimate the total milling cost. These properties have also been used with a level-set formulation in other research [12]. Because the perimeter calculation introduced a source of errors in the sensitivity computation (this was discussed in Chapter 5 as well), the use of a level-set formulation might provide a solution to this. To answer the question if the level-set method can be a better choice for the estimation of milling cost, more research is needed.

Chapter 8
Applicability

This work was inspired by the application of topology optimization in today's industry. Hence, it is suitable to end by reflecting on the work that has been done and to consider its applicability in the industry today.

8.1 Open-source versus commercial software

It takes time for developments of technology to get implemented in software that allows the industry to use them. When comparing open-source to commercial software, new developments in technology are often first developed in open-source software before they become available in commercial alternatives. This can be explained by the fact that open-source software allows more people to contribute, but also because often less testing and validation is required. While fast development can be regarded as a positive aspect of open-source software, there are also negative aspects related to it. Because anyone is able to contribute to the software, there is no clear responsibility for the correct functioning of the software. This should be taken into account when choosing the software to use in a commercial setting.

The technical support offered by commercial software development companies is another important reason for industry to use this type of software. When a project depends on the correct functioning of software, the risk of technical issues and not having support available to fix these problems is a risk that usually can not be taken. It is because of reasons like these that most commercial companies choose to work with commercial software and not rely on open-source software for their work.

Using newly developed methods like the one presented in this report is unfortunately hard to do with commercial software. Unless the software was designed to be extendable by end-users, it can be very hard, if not impossible, to achieve the level of control needed to implement these new methods. Most software allows to automate and script tasks like pre- or post-processing of results, but when methods change the fundamental functioning of the optimization loop, this is often not possible.

8.2 Demand of cutting edge developments

Because most companies rely on commercial software developers to implement new methods in their packages, it can take a long time before new methods become available. While this by itself can be regarded as a problem, it can also be questioned how big this problem actually is. While companies can be found that really operate at the cutting edge of techniques like topology optimization (especially in industries like automotive, aerospace, semi-con, etc.), it is expected that the majority of engineering companies do not use advanced design methodologies like custom-modified topology optimization for their general engineering. Therefore it can be said that the possibilities in commercial software are sufficient for the majority of users. This was confirmed in an interview with a TNO (Dutch research organization), which stated that only a small number of companies in the Netherlands use topology optimization at all, let alone run into the limitations of software packages [1].

8.3 Conclusion

While the importance of developing new methods in the field of topology optimization is extremely important for enabling innovation in all kinds of industries, most companies are currently not using the full potential of the tools available to them. It was shown how the simple formulation of the machining cost estimation already allowed significant cost reduction and its implementation in commercial software should be possible with relative ease. If such an implementation is desired, some research should be done to understand the possibilities in the commercially available software packages, but it can be expected that the simple formulation for cost estimation as presented in this report should be possible to implement in several packages.

Appendix A
Filter and Projection

The formulation used to find the machining cost presented in this work uses a density filter and smooth Heaviside projection to avoid checkerboarding, mesh-dependence and intermediate values in the resulting domains. Because these steps are often encountered in topology optimization algorithms, the implementation of them is not included in the main matter of this report, but in this appendix instead.

Density filter

When topology optimization was being developed, the problem of checkerboarding and mesh-dependence had to be solved in order to achieve desirable results. A sensitivity filter was developed that replaced the sensitivities of all design variables by a weighted average of itself and the sensitivities of the surrounding elements [25]. The weights used in this averaging are determined by the distance of the design variables to the design variable whose sensitivity is being filtered. A filter radius is defined that determines how many sensitivities are included in the averaging.

In later years, a similar approach was used to filter the design variables themselves instead of their sensitivities, which resulted in solutions free from checkerboarding and not dependent on the mesh that was used [4]. It was this method that became known as the density filter and is often found in topology optimization algorithms.

When implementing the density filter $\mathrm{DF}(\mathbf{x})$ in the optimization procedure, the relation between the filtered output value f and the input values x is described as follows:

$$f = \frac{\sum_i (w_i \cdot x_i)}{\sum_i w_i}. \tag{A.1}$$

In this formulation, x_i is the set of design variables that has a distance smaller than the filter radius to the input design variable. Their corresponding weights w_i are found by evaluating their distance to the input variable x (using any distance function 'dist') and subtracting it from the filter radius:

$$w_i = R_f - \mathrm{dist}\,(x_i,\ x)\,. \tag{A.2}$$

Note that the weights are not depending on the actual values of the input values, which allows them to be computed once before the optimization starts and re-used for each iteration that follows (assuming the mesh stays constant throughout the optimization).

When this filtering step is applied to create filtered domain $\mathbf{F}$ from the design domain $\mathbf{D}$, the sensitivity should also be adjusted accordingly:

$$\frac{\partial D_i}{\partial y} = \frac{\partial D_i}{\partial F_i}\frac{\partial F_i}{\partial y}. \tag{A.3}$$

In this equation, the value y could represent any objective or constraint that is part of the optimization. To take the effect of this filter into account when calculating sensitivities, the derivative of the output value to the input values must be determined. Because this filter does not work element-wise, but uses a subset of elements in an input domain, this sensitivity can be expressed as a sparse matrix, $\mathbf{S}$:

$$\frac{\partial \mathrm{DF}(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{S}. \tag{A.4}$$

The non-zero entries in this matrix correspond to contributions of elements in the input domain to the values in the output domain. These contributions are equal to weight factor w_i, divided by the sum of all weights that contribute to the a single output element [14].

Smooth Heaviside projection

A Heaviside function can be used to transform a continuous domain with values between 0 and 1 into a binary domain that only contains the values 0 and 1. A threshold value η is used to determine which values to set to 0 or 1. This is a useful tool when considering that intermediate values are often not desired in the context of topology optimization. However, the discrete nature of the Heaviside function makes it non-differentiable (which is a requirement for usage in gradient-based optimization), so a smooth version is created that approaches the original function but can be differentiated [27].

