Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Miguel Cortés Otero

Application of machine learning in the prediction of FDM part distortion based on finite element simulation

Master's thesis in Industrial Engineering master
Supervisor: Chao Gao
Co-supervisor: Filippo Berto
January 2022

NTNU
Norwegian University of
Science and Technology

Miguel Cortés Otero

Application of machine learning in the prediction of FDM part distortion based on finite element simulation

Master’s thesis in Industrial Engineering master
Supervisor: Chao Gao
Co-supervisor: Filippo Berto
January 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Abstract

FDM is the most widespread technology of all the additive manufacturing techniques nowadays. In recent years, much has been innovated in terms of new materials and process improvements for FDM printing. This has meant that FDM is not only used for rapid prototyping as before, but functional structures can be manufactured using this manufacturing technique. However, this printing technique still lacks dimensional accuracy for specific applications that require high accuracy. In this work, an approach of how to improve the dimensional accuracy of FDM printed parts is presented. First, a numerical simulation framework for additive manufacturing techniques is created based on Abaqus software (v. 2019 Simulia, Dassault Systems). The deformations of three different geometries are obtained and analysed using the numerical simulation framework. In addition, a study of the application of machine learning techniques to predict FDM part distortions is carried out.

Sammendrag

FDM er den mest utbredte teknologien av alle additive produksjonsteknikker i dag. De siste årene har mye blitt fornyet når det gjelder nye materialer for FDM-trykk og prosessforbedringer. Dette har gjort at FDM ikke bare brukes til rask prototyping som før, men funksjonelle strukturer kan produseres ved hjelp av denne produksjonsteknikken. Imidlertid mangler denne trykkteknikken fortsatt dimensjonsnøyaktigh et for spesifikke bruksområder som krever høy nøyaktighet. I dette arbeidet presenteres en tilnærming for hvordan man kan forbedre dimensjonsnøyaktigh eten til FDM-trykte deler. Først lages et numerisk simuleringsrammeverk for additive produksjonsteknikker basert på Abaqus software (v. 2019 Simulia, Dassault Systems). Deformasjonene til tre forskjellige geometrier oppnås ved hjelp av det numeriske simuleringsrammever ket og analyseres. I tillegg utføres en studie av broken av maskinlæringsteknikk er for å forutsi FDM-delforvrengninger.

Table of Contents

List of Figures

List of Tables

List of Abbreviations (or Symbols)

STL	Standard Tessellation Language
AM	Additive Manufacturing
SLS	Selective Laser Sintering
SLM	Selective Laser Melting
EBM	Electron Beam Melting
DMLS	Direct Metal Laser Sintering
BJ	Binder Jetting
MJF	MultiJet Fusion
DED	Direct Energy Deposition
LMD	Laser Metal Deposition
LENS	Laser Engineered Net Shape
FDM	Fused Deposition Modelling
LOM	Laminated Object Manufacturing
PLT	Paper Lamination Technology
UAM	Ultrasonic Additive Manufacturing
SLA	Stereolithography
DLP	Digital Light Processing
3D	3-Dimensional
ML	Machine Learning
PLA	Polylactic Acid
ABS	Acrylonitrile Butadiene Styrene
NT	Nodal temperature
FEA	Finite Element Analysis
KNN	K-Nearest Neighbors algorithm
SVM	Support Vector Machine algorithm
DT	Decision Tree algorithm
CNN	Convolutional Neural Network
SI	International System of Units

1 Introduction

1.1 State of the art of 3D printing technology

3D printing or additive manufacturing is the process of making three-dimensional solid objects from a digital file. The creation of the part is achieved by using additive processes. The most common technique consists in laying down successive layers of material until the part is completed. Additive manufacturing enables the production of complex shapes more efficiently than traditional manufacturing processes (subtractive manufacturing, like turning, milling, abrasion, etc.), mainly because it avoids wasting material.

There are five main steps in the printing process of a part:

1. Firstly, a sketch is made from an idea or need.
2. Create the design: after the sketch, a digital model of the part is created with 3D design software.
3. Then, the digital file is exported as an STL extension.
4. Once the digital file is converted to STL, it must be imported to a slicer program. In this program, the printing parameters and the different layers are defined. The output of the program is a file called G-code, containing the instructions that the printer must follow.
5. Upload the file to the 3D printer and start the printing.

The most used additive manufacturing techniques can be clustered into seven different groups as stated in (ISO, 2015). These groups are briefly described below:

- **Powder bed**: thermal energy selectively fuses regions of a powder bed. The raw material can be metal or plastic, both in powder form. The different processes grouped here are SLS, SLM, EBM, and DMLS.
- **Binder jetting**: a liquid bonding agent is deposited to join powder materials. The raw material can be metal, plastic or ceramic, the three of them in powder form. The processes derived from binder jetting are BJ, inkjet powder printing, and MJF.
- **Directed energy deposition**: a nozzle mounted on multi-axis arm deposits melted material. The raw material is metal in powder or wire form. The processes grouped here are laser cladding, DED, LMD, LENS, and laser or electron beam wire deposition.
- **Material extrusion**: material is drawn via a nozzle, where it is heated. It is deposited layer by layer. The raw material can be plastic or composites in solid form. There is only one process in this group, which is the FDM process, the most common of all additive manufacturing processes.
- **Sheet lamination**: sheets of material are bonded to form an object. The raw materials can be paper or metal in solid form. The processes derived from sheet lamination are LOM, PLT, and UAM.
- **Photo-polymerization**: liquid photopolymer is selectively cured by light-activated polymerization. The raw material must be a liquid photopolymer resin. There are two techniques grouped here: SLA and DLP.

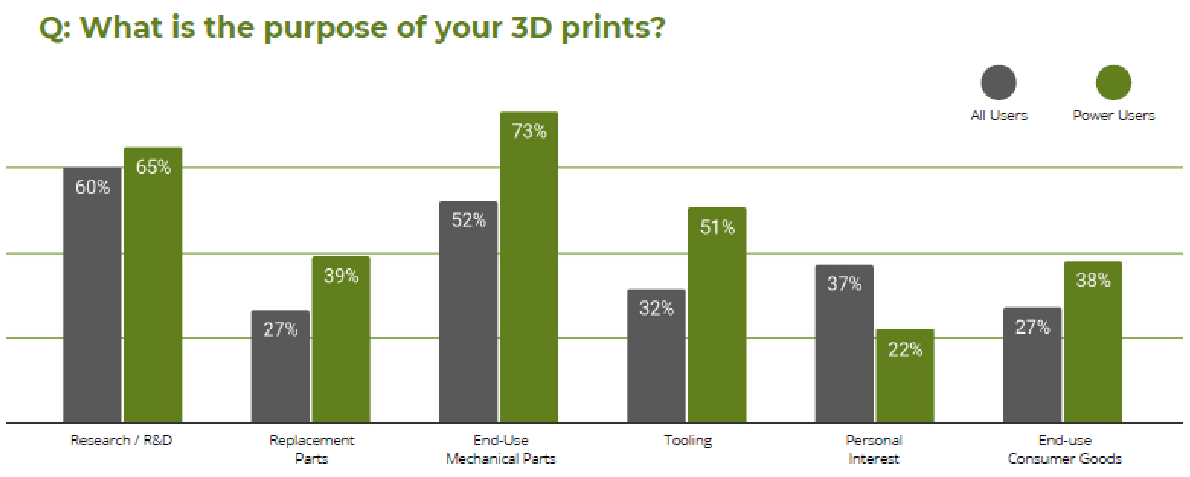
- **Material jetting**: droplets of build material are selectively deposited. The raw material must be plastic in ink form. The derived techniques from material jetting are photopolymer jetting and multi-jet modeling.

Although the techniques explained above are the most used, the 3D printing field is constantly evolving, and there is a lot of research on new novel techniques and different trends in the usage of different processes. An explanation of the trends in the use of 3D printing technology is given in the following sections, based on a survey of 3D printing users and owners of businesses that use this technology in its manufacturing processes. The survey was performed in 2021 by the company sculpteo (Moreau, 2021).

1.1.1 How is 3D printing used?

3D printing is proving to be a real manufacturing solution. It started as research and development for rapid-prototyping purpose but now is considered a real production tool. Currently, there is an increasing demand for end-use mechanical 3D printed parts, showing that additive manufacturing is considered a reliable manufacturing technology.

In the figures below, we can see the results of the survey to the questions ''What is the purpose of your 3D prints?'' and ''At which scale of manufacturing do you use 3D printing?'', respectively.



[1]

Figure 1. 1 Purpose of 3D printing technology

[1] **Power users**: users of 3D printing in a work context with significant investment and experience using the technology

Q: At which scale of manufacturing do you use 3D printing?

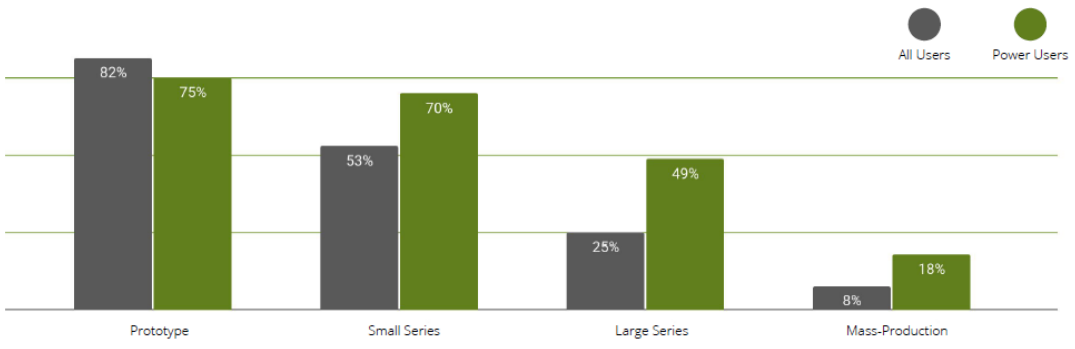


Figure 1. 2 Scale of production in 3D printed parts

1.1.2 What are the top requirements for material selection?

Here the surveyors are asked about what they consider to be the most important factor in terms of material selection. The most voted feature is strength with 72% of votes. In the second place, we have low cost, and easy to use in third place.

Q: What are your top requirements for material selection?

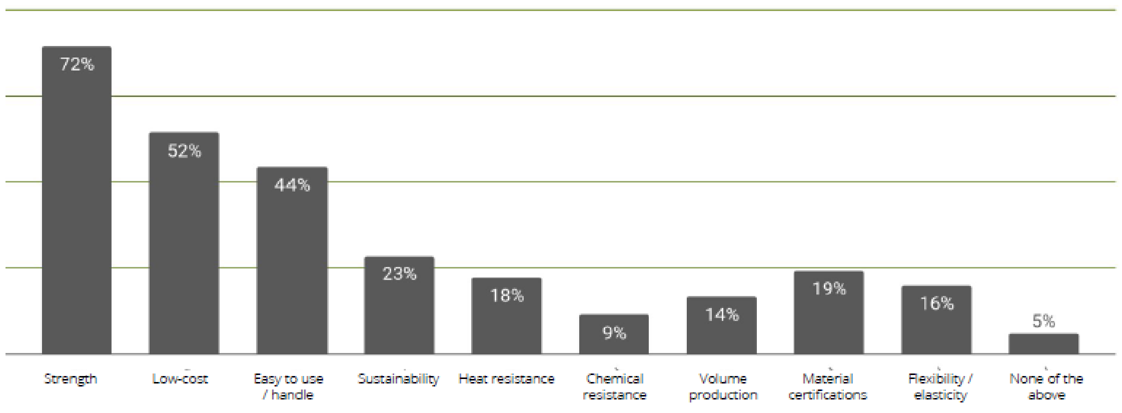


Figure 1. 3 Top requirements for material selection in 3D printing

1.1.3 What are the top challenges for using your 3D printer/s and which 3D printing technologies are mainly used?

As additive manufacturing is increasing its importance among manufacturing techniques, quality control and post-processing continue to be important challenges for the industry. Nowadays, for high production rates, the repeatability of the parts needs to be very high and so the quality control.

The respondents of the survey voted quality control as the first top challenge for 3D printing, post-processing in second place, and file preparation in third place.

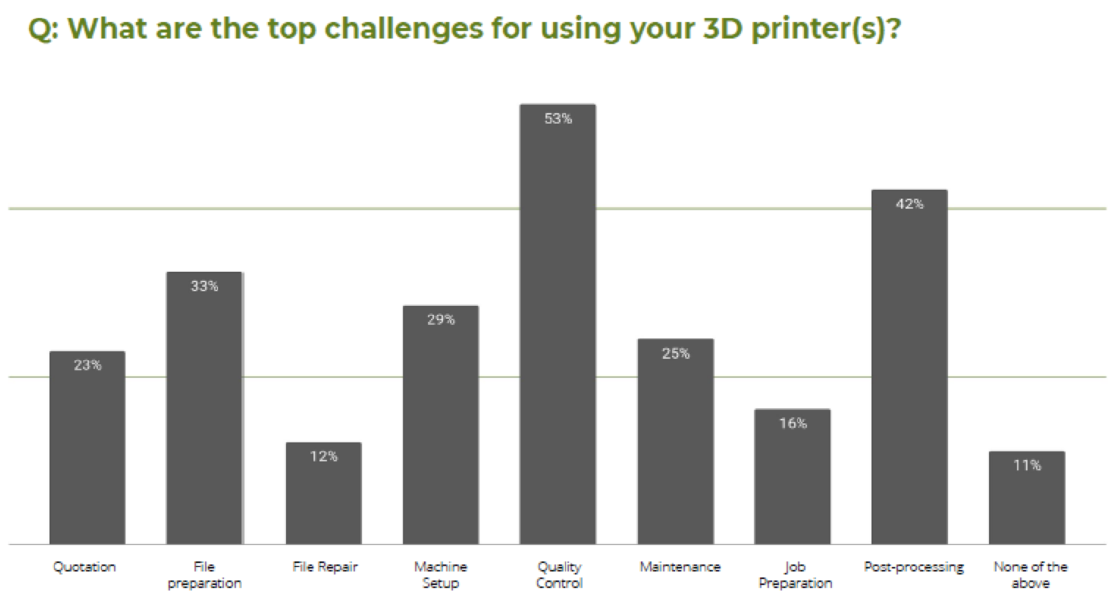


Figure 1. 4 Top challenges for using 3D printing technology

In terms of usage, FDM is the most used technique for the respondents, followed by SLS and SLA.

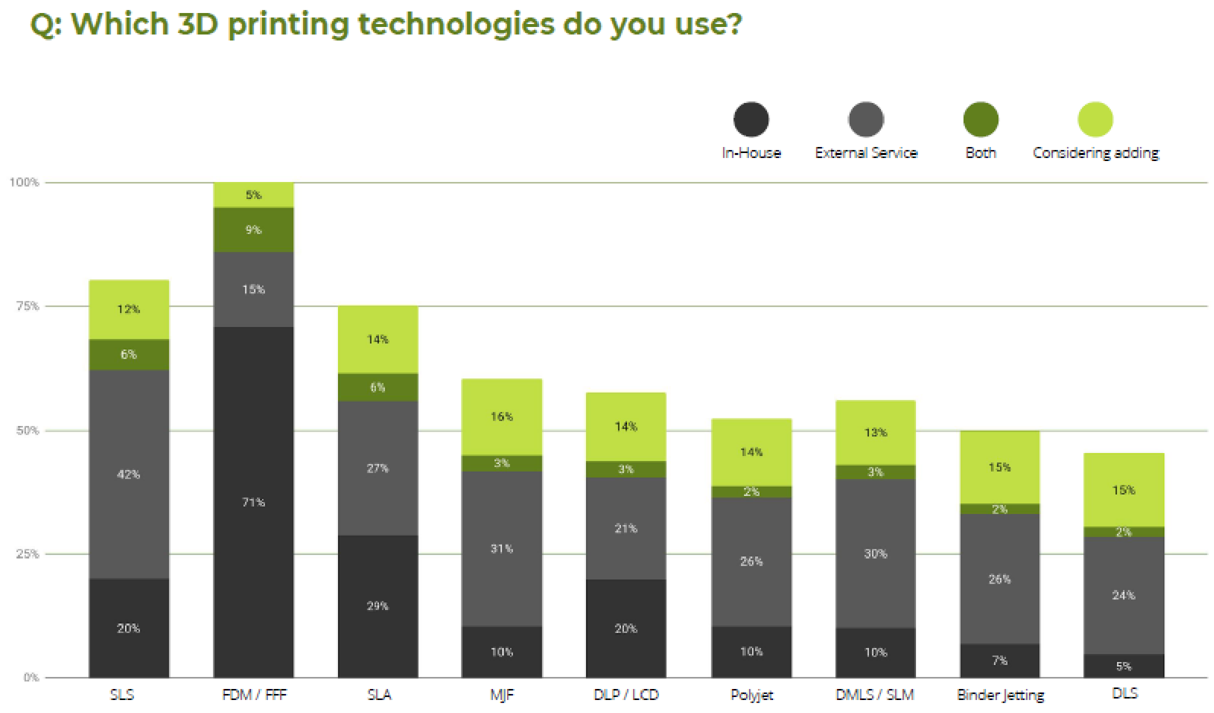


Figure 1. 5 Most used 3D printing technologies

1.1.4 Benefits and potential of 3D printing

There are lots of benefits to using additive manufacturing instead of conventional manufacturing methods. Some of the more important ones are described below:

- **Material waste and energy savings**: sometimes, for specific geometries, material supports will be required to hold the part during printing, but the overall waste is minimal.
- **Cost reduction**: specially in rapid prototyping cost savings are considerable. The 3D printer setup is much cheaper than a milling setup for example.
- **Complexity in the geometries of the parts**: additive manufacturing allows to produce a much more high range of different geometries compared to traditional manufacturing methods.
- **Weight reduction**: 3D printing allows creating light parts with increased strength because of the ability to create complex geometries. Some examples of these structures are achieved by topology optimization or lattice structures.
- **Quick iterative process**: continuous design improvement is much easier and cost saving with the additive manufacturing process.

To the respondents of the survey, the top benefit is the ability to produce complex geometry parts, followed by quick iteration and mass customization.

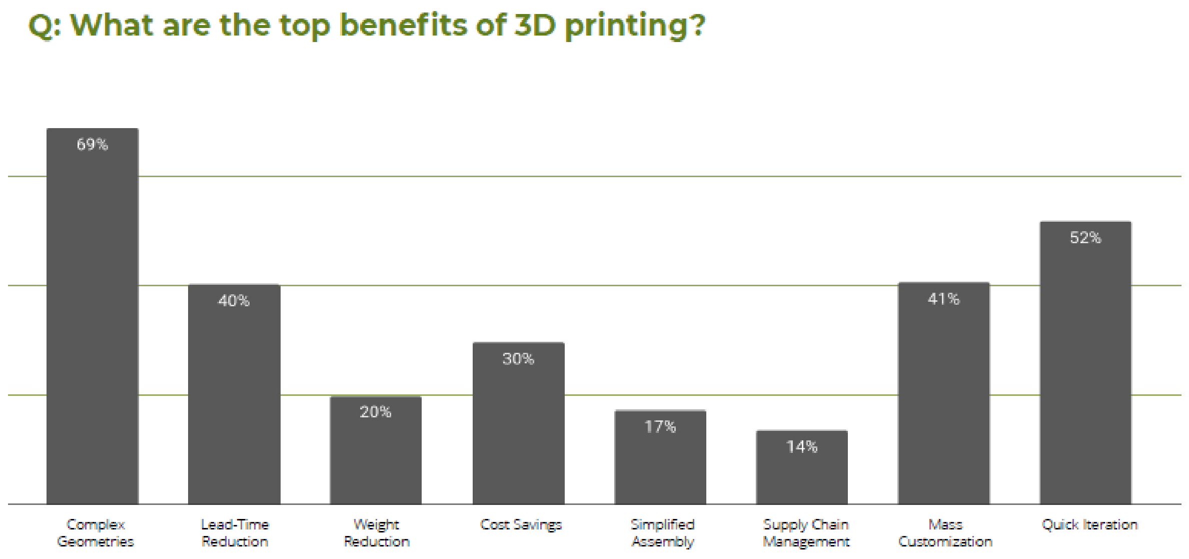


Figure 1. 6 Main benefits of implementing 3D printing technology

In terms of the potential of 3D printing, there is optimism regarding the future of this technology. The question is, ''What does the industry need to grow to become a standard of production?''. The respondents highlighted three main elements limiting the adoption of this technology: lower cost of entry, more reliable technologies, and new materials.

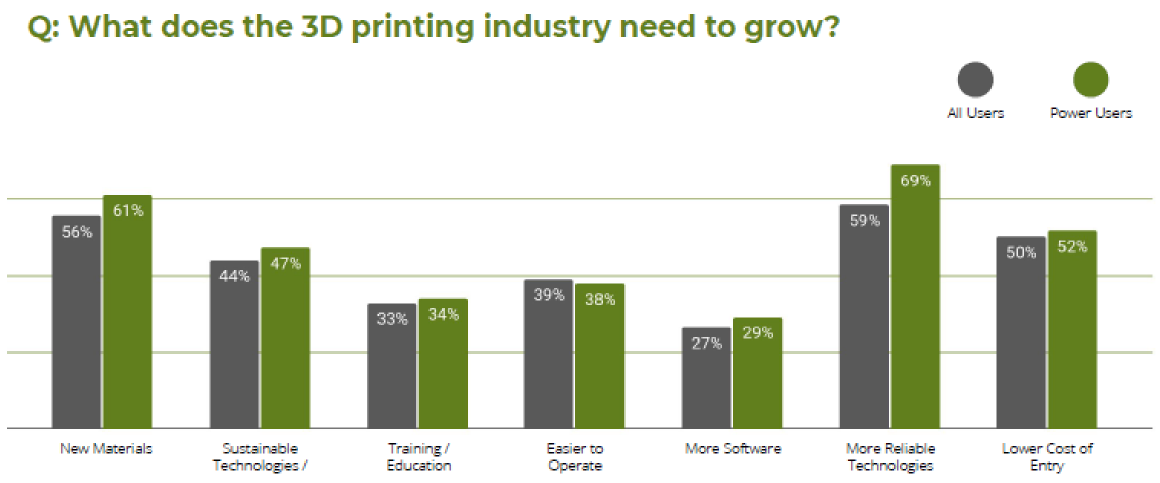


Figure 1. 7 Fields in which 3D printing needs to grow

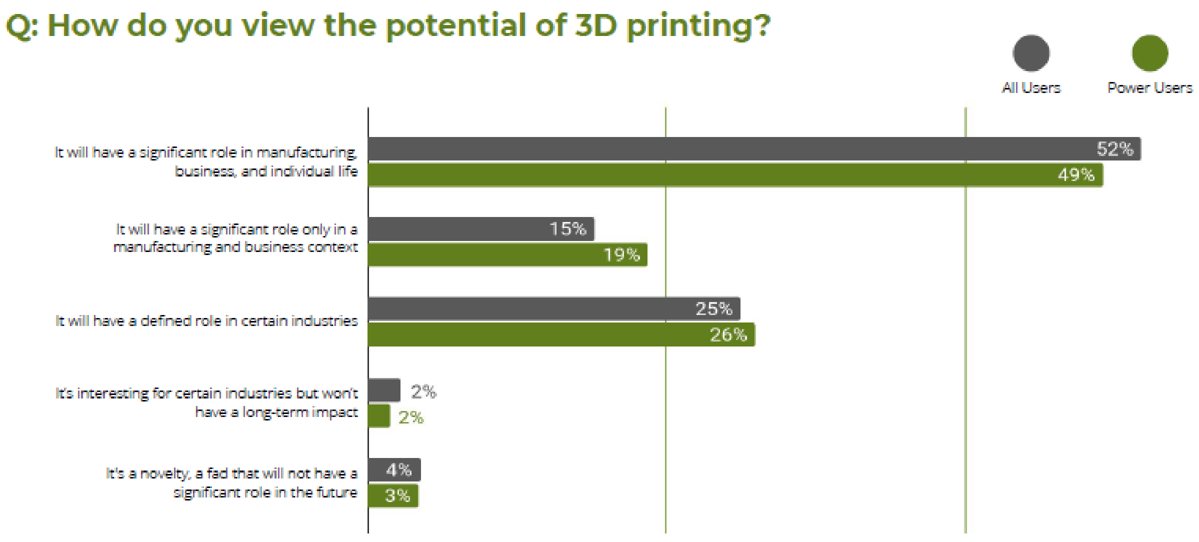


Figure 1. 8 Potential usages of 3D printing in the future

1.2 FDM process

1.2.1 Working principles and applications

This work is focused on the FDM process. In this section, the basics of this 3D printing technique are explained.

As explained in section 1.1, FDM belongs to the material extrusion group within 3D printing techniques. FDM utilizes polymers in a filament form as raw material. When the filament reaches the nozzle of the machine, it is heated to a molten state and then extruded through the nozzle. The nozzle head can move in three degrees of freedom (non-planar 3D extrusion is still under research) to deposit the molten polymer on the build plate as per the G-code instructions.

The filament is continuously fed through the extruder and nozzle of the machine via the two rollers rotating in opposite directions. The material is deposited on the build plate layer by layer until the part is completed. Multiple nozzles can be used to print composites with different polymers (Mwema & Akinlabi).

The resolution and effectiveness of the extrusion largely depend on the properties of the thermoplastic filament. In Figure 1. 9 the setup and different parts of a typical FDM 3D printer are shown.

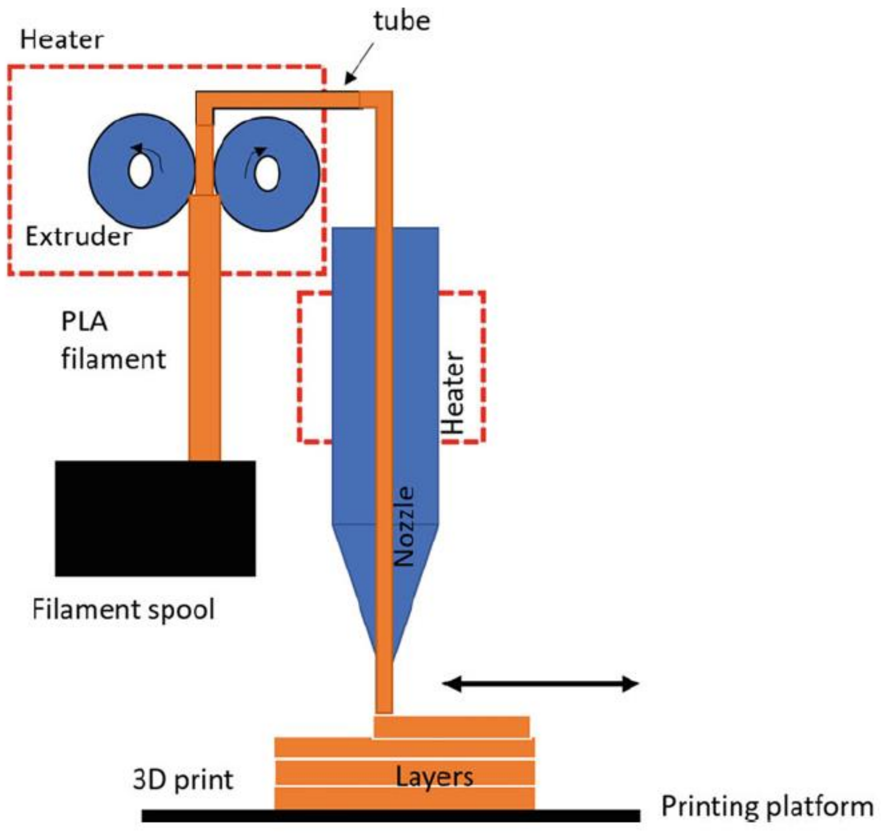


Figure 1. 9 Setup of a typical FDM 3D printer

Some of the most important applications for the FDM printing technique are listed below:

- It is a very suitable technology for prototyping and rapid tooling of complex products in low and medium batches.
- There is an increasing tendency of using the FDM process to produce molds for injection processes and the toy industry production.
- Personalization in the product development process.
- FDM is being applied in the medical field. There is a special interest in using FDM to produce molds for casting of implants. However, there are still some problems with the surface roughness of the FDM parts.
- Printing of electrochemical cells for energy storage devices, drug delivery components, the printing of conductors (electronics), etc.

1.2.2 Process parameters in FDM

The process parameters in FDM can be classified into two different groups: machine and material parameters. The material parameters depend on the material of the raw filament used in the print. Each material has different thermal and mechanical properties that strongly influence the performance of the process. On the other hand, the machine properties are defined by the G-code file. Some of them are infill pattern, infill density, raster angle, or temperature among others.

In Figure 1. 10 some of the most important FDM parameters are summarized.

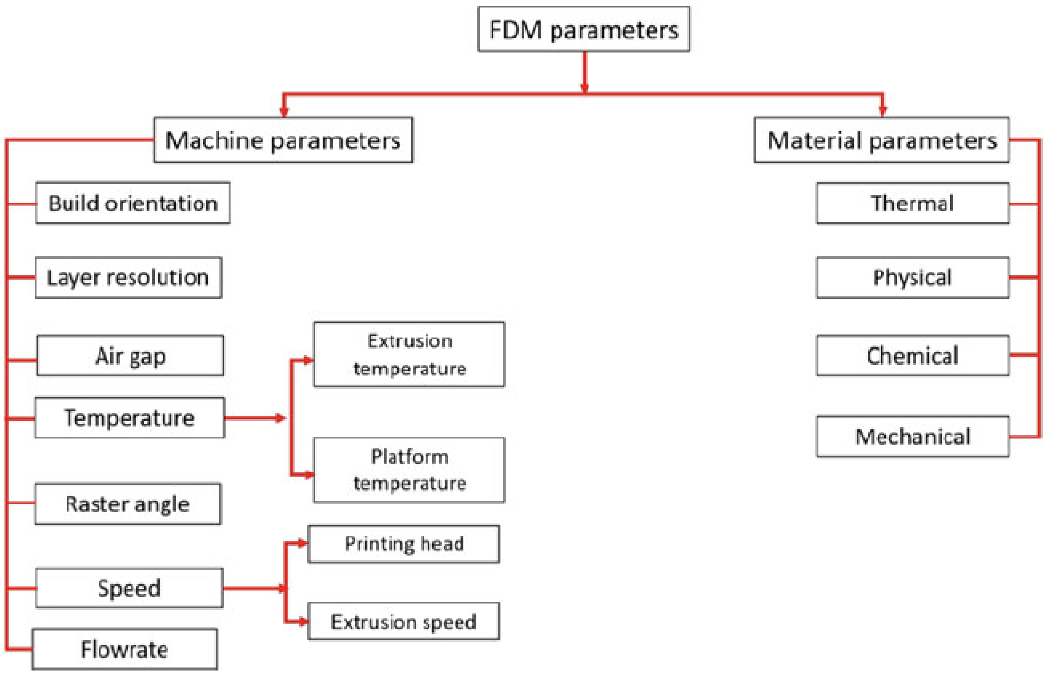


Figure 1. 10 FDM process parameters classification

- The **build orientation** indicates the angle at which the longest dimension of the part is inclined to the base of the build plate. An example of different build orientations is shown in Figure 1. 11.

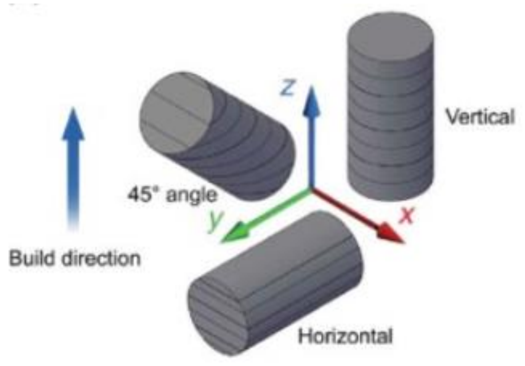


Figure 1. 11 Build orientation of 0º (horizontal), 45º, and 90º (vertical)

- The **layer resolution** indicates the thickness of the layers. It may vary from a few micrometers to millimeters depending on the accuracy of the printer (see Figure 1. 12).

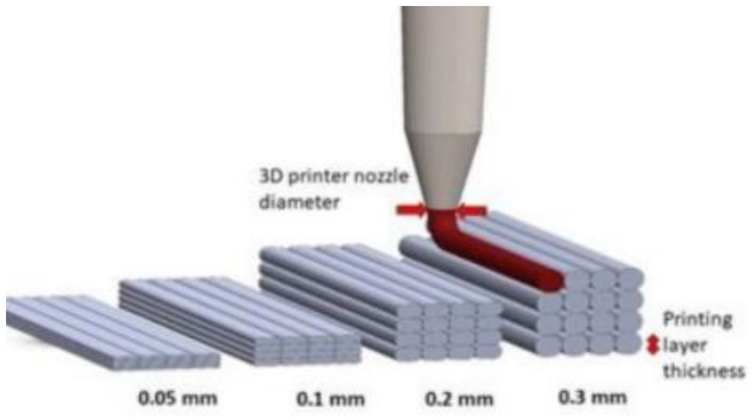


Figure 1. 12 Different layer thicknesses in the FDM process

- The **extrusion temperature** plays a very important role in the process because it must be high enough to melt the filament for easy extrusion. The **build plate temperature** is also very important to enhance the adhesion of the part with the build plate and avoid printing failure.

1.2.3 Quality issues in FDM

The quality and performance of the FDM printed parts depend on the choice of the process parameters summarised in the previous section.

The adhesion between adjacent layers and the extrusion conditions of the filament are critical for quality printings. Due to the nature of the process, the surfaces of the FDM printed parts exhibit the back-and-forth tracks of the printing nozzle known as the stair stepping effect. An image of this effect is shown in Figure 1. 13.

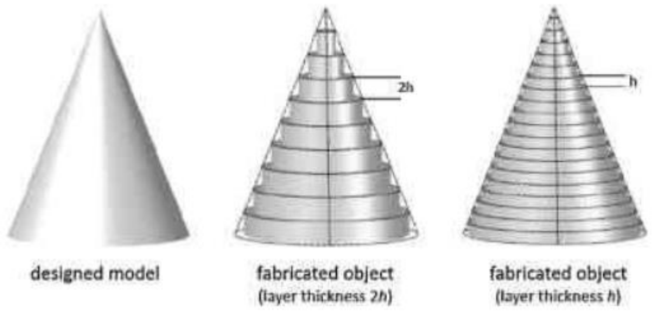


Figure 1. 13 Stair stepping effect with different layer thickness

Due to this effect, the surface roughness in FDM printed objects is one of the major drawbacks of the process. Surface roughness can be diminished at the design stage of the part by optimizing the slicing parameters and the print resolution. A direct relation exists between layer thickness and surface roughness: the higher the layer thickness, the higher the roughness and viceversa. Two main problems limit the minimum layer thickness value: one is the resolution of the printing machine and the other is the increase in the printing time, which may impact the other aspects of manufacturing, especially during mass production. There are various post-processing methods to lower the surface roughness, classified into mechanical (machining, sanding, polishing, abrasion,...) and chemical (painting, vapour deposition,...) methods.

Another problem that can cause mechanical failure and affect the dimensional accuracy of the part is the lack of adhesion between beads. Parts with a high density of defects would experience dimensional errors and low mechanical properties (hardness, toughness,...). Also if there is not enough adhesion between layers, the filament material of the adjacent layers will be forced to flow and compensate between the resulting spaces. This may lead to shrinkage of the component causing dimensional errors in the part. Additionally, porosity and cracks play a crucial role as stress raisers in the part that can cause part failure because the component cannot absorb the required energy.

There is a continued effort by the scientific community to understand the influence of the specific parameters on the FDM process and the quality of the printed parts. The interactions between these parameters are complex and require multi-objective approaches to enhance the quality of the printings.

2 Materials and methods

In this section, the main procedure of the thesis work is going to be described.

2.1 Simulation of the FDM process

In this work, numerical simulation is used to predict the part distortions for the FDM process.

2.1.1 Material

PLA was the chosen material for this work. Polylactic acid (PLA) is the most extensively researched and utilized biodegradable and renewable aliphatic polyester. Is a thermoplastic, high strength, high modulus polymer that proved to be a good substitute for petrochemical-based polymers and is useful for a lot of medical applications. Its main advantages are the following (Farah et al., 2016):

1. **Eco-friendly**: it is derived from renewable sources, biodegradable, recyclable and compostable.
2. **Biocompatibility**: it does not produce toxic or carcinogenic effects in local tissues, which makes it suitable for biomedical applications.
3. **Processibility**: PLA has better thermal processibility compared to other biopolymers, e.g., poly(hydroxyl alkanoate) or poly(ethylene glycol).
4. **Energy savings**: PLA requires 25-55% less energy to produce than petroleum-based polymers and estimations show that this can be further reduced to less than 10% in the future, which makes PLA very competitive in terms of cost. This is because the glass transition temperature varies between 50 and 70ºC and a melting point temperature ranging between 180 and 220ºC (relatively low temperatures).

PLA also has drawbacks that limit its use in some applications (Farah et al., 2016):

1. **Poor toughness**: it is a brittle polymer with only 10% of elongation at break. This could be dangerous because the brittle fracture is more difficult to detect than ductile fracture.
2. **Slow degradation rate**: PLA degrades through the hydrolysis of backbone ester groups and the rate depends on PLA crystallinity, molecular weight, and its distribution, morphology, water diffusion rate into the polymer, and the stereoisomeric content.
3. **Hydrophobicity**: the static water contact angle is approximately 80º.
4. **Lack of reactive side-chain groups**: it is chemically inert with no reactive side-chain groups making its surface and bulk modifications a challenging task.

