Degree Project for Master of Science with specialization in Robotics

Department of Engineering Science

University West

Enhancing Efficiency in Metal 3D Printing Through Heat Simulation and AI



Jean Justine

Optional figure, max 8,0 · 16,0 cm  
**If not used, the frame must be removed!**

A thesis submitted to the Department of Engineering Science

in partial fulfilment of the requirements for the degree of

Master of Science with specialization in Robotics

at University West

2025

**Date**: 14/05/2024

**Author**: Jean Justine

**Examiner**: Fredrik Sikström

**Advisor**: Xiaoxiao Zhang, Högskolan Väst

**Programme**: Master in AI and Automation

**Main field of study**: AI with a specialization in Automation

**Credits**: 60 Higher Education credits (see the course syllabus)

**Keywords:** Automation, 3D printing, Simulation, Metal, AI

**Template:** University West, IV-Master

**Publisher**: University West, Department of Engineering Science  
S-461 86 Trollhättan, SWEDEN  
Phone: + 46 520 22 30 00 Fax: + 46 520 22 32 99 Web: www.hv.se

Summary

Printing complex and unique form is increasing in demand in industry, to answer it, metal 3D printing is a good alternative, it can print those pieces without having to create a new production chain and facilitate the access those. But as a growing demand and pretty new method, there is improvement to be done, it’s in this context, that the thesis take place, as researching how to improve the efficiency through heat simulation and AI. Before presenting the results, here are the major steps to achieve that. Firstly, creating a heat simulation using a voxel representation and a heat equation to fit each piece’s geometry. Using Robot studio to simulate the printing, we have all the keys in hand to start the AI except data, to counter this problem we use reinforcement learning that doesn’t require much data to be train.

STILL IN TRAINING SO NO RESULTS FINISH THIS LATER

Affirmation

This master degree report, Enhancing Efficiency in Metal 3D Printing Through Heat Simulation and AI, was written as part of the master degree work needed to obtain a Master of Science with specialization in Robotics degree at University West. All material in this report, that is not my own, is clearly identified and used in an appropriate and correct way. The main part of the work included in this degree project has not previously been published or used for obtaining another degree.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Signature by the author Date

Jean Justine

Contents

**Preface**

Aucune entrée de table des matières n'a été trouvée.

**Main Chapters**

[1 Introduction 1](#_Toc61453532)

[1.1 No heading here 1](#_Toc61453533)

[1.1.1 Too deep structure 1](#_Toc61453534)

[1.2 Aim 1](#_Toc61453535)

[1.3 Template 1](#_Toc61453536)

[1.4 Figures 2](#_Toc61453537)

[1.5 Language 3](#_Toc61453538)

[1.6 Equations 4](#_Toc61453539)

[1.7 Program code 4](#_Toc61453540)

[1.8 Tables 5](#_Toc61453541)

[1.9 Cross-references 6](#_Toc61453542)

[1.10 Review 6](#_Toc61453543)

[1.11 Embedded objects 7](#_Toc61453544)

[2 Related work (Background) 8](#_Toc61453545)

[3 Method 9](#_Toc61453546)

[4 “Work” 10](#_Toc61453547)

[5 Results and discussion 11](#_Toc61453548)

[6 Conclusion 12](#_Toc61453549)

[6.1 Future Work and Research 12](#_Toc61453550)

[6.2 Critical Discussion 12](#_Toc61453551)

[6.3 Generalization of the result 12](#_Toc61453552)

[7 References 13](#_Toc61453553)

**Appendices**

[A. Appendix](#_Toc61453554)

[B. Appendix](#_Toc61453555)

Symbols and glossary

DED Direct Energy Deposition, a method of metal 3D printing it uses a laser to melt a metal wire into the shape requested.

WAAM Wire Arc Additive Manufacturing, it is a method of metal 3D printing that is used during this thesis, it works with a robot arm that has a metal wire and is weld using an arc onto a pieces that is being printed. It is a branch of DED

TCP Tool Center Point represent the tip of the tool where it is used in Robot Studio.

ODE Ordinary Differential Equation, this is an equation relating a function to its derivatives.

AI Artificial Intelligence

ML Machine learning is a branch of AI in which model are created and trained on data.

RL Reinforcement Learning, a Machine Learning branch in which the model learns with a system of reward and penalty to optimize what is it trained for.

Introduction

The metal additive manufacturing, has rose as a transformative technology in multiple domain such as medical, aerospace, and others, it offers the ability to produce complex and unique parts, that could be hard or impossible to produce with conventional methods and machine. Those demands continue to rise as company seek better performance and easier access.

It’s in this context that, metal 3D printing takes places, allowing manufacturers to create complex part easier and quicker without having to create a production chain. However, metal 3D printing faces multiple challenges such as not efficient build sequences, the heat propagation management is lacking, moreover, each new geometry require a new approach, new parameters tuning, resulting in trial and error cycles, using both time and hardware.

In metal additive manufacturing, as well as metal 3D printing, Thermal Management is crucial, the intense, localized heating energy from the source, creates very various thermal gradient within the piece. If these gradient are not controlled, the piece can be ruined, distorted, not smooth, and other problems. Solving this problem is key, to do so, simulating the heat and printing of the pieces as accurately as possible is mandatory, especially pieces that are more complex than others and will have a different temperature and cooling time.

AI, is a way to reduce waiting time, and learn quickly to enhance the printing path, and which pieces to print first. By doing so it can reduce the total time to print pieces, and maximize the quality and minimize the time spend.

Aim

To address these challenges, this thesis aims to develop a comprehensive simulation environment that support AI driven optimization of metal 3D printing and a complete simulation of the heat of every pieces that are printed as well as their pattern and geometry feature.

The objectives are:

* Develop and validate a physic-based model, to reproduce the real metal printing process and ensure the realism of it.
* Create an accurate representation of different pieces printed.
* Simulate the thermal environment of those printed pieces.
* Implement and evaluate algorithms, such as reinforcement learning to autonomously optimize build sequence.

Related work (Background)

In metal 3D printing, there is different methods, such as Laser Powder Bed Fusion, that cover multiple sub methods like selective laser printing, that deposit a layer of powder and use a laser to imprint on those layers the pieces they want [1].

The method that is worked on during this thesis is a branch to the one called Direct Energy Deposition, that uses a metal wire or powder and melt it using an energy source such as a laser [2].

During this thesis, the method to print is, called WAAM, Wire Arc Additive Manufacturing, a wire going through an arm robot from ABB studio, and an electrical arc that melt the wire directly onto the plate, then build layer per layer the piece requested [3].

All those methods differ in terms of capability, energy, build rates and the possibility of geometry to print.

The one used, WAAM, has multiple advantage compare to others way of printing, it can print large pieces, as it is only limited by the size of the robot. It uses common welding wires that are easy to find, already worked on for a long time so it became nearly optimum. Also it can print a lot of material quicker than others DED [3].

