

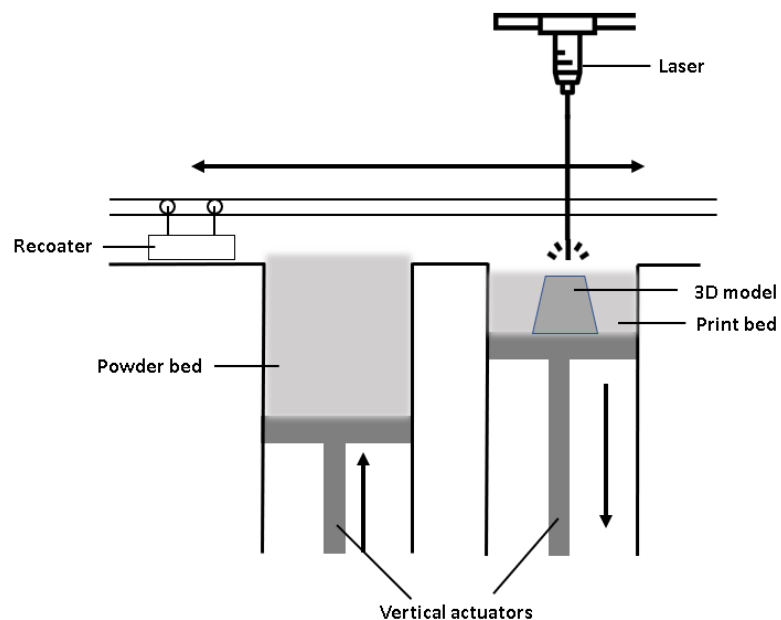
Audun Mostad, Jonatan Lærdahl, Karl Peder Mørkeseth, Mathias Kommedal

Development of miniature metal 3D printer

Utvikling av miniatyr 3D printer

Bachelor's thesis in Electrical engineering
Supervisor: Sigurd Gosse

May 2022

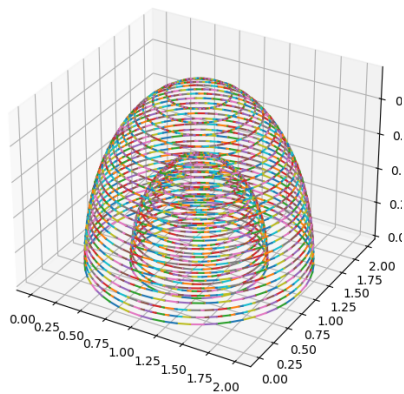


Karl Peder Mørkeseth

Audun Mostad, Jonatan Lærdahl, Karl Peder Mørkeseth, Mathias Kommedal

Development of miniature metal 3D printer

Utvikling av miniatyr 3D printer



Bachelor's thesis in Electrical engineering
Supervisor: Sigurd Gosse
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics



Kunnskap for en bedre verden

Tittelside Bacheloroppgave BIELEKTRO


Oppgavetittel (norsk og engelsk): Utvikling av metall 3D-printer	
Forfattere: Mathias Kommedal Jonathan Lærdal Audun Mostad Karl Peder Mørkeseth	Prosjektnummer: E2208
	Innleveringsdato: 20.05.2022
	Gradering: <input checked="" type="checkbox"/> åpen <input type="checkbox"/> lukket
Studium:	Elektroingeniør - BIELEKTRO
Studieretning:	Automatisering og robotikk
Veileder internt:	Sigurd Gosse
Institutt:	Institutt for teknisk kybernetikk
Oppdragsgiver:	SINTEF, Trondheim
Kontaktperson:	Eivind Johannes Øvrelid, tlf: 982 30 449, email: EivindJohannes.Ovrelid@sintef.no
Sammendrag (norsk og engelsk): SINTEF ønsket en metall 3D-printer for å teste nye metallegeringer. Det var derfor ønskelig å lage en egen 3D-printer med et lite printeområde. Hovedkomponentene var en eldre graveringslaser fra Rofin og et pulverdistribusjonsbord. Gruppen har utviklet et kommunikasjonssystem mellom de to komponentene, og utviklet programvare som gjør det mulig å printe 3D-modeller i metall. Dette gjøres ved å "slice" en STL-fil (3D-modell) til DXF-filer som laseren kan lese. Pulverdistribusjonsbordet legger ut et tynt lag metallpulver. Laseren smelter pulveret etter informasjon fra dxf-filen. Denne prosessen repeteres lag for lag til metall-modellen er ferdig printet. Det har blitt gjort mange modifikasjoner på omramningen til laseren og de elektriske kretsene i systemet for å optimalisere produktet. Engelsk: SINTEF wanted a metal 3D printer to test new metal alloys. It was therefore desirable to create a 3D printer with a small print area. The main components were an older engraving laser from Rofin and a powder distribution table. The group has developed a communication system between these two components, and developed software making 3D printing in metal possible. This is done by "slicing" an STL file (3D model) into DXF files for the laser to read. The powder distribution table applies a thin layer of metal powder. The laser melts the powder according to information in the dxf files. This process is repeated layer by layer until the metal object is finished. Multiple modifications have been done on the lasers framing and the systems electrical circuits to optimize the product.	
Stikkord norsk: 3D-printing, metallegeringer, laser, programmering, kommunikasjonssystem, automasjon, elektriske kretser	Stikkord engelsk: 3D printing, metal alloys, laser, programming, communication system, automation, electrical circuits

Preface

This bachelor thesis is the concluding project of the participants bachelors degree at the Norwegian University of Science and Technology. The project consist of 20 out of 30 credits in the spring semester of 2022. This makes up about 500 hours of work per student. Our group consist of four electrical engineering students. All four students study “automation and robotics” as their major. This bachelor thesis is written for SINTEF, with the goal of developing a small scale metal 3D printer.


We would like to thank senior researcher Eivind Johannes Øvrelid from the sustainable energy technology department at SINTEF as our supervisor through the project. We thank him for the positive continuous dialog, sound advice and aid through the course. We would also like to thank Bendik Sægrov-Sorte for his technical support and aid in 3D printing of necessary parts. We also thank Sigurd Gosse as our supervisor from NTNU for guidance and feedback throughout the project.

Signatures




Karl Peder Mørkeseth

karlpm@stud.ntnu.no



Jonatan Lærdahl

jonatala@stud.ntnu.no



Mathias Kommedal

mathiako@stud.ntnu.no



Audun Mostad

Audunmos@stud.ntnu.no

Contents

List of Figures	v
List of Tables	vii
1 Introduction	1
2 Background	2
2.1 Theory	2
2.1.1 Additive manufacturing (3D printing)	2
2.1.2 System concept	2
2.1.3 STL file	3
2.1.4 DXF file	4
2.1.5 Slicing	4
2.1.6 Metallic powder	5
2.1.7 Powder Melting and Oxidation	5
2.1.8 Communication	6
2.1.9 Electrical components	7
2.1.10 Python	8
2.2 Equipment	9
2.2.1 Rofin F30	9
2.2.2 The framing (CombiLine Advanced RT 800)	12
2.2.3 Powder distribution system (PDS)	13
2.2.4 Raspberry Pi 3 Model B (RPi)	14
2.2.5 New control panel	15
2.2.6 Inspection cam	15
2.3 Programs	16
2.3.1 VisualLaserMarker (VLM)	16
2.3.2 LaserConsole (LC)	16
2.3.3 Autodesk Fusion 360	16
2.4 Specifications the laser	17
3 Methods	19
3.1 Two solutions	19
3.1.1 Pros and cons	19
3.1.2 Decision of workpath	20
3.2 Design process	21
3.3 Approaches	22
3.3.1 The framing	22
3.3.2 Electrical circuits and terminal blocks	26
3.3.3 Printing Chamber	29
3.3.4 New powder overflow-collector	30
3.3.5 New recoater	30
3.3.6 File Transfer	32
3.4 Software development	34
3.4.1 Slicing script	34
3.4.2 Visual Laser Marker (VLM)	38
3.4.3 The powder distribution system	42

3.4.4	Graphical User Interface	44
3.5	Manual	49
3.5.1	Setting up the Powder distribution table	49
3.5.2	Setting up the laser	50
3.6	Flowchart of the system	52
4	Results	53
4.1	End product	53
4.1.1	Print process	53
4.1.2	Safety	54
4.2	Communication	54
4.2.1	Step by step	55
4.3	Print testing	56
4.4	Print quality	58
4.4.1	Without protective atmosphere	58
4.4.2	With protective atmosphere	58
5	Discussion	59
5.1	The work process	59
5.2	Product	59
5.3	Further development	60
5.3.1	Different linear actuator	60
5.3.2	Chamber	61
5.4	SINTEF's feedback	61
6	Conclusion	62
7	References	63
A	Accounting	66
B	Power diagram to supplementary components	68
C	M-functions IO	69
D	Raspberry Pi – > Relay controlling unit	70
E	Relay controlling unit	71
F	Front drawing	72
G	Control room drawings	73
H	Main program and GUI script	78
I	Controlling the PDS script	85
J	Create and show plot script	88
K	Converting STL file to DXF files script	90

L	Moving files from USB drive to folder script	93
M	Numbers keyboard script	95
N	Show video script	97
O	VLM script	98

List of Figures

1	System sketch of the 3D printing process.	3
2	Concept of tessellation with the use of triangles [5].	3
3	Slicing process [10]. From 3D model to sliced 3D model to 3D model being printed one slice at a time.	4
4	Oxidation with different levels of O_2 [18].	6
5	Example of terminal block.	7
6	24 V relay with DIN rail mount.	7
7	Siemens Sirius 24 V Contactor.	7
8	Power unit for the laser [23].	9
9	Backside of the power unit for the laser [23].	9
10	Front view of the 19" plug-in PC [23].	10
11	Back view of the 19" plug-in PC [23].	10
12	Originally external control panel [23].	10
13	Laser head, marked in the red square [23].	11
14	Marking head, marked in the red square [23].	11
15	The original frame of the system [24].	12
16	Powder distribution system in the chamber.	13
17	Raspberry Pi 3 Model B collected from [30].	14
18	10.1 inch LCD screen for Raspberry Pi.	15
19	Endoscope for the RPi [32].	15
20	The external control panel.	22
21	External control panel removed.	23
22	Front housing removed.	23
23	New control room design.	24
24	New front design.	24
25	New lighting on the ceiling.	25
26	New lighting in the control room.	25
27	Interfacing between different control units.	26
28	Light curtains and control unit.	27
29	Rotary table and engine.	27
30	Relay closes a switch on receiving an electrical signal.	27
31	The control system.	28
32	The printing chamber.	29
33	Components designed in Fusion 360.	30
34	New recoater solution.	31
35	Diagram of the file transfer setup.	32
36	Two points of intersection at the second layer.	34
37	The outline created by drawing a polyline between all of the points.	37
38	The outline hatched within VLM.	37
39	Function to define what DXF file is imported.	38
40	Function to import DXF file (dxfImport).	39
41	Declared file path for the folder that the "import.DXF"-function.	39
42	Location of the folder.	39
43	Example of the DXF files structured in the "Converted"- folder.	39
44	Function that imports hatching settings for the DXF files.	40
45	Incrementing hatch angle.	40

46	Sends signal for to the PDS to start.	41
47	Receives signal that PDS is finished.	41
48	Write, WaitOn and Read IOBit for communication between the units.	41
49	Example of communication between PC and a Zaber device from Zaber ASCII protocol [36].	42
50	Overview of the Raspberry Pi's GPIO-pins.	43
51	Snippet collected from the logging file, communication_log.txt. This is the same communication as can be seen in figure 49 taken directly from Zabers ASCII protocol.	44
52	The info box in the GUI.	45
53	The parameters-box in the GUI.	45
54	The pop up keyboard.	45
55	The progress-box in the GUI.	46
56	The text box with progress bar.	46
57	Buttons box.	47
58	Pressing the Preview-button shows a pyplot of the figure from the selected file.	47
59	Safety messages telling the user what needs to be done before the program can complete the desired task.	48
60	Parameters-box	49
61	Preview-box	49
62	Convert-box	49
63	Buttons-box.	50
64	Buttons-box.	50
65	Shutter open.	50
66	Initialize hardware.	51
67	Load job.	51
68	Start marking.	51
69	New front panel mounted.	54
70	The new control room.	54
71	Illustration of the systems general communication.	55
72	Communication between units.	56
73	An example of how print testing was done.	57
74	The upper side.	58
75	The under side.	58
76	Test print with one layer.	58
77	Test print with 2 layers.	58

List of Tables

1	Technical data on the laser head. Obtained from the Rofin manual [23].	17
2	Technical data on the marking head. Obtained from the Rofin manual [23].	18
3	Zaber_motion commands. [37]	42
4	Accounting of purchases	66
5	Accounting of purchases, pt.2	67

Abbreviations

AM - Additive Manufacturing

UI - User Interface

SLM - Selective Laser Melting

SLS - Selective Laser Sintering

EBM - Electron Beam Melting

DMLS - Direct Metal Laser Sintering

GUI - Graphic User Interface

PDS - Powder Distribution System

VLM - Visual Laser Marker

VMC2 - Visual Marker Control 2

RPi - Raspberry Pi

STL - Standard Triangle Language

DXF - Drawing Exchange Format

FTP - File Transfer Protocol

GPIO - General Purpose Input/Output

1 Introduction

This bachelor thesis is written by four students at the Norwegian University of Science and Technology (NTNU). All four students study electrical engineering and have “automation and robotics” as their major. The thesis consist of 20 out of 30 credits of the final semester of the study plan. The group wanted to work with a hands on assignment in cooperation with a company that could offer a challenging task, good resources and experienced guidance. From the many bachelor thesis presented to us, “Development of control system for metal 3D printer” announced by SINTEF Industri caught our attention. The thesis was the groups first choice and we were happy to be assigned to it.

SINTEF wanted to create a metal 3D printer for testing metal alloys. In the cellar of the Berg-building at NTNU, campus Gløshaugen, SINTEF had a laser from Rofin developed for engraving in metal. This unit was going to be one of two main components of the 3D-printer. The lasers new task is melting metal powder. The other component is a powder distribution system (PDS). It would be distributing metal powder. The PDS consisted of 3 actuators.

The task written by SINTEF was to develop a communication system between these two components and by doing so creating a functioning 3D printer. The finished system should run a printing process automatically. Meaning that the process, from pushing a print button until the finished result, should run without any manual assistance from an operator. A subtask introduced by SINTEF was to optimize the framing surrounding the components to reduce the systems volume. This included maintaining safety features while the laser is running. The group were given the freedom to explore their own solutions and approaches.

To solve the task and deliver a satisfactory end product the group has worked with multiple relevant fields. This includes disassembling and mounting a new framing for the system, software development in python and Microsoft visual basic, reporting and project management and electric circuit work. This report starts with describing used equipment and explaining the necessary theory to understand the methods used in the project. It continues with the approaches and methods used to explore the projects solutions, followed by the achieved results. The report finishes with a discussion and a conclusion. Accounting, circuit diagrams, sketches, scripts and a poster are added as appendices at the end.

In summary, the main task of the project is to make a functional metal 3D printer automatically printing from 3D models.

2 Background

This chapter is included to provide the necessary theory needed to get an understanding of the project. The background theory, chapter 2.1, consist of knowledge the group has acquired throughout the electrical engineering study program and during this bachelor thesis. The chapter also contains information about the equipment and software used in the project.

2.1 Theory

2.1.1 Additive manufacturing (3D printing)

Additive production or additive manufacturing (AM) is a production technique where 3D models are built up by layers using a marking head. Additive manufacturing is the opposite of subtractive manufacturing. Here the product is made by removing parts of the raw material around the desired product. There are some advantages with AM. The waste of raw material is far less, as unused powder can be recycled and reused. It is also much more suited for prototyping and small productions considering resources, cost and time.

Additive manufacturing can be done using multiple techniques. The most common techniques used for production of metal units are Selective laser sintering/melting (SLS/SLM), Direct Metal Laser Sintering (DMLS), and Electron Beam Melting (EBM) [1]. These techniques are all powder based. The technique used in this bachelor thesis is Selective laser melting (SLM).

2.1.2 System concept

Briefly explained, the general system is driven by four main moving components: two vertical actuators, a powder recoater and the laser. The first vertical actuator delivers powder by moving upwards. The second vertical actuator, which is the building platform, moves down to make space for a new layer of powder. The powder recoater moves horizontally, distributing powder from the powder chamber to the build platform on the second actuator before returning. When the recoater has returned back to its nominal position, the laser melts the powder in accordance with the figure for the current slice. These steps are repeated until all the "slices" are melted and the finished product has been produced. The figure below is made by the group to visualise the concept.

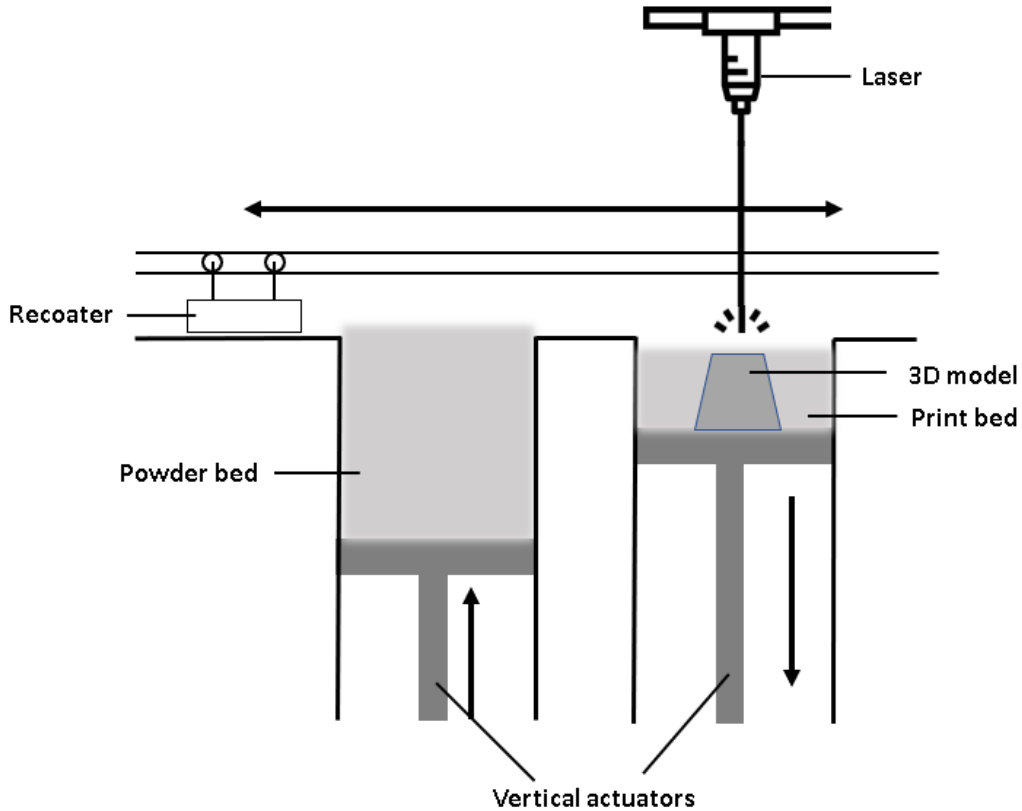


Figure 1: System sketch of the 3D printing process.

2.1.3 STL file

One of the most common file formats used for 3D printing is “.STL” [2]. STL stands for Standard Triangle Language or Standard Tessellation Language. The files consists of the surface geometry of a 3D model and is generated by using a concept called tessellation. Tessellation is when a geometric shape covers a 2D or 3D surface without overlapping or leaving any gaps by repeating itself [3]. The STL-file describes the surface of the 3D object, without considering texture and color, by using large amounts of triangles. The triangles are called facets and each facet has a perpendicular direction as well as the three points that represents the corners of the triangle [4].

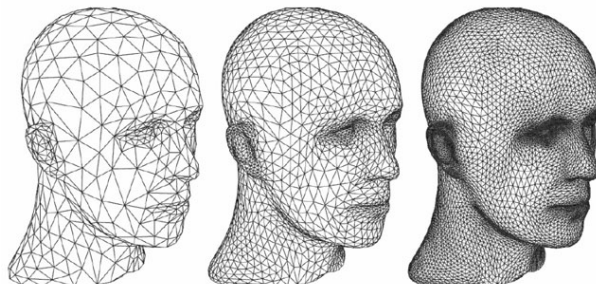


Figure 2: Concept of tessellation with the use of triangles [5].

The majority of 3D modelling programs have an STL export option. Before being used for printing the STL-file needs to be sliced. The group has built a program for this process. It will be explained in depth later on in the report.

2.1.4 DXF file

DXF stands for Drawing Exchange/Interchange Format. The file is a tagged data representation of an AutoCAD drawing file [6]. It can therefore represent both 2D and 3D models. The DXF-file type is a vector file type [7]. Vector based files use mathematical equations, lines and curves and not pixels as a raster file type does [8]. Examples of raster file types are .JPG and .PNG.

One of the advantages of using vectors is that there are no loss of resolution after scaling [8]. Pixel based files on the contrary only maintain its resolution when it is a specific size. DXF files are also open source so anyone can edit the file without requiring a specific program or license.

2.1.5 Slicing

The 3D printer builds the product up by stacking multiple layers. For the 3D structure to be built, a model needs to be processed by a slicing software. This software manipulates the 3D model by dividing it into multiple slices. These slices are 2D representations of a layer, and recreates the 3D model when stacked. The resolution depends on the layer thickness, a variable decided by the operator, it also decides how many slices the model is cut into. By having a smaller layer thickness the model is cut into more slices. The layer thickness also plays a role in how the physical performance and properties of the finished product ends up. These factors are also affected by which metals are used and different parameters set by the laser.

A good method for slicing 3D models is using data from an STL file[9]. STL files (more about these in section 2.1.3) contains information about the outline of a 3D model and is supported by most modelling software's. For example Fusion360, which is a program used for creating 2D and 3D models. Some 3D modelling programs have a slicing function implemented, but for the most part the slicing is done in another program after the model is exported as an STL file. The figure below is a simple illustration of the process:



Figure 3: Slicing process [10]. From 3D model to sliced 3D model to 3D model being printed one slice at a time.

2.1.6 Metallic powder

Metallic powder is metal broken into small particles. The powder particles used for SLM usually have a spherical form and have an average diameter of 10-45 micrometers [11]. The most common powders used are steel and iron-, nickel-, titanium- and aluminium-based alloys, alumina, silicon carbide and yttria stabilised zirconia [12].

Based on the metallic powder being used the user must choose suitable parameters such as layer thickness, laser power, frequency, scanning speed, pulse width, line width, scan strategy and more to get the desired integrity of the product.

Heat treatment is also an additional option to reduce the thermal residue stress and improving print quality [13].

2.1.7 Powder Melting and Oxidation

The metal powder melts when the laser beam is directed at it with the right effect and focus. Printing in an environment with high levels of oxygen will cause oxidation in the metal powder. Oxidation is a chemical reaction when a molecule, atom or ion loses an electron [14]. Oxidation can cause lower mechanical, thermal and electrical properties and will corrupt the end product [15]. To remove or minimize the oxidation, the system must be in a protective environment with reduced oxygen levels. A way to achieve this is by using a sealed chamber that supports the injection of an inert gas. Inert gases are used to prevent chemical reactions, in this case oxidation [16]. A noble gas, like argon or for example nitrogen can work as an inert gas for this cause.

Nitrogen is number 7 in the periodic table. As the group has learned in the physics and chemistry course, and since it is a neutral atom it is atomic number is equal to the amount of electrons in the atom. An atom has a maximum 2 electrons in the inner shell. This leaves the nitrogen atom with 5 electrons in the outer shell. Atoms “want” eight electrons in their outer shell. Therefore two nitrogen atoms can form a strong triple bond. This means that they are sharing three electrons. This creates the N_2 gas, which is very stable because of the amount of energy which is require to break the triple bond[17]. It also means that it will not react with other atoms in a melting process. SINTEF manufacturing uses nitrogen gas as they have a nitrogen gas generator.

The second gas used in SLM is argon. Argon is the inert gas used in this project. Argon is number 18 in the periodic table and is the third of total seven noble gases. Since it is a noble gas the outer shell is filled with 8 electrons. Therefore argon is non reactive in a melting process, making it the perfect option for SLM when considering oxidation.

The pictures below are taken from an article by Chalmers University of Technology [18]. The article is about the properties of argon and nitrogen gas used as shielding gasses in a laser-powder print bed making parts of stainless steel. The pictures show the materials and powder in two different atmospheres, one with 0.2% O_2 and one with 0.08% O_2 . One can clearly see the oxidation which has occurred at the upper pictures with 0.2% O_2 . The oxidation has caused bobbles and an uneven surface with fractures. This makes the product considerably weaker. The group aims to avoid such a reaction.

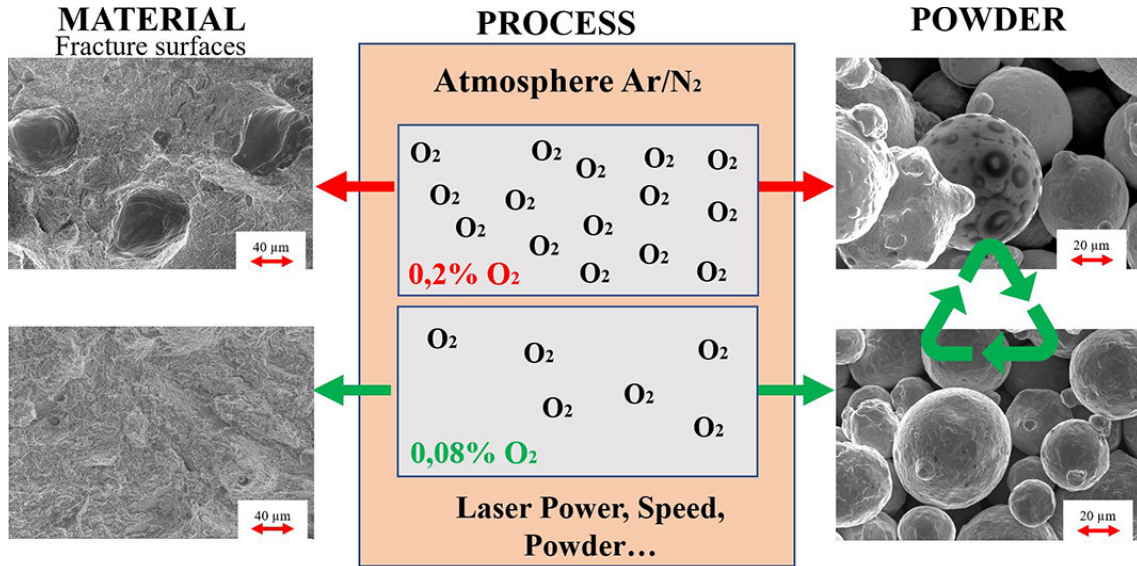


Figure 4: Oxidation with different levels of O_2 [18].

2.1.8 Communication

Serial communication

Serial communication is a type of communication where data is sent and received one bit at a time over one or two transmission lines [19]. The communication is controlled via protocols. An example of a well known serial communication protocol is USB. This protocol is used in the project.

Network

Two computers are connected to a router creating a local area network (LAN). Cat 5 Ethernet cables are used to establish communication between the different devices. Using a LAN has multiple advantages as it has low latency and high stability.

FTP-server

The file transfer protocol makes it possible to share files from one computer to another as long as they are connected to a common network [20]. An FTP-server can be set up on a device. When this is done another device can now connect to the server and these two devices can share files.

2.1.9 Electrical components

In this subsection some of the key electrical components used in the project are described.

Terminal blocks

Terminal blocks are components used to connect together two, or more wires. The most usual layout is that there are multiple individual terminals lined linearly on a strip as shown in the figure to the right.



Figure 5: Example of terminal block.

Relays

Relays are electrical components working as an electrically controlled switch. These can be used to control electrical signals and is perfect to shield components when operating with different voltages.



Figure 6: 24 V relay with DIN rail mount.

Contactor

Contactors are also electrical controlled switches and work generally in the same way as a normal relay. They can both break and make electrical power circuits. The main difference between a contactor and a relay is that the contactor is used on currents over 9A, and on voltages up to 1000 VAC. A relay is used on 10A or less, and on voltages up to 250 VAC. Contactors can also be used on circuits with one or three phases while relays can only be used on one phase circuits.



Figure 7: Siemens Sirius 24 V Contactor.

2.1.10 Python

Python is an object oriented programming language with a simple syntax [21]. This has quickly made python one of the most popular programming languages in the world. The group has used python throughout the project for the mentioned reasons, but also because it is the programming language taught throughout the electrical engineering study-program.

A python script is a collection of commands, which put together is executed as a program [22]. In this project scripts for controlling the actions of hardware, making slices from 3D-models (from one stl-file to several DXF-files) and creating a user interface has been developed. The advantage of writing a python script for a desired task over using existing software is that programs can be modified. This opens the possibility for adding improvements or necessary updates in the scripts allowing them to evolve as the project progresses.

2.2 Equipment

In this subsection the equipment used throughout project is described in detail.

2.2.1 Rofin F30

One of the main components of the system is the Rofin PowerLine F30. This is a fibre pulsed engraving laser.

The main components of the laser system are:

- 19" plug-in for electrical components where the laser beam is generated.
- Laser head that couples the laser beam in the marking head.
- Marking head.
- 19" plug-in PC.
- A monitor, mouse and keyboard. These are now replaced.

19" plug-in for electrical components

This is the component of the laser system where the fiber laser is created and where the main switch is located. The device generates all necessary supply voltages, has an emergency stop relay and also controls and monitors the fiber laser. The fiber laser is first generated in pulses, then amplified by a fiber amplifier before getting transported via a fiber optic cable to the laser head.

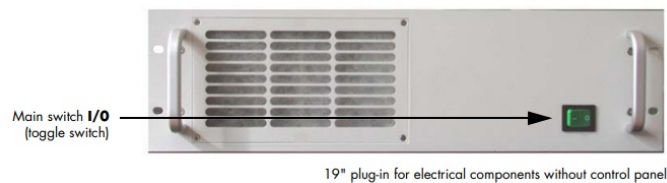


Figure 8: Power unit for the laser [23].