Thermomechanical properties used in the numerical model

Due to the limited time to perform this work, the needed properties could not be measured experimentally, so instead a review of the existing research about PLA properties was done.

There are two main options to include material properties in the numerical model. The first option is to assume the properties as constant with temperature. The other option is to include the temperature dependence of the properties in the numerical model. The second option was chosen because the results are much more accurate (Cattenone et al., 2019).

The thermomechanical properties used to model the behavior of PLA are shown in the table below:

Property	Units	Value	Temperature dependance	Reference
$Density\ (\rho)$	$\frac{Kg}{m^3}$	1240	-	(Bhandari & Lopez-Anido, 2020)
$Expansion\ coefficient\ (\alpha)$	$\frac{1}{ºC}$	$6.8 \cdot 10^{-5}$	-	(*Materials available*, 2021)
$Poisson's\ ratio\ (\vartheta)$	-	0.36	-	(Farah et al., 2016)
$Elastic\ modulus\ (E)$	$\frac{N}{m^2}$	$3.5 \cdot 10^9$	30 ºC	(Zhou et al., 2016)
		$3.35 \cdot 10^9$	40 ºC	
		$2.75 \cdot 10^9$	50 ºC	
		$2.2 \cdot 10^9$	55 ºC	
		$1.75 \cdot 10^9$	60 ºC	
		$0.95 \cdot 10^9$	70 ºC	
		$0.75 \cdot 10^9$	80 ºC	
		$0.6 \cdot 10^9$	90 ºC	
		$0.55 \cdot 10^9$	100 ºC	
		$0.5 \cdot 10^9$	120 ºC	
		$0.49 \cdot 10^9$	150 ºC	
$Yield\ strength\ (\sigma_y)$	$\frac{N}{m^2}$	$70 \cdot 10^6$	30 ºC	(Zhou et al., 2016)
		$55 \cdot 10^6$	40 ºC	
		$40 \cdot 10^6$	50 ºC	
		$32.5 \cdot 10^6$	55 ºC	
		$25 \cdot 10^6$	60 ºC	
		$8 \cdot 10^6$	70 ºC	
		$5 \cdot 10^6$	80 ºC	
		$2 \cdot 10^6$	90 ºC	
$Thermal\ Conductivity\ (\kappa)$	$\frac{W}{m \cdot ºC}$	0.111	48 ºC	(Farah et al., 2016)
		0.197	109 ºC	
		0.195	190 ºC	
$Specific\ heat\ (C_p)$	$\frac{J}{Kg \cdot ºC}$	1590	55 ºC	(Farah et al., 2016)
		1955	100 ºC	
		2060	190 ºC	
$Convective\ heat\ transfer\ coefficient\ (h)$	$\frac{W}{m^2 \cdot ºC}$	100	-	(Bhandari & Lopez-Anido, 2020)
$Emissivity\ (\varepsilon)$	-	0.95	-	(Morgan et al., 2017)

Table 2. 1 PLA properties used in the numerical model

2.1.2 Studied geometries

Three simple geometries have been chosen for the development of this work. The first one is a simple bead of PLA as shown in Figure 2. 1. The real form of the bead in the FDM process is elliptical but can be approximated as a rectangle. This assumption is made in the majority of research in the field of numerical simulation of the FDM process. This first geometry was chosen because of its simplicity and low computational cost. For this geometry, the model change method (explained in section 2.1.3.1) was used to simulate the additive manufacturing process.

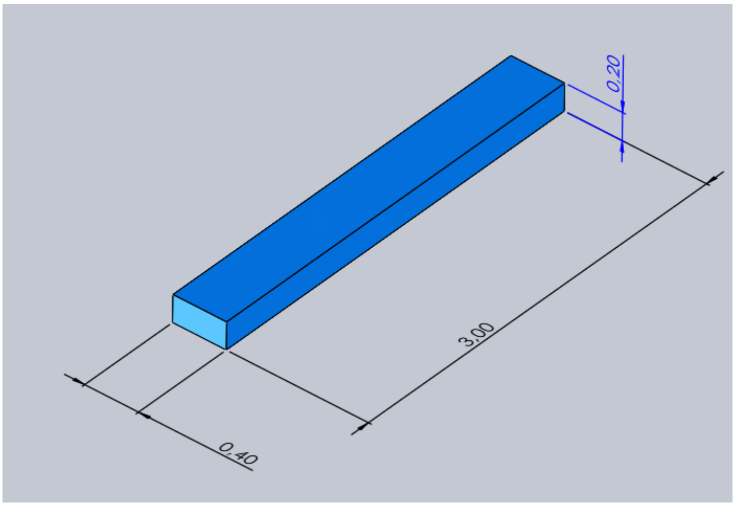


Figure 2. 1 Bead used as the first geometry for the numerical simulation (dimensions in mm)

The second geometry used was a flat spring as shown in Figure 2. 2. This geometry was used to compare the results of the simulation with the results (Cattenone et al., 2019) (benchmark) to make sure that the workflow of the simulation was done correctly. Although the material used in (Cattenone et al., 2019) is ABS and in this work we are using PLA, the results are comparable.

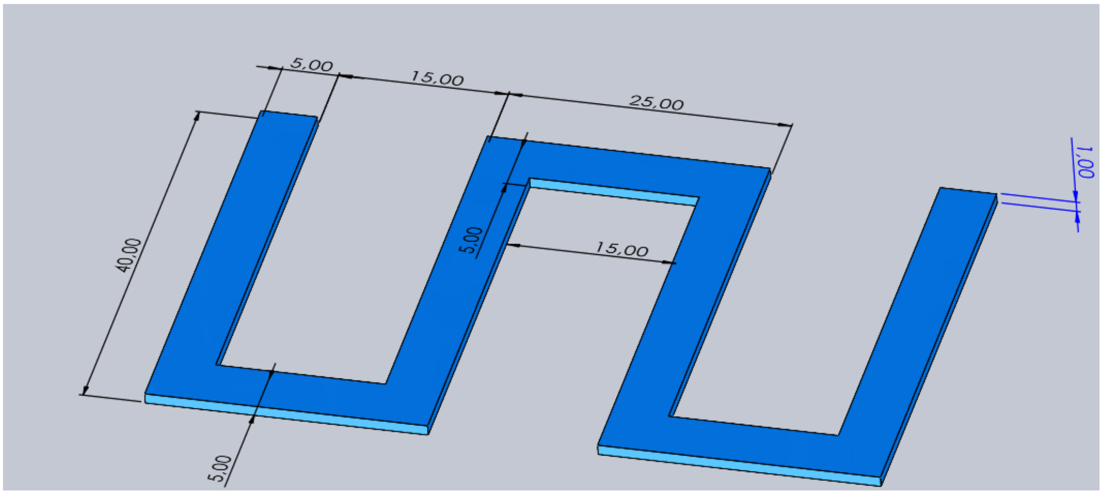


Figure 2. 2 Flat spring used as the second geometry for the numerical simulation (dimensions in mm)

Finally, the third geometry used in the work was a thin wall (see Figure 2. 3) with four different infill patterns.

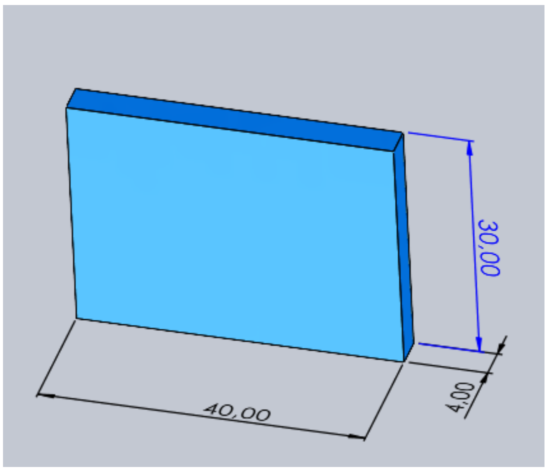


Figure 2. 3 Thin wall used as the third geometry for the numerical simulation (dimensions in mm)

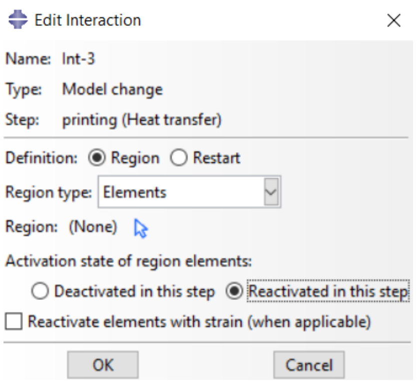
2.1.3 Thermomechanical analysis of the FDM process

The analysis is divided into two steps. Firstly, a thermal analysis is performed, solving the heat equation. The evolution of the temperature field during the printing is obtained as an output of this analysis. Then, the resulting temperature field is used in the mechanical analysis to evaluate residual stresses and distortions of the part. The software used to perform the analysis is Abaqus (v. 2019 Simulia, Dassault Systems).

2.1.3.1 Element activation

As it was described in section 1.2.1, FDM is an additive manufacturing technique, so the thermomechanical model must reflect how the material is added with time. In Abaqus software, two different options exist to add material with time: with the model change interaction and with the toolpath-mesh intersection module.

1. With the **model change** interaction (see Figure 2. 4), a set of predefined elements or geometries are deactivated or reactivated in a specific step. This option has big limitations for big or complex 3D printed models because a lot of steps and interactions should be defined in order to complete the whole part, so is not efficient to use this method for element activation.


Edit Interaction

Name: Int-3
Type: Model change
Step: printing (Heat transfer)

Definition: [✓ Region] Restart
Region type: Elements
Region: (None)

Activation state of region elements:
Deactivated in this step [✓ Reactivated in this step]
☐ Reactivate elements with strain (when applicable)

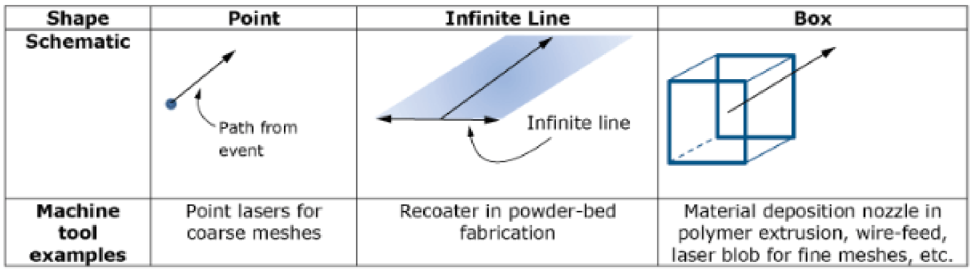
Figure 2. 4 Model change interaction

2. **Toolpath-mesh intersection module**: a toolpath represents the motion of a given component of a machine. In the case of this work, represents the movement of the printer nozzle. A toolpath is a geometric shape attached to a reference point that moves along a path. The path is defined with an event series format (see Table 2. 2), where the time and position of the reference point are indicated. The basic information needed to create the event series is: *(i)* the nozzle position, *(ii)* the nozzle velocity, and *(iii)* the cross-sectional dimensions of the extruded filament. Field variables can be added to the event series as well. The typical field variable is the on/off to indicate if the nozzle is extruding material in one toolpath segment or not.

[time]	[x]	[y]	[z]	[on/off]
0	0	0.0002	0	1
0.666	0.0398	0.0002	0	1
0.732	0.0398	0.0038	0	1
1.398	0.0002	0.0038	0	1
1.464	0.0002	0.0002	0	0
1.5	0.0004	0.0014	0	1

Table 2. 2 Event series format used for the toolpath-mesh intersection module

Three shapes are considered for toolpath-mesh intersection: a point, an infinite line, and a box (see Figure 2. 5). Depending on the application, the shapes should be chosen to describe the shape of the tool. In addition to these three shapes, a scan pattern that describes the idealized motion of the tool instead of the actual path can be used. In the case of the FDM process, the box shape is used.



Shape	Point	Infinite Line	Box
Schematic			
Machine tool examples	Point lasers for coarse meshes	Recoater in powder-bed fabrication	Material deposition nozzle in polymer extrusion, wire-feed, laser blob for fine meshes, etc.

Figure 2. 5 Different toolpath for different tool shapes

The real cross section of the beads in the FDM process is typically characterized by a rectangular shape with round corners, resulting in an ellipsoidal shape. As a simplifying assumption, the cross section is assumed to be rectangular with dimensions equal to the ellipse axes. To perform the element activation, a marching rectangle is imagined moving following the filament centerline indicated in the event series (coincident with the marching rectangle center as well). If the center of an element falls within the ideal volume described by the marching rectangle, then the element is activated. Figure 2. 6 shows how the elements are activated in this module.

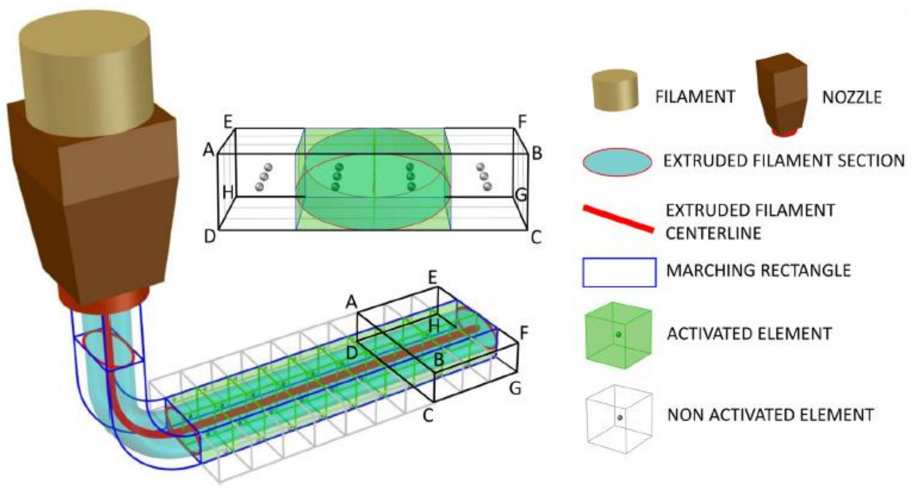


Figure 2. 6 Element activation with the box shape

2.1.3.2 Thermal analysis

In the FDM process, the filament is extruded at the temperature programmed in the G-code, the nozzle temperature (T_n). The molten filament is deposited on the build plate at the bed temperature indicated in the G-code as well (T_b). In addition, the molten filament exchanges heat with the surroundings at ambient temperature (T_a).

There are three different mechanisms for heat exchange: *(i)* conduction: heat exchange with the previously deposited material and with the build plate; *(ii)* convection: heat exchange with the surroundings via surface film conditions (eq. (3)) and *(iii)* radiation: heat exchange generated by the thermal movement of charged particles in the material (eq. (4)).

The time-spatial temperature field $T(x,t)$ is governed by the heat equation (1):

$$\boldsymbol{\rho c \dot{T} = \nabla \cdot (k \nabla T) + q} \qquad (1)$$

where ρ is the material density, $\boldsymbol{c = c\,(T)}$ is the specific heat capacity, $\boldsymbol{k = k(T)}$ is the thermal conductivity, q is the internal heat source, and t is the time. The initial conditions of the problem are the nozzle temperature, build plate temperature and ambient temperature. The Neumann boundary conditions of the process are defined as in equation (2):

$$\boldsymbol{k\frac{\partial T}{\partial n} + q_c + q_r = 0\,;\; x \in S(t)} \qquad (2)$$

where q_c is the heat flux due to convection, q_r is the heat flux due to radiation, $S(t)$ is the external surface of the body and n is a vector normal to the surface of the body.

$$\boldsymbol{q_c = h(T - T_a)} \qquad (3)$$

$$\boldsymbol{q_r = k_b(T^4 - T_a^4)\epsilon} \qquad (4)$$

Where k_b is the Stefan-Boltzmann constant, h is the convection coefficient and $\boldsymbol{\epsilon}$ is the emissivity.

2.1.3.3 Mechanical analysis

Assuming a small strain regime, is common to introduce the decomposition:

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^e + \boldsymbol{\varepsilon}^p + \boldsymbol{\varepsilon}^T$$

where ε is the total strain, ε^e is the elastic strain, ε^p is the plastic strain and ε^T is the thermal strain. The definition of these strains is given by the following equations:

$$\boldsymbol{\varepsilon}^e = \boldsymbol{D}^{-1}\boldsymbol{\sigma} \quad (5)$$

$$\boldsymbol{\varepsilon}^p = \boldsymbol{\lambda}\boldsymbol{\sigma}^{dev} \quad (6)$$

$$\boldsymbol{\varepsilon}^T = \boldsymbol{\alpha}(\boldsymbol{T} - \boldsymbol{T}_0)\boldsymbol{I} \quad (7)$$

where σ is the stress, $E = E(T)$ is the Young's modulus, v is the Poisson's ratio, $\alpha = \alpha(T)$ is the thermal expansion coefficient, I is the identity, λ is the plastic flow factor, and σ^{dev} represents de deviatoric part of the stress tensor:

$$\boldsymbol{\sigma}^{dev} = \boldsymbol{\sigma} - \frac{\mathbf{1}}{\mathbf{3}}\boldsymbol{tr}(\boldsymbol{\sigma})\boldsymbol{I} \quad (8)$$

PLA is a semicrystalline resin. Its thermal properties are strongly influenced by the glass transition temperature T_g. The glass transition indicates an increasing disorder associated with the molecular structure. It marks the transition from the glassy to the rubbery mechanical behavior of the material. Young modulus and yield stress are very influenced by temperature (see Figure 2. 7 and Figure 2. 8).

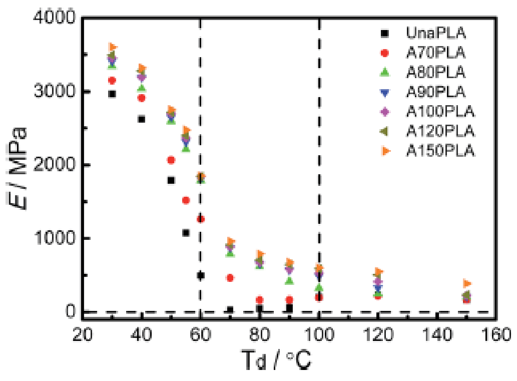


Figure 2. 7 Young's modulus dependance with temperature

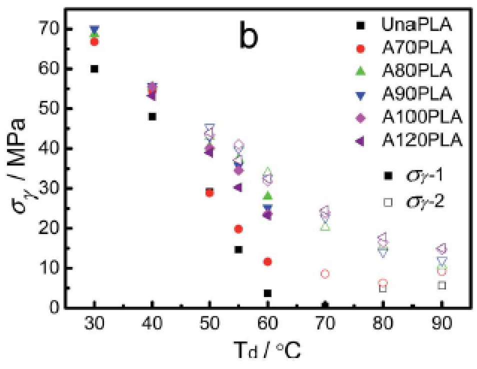


Figure 2. 8 Yield stress dependance with temperature

2.1.4 Simulation setup with Abaqus 2019

This section is going to be explained based on the wall geometry exposed in section 2.1.2.

2.1.4.1 Thermal analysis

Firstly, the part is created using SolidWorks. Then, the part is imported into ABAQUS. The simulation setup in this software is based on different modules. The detailed explanation of the simulation setup is explained below.

Property module:

In this module, the PLA properties are defined in table 2.1. Then, a solid homogeneous section is defined and then it is assigned to the whole part.

Assembly module:

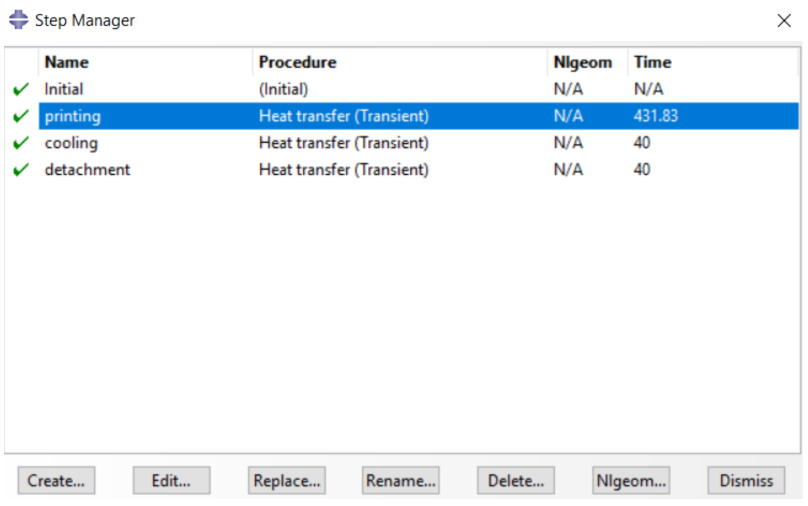
Here an instance is created. In this case, is created as a dependent instance, so it must be meshed on part. We can also adjust the position of the part with respect to the global coordinate axes.

Step module:

In this module, we define the different steps of the analysis. Three different steps are defined: printing, cooling, and detachment steps (see Figure 2. 9).

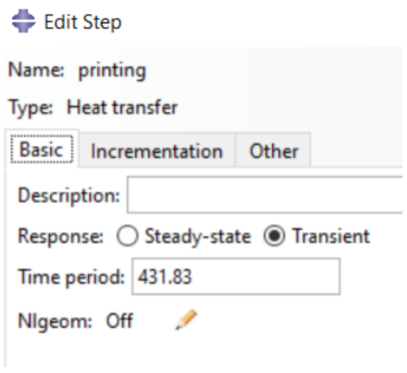
- The printing step represents the material extrusion process. The time length of the printing step varies with different 3D printed parts, infill density, infill patterns, layer thickness, etc. For this step, increments of 4 seconds were chosen to reduce the computational time of the simulation.
- The next step is the cooling, in which the material extrusion process has ended, and the part remains attached to the build plate at the bed temperature. The chosen time length, in this case, is 40 seconds with increments of 10 seconds.
- The final step in the simulation is detachment. Here, the build plate is supposed to be at ambient temperature, and the part is detached from it. The chosen time length again is 40 seconds and increments of 10 seconds.

In the three different steps, the transient option is chosen in order to know the evolution of the temperature field with time (see Figure 2. 10). The maximum number of increments of the step and the max allowable temperature change per increment is a number high enough to avoid the termination of the simulation in normal conditions (see Figure 2. 11).


Step Manager

	Name	Procedure	Nlgeom	Time
✓	Initial	(Initial)	N/A	N/A
✓	printing	Heat transfer (Transient)	N/A	431.83
✓	cooling	Heat transfer (Transient)	N/A	40
✓	detachment	Heat transfer (Transient)	N/A	40

Figure 2. 9 Different steps in the thermal analysis



Edit Step

Name: printing

Type: Heat transfer

Basic | Incrementation | Other

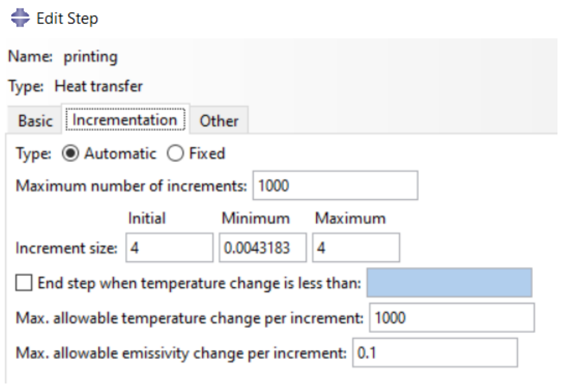
Description:

Response: ○ Steady-state [✓ ● Transient]

Time period: 431.83

Nlgeom: Off

Figure 2. 10 Basic tab in the step editor



Edit Step

Name: printing

Type: Heat transfer

Basic | Incrementation | Other

Type: [✓ ● Automatic] ○ Fixed

Maximum number of increments: 1000

	Initial	Minimum	Maximum
Increment size:	4	0.0043183	4

☐ End step when temperature change is less than:

Max. allowable temperature change per increment: 1000

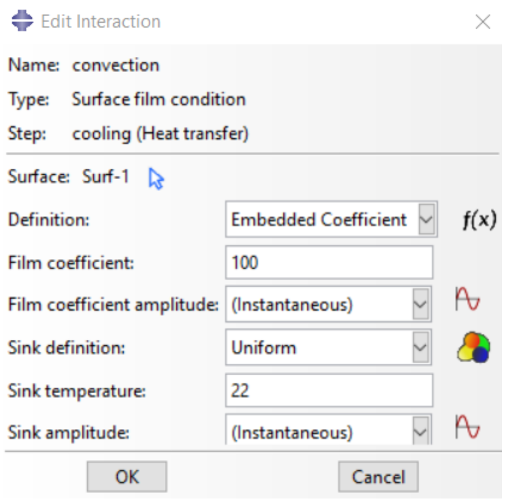
Max. allowable emissivity change per increment: 0.1

Figure 2. 11 Increment definition of the step

Interaction module:

Here the convection and thermal radiation between the part and the ambient must be defined. We only define these interactions for the cooling and detachment steps; for the printing step, these interactions are defined with the am modeler plugin (explained later).

For these interactions, we must choose the involved surfaces in the interaction, the ambient temperature, the convection coefficient (or film coefficient), and the emissivity (see Figure 2. 13 and Figure 2. 12).



Edit Interaction

Name: convection

Type: Surface film condition

Step: cooling (Heat transfer)

Surface: Surf-1

Definition: Embedded Coefficient

Film coefficient: 100

Film coefficient amplitude: (Instantaneous)

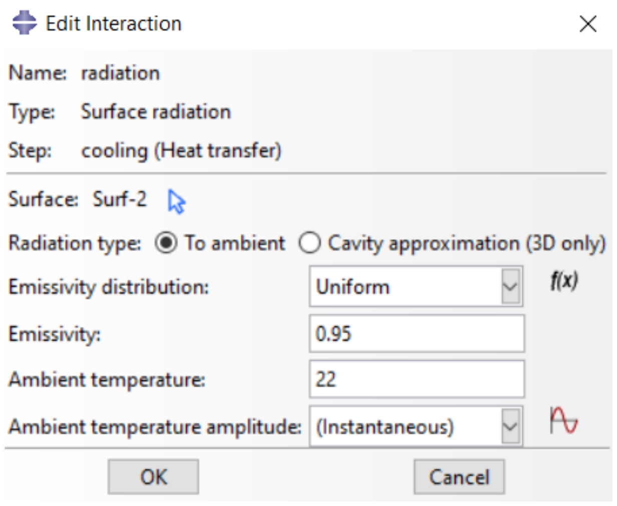
Sink definition: Uniform

Sink temperature: 22

Sink amplitude: (Instantaneous)

OK | Cancel

Figure 2. 13 Convection interaction



Edit Interaction

Name: radiation

Type: Surface radiation

Step: cooling (Heat transfer)

Surface: Surf-2

Radiation type: [✓ ● To ambient] ○ Cavity approximation (3D only)

Emissivity distribution: Uniform

Emissivity: 0.95

Ambient temperature: 22

Ambient temperature amplitude: (Instantaneous)

OK | Cancel

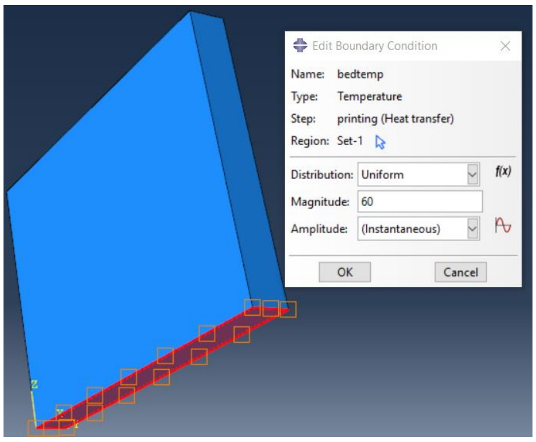
Figure 2. 12 Radiation interaction

Load module:

In this module, the build plate temperature and the initial temperature must be defined. For the build plate temperature, we simply select the bottom surface in contact with the build plate and apply a uniform temperature boundary condition at the bed temperature (see Figure 2. 14).

During the additive manufacturing process, newly deposited material comes in at a given initial temperature. This temperature is defined for the whole part in the initial step. We suppose this temperature as the nozzle temperature of the process (see Figure 2. 15).

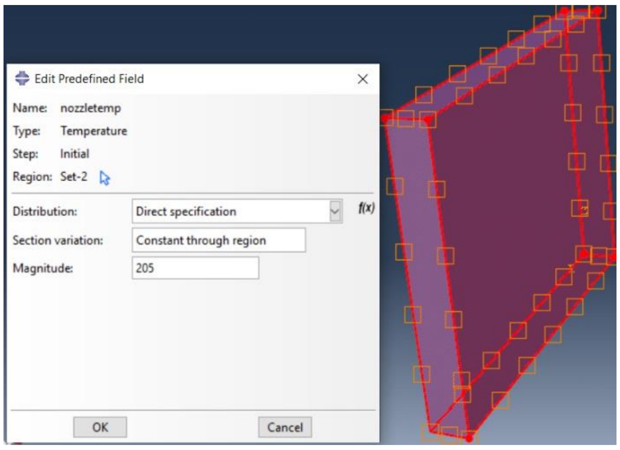
Abaqus applies the initial temperature to the material inside these elements when they are first activated.


Edit Boundary Condition

Name: bedtemp
Type: Temperature
Step: printing (Heat transfer)
Region: Set-1

Distribution: Uniform
Magnitude: 60
Amplitude: (Instantaneous)

Figure 2. 14 Boundary condition for build plate temperature


Edit Predefined Field

Name: nozzletemp
Type: Temperature
Step: Initial
Region: Set-2

Distribution: Direct specification
Section variation: Constant through region
Magnitude: 205

Figure 2. 15 Definition of the initial temperature for material extrusion

Mesh module:

The part has to be meshed to perform the numerical calculations. To perform the simulation accurately, the element height has to be equal to or a submultiple of the layer thickness (or layer height) (Cattenone et al., 2019). The other dimensions of the elements are recommended to be equal or submultiples of the bead width in the process.

In this work, the element height is equal to the layer height and the two remaining dimensions are equal to the bead width (see Figure 2. 16). The dimension selection of the elements was done with the seed edges tool. For the thermal analysis, we must use heat transfer elements, in the element type tool (see Figure 2. 17).

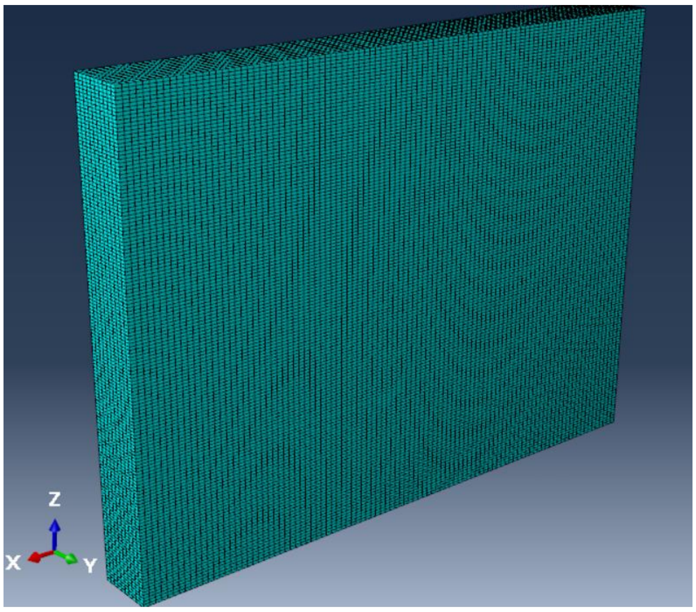
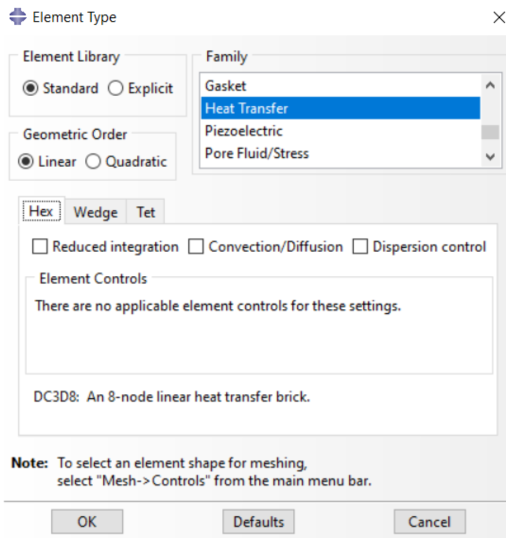


Figure 2. 16 Meshing of the wall


Element Type

Element Library: [✓ Standard] Explicit
Family: Gasket, [✓ Heat Transfer], Piezoelectric, Pore Fluid/Stress
Geometric Order: [✓ Linear] Quadratic

Hex | Wedge | Tet

☐ Reduced integration ☐ Convection/Diffusion ☐ Dispersion control

Element Controls
There are no applicable element controls for these settings.

DC3D8: An 8-node linear heat transfer brick.

Note: To select an element shape for meshing, select "Mesh->Controls" from the main menu bar.

Figure 2. 17 Element type selection

Field output requests:

Here we indicate which outputs we want to extract from the model. In the case of the thermal model, these outputs are nodal temperature (NT), heat flux vector (HFL), and reaction fluxes (RFL).

The reminder modules are the job module and the visualization module. In the job module, we run the simulation and in the visualization module, we can access the results of the simulation.

2.1.4.2 Mechanical analysis

For the mechanical analysis, we use the option copy model to duplicate the thermal model. We have to change the step, interaction, load, and mesh modules and the field output requests of the analysis.

Step module:

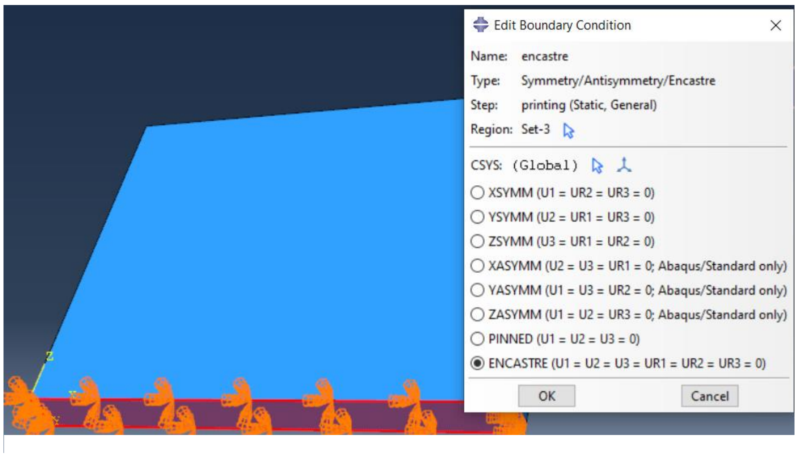
In this module, we use the option ''replace'' to change the thermal steps for general statics. The total time of the steps and the intervals remain the same as in the thermal model.

Interaction module:

In the interaction module, we simply delete the convection and radiation interactions. No mechanical interactions take place in this case.

Load module:

Here, the build temperature boundary condition and the nozzle temperature as a predefined field are deleted. Instead, the boundary condition of encastre in the bottom of the part in contact with the build plate is defined (see Figure 2. 18). In addition, the output of the thermal analysis has to be defined as a predefined field of the mechanical model (see Figure 2. 19), to consider the deformations due to the thermal gradients. In the distribution option of the predefined fields, we have to choose "From results or output database file", and then select the .odb file that we want to include in the structural analysis. It is also important to modify the predefined field to apply the correct step of the thermal analysis to the steps in the mechanical one.



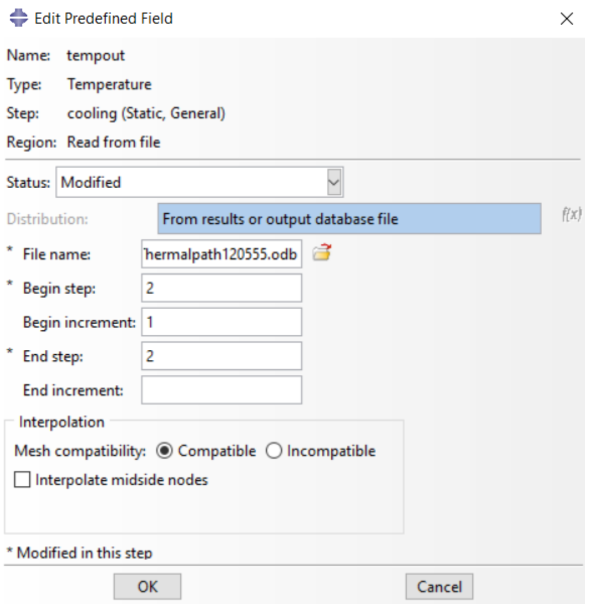
Edit Boundary Condition

Name: encastre
Type: Symmetry/Antisymmetry/Encastre
Step: printing (Static, General)
Region: Set-3

CSYS: (Global)

- XSYMM (U1 = UR2 = UR3 = 0)
- YSYMM (U2 = UR1 = UR3 = 0)
- ZSYMM (U3 = UR1 = UR2 = 0)
- XASYMM (U2 = U3 = UR1 = 0; Abaqus/Standard only)
- YASYMM (U1 = U3 = UR2 = 0; Abaqus/Standard only)
- ZASYMM (U1 = U2 = UR3 = 0; Abaqus/Standard only)
- PINNED (U1 = U2 = U3 = 0)
- [✓ ENCASTRE (U1 = U2 = U3 = UR1 = UR2 = UR3 = 0)]

Figure 2. 18 Encastre boundary condition in the mechanical analysis



Edit Predefined Field

Name: tempout
Type: Temperature
Step: cooling (Static, General)
Region: Read from file

Status: Modified
Distribution: From results or output database file
* File name: hermalpath120555.odb
* Begin step: 2
Begin increment: 1
* End step: 2
End increment:

Interpolation
Mesh compatibility: [✓ Compatible] Incompatible
☐ Interpolate midside nodes

* Modified in this step

OK Cancel

Figure 2. 19 Thermal analysis output as an input for the mechanical

Mesh module:

Here, in the assign element type tab, we must change the heat transfer elements for 3D stress elements.