The behaviour of the discrete Heaviside function is straight-forward to describe. Any value below the threshold is changed to 0 and any value above it to 1. Because the value of the element in the output domain is only dependent on the value of the corresponding element in the input domain, this operation can be applied element-wise. The smooth version uses a function including $\tanh$ functions in combination with a smoothness parameter β to approach the discrete function [14]. With input value x and output value y, the complete formulation is as follows:

$$\mathrm{HS}(x) = \frac{\tanh\left(\beta \cdot \eta\right) + \tanh\left(\beta\left(x - \eta\right)\right)}{\tanh\left(\beta \cdot \eta\right) + \tanh\left(\beta\left(1 - \eta\right)\right)}. \tag{A.5}$$

As shown in Figure A.1, the smooth Heaviside approaches the discrete version as the value of β is increased. The sensitivity of the output value with respect to the input value can be determined by differentiation of Equation A.5 with respect to x:

$$\frac{\partial \mathrm{HS}(x)}{\partial x} = \frac{\beta \cdot \left(1 - \tanh\left(\beta\left(x - \eta\right)\right)^2\right)}{\tanh\left(\beta \cdot \eta\right) + \tanh\left(\beta\left(1 - \eta\right)\right)}. \tag{A.6}$$

An important trade-off has to be made when choosing the value of β. While a high value makes the smooth function approach the discrete version, which results in an output that is closer to a binary domain, it also leads to a loss of sensitivity information when this function is used in a topology optimization procedure. This effect is caused by the fact that when using large values of β, a large portion of the input range is pushed very close to 0 or 1, resulting in a sensitivity of close to 0. For example, when looking at the line corresponding to $\beta = 16$ in Figure A.1, it can be seen that any input value below 0.3 or above 0.7 will be transformed to a value very close to 0 or 1, resulting in a function output with hardly any slope. Hence, when the total sensitivity is computed, elements in this range lose most of their sensitivity information as they are multiplied with 0. This effect is especially important to consider when the initial domain is defined with values of 0 and 1, as is the case in experiments 4 and 5 described in Chapter 3, because if all the sensitivities are multiplied with 0, no sensitivity information will remain. To this end, it is important to choose a value for β that is not too high.

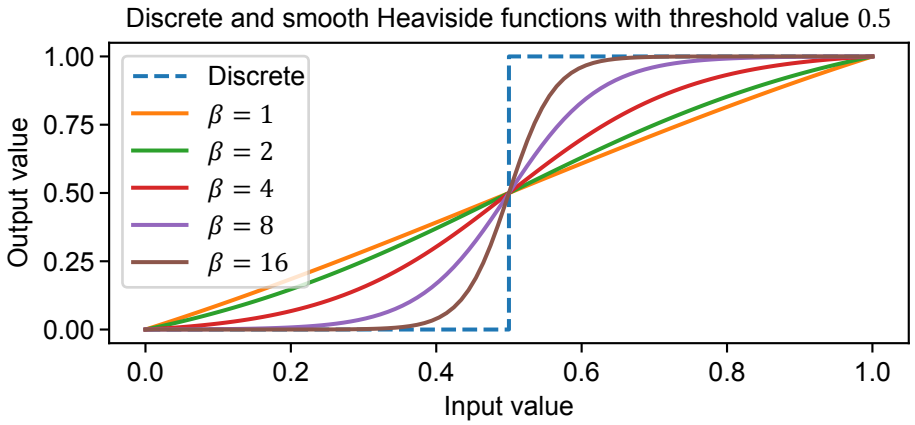


Figure A.1: Heaviside function applied on a range of input values from 0 to 1 with a threshold value of 0.5.

Appendix B
MBB-Beam Load Case

Perhaps the most-used benchmark load case for 2D optimization problems is the so-called MBB-beam. A horizontal beam has its bottom corners constrained in the vertical direction and a downwards load is applied in the centre on the beam's topside, shown in Figure B.1. In the implementation of this load case, one of the pinned corners is also constrained in the horizontal direction to constrain the rigid-body mode of translating horizontally.

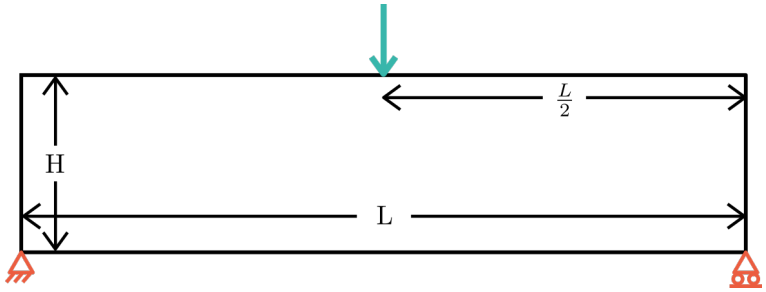


Figure B.1: Full MBB-beam load case.

The symmetry of the full MBB-beam load case allows it to be cut in half as shown in Figure B.2, reducing the computational cost of the optimization. The beam is split in the centre and the horizontal displacement of the newly formed boundary is constrained (the left boundary in Figure B.2). After the optimization is finished, the domain can be mirrored across the left boundary to create the full MBB-beam geometry again. This half MBB-beam is the load case that is used for the experiments shown in Chapter 3. Variations on this load case exist, which apply a distributed load instead of a point load for example. However, such variations are not used in this research.

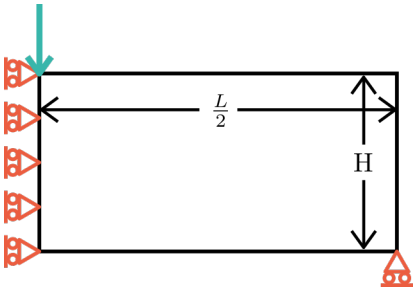


Figure B.2: Half MBB-beam load case.

Appendix C
Perimeter evaluation

Using the perimeter of a domain in topology optimization is not a new idea. In 1996, Haber et al. proposed to constrain the total perimeter as a means of combatting checkerboarding and to give the designer some control over the number of holes in the design [10]. This research however uses the perimeter to estimate the pocketing and contouring costs.