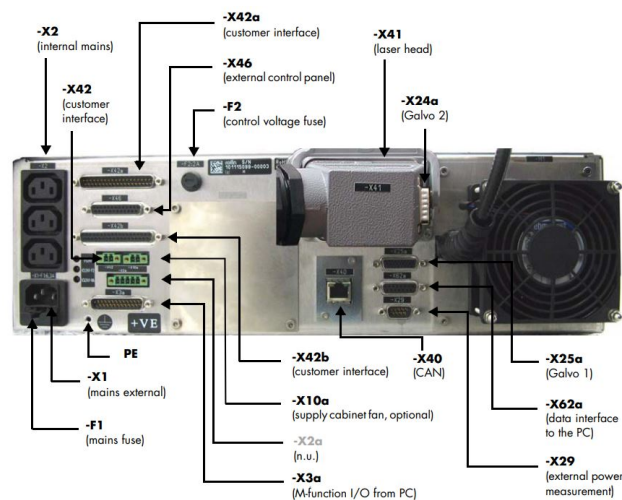


Figure 9: Backside of the power unit for the laser [23].

19" plug-in PC (Rofin PC)

This is the computer that operates the laser system and executes the printing jobs. The PC operates on Windows Embedded and is running the software required for operating the laser system. These software's are LaserConsole and VisualLaser-Marker (VLM) or Visual Marker Controller (VMC2). It also includes a DVD-RW drive and a USB port. The specs are a CPU dual core 2.4GHz, 1 GB RAM and 160 GB HDD of storage.



Figure 10: Front view of the 19" plug-in PC [23].

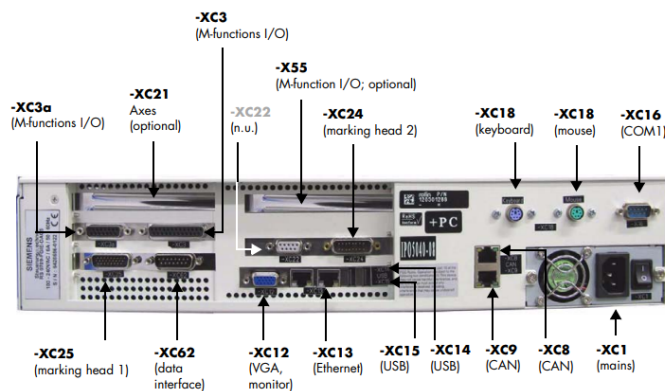
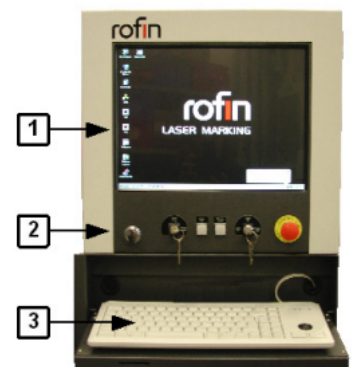


Figure 11: Back view of the 19" plug-in PC [23].

External control panel

The laser system came with an external control panel. The group has removed most of the control panel and moved it to a new location with a new layout and components. The panel marked with number 2 in figure 12 is reused in the new control panel as it is critical when initialising the system. The approach is described later in the document in section 3.3.1. The external control panels original look can be seen in figure 12.



Operating elements of the handling system RT 800

Figure 12: Originally external control panel [23].

Laser head that couples the laser beam in the marking head

This component of the laser system couples the laser beam to the marking head where the laser beam is angled. The laser head also consists of a warning light that lights up in red when the laser system is turned on and flashes if an error has occurred.

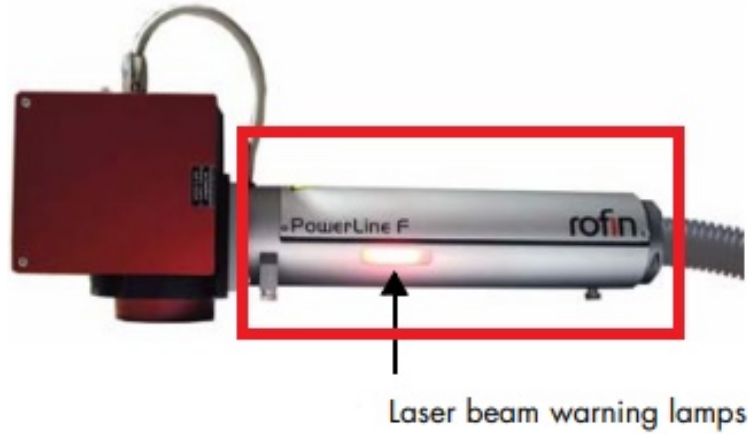


Figure 13: Laser head, marked in the red square [23].

Marking head

This is the part of the system that emits the laser beam. The emitted laser is angled by two mirrors in x and y direction. These mirrors are controlled by the plug-in PC and focuses on the desired spot on the printing area. It also contains a protective glass which is replaceable. This protection prevents dust particles from coming in conflict with the optical unit located in the marking head.

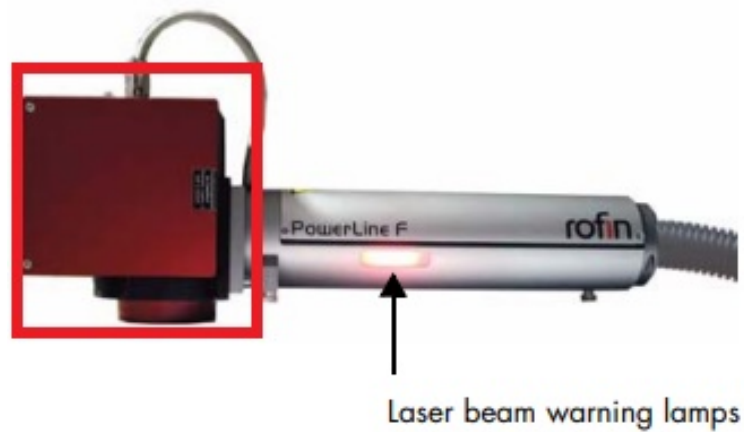


Figure 14: Marking head, marked in the red square [23].

2.2.2 The framing (CombiLine Advanced RT 800)

The name of the framing is CombiLine advanced RT 800 from Rofin. The framing protects the system from outside impact and the user from dangerous light radiation from the laser. The 19" plug-in PC and the control system is located in the space under the work table. The control system consist of a Raspberry Pi and electrical circuits connected together with terminal blocks.

On the work table there was a rotary indexing table, driven by a motor and gear unit. When the table rotated, the object that had been engraved got swapped for one waiting to be engraved. This allowed for continuous engraving. The motor with the gear unit were removed by the group to free up space and reduce weight. The laser itself is mounted in the middle of the enclosure on the work table. It can be raised or lowered along the z-axis to find optimal focus on the workpiece.

There is a slide door on the side of the machine. To ensure safe use there is a switch that triggers if the door is open or closed. If the door is open the laser will not start and any work that is running when the door is being opened is abruptly stopped. The light curtains were mounted on both sides inside the housing in the front. They were mounted to prevent the user from reaching inside while the system was running and to trigger when to rotate the indexing table. If the light curtain was breached while running the laser would stop. The extension arm with the external control panel was placed in the front of the framing. Several modifications has been done to the framing. These are described in chapter 3, "Methods".



Figure 15: The original frame of the system [24].

2.2.3 Powder distribution system (PDS)

The powder distribution system (PDS) was built as a bachelor thesis in 2021 [25] by electrical engineering students. It is built up of three Zaber stages in metal surroundings. Two of the stages (X-VSR20A, [26]) are identical and have functionality similar too pistons. The two X-VSR20A raises and lowers metal cylinders in the two chambers, the powder-chamber and the printing-chamber. The other stage (X-LSM100A, [27]) controls the recoater and moves along a horizontal axis over the top of the table. This stage has a plastic device attached to it and moves the metallic powder from the powder-chamber to the printing-chamber and over the powder deposit-chambers.

The Zaber stages were chosen for their extreme accuracy. The vertical stages smallest step size is $0.09525\ \mu\text{m}$ [28]. The travel range of the vertical stages is 20 mm, meaning 3D-models produced by the printer will be a maximum of 2 cm high. As the printing will require a thin surface to print the 3D-model on, the maximum height will be a little less than 20 mm depending on the thickness of the print plate.

The linear stages smallest step size is $0.047625\ \mu\text{m}$ [29]. This is used to move the recoater. It has a travel range of 101.6 mm, which is long enough to cover the two chambers on the PDS. The maximum travel speed is 26 mm/s. To achieve an even and smooth layer the linear stage will not operate at maximum speed. The recoater must move to the end and back, therefore it will take some time to add one layer of powder. This component started to malfunction as the project came to an end. The group came up with a temporary solution to replace it, described in 3.3.5.

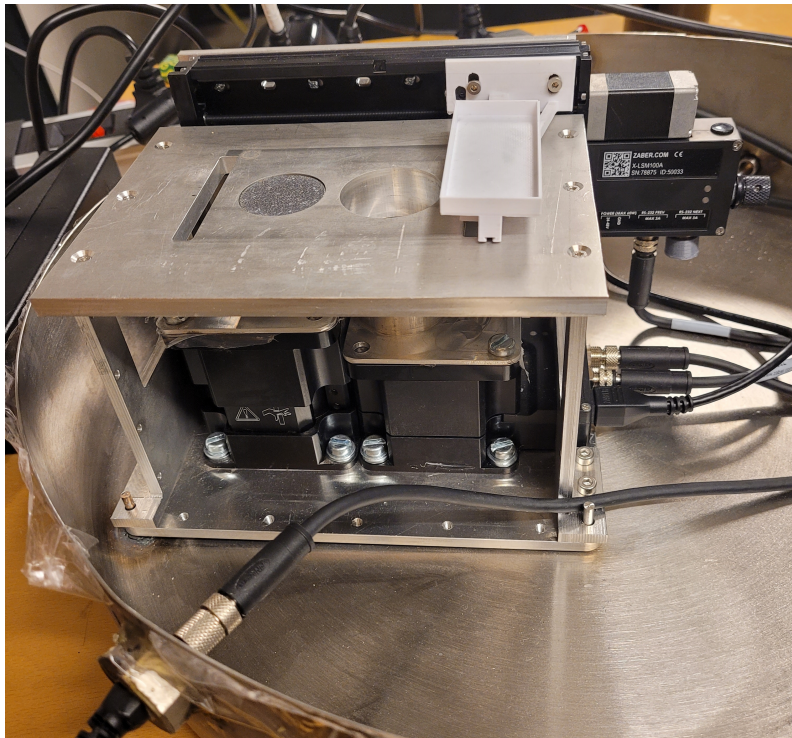


Figure 16: Powder distribution system in the chamber.

2.2.4 Raspberry Pi 3 Model B (RPi)

The Raspberry Pi 3 Model B (RPi) is one of the many models in the Raspberry Pi series. It was released in 2016. Generally, a Raspberry Pi is a small single board computer that can be used for multiple applications and services. The model 3B has a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU and 1 GB RAM. The device is equipped with general-purpose input/output (GPIO) pins. This makes it possible to externally control other electric components.



Figure 17: Raspberry Pi 3 Model B collected from [30].

In addition to these pins, it has:

- Full size HDMI output for the option of visualizing data on an external monitor.
- Ethernet port.
- 4 USB2 ports.
- Micro USB for power supply.
- Micro SD port for loading the operating system and storing data.
- DSI port for connecting touchscreen display.
- CSI port for connecting camera.
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE).

Raspberry Pi OS

Raspberry Pi OS is the installed operating system on the RPi. It is a free operating system based on Debian and is optimized for the RPi. Debian is a Linux distribution. This makes Raspberry Pi OS a Linux based operating system [31].

2.2.5 New control panel

A new control panel has been made. It is made by a HP 27" monitor, a LCD 10.1" touch screen and a new keyboard. The HP monitor and keyboard is used to replace the old control panel showed in figure 12. These were very outdated and challenging to operate. The LCD screen is designed for use with Raspberry Pi. The touch screen makes it easy to control the user interface built on the Raspberry PI.



Figure 18: 10.1 inch LCD screen for Raspberry Pi.

2.2.6 Inspection cam

A Raspberry Pi inspection cam has been installed inside the framing. This has been done so the operator can monitor the process while the laser is running. The framing has no glass or windows. This has been done as a safety measure due to the hazard of laser radiation. Therefore the system needed another way to visually monitor the process directly.



Figure 19: Endoscope for the RPi [32].

2.3 Programs

2.3.1 VisualLaserMarker (VLM)

VisualLaserMarker is a software package installed on the laser system and runs on Microsoft Windows XP Embedded. It is a powerful and flexible tool used for laser marking. Graphical objects such as text, shapes, figures, bitmaps, etc. can be made inside the program on a grid net with accurate positions referenced to the lasers marking position. Different file formats (PLO, Logo, DXF (R13), XML, AI/PS/PDF) can also be directly imported to the program. The different created objects can be grouped into individual marking objects and laser parameters can be set to each of these objects.

VLM is originally used for laser marking but has the option to use scripts and can therefore be instructed to do the same job as a laser printer.

2.3.2 LaserConsole (LC)

The LaserConsole is the user interface on the Rofin computer. It shows which programs are up and running (connected to the laser system). Three of these programs are displayed directly in the LaserConsole and is called LaserDisplay, WebVisu and Configurator. The other programs are for example VMC2 or VLM.

The LaserDisplay displays the majority of events and information connected to the laser. For example statuses, errors, inputs and outputs. Diagnostics is also located in this program and parameters such as z-axis, power, speed and frequency can be adjusted here.

WebVisu is the second program displayed directly in the LaserConsole. This program displays some of the same elements as in LaserDisplay. It contains 5 main windows: Auto mode, Manual mode, System status, Messages and Service.

The Configurator is the program used to declare variables and bits and configure different parameters to the laser and the belonging software. Unlike WebVisu and Laser Display the laser can not be controlled and set in motion via the configurator. It can only configure parameters and settings for the laser and load them up to the RCU for the next use. There are multiple configurator options such as AxesControl, GalvoControl and VLMIOMap.

2.3.3 Autodesk Fusion 360

Autodesk Fusion 360 is a cloud based 3D modelling software for product design [33]. It is used for creating 3D models. By saving the 3D model as an stl-file it can be used in a 3D-printer. Free access to the program by creating a student account through NTNU makes it an easy choice.

2.4 Specifications the laser

The Rofin PowerLine F30 is a fiber laser. The laser beam has a wave length of 1055-1070 nm and delivers a power of 30W [23]. The beam has a diameter of just 7.5 mm, which concentrate the energy in such a way that it can melt metal powder. The laser is not equipped with focusing axis FFM (Fast Focusing Module), meaning the laser must be adjusted manually to the right height for optimal focus. This height is set in the configurator on the Laser Console software by adjusting the z-axis. The two underlying tables show the technical specifications on the laser head and marking head.

Table 1: Technical data on the laser head. Obtained from the Rofin manual [23].

PowerLine F 30 laser head	
Laser Medium	Fiber laser
Wavelength [nm]	1055-1070
Linewidth [FWHM]	<10 nm
Output power, nominal [W]	>28.5
Minimal adjustable output power [W]	3
Pulse frequency [kHz]	30 - 100
Pulse length [ns]	100 ± 20 @ 30 kHz
Power stability [min/max]	<± 2.5 %
Pulse energy [mJ]	0.95 @ 30 kHz
M^2	<2.0
Beam diameter (collimator output) [nm]	7.5 ± 1.5
Beam roundness	>80 %
Beam divergence [mrad]	± 0.3
Weight without marking head [kg]	approx. 5.7
Operating surrounding temperature [°C]	15 - 35

Table 2: Technical data on the marking head. Obtained from the Rofin manual [23].

PowerLine F 30 marking head - 1064 nm	
Max diameter of the beam inside the marking head [mm]	14
Step response (settling to 1/1000 full scale)	
At 1% full scale [ms]	0.40
At 10 % full scale [ms]	1.60
Optimal performance	
Nominal deflection angle [rad]	0.82
Gain error [mrad]	<5
Zero offset [mrad]	<5
Skew [mrad]	<1.5
Non-linearity [mrad]	<2.1
Dynamical performance	
Tracking delay [ms]	0.24
Repeatability [mikrorad]	<22
Long-term drift over 8 hours at operating temp [mrad]	<0.6
Electrical connections	
Max, theoretic range of input values	0 to 65535 increments
Input and output signals	XY2-100 Standard
Supply voltage	plumin (15 + 1.5) V DC, max. 3 A per supply volt.
Mirrors	
Coating	Dielectrical high Performance coating (YAG)
Wavelength [nm]	1064
Max. permissiible power density: pulsed (at 50 ns pulselength) [MW/cm ²):	100
Weight without lens [kg]	Approx. 3
Operating sourounding temeprature [C]	15-35

3 Methods

This section is dedicated to describing the different methods and approaches used to achieve the results. The first two sections are used to explain the group's thoughts behind the main solution and the design process to complete it. After this, the subsections go into technical details on how the solutions are implemented.

3.1 Two solutions

After thoroughly discussing the task and looking through the equipment and its documentation, the group concluded that there were two main paths to a satisfying end product.

The first path was a single computer solution. This would mean using a Linux-based computer system and programming all functionality, both distribution system control and laser control, in python. The PDS would be controlled by a python script and the user interface would be programmed in the tkinter library. The group would have to find a way to communicate with the laser and marking head using G-code and implement this in python. This would be a completely open source solution.

The second option was to reuse the 19" plug-in computer for parts of the system's functionality. The reason for using this computer would be the previously installed software on the computer and their built-in functionality for controlling the laser. The installed application Visual Laser Marker (VLM) is very useful for controlling the laser. Therefore, this solution could make the route to the end product clearer and faster. Controlling the PDS will still be done with a python script.

3.1.1 Pros and cons

The advantage with option one is the freedom that comes with an open source system. A developer can change more or less every aspect of the system by adding, removing, or changing the scripts. Therefore, it is the optimal solution for further development and updates. The big challenge is to control the laser through python. The laser can be operated with g-code, and the python code must somehow be translated to g-code for controlling the laser.

The pros and cons with option two are the opposite of option one's. The built-in application VLM makes it very easy to control the laser, and it is possible to write scripts directly to the laser. The con is the age of the operating system on the computer. This could make it more difficult to create further updates and developments.

The optimization of volume, automating the slice process, sealing the chamber, and controlling the PDS with python scripts was included in both options.

3.1.2 Decision of workpath

After discussing the different options and considering the outcomes within the group, with our mentor from SINTEF and with other SINTEF employees, the group chose to go for option two. The reasoning behind the decision is that the VLM and the Rofin computers already have an implemented solution for communicating with the laser. During the discussion the group did not come up with any clear solutions for controlling the laser with G-code.

Considering the project is on a limited amount of time, and the group already has a lot of new subjects to study, the possibility of not having a finished, satisfactory product in time using method one was significantly larger than with method two. The group was afraid that with choosing option one there was a chance of getting stuck trying to solve the laser control.

3.2 Design process

During this project the group learned that the route to an end product consist of trying and failing at different approaches, until the goals are reached. In the preliminary project the group made a plan of action for the project. Here the groups main goals were set. The main goals were also given sub goals. This was done to create a clearer path to a good end product. The main goals were as follows:

- M1: Build the new framing and acquire a new chamber.
- M2: Build and set up communication between the units, the laser and the PDS, and the user interface (UI).
- M3: Successfully print in 3D.
- M4: Project reporting.

The sub goals were set as stepping stones to reach the main goals. The first sub goal to complete M1 was to dismantle the framing. The group bought tools from Würth for this task. The tools with individual prices are listed in Appendix A. Designing the new framing was an important step before building it. The group made hand drawings and 3D-models for various parts of the framing. These parts were later produced and mounted. The chamber was the last part needed for completing the frame (M1). SINTEF would require a new chamber designed with the properties needed for the 3D-printing.

Building a communication system for the units is one of the major and most challenging parts of the project. Creating a communication system requires a lot of understanding, both in communication protocols and electrical circuits. The most important sub goal for achieving M2 was to make the control scripts. These scripts have to perform to perfection as each script is a part of the whole system. A small error might fault the entire system. Creating a user interface was one of the sub goals as this is where the communication is controlled. The interface is used to start the desired 3D-printing, but also visualize the process of the printing. More in depth data like laser parameters and hatching patterns were also planned under M2.

The third main goal was to print successfully in 3D. This goal requires that M1 and M2 are completed and without errors. The group realized early that a program that could take an STL file and create slices as DXF files with the desired thickness would be a great invention. The printing process should also be automatic from start to finish. Both of these solutions are completed by scripts. The chamber must be purged for oxygen with argon gas before the printing starts.

The last main goal is successful project reporting. The grade from the bachelor thesis is mostly set considering the contents of the report. Therefore is it very important to prioritize working on it. The most important sub goal for M4 was to regularly update the report as the project went forward. This would assure the quality of the report and prevent the group for getting in time trouble close to the deadline. Good project management was also a sub goal. Close communication with SINTEF, regular group meetings, updates, two-week reports and accounting were all important parts in achieving M4.

3.3 Approaches

3.3.1 The framing

One of the criteria for the 3D printer was making the framing smaller. The original framing was unnecessarily large, containing multiple unused physical functionalities. The original idea was to make a new framing from scratch. This would be to make the framing as small as possible.

This solution would be the most convenient considering it would be taking up less space and transporting it would be easier. Despite this, the time used to achieve a completely new framing would not be worth the potential lack of results elsewhere in the project. Thorough planning and documenting all terminal blocks and wires would take a lot of time. The most important goal is to get the system to print metal 3D models, and a small framing does not directly affect this goal. Due to these conclusions the chosen approach became to use the original framing and remove the parts not needed in the final product. This approach still achieves the smaller framing, and the group saves a lot of time.

The external control panel mounted on the outside of the framing was fully removed [figure 21] due to its inconvenient positioning and replaced with a new screen and keyboard in the rear part of the system. On the front side of the printer there was a housing which made room for a rotating indexing table and a safety light curtain system. Both of these features has been removed [figure 70]. The motor and gearbox for the rotating indexing table was also removed. Figure 20 shows the external control panel on the right.

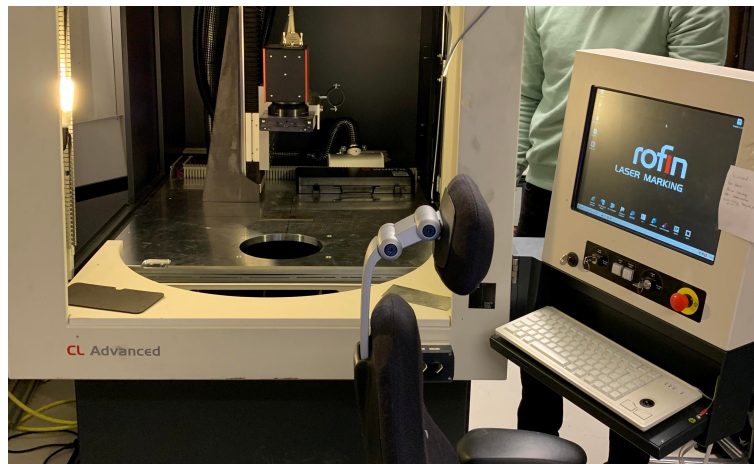


Figure 20: The external control panel.

Safety measures

There were several safety measures built into the old framing. One of these are in connection with the service door. While the door is open a signal prevents the laser from running. This is a necessary safety measure as the light radiation from the laser could be harmful. The doors safety system has been kept as it originally were.

Light curtains were attached on both sides of the framing. These were used to prevent the user from putting their hands in the machine while the rotating indexing table was active. Because the rotating indexing table was removed the light curtains no longer had any use. The light curtains have therefore now been removed.

When the safety light curtain system was removed it caused a safety alarm in the system. This is because the Rofin now received a “low” signal from the light curtains. The Rofin interprets this signal as the curtains being breached, and the laser would not run in this state.

After trial and error the solution to the error was found in the circuit diagrams. The group realised that connecting a 24V signal as an input on port A3:6 would set the light curtain value to “high”. The Rofin interprets this signal as the system being ready to run with no breaches in the light curtain.

In addition to the service door safety function there is a light column at the top of the framing. The column consists of a green, orange and red light. These lights give the operator visual indicators about the state of the system. The red light states that there is an error on the system which can for example be that the service doors has been opened.



Figure 21: External control panel removed.

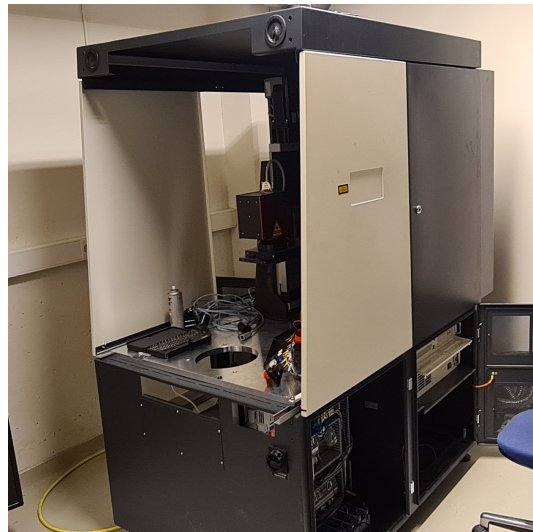


Figure 22: Front housing removed.

New control panel

The old external control panel has been removed and replaced with a new compartment, designed by the group, in the rear part of the framing. This can be done because of the extra space in the back of the framing achieved from moving unnecessary parts. The compartment was created by cutting out a part of the back wall with an angle grinder. The steel plate was then removed and a new part, designed in Fusion 360, was fitted into the hole in the framing.

The new part was designed with measured holes. This is because the group wanted the compartment to be screwed in place instead of being welded. This assessment was made to make unscrewing the compartment in case of maintenance, alterations or adjustments possible. The compartment also has flanges on every outer edge so that there are no sharp edges exposed.

The right side wall of the new compartment was designed with open areas measured to fit the old switch control panel, an LCD screen, a USB port and a cable gland. The switch panel contains switches for emergency stop, auto/manual mode, laser off/laser on/shutter open and two lamps. These are convenient to have in close range of the working space in case of the immediate need of an emergency stop and for monitoring the state of the laser.

The LCD screen has been connected to the Raspberry Pi and displays an user interface (UI) for the execution of the PDS. The UI gives the operator the opportunity to control parts of the printing process and visualize a plot of the sliced 3D model. A live stream of the printing process is also visible on the UI via the inspection camera mounted over the print bed.

The third and second smallest hole is for the USB port. The USB drive containing 3D models STL file is plugged into this port.

The fourth and smallest hole is for the cables to the lights in the ceiling. The complete sketch which were sent to the company responsible for making the two new parts are added in appendix F and G.

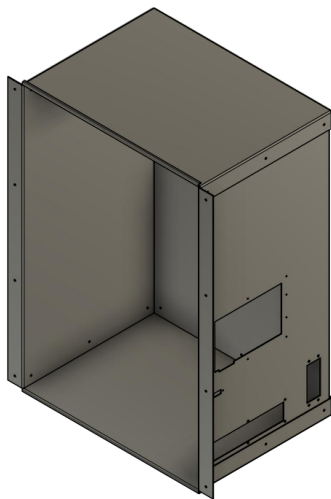


Figure 23: New control room design.

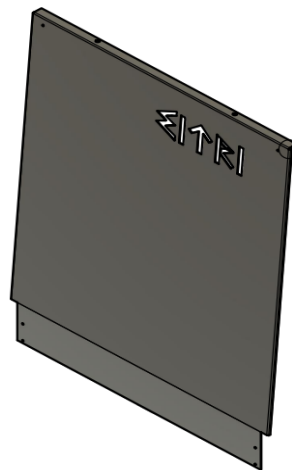


Figure 24: New front design.

Lighting

The lighting inside the framing were initially mounted on the front housing. These lights were removed as they were a part of the front. The group decided to invest in new lighting for the printer. New LED strips with aluminium lists and plastic diffusers were bought. The LED's were mounted by drilling holes in the ceiling, and aluminum lists were attached to the ceiling with nuts and bolts. The LED-strips were glued to the aluminum lists, before the plastic diffusers were latched on to complete the housing around the LED-strips. The plastic diffusers spread the light evenly.

The lights were mounted on 3 sides of the inside roof of the framing, one at the front and one on each of the long sides. The LED strips provide 1570 lumen per meter, with a total of 3 meters of LED strips. The LED's provide great and evenly spread lighting inside for the whole workspace.



Figure 25: New lighting on the ceiling.

The same approach, used in the work space to attach the new LED strips, was used inside of the new control room. Aluminium list with a length of 44 cm was mounted on both sides. This gives excellent lighting to the operator.



Figure 26: New lighting in the control room.

3.3.2 Electrical circuits and terminal blocks

To further develop the functionality of the printer there was a need for new and upgraded electrical circuits and wiring. New electrical cables has been made ready. This has been done by researching the required length of wire and cutting it with side cutter. Both ends are then stripped of isolation. If needed, a ferrule is pressed together on both sides to create the connection points. The cables are now ready to be connected in terminal blocks or directly to the system components.

