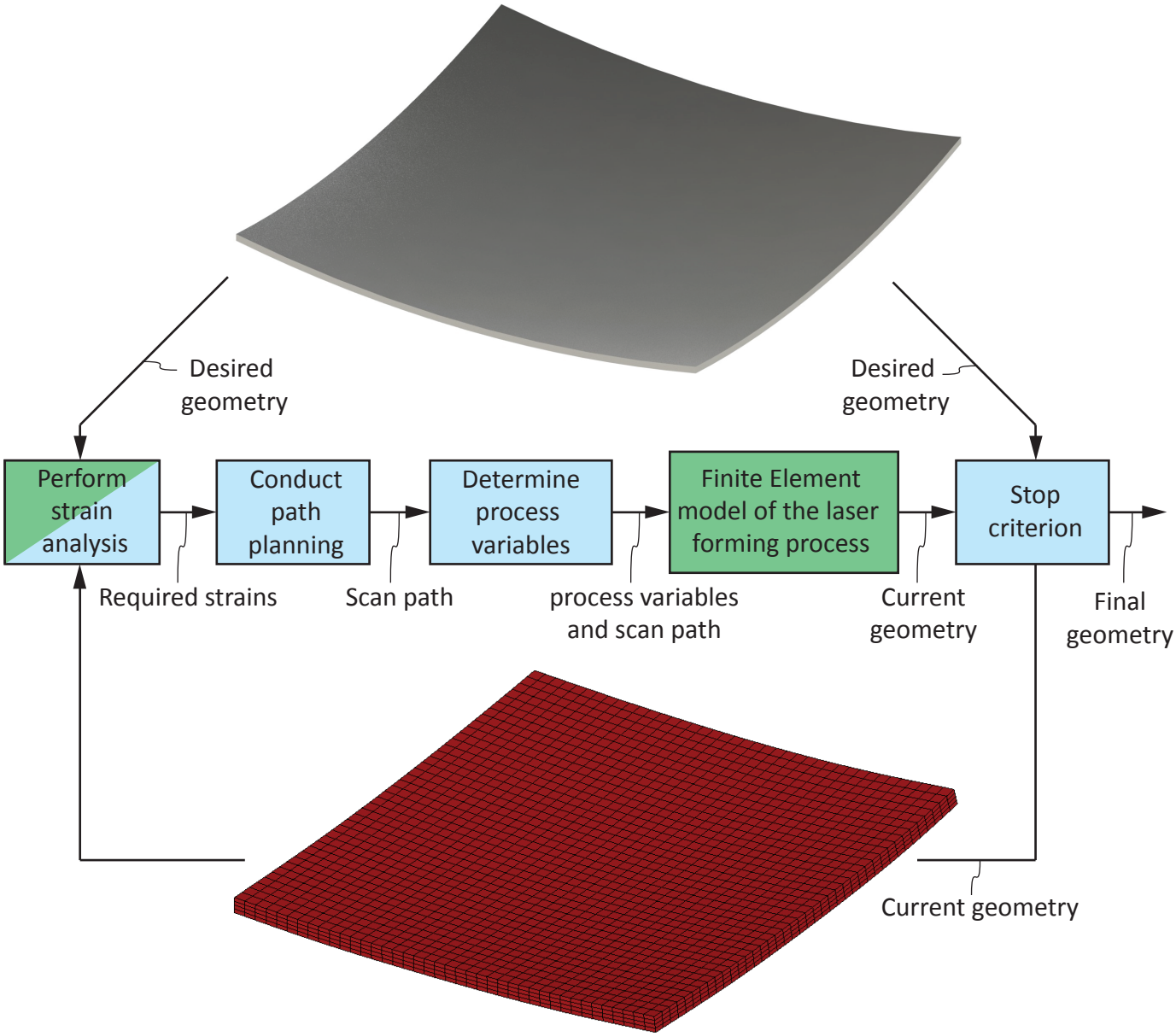
Development of a Framework for the Laser Forming Process

- For Forming of Double Curved Geometries

Master Thesis
3/02-3/06-2014



Group VT4-2.215A
Fibigerstraede 16
DK - 9220 Aalborg East

AALBORG UNIVERSITY
STUDENT REPORT

Department of Mechanical and Manufacturing Engineering
Fibigerstraede 16
DK-9220 Aalborg East
http://m-tech.aau.dk

Title:
Development of a Framework for the Laser Forming Process - For Forming of Double Curved Geometries

Theme:
Master Thesis

Project Period:
Spring Semester 2014

Project Group:
VT4-2.215A

Participants:

Kasper Madsen

Martin Søndergaard

Supervisors:
Benny Endelt
Morten Kristiansen

Copies: 5

Page Numbers: 79

Date of Completion:
June 03, 2014

Synopsis:

The objective of the master thesis is to develop a framework for the laser forming process, which enables the production of a set of desired single and double curved geometries.
The framework is based on a feedback loop which utilises the required amount of strain to go from current geometry to the desired geometry. A scan path algorithm is developed, which determines the placement of the scan paths. As the laser forming process induces compressive strains perpendicular to a given scan path, it is desirable to place the scan paths perpendicular to the orientation of the minimum principal strains. Based on a ratio between bending and in-plane strains, a suited forming mechanism is selected for the scan path. The forming mechanisms are predefined with regards to laser power and laser beam diameter, while the scan speed is selected as the control variable and adjusted as a function of the required strains. The framework iterates until a stop criterion is met, which is based on the sum of absolute error between the obtained geometry and the desired geometry.
The geometries obtained by use of the framework exert the appearance of the chosen set of desired single and double curved geometries. Thereby, validating the potential of the developed framework. However, deviations between the obtained and the desired geometries are present, why improvements are required.
The developed framework allows the production of double curved geometries. However, further work is required to ensure that the process limitations are properly established and that the developed framework functions correctly in a physical setup.

Resume

Hovedformålet med nærværende projekt, er at udvikle et framework, der muliggør produktionen af dobbeltkrumme emner i plade materiale, ved brug af laserformgivningsprocessen. Behovet for at undersøge, hvorvidt dobbeltkrumme pladedele kan formgives ved brug af laserformgivningsprocessen er affødt af en stigende efterspørgsel til kundetilpassede løsninger, hvilket stiller krav til høj fleksibilitet i produktionsanlægget. Konventionelle produktionsmetoder til pladedele fx dybtrækning og hydro formgivning, er ikke fleksible og dermed ikke egnede til at producere kundetilpassede løsninger, idet nye geometrier kræver dyr udvikling og fremstilling af formgivnings værktøjer. Derfor er der stort potentiale i en produktionsmetode der tillader høj fleksibilitet og dermed produktion af kundetilpassede løsninger. Idet laserformgivningsprocessen ikke kræver nogen formgivningsværktøjer og nye geometrier kan formgives ved justering af procesindstillinger, er det vurderet, at laserformgivningsprocessen er egnet til at producere kundetilpassede pladedele.

Projektet er udført gennem to semestre, henholdsvis 9. semester og kandidatspecialet. Gennem 9. semester lå fokus på at opnå kendskab til laserformgivningsprocessen samt udvikle et kontrolsystem, der sikrede at et V-buk på 10 grader kunne produceres med høj repeterbarhed. Til bestemmelse af parametre til kontrolsystemet, blev der udviklet en ikke-lineær simuleringsmodel. Som nævnt, er formålet med nærværende kandidatspeciale, at udvikle et framework der muliggør produktionen af dobbeltkrumme pladedele ved brug af laserformgivningsprocessen. Frameworket er udviklet som et software program. Udvikling samt evaluering af det endelig framework er sket ved hjælp af den tilegnede viden og simuleringsmodellen der blev udviklet gennem 9. semester.

Indledningsvist er laserformgivningsprocessen beskrevet for at etablere et grundliggende kendskab til processen og de formgivningsmekanismer, der er nødvendige for at producere dobbeltkrumme pladedele. Kendskabet indebærer, hvorledes mekanismerne aktiveres og kontrolleres. Laserformgivningsprocessen afhænger af adskillige materialeparametre og procesvariable. Processen er sårbar overfor varians i disse, eftersom dette medfører varians i slutproduktet. Derfor er der benyttet en konservativ bestrålingsstrategi samt en tilbagekoblingssløjfe, hvor emnet bestråles ad flere gange for at opnå den ønskede geometri. Mellem hver bestråling kontrolleres processen med henblik på at minimerer fejlen mellem den ønskede og den tilbagekoblede nuværende geometri. Bestrålingsmønsteret samt procesvariablene, der bestemmer henholdsvis hvor og hvor meget formgivning, der skal foretages, bestemmes ved hjælp af en tøjningsanalyse. Tøjnings-

analysen foretages i hver iteration af tilbagekoblingssløjfen. Tøjningsanalysen foretages ved brug af en simuleringsmodel, der indikerer størrelsen samt placeringen af de resterende tøjninger, der kræves for at gå fra nuværende til ønsket geometri. Bestrålingsmønsteret placeres vinkelret på de mindste hovedtøjninger i det resterende tøjningsfelt, eftersom de største kompressive tøjninger opnået under formgivning optræder vinkelret på bestrålingsmønsteret. Procesvariablene beregnes ud fra størrelsen af de resterende tøjninger der er i bestrålingsmønsteret. Da laserformgivningsprocessen leverer en varierende mængde formgivning på tværs af en kvadratisk blanket, grundet asymmetri i processen. Er det valgt at variere hastigheden af laseren henover bestrålingsmønsteret for at styre mængden af formgivning. Variationen i hastighed bestemmes ved hjælp af en fordelingsfunktion, da dette tillod hurtig implementering i program strukturen. Bestrålingsmønster samt procesvariable benyttes i en simuleringsmodel af laserformgivningsprocessen hvilket resulterer i en formgivet plade. Den opnåede geometri holdes op imod et stop kriterie, der tjekker den geometriske overensstemmelse mellem den opnåede og den ønskede geometri. Er stopkriteriet ikke opfyldt, køres tilbagekoblingssløjfen igen.

Det udviklede framework er testet på fire forskellige geometrier, hvoraf to af geometrierne kan betragtes som enkeltkrumme geometrier og de resterende dobbeltkrumme. Ved brug af det udviklede framework opnås overensstemmelse mellem de fremstillede geometrier og de ønskede geometrier. Dog indikerer afvigelser fra de ønskede geometrier at frameworket kan forbedres. En række af disse forbedringer af frameworket diskuteres slutteligt, med afsæt i resultaterne fra de fire tests.

Preface

This report contains the master thesis, written by the Manufacturing Technology group VT4-2.215A, studying at the Department of Mechanical and Manufacturing Engineering at Aalborg University. The project is conducted in the period between 03.02.14 and 03.06.14. The report must be perceived as further work of the groups earlier work performed during 9th. semester, however, the current report is written such that it can be read independently of the earlier work. The 9th. semester report was written with respect to the project proposal: *Laser Forming* stated by GRUNDFOS Holding A/S and is appended on the appendix-CD. The present report is based on the problem statement:

. . . . . .

"How can a framework capable of producing double curved geometries by utilisation of the laser forming process be developed?"

. . . . . .

The report is written in LaTeX, software programs are written in Java, plots are generated using gnuplot and Finite Element Analyses are solved in LS-Dyna on the Linux-cluster located at the Department of Mechanical and Manufacturing Engineering at Aalborg University. Calculations are performed using Java, Maple and Mathcad.

Great appreciation is dedicated to our supervisors Benny Endelt and Morten Kristiansen.

Reading Instructions

This thesis contains a main report with an appendix placed in the end. The report can be read independently of the appendix, but for elaborating information, references are made to the appendix. To maintain the flow in the report, programs are explained using pseudo code. A commented version of the source code is appended in the appendix. Furthermore, all code is found on the attached appendix-CD as is the main report and appendix. Source references in the report are made according to the syntax [author, year], with the bibliography in the end of the report. Titles, equations, tables and pictures are numerated in the format $x.y$, where x indicates the chapter number and y indicates the consecutive number.

Aalborg University, June 3, 2014

Contents

Nomenclature

α_i	Angle between movement option vector and the orientation of minimum principal strains	[°]
α_{max}	Allowable angular deviation with respect to perpendicularity	[°]
β	Obtained bend angle	[°]
$\theta_2(m,n)$	Field representation of the orientation of the minimum principal strains	[-]
θ_i	Orientation of principal strains	[-]
$\varepsilon'_{i,j}$	Rotated strain tensor	[-]
$\varepsilon_{i,j}$	Strain tensor	[-]
$\varepsilon_i(m,n)$	Strain field for a discretised surface	[-]
$\varepsilon_i(x,y)$	Strain field for a continuous surface	[-]
A	Absorbance coefficient	[-]
c	Iteration counter for the scan path algorithm	[-]
d	Laser beam diameter	[m]
I	Heat flux density	[$\frac{\mathrm{W}}{\mathrm{m}^2}$]
k	Iteration counter for the framework	[-]
m	Element discretisation in the x direction	[-]
n	Element discretisation in the y direction	[-]
P	Laser power	[W]
p_i	Point for the scan path	[-]
R	Laser beam radius	[m]
r	Distance from center of laser beam	[m]
V_i	Scan speed	[$\frac{\mathrm{mm}}{\mathrm{min}}$]

Chapter 1

Introduction to the Master Thesis

The introduction consists of; a brief description of the challenges related to modern manufacturing, a description of the impact these challenges has on a manufacturing facility and how these must be coped with and an introduction to the Laser Forming Project. Furthermore, an overview of the content of the report is given.

1.1 Challenges in Modern Manufacturing

A main objective within larger modern manufacturing companies has been to obtain a cost efficient production, in order to comply with their customers demand for low prices. Traditionally, a cost efficient production is obtained using a rigid mass production line, which is composed of manufacturing equipment dedicated to specific tasks [Kalpakjian and Schmid, 2006]. This results in a rigid system optimised for the production of only one product. However, customers have started to require solutions customised for their specific requirements [Kumar, 2008]. The rigid mass production line is not suited for the production of customer specific solutions, why it is necessary to incorporate or develop new manufacturing technologies which allow a higher level of flexibility. The rigid production line may consist of several different manufacturing technologies. In the present project it is chosen to focus on a flexible alternative to conventional sheet metal forming processes.

1.2 Facing the Challenge in the Industry

Sheet metal is used in a wide range of products, with applicability within the automotive industry, the aerospace industry, etc. For most applications the sheet metal must undergo a forming operation to create either a single or double curved geometry, in order to comply with functional or aesthetic requirements in the end product [Liu and Yao, 2005].

Sheet metal can be formed by using several different forming processes. Conventional processes used to obtain double curved geometries are e.g. sheet hydroforming, press stamping and deep

drawing. However, these processes require costly changeover time[1] and development and manufacturing of new dies, whenever a new geometry is to be produced. To avoid costly development and manufacturing of new dies for each customer specific product, it is necessary to utilise an alternative manufacturing technology. The alternative technology must be capable of producing similar geometries as the aforementioned processes, but without the drawback of changeover times, and costly development and manufacturing of dies. A possible alternative is the laser forming process, which is treated in this project.

The laser forming process is based on earlier thermal forming processes, such as flame forming, utilised in the ship building industry. Laser forming enables the possibility of producing parts with a double curved surface, without the use of any forming dies. The forming is performed by introducing a controlled amount of thermal expansion in a workpiece, henceforth referred to as a blank, thereby obtaining a permanent plastic deformation. Examples of components formed with the laser forming process are shown in figure 1.1.

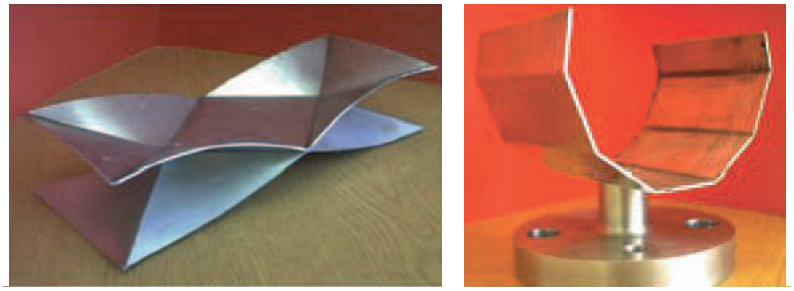


Figure 1.1: Examples of laser formed components (revised from [Dahotre and Harimkar, 2008] and [Thomson and Pridham, 1998]).

Several final geometries can be obtained from one blank geometry, by programming different process variables. As geometries can be formed by adjustment of the process variables and without the need of expensive dies, the laser forming process is assessed suited for the production of customised sheet metal parts. [Dahotre and Harimkar, 2008]

Besides increased flexibility, the laser forming process may also promote a competitive advantage with respect to rapid prototyping. As the conventional forming processes require development and manufacturing of new dies for each desired geometry, these processes may be considered fixed. Whereas the laser forming process only requires a new set of process variables in order to manufacture a new geometry. This potentially reduces the lead time from idea to finished product.

[1] Changeover consists of the tasks performed when changing to another product e.g. changing of tools in a conventional forming process.

1.3 The Laser Forming Project

To ensure successful implementation of the laser forming process with regards to industrial applications, it is necessary to investigate several different subjects e.g. process design, profitability, achievable geometries, achievable tolerances, achievable process lead times, effect on material properties etc.

The present project focuses on the subject of process design i.e. which process variables are required to form a desired geometry? This subject is chosen as the group has interest within the field of process design. To solve the subject of process design, two projects are conducted, the master thesis project and a former 9th. semester project, which is appended on the appendix-CD. The master thesis must be considered as an extension of the 9th. semester project: "Control System Development - for The Laser Forming Process" [Madsen and Søndergaard, 2013]. However, the present report is written independently of the 9th. semester report, such that a new reader does not have to read the former report.

Figure 1.2 shows the structure of the 9th. semester project. The main objective was to answer the problem statement:

"How can a controlled V-bend be obtained, when using the laser forming process?"

In order to answer this problem statement, the following subjects were treated; obtaining knowledge about the laser forming process, development of an experimental setup, development of a feedback control system, development and evaluation of a Finite Element model, establishment of controller parameters (gains) and tests of the feedback control system. It was concluded that it is possible to design a feedback control system which ensures process stability with regards to bend angle in rectangular stainless steel blanks, in this case a bend angle of $10\pm0.1^\circ$ was produced using the Finite Element model. However, during experimental testing it was observed that the measurement equipment used, did not allow sufficient precision, why only a tolerance of $\pm0.6^\circ$ could be obtained.

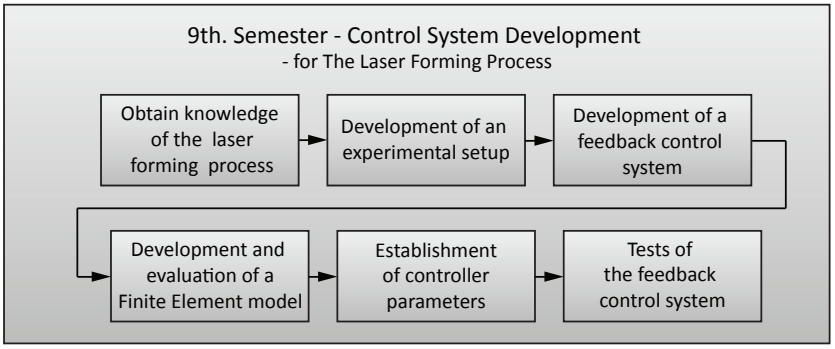


Figure 1.2: Structure of the 9th. semester project.

The work performed during the 9th. semester project assured that a proper understanding of the laser forming process was obtained. The obtained knowledge supports the work in the master thesis. To expand the applicability of the laser forming process, the next step is to develop a structured method i.e. a framework, which enables the transformation of a desired double curved geometry into suitable process variables. This leads to the problem statement for the master thesis.

"How can a framework capable of producing double curved geometries by utilisation of the laser forming process be developed?"

To answer this problem statement the subjects in figure 1.3 are treated throughout the present project. The listed subjects simultaneously define the structure of the master thesis.

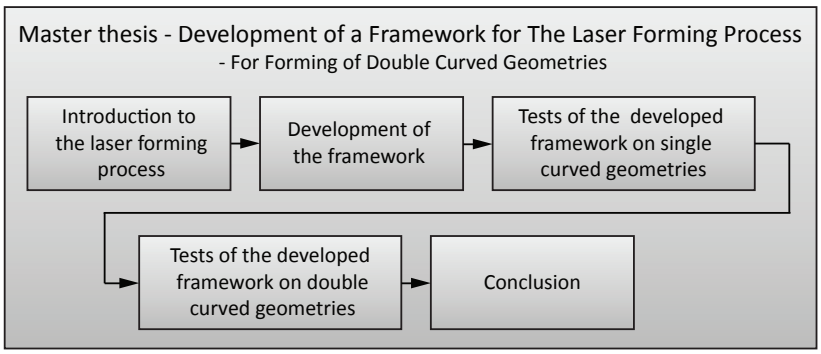


Figure 1.3: Subjects treated in the master thesis.

The laser forming process is described in chapter 2. The chapter describes; the laser forming process in general and the different forming mechanisms required to produce double curved geometries. Moreover, a Finite Element model developed in [Madsen and Søndergaard, 2013] is introduced. An adjusted version of the Finite Element model is used to develop and test the framework, throughout the master thesis.

In chapter 3, a concept for the framework is developed. The concept consists of several structured tasks related to the conversion of a desired geometry to suitable process variables.

Chapter 4 and 5 consists of four tests performed with the developed framework with regards to two single curved and two double curved geometries respectively. The purpose of the tests is to proof the concept of the developed framework.

The conclusion of the project is stated in chapter 6. The conclusion is composed of conclusions obtained throughout the entire report, as well as a conclusion to the problem statement. Finally, suggestions for future work is discussed in chapter 7.

Chapter 2

Process Description

This chapter focuses on establishing an understanding of the laser forming process, as this is essential before designing a framework capable of producing double curved geometries. Furthermore, the Finite Element model of the laser forming process, developed in [Madsen and Søndergaard, 2013], is introduced. Initially a description of the laser forming process is given. Following the process description, the concept of developable and non-developable surfaces is introduced along with the thermo-mechanical mechanisms utilised in the laser forming process to produce these surfaces. To aid the explanation of the mechanisms and introduce the Finite Element model the mechanisms are evaluated with respect to temperature distribution, strain distribution and forming using the Finite Element model.

2.1 Description of the Laser Forming Process

The laser forming process forms a blank by irradiating the surface with a defocused laser beam. Throughout this project a stainless steel 1.4301(AISI 304) blank with the material properties appended in appendix A and dimensions equal to 30x30x1mm is used. Figure 2.1 shows the terminology related to the laser forming process, which is described in the following. The description is constructed as an enumerated sequence of steps from the initial laser irradiation to final deformation of the blank. The description is based on [Dearden and Edwardson, 2003] and [Dahotre and Harimkar, 2008].

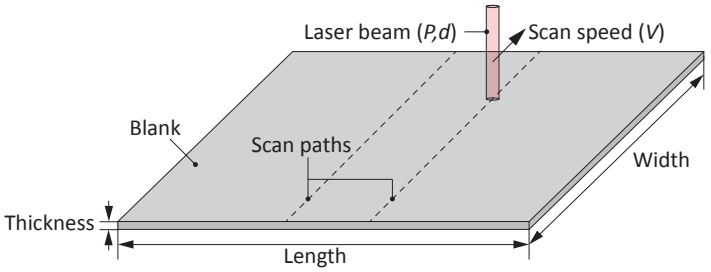


Figure 2.1: The concept of the laser forming process. A defocused laser beam with laser power P and laser beam diameter d irradiates a blank along predefined scan paths with a scan speed V.

1. **Laser irradiation:** The blank is irradiated with a defocused laser beam at a given laser power P and laser beam diameter d. The laser beam follows a scan path, denoted with dashed lines in figure 2.1, at a given scan speed V. The scan paths determine where the blank is irradiated. These can be placed arbitrarily on the surface to produce different geometries. The scan path, P, d and V are all process variables of the laser forming process. However, the term process variable henceforth refer to P, d or V. Furthermore a complete irradiation of a scan path is henceforth referred to as a laser scan.

2. **Laser absorbance:** The laser beam consists of electromagnetic waves. A part of the energy in the electromagnetic waves is absorbed by the blank material, as the laser beam traverses the blank. The remaining is either reflected or transmitted. The absorbance of electromagnetic waves occurs as the electrons of the material are forced into motion or excited by the electromagnetic waves. This results in an increase of internal energy in the blank material.

3. **Heat generation:** The increase in internal energy leads to a local generation of heat in the blank material. As heat is generated in the blank, various heat transfers occur i.e. conduction of heat into the blank, and convection and radiation from the surface of the blank. Dependent of the amount of generated heat and the heat transfers, a temperature distribution forms in the material.

4. **Blank deformation:** The heat generated, induces localised thermal expansion in the material. The thermal expansion introduces strains. The strains are resisted by the surrounding cooler material, thereby introducing stresses in the blank. If the stresses exceed the temperature dependent yield limit of the material, the blank deforms plastically. The thermal expansion, and therefore the thermal strains, are dependent of the temperature distribution, why different temperature distributions results in different types of forming. This subject is elaborated in section 2.2.

2.2 Forming Mechanisms in the Laser Forming Process

As described in section 2.1, the final deformation depends on the achieved temperature distribution in the blank. The temperature distribution is affected by the scan path and process variables i.e. laser power, laser beam diameter and scan speed, but also by workpiece and equipment parameters such as blank geometry, blank material, blank surface conditions, the use of protective air/gas for the laser etc. The influence of the variables and parameters is discussed in the 9th. semester report appended on the appendix-CD. [Shi et al., 2005]

In literature several thermo-mechanical mechanisms are suggested for understanding how different temperature distributions link to different forming scenarios. A thermo-mechanical mechanism is in the present project defined as an ideal heat distribution, which subsequently results in a strain distribution, thereby creating a unique forming type. The mechanisms are based on simple straight line scans across the blank in the width direction, as shown in figure 2.1, but are applicable to arbitrary scan paths as well. The mechanisms are an idealisation of the forming.

In practice the final forming is often a combination of the mechanisms.

In chapter 1, it was mentioned that the laser forming process must be capable of manufacturing geometries similar to conventional sheet metal forming processes. Therefore, it is required that the laser forming process enables manufacturing of both single and double curved geometries with both positive and negative curvature. These geometries can be categorised as either developable or non-developable surfaces. Figure 2.2 shows the concept of developable and non-developable surfaces. The concept is related to how a flat blank may be transformed into a surface. Developable surfaces are characterised as, surfaces, which are constructible without tearing, compressing and stretching the blank e.g. a half pipe, as shown in figure 2.2. A non-developable surface requires either tearing, compression or stretching. The dome geometry is an example of a non-developable surface, which requires in-plane shortening, as shown in figure 2.2. [Abed et al., 2005]

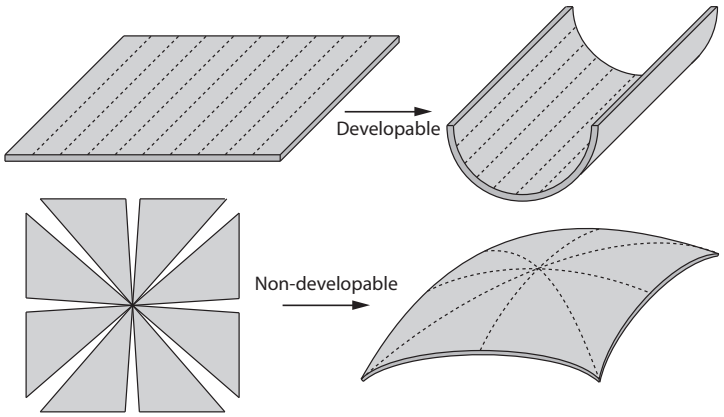


Figure 2.2: The concept of developable and non-developable surfaces (revised from [Abed et al., 2005]).

In this project, the temperature gradient and upsetting mechanism are considered as the temperature gradient mechanism is capable of creating out of plane bends that can be utilised in the creation of developable surfaces, while the upsetting mechanism is capable of creating in-plane shortening that can be utilised in the creation of non-developable surfaces. [Abed et al., 2005]

Each of the mechanisms are associated with different combinations of process variables [Dearden et al., 2003]. The following descriptions of each mechanism are based on [Dahotre and Harimkar, 2008], [Dowden, 2009] and [Shi et al., 2005]. Each description is based on a three step figure, chronologically describing the mechanism from laser beam irradiation, to final deformed geometry. In practice the forming is three dimensional. However, for clarity the descriptions only consider forming occurring perpendicular to the scan path. Hence, the forming is considered as two dimensional.

2.2.1 The Temperature Gradient Mechanism

The mechanism requires a steep temperature gradient through the thickness of the blank. The different steps of the temperature gradient mechanism are shown in figure 2.3 and described in the following.

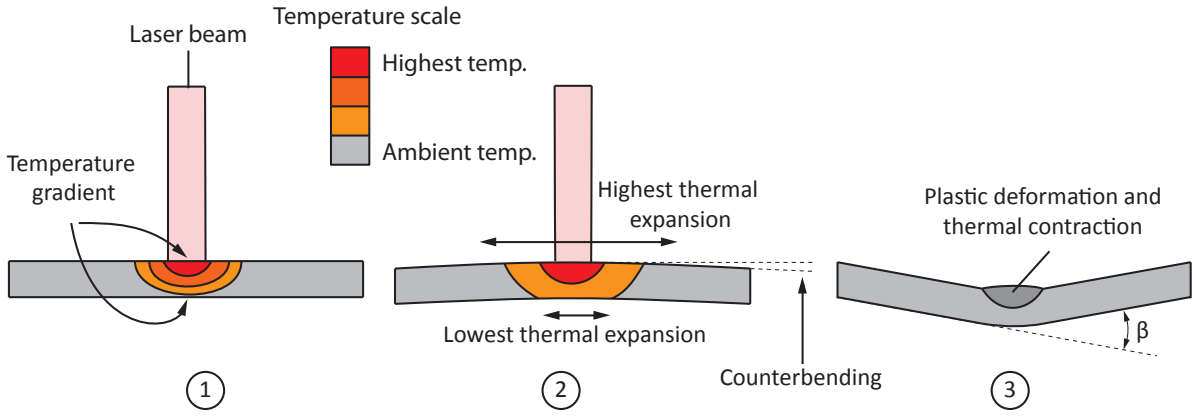


Figure 2.3: The temperature gradient mechanism. Initially a temperature gradient through the thickness is established. The gradient in temperature leads to differentiated thermal expansion through the blank which results in counterbending. The uppermost part of the blank deforms plastically due to decreased yield stress and high thermal expansion. When the blank cools the thermal expansion retracts. The plastic deformation results in a shortening relative to the original length. This results in a V-bend (revised from [Dowden, 2009] and [Dahotre and Harimkar, 2008]).

1. The blank is irradiated with a laser beam. A rule of thumb prescribes that the laser beam diameter is equivalent to the blank thickness [Edwardson et al., 2010]. A fraction of the energy from the laser beam is absorbed and causes a local temperature increase. The heat energy, obtained from the laser is simultaneously conducted to the rest of the blank. Due to the heat energy and the conductive properties of the blank a temperature gradient is formed through the thickness of the blank.

2. Due to the temperature gradient through the thickness the blank experiences different levels of thermal expansion. The thermal expansion varies from highest in the uppermost part of the blank, to lowest in the bottommost part. This results in counterbending away from the laser beam. The thermal expansion is resisted by the surrounding cooler material. Thus the thermal expansion cause compressive stresses. In the uppermost part of the blank the temperature is high, due to the irradiation of the laser beam. This lowers the yield stress of the material, as shown if figure 2.4. If the compressive stresses exceed the temperature dependent yield stress of the material, the blank deforms plastically.

3. After the surface has been laser scanned, the blank cools (dwell time). The cooling leads to thermal contraction of the blank. Since the uppermost part of the blank has been plastically deformed, the thermal contraction leads to a shortening, which is most prominent in the top of the blank. The shortening leads to a bend around the scan path. [Dowden, 2009]

states that the bend angle β after one laser scan can be up to 2°, however, this depends on the process. In [Madsen and Søndergaard, 2013] bend angles up to approximately 1.3° per laser scan were achieved. The achieved bend angle is controllable by varying the process variables. Larger bend angles are obtained by utilising multiple passes of the laser beam.

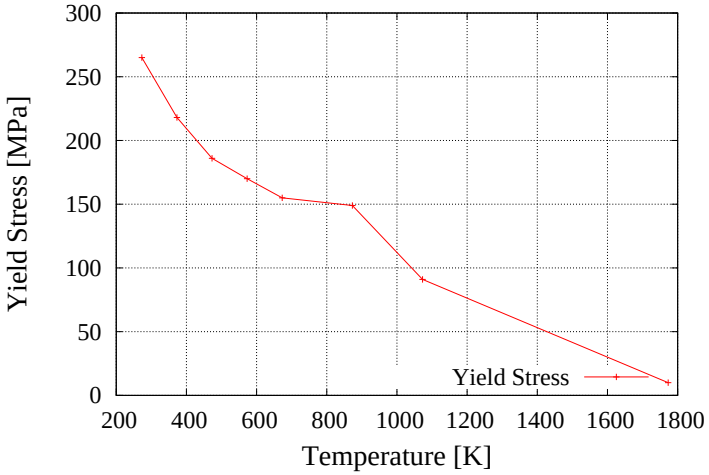


Figure 2.4: Temperature dependency of the yield stress for stainless steel 1.4301 (AISI 304).

2.2.2 The Upsetting Mechanism

The mechanism is given its name due to a local upsetting of the material beneath the laser beam through the thickness. Figure 2.5 shows the upsetting mechanism. To simplify figure 2.5 only expansion perpendicular to the plane extended by the scan path and thickness direction is considered. Ideally no temperature gradient across the thickness should exist. In practice this requirement is difficult to fulfil, thus the temperature gradient mechanism is activated as well. To minimise the influence of the temperature gradient mechanism, a high ratio of thermal conductivity to blank thickness is preferable. Moreover, a rule of thumb prescribes that the laser beam diameter must be approximately 10 times the blank thickness [Dahotre and Harimkar, 2008]. The different steps of the upsetting mechanism are shown in figure 2.5 and described in the following.

1. The blank is irradiated with a laser beam. Assuming activation of the idealised upsetting mechanism (no temperature gradient through the thickness), the material deforms uniformly perpendicular to the thickness direction, due to the thermal expansion. Due to the stiffness of the cooler surrounding material, the expansion causes compressive stresses in the heated area. If the stresses exceed the temperature dependent yield stress of the material, the material deforms plastically and local plastic upsetting occurs in the scan path.

2. As the laser beam is moved across the surface, the material near the laser beam continues to expand due to the increased temperature, whilst the material behind the laser beam is

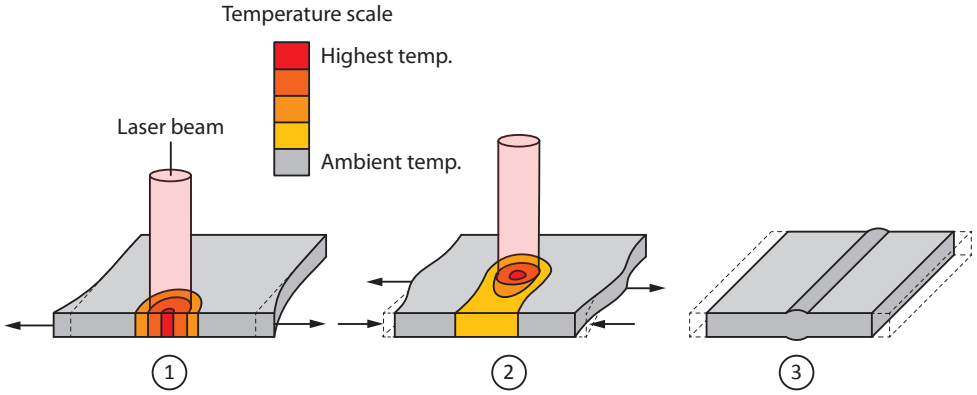


Figure 2.5: The upsetting mechanism. Initially the blank expands due to heating from the laser beam. The expansion induce uniform strains through the thickness, which causes plastic deformation if the temperature dependent yield stress is exceeded. When the material starts to cool it contracts, resulting in a shortening compared to the original shape, which is denoted with dashed lines. The arrows indicate material movement (revised from [Dowden, 2009]).

allowed to cool. The cooling decreases the thermal expansion in the blank. This leads to local shortening of the blank in the plastically deformed zone.

3. After a complete laser scan the blank is shortened, perpendicular to and in full length of the scan path.

2.3 Simulation of the Forming Mechanisms

This section introduces the Finite Element model developed in [Madsen and Søndergaard, 2013]. The forming mechanisms explained in section 2.2 are simulated using the Finite Element model of the laser forming process. The introduction briefly covers the main aspects of the Finite Element model. For a thorough description of the Finite Element model developed during the 9th. semester, the reader is referred to the 9th. semester report appended on the appendix-CD.

In the present project the Finite Element model is used to develop a framework, which enables the production of double curved geometries. To cope with this task, minor adjustments of the Finite Element model are necessary. A description of the adjustments considering; reducing the spatial discretisation of the blank and determination of process range for the upsetting mechanism, is appended in appendix B. The adjusted model is appended in appendix C. The adjusted Finite Element model has been compared to the model developed through the 9th. semester with respect to physical behaviour. An acceptable correlation was observed, why it is assessed that the adjusted Finite Element model is suitable for the development and testing of the framework. However, experimental validation of the Finite Element model is considered necessary before implementation in an experimental setup.

2.3.1 Introduction to the Finite Element Model

The Finite Element model, developed in [Madsen and Søndergaard, 2013], was developed with the purpose of establishing parameters for a feedback controller and to test a feedback control system.

The feedback control system from [Madsen and Søndergaard, 2013] is shown in figure 2.6. The control system controls the forming of a V-bend. The V-bend was formed by utilisation of the temperature gradient mechanism in a single scan path. The bend angle obtained by use of the temperature gradient mechanism is controllable by adjusting the average amount of energy transferred from the laser beam to the blank [Thomson and Pridham, 1997]. The average amount of energy was adjusted by controlling the scan speed with respect to the error between the desired bend angle and the current bend angle.

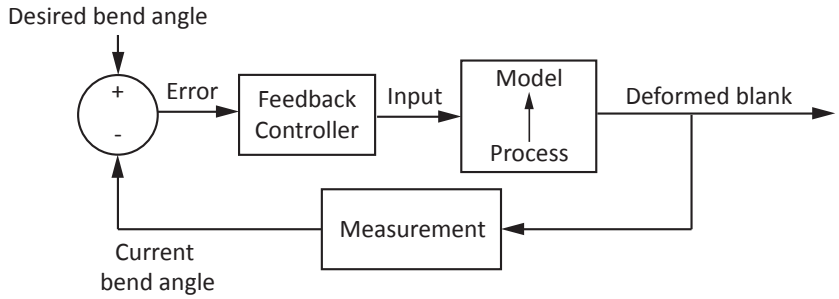


Figure 2.6: The control system developed in [Madsen and Søndergaard, 2013]. The arrow between Process and Model indicates that a Finite Element model of the process is developed and used to establish the parameters for the feedback controller.

The control system had to be capable of producing controlled V-bends in a stainless steel 1.4301 (AISI 304) blank. The temperature dependent material properties of the blank are appended in appendix A. The blank was positioned in a clamp and scanned with the laser beam in a single scan path, as shown in figure 2.7. To ensure correlation between the Finite Element model and an experimental setup, the Finite Element model was fitted to experimental data.

The laser forming process is modelled as a nonlinear isotropic coupled thermo-mechanical analysis, based on temperature dependent material properties. To simplify the Finite Element model, it is assessed that no phase changes occur. The Finite Element model is simulated using the nonlinear Finite Element Analysis code LS-Dyna. Figure 2.8 shows a visual representation of the Finite Element model. The blank is fixed at one edge to simulate the fixation of the clamp. In the solution of the coupled thermo-mechanical problem the thermal problem i.e. the temperature distribution is determined initially. Subsequently, the temperature distribution is used in the mechanical problem, to determine thermal expansion and the resulting forming. This procedure is repeated for each timestep of the solution. The temperature distribution is determined with respect to:

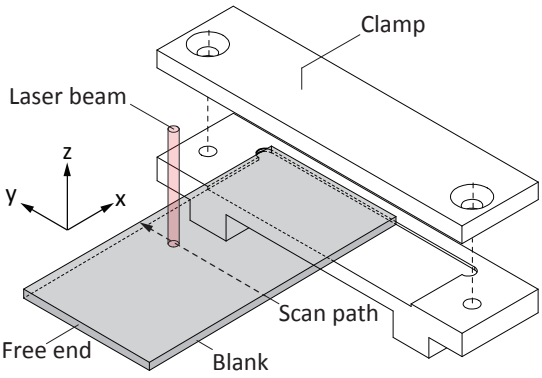


Figure 2.7: Sketch of the clamp and blank used in [Madsen and Søndergaard, 2013]. After each irradiation from the laser beam, the free end moves towards the laser beam.

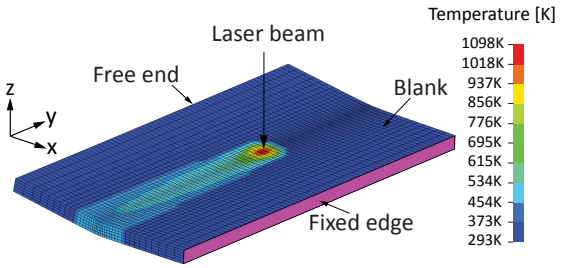


Figure 2.8: The developed Finite Element model of the laser forming process. The fixed edge is incorporated in order to simulate the clamp utilised in the experimental setup [Madsen and Søndergaard, 2013].

- A laser beam expressed as a moving Gaussian heat flux function, as shown in equation 2.1.
- Three dimensional transient heat conduction occurring in the blank.
- Convection and radiation heat losses from all surfaces. Both convection and radiation are expressed as heat flux leaving the surface of the blank.
- An ambient temperature and initial blank temperature of 293K.

The applied Gaussian heat flux function is given by equation 2.1 [Shi et al., 2005]. This approximation of the laser beam has shown good correlation with respect to experimental results in [Shi et al., 2005] and [Roll et al., 2011].

$$I(r) = \frac{2 \cdot A \cdot P}{\pi \cdot R^2} \cdot \mathrm{e}^{\left(\frac{-2 \cdot r^2}{R^2}\right)} \tag{2.1}$$

Where I is the heat flux density, A is the absorbance coefficient, P is the laser power, R is the laser beam radius and r is the distance from center of the laser beam.

As mentioned the purpose of the Finite Element model in [Madsen and Søndergaard, 2013] was to establish parameters for a control system. This implies running several simulations in a short amount of time. Hence, solving the Finite Element model must be fast, while ensuring results that correlate with experimental results. In the present project the Finite Element model is used in the development and testing of a framework enabling the production of double curved geometries, why a low solution time is still desirable. The following precautions were taken to ensure low solution time in [Madsen and Søndergaard, 2013]:

- Reduction of the blank size in order to decrease the number of total elements and thereby calculations.
- The spatial and temporal discretisation of the Finite Element model were selected to comply with the discretisation requirements, for simulating the laser forming process,

stated in [Zhang et al., 2004] i.e. a minimum of 2 elements per beam radius, a minimum of 3 elements in the thickness direction and a minimum of 4 timesteps per laser beam radius, meaning that 4 timesteps must be evaluated during the time it takes the laser beam to move a beam radius across the blank surface.

- The problem was scaled with a factor of 1000 with respect to time i.e. 3ms of simulation time equals 3s of process time. The time scaling implies an equal scaling of the thermal problem. This was treated by using LS-Dyna's "Thermal Speedup Factor" (TSF). The TSF was used to artificially scale the thermal properties of a thermal problem e.g. conductivity, heat convection coefficient etc. The TSF is set in correspondence to the time scaling i.e. if the scan speed of the laser beam is scaled by a factor 1000, the TSF is equally set to 1000. The TSF is evaluated for a simple rod example in appendix D.
- The analysis approach was switched from explicit to implicit whilst solving the Finite Element model. During the laser scan, when the deformation rate is high, the Finite Element model was solved with an explicit analysis. The explicit analysis dictates the use of small timesteps to maintain stability. Small timesteps provide a high precision solution of the deformation during the laser scan. In the subsequent dwell time the deformation rate is reduced why larger timesteps were sufficient to obtain a desired precision. During dwell time an implicit analysis was used. The implicit analysis is computationally costly compared to the explicit analysis. However, it is unconditionally stable, why large timesteps may be used. By using analysis switching high precision was obtained by using many computationally cheap explicit steps during the laser scan and few computationally costly implicit steps, during dwell time. [Lund and Lindgaard, 2012]

The correlation between the Finite Element model and an experimental setup was ensured by fitting the Finite Element model with respect to experimental data. The experimental data was obtained in the experimental setup developed in [Madsen and Søndergaard, 2013].

As mentioned the average amount of energy was controlled by varying the scan speed. However, control is only possible within a given process range. If the average energy is too high, the surface melts, as a result of the high temperatures. If the average energy is too low no plastic deformation is achieved, why no bend angle occurs. To determine a process range, that ensures an acceptable behaviour, when utilising the temperature gradient mechanism i.e. produces a significant bend angle whilst not melting the surface of the blank, a series of experiments were conducted on the experimental setup. In the experiments, the laser beam diameter was varied from 1mm to 5mm in 1mm increments. For each laser beam diameter, different scan speeds were tested to identify the upper and lower limit of the process range. The laser power was maintained at 380W to reduce the amount of experiments. All experiments with a laser beam diameter smaller than 3mm resulted in a melted surface and were therefore not acceptable. The experiments with a laser beam diameter above 3mm resulted in a narrow process range where small adjustments in the scan speed either resulted in a melted surface or a small bend angle. This was considered undesirable hence, it was decided to utilise the process range obtained for the 3mm laser beam diameter being an upper scan speed limit of $7000\frac{\text{mm}}{\text{min}}$ and a lower scan speed limit of $2750\frac{\text{mm}}{\text{min}}$.

The Finite Element model was fitted with respect to three scan speeds within the process range, as shown in figure 2.9.

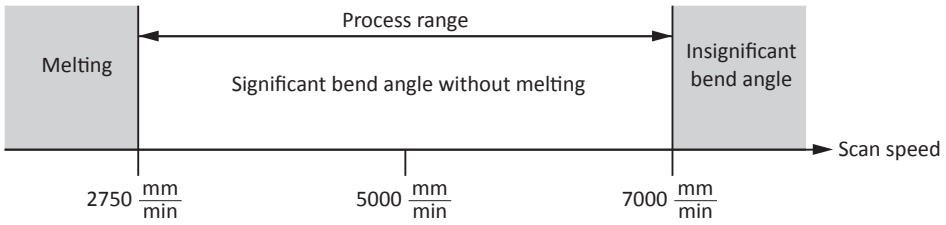
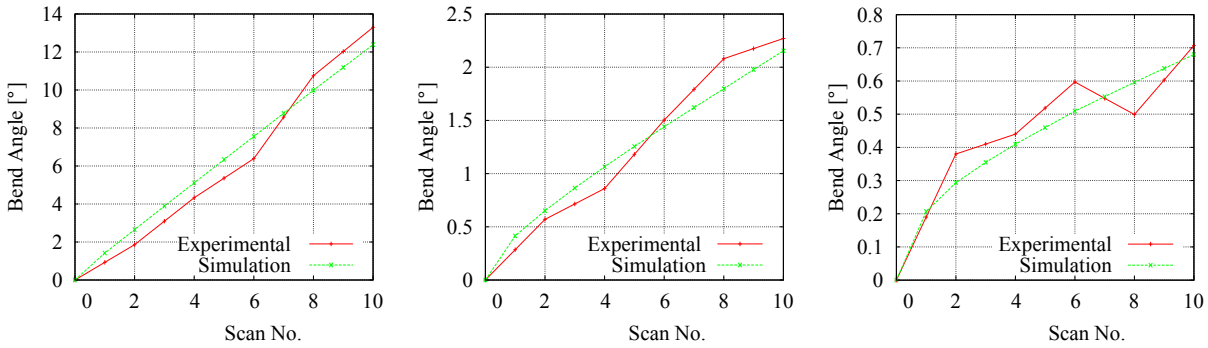


Figure 2.9: Process range for the temperature gradient mechanism when varying the scan speed and maintaining a fixed laser beam diameter of 3mm and a fixed laser power of $380W$.