C.1 Total Variance method

Calculating the perimeter in a binary domain is done by finding the boundaries between the void and solid phases. Figure C.1 shows how the mathematical circle in the left image with diameter D and perimeter πD is discretized to a binary domain in the centre image. When the total perimeter is calculated for the discretized domain, a value of $4D$ is found, which is larger than the perimeter of the original circle (πD). This is caused by the fact that in the binary domain, the perimeter can only consist of horizontal and vertical segments which together form the so-called 'taxi-cab' perimeter, inspired by the route of a taxi going through city blocks [20].

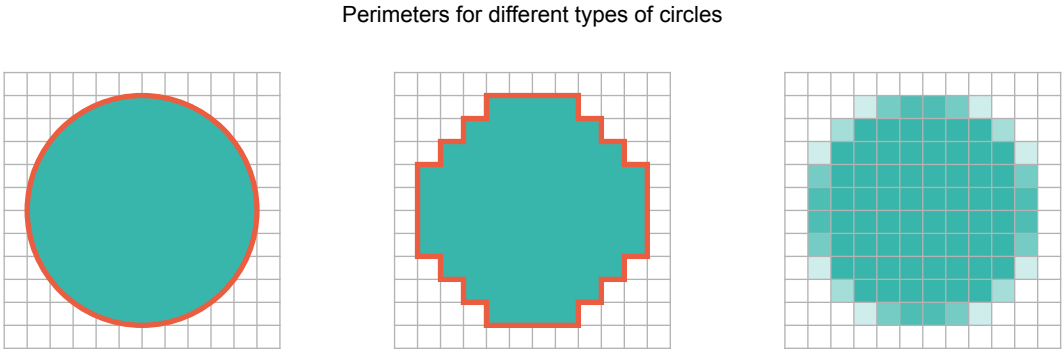


Figure C.1: Illustration of a circle (left), discretized in a binary domain (middle) and a continuous domain (right).

When the circle is discretized in a continuous domain (as shown in the right image of Figure C.1), there is no clear transition from void to solid between elements any more. Hence, a different method is required to find the perimeter in this domain.

The Total Variance (TV) is a suitable method of determining the perimeter of a continuous domain that does not have clear boundaries between the solid and void phase. It works by calculating the local gradient in a set of directions and integrating the sum of the gradients over the entire domain [21]. The simplest form for a 2-dimensional domain is the version that uses two directions (x_1 and x_2), called 'TV_2'.

$$\mathrm{TV}_2(\boldsymbol{\rho}) = \int_\Omega \left(\left| \frac{\partial \boldsymbol{\rho}}{\partial x_1} \right| + \left| \frac{\partial \boldsymbol{\rho}}{\partial x_2} \right| \right) dx = \sum_i \left(h_2 \left| \frac{\partial \rho_i}{\partial x_1} \right| + h_1 \left| \frac{\partial \rho_i}{\partial x_2} \right| \right). \tag{C.1}$$

When a discretized mesh is considered, the integral can be replaced with a summation over the elements, as shown in the equation above. The element dimensions h_1 and h_2 are multiplied with the local gradients to take their size into account. The assumption is made that all elements have the same dimensions, which is the case for all experiments shown in this report. With this formulation, the taxi-cab perimeter of discretized domains can be evaluated. However, the method developed in this research uses the perimeter to evaluate the shape factor (or circularity) of pockets. If the perimeter function

does not distinguish between squares and circles (as is the case for TV_2), the shape factor evaluation will not be accurate. This problem is solved by adding more local gradients in other directions to the summation shown in Equation C.1.

How more directions can be included is illustrated in Figure C.2. The left highlighted element has 2 arrows attached to it, pointing in the 2 directions that TV_2 uses. The middle element shows these direction for TV_4 and the right for TV_8. The arrows might suggest that the local gradient is computed using forwards or backwards differencing. However, the method is insensitive to this choice [21]. Finally, when using more than 2 local gradients in the function, the element dimension factors h are adjusted to ensure the correct scaling of the terms. A more detailed description of these factors and the complete formulation of TV_4 and TV_8 can be found in the paper by Petersson et al. [21].

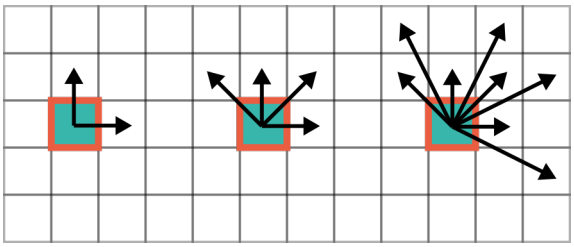


Figure C.2: Illustration of the local gradient directions used for TV_2, TV_4 and TV_8 (left to right).

How the accuracy of the perimeter evaluation depends on the number of gradients can be seen in Figure C.3. To make this plot, a square is defined in a domain and its perimeter is determined using TV_2 and TV_4. When the sides of the square align with the principle axes of the mesh, both methods are expected to obtain the exact perimeter. However, when the square is rotated such that is sides do not align with the mesh any more, an error in the perimeter evaluation shows. Because TV_2 only considers local gradients in 2 directions, its error goes up to 35%. On the other hand, the 2 extra local gradients that TV_4 uses, results in an exact perimeter evaluation when the square is rotated 45 degrees (or $\pi/4$ $\approx$ 0.78 radians). This happens because the diagonal local gradients now align with the mesh again. The result is that the perimeter evaluation using TV_4 is more accurate throughout the complete rotation of the square. Adding more local gradients makes the evaluation even more accurate, but does use more elements around the one that is considered.

In this research, the choice was made to use TV_8 in combination with a central differencing scheme to evaluate the perimeters. TV_8 was chosen for its high accuracy and central differencing was chosen because it gives the most accurate gradient [28]. It should be noted that a known pitfall of the central difference scheme is that it can yield a 0 gradient in an oscillating domain. This effect was observed in some experiments as well and discussed in Section 5.1.2.