Field output requests:

In the case of the mechanical model, the requested outputs are the stress components (S), the total strain components (E), and the translations and rotations (UT and UR).

2.1.4.3 AM modeler abaqus plugin

In the 3D printing process, material addition occurs, and a sequential thermos-mechanical analysis has to be performed in order to know the distortions of the parts. Since this problem changes with time and is also temperature dependent, a huge amount of data is needed, that will be used by user subroutines. To help with the integration of the data into the subroutines in an easy manner, various data structures have been developed as keywords in the input file of the analysis.

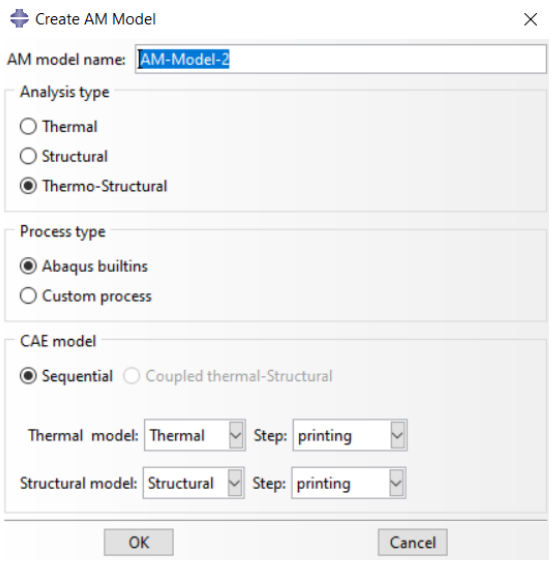
The AM modeler plug-in is available to guide the definition of the keywords from the CAE. In the following of this section, the different sections of the plug-in will be explained.

- **Creating an AM Model** (see Figure 2. 20): here, we can choose between thermal, structural, and thermo-structural analysis types. The process types are Abaqus builtins (predefined types of AM processes) and custom processes (you need to define the types yourself). In the present work, we are simulating FDM, so we select Abaqus builtins.

  Finally, we have to choose between sequential or coupled thermo-structural analysis. In the sequential option, the temperature distribution of the part is calculated first, and the results of this analysis are used as a thermal load in the structural analysis. Thermal stress calculates the shrinkage of the part due to

temperature changes. This method can be applied when the deformation of the structure does not affect the temperature distribution.
However, if the heat flow greatly varies due to thermal deformation, a coupled analysis should be performed, since thermal deformation and thermal distribution will be simultaneously changing in correlation. In the present work, sequential analysis was chosen, because it is assumed that the thermal distribution does not vary with thermal deformation.

Once the AM model is created, we can see the AM modeler tab as in Figure 2. 21. It has three main sections: *(i)data setup*, *(ii)model setup, and (iii)simulation setup.* In the data setup section, types for different data structures are mentioned: parameter tables, property tables, and event series.


Create AM Model

AM model name: AM-Model-2

Analysis type
Thermal
Structural
[✓ Thermo-Structural]

Process type
[✓ Abaqus builtins]
Custom process

CAE model
[✓ Sequential] Coupled thermal-Structural

Thermal model: Thermal Step: printing

Structural model: Structural Step: printing

OK Cancel

Figure 2. 20 Creating an AM model

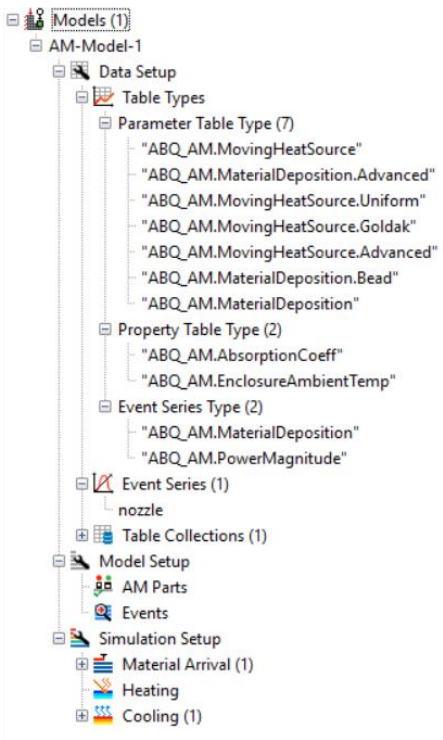
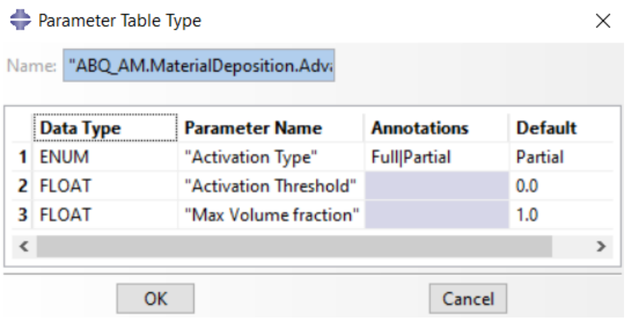

```
Models (1)
  AM-Model-1
    Data Setup
      Table Types
        Parameter Table Type (7)
          "ABQ_AM.MovingHeatSource"
          "ABQ_AM.MaterialDeposition.Advanced"
          "ABQ_AM.MovingHeatSource.Uniform"
          "ABQ_AM.MovingHeatSource.Goldak"
          "ABQ_AM.MovingHeatSource.Advanced"
          "ABQ_AM.MaterialDeposition.Bead"
          "ABQ_AM.MaterialDeposition"
        Property Table Type (2)
          "ABQ_AM.AbsorptionCoeff"
          "ABQ_AM.EnclosureAmbientTemp"
        Event Series Type (2)
          "ABQ_AM.MaterialDeposition"
          "ABQ_AM.PowerMagnitude"
      Event Series (1)
        nozzle
      Table Collections (1)
    Model Setup
      AM Parts
      Events
    Simulation Setup
      Material Arrival (1)
      Heating
      Cooling (1)
```

Figure 2. 21 AM modeler tab

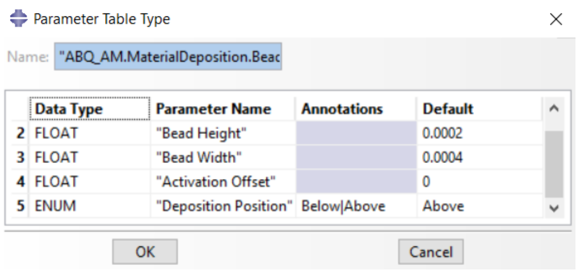
- **Data setup**: in this section, there are three different data structures:

  Parameter tables: are used to group process specific parameters that do not depend on time, space, or material state. The amount and type of parameters that are included in the parameter table, are determined by the parameter table type. For some common AM processes, the required parameter table types are predefined. The predefined parameter table types are shown in Figure 2. 21 (seven different). For simulating the FDM process, we only need to define three of them: *(i)*"ABQ_AM.MaterialDepositionAdvanced" (see Figure 2. 23), in which we can set the activation type to "full" to indicate that elements are activated and their volume fractions set to one, or set the activation type to "partial" to indicate that elements are activated when material deposits inside the element and the volume fractions progressively increase from the minimum volume fraction threshold to one as more material deposits; *(ii)* "ABQ_AM.MaterialDepositionBead (see Figure 2. 22)", where we specify the stack direction, bead height, bead width, and deposition position; *(iii)* "ABQ_AM.MaterialDeposition (see Figure 2. 24)", in which we include the event series with the nozzle movement during the printing and we select "Bead" for the FDM process instead of "Roller".



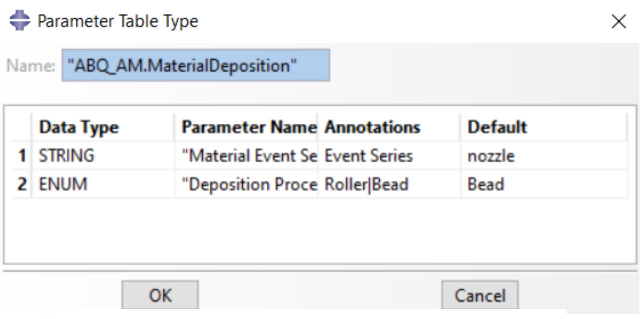
	Data Type	Parameter Name	Annotations	Default
1	ENUM	"Activation Type"	Full|Partial	Partial
2	FLOAT	"Activation Threshold"		0.0
3	FLOAT	"Max Volume fraction"		1.0

Figure 2. 23 Advanced material deposition table



	Data Type	Parameter Name	Annotations	Default
2	FLOAT	"Bead Height"		0.0002
3	FLOAT	"Bead Width"		0.0004
4	FLOAT	"Activation Offset"		0
5	ENUM	"Deposition Position"	Below|Above	Above

Figure 2. 22 Bead features of the deposition process

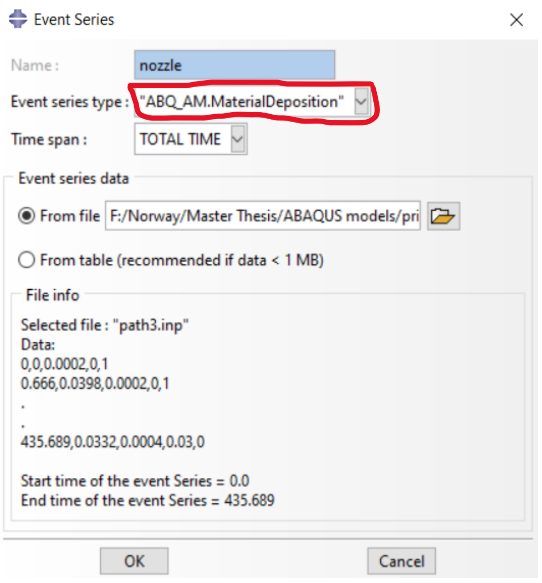


	Data Type	Parameter Name	Annotations	Default
1	STRING	"Material Event Se	Event Series	nozzle
2	ENUM	"Deposition Proce	Roller|Bead	Bead

Figure 2. 24 Nozzle event series and bead process

Property tables: are used to define dependent parameters. These parameters can depend on temperature, field, and state dependent variables. Parameters such as material properties and a film coefficient can be defined this way. In the present work, no property tables are needed because the temperature dependent properties of PLA were defined in the property module in FEA Abaqus.

Event series: event series were explained in section 2.1.3.1. There are predefined options for event series types (see Figure 2. 25). In this present work, the geometries are quite simple, so the event series were created using excel. However, in order to simulate more complex geometries, it is recommended to convert the G-code of the part into an event series format by scripting.


Event Series

Name: nozzle

Event series type: "ABQ_AM.MaterialDeposition"

Time span: TOTAL TIME

Event series data

● From file F:/Norway/Master Thesis/ABAQUS models/pri

○ From table (recommended if data < 1 MB)

File info

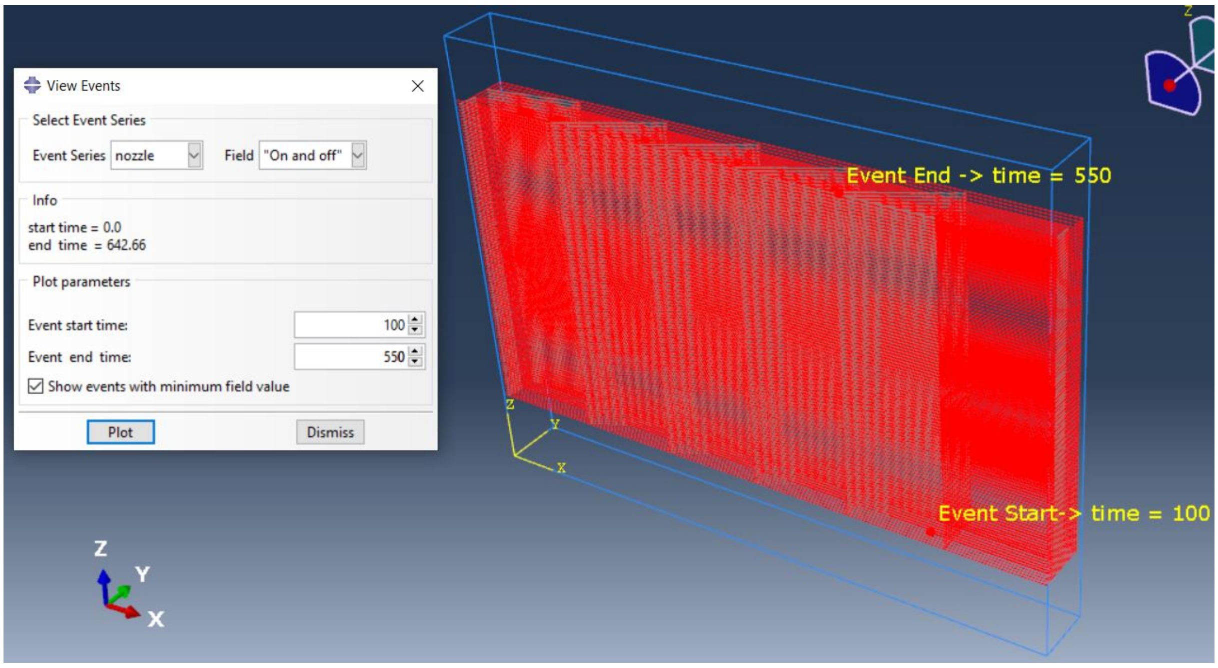
Selected file: "path3.inp"
Data:
0,0,0.0002,0,1
0.666,0.0398,0.0002,0,1
.
.
435.689,0.0332,0.0004,0.03,0

Start time of the event Series = 0.0
End time of the event Series = 435.689

Figure 2. 25 Event series definition

Finally, to end the data setup process, we use the table collections. Table collections are containers that encapsulate parameter tables and/or property tables. This way, they can be grouped together for reference in user subroutines. To simulate the FDM process as in the present work, only one table collection containing the three parameter tables is needed.

- **Model setup**: once the data is obtained and defined, we must select the parts to be included in the additive manufacturing analysis. It is also possible to visualize the nozzle movement described by the event series (see Figure 2. 26).


View Events

Select Event Series

Event Series nozzle   Field "On and off"

Info

start time = 0.0
end time = 642.66

Plot parameters

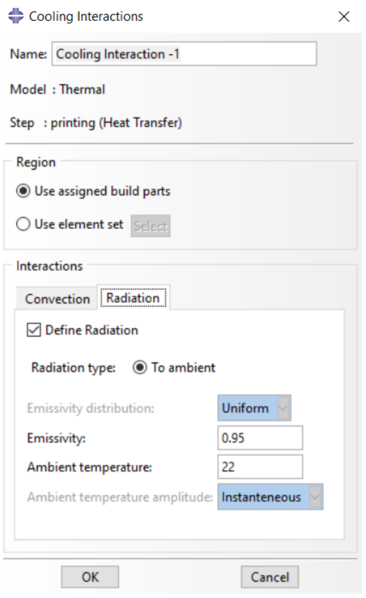
Event start time: 100

Event end time: 550

☑ Show events with minimum field value

Figure 2. 26 View of the nozzle movement between 100 s and 550 s

- **Simulation setup**: in this section, we assign the table collection to define the material arrival process, and we also define cooling by means of convection and radiation (see Figure 2. 27).


Cooling Interactions

Name: Cooling Interaction -1

Model : Thermal

Step : printing (Heat Transfer)

Region

- [✓ Use assigned build parts]
- Use element set

Interactions

Convection | Radiation

[✓ Define Radiation]

Radiation type: [✓ To ambient]

Emissivity distribution:	Uniform
Emissivity:	0.95
Ambient temperature:	22
Ambient temperature amplitude:	Instanteneous

Figure 2. 27 Defining cooling interactions

Once the AM model is completely defined, we must go to the job module and run the thermal model. Once the thermal model is completed, we use it as input for the structural model, and we run it.

3 Simulation results

3.1 First geometry (bead)

In this section, the results of the numerical simulation for the bead geometry are explained. The method for the element activation, in this case, was the model change interaction, as explained in Section 2.1.3.1. This method consists in deactivating the whole model in the first step of the analysis and then continuously activating elements step by step.

The dimensions of the bead are: length = 5 mm; width = 0.4 mm; height = 0.2 mm. The extrusion temperature was supposed 205 ºC; the bed temperature was set to 60 ºC and the ambient temperature was assumed to be 22 ºC. In addition, the printing speed of the bed was supposed to be 60 mm/s, so the printing of the simulated bead would last 0.083 seconds. Then, a cooling step of 5 seconds is computed and finally a detachment step of other 5 seconds in which the deformation of the bead can be appreciated.

3.1.1 Thermal analysis results

The requested field outputs for the thermal analysis are nodal temperature, heat flux vector, element temperature and reaction fluxes. The most important field variable is the nodal temperature (NT), which is going to be the input for the structural analysis. In Figure 3. 1, the variation of the temperature field during the cooling process is shown.

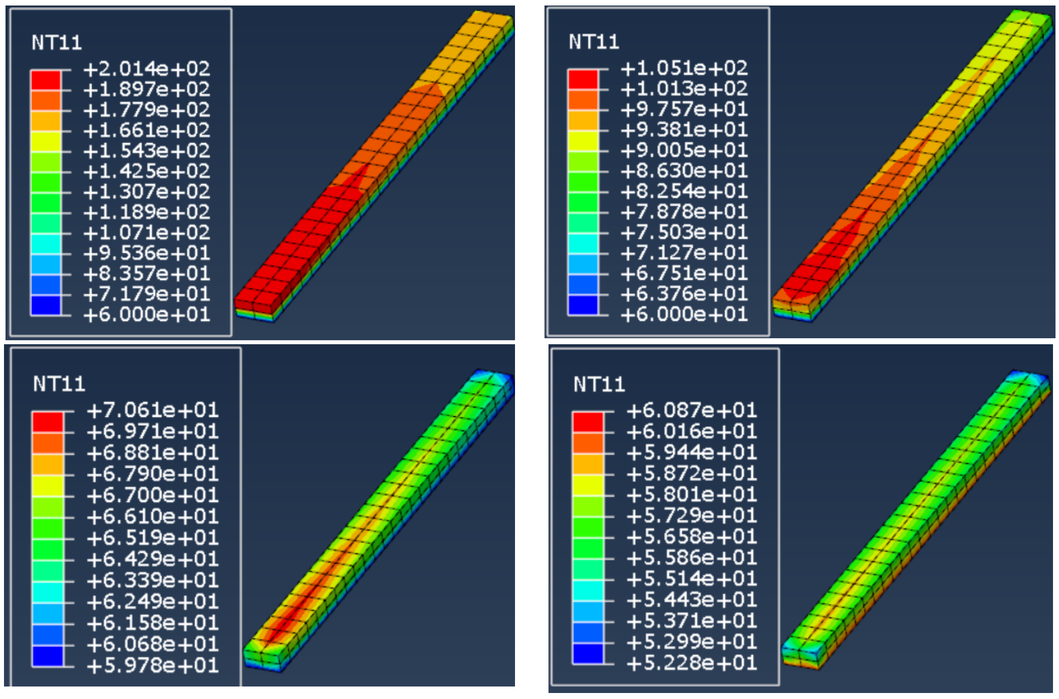


Figure 3. 1 Bead temperature field evolution in the cooling step

It is common to perform a mesh convergence study to validate the results in a finite element analysis. To check the consistency of the solution, the analysis will be performed for 2x2, 4x4, 8x8, and 16x16 elements. In the figure below, the thermal profiles at the end of the cooling process for the different mesh sizes are represented.

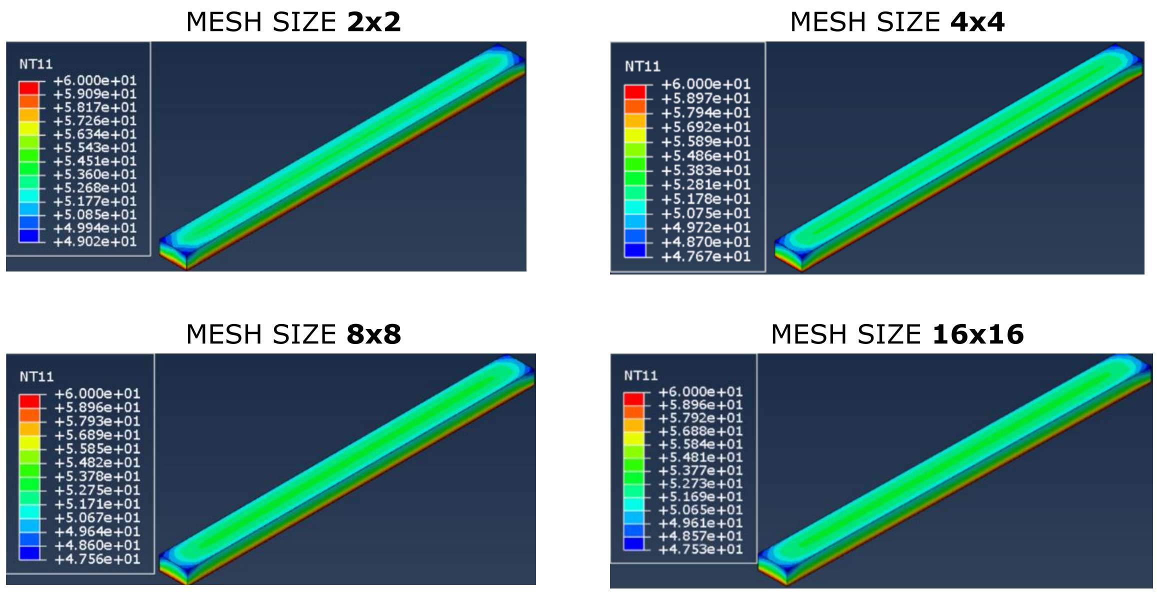


Figure 3. 2 Temperature field at the end of the cooling step for different meshes

From the results in Figure 3. 2, the temperature fields with the different meshes are almost the same. This means that the simulation converges. However, another convergency study is performed. This time, we are going to take three nodes in the middle section of the bead and plot their temperatures over time for the three different meshes. One node is at the top of the bead, another node in the middle, and the last one at the bottom (see Figure 3. 3).

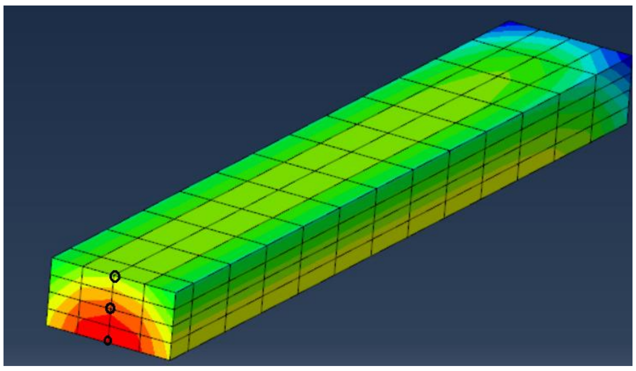


Figure 3. 3 Selected nodes to perform the convergence study

The results for each of the three points are shown in Figure 3. 4, Figure 3. 5, and Figure 3. 6. We can observe that the results for all of the different meshes are very similar. We can conclude that the analysis is convergent. In the bottom node, we can notice a big step between 205 ºC and 60 ºC. This is because of the boundary condition of the build plate temperature.

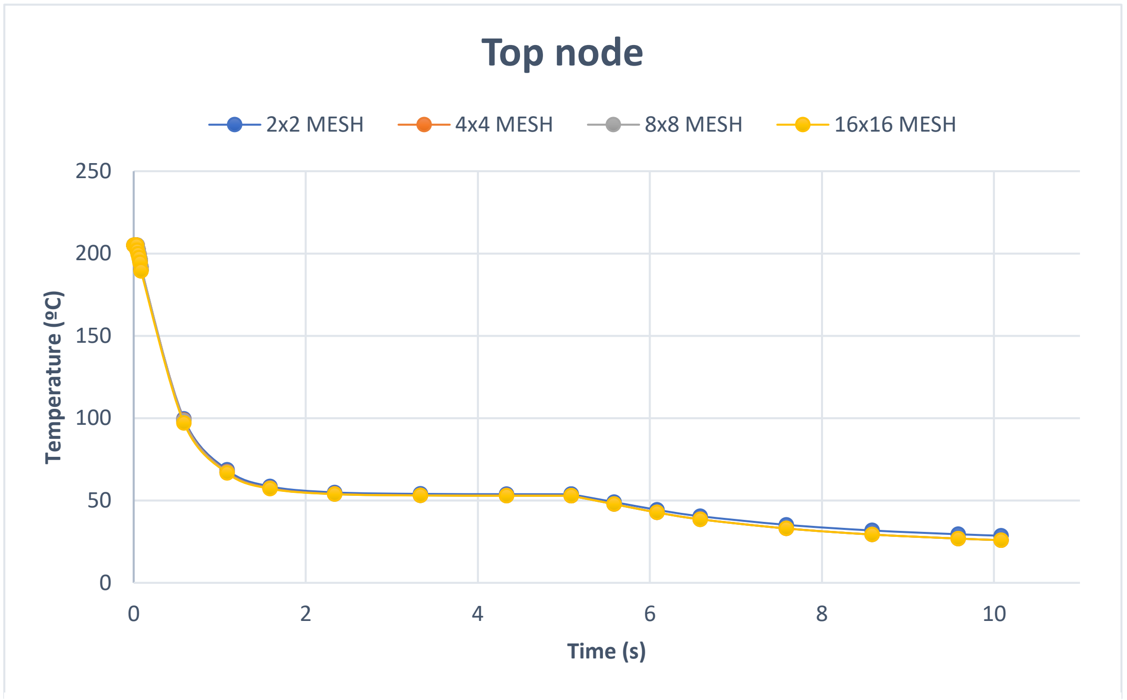


Figure 3. 4 Time-dependent temperature field for the top node and different meshes

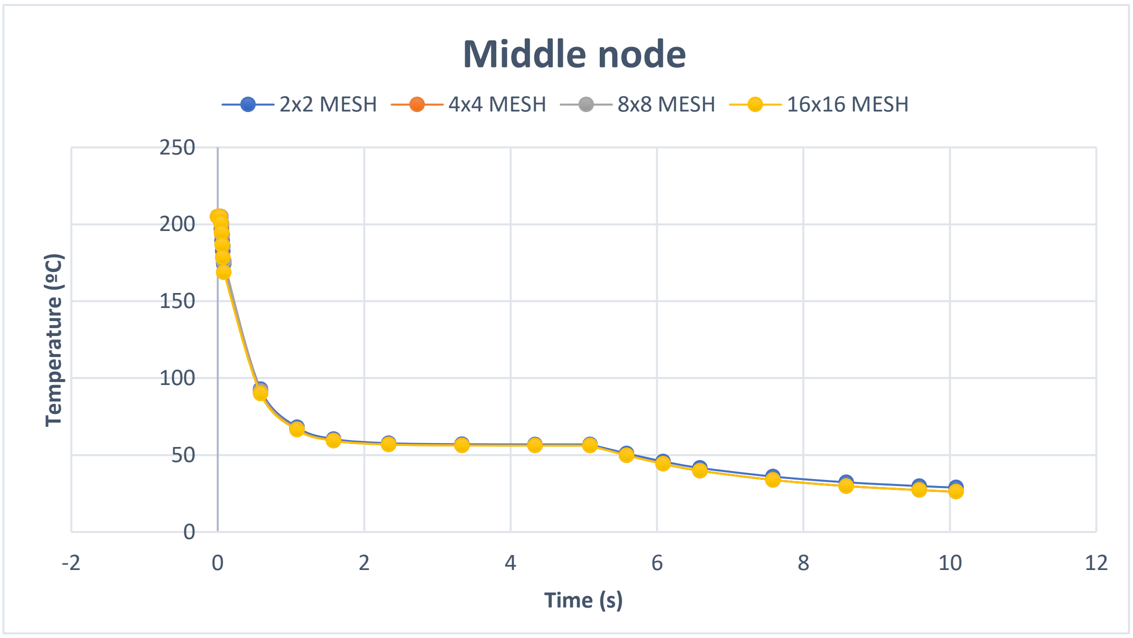


Figure 3. 5 Time-dependent temperature field for the middle node and different meshes

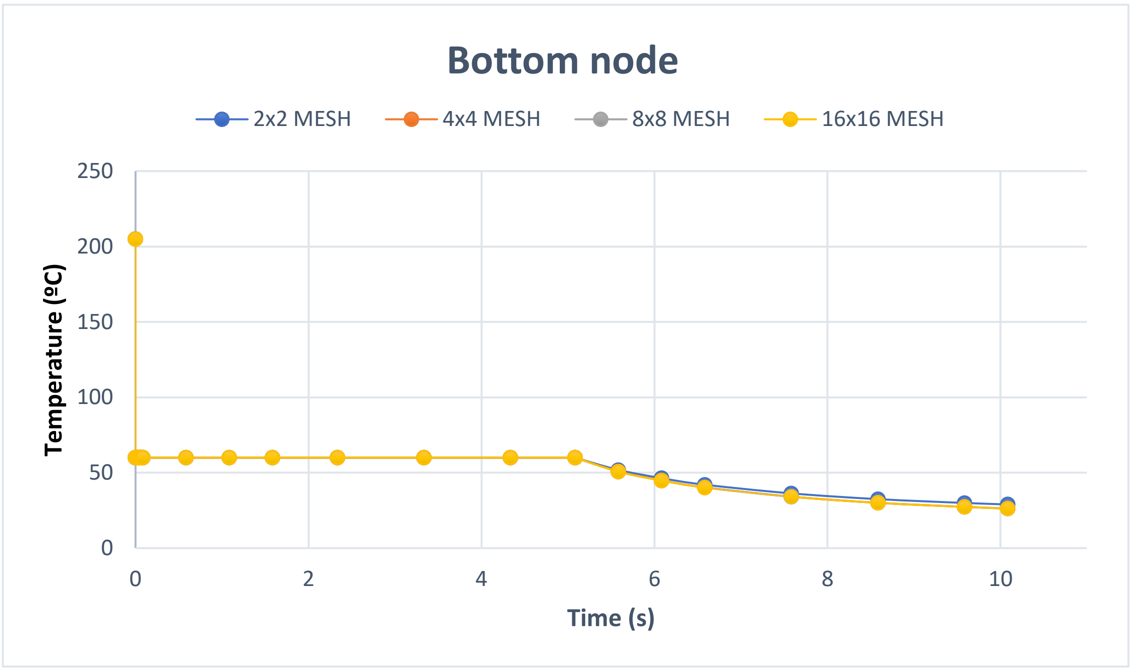


Figure 3. 6 Time-dependent temperature field for the bottom node and different meshes

3.1.2 Structural analysis results

The requested field variables in the structural analysis are stresses, strains, and displacements. We are going to focus on stresses and displacements in the three directions. In Figure 3. 7 and Figure 3. 8 we can see the evolution of the stress field and deformation field in the cooling step respectively.

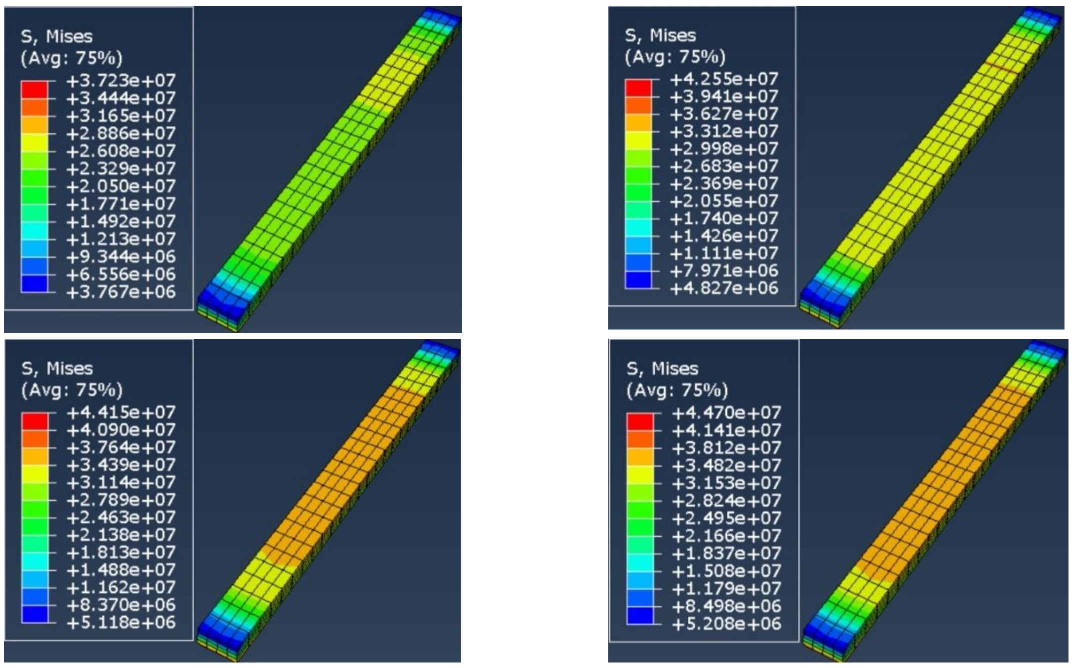


Figure 3. 7 Stress field evolution (misses) in the cooling step (between 0.5 and 2 s)

The fact that the highest stresses occur at the bottom part of the bead is noticeable. There is compressive stress in the top of the bead and tensile stress at the bottom.

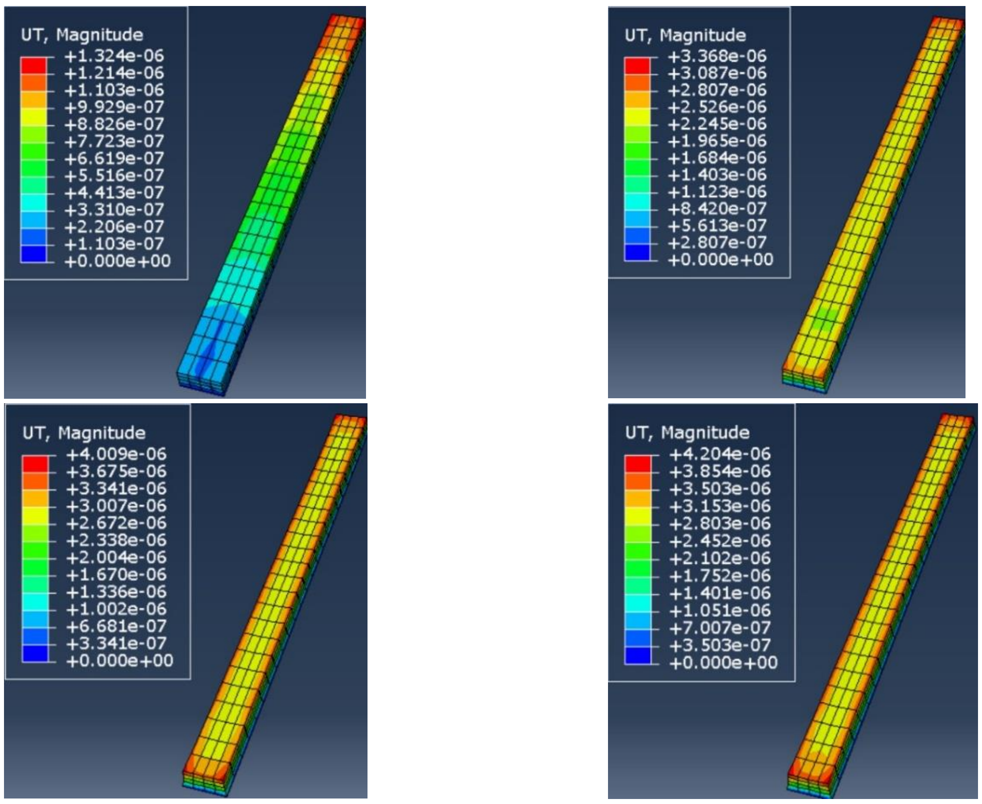


Figure 3. 8 Deformation field at the beginning of the cooling step (deformation scaled by a factor of 10)

As in the thermal analysis, we are going to perform a convergence study for the structural. In this case, the analysis will be performed for 4x4, 8x8, 16x16, and 32x16 element mesh for the deformation results at the end of the cooling step.

We are going to plot one graphic for each deformation direction (x, y, and z). For each direction, a specific node will be picked for the convergence analysis (see Figure 3. 9).