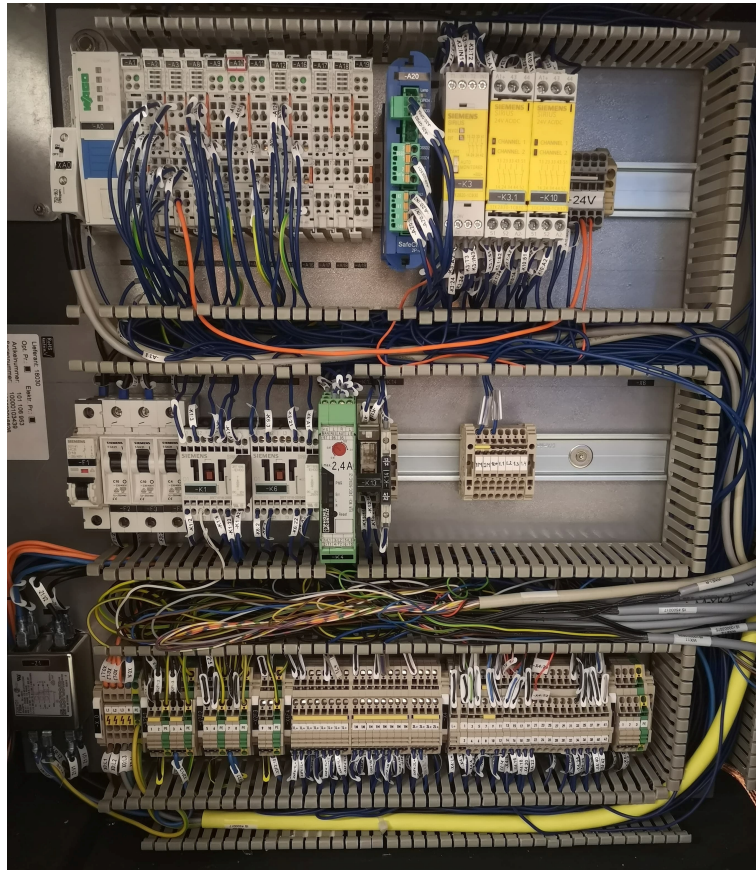


Figure 27: Interfacing between different control units.

Removed components

The group has completely removed three components. The “Cedes safe 400” emitter and receiver has been removed. These were the light curtains that previously were used to control the rotary table and engine. The rotary table and engine has also been removed. In addition, the National Instruments control unit was removed. This unit was previously used as a control unit for triggering the laser to add an additional layer. In this process the group studied the circuit diagrams to understand what wiring could be removed without affecting any other part of the system, and what wire bridges had to be implemented to keep systems in place.

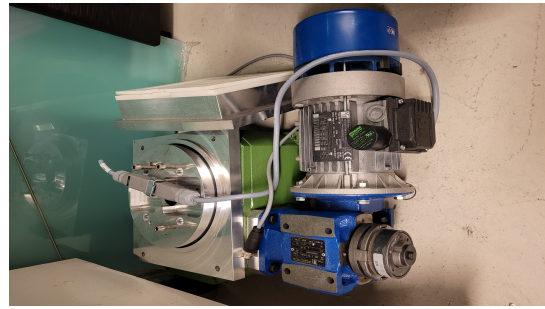
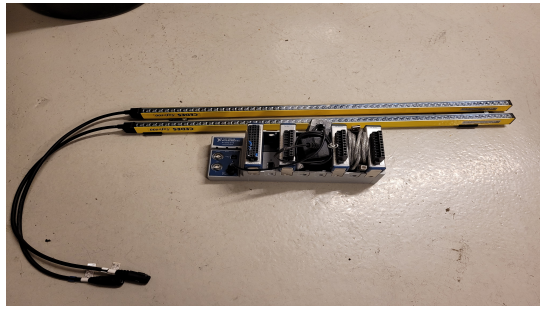


Figure 28: Light curtains and control unit. Figure 29: Rotary table and engine.

Rewiring

When modifying the machine the group removed all of the wiring used by the obsolete hardware. Any new wiring added was added in an equivalent way. Old markings were hidden inside the cable gates and new markings were added at the connecting points. This was done to ensure reversibility of the groups modifications to the highest degree possible, and to easier understand what modifications had been done.

The group used orange wiring when adding new components. This was done to avoid mixing up new wiring with previous wiring. Also the color was deemed appropriate as almost every wire added is to transmit a signal. The group made use of the existing cable canals, but needed to add additional ones as the project developed. The wires previously running through the front of the machine have now been moved and extended so they could be run through the back.

To ensure that there are no modifications done due to uncertainty regarding the extra wiring the group has kept everything structured and organized. Any additional wiring has been labeled according to the circuit diagrams. The circuit diagrams are found in the appendices.

Relay circuits between the laser and the RPi

The two major components in the systems communication, the Rofin PC and the Raspberry Pi, works on different voltage levels. The Raspberry Pi uses a maximum of 3.3V as input/output (IO) and the laser uses 24V as IO. This means that a direct signal from the laser to the RPi will fry the RPi and a direct signal from the RPi to the laser will not be registered as a high input, but as noise. Relay circuits has been integrated to solve this problem.

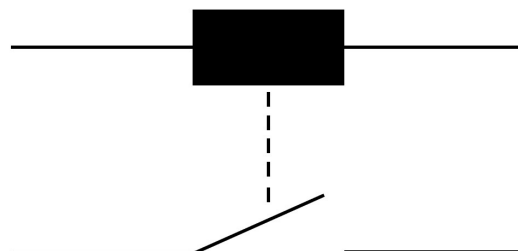


Figure 30: Relay closes a switch on receiving an electrical signal.

A relay is an electrically activated switch. In this system relays allow the RPi and the laser to use its own output signal as an input signal with the switch being controlled by the other device. This means that the RPi has a circuit from an output to an input broken by a switch. This switch is controlled by an output on the laser. When the laser outputs a high signal (24V), the switch closes and the RPi gets a high signal (3.3V) without it being fried. The laser gets a high signal through a similar circuit. Here the laser receives its own 24V output as an input controlled by a switch controlled by the RPi.

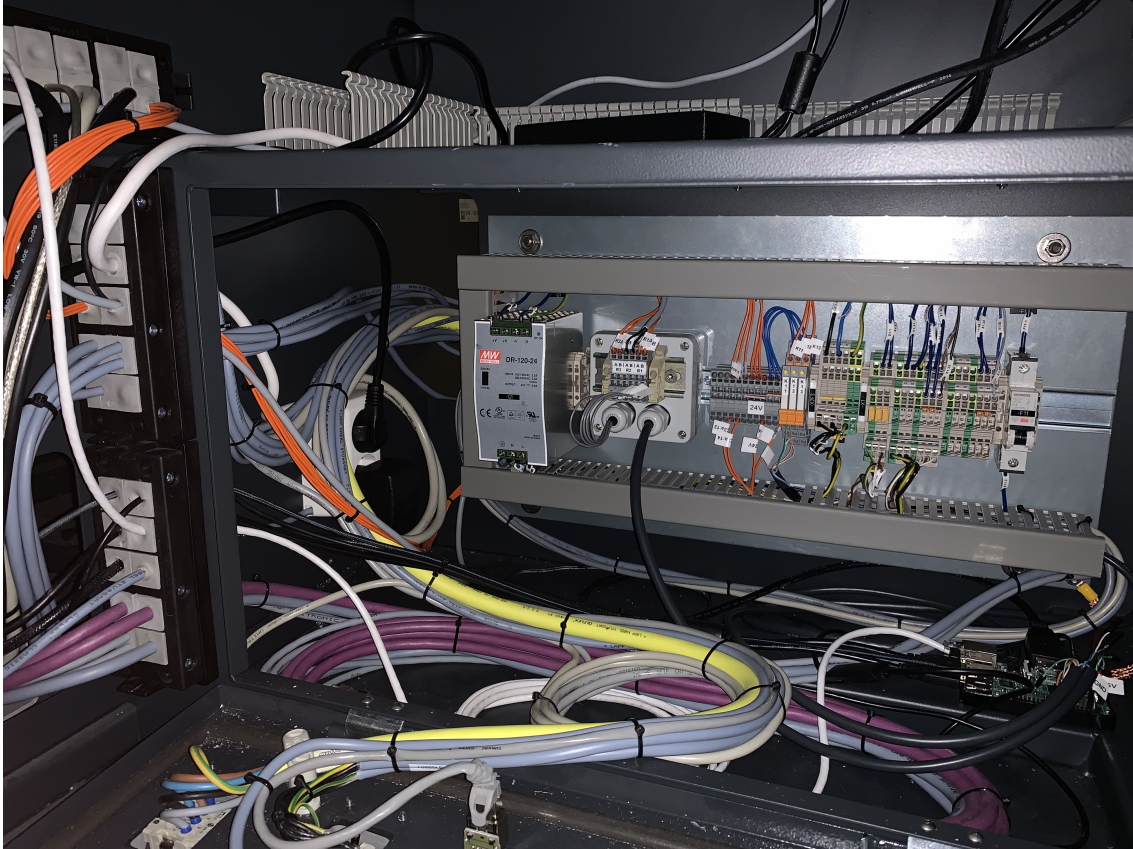


Figure 31: The control system.

The box in the middle of figure 31 contains the relays connected to a circuit board. The RPi is located in the bottom right corner. This system is built by the group.

3.3.3 Printing Chamber

The existing printing chamber was made from a stainless-steel pot with a 40 cm diameter. The chamber consist of two parts with a height of 9 cm with a plexiglass "adapter" between them. Tape was added in the joint as an attempt to further seal the chamber. The lid is also made of plexiglass, with a round hole to put a special made laser glass. This glass lets the laser through.

Originally the plan was that SINTEF would require a new chamber. This due to concern about the chamber not being sealed properly. Not having a fully sealed chamber could cause oxidation during the printing process [2.1.7]. However, due to time limitations the group made an agreement with SINTEF to continue using the old chamber as an temporary solution.

The old chamber will usable with some minor modifications. Argon gas has a higher density than air, with 1.69 kg/m^3 [34]. Air has a density of 1.225 kg/m^3 for normal conditions at 1 atmosphere pressure. This causes the argon gas to gather at the bottom of the chamber and makes it possible to purge out the air. A diffuser stone has been used to ensure that the argon gas is releases with a turbulent flow in all directions. A laminar gas flow in one direction can cause the air and argon to mix and it can take some time for the gases to separate again. By ensuring a turbulent flow the difference in density will cause the argon to slowly fill the chamber. The air will be forced out with minimal blending. This makes sure that the PDS is kept within a protective atmosphere.

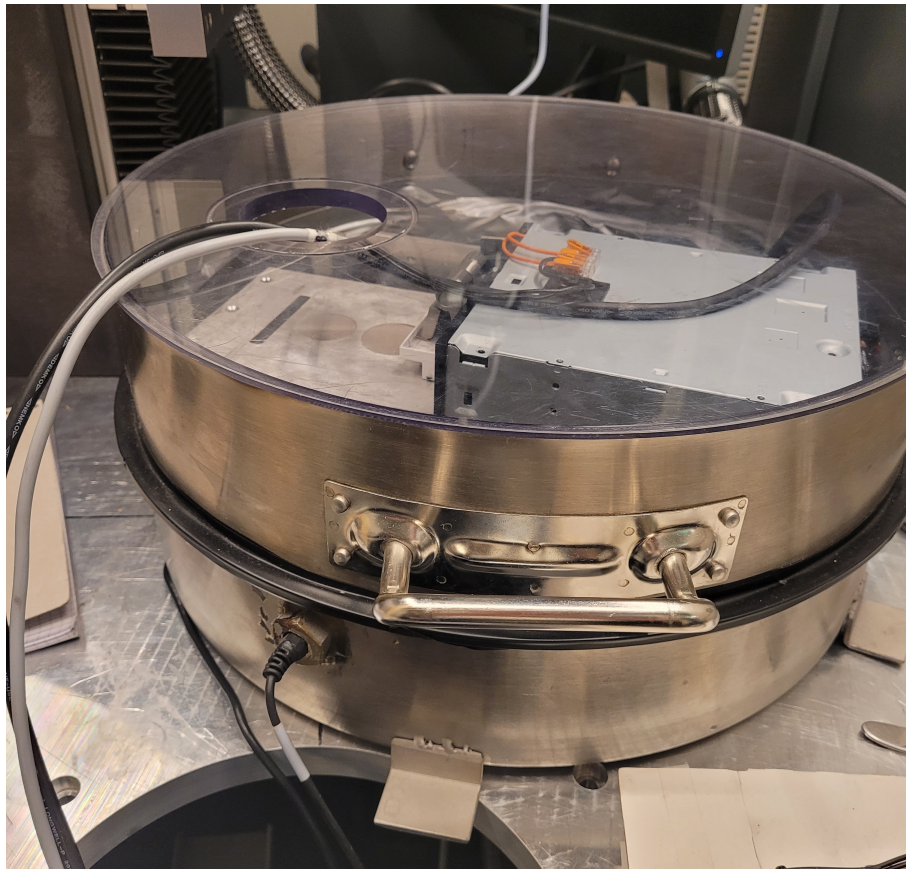


Figure 32: The printing chamber.

Argon gas

As explained in chapter 2.1.7, an unprotected atmosphere with normal levels of oxygen would allow the printing material to react and oxidise, which is undesirable. By filling the chamber with argon gas it is ensured that there is minimal amounts of oxygen and that there will be no reactions in the melting process. This assures a good print result.

3.3.4 New powder overflow-collector

The recoater moves powder from the powder chamber to the printing surface. Any excess material will be pushed down a oblong hole at the end of the PDS-table. From the previous solution of the PDS there was mounted a shelf underneath the hole. A drawer was placed in this shelf and was used to collect excess powder. This solution had a major flaw. The walls on the shelf caused the drawer to be just partly beneath the hole. This caused some of the powder to spill.

To resolve this issue a solution based on the same principle was developed, but with new design and dimensions. The wall causing the spill, located between the drawer and the PDS-table, was removed to ensure that the drawer was located directly beneath the hole. This would make sure that powder would miss. The shelf and drawer was also increased in width and length. The components were designed in Fusion 360, and 3D-printed. When the print was finished the ledge was glued on. This allowed the drawer to slide on perfectly.

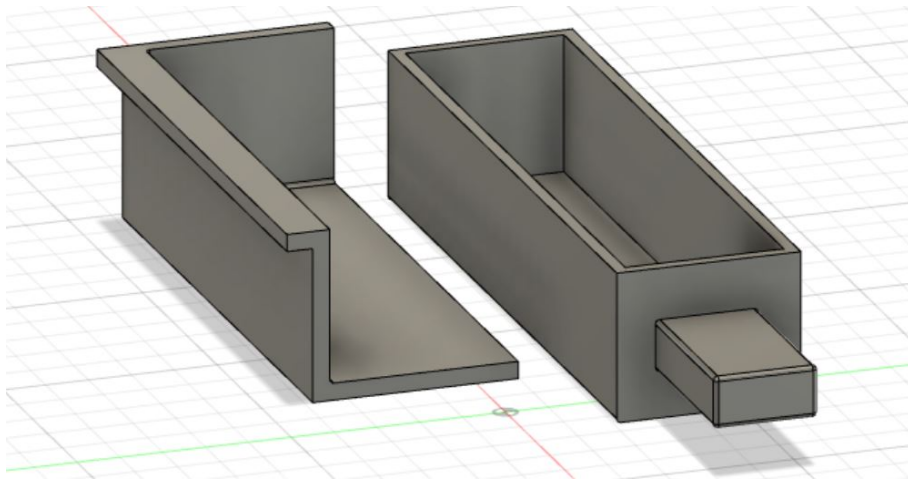


Figure 33: Components designed in Fusion 360.

3.3.5 New recoater

The recoater was, until the last two weeks of the project, moved by a linear Zaber actuator. During a testing of the PDS the linear actuator started to malfunction. When running the actuator it vibrated and made a continuous high pitched noise. When trying to locate the cause of the problem the actuator appeared to get stuck as it would refuse to move at all. By disassembling the actuator, cleaning all parts and applying lubricant the linear actuator was moving again. However, the issues persisted when running at “normal and high speeds”.

The linear actuator is made for extreme precision and not for repeated back and forth motion at max distance and high speeds. This could be the reason it started to malfunction. In testing the actuator used 32 seconds to move from start to end position and back. As the actuator was very slow and malfunctioning the group decide to find a new solution quite quickly with little time left of the project.

The solution was very inventive. It uses an old DVD-player as a linear actuator. The 3D printed end-effector previously mounted on the Zaber actuator has been attached with glue to the edge of the DVD drive. Using a relay soldered to the button of the DVD-player it is possible to simulate the eject-button being pressed and trigger it to open. This results in the powder being distributed evenly.

The new solution is more stable than its predecessor as it is attached at multiple points in the direction of work. The previous solution was an extended arm attached at one side. This caused the end to raise from the PDS-table when mounted to the actuator. In addition, having just one mounting point causes more stress and force on the outermost part of the recoater as the group has learned in physics during the study course. This could cause the recoater to bend or not being swept completely horizontal resulting in an uneven layer of powder. The solution is more stable and straight, it is also much faster.

As mentioned the old linear actuator took 32 seconds to complete a layer. The new recoater only takes 5 seconds. At 0.05 mm per layer, a model with a height of 15 mm demands 300 layers. The old recoater would use more than 2.5 hours in the recoating process. The new recoater does the same job in about 25 minutes. This is about 6 times faster.

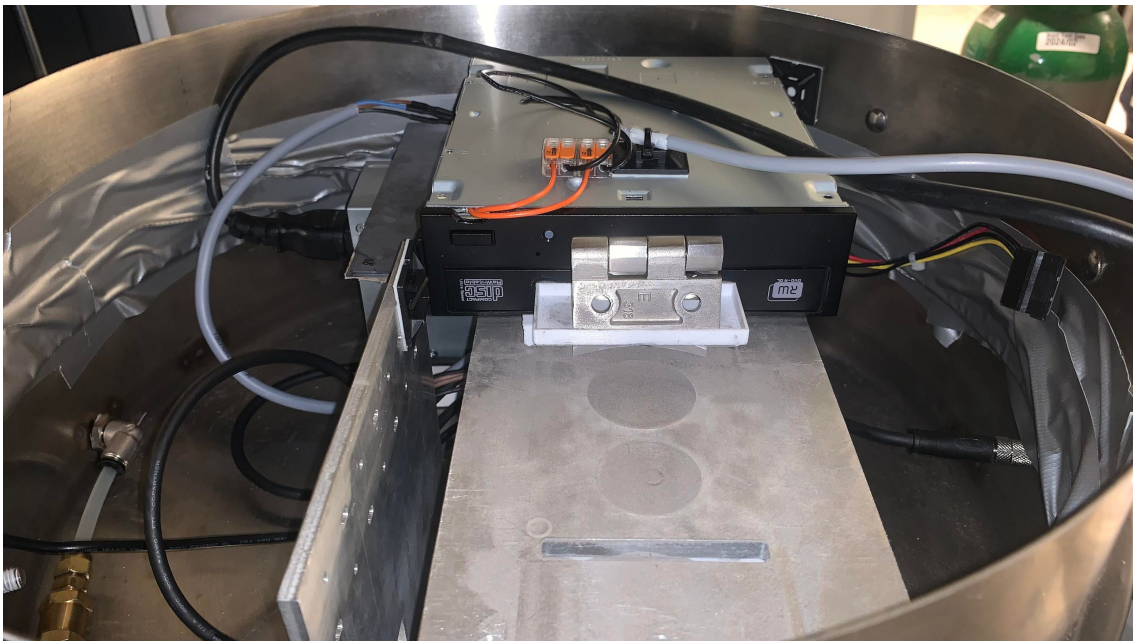


Figure 34: New recoater solution.

3.3.6 File Transfer

A necessary step in the printing process is to upload the slices to the stationary computer. Since it is running an old operating system on dated hardware it is preferred that it only controls the laser. Therefore segmenting the 3D model is done on external hardware. The DXF-files are created on the RPi and then moved to the laser.

The first attempted solution was to upload the files to a memory stick. This did not work as it appears that there are limitations implemented by Rofin. The machine refused to recognise any memory stick but their own.

The next attempt was to set up a local network from a laptop to share files. Yet again this proved to be problematic. This was because of differences in the way old and modern operative systems manage file sharing. In every configuration tested the computers responded to each other while running a network test. Still they refused to share any files.

File Transfer Protocol (FTP) Server

The solution was to set up an FTP-server on the Rofin computer. A FTP server is used for transferring files between devices over a network [35]. After attempting both directions the only functioning solution was to set up the FTP server on the Rofin computer. This setup gives the RPi the opportunity to send files to the Rofin computer by being connected to the same local area network. The network is set up on a router with a static IP configuration on both devices. This ensure a stable file transfer.

When a memory stick is inserted the RPi fetches a STL file and moves it to a folder for it to be manipulated by the slicing script. The files output by the slicing script are transferred to a directory named FTP/files_for_print. This folder is connected to the FTP folder on the Rofin computer. This means that files added to the folder on the RPi are automatically moved to the correct folder on the computer.

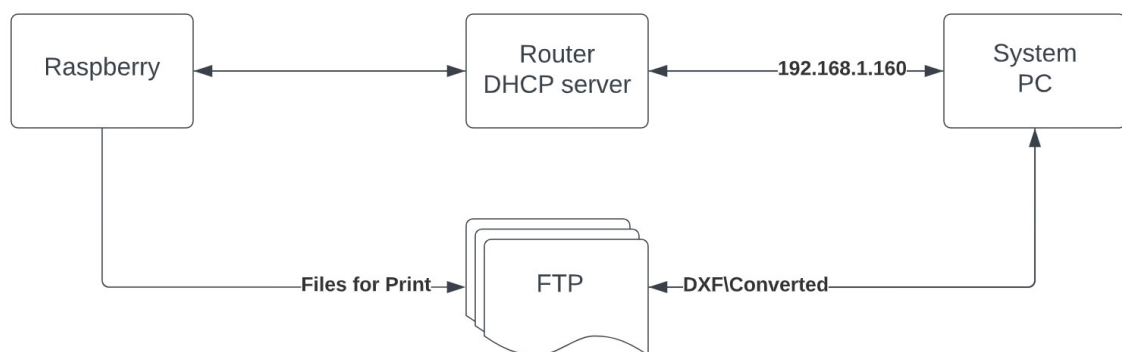


Figure 35: Diagram of the file transfer setup.

Python script

In connection with the FTP server there has been made a python script on the RPi. This script detects when a USB disk has been connected to the RPi and then handles the files used for the printing. The script holds up the process and waits until it detects a USB disk. When a USB disk has been detected the script transfers the files over to the folder in association with the FTP server. When the printing process is completed the script deletes the files from the FTP folder. This is done so that there are no left over files from the last print. This script is attached at the end of the report in the appendices.

3.4 Software development

3.4.1 Slicing script

An essential part of the printing process is to generate the slices from the STL-file. To achieve this the group have made a python script. The script checks the inserted USB drive for a STL-file and retrieves the desired layer thickness from an input in the user interface. The program checks the coordinates of every point in the STL-file and records what the highest value of the z-coordinate was. It then uses this value as the height of the model.

The script now slices through the model in increments defined by the layer thickness set in the UI. At every slice it runs through every facet and checks if the current slices height is within the minimum and maximum z-values of the facet. If that is the case, the script checks which of the three vectors intersect with an x,y plane generated at the current height. The vector are created between the three corners of the facet. It also generates a corresponding normal vector.

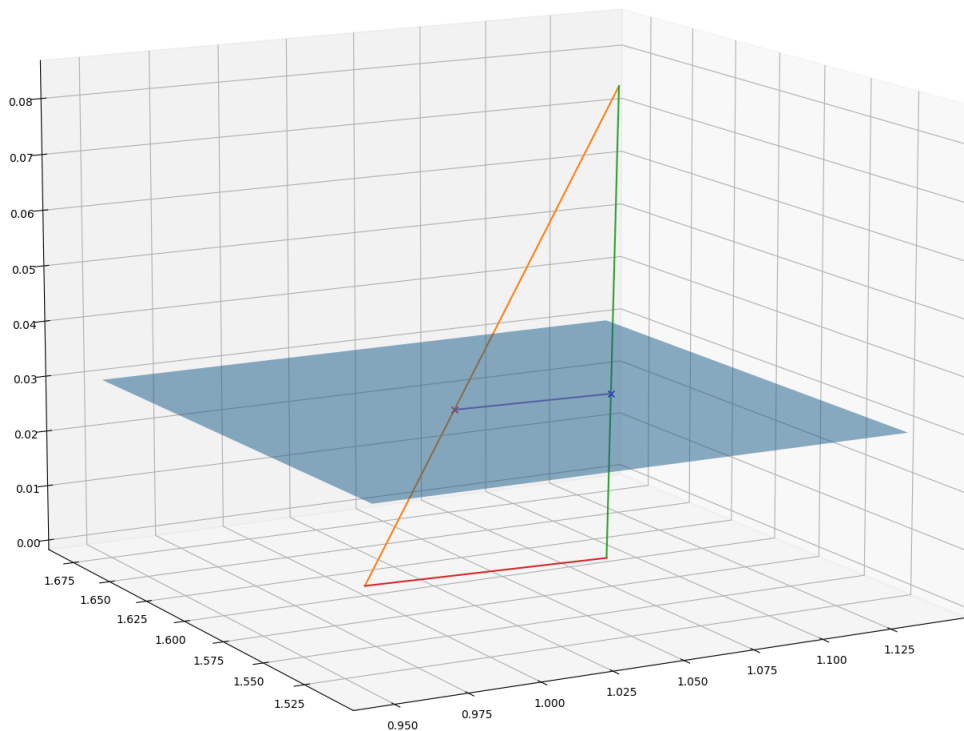


Figure 36: Two points of intersection at the second layer.

The group had to use principles that had been learned in the courses “robotics (IELET20107)” and “linear systems (TTK4225)” to tell which vectors intersect.

A line and a plane can occur in 3 different states:

1. They have no point of intersection. This means that the line is parallel with the plane.
2. They have an endless amount of points of intersection. This means that the line is in the plane itself.
3. They intersect. This means that the line is not parallel with the plane.

One way of checking if the line and the plane is parallel is to use the normal vector of the plane. If the line is perpendicular to the normal vector of the plane it will be parallel with the plane.

To check if they are parallel take the dot product between the facet vector and the normal vector. If the resulting dot product is zero, this confirms that the two vectors are perpendicular. This, in turn, means that the vector is parallel with the plane and they never intersect. Another conclusion to take from the result is that if the dot product is not equal to zero there will be a common point shared between the facet vector and the plane.

This shared point is where the line equation $\{x = x_o + at, y = y_o + bt, z = z_o + ct\}$ satisfies the scalar form of the plane equation $Ax + By + Cz + D = 0$. The general solution can be found by replacing x, y, z in the plane equation. The resulting equation is $A(x_o + a * t) + B(y_o + b * t) + C(z_o + c * t) + D = 0$ where t is a factor of the point between the start and end of the facet vector. If t is $0 \leq t \leq 1$ there is an intersection. If t is < 0.0 then there point of intersection is behind the vector. If t is > 1.0 then the point of intersection is in front of the vector. Neither of these is of any use to us.

The complete process of calculating the intersection:

- Write the equation of the line in its parametric form:

$$\{x = x_o + a * t, y = y_o + b * t, z = z_o + c * t\}$$

- Write the equation of the plane in its scalar form:

$$Ax + By + Cz + D = 0$$

- Use x, y, z corresponding parametric equations to rewrite the scalar equation of the plane. This leaves a single-variable equation, that can now be solved for t .
- Substitute t back into the parametric equations to find the x, y, z components of the intersection.

First define a line L by using a point P and a directional vector \vec{V}_d defined by point

$$P_0 = (x_o, y_o, z_o) = (1.085, 1.588, 0.085)$$

and point

$$P_1 = (x_1, y_1, z_1) = (1, 1.6, 0)$$

. P_0 and P_1 are the top and the bottom left corner of the facet in figure 36.

$$\vec{V}_d[a, b, c] = P_1 - P_0 => [-0.0845, 0.0121, -0.0854]$$

$$x = x_o + a * t$$

$$x = 1.085 - 0.085 * t$$

$$y = y_0 + b * t$$
$$y = 1.588 + 0.012 * t$$

$$z = z_0 + c * t$$
$$z = 0.085 - 0.085 * t$$

Then write the equation of a plane in it's scalar form using a known point on the plane combined with the normal vector:

$$P_p(x_p, y_p, z_p) = (1, 1, 0.03)$$
$$n = [A, B, C] = [0, 0, 1.03]$$
$$A(x - x_p) + B(y - y_p) + C(z - z_p) = 0$$
$$0(x - 1) + 0(y - 1) + 1.03(z - 0.03) = 0$$
$$1.03(z - 0.03) = 0$$

Now solve for t by rewriting the scalar equation for the plane combined with the parametric equation from the line.

$$0(x - 1) + 0(y - 1) + 1.03(z - 0.03) = 0$$
$$0(1.085 - 0.085 * t - 1) + 0(1.588 + 0.012 * t - 1) + 1.03(0.085 - 0.085 * t - 0.03) = 0$$
$$1.03(0.085 - 0.085 * t - 0.03) = 0$$
$$0.05665 - 0.08755t = 0$$
$$t = \frac{0.05665}{0.08755} = 0.06471$$

Now it is possible to substitute t back into the parametric equations to find the x, y, z components of the intersection.

$$x = x_0 + a * t$$
$$x = 1.085 - 0.085 * t$$
$$x = 1.085 - 0.085 * 0.06471 = 1.03$$

$$y = y_0 + b * t$$
$$y = 1.588 + 0.012 * t$$
$$y = 1.588 + 0.012 * 0.06471 = 1.59$$

$$z = z_0 + c * t$$

$$z = 0.085 - 0.085 * t$$

$$z = 0.085 - 0.085 * 0.06471 = 0.03$$

The line and plane intersects at $P(1.03, 1.59, 0.03)$. This process is repeated for all three vectors in a facet and every point of intersection is added to a list.