C.2 Implementation

At the heart of the TV_8 implementation lies a summation over all elements of the input domain. The local gradient is determined in each direction by central differencing and its absolute value is multiplied with the element dimension factor and added to the total perimeter. Because the absolute operator is not differentiable, a smooth version ($\widetilde{\text{abs}}$) is used [10]:

$$\widetilde{\text{abs}}(x) = \sqrt{(1+2\epsilon)x^2 + \epsilon^2} - \epsilon. \tag{C.2}$$

This function uses parameter ϵ to control how smooth the approximation to the real absolute function is. The sensitivity of the output with respect to the input value can be found by differentiation:

$$\frac{\partial \widetilde{\text{abs}}(x)}{\partial x} = \frac{(1+2\epsilon)x}{\sqrt{(1+2\epsilon)x^2 + \epsilon^2}}. \tag{C.3}$$

With the smooth absolute function defined, the rest of the implementation is straight-forward as a summation over the elements and using the function to find the absolute value of the central differences:

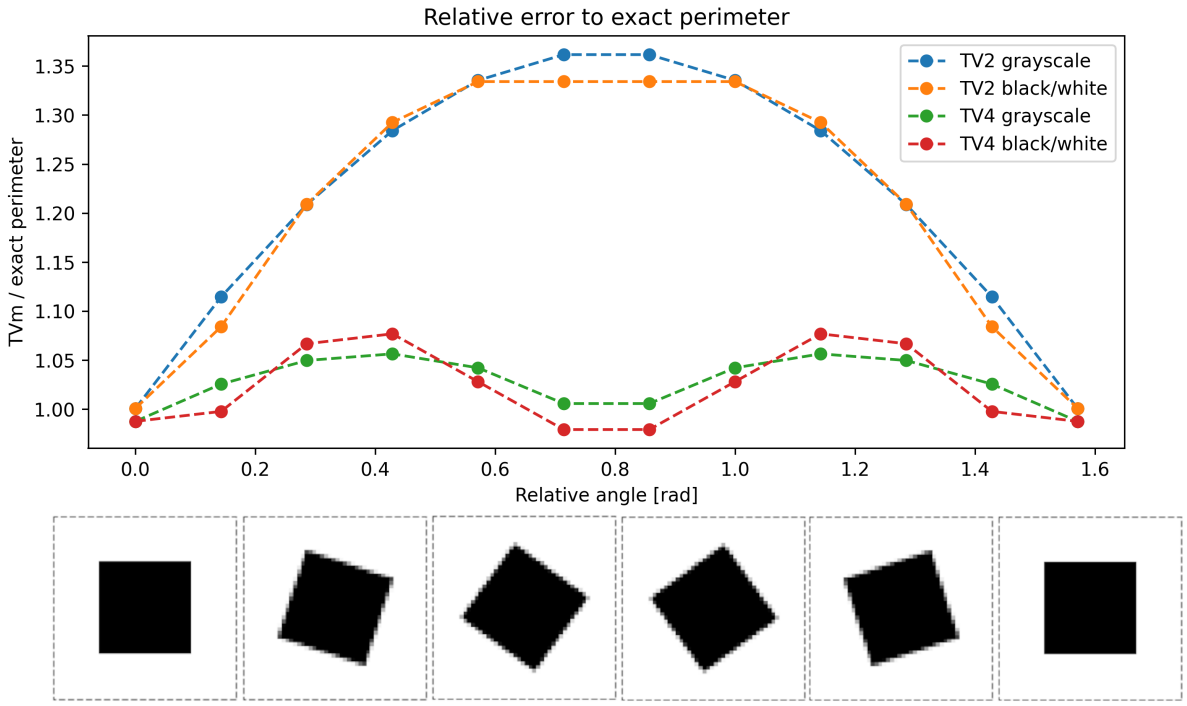


Figure C.3: Errors in perimeter evaluation when using TV_2 and TV_4 on binary and continuous domains containing a square at different angles. Some of these domains are shown below the plot for clarity.

$$TV_8(\mathbf{x}) = \sum_i \left[\sum_d \left(\frac{h_d}{2} \cdot \widetilde{\text{abs}}(x_{i,1}^d - x_{i,2}^d) \right) \right]. \tag{C.4}$$

Note that the h factors are divided by 2 because central differencing is used. In case of forward of backward differencing, the h factors are used directly.

In order to handle elements close to the boundary correctly, a padding should be added around the domain. In case of TV_8, this padding should be 2 elements wide because the central differencing uses values that are 2 rows or columns away from the considered element.

Appendix D
Milling filter

The milling filter was developed by Langelaar and is used in this work to distinguish internal from internal pockets [14]. The filter can be broken down into three basic steps, which will be explained in this appendix.

Step 1: Cumulative summing
The first step is to create a set of domains by performing a cumulative sum in a couple directions. For reasons of simplicity, the formulation in this research uses four directions, namely top-to-bottom, bottom-to-top, left-to-right and right-to-left. More directions can be added to make the method better suited for more complex geometries, but this was not deemed necessary for the experiments shown in this report. The cumulative sum function CS can be expressed as a sparse matrix $\mathbf{S}$, which is multiplied with the input domain $\mathbf{x}$ in vector form. The sensitivity of the function with respect to the input domain is therefore equal to the sparse matrix:

$$\mathrm{CS}_i(\mathbf{x}) = \mathbf{S}_i\mathbf{x}, \tag{D.1}$$

$$\frac{\partial \mathrm{CS}_i(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{S}_i. \tag{D.2}$$

In these equations, subscript i indicates the direction that is considered. Because these matrices are independent of the input domain, they can be pre-defined and re-used for every iteration of the optimization.

Step 2: Intersection
After the cumulative summing, the set of output domains (one for each direction that was considered) has to be recombined into a single domain. The value of each element in the output domain should be equal to the minimum value of the corresponding elements in the domains generated by the cumulative sums. This is achieved with a smooth minimum function to maintain differentiability of the filter. Langelaar chose a KS-function using a logarithm of summed exponentials, but this research found a different type of function (using a fraction of summed exponentials) to be giving better results. Appendix F is dedicated to the comparison between these two functions, namely LogSumExp and FracSumExp.