Welding Process Fundamentals

Arc-based WAAM, process derive directly from gas-metal arc welding combine with additive manufacturing, it is produce by a robot, that deposit a layer of metal on top of each other until the pieces wanted is ready.

To simulate the moving heat source, WAAM models used mostly: double ellipsoid Goldak equivalent heat source. In a coordinate system moving with a torch or arc welding, the front and read volumetric power densities are given with:

𝑞𝑓 (𝑥, 𝑦, 𝑧,𝑡) = (6√3𝑓𝑓𝑄/ 𝑎𝑓𝑏𝑐𝜋) e𝑥𝑝 (−3 (𝑥+𝑣(𝜏−𝑡))^ 2/ 𝑎𝑓^ 2 ) 𝑒𝑥𝑝 (−3 𝑦^2/ 𝑏^2 ) 𝑒𝑥𝑝 (−3 𝑧 2 𝑐 2 )

𝑞𝑟 (𝑥, 𝑦, 𝑧,𝑡) = (6√3𝑓𝑟𝑄/ 𝑎𝑟𝑏𝑐𝜋) 𝑒𝑥𝑝 (−3 (𝑥+𝑣(𝜏−𝑡))^2/ 𝑎𝑟^2 ) 𝑒𝑥𝑝 (−3 𝑦^2 /𝑏^2 ) 𝑒𝑥𝑝 (−3 𝑧^2/𝑐^2 )

𝑓𝑓 + 𝑓𝑟 = 2

2𝑄 = (𝑞𝑓 (𝑥, 𝑦, 𝑧,𝑡) + 𝑞𝑟 (𝑥, 𝑦, 𝑧,𝑡)) 𝑑𝑥𝑑𝑦𝑑z

𝑄 = 𝐻𝑒𝑎𝑡,[W]

Where Q is arc power, v the torch speed, (af/r​,b,c) the ellipsoid semi-axes and ff/ fr​ the front/rear energy fractions [4].

Thermal Behavior in WAAM

Using WAAM involve a thermal reaction, it is governed by the transient heat conduction equation with a moving sources:

ρCp​∂t∂T​=∇⋅(k∇T)+q(x,y,z,t) ,

where ρ is the density, Cp​ the heat capacity, k the thermal conductivity and q the spatially-distributed Goldak source term [5].

However, during this thesis, the heat equation that is adopted is a simpler format, a lump parameter thermal model to track the temperature per layer of the form,

dT /dt​​=α​(n​)Q(t)−β​(n​)[T1​(t)−Tamb​]−γ​(n​)[T^4​(t)−Tamb^4​].

Where:

* T1(t) is the mean temperature of the deposited material at time t,
* Q(t) is the instantaneous arc power input (voltage × current),
* Tamb is the ambient temperature,
* α1(n1) maps power input to temperature rise and may vary with layer index n1,
* β1(n1) is the coefficient for linear heat losses (convection/radiation),
* γ1(n1) captures nonlinear (fourth-power) radiative losses.

Robotic Simulation in WAAM

High-fidelity reproduction of the WAAM kinematics has been achieved through offline programming in ABB RobotStudio. Tools and pieces are as reality, to achieve a perfect model that allows reproduction of the 3D metal printing machine as close to reality as possible. In the optic to work in an environment that saves times, resources, allow errors and a nearly full control over it [6].

Voxel-Based Geometric for 3D printing reproduction

3D printing reproduction under a voxel-based geometry is a method that allows to recreate any shape wanted. It is used in 3D printing to have high quality, with full resolution and allow to reproduce colour and achieve a high fidelity. This allow a numerous form of application notably to simulate the heat, or to have the same quality as the pieces printed [7] and [8].

Voxel-Based Geometric Discretization for Thermal Mapping

Thermal mapping allows a lot of computation to simulate the heat correctly. When coordinate is imported, it’s first discretised into a uniform 3D structure of cubic voxels. Each of those voxel, are unique and computed, it allows to have a full rendering of different pieces in 2D or 3D. With that making calculation on every different elements or part become doable. Thermal mapping is then realized by implementing different heat equation required, and the output is the final pieces or object with a full heat map [9].

Reinforcement Learning in Additive Manufacturing

Reinforcement Learning (RL) has emerged those recent years to become a powerful tool for sequential decision-making problems in which an optimization of a set of action to maximize long term reward is requested. RL enables adaptive, closed-loop control, time-varying process.

In layer-by-layer methods such as DED, or WAAM, keys parameters such as print speed, extrusion temperature, piece temperature, layer height… strongly influence quality. RL models are, mostly on simulation trained to change and optimize those key parameters to achieve good results without having to go through mathematical models [10].

RL has also been used to optimized path planning for different pieces. Choosing a path where the high dimensional space of trajectories is complex, it is often find using trials and errors all in simulation. Model learn how to generate path that minimize errors, movement, layer misalignment as well as thermal distortion. Also in some work, when the RL is used on the real machine and take the decision, thermal camera help find errors, and the model correct them in real time [10] and [11].

Method

For this thesis, multiple methods were used to complete it. Going from simulation on ABB Robot Studio to Voxel Representation to finally create a heat simulation that sticks to reality.

All the following steps are in chronological order; some previous steps were modified later on if something needed to be changed or did not work with the previous one.

Robot Studio Simulation

Robot Studio (v2024) was selected because it provides an industry-standard digital twin of ABB manipulator and a well document Python API. Also It allows a multitude range of tool as well as a complete control over the environment like the speed, the pieces to print, the code and variable.

To achieve that Robot Studio was used to simulate the metal 3D printing machine using the same speed and paths that represent the pieces. Multiple variables are created to allow an easy access to the coordinate of the tool center point (TCP), if the machine is Welding or not, the number of layer per pieces, which pieces is being print, and another one to allow the machine to print or not.

Fetching the coordinate

From the code in python, multiple functions are created that access those variable, or set a variable. For instance, a function called “fetch\_xyz” allows the importation of the coordinate every millisecond. On top of that, there is also a verification that a pieces is being printed, and that the arm is welding. If it is finished, fetching is stopped and a variable is set that stop the Rapid code in Robot Studio until it is allowed to run again. During all this fetching time, a json file is append with every coordinate previously collected.

When the fetching time is done and the json is available, all the small outlier that could have been fetch by mistake are sorted out.

Voxel Representation

The following steps convert raw TCP logs into per-piece voxel models. It consists of multiple steps to allow a very fitting representation of the different pieces that are fetch, each in their single Voxel Representation in their json.gz file. This method was chosen over mesh or point-cloud formats due to its advantage such as saving space, accuracy, it is also wildly used, allow the representation of 3D geometrical form and a direct compatibility with the heat equation that will be used.