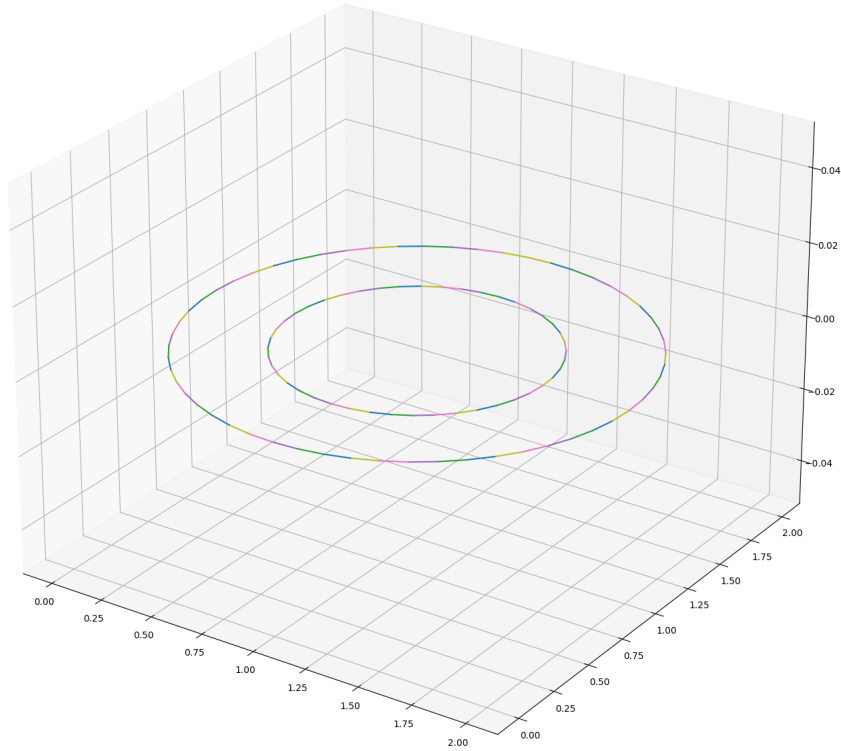


Figure 37: The outline created by drawing a polyline between all of the points.

This is done for every layer throughout the whole model and the slices are exported as DXF-files.

When the script has run through all of the facets it will have a complete list of all the points of intersection. This list is then used to create a polygonal chain also known as a polyline as shown in figure 37. A polyline is a series of connecting lines. This is done to allow VLM to recognise what lines are a part of which outline and enables it to use logic on how to properly create the hatch pattern of the layer.

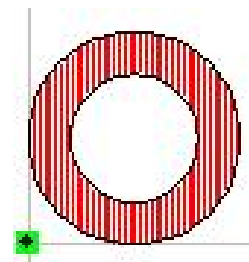


Figure 38: The outline hatched within VLM.

3.4.2 Visual Laser Marker (VLM)

After deciding to go with the selected solution the software installed on the existing computer plays an important role in the systems functionality.

VLM, as previously mentioned in subsection 2.3.1 is a program installed on the Rofin computer and is used for the laser marking process. It has a scripting environment with a variety of built in functions that enabled the group to control the laser in a desired way. The program does not have built in functions for SLM and SLS as the lasers intended use is for marking and engraving, not 3D printing.

The principle behind SLM is to melt metallic powder in layers. The slices determine the pattern which is going to be melted. The 2D drawings of the pattern is acquired from the outline created at relevant cross sections of the 3D model. Since the laser can handle 2D formats it is possible to import each slice and create a marking job in VLM.

There are multiple file formats supporting advanced 2D models that are also executable as marking objects in VLM. The first one is the original VLM file formats. These files are created directly in VLM which is not an open standard. Therefore it would be difficult for a python script to convert a STL-file into this format. The second option is to use the bitmap format (.BMP). Bitmaps are capable of storing 2D digital images. This approach is possible. However, when converting a STL model into a digital image all information regarding size is lost, making it unusable when the system depends on high accuracy between every layer created.

The DXF file format was the third option and proved to be the solution. Since it is an open standard possible to decipher just by reading through it as a text file. After coming to this conclusion the group needed to make a python script able to slice an STL-file and convert each slice into DXF format. This script is described in detail in subsection 3.4.1.

With this script developed the DXF-files are ready to be used in the VLM software. The program is as previously mentioned only designed for marking and engraving. Therefore it does not originally support the execution of a sequence of layers with varying 2D models.

For this problem the software has an opportunity to write scripts in a programming language similar to Microsoft visual basic. This functionality is used to develop a script allowing the laser to accomplish building 3D models.

For the process to be fully automated, the program would need to fetch a new DXF file for each slice. This is because the software is limited to only handle one drawing at once. The built in function “import.DXF” fetches a file from a specified folder by its numerical value in ascending order. The following pictures are taken with mobile camera. This is because it was not possible to achieve a good quality print screen on any file formats from the old computer.

```
dxfiImport.value = cstr(i) & ".dxf"
```

Figure 39: Function to define what DXF file is imported.

```

LMOSInPortConstants
Const FinishedPowderDistrubution = 10
Const StartPowderDistrubution = 14

Dim state
Sub LaserMarker_ScriptBegin ()
  Dim dxflImport,hs
  Set dxflImport = lasermarker.drawing.getmosbyname ("MO_DXF_IMPORT").item(1)

```

Figure 40: Function to import DXF file (dxflImport).

The file path of the specified folder that the function collects the files from is declared in the "Import/export" tab in the Editor settings in VLM shown in figure 41.

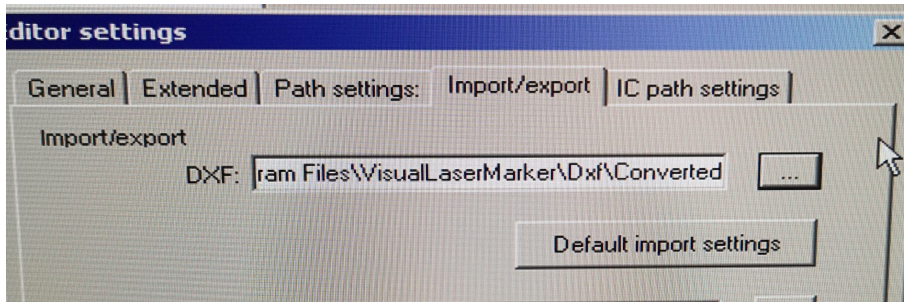


Figure 41: Declared file path for the folder that the "import.DXF"-function.

The folders location is now C: → Program Files → VisualLaserMarker → Dxf → Converted, as shown in the figure below:

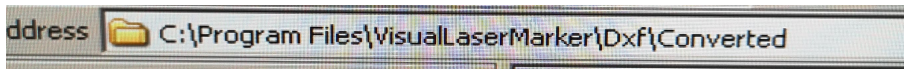


Figure 42: Location of the folder.

For all the DXF-files to be executed in the correct order they are named in ascending order, 1.dxf, 2.dxf, 3.dxf... n.dxf. The figure below shows an example of what the folder could look like when a model is divided into 10 slices.

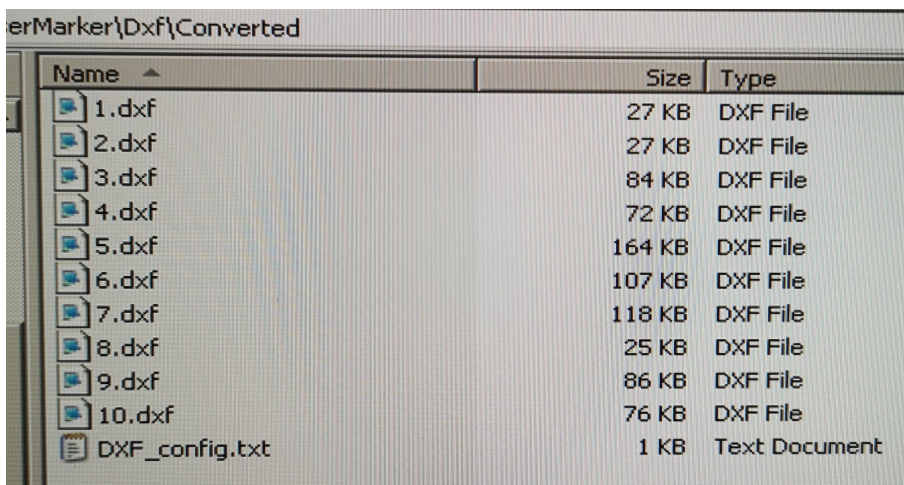


Figure 43: Example of the DXF files structured in the "Converted"- folder.

Because “while” loops are not supported in the VLM software the developed script uses a “for” loop iterating from 1 to 5000. This is enough iterations for any reasonable print. In the loop the current number is used as the DXF file name. The files are therefore fetched from the directory in the correct order. The script will run until there are no more files left in the directory.

Hatching

The script that converts the STL file to DXF files [3.4.1] only creates the outline of each slice. To print, the laser needs to melt the entire area inside the outlines. To accomplish this there is a function in VLM called hatching. With this option enabled, the figure in the DXF gets filled with a large number of lines based on the specified line width. These lines also have a certain direction, and can overlap if wanted. In the scripting environment there are functions that can import the desired settings as well as change the angle of the hatching lines.

```
Set hs = dxfImport.HatchSettings
hs.HatchAngle = 0
```

Figure 44: Function that imports hatching settings for the DXF files.

To avoid fracture, the lines should not go in the same direction for each layer. This will cause weaknesses in the product. To get a more solid structure, the hatching angle is incremented with 67 degrees for each layer.

```
hs.HatchAngle = hs.HatchAngle + 67
```

Figure 45: Incrementing hatch angle.

Communication

For the system to run automatically the units have to communicate with each other. For this task there is a port (XC3a) on the Rofin PC available for external communication and GPIO pins on the RPi. VLM can communicate with XC3a via Machine-functions (M-functions) in the scripting environment.

M-functions

The M-functions used for the communication between the units are declared and uploaded in the Configurator on the Rofin computer. ”StartPowderDistrubution” is set as an output to bit 14 which is pin nr. 13 on the XC3a port. ”FinishedPowderDistrubution” is set as an input to bit 10 which is pin nr. 1 on the XC3a port. This can be seen in appendix C.

Attribute (bit)	Value
General	
bit	14
device	ALIIO
enabled	<input checked="" type="checkbox"/>
name	StartPowderDistrubution
port	Output

Figure 46: Sends signal for to the PDS to start.

Attribute (bit)	Value
General	
bit	10
device	ALIIO
enabled	<input checked="" type="checkbox"/>
name	FinishedPowderDistrubution
port	Input

Figure 47: Receives signal that PDS is finished.

Before the script can fetch a new file, the PDS needs information that the laser is finished marking to start distributing a new layer of powder. The LaserMarker function "WriteIOBit" writes the M-function "StartPowderDistrubution" to True. The PDS receives this signal via the GPIO pins and starts to distribute a new layer of powder. While the PDS is running, the function "WaitonIOBit" waits for the M-function "FinishedPowderDistrubution" to turn True. When the PDS is finished distributing the new layer of powder, VLM uses the function "ReadIOBit" to read if the M-function "FinishedPowderDistrubution" has turned True.

```
LaserMarker.WriteIOBit "StartPowderDistrubution", 1
LaserMarker.WaitOnIOBit "FinishedPowderDistrubution", 1, 5000
Lasermarker.ReadIOBit "FinishedPowderDistrubution", state
```

Figure 48: Write, WaitOn and Read IOBit for communication between the units.

If the state of "FinishedPowderDistrubution" is "True", VLM imports the next DXF file from the directory and starts marking that file. This sequence will continue to loop until there are no more files left in the directory to fetch and the 3D model is complete. The complete VLM script is in the appendices.

3.4.3 The powder distribution system

USB to serial communication

The three stages described in subsection 2.2.3 communicates with the Raspberry Pi (RPi) through Universal Serial Bus (USB) to RS232 signals. On the RPi side a python script controls what is sent to the stages, and on the stages side the data is received and executed on. The Zaber stages uses Zaber's ASCII protocol [36] for the devices settings and commands.

The Zaber protocol exists in two versions, the Zaber ASCII protocol and the Zaber binary protocol. In this project the ASCII protocol [36] was used because it is the default protocol for the Zaber devices and because it is a requirement for a python library described later. Zaber devices listen to commands sent through the serial port and replies immediately. The protocol consists of the commands used to communicate with Zaber devices. These commands always start with a "/" and ends with a newline. The devices answers starts with an "@" and ends with a newline.

```
/1 move abs 10000d
@01 0 OK BUSY -- 0
```

Figure 49: Example of communication between PC and a Zaber device from Zaber ASCII protocol [36].

The libraries

The python script uses two libraries to control and communicate with outside hardware. These two libraries are Zabers zaber motion library [37] and a library called RPi.GPIO [38] to control the RPi's GPIO-pins. The full code is added as an appendix in this report [Appendix I].

The zaber motion library is simple and well documented on Zabers website [37]. Zaber motion provides the possibility to communicate directly with the stages with simple, built in commands. The library requires the connected devices to use the ASCII protocol. With zaber motion implemented, the python script sends data through the commands shown and described in the table below.

Table 3: Zaber_motion commands. [37]

Function	Functionality
axis.move_absolute	Moves the acquired axis to an absolute position.
get_axis	Gets an Axis class instance. Allowing control over an axis.
axis.home	Returns all axes to their homing positions.
axis.move_max/min	Moves the axis to the maximum/minimum position.
axis.get_position	Returns current axis position.
axis.move_velocity	Begins to move axis at specified speed.
library.set_log_output	Sets library logging output.

The RPi provides the system with important functionality through its GPIO-pins (General-Purpose Input Output-pins), see figure 50. These pins are digital signal

pins on the RPi's circuit board and may be used as both input- and output-pins. The GPIO pins are a way for the Raspberry Pi to control and monitor the outside world, in this case outside hardware such as the laser and the powder distribution table. This is done by integrating the pins in electronic circuits [39].



Figure 50: Overview of the Raspberry Pi's GPIO-pins.

The GPIO-pins can be controlled through a python library called RPi.GPIO [38]. This library makes it possible to read and write to the GPIO-pins on the RPi and has been integrated into the python control script. By doing this the powder distribution table and the laser can communicate through the RPi. It is crucial that the powder distribution table and the laser communicates. This is so that the process can work in the correct order and nothing goes wrong during the printing process. Small misses in the timings will lead to a corrupted end product, and potentially damage the equipment.

Example of communication (figure 71):

- Laser says that the table can do its job.
- The table lifts and lowers the chambers then moves the powder from one chamber to the other with the recoater and then back to nominal position.
- The table tells the laser "I am done with my job, you can do yours".
- The laser does it's job.

The code

The script used to control the distribution table is called `driving_the_stages.py`. The script uses the libraries discussed above to send messages using the ASCII-protocol to the stages and to give outputs and collect input through the GPIO-pins. This script is called in the main program when the user pushes the run-button in the UI and is fully automated.

The python script also produces a logging file named `communication_log.txt`. This file is produced through the function `Library.set_log_output` and it logs the data messages sent from the computer to the stages and the stages' answers.

```
≡ communication_log.txt
1 2022/03/31 14:44:48.535231 TX Line: /1 1 25 move abs 146982:6C
2 2022/03/31 14:44:48.549700 RX Line: @01 1 25 OK BUSY -- 0
3 2022/03/31 14:44:48.549700 TX Line: /1 1 26:F6
4 2022/03/31 14:44:48.565656 RX Line: @01 1 26 OK IDLE -- 0
5 2022/03/31 14:44:48.565656 TX Line: /2 1 27 home:2B
6 2022/03/31 14:44:48.581613 RX Line: @02 1 27 OK BUSY -- 0
```

Figure 51: Snippet collected from the logging file, communication_log.txt. This is the same communication as can be seen in figure 49 taken directly from Zabers ASCII protocol.

Approach

The PDS is a very important part of the system. The group therefore wanted to achieve its functionality early on in the project. The first attempt at communication between the computer and the PDS was with the USB protocol. The group decided to use libraries called pyusb [40] and libusb [41]. These libraries are pretty intricate, with little documentation and requires a large amount of knowledge of the USB protocol. Using this library the group achieved little to no progress. At first it seemed the group had missed a part of the communication. For example a communication setup, like the three-way-handshake for the TCP-protocol [42].

After this attempt lead nowhere the group started to research other possible libraries and found a library made by Zaber for use with Zaber devices. This library worked almost immediately and the group got control over the PDS-table.

During further testing and development of the distribution table an error was found. The Zaber devices moves on an absolute scale where the nominal position, the starting position, equals zero. If the device is not in this nominal position when the devices are turned on they might get desynchronized and move according to the wrong nominal position. This was fixed by always starting with an initialization-process and is why initializing is a obligatory step in the manual 3.5 before running the printing process. The initialization process is done by setting all devices back to their nominal position and then moving them to their assigned “home” position. The devices now move according the correct nominal position.

3.4.4 Graphical User Interface

The Graphical User Interface (GUI) is made in python with the library tkinter [43]. Tkinter is the built in python module to make GUI's. The interface created in this project controls all functionality on the powder distribution table, sets variables, shows a live video feed and a preview of the sliced figure and shows information about the execution. It is built on the RPi and interacts through a touchscreen.

Design

The design of the GUI is made for simple interaction. It is built up by boxes with each box being clearly separated. The boxes are set up underneath each other for simple viewing. They are called information, parameters, progress, files, preview and buttons.

The first box is an information box. It contains a step by step guide to start the process. This is done for the user to have a clear overview of what needs to be done in order to be ready for running the 3D printer.

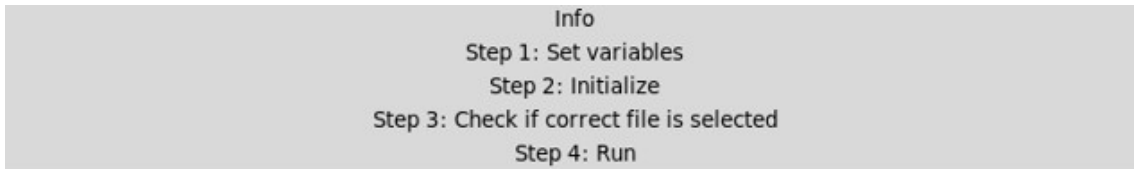


Figure 52: The info box in the GUI.

The next box is called “Parameters”. In this box the user sets the necessary variables to a desired value. This is done by writing the desired value in the text box. As the GUI is shown on a touch screen it has no connected keyboard. Because of this, when the user clicks to edit the parameter a number keyboard opens up on the screen, seen in figure 54. Here the user can type in the desired value. The set value is clearly shown on the right side of the box and is set as the user presses the “Done” button.



Figure 53: The parameters-box in the GUI.

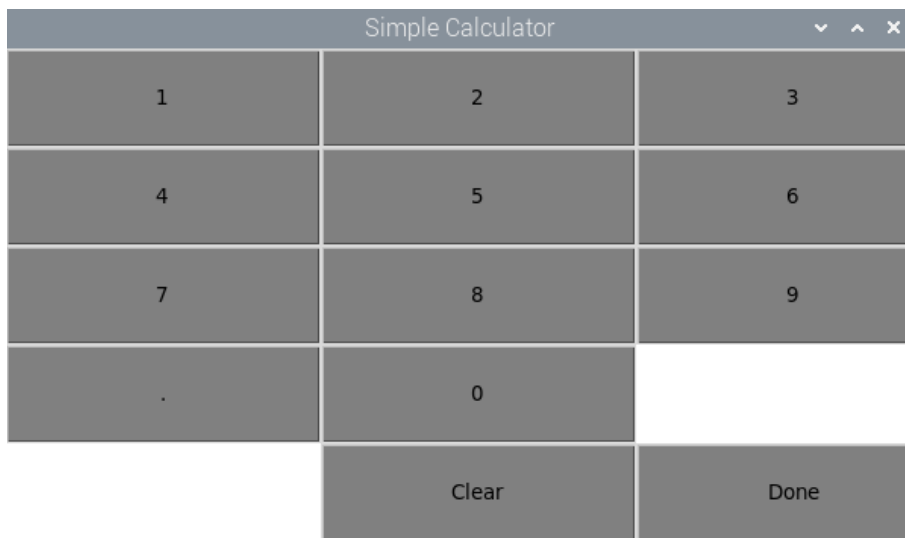


Figure 54: The pop up keyboard.

The progress box of the GUI is a text screen. This screen is updated with information about progress and instructions for moving forward as the user interacts with the system.

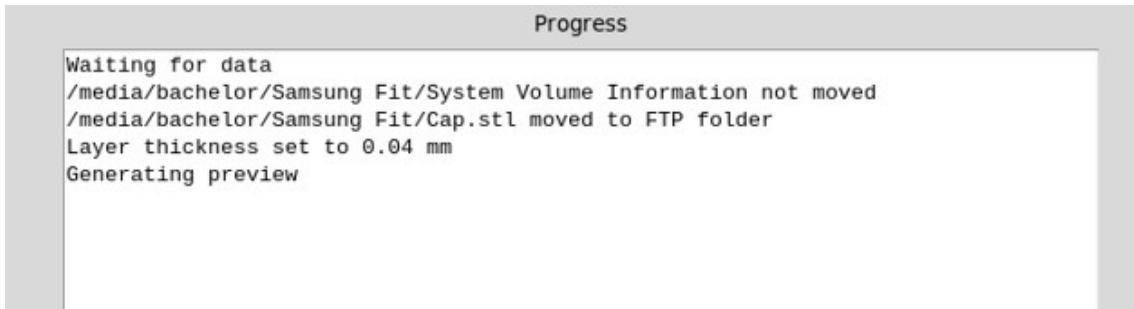


Figure 55: The progress-box in the GUI.

The next two boxes are “Preview” and “Files”. Both of these contain a single button. When the “Preview” button is pressed a popup window with a pyplot of the figure is shown. When the “Convert file” button is pressed the progress text box is updated with a progress-bar. The bar indicates how far the slicing-process has come. There is also a percentage indicating the same thing.

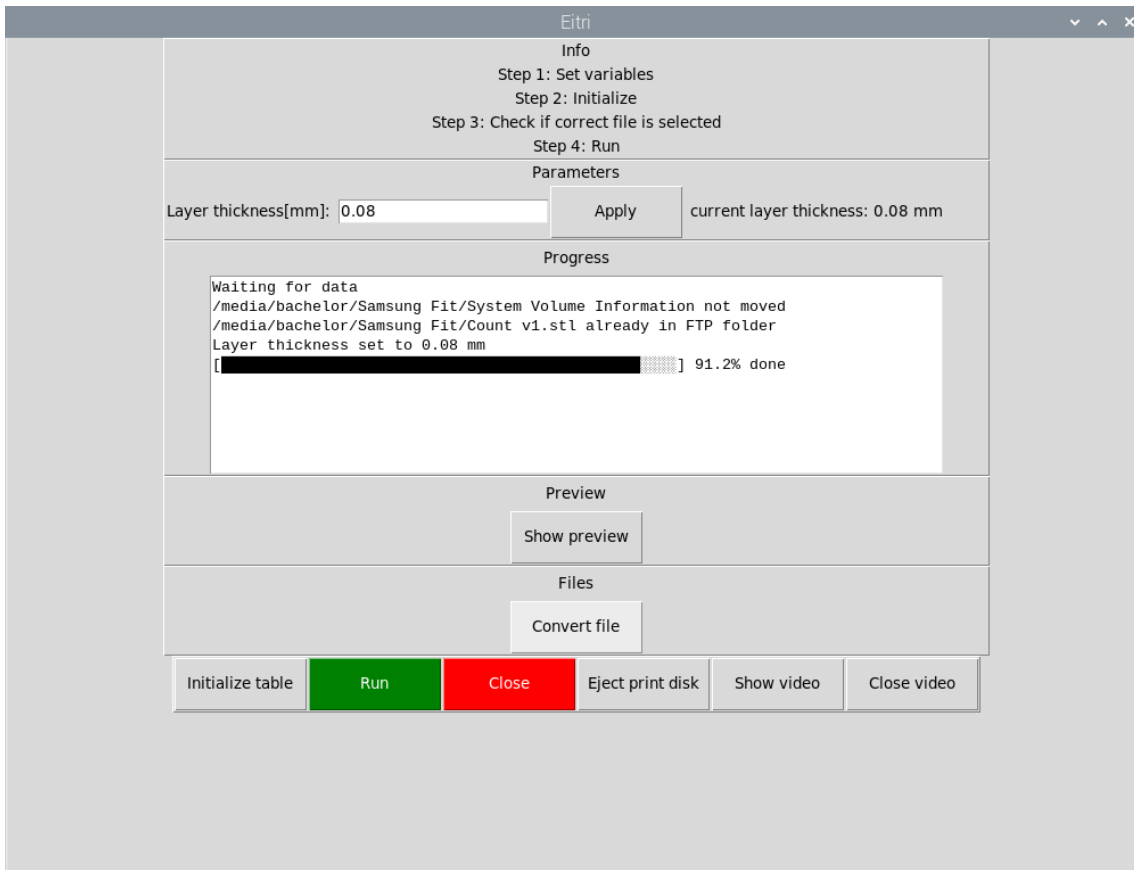


Figure 56: The text box with progress bar.

The last part of the GUI is four buttons. The buttons say “Initialize table”, “Run”, “Close”, “Eject Print Disk”, “Show video” and “Close video”. The design also uses simple color coding to improve user experience. The “Run” button is colored green and the “Close” button is colored red.

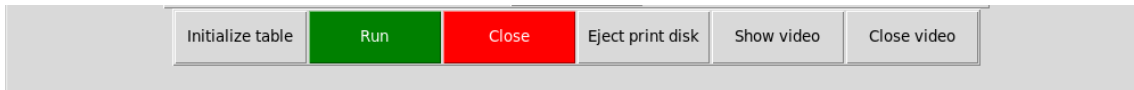


Figure 57: Buttons box.

Functionality

The GUI controls the initialization and the start of the powder distribution table through the buttons named “Initialize” and “Run”. These buttons trigger functions in the python script “drive_the_stages.py”. This is the script made for functionality, such as movement, on the powder distribution table.

The “Preview” button in the GUI shows a preview of the figure from the selected file. Behind this button is a python script called “Butcher.py”. The butcher-script slices the figures outline with the desired layer thickness and plots the result, as can be seen in figure 58. In this figure a 3D-model of a hollow cap has been sliced up and plotted layer by layer.

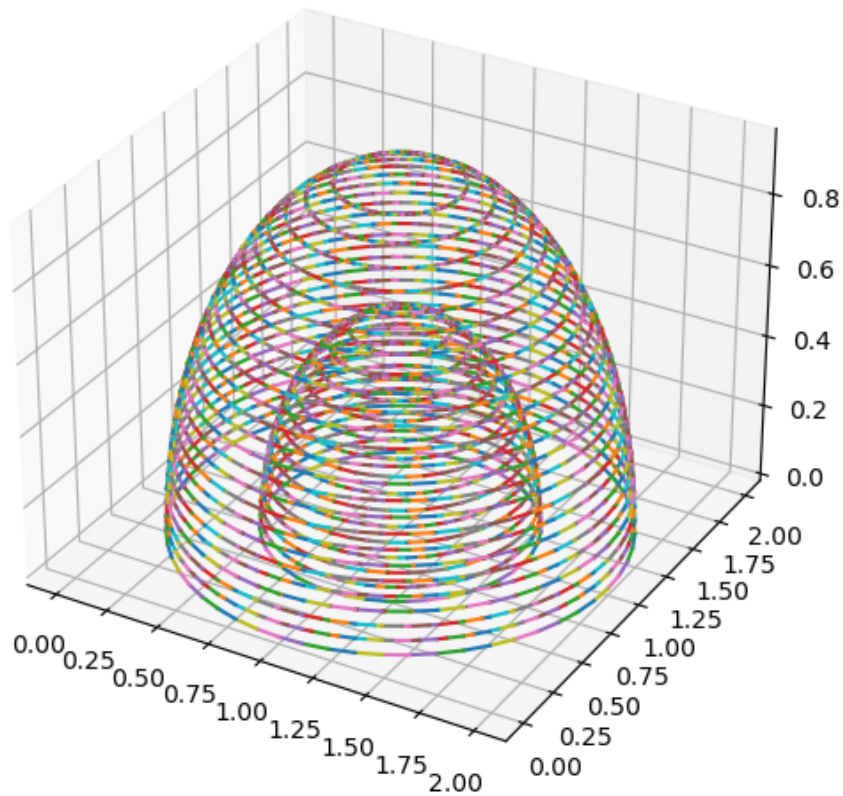


Figure 58: Pressing the Preview-button shows a pyplot of the figure from the selected file.

In the Files box the “Convert file” button triggers the slicing script. This script converts the STL figure into slices based on layer thickness and puts them in a folder on the FTP server called files_for_print. The close button closes the GUI and the eject print disk button lifts the printing chamber so the print disk can be removed. If the show video button is pressed a window with a live video from a webcam is displayed. The video can be closed again with the close video button.

Safety

The distribution table has no concept of what goes on in the GUI unless the script tells it. Therefore, to make sure that the printing is running smoothly and nothing is running in the wrong order, the GUI has been implemented with some safety mechanisms.

As the info box suggests the distribution table should be initialized and a layer thickness set before running the program. If these criteria are not fulfilled when the program starts, the program will stop and output an error message. Because of this the script has been implemented with code that stops the user from pressing the run button without setting a layer thickness and running the initialize sequence. If this is still done, the progress box is updated with a messages telling the user to fulfill the criteria.

As it is not possible to create a preview of the figure without setting a layer thickness there has been implemented an identical safety method. Pressing the preview button before setting a layer thickness will update the progress box with a message telling the user to set the layer thickness before generating a preview.

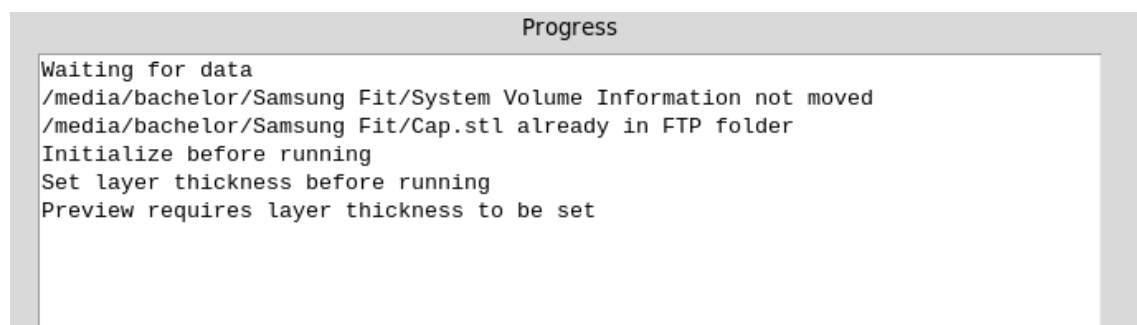


Figure 59: Safety messages telling the user what needs to be done before the program can complete the desired task.