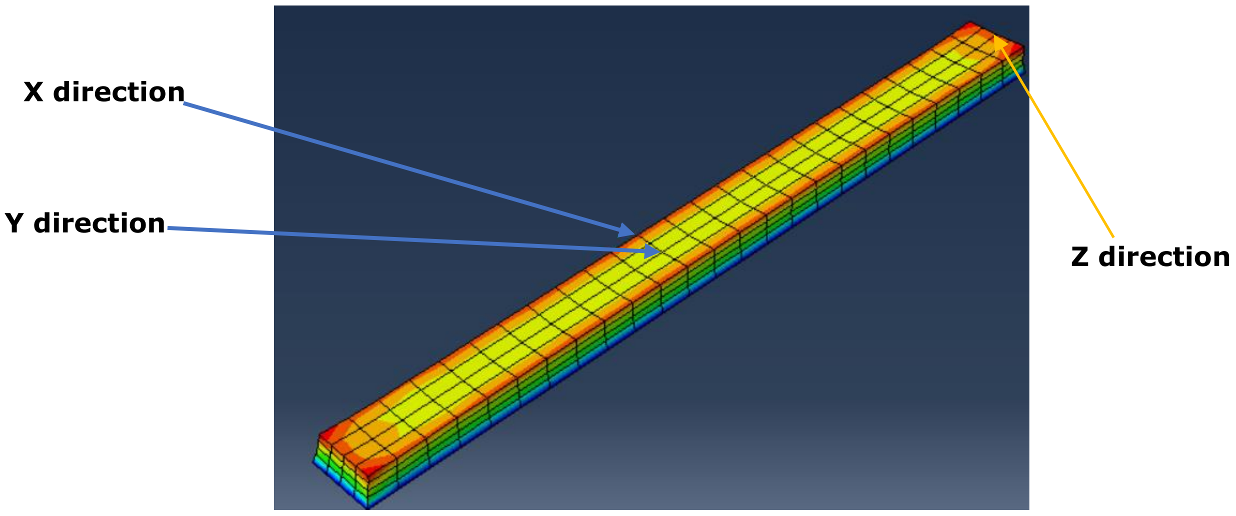


Figure 3. 9 Picked nodes for the structural convergence analysis

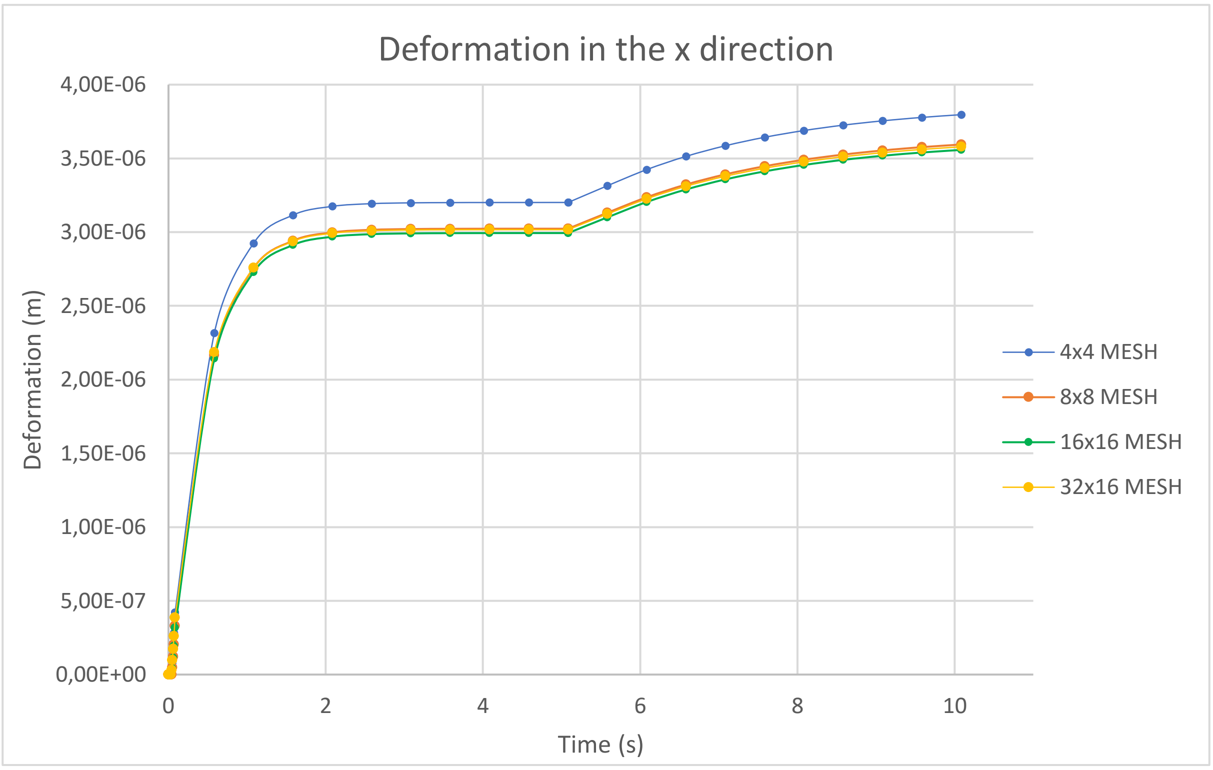


Figure 3. 10 Deformation in the x direction for the different mesh sizes

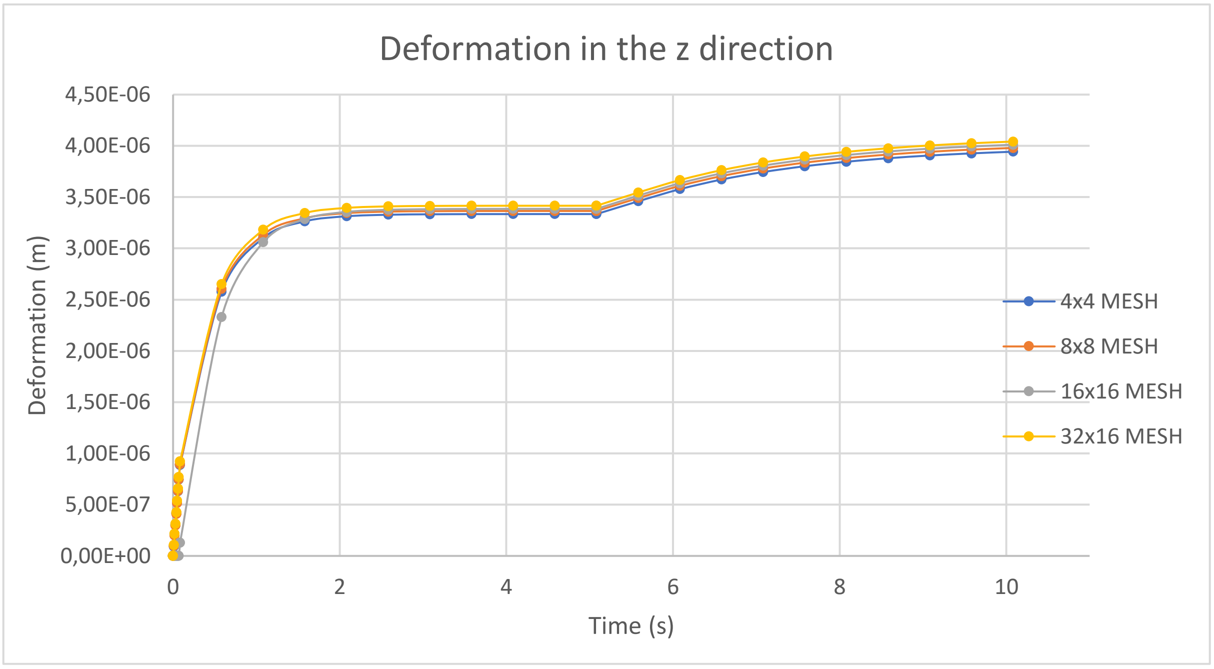


Figure 3. 12 Deformation in the z direction for the different mesh sizes

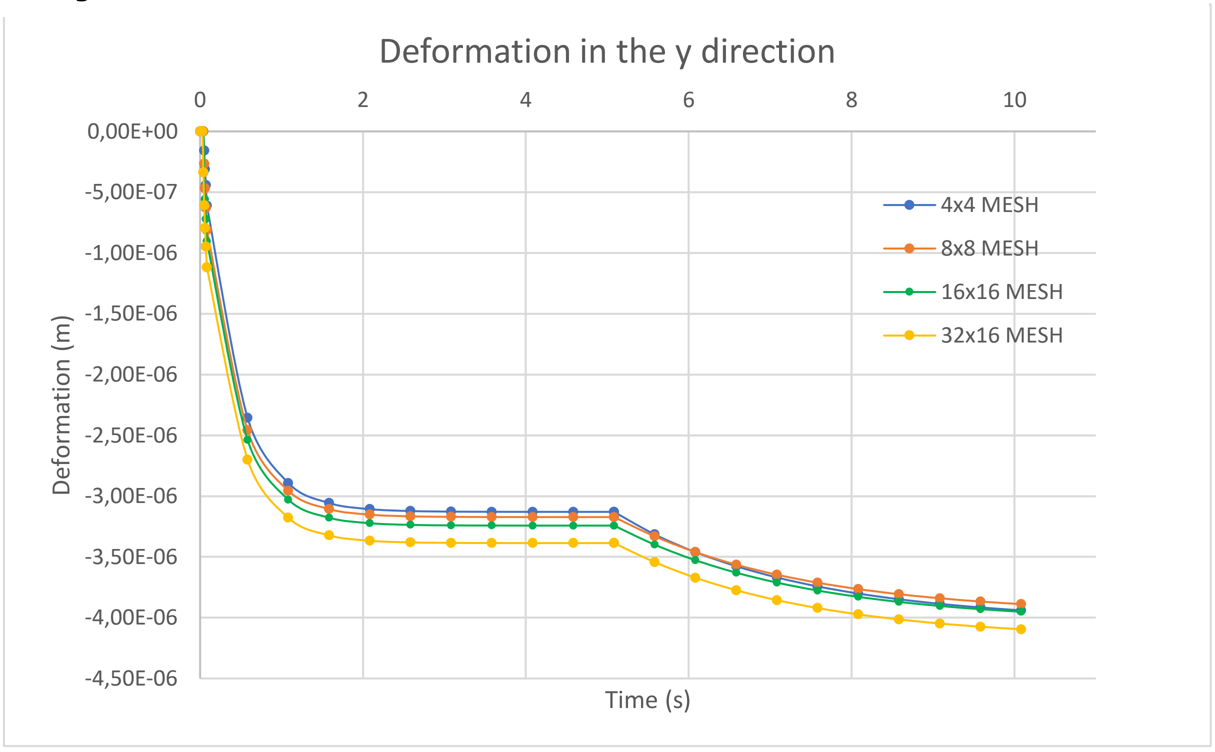


Figure 3. 11 Deformation in the y direction for the different mesh sizes

The results of the convergence analysis are shown in Figure 3. 10, Figure 3. 11, and Figure 3. 12. We can see that the three different components of the deformation have very similar values with the different mesh sizes.

3.1.3 Part distortion evaluation

In this section, we are going to pick the higher deformation values in each direction. Then, the distortion percentage of the part in each direction is calculated. The deformation results are picked from the 32x16 mesh size.

Dimensions of the bead in the different directions:

- ***X direction***: 0.4 mm.
- ***Y direction***: 0.2 mm.
- ***Z direction***: 5 mm.

Maximum displacements in the different directions:

- ***X direction***: 0.00358 mm.
- ***Y direction***: -0.00418 mm.
- ***Z direction***: 0.00414 mm.

Distortion percentage in the different directions:

- **X direction** = $\frac{0.00358}{0.4} \cdot 100 \; = 0.895\%$
- **Y direction** = $\left|\frac{-0.00418}{0.2} \cdot 100\right| \; = \; 2.09\%$ (absolute value)
- **Z direction** = $\frac{0.00414}{5} \cdot 100 = 0.083\%$

The Y direction (stack direction in this case), has the highest percentage of distortion, followed by the X direction and finally the Z direction (material deposition direction in this case).

3.2 Second geometry (flat spring)

In this section, the numerical simulation results of the flat spring are discussed. The element activation, in this case, is performed by the toolpath-mesh intersection module integrated into the AM modeler plugin described in section 2.1.4.3. For this reason, the toolpath describing the nozzle movement is introduced with an event series format and then the software activates elements based on this event series and the user subroutine UEPACTIVATIONVOL. The toolpath used for the flat spring is divided into a contour path and an infill path.

- It has two **contour** beads (see Figure 3. 13) delimiting the perimeter of the part.

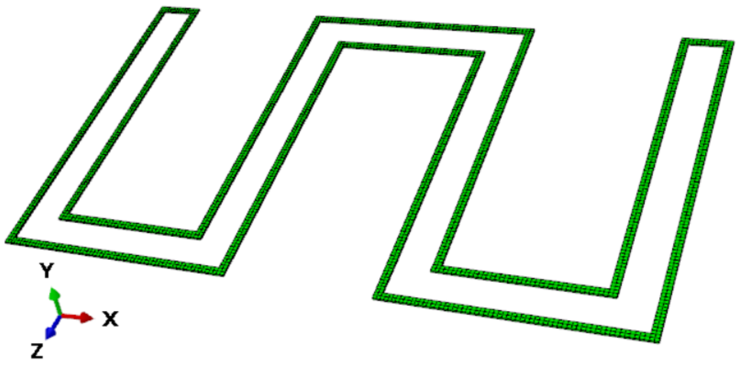


Figure 3. 13 Contour beads in the first layer of the flat spring

- The **infill pattern** is bidirectional and has a raster angle between layers of 90º. The stack direction is Y in this case (see Figure 3. 13). The dimension of the flat spring in this direction is 1 mm. Considering a layer height of 0.2 mm, five layers are needed to build the part. In the first layer, the extrusion direction is the X, in the second, the Z direction. This pattern is repeated until the part is completely finished. The extrusion speed is assumed to be 60 mm/s and the infill density of the part is 100% (all the elements of the model are activated). The infill pattern is shown in Figure 3. 14. The total time of the printing is 232.85 seconds.

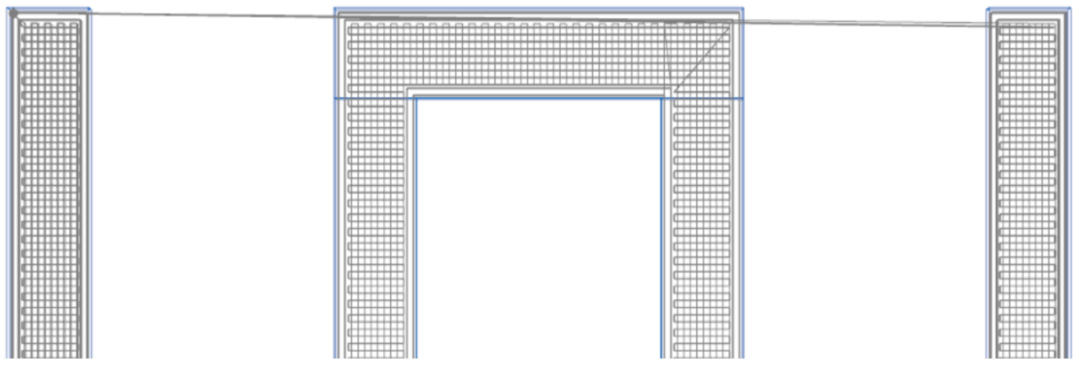


Figure 3. 14 Infill pattern used in the flat spring printing

- **Meshing strategy**: in order to correctly reproduce the printing process, it is necessary that the element height is equal to or a submultiple of the filament height. It is also highly recommended that the element width is equal to or a submultiple of the filament width (Cattenone et al., 2019). So, in this case, the element height of the mesh is set to the value of the layer height (0.2 mm) and the element width and element length are set to the value of the extruded bead width (0.4 mm). The mesh is shown in Figure 3. 15. In this case, we are not going to perform the analysis with different mesh sizes because it hardly affects the final results (Cattenone et al., 2019) and would suppose a high computational cost.

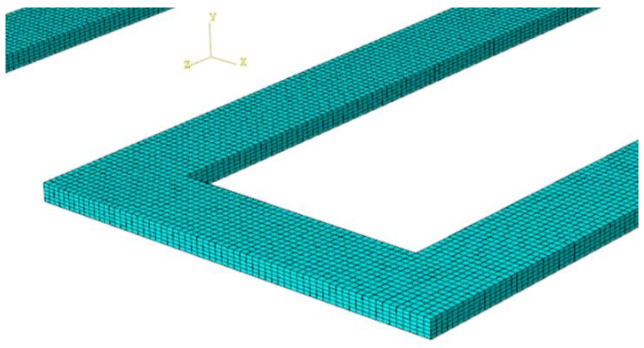


Figure 3. 15 Meshing strategy for the flat spring analysis

3.2.1 Influence of the time step in the analysis

Time step in the printing analysis has a noticeable effect on the performance of the simulation. Firstly, it strongly influences the activation temperature. The activation temperature should be equal to the extrusion temperature of the nozzle, but it has been proven that the higher the time step, the lower the activation temperature (see Figure 3. 16).

Some dispersion metrics of the data are calculated: $Variance\ (\sigma^2) = 1548.7$; $Standard\ deviation\ (\sigma) = 39.35\ ºC$; $Coefficient\ of\ variation\ (CV) = 31.6\ \%$.

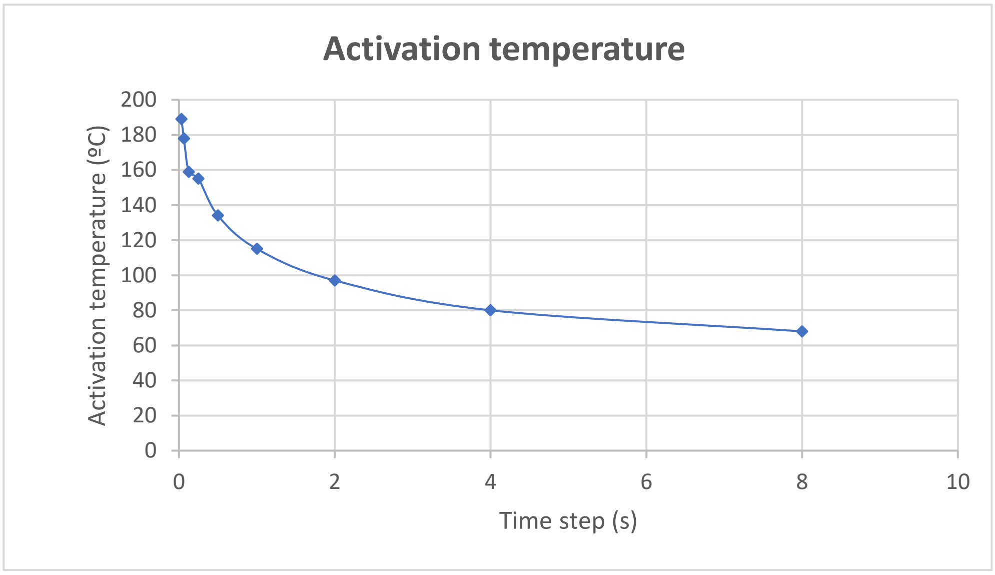


Figure 3. 16 Element activation temperature as a function of time step

Now, a study of the influence of the time step on the maximum stress values (von misses criterion) is performed (see Figure 3. 17). The dispersion metrics in this case have the following values: $Variance\ (\sigma^2) = 6.546$; $Standard\ deviation\ (\sigma) = 2.56$; $Coefficient\ of\ variation\ (CV) = 10.4\%$. We can conclude that the time step influences the thermal analysis more than the structural one.

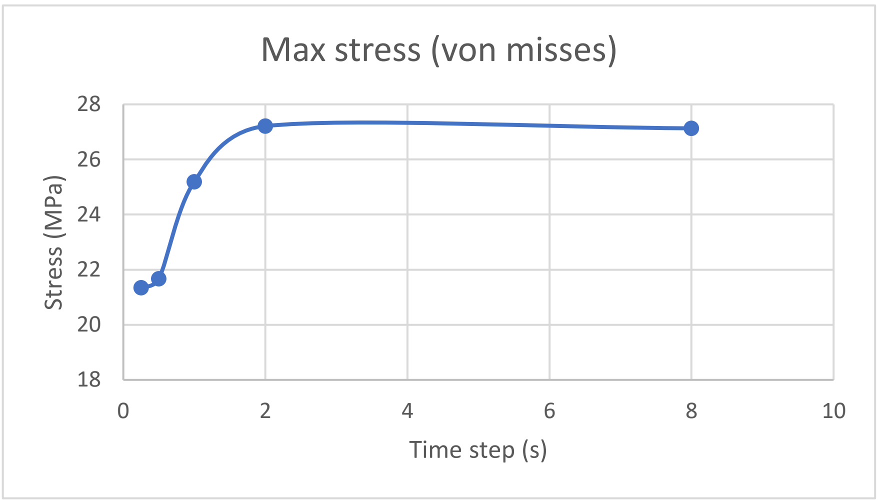


Figure 3. 17 Maximum stress at the end of the cooling step with different time steps for the analysis

In addition, the time step also greatly influences the computational time required to perform the thermal and structural simulation. The smaller the time interval the greater the computational time required (see Figure 3. 18).

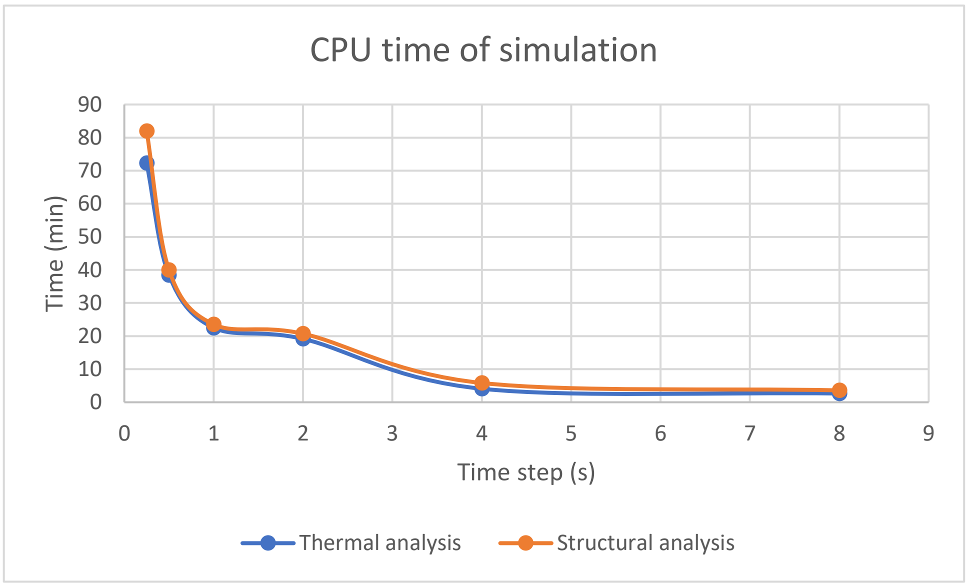


Figure 3. 18 Computational time for the analysis as a function of time step

3.2.2 Thermal analysis results

In the thermal analysis, we want to know the temperature distribution in order to use it as input in the structural analysis, so the nodal temperature is the main output. For the thermal analysis, the main temperatures of the process were set to the following values:

- Extrusion temperature = 205 ºC
- Bed temperature = 60 ºC
- Ambient temperature = 22 ºC

In Figure 3. 19, a section view of the temperature field is shown. It corresponds to the printing of the last layer. We can see how the bottom layers are reheated as molten material is deposited above them.

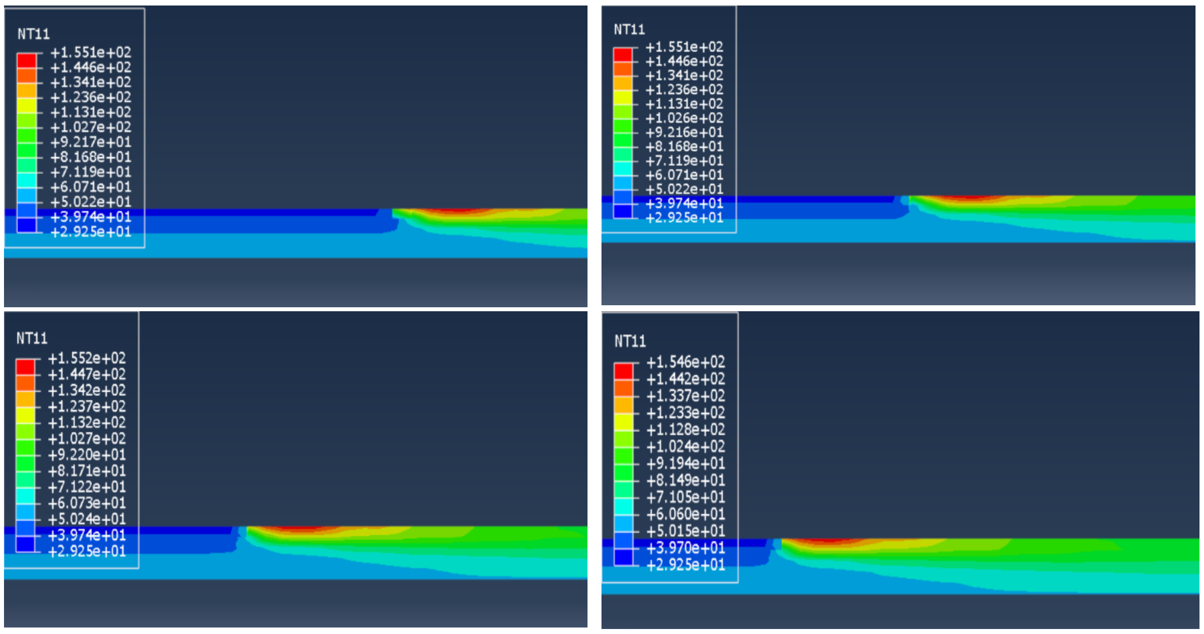


Figure 3. 19 Thermal distribution as the material is continuously added (top layer)

In the FDM process, we can appreciate how the layers below are reheated by the addition of layers on the top. These reheating processes contribute to inducing thermal gradients in the part and therefore residual stresses. We are going to compare the thermal evolution of nodes on the top of the first layer, third layer, and fifth layer (see Figure 3. 20) to see the reheating process in FDM printing.

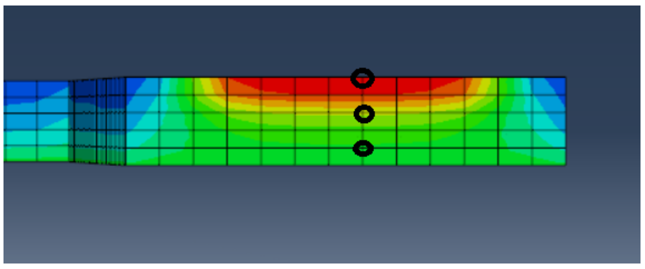


Figure 3. 20 Selected nodes to plot their temperature profiles

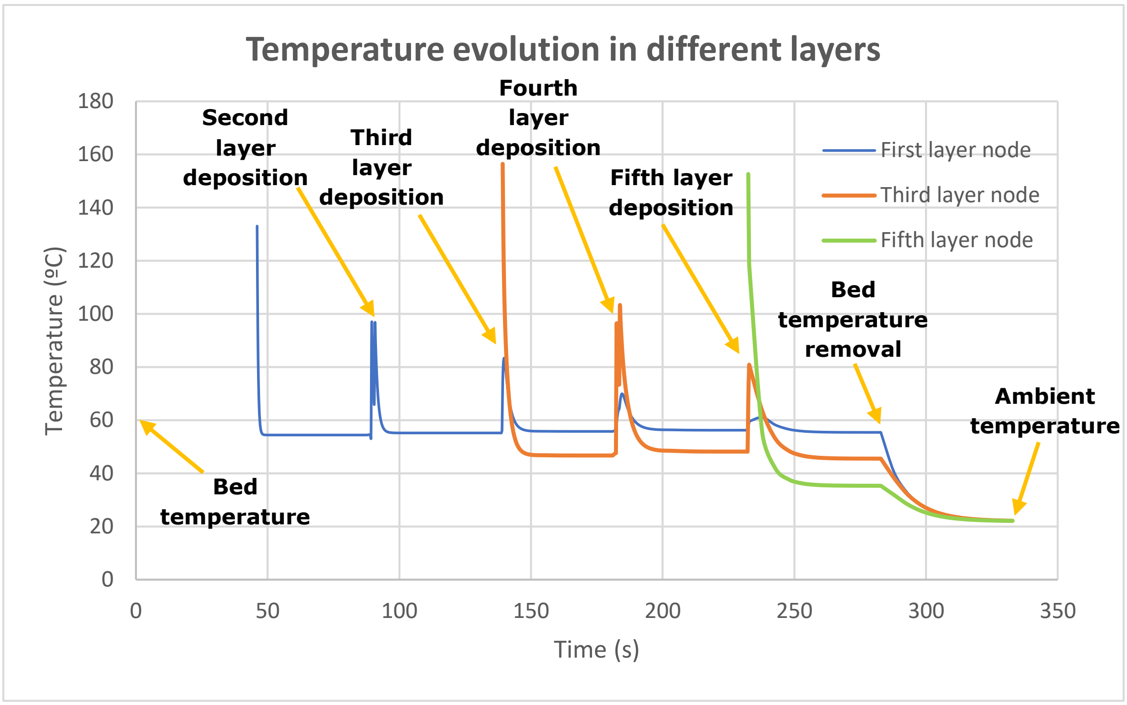


Figure 3. 21 Thermal profiles of three nodes in different printed layers

In Figure 3. 21 we can see the thermal profiles for points in different layers. These results were achieved with a time step of 0.25 seconds, so the activation temperature is less than 205 ºC (extrusion temperature).

- In the first layer, we can see the first temperature drop between the activation temperature and a temperature a bit lower than the bed temperature (60 ºC). Then, a reheat takes place due to the deposition of the second layer. The same process is repeated for the third, fourth and fifth layers. The reheating temperature decreases layer by layer because there is more distance between the first layer and the new extruded layers.
- For the third layer node, we can also see the temperature drop at the beginning between the activation temperature and a temperature around 47 ºC. Then, two reheat processes take place due to the deposition of the fourth and fifth layers.
- Finally, for the fifth layer node, we only have the initial temperature drop because is the top layer.
- At time of 282 seconds, the bed temperature condition is removed and there is a final cooling of the part to the ambient temperature.

3.2.3 Structural analysis results

In Figure 3. 22 and Figure 3. 23 we can see the results for the stress and deformation fields in the flat spring geometry. We can see the evolution of the stress field at the beginning of the cooling step and the evolution of the deformation field at the end of the detachment step, respectively.

For the stress field evolution, we can see that the right side of the part has the lowest stress values. This is because it was the last printed region of the part and the cooling happens later than in the rest. In addition, we can notice an increase in the residual stress values due to the cooling process.

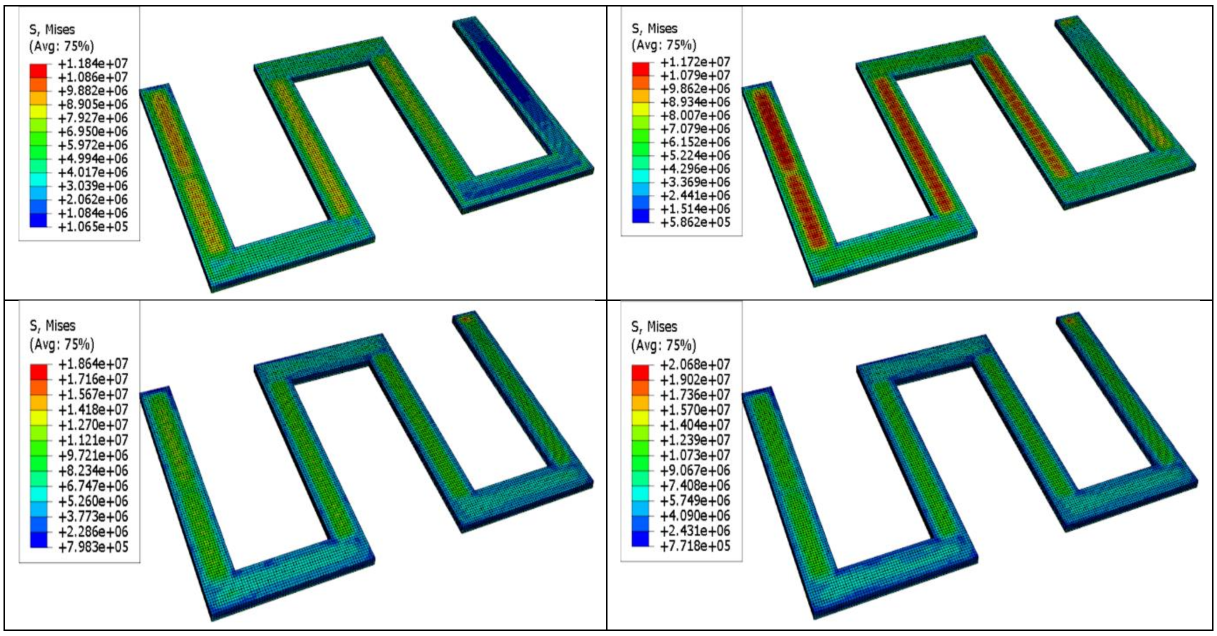


Figure 3. 22 Von Misses stress field at the beginning of the cooling step

For the deformation field in the detachment step (see Figure 3. 23), we can see that some of the bottom nodes of the spring are still in contact with the build plate (dark blue, with zero displacement), and others have different displacements due to the shrinkage effect. The corners of the model have the highest displacement values in the stack direction. The displacements in Figure 3. 23 were scaled by a factor of 5 to see the deformation pattern clearly.

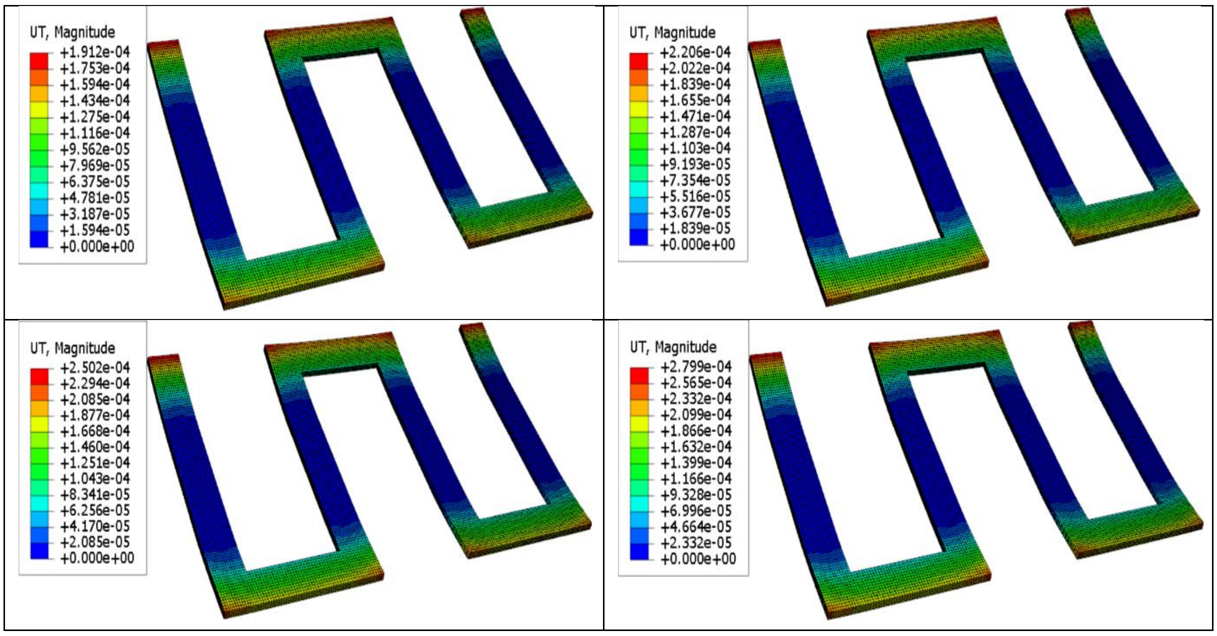


Figure 3. 23 Deformation field at the end of the detachment step with a scale deformation factor of 5

3.2.4 Part distortion evaluation

Dimensions of the flat spring in the different directions:

- **X direction**: 65 mm
- **Y direction**: 1 mm (stack direction in this case)
- **Z direction**: 40 mm
- **Thickness of the flat spring pattern**: 5 mm

Maximum displacements in the different directions:

- **X direction** (see Figure 3. 24): -0.0314 mm.

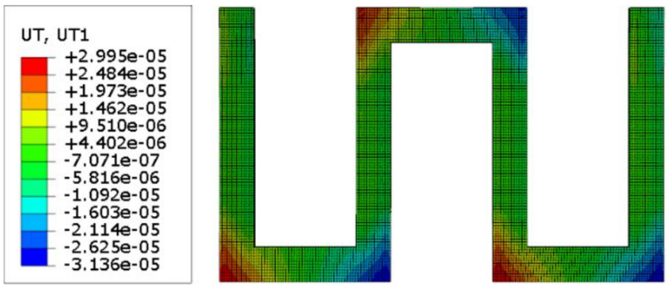


Figure 3. 24 Displacement field in the X direction

- **Y direction** (see Figure 3. 25): 0.278 mm.