The model was fitted to the three scan speeds by systematic variation of the absorbance coefficient, as recommended in [Dowden, 2009]. Results of the fitted Finite Element model are shown in figure 2.10.



(a) Results with a scan speed of $2750\frac{\text{mm}}{\text{min}}$ and an absorbance coefficient A = 0.4250.

(b) Results with a scan speed of $5000\frac{\text{mm}}{\text{min}}$ and an absorbance coefficient A = 0.3125.

(c) Results with a scan speed of $7000\frac{\text{mm}}{\text{min}}$ and an absorbance coefficient A = 0.3000.

Figure 2.10: Comparison of the fitted Finite Element model and experimental results.

As mentioned, the purpose of the model in [Madsen and Søndergaard, 2013] was to aid in the establishment of parameters for a control system. As the correlation between the model and the physical behaviour achieved in experiments was found acceptable, it was assessed sufficient for this task.

2.3.2 Simulating the Mechanisms

The introduced Finite Element model is used to demonstrate the mechanisms discussed in section 2.2. In this demonstration a straight line laser scan in the Y direction is performed. The demonstration focuses on the ability of the model to demonstrate correct physical behaviour prescribed by the mechanisms with respect to temperature distribution, strains and forming.

The behaviour is inspected at nodes located on the centerplane, perpendicular to the Y axis, as shown in figure 2.11.

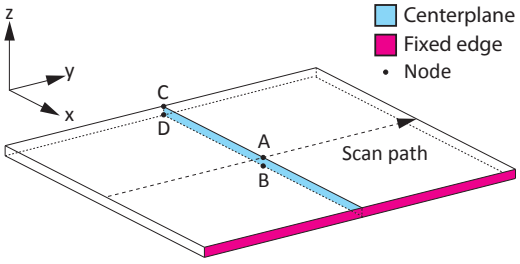


Figure 2.11: Inspected nodes in the demonstration of the temperature gradient mechanism and upsetting mechanism.

Table 2.1 shows the process variables used to activate the temperature gradient mechanism and upsetting mechanism respectively. The temperature gradient mechanism is demonstrated using the process variables accounting for the lower process range limit identified in [Madsen and Søndergaard, 2013], as this is known to activate the temperature gradient mechanism. No experimental tests have been conducted with respect to the upsetting mechanism[1]. Hence, the process variables selected for this demonstration are based on rules of thumb i.e. the laser beam diameter is set equal to 10 times the blank thickness, as suggested in section 2.2.2. The scan speed is lowered as this facilitates a uniform temperature distribution through the thickness of the blank and the laser power is maintained fixed.

Variable:	**Temperature gradient mechanism:**	**Upsetting mechanism:**
Laser beam diameter:	3mm	10mm
Scan speed:	$2750\frac{\text{mm}}{\text{min}}$	$500\frac{\text{mm}}{\text{min}}$
Laser power:	380W	380W

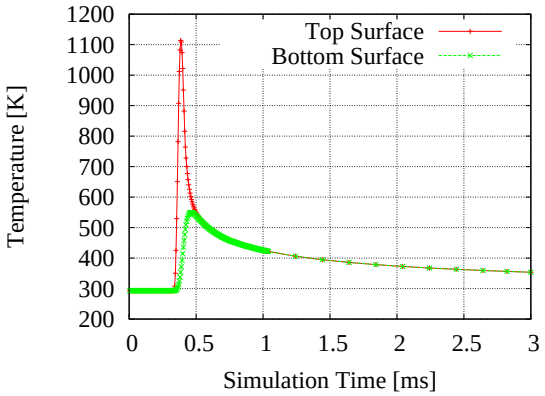
Table 2.1: Process variables used to activate the forming mechanisms.

Temperature Distribution

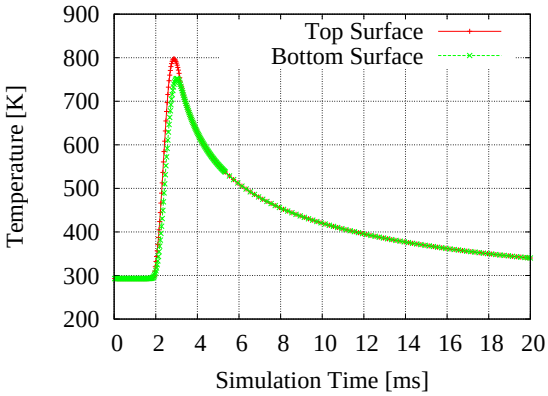
Achieving the temperature gradient mechanism implies selecting process variables, which guarantee a steep temperature gradient through the thickness of the blank. Achieving the upsetting mechanism implies selecting process variables, which guarantee a uniform temperature distribution through the thickness. As shown in figure 2.12, a temperature difference from the top to bottom surface is present in both mechanisms, why a temperature gradient through the thickness is present. However, the gradient in the upsetting mechanism is less prominent. As mentioned in section 2.2.2, it is difficult to achieve a uniform temperature through the thickness, due to the conductive properties of the material. In both mechanisms the gradient in the centerplane is

[1]This demonstration of the mechanisms was conducted prior to determination of the process range for the upsetting mechanism. Therefore, the process variables used throughout the demonstration correspond to rules of thumb, instead of a determined process range.

largest as the laser beam traverses the centerplane. After the laser beam has passed, the surface temperature converges towards ambient temperature.



(a) Temperature development of the temperature gradient mechanism.

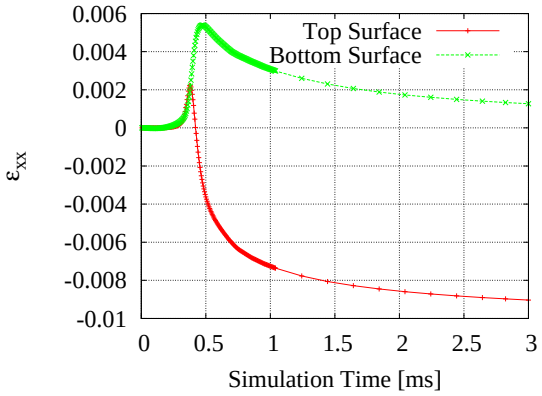


(b) Temperature development of the upsetting mechanism.

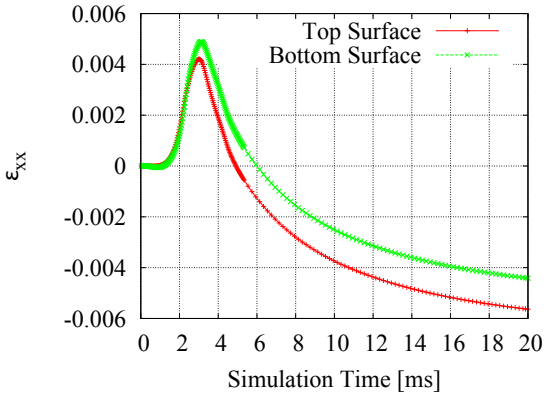
Figure 2.12: Temperature developments for the temperature gradient and upsetting mechanism in node A and node B, shown in figure 2.11.

Strain Distribution

[Liu and Yao, 2005] states that, the highest compressive strains in the laser forming process, occur in the direction perpendicular to the scan path, why only the X directional strains are investigated. As described in section 2.2, a temperature gradient through the thickness induces differentiated thermal expansion. Figure 2.13a shows that, when utilising the temperature gradient mechanism, the top surface is subject to negative X directional strain and the bottom surface is subject to positive strain after a complete laser scan. Hence, the top surface is in compression and the bottom surface is in tension, as is expected in a bending scenario. Figure 2.13b shows that the upsetting mechanism results in negative X directional strain through the entire thickness. The difference in magnitude from top to bottom is assumed to be caused by the influence of the temperature gradient mechanism. The strain implies that the entire scan path is in compression perpendicular to the scan path, as is expected in a shortening scenario.



(a) Strain development of the temperature gradient mechanism.

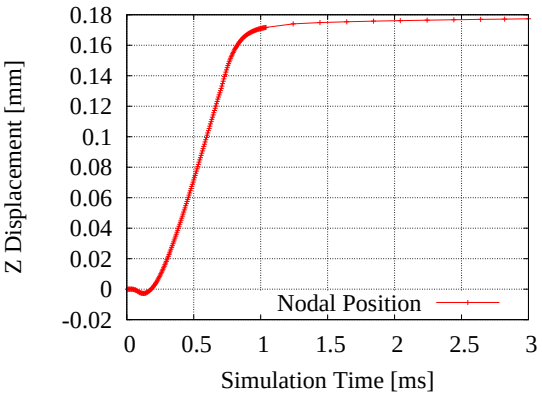


(b) Strain development of the upsetting mechanism.

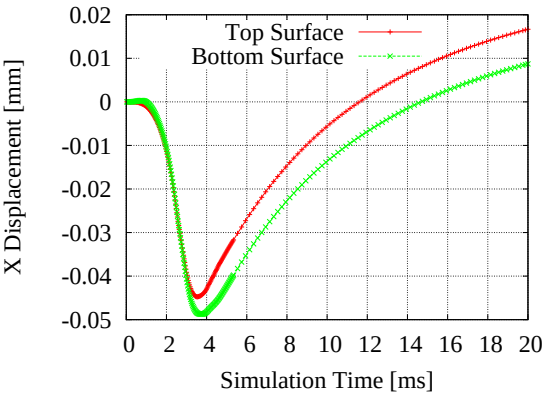
Figure 2.13: Strain developments in the X direction for the temperature gradient and upsetting mechanism at node A and node B, shown in figure 2.11.

Forming

Figure 2.14a shows the Z displacement of node C whilst utilising the temperature gradient mechanism. Initially a negative displacement of node C is observed. This is assumed to be caused by the counterbending of the blank, as a result of the differentiated expansion. This is followed by a positive displacement of node C, which is a result of the development in bend angle. Figure 2.14b shows the X displacement of node C and D whilst utilising the upsetting mechanism. During heating, the blank expands causing an elongation of the blank in the X direction. After heating the blank contracts and is shortened in comparison to the original geometry. The shortening is more prominent in the top. This is assumed to be caused by the influence of the temperature gradient mechanism.



(a) Z displacement of node C shown in figure 2.11, as a result of the temperature gradient mechanism.



(b) X displacement of node C and D shown in figure 2.11, as a result of the upsetting mechanism.

Figure 2.14: Displacements from the temperature gradient and upsetting mechanism.

2.4 Sub Conclusion

In the laser forming process a defocused laser beam irradiates a blank in predefined scan paths. The absorbed laser energy causes heat generation in the material, which results in expansion of the blank. The expansion is resisted by the surrounding material thereby inducing compressive stresses. If the stresses exceed the temperature dependent yield stress, plastic deformation is introduced. To produce double curved geometries it is necessary to utilise the temperature gradient mechanism and the upsetting mechanism. The mechanisms are important, as they enable the laser forming process to create developable and non-developable surfaces. Both mechanisms were demonstrated using an adjusted version of the Finite Element model developed in [Madsen and Søndergaard, 2013]. The Finite Element model successfully demonstrated correct physical behaviour with respect to the temperature distribution, strain distribution and forming obtained by both mechanisms.

Chapter 3

The Developed Framework for the Laser Forming Process

This chapter describes the developed framework for the laser forming process. The framework is capable of transforming a desired geometry into the required scan path and process variables. The framework is developed as a software program. The development and evaluation of the framework is based on the Finite Element model of the laser forming process, described in chapter 2. However, the framework is developed with regards to a physical setup i.e. the framework can be implemented directly in a physical setup. This chapter consists of an introduction to the framework, where the background and structure of the framework are presented. Following the introduction, the individual tasks performed in the framework are elaborated.

3.1 Introducing the Framework

This section introduces a concept for process design, for the laser forming process, developed by [Liu and Yao, 2005]. The concept serves as the background for the framework, which is described in section 3.1.2.

3.1.1 Background for the Framework

[Liu and Yao, 2005] developed a concept for calculating scan paths and selecting of process variables suitable for creating double curved geometries. The concept must be utilised prior to the execution of the laser forming process. Figure 3.1 shows the concept used by [Liu and Yao, 2005], which is described below.

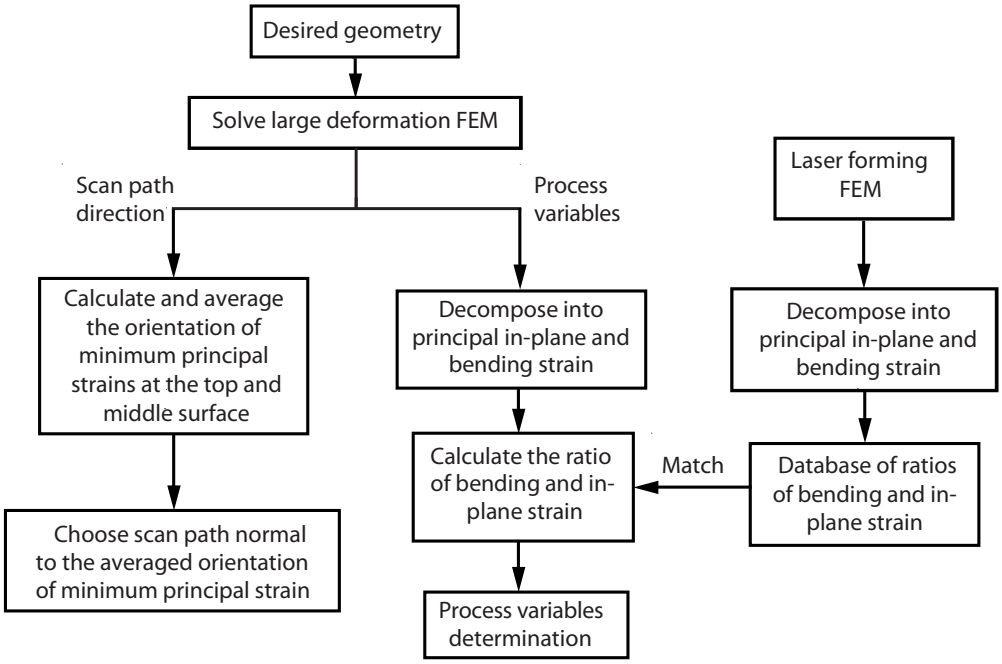


Figure 3.1: Flowchart of the concept used in [Liu and Yao, 2005].

Initially a Finite Element representation of the desired geometry is flattened between two rigid planes in a large deformation Finite Element model. The desired geometry is flattened to a thickness equal to the original blank thickness. From the Finite Element model a strain field is obtained, which corresponds to the developed strains, when the desired geometry is compressed to flat. The developed strain field is equal to the strain field required to obtain the desired geometry. The required strain field is used to determine suitable scan paths and process variables. The scan paths are oriented perpendicular to the average of the orientation of the minimum principal strains in the topplane and in the midplane of the blank. This is shown in figure 3.2, where the bars indicate the size and orientation of minimum principal strains. The orientation of the bars are equal to the orientation of minimum principal strains, while the lengths are equal to the relative size of the minimum principal strains. The red line in figure 3.2, indicates the chosen scan path with respect to the orientation of minimum principal strain.

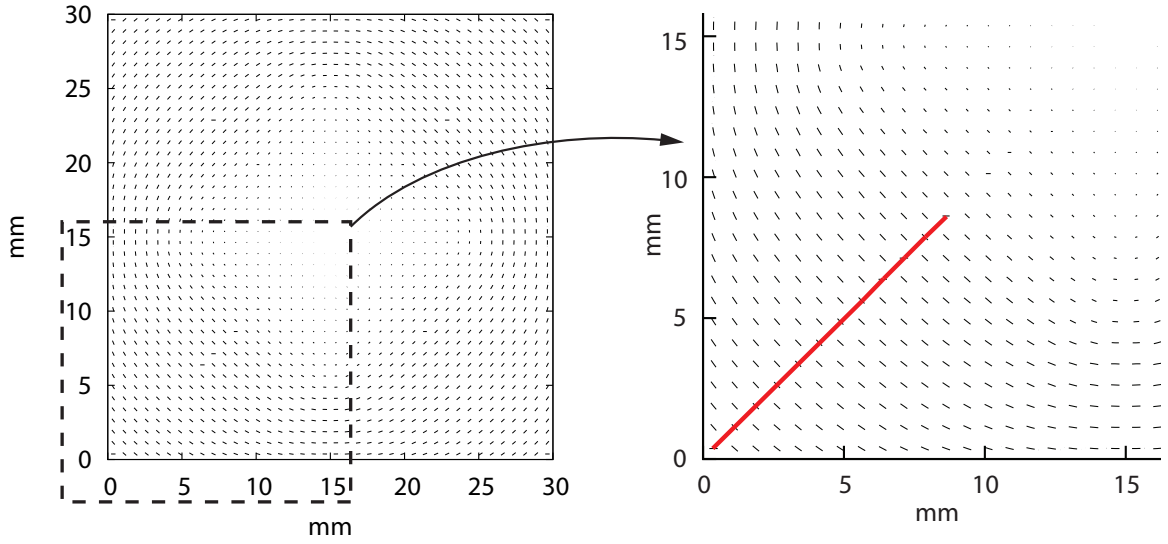


Figure 3.2: The orientation of the average of the minimum principal strains in the topplane and in the midplane of the blank. An excerpt is shown, which illustrates a scan path located perpendicular to the orientation of minimum principal strains.

As discussed in section 2.2 the temperature gradient mechanism and the upsetting mechanism are required to develop double curved geometries. The selection of forming mechanism is based on the remaining topplane and midplane strains. These are considered indicators of the in-plane and bending strains required to achieve the desired geometry. The forming mechanisms are controlled by proper selection of process variables. The process variables are selected from a database, based on the required in-plane and bending strains in the scan path. The database has been established, prior to the forming operation, using a Finite Element model of the laser forming process. When a scan path has been calculated for the desired geometry and suited process variables have been chosen, a laser scan can be performed.

The concept introduced in [Liu and Yao, 2005] was evaluated in an experimental setup with respect to a dome and a saddle geometry. Correlation was obtained between the desired and the obtained geometry for both the dome and saddle. The discrepancy between the desired geometry and the obtained geometry was assessed to be caused by the discretisation of the desired geometry, which is necessary when implementing the geometry in the Finite Element program, as well as the finite number of scan paths.

3.1.2 Introduction to the Developed Framework

The concept developed by [Liu and Yao, 2005] proofed successful, for the tested geometries. The approach used in [Liu and Yao, 2005], depends on achieving the desired geometry in one iteration, while using process variables from a database, which are selected based on the required strain field. However, the output of the laser forming process suffers from variance, residual

stresses and non-linearities as described in the following.

- **Process variance:** The laser forming process depends on several variables and parameters e.g. process variables and material parameters, which can be subject to fluctuation. This results in variation in the end product [Thomson and Pridham, 1997].
- **Residual stresses:** From chapter 1 it was determined that the developed framework must enable the production of double curved geometries. It is assessed that the production of double curved geometries introduce residual stresses outside the scan path. These stresses affect the output of the remaining process by either increasing or decreasing the obtainable bending or in-plane strain.
- **Process non-linearities:** Furthermore, the laser forming process suffers from process non-linearities. These are defined as edge effects and bend rate decay, which are thoroughly described for the temperature gradient mechanism in [Madsen and Søndergaard, 2013] and briefly described below.
  - Edge effects refer to a varying amount of forming, when the laser beam approaches the edge of a blank. The edge effect can be explained by the asymmetry of the process [Shen et al., 2009]. The asymmetry is both due to a changing mechanical resistance towards forming and an asymmetric temperature distribution along the scan path [Dahotre and Harimkar, 2008].
  - The bend rate decay refers to a decreased amount of bending achieved, when utilising the temperature gradient mechanism through multiple laser scans in a single scan path [Dahotre and Harimkar, 2008]. The decay in bend rate is caused by a series of different effects e.g. strain hardening of the blank. As the material hardens it becomes more reluctant to form. It is assumed that the effect of bend rate decay is also present in a general forming scenario.

With respect to the variance, residual stresses and non-linearities present in the laser forming process, it is assessed that the concept described in [Liu and Yao, 2005] using a single iteration strategy and process variables selected from a database is not a robust approach. To increase robustness of the laser forming process, it is selected to combine the concept developed by [Liu and Yao, 2005] with process control.

Basic Control Theory

Process control may be obtained with two basic control system approaches being, feedforward and feedback. The approaches are shown in figure 3.3 and described in the following.

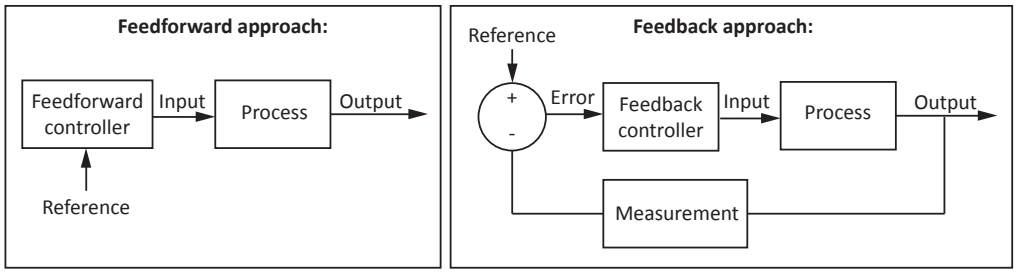


Figure 3.3: The two basic control system approaches. The feedforward approach relies on process knowledge to control the process input, such that a desired output is produced. The feedback approach relies on measuring the process output and correcting the input with respect to an error between the current and the desired output.

The feedforward approach relies on the ability to predict the process output, similar to the concept presented by [Liu and Yao, 2005]. A reference of the process output is given to the feedforward controller. Based on the reference output and knowledge of the behaviour of the process, the processing input is calculated. The knowledge can be based on empirical results or predictive models of the process, which is the case in [Liu and Yao, 2005] where a database is used. The major disadvantage of the feedforward approach is the demand for high repeatability of the process, in order to predict the process behaviour correctly. This is assessed to be problematic as the laser forming process suffers from poor repeatability e.g. bending tests with nominally identical process variables for a 10mm wide and 1mm thick mild steel blank, produced a spread in bend angle around the process mean of $\pm 15\%$, when formed with 30 minute intervals. [Thomson and Pridham, 1997]

The feedback approach relies on measuring the output of the process and adjust the process input accordingly. A reference for the output is given. By measuring the output of the process with a sensor and comparing it with the reference, at each iteration k of the feedback loop, an error is obtained[1]. Based on the error the feedback controller adjusts the input to the process for the next iteration of the feedback loop $k + 1$. Compared to the feedforward approach, the feedback approach is able to cope with poor repeatability, as the input is adjusted with respect to the current output of the process. [Thomson and Pridham, 1997]

Based on the description of the basic control system approaches, it is assessed that feedback control is the best suited control system approach for the laser forming process.

[1]The iteration counter k is introduced to designate the iterations of the feedback loop

Incorporating Feedback Control in the Framework

The feedback system shown in figure 3.3, is the foundation for the framework developed in this project, the structure of the framework is shown in figure 3.4. To allow feedback control of the laser forming process a conservative multipass strategy is utilised, such that the laser forming is conducted in small incremental steps defined by the iteration counter k of the framework. The scan path and process variables are determined between every laser scan.

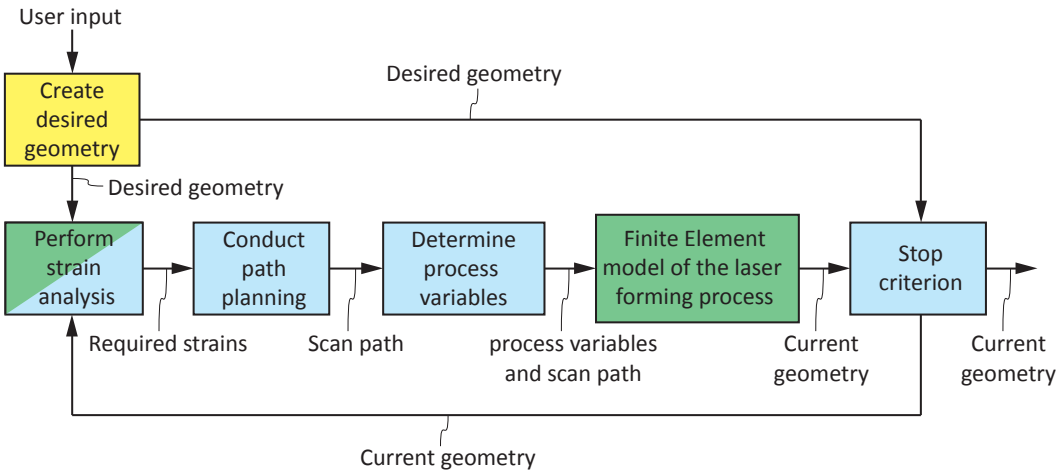


Figure 3.4: Structure of the framework for the laser forming process. The yellow box indicates a task performed by utilisation of SolidWorks and LS-Prepost, the green boxes indicate tasks solved by a Finite Element model in the non-linear Finite Element code LS-Dyna and the blue boxes designate tasks performed by a developed software program.

The boxes which constitute the framework shown in figure 3.4 are explained in the following list, references are made to the sections where the boxes are elaborated further. Pseudo code is used to assist the explanation of selected boxes.

- In *"Create desired geometry"* a desired geometry is defined by user input. The desired geometry is used as a reference for the framework. This box is elaborated in section 3.2.
- In *"Perform strain analysis"* a strain field is determined, which define the strains required to go from current geometry[2] to the desired geometry. The strain field is obtained by use of a Finite Element model and manipulated to suit the purpose of the subsequent path planning. This box is elaborated in section 3.3.
- In *"Conduct path planning"* a scan path is determined on the basis of the strain field obtained in *"Perform strain analysis"*. This box is elaborated in section 3.4.
- In *"Determine process variables"* process variables are determined based on the required strain in the scan path. This box is elaborated in section 3.5.
- In *"Finite Element model of the laser forming process"* the Finite Element model of the

[2]Current geometry defines the initial flat blank in the first iteration and the formed blank in the following iterations of the framework.

laser forming process is solved with regards to the scan path and process variables determined in *"Conduct path planning"* and *"Determine process variables"*. The *"Finite Element model of the laser forming process"* is introduced in section 2.3.1. The keydeck for the Finite Element model is appended in appendix C.

- In *"Stop criterion"* the current geometry received from *"Finite Element model of the laser forming process"* is subtracted the desired geometry which results in a surface error. The surface error is used as a stop criterion for the framework. This box is elaborated in section 3.6.

If the criterion in *"Stop criterion"* is not achieved, a new iteration of the framework must be performed.

The boxes in figure 3.4 are marked with different colours. The yellow box designates the task of creating a desired geometry which in the present project is solved in SolidWorks and LS-Prepost respectively. The green boxes designate tasks which are solved using two separate Finite Element models, which are solved using the nonlinear Finite Element code LS-Dyna. Remark that the Finite Element model used for the strain analysis must be used during the execution of the framework when implemented in a physical setup, whereas the Finite Element model of the laser forming process, functions as a substitute for the physical process. The blue boxes designate tasks which are performed by a software program. The software program is written in the object oriented programming language Java. The software program consists of a Java main program, utilising a chronological structure of the tasks which are to be performed. To perform these tasks, a toolbox is utilised, which is a separate Java class containing "tools" that are developed for the purpose of the individual tasks. The toolbox is used to reduce the extent of the Java main program. Furthermore, a series of shell scripts are developed, these shell scripts contain simple programs that may be written to the shell[3], thereby performing tasks based on shell commands. All developed software is executed on the Linux-based cluster located at the Department of Mechanical and Manufacturing Engineering at Aalborg University.

For a thorough understanding of the software program, a commented version of the source code for the Java main program, toolbox and shell scripts are appended in appendix E, F and G respectively.

3.2 Create Desired Geometry

This section describes the box "*Create desired geometry*", shown in figure 3.4. The box treats the problem of creating a desired geometry for the framework by means of user input.

Figure 3.4 shows that an initial desired geometry must be developed prior to process execution. The process of defining a geometry is shown for a CAD geometry of a 10° V-bend in figure 3.5. The 10° V-bend is used as an example geometry throughout the remaining sections of this chapter. The desired geometry can be generated by the user in e.g. a CAD program or via

[3]Command-line interpreter for Linux based operating systems.

a point cloud file of a desired surface. In this project the desired geometries are modelled in SolidWorks and exported to LS-Prepost, which is used for preprocessing. LS-prepost is used to mesh various desired geometries, which are introduced in chapter 4 and 5. The desired geometry must be saved in a LS-Prepost readable format e.g. .IGS or .vda. This file is manually preprocessed in LS-Prepost, where a meshed shell representation of the blank is produced.

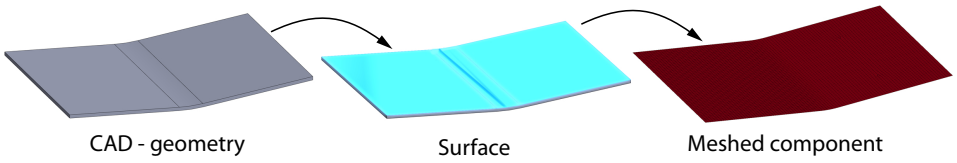


Figure 3.5: The surface of the desired geometry is exported in .IGS format, which allows preprocessing in LS-Prepost. This results in the meshed geometry.

3.3 Perform Strain Analysis

The approach for determining scan paths, utilised in this project resembles the approach used in the concept developed by [Liu and Yao, 2005], introduced in section 3.1. The approach dictates that scan paths are placed perpendicular to the orientation of the minimum principal strains. This approach is utilised, as the most compressive strain introduced by the laser forming process occurs perpendicular to the scan path. The orientation of the minimum principal strains is determined with respect to a strain field. This section describes the box *"Perform strain analysis"*, shown in figure 3.4. The box treats the problem of determining the strain field used in *"Conduct path planning"*, with respect to a desired geometry and a current geometry received from *"Create desired geometry"* and *"Finite Element model of the laser forming process"*, as shown in figure 3.4.

In this project, a strain field is defined as a field representation of the required strains in a certain plane to go from current to desired geometry. During the strain analysis several different strain field representations are introduced. A brief description of all the introduced strain field representations is appended in appendix H. All strain fields are approximated by means of a Finite Element model. Therefore, a strain field is only known at discrete points dictated by the element discretisation i.e. a continuous strain field $\varepsilon_{example}(x, y)$ is defined as $\varepsilon_{example}(m, n)$. m and n refers to the utilised element discretisation. The element discretisation in the X and Y direction and an example of the blank discretisation is shown in figure 3.6.

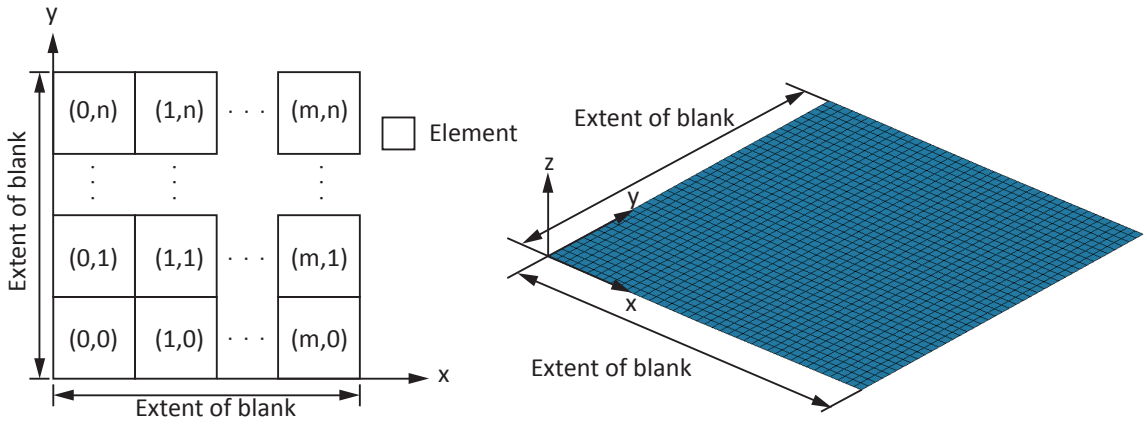


Figure 3.6: Discretisation of strain fields and an example of the discretisation of a blank.

The description of the strain analysis is divided into two separate tasks being; determination of the strain field, for generating scan paths and manipulation of the determined strain field. Each task is elaborated in section 3.3.1 and 3.3.2.

3.3.1 Determination of the Strain Field

The framework for the laser forming process must be capable of producing double curved geometries. In order to produce double curved geometries the placement of scan paths and selection of process variables must correspond to the bending and in-plane strain required to go from the current to desired geometry. Therefore, it is necessary to determine how bending and in-plane strains can be obtained from the strain field determined by the strain analysis.

The laser forming process induces plastic deformation to permanently form a blank. To simplify the problem of identifying the required bending and in-plane strains, it is assumed that the deformation exhibits a linear elastic behaviour and that deflections are small. As the framework is based on the feedback approach, errors introduced by the assumption are accepted, as these are reduced in the subsequent iterations of the framework. [Ventsel and Krauthammer, 2001] states that the general assumptions, when considering linear, elastic, small deflection theory of bending for thin plates are analogous to the general assumptions in beam theory. Therefore, it is assessed, that the distribution of required bending and in-plane strain can be approximated by considering the strain distributions in an elastic beam in bending and in shortening, as shown in figure 3.7. In the bending scenario the strains vary from positive to negative through the thickness [Gere and Goodno, 2009]. Assuming a linear distribution of strains, bending is identified by investigating the strain fields $\varepsilon_{top}(m,n)$ or $\varepsilon_{bot}(m,n)$ on the top or bottom surface respectively. In the shortening scenario the strains are negative and of equal size through the thickness. Therefore, shortening may also be identified by the surface strains. However, in the case of combined deformation scenario, this is undesirable, as bending and in-plane shortening are difficult to decouple. Instead $\varepsilon_{mid}(m,n)$ is a better indicator of in-plane shortening, as the strains induced by bending are assessed to be equal to zero in the midplane.

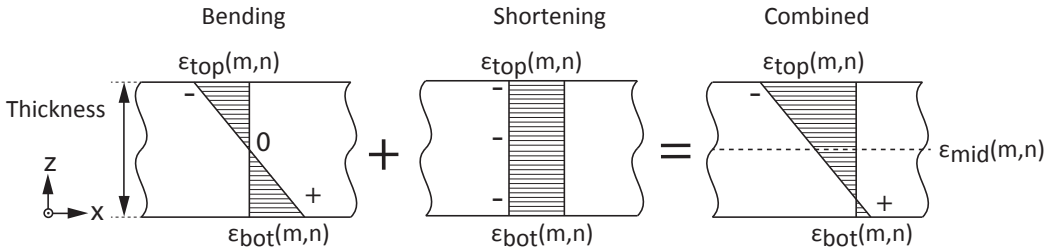


Figure 3.7: 2D examples of strain distributions. In the bending scenario the strain varies through the thickness and is indicated by $\varepsilon_{bot}(m,n)$ or $\varepsilon_{top}(m,n)$. In the in-plane shortening scenario the strains are evenly distributed through the thickness and are indicated by $\varepsilon_{mid}(m,n)$.

$\varepsilon_{top}(m,n)$ and $\varepsilon_{bot}(m,n)$ indicates bending strain and $\varepsilon_{mid}(m,n)$ indicates in-plane strain. To ease implementation, it is desired to establish one strain field $\varepsilon_{path}(m,n)$ for the determination of scan paths. Therefore, $\varepsilon_{path}(m,n)$ must contain information regarding both the required bending and in-plane strain, such that both can be accounted for during a laser scan by selection of process variables. To include information of the required bending and in-plane strain $\varepsilon_{path}(m,n)$ is created as an average of $\varepsilon_{mid}(m,n)$ and one of the surface strain fields.

From section 2.2 it is known that the temperature gradient mechanism is present in both forming mechanisms, why the blank bend towards the laser beam. When the blank bend towards the laser beam, the largest compressive strains occur in the laser scanned surface. Therefore, the surface requiring the most compressive strain is selected for the average, creating $\varepsilon_{path}(m,n)$. Figure 3.8 shows the combined bending and shortening scenario, where $\varepsilon_{path}(m,n)$ is the average of $\varepsilon_{top}(m,n)$ and $\varepsilon_{mid}(m,n)$.

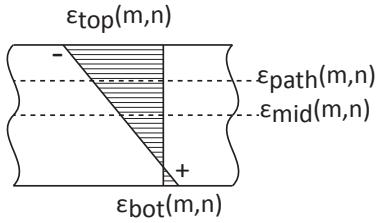


Figure 3.8: The placement of $\varepsilon_{path}(m,n)$ in a combined bending and shortening scenario.

The pseudo code for determining $\varepsilon_{path}(m,n)$ is shown in algorithm 1 and described below. The pseudo code corresponds to line 127 to 217 of the Java main program appended in appendix E.

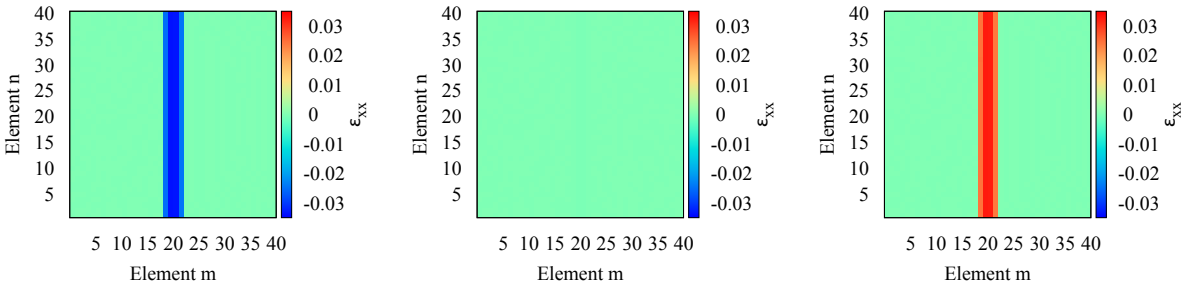
Algorithm 1 Pseudo code for determining $\varepsilon_{path}(m,n)$.

1: Extract $\varepsilon_{top}(m,n)$ and $\varepsilon_{bot}(m,n)$ from the OneStep solution
2: Calculate $\varepsilon_{mid}(m,n)$ = average of $\varepsilon_{top}(m,n)$ and $\varepsilon_{bot}(m,n)$
3: Determine most compressive surface strain of $\varepsilon_{top}(m,n)$ and $\varepsilon_{bot}(m,n)$
4: **if** most compressive surface strain is located in $\varepsilon_{top}(m,n)$ **then**
5: $\quad\varepsilon_{path}(m,n)$ = average of $\varepsilon_{mid}(m,n)$ and $\varepsilon_{top}(m,n)$
6: **end if**
7: **if** most compressive surface strain is locted in $\varepsilon_{bot}(m,n)$ **then**
8: $\quad\varepsilon_{path}(m,n)$ = average of $\varepsilon_{mid}(m,n)$ and $\varepsilon_{bot}(m,n)$
9: **end if**

As shown in line 1 of algorithm 1 the determination of $\varepsilon_{path}(m,n)$ starts with an extraction of $\varepsilon_{bot}(m,n)$ and $\varepsilon_{top}(m,n)$ from LS-Dyna's OneStep solver, the Finite Element model which represents the OneStep solver is appended in appendix I. The OneStep solver is selected for determining the strains constituting $\varepsilon_{path}(m,n)$. The OneStep solver is a complete piece of Finite Element code provided in LS-Dyna, which eases the development of the framework. Furthermore, the OneStep solver provides a low computation time compared to the flattening model introduced in chapter 5. The OneStep solver receives inputs in the form of a shell model representation of either the current or desired geometry and the material properties of the blank. Based on the input geometry and the material properties the OneStep solver determines an initial unformed flat state. The output is a flattened blank along with the strain tensor ε_{ij} at the outermost integration points of each element. ε_{ij} represents the required strain to go from a flat blank to the input geometry [LSTC, 2013]. The integration points are distributed through the thickness, at the element centroid. The outermost integration points are the integration points closest to the element surface. The uppermost integration point is used to generate $\varepsilon_{top}(m,n)$ and the bottommost integration point is used to generate $\varepsilon_{bot}(m,n)$. As the integration points are not located precisely at the surface, the procedure is to use four or five integration points through the thickness and ignore the error, when estimating a strain field located at the surface [Support, 2013]. In this project five integration points are used. It is assessed that the error introduced is insignificant, as long as the distribution of the strains demonstrate correct behaviour. In the initial iteration $k = 1$, the input geometry to the OneStep solver is the desired geometry e.g. the V-bend geometry in figure 3.5. Therefore, the output is the total $\varepsilon_{bot}(m,n)$ and $\varepsilon_{top}(m,n)$ required to go from flat blank to desired geometry. In the following iterations the input is the current geometry i.e. the geometry achieved after each laser scan. The remaining strain in each iteration, necessary to achieve the final geometry, is obtained by subtracting the current result of the OneStep solver from the initial result.

The operations in Line 2-9 of algorithm 1 convert the extracted $\varepsilon_{bot}(m,n)$ and $\varepsilon_{top}(m,n)$ into $\varepsilon_{path}(m,n)$. As mentioned the surface, which requires the most compressive strain, is used to calculate $\varepsilon_{path}(m,n)$ by averaging the surface strain field with $\varepsilon_{mid}(m,n)$. Assuming a linear distribution of strain through the thickness, as shown in figure 3.8, $\varepsilon_{mid}(m,n)$ can be approximated as the average of $\varepsilon_{bot}(m,n)$ and $\varepsilon_{top}(m,n)$.

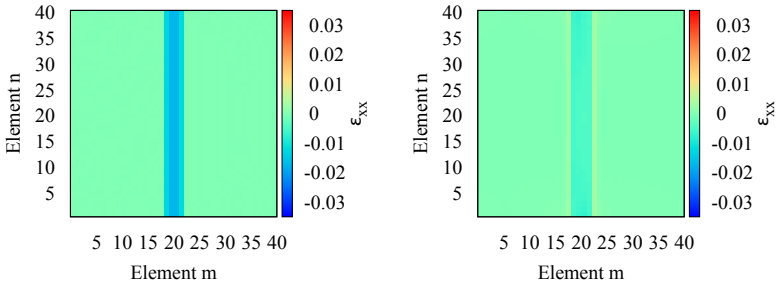
Considering the V-bend from figure 3.5, the most compressive and tensile strains occur at the surfaces in the X direction, while the midplane strain in this direction is assumed to be zero. The X directional strain component for the initial iteration $k = 1$, is shown for $\varepsilon_{top}(m, n)$, $\varepsilon_{mid}(m, n)$ and $\varepsilon_{bot}(m, n)$ in figure 3.9a, 3.9b and 3.9c respectively[4].



(a) $\varepsilon_{top}(m, n)$. **(b)** $\varepsilon_{mid}(m, n)$. **(c)** $\varepsilon_{bot}(m, n)$.

Figure 3.9: X directional strain component of $\varepsilon_{top}(m, n)$, $\varepsilon_{mid}(m, n)$ and $\varepsilon_{bot}(m, n)$. The most compressive strains are required in $\varepsilon_{top}(m, n)$ while the largest tensile strains are required in $\varepsilon_{bot}(m, n)$.

The X directional component of $\varepsilon_{path}(m, n)$, in the initial iteration $k = 1$ is shown in figure 3.10a. The required strain in $\varepsilon_{path}(m, n)$ reduces after each successful laser scan. This is shown in figure 3.10b where 10 laser scans, activating the temperature gradient mechanism[5], are performed in the Y direction.



(a) $\varepsilon_{path}(m, n)$ in the initial iteration $k = 1$. **(b)** $\varepsilon_{path}(m, n)$ after 10 laser scans.

Figure 3.10: $\varepsilon_{path}(m, n)$ in the initial iteration $k = 1$ and after 10 laser scans with static process variables activating the temperature gradient mechanism.

[4]Remark that the axes in figure 3.9 correspond to a 40x40 element discretisation of the 30x30x1mm blank used in all tests of framework.

[5]Utilised process variables; scan speed $V = 2750\frac{\text{mm}}{\text{min}}$, laser power $P = 380$W and laser beam diameter $d = 3$mm.

3.3.2 Manipulating the Strain Field

$\varepsilon_{path}(m,n)$ must be manipulated before path planning can be conducted. Three manipulated fields of $\varepsilon_{path}(m,n)$ are calculated being; The size of the minimum principal strain $\varepsilon_2(m,n)$, the orientation of the minimum principal strain $\theta_2(m,n)$ and a threshold $\varepsilon_{thresh}(m,n)$ of $\varepsilon_2(m,n)$. This section focuses on the procedure for calculating the three manipulations. The utilisation of the manipulations is elaborated in section 3.4.1. The pseudo code for manipulating $\varepsilon_{path}(m,n)$ is shown in algorithm 2 and described below. The pseudo code corresponds to line 220 to 221 of the Java main program appended in appendix E.

Algorithm 2 Pseudo code for manipulating $\varepsilon_{path}(m,n)$.

1: Receive $\varepsilon_{path}(m,n)$
2: Calculate size of the minimum principal strain $\varepsilon_2(m,n)$ of $\varepsilon_{path}(m,n)$
3: Calculate orientation of the minimum principal strain $\theta_2(m,n)$ of $\varepsilon_{path}(m,n)$
4: Define percentage of $\varepsilon_2(m,n)$ to threshold
5: Calculate threshold $\varepsilon_{thresh}(m,n)$ of $\varepsilon_2(m,n)$

Calculating Size and Orientation of Minimum Principal Strain

Line 1-3 of algorithm 2 shows that the first manipulations are related to calculating the size and orientation of the minimum principal strain. $\varepsilon_{path}(m,n)$ contains a strain tensor ε_{ij} for each element, describing the three dimensional state of strain required with respect to the cartesian coordinate system. The element may be rotated to a unique principal orientation, where only three principal strains are present and all shear strains are equal to zero. This orientation is referred to as the principal orientation. It is necessary to determine the size and orientation of the minimum principal strains as these are utilised in the scan path algorithm presented in section 3.4.1. To simplify the determination of the size and orientation it is assumed that through thickness deformations are negligible. Therefore, the problem is considered two dimensional ($\varepsilon_{zz} = \varepsilon_{yz} = \varepsilon_{zx} = 0$). In this case only the maximum principal strain ε_1 and minimum principal strain ε_2 corresponding to two principal orientations θ_1 and θ_2 acting in the XY plane are considered for each element in $\varepsilon_{path}(m,n)$. The size and orientation of the minimum principal strain is found by means of 2D strain rotation. Figure 3.11 shows that a 2D strain element rotated by a given angle θ, creates a set of new strains, ε'_{xx}, ε'_{yy} and ε'_{xy} acting on the sides of the element. By varying θ it is possible to identify the principle orientation of the element where only ε'_{xx}, ε'_{yy} act on the element and ε'_{xy} is equal to zero. At this direction ε'_{xx} equals the maximum principal strain ε_1 with the principal orientation θ_1 and ε'_{yy} equals the minimum principal strain ε_2 with the orientation θ_2, as shown in figure 3.11. The equations for determining the size and orientation of the minimum principal strain are appended in appendix J. [Gere and Goodno, 2009]

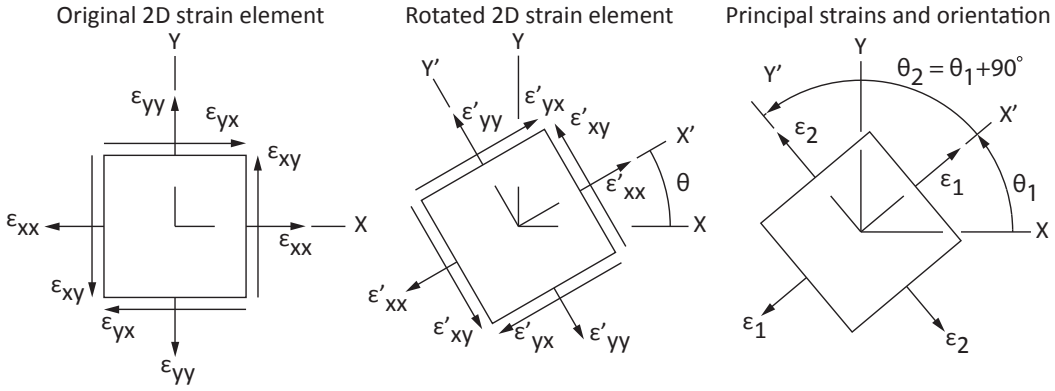
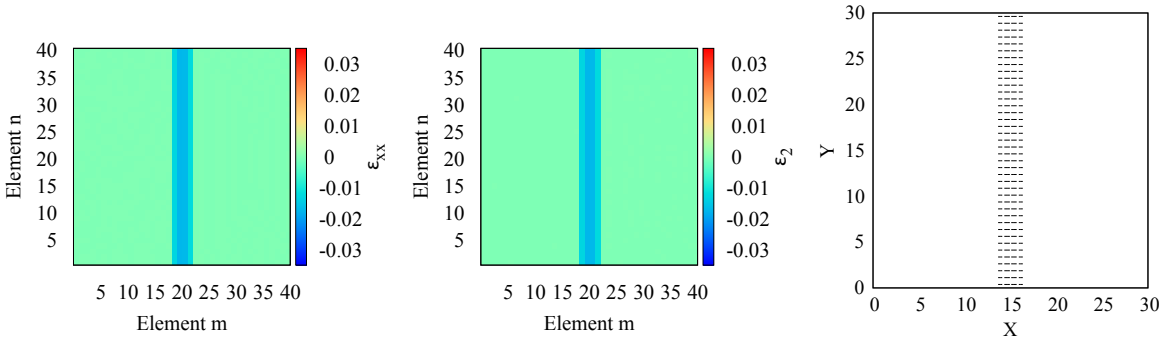


Figure 3.11: Strain rotation of a 2D element.