Mapping deposition points to Voxel Indices

Firstly, the TCP brute coordinate (‘x’, ‘y’, ‘z’) are converted into integer to voxel indices (‘ix’, ‘iy’, ‘iz’) by first normalizing each axis to a 0,1 range based on the observed minimum and maximum. So they correspond to the first grid position and the last. This places each point proportionally within the grid. Then those value are rounded to the nearest integer. The same principle is applied for each vertical coordinate, but, if a fixed layer thickness is specified, each vertical coordinate is divided by this thickness to assign it to a particular layer index

Local 2D Filling of Each Deposition Point

Each mapped voxel (ix, iy, iz) is expanded into a small square footprint in its Z slice to represent the toolhead diameters and the surface printed. A radius defines a neighbourhood of offsets (dx,dy), for each offset within the radius, the corresponding voxel grid cell at (ix+dx, iy+dy) is filled. It is to ensure that small gaps between sampled point are connected and close among the same layer.

Vertical Smoothing Across Layers

To reduce layer to layer discretization noise, a sliding window of adjacent Z slices, with a default window of 3, is checking each slice index z. The sum of binary values across the window is compared to half the window depth, if the local sum meet or exceeds that threshold, the voxel at (x,y,z) is also filled. This majority voting system ensure the vertical consistency across all layers.

Extraction of Per-Layer Bounding Boxes

Connected region in the binary voxel grid are first identified using a 3x3x3 structure to assign each cluster a label. For each label, all voxel coordinates are grouped by their Z-slice, and the minimum and maximum X, Y indices in each slice define that layer’s 2D bounding box.

Then, a nested dictionary mapping each component label to its per-slice "bounding\_box" and list of "active\_pixels", is then written as a gzipped json file, named after the pieces the voxel is from. This compact layer representation allows saves of space without a drop in quality of the representation.

Cooling time

Cooling intervals were computed for each pieces by comapring timestamps at the end of each layer. An internal dictionary that maps each pieces to its cool time was created and enable on demand retrieval of elapsed cooling time.

When printing of a layer begins, the current time is fetched and compare to the stored end time for that piece. The difference defines the cooling time since the last build of the piece, this value is return and the last end timestamp is reset.

If a piece’s layer is complete the end function record the current timestamps in the dictionary and this timestamps become the reference for all future cooling time calculation for that piece.

All timing is based on python’s perf counter allowing a very high precision.

Heat simulation

The voxelized geometry done previously can now be subjected to a transient heat transfer simulation that solves ordinary differential equation for each active voxel over successive layers. This procedure in implements in multiple steps

Data Loading and initialization

The gzipped json file produce in section 3.3 is decompressed and parsed to reconstruct the pieces per slice voxels map. A three-dimensional temperature grid of size (nz \* nx \* ny) is initialized to the ambient temperature for every voxel.

Heat-equation Formulation

A local Heat ordinary differential equation (ODE) is defined for each active voxel:

dT /dt​​=α​Q−β​[T1​−Tamb​]−γ​[T^4​−Tamb^4​].

Where:

* Q is the constant heat input per step.
* α β γ are empirical parameters that depend on the local geometry of the pieces as well as its cool time and the number of layers.
* T is the current voxel temperature.
* Tamb is the ambient temperature.

Geometry Analysis

A geometry analysis of the different pieces is conducted to obtain the compactness, filled area, maximum internal gap, number of wall, the average thickness of wall as well as the bounding box area.

In-Slice Neighbourhood Averaging

To approximate lateral heat conduction (among the layers), each temperature is blended with the mean temperature of its eight adjacent neighbours on the same Z slice. A weighted average is then used as the temperature in the ODE.

### Layer-Wise Time Integration

The simulation advance layer per layer, for each of them, a specified number of substep is computed, with a time increment dt, in addition to the cooling time that influences the ODE equation. During each substep, all active voxels in the current layer are updated at once by adding dt x ODE to their temperature. The updated grid with new temperatures serves then for the initial condition for the next layer.

Saving heat information

To save the computed voxel with the new temperature, the heat map is saved under an .npy file and a new internal dictionary called stats is created that stores, the average temperature of the piece’s active voxel, without taking in account the “air”, it’s dimension as well as its cool time and the name of the files where the heat map are saved.

Choosing Path Algorithm

To determine which pieces to print first a simple algorithm is created.

Once all the pieces had one layer printed. A list of the piece’s id is created, and an algorithm go through the stats dictionary previously created and selected the pieces with the lowest average temperature to be printed, if this temperature is below the threshold where it is allowing to print on, it print it and its number of layer is incremented by one, if it’s not allowed, it’s waiting 10 seconds to add to the cool time, it compute again the heat map as well as the average temperature of all pieces and it start again. Once a pieces has reach the number of layers it is needed, this piece’s id is removed from the list and only the other pieces can now be printed.

Reinforcement Learning implementation

This section describe how a Q-learning agent was integrated into the pipeline, which take decision of which pieces to print based on its temperature, cooling time and numbers of layers.

Justification of Reinforcement Learning

A simple model of RL was chosen, because the thermal dynamic, time constraint of each part are complex time varying and not easily understand by normal algorithm. Q-learning allows to learn an optimal scheduling policy directly from the simulation and is easy and enough to work for selecting which pieces to print next.

Problem Formulation

* State (S): at each decision point, the state is encoded as a tuple per active piece (avg\_temp/10, cool\_time./10, layer\_count) where:
  + Cool time is the time since the last time the piece has been printed.
  + Avg temp is the average temperature of this pieces
  + Layer count is the number of layers printed so far
* Action (A): Selecting one pieces from the set of pieces where their average temperature is below the threshold.
* Rewards (R):
  + Always – 1 as a baseline penalty
  + +5 if a piece’s layer is printed immediately (no waiting time)
  + +1 if there is less than 30 seconds of waiting
  + -5 in other cases

Q-Learning Algorithm

* Update rule:
  + *Q* is the current estimate value of the action *a* in the state *s*, *α*∈[0,1] the learning rate, *r* the reward, *γ*∈[0,1] the discount factor, *s’* the following state after taking action *a*, *maxa′​Q(s′,a′)* is the estimated best possible value for the next state *s′* over all possible actions *a.*
* **Policy:** An ε-greedy strategy was implemented, with initial ε=0.2 decayed by 0.995 per episode to a minimum of 0.05.

All Q-table operations and policy decisions were handled by the class Q Agent in the q\_agent.py.

Integration into the Printing Loop

To integrate the model into the printing loop for it to take decision on which pieces to print, it is first loaded, and if the model wasn’t saved or was deleted, a blank model is created called Q-table.