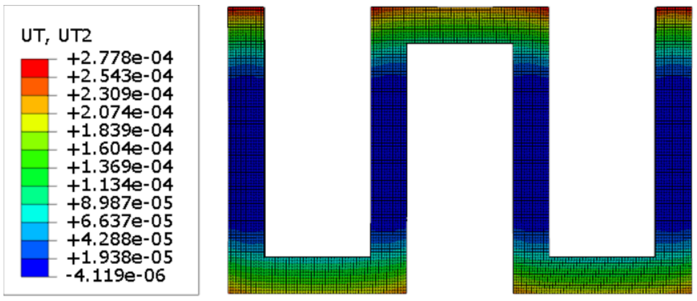


Figure 3. 25 Displacement field in the Y direction

- **Z direction** (see Figure 3. 26): 0.0511 mm.

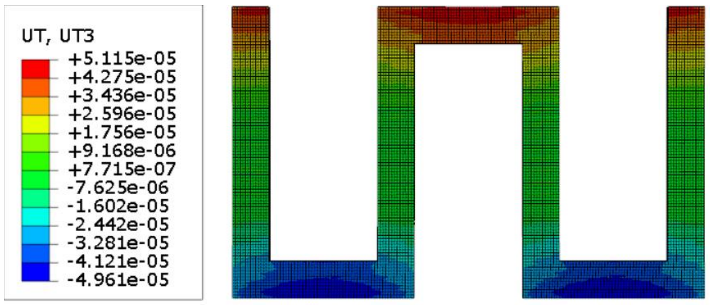


Figure 3. 26 Displacement field in the Z direction

Distortion percentages in different directions (X and Z directions compared to the thickness of the flat spring pattern):

- **X direction** = $\frac{|-0.0314|}{5} \cdot 100 = 0.63\ \%$
- **Y direction** = $\frac{0.278}{1} \cdot 100 = \ 27.8\%$
- **Z direction** = $\frac{0.0511}{5} \cdot 100 = \ 1.02\%$

From the results we can extract that deformation in the stack direction is much higher than in the rest of the dimensions due to warpage. There is a high contact area between the piece and the build plate, and the dimension in the stack direction is only 1 mm, which also contributes to the high displacement value in this direction.

3.2.5 Validation of the results

(Cattenone et al., 2019) was used as a benchmark for the flat spring simulation. In this work, the simulation is performed with PLA material instead ABS and with different process parameters. But the field distribution should be comparable, so we use (Cattenone et al., 2019) to validate the results. If we validate the results, this means that the numerical simulation procedure is correct, so we can perform analysis in the same way to different geometries, such as the thin wall explained in section 2.1.2.

We are only going to compare the structural results. This is because the structural analysis depends on the thermal results. So if the thermal results are not in agreement with (Cattenone et al., 2019), the structural ones will not be either.

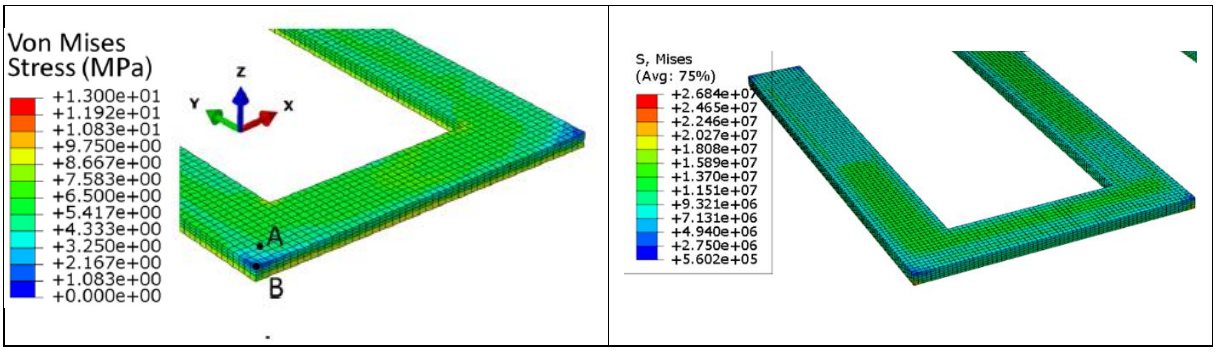


Figure 3. 27 On the left, stress results extracted from (Cattenone et al., 2019), on the right, stress results from this present work (end of the printing step)

In Figure 3. 27, we can see the stress distribution comparison between both works at the end of the printing process. The stress distribution is almost the same, with the smaller stresses in the corners of the model and almost a uniform distribution in the rest of the part. The values in the present work are higher, this is because we are using PLA, which has higher values for Young’s modulus and yield stress than in the case of ABS.

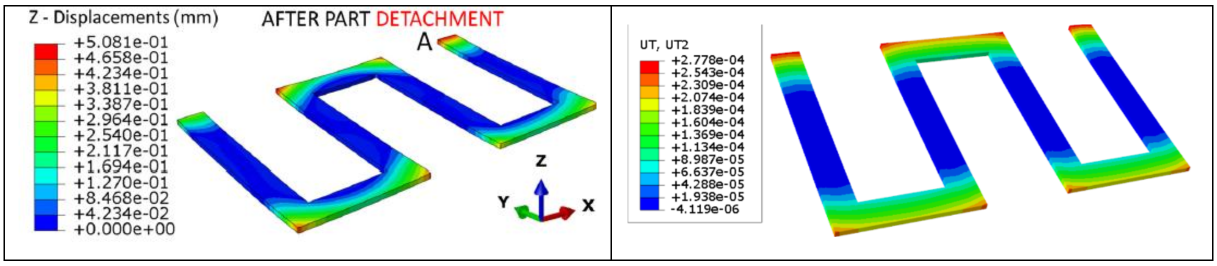


Figure 3. 28 On the left, deformation results for the stack direction extracted from (Cattenone et al., 2019), on the right, deformation results for the stack direction extracted from the present work (end of the detachment step)

In Figure 3. 28, we can see the deformation results for the stack direction in each work. There are big similarities between the two deformation fields. The values in the present work are smaller than in (Cattenone et al., 2019), because the thermal expansion coefficient in the ABS was set to $9\, x\, 10^{-5}\ ºC^{-1}$ and in the PLA was set to $6.8\, x\, 10^{-5}\ ºC^{-1}$.

3.3 Third geometry (thin wall)

As with the flat spring, the element activation method is performed by the toolpath-mesh intersection module integrated into the AM modeler plugin in Abaqus. In this case, we are going to study four different infill patterns, so four different event series were generated representing the corresponding nozzle movement. These event series were generated with excel because of the simplicity of the part. We are also going to perform a parametric study in sections later to see the impact of some parameters on the part distortion.

- In this case, only one contour bead is printed (see Figure 3. 29). The layer thickness, in this case, is also set to 0.2 mm. The wall height is 30 mm, so there are 150 layers in the printing process. This entails a high computational cost for the analysis. The printing time for the thin wall strongly varies depending on the infill pattern, being the minimum printing time of 431.83 seconds for the first infill pattern. The dimensions of the wall are 30 mm in height; 40 mm long and 4 mm wide.

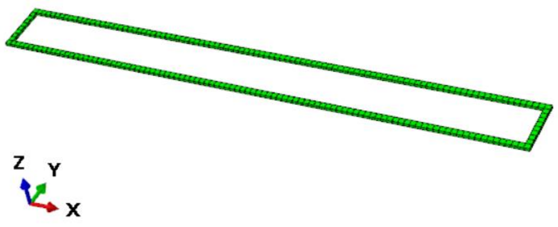


Figure 3. 29 Contour bead for the thin wall geometry

- For the meshing strategy, in this case, we set the element heigh as the filament height (0.2 mm) and the element width and length as the filament width (0.4 mm) (see Figure 3. 30).

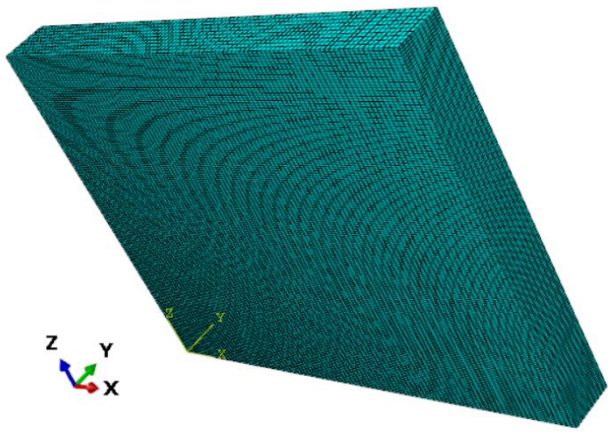


Figure 3. 30 Meshing strategy for the thin wall

3.3.1 Infill patterns

In this section, the four different infill patterns are described in detail. As it was said, the infill pattern influences the printing and computational times.

Infill pattern nº1:

The first infill pattern is shown in Figure 3. 31. It is a unidirectional pattern printed in the X direction. It has 0.8 mm spacing between beads. The printing time for the whole part with this pattern is 431.83 seconds. For this pattern we can calculate the part density as follows:

- **Total area of the base** $= 40 \cdot 4 = 160\, mm^2$
- **Contour area** $= 2 \cdot 39.2 \cdot 0.4 + 2 \cdot 4 \cdot 0.4 = 34.56\, mm^2$
- **Infill area** $= 2 \cdot 39.2 \cdot 0.4 = 31.36\, mm^2$
- **Part density** $= \frac{34.56+31.36}{160} \cdot 100 = 41.2\%$

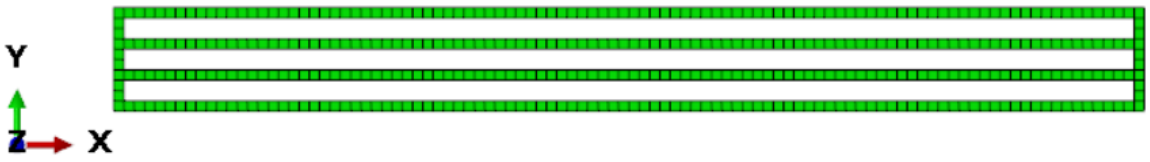


Figure 3. 31 First infill pattern

Infill pattern nº2:

The second infill pattern is shown in Figure 3. 32. It is also a unidirectional pattern printed in the X direction. It has a 0.4 mm spacing between beads. The printing time for the whole part, in this case, is 661.79 seconds. The part density in this case:

- **Total area of the base** = $40 \cdot 4 = 160\ mm^2$
- **Contour area** = $34.56\ mm^2$
- **Infill area** = $4 \cdot 39.2 \cdot 0.4 = 62.72\ mm^2$
- **Part density** = $\frac{34.56+62.72}{160} \cdot 100 = 60.8\%$

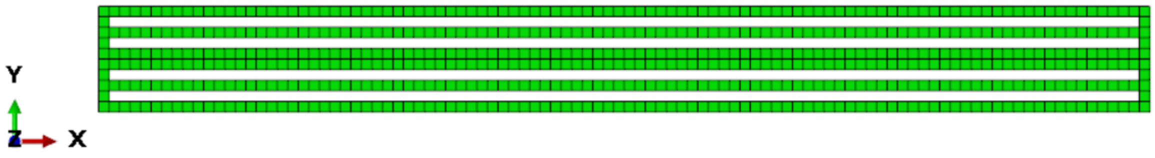


Figure 3. 32 Second infill pattern

Infill pattern nº3:

The third infill pattern is shown in Figure 3. 33. It is a unidirectional pattern but in this case printed in the Y direction. The spacing between beads is 6 mm. The printing time is 435.689 seconds and the part density:

- **Total area of the base** = $160\ mm^2$
- **Contour area** = $34.56\ mm^2$
- **Infill area** = $3 \cdot 0.8 \cdot 3.2 + 2 \cdot 0.4 \cdot 3.2 = 10.24\ mm^2$
- **Part density** = $\frac{34.56+10.24}{160} \cdot 100 = 28\%$

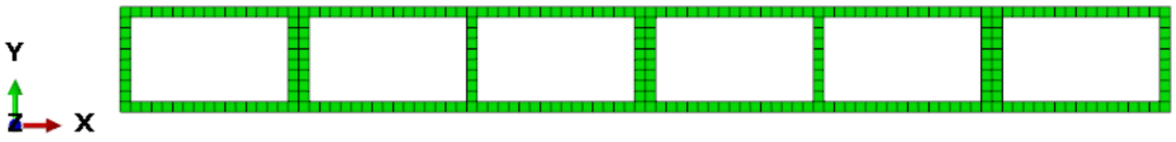


Figure 3. 33 Third infill pattern

Infill pattern nº4:

The fourth and final infill pattern is shown in Figure 3. 34. In this case, is a bidirectional pattern printed in the X and Y directions with a raster angle of 90º. The printing time is 642.66 seconds and the part density:

- **Total area of the base** = $160\ mm^2$
- **Contour area** = $34.56\ mm^2$

- **Infill area** $= 2 \cdot 0.4 \cdot 3.2 + 3 \cdot 0.8 \cdot 3.2 + 2 \cdot 39.2 \cdot 0.4 = 41.6\ mm^2$
- **Part density** $= \frac{31.36+34.56}{160} \cdot 100 = 47.6\%$

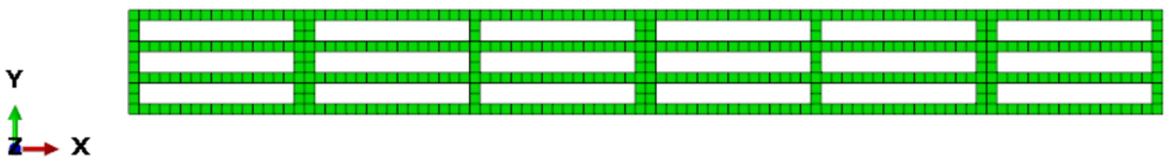


Figure 3. 34 Fourth infill pattern

3.3.2 Thermal analysis results

Before starting with the results, some aspects of the simulation have to be clarified. As it was said in sections before, the simulation requires high computational time. So, in order to simulate in a comprehensive time-lapse, the printing step was simulated with a time step of 4 seconds both for the thermal and structural simulations. This will cause the activation temperature will not be the same as the extrusion temperature as it is explained in section 3.2.1.

In Figure 3. 35, the thermal distributions of the different infill patterns are shown for the printing time of 220 seconds. The extrusion temperature was set to 205 ºC and the bed temperature to 60 ºC.

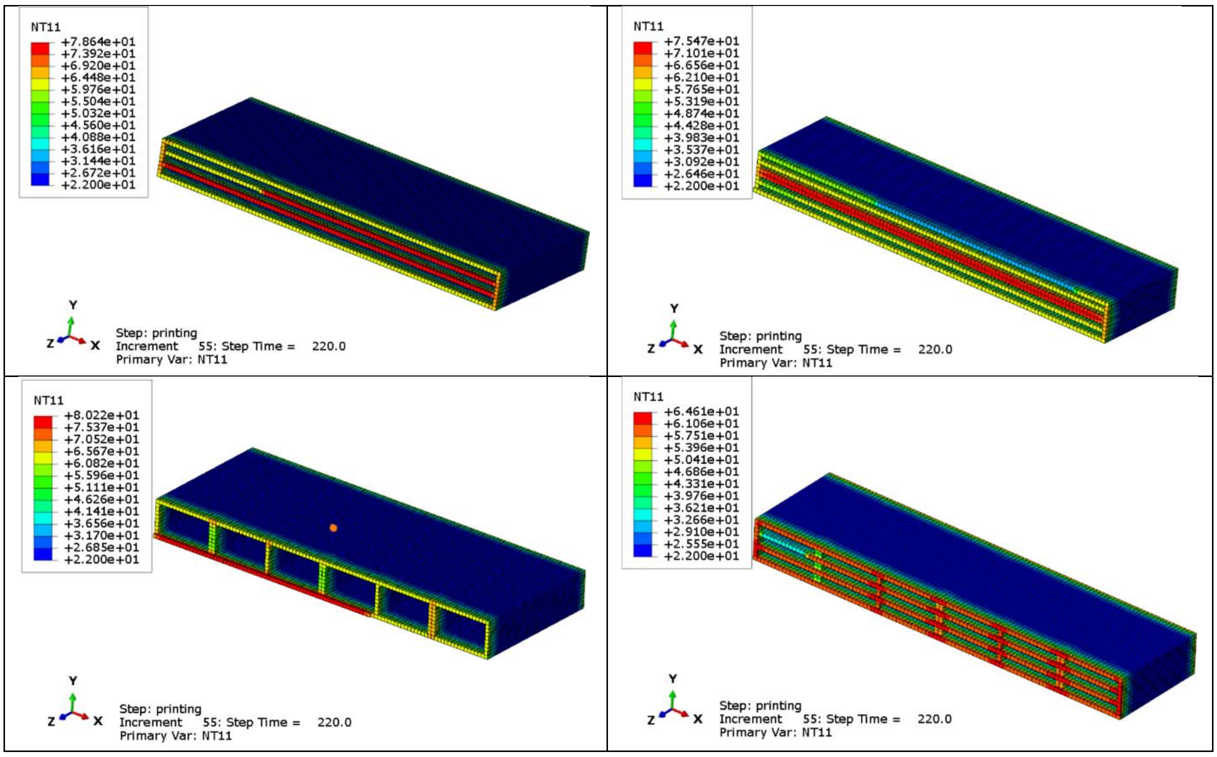


Figure 3. 35 Thermal distributions of the four different infill patterns at the same printing time

We can appreciate how the bottom layers are at the bed temperature, then there is a transition to the ambient temperature (dark blue) and the newly extruded material is represented by the red color. We can also notice that infill patterns 1 and 3 have more extruded layers at 220 seconds of printing. This is because the part density in these infill patterns is less than in the second and fourth patterns.

In Figure 3. 36, we pick the same node in the outer surface of the wall for all the different infill patterns and compare the thermal evolution with time. In Figure 3. 37, we pick an internal node, and we do the same. The picked nodes are in the middle of the X dimension of the wall and in the top layer.

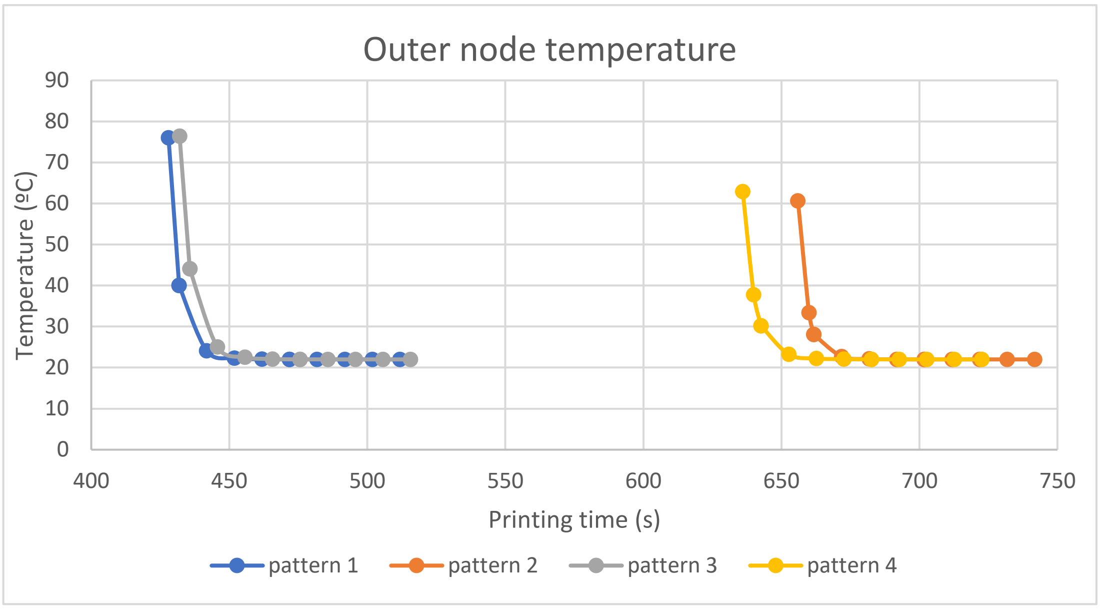


Figure 3. 36 Outer node temperature distribution in the top layer

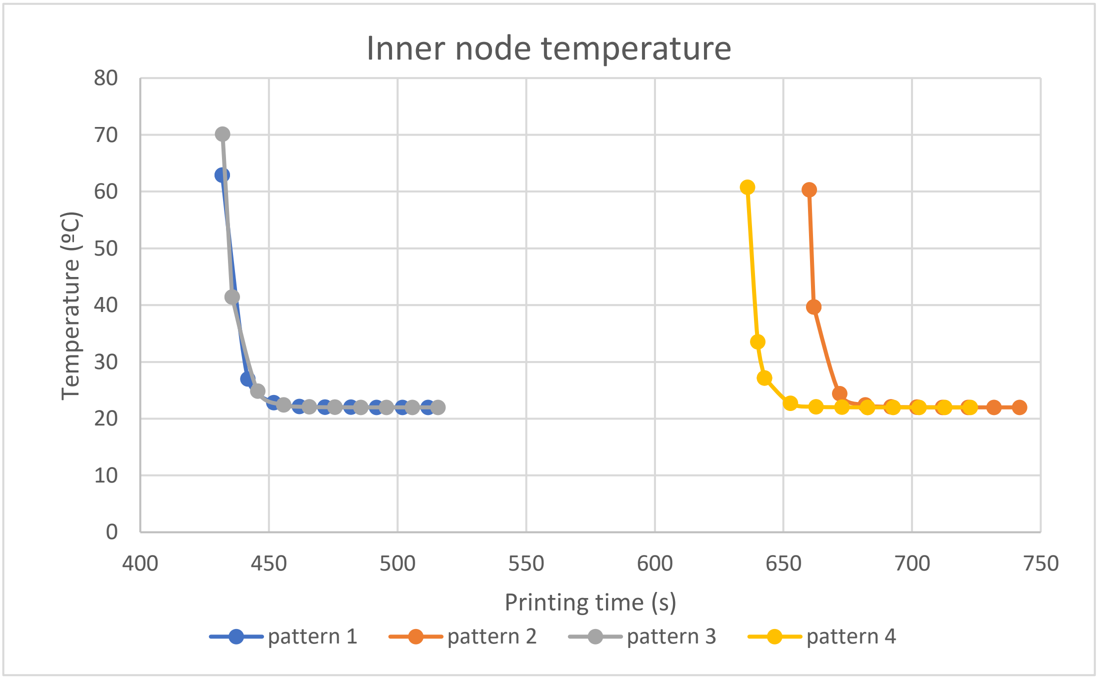


Figure 3. 37 Inner node temperature distribution in the top layer

From Figure 3. 37 and Figure 3. 36, we can notice that the activation temperature for infill patterns 1 and 3 is higher than for patterns 2 and 4, which are the patterns with less part density. Temperatures tend to ambient temperature as time goes on. A rough calculation of the cooling rate at the beginning of the nodes cooling was performed taking into account the two first points of each infill pattern (time, temperature) and calculating the slope.

For the outer node:

- Infill pattern 1: -9.4 ºC/s
- Infill pattern 2: -6.82 ºC/s
- Infill pattern 3: -8.76 ºC/s
- Infill pattern 4: -6.28 ºC/s

For the inner node:

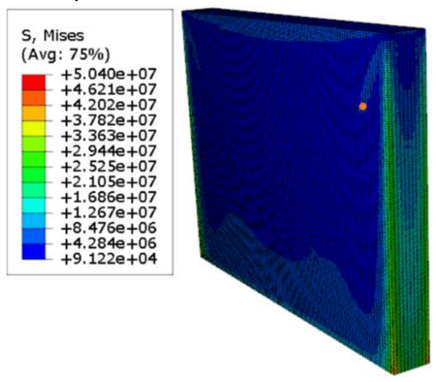
- Infill pattern 1: -3.6 ºC/s
- Infill pattern 2: -11.55 ºC/s
- Infill pattern 3: -7.77 ºC/s
- Infill pattern 4: -6.8 ºC/s

The cooling rate is related to the polymer crystallization temperature (Poel et al., 2011). The higher the cooling rate, the lower the crystallization temperature of the polymer.

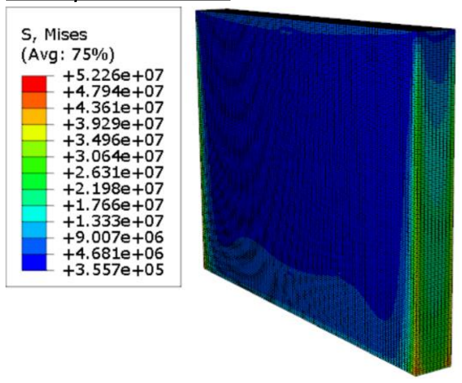
3.3.3 Structural analysis results

In Figure 3. 38, the stress field for the different infill patterns is shown. The extrusion temperature was set to 205 ºC and the bed temperature was set to 60 ºC.

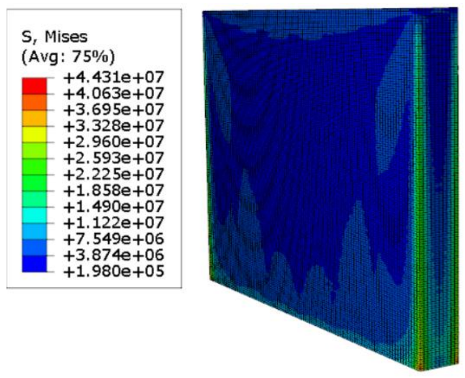
Infill pattern nº1



Infill pattern nº2



Infill pattern nº3



Infill pattern nº4

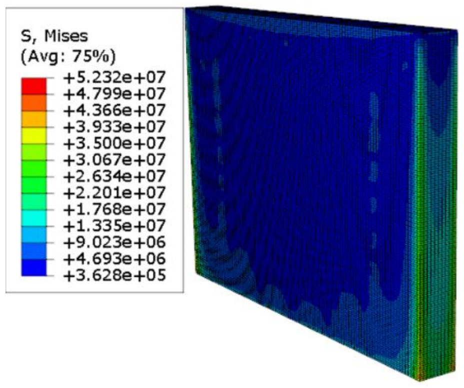


Figure 3. 38 Stress field at the end of the cooling process

From Figure 3. 38, we can see that the higher stresses take place in the bottom corners and along the edges in the stack direction. The higher stresses take place in infill patterns nº2 and nº4 (very close to each other), followed by pattern nº1 and finally pattern nº3 with the lowest stress values.

In Figure 3. 39, Figure 3. 40, and Figure 3. 41, we can see the nodal displacement results in the three different directions for the different infill patterns. The results are scaled by a factor of 10 to clarify the deformation behavior of the part.

Displacements in the X direction

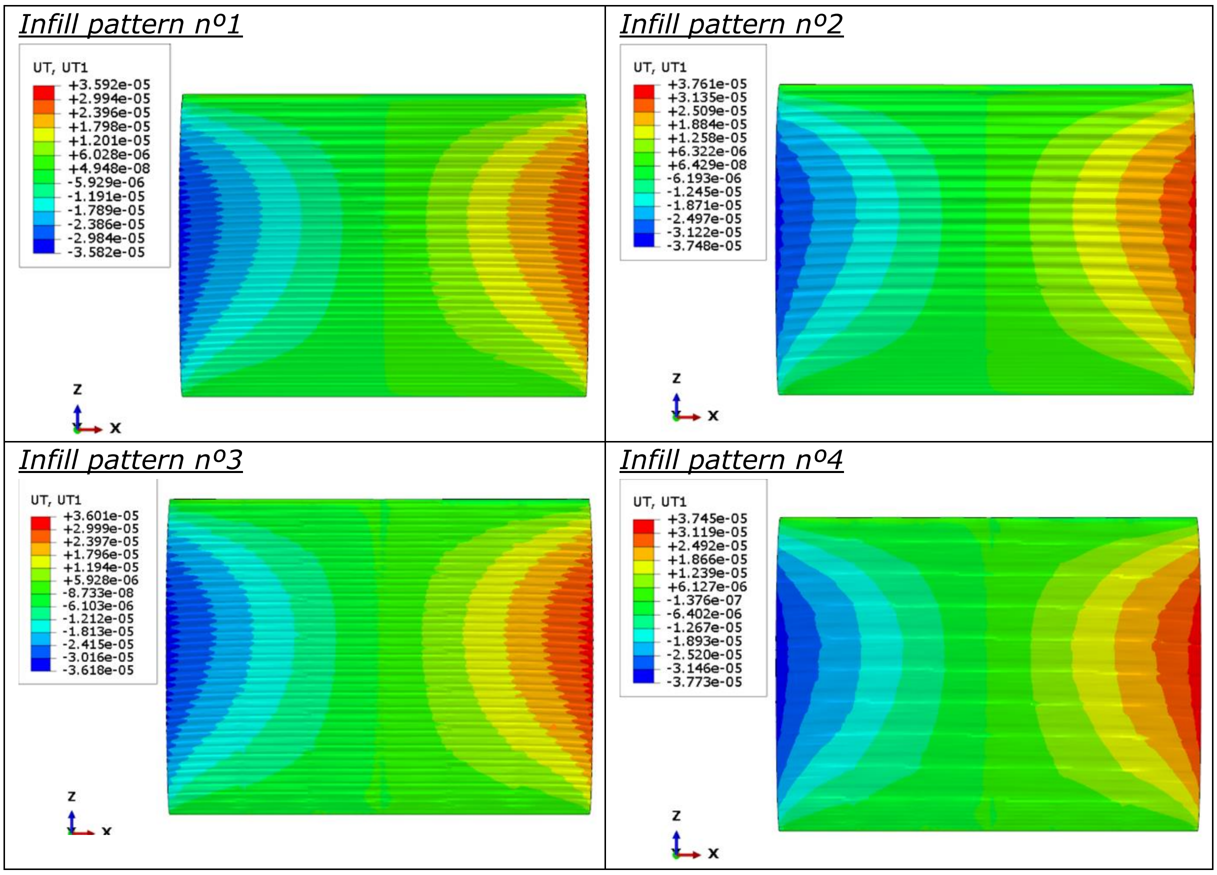
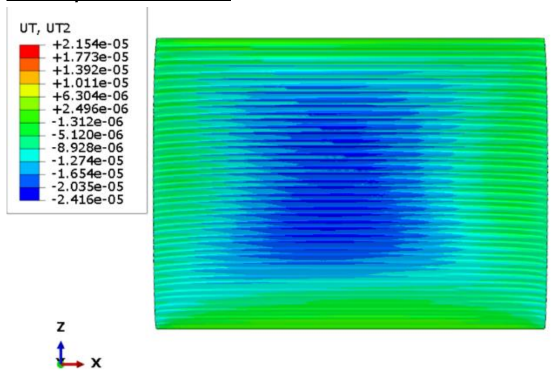


Figure 3. 39 Displacement field in the X direction for different infill patterns

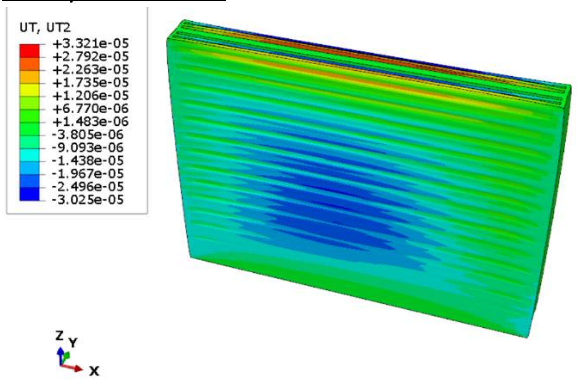
For the X displacements, we can notice symmetry in the deformation pattern with respect to the plane x=20 mm in all four infill patterns. The maximum displacement in the X direction takes place for the infill pattern nº 4, with a value of -0.0377 mm, followed by infill pattern nº 2 with 0.0376 mm, infill pattern nº 3 with -0.0362 and finally, the infill pattern nº 1 with 0.0358 mm.

Displacements in the Y direction

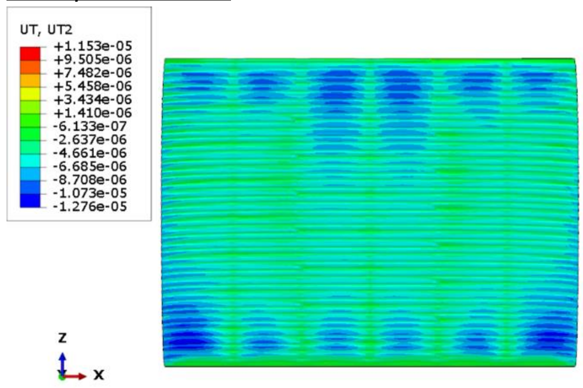
Infill pattern nº1



Infill pattern nº2



Infill pattern nº3



Infill pattern nº4

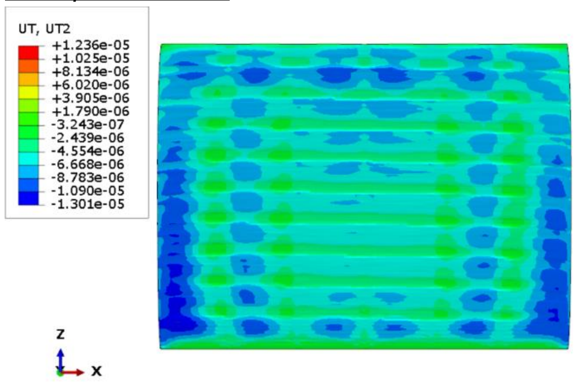
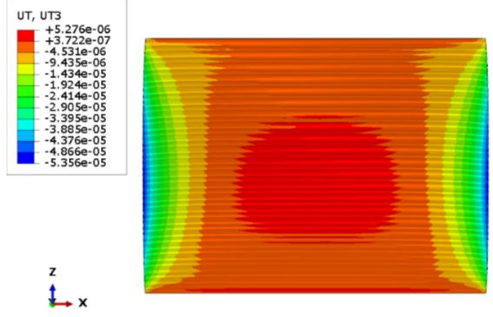


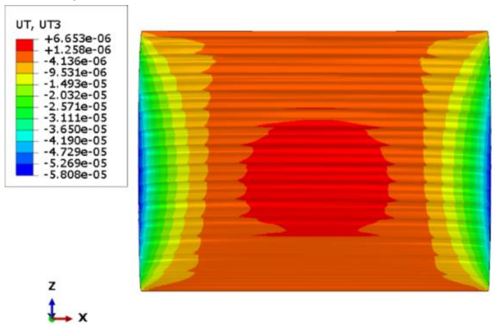
Figure 3. 40 Displacement field in the Y direction for different infill patterns

Displacements in the Z direction

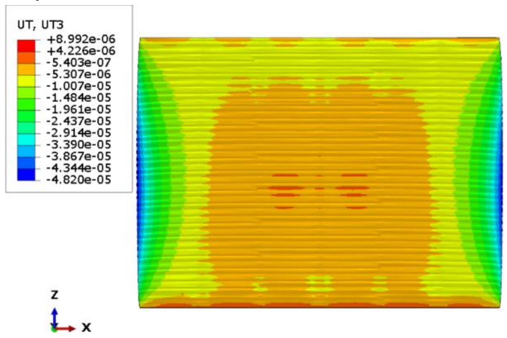
Infill pattern nº1



Infill pattern nº2



Infill pattern nº3



Infill pattern nº4

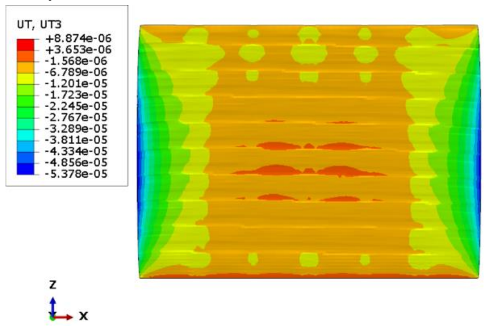


Figure 3. 41 Displacement field in the Z direction for different infill patterns

For the Y displacements (see Figure 3. 40), we can assume symmetry in the deformation pattern with respect to the plane Y=2mm except for the infill pattern nº 2. Anyway, Y displacement values are not as symmetric as the X displacement values.

The maximum displacement for the Y direction takes place for the infill pattern nº 2, with a value of 0.0332 mm, followed by infill pattern nº 1 with -0.0242 mm, infill pattern nº 4, with -0.013 mm and finally infill pattern nº 3 with -0.0128 mm. Infill patterns nº4 and nº3 are the only ones with infill patterns in the Y direction, so this is the reason why the deformation in the Y direction is smaller.

For the Z displacements, there is symmetry with respect to the plane Y=20 mm. The maximum displacement for the Z direction takes place for the infill pattern nº 2, with a value of -0.0058 mm, followed by infill pattern nº 4 with -0.00538 mm, infill pattern nº 1 with -0.00536 mm and finally, infill pattern nº 3 with -0.00482 mm.