The functions used for the smooth minimum operation are sensitive to overflow if its input domains contain values that are much larger than 1. Because the cumulative sum method often yields domains that do contain these large values, an overflow protection step is required. Before the domains are used in the smooth minimum function, any value that is larger than 1 is replaced by the natural logarithm of that value. With this step, numerical problems in the smooth minimum function are avoided.

Step 2: Projection
The cumulative sum method creates domains with maximum values that can be larger than 1 (even after the overflow protection step has been applied) and therefore these can also show up in the intersected domain. To end up with a final domain with values in the desired range of 0 to 1, a smooth Heaviside projection is used. An explanation of this projection is included in Appendix A. This step concludes the milling filter and allows the internal pockets to be removed from a given domain.

Appendix E
Connected Component Labelling

The name 'Connected Component Labelling' is a term used to group many algorithms that can distinguish features from the background of the input domain. Their input is always a binary image, where each pixel either belongs to the background (background-pixel) or to a feature (feature- or object-pixel). In the context of these algorithms, the term 'pixel' can be interchanged with 'element' as it is used in the context of optimization. Because the input has to be binary, this usually means that pre-processing of the image is required, as images taken from a camera are usually not binary (nor are the domains generated by the optimizer). Figure E.1 shows how the algorithm works, where the image on the left is the pre-processed input.

In the end, the goal of the algorithm is to assign a label to each feature-pixel in the image, where pixels that belong to the same feature are given the same label. Many approaches have been implemented, but the most basic version is the so-called 'two pass method', which uses two scans to assign the labels. In the first pass, all feature-pixels are scanned, usually from left to right and top to bottom. The neighbours of each pixel are checked and depending on their labels, the current pixel is either given a new label or one of the neighbouring labels. If the current pixel connects two or more pixels with different labels, it means that those labels belong to the same feature. In that case, all the neighbouring labels are stored in an equivalence set and one of those labels is given to the current pixel. In Figure E.1, a total of 4 labels are assigned and the equivalence sets are $\{1, 2\}$ and $\{3, 4\}$ because pixels with labels 1 and 2 are connected and the same holds for pixels with labels 3 and 4.

After the first scan, all pixels belonging to features have been given a label, but pixels belonging to the same feature can still have different labels. However, the equivalence sets describe these relations and can be used to fix this. In the second pass, every label is changed to the label with the lowest number found in the equivalence set that the original label belongs to. If the original label does not belong to any equivalence set, it remains unchanged. After this second pass, all pixels belonging to a feature have been given the same label. In the example shown in Figure E.1, it can be seen that the labels with number 2 are changed to 1 and the labels with number 4 are changed to 3. These labels can now be used to extract the pixels belonging to each feature as shown in the rightmost image of Figure E.1 and the algorithm is finished.

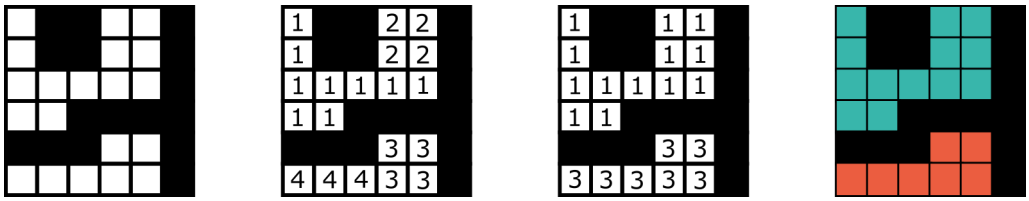


Figure E.1: Illustration of the two-pass CCL algorithm. Black pixels represent background while white pixels represent features. From left to right: input image, labels after first scan, labels after second scan, output.

Implementation for optimization

With a clear understanding of how the two-pass CCL algorithm works, its implementation in an optimization scheme can now be investigated. Because the SIMP method is used, densities in the design domain are represented as floating point numbers. Similarly to images taken with a camera, such a domain needs pre-processing to transform it into a binary domain where solid elements are represented with the value 1 and void is represented with value 0.

Pushing intermediate values to 0 or 1 is a common operation in optimization and can be done using the smooth Heaviside function (explained in Appendix A, Equation A.5). However, using the Heaviside function already poses a few problems. No matter how large the smoothness parameter β is made, an

input value that is not already equal to 0 or 1 will never be pushed to be exactly equal to 0 or 1. Output values can be pushed arbitrarily close to 0 or 1 by increasing β, but they will never reach them. This means that the smooth Heaviside function can be used to push intermediate values closer to 0 or 1, but to make a truly binary domain, it is not sufficient. Furthermore, no matter the value of β, the input value will remain unchanged if it is equal to the threshold value. While it is unlikely that a density value will be exactly equal to this threshold, it can happen.

Both problems can be mitigated by setting all values below the threshold equal to 0 and all values equal or above the threshold equal to 1. However, this operation is not differentiable, so doing this will introduce an error in the sensitivity calculation. By first applying a smooth Heaviside function, the error can be reduced, but it can not be completely avoided.

After pre-processing, which can not be done without introducing errors in the sensitivity calculations, the labelling process can start. Another problem arises when considering that all operations should be differentiable to allow proper sensitivity calculation. This problem is illustrated in Figure E.2 and comes down to the discrete nature of connectivity. Two features are either connected or not and there is no state in between these two. When features are only separated by a single 'bridging' element, changing it from 0 to 1 will reduce the number of features from 2 to 1. In combination with how the domain is pre-processed to become binary, this means that an infinitesimal change in the original input domain can reduce (or increase) the number of features. Since the number of features is always an integer number, this leads to infinite sensitivities for these bridging elements.

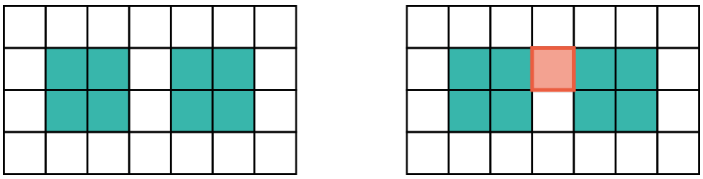


Figure E.2: Left: a binary domain with two unconnected features highlighted in blue. Right: the same domain with a bridging pixel highlighted in orange.