To check the validity of the presented procedure for determining the size and orientation of the minimum principal strain, the V-bend from figure 3.5 is considered. It is assessed that the most compressive strains are required in the top surface in the X direction. Thus, $\varepsilon_2(m,n)$ should be similar in size to the X directional strains of $\varepsilon_{path}(m,n)$. Figure 3.12a and figure 3.12b shows the X directional strains of $\varepsilon_{path}(m,n)$ and $\varepsilon_2(m,n)$ of $\varepsilon_{path}(m,n)$. It is seen that the strain fields are similar with respect to the size of the strains, as shown in figure 3.12a and 3.12b. Furthermore, as the largest compressive strains occur in the X direction, $\theta_2(m,n)$ produces orientations of minimum principal strain parallel to the X axis, as shown in figure 3.12c[6]. Due to the results presented in figure 3.12, it is assessed that the applied procedure for determining $\varepsilon_2(m,n)$ and $\theta_2(m,n)$ is satisfactory.



(a) ε_{xx} of $\varepsilon_{path}(m,n)$. **(b)** $\varepsilon_2(m,n)$. **(c)** $\theta_2(m,n)$.

Figure 3.12: Figure 3.12a and 3.12b shows the X directional strains of $\varepsilon_{path}(m,n)$ and $\varepsilon_2(m,n)$ of $\varepsilon_{path}(m,n)$. Figure 3.12c shows $\theta_2(m,n)$ for the V-bend.

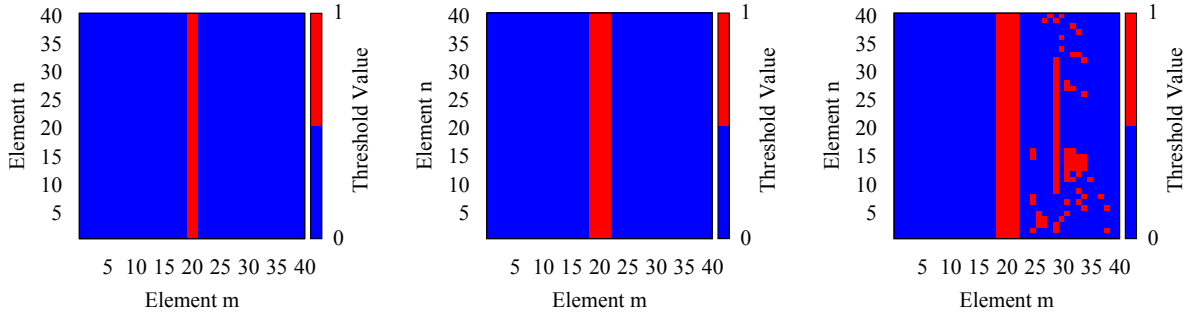
Calculating the Threshold

The amount of forming required to create a desired geometry varies across the surface, dependent of the desired geometry. When forming is induced in one area of the blank, residual stresses

[6]Remark that $\varepsilon_2(m,n)$ is shown with respect to the 30x30x1mm blank.

are introduced in the blank. The residual stresses may form the areas outside the scan path. This potentially over forms the blank in areas, where a smaller amount of forming is required. To reduce the risk of over forming, it is chosen to form the area requiring the largest amount of forming first. To ensure this, a threshold field $\varepsilon_{thresh}(m,n)$ of $\varepsilon_2(m,n)$ is created in line 4-5 of algorithm 2. An element in $\varepsilon_{thresh}(m,n)$ is set equal to zero if $\varepsilon_2(m,n)$ of the element is larger than a limit value and equal to 1 if $\varepsilon_2(m,n)$ is less than the limit. The limit value is determined by use of a threshold percentage. A threshold percentage of 10% results in a $\varepsilon_{thresh}(m,n)$ where 10% of the elements are set equal to 1 and the remaining are set to 0. As a result, 10% of the elements requiring the most compressive strains are set equal to 1 in $\varepsilon_{thresh}(m,n)$. The scan path can only be created in the area, where $\varepsilon_{thresh}(m,n)$ equals 1. This assures that forming is performed, where the largest deformation is required.

Considering the V-bend from figure 3.5, the most compressive strains of $\varepsilon_2(m,n)$ are required near the center along the Y direction. $\varepsilon_{thresh}(m,n)$ with threshold percentages of 5%, 10% and 15% is shown in figure 3.13a, 3.13b and 3.13c respectively. It is seen that the allowable area for the scan path increases with increasing threshold percentage. A threshold percentage of 5% results in a thin line across the surface, where $\varepsilon_{thresh}(m,n) = 1$. When the percentage is increased to 10% the line becomes wider and when increased to 15% it is seen that elements far from the bend are set equal to 1. This is due to a fixed amount of elements being set equal to 1. The task of selecting a threshold percentage depends on the selected desired geometry. In chapter 4 and 5 four geometries are tested using the developed framework. In each test, the chosen threshold percentage is discussed.



(a) Threshold percentage equal to 5%.

(b) Threshold percentage equal to 10%.

(c) Threshold percentage equal to 15%.

Figure 3.13: $\varepsilon_{thresh}(m,n)$ as a result of different threshold percentages. As the threshold percentage increases, the amount of elements set equal to 1 in $\varepsilon_{thresh}(m,n)$ increases as well.

3.4 Conduct Path Planning

This section describes the box *"Conduct path planning"*, shown in figure 3.4. The box treats the problem of generating scan paths suited for achieving the desired geometry based on the received strain fields from *"Perform strain analysis"*, as shown in figure 3.4. As mentioned in section 3.1, the scan path is placed perpendicular to the orientation of minimum principal strains. The main component of *"Conduct path planning"* is the scan path algorithm, which is elaborated in section 3.4.1.

3.4.1 The Scan Path Algorithm

The manipulated versions of $\varepsilon_{path}(m,n)$ are used as input to the scan path algorithm. The scan path algorithm generates a scan path for each iteration k of the framework by iteratively scanning the manipulated versions of $\varepsilon_{path}(m,n)$. For this purpose, an iteration counter c is introduced for the scan path algorithm. A search conducted by the scan path algorithm, iterates until a complete scan path is generated. For each search the iteration counter c is iterated. Scan paths are allowed to be placed similarly through several iterations. However, it is known that this affects the formability of the blank, why it must be addressed when implementing the framework in a physical setup [Madsen and Søndergaard, 2013]. The pseudo code for the scan path algorithm is shown in algorithm 3 and described in the following. The pseudo code corresponds to line 177 of the Java main program appended in appendix E.

Algorithm 3 Pseudo code for the scan path algorithm.

1: Receive $\theta_2(m,n)$, $\varepsilon_2(m,n)$, $\varepsilon_{thresh}(m,n)$ and define maximum accepted deviation angle α_{max}
2: Initialise iteration counter for the scan path algorithm $c = 1$
3: Determine initial start point p_{start} = element with largest compressive strain in $\varepsilon_2(m,n)$
4: Determine appropriate search stencil for p_{start}
5: Determine movement option vectors of the search stencil
6: Determine angle α between movement option vectors and $\theta_2(m,n)$
7: Select p_{best} as the movement option with α closest to 90° satisfying:
8: Criterion 1. $\varepsilon_{thresh}(p_{best}) = 1$
9: Criterion 2. α must be equal to $90° \pm \alpha_{max}$
10: Criterion 3. p_{best} not visited in current iteration k of the framework
11: **if** Criteria are met **then**
12: Define $p_{start} = p_{best}$
13: Iterate counter $c = c + 1$
14: Repeat from line 4
15: **else**
16: # Only reset the scan path algorithm once per iteration k of the framework
17: Reset the scan path algorithm from line 2
18: **end if**
19: End scan path algorithm

Line 1-2 of algorithm 3, shows that the scan path algorithm starts with receiving $\theta_2(m,n)$,

$\varepsilon_2(m,n)$ and $\varepsilon_{thresh}(m,n)$. Furthermore, a maximum accepted deviation angle α_{max} is defined and the iteration counter c is initialised. α_{max} defines the acceptable deviation with respect to perpendicularity between the scan path and the orientation of the minimum principal strain, as shown in figure 3.15b. Line 3 shows that the initial start point p_{start} of the scan path algorithm is determined as the element with the largest compressive strain of $\varepsilon_2(m,n)$. This ensures that the scan path algorithm is started at the element requiring the largest amount of forming. From p_{start} a search stencil is selected in line 4. In the initial iteration of the scan path algorithm $c = 1$, an eight point search stencil is selected. This search stencil provides eight movement options being the surrounding eight element centroids, as shown in figure 3.14. The angle α between the vector of the movement option and the orientation of the minimum principal strains is determined for all movement options, by means of the dot product, in line 6. The movement option which is closest to perpendicular and satisfies three criteria, given in line 8-10, with respect to threshold, perpendicularity and point repetition is considered the best solution p_{best}. p_{best} is used as p_{start} in iteration $c+1$ of the scan path algorithm.

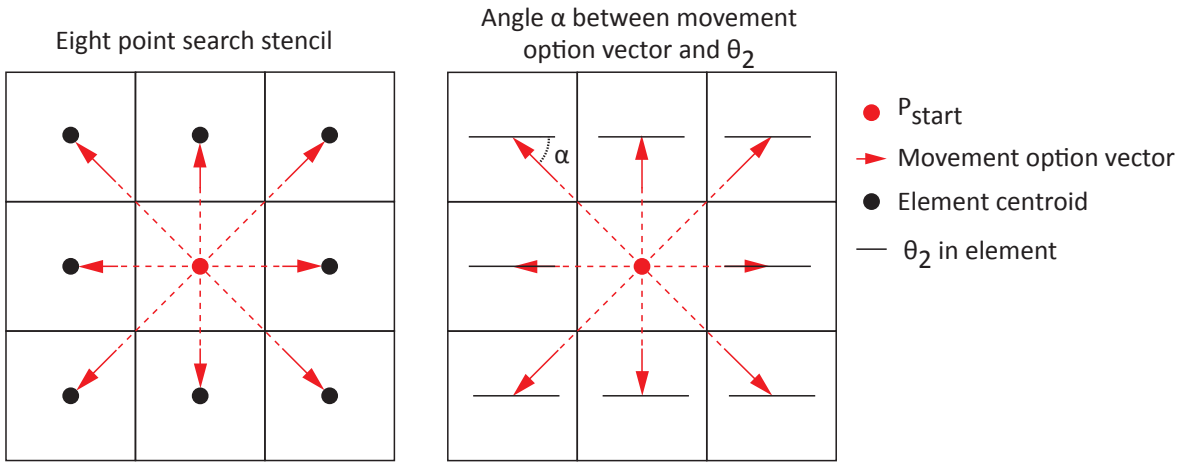
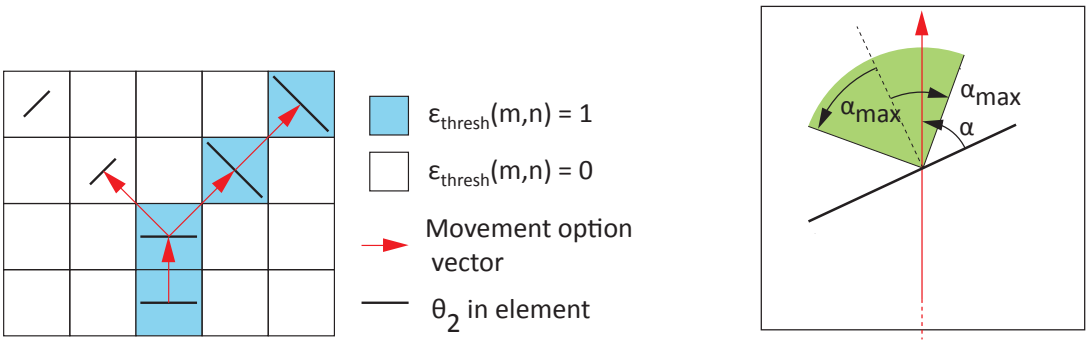


Figure 3.14: The eight point search stencil used if $c = 1$. This stencil provides eight movement options corresponding to the surrounding elements. The angle α is calculated between all vectors of the movement options and the orientation of the minimum principal strain in $\theta_2(m,n)$.

Criterion 1, shown in line 8 of algorithm 3, relates to $\varepsilon_{thresh}(m,n)$. The element located at p_{best} must equal 1 in $\varepsilon_{thresh}(m,n)$ to ensure that high areas requiring the largest amount of forming are formed first. As mention in section 3.3.2 this prevents over forming of the blank. The principle is shown in figure 3.15a. The blue squares denote the area where $\varepsilon_{thresh}(m,n) = 1$ and the white squares where $\varepsilon_{thresh}(m,n) = 0$. In figure 3.15a the scan path algorithm identifies two equally good solutions with respect to perpendicularity. However, $\varepsilon_{thresh}(m,n)$ determines that only one solution is suitable, as this requires more forming.

Criterion 2, shown in line 9 of algorithm 3, relates to allowable angular deviation from perpendicularity in the scan path. As the angle is determined by means of the dot product, the angle is always in the range of 0°→180°. A movement perpendicular to the orientation of minimum

principal strains ideally creates an angle α equal to 90°. However, the scan path algorithm is forced to search in discrete intervals dictated by the search stencil and the discretisation of the blank used in the Finite Element model. Therefore, a perfect perpendicularity cannot always be guaranteed. This is shown in figure 3.15b. To overcome this problem an angular deviation from perpendicularity is allowed. The maximum amount of angular deviation from perpendicularity is controlled by α_{max}.



(a) Criterion 1. **(b)** Criterion 2.

Figure 3.15: Figure 3.15a and 3.15b shows criterion 1 and 2 respectively. In criterion 1 the scan path algorithm is only allowed to move to element centroids, where $\varepsilon_{thresh}(m,n) = 1$. Criterion 2 prescribes that the scan path must not deviate more than α_{max} from perpendicularity. This is necessary as the discretisation of the Finite Element model and search stencil does not guarantee perpendicularity.

Criterion 3, shown in line 10 of algorithm 3, ensures that the same element is not scanned twice in a scan path. If the scan path algorithm is allowed to use the same element twice during path planning, the scan path algorithm may stagnate between two points, as shown to the left in figure 3.16. The stagnation occurs when the previous point is providing a better perpendicularity than the new movement option vectors. To overcome this, the scan path algorithm is not allowed to go to previously visited element centroids. In the scan path algorithm this is managed by "burning" the visited element in each iteration c, such that they are eliminated from the following iterations of the scan path algorithm. In the framework this is solved by comparing p_{best} with a new field $\varepsilon_{burn}(m,n)$, where all burned points are set equal to 1 and all non-visited points are 0. The principle is exemplified to the right in figure 3.16.

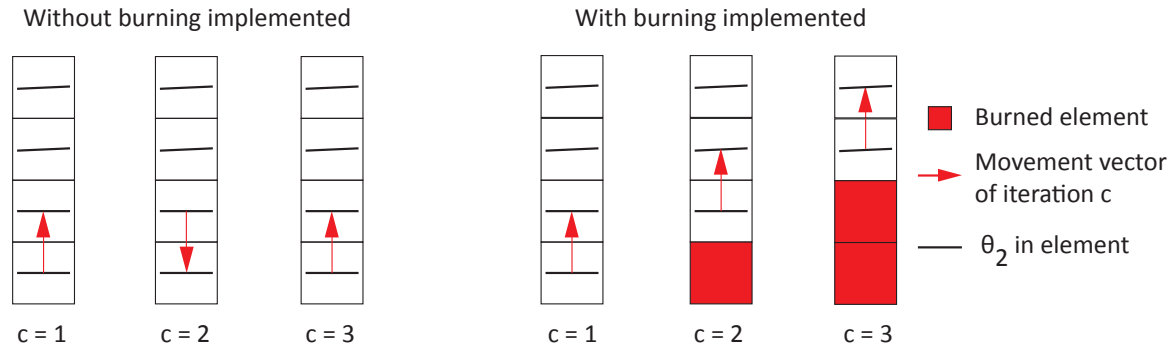


Figure 3.16: Criterion 3. To avoid stagnations of the scan path algorithm visited elements are burned, such that only element centroids not visited before are considered in the following iterations of the scan path algorithm.

If p_{best} satisfies all three criteria, it is accepted as p_{start} for iteration $c+1$ of the scan path algorithm. As described in line 4 of algorithm 3, an appropriate search stencil is selected for each iteration c of the scan path algorithm. Initially, the eight point search stencil, shown in figure 3.14, was utilised for all iterations of the scan path algorithm. The use of the eight point search stencil is exemplified in figure 3.17. The eight point search stencil determines a scan path, which moves upwards from $c = 1 \rightarrow 2$, but starts moving downwards right from $c = 2 \rightarrow 3$ and downwards from $c = 3 \rightarrow 4$. This is considered undesirable, as the scan path may stagnate in the bottom of the strain field, instead of producing a straight scan path, as shown in figure 3.17.

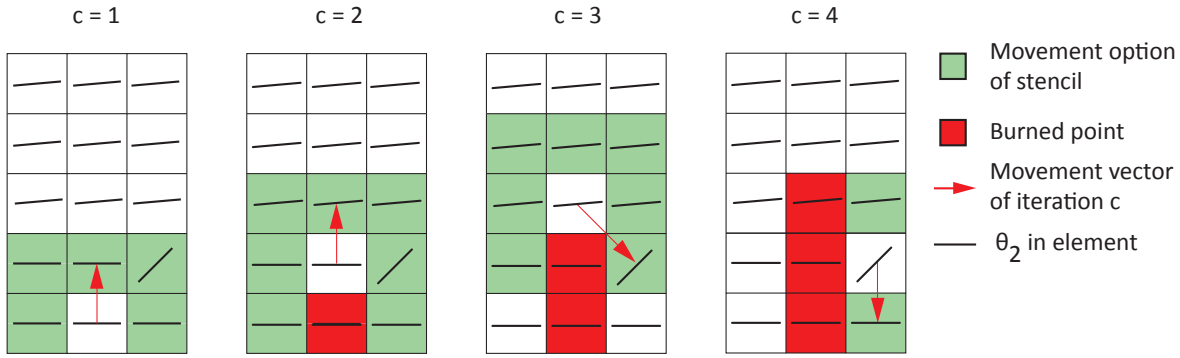


Figure 3.17: The scan path selected by use of the eight point search stencil.

To avoid the complications exemplified with the eight point search stencil, eight reduced search stencils are introduced, as shown in figure 3.18. The reduced search stencils correspond to the movement option vector selected in the previous iteration of the scan path algorithm $c-1$. The correlation between the movement direction of iteration $c-1$ and appropriate reduced search stencil for the current iteration c, is shown in figure 3.18. E.g. if the movement in $c-1$ is in an upwards direction, search stencil 2 is utilised in iteration c. Search stencil 2 provides three movement options in the upwards direction.

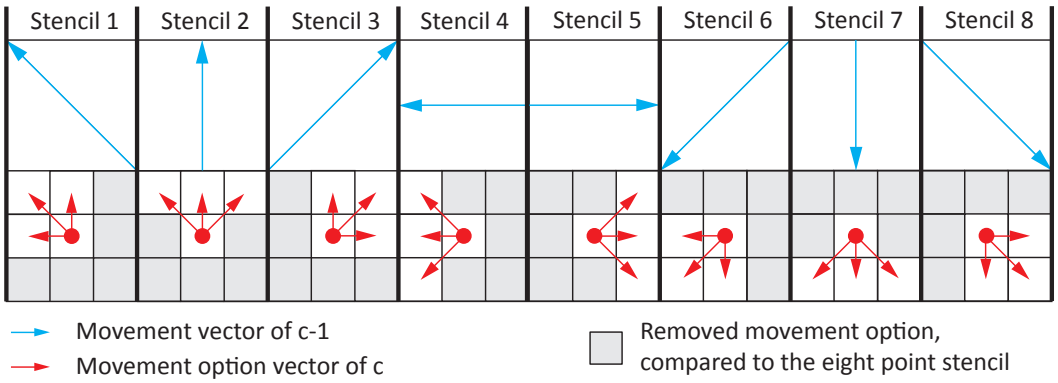


Figure 3.18: Correlation between the movement option vector selected in iteration $c-1$ of the scan path algorithm and the reduced search stencil selected for the current iteration c of the scan path algorithm.

The reduced search stencils are used to increase the robustness of the scan path algorithm, with regards to stagnation. This is exemplified in figure 3.19, where the reduced search stencil is applied to the same orientation field, as the eight point search stencil, for four iterations of the scan path algorithm. However, the eight point search stencil is used in $c = 1$, in order to initialise the scan path algorithm. In the following iterations the appropriate reduced search stencil is applied. The reduced search stencils result in a preferred direction of the scan path algorithm, as it continuously moves upwards in the present example. When moving upwards, the selected search stencil limits the ability of the scan path algorithm to move in a downwards oriented direction, in the following iteration $c+1$.

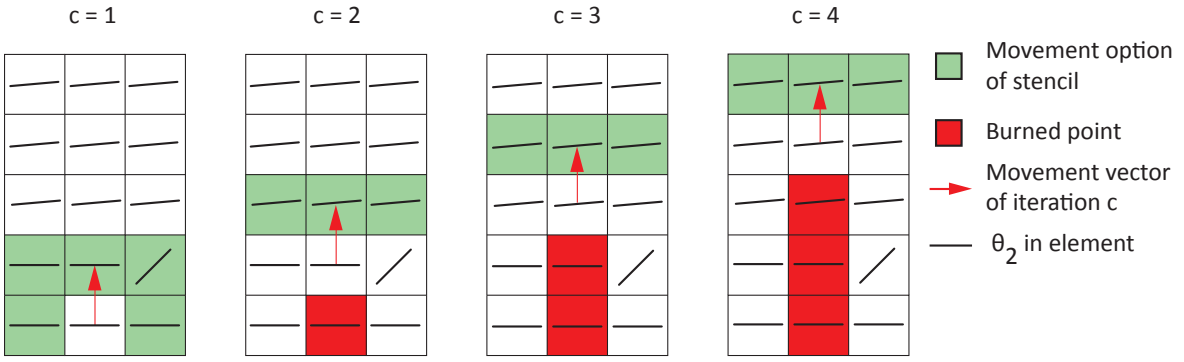


Figure 3.19: A reduced search stencil dictates a continuous upwards movement and results in a straight scan path.

The procedure of selecting an appropriate search stencil and identifying a new $p_{start} = p_{best}$ continues until no solutions are available for the scan path algorithm. When no solutions are obtained by the scan path algorithm, it is restarted from the initial p_{start}, as shown in line 16 of algorithm 3. This is exemplified in figure 3.20. In the initial search, the scan path algorithm searches upwards, until no solutions are available. In the second search, the scan path algorithm

starts at p_{start} of the initial iteration $c = 1$. As the initial search has burned the movement options in the upwards direction, the second best solution is moving downwards, until the scan path algorithm does not obtain a solution. After two searches, all visited points are combined to create a complete scan path for iteration k of the framework. In the scan path algorithm the start point of the scan path is always set equal to the end point of the second search while the end point of the scan path is set equal to the end point of the first search.

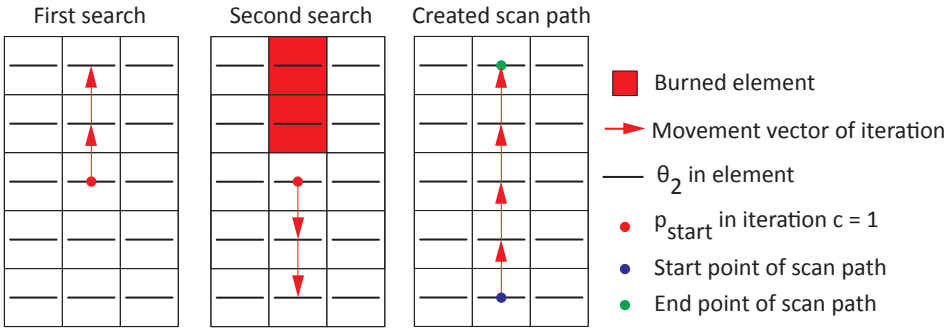
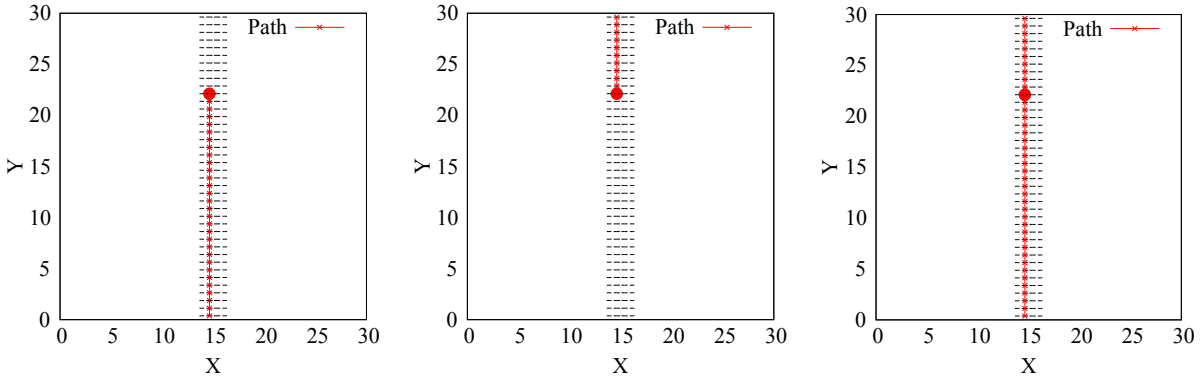


Figure 3.20: When no solutions are obtained by the scan path algorithm in the first search, it is restarted from the initial p_{start}. Elements which have been visited are burned, such that the second best solution is to move downwards. The two searches are combined to form one consecutive scan path.

Considering the V-bend from figure 3.5, figure 3.21 shows that the orientation of the minimum principal strains is parallel to the X axis. Therefore, the scan path algorithm should produce a scan path, moving in the Y direction. Figure 3.21 shows the scan paths obtained using the scan path algorithm with a threshold set to 10% and α_{max} equal til 50°. Figure 3.21a and 3.21b shows the first and second search of the scan path algorithm respectively. The combined scan path, for iteration $k = 1$ of the framework, is shown in figure 3.21c. The scan path algorithm is tested in chapter 4 and 5.



(a) Search one. **(b)** Search two. **(c)** Combined scan path.

Figure 3.21: Results of the scan path algorithm for the V-bend example. In search one, the scan path algorithm searches downwards and in search two, it searches upwards. When the two searches are combined, they produce the scan path for iteration $k = 1$ of the framework.

3.5 Determine Process Variables

This section describes the box *"Determine process variables"*, shown in figure 3.4. The box treats the problem of determining process variables with respect to the scan path received from *"Conduct path planning"*, as shown in figure 3.4.

The determination of process variables is performed by a developed control strategy, which determines how a state variable is transformed into suited process variables. In the framework, the state variable is defined as the required strain to go from current to desired geometry. Suited process variables are defined as, process variables ensuring that the formed geometry converges towards the desired geometry in each iteration of the framework.

It is chosen to use the Finite Element model of the laser forming process, to exemplify the importance of a control strategy. Figure 3.23 shows the output of the Finite Element model of the laser forming process, after one laser scan across the 30x30x1mm blank, introduced in chapter 2. The laser scan is performed with the scan path shown in figure 3.22 and static process variables, which activates the temperature gradient mechanism[7].

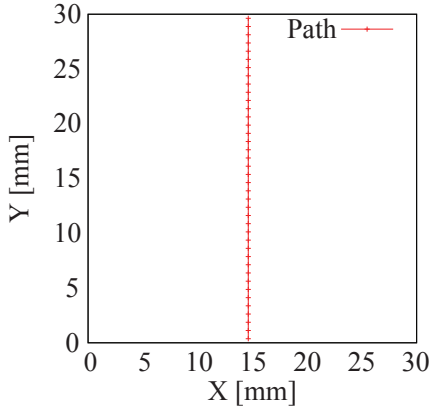


Figure 3.22: Scan path used in figure 3.23.

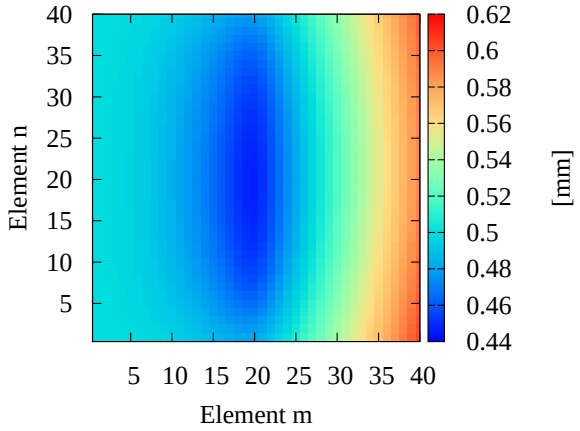


Figure 3.23: Output of the laser forming process after one laser scan with the scan path shown in figure 3.22.

From figure 3.23 it is observed that the amount of forming introduced in the blank, varies along the scan path, when using static process variables. To reduce the variance in the next iteration $k+1$ of the framework, the control strategy must adjust the process variables during the laser scan.

It is necessary to determine the process variables, which are to be controlled by the control strategy. During execution of the laser forming process, it is possible to adjust the laser beam

[7]Scan speed $V = 2750\frac{\text{mm}}{\text{min}}$, laser power $P = 380$W and laser beam diameter $d = 3$mm.

diameter, laser power and scan speed [Thomson and Pridham, 1997]. In [Madsen and Søndergaard, 2013], the control strategy adjusted the scan speed. An increase in scan speed results in less energy being absorbed by the blank, which leads to less forming. Inversely, a decrease in scan speed leads to increased forming. Acceptable results were obtained using the scan speed as the control variable, why this is also utilised in the present project. The remaining process variables i.e. laser beam diameter and laser power, are kept static through the process with respect to the given forming mechanism.

The pseudo code for the control strategy is shown in algorithm 4 and explained in the following. The pseudo code corresponds line 225 of the Java main program appended in appendix E. The control strategy is applied for all elements in the scan path, but to ease the explanation, only one element is covered in the pseudo code.

Algorithm 4 Pseudo code for *Determine process variables* in figure 3.4.

1: Receive scan path (array of element numbers)
2: Read required minimum principal surface strain in element $\varepsilon_{2,surf}$
3: Read required minimum principal midplane strain in element $\varepsilon_{2,mid}$
4: Calculate required in-plane strain in element ε_{in} and bending strain in element ε_{bend}
5: # Select forming mechanism based on the ratio between ε_{bend} and ε_{in}
6: **if** $\frac{\varepsilon_{bend}}{\varepsilon_{in}} < 1.0$ **then**
7: The upsetting mechanism is chosen
8: **else**
9: The temperature gradient mechanism is chosen
10: **end if**
11: # Select process variables with respect to forming mechanism and required strain
12: Determine element with largest required strain ε_{max} and smallest required strain ε_{min} in the scan path
13: **if** $\varepsilon_{max} > \varepsilon_{maxlim}$ **then**
14: Element with ε_{max} is assigned lowest scan speed
15: A scan speed for the remaining elements in the scan path is established with respect to a fraction distribution
16: **else**
17: Determine scan speed for element with ε_{max}
18: A scan speed for the remaining elements in the scan path is established with respect to a fraction distribution
19: **end if**
20: **if** $\varepsilon_{min} < \varepsilon_{minlim}$ **then**
21: Set laser power to zero
22: **end if**
23: Repeat from line 2 until all element in the scan path are treated
24: Write scan path and determined process variables to files, which are used by the Finite Element model

Initially, an array of the elements in the scan path is received from the scan path algorithm, described in section 3.4. For each element, the size of the minimum principal strain is obtained for the surface $\varepsilon_{2,surf}$ which is to be scanned and for the midplane $\varepsilon_{2,mid}$. $\varepsilon_{2,surf}$ and $\varepsilon_{2,mid}$ are used to calculate ε_{in} and ε_{bend}, as shown in figure 3.24. ε_{in} is set equal to $\varepsilon_{2,mid}$ and ε_{bend} is equal to $\varepsilon_{2,surf}$ subtracted ε_{in}.

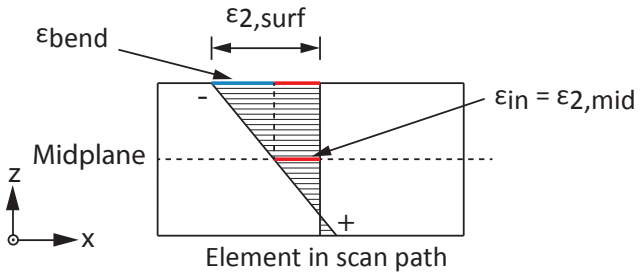


Figure 3.24: Definition of ε_{in} and ε_{bend}, where the red line indicate ε_{in} and the blue line indicates ε_{bend} equal to $\varepsilon_{2,surf}$ subtracted ε_{in}.

Line 6 of algorithm 4 defines the condition used to switch between the temperature gradient mechanism and the upsetting mechanism. The switch condition is the ratio between ε_{bend} and ε_{in}, if less than 1 then ε_{in} is dominant, why the upsetting mechanism must be utilised. If the ratio is greater than 1 the temperature gradient mechanism is utilised. When a forming mechanism has been determined, it is necessary to calculate suited process variables for each individual element in the scan path. Line 11-22 of algorithm 4 introduces the control strategy for determining process variables. In the following, the strategy is explained with regards to a scenario, where the temperature gradient mechanism has been chosen with respect to the switch condition. This implies that ε_{max} and ε_{min} in algorithm 4 are given with regards to bending strains, why these are reformulated as $\varepsilon_{max,bend}$ and $\varepsilon_{min,bend}$ respectively. If the upsetting mechanism is chosen, the process variables would be determined with regards to in-plane strains.

Prior to the calculation, the largest bending strain $\varepsilon_{max,bend}$ and smallest bending strain $\varepsilon_{min,bend}$ located in the scan path are extracted. These are used to evaluate whether a large or small amount of forming is required to obtain the desired geometry. It is desired to design a control strategy which ensures a fast rate of convergence, whilst maintaining robustness. Therefore, the developed control strategy must ensure a maximum amount of forming when large strains are required, whilst reducing the amount of forming when approaching the desired geometry. To comply with this requirement, it is necessary to determine the limitations of the laser forming process with regards to obtainable strains. These limitations represent the amount of forming which can be produced with the lowest and highest scan speed within the process range for the given mechanism. The limitations are defined as ε_{maxlim} and ε_{minlim} for the lowest and highest scan speed respectively. The limitations refer to bending strain with regards to the temperature gradient mechanism and in-plane strain with regards to the upsetting mechanism. These limitations assure that the maximum amount of forming is produced if the required strains are larger than ε_{maxlim}. If lower than ε_{maxlim}, then the forming is reduced and if lower than ε_{minlim} the

laser power is set equal to zero, such that no forming is induced.

As shown in line 13, if $\varepsilon_{max,bend}$ is larger than ε_{maxlim}, then the element is prescribed the lowest scan speed within the process range and a fraction distribution is used to determine the scan speed V_{el} for the remaining elements. The fraction distribution is shown in equation 3.1. The fraction distribution ensures that even if all elements are above ε_{maxlim}, the scan speed is varied as a function of strain. This is assessed to reduce the variance in the forming observed in figure 3.23, where static process variables resulted in variance.

$$V_{el} = 2750\frac{\text{mm}}{\text{min}} + (7000\frac{\text{mm}}{\text{min}} - 2750\frac{\text{mm}}{\text{min}}) \cdot \left(\frac{\varepsilon_{\text{max,bend}} - |\varepsilon_{\text{bend}}|}{\varepsilon_{\text{max,bend}}}\right) \tag{3.1}$$

In equation 3.1, the fraction between $\varepsilon_{max,bend}$ and the bending strain ε_{bend} of the individual element is multiplied by the process range of the temperature gradient mechanism ($7000\frac{\text{mm}}{\text{min}} - 2750\frac{\text{mm}}{\text{min}}$). If the fraction equals zero, the element requires the same amount of strain as $\varepsilon_{max,bend}$ i.e. receives the lowest scan speed equal to $2750\frac{\text{mm}}{\text{min}}$. The value of the fraction increases as smaller ε_{bend} are measured, resulting in an increase in scan speed up to $7000\frac{\text{mm}}{\text{min}}$.

If $\varepsilon_{max,bend}$ is lower than ε_{maxlim}, it is necessary to increase the scan speed V_{el} assigned the element requiring $\varepsilon_{max,bend}$ to avoid overforming. The scan speed V_{el} is increased with respect to equation 3.2.

$$V_{inc} = 2750\frac{\text{mm}}{\text{min}} + \left(\frac{\varepsilon_{\text{maxlim}} - \varepsilon_{\text{max,bend}}}{\varepsilon_{\text{maxlim}}}\right) \cdot \left(7000\frac{\text{mm}}{\text{min}} - 2750\frac{\text{mm}}{\text{min}}\right) \tag{3.2}$$

Where V_{inc} is the increased scan speed. The increase in scan speed, is based on the fraction between ε_{maxlim} and $\varepsilon_{max,bend}$. The increased scan speed is subtracted the highest scan speed, as shown in equation 3.3, in order to obtain a new range V_{new} in which the temperature gradient mechanism can be controlled. V_{inc} and V_{new} are used to create the new fraction distribution, shown in equation 3.4.

$$V_{new} = 7000\frac{\text{mm}}{\text{min}} - \text{V}_{\text{inc}} \tag{3.3}$$

$$V_{el} = V_{inc} + V_{new} \cdot \left(\frac{\varepsilon_{max,bend} - |\varepsilon_{bend}|}{\varepsilon_{max,bend}}\right) \tag{3.4}$$

Furthermore, ε_{bend} is checked with regards to ε_{minlim}, if lower, then the laser power is set to zero for that element, such that no forming is introduced.

The control strategy explained through this section is exemplified in figure 3.25 and 3.26. Figure 3.25 shows the required bending strains to go from a current geometry to the desired V-bend. The current geometry has been laser scanned 15 times with static process variables, activating the temperature gradient mechanism[8]. It is seen, that the required bending strains vary across the width of the blank, which corresponds to the uneven amount of forming shown in figure

[8]Scan speed $V = 2750\frac{\text{mm}}{\text{min}}$, laser power $P = 380$W and laser beam diameter $d = 3$mm.

3.23. Using the control strategy introduced through this section, the scan speed is distributed as shown in figure 3.26. In 3.26 it is shown that, the elements from figure 3.25, which requires less strain are prescribed a higher scan speed.

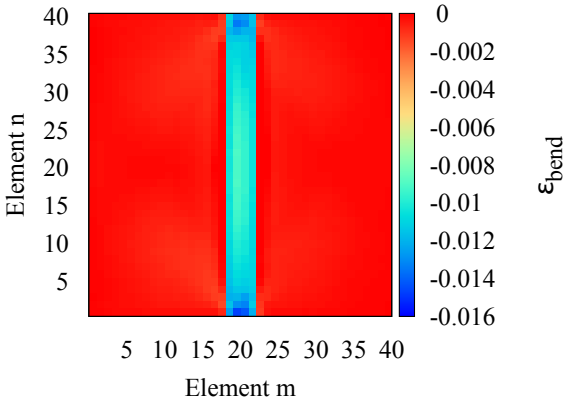


Figure 3.25: Required bending strains in order to obtain a V-bend geometry.

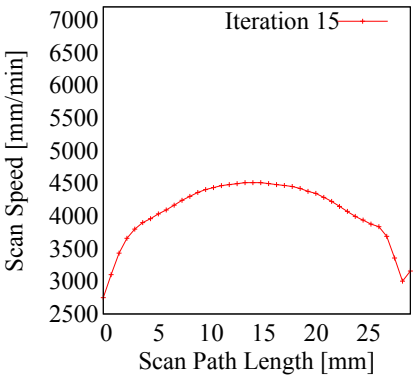


Figure 3.26: Determined scan speed in order to obtain the required bending strains in figure 3.25.

The functionality of the control strategy is tested in chapter 4 and 5.

3.6 Stop Criterion

This section describes the box *"Stop criterion"*, shown in figure 3.4. The box treats the problem of defining a stop criterion for the framework. The stop criterion is determined with respect to a desired geometry and a current geometry received from *"Create desired geometry"* and *"Finite Element model of the laser forming process"*, as shown in figure 3.4.

From section 3.1.2 it was determined to use a conservative multipass approach to allow the implementation of feedback control in the laser forming process. The current geometry is evaluated with respect to the desired geometry in every iteration of the framework. The evaluation is performed with respect to a stop criterion described in this section.

A stop criterion must be formulated, which terminates the framework, when the deviation between the current and the desired geometry is acceptable. The stop criterion can be formulated in several different ways e.g. by curvature or maximum deviation between the current and the desired geometry, the formulation depends on requirements to the end product. Since there is no specific end product in this project, the stop criterion is formulated as the sum of absolute error between the current and the desired geometry. The sum of absolute error means that for every element centroid in the current geometry, the distance in the Z direction between the current and the desired geometry is calculated in order to obtain an error. The absolute value of this error is summed for all elements. This formulation of the stop criterion provides a scalar measure of the correlation between the current and the desired geometry.

To evaluate the current geometry with respect to the desired geometry, it is chosen to extract the Z coordinates for all element centroids[9]. In order to compare the extracted Z coordinates it is necessary to perform a mapping from the desired to the current geometry. Initially a 1-1 mapping between the element centroids of the current and the desired geometry was considered. The 1-1 mapping approach is exemplified using a 2D scenario, which is shown in figure 3.27. An error in the X direction is present between the element centroids of the current and the desired geometry. This error increases as the desired geometry is more heavily formed. The error in the X direction results in a misleading measurement of the Z coordinate and thereby a poor approximation of the geometrical correlation between the current and the desired geometry.

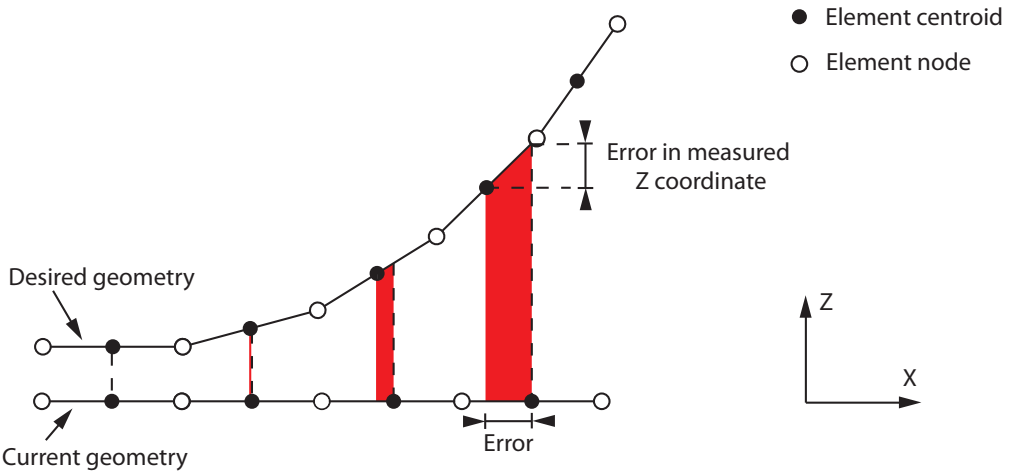


Figure 3.27: 1-1 mapping approach between the blank and the desired geometry. The hollow circles represent nodal points, the solid circles represent element centroids and the red area indicates the error in in-plane location of the element centroid.

To overcome the issue related to the 1-1 mapping, it is chosen to evaluate the location of each element centroid with respect to their X coordinate (2D example). The mapping is performed by reading the X coordinates of all the element centroids in the blank and structure these with respect to the value of the X coordinate. Subsequently, all elements in the desired geometry are compared to the elements of the blank and the best match with regards to X coordinate is stored. The improved mapping approach is shown in figure 3.28, where it is shown that the error in X direction is generally minimised, thereby minimising the value of the misleading Z coordinate. To improve the performance of this approach, it is chosen to increase the mesh density of the desired geometry, thereby minimising the maximum distance to the best match.

[9]Through this project the information located at the element centroids is used, as all strain information used for the path planning described in section 3.4 is given at the element centroids.

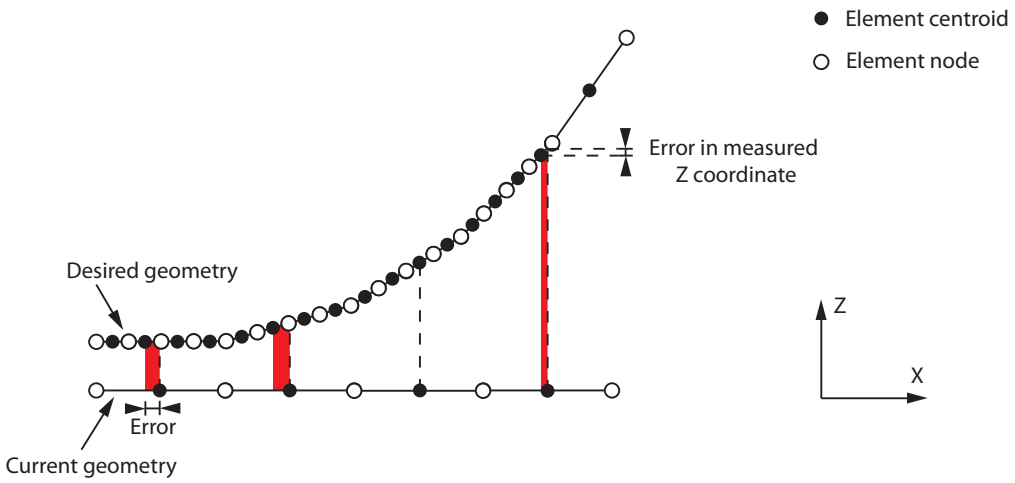


Figure 3.28: Improved mapping approach between the blank and the desired geometry. The hollow circles represent nodal points, the solid circles represent element centroids and the red area indicates the error in in-plane location of the element centroid.

The mapping is performed for all elements in the current geometry. The mapped Z coordinates from the desired geometry are subtracted from the Z coordinates of the current geometry, thereby, obtaining an error. The absolute value of the error is summed for all elements to create the scalar measure i.e. the sum of absolute error.

As the purpose of this project is to proof the concept of the framework, no exact value of the stop criterion is set. However, an analysis of the development of the sum of absolute error, is performed in chapter 4 and 5. Since, no exact value of the stop criterion is set, it is chosen to implement a requirement with regards to maximum number of iterations to ensure that the framework is terminated.