The work flow of the RL model is as follow:

* The cooling time for each pieces is retrieve with the “get\_cooling\_time” function.
* Thermals information is update with “save\_heat\_stats” which save the statistics and call the main heat function, is access via this dictionnnary.
* The numbers of layer already printed for this pieces are retrieved with the “fetch\_number\_of\_layers” function that connect to Robot Studio and take directly the value.
* The current set is encoded via “agent.encode\_state(stats, piece\_id)”.
* A valid action set was filtered by temperature threshold.
* The model chose an action using “agent.choice\_action”, among the available pieces that can be print, then it print the corresponding choice using the “set\_piece\_choice()” function.
* After execution, a reward was computed, and “agent.update(...)” was used to adjust the Q-table.
* ε, the greedy strategy was decayed via agent.decay\_epsilon().

This loop continued until all parts reached the target layer count, at which point the Q-table was saved.

Experiment

To make every bit of code work, from robot studio to python, to do a voxel representation that works, connect python to robot studio, create the heat simulation and check if the results are realists, find out if the choosing path algorithms work and save time, implement a RL model, took approximately five hundred attempt that were made during this thesis.

Objectives

* Verify that the pipeline Robot Studio, Voxel, Heat Simulation produces temperature within realistic range
* Evaluate the efficiency of the path-selection algorithm in reducing total build time.
* Evaluate the efficiency of the RL model, in reducing total build time.

Experimental Setup

The equipment used during all of this thesis is a laptop Lenovo Legion with 24 gigabit of Ram, an intel core i5 processor as well as 6gb of Video Ram using Nvidia RTX 2060.

The simulation of the printing machine was done through Robot studio.

The python pipeline was used with Visual Studio Code with python v3.9.1.

Protocol

The experiments were carried out steps by steps:

For each geometry and run:

* Simulate one layer in robot studio while recording TCP coordinate.
* Generate voxel model and compute heat profile for all layers of every piece.
* Record Average temperature and cooling time.
* Apply the path selection algorithm and after the RL model to find which pieces to print.
* Record the total simulated build time for the two methods and compare it to a simple path.

Results Summary

In total, approximately 500 runs were effected, with 250 to create the fetching of coordinate as well as the voxel representation then 150 to create the heat simulation as realist as possible, and finally 100 to test the path choosing algorithm as well as the RL model.

Sources of errors and error margins

Simulation vs reality: as no data on the actual machine was available a margin of ±10 % on the thermal properties is allowed

Voxel resolution may neglect feature under 2 mm due to the reduction and allowed computation of the size of the voxel, it can be reduced even further with a higher voxel resolution.

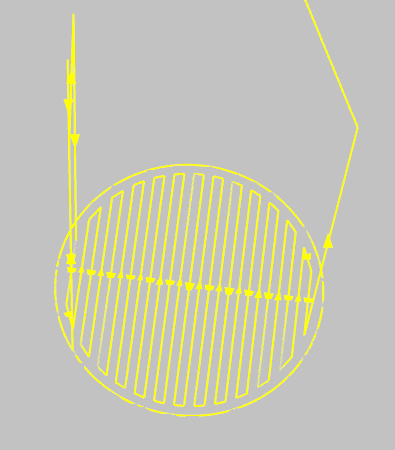
Cooling time measurement error is negligible.

The sources of errors come mainly from the code’s error as well as all the calculation that needed to be changed in order to fit to reality resulting in a lot of experimentation.

Work

The work conducted during this thesis has multiple steps and every one of them have to connected between each other’s perfectly otherwise errors, or miscalculation would have appeared, that is why it require precise and flawless execution.

Virtual WAAM Environment Setup

A virtual simulation of the machine 3D printing was created in Robot Studio that mimic the movement and speed of the real arm, multiple path for different pieces were done.

The movement when printing is 12 mm/s, and in between printed during transition phases it’s 800 mm/s. The TCP, world base coordinate and each piece coordinate were defined as separate tools and work object. When a piece is printed it send a signal call “weld” to indicate when it’s welding, this signal is turn to false when it is in transition phases as well as when the arm is reorienting itself in the same piece phase. At the end of every layer’s printed, the work object z dimension of this piece is incremented by 1 to simulate the layer’s thickness, whereas the tool work object doesn’t change so the TCP can print above the layer. When the printing is finished, the piece’s work object is put back to normal at z = -100. Also when a layer is done, the program waits for order from python’s main code to go and select a new piece to print.

Every pieces will be printed with a defined number of layers, to train and optimize everything, the number of layers printed throughout the whole thesis was from 2 to 15. As seen in Fig. 1, the raw path of the machine to print one layer of the piece number one. This part aligns with the aim of creating a realistic simulation, by creating a twin of the real machine and real pieces.

Figure 1 Raw path of the printing of piece 1 in Robot Studio

TCP Data Acquisition

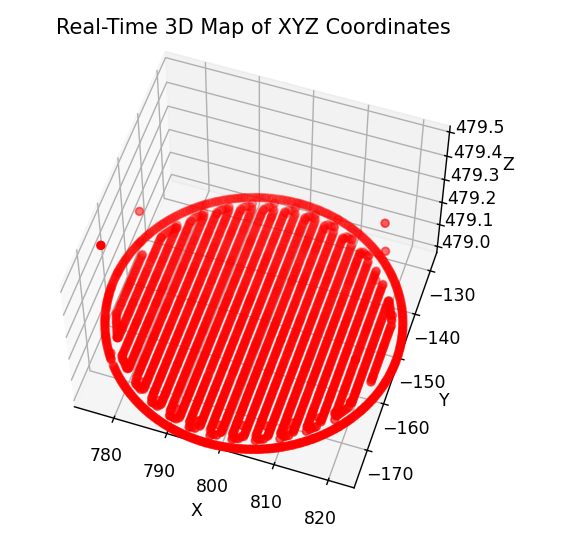


Figure 2 Coordinate fetched from Robot Studio Represented in 3D

To acquire the TCP coordinate, a function called “fetch\_xyz” is implemented in another function that fetch continuously: the TCP coordinate with “fetch\_xyz”, if it’s welding or not with “fetch\_weld”, and the state of the layer, if it’s finished or no. The coordinate are fetch at the frequency of 1kHz as the condition that the simulation is welding and the layer is not finished, those coordinate are save in a json file called “depositions\_points\_piece\_{piece\_id}.json”, so there is a separate json for every pieces, depending on the piece, but approximately 25 points are fetch each seconds when printing and fetching. Those are reset every time the main function is called. In Fig 2, the representation of all the points for one layer fetched are put into 3D.

Outlier Detection & JSON Logging Structure

Some outliers could have slipped by even with the welding and layer’s security, to fix this problem, the json of the current piece is load in “deposition\_point” variable, then the outliers are filter out by the function “filter\_points\_by\_layer“, that take as parameters, depositions\_points, min\_pts, for the minimum point per layer to be consider not outliers, set on 50, and the layer height set as 1.