3.5 Manual

To start the printing process there are two parts that need to run. The first one is the python scripts on the Raspberry Pi controlled by the user interface and the second one is the visual basic script made in the VLM software on the Rofin computer.

3.5.1 Setting up the Powder distribution table

Readying the distribution table for execution is done through the user interface on the Raspberry Pi, and is done through four obligatory steps. There are also a few optional steps to get a better understanding and to make sure the process is going as expected.

Step 1: Enter data

Plug a memory USB drive with the STL file into the USB port on the inside wall of the control room. The Raspberry Pi automatically moves the files from the USB-drive to the correct folder.

Step 2: Set the layer thickness

Setting the layer thickness is a necessary step. This is done by pressing the entry box in the parameters box. A numbers keyboard will appear and the user input the desired value. Click “Done” in the numbers keyboard. The applied layer thickness is shown in the same box.



Figure 60: Parameters-box

Step 3: Show preview (Optional)

Pressing the “Show Preview”-button displays the input STL file in a plot. Here the user can see the outline of the figure about to be printed.

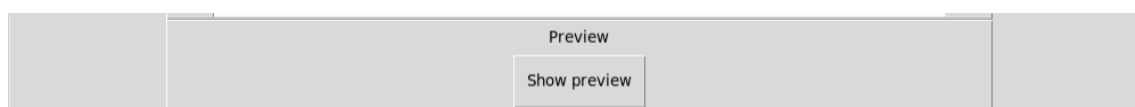


Figure 61: Preview-box

Step 4: Convert file

Press the “Covert file”-button. This takes the STL file and converts each layer into its own DXF file. These are the files that are sent to the FTP-folder on the Rofin computer.

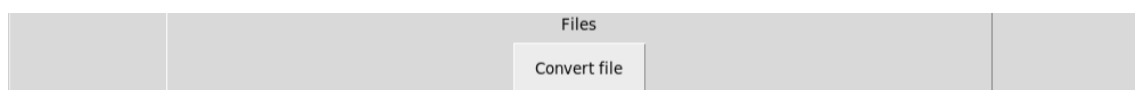


Figure 62: Convert-box

Step 5: Run

Pressing the “Run” button sets the distribution table ready, and is now waiting for a signal from the laser.



Figure 63: Buttons-box.

Step 6: Show video (Optional)

If the user wants to view the process while it is going the user can press the “Show video” button. This opens up a window with a live video from the inspection camera. The window can be closed again with the “Close video” button.

Step 7: Eject the print disk

After the print it is necessary to lift the print bed to its top position in order to remove the finished product. This is done by pressing the “Eject print disk” button.

3.5.2 Setting up the laser

Readying the laser for execution is done through the laser console software and the VLM software. It is done through four obligatory steps.

Step 1: Start reference axis

Navigate into the laser console. On the bottom of the page press the “start reference axis” button.

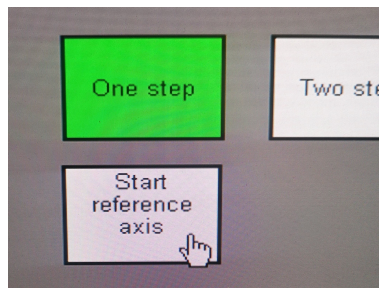


Figure 64: Buttons-box.

Step 2: Open shutter

On the button panel turn the right key to “shutter open” position. This opens the shutter, allowing the laser to come through.



Figure 65: Shutter open.

Step 3: Initialize hardware

Open the VLM software. Press the “initialize hardware” button. This button establishes the connection between the laser and the VLM software.

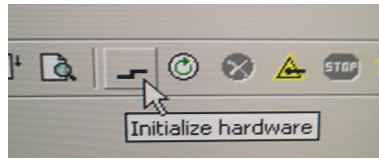


Figure 66: Initialize hardware.

Step 4: Load job

Press the “Load Job” button. This button loads the job onto the laser.

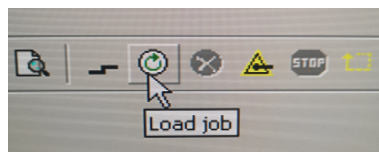


Figure 67: Load job.

Step 5: Start marking

Press the “Start marking” button. This runs the laser and the printing process is now running.

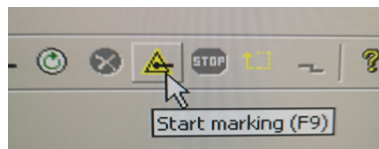
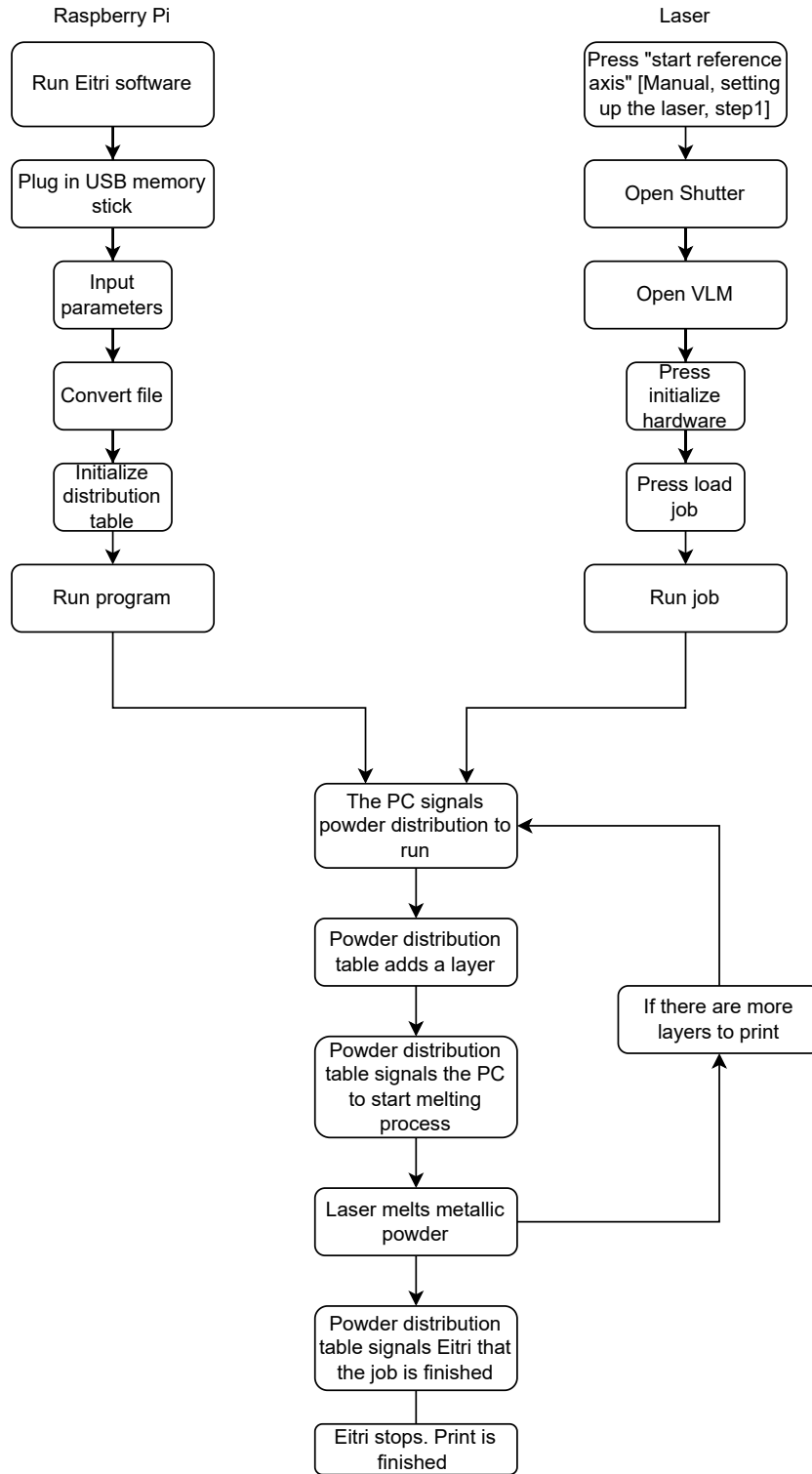


Figure 68: Start marking.

3.6 Flowchart of the system



4 Results

This section describes the groups results from the bachelor thesis.

4.1 End product

The end product of this bachelor is a functional 3D printer. The group has developed software and a communication system between all the units in the printer system. The physical system consist of the framing, the chamber, the powder distribution system and the laser.

There has been major changes to the framing of the system. The new housing fits well in the rear, and creates the space for a brand new screen and keyboard. Together with these two components a touch screen, connected to the RPi, mounted on the right wall in the housing makes up the new control unit. The programmed GUI appears on this screen. An endoscope attached underneath the laser makes it possible to watch the printing process on the RPi's screen. This control panel is a major upgrade to the previous control panel. It also takes up significantly less space.

The system has become lighter and smaller. The length has been reduced from 2.21 meters to approximately 1.5m. This was achieved by removing the old front housing and control panel. Inside the framing the light bulbs, safety light curtains, motor and rotary indexing table has been removed as they are not necessary in the new system. The result of these modifications are an optimization of volume and a major weight reduction.

4.1.1 Print process

The print process is now automatic from the moment the operator pushes the “Run” button on the GUI and the “Start Marking” button in VLM. Before the operator can start the process, the powder bed must be filled with powder and the chamber must be purged with argon gas. The operator needs a 3D model of the desired print saved as a STL file. This file must be transferred to a memory stick. This stick is connected to the USB port in the control room.

On the touch screen the operator can click on the “convert file” button. The STL file will now be sliced, in the slicing script made by the group, using the desired layer thickness. The script will produce DXF files and store them in a folder on the RPi. This folder is connected to a folder on the Rofin computer. This makes it possible for the VLM to import the files.

A VLM script has been created to give the system functionality to print 3D models with varying x and y coordinates and to import the previously mentioned DXF files. This script automatically opens when the system is initialised. The process is now ready to start. When the process is running the communication system will move the process forward described in subsection 4.2.

The laser melting one layer equals one DXF file used. This is repeated until every DXF file is used, and the 3D model is finished.

4.1.2 Safety

The new framing is fully closed. This protects the operator and others in the area from dangerous light radiation. A safety mechanism prevents the laser from running when the doors are open. This is a necessary feature that prevents users or spectators from looking directly at the laser while it is running. A camera is mounted next to the PDS. This allows the operator to overlook the process in a safe way from the control panel.

The lasers control panel is mounted in the control panel compartment underneath the RPi screen. An emergency stop button is included. Pushing this button would immediately stop the laser from running and is an essential safety mechanism for urgent stops.



Figure 69: New front panel mounted.

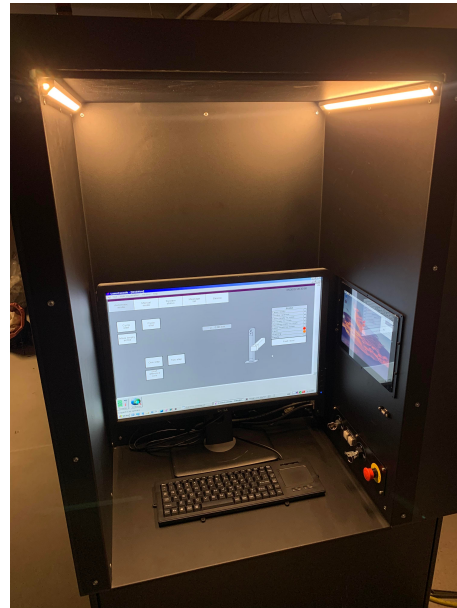


Figure 70: The new control room.

4.2 Communication

The systems general communication is summed up in four steps. The first step is the RPi being notified by the laser to start the printing process. The RPi then signals the powder distribution system (PDS) to apply a layer. The PDS signals back to the RPi after the powder distribution is finished. When the RPi receives this signal it tells the laser to run the melting process. After the melting process is finished, the laser tells the RPi that it has completed the current layer and a new layer can be distributed. Figure 71 illustrates the communication.

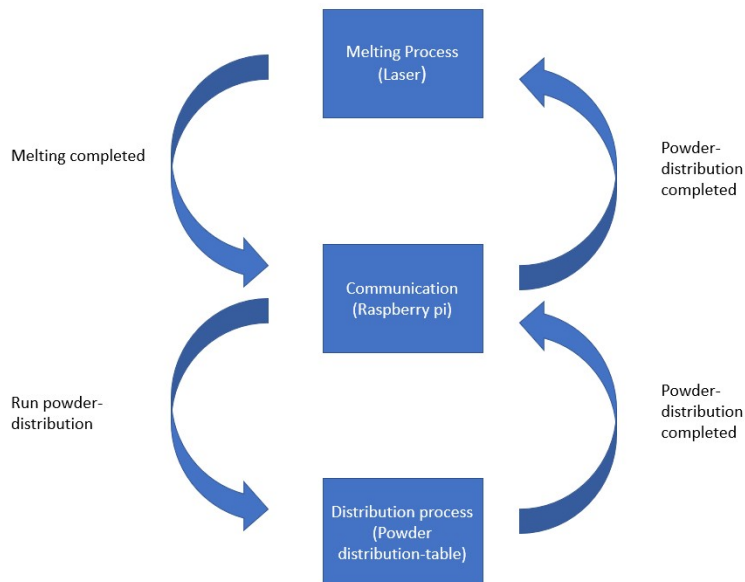


Figure 71: Illustration of the systems general communication.

4.2.1 Step by step

Step 1: Raspberry Pi → Powder distribution-table:

The communication between the RPi and the powder distribution-table is done through a python script on the RPi. This python script uses two libraries to connect software and hardware. Only one of these are used for communicating between the RPi and the powder distribution-table. This is the `zaber_movement` library [37]. The `zaber_movement` library is specifically built for moving zaber devices like the ones that control the distribution table. This library constructs messages following the Zaber ASCII-protocol [36]. An example of a message is `"/1 25 move abs 146982:6C"`. More on how this communication works is described in subsection 2.2.3.

Step 2: Powder distribution-table → Raspberry Pi:

The communication between the powder distribution table and RPi is the table's answer to the previous step. The `zaber_motion` library has functionality to read the answer from the devices as well as generate and send messages. Because of the answers from the devices can be read. When the devices are done with their task they reply with a message including `"OK IDLE"`. By listening for this message the script can understand when the powder distribution is done. An example of a message is `"@01 1 25 OK IDLE -0"`. More on how this communication works is described in subsection 2.2.3.

Step 3: Raspberry Pi → Laser:

The communication between the RPi and the laser is the RPi sending a high signal through one of its GPIO pins. This is done by using a python library called `RPi.GPIO` [38]. A high output on the RPi means a 3.3V output. As the laser needs a 24V input for it to be read as high the signal needed to be run through a relay. This relay closes a switch on a high signal from the RPi and allows the laser to read one of its own 24V outputs as an input. This signal tells the laser that the distribution process is done and it can start the melting process again.

Step 4: Laser → Raspberry Pi:

The communication between the laser and the RPi is a message from the laser saying that it has finished the melting process. This message is sent by the laser setting a high output. This means setting a 24V signal on an output for the RPi to read. As the RPi cannot endure a 24V signal [44], the signal runs through a relay. This relay controls a switch that allows the RPi to read on one of its own output as an input.

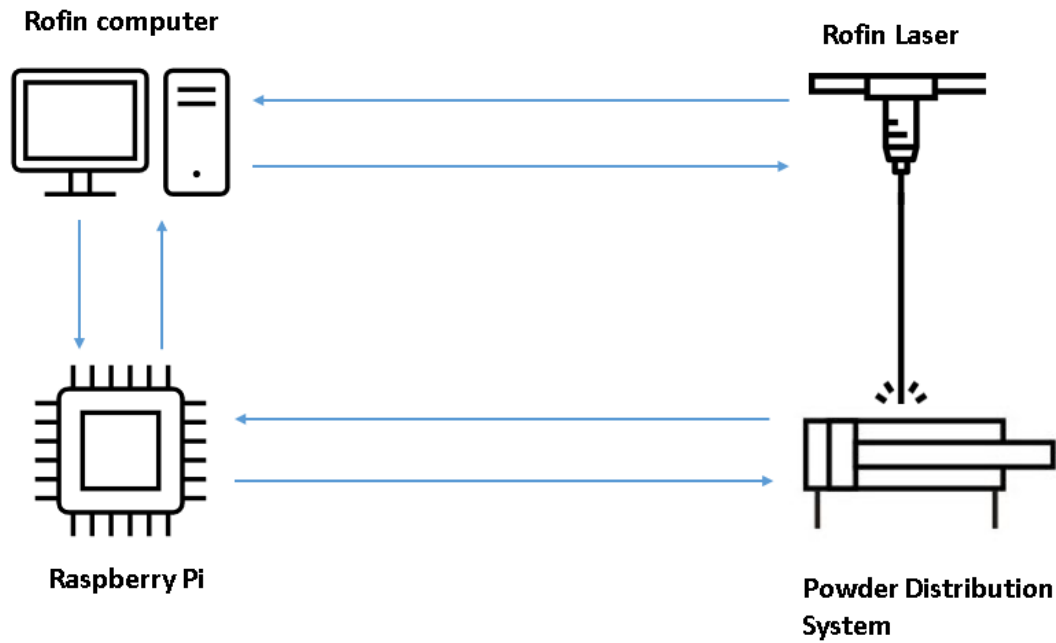


Figure 72: Communication between units.

This communication works without any issues. Throughout the print testing there has been no issues in connection with the timings of the communication system or any signals getting lost in transmission.

4.3 Print testing

When the full system was up and running the group started testing printing parameters. The testing was done to acquire an understanding of how the parameters affected the printing result. This was done by gradually changing the parameters of the laser and looking at what parameter changes affected the print in what way. The group was also looking for what combination of parameters provided the best print result.

The specific parameters being tested and experimented with were the laser power, layer thickness, laser pulse frequency, the lasers speed, the overlap and adjusting the height of the laser compared to the print table.

The discovered results from testing each individual variable is:

- The lasers power affects how much energy the metallic powder is melted with. More power equals more energy delivered to the powder.

-
- The layer thickness controls how much powder is distributed. This affects the amount of energy required for the laser to properly melt it.
 - The laser pulse frequency represents the rate of the pulses. This affects how long each pulse last and then how much energy they deliver. Higher frequencies gives shorter pulses. Shorter pulses transfers less energy per pulse, but at a more consistent rate. Lower frequencies gives longer pulses. Longer pulses transfers more energy but can quickly result in the powder layer being disrupted by the impact.
 - The lasers speed defines how fast the laser moves. The slower the laser moves, the more contact it has with the powder. Therefore, a slower laser speed will result in more melting of the powder and a higher laser speed will result in less melting of the powder.
 - The overlap variable controls how much one line in the hatching will overlap with the previous one. A higher percentage of overlap will result in the powder being exposed to the laser for a longer amount of time. This helps contribute to a smoother melting.



Figure 73: An example of how print testing was done.

Figure 73 shows an example of a test print. In this test the group was trying to figure out how to get a complete printed figure without the metal melting into small pearls. This was done by changing the parameters step by step for each print of a star. The metal gathering in small pearls can clearly be seen in print number 7 (fourth from the left on the second row).

4.4 Print quality

4.4.1 Without protective atmosphere

As explained in the background theory in subsection 2.1.7 about argon gas and oxidation, printing in an unprotected atmosphere will cause a corrupted result. Before the group could acquire argon gas, printing in an unprotected environment was tested. The result from this test can be seen in figure 74 and 75. This is the front and back of the same print.



Figure 74: The upper side.

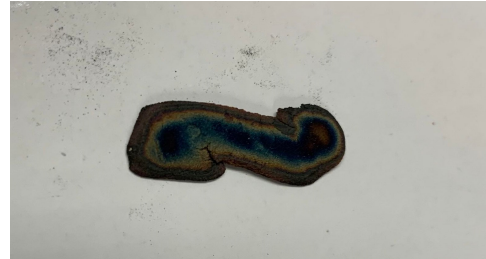


Figure 75: The under side.

The two pictures clearly show that the test results are poor without a protective atmosphere. The structural integrity of the product is greatly reduced. The product is very porous and it breaks easily. The product also has low conductivity. All of these qualities are unwanted and a result of the print process being in an unprotected environment with too high levels of oxygen. This product is unfit for any structural testing of alloys.

4.4.2 With protective atmosphere

The printed result with a protective atmosphere is a lot better quality than without protective atmosphere. In this print the layer thickness is set to 0.8 mm. The first figure consists of only one layer and the second consists of two layers.

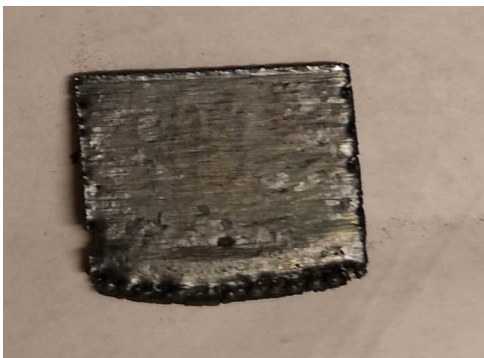


Figure 76: Test print with one layer.



Figure 77: Test print with 2 layers.

You can see from the pictures that the metal looks a lot more coherent than the ones printed without a protective atmosphere. Further testing with different parameters was done to achieve a continuous improvement of the print quality, but no single solution exists as this varies depending on the size of the figure being printed as well as the thickness of the layer.

5 Discussion

5.1 The work process

The development of the metal 3D printer and its communication was a complex and difficult task. The process required a lot of knowledge within several different fields of study. Programming and work with industrial electrical circuits has been the two main jobs. The group had a considerable amount of background information to read regarding the laser systems functionality and previous work which had been done. Considering that the group had little experience with 3D printing in general, it proved difficult to estimate the projects time consumption and how the end product would turn out.

The project has without a doubt been very educational for the group members. Throughout the project theory from various subject has been put to good use. Especially subjects regarding mathematics, digital communication, electrical circuits and industrial control systems has come in handy. The project has given great practice in practical work. This is something which regular coursework provides little training in, as most subject are focused on theoretical aspects.

The bachelor thesis has also given good training in teamwork. This is an essential part of the students future careers as engineers. The group has realised that good project management requires a collective effort and continuous information flow within the group and to the employer.

In the preliminary project report the group decided on four main points to achieve good project management. These were, weekly Monday meetings to decide the weeks goal, weekly Friday meetings, and more frequently later on in the project, for reporting, collective discussions surrounding disagreements and continuous documentation. All of these four points has been achieved during the project. The group is specially pleased with the decision to update the report week by week. This is because, as the project developed, the group could look back on when and why certain solutions were chosen and why problems occurred. This helped majorly in further development and in understanding the process of development.

5.2 Product

The main task of the project was to develop a communication system between the 3D printers components. To accomplish this goal the group could have gone for several different solutions considering the amount of communication protocols that exist today. The operating system Windows XP Embedded on the Rofin computer is very old in computer context. This made it challenging to communicate with newer computers. For this reason the group decided to develop a hardware solution for connecting the old computer and the RPi. This proved to be a good choice as no signals will be interfered with or get lost in transmission. The communication is very robust.

Implementing the Raspberry Pi was a good decision. It became an intermediary between the PDS and old computer, with the ability to handle operations the old computer could not. For instance it runs the scripts for controlling the PDS and

the slicing script. Also the GPIO pins was needed for controlling the sequence in the printing process. The installed Raspberry Pi OS operating system on the RPi is based on Linux, and proved useful in making the GUI. A RPi also cost a fraction of the price compared to a industrial computer. The project would not have been possible without this component.

The group decided to use a memory stick transferring STL files to the Rofin computer. The desirable solution would be to wirelessly transfer the files, but this proved impossible due to the differences between the Rofin computer and the RPi.

Another task was to reduce the volume used by the system. The first thought were to completely build a new frame design for the laser and 3D printer, but the group realized it was possible to optimize the framing which was already in use. The laser required more or less the same height and width as the working space already provided. The terminal blocks, power supply and wiring also needed space underneath the work table. It is also favorable to have the work table about 70 cm above the floor, for easy access and good working conditions. These requirements were fulfilled with the existing framing and the group agrees that it was the right decision to keep the old framing. Both because it works ideally and because it saved the project a lot of time to focus on the communication and functionality of the 3D printer. The group believes the potential of the framing was maximized, and are extremely pleased with the new control panel.

The linear actuator malfunctioned at the same time as the rest of the system was ready for testing. This costed the project about a week of work as the group could not run tests until a new recoater was acquired. This was a frustrating period of the project as the actuator was not a solution the group was responsible for. This extra week could have been used for testing 3D prints with different layer thickness and parameters to get an optimal result. Much of the time after the new recoater was finished were used on completing the report due to the deadline.

The very first printing tests were done without argon gas in the chamber. The result was a super porous and weak “metal” sample, as shown in 4.4.1. Later print tests with the chamber filled with argon gas left a shiny and compact metal object. This proved the importance of using an inert gas to the group.

5.3 Further development

Although the group is very pleased with the solutions achieved in this project, there are a few possible improvements and alternations. Due to the broadness and magnitude of the project there are some problems the group would have solved differently had there been more time available.

5.3.1 Different linear actuator

The linear actuator from Zaber used from a previous project was not suited for its task. Due to mechanical wear it malfunctioned and was replaced with a creative solution during the last two weeks. This solution was necessary for testing the system before the delivery date arrived and worked its purpose but is obviously not a satisfactory permanent solution.

A more optimal solution would be to invest in a motor and actuator suited for moving back and forth horizontally many times. This would be a more industrial and stable solution and would also look better. The new solution would also have to be rigid in the vertical direction so that the recoater distributes the powder smooth every time. This is an easy implementation but the delivery time for this kind of motor was too long.

5.3.2 Chamber

It's not possible to achieve a good quality print in open atmosphere due to oxidation. The chamber where the PDS is located has to be free of oxygen, and is purged with a noble gas to achieve this. The current chamber is not ideal because it is not properly sealed, neither on the sides or in the top. This makes it difficult to maintain a low oxygen environment during the printing process. Because this part of the system was not a part of our task the group continued using the existing chamber in agreement with SINTEF.

The area where the laser started to melt a new layer left a uneven and black surface. This can be seen on the right side of the metal object at figure 77. The groups supervisor at SINTEF was certain that this is caused by oxidation. This means the chamber has problems with keeping the print area free of oxygen.

To get an even better result the current chamber has to be replaced with a more fitting chamber optimized for maintaining an oxygen free environment. This is a development that also could have been completed had the group had more time left at the end of the project.

5.4 SINTEF's feedback

At the end of the project the group received feedback from the supervisor at SINTEF. The feedback was very good. It was as follows:

SINTEF is very satisfied with the solutions chosen by the group in the development of a miniature 3D printer for alloy development. There has been developed multiple innovative solutions. The communication system between all components is a very good and well thought out innovation. The algorithms for slicing the geometry into layers that will be printed is completely new and innovative. The interface and physical layout is very good. The physical restructuring of the framing makes the printer more functional and flexible for research. First attempts at printing shows that it is possible to print 3D geometries. For SINTEF and NTNU this new machine will reduce time and cost and we have now received a tool suitable for master and Phd work.

6 Conclusion

This chapter will summarize the project through a few conclusions. The main challenge was to create a communication system between the Rofin laser and the PDS This challenge has been successfully accomplished through the use of a RPi. The communication between the units runs without faults and errors. This allows the system to print in 3D.

The system will also be able to produce different three dimensional geometric structures from metal powder with the correct parameters. The printing results can be optimized with further research into the parameters and making sure the printing area is free of oxygen.

The physical components designed by the group fits well into the assigned areas of the framing. By making the framing smaller the system now takes up a lot less area and has a drastically reduced weight. The old control panel has been removed and replaced with a new upgraded control panel and work station in the new compartment in the rear part of the system.

The group is very pleased with the outcome of the bachelor thesis as a whole. When starting to work on the project the group knew that it was a challenging task. The group was set on delivering a functional system through consistent work and by putting our knowledge into a physical product.

As the product is finished and the 3D printer is running the group is confident that the product is satisfactory and is going to be of good use to SINTEF. This conclusion is based of the results seen in testing and the systems communication working from start to finish without any malfunctions.

As can be seen in subsection 5.4 SINTEF's assessment of the project is in accordance with the groups assessment. SINTEF agrees that the finished product is well produced and is going to be of good use to SINTEF.

7 References

- [1] *3D Metal Printing Technology - ScienceDirect*. <https://www.sciencedirect.com/science/article/pii/S2405896316325496>. (Accessed on 05/13/2022).
- [2] *3D Printing File Formats: Everything you need to Know - Additive-X*. <https://www.additive-x.com/blog/file-formats-used-3d-printing/>. (Accessed on 05/05/2022).
- [3] *The History of Tessellations: The Mathematical Art of Repeating Patterns*. <https://mymodernmet.com/tessellation-art/>. (Accessed on 04/20/2022).
- [4] *STL File Format*. <https://docs.fileformat.com/cad/stl/>. (Accessed on 05/19/2022).
- [5] *What is an STL file? — Everything You Need to Know About This File Format*. <https://www.sculpteo.com/en/3d-learning-hub/create-3d-file/what-is-an-stl-file/>. (Accessed on 05/05/2022).
- [6] *DXF File Format*. <https://docs.fileformat.com/cad/dxf/>. (Accessed on 04/20/2022).
- [7] *DXF file - What it is and how do you open it? — Adobe*. <https://www.adobe.com/creativecloud/file-types/image/vector/dxf-file.html#:~:text=DXF%20is%20short%20for%20Drawing,3D%20drawings%20during%20product%20design..> (Accessed on 05/19/2022).
- [8] *Raster vs. Vector: Understanding Design File Types 101*. <https://www.vecteezy.com/blog/design-tips/raster-vs-vector#:~:text=Raster%20files%20consist%20of%20tiny,distortion%20or%20loss%20of%20quality..> (Accessed on 05/19/2022).
- [9] *Study on STL-Based Slicing Process for 3D Printing*. <https://repositories.lib.utexas.edu/handle/2152/89888>. (Accessed on 05/13/2022).
- [10] *IV. Slicing. As Lulzbot says, “slicing software... — by Garrett Spiegel — 3D Printing in O&P — Medium*. <https://medium.com/3d-printing-in-o-p/iv-slicing-72a9515f44bc>. (Accessed on 04/04/2022).
- [11] *Metal powder for additive manufacturing — Metal powder — Sandvik*. <https://www.metalpowder.sandvik/en/products/applications/additive-manufacturing/>. (Accessed on 05/13/2022).
- [12] *Materials Used in Selective Laser Melting (SLM) - Matmatch*. <https://matmatch.com/learn/material/materials-used-in-selective-laser-melting-slm>. (Accessed on 03/31/2022).
- [13] Sudha Cheruvathur, Eric A Lass, and Carelyn E Campbell. “Additive manufacturing of 17-4 PH stainless steel: post-processing heat treatment to achieve uniform reproducible microstructure”. In: *Jom* 68.3 (2016), pp. 930–942.
- [14] *What Is Oxidation? Definition and Example*. <https://www.thoughtco.com/definition-of-oxidation-in-chemistry-605456#:~:text=Oxidation%20is%20the%20loss%20of,%2C%20molecule%2C%20or%20ion%20decreases..> (Accessed on 05/12/2022).