3.7 Sub Conclusion

The developed framework is based on the concept developed by [Liu and Yao, 2005], in which scan paths are placed perpendicular to the orientation of the minimum principal strain and process variables are selected, based on the required strain in the scan path. The developed framework was created as a feedback loop, to cope with the poor repeatability of the laser forming process. The framework calculates a scan path and suited process variables based on the required strain to go from the current to the desired geometry, in each iteration. The scan path is generated with a developed scan path algorithm. To reduce the risk of over forming the blank, scan paths are generated in areas requiring the largest amount of forming first. The scan path is placed perpendicular to the orientation of minimum principal strain, in a scan path strain field, as the largest compressive strains induced by the laser forming process occurs perpendicular to the scan path. The scan path strain field is created as the average of the required midplane strain field and the surface strain field requiring the largest compressive strain, as these strain fields indicate

the in-plane and bending strain required to form a desired geometry. To ensure that the current geometry converges towards the desired geometry, suited process variables are determined for all elements in the scan path, with respect to the required bending and in-plane strains. The scan speed was selected as the control variable, as it is possible to control the amount of forming induced in the blank by adjustment of the scan speed. The scan speed is varied along the scan path as a function of the required strains to reduce the variance in the process output. A stop criterion was formulated to stop the framework. The stop criterion is formulated with respect to the sum of absolute error, as this provides a scalar measure of the correlation between the current and the desired geometry. As the purpose of this project is to proof the concept of the framework, no exact value of the stop criterion was set.

Chapter 4

Test of the Framework on Single Curved Geometries

This chapter presents two tests performed on single curved geometries, using the framework developed in chapter 3. The purpose of this chapter is to perform a proof of concept of the framework, with regards to the production of single curved geometries. This chapter consists of; tests of the framework and discussions of results and selected observations.

The desired geometries are formed from a square blank measuring 30x30x1mm, as mentioned in section 2.1. The blank is fixed at X=0, as shown in figure 4.1. The desired geometries are shown in figure 4.2. The desired geometries are a 10° V-bend geometry and a cosinusoidal geometry. CAD-files of the desired geometries are appended on the appendix-CD. The geometries are single curved, why forming is only required parallel to the XZ plane. Both desired geometries are developable surfaces, why they are obtainable by utilising the temperature gradient mechanism.

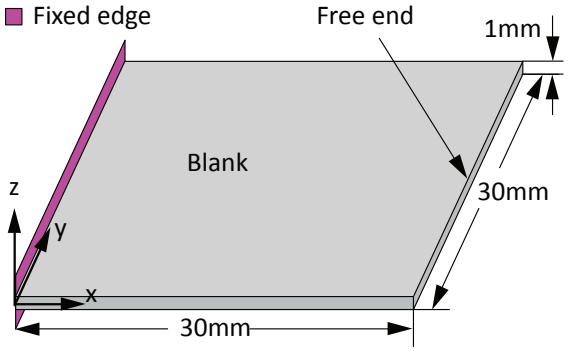


Figure 4.1: Dimensions of the square blank. The blank is fixed at X=0.

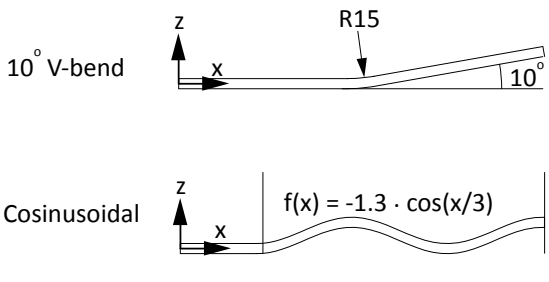


Figure 4.2: Tested geometries being; a 10° V-bend geometry and a cosinusoidal geometry.

In tests of single curved geometries, the relative size and orientation of the minimum principal strains, and therefore also placement of scan paths and selection of process variables, are predictable by intuition. The 10° V-bend was used for preliminary testing and debugging of the

tasks constituting the framework. The cosinusoidal geometry was used to test the ability of the framework to perform laser scans on both sides of the blank. As the tests function as proof of concept, no concrete requirements are established for the output of the framework. Instead it is chosen to analyse the output of a series of iterations of the framework.

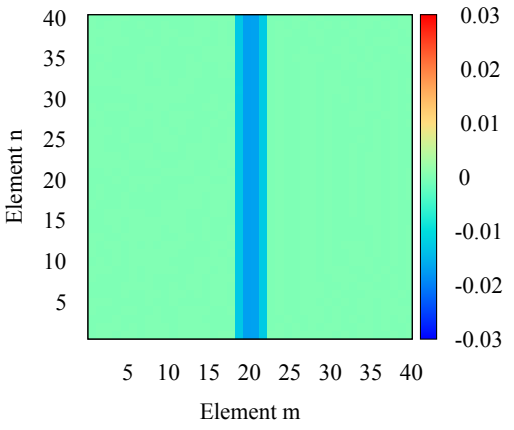
The tests of the single curved geometries are conducted with the settings of the framework shown in table 4.1. The selected settings were determined by trial of the framework. Settings providing a higher rate of convergence or a better correlation between the current and desired geometry may be found. However, as the purpose is to proof the concept of the framework, the settings are accepted.

Parameter	**10° V-bend geometry**	**Cosinusoidal geometry**
Threshold percentage	10%	50%
Allowable angular deviation α_{max}	50°	50°
Maximum number of iterations	100	200

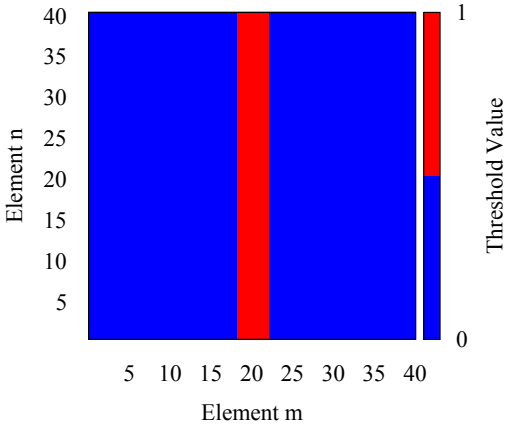
Table 4.1: Settings used for the tests performed on the single curved geometries.

4.1 Test of the 10° V-bend Geometry

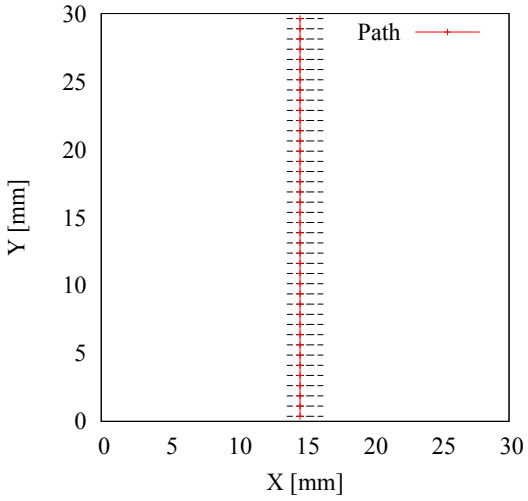
The framework receives the 10° V-bend geometry shown in figure 4.2 and the settings shown in table 4.1. Test results are shown in figure 4.3. Figure 4.3a to 4.3c shows; the size of the minimum principal strains $\varepsilon_2(m,n)$ in $\varepsilon_{path}(m,n)$, the threshold $\varepsilon_{thresh}(m,n)$ of $\varepsilon_2(m,n)$ and the orientation of the minimum principal strains $\theta_2(m,n)$ and scan path, for the first iteration of the framework. Figure 4.3d shows the development of the sum of absolute error as a function of the iterations of the framework.



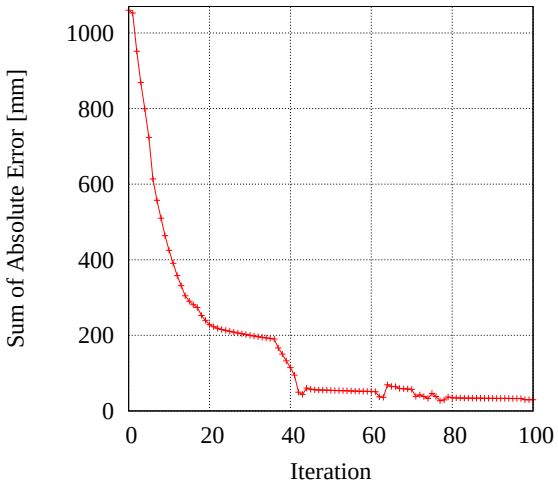
(a) $\varepsilon_2(m,n)$.



(b) $\varepsilon_{thresh}(m,n)$.



(c) Scan path and $\theta_2(m,n)$.



(d) Sum of absolute error.

Figure 4.3: Results from the 10° V-bend test, with a threshold percentage of 10%, a maximum amount of iterations equal to 100 and $\alpha_{max} = 50°$. Figure 4.3a to 4.3c show the results from the first iteration of the framework, while figure 4.3d shows the sum of absolute error for all 100 iterations.

The V-bend requires the largest compressive strains in the top surface near the bend. Therefore, figure 4.3a to 4.3c relates to the upper $\varepsilon_{path}(m,n)$, shown in figure 4.4.

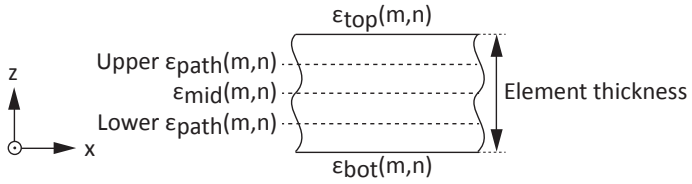
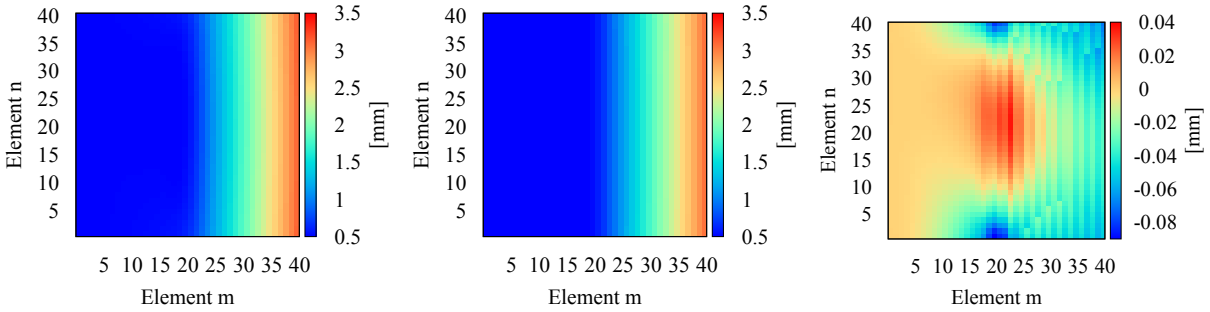


Figure 4.4: Placement of the strain fields.

Figure 4.3d shows, that the sum of absolute error converges towards zero. The smallest sum of absolute error is reached at a value of 29.60 in iteration 100 of the framework. Figure 4.5a shows a surface plot of the current geometry in iteration 100 of the framework. By comparing the current geometry with the desired geometry, shown in figure 4.5b, the surface error, shown in figure 4.5c, is obtained. It is shown, that the maximum deviation between the current and the desired geometry is |-0.08mm|, which is assessed to proof the concept of the developed framework. However, figure 4.5c, shows that there is a potential for improving the selection of process variables, as the error distribution varies across the width of the blank. To improve the selection of process variables, it is assessed that the utilised control strategy must be redesigned. A suggestion for a redesign is discussed in chapter 7.



(a) Current geometry.

(b) Desired geometry.

(c) Error between desired and current geometry.

Figure 4.5: Figure 4.5a shows the obtained geometry after 100 iterations of the framework. Figure 4.5b shows the desired geometry and figure 4.5c shows the error between the current and the desired geometry.

4.1.1 Observations from the Test of the 10° V-bend Geometry

Through this section two observations are discussed with respect to the results presented in figure 4.3.

Observation 1 - Decreasing Rate of Convergence from Iteration 1 to 36

Figure 4.3d, shows that the rate of convergence reduces with increasing iterations of the framework from iteration 1 to 36. This is assumed to be a result of the determined process variables during the laser scan. As mentioned in section 3.5, the control strategy adjusts the scan speed in order to cope with the varying amount of required strain. Figure 4.6 shows the determined scan

speed for iteration 1, 10, 20 and 30 of the framework. It is seen that the scan speed increases, as the number of iterations increase.

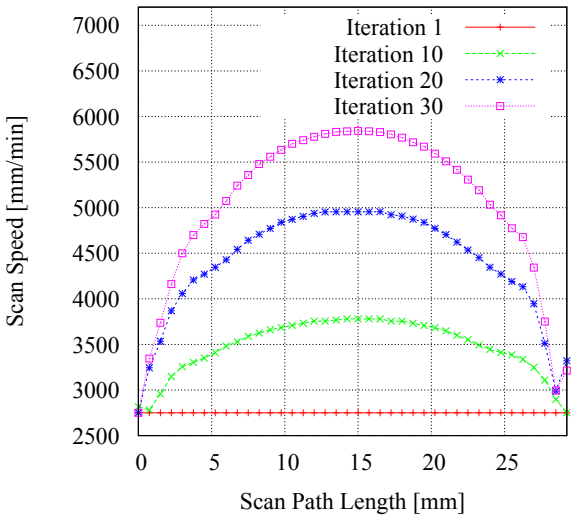


Figure 4.6: Scan speed iteration 1, 10, 20 and 30 of the framework.

As the scan speed is increased, the amount of forming performed in each iteration of the framework decreases. Thus, the decreasing rate of convergence is assessed to be caused by the increase in scan speed in each iteration.

Observation 2 - Fast Rate of Convergence from Iteration 37 to 42

A fast rate of convergence is achieved from iteration 37 to 42, compared to preceding iterations, as shown in figure 4.7. By comparing the scan paths constructed from iterations 37 to 42 and the scan paths constructed in the preceding iterations, it is observed that the scan paths from iteration 37 to 42 are placed closer to the fixed edge of the blank than the preceding iterations. Laser scans that are placed closer to the clamp, as shown in figure 4.8, have a larger impact on the sum of absolute error, as the area of the blank moved towards the desired geometry is increased i.e. the area between the scan path and the free end. Furthermore, several of the preceding scan paths were identical. Performing laser scans in the same scan path increases the amount of strain hardening in the area around the scan path. This is known to result in a decreased bend rate [Dahotre and Harimkar, 2008]. Therefore, the amount of forming increases, when moving the scan path to a new position.

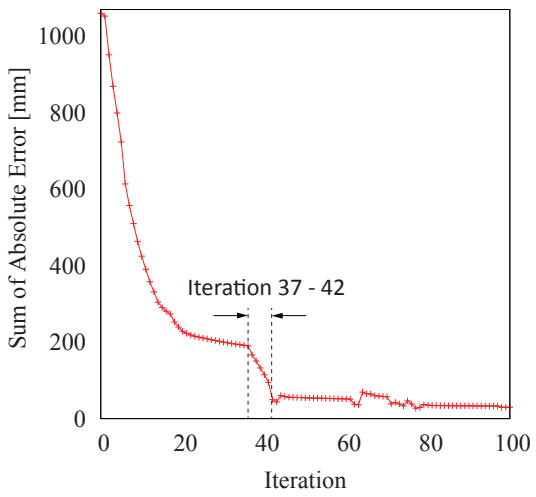


Figure 4.7: The sum of absolute error, where iteration 37 and 42 of the framework are designated.

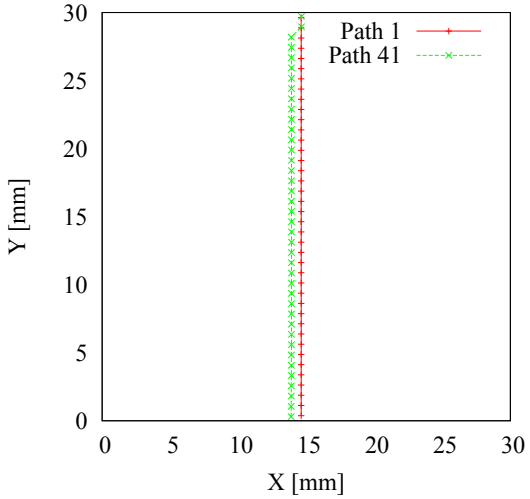
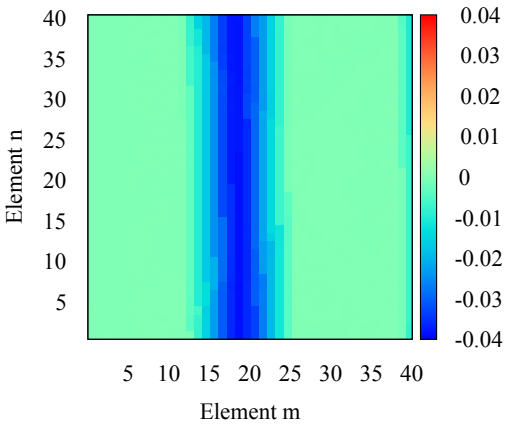


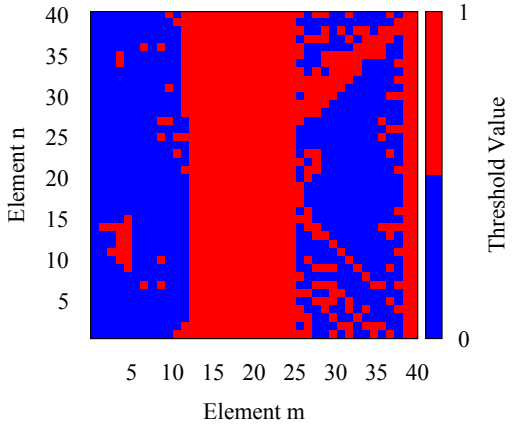
Figure 4.8: The scan paths of iteration 1 and 41 of the framework. The scan path of iteration 41 is placed closer to the fixed edge (X = 0).

4.2 Test of the Cosinusoidal Geometry

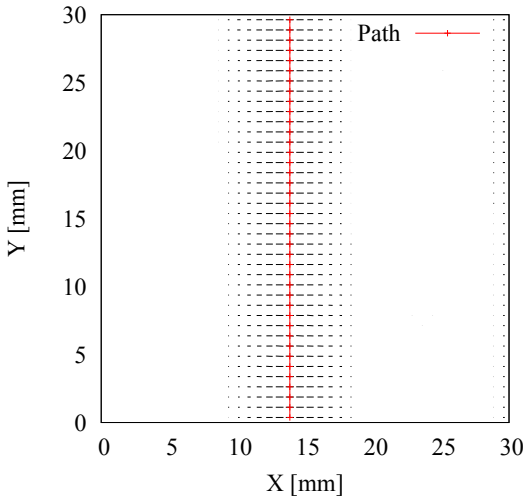
The framework receives the cosinusoidal geometry shown in figure 4.2 and the settings shown in table 4.1. Results are shown in figure 4.9. Figure 4.9a to 4.9c shows; the size of the minimum principal strains $\varepsilon_2(m,n)$ in $\varepsilon_{path}(m,n)$, the threshold $\varepsilon_{thresh}(m,n)$ of $\varepsilon_2(m,n)$ and the orientation of the minimum principal strains $\theta_2(m,n)$ and scan path, for the first iteration of the framework. Figure 4.9d shows the development of the sum of absolute error as a function of the iterations of the framework. The cosinusoidal geometry requires laser scans on both the top and bottom surface. In the first iteration of the framework, the most compressive strain is required on the bottom surface. Therefore, figure 4.9a to 4.9c relates to the lower $\varepsilon_{path}(m,n)$.



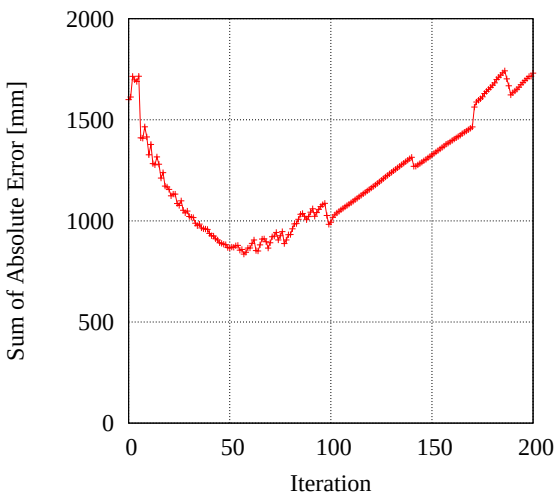
(a) $\varepsilon_2(m,n)$.



(b) $\varepsilon_{thresh}(m,n)$.



(c) Scan path and $\theta_2(m,n)$.



(d) Sum of absolute error.

Figure 4.9: Results from the test of the cosinusoidal geometry, with a threshold percentage of 50%, a maximum amount of iterations equal to 200 and $\alpha_{max} = 50°$. Figure 4.9a to 4.9c show the results from the first iteration of the framework, while figure 4.9d shows the sum of absolute error for all 200 iterations.

Through the iterations, the surface requiring the most compressive strain switch. Figure 4.10 shows the switching between the surfaces for the first 30 iterations of the framework. The test of the cosinusoidal geometry shows that the framework successfully performs laser scans on both surfaces of the blank. Figure 4.10, shows that the bottom surface is scanned in iteration 1, while the top surface is scanned in iteration 2. Figure 4.11 show the orientation of the minimum principal strains $\theta_2(m,n)$ and generated scan path for iteration 2. By comparing figure 4.9c and 4.11, it is seen that $\theta_2(m,n)$ is located inversely in the two iterations. This is a result of the cosinusoidal geometry varying with respect to the function $f(x) = -1.3 \cdot \cos(\frac{x}{3})$.

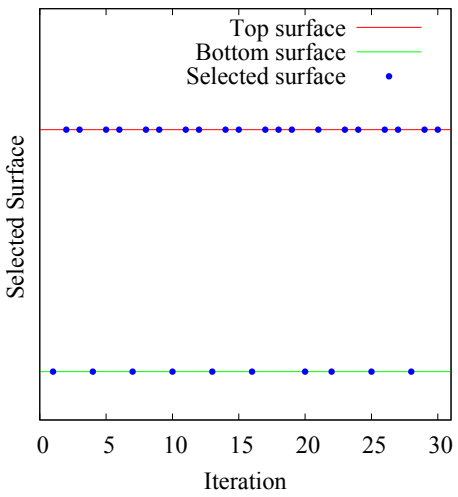


Figure 4.10: Switching of surfaces to be laser scanned, with respect to iterations of the framework.

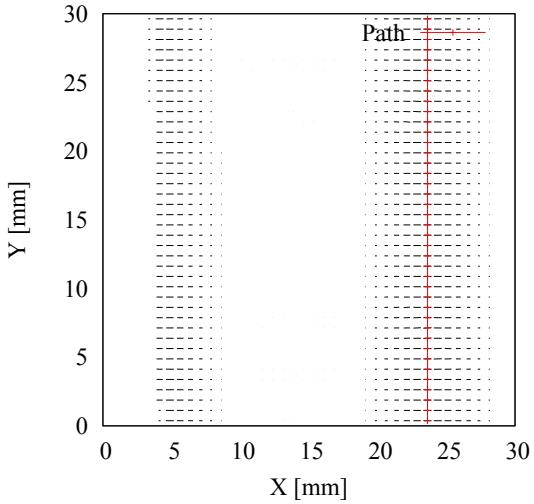


Figure 4.11: $\theta_2(m,n)$ and scan path for iteration 2 of the framework.

4.2.1 Observations from the Test of the Cosinusoidal Geometry

Through this section one observation is discussed with respect to the results presented in figure 4.9.

Observation 1 - Divergence of the Sum of Absolute Error

Figure 4.12, shows that the sum of absolute error converges in the initial 57 iterations of the framework. However, divergence is seen from iteration 58 to 200. To investigate the cause of the divergence, the constructed scan paths from iteration 58 to 200 are investigated. Figure 4.13 shows the distribution of laser scans performed on the top and bottom surface, within certain intervals of the blank, from iteration 58 to 200. 84.6% of the laser scans are placed on the top surface within an interval of 1-6mm from the fixed edge.

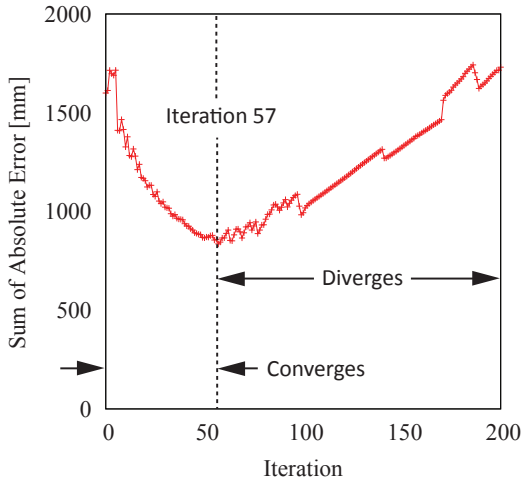


Figure 4.12: The sum of absolute error.

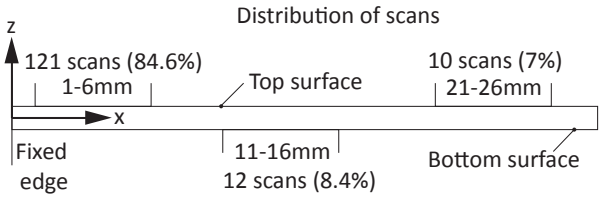


Figure 4.13: Distribution of laser scans from iteration 58 to 200 of the framework.

To check the effect of the laser scans performed in the divergent interval, the current geometry after 58 iterations and 200 iterations are compared. Figure 4.14 compares both geometries to the desired geometry. It is seen that both geometries exert the appearance of a cosinusoidal geometry. Furthermore, it is shown that the current geometry from iteration 200 has obtained a greater bend, close to the clamp, as more laser scans have been performed here, as shown in figure 4.13. The increased amount of forming in the interval, results in a rotation of the cosinusoidal appearance of the current geometry. The rotation causes a local geometrical convergence towards the desired geometry. However, it is also assessed to be the reason for the increased divergence in sum of absolute of error, as the current geometry further away from the clamp is moved away from the desired geometry.

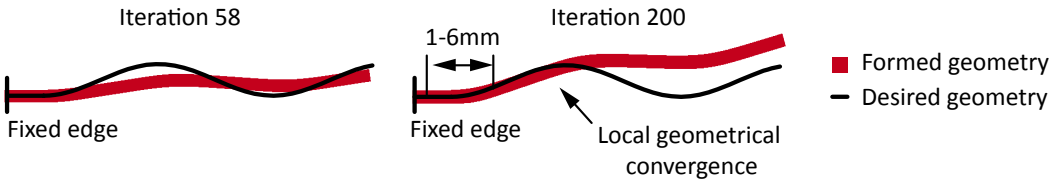


Figure 4.14: The formed geometries after 58 and 200 iterations of the framework.

As the current geometry after 200 iterations has converged locally and resembles the geometry of a cosinusoidal geometry, it is assessed that the sum of absolute error can provide a misleading measure of the current geometry's progress towards the desired cosinusoidal geometry. It is chosen to investigate the required strain to be achieved in the blank, as this can also be used to check the progress of the process. This is suitable as each successive laser scan decreases the required strain and the desired geometry is achieved when the required strain reaches zero. As a measure of the required strain, figure 4.15 shows the sum of absolute size of minimum principal strains for the upper and lower $\varepsilon_{path}(m,n)$, as a function of laser scans performed on

the respective surfaces. It is seen that both converge towards zero. As neither equals zero, further forming is required to reach the desired geometry, which is also evident from the formed geometry achieved in iteration 200, shown in figure 4.14.

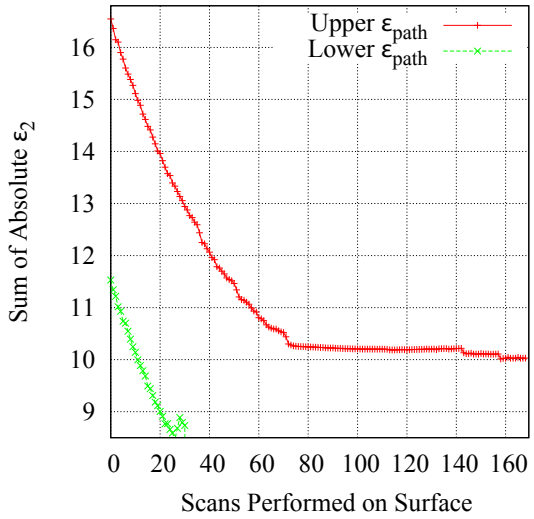


Figure 4.15: The sum of absolute size of minimum principal strains in the lower and upper $\varepsilon_{path}(m,n)$ with respect to the amount of laser scans placed on the respective surface.

4.3 Sub Conclusion

Two geometries were tested in order to proof the concept of the framework. The geometries being; a 10° V-bend and a cosinusoidal geometry. The framework formed the 10° V-bend to a sum of absolute error of 29.6 with a maximum deviation of |-0.08mm| in 100 iterations, thereby, proving the concept of the developed framework. The cosinusoidal geometry validated the frameworks capability of forming on both sides of a blank. The framework was not able to form the cosinusoidal geometry in 200 iterations of the framework. However, the obtained geometry exerts the appearance of the desired cosinusoidal geometry. It was chosen to use a combination of the sum of absolute error and the sum of absolute size of minimum principal strain as the sum of absolute error provided a misleading measure of the progress of the process.

Chapter 5

Test of the Framework on Double Curved Geometries

This chapter presents two tests performed on double curved geometries, using the framework developed in chapter 3. The purpose of this chapter is to perform a proof of concept of the framework, with regards to the production of double curved geometries. This chapter consists of; tests of the framework and discussions of results and selected observations.

Two desired geometries are tested throughout this chapter. The desired geometries are formed from a square blank measuring 30x30x1mm, as mentioned in section 2.1. The blank is fixed at four nodes on the bottom surface of the blank, as shown in figure 5.1. This is necessary to assure that the blank is kept in place during forming.

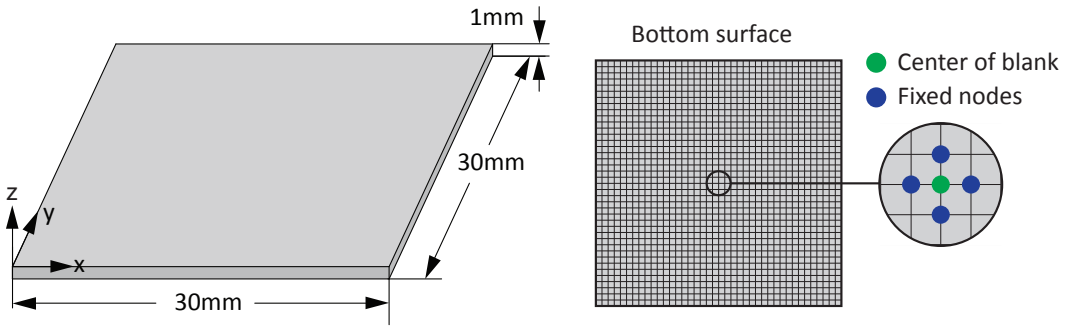


Figure 5.1: The blank used in the tests of the double curved geometries. The blank is fixed at four nodes on the bottom surface to ensure that the blank is kept in place during forming.

The desired geometries are shown in figure 5.2. The desired geometries are a dome geometry and a saddle geometry. Similar geometries are used in [Liu and Yao, 2005], why these are assessed suitable for preliminary testing. Both geometries are created as swept surfaces, generated from three point arcs. Nine coordinates describing the geometry are shown for each of the desired geometries. CAD-files of the desired geometries are appended on the appendix-CD. Both desired geometries are non-developable surfaces, why they require the utilisation of both the temperature

gradient mechanism and the upsetting mechanism.

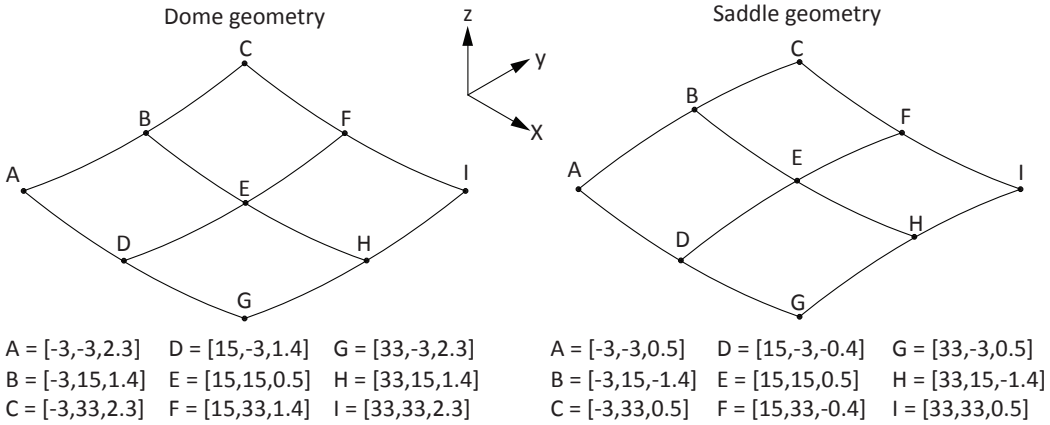


Figure 5.2: Tested geometries being; a dome geometry and a saddle geometry.

As discussed in section 3.3.1 the strain field utilised in the framework is estimated by use of LS-Dyna's OneStep solver. This approach is inapplicable for the double curved geometries, as the achieved orientation of the minimum principal strains, for the double curved geometries, depends on the orientation of the input geometry to the OneStep solver. Tests documenting this behaviour are appended in appendix K. As an alternative, the OneStep solver is substituted with a developed "flattening" Finite Element model. The flattening model is appended in appendix L. Figure 5.3 shows the principle of the flattening model for a V-bend geometry.

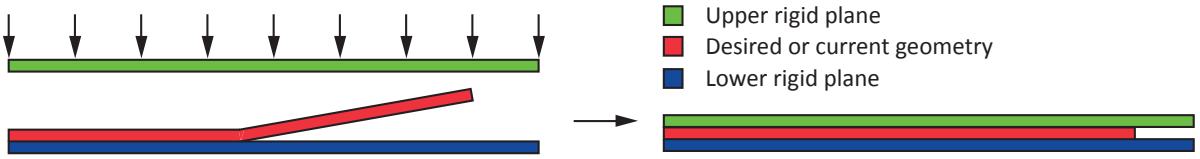


Figure 5.3: Principle of the flattening model.

In the flattening model, a shell representation of the desired or current geometry is placed between two rigid planes. The upper rigid plane is moved towards the lower rigid plane, which is fixed in space. During the movement, the V-bend deforms, thereby, inducing strains in the flattened geometry. The strain fields $\varepsilon_{top}(m,n)$ and $\varepsilon_{bot}(m,n)$ are extracted from the flattened geometry. In opposite to the OneStep solution of a V-bend, $\varepsilon_{top}(m,n)$ prescribes tensile strains in the top surface and $\varepsilon_{bot}(m,n)$ prescribes compressive strains in the bottom surface, due to the deformation of the blank. Therefore, it is necessary to convert $\varepsilon_{top}(m,n)$ and $\varepsilon_{bot}(m,n)$, such that they represent the required strain instead of the induced strain. The conversion is done by reversing the strain fields. This is done by multiplying all components in the strain tensor ε_{ij} with (-1.0). After the conversion $\varepsilon_{top}(m,n)$ and $\varepsilon_{bot}(m,n)$ are used as described in chapter 3.

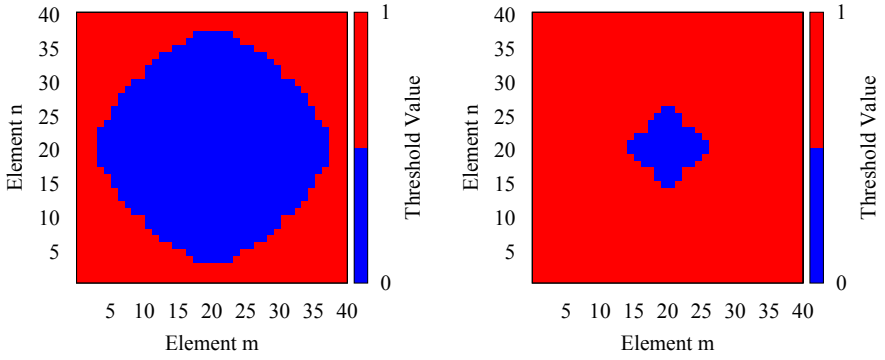
The tests of the double curved geometries are conducted with the settings of the framework

shown in table 5.1. The selected settings were determined by trial of the framework. Settings providing a higher rate of convergence or a better correlation between the current and desired geometry may be found. However, as the purpose is to proof the concept of the framework, the settings are accepted. Furthermore, the functionality of ε_{minlim} for the upsetting mechanism and the temperature gradient mechanism, discussed in section 3.5, is deactivated as this caused early stagnation of the convergence in both tests.

Parameter	**Dome geometry**	**Saddle geometry**
Threshold percentage	95%	95%
Allowable angular deviation α_{max}	50°	50°
Maximum number of iterations	60	200

Table 5.1: Settings used for the tests performed on the double curved geometries.

Comparing the settings in table 5.1 with the settings utilised in the tests of the single curved geometries, presented in chapter 4, shows that the threshold percentage is increased. This is done to increase the solution space for the scan path algorithm. This is exemplified in figure 5.4, which shows $\varepsilon_{thresh}(m,n)$ for the dome geometry with different values of the threshold percentage. If the threshold percentage is low, the solution space is reduced which results in short scan paths for the tests with the double curved geometries. The short scan paths resulted in stagnation of the process, as insignificant amounts of forming was introduced.



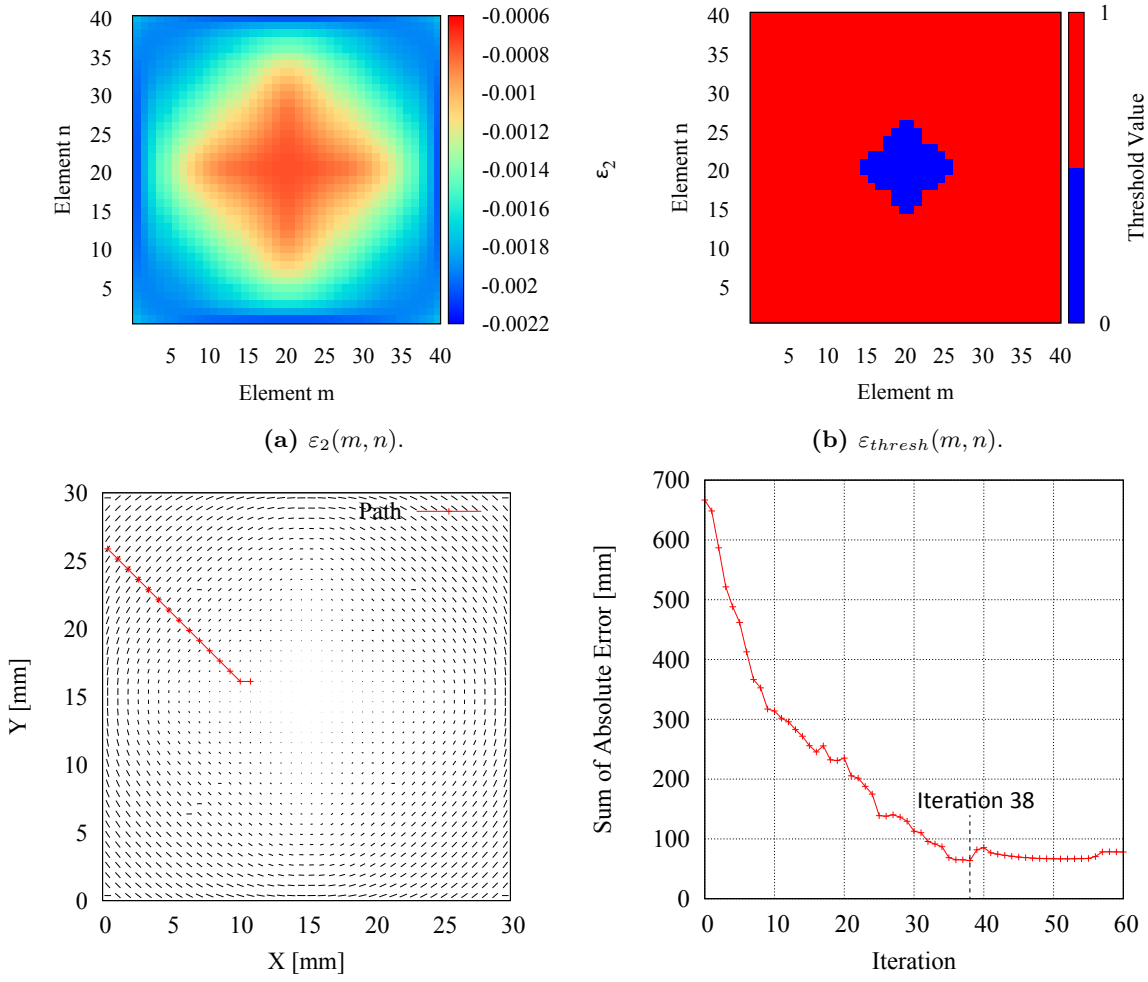
(a) $\varepsilon_{thresh}(m,n)$ for a threshold percentage of 50%.

(b) $\varepsilon_{thresh}(m,n)$ for a threshold percentage of 95%.

Figure 5.4: $\varepsilon_{thresh}(m,n)$ for different threshold percentages.

5.1 Test of the Dome Geometry

The framework receives the dome geometry shown in figure 5.2 and the settings shown in table 5.1. Test results are shown in figure 5.5. Figure 5.5a to 5.5c shows; the size of the minimum principal strain $\varepsilon_2(m,n)$ in $\varepsilon_{path}(m,n)$, the threshold $\varepsilon_{thresh}(m,n)$ of $\varepsilon_2(m,n)$ and the orientation of the minimum principal strains $\theta_2(m,n)$ and scan path, for the first iteration of the framework. Figure 5.5d shows the development of the sum of absolute error as a function of the iterations of the framework.



(a) $\varepsilon_2(m,n)$.

(b) $\varepsilon_{thresh}(m,n)$.

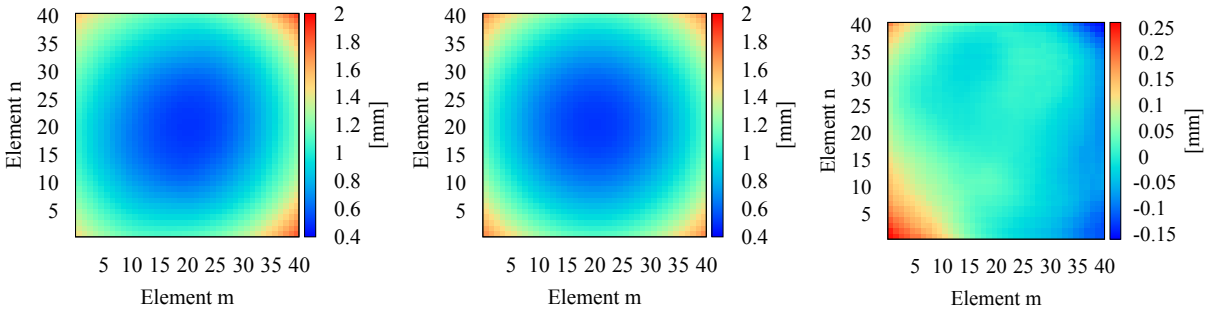
(c) Scan path and $\theta_2(m,n)$.

(d) Sum of absolute error.

Figure 5.5: Results from the test of the dome geometry, with a threshold percentage of 95%, a maximum amount of iterations equal to 60 and $\alpha_{max} = 50°$. Figure 5.5a to 5.5c show the results from the first iteration of the framework, while figure 5.5d shows the sum of absolute error for all 60 iterations.

Figure 5.5d, shows that the smallest sum of absolute error is reached at a value of 63.71 in iteration 38 of the framework. The sum of absolute error converges towards zero until iteration

38, where it stagnates. Figure 5.6a shows the current geometry in iteration 38 of the framework. By comparing the current geometry with the desired geometry, shown in figure 5.6b, the surface error, shown in figure 5.6c, is obtained. The obtained geometry exerts the appearance of the desired geometry with a maximum deviation of |0.25mm|, which is assessed to proof the concept of the developed framework. However, deviations from the desired geometry are present, as shown in figure 5.6c, why improvements of the framework are required.



(a) Current geometry.

(b) Desired geometry.

(c) Error between desired and current geometry.

Figure 5.6: Figure 5.6a shows the obtained geometry after 38 iterations of the framework. Figure 5.6b and 5.6c show the desired geometry and the error between the obtained and desired geometry respectively.

5.1.1 Observations from the Test of the Dome Geometry

Through this section, two observations are discussed with respect to the results presented in figure 5.5.

Observation 1 - Remaining Strain in the Circumference of the Blank

From figure 5.5d it is seen that the development of the sum of absolute error stagnates after 38 iterations of the framework. As discussed in section 4.2, the sum of absolute error can provide a misleading measure of the current geometry's progress towards the desired geometry. Therefore, the development in required strain is checked as well. Only the top surface of the blank is laser scanned in the 60 iterations of the framework. Figure 5.7 shows the sum of absolute size of minimum principal strain for the upper $\varepsilon_{path}(m, n)$. Similar behaviour with respect to stagnation is observed in the sum of absolute size of minimum principal strain, thereby substantiating that the process stagnates.

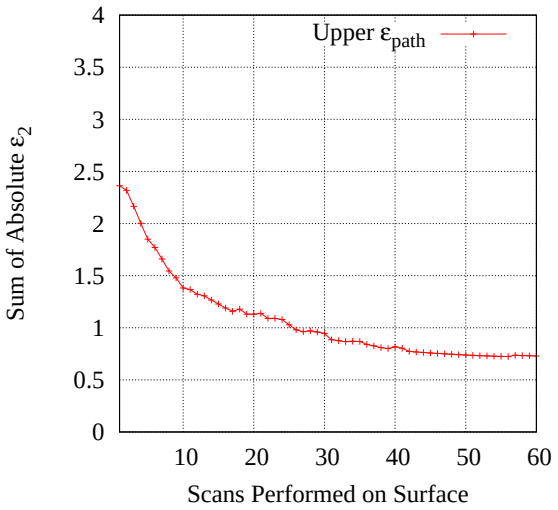
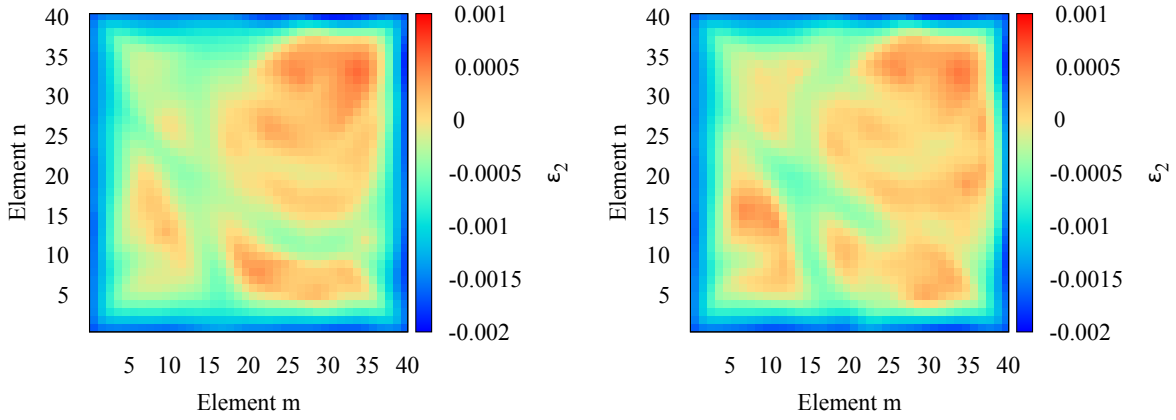


Figure 5.7: The sum of absolute size of minimum principal strains in the upper $\varepsilon_{path}(m, n)$.

As both the sum of absolute error and sum of absolute size of minimum principal strain stagnates, it is assessed that the laser forming process is unable to form the blank beyond iteration 38. To identify the cause of the stagnation, $\varepsilon_2(m, n)$ of iteration 38 and 60 is investigated. $\varepsilon_2(m, n)$ for the two iterations is shown in figure 5.8a and 5.8b.