3.3.4 Part distortion evaluation

Dimensions in the different directions of the thin wall:

- **X direction**: 40 mm
- **Y direction**: 4 mm
- **Z direction**: 30 mm

-Infill pattern nº1

Maximum displacements in the different directions:

- **X direction**: -0.0358 mm
- **Y direction**: -0.0242 mm
- **Z direction**: -0.0536 mm

Distortion percentages in different directions:

- **X direction** = $\frac{|-0.0358|}{40} \cdot 100 = 0.0895\%$
- **Y direction** = $\frac{|-0.0242|}{4} \cdot 100 = 0.605\%$
- **Z direction** = $\frac{|-0.0536|}{30} \cdot 100 = 0.179\%$

-Infill pattern nº2

Maximum displacements in the different directions:

- **X direction**: 0.0376 mm
- **Y direction**: 0.0332 mm
- **Z direction**: -0.058 mm

Distortion percentages in the different directions:

- **X direction** = 0.094%
- **Y direction** = 0.83%

- **Z direction** = 0.19%

-Infill pattern nº3

Maximum displacements in the different directions:

- **X direction**: -0.0362 mm
- **Y direction**: -0.0128 mm
- **Z direction**: -0.0482 mm

Distortion percentages in the different directions:

- **X direction** = 0.09%
- **Y direction** = 0.32%
- **Z direction** = 0.16%

-Infill pattern nº4

Maximum displacements in the different directions:

- **X direction**: -0.0377 mm
- **Y direction**: -0.013 mm
- **Z direction**: -0.054 mm

Distortion percentage in the different directions:

- **X direction** = 0.094%
- **Y direction** = 0.325%
- **Z direction** = 0.18%

3.3.5 Parametric study of the influence of process parameters in the part distortion

In this section, we are performing a parametric study to see the influence of three different process parameters on the part distortion (for the thin wall geometry). The three parameters that we are analyzing here are infill pattern, extrusion temperature, and bed temperature. These are called factors, and we are performing the analysis with different values of these factors, called levels. The factors and their levels are the following:

- Infill pattern: pattern 1, pattern 2, pattern 3, and pattern 4 (the same infill patterns explained in section 3.3.1).
- Extrusion temperature: 195 ºC, 205 ºC and 210 ºC.
- Bed temperature: 55 ºC, 60 ºC and 65 ºC

To perform the experiments with all the combinations of levels, we should perform $4 \cdot 3 \cdot 3 = 36$ different simulations. All combinations of factors are shown in Table 3. 1, Table 3. 2, and Table 3. 3.

Pattern	**Extrusion temperature**		
1	195 ºC	195 ºC	195 ºC
2	195 ºC	195 ºC	195 ºC
3	195 ºC	195 ºC	195 ºC
4	195 ºC	195 ºC	195 ºC

Bed temperature	55 ºC	60 ºC	65 ºC

Table 3. 1 Combination of levels of the three different factors (extrusion temperature = 195 ºC)

Pattern	**Extrusion temperature**		
1	205 ºC	205 ºC	205 ºC
2	205 ºC	205 ºC	205 ºC
3	205 ºC	205 ºC	205 ºC
4	205 ºC	205 ºC	205 ºC
Bed temperature	55 ºC	60 ºC	65 ºC

Table 3. 2 Combinations of levels of the three different factors (extrusion temperature = 205 ºC)

Pattern	**Extrusion temperature**		
1	210 ºC	210 ºC	210 ºC
2	210 ºC	210 ºC	210 ºC
3	210 ºC	210 ºC	210 ºC
4	210 ºC	210 ºC	210 ºC
Bed temperature	55 ºC	60 ºC	65 ºC

Table 3. 3 Combinations of levels of the three different factors (extrusion temperature = 210 ºC)

The objective is to calculate the influence of each parameter in the maximum displacements of the different directions (X, Y, and Z) using the ANOVA table tool. For doing this, we will need to calculate three different ANOVA tables, one for each direction. The results of the experiments are shown in Table 3. 4. The deformation values are given in absolute value and in milimeters.

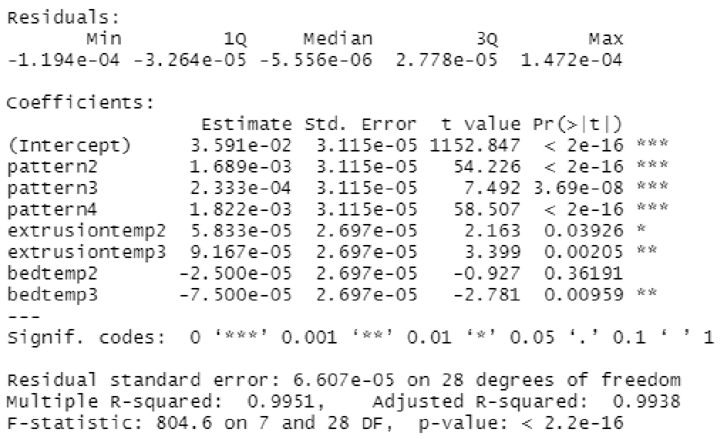
nº	**Pattern**	**Extrusion temperature**	**Bed temperature**	**X maximum displacement**	**Y maximum displacement**	**Z maximum displacement**
1	1	195 ºC	55 ºC	0.0359	0.0238	0.0529
2	1	205 ºC	55 ºC	0.0360	0.0242	0.0533
3	1	210 ºC	55 ºC	0.0360	0.0246	0.0538
4	1	195 ºC	60 ºC	0.0359	0.0238	0.0531
5	1	205 ºC	60 ºC	0.0359	0.0242	0.0536
6	1	210 ºC	60 ºC	0.0360	0.0243	0.0538
7	1	195 ºC	65 ºC	0.0358	0.0238	0.0534
8	1	205 ºC	65 ºC	0.0359	0.0242	0.0538
9	1	210 ºC	65 ºC	0.0359	0.0244	0.0540
10	2	195 ºC	55 ºC	0.0375	0.0330	0.0573
11	2	205 ºC	55 ºC	0.0377	0.0328	0.0578
12	2	210 ºC	55 ºC	0.0377	0.0328	0.0580
13	2	195 ºC	60 ºC	0.0375	0.0334	0.0576
14	2	205 ºC	60 ºC	0.0376	0.0332	0.0581
15	2	210 ºC	60 ºC	0.0378	0.0332	0.0583
16	2	195 ºC	65 ºC	0.0375	0.0338	0.0580
17	2	205 ºC	65 ºC	0.0376	0.0337	0.0584
18	2	210 ºC	65 ºC	0.0376	0.0337	0.0586
19	3	195 ºC	55 ºC	0.0361	0.0125	0.0475
20	3	205 ºC	55 ºC	0.0362	0.0127	0.0479

21	3	210 ºC	55 ºC	0.0363	0.0127	0.0481
22	3	195 ºC	60 ºC	0.0361	0.0126	0.0478
23	3	205 ºC	60 ºC	0.0362	0.0128	0.0482
24	3	210 ºC	60 ºC	0.0362	0.0128	0.0484
25	3	195 ºC	65 ºC	0.0360	0.0128	0.0481
26	3	205 ºC	65 ºC	0.0361	0.0129	0.0485
27	3	210 ºC	65 ºC	0.0362	0.0130	0.0487
28	4	195 ºC	55 ºC	0.0378	0.0128	0.0531
29	4	205 ºC	55 ºC	0.0378	0.0130	0.0535
30	4	210 ºC	55 ºC	0.0377	0.0130	0.0537
31	4	195 ºC	60 ºC	0.0378	0.0129	0.0534
32	4	205 ºC	60 ºC	0.0377	0.0130	0.0538
33	4	210 ºC	60 ºC	0.0377	0.0131	0.0540
34	4	195 ºC	65 ºC	0.0378	0.0130	0.0537
35	4	205 ºC	65 ºC	0.0377	0.0131	0.0541
36	4	210 ºC	65 ºC	0.0377	0.0132	0.0543

Table 3. 4 Results for all the 36 combinations of levels

In this study, the additive model is going to be used for ANOVA. The additive model assumes that the effects on the outcome of a particular level change for one explanatory variable do not depend on the level of another explanatory variable. If we used the interaction model, overfitting would happen.

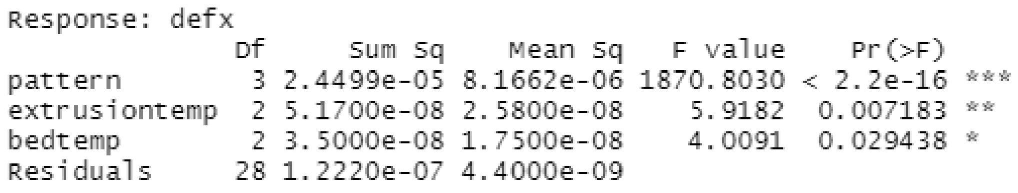
ANOVA table for the X displacement and coefficients of the linear additive model



```
Residuals:
       Min         1Q     Median         3Q        Max 
-1.194e-04 -3.264e-05 -5.556e-06  2.778e-05  1.472e-04 

Coefficients:
                 Estimate Std. Error  t value Pr(>|t|)    
(Intercept)     3.591e-02  3.115e-05 1152.847  < 2e-16 ***
pattern2        1.689e-03  3.115e-05   54.226  < 2e-16 ***
pattern3        2.333e-04  3.115e-05    7.492 3.69e-08 ***
pattern4        1.822e-03  3.115e-05   58.507  < 2e-16 ***
extrusiontemp2  5.833e-05  2.697e-05    2.163  0.03926 *  
extrusiontemp3  9.167e-05  2.697e-05    3.399  0.00205 ** 
bedtemp2       -2.500e-05  2.697e-05   -0.927  0.36191    
bedtemp3       -7.500e-05  2.697e-05   -2.781  0.00959 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.607e-05 on 28 degrees of freedom
Multiple R-squared:  0.9951,	Adjusted R-squared:  0.9938 
F-statistic: 804.6 on 7 and 28 DF,  p-value: < 2.2e-16
```



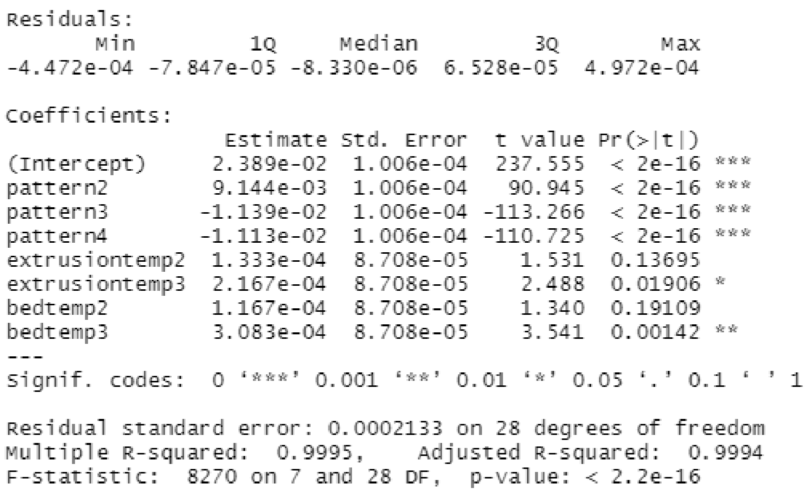
```
Response: defx
              Df     Sum Sq    Mean Sq   F value    Pr(>F)    
pattern        3 2.4499e-05 8.1662e-06 1870.8030 < 2.2e-16 ***
extrusiontemp  2 5.1700e-08 2.5800e-08    5.9182  0.007183 ** 
bedtemp        2 3.5000e-08 1.7500e-08    4.0091  0.029438 *  
Residuals     28 1.2220e-07 4.4000e-09                        
```

Table 3. 5 ANOVA table for the X direction deformation

To accept the hypothesis that one factor is statistically relevant, we are assuming a level of significance higher than 95%, or what is the same, a p-value less than 0.05.

In the case of Table 3. 5, the results show that all the factors are statistically relevant for the X direction displacement. The infill pattern is the most influential factor (lowest p-value), followed by the extrusion temperature and finally the bed temperature (highest p-value).

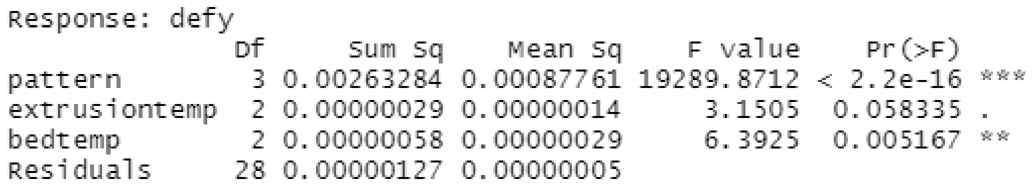
ANOVA table for the Y displacement and coefficients of the linear additive model



```
Residuals:
       Min         1Q     Median         3Q        Max 
-4.472e-04 -7.847e-05 -8.330e-06  6.528e-05  4.972e-04 

Coefficients:
                 Estimate Std. Error  t value Pr(>|t|)    
(Intercept)     2.389e-02  1.006e-04  237.555  < 2e-16 ***
pattern2        9.144e-03  1.006e-04   90.945  < 2e-16 ***
pattern3       -1.139e-02  1.006e-04 -113.266  < 2e-16 ***
pattern4       -1.113e-02  1.006e-04 -110.725  < 2e-16 ***
extrusiontemp2  1.333e-04  8.708e-05    1.531  0.13695    
extrusiontemp3  2.167e-04  8.708e-05    2.488  0.01906 *  
bedtemp2        1.167e-04  8.708e-05    1.340  0.19109    
bedtemp3        3.083e-04  8.708e-05    3.541  0.00142 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0002133 on 28 degrees of freedom
Multiple R-squared:  0.9995,	Adjusted R-squared:  0.9994 
F-statistic:  8270 on 7 and 28 DF,  p-value: < 2.2e-16
```

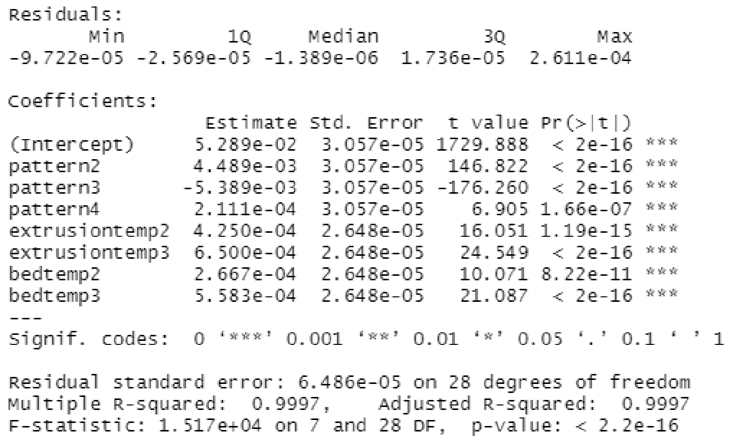


```
Response: defy
              Df     Sum Sq    Mean Sq    F value    Pr(>F)    
pattern        3 0.00263284 0.00087761 19289.8712 < 2.2e-16 ***
extrusiontemp  2 0.00000029 0.00000014     3.1505  0.058335 .  
bedtemp        2 0.00000058 0.00000029     6.3925  0.005167 ** 
Residuals     28 0.00000127 0.00000005                         
```

Table 3. 6 ANOVA table for the Y direction deformation

As shown in Table 3. 6, there are only two statistically relevant factors for the Y direction displacement: the infill pattern and the bed temperature. Both have lower p-values than 0.05 but the infill pattern has more statistical significance. On the contrary, the extrusion temperature, in this case, is not statistically relevant because its p-value is higher than 0.05.

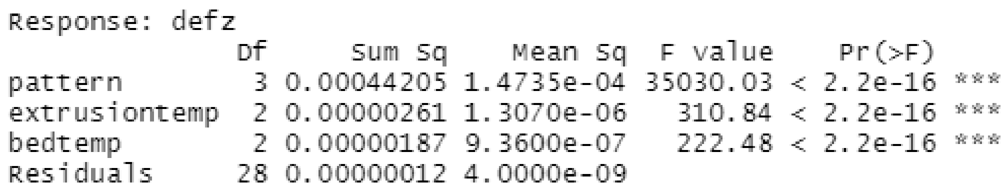
ANOVA table for the Z displacement



```
Residuals:
       Min         1Q     Median         3Q        Max 
-9.722e-05 -2.569e-05 -1.389e-06  1.736e-05  2.611e-04 

Coefficients:
                 Estimate Std. Error  t value Pr(>|t|)    
(Intercept)     5.289e-02  3.057e-05 1729.888  < 2e-16 ***
pattern2        4.489e-03  3.057e-05  146.822  < 2e-16 ***
pattern3       -5.389e-03  3.057e-05 -176.260  < 2e-16 ***
pattern4        2.111e-04  3.057e-05    6.905 1.66e-07 ***
extrusiontemp2  4.250e-04  2.648e-05   16.051 1.19e-15 ***
extrusiontemp3  6.500e-04  2.648e-05   24.549  < 2e-16 ***
bedtemp2        2.667e-04  2.648e-05   10.071 8.22e-11 ***
bedtemp3        5.583e-04  2.648e-05   21.087  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.486e-05 on 28 degrees of freedom
Multiple R-squared:  0.9997,	Adjusted R-squared:  0.9997 
F-statistic: 1.517e+04 on 7 and 28 DF,  p-value: < 2.2e-16
```



```
Response: defz
              Df     Sum Sq    Mean Sq  F value    Pr(>F)    
pattern        3 0.00044205 1.4735e-04 35030.03 < 2.2e-16 ***
extrusiontemp  2 0.00000261 1.3070e-06   310.84 < 2.2e-16 ***
bedtemp        2 0.00000187 9.3600e-07   222.48 < 2.2e-16 ***
Residuals     28 0.00000012 4.0000e-09                       
```

Table 3. 7 ANOVA table for the Z direction deformation

In the case of Table 3. 7, the three factors have high statistical significance. First, the highest F value corresponds to the infill pattern (more statistical significance), followed by the extrusion temperature and, finally the bed temperature.

4 Machine Learning applications in FDM

Until now, we have only used numerical simulation for predicting the deformations due to the FDM process. As we saw in other sections of this present work, the numerical simulation of this type of process involves a high computational and time cost.

In this situation, we need to use other methods that allow us to achieve results in a reasonable time and computational cost. This could be an opportunity to use a machine learning model.

In this section, the basic machine learning algorithms are described with some examples, and then a more detailed analysis is made of the neural networks applied to FDM.

4.1 Machine learning algorithms

An algorithm can be described as a set of logical instructions following a specific flow in order to solve one specific problem.

Machine learning (ML) is a branch of artificial intelligence (AI) and computer science. There are a lot of possible definitions for machine learning, for example: ‶A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E″.

The most common machine learning algorithms can be divided into three main groups: supervised learning, unsupervised learning, and reinforcement learning. In the upcoming sections, these three groups are described (Brownlee, 2016).

4.1.1 Supervised learning

These are algorithms that use a known dataset (called training dataset) to make predictions. The training dataset includes input data and response values. From it, the supervised learning algorithm seeks to build a model that can make predictions of the response values for a new dataset.

A common example used to explain the supervised learning algorithms is the similarity with a classroom scene. The teacher explains difficult content to the students that can not learn on their own by giving them the answers. This is how we teach the machine with supervised learning algorithms.

Supervised machine learning includes two major processes: classification and regression.

- **Classification**: in this type of process, a set of data is categorized into classes. It can be a binary classification problem (e.g. the prediction of whether it will rain or not, with two outputs ‶yes″ or ‶not″) or a multi-class classification problem (more possible outputs). The most common classification problems nowadays are speech recognition, face detection, handwriting recognition, document falsification, etc.
- **Regression**: it is a model prediction method in which the output is a continuous numerical value. It establishes a relationship between a single dependent variable dependent on several independent ones. For example, calculate a house price based on different parameters (e.g. house size, number of rooms, etc.).

Now we are going to talk about the most used algorithms for supervised learning. Some of these algorithms can be used in regression, classification, or even both.

1. *Linear regression*

It is only used in regression problems and is one of the most used because of its simplicity in implementation.

Linear regression is a linear model, i.e., a model that assumes that the output variable (y) can be calculated as a linear combination of the input variables (x). When we only have a single input variable, it is called simple linear regression (see Figure 4. 1). Otherwise, it is called multiple linear regression.

A linear regression model assigns coefficients to each input value. The representation of a simple regression problem would be:

$$y = \alpha_0 + \alpha_1 \cdot x$$

which is the equation of a straight line, being α_0 the so-called intercept or the bias coefficient and α_1 the weight of the input variable x.

Training a linear regression model means estimating the values of the coefficients that correlate the input values with the output value. There are different techniques to estimate these coefficients. Here we are going to talk about the most used three techniques.

- **Ordinary least squares**: seeks to minimize the sum of the squared residuals. Given a regression line through the data, we calculate the distance from each point of the training data to the regression line, square it, and sum all of the squared errors together. This is the quantity that the least square method seeks to minimize.
- **Gradient descent**: starts assigning random values to each coefficient. For each pair of input and output values, the sum of the squared errors is calculated. A learning rate is used as a scale factor and the coefficients are updated in the direction of minimizing the error. It is an iterative process repeated until a minimum sum squared error is achieved or no further improvement is possible. The learning rate defines the size of the improvement step to take in each iteration.
- **Regularization**: these methods seek to minimize the squared error of the model and also reduce the complexity of the model. These methods are effective to use when there is collinearity in the input data and least squares would overfit the training data.

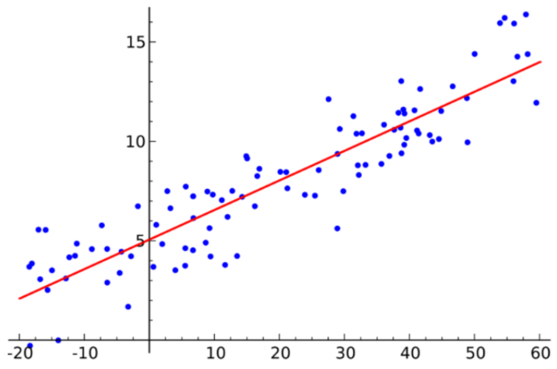


Figure 4. 1 Representation of a simple linear regression problem

2. *Logistic regression*

It is the primary option for binary classification problems. The function used at the core of the method is the logistic or sigmoid function (see Figure 4. 2). It is like a smooth step function. For very negative values, the sigmoid function returns values close to 0; on the contrary, for very positive values the function returns values close to 1. The difference between the sigmoid and step function is that for any input value, the sigmoid function will return a value in the interval (0,1). The analytical expression of the logistic function:

$$\frac{1}{1+e^{-x}}$$

Being x the input value.

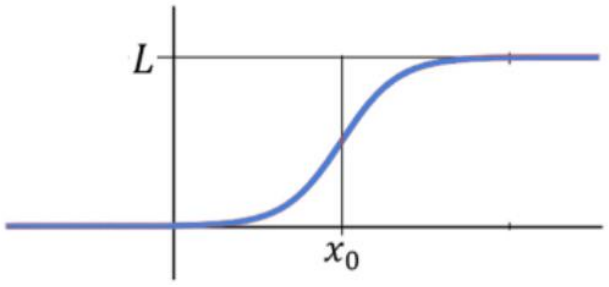


Figure 4. 2 Logistic function representation

The representation of a logistic regression model looks similar to the representation in linear regression. In this case, we are also calculating linear coefficients to predict an output dependent on input values, but the output is a binary value rather than a numerical value. The representation of a logistic regression model is shown below:

$$y = \frac{e^{\alpha_0+\alpha_1 \cdot x}}{1+e^{\alpha_0+\alpha_1 \cdot x}}$$

The logistic regression models the probability of the default class. Then, the probability prediction must be transformed into a binary value (0 or 1). Again, to estimate the coefficients of the model (learning process), an algorithm must be used. This is done using maximum-likelihood estimation from the training data. This learning algorithm is used by a variety of machine learning algorithms, although it does make assumptions about the distribution of the data. The best coefficients would result in a model that would predict a value very close to 1 for the default class and a value very close to 0 for the other class.

3. *K-Nearest neighbors*

This algorithm can be used for classification and regression problems. KNN has no model other than storing the entire dataset, learning is not required in this case.

Predictions with this algorithm are made for a new data point by searching through the entire training set for the K most similar instances and summarizing the output variable for those K instances. For classification this might be the mode class value, in regression this might be the mean output variable.

A distance measure has to be used in order to determine which of the K instances in the dataset are most similar to a new input. For real value input variables, the euclidean distance is the most used one, but there are other popular distance measures (e.g., Minkowski distance, Manhattan distance, etc.). This measure has to be selected based on the properties of the data.

KNN works well with a small number of input variables but struggles when the number of input variables is very large. This is because the dimensionality of the problem increases as more input values are added. Distance in high-dimensional problems is unintuitive.

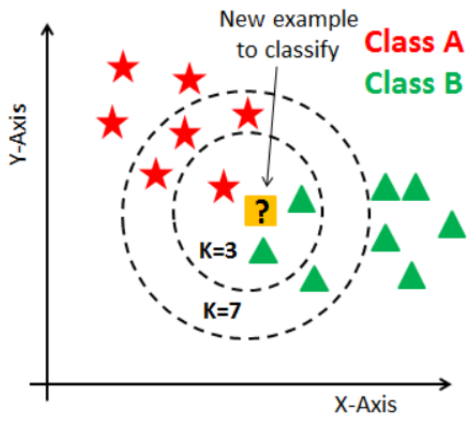


Figure 4. 3 Example of how KNN algorithm works

4. *Decision tree*

This algorithm is also called CART (Classification and Regression Trees). It can be used for classification or regression predictive modeling. The representation of a decision tree is a binary tree where each node represents an input variable (x) and a split on that variable. The leaf nodes contain an output variable (y), used to make a prediction. The tree can be stored as a graph or a set of rules (see Figure 4. 4).

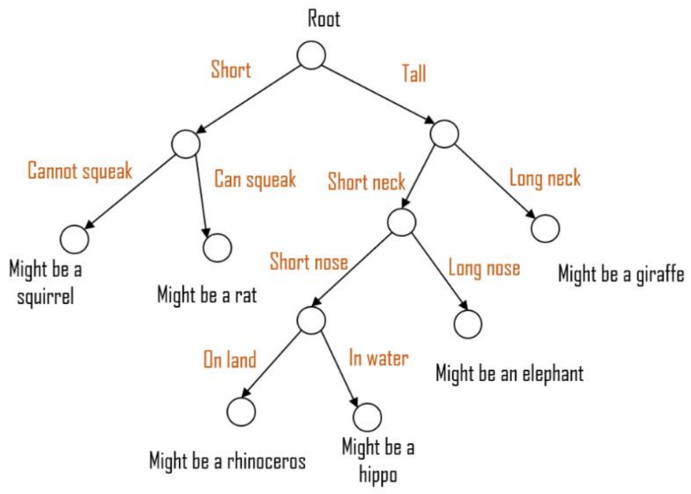


Figure 4. 4 Decision tree structure

Creating a binary decision tree is a process of dividing up the input space. Recursive binary splitting is used to divide the input space. It consists of a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function. The split with the lowest cost is selected. For regression, the cost function is the sum squared error across all training examples:

$$\sum_{i=1}^{n}(y_i - prediction_i)^2$$

Where y is the output for the training example and prediction is the predicted output of the tree. For classification problems, the Gini cost function is used, and it provides an indicator of how mixed the training data assigned to each node is (how pure the leaf nodes are):

$$G = \sum_{k=1}^{n} p_k \cdot (1 - p_k)$$

Where G is the cost function, p_k is the number of training instances with class k in the rectangle of interest. A node that has all classes of the same type will have G=0. The algorithm must have a stopping criterion, and the most used one is to use a minimum count of the number of training instances assigned to each leaf node.

5. *Support Vector Machines (SVM)*

Maximal-Margin Classifier: hypothetical classifier that best explains how SVM works in practice. It consists in calculating a hyperplane that splits the input variable space (e.g., two input variables create a two-dimensional space). In two dimensions, the hyperplane can be visualized as a line with the notation:

$$\alpha_0 + \alpha_1 \cdot x_1 + \alpha_2 \cdot x_2 = 0$$

Where x_1 and x_2 are the input variables, α_1 and α_2 are the coefficients that determine the slope of the line, and α_0 is the intercept. The coefficients are found by the learning algorithm. Using this line, classifications can be made.

The distance between the line and the closest data points is called the margin. The optimal line is the one that can separate two classes with the largest margin. The relevant points to defining the line are called support vectors.

Soft Margin Classifier: real data is messy and cannot be separated perfectly with a hyperplane. In this case, the constraint of maximizing the margin is relaxed, so some points in the training data can violate the separating line. New coefficients are introduced to give the margin wiggle room in each dimension (slack variables). The parameter that tunes this margin wiggle is called C. It defines the magnitude of the wiggle allowed across all dimensions. The higher the value of C, the more wiggle allowed.

SVM (kernels): SVM is implemented using a kernel. The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. The equation for making a prediction for a new input:

$$f(x) = B_0 + \sum_{i=1}^{n} (a_i * (x \ * x_i))$$

Where x_i are each of the support vectors, x is the input, and B_0 and a_i are estimated from the training data by the learning algorithm. There are different kernels apart from the linear that allow different kinds of classification.

The most used method to calculate the different parameters of the SVM model is the Sequential Minimal Optimization (SMO), which breaks the problem into sub-problems that can be solved analytically.

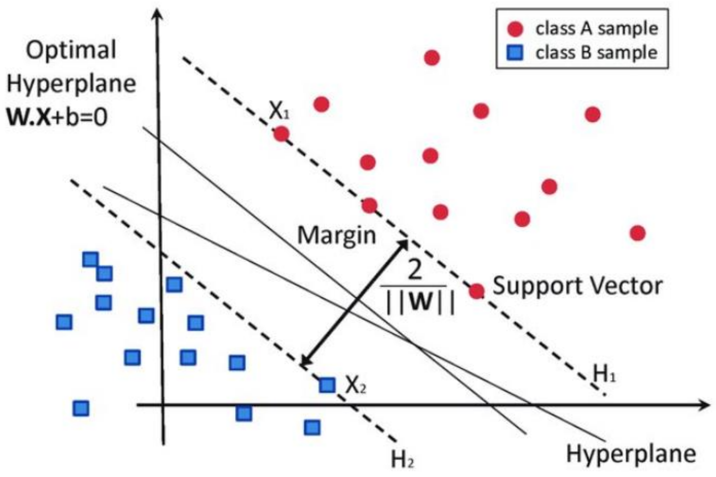


Figure 4. 5 SVM setup for a 2-Dimensional problem

6. *Bagging and Random forest*

Random Forest is a type of ensemble machine learning algorithm called Bootstrap Aggregation or bagging.

Bootstrap method: used to estimate a quantity from a data sample. The procedure is to create many random sub-samples with replacement of the original data set, extract the quantity from each sub-sample and finally calculate the average. This method is used in machine learning algorithms to estimate the learning coefficients of different models.

Bootstrap Aggregation (Bagging): this is a simple and powerful ensemble method. It combines the predictions from multiple machine learning algorithms to make more accurate predictions. Bagging can be used to reduce the variance of some algorithms (like decision trees). Decision trees are sensitive to specific data when they are trained. If the training data is changed (e.g. trained with a subset of the data) the predictions may differ. So in this case we are generating a lot of different samples of data and decreasing the variance of the model. The only parameter when bagging decision trees is the number of trees to create.

Random Forests are an improvement over bagged decision trees. Bagged decision trees still have a lot of structural similarities with decision trees that can result in high correlation in their predictions. In Decision Trees, when selecting a split point, the learning algorithm is allowed to look through all variables. In Random Forest, the learning algorithm is limited to a random sample of features which to search. This number has to be specified as a parameter of the algorithm.

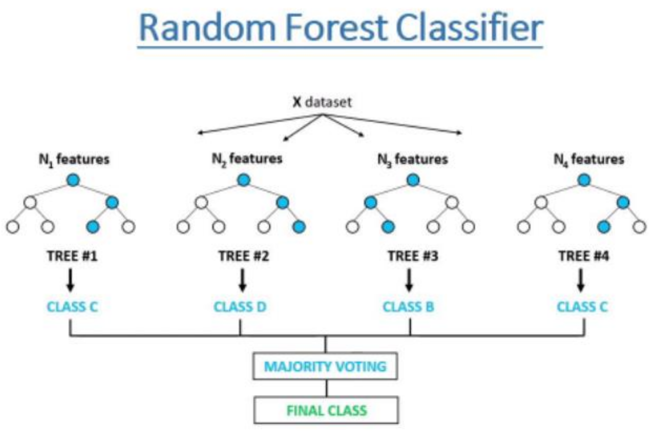


Figure 4. 6 Structure of a random forest classifier

7. *Naive Bayes*

It is a simple but powerful classification algorithm. This algorithm is based on the Bayes' Theorem with the independence assumptions between predictors. Based on naive Bayes, Gaussian naive Bayes is used for classification based on the binomial distribution of data.

$$P(class|data) = \frac{P(data|class) \cdot P(class)}{P(data)}$$

Where P(class/data) is the probability of a new data point having either class, given the data point, which is the value that we want to calculate; P(class) is the prior probability of class; P(data/class) is the probability of predictor given class and P(data) is the prior probability of predictor or marginal likelihood. Finally, the class with a higher probability is the chosen one.

4.1.2 Unsupervised learning

In unsupervised learning algorithms, direct control of the developer is not needed. The desired results are unknown and yet to be defined, and they work with unlabeled data. Some usages for unsupervised learning algorithms are: exploring the structure of the information, extracting valuable insights, and detecting patterns in the data (very interesting for big-data applications).

Unsupervised learning includes two major processes: clustering and dimensionality reduction.

- **Clustering**: the exploration of data used to segment it into different groups or clusters based on their internal patterns without prior knowledge of group credentials.
- **Dimensionality reduction**: if there is a lot of noise in the input data, these algorithms use dimensionality reduction to remove this noise and keep the relevant information.

Now, we are going to talk about the most common unsupervised learning algorithms.

1. *K-means clustering*

The objective of this algorithm is to group similar data points together and discover hide patterns in the data. This algorithm looks for a fixed number (k) of clusters in the dataset. The parameter of the model "k" is the number of needed centroids in the dataset. A centroid represents the center of the cluster. Once the centroids are defined, then every data point is allocated to each of the clusters through reducing the cluster sum of squares.

For the learning process, the algorithm starts with a first group of random centroids, that are used as the beginning points for each cluster and then performs iterative calculations to optimize the position of the centroids. There are two ways to stop the iteration: the centroids have stabilized or the defined number of iterations has been reached.

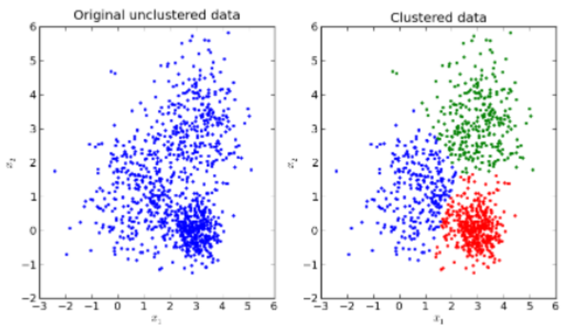


Figure 4. 7 Before and after using k-means clustering algorithm

2. *t-SNE*

This algorithm is a dimensionality reduction technique and its primary use is for visualization purposes. We are going to dissect the t-SNE name (t-Distributed Stochastic Neighbor Embedding):

- Embedding: high-dimensional data represented in a lower dimensional space.
- Neighbor: data-point that is close to the data point of interest.
- Stochastic: use of randomness in the iterative process when searching for a representative embedding.
- T-Distributed: probability distribution used by the algorithm to calculate similarity scores in the lower dimensional embedding.

Step 1: determining the similarity of points by measuring distances between the point of interest and the rest and placing them on a normal curve. Then, some scaling is applied to account for variations in the density of different regions. The result of the calculations is a matrix with similarity scores between each pair of points.

Step 2: then, the algorithm maps all the points onto a lower-dimensional space and calculates again similarities between points. This time, the algorithm uses a t-distribution

Step 3: now, the goal of the algorithm is to make the new similarity matrix look like the original one using an iterative approach.

3. *Principal component analysis*

It is a dimensionality reduction algorithm to transform a set of features in a dataset into a smaller number of features trying to retain as much information as possible. Some advantages of using this algorithm are: removing correlated features (reduces the training time of a machine learning model) and reduces overfitting. Some disadvantages are: independent variables are less interpretable, there is some part of the information that is lost and scaling of the features is required prior to processing.

4.2 Deep Learning Neural Networks applied to FDM simulation

In this section, we introduce and explain the concept of neural networks. Then, a review of its applications in the FDM technology is done.

4.2.1 What is a neural network? Types of neurons and architecture

Neural networks, also known as Artificial Neural Networks (ANN) are a subset of machine learning and the core of deep learning. Their structure is inspired by the human brain, mimicking the way that biological neurons signal to one another.

Neural networks are structured in different layers. There is one input layer, one output layer, and one or more hidden layers (see Figure 4. 8).

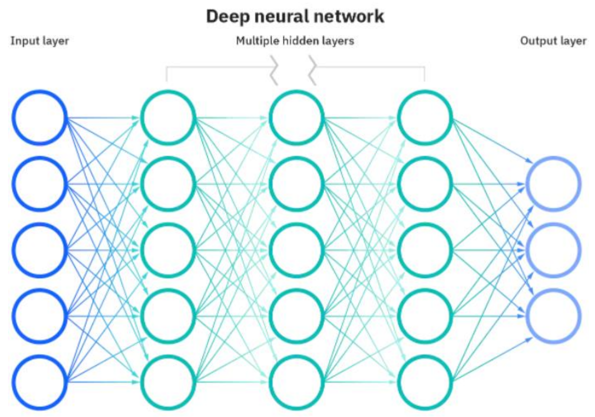


Figure 4. 8 Layered structure of a neural network

There are different types of artificial neurons. The most used ones are the perceptron and the sigmoid neuron (Nielsen, 2019).