Every points are now sorted out and clean, ready to have voxel transformation.

Voxel-Based Geometry Pipeline

To create the voxel representation after filtering the outliers and cleaning the data, multiple steps are to be done, first is to map the coordinate to the voxel indices, then to fill gaps between indices to be sure to have the full pieces. This is connected to the aim of creating a fully accurate representation of the pieces, in 3D.

Voxel Grid Parameters (resolution, layer thickness)

3 value are required to create the voxel representation: the number of layers, obtain once again with the function “fetch\_number\_of\_layer(url\_number\_of\_layers)” that take into parameter the URL to have access to the value of the number of layers of this pieces, it is nz, and will define the number of layer the voxel will have. Ny, and nx, will define the horizontal dimension and are defined by hand to 400 by 400 for the majority of the work, but as far as 2000 by 2000 voxel dimension were used.

Deposition-Point Voxel-Index Mapping

The deposition point are then mapped to the voxel’s indices using the function “point\_to\_voxel\_indices()” that takes into argument, the coordinate x, y, z from depositions points, nz, nx, ny, the layer height, as well as the range of the coordinate and their minimum.

In-Plane Filling & Vertical Smoothing

To fill gap in between points, a fill radius of dimension 3 by 3 is used, that fill voxel around one active voxel, with a matrix 3 by 3.

Also if needed a vertical smoothing is applied to allow a better consistency between layers. None is done here.

Bounding-Box Extraction and storage

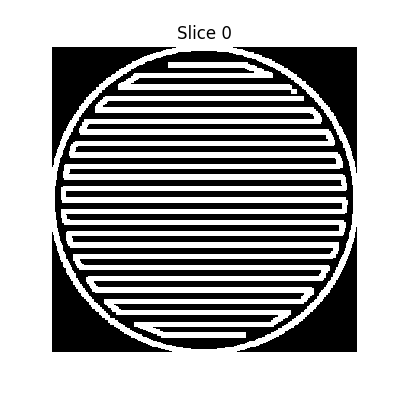
After having correct representation, it is saved using “save\_bounding\_boxes\_from\_grid()” function, that saves the bounding boxes with inside every active pixel of the piece, so it saves space. To use it later, what to do is just to reproduce a black out voxel within the bounding boxes dimension and put active pixel as active voxel. It is saved under a gzipped json to save even more space. All of that allowed to have even more precision, to go to thousands of voxels length and width without having to save too much, but it will take a lot of place and computer resources to do the heat simulation.

Figure 3 Voxel representation of the first layer of piece 1

Transient Heat-Transfer Simulation

The Heat-Transfer simulation is done using the gzipped json previously created after, the voxel are reproduce inside the “heat\_simulation()” function using the active pixel and bounding box dimension. After that the geometry of each pieces is find and saved in a dictionary using the function “analyze\_geometry()”, that takes into parameters active pixels and the bounding box dimension. It is then returning the density, the average thickness of “wall” inside piece, the average distance within the piece.

ODE Solver Implementation for Each Voxel

For each voxel this ODE equation is used to calculate and update its temperature:

T [z, y, x] = T + dt \* alpha \* Q - beta \* (T - T\_amb) - gamma \* (T\*\*4 - T\_amb)

With every piece’s part that is connected compute its own temperature according to this equation, dt is set to 1, otherwise it takes too much computer power, but for more accurate heat propagation other value must be attributed, same for steps\_per\_layer, which is how many seconds in the future the heat is to be predicted, it is also set to one for the same reason.

Empirical Parameter Calibration (α, β, γ)

The heat equation, used to calculate the temperature of each part of pieces, are modified according to the pieces geometric property disclosed previously with this ratio:

α= (0.8 \* compactness + 0.75 \* (1 / (1 + thickness)) + 1 \* density) \*((number\_layer+11)/13),

β= (0.03 + 0.01 \* (1 - compactness) + 0.02 \* (gap / 50.0)) \*(((time\_cooling))\*\*2),

γ= (1e-6 + 5e-6 \* compactness + 1e-6 \* (distance / 15.0)) \*(((time\_cooling))\*\*1.05).

The ratio within α, β, γ changed during the thesis work to try and manage to fit appropriately the reality of temperature, especially with the cooling time which here is boosted to accelerate the learning of the AI, but should not be if more realistic over time cooling is wanted. Otherwise, the equations are balanced so the more layers the more heat and less quick to dissipated the heat. It is also heavily influence by the compactness and density, which are similar, and the heat dissipation is influence by the gab within the piece as well as its compactness too.

In-Slice Neighbourhood Averaging & Time-Stepping

Finally, the temperature of each voxel to be put in the ODE equation is calculated using the average temperature of its neighbour, using a 3\*3 matrix around the voxel with a 40% ratio and 60% ratio for the voxel in itself, and that’s the T in the formula.

Saving the heat map

Using the function “compute\_piece\_avg\_temp()”, return the heat map as well as the average temperature of the piece, to create the threshold needed later. Implemented in another function, called “save\_heat\_stats()”, the map is saved under a .npy file, as well as the average temperature, the file\_name, the cool time, and the dimension of the map under a dictionary to access it at any time, and display all those statistics needed.

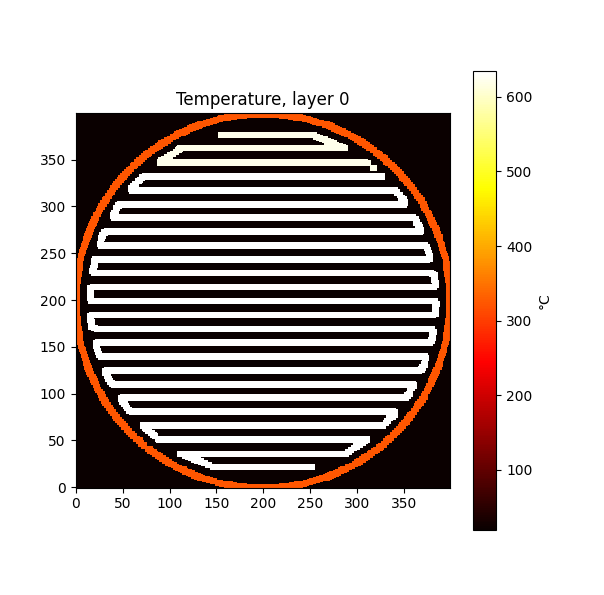
With that the whole heat pipeline is done, and the heat representation is clear and working.

Figure 4 Heat map of piece 1

Reinforcement-Learning Driven Build-Sequence Scheduling

The build sequence scheduling is an important part of the work, as it chooses which pieces to print next and is touching the core of the thesis’ goal which is to save time and have good quality over the pieces printed. For that a Reinforcement Learning model is build.