-
- [15] *The effect of powder oxidation on defect formation in laser additive manufacturing - ScienceDirect*. <https://www.sciencedirect.com/science/article/pii/S1359645418309698>. (Accessed on 05/13/2022).
- [16] *inertgass - Store norske leksikon*. <https://snl.no/inertgass>. (Accessed on 03/30/2022).
- [17] *Bond Energy — Chemistry for Non-Majors*. <https://courses.lumenlearning.com/cheminter/chapter/bond-energy/>. (Accessed on 05/10/2022).
- [18] Pauzon, Camille. *Effect of argon and nitrogen atmospheres on the properties of stainless steel 316 L parts produced by laser-powder bed fusion*. https://research.chalmers.se/publication/510755/file/510755_Fulltext.pdf. (Accessed on 04/26/2022). Mar. 2019.
- [19] *Serial communication Basic Knowledge -RS-232C/RS-422/RS-485- — CONTEC*. <https://www.contec.com/support/basic-knowledge/daq-control/serial-communicatin/#:~:text=Serial%20communication%20is%20a%20communication,one%20bit%20at%20a%20time..> (Accessed on 05/12/2022).
- [20] *File Transfer Protocol (FTP) Definition*. <https://www.investopedia.com/terms/f/ftp-file-transfer-protocol.asp>. (Accessed on 05/12/2022).
- [21] *What is Python? Executive Summary — Python.org*. <https://www.python.org/doc/essays/blurb/>. (Accessed on 05/19/2022).
- [22] *Python scripts and modules — AMath 483/583, Spring 2013 1.0 documentation*. https://faculty.washington.edu/rjl/classes/am583s2014/notes/python_scripts_modules.html. (Accessed on 05/19/2022).
- [23] *PowerLine F Operator's Manual Version 1.2.6 (RCU)*.
- [24] *Combiline Advanced RT 800/RT 1000 Manual, Version 1.0, 2007*.
- [25] *Hovedrapport.pdf*. <https://drive.google.com/file/d/12y9oDkHQdA5QgiTuCuc4MMH6sB-YDepq/view?usp=sharing>. (Accessed on 04/20/2022).
- [26] *X-VSR-SV1 User's Manual - Zaber*. <https://www.zaber.com/manuals/X-VSR-SV1>. (Accessed on 04/01/2022).
- [27] *X-LSM User's Manual - Zaber*. <https://www.zaber.com/manuals/X-LSM>. (Accessed on 04/01/2022).
- [28] *X-VSR20A Specifications - Zaber*. <https://www.zaber.com/products/vertical-stages/X-VSR/specs?part=X-VSR20A>. (Accessed on 04/21/2022).
- [29] *X-LSM100A Specifications - Zaber*. <https://www.zaber.com/products/linear-stages/X-LSM/specs?part=X-LSM100A>. (Accessed on 04/21/2022).
- [30] *Best pris på Raspberry Pi 3 Model B+ - Se priser før kjøp i Prisguiden*. <https://prisguiden.no/produkt/raspberry-pi-3-model-b-326717>. (Accessed on 04/28/2022).
- [31] *Raspberry Pi OS - Raspberry Pi*. <https://www.raspberrypi.com/software/>. (Accessed on 05/18/2022).
- [32] *3 in 1 USB Type C Endoscope 7mm Inspection HD Camera For Android PC Borescope kjøp billig — fri frakt, ekte anmeldelser med bilder — Joom*. <https://www.joom.com/nb/products/60b5cc0ced7e3201f0fa7185>. (Accessed on 05/15/2022).
-

-
- [33] *Fusion 360 — 3D CAD, CAM, CAE & PCB Cloud-Based Software — Autodesk*. <https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR&tab=subscription>. (Accessed on 05/11/2022).
- [34] *Gas Technologies & Chemicals - Argon*. <https://www.gastech.co.il/en/argon>. (Accessed on 05/05/2022).
- [35] *File Transfer Protocol (FTP) Definition*. <https://www.investopedia.com/terms/f/ftp-file-transfer-protocol.asp>. (Accessed on 04/21/2022).
- [36] *Zaber ASCII Protocol Manual - Zaber*. <https://www.zaber.com/protocol-manual?device=X-LSM100A&peripheral=N%2FA&version=7.26&protocol=ASCII>. (Accessed on 04/01/2022).
- [37] *Zaber Motion Library (ASCII)*. <https://www.zaber.com/software/docs/motion-library/ascii/>. (Accessed on 04/01/2022).
- [38] *Control Raspberry Pi GPIO Pins from Python — ICS*. <https://www.ics.com/blog/control-raspberry-pi-gpio-pins-python>. (Accessed on 04/20/2022).
- [39] *General Purpose Input Output on the Raspberry Pi*. <https://www.futurelearn.com/info/courses/physical-computing-raspberry-pi-python/0/steps/23033#:~:text=The%20GPIO%20pins%20are%20one,interact%20with%20many%20other%20objects..> (Accessed on 04/04/2022).
- [40] *GitHub - pyusb/pyusb: Easy USB access for Python*. <https://github.com/pyusb/pyusb>. (Accessed on 05/10/2022).
- [41] *libusb*. <https://libusb.info/>. (Accessed on 05/10/2022).
- [42] *Three-Way Handshake - an overview — ScienceDirect Topics*. <https://www.sciencedirect.com/topics/computer-science/three-way-handshake#:~:text=The%20TCP%20handshake,as%20shown%20in%20Figure%203.8..> (Accessed on 05/10/2022).
- [43] *tkinter — Python interface to Tcl/Tk — Python 3.10.4 documentation*. <https://docs.python.org/3/library/tkinter.html>. (Accessed on 05/03/2022).
- [44] *Raspberry Pi and General-Purpose Input/Output*. <https://www.futurelearn.com/info/courses/robotics-with-raspberry-pi/0/steps/75878#:~:text=A%20voltage%20between%201.8V,you%20will%20fry%20your%20Pi!> (Accessed on 04/25/2022).

A Accounting

Throughout the project the group have made several purchases necessary for completing the product. This includes difference tools, components, electrical components and minor things for the work space. The cost of the different objects are listed in the table below.

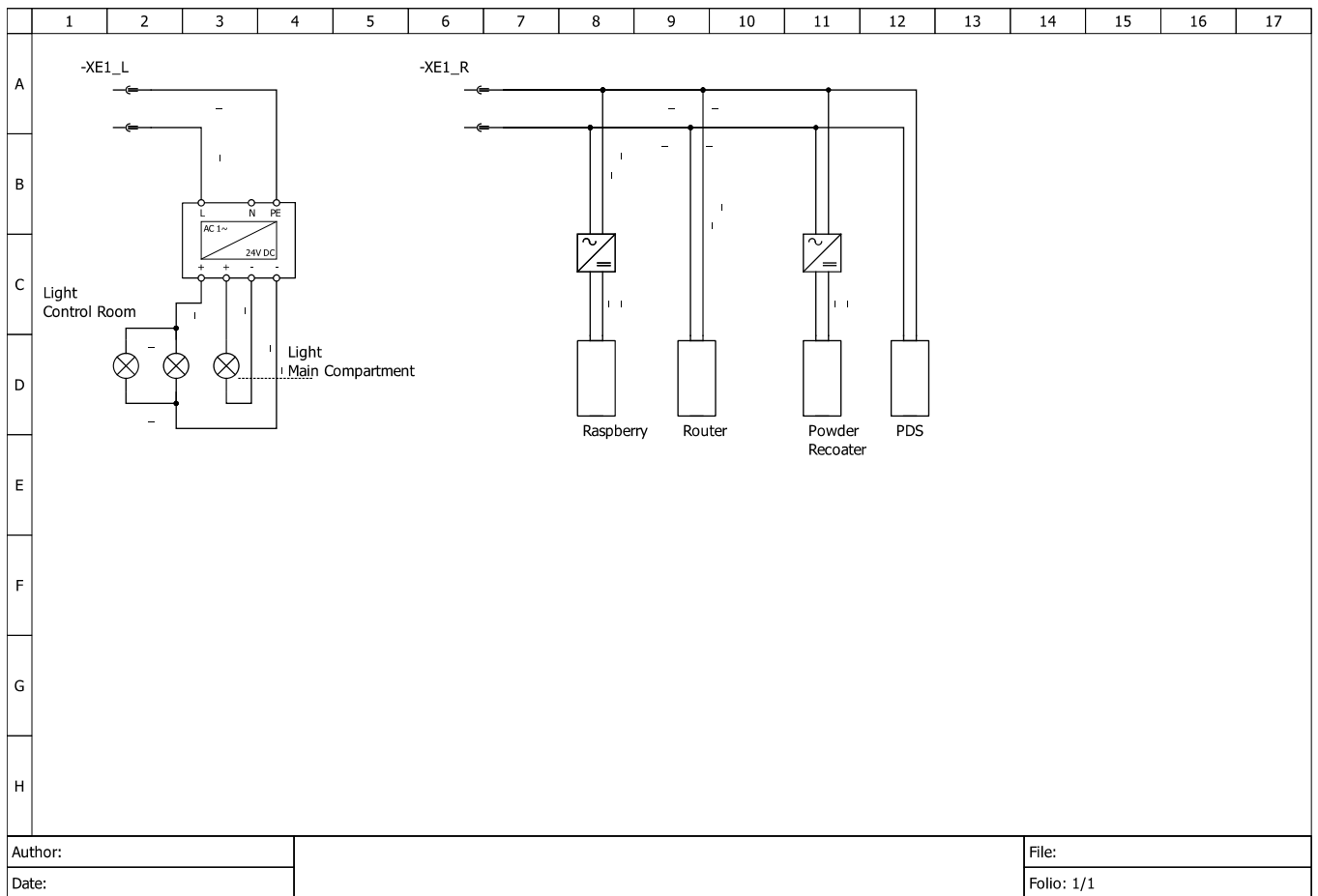
Table 4: Accounting of purchases

Product	Price	Comment
Würth, Drill	-	-
Würth, Angle grinder	-	-
Würth, battery screwdriver S4-A	-	-
Würth, battery package for tools	-	-
Würth, pipe and bit set	-	-
Würth tools (Total)	7300.00 NOK	Equipment for disassembly and building framework
Earplugs 10 pairs	36.90 NOK	Safety
Safety goggles	29.90 NOK	Safety
Solder Sleeves	99.80 NOK	Joining cables
Heat shrink tubing	69.90 NOK	Use to seal joint cable
HDMI to DVI-D adapter	159.90 NOK	Adapter for new screen
USB2-HUB 4-ports	199.90 NOK	USB2 extra inputs
Electrical tape, red & black	59.80 NOK	Tape
Wago terminal block, 2 and 3-ways	219.80 NOK	Connect cables
Storing box	35.90 NOK	Store screws etc
Strips (Cable tie set)	159.90 NOK	Secure cables to the framing
Extension lead, 3 and 6-ways	349.90 NOK	Power supplies to components and PC's
Drill set	159.90 NOK	For drilling
Bits screwdriver (2x)	499.80 NOK	For bits
Cabel FQ $1.5mm^2$	49.90 NOK	Cables for wiring
Cable collector	69.90 NOK	Collecting wires
Tape measure	119.90 NOK	Measuring
HDMI 3m	169.90 NOK	HDMI for screen
Wall hooks	179.90 NOK	Hanging up cloths in work space.
HP V27i 27" monitor	1490.00 NOK	New screen for framing
LableManager + lable tape	475.00 NOK	Put lable on new wires
Bootlace Ferrule $2.5mm^2$, 100 pieces	29.10 NOK	Make terminals
Twin Entry Ferrule $2.5mm^2$, 100 pieces	38.45 NOK	Make terminals

Table 5: Accounting of purchases, pt.2

Electronic side cutter	214.80 NOK	For cutting cables
TRAP 22-10 Crimp Tool	421.18 NOK	Seal cables
A-DS15-HOOD-WP	128.07 NOK	Connection from PC to Raspberry PI
Network cable	98.37 NOK	Connection from PC to Raspberry PI
Orange Ferrule Connector 100 pieces	77 NOK	Make terminals
Namron LED strip 5m	719.10 NOK	Lighting
Namron Driver dimbar LED strip	879.12 NOK	Lighting
Namron Aluminiumsprofil 2m, 2x	898.20 NOK	Lighting
Bolts and nuts, M4	89.70 NOK	Attach list for lighting
TP-Link Omada TL-R605	668.00 NOK	Router
USB endoscope	185.47 NOK	Camera inside framing
USB-extension	36.81 NOK	USB-extension
LCD screen, 10.1 inch	1258.2 NOK	Screen
Power-extension	59.09 NOK	For PC-screen
HDMI 3m	185.65 NOK	HDMI cable
Relay, 3x	395.64 NOK	Control system
USB to micro-B 3m	228.43 NOK	Power/Charger
USB 2m	106.06 NOK	USB cable
Raspeberry Pi power supply	133.93 NOK	Power supply
Keyboard	698.44 NOK	Keyboard
HUB USB 4-port	95.90 NOK	USB HUB
Socket	89.00 NOK	Socket for power supply
Silicon spray	59.90 NOK	Lubricant spray
Cable gland, 2x	73.80 NOK	For new cables
Junction box	36.90 NOK	Protect the wire connection.
Razor blad	139.00 NOK	Used as new recoater
Superglue	99.00 NOK	Glue components
Different M4 screws	167.70 NOK	Building
Metal discs	106.80 NOK	Building
Namron mounting set	79.00 NOK	Mount aluminiumprofil
Cabel cutter	219.00 NOK	Cutting cables
Springs	119.00 NOK	Making new recoater
Different cables	353.70 NOK	Building
Mounting table	26.90 NOK	Mounting
Drill screws 4, 2X19mm	54.90 NOK	Mounting
Electronic cleaning	89.90 NOK	Clean components
Front cover and control cabinet	13937.50 NOK	New components to the framing
Total	35236.41 NOK	

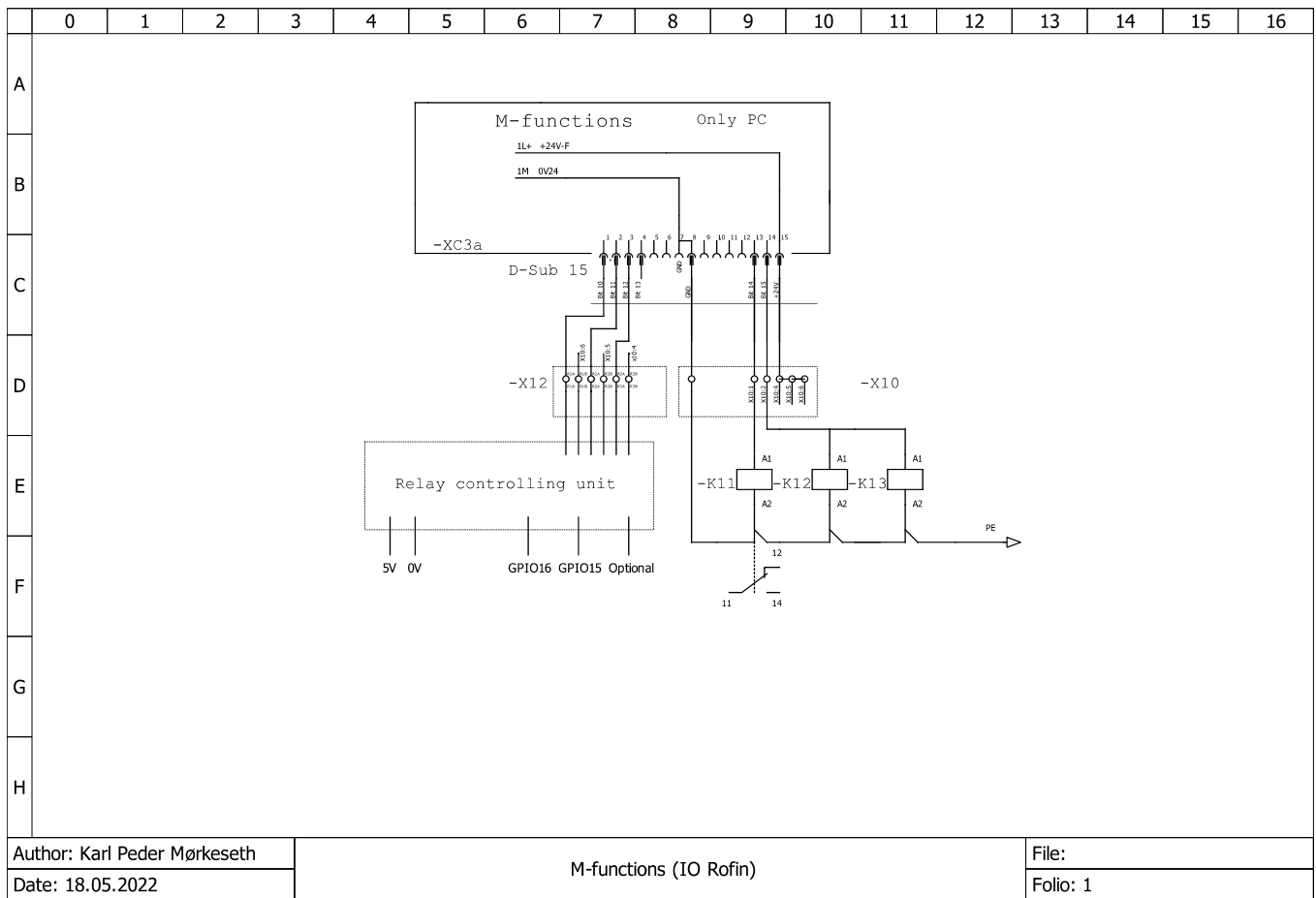
B Power diagram to supplementary components



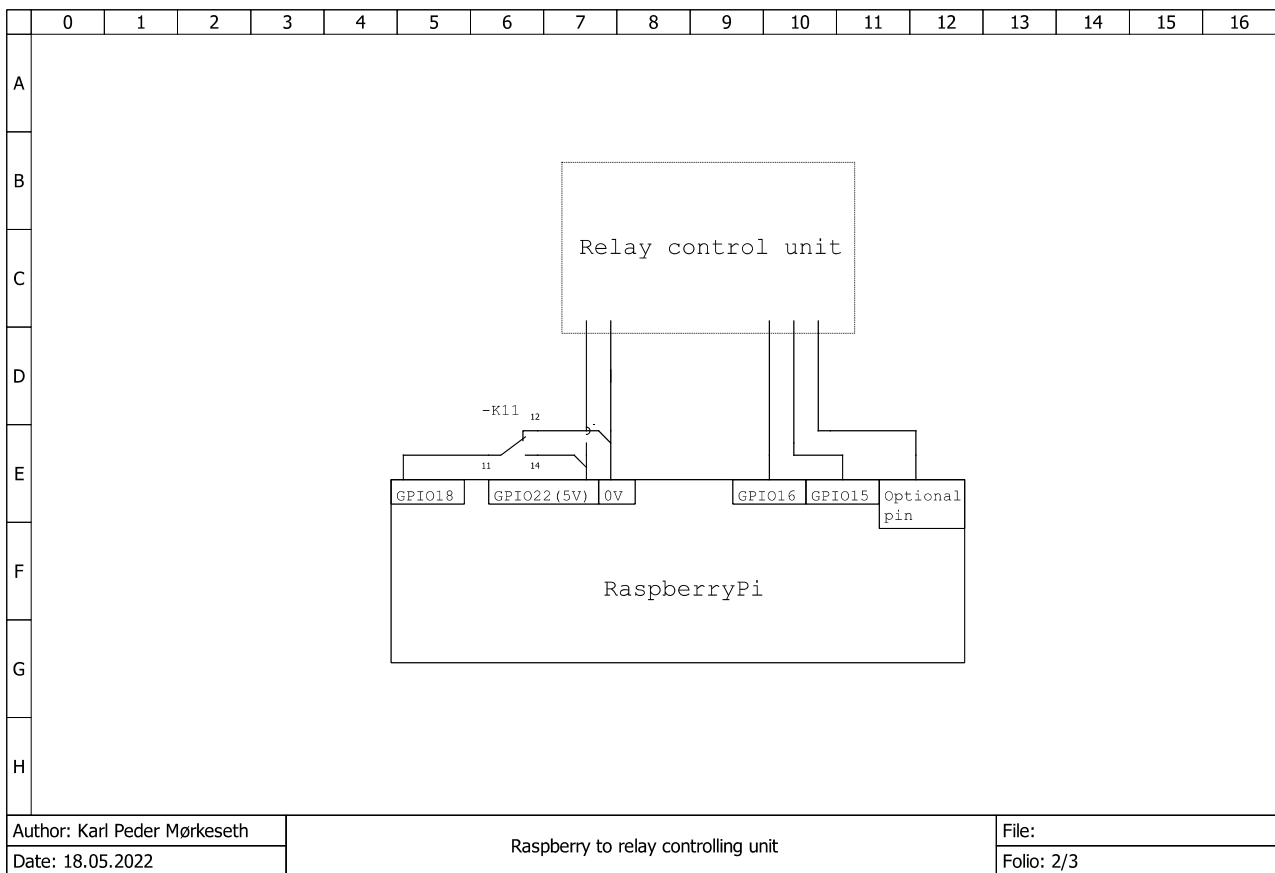
Author:
Date:

File:
Folio: 1/1

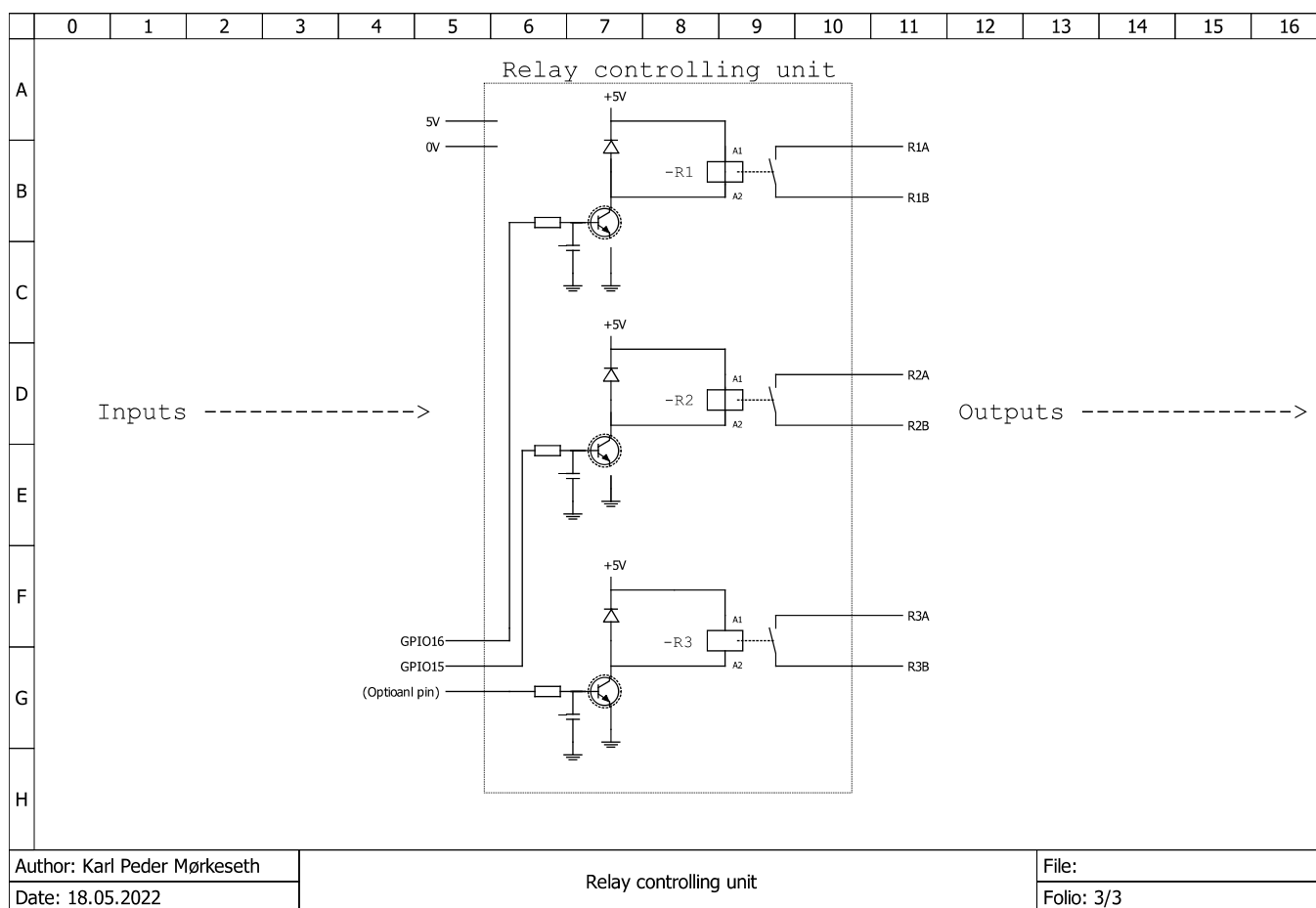
C M-functions IO



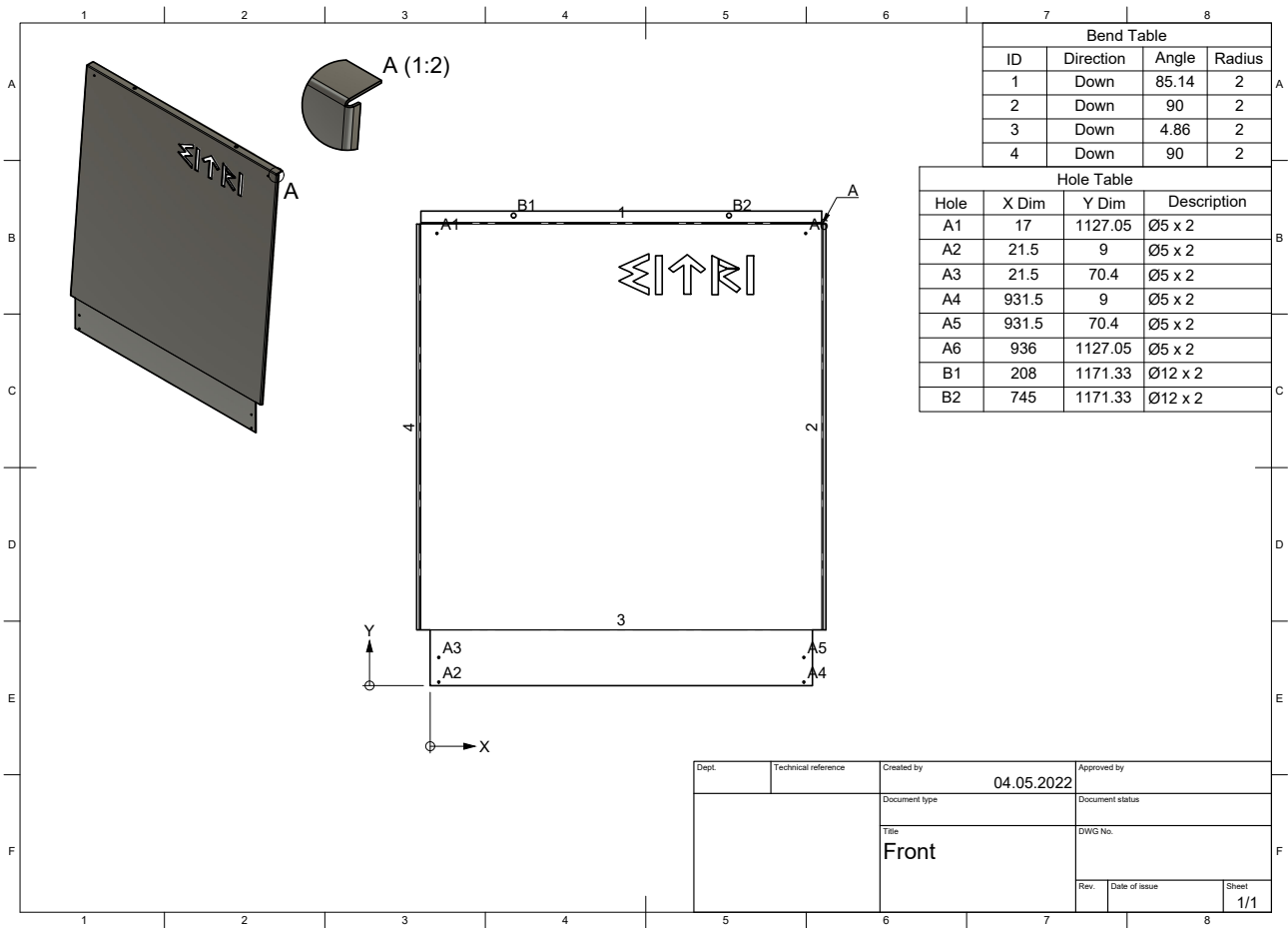
D Raspberry Pi – > Relay controlling unit