- **Perceptron**: nowadays, other types of neural networks are commonly used. But to understand the basics of neural networks, it is worth understanding perceptrons. A perceptron takes binary inputs (x_i) and produces a single binary output (y). To compute the output, weights (ω_i) that express the importance of the respective inputs are introduced. The neuron's output is determined by whether the weighted sum of its inputs is less or greater than a threshold value:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

  Now, the concept of bias is introduced to simplify the notation. The bias is calculated as b = -threshold. So the notation changes to:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

  If the bias is very positive, it is easy for the neuron to output a 1. On the contrary, if the bias is very negative, it is difficult for the neuron to output a 1. To tune the weight and biases of each neuron, learning algorithms are used. The bad thing about perceptrons is that a small change in the weights or biases can cause a complete flip in the output, this is because we are introducing sigmoid neurons below.

- **Sigmoid neurons**: small changes in their weights and biases cause a small change in their output (y). The inputs of this neuron can have values between 0 and 1. It also has weights for each input ($\omega_1, \omega_2, \ldots$) and a bias, b. The output, in this case, is calculated as $\sigma(\omega \cdot x + b)$, where σ is the sigmoid function, expressed by:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

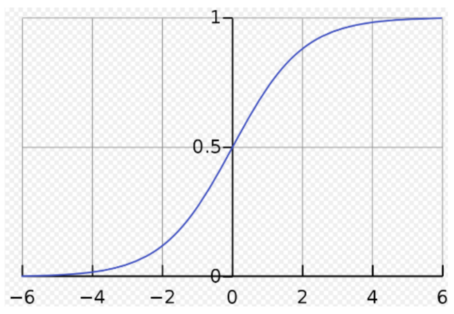


Figure 4. 9 Shape of a sigmoid function

So the output of a sigmoid neuron can be expressed as:

$$\frac{1}{1 + exp(-\sum_j \omega_j x_j - b)}$$

When $\sum_j \omega_j x_j - b$ is a very high value, the output is close to 1, just like the perceptron. Otherwise, if it is a very low value, the output is close to 0, again like the perceptron. But it is only when $\sum_j \omega_j x_j - b$ is of modest size that there is much deviation from the perceptron model.

The sigmoid function is called an activation function of the neuron. There are more activation functions, and the sigmoid is commonly used in the neural network field. The output of a sigmoid function can be any real number between 0 and 1.

- **ReLu activation function**: may be the most used activation function for neural networks nowadays. The definition of this function f(x) is simple: it takes the value zero for values of x<0 and the value x when x>0 (see Figure 4. 10)

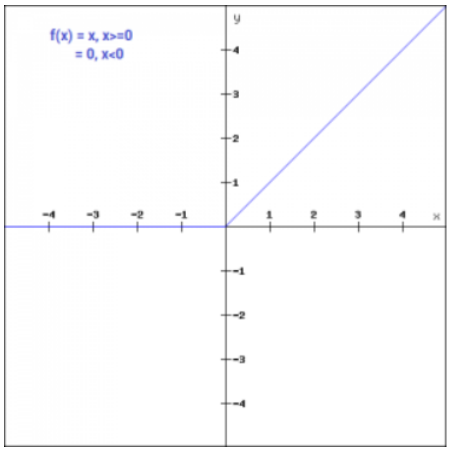


Figure 4. 10 ReLu activation function

- **Softmax activation function**: takes vectors of real numbers as inputs, and normalizes them into a probability distribution proportional to the exponentials of the input numbers. The output value of this function will be in the range of 0 and 1 and the elements will add up to 1 (probability distribution). It is used in the output layer of neural network classifiers, other multiclass classification methods, and reinforcement learning. The analytical form of the softmax function is shown below.

$$softmax(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; for\; j = 1, \ldots, K$$

Architecture of neural networks

Suppose we have the network in Figure 4. 11:

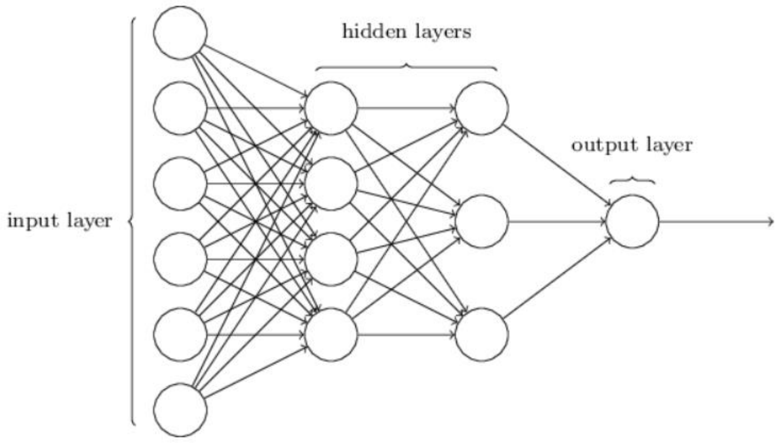


Figure 4. 11 Architecture of a neural network

The leftmost layer is the input layer, which contains the input neurons. The rightmost layer is the output layer and contains the output neurons (in this case only one output). The layers in the middle are the hidden layers (not input or output layers).

The design for the input and output layers is often straightforward because it depends on the format of the input data and the output that we want to obtain. To design the hidden layers, there are no simple rules of thumb. There are many design heuretics developed by researchers which help to get the desired behavior of the net. Up to now, we only talked about networks where the output from one layer is the input of the next layer, so there are no loops. However, there are other networks in which feedback loops are possible. These models are called recurrent neural networks, which are very promising.

4.2.2 Most used algorithms to train neural networks

For a neural network to learn, the objective is the minimization of a loss index. The loss index is a function that measures the performance of the neural network on a data set.

The loss function has an error and a regularization term. The error term evaluates how the model fits the data set and the regularization term prevents overfitting of the data. The loss function depends on the biases and weights of the neural network.

Summed up, the learning process for neural networks is basically searching the weights and biases of the network at which the loss function takes a minimum value. This process often is a non-linear problem (the loss function is non-linear). Iterative algorithms have to be applied in order to solve it. The start point of the process is to pick random values of weights and biases and then, with each iteration of the algorithm, the weights and biases are changing while the loss function decreases.

Now, we are going to describe the most used algorithms in the neural network learning process.

1. *Gradient descent*

We will denote the weights and biases as a parameter vector ω, the loss function in one point as $f^{(i)}$ and the gradient as $g^{(i)}$. The method begins at a point $\omega^{(0)}$ and, until a stop criterion is satisfied, moves from $\omega^{(i)}$ to $\omega^{(i+1)}$ in the training direction of $-g^{(i)}$, so one iteration of this algorithm works as we can see below:

$$\omega^{(i+1)} = \omega^{(i)} - g^{(i)}\eta^{(i)}$$

Where the parameter η is the learning rate. This value is obtained by one-dimensional optimization along the training direction at each step. In some cases, a fixed value for the learning rate is used.

The gradient descent requires many iterations for some functions with narrow valley structures. In addition, the gradient direction is in which the loss function decreases the most rapidly, but not necessarily produce the fastest convergence. Even so, is the recommended algorithm for massive neural networks because it stores the gradient vector (n size), not the Hessian matrix (n^2 size).

2. *Newton's method*

It is a second order algorithm because it uses the Hessian matrix, H (second derivatives). Considering the cuadratic approximation of the loss function f at $\omega^{(0)}$ using Taylor's series:

$$f = f^{(0)} + g^{(0)} \cdot (\omega - \omega^{(0)}) + 0.5 \cdot (\omega - \omega^{(0)})^2 \cdot H^{(0)}$$

The Newton's method equation is usually denoted as:

$$\omega^{(i+1)} = \omega^{(i)} - (H^{(i)-1} \cdot g^{(i)})\eta$$

Where η is the training rate. The vector $H^{(i)-1} \cdot g^{(i)}$ is called Newton's training direction. Newton's method requires fewer steps than the gradient descent but the computational cost is much higher because it has to store the Hessian matrix and its inverse.

3. *Conjugate gradient*

Motivated to accelerate the convergence in the gradient descent algorithm. It avoids the storage of the Hessian matrix, evaluation, and inversion as Newton's requires.

In this algorithm, the search to diminish the loss function is performed along with conjugate directions, which generally provide faster convergence than gradient descent directions. It starts with an initial parameter vector $\omega^{(0)}$and an initial training direction vector $d^{(0)} = -g^{(0)}$. Then, the algorithm constructs a sequence of training directions as:

$$d^{(i+1)} = g^{(i+1)} + d^{(i)} \cdot \gamma^{(i)}$$

Where γ is called the conjugate parameter. The parameters are then improved as:

$$\omega^{(i+1)} = \omega^{(i)} + d^{(i)} \cdot \eta^{(i)}$$

The training rate η is found by line minimization. This method has proved to be more effective than gradient descent.

4. *Stochastic gradient descent*

The gradient descent has a big problem for large datasets, i.e. its amount of computation for each step. Let's imagine we have 20000 data points and 15 features. The sum of squared residuals consists of as many terms as data points, 20000. In each iteration, the derivative of this function with respect to all the features has to be computed, 20000*15=300000 computations per iteration. If we are taking 1000 iterations, 300000000 computations per step are needed. This would be a very slow process.

The SGD randomly picks one data point from the whole data set at each iteration to reduce the computations enormously. The algorithm can also sample a small number of data points at each step and that is called mini-batch gradient descent.

5. *Backpropagation method*

Is a method of adjusting the weights and biases to minimize the loss function by moving from the right (output layer) to the left (input layer). Is one of the preferred methods to train a neural network.

The first step in the algorithm is to initialize the weights and biases randomly and calculate the error between the prediction and the label. Then, the weights and biases of the previous layers are updated using the stochastic gradient descent algorithm, following the equation:

$$\omega_{new} = \omega_{old} - (\alpha \cdot \frac{dE}{d\omega})$$

Where ω_{new} is the new weight, ω_{old} is the old one, α is the learning rate and $\frac{dE}{d\omega}$ is the partial derivative of the error with respect to the weight. This is an iterative process for each layer.

For the calculation of the rate of change of error to the weights and biases ($\frac{dE}{d\omega}$), the chain rule in different steps is used.

4.2.3 Review of neural networks applied to the FDM printing process

In (Mohamed et al., 2021) an application of ANN (Artificial Neural Networks) for modeling and optimization of dimensional accuracy in cylindrical FDM parts is explained. The proposed ANN model has six process parameters as inputs (i.e. slice thickness, air gap, raster angle, part print direction, bead width, and number of contours) and two different part distortions of the cylindrical parts (percentage difference in length and percentage difference in diameter).

The objective of the neural network is to understand the input-output relationships and also determine the optimum process setting for the minimum distortion in the part. To obtain the optimum settings, K-fold cross-validation was used. This algorithm partitions the data into K-subsets. Some of the K subsets are used to train the data and the remaining subsets are used for prediction. The fit goodness of the ANN model was evaluated based on R^2, root mean square error (RMSE), and mean absolute deviation (MAD).

The number of hidden layers was determined by trial and error and the optimum value was found to be 10 hidden layers. The neural network structure is shown in Figure 4. 12

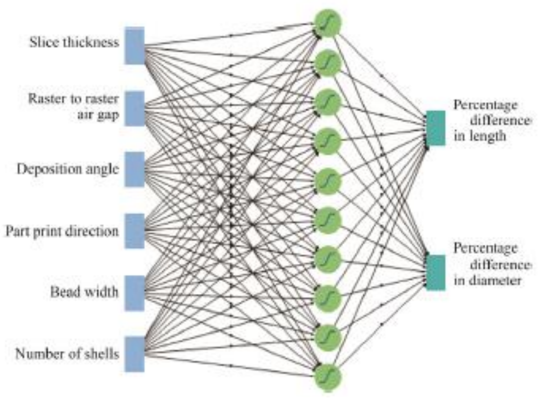


Figure 4. 12 Neural network architecture

10 hidden layers were selected based on the high R^2 and minimum sum of squares of the error.

The R^2 for the validation set in both models exceeds 98% and the RMSE values are very close to zero. This is a confirmation of the accurate predictions that the model can perform in data not used for training.

In (Khanzadeh et al., 2018), an unsupervised neural network is used to categorize point cloud measurements of FDM printed parts to cluster those measurements which have similar deviations in terms of severity and direction.

The point cloud of the parts is generated by a 3D laser scanner, and the geometric deviations can be calculated by comparing the point cloud data with the CAD model of the part.

For the experimental results, two process parameters are varied: the extrusion temperature, with five different levels (220 ºC, 225 ºC, 230 ºC, 235 ºC, and 240 ºC), and the infill percentage, with four levels (70%, 80%, 90%, and 100%). This means that there are twelve possible treatments, but in some of them, the failure of the printing occurred.

The chosen unsupervised approach is the concept of SOM (Self Organizing Map), useful to identify different parameters types of geometric deviations associated with specific process conditions. It maps high-dimensional input data into a 2-D space, preserving the topological relationship between the data. The clustered data reduce the dimensionality and characterize the similarity along the data points.

In Figure 4. 13, we can see a neural network schematic, with three input attributes (the three geometric deviations in the three directions $\Delta x,\ \Delta y,\ and\ \Delta z$) and 16 different clusters as the output of the network. This network is composed of three types of entities: input neurons, connection vectors, and output neurons. The vectors connect the input neurons to the output neurons and neighboring output neurons to each other. Each vector has a weight produced randomly in the initialization procedure and updated in the training. The network in Figure 4. 13 yields a quadrilateral map with at most four connections for each neuron with its neighbors.

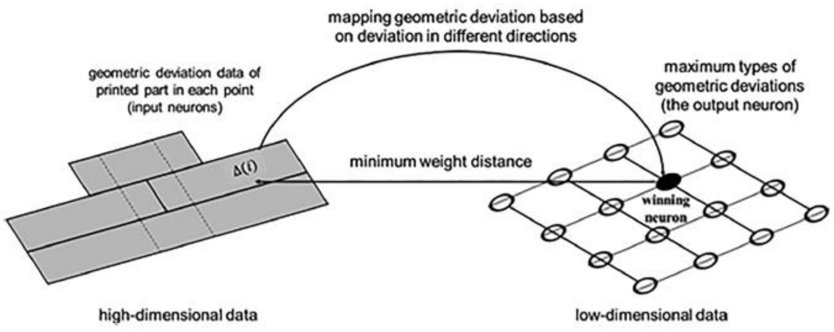

mapping geometric deviation based on deviation in different directions
geometric deviation data of printed part in each point (input neurons)
maximum types of geometric deviations (the output neuron)
Δ(i)
minimum weight distance
winning neuron
high-dimensional data
low-dimensional data

Figure 4. 13 Schematic of the SOM neural network

In (Khanzadeh et al., 2018), a 5x5 hexagonal (at most 6 connections for each neuron with its neighbors) SOM was used, which means 25 different clusters. The more clusters in the SOM model, the more subtle differences among geometric deviations can capture, but the model is more sensitive to noise. Lower-order models capture the major types of geometric deviations, but not the subtle differences. In Figure 4. 14, it can be seen the results for specific process parameters. Each hexagon represents one cluster, and the number inside is the number of points classified in each cluster.

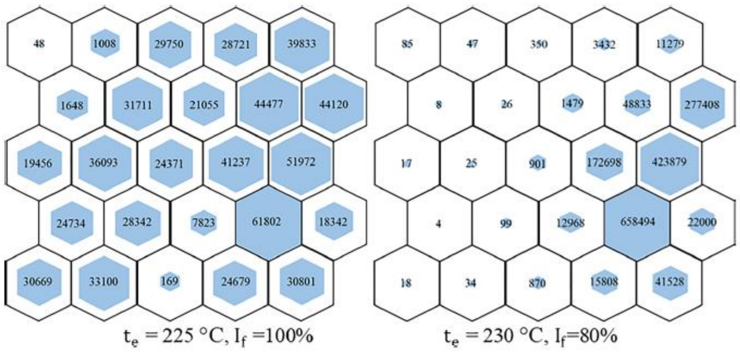


Figure 4. 14 Results of the SOM-based neural network for specific process parameters

The results were verified using the k-means unsupervised algorithm and both showed concordance in the results.

In (Williams et al., 2019), the objective is to investigate the extent to which enforcing design repository standardization impacts the capability of a machine learning model to analyze new geometrical data. For this purpose, a 3D CNN (convolutional Neural Network) that assesses build metrics of part mass, required support part mass, and build time for creating new FDM printed parts is implemented.

An artificial design repository was created to achieve greater sample size and consistency. It includes planar and curved surfaces, a variety of bounding box aspect ratios, and concave and convex geometries. The repository was based on 18 parametric design templates (see Figure 4. 15).

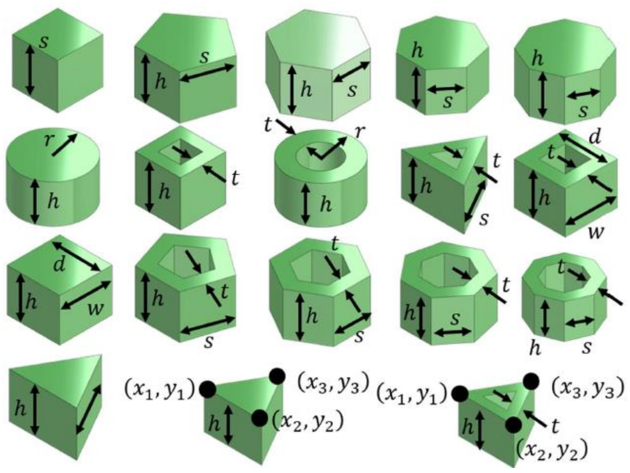


Figure 4. 15 Parametric design templates for the repository

Modeling the designs in Figure 4. 15 parametrically, thousands of different designs were obtained. Designs are modeled in four steps: defining the design parameters, creating the CAD model, creating a tessellated mesh, and finally the conversion to a binary voxel representation (64x64x64 element matrix). The material presence is represented by a 1 and the void is represented by 0.

Four different treatments were applied to the artificial repository: no transformation (N), translation (T), rotation (R), and translation plus rotation (T+R). In total, 16 artificial repositories were created, four for each treatment. The total number of unique geometries in the study was 72000. All artificial repositories were used as training and testing data but never mixed during the training process.

3 different 3D convolutional neural networks were used, one for each quantitative build metric estimation. In Figure 4. 16, the chosen architecture for the neural networks is shown. They accept a voxel-based 3D geometric input (262,144 individual points) and consist of 10 layers.

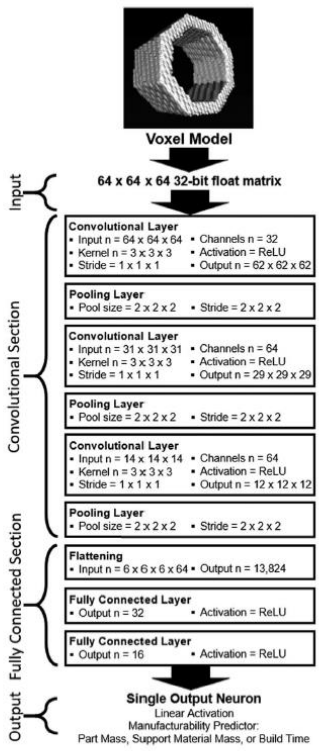

Voxel Model
Input
64 x 64 x 64 32-bit float matrix
Convolutional Section
Convolutional Layer
• Input n = 64 x 64 x 64 • Channels n = 32
• Kernel n = 3 x 3 x 3 • Activation = ReLU
• Stride = 1 x 1 x 1 • Output n = 62 x 62 x 62
Pooling Layer
• Pool size = 2 x 2 x 2 • Stride = 2 x 2 x 2
Convolutional Layer
• Input n = 31 x 31 x 31 • Channels n = 64
• Kernel n = 3 x 3 x 3 • Activation = ReLU
• Stride = 1 x 1 x 1 • Output n = 29 x 29 x 29
Pooling Layer
• Pool size = 2 x 2 x 2 • Stride = 2 x 2 x 2
Convolutional Layer
• Input n = 14 x 14 x 14 • Channels n = 64
• Kernel n = 3 x 3 x 3 • Activation = ReLU
• Stride = 1 x 1 x 1 • Output n = 12 x 12 x 12
Pooling Layer
• Pool size = 2 x 2 x 2 • Stride = 2 x 2 x 2
Fully Connected Section
Flattening
• Input n = 6 x 6 x 6 x 64 • Output n = 13,824
Fully Connected Layer
• Output n = 32 • Activation = ReLU
Fully Connected Layer
• Output n = 16 • Activation = ReLU
Output
Single Output Neuron
Linear Activation
Manufacturability Predictor:
Part Mass, Support Material Mass, or Build Time

Figure 4. 16 Arquitecture of the 3D CNN

In the results, it is shown how the CNN model can predict with high accuracy the needed mass for the part and the build time (average COD values above 0.9 for all the training treatments) and with much less accuracy the support material mass (average COD below 0.8 for all the training treatments).

4.3 Possible applications of neural networks in this work

In the previous, we have reviewed some of the applications of machine learning in the FDM printing field with satisfactory results.

In the present work, a numerical simulation method was studied in order to simulate the FDM printing process. The thermal and structural results of the simulation could be used to develop predictive machine learning models. At the same time, with these predictive tools, the FDM techniques could be enhanced to achieve more accuracy and standardization in the parts. This would be a very important step since in some fields FDM cannot be used yet due to its lack of dimensional precision.

In this section, we are going to describe some ideas of how neural networks could be implemented using the results of the studied numerical simulation in this work.

1. *Deformation prediction of a thin wall FDM printed part in the three directions based on the infill pattern*

For this case, the starting point is a fixed wall geometry, with defined height, width, and thickness. The objective of this application is to predict the deformation values for different infill patterns. In the case of this work (section 3.3.1), only four infill patterns were studied. In order to train a neural network successfully, thousands of different infill patterns would be needed.

To reach this amount of different data, a cellular automaton-like method could be used to generate different infill patterns. The input data would be pixels that define a 2D image as we can see in Figure 4. 17, where the black pixels represent material and the white pixels represent void.

A cellular automaton method consists of a collection of colored cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells.

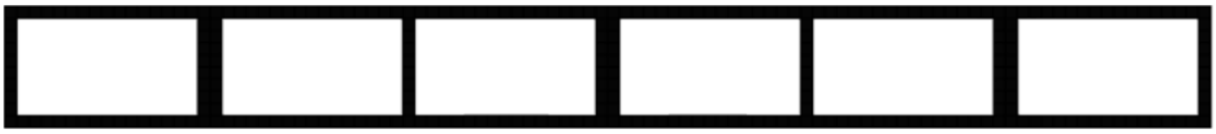


Figure 4. 17 2D image representing an infill pattern for developing a neural network

One possible configuration for the input data could be 100x10, having a total of 1000 pixels for each input image. A very used approach for processing 2D images is the CNNs (Convolutional Neural Networks).

The prediction of the deformation in the three directions based on the infill pattern is a regression supervised problem. This means that we need to create labeled training data. FEA (Finite Element Analysis) can be used to calculate the deformations of the training data. The training data could be created experimentally, but it is not cost worthy since a lot of different parts should be printed.

Knowing that we have 1000 pixels as inputs and 3 deformation values as outputs, we have the number of input and output neurons defined, respectively.

To choose the number of hidden layers, we have to look at the dimensionality of the input data. In this case, our input data (pixels) only has two states (1 for material, 0 for void), so one or two hidden layers could work. Let’s say we choose two hidden layers.

Then, to choose the number of neurons in the hidden layers, a typical criterion is used:

$$n^{o}\ hidden\ neurons = \sqrt{n^{o}\ input\ neurons * n^{o}\ output\ neurons} = \sqrt{3000} = 54.77$$

Also, the number of neurons should keep on decreasing in subsequent layers to get more and more close to pattern and feature structure. Having this into account, 35 neurons are chosen for the first hidden layer and 15 for the second hidden layer, making a total of 50 hidden neurons.

The activation function chosen for this example is the ReLU, which works very well with regression problems like in this case. The chosen loss function is the mean squared error (MSE) between the predicted value and the true value (given by FEA or experimental measurement). The algorithm used to find the weigths and biases that minimize the loss function is the Adam optimizer mechanism. This is basically an optimized gradient descent algorithm that converges faster. For this optimizer to work, the learning rate and batch size have to be defined. Common values for these hyper-parameters are 0.01 and 64 respectively.

2. *Deformation prediction of a thin wall FDM printed part in the three directions dependent on the infill density, extrusion temperature, and bed temperature*

In this work, we performed simulations with different infill patterns (and different infill densities), extrusion temperatures, and bed temperatures. With each combination of factors, the deformation results are different. Instead of having to predict the deformation results by running long numerical simulations, we could develop a machine learning model to predict the results much faster.

In this case, this problem is also a regression supervised problem. We need labeled data (FEA or experimental measurements) to train the model. We have three inputs and three outputs (deformations in the three directions), so the architecture of the network is 3-n-3. We have low-dimensional data as input, so one hidden layer should be enough. The number of neurons is often chosen by trying different options and choosing the optimum value, but between 6 and 10 hidden neurons seem reasonable.

The chosen activation function, in this case, is also ReLU and the loss function is the mean squared error. The training algorithm used could be the Adam optimizer again, but due to the simplicity of this neural network, the stochastic gradient descent could be used to avoid too much complexity in the coding.

5 Potential applications and future work

The simulation framework developed in this work has lots of applications in many industries nowadays. It is worth mentioning that with the developed simulation framework in this work, almost every additive manufacturing process could be simulated.

Numerical simulation of manufacturing processes like additive manufacturing has lots of advantages, some of them are:

- Predicting material behavior and validating additive manufacturing designs when optimizing for manufacturing.
- Flexibility: finite element is a flexible method of analysis. It can be applied to different geometries and process conditions and obtain robust results.
- Troubleshooting: when the behavior of the process does not agree with the expectations, numerical simulations can help to determine the main reason for the problem and suggest solutions.
- Numerical simulations can obtain accurate results for high non-linear models, like in the case of additive manufacturing models.
- Numerical simulations can be used to generate data to develop data-driven prediction models with high accuracy. The accuracy of the data-driven models depend on the quality of the data used to train the model.

There are a lot of engineering fields in which high-dimensional accuracy parts are needed. In these fields, lightweight parts play an important role in order to minimize the material and energy needed in the process and improving efficiency. Thus, additive manufacturing is a very promising technique to achieve these goals. The most important engineering fields in which high accuracy additive manufacturing parts could play a major role in the future are:

- Aerospace: aerospace components require highly complex geometric structures in sometimes very tight spaces, so the high dimensional precision of the printing is a key factor. Thus, consistent manufacturing processes are needed to improve the repeatability of the parts. Adittive manufacturing can be used to produce lightweight parts by mass optimization and reduce storage needs.
- Medical: additive manufacturing has increasing importance in the medical field. The main tasks for the AM technology in the medical field are medical models, used for pre- and postoperative planning and training, training for students, etc; implants, that can be directly or indirectly additively manufactured to replace defective or missing tissue (e.g. dental applications); tools and medical devices that allow or enhance clinical performance; biomanufacturing, in which the materials need to be biologically compatible with the human body. All of these applications require high dimensional accuracy in the 3D printing process.
- Transportation: automotive companies are adopting 3D printing technology to fabricate car components and develop personalized services, especially in luxury brands, with components like carbon fiber.

For all these applications that require high accuracy in the 3D printing, the optimization of the process parameters is a promising perspective in order to achieve high dimensional fidelity and repeatability in the process. Machine learning models could be used to predict the geometric deviation of the parts and obtain the optimum process parameters for a minimum part distortion.

There are still some challenges to taking advantage of the full potential of the coupling between numerical simulation and machine learning in the field of additive manufacturing. Some of the most important challenges are described below.

- Regarding the numerical simulation, the most field of improvement, especially for the FDM simulation, is the modeling of the contact between the part and the build plate, to evaluate the adhesion force, which is influenced by local imperfections, impurities, and a non-uniform temperature distribution of the build plate. To avoid the influence of this adhesion force, a raft is usually printed first. The raft is wasted material, so if the numerical simulation could model this interaction, we may avoid this material waste by adjusting the process parameters. Another field of improvement regarding numerical simulation is reducing the time and computational costs needed. This could be improved by optimizing the element activation algorithms.
- More research on how to integrate ML into simulation has to be done. ML techniques can be integrated into a model in order to reduce model order and offer approximate but simpler solutions. This would cause a noticeable reduction in simulation time.
- Specialization of the ML predictive models: existing research on the application of ML techniques to additive manufacturing has a very general vision. Thus, the general results of this research are not applicable to all cases. To fully extract the ML potential, the ML models should be personalized for each part geometry and machine. This way, the specialized ML model would predict the performance of the process with more accuracy than using a generalized model.

6 Conclusions

In this work, we developed a simulation framework based on Abaqus software and we studied the application of machine learning techniques to the FDM process using the simulation results. Regarding the simulation, we can conclude that:

- Numerical simulation of additive manufacturing processes requires a lot of time and computational resources.
- For element activation in the printing process simulation, we have compared two methods: model change and the toolpath-mesh intersection method. It was found that the toolpath intersection method is more efficient since there is no need to define a lot of steps and interactions for activating subsets of elements in each step, unlike the model change method.
- We simulated with different time steps in the printing step of the simulation. It was found that the printing step influences more the thermal analysis results than the structural results. For the thermal results, the smaller the time step, the higher the activation temperature of the elements (in the limit of a time step of zero, should be the same as the extrusion temperature of the nozzle). The time step also greatly influences the simulation time (the smaller the time step, the more time required for the simulation).
- From the deformation results, it is worth mentioning that the infill pattern has the least dimensional distortion in the Y and Z directions. All the infill patterns have approximately the same dimensional distortion in the X direction.
- A parametric study for the thin wall geometry was performed. The parameters included in the study were four different infill patterns, three bed temperatures, and three extrusion temperatures. The response was the maximum displacement in the three directions, so one ANOVA table for each direction was computed. The results showed that the infill pattern was the most influential factor for the three directions. For the X direction, all the factors showed to be influential for the deformation response; for the Y direction the only factor that was not influential was the extrusion temperature; for the Z direction, the three parameters are influential.

Regarding the machine learning application in the additive manufacturing prediction:

- It was concluded that ML techniques can be applied to additive manufacturing processes with high accuracy results.
- Interaction between numerical simulation and machine learning models is needed in order to develop more efficient predictive models. This can be done in two ways: generating the training data with the simulation results or introducing machine learning models into the simulation to reduce the problem order and get simpler and faster solutions.

7 References

Akbas, O. E., Hira, O., Hervan, S. Z., Samankan, S., & Altinkaynak, A. (2020). Dimensional accuracy of FDM-printed polymer parts.

An, N., Yang, G., Yang, K., Wang, J., Li, M., & Zhou, J. (2021) Implementation of Abaqus user subroutines and plugin for thermal analysis of powder-bed electron-beam-melting additive manufacturing process.

Bhandari, S., & Lopez-Anido, R. A. (2020). Discrete-event simulation thermal model for extrusion-based additive manufacturing of PLA and ABS. *Materials*, *13*(21), 4985.

Brownlee, J. (2016). *Master machine learning algorithms*.

Cattenone, A., Morganti, S., Alaimo, G., & Auricchio, F. (2019). Finite element analysis of additive manufacturing based on fused deposition modeling: distortions prediction and comparison with experimental data. *Journal of Manufacturing Science and Engineering*, *141*(1).

Courter, B., Jing Bi, V.S, & J. Hansen, S. C. (2017). Finite Element Simulation of the Fused Deposition Modelling Process.

Equbal, A., Akhter, S., Equbal, Md. A., & Sood, A. K. (2021). Application of Machine Learning in Fused Deposition Modeling: A Review.

Farah, S., Anderson, D. G., & Langer, R. (2016). Physical and mechanical properties of PLA, and their functions in widespread applications—A comprehensive review. *Advanced drug delivery reviews*, *107*, 367-392.

Gardan, J. (2016). Additive manufacturing technologies: state of the art and trends.

ISO, A. (2015). ASTM52900-15. *Standard Terminology for Additive Manufacturing—General Principles—Terminology*.

Khanolkar, M. P., Abraham, A., McComb, C., & Basu, S. (2020). Using Deep Image Colorization to Predict Microstructure-Dependent Strain Fields. *48th SME North American Manufacturing Research Conference, NAMRC 48.*

Khanzadeh, M., Rao, P., Jafari-Marandi, R., Smith, B. K., Tschopp, M. A., & Bian, L. (2018). Quantifying geometric accuracy with unsupervised machine learning: Using self-organizing map on fused filament fabrication additive manufacturing parts. *Journal of Manufacturing Science and Engineering*, *140*(3).

Kiendl, J., Gao, C. (2020). Controlling toughness and strength of FDM 3D-printed PLA components through the raster layup.

Liu, G., Zhang, X., Chen, X., He, Y., Cheng, L., Huo, M., Yin, J., Hao, F., Chen, S., Wang, P., Yi, S., Wan, L., Mao, Z., Chen, Z., Wang, X., Cao, Z., & Lu, J. (2021). Additive manufacturing of structural materials.

Materials available. (2021).

Mohamed, O. A., Masood, S. H., & Bhowmik, J. L. (2021). Modeling, analysis, and optimization of dimensional accuracy of FDM-fabricated parts using definitive screening design and deep learning feedforward artificial neural network. *Advances in Manufacturing*, *9*(1), 115-129.

Mohamed, O. A., Masood, S. H., & Bhowmik, J. L. (2016). Mathematical modeling and FDM process parameters optimization using response surface methodology based on Q-optimal design. www.elsevier.com/locate/apm

Moreau, C. (2021). The state of 3D printing. *Sculpteo*.

Morgan, R. V., Reid, R. S., Baker, A. M., Lucero, B., & Bernardin, J. D. (2017). *Emissivity measurements of additively manufactured materials*.

Mwema, F. M., & Akinlabi, E. T. *Fused Deposition Modeling: Strategies for Quality Enhancement*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-48259-6

Nasiri, S., & Khosravani, M. R. (2021). Machine learning in predicting mechanical behavior of additively manufactured parts.

Nielsen, M. (2019). *Neural Networks and Deep Learning*.

Poel, G. V., Mathot, V. B., & Ye, P. (2011). Crystallization Temperature vs. Cooling Rate: the Link with "Real-Life" Polymer Processes. *Inc. PerkinElmer. Differential Scanning Calorimetry*.

Pereira, L. N. (2019). The effects of 3D printing Parameters and Surface Treatments on Convective Heat Transfer Performance.

Pérez, M., Medina-Sánchez, G., García-Collado, A., Gupta, M., & Carou, D. (2018). Surface Quality Enhacement of Fused Deposition Modeling (FDM) Printed Samples Based on the Selection of Critical Printing Parameters.

Song, X., Feih, S., Zhai, W., Sun, C., Li, F., Maiti, R., Wei, J., Yang, Y., Oancea, V., Brandt, L. R, & Korsunsky, A. M. (2020). Advances in additive manufacturing process simulation: Residual stresses and distortion predictions in complex metallic components.

Sun, Q., Rizvi, G. M., Bellehumeur, C. T., & Gu, P. (2008). Effect of processing conditions on the bonding quality of FDM polymer filaments.

Sun, Y., Hanhan, I., Sangid, M. D., & Lin, G. (2020) Predicting Mechanical Properties from Microstructure Images in Fiber-reinforced Polymers using Convolutional Neural Networks.

Vahabli, E. & Rahmati, S. (2017). Improvement of FDM parts' surface quality using optimized neural networks - medical case studies.

Williams, G., Meisel, N. A., Simpson, T. W., & McComb, C. (2019). Design repository effectiveness for 3D convolutional neural networks: application to additive manufacturing. *Journal of Mechanical Design*, *141*(11).

Zhou, C., Guo, H., Li, J., Huang, S., Li, H., Meng, Y., Yu, D., de Claville Christiansen, J., & Jiang, S. (2016). Temperature dependence of poly (lactic acid) mechanical properties. *RSC advances*, *6*(114), 113762-113772.