(a) $\varepsilon_2(m, n)$ of iteration 38.

(b) $\varepsilon_2(m, n)$ of iteration 60.

Figure 5.8: $\varepsilon_2(m, n)$ for iteration 38 and 60 of the framework.

Figure 5.8, shows that only small variations in $\varepsilon_2(m, n)$ are present from iteration 38 to 60 of the framework. Furthermore, it is seen that the most compressive strains are required in the circumference of the blank. By investigation of the generated scan paths from iteration 38 to 60, it is seen that the scan paths are generally placed near the edge of the blank. This is exemplified in figure 5.9 for iteration 60 of the framework. It is assessed that the scan paths determined by the scan path algorithm are placed correctly. However, the laser forming process is not able to

remove the required compressive strains in the circumference of the blank.

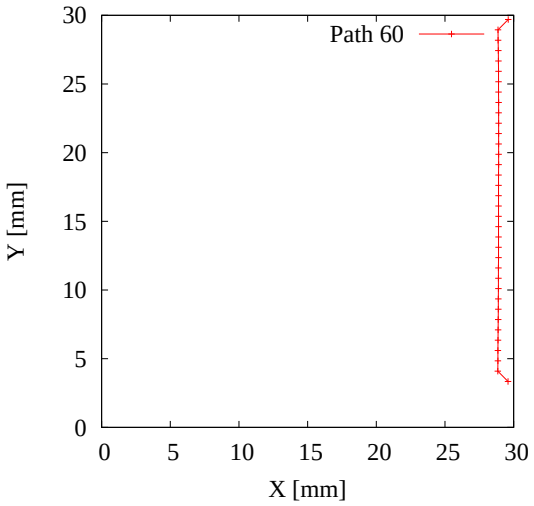


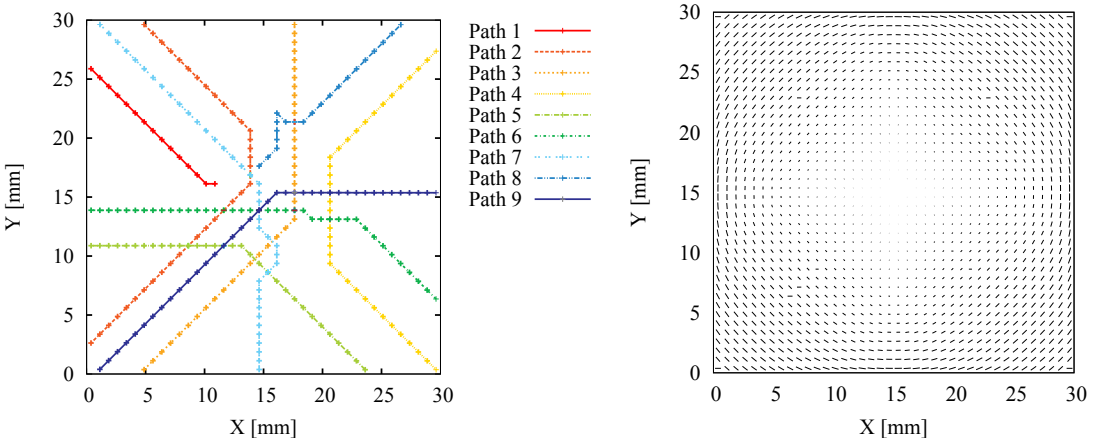
Figure 5.9: Scan path placement in iteration 60 of the framework.

The inability to remove the required compressive strains in the circumference of the blank is caused by limitations of the laser forming process. It is difficult to activate the forming mechanisms close to the edge of the blank, as the surrounding material provides insufficient mechanical resistance towards the thermal expansion. As a result no plastic deformation is obtained.

Observation 2 - Asymmetry in the Final Geometry

The current geometry in iteration 38 of the framework exerts the appearance of a dome geometry. However, the current geometry is asymmetrical, whereas the desired dome geometry is symmetrical around the center with respect to the XZ and YZ plane.

Figure 5.10a shows the placement of scan paths for the first nine iterations of the framework. Note that $\varepsilon_{thresh}(m, n)$, shown in figure 5.5b changes between iterations of the framework, why the scan paths from iteration 2 to 9, passing through the center, are related to a different representations of $\varepsilon_{thresh}(m, n)$. The generated scan paths are perpendicular to the orientation of the minimum principal strains shown in figure 5.10b. This corresponds to the intended placement of scan paths. However, as the laser scans are performed in separate iterations of the framework, asymmetric forming cannot be avoided.



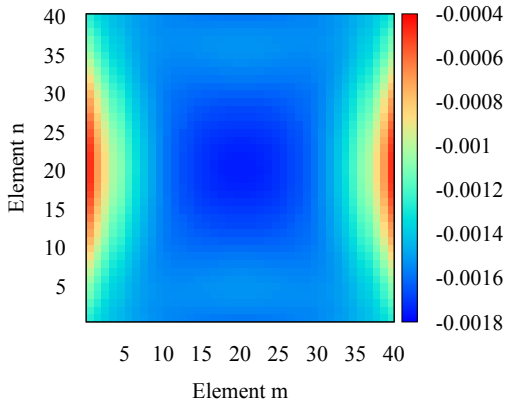
(a) Scan paths generated for iteration 1 to 9 of the framework.

(b) $\theta_2(m, n)$ from the first iteration of the framework.

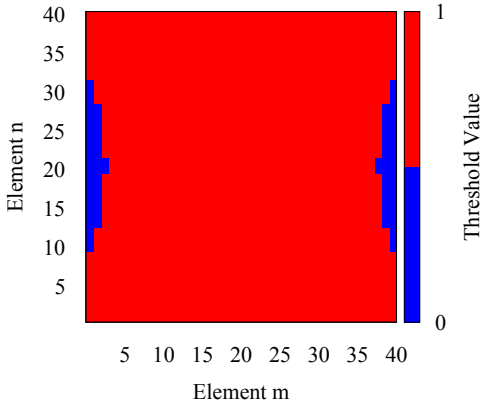
Figure 5.10: Figure 5.10a shows the scan paths generated for iteration 1 to 9 of the framework. The placement of scan paths correspond to the orientation of minimum principal strains, as shown in figure 5.10b.

5.2 Test of the Saddle Geometry

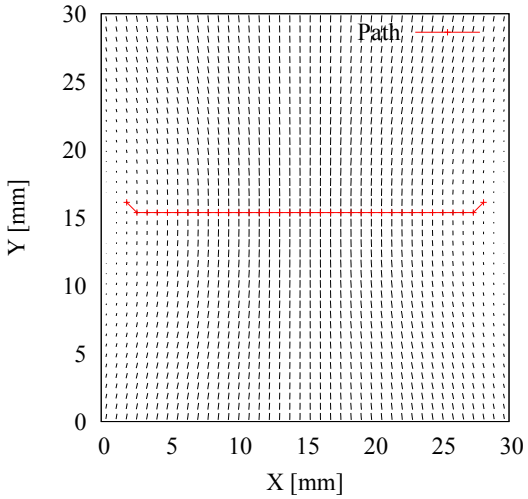
The framework receives the saddle geometry shown in figure 5.2 and the settings shown in table 5.1. Test results are shown in figure 5.11. Figure 5.11a to 5.11c shows; the size of the minimum principal strains $\varepsilon_2(m,n)$ in $\varepsilon_{path}(m,n)$, the threshold $\varepsilon_{thresh}(m,n)$ of $\varepsilon_2(m,n)$ and the orientation of the minimum principal strains $\theta_2(m,n)$ and scan path, for the first iteration of the framework. Figure 5.11d shows the development of the sum of absolute error as a function of the iterations of the framework. The saddle geometry requires laser scans on both the top and bottom surface. In the first iteration of the framework, the most compressive strain is required on the bottom surface. Therefore, figure 5.11a to 5.11c relates to the lower $\varepsilon_{path}(m,n)$.



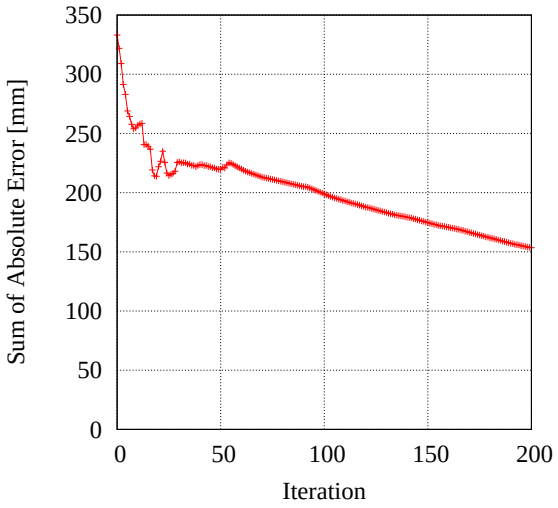
(a) $\varepsilon_2(m,n)$.



(b) $\varepsilon_{thresh}(m,n)$.



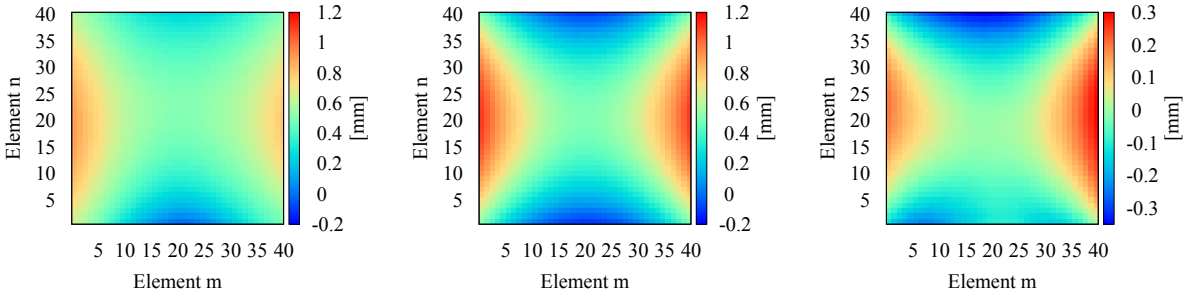
(c) Scan path and $\theta_2(m,n)$.



(d) Sum of absolute error.

Figure 5.11: Results from the test of the saddle geometry, with a threshold percentage of 95%, a maximum amount of iterations equal to 200 and $\alpha_{max} = 50°$. Figure 5.11a to 5.11c show the results from the first iteration of the framework, while figure 5.11d shows the sum of absolute error for all 200 iterations.

Figure 5.11d, shows that the smallest sum of absolute error is reached at a value of 153.43 in iteration 200 of the framework. The sum of absolute error converges towards zero, but fails to reach zero before reaching the maximum amount of iterations. Figure 5.12a shows the current geometry in iteration 200 of the framework. By comparing the current geometry with the desired geometry, shown in figure 5.12b, the surface error, shown in figure 5.12c, is obtained.



(a) Current geometry.

(b) Desired geometry.

(c) Error between desired and current geometry.

Figure 5.12: Figure 5.12a shows the obtained geometry after 200 iterations of the framework. Figure 5.12b and 5.12c shows the desired geometry and the error between the obtained and desired geometry respectively.

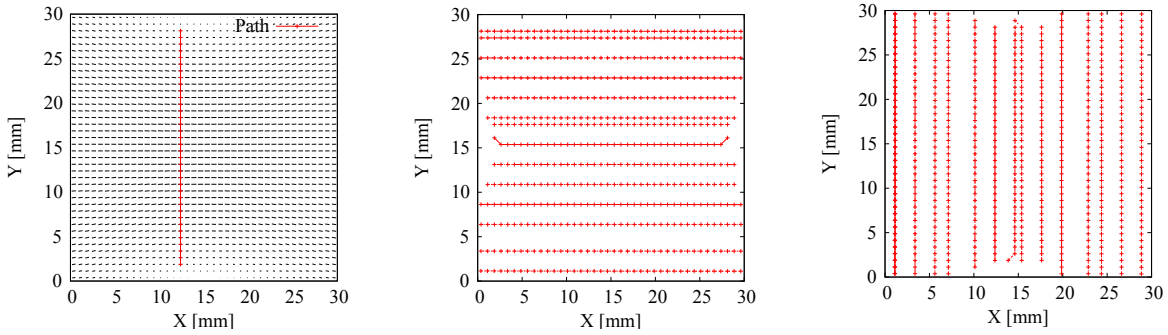
The obtained geometry exerts the appearance of the saddle geometry with a maximum deviation of $|0.3\text{mm}|$, which is assessed to proof the concept of the developed framework. However, deviations from the desired geometry are present, as shown in figure 5.12c, why improvements of the framework are required.

5.2.1 Observations from the Test of the Saddle Geometry

Through this section two observations are discussed with respect to the results presented in figure 5.11.

Observation 1 - The Scan Path Algorithm Suffers from Discretisation

The bottom surface is scanned in iteration 1, while the top surface is scanned in iteration 2 of the framework. Figure 5.13a shows the orientation of the minimum principal strains $\theta_2(m, n)$ and generated scan path for iteration 2 of the framework. Figure 5.11c and 5.13a, shows that straight scan paths are generated in the first two iterations of the framework. By investigation of the remaining scan paths it is observed that the scan path algorithm continuously generates straight scan paths for the first 62 iterations of the framework. Figure 5.13b and 5.13c show the scan paths generated during the first 30 iterations, for the bottom and top surface respectively.



(a) $\theta_2(m,n)$ and scan path for iteration 2 of the framework.

(b) Scan paths on the bottom surface.

(c) Scan paths on the top surface.

Figure 5.13: Figure 5.13a shows the scan path generated in iteration 2 of the framework. Figure 5.13b and 5.13c show the scan paths generated during the first 30 iterations, for the bottom and top surface.

The generated scan paths have been compared with $\theta_2(m,n)$ for the respective iterations. The straight scan paths do not comply with the intend of generating scan paths perpendicular to the orientation of the minimum principal strains. This is exemplified in figure 5.14, which shows the scan path generated for iteration 19 of the framework. The red line indicates the scan path generated by the scan path algorithm. The green line represents a scan path with the same start point, but satisfying the intend of placing scan paths perpendicular to the orientation of minimum principal strains.

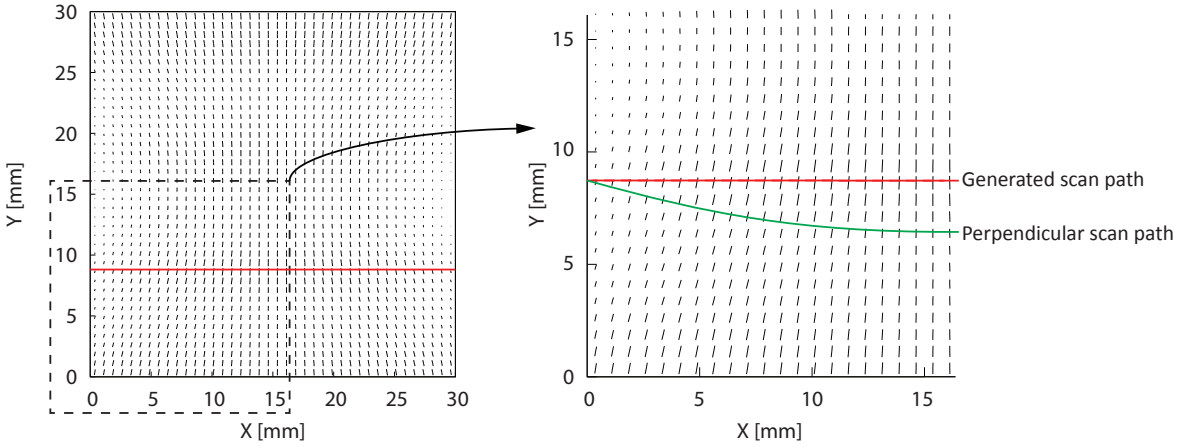


Figure 5.14: The generated scan path from the scan path algorithm and the desired scan path, which is perpendicular to the orientation of the minimum principal strains, for iteration 19 of the framework.

The problem is caused by the search stencil, utilised in the scan path algorithm, described in section 3.4.1. The problem is exemplified in figure 5.15. The search stencil dictates discrete movement options in approximately 45° intervals. The movement options are dictated by the vectors between the start point and the element centroids of the neighbouring elements. In the example $\alpha_1 < \alpha_2$, why a scan path moving upwards is generated. The movements orientated

upwards contradict the intended scan path, which is supposed to move perpendicular to the orientation of minimum principal strains. However, this movement is not possible due to the discrete movement options of the search stencil.

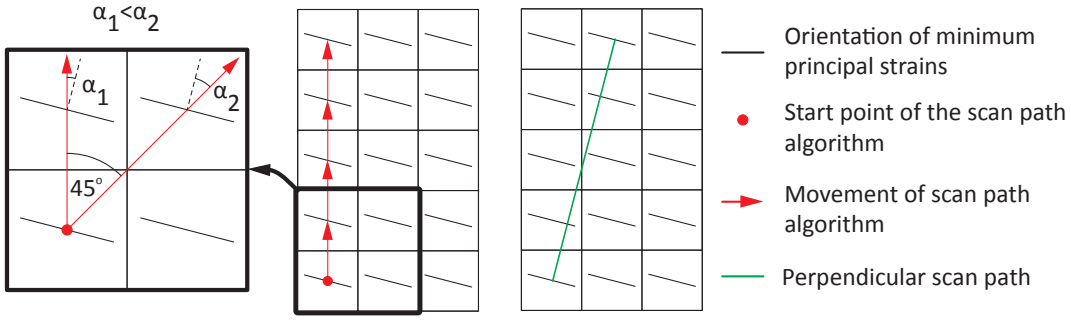


Figure 5.15: The scan path generated by the scan path algorithm and the desired scan path with respect to perpendicularity to the orientation of minimum principal strains.

Observation 2 - Short Scan Paths Generated by the Scan Path Algorithm

Straight laser scans, across the width of the blank, are generated for the first 62 iterations of the framework. In the remaining iterations shorter scan paths are generated. The scan paths are placed on the bottom surface near the edge closest to the X axis, as exemplified for iteration 110 and 120 in figure 5.16. The short scan paths are a result of an early violation of the criteria with respect to threshold, perpendicularity and point repetition, as described in section 3.4.1. An undesirable selection of the initial start point, for the scan path algorithm, can result in early violation of the three criteria. The potential amount of forming introduced in a short scan path is generally smaller than the potential amount of forming in a longer scan path. As a result the convergence rate may be reduced when generating short scan paths.

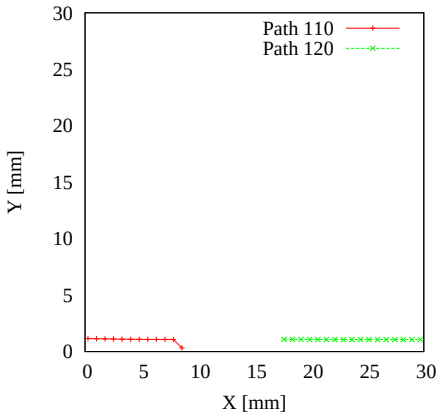


Figure 5.16: Scan paths generated in iteration 110 and 120 of the framework.

5.3 Sub Conclusion

Two geometries were tested in order to proof the concept of the framework. The geometries being; a dome geometry and a saddle geometry. The framework formed the dome geometry to a sum of absolute error equal to 63.71 with a maximum deviation of |0.25mm| in 38 iterations of the framework. The saddle geometry was formed to a sum of absolute error equal to 153.43 with a maximum deviation of |0.30mm| in 200 iterations. The tested geometries exert the appearance of the dome and saddle, thereby proving the concept of the developed framework. The two tests assisted in identifying four areas in need of improvement in the current framework:

- The framework is unable to reduce the required strain in the circumference of the blank, due to limitations in the laser forming process
- The formed dome geometry is asymmetric, as a result of the placement of scan paths in each iteration of the framework
- The scan path algorithm is unable to generate scan paths perpendicular to the orientation of minimum principal strains, as a result of the discrete search stencil
- The scan path algorithm generates short scan paths, as a result of an undesirable selection of the initial start point.

Chapter 6

Conclusion

In this project a framework was developed, which enables the development of double curved geometries by use of the laser forming process. The framework was developed with respect to the problem statement:

"How can a framework capable of producing double curved geometries by utilisation of the laser forming process be developed?"

To develop the framework, it was necessary to establish an understanding of the laser forming process. In the laser forming process a defocused laser beam irradiates a blank in predefined scan paths. The absorbed laser energy causes heat generation in the material, which results in expansion of the blank. The expansion is resisted by the surrounding material thereby inducing compressive stresses. If the stresses exceed the temperature dependent yield stress, plastic deformation is introduced. To produce double curved geometries it is necessary to utilise the temperature gradient mechanism and the upsetting mechanism. The mechanisms are important, as they enable the laser forming process to create developable and non-developable surfaces.

To develop and test the framework it was chosen to utilise a Finite Element model of the laser forming process. The Finite Element model developed through the 9th. semester project was adjusted to cope with both the upsetting mechanism and the temperature gradient mechanism, as these are required when forming double curved geometries. The Finite Element model successfully demonstrated correct physical behaviour with respect to the temperature distribution, strain distribution and forming obtained by both mechanisms. However, experimental validation of the Finite Element model is necessary to ensure correlation with the physical laser forming process.

The framework was created as a feedback loop, to cope with the poor repeatability of the laser forming process. The framework determines a scan path and suited process variables in each iteration, based on the required strain to go from the current to the desired geometry. The scan path is generated with a developed scan path algorithm. To reduce the risk of over forming the blank, scan paths are placed in areas requiring the largest amount of forming first. The scan

path is placed perpendicular to the orientation of minimum principal strains, in a scan path strain field, as the largest compressive strains induced by the laser forming process occur perpendicular to the scan path. The scan path strain field is created as the average of the required midplane strain field and the surface strain field requiring the largest compressive strain, as these strain fields indicate the in-plane and bending strain, required to form a desired geometry. To ensure that the current geometry converges towards the desired geometry, suited process variables are determined for all elements in the scan path, with respect to the required bending and in-plane strains. The scan speed was selected as the control variable, as it is possible to control the amount of forming induced in the blank by adjustment of the scan speed. The scan speed is varied along the scan path as a function of the required strains, in order to reduce the variance in the process output. A stop criterion was formulated to stop the framework. The stop criterion is formulated with respect to the sum of absolute error, as this provides a scalar measure of the correlation between the current and the desired geometry.

The framework was tested with regard to four tests, which are divided into tests of single curved geometries and tests of double curved geometries. The tests of the single curved geometries considered a 10° V-bend geometry and a cosinusoidal geometry. The 10° V-bend geometry was used for preliminary testing and debugging purposes, which ensured a functioning framework. The cosinusoidal geometry was tested, to ensure the frameworks ability to perform laser scans on both sides of the blank. The tests of the double curved geometries considered a dome geometry and a saddle geometry. Both tests were used for preliminary tests of the double curved geometries to proof the concept of the framework.

The framework formed the 10° V-bend to a sum of absolute error of 29.6 with a maximum deviation of |-0.08mm| in 100 iterations, thereby, proving the concept of the developed framework. The cosinusoidal geometry validated the frameworks capability of forming on both sides of a blank. The framework was not able to form the cosinusoidal geometry in 200 iterations of the framework. However, the obtained geometry exerts the appearance of the desired cosinusoidal geometry. It was chosen to use a combination of the sum of absolute error and the sum of absolute size of minimum principal strains, as the sum of absolute error provided a misleading measure of the progress of the process.

The framework formed the dome geometry to a sum of absolute error equal to 63.71 with a maximum deviation of |0.25mm| in 38 iterations of the framework. The saddle geometry was formed to a sum of absolute error equal to 153.43 with a maximum deviation of |0.30mm| in 200 iterations. The tested geometries exert the appearance of the dome and saddle, thereby proving the concept of the developed framework. The two tests assisted in identifying four areas in need of improvement in the current framework:

- The framework is unable to reduce the required strain in the circumference of the blank, due to limitations in the laser forming process
- The formed dome geometry is asymmetric, as a result of the placement of scan paths in each iteration of the framework

- The scan path algorithm is unable to generate scan paths perpendicular to the orientation of minimum principal strains, as a result of the discrete search stencil
- The scan path algorithm generates short scan paths, as a result of an undesirable selection of the initial start point.

In summation, the developed framework enables the forming of double curved geometries by utilisation of the laser forming process. Thus, it is concluded that a framework based on a feedback control loop, utilising strain information for the determination of scan paths and process variables enables the forming of double curved geometries. However, several areas requiring improvement were identified. These are discussed in chapter 7.

Chapter 7

Future Work

In this chapter, the industrial potential of a framework for the laser forming process is discussed. Furthermore, the future work required to improve the developed framework is discussed.

As discussed in chapter 1, the laser forming process is a potential substitute to conventional sheet forming processes. Compared to conventional processes, laser forming introduces a higher degree of flexibility, as the framework enables the production of new geometries by changing the desired geometry, whereas conventional sheet forming processes require costly development and manufacturing of new dies. The increase in flexibility introduces great potential within the area of rapid prototyping, customised products and small-batch production. Furthermore, the laser forming process may be combined with other processes performed with a laser source. Such that a single work cell has the capability of performing e.g. laser welding, laser cutting and laser forming. A successful implementation of the laser forming process with regards to industrial application requires investigation of several subjects e.g. process design, profitability, achievable geometries, achievable tolerances, achievable process lead times, effect on material properties etc.

This project focused on the area of process design. The subject of process design was handled with the developed framework, capable of determining scan paths and process variables, with respect to a desired geometry.

7.1 Improvements of the Developed Framework

To ensure the full potential of the developed framework, several tasks must be treated. The tasks are discussed in the following.

Performance Requirements for the Framework

In this project four tests were conducted with the developed framework. The purpose of the tests was to proof the concept of the developed framework and assist in identifying areas of the framework which required further work. When these improvements have been treated, it is necessary to perform new tests of the framework. Prior to these tests it is necessary to establish

a set of performance requirements for the framework. The performance requirements must relate to competing forming processes with respect to e.g. process lead time, achievable geometries and achievable tolerances. The performance requirements must be satisfied to ensure the industrial applicability of the framework.

Evaluation of the Finite Element Model of the Laser Forming Process

The framework was developed and evaluated using a Finite Element model. The utilised Finite Element model was fitted to experimental data, with respect to the temperature gradient mechanism, in the 9th. semester project. However, the utilisation of the upsetting mechanism in the present project, required adjustments of the Finite Element model. To ensure that the output of the Finite Element model and the physical laser forming process correlates, it is necessary to fit the simulation model with respect to both the temperature gradient mechanism and upsetting mechanism. When the Finite Element model has been fitted, it is necessary to update the parameters used throughout the framework.

Development of an Experimental Setup for Testing of the Framework

To ensure that the results achieved using the developed framework correlates when implemented in a physical setup, it is necessary to develop an experimental setup, that enables testing of the framework. The experimental setup must enable forming of single and double curved geometries. This requires considerations with respect to:

- **Measurement of the blank:** The framework requires that the surface of the current geometry is measured in each iteration. The measurement must provide a satisfactory representation of the current geometry that can be handled in the framework software e.g. a point cloud representation.
- **Irradiation of the blank:** The framework requires that the entire surface of the blank can be irradiated with respect to the determined scan path and control variables. The movement of the laser beam must comply with the scan speed specified by the process ranges defined for the temperature gradient mechanism and upsetting mechanism.
- **Fixation of the blank:** The laser forming process requires fixation of the blank. For single curved geometries a fixation at one edge is sufficient. The forming of double curved geometries require an alternative fixation method. The alternative fixation method must not introduce any additional structural stiffness to the blank, as this will influence to output of the process.

Improvement of the Strain Analysis

The strain analysis used in the framework relied on the use of LS-Dyna's OneStep solver for the single curved geometries and a developed flattening model for the double curved geometries. As described in appendix K, the OneStep solver resulted in a misleading strain field for the double curved geometries, why a flattening model was developed. However, the flattening model becomes unstable for heavily formed geometries. A new approach must be developed for obtaining a correct strain field. The new approach should allow for fast determination of the required

strain field, as it is used in each iteration of the framework. Furthermore, it must be applicable for all geometries, achievable by the laser forming process.

Improvements of the Scan Path Algorithm

The scan path algorithm must be revised to ensure that scan paths are perpendicular to the orientation of the minimum principal strains. In the current framework the algorithm suffers from the discretisation of the search stencil. Therefore, the improvement must ensure a path planning algorithm, independent of the limitation set by the discretisation.

Furthermore, it is assumed that the convergence rate can be improved by revising the scan path algorithm. The current strategy is to start the scan path algorithm from the element requiring the largest amount of strain and accepting the generated scan path. It is suggested to investigate several different initial start points prior to execution of the laser forming process. Each of the initial start points results in a new scan path. The scan path and corresponding process variables providing the largest amount of forming can then be selected for the current iteration of the framework. Another argument for considering different start points, is that the most compressive strain in a single element may be a result of numerical errors in the Finite Element model, thus leading to a wrong selection of the start point.

The current framework produces one consecutive scan path per iteration. An alternative to this approach is to generate several scan paths for each iteration. By performing several laser scans, a larger amount of the required strain is minimised in each iteration. However, several considerations are necessary to address before implementing a solution, which performs several laser scans, such as; how to define new start points for the scan path algorithm, how close scan paths can be placed to each other without influencing the process output achieved in each laser scan and how many laser scans can be performed per iteration while maintaining a robust process.

Improvement of the Stop Criterion

The formulated stop criterion evaluated the geometrical correlation of the obtained geometry with respect to the desired geometry. The formulated stop criterion calculates the sum of absolute error between the current and the desired geometry. However, the sum of absolute error can provide a misleading measure of the current geometry's progress towards the desired geometry. It was chosen to investigate the required strain to be achieved in the blank, as each successive laser scan decreases the required strain. A converging development was observed, why it is assessed that a stop criterion must be formulated with regards to both the geometrical correlation and the required strain correlation.

Increasing the Size of the Blank

The laser forming process is unable to successfully reduce the required compressive strain in the circumference of the blank, due to limitations of the process. A possible solution is to use a larger

blank, to ensure that the required compressive stresses, at the circumference of the blank, can be induced into the material. If this strategy is utilised it is necessary to consider the springback which occurs, when the desired geometry is cut from the larger initial blank. The springback is a consequence of the residual stresses in the blank. It is assessed that the springback can be approximated using the Finite Element model of the laser forming process, as the results from the Finite Element model contains the required stress and strain information. Furthermore it must be investigated, how excess material is removed. The influence of the process used to remove the excess material must be investigated e.g. if laser cutting is to be used, the thermal effect of cutting must be investigated.

Historical Error Development

The laser forming process suffers from poor repeatability, as discussed in section 3.1.2. To increase the repeatability of the laser forming process, the scan speed is controlled along the scan path. The scan speed is determined with respect to the selected forming mechanism and the required amount of either bending or in-plane strain. The scan speed is adjusted along the scan path with regards to a fraction distribution. However, the output of the laser forming process varies as more scans are performed, as a consequence of e.g. induced residual stresses and strain hardening. Therefore, it is assessed that the historical error development of the process must be utilised to select suitable process variables. If historical error development is considered, the development in the introduced forming over several scans can be taken into account when selecting process variables. E.g. If the forming is slower than expected, then the scan speed is lowered, such that more forming is introduced.

Bibliography

Abed, E., Edwardson, S. P., Dearden, G., and Watkins, K. G. (2005). Closed loop 3-dimensional laser forming of developable surfaces. *International Workshop on Thermal Forming, Bremen.*

Dahotre, N. B. and Harimkar, S. P. (2008). *Laser Fabrication and Machining of Materials.* Springer Science + Business Media, 1. ed. edition.

Dearden, G. and Edwardson, S. P. (2003). Laser assisted forming for ship building. *Shipyard application of industrial lasers (SAIL) Williamsburg VA. June 2-4* .

Dearden, G., Taylor, C., Bartkowiak, K., Edwardson, S. P., and Watkins, K. (2003). An experimental study of laser micro-forming using a pulsed nd:yag laser and scanning optics (m409). *22nd International Congress on Applications of Lasers and Electro-Optics (ICALEO 2003).*

Deng, D. and Murakawa, H. (2006). Numerical simulation of temperature field and residual stress in multi-pass welds in stainless steel pipe and comparison with experimental measurements. *Computational Materials Science*, 37, 269-277.

Dowden, J. (2009). *The Theory of Laser Materials Processing.* Springer Science + Business Media, 1. ed. edition.

Edwardson, S. P., Griffiths, J., Edwards, K. R., Dearden, G., and Watkins, K. G. (2010). Laser forming: overview of the controlling factors in the temperature gradient mechanism. *Journal of Mechanical Engineering Science*, 224, 1031-1040.

Gere, J. M. and Goodno, B. J. (2009). *Mechanics of Materials.* CENGAGE Learning, 7 ed. edition.

Kalpakjian, S. and Schmid, S. (2006). *Manufacturing Engineering and Technology.* PEARSON Prentice Hall, 5 ed. edition.

Kennedy, J. M. (2013). Simulation of gaussian distributed heat flux. *Suggestion from LS-Dyna Support Resources- Yahoo Tech Group.*

Kumar, A. (2008). From mass customization to mass personalization: a strategic transformation. *Int J Flex Manuf Syst*, 19, 533-547.

Liu, C. and Yao, Y. (2005). Fem-based process design for laser forming of doubly curved shapes. *Journal of Manufacturing Processes*, 7, 109-121.

LSTC, L. S. T. C. (2007). *LS-DYNA KEYWORD USER'S MANUAL*. Livermore Software Technology Corporation., 1 ed. edition.

LSTC, L. S. T. C. (2013). *LS-DYNA KEYWORD USER'S MANUAL*. Livermore Software Technology Corporation., 1 ed. edition.

Lund, E. and Lindgaard, E. (2012). Course on finite element methods 2012. *Lecture at Aalborg University*.

Madsen, K. and Søndergaard, M. (2013). *Control System Development- for the Laser Forming Process*. Aalborg University, 1 ed. edition.

Mikron Instrument Company Inc. (2013). Table of emissivity of various surfaces. `http://www.czlazio.com/tecnica/Tabella%20delle%20Emissivit%C3%A0.pdf` [Accessed 04 December 2013].

Roll, K., Kleeh, T., and Merklein, M. (2011). Modeling laser heating for roller hemming applications. *Key Engineering Materials*, 473, 501-508.

Samuel, K. G., Mannan, S. L., and Radhakrishnan, V. M. (1992). The influence of temperature and prior cold work on the strain-hardening parameters of a type 316 ln stainless steel. *Int. J. Pres. Ves. and Piping*, 52, 151-157.

Shapiro, A. and Lo, D. (2009). Heat transfer and coupled thermal stress problems with ls-dyna. *Lecture at DYNAMORE*.

Shen, H., Hu, J., and Yao, Z. (2009). Analysis and control of edge effects in laser bending. *Optics and Lasers in Engineering*, 48, 305-315.

Shi, Y., Yao, Z., Shen, H., and Hu, J. (2005). Research on the mechanisms of laser forming for the metal plate. *International Journal of Machine Tools and Manufacture*, 46, 1689-1697.

Support, L.-D. (2013). Ls-dynas user guide - elements. `http://www.dynasupport.com/tutorial/ls-dyna-users-guide/elements` [Accessed 09 May 2013].

Thomson, G. and Pridham, M. (1997). A feedback control system for laser forming. *Mechatronics Vol. 7*, 5, 429-441.

Thomson, G. and Pridham, M. (1998). Improvements to laser forming through process control refinements. *Optics & Laster Technology*, 30, 141-146.

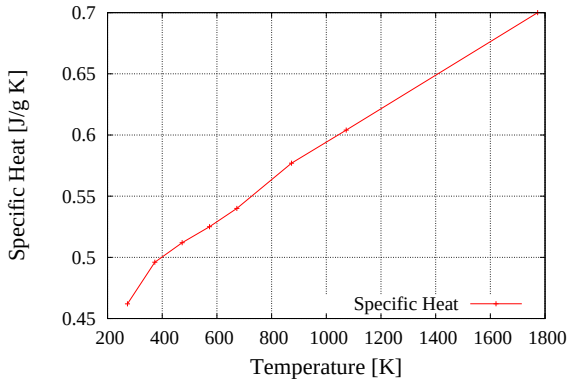
Ventsel, E. and Krauthammer, T. (2001). *Thin Plates and Shells - Theory, Analysis and Application*. Marcel Dekker, 1 ed. edition.

Zhang, L., Reutzel, E., and Michaleris, P. (2004). Finite element modeling discretization requirements for the laser forming process. *International Journal of Mechanical Sciences*, 46, 623-637.

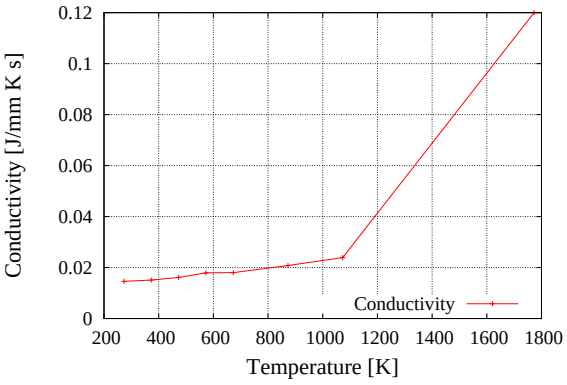
Appendix A

Data for the Simulation Models

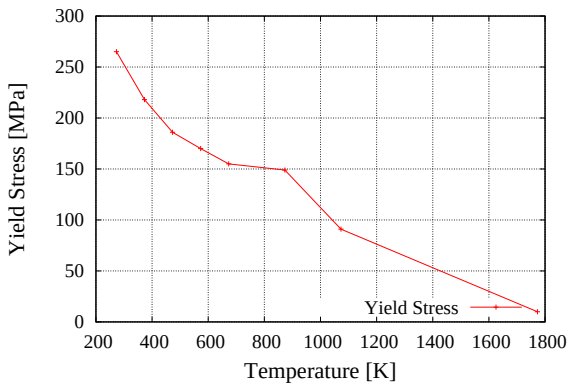
This appendix lists the temperature dependent material data used in both the previous and current simulation model [Deng and Murakawa, 2006] and [Samuel et al., 1992].



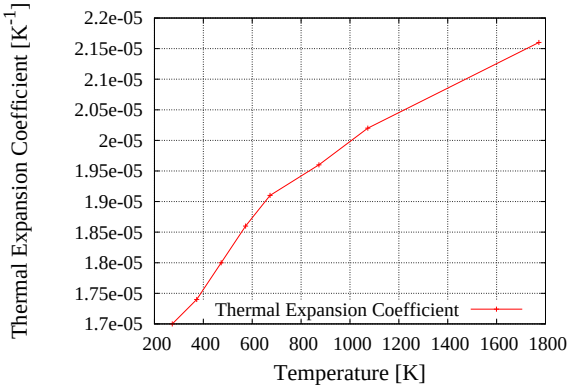
(a) Specific heat.



(b) Conductivity.

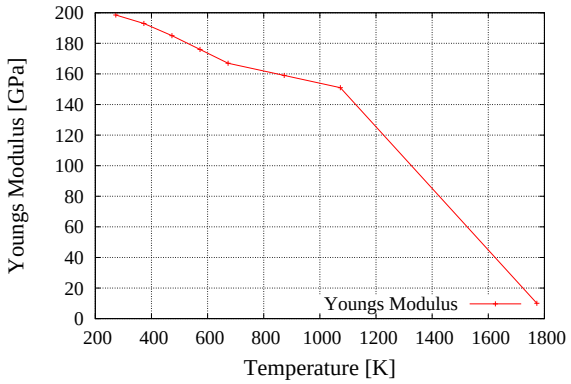


(c) Yield stress.

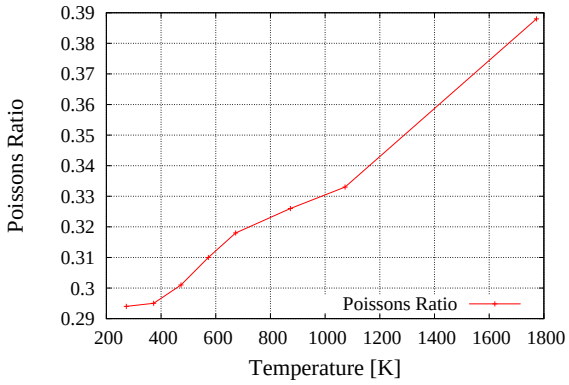


(d) Thermal expansion coefficient.

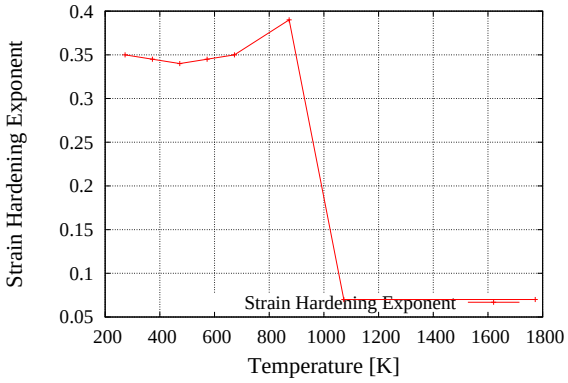
Figure A.1: Temperature dependent material properties (figure 1 of 2).



(a) Young's modulus.



(b) Poisson's ratio.



(c) Strain hardening.

Figure A.2: Temperature dependent material properties (figure 2 of 2).

Temperature [K]	273	373	473	573	673	873	1073	1773
Specific Heat $\left[\frac{\mathrm{J}}{\mathrm{g \cdot K}}\right]$	0.462	0.496	0.512	0.525	0.540	0.577	0.604	0.700
Conductivity $\left[\frac{\mathrm{J}}{\mathrm{mm \cdot K \cdot s}}\right]$	0.0146	0.0151	0.0161	0.0179	0.0180	0.0208	0.0239	0.120
Yield Stress [MPa]	265.00	218.00	186.00	170.00	155.00	149.00	91.00	10.00
Thermal Expansion $[\mathrm{K}^{-1}]$ Coefficient	1.70e-5	1.74e-5	1.80e-5	1.86e-5	1.91e-5	1.96e-5	2.02e-5	2.16e-5
Young's Modulus [GPa]	198.50	193.00	185.00	176.00	167.00	159.00	151.00	10.00
Poisson's Ratio	0.294	0.295	0.301	0.310	0.318	0.326	0.333	0.388
Strain Hardening Exponent	0.350	0.345	0.340	0.345	0.350	0.390	0.070	0.070

Table A.1: Temperature dependent material properties.

Appendix B

Adjustments for the Finite Element Model of the Laser Forming Process

This appendix describes the adjustments made to the Finite Element model of the laser forming process. The Finite Element model was developed during the 9th. semester project, with the purpose of simulating the temperature gradient mechanism. The adjustments are performed to ensure that the Finite Element model is suited for the purpose of this project. It is chosen to adjust the Finite Element model with regards to: Reduction of the discretisation of the blank and implementation of the upsetting mechanism.

Reducing the Spatial Discretisation of the Blank

As mentioned in section 1.3, the Finite Element model is used to develop and test the framework. During the development and tests of the framework, it is required to perform several simulations of the Finite Element model, why a low computation time is desired. It is chosen to reduce the spatial discretisation of the blank used in the Finite Element model, as this results in a decreased computation time. The discretisation is reduced to 40x40x4 elements. This complies with the requirements from [Zhang et al., 2004] concerning spatial discretisation i.e. a minimum of 2 elements per beam radius and a minimum of 3 elements in the thickness direction.

Determination of Process Range for the Upsetting Mechanism

In section 2.2, it is determined that the Finite Element model must be able to represent the physical behaviour of both the temperature gradient mechanism and the upsetting mechanism, as these are required when forming double curved geometries. The Finite Element model developed during the 9th. semester project, already provides an acceptable correlation with regards to simulating the temperature gradient mechanism, why only the implementation of the upsetting mechanism is considered in this appendix. To simulate the upsetting mechanism, it is necessary to adjust the process variables used for the simulation of the temperature gradient mechanism. The upsetting mechanism requires that the laser beam diameter is increased and the scan speed is decreased. This facilitates a uniform temperature distribution through the thickness of the

blank, thereby activating the upsetting mechanism.

It is desired to determine a process range for the upsetting mechanism with regards to scan speed, such that the amount of in-plane strain can be controlled. The process range for the upsetting mechanism is defined as the range which ensures an acceptable behaviour i.e. produces a significant amount of in-plane strain whilst not performing a significant bend, which indicates an undesirable influence of the temperature gradient mechanism. To determine a process range, that ensures an acceptable behaviour, when utilising the upsetting mechanism, a series of experiments were conducted using the Finite Element model. In the experiments, the laser beam diameter was varied from 6mm to 10mm in 2mm increments. For each laser beam diameter, different scan speeds were tested to identify the upper and lower limit of the process range. The laser power was maintained at 380W to reduce the amount of experiments. The laser beam diameter is chosen to be 6 mm and the process range, which ensures an acceptable behaviour is given as $200\frac{\text{mm}}{\text{min}}$ to $400\frac{\text{mm}}{\text{min}}$. Results of the experiments are appended on the appendix-CD.

Appendix C

Keydeck for the Finite Element Model of the Laser Forming Process

This appendix documents the keydeck constituting the Finite Element model of the laser forming process. The keydeck consists of several keywords. All information with regards to the keywords are provided by [LSTC, 2013] and [LSTC, 2007]. All keywords are shown throughout this appendix and the overall functionality is described. To ease the reading, default values have been removed. The keydeck and included files are appended on the appendix-CD.

The initial part of the keydeck is shown in listing L.1. The initial part designates the authors of the present keydeck and displays the unit scheme chosen for the Finite Element model along with a part summary.

```
1  *KEYWORD
2  *TITLE
3  Coupled thermal-mechanical simulation of the laser forming process in LS-Dyna
4  $ By Group VT4-2.215
5  $ - Kasper Madsen
6  $ - Martin Soendergaard
7  $                                    |        |
8  $                                    |________|
9  $                                     \      /    /
10 $                                      \____/    /
11 $                                        <|>    /
12 $                                        <|>   /
13 $                                        <|>  /
14 $                                        <|> /
15 $                                        <|>/
16 $                  ____________________|/
17 $
18 $The unit scheme used is given below:
19 $ Unit System Scheme (C): mm, ms, kg ==> kN, GPa ,Joule, kW
20 $ Part summary:
21 $      pid      name    nid-start   elid-start
22 $        1      blank   1            1
```

Listing C.1: The initial part of the keydeck.

C.1 Include

Listing C.2 includes the necessary files for the keydeck, extraction of these makes the keydeck easier to read and allows alterations to be performed to the files between simulations. The blank.k file is only included in the initial simulation, after that a dynain file is used. The dynain file represents the deformed state of the blank after each scan i.e. geometry of the blank, stresses and plastic strains. The SEGMENT file contains all segments used in the Finite Element model e.g. surface segments of the blank, which are used to apply convection and radiation to the model. The PARAMETRES file is used to alter the scan speed of the Finite Element model, why this is updated between scans

```
1 *INCLUDE
2 blank.k
3 *INCLUDE
4 SEGMENTS
5 *INCLUDE
6 PARAMETRES
7 *INCLUDE
8 DEFINECURVES
```

Listing C.2: Including files.

C.2 Dynain

Listing C.3 consists of the INTERFACE_SPRINGBACK_LSDYNA and the SET_PART_LIST keyword which creates a dynain file. As mentioned in section C.1, the dynain file contains the geometry of the blank, stresses and plastic strains.

```
1 *INTERFACE_SPRINGBACK_LSDYNA
2 $     psid
3           1
4 $
5 *SET_PART_LIST
6 $     psid
7           1
8 $      pid
9           1
```

Listing C.3: Interface keyword which creates a dynain file.

C.3 Control Keywords

Listing C.4 is used to ensure that no cpu limit is set.

```
1 *CONTROL_CPU
2 $#  cputim
3      0.000
```

Listing C.4: Control CPU.

Listing C.5 ensures that hourglass control is implemented, thereby preventing zero-energy modes.

```
1 *CONTROL_HOURGLASS
2 $#     ihq        qh
3         6  0.100000
```

Listing C.5: Control hourglass.

```
1 *HOURGLASS
2 $#    hgid       ihq        qm       ibq        q1        q2    qb/vdc        qw
3         1         6
```

Listing C.6: Hourglass.