Design of the Reward Function

The reward function was designed so the model gets positive reward if it saves time.

It is like that:

* Always -1 as a baseline penalty to discourage unnecessary delay
* +5 if it prints immediately
* + 1 if it prints within 30 seconds of wait maximum
* -5 if there were more than a 30 seconds wait

This scheme strongly rewards immediate prints, which is the goal, to have less time possible while still allowing some short waiting time.

Scheduling Algorithm Logic

**State Vector:** To be the most efficient possible but still keeping a relatively simple model, the agent has access to the cool time of each pieces, its number of layers as well as its average temperature, the average temperature and cool time are scaled down by dividing them by 10 each.

**Valid Action:** The model has to make a choice of a piece to print where its average temperature falls below a threshold of 400-degre °C. Otherwise it waits 10 seconds before it computes again the temperature and be able to make a choice again, doing so until at least one pieces become printable.

**Action Selection:** ε-greedy policy with ε starting at 0.2 and decayed by a factor of 0.995 per decision to a minimum of 0.05.

Q-Learning Algorithm and Hyperparameters

The Q-learning update: .

was implemented with a learning rate α = 0.1 and discount factor γ = 0.9, as specified by the Q-Agent constructor in q\_agent.py. These values were selected based on an initial set of short trial episodes, which indicated stable convergence behaviour without further tuning.

Integration with Thermal Model and Python Pipeline

To integrate the Reinforcement Learning model with the python pipeline, and the thermal model which is done inside the pipeline, the model, a Q-table, is first load if it exists otherwise it is initialized. After printing one layer per piece, and doing a heat simulation in all of them in a dictionary called “stats” using “save\_heat\_stats()”. A new loop is started where the model observes the new state, computes the reward for the preceding action. Then the model takes an action using “agent.choose\_action(state, valid\_actions)” and choose a piece’s to be printed that is available, the rewards is then calculated, the pieces are printed using ‘set\_piece\_choice(choice)” with the choice of the model. After the layers is complete “agent.update(prev\_state, prev\_action, reward, state, valid\_actions)” is called and ε is decayed via “agent.decay\_epsilon()”.

Finally, when all the pieces are printed, the new agent is saved and a graph of the rewards progress is saved. And the total time to print everything is shown.

Experimental Trials and Validation

To train and optimize the build sequence, a series of trials were done using different heat simulation parameters, blank RL models, using different number of layer to print, and different path choosing algorithm to compare.

### Selection of Test Geometries and Simulation Parameters

Experience included different piece complexities, layer numbers ranging from 5 to 10, and various empirical parameter calibrations to explore their impact on the build sequence optimization. For instance:



Figure 5 Evolution of AI performance for the first run under case 1 condition, reward per loop according to which loop run

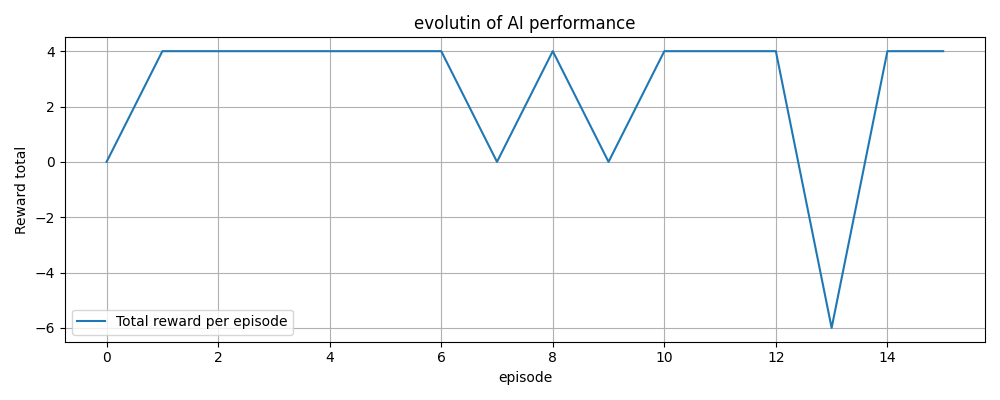


Figure 6 Evolution of AI performance for the second run under case 1 condition, reward per loop according to which loop run

### ****Test Case 1 (Trial 1 - Figure 5 & 6):****

Accelerated cooling parameters, blank RL model:

* **α**: (0.8 × compactness + 0.75 × (1 / (1 + thickness)) + 1 × density) × ((number\_layer+11)/13)
* **β**: (0.03 + 0.01 × (1 - compactness) + 0.02 × (gap / 50.0)) × (cool\_time²)
* **γ**: (1e-6 + 5e-6 × compactness + 1e-6 × (distance / 15.0)) × (cool\_time^1.05)

**Result:**

* RL: 1548.43 s
* Algorithm: 1604.12 s

### ****Test Case 2 (Trial 2 - Image 7):****

Modified parameters for extended cooling and sensitivity exploration, with the model that trained under case 1 :

* **α**: (0.4 × compactness + 0.5 × (1 / (1 + thickness)) + 0.9 × density) × ((number\_layer+8)/10)
* **β**: (0.02 + 0.03 × (1 - compactness) + 0.01 × (gap / 50.0)) × ((cool\_time + 0.01)³)

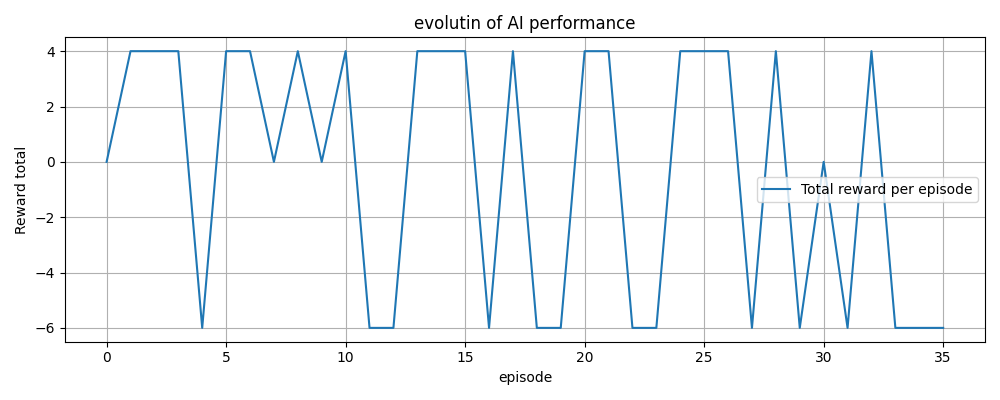


Figure 7 Evolution of AI performance for the first run under case 2 condition, reward per loop according to which loop run

* **γ**: (1e-6 + 5e-6 × compactness + 1e-6 × (distance / 10.0)) × ((cool\_time + 1/120)^1.05)