Concluding, the discrete nature of the two pass CCL algorithm introduces errors in the sensitivity analysis. It was also shown how an infinitesimal change in the input domain can lead to integer changes in the output, causing more problems in the sensitivity analysis. Since these effects are not specific to the two pass strategy, but hold for the complete group of CCL algorithms, these algorithms are deemed not suitable for direct use in gradient-based optimization where densities are represented with floating point numbers. Perhaps CCL-inspired algorithms for non-binary domains can be conceived that do maintain differentiability, but this direction is outside the scope of this research. Instead, an alternative approach to deal with multiple internal pockets is used as explained in Section 3.5.

Appendix F

Comparison of smooth minimum functions

In order to determine the minimum value from a set in a differentiable manner, a smooth minimum function can be used. When the input is a set of matrices and the minimum should be determined element-wise, this operation is also known as 'intersection'. Several functions with different characteristics can be found to implement this operation. This appendix investigates the differences between two common smooth minimum functions, namely LogSumExp and FracSumExp, to determine which function is the most suitable for use in domain composition as described in Section 3.5.2.

LogSumExp

The LogSumExp is defined as shown in Equation F.1. It determines output value ρ_j based on m inputs $P_j^{(i)}$. Constant C_I (constant of intersection) is used to determine the smoothness of the function. The sign of C_I determines whether the function acts as a smooth minimum (negative C_I) or maximum (positive C_I) [14].

$$\rho_j = \frac{1}{C_I} \ln\left(\frac{1}{m}\sum_{i=1}^{m} e^{C_I P_j^{(i)}}\right). \tag{F.1}$$

To visualize the characteristics of this function, a plot is made by supplying two inputs to the LogSumExp function, shown in Figure F.1. The inputs are $y = x$ and $y = -x$, which means that the exact minimum equals $y = x$ when $x < 0$ and $y = -x$ when $x > 0$. At $x = 0$, the exact minimum shows a discontinuity in its slope which makes it non-differentiable. Different values of C_I are used to show its influence on the output. Because the maximum difference between inputs is 1 in the context of topology optimization, the plot is made in the range $-0.5 < x < 0.5$ (at $x = -0.5$ and $x = 0.5$, the difference between input equals 1).

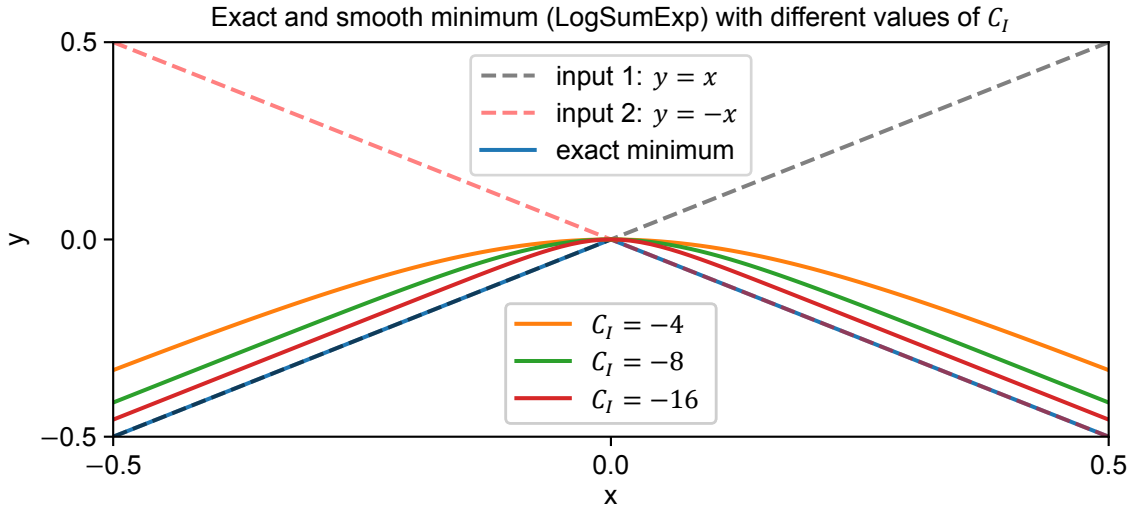


Figure F.1: Output of exact minimum and smooth LogSumExp function using different values of C_I with inputs $y = x$ and $y = -x$.

When looking at the figure, it can observed that the output of the LogSumExp function approaches the exact minimum as the absolute value of C_I is increased. It can also be seen that the functions converge on the same slope as the exact minimum, but not on the same value. While all functions match the exact minimum at $x = 0$ (no difference between inputs), as the difference between inputs is increased, an error appears that does not disappear as the difference is further increased.

In the context of domain compilation for topology optimization, this error has an undesired effect on the sensitivity calculations. When objectives or constraints are calculated with the compiled domain, their sensitivities with respect to the input domains must be calculated. Because this error in the smooth minimum is not zero, these objectives or constraints will have a non-zero sensitivity with respect to all input domains. In the ideal case, the objectives and constraints are only sensitive to changes in the minimum elements.

FracSumExp

The second function that will be investigated is the FracSumExp function, which is defined as shown in Equation F.2. Similarly to the LogSumExp function, constant of intersection C_I is used to determine the smoothness of the function, where a negative value makes the function act as a smooth minimum and a positive value makes it act as a smooth maximum.

$$\rho_j = \frac{\sum_{i=1}^{m} P_j^{(i)} e^{C_I P_j^{(i)}}}{\sum_{i=1}^{m} e^{C_I P_j^{(i)}}}. \tag{F.2}$$

To visualize this function and compare it to the LogSumExp function, the same plot is made as before, with $y = x$ and $y = -x$ as the two inputs. The result is shown in Figure F.2. A first observation is that just like LogSumExp, this function matches the exact minimum at $x = 0$. However, the behaviour when x moves away from 0 (and the difference between inputs is increased) is different from the LogSumExp behaviour. The error that appears when the difference between inputs is increased converges to zero instead of remaining constant.