Listing C.7 ensures that the Finite Element model is calculated as a coupled thermo-mechanical analysis (soln=2).

```
1 *CONTROL_SOLUTION
2 $#    soln       nlq     isnan     lcint
3         2
```

Listing C.7: Control solution.

Listing C.8 sets the termination time for the Finite Element model, which in the present simulation depends on the time of the laser scan and a fixed cooling time.

```
1 *CONTROL_TERMINATION
2 $#  endtim    endcyc     dtmin    endeng    endmas
3   &dt2+1.6
```

Listing C.8: Control termination.

Listing C.9 controls the structural timestep of the analysis, in this present Finite Element model LS-Dyna calculates the timestep.

```
1 *CONTROL_TIMESTEP
2 $#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st
3             0.900000
```

Listing C.9: Control timestep.

Listing C.10 is used to set parameters for the nonlinear coupled thermo-mechanical analysis.

```
1 *CONTROL_THERMAL_NONLINEAR
2 $#  refmax       tol       dcp    lumpbc    thlstl    nlthpr    phchpn
3                1E-4  0.500000
```

Listing C.10: Control thermal nonlinear.

Listing C.11 defines the Finite Element model as a transient problem (atype=1) and a nonlinear problem with material properties evaluated at gauss point temperatures (ptype=1). Solver 12 is used, as this is the default solver for MPP (Massively Parallel Processing) simulations. TSF is set to 1000 which speeds up the thermal problem by a factor1000. This scaling is elaborated in appendix D.

```
*CONTROL_THERMAL_SOLVER
$#   atype     ptype    solver     cgtol       gpt    eqheat     fwork       sbc
         1         1        12
$#                                                                          TSF
                                                                           1000
```

Listing C.11: Control thermal solver.

Listing C.12 defines the timesteps used by the thermal solver, in the present Finite Element model these are controlled by a load curve (Listing C.13).

```
*CONTROL_THERMAL_TIMESTEP
$#      ts       tip       its      tmin      tmax     dtemp      tscp      lcts
                   1   1.0E-03                                                4
```

Listing C.12: Control thermal timestep.

Listing C.13 is a load curve which defines the timesteps for the thermal solver, the size of these timesteps is explained in the 9th. semester report appended on the appendix-CD.

```
*DEFINE_CURVE_TITLE
Time_step
$#    lcid      sidr       sfa       sfo      offa      offo    dattyp
         4
$#                a1                  o1
              0.000              0.0032
           &dt1                  0.0032
           &dt2                     0.1
             20.000                 0.1
```

Listing C.13: Timestep for the thermal solver.

Listing C.14 is used to define the response of the solid element.

```
*CONTROL_SOLID
$#   ESORT    FMATRX   NIPTETS    SWLOCL
                   2
```

Listing C.14: Control solid.

C.4 Database

Listing C.15 contains the database keywords, which in the present simulation are used to obtain a desired amount of plots of the simulation.

```
*DATABASE_BINARY_D3DUMP
$#      dt      lcdt      beam     npltc    psetid
      0.01
$
$
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc    psetid
      0.03
$
$
*DATABASE_EXTENT_BINARY
$    neiph     neips    maxint    strflg    sigflg    epsflg    rltflg    engflg
                             3                 1         1         1         1
$   cmpflg    ieverp    beamip     dcomp      shge     stssz    n3thdt   ialemat
                                                2
$  nintsld   pkp_sen      sclp    unused     msscl     therm    iniout    iniout
                                                2         2       ALL STRESS_GL
```

Listing C.15: Database keywords.

C.5 Implicit

During the development of the Finite Element model through the 9th. semester, it was determined to run the Finite Element model as part explicit analysis and part implicit. The -5 placed under IMFLAG in listing C.16 allows switching between explicit analysis and implicit analysis to be performed by using a load curve. The load curve dictates that an explicit solver must be used during the time of the laser scan and an additional 0.3ms, (dt1) + 0.3ms, and then switch, such that an implicit solver is used during dwell time.

```
*CONTROL_IMPLICIT_GENERAL
$#  IMFLAG        DT0    IMFORM      NSBS       IGS     CNSTN      FORM    ZERO_V
        -5      0.001                             1                             0
```

Listing C.16: Control implicit general.

Listing C.17 is used to define parameters for the automatic step size control.

```
*CONTROL_IMPLICIT_AUTO
$#    IAUTO    ITEOPT    ITEWIN     DTMIN     DTMAX     DTEXP
          1        11         5       0.2
```

Listing C.17: Control implicit auto.

Listing C.18 is used to define that a nonlinear solution is desired.

```
*CONTROL_IMPLICIT_SOLUTION
$#  NSOLVR    ILIMIT    MAXREF     DCTOL     ECTOL     RCTOL     LSTOL    ABSTOL
         2                  15
```

Listing C.18: Control implicit solution.

Listing C.19 is used to output the calculation summary given by LS-Dyna.

```
*CONTROL_IMPLICIT_SOLVER
$#  LSOLVR    LPRINT     NEGEV     ORDER      DRCM    DRCPRM   AUTOSPC   AUTOTOL
                    1                                   100         1     1.E-8
$#  LCPACK    MTXDMP
         2
```

Listing C.19: Control implicit solver.

C.6 Boundary Conditions

Listing C.21 applies boundary convection to the Finite Element model.

```
*BOUNDARY_CONVECTION_SET
$#    SSID
         1
$#   HLCID     HMULT     TLCID     TMULT       LOC
        12         1                 293
```

Listing C.20: Boundary convection.

```
*DEFINE_CURVE_TITLE
Convection
$#    lcid      sidr       sfa       sfo      offa      offo    dattyp
        12
$#                a1                  o1
              0.000             1.5E-8
           &dt1                 1.5E-8
            &dt2                1.5E-6
              20.000            1.5E-6
```

Listing C.21: Load curve for the boundary convection heat transfer.

Listing C.22 applies boundary radiation to the Finite Element model, where FMULT is obtained by equation C.1 [Shapiro and Lo, 2009]. Remark that FMULT is converted to the unit scheme used in the present Finite Element model.

$$\text{FMULT} = \sigma \cdot \epsilon \cdot F \rightarrow 5.67 \cdot 10^{-8} \frac{\text{J}}{\text{s} \cdot \text{m}^2 \cdot \text{K}^4} \cdot 0.4 \cdot 1 = 2.268 \cdot 10^{-17} \frac{\text{J}}{\text{ms} \cdot \text{mm}^2 \cdot \text{K}^4} \tag{C.1}$$

Where σ is the Stefan Boltzmann constant, ϵ is the emissivity which describes the fraction of energy radiated from the surface (set to 0.4 [Mikron Instrument Company Inc., 2013]) and F is an exchange factor, which is set to 1 when radiation is between a component and the environment [Shapiro and Lo, 2009].

```
1 *BOUNDARY_RADIATION_SET
2 $#     SSID      TYPE
3          1         1
4 $#    FLCID     FMULT     TLCID     TMULT       LOC
5          0 2.268E-17         0       293
```

Listing C.22: Boundary radiation.

C.7 Laser Beam

To deliver heat flux to the blank, the keyword shown in listing C.23 is utilised, where ssid designates the surface segment of the blank and lcid designates the load curve which must be applied to this segment. Remark that the developed framework, adjusts ssid, such that both the top and bottom surface can be irradiated. The load curve is defined as a Gaussian function, which is created by means of the DEFINE_FUNCTION keyword, shown in listing C.24. The working principle of this method is that a circle with a Gaussian heat flux distribution and with the radius of the laser beam, is moved across the aforementioned surface segment. During this movement, all surface elements are evaluated with respect to their euclidian distance from the center of the laser beam (distc line 15 of listing C.24). If the elements are within the radius of the laser beam, they receive a calculated amount of heat flux with respect to the Gaussian distribution(fl line 18 and 20 of listing C.24). Similar implementation has been used by [Roll et al., 2011] with good results, why this is assessed to be suited. Remark that the variables marked "name"(time) listed in line 9-14 in listing C.24, contains process variables which are included as DEFINECURVES. This is done as these are determined by use of the framework in between every laser scan. E.g. the movement of the laser beam(xloc, yloc) is determined by the framework as X and Y movement load curves.

```
1 *BOUNDARY_FLUX_SET
2 $#   ssid
3          2 (top surface)
4 $#    lcid      mlc1      mlc2      mlc3      mlc4       loc     nhisv       fid
5          3         0         0         0         0
```

Listing C.23: Boundary flux set.

Listing C.24 contains the Gaussian function that is used to simulate the laser beam, the function was received from [Kennedy, 2013], but has been heavily altered for the purposes of the simulation.

```
*DEFINE_FUNCTION
$#     fid     defintion
         3     flux a function of position
float flux(float x,float y,float z,float vx,float vy,float vz,
        float tinf,float time)
{
 float radius, xspot, yspot, fl, distc, A, P, Pi ;

 A = &A ;
 P = Power(time);
 Pi = 3.1416 ;
 radius = radi(time);
 xspot = xloc(time) ;
 yspot = yloc(time) ;
 distc = sqrt((x-xspot)**2 + (y-yspot)**2) ;

 if ((x-xspot)**2 + (y-yspot)**2 <= radius**2)
   fl = -((2 * A * P) / (Pi * radius**2) * exp((-2 * distc**2) / radius**2));
 else
   fl = 0. ;
 return (fl) ;
}
```

Listing C.24: Gaussian function that is used to simulate the laser beam.

C.8 Blank and Material

Listing C.25 is the PART keyword for the blank used in the Finite Element model. It defines the part ID, section ID as well as the mechanical and thermal material ID's.

```
*PART
$# title
Blank
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1                                              2
```

Listing C.25: Part keyword.

Listing C.26 defines the element formulation used, which in this case is a default formulation.

```
*SECTION_SOLID_TITLE
Blank
$#   secid    elform
         1         1
```

Listing C.26: Section keyword.

To ensure that the model exhibits proper physical behaviour during both the thermal and the mechanical analysis. It is assessed necessary to incorporate the nonlinear behaviour of the material properties, as a function of temperature. It is assessed that phase change does not occur during the process i.e. melting of the surface. The temperature dependency of the material is

coped with by introducing two material keywords in LS-Dyna. One which defines temperature dependency of the thermal properties e.g. heat capacity and conductivity and one which defines the temperature dependent mechanical properties e.g. yield stress. The material keywords are listed in listing C.27 and C.28. For both material keywords it is assessed that the material shows isotropic behaviour. Both material keywords consists of material properties which are linearly interpolated between 8 discrete temperature values in the range 273K-1773K. This discretisation is assessed to be sufficient for the purpose of the model. The material data for stainless steel 1.4301, which is used throughout this project is obtained through [Deng and Murakawa, 2006] and [Samuel et al., 1992] and is appended in appendix A. The plastic hardening modulus (etan) in listing C.27, is approximated by use of Hollomon's equation, shown in equation C.2. This was necessary as no data has been available. The approximation was performed by utilising the temperature dependent strain index (K) and strain hardening exponent (n), obtained from [Samuel et al., 1992]. The data is inserted into Hollomon's equation and differentiated. Etan is obtained by evaluating the differentiated expression, with respect to the strain value for which etan is required. As the strain value varies across the surface of the blank, it is not feasible to determine etan for all values, why it is chosen to determine etan from one strain value (0.1). Calculations of etan are appended on the appendix-CD.

$$\sigma = K + \epsilon^n \tag{C.2}$$

```
*MAT_ELASTIC_PLASTIC_THERMAL_TITLE
Blank mechanical properties
$#     mid        ro
         1 7.9000E-6
$#      t1        t2        t3        t4        t5        t6        t7        t8
    273.00    373.00    473.00    573.00    673.00    873.00   1073.00   1773.00
$#      E1        E2        E3        E4        E5        E6        E7        E8
   198.500    193.00    185.00    176.00    167.00    159.00    151.00     10.00
$#     pr1       pr2       pr3       pr4       pr5       pr6       pr7       pr8
     0.294     0.295     0.301     0.310     0.318     0.326     0.333     0.388
$#  alpha1    alpha2    alpha3    alpha4    alpha5    alpha6    alpha7    alpha8
 1.7000E-5 1.7400E-5 1.8000E-5 1.8600E-5 1.9100E-5 1.9600E-5 2.0200E-5 2.1600E-5
$#   sigy1     sigy2     sigy3     sigy4     sigy5     sigy6     sigy7     sigy8
  0.265000  0.218000  0.186000  0.170000  0.155000  0.149000  0.091000  0.010000
$#   etan1     etan2     etan3     etan4     etan5     etan6     etan7     etan8
     1.986     1.787     1.632     1.554     1.642     1.589     0.159     0.159
```

Listing C.27: Mechanical properties of the blank with respect to temperature.

```
*MAT_THERMAL_ISOTROPIC_TD_TITLE
Blank thermal properties
$#     tmid       tro     tgrlc    tgmult      tlat      hlat
         2
$#       t1        t2        t3        t4        t5        t6        t7        t8
    273.00    373.00    473.00    573.00    673.00    873.00   1073.00   1773.00
$#       c1        c2        c3        c4        c5        c6        c7        c8
    462.00    496.00    512.00    525.00    540.00    577.00    604.00    700.00
$#       k1        k2        k3        k4        k5        k6        k7        k8
 1.4600E-5 1.5100E-5 1.6100E-5 1.7900E-5 1.8000E-5 2.0800E-5 2.3900E-5  1.200E-4
```

Listing C.28: Thermal properties of the blank with respect to temperature.

C.9 Miscellaneous

Listing C.29 implements damping in the model in order to stabilise the deformation rate of the blank.

```
*DAMPING_GLOBAL
                  50
```

Listing C.29: Damping global.

Listing C.30 is used to set the initial temperature of the blank to room temperature (293K $\approx$ 20°C).

```
*INITIAL_TEMPERATURE_SET
$     NSID      TEMP       LOC
         0      293.
```

Listing C.30: Initial temperature set.

Listing C.31 is used to set a boundary temperature for the nodes at the fixed edge, in the tests of single curved geometries. The boundary temperature represents the conductive cooling from the clamp fixture. The boundary temperature is set to room temperature (293K $\approx$ 20°C).

```
*BOUNDARY_TEMPERATURE_SET
$     NSID      LCID     CMULT       LOC
         7                 293
```

Listing C.31: Boundary temperature set.

Appendix D

Verification of the Temperature Speedup Factor in LS-DYNA

LS-Dyna's thermal solver allows utilisation of a "Thermal Speedup Factor" (TSF). The TSF is used to artificially scale the thermal properties of a thermal problem e.g. conductivity, heat convection coefficient etc. The TSF is set in correspondence to the time scaling i.e. if the scan speed of the laser beam is scaled by a factor 1000, the TSF is equally set to 1000. In cooperation with a scaling of the speed the TSF allows a shorter simulation time. [LSTC, 2013]

To test if the TSF is applicable in the problem of laser forming a stainless steel blank with nonlinear thermal properties with respect to temperature, a thermal test example is created. The example is a stainless steel rod, shown in figure D.1. The rod has equivalent properties as the blank used in the laser forming process. The rod is prescribed with a constant boundary temperature of 1273K prescribed at the end nodes. Furthermore, all surfaces are subject to heat loss from convection and radiation. The initial temperature of the rod is set equal to 293K. The rod is discretised into 10 equally sized solid elements. The problem is solved using the same thermal solver as in the simulation model for the laser forming process.

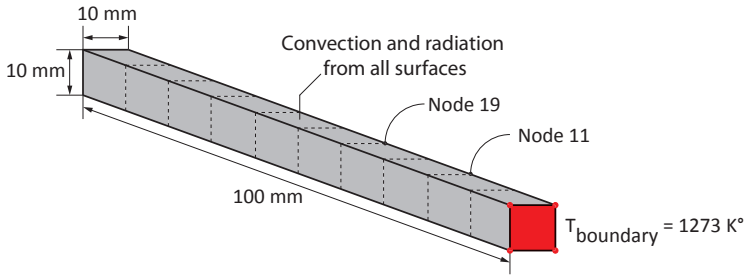


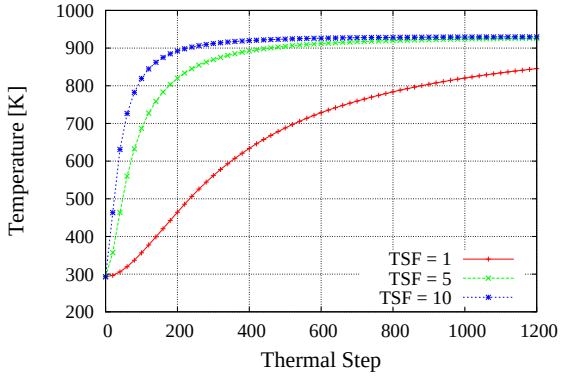
Figure D.1: The rod example.

Three different TSF values being 1, 5 and 10 are tested. The temperature development with respect to thermal timesteps is investigated in node 11 and 19, shown in figure D.1. The temperature development of node 11, shown in figure D.2a increases with increasing size of the TSF

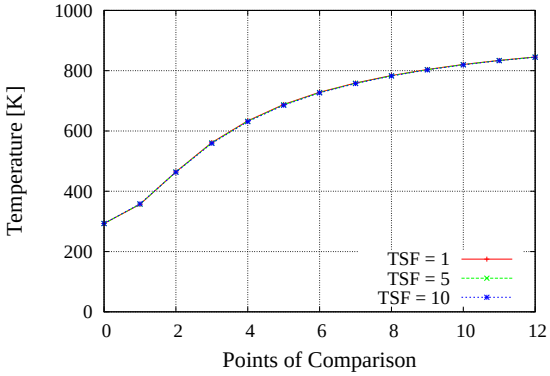
hence the scaling affects that rate of heat transfer in the solution as, expected. Moreover, it is observed that all solutions converge towards a steady state where the heat transferred to the node equals the heat leaving the node.

To test if the TSF works i.e. if it is possible to decrease the simulation time, by scaling of the thermal problem, the temperature development of the three different TSF values is compared at predefined comparison points. For TSF = 1, a comparison point is set at every 100^{th} thermal step. As the solution for TSF = 5 is scaled by a factor of five a comparison point is set at every 20^{th} thermal step to compensate for the scaling. Similarly the solution with TSF = 10 has comparison points at every 10^{th} thermal step. In each comparison point the node temperature should be similar for each of the three solutions. The comparison for node 11 is shown in figure D.2b. It is shown that the scaled temperatures correlate with the unscaled.

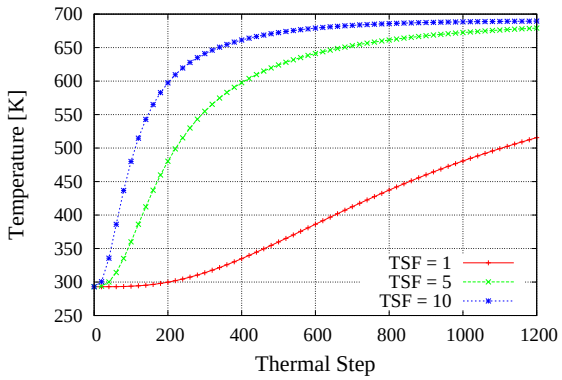
Similar results are obtained for node 19, these are presented in figure D.2c and D.2d. In addition, it is observed that the temperature converges towards a lower temperature in node 19. This is due to the position of node 19, as the heat is required to be conducted into more material. Moreover, a larger surface area is present between the heat source and node 19, why convection and radiation removes more heat from the rod. Based on the obtained results from the example, it is assumed that the TSF is applicable in the laser forming simulation.



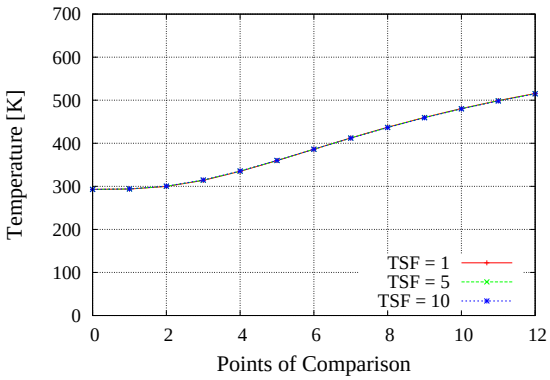
(a) Temperature development in node 11 for the different solutions.



(b) Temperature development in node 11 at different comparison points.



(c) Temperature development in node 19 for the different solutions.



(d) Temperature development in node 19 at different comparison points.

Figure D.2: Results from the rod experiment.

Appendix E

LaserForm.Java

This appendix contains the Java main program. When executed the Java main program utilises tools located in the ToolBox, which is appended in appendix F. The Java main program is also appended on the appendix-CD.

```
1  /*
2   *                                    |        |    VT4-2.215 Laser Forming
3   *                                    |        |    3 Februar 2014
4   *                                    |________|
5   *                                     \      /    /
6   *                                      \____/    /
7   *                                        <|>    /
8   *                                        <|>   /
9   *                                        <|>  /
10  *                                        <|>/
11  *                        ________________|/
12  *
13  */
14
15 // Importing packages
16 import java.io.*;
17 import java.lang.*;
18 import java.util.*;
19 import bek.opti.ObjectUtility;
20
21 public class LaserForm extends ObjectUtility {
22 //%%%%%%%%%%%%%%%%%%%%%%%% USED VARIABLES %%%%%%%%%%%%%%%%%%%%%%%%
23
24 //****************** Must be specified ******************
25     private String Geometry = "dome";//Desired geometry;
26     private int MaxScan = 60;//Max number of iterations
27     private int toleranceaccep = 1;//Acceptable sum of absolute error
28
29 //****************** Must be specified ******************
30     private int k = 1;//Iteration counter for the framework
31     private boolean doesExist = true;//Boolean used for conditioned wait functions
32     private int tolerancemeas = 1000;//Measured sum of absolute error
33     private String shell_clean = "./shellscripts/shell_clean.sh";//Shell-script
```

```
        which cleans the directory
34      private String shell_run = "./shellscripts/shell_run.sh";//Shell-script which
            runs the Finite Element model of the laser forming process for k=1
35      private String shell_run2 = "./shellscripts/shell_run2.sh";//Shell-script which
            runs the Finite Element model of the laser forming process for k>1
36      private String shell_runonestepinit = "./shellscripts/shell_runonestepinit.sh";
            //Shell-script which runs the OneStep solver for iteration k=1
37      private String shell_runonestep = "./shellscripts/shell_runonestep.sh";//Shell-
            script which runs the OneStep solver for iteration k>1
38      private String shell_generateshell = "./shellscripts/shell_generateshell.sh";//
            Shell-script which generates the shell model used for the onestep analysis
39      private String shell_centroids = "./shellscripts/shell_centroids.sh";//Shell-
            script which outputs the element centroids from prepost from the current
            geometry
40      private String shell_centroidsinit = "./shellscripts/shell_centroidsinit.sh";//
            Shell-script which outputs the element centroids from prepost from the
            desired geometry
41      private String shell_centroidsblank = "./shellscripts/shell_centroidsblank.sh";
            //Shell-script which outputs the element centroids from prepost from the
            blank.k
42      private String shell_centroidsonestep = "./shellscripts/shell_centroidsonestep.
            sh";//Shell-script which outputs the element centroids from prepost for the
            flattened geometry in the OneStep solver
43      private String centroidfile = "centroidsblank.data";//Element centroids used in
            the scan path algorithm
44      File myFile = new File("status.out");//Status.out is a file created by Dyna (
            used to perform conditioned wait functions)
45      private double[][][] strainstopfirst = new double[40][40][6];//Top surface
            strain field for k=1
46      private double[][][] strainsbotfirst = new double[40][40][6];//Bottom surface
            strain field for k=1
47      private double[][][] strainscenterfirst = new double[40][40][6];//Midplane
            strain field for k=1
48      private double[][][] strainsinfinalupper = new double[40][40][6];//Upper scan
            path strain field for iteration k=1
49      private double[][][] strainsinfinallower = new double[40][40][6];//Lower scan
            path strain field for iteration k=1
50      private double[][][] strainstop = new double[40][40][6];//Top surface strain
            field extracted from OneStep solver for k>1 (Currently achieved strain)
51      private double[][][] strainsbot = new double[40][40][6];//Bottom surface strain
            extracted from OneStep solver for iteration k>1 (Currently achieved strain)
52      private double[][][] strainscenter = new double[40][40][6];//Midplane strain
            field extracted from OneStep solver for iteration k>1 (Currently achieved
            strain)
53      private double[][][] strainsavgupper = new double[40][40][6];//Average of
            midplane and top surface strain field extracted from OneStep solver for
            iteration k>1  (Currently achieved strain)
54      private double[][][] strainsavglower = new double[40][40][6];//Average of
            midplane and bottom surface strain field extracted from OneStep solver for
            iteration k>1  (Currently achieved strain)
55      private double[][][] strainspathupper = new double[40][40][6];//
            strainsinfinalupper-strainsavgupper = the required strain to be achieved in
            the upper scan path strain field for iteration k>1
```

```
56      private double[][][] strainspathlower = new double[40][40][6];//
            strainsinfinallower-strainsavglower = the required strain to be achieved in
            the lower scan path strain field for iteration k>1
57      private double[][][] strainscenterreq = new double[40][40][6];//The midplane
            strain field = the required strain to be achieved in the midplane for
            iteration k>1
58      private double[][][] strainssurreq = new double[40][40][6];//The required
            strains in either the top or bottom surface strain field (dependent of the
            surface scanned in the current iteration)
59      private double[][][] ZValue = new double[40][40][3];//Z coordinates of desired
            geometry above element centroids in the deformed shell
60      private double[][][] strainspath = new double[40][40][6];//The scan path strain
            field i.e. the remaining strain to be achieved in either the upper or lower
            scan path strain field
61      private double[] AbSumError = new double[40 * 40];//Sum of absolute error in Z
            for all iterations (for post processing)
62      private double[][] Thresh;//Threshold field of 2nd principal strain (1 if above
            threshold value 0 if below)
63      private double[][][] Prin;//Size and orientation of minimum principal strains
            for the scan path strain field
64      private double[][][] PrinSur;//Size and orientation of minimum principal strains
             for the surface strain field used to create the scan path strain field
65      private double[][][] Princenter;//Size and orientation of minimum principal
            strains for the midplane strain field
66      private int[][] ElPath;//Elements visited in the scan path
67      private double[][] ElPathPara;//Determined processing parameters in scan path
68      private String strainPathPosition;//Position of strain field (upper or lower)
69
70  //%%%%%%%%%%%%%%%%%%%%%%%%% MAIN METHOD %%%%%%%%%%%%%%%%%%%%%%%%%
71      public static void main(String[] args) {
72          LaserForm Autokoer = new LaserForm();
73          System.out.println("START - The Program is Started");
74          Autokoer.run();
75          System.out.println("END - The Program is Finished");
76      }
77  //%%%%%%%%%%%%%%%%%%%%%% LASER FORMING METHOD %%%%%%%%%%%%%%%%%%%%%%
78
79      public void run() {
80          ToolBox tool = new ToolBox();// The Toolbox class is initialised
81
82          //Extract the desired geometry from directory storing all geometries
83          tool.SystemCall("cp /home/tube/Laserforming/new/desiredgeometries/kfiles/" +
                Geometry + ".k desired.k");
84          tool.SystemCall("cp /home/tube/Laserforming/new/desiredgeometries/kfilesfine
                /" + Geometry + "fine.k desiredfine.k");
85
86  //%%%%%%%%%%%%%%%% CHECK INITIAL GEOMETRICAL ERROR %%%%%%%%%%%%%%%%
87          //Centroids are extracted from the initial model (desired geometry)
88          System.out.println("Open prepost and output the coordinates of the element
                centroids(desired geometry)");
89          tool.SystemCall(shell_centroidsinit);
90
91          //Remark in the following shell script, several tasks are performed, such as
```

```
            extraction of a shell model from the
92          //solid blank and extraction of centroids from the shell model
93          System.out.println("Open prepost and output the coordinates of the element
                centroids for the undeformed blank.k");
94          tool.SystemCall(shell_centroidsblank);
95
96          //Check geometrical error (Surface error & sum of absolute error)
97          ZValue = tool.MapElementCentroidToCurrent("centroidsblank.data");
98          AbSumError[k - 1] = tool.ErrorData(ZValue, "centroidsblank.data");
99          tolerancemeas = AbSumError[k - 1];
100         System.out.println("Absolute error: " + AbSumError[k - 1]);//Current sum of
                absolute error
101         ToFile(AbSumError, "debugplot/AbSumError.data");//File containing the sum of
                absolute error
102
103 //%%%%%%%%%%%%%%%%%%%% START LASER FORMING LOOP %%%%%%%%%%%%%%%%%%
104         while (toleranceaccep < tolerancemeas && k - 1 < MaxScan) {
105
106 //%%%%%%%%%%%%%%%%%%%%%%%% CLEAN DIRECTORY %%%%%%%%%%%%%%%%%%%%%%%%
107             System.out.println("Cleaning the directory");
108             tool.SystemCall(shell_clean);// run shell script which cleans directory
109
110             // Wait until the directory is cleaned
111             while (doesExist == true) {
112                 System.out.println("Not finished cleaning the directory");
113                 doesExist = myFile.exists();
114             }
115             doesExist = true;
116             System.out.println("Finished cleaning the directory");
117
118 //%%%%%%%%%%%%%%%%%%% PERFORM ONESTEP SIMULATION %%%%%%%%%%%%%%%%%%
119             if (k == 1) {
120                 System.out.println("Performing the initial(k=1) onestep analysis for
                        the desired geometry");
121                 tool.SystemCall(shell_runonestepinit);// must be run on fine meshed
                        desired geometry
122             } else {
123                 System.out.println("Performing the " + k + ". onestep analysis");
124                 tool.SystemCall(shell_runonestep);// must be run on the shell model
                        of current geometry created from the solid model
125             }
126
127 //%%%%%%%%%%%%%%%%%%%% ANALYSE ONESTEP RESULTS %%%%%%%%%%%%%%%%%%%%
128             //Strain fields are extracted from the results of the OneStep solver
129             //Furthermore, the size and orientation of minimum principal strains are
                        determined for the laser scanned surface strain field and the
                    midplane strain field (used for determining processing parameters)
130             //Extract centroids from flattened blank in onestep (d3plot)
131             System.out.println("Open prepost and output the coordinates of the
                    element centroids for flat geometry (d3plot)");
132             tool.SystemCall(shell_centroidsonestep);
133
134             if (k == 1) {
```

```
135                 //In first iteration the strain fields are determined by mapping the
                        strains from desired geometry to blank geometry
136                 strainsbotfirst = tool.MapDesiredToShell("bottom", "flip", Geometry)
                        ;
137                 strainstopfirst = tool.MapDesiredToShell("top", "flip", Geometry);
138                 strainscenterfirst = tool.AvgArray(strainstopfirst, strainsbotfirst)
                        ;
139                 Princenter = tool.PrincipalStrain(strainscenterfirst);
140
141                 //Determine the surface to scan (surface with the largest negative
                        principal strain) and create scan path strain field
142                 strainPathPosition = tool.PathPosition(strainsbotfirst,
                        strainstopfirst);//Determine surface with largest negative
                        principal strain
143                 if (strainPathPosition == "upper") {
144                     strainspath = tool.AvgArray(strainstopfirst, strainscenterfirst)
                            ;//Use the upper scan path strain field
145                     PrinSur = tool.PrincipalStrain(strainstopfirst);//Size and
                            orientation of minimum principal strains for the top surface
                             strain field
146                     System.out.println("Upper surface has the largest negative
                            principal strain");
147                 }
148                 if (strainPathPosition == "lower") {
149                     strainspath = tool.AvgArray(strainsbotfirst, strainscenterfirst)
                            ;//Use the lower scan path strain field
150                     PrinSur = tool.PrincipalStrain(strainsbotfirst);//Size and
                            orientation of minimum principal strains for the bottom
                            surface strain field
151                     System.out.println("Lower surface has the largest negative
                            principal strain");
152                 }
153             } else {
154                 System.out.println("Open prepost and output the coordinates of the
                        element centroids for current geometry (d3plot)");
155                 tool.SystemCall(shell_centroids);
156                 centroidfile = "deformedcentroid.data";//Current geometry centroids
                        used in subsequent scan path
157
158                 //In the second and following iterations, calculate required strain
                        fields by extracting the results from the OneStep solver (
                        current geometry) and subtracting these from the initial OneStep
                         result (desired geometry)
159                 //Results from OneStep solution of current geometry
160                 strainsbot = tool.StrainToShell("bottom");
161                 strainstop = tool.StrainToShell("top");
162                 strainscenter = tool.AvgArray(strainstop, strainsbot);
163                 strainsavgupper = tool.AvgArray(strainstop, strainscenter);
164                 strainsavglower = tool.AvgArray(strainsbot, strainscenter);
165
166                 //Scan path strain fields from OneStep of desired geometry
167                 strainsinfinalupper = tool.AvgArray(strainstopfirst,
                        strainscenterfirst);
```

```
168                 strainsinfinallower = tool.AvgArray(strainsbotfirst,
                        strainscenterfirst);
169
170                 //Subtract the current OneStep results (current geometry) from the
                        initial OneStep results (desired geometry) to determine required
                         strain
171                 for (int x = 0; x < 40; x++) {
172                     for (int y = 0; y < 40; y++) {
173                         for (int i = 0; i < 6; i++) {
174                             strainspathupper[x][y][i] = strainsinfinalupper[x][y][i]
                                    - strainsavgupper[x][y][i];
175                             strainspathlower[x][y][i] = strainsinfinallower[x][y][i]
                                    - strainsavglower[x][y][i];
176                             strainscenterreq[x][y][i] = strainscenterfirst[x][y][i]
                                - strainscenter[x][y][i];
177                         }
178                     }
179                 }
180
181                 //Select scan path surface (Surface with largest remaining negative
                        principal strain) and determine the scan path strain field
182                 strainPathPosition = tool.PathPosition(strainspathlower,
                        strainspathupper);
183                 for (int x = 0; x < 40; x++) {
184                     for (int y = 0; y < 40; y++) {
185                         for (int i = 0; i < 6; i++) {
186                             if (strainPathPosition == "upper") {
187                                 strainspath[x][y][i] = strainspathupper[x][y][i];
188                             }
189                             if (strainPathPosition == "lower") {
190                                 strainspath[x][y][i] = strainspathlower[x][y][i];
191                             }
192                         }
193                     }
194                 }
195
196                 //Determine required strain in laser scanned surface
197                 for (int x = 0; x < 40; x++) {
198                     for (int y = 0; y < 40; y++) {
199                         for (int i = 0; i < 6; i++) {
200                             if (strainPathPosition == "lower") {
201                                 strainssurreq[x][y][i] = strainsbotfirst[x][y][i] -
                                        strainsbot[x][y][i];//Use the lower scan path
                                        strain field
202                             }
203                             if (strainPathPosition == "upper") {
204                                 strainssurreq[x][y][i] = strainstopfirst[x][y][i] -
                                        strainstop[x][y][i];//Use the upper scan path
                                        strain field
205                             }
206                         }
207                     }
208                 }
```

```
209
210                 //Determine size and orientation of minimum principal strains
211                 PrinSur = tool.PrincipalStrain(strainssurreq);
212                 Princenter = tool.PrincipalStrain(strainscenterreq);
213             }
214
215             ToFile(tool.getComponent(strainspath, 0, 0), "debugplot/avg1.data");//
                    Data for post processing - Scan path strain field
216
217 //%%%%%%%%%%%% DETERMINE SCAN PATH & PROCESS VARIABLES %%%%%%%%%%
218             //Path planning must be performed - This is done by means of the scan
                    path algorithm
219             //Furthermore process variables are determined with respect to the
                    required in-plane and bending strain
220             Prin = tool.PrincipalStrain(strainspath);//Size and orientation of
                    minimum principal strains for the scan path strain field
221             Thresh = tool.Thresh(Prin, 50.0);//50.0 refer to the threshold
                    percentage
222             ToFile(tool.getComponent(Prin, 0, 0), "debugplot/prinstrain.data");//
                    Data for post processing - size of minimum principal strain
223             tool.VectorPlot(Prin);//Data for post processing - Vector field plot of
                    the orientation of minimum principal strain field
224             ElPath = tool.ElementPath(Thresh, Prin, 50.0, centroidfile);//Determine
                    scan path w.r.t. scan path algorithm (50.0 refers to the allowable
                    angular deviation)
225             ElPathPara = tool.ElPathPara(ElPath, PrinSur, Princenter, 0,
                    strainPathPosition);//Determine process variables (0=fraction
                    distribution, 1=pure tgm with fixed variables, 2=pure um with fixed
                    variables, 3=combined um/tgm with fixed variables
226             tool.GenerateDefineCurve(ElPath, centroidfile, ElPathPara);//Generate
                    define curves (input for to the Finite Element model of the laser
                    forming process)
227
228 //%%%%%%%%%%%%%%%% PERFORM LASER FORMING SIMULATION %%%%%%%%%%%%
229             if (k == 1) {
230                 System.out.println("
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
231                 System.out.println("%%%%%%%%% Simulation model is initiated
                        %%%%%%%%%");
232                 System.out.println("
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
233                 tool.SystemCall(shell_run);
234             } else {
235                 System.out.println("
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
236                 System.out.println("%%%%%%%%% Simulation model " + k + " is
                        initiated %%%%%%%%");
237                 System.out.println("
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
238                 tool.SystemCall(shell_run2);
239             }
240             System.out.println("Simulation " + k + " is finished");
241
```

```
242 //%%%% GENERATE SHELL MODEL FOR NEXT ITERATION OF FRAMEWORK %%%%
243             //Generate shell model which is to be used in a new OneStep analysis
244             tool.SystemCall(shell_generateshell); // run shell script "generate
                    shell model"
245             System.out.println("A shell model has been created");
246 
247 //%%%%%%%%%%%% CHECK CURRENT GEOMETRICAL ERROR %%%%%%%%%%%%
248             tool.SystemCall(shell_centroids);
249             ZValue = tool.MapElementCentroidToCurrent("deformedcentroid.data");
250             AbSumError[k] = tool.ErrorData(ZValue, "deformedcentroid.data");
251             tolerancemeas = AbSumError[k];
252             System.out.println("Absolute error: " + AbSumError[k]);
253             ToFile(AbSumError, "debugplot/AbSumError.data"); //Data for post
                    processing  - File containing the sum of absolute error
254 //%%%%%%%%%%%%%%% DATA LOGGING FOR POST-PROCESSING %%%%%%%%%%%%%%
255             tool.SystemCall("mkdir Scan" + k);
256             tool.SystemCall("mkdir " + Geometry);
257             tool.SystemCall("cp deformedshell.k Scan" + k);
258             tool.SystemCall("cp d3dump01 Scan" + k);
259             tool.SystemCall("cp d3plot Scan" + k);
260             tool.SystemCall("cp d3hsp Scan" + k);
261             tool.SystemCall("cp -r debugplot Scan" + k);
262             tool.SystemSleep(1000);        // Wait until the information is copied
263             tool.SystemCall("cp d3plot01 Scan" + k);
264             tool.SystemSleep(1000);        // Wait until the information is copied
265             tool.SystemCall("rm d3plot01");
266             tool.SystemCall("mv deformedshell.data Scan" + k);
267             tool.SystemCall("mv deformedcentroid.data Scan" + k);
268             tool.SystemCall("mv PARAMETRES Scan" + k);
269             tool.SystemCall("mv X_DEFINE_CURVE Scan" + k);
270             tool.SystemCall("mv Y_DEFINE_CURVE Scan" + k);
271             tool.SystemCall("mv P_DEFINE_CURVE Scan" + k);
272             tool.SystemCall("mv laststate.k Scan" + k);
273             tool.SystemCall("mv laststatecomplete.k Scan" + k);
274             tool.SystemCall("mv dyna.temp Scan" + k);
275             tool.SystemCall("mv onestepresult Scan" + k);
276             tool.SystemCall("mv Scan* " + Geometry);
277             System.out.println("Done moving files! -> restart");
278             k++;
279         }
280 //%%%%%%%%%%%%%%% PRINT OUT STOP CRITERIA  %%%%%%%%%%%%%%
281         if (k - 1 == MaxScan) {
282             System.out.println("MaxScan has been reached.");
283         }
284         if (tolerancemeas == toleranceaccep) {
285             System.out.println("Tolerance is acceptable");
286         }
287         tool.SystemCall(shell_clean);
288         tool.SystemCall("rm desired.k");
289     }
290 }
```