**Result:**

* RL: 7274.27 s
* Algorithm: 7247.04 s
* Simple queue: 6535.71 s

### ****Test Case 3 (Trial 3 – Figure 8):****

Adjusted parameters for increased realism, with now model train on the first two cases:

* **α**: (0.8 × compactness + 0.9 × (1 / (1 + thickness)) + 0.8 × density) × ((number\_layer+18)/20)
* **β**: (0.03 + 0.01 × (1 - compactness) + 0.05 × (gap / 50.0)) × (cool\_time²) *(cool\_time/10 for realism)*
* **γ**: (1e-6 + 5e-6 × compactness + 1e-6 × (distance /25.0)) × (cool\_time^1.03) *(cool\_time/12 for realism)*

**Result:**

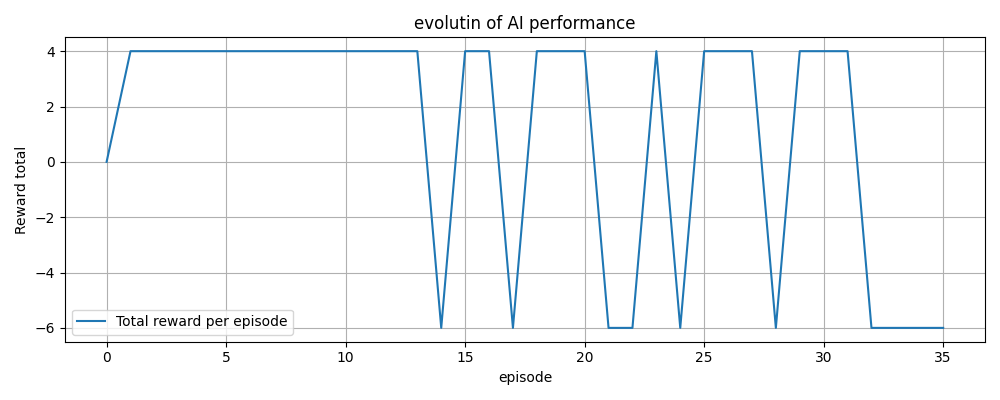
* RL: 6314.76 s
* Algorithm: 7271.21 s *(RL significantly better*)

Figure 8 Evolution of AI performance for the first run under case 3 condition, reward per loop according to which loop run

### ****Test Case 4 (Trial 4 – Figure 9):****

Further adjusted parameters for fine-tuning realism and balance, started with a blank RL model:

* **α**: (0.8 × compactness + 0.9 × (1 / (1 + thickness)) + 0.8 × density) × ((number\_layer+23)/25)
* **β**: (0.03 + 0.01 × (1 - compactness) + 0.05 × (gap / 50.0)) × (cool\_time²) *(cool\_time/10 for realism)*
* **γ**: (1e-6 + 5e-6 × compactness + 1e-6 × (distance /50.0)) × (cool\_time^1.05) *(cool\_time/12 for realism)*

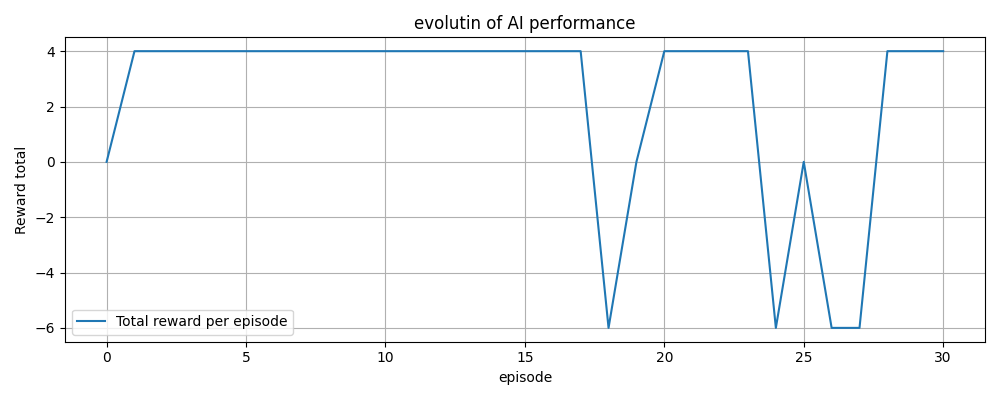


Figure 9 Evolution of AI performance for the first run under case 4 condition, reward per loop according to which loop run

**Results (First Run):**

* RL: 2668.89 s
* Algorithm: 2629.95 s *(Algorithm slightly better initially)*

### ****Test Case 5 (Trial 5 - Figure 10):****

Identical parameters to Test Case 4, but with an additional run to confirm RL performance improvement after continued training.

**Results (Second Run after additional training):**

* RL: 2575.98 s (Improved from previous RL run)
* Algorithm: 2629.95 s (Algorithm unchanged, RL now better)

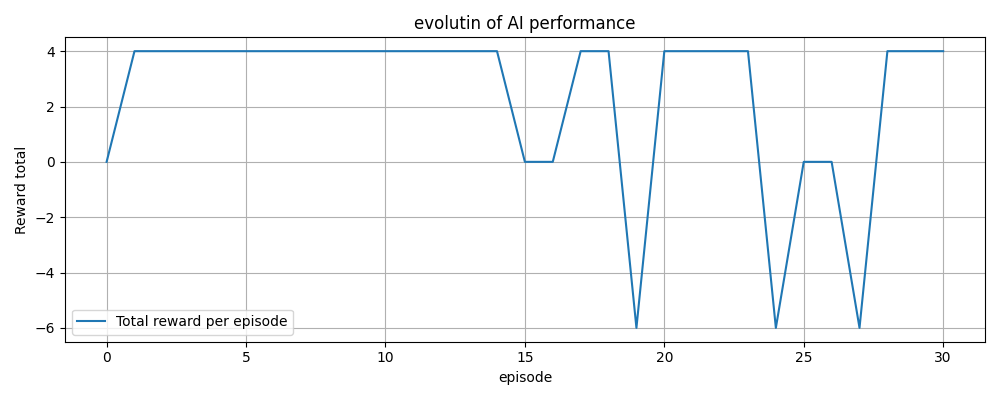


Figure 10 Evolution of AI performance for the second run under case 4 condition, reward per loop according to which loop run

All the trials were done, only the most relevant one were retain.

### Trial Protocol (number of runs, data captured)

A systematic experimental protocol was followed:

* Number of trials: Over 100 experiments.
* Data captured: TCP coordinates, voxel representations, cooling times, temperature distributions, RL rewards, and total build times.

Only the relevant experiment were selected and display here. Most of them had to be abort during the training to change values to optimize the heat or the Reinforcement Learning model.