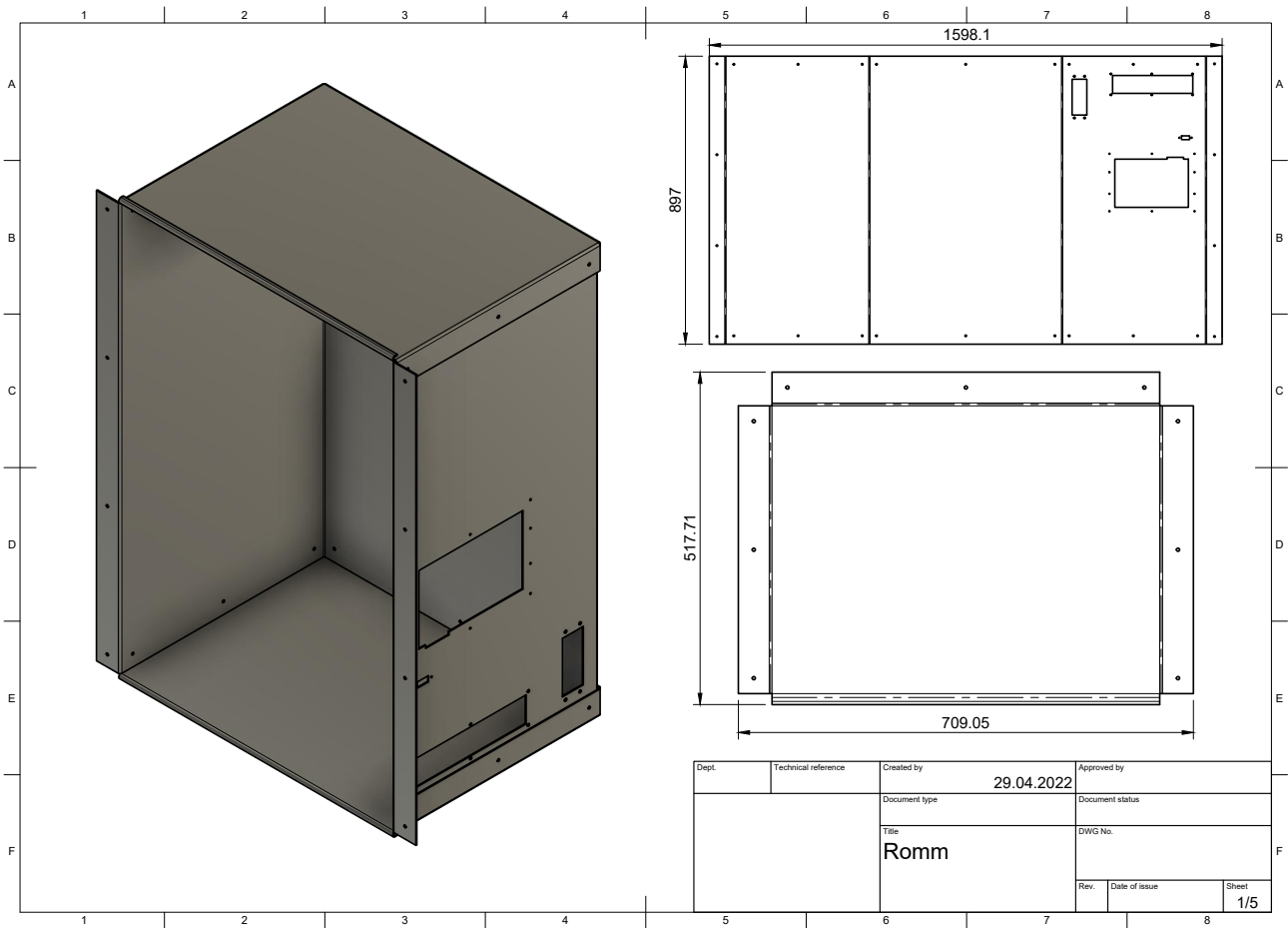
E Relay controlling unit

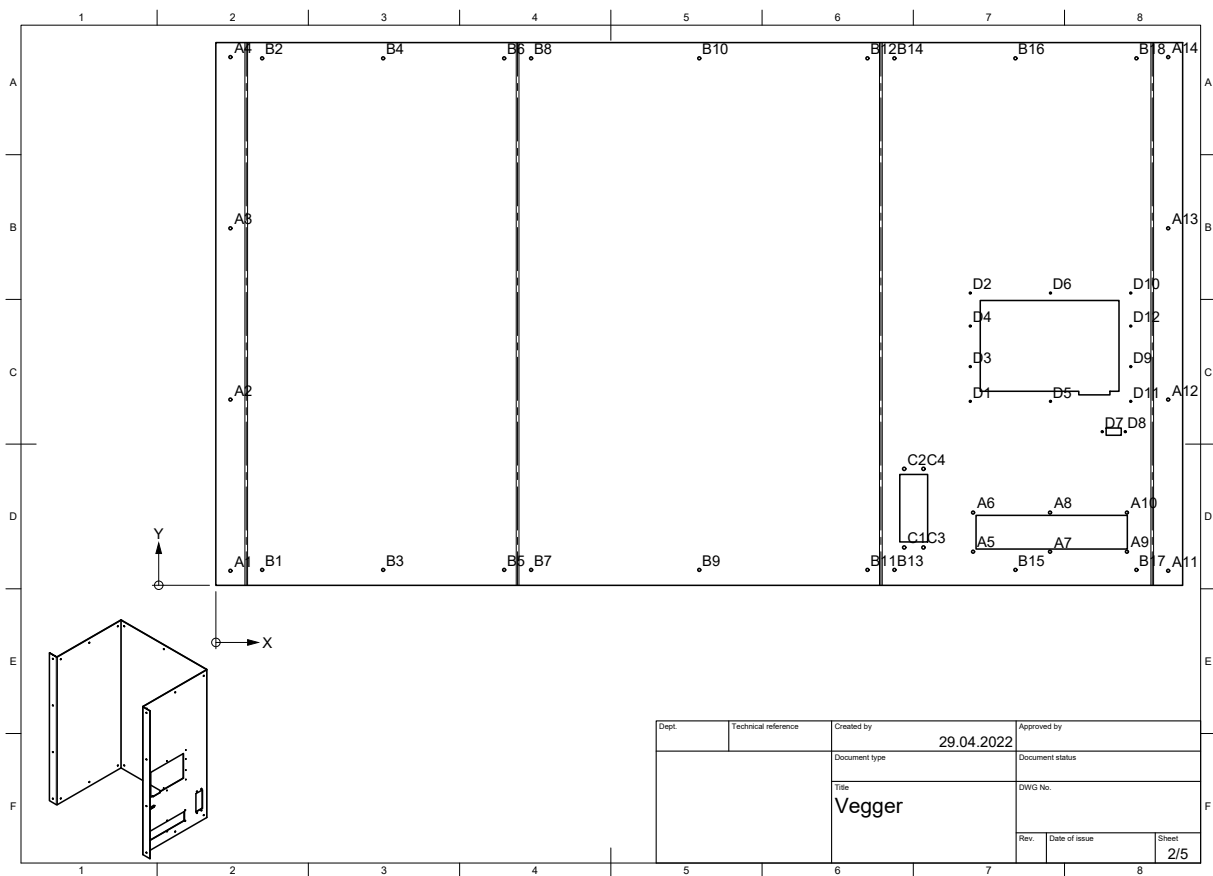


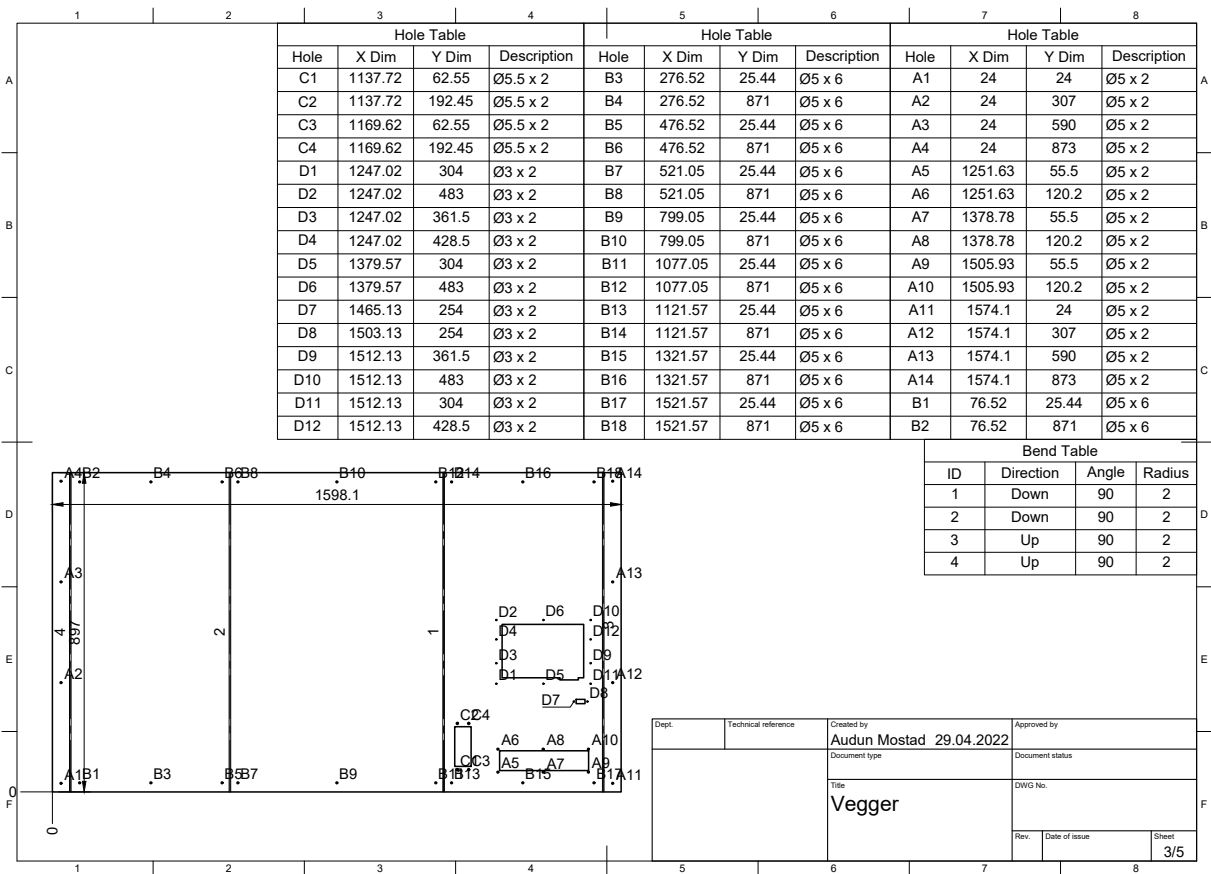
F Front drawing

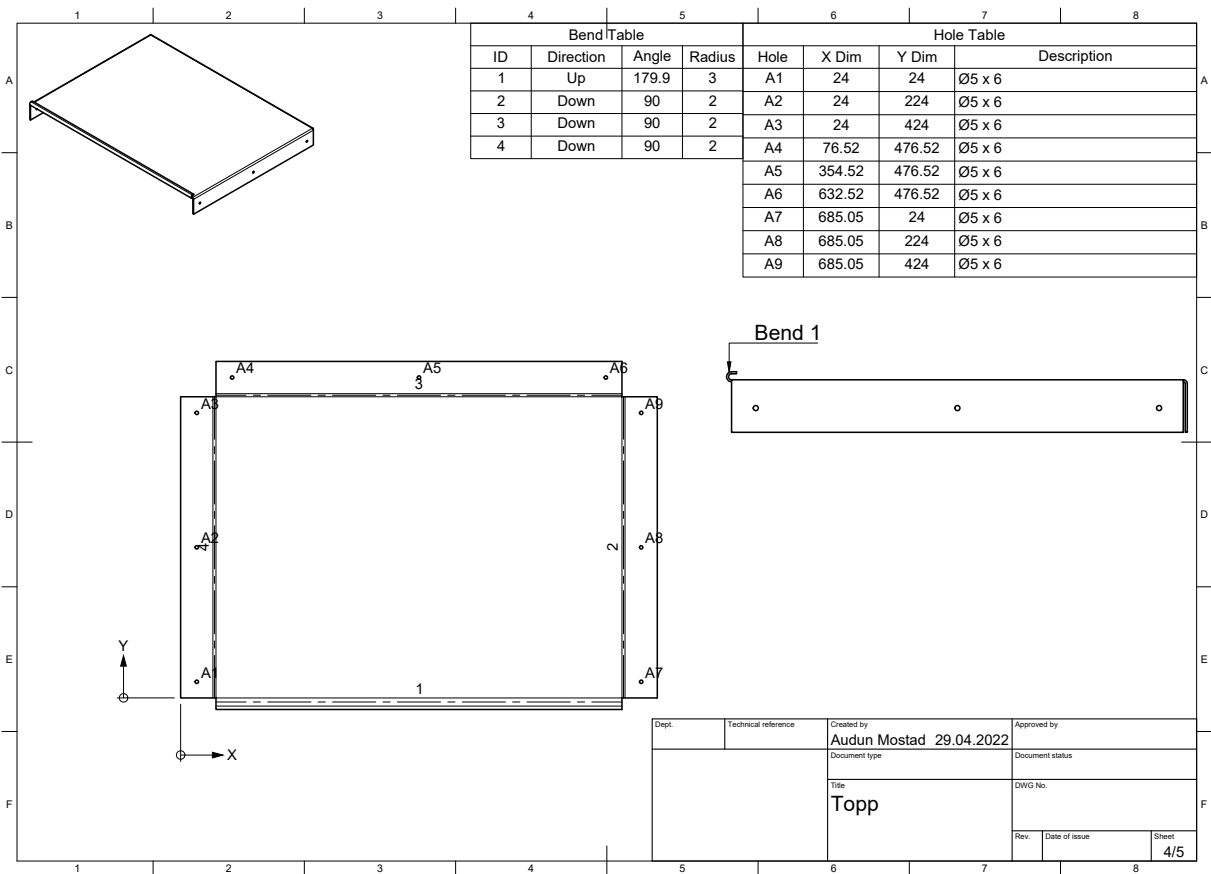


G Control room drawings

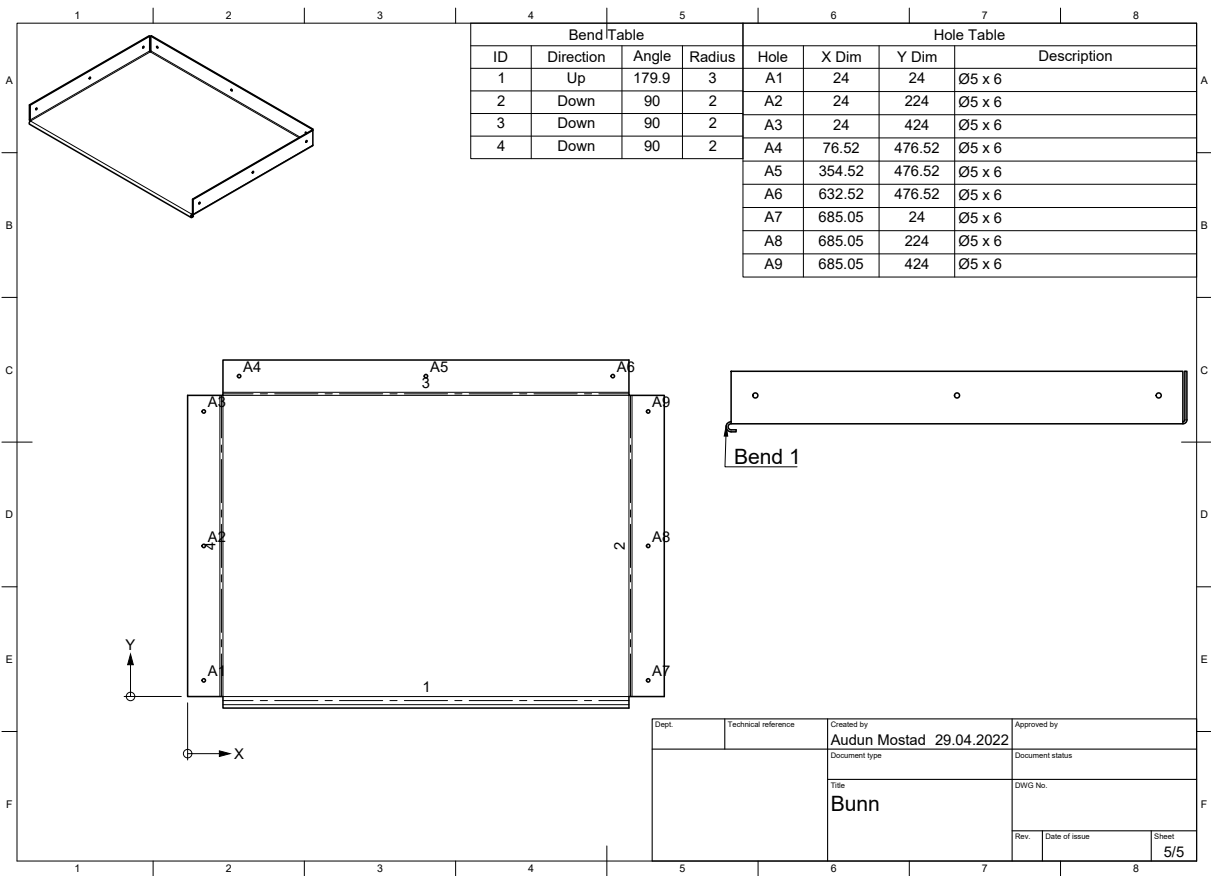








Dept.	Technical reference	Created by Audun Mostad 29.04.2022	Approved by
		Document type	Document status
		Title Topp	DWG No.
		Rev.	Date of issue
			Sheet 4/5



H Main program and GUI script

5/20/22, 9:09 PM

Eitri.py

```
1 #!/bachelor/bin/python
2
3 # Importing necessary libraries
4 import tkinter as tk
5 from tkinter import ttk
6 import os
7 import matplotlib.pyplot as plt
8 import matplotlib.figure as Figure
9 from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)
10 import drive_the_stages as dts
11 import zaber_motion
12 from zaber_motion import Library, log_output_mode, Units
13 from zaber_motion.ascii import Connection
14 from zaber_motion.gcode import OfflineTranslator
15 import time
16 import RPi.GPIO as GPIO
17 from moving_files import run_moving_files, delete_files_from_FTP_server
18 # from Butcher import butcher
19 from Damascus import run_damascus
20 from numbers_table import run_numbers
21 import numbers_table
22 from Mandoline import chef
23 import threading as thr
24 import cv2
25 from video_file import run_video
26 from ftplib import FTP
27 from mpl_toolkits import mplot3d
28
29 # Function for opening the numbers-popup-window
30 def click_entry(event):
31     global lt_entry, window, calc_gui, txt_edit, label_lt
32
33     if calc_gui == 0:
34         # Creating pop-up-window
35         calc_gui = tk.Toplevel(window)
36
37         calc_gui.attributes('-topmost', True)
38         calc_gui.update()
39
40         calc_gui.configure(background="white")
41
42         calc_gui.title("Numbers input")
43
44         calc_gui.geometry("600x340")
45
46         # Runs the main function in the numbers_table-script
47         run_numbers(window, lt_entry, calc_gui)
48
49         done_b = tk.Button(calc_gui, text='Done', fg='black', bg='grey',
50                             command=done, height=3, width=20)
51         done_b.grid(row=6, column='2')
52
53     else:
54         calc_gui.deiconify()
55
56 # Function to hide the numbers popup-window
57 def done():
58     global lt, calc_gui, plot_req
```

localhost:4649/?mode=python

1/8

```

59     lt = lt_entry.get()
60     label_lt["text"] = f"current layer thickness: {lt} mm"
61     f_vars = open('print_variables.txt', 'w')
62     f_vars.write(f'layer_thickness={lt}\n')
63     plot_req = True
64     txt_edit.insert(tk.END, f'Layer thickness set to {lt} mm\n')
65     txt_edit.update()
66     txt_edit.yview_pickplace('end')
67     calc_gui.withdraw()
68
69 # Function to eject the print disk
70 def eject_disk():
71     global axis_list
72     axis_list[0].move_absolute(19.7, Units.LENGTH_MILLIMETRES)
73
74 # Function to show the preview-plot
75 def plot_show():
76     global lt, plot_req, txt_edit
77     if plot_req == True:
78         txt_edit.insert(tk.END, 'Generating preview. This might take a while...\n')
79         txt_edit.update()
80         txt_edit.yview_pickplace('end')
81         path = '/home/bachelor/FTP/files_for_transfer'
82         for filename in os.listdir(path):
83             stl_file = path+'/' +str(filename)
84             plot_to_show = plt.figure()
85             axes = mplot3d.Axes3D(plot_to_show)
86             plot_to_show.add_axes(axes, auto_add_to_figure = False)
87             run_damascus(stl_file, lt, txt_edit, plot_to_show)
88     else:
89         txt_edit.insert(tk.END, 'Preview requires layer thickness to be set\n')
90         txt_edit.update()
91         txt_edit.yview_pickplace('end')
92
93 # Function to set the layer thickness
94 def lt_set():
95     global lt, plot_req
96     lt = lt_entry.get()
97     label_lt["text"] = f"current layer thickness: {lt} mm"
98     f_vars = open('print_variables.txt', 'w')
99     f_vars.write(f'layer_thickness={lt}\n')
100    plot_req = True
101    txt_edit.insert(tk.END, f'Layer thickness set to {lt} mm\n')
102    txt_edit.update()
103    txt_edit.yview_pickplace('end')
104    f_vars.close()
105
106 # Function to convert the stl file into many dxf-files (one for each slice)
107 def chop_files():
108     global lt, txt_edit, chop_req
109     float_lt = float(lt)
110     txt_edit.insert(tk.END, '\n')
111     t1 = thr.Thread(target=chef,args=(float_lt, txt_edit))
112     t1.start()
113     chop_req = True
114
115 # Function to close the GUI
116 # Is called when close-button is pressed
117 def close_window():
118     os.system('fusermount -u /home/bachelor/FTP/files_for_print')

```

```

119     window.destroy()
120
121 # Function to initialize the distribution table
122 # Is called when initialize the stages-button is pressed
123 def init():
124     global axis_list, txt_edit, run_req
125     txt_edit.insert(tk.END, 'Initializing stages\n')
126     txt_edit.update()
127     txt_edit.yview_pickplace('end')
128     dts.initialize(axis_list)
129     run_req = True
130     txt_edit.insert(tk.END, 'Initialisation done\n')
131     txt_edit.update()
132     txt_edit.yview_pickplace('end')
133
134 # Function to show the video
135 # Is called when show video button is pressed
136 def show_video():
137     global t2, stop_video, lt
138     stop_video = False
139     stop_video_args = [lambda : stop_video]
140     t2 = thr.Thread(target=run_video, args=(stop_video_args))
141     t2.start()
142
143 # Function to close the video
144 # Is called when the close video button is pressed
145 def close_video():
146     global t2, stop_video
147     stop_video = True
148
149 # Function to run the distribution table
150 # Is called when run-button is pressed
151 def run_eitri():
152     global pin_IN, pin_OUT, pin_const_1, pin_coater, run_pin, run, axis_list,
run_req, txt_edit, plot_req, lt, chop_req
153
154     # If all requirements are met
155     # Calls the drive_the_stages script
156     if run_req == True and plot_req == True and chop_req == True:
157         txt_edit.insert(tk.END, 'Running...\n')
158         txt_edit.update()
159         txt_edit.yview_pickplace('end')
160         dts.main_part(pin_IN, pin_OUT, pin_const_1, run_pin, run, axis_list,
txt_edit, lt, pin_coater)
161         dts.home_all_devices(axis_list, 14)
162         txt_edit.insert(tk.END, 'Printing complete\n')
163         txt_edit.update()
164         txt_edit.yview_pickplace('end')
165         delete_files_from_FTP_server(txt_edit)
166         GPIO.cleanup()
167     # Gives output of what is wrong if all requirements are not met
168     else:
169         if run_req == False:
170             txt_edit.insert(tk.END, 'Initialize before running\n')
171             txt_edit.update()
172             txt_edit.yview_pickplace('end')
173         if plot_req == False:
174             txt_edit.insert(tk.END, 'Set layer thickness before running\n')
175             txt_edit.update()
176             txt_edit.yview_pickplace('end')

```

```

177         if chop_req == False:
178             txt_edit.insert(tk.END, 'Convert files before running\n')
179             txt_edit.update()
180             txt_edit.yview_pickplace('end')
181
182 # Unmounts the FTP folder
183 try:
184     os.system('fusermount -u /home/bachelor/FTP/files_for_print')
185 except:
186     pass
187
188 open('communication_log.txt', 'w').close() # Starts the communication logging
189
190 # Defining variables for running the program
191 run_pin = 'True' #
192 run = True
193 ftp_state = 2
194
195 # Defining variables for GUI-safety
196 plot_req = False
197 run_req = False
198 chop_req = False
199 calc_gui = 0
200 cam = True
201
202 #Defining the GPIO-pins
203 pin_IN = 18 # Input GPIO-pin. Pin to get value from laser
204 pin_OUT = 16 # Output GPIO-pin. Pin to send value to laser
205 pin_const_1 = 22 # Output GPIO-pin. For sending constant value to a relay
206 pin_coater = 15
207 GPIO.setmode(GPIO.BOARD)
208 GPIO.setup(pin_IN, GPIO.IN) # Sets up the input-pin
209 GPIO.setup(pin_coater, GPIO.OUT)
210 GPIO.setup(pin_OUT, GPIO.OUT) # Sets up the output-pins
211 GPIO.setup(pin_const_1, GPIO.OUT)
212
213 GPIO.output(pin_const_1, GPIO.HIGH) # Sets the constant pin high
214
215 Library.enable_device_db_store()
216
217 # If there is connection with the laser
218 try:
219     with Connection.open_serial_port("/dev/ttyUSB0") as connection: # Sets up
communication with the stages
220         axis_list = dts.get_devices(connection, pin_OUT) # Collects the axes for
communication
221
222         # Builds the GUI-main window
223         window = tk.Tk()
224         window.title('Eitri')
225         window.geometry('1000x800')
226
227         window.resizable(width=True, height=True)
228
229         # Builds info-box
230         frame_0 = tk.Frame(
231             master=window,
232             relief=tk.RAISED,
233             borderwidth=1
234         )

```

```

235     frame_0.pack(expand=False)
236     info_header = tk.Label(master=frame_0, width=80, text=f"Info")
237     info_lable_1 = tk.Label(master=frame_0, width=80, text=f"Step 1: Set
variables")
238     info_lable_2 = tk.Label(master=frame_0, width=80, text=f"Step 2:
Initialize")
239     info_lable_3 = tk.Label(master=frame_0, width=80, text=f"Step 3: Convert the
file")
240     info_lable_4 = tk.Label(master=frame_0, width=80, text=f"Step 4: Run")
241     info_header.pack()
242     info_lable_1.pack()
243     info_lable_2.pack()
244     info_lable_3.pack()
245     info_lable_4.pack()
246
247     # Builds parameters-box
248     frame_1 = tk.Frame(
249         master=window,
250         relief=tk.RAISED,
251         borderwidth=1
252     )
253     frame_1.pack(expand=False)
254
255     label_1 = tk.Label(master=frame_1, width=80, text=f"Parameters")
256     label_1.pack()
257     lt = tk.Label(pady=5, master=frame_1, text=f"Layer thickness[mm]: ")
258     lt.pack(side=tk.LEFT)
259     lt_entry = tk.Entry(master=frame_1)
260     lt_value = tk.Entry(master=lt_entry, width=10)
261     lt_entry.pack(side=tk.LEFT)
262     lt_entry.bind('<1>', click_entry)
263
264     label_lt = tk.Label(master=frame_1, padx=5, text=f"Current layer thickness:
NONE")
265     label_lt.pack(side=tk.LEFT)
266
267     # Builds Progress-box
268     frame_2 = tk.Frame(
269         master=window,
270         relief=tk.RAISED,
271         borderwidth=1
272     )
273     frame_2.pack(expand=False)
274
275     label_2 = tk.Label(pady=5, width=80, master=frame_2, text=f"Progress")
276     label_2.pack()
277     txt_edit = tk.Text(master=frame_2, height=10, width=80)
278     txt_edit.pack()
279
280     # Builds Preview-box
281     frame_4 = tk.Frame(
282         master=window,
283         relief=tk.RAISED,
284         borderwidth=1
285     )
286     frame_4.pack(expand=False)
287
288     label_4 = tk.Label(pady=5, width=80, master=frame_4, text=f"Preview")
289     label_4.pack()
290