Appendix F

ToolBox.Java

This appendix contains the ToolBox used by the Java main program. The ToolBox is also appended on the appendix-CD.

```
1  /*
2   *                                    |        |    VT4-2.215 Laser Forming
3   *                                    |        |    3 Februar 2014
4   *                                    |________|
5   *                                     \      /   /
6   *                                      \____/   /
7   *                                        <|>    /
8   *                                        <|>   /
9   *                                        <|> /
10  *                                        <|>/
11  *                      ____________________|/
12  *
13  * This Toolbox is for the LaserForm.java program. The program is made by the VT4
        -2.215 Laser Forming group.
14  *
15  *                     '-----------''-----------'
16  *                     |___ ===============___|
17  *                        |      Toolbox      |
18  *                        |___________________|
19  */
20
21  // Importing packages
22  import java.io.*;
23  import java.util.*;
24  import java.text.*;
25  import java.nio.*;
26  import bek.opti.ObjectUtility;
27
28  public class ToolBox extends ObjectUtility {
29  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
30      // Method for executing the System Calls.
31      public void SystemCall(String executing) {
32
33          //METHOD DESCRIPTION:
```

```
34          /* The method is capable of executing the system calls.
35           */
36          try {
37              Process proc = Runtime.getRuntime().exec(executing);
38              try {
39                  proc.waitFor();
40                  System.out.println("Process " + executing + " executed");
41              } catch (InterruptedException e) {
42                  System.err.println("process was interrupted");
43              }
44
45              proc.getInputStream().close();
46              proc.getOutputStream().close();
47              proc.getErrorStream().close();
48
49          } catch (IOException e) {
50              System.err.println("IOException starting process!");
51          }
52      }
53      //End of method
54  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55      // Method for implementing a Sleep
56      public void SystemSleep(int time) {
57
58          //METHOD DESCRIPTION:
59          /* The method is capable of forcing the java code to sleep for a specified
                amount of time (ms):
60           */
61          try {
62              Thread.sleep(time);
63          } catch (InterruptedException ex) {
64              System.out.println("Error, when trying to sleep for " + time + " ms");
65          }
66      }
67      //End of method
68  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69      // Method for reading onestepresult file
70      public double[][] ReadOnestep(String surface, String onestepfile, int noElem) {
71
72          //METHOD DESCRIPTION:
73          /* The method is capable of reading files formated as a onestepresult file
                and output an Array[element ID][strain component] by performing the
                following step:
74           * - Select surface of interest
75           * - Read all lines in file
76           * - Identify strain part of file (*INITIAL_STRAIN_SHELL INDEX)
77           * - Create strings holding the strains of the selected surface
78           * - Split strings into components
79           * - Convert string to double format
80           * - Output array
81           */
82          System.out.println("Starting ReadOneStep");
83          //Variables used in the method
```

```
84          String selector = surface;//Selects the surface of interest (top or bottom)
85          String input = onestepfile;//Title of the onestepresults file
86          int nElem = noElem;//No. of elements in the onestep file
87          int type = 3;//top/bottom selector variable (0 = top, 1=bottom) initiate at
                value different than 1,0
88          int x = 0;//Index identified for specific String
89          String[] onestep = null;//Onestepresults string array initialiser
90          List<String> onestepList = new ArrayList<String>();     //Onestepresults in
                list format
91          int c = 4;//Counter used to filter top and bottom strains from onestep[]
92          String[] StrainsString = new String[nElem];//Strains in String format
93          int[] Operational = new int[nElem];//Operational sign checker (required to
                read the file)
94          String[][] StrainSplit = new String[nElem][6];//String split into the 6
                strain components for the top of the blank
95          int r = 0;//start character used in split
96          int s = 9;//end character used if first operational sign is negative
97          int interval = 10;//character split interval
98          Double format = new Double("6.35");//Double format used in conversion from
                String to double
99          double[][] StrainDouble = new double[nElem][6];//Strains in desired surface
                of the blank, in double format
100
101         //CHOOSE TOP/BOTTOM SURFACE
102         if (selector.equals("top")) {
103             type = 0;
104         }
105         if (selector.equals("bottom")) {
106             type = 1;
107         }
108
109         //READ ONESTEPRESULTS AND DETECT *INITIAL_STRAIN_SHELL INDEX
110         try {
111             FileInputStream file = new FileInputStream(input);
112             DataInputStream data_input = new DataInputStream(file);
113             BufferedReader buffer = new BufferedReader(new InputStreamReader(
                    data_input));
114             String str_line;
115             while ((str_line = buffer.readLine()) != null) {
116                 str_line = str_line.trim();
117                 if ((str_line.length() != 0)) {
118                     onestepList.add(str_line);
119                 }
120             }
121             onestep = (String[]) onestepList.toArray(new String[onestepList.size()])
                    ; //Convert list to array (each index correspond to a line in the
                    onestepresult file)
122         } catch (IOException e) {
123         }
124         try {
125             x = grepLineNumber(input, "*INITIAL_STRAIN_SHELL"); //Identify index of
                    *INITIAL_STRAIN_SHELL
126         } catch (IOException e) {
```

```
127         }
128
129         //CREATE STRING ARRAYS FOR THE STRAINS (STRING)
130         if (type == 0) {
131             StrainsString[0] = onestep[x + 2];
132         } else {
133             StrainsString[0] = onestep[x + 1];
134         }
135         for (int k = 1; k < nElem; k++) {
136             if (type == 0) {
137                 StrainsString[k] = onestep[x + 1 + c];
138             } else {
139                 StrainsString[k] = onestep[x + c];
140             }
141             c = c + 3;
142         }
143
144         //SPLIT STRINGS INTO THE 6 STRAIN COMPONENTS FOR TOP AND BOTTOM
145         //Check operational sign (necessary as the substring commmand can't handle
                empty char. in positive strains)
146         for (int f = 0; f < nElem; f++) {
147             if (StrainsString[f].substring(0, 1).equals("-")) {
148                 Operational[f] = 1;
149             }
150         }
151
152         //Split Array into 6 components
153         for (int k = 0; k < nElem; k++) {
154             if (Operational[k] == 1) {
155                 for (int i = 0; i < 6; i++) {
156                     StrainSplit[k][i] = StrainsString[k].substring(r, r + interval);
                            //split
157                     r += interval; //Iterate split start character
158                 }
159             } else {
160                 StrainSplit[k][0] = StrainsString[k].substring(r, interval - 1);
161                 for (int i = 1; i < 6; i++) {
162                     StrainSplit[k][i] = StrainsString[k].substring(s, s + interval);
                            //split
163                     s += interval; //Iterate split start character
164                 }
165             }
166             r = 0;
167             s = 9;
168         }
169
170         //CONVERT STRINGS INTO DOUBLE VALUE
171         for (int k = 0; k < nElem; k++) {
172             for (int i = 0; i < 6; i++) {
173                 StrainDouble[k][i] = format.parseDouble(StrainSplit[k][i]);
174             }
175         }
176         System.out.println("ReadOneStep done");
```

```
177         return StrainDouble;
178     }
179     //End of method
180 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
181     // Method for appending strains in onestepresult to a structured element array
182     public double[][][] StrainToShell(String surface) {
183 
184         //METHOD DESCRIPTION:
185         /* The method is capable of appending strains in the format (Array[element
                ID][strain component]) to an ordered m by n array  by performing the
                following step:
186          * - Create ordered element array - [m][n] which corresponds to spatial
                dimensions [x][y] of a square blank
187          * - Read strains from onestepresult
188          * - Combine strains with ordered element array [m][n][strain compontent]
189          * - Output ordered array with strains components
190          */
191         System.out.println("Starting StrainToShell");
192         //Variables used in the method
193         String DesiredSurface = surface;//Inspected surface of the model
194         int ex = 40;//Elements in the X direction
195         int ey = 40;//Elements in the Y direction
196         double[][] C = new double[ex][ey];//Array holding the ordered element ID's
197         int nElem = 1600;//No. of elements in the shell model
198         double[][] StrainDouble = new double[nElem][6];//Strains in the desired
                surface of the blank in double format
199         int ElstartID = 1;//Start element ID for the shell model
200         int ElId = ElstartID;//Iteration counter to loop through elements
201         double[][][] StrainFinal = new double[ex][ey][6];//Strains ordered with
                respect to the element ID
202 
203         //READ ELEMENT CENTROID FILE AND PLACE IN ARRAY C[][]
204         double A[][] = ReadFile("centroidsblank.data", " ");//Centroid coordinates
                of flat model
205         ArraySort(A, 1, 2);//Sorting Array with respect to x and y coordinates of
                the centroid
206         for (int m = 0; m < ex; m++) {
207             for (int n = 0; n < ey; n++) {
208                 int a = n + ex * m;//Integer that allows me to scan entire array
                        length of A[][]
209                 C[m][n] = A[a][0];//Generate array with ordered element ID's
210             }
211         }
212 
213         //READ STRAIN FROM ONESTEPRESULTS
214         StrainDouble = ReadOnestep(DesiredSurface, "onestepresult", nElem);
215 
216         //COMBINE STRAIN VALUES WITH THE ORDERED ELEMENT ARRAY
217         while (ElId <= nElem) {
218             for (int z = 0; z < ex; z++) {
219                 for (int y = 0; y < ey; y++) {
220                     if (C[z][y] == ElId) {
221                         for (int i = 0; i < 6; i++) {
```

```
                            StrainFinal[z][y][i] = StrainDouble[ElId - 1][i];
                        }
                    }
                }
            }
            ElId++;
        }
        System.out.println("StrainToShell done");
        return StrainFinal;
    }
    //End of method
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    // Method for calculating size and orientation of minimum principal strain in a
       strain field
    public double[][][] PrincipalStrain(double[][][] Strains) {

        //METHOD DESCRIPTION:
        /* The method is capable of converting strains in format [m][n][strain
           components] to the size of minimum principal strain and orientaion of
           minimum principal strain (2D) in each element by performing the
           following step:
         * - Calculate size of minimum principal strain in each element
         * - Calculate orientation of minimum principal in each element
         * - Output ordered array [m][n][size / orientation] with size of minimum
             principal strains and orientation of minimum principal strains
         */
        System.out.println("Starting PrincipalStrain");
        //Variables used in the method
        double[][][] StrainComps = Strains;//All strain components in the desired
           surface of the shell model
        int ex = 40;//Elements in the X direction
        int ey = 40;//Elements in the Y direction
        double[][][] PrincipalStrain = new double[ex][ey][2];//Size and orientation
           of minimum principal strains ordered with respect to the element ID

        //CALCULATE SIZE OF MINIMUM PRINCIPAL PRINCIPAL STRAIN
        for (int x = 0; x < ex; x++) {
            for (int y = 0; y < ey; y++) {
                PrincipalStrain[x][y][0] = (StrainComps[x][y][0] + StrainComps[x][y
                    ][1]) / 2 - Math.sqrt(Math.pow((StrainComps[x][y][0] -
                    StrainComps[x][y][1]) / 2, 2) + Math.pow(StrainComps[x][y][3],
                    2));
            }
        }

        //CALCULATE ORIENTATION OF MINIMUM PRINCIPAL STRAIN
        for (int x = 0; x < ex; x++) {
            for (int y = 0; y < ey; y++) {
                if (StrainComps[x][y][3] < 0 && StrainComps[x][y][0] < StrainComps[x
                    ][y][1]) {
                    PrincipalStrain[x][y][1] = 0.5 * Math.atan((2 * StrainComps[x][y
                        ][3]) / (StrainComps[x][y][0] - StrainComps[x][y][1]));
                }
```

```
263                 if (StrainComps[x][y][3] > 0 && StrainComps[x][y][0] < StrainComps[x
                        ][y][1]) {
264                     PrincipalStrain[x][y][1] = (Math.PI) + 0.5 * Math.atan((2 *
                            StrainComps[x][y][3]) / (StrainComps[x][y][0] - StrainComps[
                            x][y][1]));
265                 }
266                 if (StrainComps[x][y][3] > 0 && StrainComps[x][y][0] > StrainComps[x
                        ][y][1]) {
267                     PrincipalStrain[x][y][1] = (Math.PI / 2.0) + 0.5 * Math.atan((2
                            * StrainComps[x][y][3]) / (StrainComps[x][y][0] -
                            StrainComps[x][y][1]));
268                 }
269                 if (StrainComps[x][y][3] < 0 && StrainComps[x][y][0] > StrainComps[x
                        ][y][1]) {
270                     PrincipalStrain[x][y][1] = (Math.PI / 2.0) + 0.5 * Math.atan((2
                            * StrainComps[x][y][3]) / (StrainComps[x][y][0] -
                            StrainComps[x][y][1]));
271                 }
272             }
273         }
274         System.out.println("PrincipalStrain done");
275         return PrincipalStrain;
276     }
277     //End of method
278 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
279     // Method for producing a data file used to plot a vector plot in gnuplot
280     public void VectorPlot(double[][][] Principal) {
281 
282         //METHOD DESCRIPTION:
283         /* The method is capable of converting the size and orientation of minimum
                principal strains into a vector plot data file by performing the
                following step:
284          * - Calculate scaling of vector size (scaling determined from normalised
                absolute size of minimum principal strain)
285          * - Calculate component in the vector plot file
286          * - Output data
287          */
288         System.out.println("Starting VectorPlot");
289         //Variables used in the method
290         double[][][] PrincipalDirections = Principal;//Size and orientation of
                minimum principal strains, ordered with respect to the element ID
291         int ex = 40;//Elements in the X direction
292         int ey = 40;//Elements in the Y direction
293         double lengthx = 30;//Original extent of the blank in the X direction
294         double lengthy = 30;//Original extent of the blank in the Y direction
295         int nElem = 1600;//No. of element in the shell model
296         double[][] PlotData = new double[nElem * 2][4];//All data necesarry to plot
                vectorplot in gunplot (x,y,dx,dy), each line corresponds to [start x,
                start y, length x, length y] of each vector
297         double[][] Abs = new double[ex][ey];//Absolute size of minimum principal
                strains - used for scaling vector size
298         double[][] Norm = new double[ex][ey];//Normalised size of principal strains
                - used for scaling vector size
```

```
299         double min;//Minimum absolute size of the minimum principal strain - used to
                normalise
300         double max;//Maximum absolute size of the minimum principal strain - used to
                normalise
301         int a = 0;//Counter used generate correct ordering of x,y,dx,dy in PlotData
                [][]
302         int b = 0;//Counter used generate correct ordering of x,y,dx,dy in PlotData
                [][]
303         int c = 0;//Counter used generate correct ordering of x,y,dx,dy in PlotData
                [][]
304
305         //CALCULATE SCALING OF VECTOR W.R.T. NORMALISED SIZE OF MINIMUM PRINCIPAL
                STRAIN
306         //Determine absolute strain and identify minimum and maximum value for
                normalisation
307         for (int x = 0; x < ex; x++) {
308             for (int y = 0; y < ey; y++) {
309                 Abs[x][y] = Math.abs(PrincipalDirections[x][y][0]);
310             }
311         }
312         min = max = Abs[0][0];
313         for (int x = 0; x < ex; x++) {
314             for (int y = 0; y < ey; y++) {
315                 if (Abs[x][y] < min) {
316                     min = Abs[x][y];
317                 }
318                 if (Abs[x][y] > max) {
319                     max = Abs[x][y];
320                 }
321             }
322         }
323
324         //Normalise strain (values between 0-1)
325         for (int x = 0; x < ex; x++) {
326             for (int y = 0; y < ey; y++) {
327                 Norm[x][y] = (Math.abs(PrincipalDirections[x][y][0]) - min) / (max -
                        min);
328             }
329         }
330
331         //CALCULATE X,Y,DX,DY
332         //x and y are placed in the centroid of the element - Two vectors are
                plotted from each centroid one in the correct orientation of minimum
                principal strain and one rotated 180 degress around the element
                centroids
333         //this is necessary to achieve a better visual representation of the
                orientation of minimum principal strain in the element
334         //Correct orientation (Note that all dx,dy are scaled further with a 0.3
                value, this ensures that all arrows fit within the element size of 0.75
                mm x 0.75mm
335         for (int i = 0; i < nElem; i++) {
336             if ((i) % 40 == 0 && i != 0) {
337                 a++;
```

```
338             }
339             if ((i) % 40 == 0 && i != 0) {
340                 c++;
341             }
342             b = i - 40 * a;
343             PlotData[i][0] = lengthx / ex * 0.5 + lengthx / ex * (b);
                                //X
344             PlotData[i][1] = lengthy / ey * 0.5 + lengthx / ex * (c);
                                //Y
345             PlotData[i][2] = Math.cos(PrincipalDirections[b][c][1]) * Norm[b][c] *
                    0.3; //DX - correct direction
346             PlotData[i][3] = Math.sin(PrincipalDirections[b][c][1]) * Norm[b][c] *
                    0.3; //DY- correct direction
347         }
348         a = b = c = 0;
349         //Rotated orientation
350         for (int i = 1600; i < nElem * 2; i++) {
351             if ((i) % 40 == 0 && i != 1600) {
352                 a++;
353             }
354             if ((i) % 40 == 0 && i != 1600) {
355                 c++;
356             }
357             b = i - 40 * a - 1600;
358             PlotData[i][0] = lengthx / ex * 0.5 + lengthx / ex * (b);
                                //X
359             PlotData[i][1] = lengthy / ey * 0.5 + lengthx / ex * (c);
                                //Y
360             PlotData[i][2] = -Math.cos(PrincipalDirections[b][c][1]) * Norm[b][c] *
                    (0.3); //DX - negative direction
361             PlotData[i][3] = -Math.sin(PrincipalDirections[b][c][1]) * Norm[b][c] *
                    (0.3); //DY- negative direction
362         }
363         ToFile(PlotData, "debugplot/vector.data");
364         System.out.println("VectorPlot done");
365     }
366     //End of method
367 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
368     // Method for appending centroid coordinates to structured element array
369     public double[][][] CenterAppender(String centroidfile) {
370 
371         //METHOD DESCRIPTION:
372         /* The method is capable appending centroid coordinates to an ordered
                element array by performing the following steps:
373          * - Create ordered element array - [m][n] corresponds to spatial dimensions
                 [x][y]
374          * - Append coordinates to array [m][n][coordinates]
375          * - Output array
376          */
377         System.out.println("Starting CenterAppender");
378         //Variables used in the method
379         int ex = 40;//Elements in the X direction
380         int ey = 40;//Elements in the Y direction
```

```
381         double[][] C = new double[ex][ey];//Array holding the ordered element ID's
382         double[][][] CenterCoord = new double[ex][ey][4];//Array with ordered
                centroid coordinates
383         int ElstartID = 1;//Start element ID
384         int ElId = ElstartID;//Iteration counter to loop through elements
385         int nElem = 1600;//No. of elements
386 
387         //READ ELEMENT CENTROID FILE AND PLACE IN ORDERED ARRAY C[][]
388         double A[][] = ReadFile("centroidsblank.data", " ");
389         ArraySort(A, 1, 2);//Sorting Array with respect to x and y coordinates of
                the centroid
390         for (int m = 0; m < ex; m++) {
391             for (int n = 0; n < ey; n++) {
392                 int a = n + ex * m;//Integer that allows for scanning the entire
                        array lengh of A[][]
393                 C[m][n] = A[a][0];//Generate array with ordered element ID's
394             }
395         }
396 
397         //COMBINE COORDINATES WITH THE ORDERED ELEMENT ARRAY
398         double B[][] = ReadFile(centroidfile, " ");
399         while (ElId <= nElem) {
400             for (int z = 0; z < ex; z++) {
401                 for (int y = 0; y < ey; y++) {
402                     if (C[z][y] == ElId) {
403                         CenterCoord[z][y][0] = ElId;
404                         for (int i = 1; i < 4; i++) {
405                             CenterCoord[z][y][i] = B[ElId - 1][i];
406                         }
407                     }
408                 }
409             }
410             ElId++;
411         }
412         System.out.println("CenterAppender done");
413         return CenterCoord;
414     }
415     //End of method
416 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
417     // Method for mapping strains in the desired geometry to the ordered array
418     public double[][][] MapDesiredToShell(String surface, String flip, String
            geometry) {
419 
420         //METHOD DESCRIPTION:
421         /* The method is capable of mapping the strains from the desired geometry to
                the shell model by performing the following step:
422          * - Read centroid data for flattened desired geometry and shell model of
                blank
423          * - Find "best match" element ID in desired geometry with shortest in-plane
                centroid distance to the shell model
424          * - Map strain components from "best match" to the shell model
425          * - Output array [m][n][mapped strain components]
426          */
```

```
427         System.out.println("Starting MapDesiredToShell");
428         //Variables used in this method
429         String DesiredSurface = surface;//Inspected surface of the model
430         int exs = 40;//Elements in the X direction (40x40 shell model of blank)
431         int eys = 40;//Elements in the Y direction (40x40 shell model of blank)
432         double[][] C = new double[exs][eys];//Ordered array with element ID
433         int nElem = exs * eys;//No. of elements in shell model of blank
434         int ElstartID = 1;//Start element ID
435         int ElId = ElstartID;//Iteration counter to loop through elements
436         double dist = 0;//Euclidean distance, in XY plane, between centroid in 40x40
                shell model of blank and the flattened desired geometry
437         double minDist = 10000;//Variable storing the shortest distance between to
                element centroids (Must be initiated at a value higher than the minimum
                distance between elements)
438         int[] minDistElId = new int[nElem];//Element IDs corresponding to shortes
                distances
439         double[][][] StrainFinal = new double[exs][eys][6];//Mapped strains ordered
                with respect to the element ID
440
441         //READ CENTROID COORDINATES FOR DESIRED GEOMETRY (A[][]) AND  SHELL MODEL OF
                BLANK (B[][])
442         double A[][] = ReadFile("onestepcentroid.data", " ");//Centroids of the fine
                meshed desired geometry (flat model)
443         double B[][] = ReadFile("centroidsblank.data", " ");//Centroid of the 40x40
                shell model of blank (not formed)
444
445         //FIND BEST MATCH CENTROID (SHORTEST EUCLIDEAN DISTANCE IN XY PLANE) BETWEEN
                B[][] AND A[][]
446         while (ElId <= nElem) {
447             for (int i = 0; i < A.length; i++) {
448                 if (flip == "flip") {
449                     dist = Math.sqrt(Math.pow(B[ElId - 1][1] - (A[i][1]), 2) + Math.
                        pow(B[ElId - 1][2] - (-A[i][2]), 2)); //Euclidean distance
                        in XY plane
450                 } else {
451                     dist = Math.sqrt(Math.pow(B[ElId - 1][1] - (A[i][1]), 2) + Math.
                        pow(B[ElId - 1][2] - (A[i][2]), 2));
452                 }
453                 if (dist < minDist) {
454                     minDist = dist; //Save shortest distance
455                     minDistElId[ElId - 1] = i + 1;
456                 }
457             }
458             minDist = 10000; //reset distance variable
459             ElId++;
460         }
461         ElId = ElstartID;
462
463         //SORT AND ORDER ELEMENT ARRAY
464         ArraySort(B, 1, 2); //Sorting Array with respect to X and Y coordinates of
                the centroid
465         for (int m = 0; m < exs; m++) {
466             for (int n = 0; n < eys; n++) {
```

```
467                 int a = n + exs * m; //Integer that allows scanning of the entire
                        array lengh of A[][]
468                 C[m][n] = B[a][0]; //Generate array with ordered element ID's
469             }
470         }
471
472         //READ STRAIN COMPONENTS
473         double strains[][] = ReadOnestep(DesiredSurface, "onestepresult", A.length);
474
475         //COMBINE STRAIN VALUES WITH THE ORDERED ELEMENT ARRAY
476         while (ElId <= nElem) {
477             for (int z = 0; z < exs; z++) {
478                 for (int y = 0; y < eys; y++) {
479                     if (C[z][y] == ElId) {
480                         for (int i = 0; i < 6; i++) {
481                             StrainFinal[z][y][i] = strains[minDistElId[ElId - 1] -
                                    1][i]; //Find the strain value at the element
                                    closest to the centroid of the 40x40 shell model of
                                    the blank
482
483                         }
484                     }
485                 }
486             }
487             ElId++;
488         }
489         System.out.println("MapDesiredToShell done");
490         return StrainFinal;
491     }
492     //End of method
493 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
494     // Method for identifying index of specific string in data file
495     public int grepLineNumber(String file, String word) throws IOException {
496         //METHOD DESCRIPTION:
497         /* The method is capable of identifying the index of a specific string in
                data file by performing the following steps:
498          * - scan file until string is met and store index value
499          */
500
501         System.out.println("Starting grepLineNumber");
502         //Variables used in this method
503         String line;//String storing a line from the file
504         int lineNumber = 0;//Line number / index of line holding desired string (
                word)
505
506         //READ FILE UNTIL THE STRING IS DETECTED AND RETURN THE INDEX
507         BufferedReader buf = new BufferedReader(new InputStreamReader(new
                DataInputStream(new FileInputStream(file))));
508         while ((line = buf.readLine()) != null) {
509             lineNumber++;
510             if (word.equals(line)) {
511                 return lineNumber;
512             }
```

```
513         }
514         System.out.println("grepLineNumber done");
515         return -1;
516     }
517     //End of method
518 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
519     // Method for going from 3D to 2D array and manipulation
520     public double[][] getComponent(double[][][] Data, int component, double addition
        ) {
521         //METHOD DESCRIPTION:
522         /* The method is capable of converting a 3D array to 2D ,filled with one of
               the components in the 3D arry, by performing the following steps:
523          * - Place Data[x][y][component] in ComponentArray[selected component at x][
                selected component at y]
524          */
525 
526         //Variables used in this method
527         int ex = 40;//Elements in the X direction
528         int ey = 40;//Elements in the Y direction
529         double[][] ComponentArray = new double[ex][ey]; //Selected 2D array in
               ordered array
530 
531         for (int x = 0; x < ex; x++) {
532             for (int y = 0; y < ey; y++) {
533                 ComponentArray[x][y] = Data[x][y][component] + addition; //Addition
                        can be used to make an offset in the selected component if
                        desired
534             }
535         }
536         return ComponentArray;
537     }
538     //End of method
539 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
540     // Method for performing thresholding on size of minimum principal strain
541     public double[][] Thresh(double[][][] Strain, double ThreshPercentage) {
542         //METHOD DESCRIPTION:
543         /* The method is capable of thresholding the size of the minimum principal
               strains [][] by performing the following steps:
544          * - Sort strains in ordered array
545          * - Identify the threshold limit with respect to threshold percentage
546          * - Perform thresholding
547          * - Output ordered array [m][n] with entry = 1 if the threshold is
                satisfied and 0 if not
548          */
549 
550         System.out.println("Starting Thresh");
551         //Variables used in this method
552         double[][] ThreshArray = new double[Strain.length][Strain.length];//Array
               filled with ordered 1's and 0's (1 if >= thresh else 0)
553         int i = 0;//Counter used to generate sorted array
554         double[] Sorted = new double[Strain.length * Strain.length];//Array sorted
               with sorted values from lowest to highest
555         int ThreshIndex = 0;//Index defining the index in the sorted array
```

```
            corresponding to the threshPercentage
556         double ThreshValue = 0.0;//Size of minimum principal strain value defining
                threshold limit
557
558         //GENERATE AND SORT ALL STRAINS IN 1D ARRAY
559         for (int x = 0; x < Strain.length; x++) {
560             for (int y = 0; y < Strain.length; y++) {
561                 Sorted[i] = Strain[x][y][0];
562                 i = i + 1;
563             }
564         }
565         Arrays.sort(Sorted);
566
567         //DETERMINE THRESHOLD LIMIT WITH RESPECT TO THRESHOLD PERCENTAGE
568         ThreshIndex = (int) Math.round(((Strain.length * Strain.length) / 100) *
                ThreshPercentage);
569         ThreshValue = Sorted[ThreshIndex - 1];
570
571         //PERFORM THRESHOLDING OF THE SIZE OF MINIMUM PRINCIPAL STRAIN
572         for (int x = 0; x < Strain.length; x++) {
573             for (int y = 0; y < Strain.length; y++) {
574                 if (Strain[x][y][0] <= ThreshValue) {
575                     ThreshArray[x][y] = 1.0;
576                 }
577             }
578         }
579         ToFile(ThreshArray, "debugplot/thresh.data");
580         System.out.println("Thresh done");
581         return ThreshArray;
582     }
583     //End of method
584 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
585     // Method for performing pathplanning
586     public int[][] ElementPath(double[][] Thresh, double[][][] PrinData, double
            angleaccept, String centroid) {
587         //METHOD DESCRIPTION
588         /* The method is capable of determining scan paths by performing the scan
                path algorithm described in the main report.
589          */
590
591         System.out.println("Starting ElementPath");
592         //Variables used in this method
593         int ex = 40;//Elements in the X direction
594         int ey = 40;//Elements in the Y direction
595         double min = 1000;//minimum size of minimum principal strain (must be
                initiated at a value above 0)
596         int startx = 0;//Starting X point of the scan path algorithm
597         int starty = 0;//Starting Y point of the scan path algorithm
598         double[][] orientation = getComponent(PrinData, 1, 0);//Orientation of
                minimum principal strains field
599         double[][] grassFire = new double[ex][ey];//Burned elements field (visited
                elements = 1, non visited elements = 0)
600         double[][][] Vectororientation = new double[ex][ey][2];//A vector
```

```
            representation of the orientation of principal strain
601         double[][][] coords = CenterAppender(centroid);//Coordinates of element
                centroids
602         double[][] V = new double[8][2];//Search stencil representation of movement
                options vectors in current point
603         double[] Angles = new double[8];//Search stencil representation of the angle
                 between the orienation of principals strain and movement option vector
604         int option = 0;//trigger, detecting when the scan path algorithm is out of
                movement options
605         ArrayList<Integer> listxf = new ArrayList<Integer>();//X coordinates in the
                1st search of the scan path algorithm
606         ArrayList<Integer> listyf = new ArrayList<Integer>();//Y coordinates in the
                1st search of the scan path algorithm
607         ArrayList<Integer> listxb = new ArrayList<Integer>();//X coordinates in the
                2nd search of the scan path algorithm
608         ArrayList<Integer> listyb = new ArrayList<Integer>();//Y coordinates in the
                2nd search of the scan path algorithm
609         int dircount = 0;//Counter for the scan path algorithm
610         int a = 0;//Value used to check if the scan path algorithm has run to a dead
                 end (if equal to 7 = dead end)
611         int dir = 8;//variable used to determine appropriate redced search stencil
612         int length = 8;//size of the  search stencil (3 points or 8 points)
613         int temp = 0;//temporary variable uses to select appropriate search stencil
614         int[] stencilx = {-1, 0, 1, -1, 1, -1, 0, 1};//X stencil updated during
                iterations
615         int[] stencily = {1, 1, 1, 0, 0, -1, -1, -1};//Y stencil updated during
                iterations
616
617         //IDENTIFY MOST COMPRESSIVE STRAIN I.E. START POINT OF SCAN PATH ALGORITHM)
618         for (int x = 1; x < ex - 1; x++) {
619             for (int y = 1; y < ey - 1; y++) {
620                 if (Thresh[x][y] == 1 && PrinData[x][y][0] < min) {
621                     min = PrinData[x][y][0];
622                     startx = x;
623                     starty = y;
624                 }
625             }
626         }
627
628         System.out.println("min prin strain " + min + " start(" + startx + "," +
                starty + ")");
629
630         //START SCANPATH ALGORITHM
631         while (dircount < 2) {
632             int pointx = startx;
633             int pointy = starty;
634             if (dircount == 1) {
635                 option = 0;
636             }
637
638             //CREATE UNITIY VECTOR PRINCIPAL ORIENTATION FIELD (USED TO DETERMINE
                    ANGLE BETWEEN MOVEMENT OPTION VECTORS ORIENTATION OF MINUMUM
                    PRINCIPAL STRAIN)
```

```
639             for (int i = 0; i < ex; i++) {
640                 for (int j = 0; j < ey; j++) {
641                     Vectororientation[i][j][0] = Math.cos(orientation[i][j]);
642                     Vectororientation[i][j][1] = Math.sin(orientation[i][j]);
643                 }
644             }
645 
646             int[] stencilxs = {-1, 0, 1, -1, 1, -1, 0, 1};//X stencil at start point
                     (cannot be moved)
647             int[] stencilys = {1, 1, 1, 0, 0, -1, -1, -1};//Y stencil at start point
                     (cannot be moved)
648             stencilx = stencilxs;//X stencil updated during iterations
649             stencily = stencilys;//Y stencil updated during iterations
650 
651             //START SEARCHING
652             while (option < 1 && pointx < orientation.length - 1 && pointy <
                    orientation[0].length - 1 && pointx > 0 && pointy > 0) {
653         //SELECT APPROPIATE STENCIL FOR SEARCH
654                 //The appropriate search stencil is selected in the initial
                        iteration the 8pt stencil is selected in the remain iterations a
                         reduced search stencil is selected with respect to the movement
                         option vector
655                 //selected in the previous iteration.
656                 if (dir == 0) {//upwards-left - reduced search stencil
657                     stencilx[0] = -1;
658                     stencilx[1] = 0;
659                     stencilx[2] = -1;
660                     stencily[0] = 1;
661                     stencily[1] = 1;
662                     stencily[2] = 0;
663                     length = 3;
664                 }
665                 if (dir == 1) {//upwards - reduced search stencil
666                     stencilx[0] = -1;
667                     stencilx[1] = 0;
668                     stencilx[2] = 1;
669                     stencily[0] = 1;
670                     stencily[1] = 1;
671                     stencily[2] = 1;
672                     length = 3;
673                 }
674                 if (dir == 2) {//upwards-right - reduced search stencil
675                     stencilx[0] = 0;
676                     stencilx[1] = 1;
677                     stencilx[2] = 1;
678                     stencily[0] = 1;
679                     stencily[1] = 1;
680                     stencily[2] = 0;
681                     length = 3;
682                 }
683                 if (dir == 3) {//left - reduced search stencil
684                     stencilx[0] = -1;
685                     stencilx[1] = -1;
```

```
686                     stencilx[2] = -1;
687                     stencily[0] = 1;
688                     stencily[1] = 0;
689                     stencily[2] = 1;
690                     length = 3;
691                 }
692                 if (dir == 4) {//right - reduced search stencil
693                     stencilx[0] = 1;
694                     stencilx[1] = 1;
695                     stencilx[2] = 1;
696                     stencily[0] = 1;
697                     stencily[1] = 0;
698                     stencily[2] = -1;
699                     length = 3;
700                 }
701                 if (dir == 5) {//downwards-left - reduced search stencil
702                     stencilx[0] = -1;
703                     stencilx[1] = -1;
704                     stencilx[2] = 0;
705                     stencily[0] = 0;
706                     stencily[1] = -1;
707                     stencily[2] = -1;
708                     length = 3;
709                 }
710                 if (dir == 6) {//downwards - reduced search stencil
711                     stencilx[0] = -1;
712                     stencilx[1] = 0;
713                     stencilx[2] = 1;
714                     stencily[0] = -1;
715                     stencily[1] = -1;
716                     stencily[2] = -1;
717                     length = 3;
718                 }
719                 if (dir == 7) {//downwards-right - reduced search stencil
720                     stencilx[0] = 1;
721                     stencilx[1] = 0;
722                     stencilx[2] = 1;
723                     stencily[0] = 0;
724                     stencily[1] = -1;
725                     stencily[2] = -1;
726                     length = 3;
727                 }
728                 if (dir == 8) {//8 point search stencil
729                     length = 8;
730                     stencilx = stencilxs;
731                     stencily = stencilys;
732                 }
733 
734                 //DETERMINE CURRENT MOVEMENT OPTION VECTORS BASED ON ACTIVE STENCIL
735                 for (int i = 0; i < length; i++) {
736                     for (int k = 1; k < 3; k++) {
737                         V[i][k - 1] = coords[pointx + stencilx[i]][pointy + stencily
                                [i]][k] - coords[pointx][pointy][k];
```

```
738                     }
739                 }
740
741                 //DETERMINE ANGLES BETWEEN MOVEMENT OPTIONS VECTOR AND ORIENTATION
                        OF MINIMUM PRINCIPAL STRAIN  (DOTPRODUCT)
742                 for (int i = 0; i < length; i++) {
743                     Angles[i] = Math.acos((Vectororientation[pointx + stencilx[i]][
                            pointy + stencily[i]][0] * V[i][0] + Vectororientation[
                            pointx + stencilx[i]][pointy + stencily[i]][1] * V[i][1]) /
                            (Math.sqrt(Math.pow(Vectororientation[pointx + stencilx[i]][
                            pointy + stencily[i]][0], 2) + Math.pow(Vectororientation[
                            pointx + stencilx[i]][pointy + stencily[i]][1], 2)) * Math.
                            sqrt(Math.pow(V[i][0], 2) + Math.pow(V[i][1], 2)))) * 180.0
                            / Math.PI;
744                 }
745
746                 //SELECT MOVEMENT OPTION WHICH IS CLOSEST TO PERPENDICULAR AND
                        SATISFIES CRITERIA WITH RESPECT TO THRESHOLD, PERPENDICULARITY
                        AND POINT REPETITION
747                 double minangle = 180;//Initialised minimum angle, must be greater
                        than 90 deg.
748                 int minindex = 8;//Initialised mininum index must be equal to 8
749                 int bestx = pointx;//initialise best x direction
750                 int besty = pointy;//initialise best y direction
751                 grassFire[bestx][besty] = 1;//Burn start point
752                 for (int i = 0; i < length; i++) {
753                     if (Math.abs(90.0 - Angles[i]) <= minangle && Thresh[pointx +
                            stencilx[i]][pointy + stencily[i]] == 1 && grassFire[pointx
                            + stencilx[i]][pointy + stencily[i]] == 0) {
754                         minindex = i;
755                         minangle = Math.abs(90.0 - Angles[i]);
756                     }
757                 }
758                 //UPDATE TEMPORARY VARIABLE WITH THE CHOSEN MOVEMENT DIRECTION, SUCH
                         THAT A NEW APPROPRIATE SEARCH STENCIL CAN BE SELECTED FOR THE
                        FOLLOWING ITERATION
759                 if (minangle < angleaccept && a % 7 == 0) {
760                     for (int i = 0; i < 8; i++) {
761                         if (minindex == i) {
762
763                             if (dir != 8) {//for the reduced stencil temp is set
                                    equal to the correlating index in the 8pt stencil,
                                    this is a requirement of the implementation method
764                                 if (dir == 0 && i == 0) {
765                                     temp = 0; //next search stencil is the upwards-
                                            left reduced search stencil
766                                 }
767                                 if (dir == 0 && i == 1) {
768                                     temp = 1; //next search stencil is the upwards
                                            reduced search stencil
769                                 }
770                                 if (dir == 0 && i == 2) {
771                                     temp = 3; //next search stencil is the left
```

```
                reduced search stencil
772         }
773         if (dir == 1 && i == 0) {
774             temp = 0; //next search stencil is the upwards-
                    left reduced search stencil
775         }
776         if (dir == 1 && i == 1) {
777             temp = 1; //next search stencil is the upwards
                    reduced search stencil
778         }
779         if (dir == 1 && i == 2) {
780             temp = 2; //next search stencil is the upwards-
                    right reduced search stencil
781         }
782         if (dir == 2 && i == 0) {
783             temp = 1; //next search stencil is the upwards
                    reduced search stencil
784         }
785         if (dir == 2 && i == 1) {
786             temp = 2; //next search stencil is the upwards-
                    right reduced search stencil
787         }
788         if (dir == 2 && i == 2) {
789             temp = 4; //next search stencil is the right
                    reduced search stencil
790         }
791         if (dir == 3 && i == 0) {
792             temp = 0; //next search stencil is the upwards-
                    left reduced search stencil
793         }
794         if (dir == 3 && i == 1) {
795             temp = 3; //next search stencil is the left
                    reduced search stencil
796         }
797         if (dir == 3 && i == 2) {
798             temp = 5; //next search stencil is the downwards
                    -left reduced search stencil
799         }
800         if (dir == 4 && i == 0) {
801             temp = 2; //next search stencil is the upwards-
                    right reduced search stencil
802         }
803         if (dir == 4 && i == 1) {
804             temp = 4; //next search stencil is the right
                    reduced search stencil
805         }
806         if (dir == 4 && i == 2) {
807             temp = 7; //next search stencil is the downwards
                    -right reduced search stencil
808         }
809         if (dir == 5 && i == 0) {
810             temp = 3; //next search stencil is the left
                    reduced search stencil
```

```
811                                 }
812                                 if (dir == 5 && i == 1) {
813                                     temp = 5; //next search stencil is the downwards
                                        -left reduced search stencil
814                                 }
815                                 if (dir == 5 && i == 2) {
816                                     temp = 6; //next search stencil is the downwards
                                         reduced search stencil
817                                 }
818                                 if (dir == 6 && i == 0) {
819                                     temp = 5; //next search stencil is the downwards
                                        -left reduced search stencil
820                                 }
821                                 if (dir == 6 && i == 1) {
822                                     temp = 6; //next search stencil is the downwards
                                         reduced search stencil
823                                 }
824                                 if (dir == 6 && i == 2) {
825                                     temp = 7; //next search stencil is the downwards
                                        -right reduced search stencil
826                                 }
827                                 if (dir == 7 && i == 0) {
828                                     temp = 4;  //next search stencil is the right
                                        reduced search stencil
829                                 }
830                                 if (dir == 7 && i == 1) {
831                                     temp = 6; //next search stencil is the downwards
                                         reduced search stencil
832                                 }
833                                 if (dir == 7 && i == 2) {
834                                     temp = 7; //next search stencil is the downwards
                                        -right reduced search stencil
835                                 }
836                                 dir = temp;
837                             } else {
838                                 dir = i; //next search stencil is the eight point
                                    search stencil
839                             }
840                             //Update the best point identified
841                             bestx = bestx + stencilx[i];
842                             besty = besty + stencily[i];
843                         } else {
844                             a++;
845                         }
846                     }
847                     //write the best point to list
848                     if (dircount == 0) {
849                         listxf.add(bestx);
850                         listyf.add(besty);
851                     } else {
852                         listxb.add(bestx);
853                         listyb.add(besty);
854                     }
```

```
855                 } else {
856                     option = 1;
857                     System.out.println("No further movement in this direction");
858                 }
859                 //update the start point for the next iteration of the algorithm
860                 pointx = bestx;
861                 pointy = besty;
862                 grassFire[bestx][besty] = 1;//Burn visited point
863             }
864             a = 0;
865             dir = 8;
866             dircount++;
867         }
868 
869         //GENERATE ORDERED ARRAY FROM FORWARD LIST, START POINT AND BACKWARDS LIST
870         int[][] Points = new int[listxb.size() + 1 + listxf.size()][2];
871         for (int i = 0; i < listxb.size(); i++) {
872             Points[i][0] = listxb.get(listxb.size() - 1 - i);
873             Points[i][1] = listyb.get(listyb.size() - 1 - i);
874         }
875         Points[listxb.size()][0] = startx;
876         Points[listyb.size()][1] = starty;
877         for (int i = listxb.size(); i < listxb.size() + listxf.size(); i++) {
878             Points[i + 1][0] = listxf.get(i - listxb.size());
879             Points[i + 1][1] = listyf.get(i - listxb.size());
880         }
881 
882         for (int j = 0; j < Points.length; j++) {
883             System.out.println(Points[j][0] + "    " + Points[j][1]);
884         }
885 
886         System.out.println("ElementPath done");
887         return Points;
888 
889     }
890      //End of method
891 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
892     // Method for averaging two arrays
893     public double[][][] AvgArray(double[][][] top, double[][][] bottom) {
894 
895         //METHOD DESCRIPTION:
896         /* The method is capable of averaging the last components of two 3D arrays
                by performing the following steps:
897          * - Take average between first and second array for all components
898          */
899         System.out.println("Starting AvgArray");
900         //Variables used in this method
901         double[][][] Avg = new double[top.length][top.length][6];//Array with
                averaged components
902 
903         //CALCULATE AVERAGE FOR ALL COMPONENTS
904         for (int x = 0; x < top.length; x++) {
905             for (int y = 0; y < top.length; y++) {
```

```
                for (int i = 0; i < 6; i++) {
                    Avg[x][y][i] = (top[x][y][i] + bottom[x][y][i]) / 2;
                }
            }
        }
        System.out.println("AvgArray done");
        return Avg;
    }
     //End of method
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    // Method for selecting process variables
    public double[][] ElPathPara(int[][] ElPath, double[][][] Prinsur, double[][][]
        Princenter, int ControlOrMech, String strainPathPosition) {

        //METHOD DESCRIPTION:
        /* The method is capable of determining process variables for the scan path
            by performing the following steps:
         * - Calculate required bending and in-plane strain
         * - Select forming mechanism with respect to required bending and in-plane
             strain
         * - Calculate required bending and in-plane strain
         * - Select process variables with respect to selected forming mechanism and
              required bending and in-plane strain (Fraction distribution)
         */
        System.out.println("Starting ElPathPara");
        //Variables used in this method
        double[][] Parameters = new double[ElPath.length][4];//Array with scan speed
            , laser power, laser beam radius and selected surface
        double[][][] PrinMidplane = Princenter;//Size and orientation of minimum
            principal strains for the midplane strain field
        double[][][] PrinSur = Prinsur;//Size and orientation of minimum principal
            strains for the surface strain field
        double[] BendingStrain = new double[ElPath.length];//Bending strain in the
            scan path
        double[] MidStrain = new double[ElPath.length];//In-plane strain in the scan
             path
        double[] ratio = new double[ElPath.length];//ratio between bending strain
            and in-plane strain
        double MaxUM = -1000;//The maximum in-plane strain in the scan path
        double MaxTGM = -1000;//The maximum bending strain in the scan path
        double MinUM = 1000;//The minimum in-plane strain in the scan path
        double MinTGM = 1000;//The minimum bending strain in the scan path
        double UMlimit = 0.027;//Absolute value of the maximum achievable in-plane
            strain by the UM
        double TGMlimit = 0.0021;//Absolute value of the maximum achievable bending
            strain by the TGM
        double UMpowerlimit = 0.014;//Absolute value of the minimum achievable in-
            plane strain by the UM
        double TGMpowerlimit = 0.00035;//Absolute value of the minimum achievable
            bending strain by the TGM
        double V;//Scan speed
        double Vmoving; //Scan speed update value (V_new in project) used to ensure
            movement within processrange
```

```
944
945         //FIND ELEMENTS CORRESPONDING TO SCAN PATH, CALCULATE BENDING AND IN-PLANE
                STRAIN AND SPECIFY SURFACE TO BE SCANNED
946         for (int i = 0; i < ElPath.length; i++) {
947             BendingStrain[i] = PrinSur[ElPath[i][0]][ElPath[i][1]][0] - PrinMidplane
                    [ElPath[i][0]][ElPath[i][1]][0];
948             MidStrain[i] = Princenter[ElPath[i][0]][ElPath[i][1]][0];
949             if (strainPathPosition == "lower") {
950                 Parameters[i][3] = 1; //Define surface lower
951             }
952             if (strainPathPosition == "upper") {
953                 Parameters[i][3] = 0; //Define surface upper
954             }
955         }
956
957         //SELECT MECHANISM BY RATIO
958         for (int i = 0; i < ElPath.length; i++) {
959             if (Math.abs(BendingStrain[i] / MidStrain[i]) < 1.0) {
960                 ratio[i] = 0; //UM selected
961                 Parameters[i][2] = 3;//Define beam radius
962             } else {
963                 ratio[i] = 1;//TGM selected
964                 Parameters[i][2] = 1.5;//Define beam radius
965             }
966         }
967
968         if (ControlOrMech == 0) {
969             //SELECT PROCESS VARIABLES WITH RESPECT TO FORMING MECHANISM AND
                    REQUIRED STRAIN
970             for (int i = 0; i < ElPath.length; i++) {
971                 if (ratio[i] == 0) {//um selected
972                     //calc largest and smallest in-plane strain
973                     for (int j = 0; j < ElPath.length; j++) {
974                         if (Math.abs(MidStrain[j]) > MaxUM) {
975                             MaxUM = Math.abs(MidStrain[j]);
976                         }
977                         if (Math.abs(MidStrain[j]) < MinUM) {
978                             MinUM = Math.abs(MidStrain[j]);
979                         }
980                     }
981
982                     //Define scan speed
983                     if (MaxUM > UMlimit) {
984                         V = 200;
985                         Parameters[i][0] = V + (400 * (MaxUM - Math.abs(MidStrain[i
                                ])) / MaxUM);
986                     } else {
987                         V = 200 + ((UMlimit - MaxUM) / UMlimit) * 400;
988                         Vmoving = 400 - V; //Adjusted upper speed, to ensure that
                                the scan speed is always within the processrange i.e.
                                200-400mm/min
989                         Parameters[i][0] = V + (Vmoving * (MaxUM - Math.abs(
                                MidStrain[i])) / MaxUM);
```

```
990                     }
991
992                     //Define power (kill if forming is low)
993                     if (MinUM < UMpowerlimit) {
994                         Parameters[i][1] = 0;
995                     } else {
996                         Parameters[i][1] = 0.38;
997                     }
998                     System.out.println("UM with V= " + Parameters[i][0] + "     P =
                        " + Parameters[i][1] + "    r = " + Parameters[i][2] + "
                        surface (0=t,1=b) = " + Parameters[i][3]);
999                 }
1000
1001                if (ratio[i] == 1) {//tgm selected
1002                    //calc largest and smallest bend strain
1003                    for (int j = 0; j < ElPath.length; j++) {
1004                        if (Math.abs(BendingStrain[j]) > MaxTGM) {
1005                            MaxTGM = Math.abs(BendingStrain[j]);
1006                        }
1007                        if (Math.abs(BendingStrain[j]) < MinTGM) {
1008                            MinTGM = Math.abs(BendingStrain[j]);
1009                        }
1010                    }
1011
1012                    //Define scan veloity
1013                    if (MaxTGM > TGMlimit) {
1014                        V = 2750;
1015                        Parameters[i][0] = V + (4250 * (MaxTGM - Math.abs(
                            BendingStrain[i])) / MaxTGM);
1016                    } else {
1017                        V = 2750 + ((TGMlimit - MaxTGM) / TGMlimit) * 4250;
1018                        Vmoving = 7000 - V; //Adjusted upper speed, to ensure that
                            the scan speed is always within the processrange i.e.
                            2750-7000mm/min
1019                        Parameters[i][0] = V + (Vmoving * (MaxTGM - Math.abs(
                            BendingStrain[i])) / MaxTGM);
1020                    }
1021
1022                    Parameters[i][1] = 0.38;
1023
1024                    Define power (kill if forming  {
1025                        is
1026                    }
1027                    low)
1028                    if (MinTGM < TGMpowerlimit) {
1029                        Parameters[i][1] = 0;
1030                    } else {
1031                        Parameters[i][1] = 0.38;
1032                    }
1033                    System.out.println("TGM with V= " + Parameters[i][0] + "    P =
                        " + Parameters[i][1] + "    r = " + Parameters[i][2] + "
                        surface (0=t,1=b) = " + Parameters[i][3]);
1034                }
```

```
            }
        } else {
            if (ControlOrMech == 1) { //Pure TGM
                System.out.println("Pure TGM without control selected");
                for (int j = 0; j < ElPath.length; j++) {
                    Parameters[j][0] = 2750;
                    Parameters[j][1] = 0.38;
                    Parameters[j][2] = 1.5;
                }
            }
            if (ControlOrMech == 2) { //Pure UM
                System.out.println("Pure UM without control selected");
                for (int j = 0; j < ElPath.length; j++) {
                    Parameters[j][0] = 200;
                    Parameters[j][1] = 0.38;
                    Parameters[j][2] = 3.0;
                }
            }
            if (ControlOrMech == 3) { //Combined UM/TGM without control
                for (int j = 0; j < ElPath.length; j++) {
                    System.out.println("Combined TGM/UM without control selected");
                    if (ratio[j] == 0) {
                        Parameters[j][0] = 200;
                        Parameters[j][1] = 0.38;
                        Parameters[j][2] = 3.0;
                    }
                    if (ratio[j] == 1) {
                        Parameters[j][0] = 2750;
                        Parameters[j][1] = 0.38;
                        Parameters[j][2] = 1.5;
                    }
                }
            }
        }

        ToFile(Parameters, "debugplot/parameters.data");
        System.out.println("ElPathPara done");
        return Parameters;
    }
    //End of method
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    // Method for mapping Z values of element centroids in the desired geometry to
       the current geometry
    public double[][][] MapElementCentroidToCurrent(String InitOrDeformed) {

        //METHOD DESCRIPTION:
        /* The method is capable of mapping the Z values from the desired geometry
           to the current geometry by performing the following steps:
         * - Read centroid data for both desired and current geometry
         * - Find "best match" element ID in desired geometry with shortest
             euclidean(x,y) distance to the current geometry
         * - Subtract Z values from "best match" in the desired geometry and the
             current geometry
```

```
1084            * - Output an error array [m][n]
1085            */
1086           System.out.println("Starting MapElementCentroidToCurrent");
1087           //Variables used in this method
1088           String Iteration = InitOrDeformed;
1089           int exs = 40;//Elements in the X direction
1090           int eys = 40;//Elements in the Y direction
1091           double[][] C = new double[exs][eys];//Ordered array with element ID
1092           int nElem = exs * eys;//No. of elements
1093           int ElstartID = 1;//Start element ID
1094           int ElId = ElstartID;//Iteration counter to loop through elements
1095           double dist = 0;//Euclidean distance, in XY plane, between centroid in
                  current geometry and the desired geometry
1096           double minDist = 10000;//Variable storing the shortest distance between two
                  element centroids (Must be initiated at a high value)
1097           int[] minDistElId = new int[nElem];//Element IDs corresponding to shortest
                  distances
1098           double[][] ZValueFinal = new double[exs][eys];//Complete Z coordinate values
                   ordered with respect to the element ID
1099           double[][][] ValueFinal = new double[exs][eys][4];//Z coordinate value
                  ordered with respect to the element ID
1100           double D[][] = ReadFile("centroidsblank.data", " ");//Centroid of the blank
                  before forming
1101
1102           //READ CENTROID COORDINATES FOR DESIRED GEOMETRY (A[][]) AND CURRENT
                  GEOMETRY (B[][])
1103           double A[][] = ReadFile("centroiddesired.data", " "); //Centroids of the
                  fine meshed desired geometry
1104           double B[][] = ReadFile(Iteration, " "); //Centroid of the current geometry
1105
1106           //FIND BEST MATCH CENTROID (SHORTEST EUCLIDEAN DISTANCE) BETWEEN B[][] AND A
                  [][]
1107           while (ElId <= nElem) {
1108               for (int i = 0; i < A.length; i++) {
1109                   dist = Math.sqrt(Math.pow(B[ElId - 1][1] - (A[i][1]), 2) + Math.pow(
                          B[ElId - 1][2] - (A[i][2]), 2)); //Euclidean distance in XY
                          plane
1110                   if (dist < minDist) {
1111                       minDist = dist; //Saves shortest distance
1112                       minDistElId[ElId - 1] = i + 1; // Controls the placement of the
                              shortest minDist
1113                   }
1114               }
1115               minDist = 10000; //reset distance variable
1116               ElId++; //The counter is iterated
1117           }
1118
1119           ElId = ElstartID; // The counter is reset
1120
1121           //SORT AND ORDER ELEMENT ARRAY
1122           ArraySort(D, 1, 2); //Sorting Array with respect to x and y coordinates of
                  the centroid
1123           for (int m = 0; m < exs; m++) {
```

```
1124            for (int n = 0; n < eys; n++) {
1125                int a = n + exs * m; //Integer that allows scanning of the entire
                        array length of A[][]
1126                C[m][n] = D[a][0]; //Generate array with ordered element ID's
1127            }
1128        }
1129
1130        //COMBINE COORDINATE VALUES WITH THE ORDERED ELEMENT ARRAY
1131        while (ElId <= nElem) {
1132            for (int z = 0; z < exs; z++) {
1133                for (int y = 0; y < eys; y++) {
1134                    if (C[z][y] == ElId) {
1135                        for (int i = 0; i < 4; i++) {
1136                            ValueFinal[z][y][i] = A[minDistElId[ElId - 1] - 1][i];
                                //Find the coordinates (xyz) of the element closest
                                to the centroid of the current geometry
1137                        }
1138                    }
1139                }
1140            }
1141            ElId++;
1142        }
1143
1144        //Selecting the Z value
1145        for (int z = 0; z < exs; z++) {
1146            for (int y = 0; y < eys; y++) {
1147                ZValueFinal[z][y] = ValueFinal[z][y][3]; // Creates separate array
                        for the Z coordinate
1148            }
1149        }
1150
1151        System.out.println("MapElementCentroidToCurrent done");
1152        return ValueFinal;
1153    }
1154        //End of method
1155 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1156    // Method for sorting arrays
1157
1158    public double[][] ArraySort(double A[][], int i) {
1159        return QuickSort(A, i);
1160    }
1161
1162    public double[][] ArraySort(double A[][], int i, int j) {
1163        return QuickSort(A, i, j);
1164    }
1165    //End of method
1166 //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1167    // Method for calculating the sum of absolute error w.r.t. Z-coord
1168    public double ErrorData(double[][][] ValueFinal, String centroids) {
1169
1170        //METHOD DESCRIPTION:
1171        /* The method is capable of calculating the sum of absolute of error w.r.t.
               Z-coord  by performing the following steps:
```

```
1172          * - Determine absolute error in the Z directions, between the current and
                  desired geometry for each element
1173          * - Calculate sum of the absolute error in each elements
1174          */
1175         System.out.println("ErrorData started");
1176
1177         int exs = 40;//Elements in the X direction
1178         int eys = 40;//Elements in the Y direction
1179         double[][][] ZValueFinal = ValueFinal;//Z coordinate of desired geometry
1180         String Iteration = centroids;//Name of current geometry centroid file
1181         double A[][] = ReadFile("centroidsblank.data", " ");//Centroid of the
                 unformed geometry
1182         double B[][] = ReadFile(Iteration, " ");//Centroids current geometry
1183         double Error = 0;//Error in Z distance between current and desired geometry
1184         double[][] Error2 = new double[exs][eys];//Error field i.e. the error in Z
                 distance in all elements
1185         double AbSumError = 0;//The sum of absolute error
1186         double[][] C = new double[exs][eys];//Ordered array with element ID
1187         double[][][] ZCoord = new double[exs][eys][4];//Coordinates of current
                 geometry
1188         int nElem = exs * eys;//No. of elements in current geometry
1189         int ElstartID = 1;//Start element ID
1190         int ElId = ElstartID;//Iteration counter to loop through elements
1191         double[][] ZCoordCurrent = new double[exs][eys];//Z Coordinates of current
                 geometry
1192
1193         //SORT AND ORDER ELEMENT ARRAY
1194         ArraySort(A, 1, 2); //Sorting Array with respect to x and y coordinates of
                 the centroid
1195         for (int m = 0; m < exs; m++) {
1196             for (int n = 0; n < eys; n++) {
1197                 int a = n + exs * m; //Integer that allows scanning of the entire
                         array length of A[][]
1198                 C[m][n] = A[a][0]; //Generate array with ordered element ID's
1199             }
1200         }
1201
1202         //COMBINE COORDINATES WITH THE ORDERED ELEMENT ARRAY
1203         while (ElId <= nElem) {
1204             for (int z = 0; z < exs; z++) {
1205                 for (int y = 0; y < eys; y++) {
1206                     if (C[z][y] == ElId) {
1207                         for (int i = 0; i < 4; i++) {
1208                             ZCoord[z][y][i] = B[ElId - 1][i];
1209                             ZCoordCurrent[z][y] = ZCoord[z][y][3];
1210                         }
1211                     }
1212                 }
1213             }
1214             ElId++;
1215         }
1216
1217         //CALCULATING THE SUM OF ABSOLUTE ERROR BETWEEN CURRENT GEOMETRY AND DESIRED
```

```
            GEOMETRY
        for (int z = 0; z < exs; z++) {
            for (int y = 0; y < eys; y++) {
                Error = ZValueFinal[z][y][3] - ZCoord[z][y][3];//Error in a single
                    array entrance
                Error2[z][y] = ZValueFinal[z][y][3] - ZCoord[z][y][3];//Complete
                    error array used for plots
                AbSumError = AbSumError + Math.abs(Error);//Saves the sum of
                    absolute error
            }
        }

        ToFile(getComponent(ZValueFinal, 3, 0), "debugplot/Zdesired.data");//data
            for postprocessing
        ToFile(getComponent(ZCoord, 3, 0), "debugplot/ZCurrent.data");//data for
            postprocessing
        ToFile(Error2, "debugplot/Error.data");//data for postprocessing
        System.out.println("ErrorData finished");
        return AbSumError;
    }
    //End of method
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    //Method for generating DEFINE CURVES and PARAMETRES file
    public void GenerateDefineCurve(int[][] ElPath, String centroid, double[][]
        ElPathPara) {

        //METHOD DESCRIPTION:
        /* The method is capable of generating the DEFINE_CURVES and PARAMETRES
            input file for the Finite Element model of the laser forming process by
            performing the following steps:
         * - convert scan path and process variables into DEFINE_CURVES AND
             PARAMETRES format
         */
        //Variables used in the method
        double[][] A = new double[ElPath.length][2];//X and Y coordinates of scan
            path used in DEFINE_CURVES
        double[] Nodes = new double[A.length];//Points in the scan path
        double[][][] coords = CenterAppender(centroid);//Coordinates of all element
            centroids
        int exs = 40;//Elements in the X direction
        int eys = 40;//Elements in the Y direction
        int ElstartID = 1;//Start element ID
        int ElId = ElstartID;//Iteration counter to loop through elements
        int nElem = 1600;//No. of elements
        double[] DefineTime = new double[Nodes.length + 1];     //Array storing the
            time, at which the laser beam must be at a given point in the scan path
        String name = null;//String used to designate function names in the generate
             DEFINE_CURVES file
        int id;//Function id in DEFINE_CURVES
        double[] PLaser = new double[Nodes.length + 1];//Laser power for all points
            in the scan path
        double[] radius = new double[Nodes.length + 1];//Laser beam radius for all
            points in the scan path
```

```
1255        double[] PLasersurface = new double[Nodes.length + 1];//The selected scan
                surface for all points in the scan path
1256        int surface = 0;//Section set ID for the selected surface
1257
1258        //GENERATE X,Y COORDINATES FOR THE SCANPATH
1259        double[] PointElId = new double[ElPath.length];
1260        for (int i = 0; i < PointElId.length; i++) {
1261            PointElId[i] = coords[ElPath[i][0]][ElPath[i][1]][0];
1262        }
1263
1264        double[][] curvecoords = new double[PointElId.length][2];
1265        for (int k = 0; k < PointElId.length; k++) {
1266            for (int x = 0; x < exs; x++) {
1267                for (int y = 0; y < eys; y++) {
1268                    if (PointElId[k] == coords[x][y][0]) {
1269                        A[k][0] = coords[x][y][1];
1270                        A[k][1] = coords[x][y][2];
1271                    }
1272                }
1273            }
1274        }
1275
1276        ToFile(A, "debugplot/path.data"); //Data for postprocessing
1277
1278        //CALCULATE TIME BETWEEN POINTS (used to calculate scan time from point to
                point in the scan path)
1279        for (int i = 1; i < Nodes.length; i++) {
1280            DefineTime[i] = DefineTime[i - 1] + (Math.sqrt(Math.pow(A[i][0] - A[i -
                    1][0], 2) + Math.pow(A[i][1] - A[i - 1][1], 2))) / (ElPathPara[i][0]
                     / 60.0); //dist between points/determined scan speed
1281            DefineTime[i] = Math.round(DefineTime[i] * 1000000) / 1000000.0; //round
                     the time
1282        }
1283
1284        //WRITE OUTPUT AS X AND Y DEFINE CURVE
1285        for (int k = 0; k < 2; k++) {
1286            if (k == 0) {
1287                name = "X";
1288                id = 1;
1289            } else {
1290                name = "Y";
1291                id = 2;
1292            }
1293            try {
1294                PrintWriter writer = new PrintWriter(name + "_DEFINE_CURVE", "UTF-8"
                        );
1295                writer.println("*DEFINE_FUNCTION_TABULATED");
1296                writer.println("$#     fid    definition");
1297                writer.println("         " + id + "     (t," + name + ") data pairs"
                        );
1298                writer.println("$# title");
1299                writer.println(name + "loc");
1300                writer.println("$#                t                    " + name);
```

```
1301                for (int i = 0; i < Nodes.length; i++) {
1302                    writer.println("          " + DefineTime[i] + "            " + A
                            [i][k]);
1303                }
1304                writer.println("          " + DefineTime[Nodes.length - 1] + "
                                " + A[Nodes.length - 1][k]);
1305
1306                writer.close();
1307            } catch (Exception ex) {
1308            }
1309        }
1310
1311        for (int i = 0; i < Nodes.length; i++) {
1312            PLaser[i] = ElPathPara[i][1];
1313            PLasersurface[i] = ElPathPara[i][3];
1314            radius[i] = ElPathPara[i][2];
1315        }
1316        PLaser[Nodes.length] = 0;
1317
1318        //WRITE OUTPUT AS P DEFINE CURVE for the laser model
1319        try {
1320            PrintWriter writer = new PrintWriter("P_DEFINE_CURVE", "UTF-8");
1321            writer.println("*DEFINE_FUNCTION_TABULATED");
1322            writer.println("$#     fid    definition");
1323            writer.println("         " + 9 + "     (t,P) data pairs");
1324            writer.println("$# title");
1325            writer.println("Power");
1326            writer.println("$#                  t                    Power");
1327            for (int i = 0; i < Nodes.length; i++) {
1328                writer.println("          " + DefineTime[i] + "           " + PLaser
                        [i]);
1329            }
1330            writer.println("          " + DefineTime[Nodes.length - 1] + "
                            " + PLaser[Nodes.length]);
1331            writer.close();
1332        } catch (Exception ex) {
1333        }
1334
1335        //WRITE OUTPUT AS Radius
1336        try {
1337            PrintWriter writer = new PrintWriter("RADIUS_DEFINE_CURVE", "UTF-8");
1338            writer.println("*DEFINE_FUNCTION_TABULATED");
1339            writer.println("$#     fid    definition");
1340            writer.println("         " + 11 + "     (t,R) data pairs");
1341            writer.println("$# title");
1342            writer.println("radi");
1343            writer.println("$#                  t                    radi");
1344            for (int i = 0; i < Nodes.length; i++) {
1345                writer.println("          " + DefineTime[i] + "           " + radius
                        [i]);
1346            }
1347            writer.println("          " + DefineTime[Nodes.length - 1] + "
                            " + radius[Nodes.length]);
```

```
1348            writer.close();
1349        } catch (Exception ex) {
1350        }
1351
1352        //WRITE OUTPUT AS SWITCH FOR EXPLICIT IMPLICIT
1353        try {
1354            PrintWriter writer = new PrintWriter("EXP_IMP_SWITCH");
1355            writer.println("*DEFINE_CURVE_TITLE");
1356            writer.println("Explicit to implicit switching");
1357            writer.println("$#    lcid      sidr       sfa       sfo      offa
                    offo    dattyp");
1358            writer.println("         5");
1359            writer.println("$#                 a1                  o1");
1360            writer.println("              0.000                   0");
1361            writer.println("          &dt1+0.3                    0");
1362            writer.println("          &dt2+0.3                    1");
1363            writer.println("             20.000                   1");
1364            writer.close(); // Closing the file
1365        } catch (IOException ex) {
1366        }
1367
1368        //WRITE OUTPUT AS PARAMETER
1369        try {
1370            PrintWriter writer = new PrintWriter("PARAMETRES");
1371            writer.println("*PARAMETER");
1372            writer.println("R      dt1   " + DefineTime[Nodes.length - 1]);
1373            writer.println("R        A " + 0.4);
1374            writer.println("*PARAMETER_EXPRESSION");
1375            writer.println("R      dt2      (&dt1+0.01)"); // Necessary to obtain
                    the format required by LS-Dyna
1376            writer.close(); // Closing the file
1377        } catch (IOException ex) {
1378        }
1379
1380        //WRITE OUTPUT BOUNDARY FLUX SET
1381        if (PLasersurface[1] == 0) {
1382            System.out.println("we choose the top");
1383            surface = 2;
1384        } else {
1385            surface = 4;
1386        }
1387
1388        try {
1389            PrintWriter writer = new PrintWriter("BOUNDARY_FLUX_SET");
1390            writer.println("*BOUNDARY_FLUX_SET");
1391            writer.println("$#    ssid");
1392            writer.println("         " + surface);
1393            writer.println("$#    lcid      mlc1      mlc2      mlc3      mlc4");
1394            writer.println("         3         0         0         0         0");
1395            writer.close(); // Closing the file
1396        } catch (IOException ex) {
1397        }
1398
```

```
1399          //End of method
1400      }
1401  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1402      // Method for determining the surface to be scanned
1403
1404      public String PathPosition(double[][][][] bottomstrains, double[][][][] topstrains)
              {
1405          //METHOD DESCRIPTION:
1406          /* The method is capable of determining the surface to be scanned by
                 performing the following steps:
1407           * - Determine most compressive strain in upper and lower surface
1408           * - Select surface with most compressive strain
1409           */
1410          System.out.println("PathPosition started");
1411          //Variables used in the method
1412          int ex = 40;
1413          int ey = 40;
1414          double[][][][] botPrin = PrincipalStrain(bottomstrains);
1415          double[][][][] topPrin = PrincipalStrain(topstrains);
1416          double minbot = 1000;
1417          double mintop = 1000;
1418          String PathPos;
1419
1420          //DETERMINE SIZE OF MINIMUM PRINCIPAL STRAIN (MOST COMPRESSIVE STRAIN)
1421          for (int x = 0; x < ex; x++) {
1422              for (int y = 0; y < ey; y++) {
1423                  if (botPrin[x][y][0] < minbot) {
1424                      minbot = botPrin[x][y][0];
1425                  }
1426                  if (topPrin[x][y][0] < mintop) {
1427                      mintop = topPrin[x][y][0];
1428                  }
1429              }
1430          }
1431
1432          //SELECT SURFACE WITH MOST COMPRESSIVE STRAIN
1433          if (mintop < minbot) {
1434              PathPos = "upper";
1435          } else {
1436              PathPos = "lower";
1437          }
1438
1439          System.out.println("PathPosition finished");
1440          return PathPos;
1441      }
1442      //End of method
1443  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

Appendix G

Shell Scripts

This appendix consists of the shell scripts utilised by the Java main program, appended in appendix E, to perform commands in the shell. The shell scripts are also appended on the appendix-CD.

Shell Script: clean_centroidonestep