### Performance Metrics (temperature error, build-time saving)

Performance was primarily evaluated using total build time, reinforcement learning reward metrics, and comparative performance against a deterministic selection algorithm:

Table 1 Trials Results

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | RL Total Time (s) | Algorithm Total Time (s) | Time Improvement (%) |
| 5 layers (Case 1) | 1548.43 | 1604.12 | +3.47% (RL Faster) |
| 5 layers (additional run) | 1560.87 | 1604.12 | +2.70% (RL Faster) |
| 10 layers (Case 2) | 7274.27 | 7247.04 | -0.38% (Algorithm Faster) |
| 10 layers Realism Adjusted (Case 3) | 6314.76 | 7271.21 | +13.16% (RL Faster) |
| 10 layers (Second Trial) | 2575.98 | 2629.95 | +2.05% (RL Faster) |

Interpretation:

RL consistently provided improvements in total printing time in scenarios tuned for realistic and computationally balanced parameters.

RL exhibited slight disadvantages in some specific setups, indicating sensitivity to parameter tuning.

### Error Sources and Uncertainty Quantification

Results and discussion

The Results and discussion section presents a detailed and objective description of the results. This can be done by showing tables, figures which are numbered and given explanatory figure/table texts. Figure numbering is below the figure while the tables get their numbering above. If tables or figures are too big or too many, the most interesting are added in this section and the rest must be placed in the appendix, see Appendix. In graphs don’t forget to state units and to use the figure area optimal. Avoid strange/hard reading colours and symbols; it must be easy for the reader to understand the graphs. This section must also conclude the results and findings (or if too extensive in a new section for result discussion). The purpose is to discuss and analyse the meaning of the results related to the theoretical framework, related jobs and methodology. Questions to be asked when you write this section: what do the readings say, how likely are they, can they be generalized, how significant are the results, outliers, etc. In this part, personal reflections are allowed.

Conclusion

The final aim of the study is to make conclusions from what is analysed, implemented, tested or measured. Discuss and answer all the aims and questions formulated in the introduction. The conclusions drawn should refer to the initial aims, objectives, literature and problems. The validity of the conclusions should also be discussed and evaluated.

Future Work and Research

It is common to conclude with a section on future work, either as part of the conclusion section or as a new chapter. It is sometimes important to recognize how the project can move forward in solving future problems.

Critical Discussion

Personal reflections and personal experiences during the work.

Generalization of the result

Discuss how general the result is and if they can be useful in other areas. How unique is this study?

References

|  |  |
| --- | --- |
| [1] | Unknown, ”Advanced Metal 3D Printing Center,” Michigan Technological University, 2025. [Online]. Available: https://www.mtu.edu/metal-3d/services/orientation/methods/. [Använd 05 2025]. |
| [2] | D. M. Z. B. V. A. B. S. B. A. S. J. L. E. E. N. Svetlizky D, ”Directed energy deposition (DED) additive manufacturing: Physical characteristics, defects, challenges and applications,” *Materials Today,* vol. 49, nr 10.1016/j.mattod.2021.03.020, p. 25, 30 11 2021. |
| [3] | Unknown, ”Wire Arc Additive Manufacturing (WAAM),” Migal, 2025. [Online]. Available: https://www.migal.co/en/waam#:~:text=Wire%20Arc%20Additive%20Manufacturing%20(WAAM)%20is%20a%20manufacturing%20process%20used,print%20or%20repair%20metal%20parts.. [Använd 05 2025]. |
| [4] | M. S. A. S. a. U. R. O. MOKROV, ”A FINE MODIFICATION OF THE DOUBLE,” i *ISF – Welding and Joining Institute,*, Aachen, 2019. |
| [5] | M. T. a. V. Petkov, "A THERMAL MODEL FOR WIRE ARC ADDITIVE MANUFACTURING," in *14th International Scientific and Practical Conference*, Latvia, 2024. |
| [6] | W. Shen, ”Research on virtual simulation design of ABB robot welding operation based on Robotstudio,” i *2020 IEEE International Conference on Artificial Intelligence and Computer Applications*, Dalian, China, 2020. |
| [7] | S. Y. S. Y. J. M. Ruixiang Cao, ”Printing, Poxel: Voxel Reconstruction for 3D,” p. 3, 2025. |
| [8] | A. P. Alexey Tolok, ”Functional-Voxel Method in Problems of Geometric,” Moscow, 2020. |
| [9] | F. F. O. F. M. M. Etienne Chassaing, ”Thermoxels: a voxel-based method to generate simulation-ready 3D thermal models,” Lausanne, 2025. |
| [10] | D. C. X. Z. C. G. L. L. Y. H. W. B. Xijun Zhang, ”Machine learning-driven 3D printing: A review,” *Applied Materials Today,* vol. 39, nr https://doi.org/10.1016/j.apmt.2024.102306, 2024. |
| [11] | V. S. N. P. A. M. M. a. M. P. F. J. M. A. Fabio Parisi, ”A new concept for large additive manufacturing in construction: tower crane-based 3D printing controlled by deep reinforcment learning,” *Construciton Innovation,* vol. 24, nr doi/10.1108/ci-10-2022-0278/, p. 25, 2023. |

* 1. Appendix

The appendices gather everything that is not necessary, for the reader, to be able to follow and understand the main part of the report. An appendix might include detailed information such as tables of measured data, calculations, program codes etc. It can also include figures that are too large to insert in the text. An appendix is an independent document and pages are numbered separately from the rest of the document or other appendices.

Extensive program listings or measurement sets are usually not ok even for an appendix. It is impossible to read and interpret. These are usually best distributed as files and cannot be part of the degree report. If you work together with a company, make sure that any pictures or data presented are approved by the company to be published.

* 1. Appendix

Delete all appendices if not needed. Note, to be able to delete all appendices you must follow these steps to preserve the template. This guide is tested on Word 365 and 2019 on Windows but works in the same way for older versions.

|  |  |  |
| --- | --- | --- |
| Step | Action |  |
| 1 | Double click with the mouse in the Header area of the Appendix A page. It is at the very top of the page. Note, it must be Appendix A. |  |
| 2 | In the “Header & Footer” menu ribbon select [Link to Previous] option in the Navigation pane. |  |
| 3 | In the dialog select Yes. |  |
| 4 | Double click with the mouse in the Footer area of the Appendix A page. At the very bottom of the page where you see the page number [A:1]. |  |
| 5 | Redo the [Link to Previous] manoeuvre. |  |
| 6 | Close the Header and Footer. |  |
| 7 | Turn on show paragraphs to be able to see all section breaks. |  |
| 8 | Move the cursor to before the [section break] on the page of the References list chapter 7. |  |
| 9 | Press [F8]. |  |
| 10 | Move the cursor to the end of the document. |  |
| 11 | Press [Delete]. |  |