```

```
291     btn_preview= tk.Button(  
292         master=frame_4,  
293         text='Show preview',  
294         command=plot_show,  
295         height=2,  
296         width=10  
297     )  
298     btn_preview.pack()  
299  
300     # Builds files-box  
301     frame_3 = tk.Frame(  
302         master>window,  
303         relief=tk.RAISED,  
304         borderwidth=1  
305     )  
306     frame_3.pack(expand=False)  
307  
308     label_3 = tk.Label(pady=5, width=80, master=frame_3, text=f"Files")  
309     label_3.pack()  
310     btn_files = tk.Button(  
311         master=frame_3,  
312         text='Convert file',  
313         command=chop_files,  
314         height=2,  
315         width=10  
316     )  
317     btn_files.pack()  
318  
319     # Builds buttons-box  
320     frame_5 = tk.Frame(  
321         master>window,  
322         relief=tk.RAISED,  
323         borderwidth=1  
324     )  
325     frame_5.pack(expand=False)  
326  
327     btn_init= tk.Button(  
328         master=frame_5,  
329         text='Initialize table',  
330         command=init,  
331         height=2,  
332         width=10  
333     )  
334  
335     btn_init.pack(side=tk.LEFT)  
336  
337     btn_run= tk.Button(  
338         master=frame_5,  
339         text='Run',  
340         command=run_eitri,  
341         bg='green',  
342         fg='white',  
343         height=2,  
344         width=10  
345     )  
346  
347     btn_run.pack(side=tk.LEFT)  
348  
349     btn_close= tk.Button(  
350         master=frame_5,
```

```

351         text='Close',
352         command=close_window,
353         bg='red',
354         fg='white',
355         height=2,
356         width=10
357     )
358
359     btn_close.pack(side=tk.LEFT)
360
361     btn_eject= tk.Button(
362         master=frame_5,
363         text='Eject print disk',
364         command=eject_disk,
365         height=2,
366         width=10
367     )
368
369     btn_eject.pack(side=tk.LEFT)
370
371     btn_video= tk.Button(
372         master=frame_5,
373         text='Show video',
374         command=show_video,
375         height=2,
376         width=10
377     )
378
379     btn_video.pack(side=tk.LEFT)
380
381     btn_c_video= tk.Button(
382         master=frame_5,
383         text='Close video',
384         command=close_video,
385         height=2,
386         width=10
387     )
388
389     btn_c_video.pack(side=tk.LEFT)
390
391     # Tries to connect the FTP folder to the laser
392     if os.system('curlftpfs -o nonempty @192.168.1.160
/home/bachelor/FTP/files_for_print') == 0:
393         txt_edit.insert(tk.END, 'Connected to laser\n')
394         txt_edit.update()
395         txt_edit.yview_pickplace('end')
396     else:
397         txt_edit.insert(tk.END, 'No connection or with laser\n')
398         txt_edit.update()
399         txt_edit.yview_pickplace('end')
400
401     # runs_the script for moving files to the FTP-folder
402     run_moving_files(txt_edit)
403     lt = lt_entry.get()
404     window.mainloop()
405
406 # If there is no connection with the laser
407 except zaber_motion.exceptions.serial_port_busy_exception.SerialPortBusyException:
408     print('Ensure laser system is powered on completely and connected to router')
409

```

I Controlling the PDS script

5/20/22, 9:04 PM

drive_the_stages.py

```
1 # All code used to control the old recoater is commented out
2
3 # Imports all libraries
4 # Zaber_motion controls the Zaber stages
5 # Time is not implemented and was used for testing
6 # RPi.GPIO is used for controlling the GPIO-pins on the Raspberry Pi
7 from zaber_motion import Library, log_output_mode, Units
8 from zaber_motion.ascii import Connection
9 from zaber_motion.gcode import OfflineTranslator
10 import time
11 import RPi.GPIO as GPIO
12 from moving_files import run_moving_files, delete_files_from_FTP_server
13 import tkinter as tk
14 import os
15
16 # Function moves a device to a position compared to the home position (all of the
17 # devices in our project have only one axis)
18 def move_device_abs(length, axis):
19     print('moves', axis)
20     axis.move_absolute(length, Units.LENGTH_MILLIMETRES)
21
22 # Function moves a device to a position compared to the devices relative position
23 # (all of the devices in our project have only one axis)
24 def move_device_rel(length, axis):
25     axis.move_absolute(length, Units.LENGTH_MILLIMETRES)
26
27 # Function initializes the stages. This is to avoid potential startup errors
28 def initialize(axis_list):
29     home_all_devices(axis_list, 0)
30     # axis_list[2].move_max()
31     # axis_list[2].move_min()
32     home_all_devices(axis_list, 14.35)
33
34 # Function homes all devices (moves all devices to their home positions)
35 # The home position of axis[0] (the printing chamber) has been changed to 14
36 # millimetres from original home position.
37 # This is because the printing chamber must start at the top.
38 def home_all_devices(axis_list, axis_0_home_mm):
39     axis_list[0].move_absolute(axis_0_home_mm, Units.LENGTH_MILLIMETRES)
40     axis_list[1].home()
41     # axis_list[2].home()
42
43 # Function moves the printing chamber
44 def move_bed_1(length, axis_list):
45     axis_list[0].move_absolute(length, Units.LENGTH_MILLIMETRES)
46
47 # Function moves the powder chamber
48 def move_bed_2(length, axis_list):
49     axis_list[1].move_absolute(length, Units.LENGTH_MILLIMETRES)
50
51 # Function moves the recoater
52 def move_recoater(length, axis_list):
53     axis_list[2].move_absolute(length, Units.LENGTH_MILLIMETRES)
54
55 def drive_cd():
56     global pin_out_coater
57     GPIO.output(pin_out_coater, GPIO.HIGH)
58     time.sleep(0.5)
59     GPIO.output(pin_out_coater, GPIO.LOW)
```

localhost:4649/?mode=python

1/3


```

57     time.sleep(2.25)
58     GPIO.output(pin_out_coater, GPIO.HIGH)
59     time.sleep(0.5)
60     GPIO.output(pin_out_coater, GPIO.LOW)
61     time.sleep(2.25)
62
63
64 def get_devices(connection, pin_OUT):
65     # While loop for the main code. Runs while connected to the powder distribution-
table
66     # Gets the devices in a list
67     device_list = connection.detect_devices()
68     print("Found {} devices".format(len(device_list)))
69
70     # Setting logging file
71     Library.set_log_output(log_output_mode.LogOutputMode.FILE,
"communication_log.txt")
72
73     # Finds every device in the device list and gives it too a descriptive variable
74     bed_1 = device_list[0]
75     bed_2 = device_list[1]
76     #     recoater = device_list[2]
77
78     # Finds every axis in the device list and gives it too a descriptive variable.
And adds them all to a list
79     #     recoater_axis = recoater.get_axis(1)
80     bed_1_axis = bed_1.get_axis(1)
81     bed_2_axis = bed_2.get_axis(1)
82     #     axis_list = [bed_1_axis, bed_2_axis, recoater_axis]
83     axis_list = [bed_1_axis, bed_2_axis]
84     GPIO.output(pin_OUT, GPIO.LOW)
85     return axis_list
86
87
88 def main_part(pin_IN, pin_OUT, pin_const_1, run_pin, run, axis_list, txt_edit, lt,
pin_coater):
89     global pin_out_coater
90     lt = float(lt)
91     pin_out_coater = pin_coater
92
93     width_microm = 30 # This variable tells the width of each slice and is
implemented in the movement of the chambers as moveemnt per slice. Value is
collected from the laser.
94     home_bed_1 = 14.4 # Defines the starting position of the printing chamber
95     move_bed_1_mm = home_bed_1 # Variable to control the movement of the printing
chamber
96     move_bed_2_mm = 1.5 # Variable to control the movement of the chambers
97     i = 0 # Variable to keep count of the number of layers finished
98
99     f = open('/home/bachelor/FTP/files_for_print/DXF_config.txt', 'r')
100    f_cont = f.read()
101    f_split = f_cont.split(':')
102    max_count = int(f_split[1].split('\n')[0])
103
104    # Checks for connection with the laser
105    if os.system('ping -c 1 192.168.1.160') == 0:
106        # Loop that runs while the printing process is underway.
107        while run_pin == 'True': # GPIO.input(run_pin) == 1:
108            # If the laser says that it is the powder distribution-tables turn to
move.

```

```
109     GPIO.output(pin_OUT, GPIO.LOW)
110     if GPIO.input(pin_IN) == 1:
111         # Moves the chambers and the recoater
112         i += 1
113         txt_edit.insert(tk.END, f'Running layer {i}\n')
114         txt_edit.update()
115         txt_edit.yview_pickplace('end')
116
117         move_bed_1_mm -= lt
118         move_bed_2_mm += lt*1.5
119
120         move_bed_1(move_bed_1_mm, axis_list)
121         move_bed_2(move_bed_2_mm, axis_list)
122
123         drive_cd()
124
125         #         axis_list[2].move_max()
126         #         axis_list[2].move_min()
127
128         txt_edit.insert(tk.END, f'Done with layer {i}\n')
129         txt_edit.update()
130         txt_edit.yview_pickplace('end')
131
132
133         GPIO.output(pin_OUT, GPIO.HIGH)
134         time.sleep(0.1)
135
136         if i >= max_count-1:
137             run_pin = False
138             GPIO.output(pin_OUT, GPIO.LOW)
139
140     # If there is no connection with the laser
141     else:
142         txt_edit.insert(tk.END, 'No connection or with laser\n')
143         txt_edit.update()
144         txt_edit.yview_pickplace('end')
145
```

J Create and show plot script

5/20/22, 9:29 PM

Damascus.py

```
1 # Imports libraries
2 import numpy as np
3 from stl import mesh
4 from mpl_toolkits import mplot3d
5 from matplotlib import pyplot
6
7 # Function to find intersections
8 def isect_line_plane_v3(p0, p1, elevation):
9     epsilon = 1e-6
10    planePoint = np.array([0,0,elevation])
11    planeNormal = np.array([0,0,1])
12
13    LineDirection = p1-p0
14    LinePoint = p0
15    dot = np.dot(planeNormal, LineDirection)
16    if abs(dot) > epsilon:
17        # The factor of the point between p0 -> p1 (0 - 1)
18        # if 'si' is between (0 - 1) the point intersects with the segment.
19        # Otherwise:
20        # < 0.0: behind p0.
21        # > 1.0: infront of p1.
22        w = LinePoint - planePoint
23        si = -planeNormal.dot(w) / dot
24        u = LineDirection*si
25        impact = p0 + u
26        if (p0[0] < impact[0] > p1[0] or p0[0] > impact[0] < p1[0]) or p0[1] <
impact[1] > p1[1] or p0[1] > impact[1] < p1[1]:
27            return None
28        return impact
29    # The segment is parallel to plane.
30    return None
31
32 # Function checks if lines with the correct z values intersect with the plane
33 def slicer(list,h):
34    lines = []
35    LAB = []
36    LAC = []
37    LBC = []
38    if np.all(list == list[2,:], axis = 0)[2]:
39        return
40    corners = list[list[:,-1].argsort()[::-1]]
41    A = corners[0]
42    B = corners[1]
43    C = corners[2]
44
45    #Check if the plan intersect the line between A and B.
46    AB = isect_line_plane_v3(A,B,h)
47    if np.all(AB) != None:
48        for val in AB:
49            LAB.append(val)
50        lines.append(LAB)
51
52    #Check if the plan intersect the line between A and C.
53    AC =isect_line_plane_v3(A,C,h)
54    if np.all(AC) != None:
55        for val in AC:
56            LAC.append(val)
57        lines.append(LAC)
58
```

localhost:4649/?mode=python

1/2

```
59     #Check if the plan intersect the line between B and C.
60     BC = isect_line_plane_v3(B,C,h)
61     if np.all(BC) != None:
62         for val in BC:
63             LBC.append(val)
64             lines.append(LBC)
65
66     if len(lines) >= 2:
67         pyplot.plot([lines[0][0],lines[1][0]],[lines[0][1],lines[1][1]],[lines[0]
68 [2],lines[1][2]])
69 # Main function to be called from Eitri (main program)
70 # Function to create the plot
71 def run_damascus(stl_file, lt, txt_edit, figure):
72     # Load the STL files.
73     lt = float(lt)
74     your_mesh = mesh.Mesh.from_file(stl_file)
75     height = 0
76     for val in your_mesh.z:
77         if max(val) > height:
78             height = max(val)
79     current_height = 0
80     load = 0
81
82     # Build the plot
83     while current_height < height:
84         for i in range(0,len(your_mesh.vectors)):
85             matrix = your_mesh.vectors[i]
86             cords = matrix[:,0],matrix[:,1],matrix[:,2]
87             vectors = np.array([cords[0],cords[1],cords[2]])
88             if (np.all(np.round(matrix[:,2],2) > current_height) or
89                 np.all(np.round(matrix[:,2],2) < current_height)):
90                 pass
91             else:
92                 slicer(your_mesh.vectors[i],current_height)
93             load += 1
94             current_height = round(current_height+lt,3)
95     pyplot.show()
96
```

K Converting STL file to DXF files script

5/20/22, 9:21 PM

Mandoline.py

```
1 # Imports libraries
2 import os, os.path
3 import os, sys
4 import numpy as np
5 import ezdxf
6 from stl import mesh
7 import tkinter as tk
8
9 # Function creates and writes the progress bar
10 def print_percent_done(index, total, title, bar_len=50):
11     global progress_file
12     '''
13     index is expected to be 0 based index.
14     0 <= index < total
15     '''
16     percent_done = (index+1)/total*100
17     percent_done = round(percent_done,1)
18     if percent_done > 100:
19         percent_done = 100.0
20
21     done = round(percent_done/(100/bar_len))
22     togo = bar_len-done
23
24     done_str = '█'*int(done)
25     togo_str = '░'*int(togo)
26
27     last_index = progress_file.index('end')
28
29     last_index_int = float(last_index)-1.0
30
31     progress_file.insert(str(last_index_int),f'[{done_str}{togo_str}]
32     {percent_done}% done\n')
33     progress_file.delete(str(last_index_int-1.0), str(last_index_int))
34
35     if round(percent_done) == 100:
36         progress_file.insert(tk.END, f'✅{title}: done slicing\n')
37
38 # Function finds the intersections
39 def isect_line_plane_v3(p0, p1, elevation):
40     epsilon = 1e-6
41     planePoint = np.array([0,0,elevation])
42     planeNormal = np.array([0,0,1])
43
44     LineDirection = p1-p0
45     LinePoint = p0
46     dot = np.dot(planeNormal, LineDirection)
47     if abs(dot) > epsilon:
48         # The factor of the point between p0 -> p1 (0 - 1)
49         # if 'si' is between (0 - 1) the point intersects with the segment.
50         # Otherwise:
51         # < 0.0: behind p0.
52         # > 1.0: infront of p1.
53         w = LinePoint - planePoint
54         si = -planeNormal.dot(w) / dot
55         u = LineDirection*si
56         impact = p0 + u
57         if (p0[0] < impact[0] > p1[0] or p0[0] > impact[0] < p1[0]) or p0[1] <
58         impact[1] > p1[1] or p0[1] > impact[1] < p1[1]:
```

localhost:4649/?mode=python

1/3

```

58         return None
59     return impact
60     # The segment is parallel to plane.
61     return None
62
63 # Function checks if lines with the correct z values intersect with the plane
64 def slicer(list,h):
65     list = np.round(list,3)
66     #If all of the z values are above or below the current height.
67     #Then no calculation is needed.
68     if np.all(list == list[:,2], axis = 0)[2]:
69         return
70     if not np.any(h >= list[:,2], axis = 0) or not np.any(h <= list[:,2], axis = 0):
71         return
72     #Retrives the cordiantes for A,B and C.
73     corners = list[list[:, -1].argsort()[::-1]]
74     A = corners[0]
75     B = corners[1]
76     C = corners[2]
77     fragments = []
78     LAB = []
79     LAC = []
80     LBC = []
81     #Check if the plan intersect the line between A and B.
82     AB = isect_line_plane_v3(A,B,h)
83     if type(AB) != type(None):
84         for val in AB:
85             LAB.append(val)
86         fragments.append(LAB[0:2])
87
88     #Check if the plan intersect the line between A and C.
89     AC = isect_line_plane_v3(A,C,h)
90     if type(AC) != type(None):
91         for val in AC:
92             LAC.append(val)
93         fragments.append(LAC[0:2])
94
95     #Check if the plan intersect the line between B and C.
96     BC = isect_line_plane_v3(B,C,h)
97     if type(BC) != type(None):
98         for val in BC:
99             LBC.append(val)
100         fragments.append(LBC[0:2])
101     return fragments
102
103 # Function to check for STL files
104 def countImage():
105     #Retrivers the path to current directory
106     source_dir = "/home/bachelor/FTP/files_for_transfer"
107     files = []
108     #Finds a STL file.
109     for file in os.listdir(source_dir):
110         if file.endswith(".stl"):
111             stl_file = os.path.join(source_dir, file)
112             print(stl_file)
113             files.append(stl_file)
114             #Move return outside for: to get all STL files in directory.
115             return files,source_dir
116
117 # Function to convert files to DXF

```

```

118 def convertImage(file,lt):
119     final_dir = '/home/bachelor/FTP/files_for_print'
120     # Retrivers the tringular polygons from the STL files.
121     for filename in os.listdir(final_dir):
122         os.remove(final_dir+'/'+filename)
123     your_mesh = mesh.Mesh.from_file(file)
124     height = 0
125     for val in your_mesh.z:
126         if max(val) > height:
127             height = max(val)
128     nmb = height/lt
129     layers = 0
130     current_height = 1e-6
131     while current_height < height:
132         layers = layers+1
133         print_percent_done(layers,nmb,file.split("\\")[1])
134         name = str(layers)+".dxf"
135         lines = []
136         for i in range(0,len(your_mesh.vectors)):
137             if (np.all(np.round(your_mesh.vectors[i][:,2],2) > current_height) or
138                 np.all(np.round(your_mesh.vectors[i][:,2],2) < current_height)):
139                 pass
140             else:
141                 lines.append(slicer(your_mesh.vectors[i],current_height))
142         dwg = ezdxf.new("R2007")
143         msp = dwg.modelspace()
144         dwg.layers.new(name="RofinStandard")
145         exp_ind = final_dir+'/'+name
146         for ctr in lines:
147             if ctr != None:
148                 msp.add_lwpolyline(ctr)
149         dwg.saveas(exp_ind)
150         dwg.save()
151         current_height += lt
152     fullname = final_dir+"/"+DXF_config.txt"
153     f = open(fullname,"w")
154     f.write("LayerCount:"+str(int(nmb+1))+"\n")
155     f.write("LayerThickness:"+str(lt)+"\n")
156     f.close
157
158 # Function to be called from Eitri (main program)
159 def chef(SliceThickness, txt_edit):
160     global progress_file
161
162     progress_file = txt_edit
163     images,parent_dir = countImage()
164     nmb = 0
165     for files in images:
166         convertImage(files,SliceThickness)

```

L Moving files from USB drive to folder script

5/20/22, 9:13 PM

moving_files.py

```
1 # Importing necessary libraries
2 import os
3 import time
4 import shutil
5 import tkinter as tk
6
7 # Checks for a USB disk
8 # Waits until USB disk is found
9 # Moves the files from the USB drive into the correct folder
10 # For the slicing script to collect
11 def run_moving_files(txt_edit):
12
13     run = True # Setting up a run variable
14
15     directory = '/media/bachelor' # Empty directory in wich the disk will show up
16
17     txt_edit.insert(tk.END, f'Waiting for data\n')
18     txt_edit.update()
19     txt_edit.yview_pickplace('end')
20
21     # While loop to check for disk in empty directory
22     while run == True:
23         for disk in os.listdir(directory):
24             if disk:
25                 run = False
26
27     time.sleep(1)
28
29     FTP_path = '/home/bachelor/FTP/files_for_transfer'
30     current_directories_list = []
31
32     for filename in os.listdir(FTP_path):
33         current_directories_list.append(str(filename))
34
35
36     for filename in os.listdir(directory+'/'+str(disk)):
37         disk_path = directory+'/'+str(disk)+'/'+str(filename)
38
39         # Checks if files is already in the FTP folder
40         # If it is. The file is not moved
41         if '.stl' not in disk_path:
42             txt_edit.insert(tk.END, f'{disk_path} not moved\n')
43             txt_edit.update()
44             txt_edit.yview_pickplace('end')
45
46         # Moves if the path is a file
47         else:
48             if os.path.isfile(disk_path):
49                 if filename not in current_directories_list:
50                     shutil.copy(disk_path, FTP_path)
51                     txt_edit.insert(tk.END, f'{disk_path} moved to FTP folder\n')
52                     txt_edit.update()
53                     txt_edit.yview_pickplace('end')
54                 else:
55                     txt_edit.insert(tk.END, f'{disk_path} already in FTP folder\n')
56                     txt_edit.update()
57                     txt_edit.yview_pickplace('end')
58
59         # Moves if the path is a directory
```

localhost:4649/?mode=python

1/2


```
60         elif os.path.isdir(disk_path):
61             if filename not in current_directories_list:
62                 shutil.copytree(disk_path, FTP_path+'/'+str(filename))
63                 txt_edit.insert(tk.END, f'{disk_path} moved to FTP folder\n')
64                 txt_edit.update()
65                 txt_edit.yview_pickplace('end')
66             else:
67                 txt_edit.insert(tk.END, f'{disk_path} already in FTP folder\n')
68                 txt_edit.update()
69
70 # Deletes files from the folder
71 def delete_files_from FTP_server(txt_edit):
72
73     FTP_path = '/home/bachelor/FTP/files_for_transfer'
74
75     for filename in os.listdir(FTP_path):
76         # Deletes if path is a file
77         if os.path.isfile(FTP_path+'/'+str(filename)):
78             os.remove(FTP_path+'/'+str(filename))
79             txt_edit.insert(tk.END, f'File{FTP_path}/{str(filename)} deleted\n')
80             txt_edit.update()
81             txt_edit.yview_pickplace('end')
82
83         # Deletes is path is a folder
84         elif os.path.isdir(FTP_path+'/'+str(filename)):
85             shutil.rmtree(FTP_path+'/'+str(filename))
86             txt_edit.insert(tk.END, f'Directory{FTP_path}/{str(filename)} and file
87 contents deleted\n')
88             txt_edit.update()
89             txt_edit.yview_pickplace('end')
```

M Numbers keyboard script

5/20/22, 9:16 PM

numbers_table.py

```
1 # import tkinter (everything)
2 from tkinter import *
3
4
5 # Function to update the entry-textbox
6 def press(num):
7     global expression, equation, lt_entryen
8
9     expression = str(num)
10
11     # update variable
12     equation.set(expression)
13
14     # Get variable from insert box
15     lt_streng = equation.get()
16
17     lt_entryen.insert(END, lt_streng)
18
19 # Function to clear the input
20 def clear():
21     global expression
22     lt_entryen.delete(0, END)
23
24 # Main function to be called from eitri-script
25 def run_numbers(win, lt_entry, calc_gui):
26     global expression, equation, gui, lt_entryen
27     lt_entryen = lt_entry
28
29     lt_entryen.delete(0, END)
30
31     # Sets up the entry as variable
32     gui = calc_gui
33     expression = ""
34     equation = StringVar()
35
36     # Create all the buttons with a number or functionality attached to it
37     button1 = Button(gui, text=' 1 ', fg='black', bg='grey',
38                     command=lambda: press(1), height=3, width=20)
39     button1.grid(row=2, column=0)
40
41     button2 = Button(gui, text=' 2 ', fg='black', bg='grey',
42                     command=lambda: press(2), height=3, width=20)
43     button2.grid(row=2, column=1)
44
45     button3 = Button(gui, text=' 3 ', fg='black', bg='grey',
46                     command=lambda: press(3), height=3, width=20)
47     button3.grid(row=2, column=2)
48
49     button4 = Button(gui, text=' 4 ', fg='black', bg='grey',
50                     command=lambda: press(4), height=3, width=20)
51     button4.grid(row=3, column=0)
52
53     button5 = Button(gui, text=' 5 ', fg='black', bg='grey',
54                     command=lambda: press(5), height=3, width=20)
55     button5.grid(row=3, column=1)
56
57     button6 = Button(gui, text=' 6 ', fg='black', bg='grey',
58                     command=lambda: press(6), height=3, width=20)
59     button6.grid(row=3, column=2)
```

localhost:4649/?mode=python

1/2

```
60
61 button7 = Button(gui, text=' 7 ', fg='black', bg='grey',
62                  command=lambda: press(7), height=3, width=20)
63 button7.grid(row=4, column=0)
64
65 button8 = Button(gui, text=' 8 ', fg='black', bg='grey',
66                  command=lambda: press(8), height=3, width=20)
67 button8.grid(row=4, column=1)
68
69 button9 = Button(gui, text=' 9 ', fg='black', bg='grey',
70                  command=lambda: press(9), height=3, width=20)
71 button9.grid(row=4, column=2)
72
73 button0 = Button(gui, text='.', fg='black', bg='grey',
74                  command=lambda: press('.'), height=3, width=20)
75 button0.grid(row=5, column=0)
76
77 clear_b = Button(gui, text='Clear', fg='black', bg='grey',
78                  command=clear, height=3, width=20)
79 clear_b.grid(row=6, column='1')
80
81 Decimal= Button(gui, text=' 0 ', fg='black', bg='grey',
82                  command=lambda: press(0), height=3, width=20)
83 Decimal.grid(row=5, column=1)
84
```

N Show video script

5/20/22, 9:17 PM

video_file.py

```
1 # Imports libraries
2 import cv2
3
4 # Opens video in a new window
5 def run_video(stop):
6     cam = cv2.VideoCapture(0)
7
8     # Runs the video stream until
9     # close video button is pressed
10    while True:
11        ret, image = cam.read()
12        cv2.imshow('Imagetest',image)
13        k = cv2.waitKey(1)
14        if stop():
15            break
16    cam.release()
17    cv2.destroyAllWindows()
18
```

O VLM script

```
Option Explicit
'LMOSInPortConstants
Const FinishedPowderDistrubution = 10
Const StartPowderDistrubution = 14

Dim state
Sub LaserMarker_ScriptBegin ()
    Dim dxfImport,hs
    Set dxfImport =
lasermarker.drawing.getmosbyname("MO_DXF_IMPORT").item(1)
    Set hs = dxfImport.HatchSettings
    hs.HatchAngle = 0
    Dim i,var,Layers
    'arbitrary number of layers
    'ending the print is controlled by the print table.
    Layers = 5000
    For i = 1 To Layers
        'Sends signal to distrubute a new powder layer.
        LaserMarker.WriteIOBit "StartPowderDistrubution",1
        'Waiting for a signal that the layer has been completed.
        LaserMarker.WaitOnIOBit "FinishedPowderDistrubution", 1, 90000
        'Writing the state of the signal to the "state" variable.
        Lasermarker.ReadIOBit "FinishedPowderDistrubution", state
        'Stops the print job if failed to receive signal.
        If state = False Then
            lasermarker.scriptutils.message "Job ended by print
table."

            LaserMarker.StopMarking()
            LaserMarker.WriteIOBit "StartPowderDistrubution",0
            i = Layers
        Else
            LaserMarker.WriteIOBit "StartPowderDistrubution",0
            'Incrementally work through the layers in the print folder.
            dxfImport.value = cstr(i) & ".dxf"
            'Print the current layer being printed.
            lasermarker.scriptutils.message dxfImport.value
            'Notifies the laser to start printing.
            dxfImport.Mark
            'Cache Purge.
            LaserMarker.Synchronize
            'Rotate the hatching pattern 67 degrees every layer.
            hs.HatchAngle = hs.HatchAngle + 67
            End If
        Next
    'When done set the import value
    'so the next print starts at 1.
    dxfImport.value = cstr(1) & ".dxf"
End Sub
```

Development of metal 3D printer

By Mathias Kommedal, Karl Peder Mørkeseth, Audun Mostad and Jonatan Lærdahl

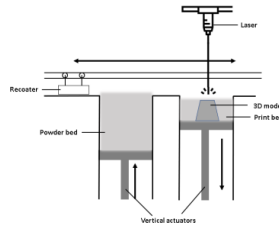
May 2022

Motivation

Metal 3D printers are rapidly increasing in popularity. The material used in the printing process are limited. SINTEF wants to research new metal alloys, which can be used in 3D printing. The technology is new and expensive. Building a new 3D printer which is capable of printing small samples gives SINTEF the opportunity to research this technology. A powerful engraving laser was available for melting metal powder, and a powder distribution system had previously been made. The challenge was to make a communication system between the two units and to automate the process. The project was completed by four students as a bachelor thesis in electrical engineering.

Metal 3D printing

The printing is achieved by melting metal powder with a high energy source (the laser). A 3D model is printed layer by layer. The powder distribution system (PDS) has two chambers, one powder bed and one printing bed. These are adjusted vertically by actuators. The powder bed is slightly lifted, and the print bed is slightly lowered. A recoater sweeps over the PDS which leaves a thin layer of powder in the print bed. The laser reads a DXF file which describes how the layer shall be melted. This process is repeated layer by layer until the model is completed.



Framing

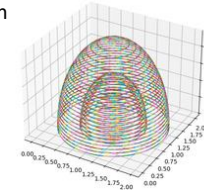
The laser was mounted in a large enclosure. By removing an external control panel and the front housing the size has been significantly reduced. New LED-strips have been attached to the ceiling for optimal lighting. The various measures have led to an optimized volume and weight reduction. The system now utilizes the space available by unnecessary parts being removed and the control panel being relocated.

Control Panel

The control panel was moved to the rear of the framing. The rear wall was partly removed by an angle grinder. A special made housing was mounted. A new screen and keyboard were added to operate the laser. Holes in the new designed housing made place for a new touch screen and a USB port. The touch screen is integrated with a graphical user interface (GUI) made by the group. The GUI is used to start the print process. It also provides the option to view the print process live through a camera mounted inside the framing.

Slicing script

The laser can read a DXF file and melt a layer of powder accordingly. Converting 3D models (STL file) to several DXF files is therefore an essential part. A python script takes a STL file and the desired layer thickness as a variable in the GUI. By using vector algebra, the script can find the outline, and potentially an inner rim like the figure shows. The script works its way from the bottom to the top of the model, where each layer is saved as a DXF file.



Communication

The communication is controlled by a Raspberry Pi (RPI). This is achieved with a relay circuit. The laser operates with 24V I/O, and the RPI I/O pins use 3.3V. A relay circuit prevents the RPI from getting fried. The RPI tells the PDS to coat a layer using a python library for communicating with the devices on the PDS. The PDS responds to the RPI when it is done. Next, the RPI sends a 3.3V signal via a relay circuit to the laser. The laser melts the first layer according to information in the first DXF file and sends a 24V signal through a relay circuit when it is finished. This process repeats itself until all the DXF files are used. This results in the finished 3D model in metal.

The end product

The product works as intended. The system is an automated process from the moment a STL file is uploaded, and the operator pushes the "RUN" button on the GUI and the "Start marking" button in the VisualLaserMarker (VLM) software. The new framing protects the spectators from dangerous light radiation. They can control and watch the process from the new control panel. The PDS is attached in a sealed chamber, which is filled with argon gas before printing. Argon gas protects the powder from oxidation, which would cause fractures in the metal. The group has successfully fulfilled their tasks and are very satisfied with the results. We would like to thank our supervisor Eivind Johannes Øvrelid from SINTEF and Bendik Sægrov-Sorte for advice and technical support and our supervisor at NTNU, Sigurd Gosse.