```
# Remove unwanted data from file
sed '/*KEYWORD/,/*SHELL_ELEMENT_CENTROID/d' flatcentroid.k > temp.data
sed '$d' temp.data > onestepcentroid.data
sync
```

Shell Script: shell_centroids

```
# Opens d3plot in prepost and the .ses file saves the last state of the plot
prepost -nographics c=centroids.ses
# Remove unwanted data from file
sed '/*KEYWORD/,/*SHELL_ELEMENT_CENTROID/d' deformedcentroid.k > temp.data
sed '$d' temp.data > deformedcentroid.data
sync
```

Shell Script: shell_centroidsblank

```
# combines the blank.k with the segments
cp blank.k blank1.k
cat SEGMENTS blank1.k > blankSEG.k
rm blank1.k

# Opens the solid blank in prepost and saves the midplane as a shell model
DISPLAYTEMP=`echo $DISPLAY`
Xvfb :7 -ac -screen 0 1600x1200x24 &
DISPLAY=:7.0

prepost -nographics c=generateshellblank.ses


DISPLAY=`echo $DISPLAYTEMP`
killall Xvfb

# Opens the shell blank in prepost and saves the centroids of the model
prepost -nographics c=centroidsblank.ses
# Remove unwanted data from file
sed '/*KEYWORD/,/*SHELL_ELEMENT_CENTROID/d' centroidsblank.k > temp.data
sed '$d' temp.data > centroidsblank.data
rm centroidsblank.k
sync
```

Shell Script: shell_centroidsinit

```
# Opens d3plot in prepost and the .ses file saves the last state of the plot
prepost -nographics c=centroiddesired.ses
# Remove unwanted data from file
sed '/*KEYWORD/,/*SHELL_ELEMENT_CENTROID/d' centroiddesired.k > temp.data
sed '$d' temp.data > centroiddesired.data
rm centroiddesired.k
sync
```

Shell Script: shell_centroidsonestep

```
# Opens d3plot in prepost and the .ses file saves the last state of the plot
prepost -nographics c=centroidsonestep.ses
# Remove unwanted data from file
sed '/*KEYWORD/,/*SHELL_ELEMENT_CENTROID/d' flatcentroid.k > temp.data
sed '$d' temp.data > flatcentroid.data
sync
```

Shell Script: shell_clean

```
# Moving old simulation data files
rm new* messag nodout status.out d3* lspost* *~ tprint eloutdet part_des lsp* node*
    new_temp* spoo* disk* dyna.temp unfolded.k temp.data flatname.tmp
sync
```

Shell Script: shell_generateshell

```
# Opens d3plot in prepost and the .ses file saves the last state of the plot
prepost -nographics c=openplot.ses
# Combining segments and the data from the laststate of the d3plot
cat SEGMENTS laststate.k > laststatecomplete.k
# Generate shell representation of the current geometry
DISPLAYTEMP=`echo $DISPLAY`
Xvfb :7 -ac -screen 0 1600x1200x24 &
DISPLAY=:7.0

prepost -nographics c=generateshell.ses

DISPLAY=`echo $DISPLAYTEMP`
killall Xvfb

sync
```

Shell Script: shell_openplot

```
# Opens d3plot in prepost and the .ses file saves the last state of the plot
prepost -nographics c=openplot.ses
sync
```

Shell Script: shell_openplotonestep

```
# Opens d3plot in prepost and the .ses file saves the last state of the plot
prepost -nographics c=openplotonestep.ses
sync
```

Shell Script: shell_run

```
# Starts the simulation in the initial iteration
dyna_d i=lasermodel.k ncpu=-8 memory=100M > dyna.temp
sync
```

Shell Script: shell_run2

```
# Starts the simulation after initial iteration
dyna_d i=lasermodel2.k ncpu=-8 memory=100M > dyna.temp
sync
```

Shell Script: shell_runonestepinit

```
# Runs the onestep analysis for the initial iteration
dyna_d i=onestepinit.k ncpu=-8 memory=100M
sync
```

Shell Script: shell_runonestep

```
# Runs the onestep analysis after the initial iteration
dyna_d i=onestep.k ncpu=-8 memory=100M
sync
```

Appendix H

Fields used in Strain Analysis

This appendix lists the field representations used in the strain analysis described in section 3.3. Note that only the discrete representations of the fields are included.

- $\varepsilon_{top}(m,n)$ - A strain field placed on the top surface of the current geometry. Each element in the strain field contains the required strain in all six components of ε_{ij} related to go from the current to the desired geometry, in the top surface.
- $\varepsilon_{mid}(m,n)$ - A strain field placed on the midplane (between top and bottom surface) of the current geometry. Each element in the strain field contains the required strain in all six components of ε_{ij} related to go from the current to the desired geometry, in the midplane.
- $\varepsilon_{bot}(m,n)$ - A strain field placed on the bottom surface of the current geometry. Each element in the strain field contains the required strain in all six components of ε_{ij} related to go from the current to the desired geometry, in the bottom surface.
- $\varepsilon_{surf}(m,n)$ - Is a common designation of either the top or bottom surface strain field.
- $\varepsilon_{path}(m,n)$ - A strain field placed between the midplane and a surface. $\varepsilon_{path}(m,n)$ is used as the basis for the scan path algorithm, described in section 3.4.1. Each element in the strain field contains the average of the corresponding element in $\varepsilon_{mid}(m,n)$ and one of the surface strain fields i.e. $\varepsilon_{top}(m,n)$ or $\varepsilon_{bot}(m,n)$, for all six strain components. The surface strain field with the most compressive strain is selected for the average.
- $\varepsilon_2(m,n)$ - A strain field containing the size of the minimum principal strains of $\varepsilon_{path}(m,n)$.
- $\theta_2(m,n)$ - A strain field containing the orientation of the minimum principal strains of $\varepsilon_{path}(m,n)$.
- $\varepsilon_{thresh}(m,n)$ - A threshold field of $\varepsilon_2(m,n)$. Each element in the strain field contains either 0 or 1. The element equals 0 if $\varepsilon_2(m,n) <$ a limit defined by the threshold percentage and 1 if $\varepsilon_2(m,n) >$ the limit.

Appendix I

Keydeck for the Onestep Analysis

This appendix contains the keydeck for the OneStep solver, used to establish a strain field for the single curved geometries in chapter 4. The OneStep solver is a complete piece of Finite Element code provided in LS-Dyna. The OneStep solver receives inputs in the form of a shell model representation of either the current or desired geometry and the material properties of the blank. Based on the input geometry and the material properties the OneStep solver determines an initial unformed flat state. The output is a flattened blank along with the strain tensor ε_{ij} at the outermost integration points of each element. ε_{ij} represents the required strain to go from the flattened blank to the input geometry. [LSTC, 2013]

```
1  *KEYWORD
2  *TITLE
3  $# title
4  OneStep solution
5  $ By Group VT4-2.215
6  $ - Kasper Madsen
7  $ - Martin Soendergaard
8  $                                    |       |
9  $                                    |       |
10 $                                    |_______|
11 $                                     \     /   /
12 $                                      \___/   /
13 $                                        <|>   /
14 $                                        <|>  /
15 $                                        <|> /
16 $                                        <|>/
17 $                  ____________________|/
18 $
19 $
20 $ Unit System Scheme (C): mm, ms, kg ==> kN, GPa ,Joule, kW
21 $
22 $============================Include Files=====================================
23 $
24 *INCLUDE
25 deformedshell.k
26 $
27 $
```

```
$
$
$============================Control Cards=====================================
$
*CONTROL_TERMINATION
$   ENDTIM
       1.0
$
*CONTROL_IMPLICIT_GENERAL
$   IMFLAG       DT0
         1      0.25
$
*CONTROL_FORMING_ONESTEP
$   OPTION              AUTODB
         7                  -1

$
*CONTROL_FORMING_ONESTEP_AUTO_CONSTRAINT
$  AUTOSPC
         1
$
*CONTROL_IMPLICIT_TERMINATION
$   DELTAU
     0.001
$
*CONTROL_IMPLICIT_SOLUTION
$  NSSOLVR    ILIMIT    MAXREF     DCTOL     ECTOL
         2        11      1200     0.001      0.10
$
*CONTROL_IMPLICIT_SOLVER
$   LSOLVR
         4
$
*CONTROL_IMPLICIT_AUTO
$    IAUTO    ITEOPT    ITEWIN     DTMIN     DTMAX
         0         0         0       0.0       0.0
$
$=======================Parts and Material Models============================
$
*PART
$# title
Deformshell
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         2         1         1
$
*SECTION_SHELL_TITLE
blank
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         1        16  1.000000         5         1         0         0         1
$#      t1        t2        t3        t4      nloc     marea      idof    edgset
    1.0000    1.0000    1.0000    1.0000  1.000000
$
$
```

```
*MAT_PIECEWISE_LINEAR_PLASTICITY
$#     MID        RO         E        PR      SIGY      ETAN      FAIL      TDEL
         1 7.9000E-6   197.400    0.2942    0.2556    1.9462
$
*END
```

Listing I.1: Keydeck for the OneStep solver.

Appendix J

Principal Strain Equations

This appendix documents the equations used for determining the orientation and size of the minimum principal strains in a two dimensional strain element.

A given 2D strain element rotated by a given angle θ, creates a set of new strains, ε'_{xx}, ε'_{yy} and ε'_{xy} acting on the sides of the element. [Gere and Goodno, 2009]. By varying θ it is possible to identify the principle orientation of the element where only ε'_{xx}, ε'_{yy} act on the element and ε'_{xy} is equal to zero. At this direction ε'_{xx} equals the maximum principal strain ε_1 with the principal orientation θ_1 and ε'_{yy} equals the minimum principal strain ε_2 with the orientation θ_2, as shown in figure J.1.

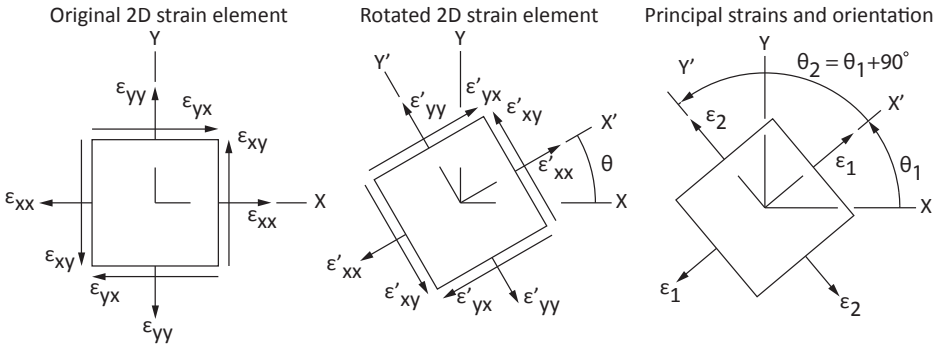


Figure J.1: Strain rotation of a 2D element

The strain transformation equations for rotating a 2D element by θ degrees are given by equation J.1 to J.3 [Gere and Goodno, 2009].

$$\varepsilon'_{xx} = \frac{\varepsilon_{xx} + \varepsilon_{yy}}{2} + \left(\frac{\varepsilon_{xx} - \varepsilon_{yy}}{2}\right)\cos(2\theta) + \varepsilon_{xy}\sin(2\theta) \tag{J.1}$$

$$\varepsilon'_{yy} = \frac{\varepsilon_{xx} + \varepsilon_{yy}}{2} - \left(\frac{\varepsilon_{xx} - \varepsilon_{yy}}{2}\right)\cos(2\theta) - \varepsilon_{xy}\sin(2\theta) \tag{J.2}$$

$$\varepsilon'_{xy} = \left(\frac{\varepsilon_{xx} - \varepsilon_{yy}}{2}\right) \sin(2\theta) + \varepsilon_{xy} \cos(2\theta) \tag{J.3}$$

By varying θ it is possible to identify the principle orientation of the element where only ε'_{xx}, ε'_{yy} act on the element and ε'_{xy} is equal to zero. At this direction ε'_{xx} equals the maximum principal strain ε_1 with the principal orientation θ_1 and ε'_{yy} equals the minimum principal strain ε_2 with the orientation θ_2, as shown in figure 3.11. By setting ε'_{xy} equal to 0 in equation in J.3 and solving for θ it is possible to determine the principal orientation. As a result ε_2, θ_1 and θ_2 are found, as shown in equation J.4 to J.6. [Gere and Goodno, 2009]

$$\varepsilon_{1,2} = \frac{\varepsilon_{xx} + \varepsilon_{yy}}{2} \pm \sqrt{\left(\frac{\varepsilon_{xx} - \varepsilon_{yy}}{2}\right)^2 + \varepsilon_{xy}^2} \tag{J.4}$$

$$\theta_1 = \frac{1}{2} \arctan\left(\frac{2\varepsilon_{xy}}{\varepsilon_{xx} - \varepsilon_{yy}}\right) \tag{J.5}$$

$$\theta_2 = \frac{1}{2} \arctan\left(\frac{2\varepsilon_{xy}}{\varepsilon_{xx} - \varepsilon_{yy}}\right) + \frac{\pi}{2} \tag{J.6}$$

Appendix K

Issues Concerning the OneStep Solver

This appendix concerns an issue detected, when using the OneStep solver with double curved geometries. The appendix presents the obtained orientation of minimum principal strains from three different tests of LS-Dyna's OneStep solver. The input for each test is the dome geometry tested in section 5.1. In each test the orientation of the dome geometry is changed i.e. rotated around the Z axis. The following orientations are tested:

- Test 1 - 0.0° around Z axis (no rotation)
- Test 2 - 22.5° around Z axis
- Test 3 - 45.0° around Z axis

The orientations of the minimum principal strains are shown in figure K.1, K.2 and K.3 respectively. The orientations of the minimum principal strains are obtained by using the post-processing software LS-Prepost. The orientations of the bars correspond to the orientation of minimum principal strain in the midplane. Fringes are omitted from the figures, as only the orientation of minimum principal strain is of interest for this test.

The orientation of minimum principal strain must be independent of the orientation of the input geometry. However, the three tests show that the orientation of minimum principal strain varies with varying orientation of the input geometry, why it is concluded that the OneStep solver is not suited for determination of the required strain in the developed framework.

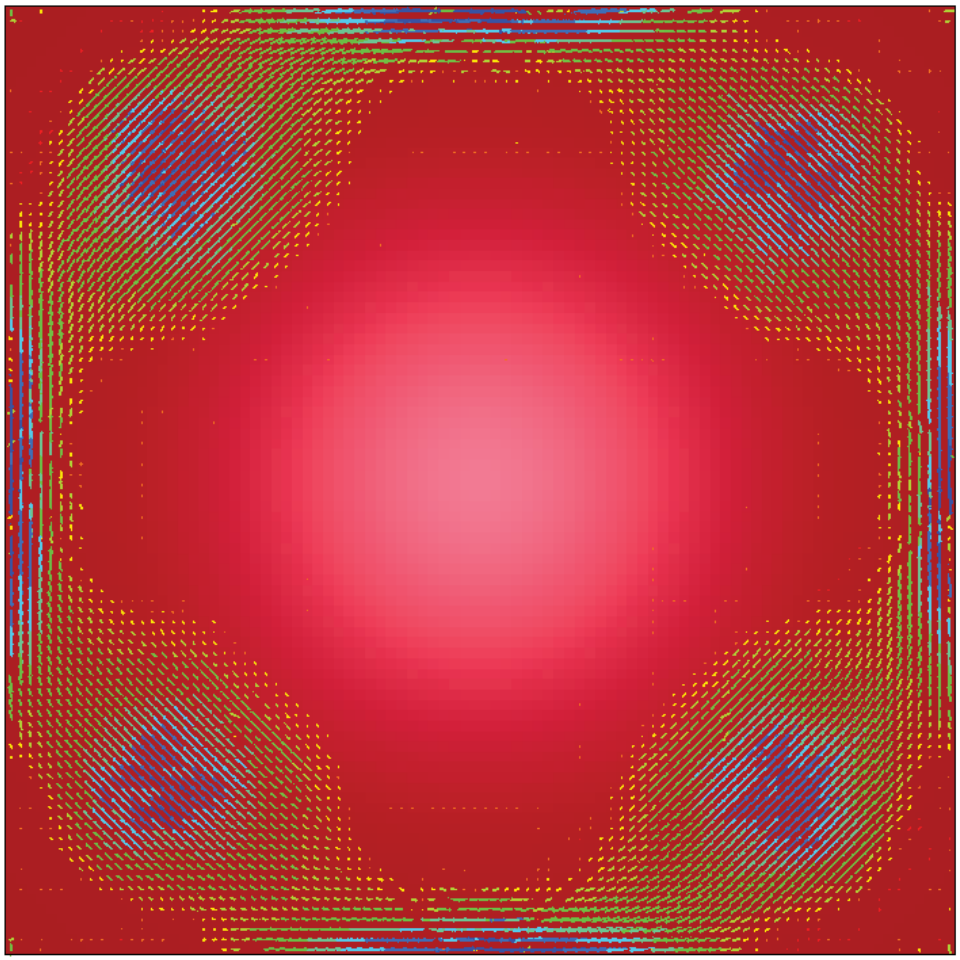


Figure K.1: Test 1 - The orientation of minimum principal strains with the dome geometry rotated 0.0° around the Z axis.

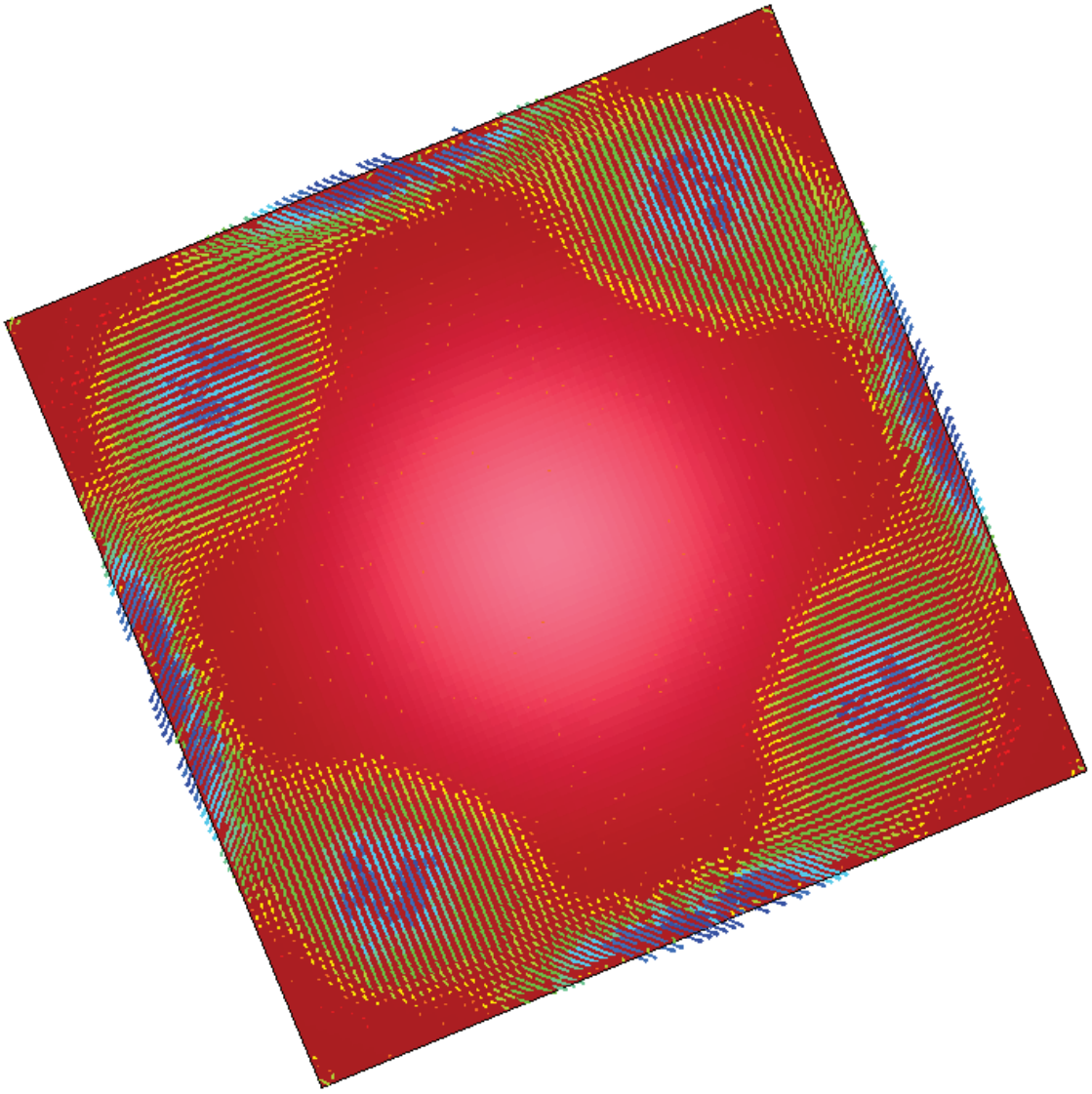


Figure K.2: Test 2 - The orientation of minimum principal strains with the dome geometry rotated 22.5° around the Z axis.

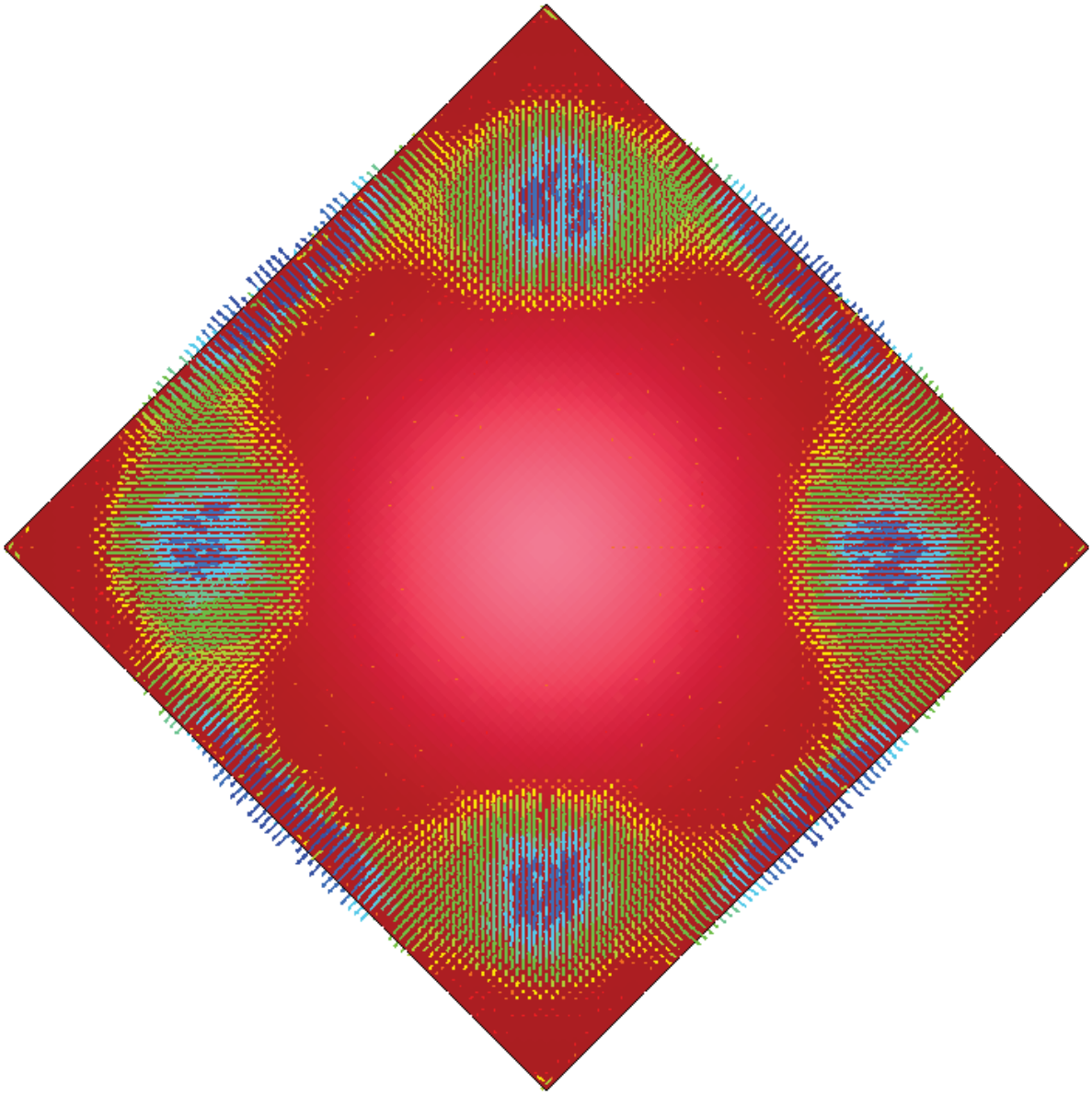


Figure K.3: Test 3 - The orientation of minimum principal strains with the dome geometry rotated 45.0° around the Z axis.

Appendix L

Keydeck for the Flattening Model used for Double Curved Geometries

This appendix contains the keydeck for the flattening model used to establish a strain field for the double curved geometries in chapter 5. The flattening model is developed as a replacement for the OneStep solver, as the OneStep solver provides misleading results when utilised on double curved geometries, as explained in appendix K. In the flattening model, a shell representation of either the desired or the current geometry is placed between two rigid planes. The upper rigid plane is moved towards the lower rigid plane, which is fixed in space. During the movement, the input geometry deforms, thereby, inducing strains in the flattened geometry.

```
1  *KEYWORD
2  $
3  $ Part summary:
4  $  pid  name            nid-start       elid-start
5  $      5        bottom          500000  500000
6  $      4        top             400000  400000
7  $
8  *TITLE
9  $# title
10 Flattening of desired geometries.
11 $
12 $-------------------INCLUDE FILES---------------
13 $
14 *INCLUDE
15 top.k
16 *INCLUDE
17 bottom.k
18 *INCLUDE
19 desired.k
20 $
21 *PARAMETER
22 $# Materiale parametre:
23 R my        0.000100
24 R pr        0.294000
25 R th        1.000000
```

```
R E          198.500
R K        425.70000
R n         0.256200
R ro       7.9000E-6
$
*INTERFACE_SPRINGBACK_LSDYNA
$     psid
         1
$
*SET_PART_LIST
$     psid
         1
$      pid
         1
$
*CONTROL_IMPLICIT_GENERAL
$#  IMFLAG       DT0    IMFORM      NSBS       IGS     CNSTN      FORM    ZERO_V
         1     0.001                             2                             0
$
*CONTROL_IMPLICIT_AUTO
$#   IAUTO    ITEOPT    ITEWIN     DTMIN     DTMAX     DTEXP
         1        11         5       0.2
$
$----------------Strains Out---------------------
$
*DATABASE_EXTENT_BINARY
$#   neiph     neips    maxint    strflg    sigflg    epsflg    rltflg    engflg
         0         0         0         1         2         1         2         2
$#  cmpflg    ieverp    beamip     dcomp      shge     stssz    n3thdt   ialemat
         0         0         0         1         1         1         1
$# nintsld   pkp_sen      sclp    unused     msscl     therm    iniout    iniout
         0         0  1.000000         0         0         0ALL       ALL
$
$----------------Control cards--------------------
$
*CONTROL_CONTACT
$#  slsfac    rwpnal    islchk    shlthk    penopt    thkchg     orien    enmass
     0.000     0.000         2         1         0         1         1
$#  usrstr    usrfrc     nsbcs    interm     xpene     ssthk      ecdt   tiedprj
         0         0         0         0     0.000         1
$#   sfric     dfric       edc       vfc        th     th_sf    pen_sf
     0.000     0.000     0.000     0.000     0.000     0.000     0.000
$#  ignore    frceng   skiprwg    outseg   spotstp   spotdel   spothin
         0         0         0         0         0         0     0.000
$#    isym    nserod    rwgaps    rwgdth     rwksf      icov         x    ithoff
         0         0         0     0.000  1.000000
$#  shledg
         0
$
*CONTROL_CPU
$#  cputim
     0.000
$
```

```
*CONTROL_DAMPING
$#  nrcyck     drtol    drfctr    drterm    tssfdr    irelal     edttl    idrflg
         0     0.000     0.000     0.000     0.000         0     0.000         0
$
*CONTROL_ENERGY
$#    hgen      rwen    slnten     rylen
         2         2         2         2
$
*CONTROL_HOURGLASS
$#     ihq        qh
         4
$
*CONTROL_OUTPUT
$#   npopt    neecho    nrefup    iaccop     opifs    ipnint    ikedit    iflush
         0         0         0         0     0.000         0         0         0
$#   iprtf    ierode     tet10    msgmax    ipcurv
         0         0         2        50
$
*CONTROL_SHELL
$#  wrpang     esort     irnxx    istupd    theory       bwc     miter      proj
     0.000         0         0         1        25         2         1
$# rotascl    intgrd    lamsht    cstyp6    tshell    nfail1    nfail4   psnfail
  1.000000         0         0         1
$# psstupd    irquad
         0         0
$
*CONTROL_TERMINATION
$#  endtim    endcyc     dtmin    endeng    endmas
  7.00000
$
*CONTROL_TIMESTEP
$#  dtinit    tssfac      isdo    tslimt     dt2ms      lctm     erode     ms1st
     0.000     0.000         0     0.000 -7.500E-4
$#  dt2msf   dt2mslc     imscl
     0.000         0         0
$
*DATABASE_BINARY_D3PLOT
$#      dt      lcdt      beam     npltc    psetid
  0.050000
$#   ioopt
         0
$
$
*DATABASE_RCFORC
$#      dt
  0.100000
$
$
$
$
$
$
$
```

```
$-----------------------CONTACT CARDS--------------------
$
*CONTACT_AUTOMATIC_SURFACE_TO_SURFACE_ID
$#     cid                                                                   title
         1
$#    ssid      msid     sstyp     mstyp    sboxid    mboxid       spr       mpr
         1         4         3         3
$#      fs        fd        dc        vc       vdc    penchk        bt        dt
       &my       &my     0.000     0.000     0.000         1     0.0001.0000E+20
$#     sfs       sfm       sst       mst      sfst      sfmt       fsf       vsf
  1.000000  1.000000     0.000     0.000  1.000000  1.000000  1.000000  1.000000
$
$
*CONTACT_AUTOMATIC_SURFACE_TO_SURFACE_ID
$#     cid                                                                   title
         2
$#    ssid      msid     sstyp     mstyp    sboxid    mboxid       spr       mpr
         1         5         3         3
$#      fs        fd        dc        vc       vdc    penchk        bt        dt
       &my       &my     0.000     0.000     0.000         1     0.0001.0000E+20
$#     sfs       sfm       sst       mst      sfst      sfmt       fsf       vsf
  1.000000  1.000000     0.000     0.000  1.000000  1.000000  1.000000  1.000000
$
$
$----------------------LOADS AND MOTION-------------------------
$
*BOUNDARY_PRESCRIBED_MOTION_RIGID
$#     pid       dof       vad      lcid        sf       vid     death     birth
         4         3         2         2  1.000000         01.0000E+28
$
$
*PART_MOVE
$#     pid            xmov            ymov            zmov     cid
       1           0.000           0.000              1
*PART_MOVE
$#     pid            xmov            ymov            zmov     cid
       4           0.000           0.000              6
$
$-------------------------DEFINE CURVES-----------------------
$
*DEFINE_CURVE
         2         0 0.0000000 0.0000000 0.0000000 0.0000000
                   0                   0
                 7.0             -11.75
                 7.1             -11.75
$
$--------------------------PARTS AND PROPERTIES--------------------------------
$
*PART
$# title
bottom
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         5         5         5
```

```
*SECTION_SHELL
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         5         2     0.000         0         1         0         0         1
$#      t1        t2        t3        t4      nloc     marea      idof    edgset
  1.000000  1.000000  1.000000  1.000000
*MAT_RIGID_TITLE
top
$#     mid        ro         e        pr         n    couple         m     alias
         5       &ro        &E       &pr        &n     0.000     0.000
$#     cmo      con1      con2
  1.000000         4         7
$#lco or a1       a2        a3        v1        v2        v3
     0.000     0.000     0.000     0.000     0.000     0.000
$
*PART
$# title
top
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         4         4         4
*SECTION_SHELL
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         4         2     0.000         0         1         0         0         1
$#      t1        t2        t3        t4      nloc     marea      idof    edgset
  1.000000  1.000000  1.000000  1.000000
*MAT_RIGID_TITLE
top
$#     mid        ro         e        pr         n    couple         m     alias
         4       &ro        &E       &pr        &n     0.000     0.000
$#     cmo      con1      con2
  1.000000         4         7
$#lco or a1       a2        a3        v1        v2        v3
     0.000     0.000     0.000     0.000     0.000     0.000
$
*PART
$# title
desired geometry
$#     pid     secid       mid     eosid      hgid      grav    adpopt      tmid
         1         1         1         0         0         0         0         0
*SECTION_SHELL
$#   secid    elform      shrf       nip     propt   qr/irid     icomp     setyp
         1        16                   5
$#      t1        t2        t3        t4      nloc     marea      idof    edgset
       1.0       1.0       1.0       1.0     0.000     0.000     0.000         0
*MAT_PIECEWISE_LINEAR_PLASTICITY
$#     MID        RO         E        PR      SIGY      ETAN      FAIL      TDEL
         1 7.9000E-6   198.500     0.294     265.0     1.986
$
*DAMPING_GLOBAL
         0        50
*END
```

Listing L.1: Keydeck for the flattening model.