Zhu, Z., Anwer, N., Huang, Q., & Mathieu, L. (2018). Machine learning in tolerancing for additive manufacturing. *CIRP Annals*

Appendix 1

In this appendix, the code to obtain the ANOVA tables needed for the parametric study of the wall geometry. The code was implemented in the statistical tool R.

```
defx <-
c(0.0359,0.036,0.036,0.0359,0.0359,0.036,0.0358,0.0359,0.0359,0.0375,0.0377
,0.0377,0.0375,0.0376,0.0378,0.0375,0.0376,0.0376,0.0361,0.0362,0.0363,0.03
61,0.0362,0.0362,0.0360,0.0361,0.0362,0.0378,0.0378,0.0377,0.0378,0.0377,0.
0377,0.0378,0.0377,0.0377)

defy <-
c(0.0238,0.0242,0.0246,0.0238,0.0242,0.0243,0.0238,0.0242,0.0244,0.033,0.03
28,0.0328,0.0334,0.0332,0.0332,0.0338,0.0337,0.0337,0.0125,0.0127,0.0127,0.
0126,0.0128,0.0128,0.0128,0.0129,0.013,0.0128,0.013,0.013,0.0129,0.013,0.01
31,0.013,0.0131,0.0132)

defz <-
c(0.0529,0.0533,0.0538,0.0531,0.0536,0.0538,0.0534,0.0538,0.054,0.0573,0.05
78,0.058,0.0576,0.0581,0.0583,0.058,0.0584,0.0586,0.0475,0.0479,0.0481,0.04
78,0.0482,0.0484,0.0481,0.0485,0.0487,0.0531,0.0535,0.0537,0.0534,0.0538,0.
054,0.0537,0.0541,0.0543)


A <-
c(1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4)

B <-
c(1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3)

C <-
c(1,1,1,2,2,2,3,3,3,1,1,1,2,2,2,3,3,3,1,1,1,2,2,2,3,3,3,1,1,1,2,2,2,3,3,3)


pattern <- as.factor(A)

extrusiontemp <- as.factor(B)

bedtemp <- as.factor(C)


resx <- lm(defx~pattern+extrusiontemp+bedtemp)

resy <- lm(defy~pattern+extrusiontemp+bedtemp)

resz <- lm(defz~pattern+extrusiontemp+bedtemp)


anova(resx)

anova(resy)

anova(resz)
```

Appendix 2

In this work, the used geometries for the numerical analysis were simple, so the toolpath was calculated using excel. For complex geometries is not efficient to use excel, and scripting should be used to convert the G-code directly into an event series format. In this appendix, a code that allows this conversion is presented. It is only compatible with the slicer Replicator G, so if another slicer is used, the code will have to be modified. To generate the event series, the command line should have the following format: abaqus python generateEventSeries.py g_code_filename roller_pad_dimension delay_between_layers power. Two output files are generated, the laser toolpath and the roller pad dimension for each layer. If we want only the toolpath for FDM analysis, we simply set the power to zero.

1. Main Function

```
if __name__ =="__main__":
#def main():

    import os,sys

    #---------
    #Handling arguments
    #---------
    if len(sys.argv)!=5:
        print("The python script should be called with 4 arguments -
'generateEventSeries.py toolPathFileName rollerPadDimension
delayBetweenLayers powerOftheLaser' Eg generateEventSeries.py cube.gCode
0.5 10 400")
        exit()

    fileName  = sys.argv[1]
    rollerPad = sys.argv[2]
    userDelay = sys.argv[3]
    userPower = sys.argv[4]

    #fileName = 'Model-Horizontal.gcode'
    #rollerPad = .5

    #userDelay=10
    #userPower=300

    try:
        rollerPad = float(rollerPad)
    except:
        print ("The roller pad value should be a float")
        exit()

    try:
        userDelay = float(userDelay)
    except:
        print ("The delayBetweenLayers value should be a float")
        exit()

    try:
        userPower = float(userPower)
    except:
        print ("The power value should be a float")
        exit()


    if not os.path.isfile(fileName):
        print ("The file path does not exist")
        exit()
    head,onlyFileName = os.path.split(fileName)


    #-----
    #Importing the appropriate reader
    #--------
    neutralFormList = ['txt','npf','sim']
    if onlyFileName[-3:].lower() in neutralFormList:
        onlyFileName = onlyFileName[:-4]
        import LayerObjectReader as myGcode

    elif onlyFileName[-5:].lower() =='gcode' or gCodeFileName[-2:].lower()
==='nc':
        if onlyFileName[-5:].lower() =='gcode':
            onlyFileName = onlyFileName[:-6]
        else:
            onlyFileName = onlyFileName[:-3]
        import GcodeReader as myGcode

    constWidth= 0.4
    Layers = myGcode.CreateLayers(fileName,constWidth)


    #-----------------
    #Writing the Event Series data
    #-----------------

    PowerFile = onlyFileName+"_EventSeries_Power.inp"
    RollerFile =onlyFileName+"_EventSeries_Roller.inp"
    f1=open(PowerFile,'w')
    f2= open(RollerFile,'w')
    tPrev=0.0

    #-----
    #Getting the yMin.yMax for all the layers
    #-------



    ymin,ymax=None,None
    for layer in Layers:
        if len(layer.pathPoints)==0:
            continue

        for path in layer.pathPoints:
            y = path.y
            ymin = y
            ymax=y
            break

    try:
        v1= float(ymin)
        v2 =float(ymax)
    except:
        raise ValueError("The minimum y value of the layers could not be
determined the plugin will now exit")
        sys.exit()



    for layer in Layers:
        if len(layer.pathPoints)==0:
            continue

        for path in layer.pathPoints:
            y = path.y

            if ymin > y:
                ymin=y
            if ymax<y:
                ymax=y

    range = ymax- ymin
    ymin -= rollerPad*range
    ymax+= rollerPad*range

    modifiedlayerStartTime = 0
    prevLayerEndTime = 0


    #Modifying time for user delay
    for layer in Layers:
        if len(layer.pathPoints)==0:
            continue

        modifiedlayerStartTime=prevLayerEndTime+userDelay
        originalLayerStartTime = 0.0+layer.pathPoints[0].t

        layer.pathPoints[0].t = modifiedlayerStartTime

        cnt=0
        lPathPoints = layer.pathPoints
        for path in lPathPoints[1:]:
            cnt+=1
            currTime = 0.0+path.t
            newTime = modifiedlayerStartTime+currTime-
originalLayerStartTime
            path.t= newTime
        prevLayerEndTime=lPathPoints[-1].t


    for layer in Layers:

        if len(layer.pathPoints)==0:
            continue

        z=layer.modifiedZpos + 0.5*layer.avgHeight
        xAv= 0.0
        tCurr = layer.pathPoints[0].t
        vecPow = []


        #--------
        #Switching the power by one row
        #----------
        lPathPoints = layer.pathPoints
        for path in lPathPoints[1:]:
            vecPow.append(path.width)
        vecPow.append(0.0)

        #for path in layer.pathPoints:
        pathCnt=-1
        for path in layer.pathPoints:
            pathCnt+=1
            x=  path.x
            y = path.y
            t = path.t
            pow = vecPow[pathCnt] #This is used to store power for .npf
file

            if pow>0 and userPower>0:
                pow = userPower

f1.write(str(t)+","+str(x)+","+str(y)+","+str(z)+","+str(pow)+"\n")

        xAv/=len(layer.pathPoints)


        f2.write(str(tPrev)+","+str(xAv)+","+str(ymin)+","+str(z)+",1\n")
        f2.write(str(tCurr)+","+str(xAv)+","+str(ymax)+","+str(z)+",0\n")

        tPrev = layer.pathPoints[-1].t

    f1.close()
    f2.close()
    #f3.close()

    print ("Successfully generated the Event Series files \n"+PowerFile
+"\n"+RollerFile+"\n")

#main()
```

2. Gcode reader

```
import re
from ReaderSupporClasses import Point as myPoint
from ReaderSupporClasses import StateObject as myState
from ReaderSupporClasses import Layer as myLayer
from ReaderSupporClasses import PathPoint as myPath
import sys,math

tol = 1e-5

###########################################################################
########################################################################
#Support Functions
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

def updateX(inpString):
    reX = re.compile(r" X([-+]?\d+\.?\d*)",re.I)


    match=reX.search(inpString)

    if match:
```

```
        val = match.group(1)
        fval=float(val)
        endPos=match.end() #Gives the end position in the string...
        return ('X',True,fval,endPos)
    else:
        return ('X',False,None,None)


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def updateY(inpString):
    #reY = re.compile(r" Y([-+]?\d+\.?\d*?)[ /b]+",re.I|re.M)
    reY = re.compile(r"Y([-+]?\d+\.?\d*)",re.I)

    match=reY.search(inpString)

    if match:
        val = match.group(1)
        fval=float(val)
        endPos=match.end() #Gives the end position in the string...
        return ('Y',True,fval,endPos)
    else:
        return ('Y',False,None,None)



#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def updateZ(inpString):
    reZ = re.compile(r" Z([-+]?\d+\.?\d*)",re.I)

    match=reZ.search(inpString)

    if match:
        val = match.group(1)
        fval=float(val)
        endPos=match.end() #Gives the end position in the string...
        return ('Z',True,fval,endPos)
    else:
        return ('Z',False,None,None)


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def updateF(inpString):

    reF = re.compile(r" F([-+]?\d+\.?\d*)",re.I)
    reW = re.compile(r" W([-+]?\d+\.?\d*)",re.I)


    match=reF.search(inpString)

    if not match:
        return ('F',False,None,None)

    tableMatch = reW.search(inpString) #if F and W comes together then the
F represents the motion of the table..
    if tableMatch:
        return ('F',False,None,None)              #which is ignored..


    val = match.group(1)
    fval=float(val)
    endPos=match.end() #Gives the end position in the string...
```

```
    return ('F',True,fval,endPos)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def updateE(inpString):
    reE = re.compile(r" E([-+]?\d+\.?\d*)",re.I)

    match=reE.search(inpString)

    if match:
        val = match.group(1)
        fval=float(val)
        endPos=match.end() #Gives the end position in the string...
        return ('E',True,fval,endPos)
    else:
        return ('E',False,None,None)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def updatePumpSwitch(inpString):

    rePumpOn = re.compile(r"M101\b",re.I)
    rePumpOn2 = re.compile(r"M102\b",re.I) #This paramter has to be
understoood For time being (4/19/2015) this is assumed to be turning the
pump on....
    rePumpOff = re.compile(r"M103\b",re.I)

    matchOn1 = rePumpOn.search(inpString)
    matchOn2 = rePumpOn.search(inpString)
    matchOff = rePumpOff.search(inpString)

    #pumpSwitch = existingPumpState
    if matchOn1 or matchOn2:
        pumpSwitch = True
    elif matchOff:
        pumpSwitch = False

    #didItSwitch = existingPumpState ^ pumpSwitch
    return ('PS',True,pumpSwitch,0) #Improve

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def updatePumpSwitchNL(inpString):
    #This code is for the national lab... they work with M62 and m64 for
pumps and not m101 and m103 (which are extruders)

    #This function takes in both the existing pumpState checks whether the
pump has been turned on or off...
    #This is added to fix the discrepancy in the ORNL path.. in which the
pump still oozes out material even after it is turned off..
    #This lead to some materials not being activated...

    #The idea now is to understand whether the pump is switched on or off
..which is done using an exclusive or..
    #Once it is determined then if the pump is turned on .. then it is
updated otherwise
    #The pump is only turned off when the next F is encountered..
    #This is a fix ..entirely specific to the ORNL data...

    rePumpOn = re.compile(r"\(Turn Pump ON\)",re.I)
    rePumpOff = re.compile(r"\(Turn Pump OFF\)",re.I)

    match1 = rePumpOn.search(inpString)
    match2 = rePumpOff.search(inpString)
```

```
    #pumpSwitch = existingPumpState
    if match1:
        pumpSwitch = True
    elif match2:
        pumpSwitch = False

    #didItSwitch = existingPumpState ^ pumpSwitch
    return ('PS',True,pumpSwitch,0) #Improve

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def getDistanceCoords(x1,y1,x2,y2):
    d = (x1-x2)**2 + (y1-y2)**2
    dist = d**.5
    return dist
#main()

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def getDistance(P1,P2):
    d = (P1.x-P2.x)**2 + (P1.y-P2.y)**2 + (P1.z-P2.z)**2
    dist = d**.5
    return dist
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def updateIdleTime(inpString):
    #This function matches the P values ...which represents the dead
state...
    reDeadStP = re.compile(r" P(\d+\.?\d*?) ",re.I|re.M)
    reDeadStS = re.compile(r" S(\d+\.?\d*?) ",re.I|re.M)

    matchP=reDeadStP.search(inpString)
    matchS=reDeadStS.search(inpString)

    if matchP:
        val = matchP.group(1)
        fval=float(val) #this value represents the time in milliSeconds..
        fval /= 1000.0
        endPos=matchP.end() #Gives the end position in the string...
        return ('IT',True,fval,endPos)
    elif matchS:
        val = matchS.group(1)
        fval=float(val) #this value represents the time in seconds..
        endPos=matchS.end() #Gives the end position in the string...
        return ('IT',True,fval,endPos)
    else:
        return ('IT',False,None,None)


###########################################################################
########################################################################
# Main Function
###########################################################################
########################################################################

def CreateLayers(fileName,constWidth):


    print("Starting GCode Translator")

    if not fileName:
        return None
```

```
    #------------------------
    #Input data
    #------------------------
    f=open(fileName,'r')
    #f2=open("MotionStates.txt",'w')
    GcodeTxt= f.read()

    #------------------------------------------------
    #Creting the GCommand dictionary
    #The reader is the main dictionary with the information on what to to
do after reading the first parameter....
    #The key will be the first field like G0,G01 or G11 .. which will be
pointing to the appropriate read function...
    #------------------------------------------------
    reader={}
    reader['G01'] =  [updateX,updateY,updateZ,updateF,updateE]
    reader['G1']  =  [updateX,updateY,updateZ,updateF,updateE]
    reader['G0']  =  [updateX,updateY,updateZ,updateF,updateE]
    reader['M62'] =  [updatePumpSwitchNL,updateIdleTime] #This is for the
pump to turn on... ORNL thing..
    reader['M63'] =  [updatePumpSwitchNL,updateIdleTime] #This is for the
pump to turn off... ORNL thing..
    reader['M101'] = [updatePumpSwitch,updateIdleTime]
    reader['M103'] = [updatePumpSwitch,updateIdleTime]
    reader['G4'] = [updateIdleTime]


    #------
    #Globals
    #------
    Layers=[] #list of layers



    #-----------------------------------------------------------------------
-
    #Getting the first point from the GCode data....
        #The code will be starting to look for the line with the first
material deposit.
        #The idea is to match code 64 or 101 and then look for the last
state in the strings before the first match..
        #This will get the last point before the machine starts depositing
material .. this is deemed as the first point before the material
deposition starts
    #-----------------------------------------------------------------------
-

    repumpStart  = re.compile(r"M101\b.*?\n",re.I|re.M|re.DOTALL)
    rePumpSt2 = re.compile(r"M64\b",re.I)
    pumpStMatch = repumpStart.search(GcodeTxt)
    pumpStMatch2=  rePumpSt2.search(GcodeTxt)


    if pumpStMatch: #Depending on various machines, the search code could
be expanded
        pumpStindex =  pumpStMatch.start()
    elif pumpStMatch2:
        pumpStindex =  pumpStMatch2.start()
    else:
        raise ValueError,"The Gcode data is not found as expected when
looking for the first instance for the extruder/pump start"
```

```
    x=None
    y=None
    z=None
    f=None

    MachineReadiness = GcodeTxt[:pumpStindex]
    reXval = re.compile(r" X([-+]?\d+\.?\d*)",re.I|re.M)
    reYval = re.compile(r" Y([-+]?\d+\.?\d*)",re.I|re.M)
    reZval = re.compile(r" Z([-+]?\d+\.?\d*)",re.I|re.M)
    reFval = re.compile(r" F([-+]?\d+\.?\d*)",re.I|re.M)
    reEval = re.compile(r" E([-+]?\d+\.?\d*)",re.I|re.M)
    ReValPos=[reXval,reYval,reZval]
    ReValRate = [reFval,reEval]
    firstPoint =[]

    for res in ReValPos:
        matches = res.findall(MachineReadiness)
        if matches:
            val = matches[-1]
            fval=float(val)
            firstPoint.append(fval)
        else:
            raise ValueError,"The Translator could not determine the x,y,z
position before the Extruder was turned on"

    x,y,z=firstPoint[0],firstPoint[1],firstPoint[2]

    for res in ReValRate:
        matches = res.findall(MachineReadiness)
        if matches:
            val = matches[-1]
            fval=float(val)
            firstPoint.append(fval)
        else:
            firstPoint.append(0.0) #In case initial F and E are missing it
is set to 0

    #startPoint = myPoint(x,y,z)
    startSpeed = firstPoint[3]
    startPumpState = False
    startExtrudeRate = firstPoint[4]
    startIdleTime=0.0
    startAcceleration = False
    startState =
myState(startIdleTime,x,y,z,startSpeed,startAcceleration,startPumpState,sta
rtExtrudeRate)


    #--------------------------------------------------------------
    #Finding the Units....
        #G20 - inches
        #G21 - mm
    #--------------------------------------------------------------
    ReUnit = re.compile(r"^G20 ",re.I|re.M)
    UnitMatch = re.search(ReUnit,MachineReadiness)
    if UnitMatch:
        print "\tUnits are in Inches"
        unitScale = 25.4
    else:
        print "\tUnits are in mm"
        unitScale = 1.0 # Not sure what to do with these now..



    #----------------------------------------
    #Finding the filament diameter..
        #Determining the cross section of the filament.
    #----------------------------------------
    if constWidth <=0.0:
        reFilDia = re.compile(r'Filament_Diameter.*: (\d*\.\d*)',re.I|re.M)
        fildiaMatch  = reFilDia.search(GcodeTxt)
        if fildiaMatch:
            filDia = fildiaMatch.group(1)
            filDia = float(filDia)
            print "\tthe filament dia was found to be  = ",filDia
        else:
            raise ValueError," The Gcode reader could not find the filament
diameter\nAdd a comment of the form 'Filament_Diameter_(mm): 1.82' to the
gCode file or use the constant width option"
        filXsection = 0.25*filDia*filDia*(math.pi)



    #------------------------------------
    #Geting the rest of the Gcode block..
    #------------------------------------

    ActiveGcodeTxt = GcodeTxt[pumpStindex:]
    linesinGcode = ActiveGcodeTxt.splitlines()
    motionStates = []
    motionStates.append(startState)


    # This is going to grasp the first line parameter...
    StartcodeRe=re.compile(r'(^\w*)')
    import copy

    for line in linesinGcode:

        line+="\n" #this is to match the end of line ..
        GcodeMatch = StartcodeRe.search(line)

        if GcodeMatch:
            command = GcodeMatch.group(1)
            command =command.upper()

        if not reader.has_key(command):
            continue

        FuncTocheck = reader[command] #this will give a list of function to
match for...
        newState=copy.deepcopy(motionStates[-1]) #Only when the Gcode
matches and the state has to be updated ..then the copy operation is done..

        # Idletime should be reverted back to 0...
        newState.setIdleTime(0.0)

        #This is still expensive.. should think about something better ...
#Improve..
```

```
        paramPos={} #This is a dictionary of parameter and position and is
needed for determining acceleration.. Might have to be changed later..
        for funcs in FuncTocheck:

            #Fupdate = False
            #layer=myLayer(cnt)
            param,isNew,newVal,pos = funcs(line)

            #Now to get the appropriate update function from the
motionState object
            if isNew :
                updateFunc = newState.updateParam(param)
            else:
                continue

            if param=='E'and constWidth<=0.0:
                newVal*=filXsection


            updateFunc(newVal)
            paramPos[param]=pos


        if paramPos.has_key('F'):
            fpos= paramPos['F']
            newState.setAcceleration(False)
            keyList = paramPos.keys()
            keyList.remove('F')

            for ps in keyList:
                position = paramPos[ps]

                if position < fpos:
                    newState.setAcceleration(True)
                    break
        motionStates.append(newState)


    #Now to get the material deposition width for each pass..
    motionStates[0].width = 0.0
    motionStates[0].time = 0.0
    cumTime = 0.0

    #layerHeight = 1.0
    layerHeight = 1.0
    widthScaleFactor=1.0

    for i in range(1,len(motionStates)):
        preState = motionStates[i-1]
        currState = motionStates[i]
        x1,y1 = currState.x,currState.y
        x2,y2 = preState.x,preState.y
        dist = getDistanceCoords(x1,y1,x2,y2)
        matDeposited =  currState.getE() - preState.getE()


        if dist and constWidth<=0.0:
            width = widthScaleFactor*matDeposited /(dist*layerHeight)
            currState.width = width
        elif dist and constWidth>0.0:
            width = constWidth
            currState.width = width
        elif not dist:
            currState.width = 0.0

        #Time Calculations..
        DeadState = currState.getIdleTime()
        Accel = currState.getAcceleration()

        if DeadState: #Dead state ..then add the necessary time
            cumTime +=DeadState
            currState.setTime(cumTime)
            continue

        if dist<=tol:
            prevTime = preState.getTime()
            currState.setTime(prevTime)
            continue

        v=currState.getF()
        u=preState.getF()

        if Accel: #If acceleration found .. add the time for acceleration

            if abs(v - u) > tol:
                a = (v**2 - u**2)/(2*dist)
                t = (v - u)/a
                t*=60 #to change it to s
                cumTime+=t
                currState.setTime(cumTime)
            else: #constant velocity motion...
                t= dist/v #assuming that v is always going to be there ..
                t*=60 #To get time in
                cumTime+=t
                currState.setTime(cumTime)

        else:  #constant velocity motion..
            t= dist/v #assuming that v is always going to be there ..
            t*=60 #To get time in
            cumTime+=t
            currState.setTime(cumTime)

    #for mStates in motionStates:
    #    f2.write(str(mStates))
    #f2.close
    #----------------------------------------------------------------------
------------------------------------------------------------------------
    #The assumption is that when E changes that is where the model starts
...All other points are ignored...
    #Creating the layer object...
    Layers=[]

    heightNew=None
    bNewLayer = False


    #Creating the layers
    cnt=0
    totHeight=0.0
    for mStates in motionStates:
       MatWidth = mStates.width
       height = mStates.z
       if cnt==0 and MatWidth>tol:
           newLayer = myLayer(0)
           newLayer.zPos = height
           Layers.append(newLayer)
           cnt+=1
           continue

       elif cnt>0 :
           prevLayerHeight = Layers[cnt-1].zPos

           if prevLayerHeight < height and MatWidth>tol:
               newLayer = myLayer(cnt)
               newLayer.zPos = height
               newLayer.avgHeight = height-prevLayerHeight
               newLayer.modifiedZpos = height - 0.5*(height-
prevLayerHeight)
               totHeight+= newLayer.avgHeight
               Layers.append(newLayer)
               cnt+=1
               continue

    avgHeight = totHeight/(cnt-1)

    #Populating the layer with path points..
    for layer in Layers:

        height = layer.zPos
        for i in range(1,len(motionStates)):
            currState = motionStates[i]
            prevState = motionStates[i-1]


            if currState.width>tol and abs(currState.z-height)<tol:
                #Set the first path Point
                lenPaths = len(layer.pathPoints)
                if lenPaths==0:

                    if layer.layerNumber ==0:
                        timeToSubtract = prevState.time

                    X = prevState.x
                    Y = prevState.y
                    T = prevState.time - timeToSubtract
                    layer.setPathPoints(X,Y,T,0.0)


                    XCurr= currState.x
                    YCurr= currState.y
                    TCurr= currState.time - timeToSubtract
                    WCurr= currState.width
                    layer.setPathPoints(XCurr,YCurr,TCurr,WCurr)

                #This is another important bug fix..
                #The issue was that while looking for only paths with
positive width .. the start points were not being recognized
                # These are points which could be determined in two ways
                #Checking if the pumps were turned on or off at that point
.. This method is not good becuase sometimes the pump turns on and does not
move..
```

```
                #The other method is to see if the previous state has a
width =0.0 then the previous point is added to the path..#More
brainstorming needed..
                elif( lenPaths > 0 and prevState.width <tol):

                    X = prevState.x
                    Y = prevState.y
                    T = prevState.time - timeToSubtract
                    W = prevState.width
                    layer.setPathPoints(X,Y,T,W)

                    #Adding data from the current path
                    X = currState.x
                    Y = currState.y
                    T = currState.time-timeToSubtract
                    W = currState.width
                    layer.setPathPoints(X,Y,T,W)


                elif( lenPaths > 0 and prevState.width >tol):
                    X = currState.x
                    Y = currState.y
                    T = currState.time-timeToSubtract
                    W = currState.width
                    layer.setPathPoints(X,Y,T,W)


    #Scaling the width to adjust for layer height..
    #The width for each path was determined by using unit layer height.

    if constWidth<=0.0:
        for layer in Layers:
            for paths in layer.pathPoints:
                currWidthVal = paths.width

                if currWidthVal<tol:
                    paths.width = 0.0

                if currWidthVal > tol:
                    a=math.pi*avgHeight*avgHeight*0.25
                    newWidth = currWidthVal - a
                    newWidth /= avgHeight
                    newWidth = newWidth + avgHeight
                    paths.width = newWidth


    print "\tnumber of layers = ",len(Layers)
    print "\tAverage height of layers = ",avgHeight
    print "Gcode Reader successful"


    #-----------------------------------------------------------------------
-----------------------------------------------------------------------------

    #Before beginning the origin has to be found out
    #And the start time has to be scaled...
    #All this data is obtained from the first Layer..


    #Determining the origin of the model..
    layer0 = Layers[0]
    modelOrigX = layer0.pathPoints[0].x
    modelOrigY = layer0.pathPoints[0].y
    modelOrigZ = layer0.zPos -avgHeight
    layer0.modifiedZpos = modelOrigZ + 0.5* avgHeight

    modelOrigin = myPoint(modelOrigX,modelOrigY,modelOrigZ)

    layer0.setOrigin(modelOrigin)
    layer0.setAvgHeight(avgHeight)

    #Scaling the VAlues to mm...
    #for layer in Layers:
    #    layer.setUnits(unitScale)


    #---------
    #Writing the layer object dump
    #---------
    f2 = open('LayerObjectDump.txt','w')
    for layer in Layers:
        layer.setMinWidth()
        f2.write(str(layer))
    f2.close()


    return Layers
```

3. Layer Object Reader

```
import re
from ReaderSupporClasses import Layer as myLayer
from ReaderSupporClasses import Point as myPoint


def CreateLayers(fileName,constWidth):

    constWidth=None #This width is not used. This argument is added so as
to have consistency between the CreateLayer function in Gcode reader and
Layer object reader


    f=open(fileName,'r')
    sLayer = f.read()
    Layers=[]
    srcStr1 =r"^                  Layer Number ="
    reLayerNums = re.compile(srcStr1,re.I|re.DOTALL|re.M)
    #lsLayers =re.findall(reLayerNums,sLayer)
    lSplitLayers = re.split(reLayerNums,sLayer)

    #----------
    #Getting the layer height
    #---------
    srcStr2 =r"step between layer =(.*)\n"
    reLayerNums = re.compile(srcStr2,re.I|re.M)
    srcStepSize = re.search(reLayerNums,sLayer)
    sStepSize =  srcStepSize.groups()[0]
    sStepSize = sStepSize.strip()
    steplayerHeight = float(sStepSize)
```

```
LzPositions=[]
avgLayerHeight=0.0
cnt=0

for layer in lSplitLayers[1:]:

    #Improvement .. Get the string till path points...

    #Get the layer number...
    srcLayNum = r"^ Layer Number = (\d+)"
    reLayerNum = re.compile(srcLayNum,re.M|re.I)
    reLaySrch = re.search(reLayerNum,layer)
    layNum= int(reLaySrch.groups(1)[0])
    layNum-=1
    #print layNum


    #Get the layer position...
    srcLayPos = r"^ Layer Zpos = ([-+]?\d+\.?\d*)"
    reLayerZPos = re.compile(srcLayPos,re.M|re.I)
    reLayZpos = re.search(reLayerZPos,layer)
    layerZpos= float(reLayZpos.groups(1)[0])
    LzPositions.append(layerZpos)
    #print layerZpos


    #Get the main data block...
    sreBlock = r"power\n(.*?)\n\n"
    reBlock = re.compile(sreBlock,re.I|re.M|re.DOTALL)
    sBlock = re.search(reBlock,layer)
    #dataBlock = sBlock.groups(1)[0]
    #listDataBlocks = dataBlock.split("\n")

    currLayer= myLayer(layNum)
    currLayer.setZpos(layerZpos)
    currLayer.setAvgHeight(steplayerHeight)
    currLayer.setModifiedZpos(layerZpos)

    if sBlock==None:
        Layers.append(currLayer)
        continue
    else:

        dataBlock = sBlock.groups(1)[0]
        listDataBlocks = dataBlock.split("\n")

        for datalines in listDataBlocks:
            path = datalines.split(",")
            try:
                x = float(path[0])
                y = float(path[1])
                t = float(path[2])
                w = float(path[4])
            except:
                break


            #
```

```
                    currLayer.setPathPoints(x,y,t,w)

                try:
                    currLayer.setMinWidth()
                except:
                    print currLayer.layerNumber,"Is the layer number"
                    print currLayer

                Layers.append(currLayer)


        #Determining the layer height and average layer height..

        #f=open("NewLayerObject.txt",'w')
        #for layer in Layers:
        #    f.write(str(layer))
        #f.close()
        return Layers
```

4. Reader Support Classes

```
import copy
class StateObject(object):
    """description of class"""


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def __init__(self,IdleTime=
None,X=None,Y=None,Z=None,Rate=None,Acceleration=None,PumpState=None,Extrud
eRate=None,Time=None):
        self.idleTime= IdleTime
        self.x = X
        self.y = Y
        self.z = Z
        self.rate=Rate #This is the F parameter
        self.acceleration = Acceleration
        self.pumpSwitch=PumpState
        self.extrudeRate = ExtrudeRate #This is the E parameter..
        self.time=Time
        self.width = None

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def printState(self):

        print "\nx = ",self.point.x,"  y = ",self.point.y," z =
",self.point.z," Feed Rate = ",self.rate, "Is pump On = ",self.pumpSwitch,\
            "ExtrudeRate = ",self.extrudeRate


###########################################################################
########
    # GET FUNCTIONS
    #######################


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
```

```
    def getIdleTime(self):
        return self.idleTime


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getX(self):
        return self.x

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getY(self):
        return self.y

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getZ(self):
        return self.z

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getF(self):
        return self.rate

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getAcceleration(self):
        return self.acceleration

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getPumpState(self):
        return copy.deepcopy(self.pumpSwitch)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getE(self):
        return self.extrudeRate

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def getTime(self):
        return self.time



###########################################################################
########
    # SET FUNCTIONS
    #######################
    def setIdleTime(self,IdleTime):
        self.idleTime = IdleTime

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setX(self,xval):
        self.x = xval

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setY(self,yval):
```

```
        self.y = yval

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setZ(self,zval):
        self.z = zval

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setF(self,frate):
        self.rate = frate

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setAcceleration(self,Acceleration):
        self.acceleration = Acceleration

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setPumpState(self,pumpSw):
        self.pumpSwitch = pumpSw

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setE(self,ExtrudeRate):
        self.extrudeRate = ExtrudeRate

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~
    def setTime(self,Time):
        self.time = Time

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~


############################################################################
########
    # SUPPORT FUNCTIONS
    #######################

    #This is only for setting up the x,y,z,f and E paramters .. with the
idea that in future the I,J parameters will be added.
    #This will retrun the appropriate set functions ..and the arguments
will be passed by the client code...
    #The input argument has to be a string of the form shown in the
dictionary key ..
    def updateParam(self,arg):
        paramDict={}
        paramDict['X']=self.setX
        paramDict['Y']=self.setY
        paramDict['Z']=self.setZ
        paramDict['F']=self.setF
        paramDict['E']=self.setE
        paramDict['IT']=self.setIdleTime
        paramDict['PS']=self.setPumpState
        return paramDict[arg]
```

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
    #String representation.
    def __str__(self):

        str2return = "\n"+ "IdleT= "+str(self.idleTime).ljust(10)+ "   ,
"+"X= "+str(self.x).ljust(10)+ "   ,   " +"Y= "+str(self.y).ljust(10)+ "   ,
"+"Z= "+str(self.z).ljust(10) \
                +"   ,   "+"F= "+str(self.rate).ljust(10)+ "   ,   "+"IsA=
"+str(self.acceleration)+"   ,   "+"Pump = "+str(self.pumpSwitch)+"   ,   "+"E=
"+str(self.extrudeRate)+\
                "   ,   "+"W= "+str(self.width)+"   ,   "+"Time=
"+str(self.time)

        return str2return




############################################################################
#################################
#Class 2 -> Layer Class
############################################################################
################################




#from stateObject import StateObject as myState
#from point import Point as myPoint
#from PathPoint import PathPoint as myPath

tol =0.001

class Layer(object):
    """description of class"""
    def __init__(self,LayerNumber):
        self.layerNumber = LayerNumber
        self.origin = None

        self.avgHeight=None
        self.zPos = None
        self.modifiedZpos= None
        self.minWidth = None
        self.pathPoints=[]

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setOrigin(self,point):
        self.origin = point
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    #The modified Zpos is requried to make the tip head travel through the
thickness of the layer instead of being at the top
    #This is done by subtracting the half of the layer height from the
zposition
    def setModifiedZpos(self,Zmodified):
        self.modifiedZpos = Zmodified

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setZpos(self,Z):
        self.zPos = Z
```

```
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setAvgHeight(self,height):
        self.avgHeight = height

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setOrigin(self,point):
        self.origin = point

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def appendState(self,state):
        self.motionStates.append(state)

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def appendPathPoints(self,pathPoint):
        self.pathPoints.append(pathPoint)



    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    #def __repr__(self):
    #    print "\n\n\n---------------------------------------------------------
-----------------"
    #    print "\n                  Layer Number = ", self.layerNumber
    #    print "\n---------------------------------------------------------------
-------------"

    #    print "\n Layer Number = ",self.layerNumber
    #    print "\n\n  x   ,   y   ,  z   ,   F ,   pump\n"
    #    for m in self.motionStates:
    #        print "\n",m.getX,m.getY,m.getZ,m.getF,m.getPumpState

        #        self.origin = None

        #self.avgHeight=None
        #self.zPos = None
        #self.modifiedZpos= None
        #self.pathPoints=[]
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def __str__(self):

        if self.origin:
            origString = "\n Layer Origin = "+ str(self.origin.x)+ " , " +
str(self.origin.y)+ " , " + str(self.origin.z)
        else:
            origString = ''

        str1=\
            "\n\n\n-----------------------------------------------------------
---------------"+\
            "\n                  Layer Number = " + str(self.layerNumber) +
\
            "\n---------------------------------------------------------------
----------"+\
            "\n Layer Number = "+str(self.layerNumber) + origString + \
            "\n Layer Zpos = "+ str(self.zPos) + \
            "\n Layer Modified zPos = "+str(self.modifiedZpos) + \
            "\n Layer height = " +str(self.avgHeight) +\
            "\n Layer Min Width = " +str(self.minWidth) +\
            "\n\n Path Points are ...... \n"+\
            "\nx                   y            Time               width"
```

```
        str2=''
        for paths in self.pathPoints:
            str2+="\n"+str(paths.x).ljust(10)+ "   ,
"+str(paths.y).ljust(10)+ "   ,   " +str(paths.t).ljust(10)+"   ,   "
+str(paths.width).ljust(10)
        strRep = str1+str2

        return strRep


    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setZPosFromLayer(self):

        if len(self.motionStates)==0:
            print 'The motion States are empty and Z position cannot be
set'
            return

        for motionStates in self.motionStates:
            if motionStates.pumpSwitch:
                zPos = motionStates.getZ()
                self.setZpos(zPos)
                break

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setPathPoints(self,X,Y,T,Width):

        #The idea of Time Modifier is to subtract a certain value from the
time..
        #normally the start time is not when the machine starts but a later
value after the machine checks for readines..
        #This time is there in the motionStates object which is not useful
from an analysis standPoint..
        x = X
        y=  Y
        t= T
        w = Width
        myPathPt = PathPoint(x,y,t,w)
        self.appendPathPoints(myPathPt)




    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # GET Functions ...........
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def getZpos(self):
        return float(self.zPos)




    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Support Functions ...........
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    def setUnits(self,scaleFactor):
```

```
        if self.origin:
            self.origin.scale(scaleFactor)
        self.avgHeight *=scaleFactor
        self.zPos *=scaleFactor
        self.modifiedZpos *=scaleFactor

        for pathPoints in self.pathPoints:
            pathPoints.x *=scaleFactor
            pathPoints.y *=scaleFactor

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setMinWidth(self): #This functions sets the minimum width for the
layer.. this has to be used by the Translator code for Wgrid creation..
        wmin = self.pathPoints[1].width #the first path point width is 0..
assuming the second path point width as minimum..
        if wmin<tol:
            raise ValueError, "The minimum width cannot be 0....."

        for i in range(1,len(self.pathPoints)):
            w = self.pathPoints[i].width
            if wmin > w and w>tol:
                wmin =w

        if wmin<tol:
            raise ValueError, "The minimum width cannot be 0....."

        self.minWidth = wmin
        pass

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setWidthScale(self,scaleFactor):   #This is to scale the width of
the path..

        if scaleFactor<=0.0:
            raise ValueError, "The scale factor for width should be
positivie"
            return None
        for paths in self.pathPoints:
            paths.width*=scaleFactor

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def setTimeScale(self,scaleFactor):  #This is to scale the time factor
of the path...

        if scaleFactor<=0.0:
            raise ValueError, "The scale factor for width should be
positivie"
            return None
        for paths in self.pathPoints:
            paths.t*=scaleFactor


###########################################################################
#################################
#Class 3 -> PathPoint Class
###########################################################################
################################
class PathPoint(object):
    """description of class"""
```

```
    def __init__(self,X,Y,T,W):
        self.x = X
        self.y = Y
        self.t = T
        self.width = W




###########################################################################
#################################
#Class 4 -> Point Class
###########################################################################
################################

class Point(object):
    """description of class"""

    def __init__(self,X,Y,Z):
        self.x = X
        self.y = Y
        self.z = Z

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def scale(self,Scale):
        self.x *= Scale
        self.y *= Scale
        self.z *= Scale
```

Miguel Cortés Otero

NTNU
Norwegian University of
Science and Technology