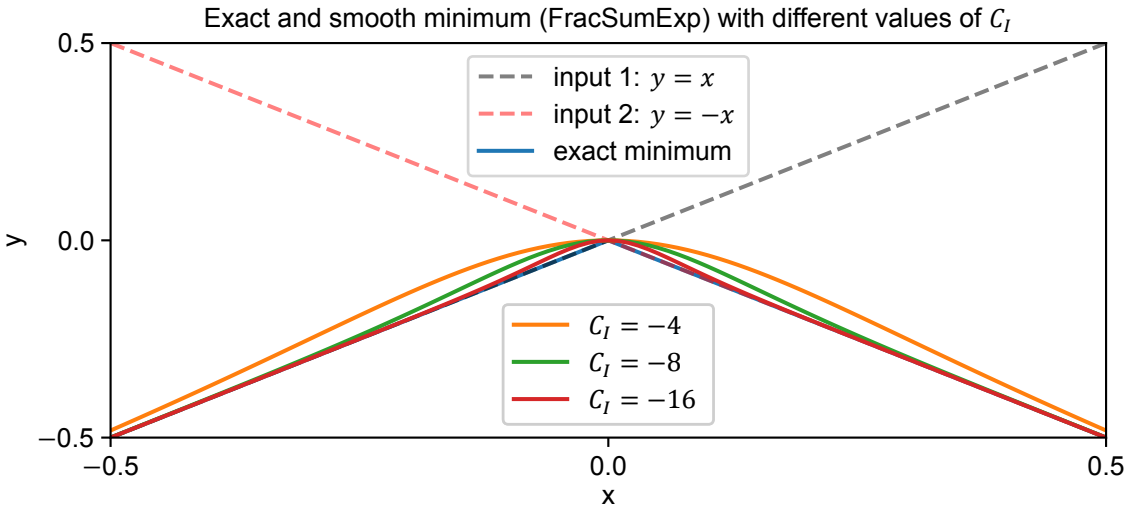


Figure F.2: Output of exact minimum and smooth FracSumExp function using different values of C_I with inputs $y = x$ and $y = -x$.

Comparison

When comparing the LogSumExp and FracSumExp functions, two properties are most interesting to look at in the context of topology optimization. The first property is the error between function's output and an exact minimum when all inputs are 1 and a single input is 0. This would be the situation in a converged topology optimization where each pocket is defined by a single pocket domain. When

objectives and constraints are determined with the compiled domain, their sensitivities with respect to the pocket domains have to be determined. If the error in the smooth minimum is non-zero, the objectives and constraints will be sensitive to all input elements, while they should ideally only be sensitive to the element with the lowest density value.

On first observation, it seems like the FracSumExp function is a better choice because its error converges to zero while the error of the LogSumExp remains constant. However, the error of both functions can be made arbitrarily close to zero by increasing the absolute value C_I. Because of this, no choice can be made between the functions solely based on the behaviour of the error.

The second interesting property of the functions is their curvature and how it behaves when the difference between inputs is increased. Because the optimizer uses a gradient-based update scheme, large peaks in the curvature will hinder convergence as this is non-linear behaviour. Hence, in order to choose the best function to use, this curvature should be taken into account.

The starting point of this comparison is to choose a certain error which is allowed when the input difference is 1. The constants of intersection can then be found such that both functions produce this error. Note that in order to make both functions produce the same error, different values for C_I are required. The next step is to compare the curvatures of both functions, where the function with lowest peak curvature is considered to be the better choice.

The results of this comparison are displayed in Figure F.3. The three errors which were chosen are 0.12, 0.06 and 0.03 (12%, 6% and 3%), corresponding to the blue, orange and green lines respectively. Solid lines in both plots correspond to the LogSumExp function while dashed lines correspond to the FracSumExp function. The left plot in this figure shows how the errors develop when the input difference is increased. The right plot shows the curvature of the functions when their constants of intersection are tuned to produce these errors. In this right plot, it can also be seen that for every error considered, the FracSumExp function has a lower peak in the curvature than the LogSumExp function. This means that for every error considered, the FracSumExp function is a better choice. The conclusion is therefore made that the FracSumExp function is the better choice in the context of topology optimization.

It should be noted that the errors produced by the FracSumExp function when the input difference is smaller than 1 are always larger than those produced by the LogSumExp function. This means that until the solution is converged, the undesired coupling between input domains and the objectives and constraints will be stronger. Whether this effect has a larger negative effect on the convergence than the peak curvature could be determined by experimentation, but due to time constraints this was not done in this research.

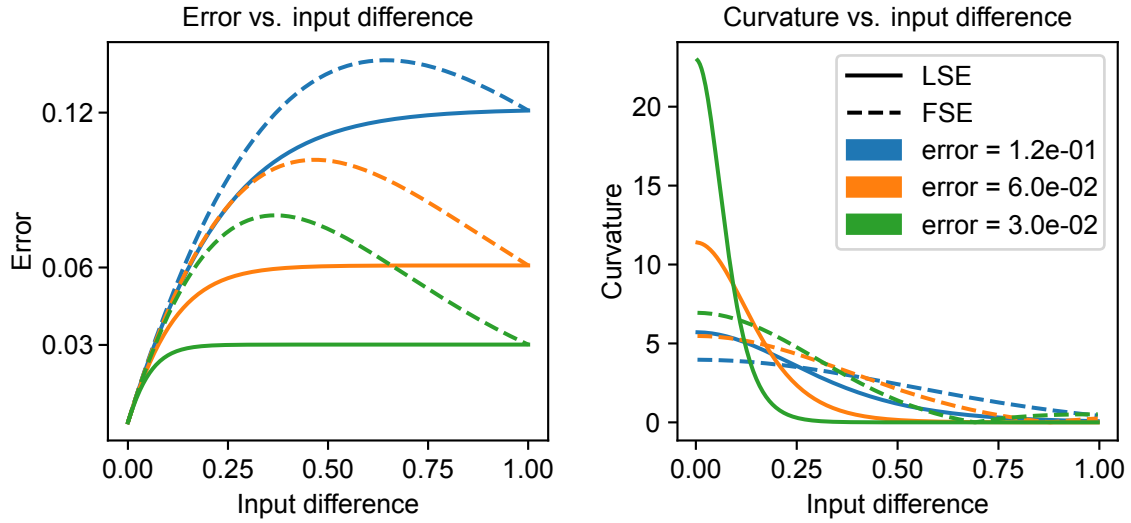


Figure F.3: Comparison between LogSumExp and FracSumExp functions. Left: error for both functions tuned for three target errors versus the input difference. Right: curvature of both functions tuned for target error versus the input difference.

Bibliography

[1] Personal communication, May 18 2021.

[2] D Ben-Arieh. Cost estimation system for machined parts. *InternationalJournal of Production Research*, 38(17):4481–4494, 2000. doi: 10.1080/00207540050205244. URL https://www.tandfonline.com/doi/abs/10.1080/00207540050205244?tab=permissions&scroll=top.

[3] M. P. Bendsøe and O. Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer, 2003.

[4] B. Bourdin. Filters in topology optimization. *International Journal for Numerical Methods in Engineering*, 50(9):2143–2158, 2001. ISSN 00295981. doi: 10.1002/nme.116.

[5] E. R. Davies. *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Elsevier, 4 edition, 2012. ISBN 9780123869081.

[6] P. Dewhurst and G. Boothroyd. Early cost estimating in product design. *Journal of Manufacturing Systems*, 7(3):183–191, 1988. ISSN 02786125. doi: 10.1016/0278-6125(88)90003-9.

[7] J. K. Guest. Topology optimization with multiple phase projection. *Computer Methods in Applied Mechanics and Engineering*, 199(1-4):123–135, 2009. ISSN 00457825. doi: 10.1016/j.cma.2009.09.023.

[8] J. K. Guest, J. H. Prévost, and T. Belytschko. Achieving minimum length scale in topology optimization using nodal design variables and projection functions. *International Journal for Numerical Methods in Engineering*, 61(2):238–254, 2004. doi: 10.1002/nme.1064. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-4444271810&doi=10.1002%2Fnme.1064&partnerID=40&md5=ddc175f2e5934f2fcf9dde5742b27379.

[9] S. K. Gupta, W. C. Regli, D. Das, and D. S. Nau. Automated manufacturability analysis: A survey. *Research in Engineering Design - Theory, Applications, and Concurrent Engineering*, 9(3):168–190, 1997. ISSN 09349839. doi: 10.1007/BF01596601.

[10] R. B. Haber, M. P. Bendsøe, and C. S. Jog. Perimeter Constrained Topology Optimization of Continuum Structures BT - IUTAM Symposium on Optimization of Mechanical Systems. pages 113–120, Dordrecht, 1996. Springer Netherlands. ISBN 978-94-009-0153-7.

[11] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao. The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition*, 70:25–43, 2017. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.04.018.

[12] Lester O Hedges, H Alicia Kim, and Robert L Jack. Stochastic level-set method for shape optimisation. *Journal of Computational Physics*, 348:82–107, 2017. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2017.07.010. URL https://www.sciencedirect.com/science/article/pii/S0021999117305120.

[13] R. Janardan, M. Smid, and D. Dutta. Machinability: Geometric Reasoning for Cutting. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 67:251–252, 2005. doi: 10.1090/dimacs/067.

[14] M. Langelaar. Topology optimization for multi-axis machining. *Computer Methods in Applied Mechanics and Engineering*, 351:226–252, 2019. ISSN 0045-7825. doi: 10.1016/j.cma.2019.03.037.

[15] P.R. Mercer. *More Calculus of a Single Variable*. Springer, 2014. ISBN 9781493919253.

[16] David F. Noble. *Forces of Production*. Knopf, 1 edition, 1984. ISBN 9780394512624.

[17] E. Oberg and F. D. Jones. *Machinery's Handbook*. Industrial Press, New York, New York, 1 edition, 1914.

[18] A. Overby. *CNC Machining Handbook*. Mcgraw-Hill Education - Europe, 1 edition, 2010. ISBN 9780071623018.

[19] Tanisha Pereira, John V. Kennedy, and Johan Potgieter. A comparison of traditional manufacturing vs additive manufacturing, the best method for the job. *Procedia Manufacturing*, 30: 11–18, 2019. ISSN 23519789. doi: 10.1016/j.promfg.2019.02.003. URL https://www.sciencedirect.com/science/article/pii/S2351978919300332.

[20] J. Petersson. Some convergence results in perimeter-controlled topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 171(1):123–140, 1999. ISSN 0045-7825. doi: 10.1016/S0045-7825(98)00248-5.

[21] J. Petersson, M. Beckers, and P. Duysinx. Almost Isotropic Perimeters in Topology Optimization: Theoretical and Numerical Aspects. jun 1999.

[22] C. Poli. *Design for manufacturing: a structured approach*. Elsevier, 2001. ISBN 978-0-7506-7341-9.

[23] A J G Schoofs. Structural Optimization History and State-of-the-Art BT - Topics in Applied Mechanics: Integration of Theory and Applications in Applied Mechanics. pages 339–345. Springer Netherlands, Dordrecht, 1993. ISBN 978-94-011-2090-6. doi: 10.1007/978-94-011-2090-6_37. URL https://doi.org/10.1007/978-94-011-2090-6_37.

[24] O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, 2001. ISSN 1615147X. doi: 10.1007/s001580050176.

[25] 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, 1998. ISSN 09344373. doi: 10.1007/BF01214002.

[26] S. L. Vatanabe, T. N. Lippi, C. R. Lima, G. H. Paulino, and E. C. N. Silva. Topology optimization with manufacturing constraints: A unified projection-based approach. *Advances in Engineering Software*, 100:97–112, 2016. ISSN 18735339. doi: 10.1016/j.advengsoft.2016.07.002.

[27] F. Wang, B. S. Lazarov, and O. Sigmund. On projection methods, convergence and robust formulations in topology optimization. *Structural and Multidisciplinary Optimization*, 43(6):767–784, 2011. ISSN 1615147X. doi: 10.1007/s00158-010-0602-y.

[28] Young and Mohlenkamp. Numerical Differentiation. Technical report, Ohio University, 2021.