Freie Universität Berlin

Masterarbeit am Institut für Informatik der Freien Universität Berlin in Zusammenarbeit mit dem Fachgebiet SVFS der TU-Berlin

Investigating the complexity, improvements, and limitations of a 4-axis 3D printer and a correspondingly implemented slicing strategy compared to a conventional 3-axis 3D printer with planar slicing on the example of a 3D printer modification by FullControl.

Malte Schlichting
Enrolment number: 5335739
maltes99@zedat.fu-berlin.de

Advisor:	Prof. Dr. Günter Rote
First examiner:	Prof. Dr. Günter Rote
Second examiner:	Prof. Dr. Christina Völlmecke

Berlin, June 11, 2025

Abstract

This thesis explores the modification of a 4-axis 3D printer by FullControl, with a particular focus on enabling non-planar and non-horizontal printing through the implementation of a B-axis slicing pipeline. A Python Jupyter Notebook is developed to generate the corresponding G-code, utilizing all four axes. The work presents the conceptual design, implementation process, and technical challenges involved in creating a slicer for this type of 3D printer. Additionally, the study investigates the complexity, capabilities, and limitations of non-planar fused filament fabrication (FFF) by comparing it with conventional planar printing techniques. A series of experiments are conducted to evaluate the differences and potential benefits of non-planar 4-axis printing.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht. Mir ist bewusst, dass ich, sofern ich zur Erstellung dieser Arbeit KI-basierte Tools verwendet habe, die Verantwortung für eventuell durch die KI generierte fehlerhafte oder verzerrte (bias) Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate trage.

June 11, 2025

Malte Schlichting

Used Tools

If not stated differently, ChatGPT4o (free version) was used on the website https://chatgpt.com for the following purposes: 1. Convert other citation formats into BibTeX with the prompt: "Can you convert this into bibtex: [citation not in BibTeX format]". The output was then manually checked for correctness before use. 2. For minor parts, convert bullet points into whole sentences with the prompt: "Can you convert the following bullet points into one scientific continuous text: [bullet points]". The results were then manually verified for accuracy. It is worth mentioning that the bullet points were written completely by myself. 3. Rewrite my version of the abstract, which I then manually checked for correctness and modified to fit the thesis. The prompt "Can you rewrite the abstract of my master thesis to sound more professional? The abstract is: [my abstract]" was used.

Contents

Abbreviations

3D	three-dimensionl.
AM	additive manufacturing.
DOF	degrees of freedom.
FFF	fused filament fabrication.
GUI	graphical user interface.
STL	standard tessellation language.

List of Figures

List of Tables

1 Introduction

Three-dimensionl (3D) printing and especially fused filament fabrication (FFF) 3D printing, has become one of the most important and widespread technologies for additive manufacturing (AM). This technology enables fast prototyping and is on the way to replace conventional manufacturing technologies such as subtractive manufacturing [1]. In addition to advantages such as low cost production and low energy consumption, conventional FFF 3D printing has various limitations, such as the need for support structures and the production of fragile parts, especially in the direction of z [2]. Most of the downsides are due to the 3-axis design of the 3D printers. This is why the field of non-planar printing and multi-axis printers has emerged in the last few years [3]. An example of a 4-axis printer is the modification of a 3-axis Prusa printer by FullControl [4]. FullControl is an open-source G-code designer by Andrew Gleadall [5]. His 4-axis printer has an added b-axis that enables the print head to turn around the y-axis and print sideways. However, 3-axis printers are more popular than 4-axis printers for a reason. With higher degrees of freedom (DOF) comes higher complexity. A comparison between the original 3-axis Prusa printer and the modified 4-axis version, along with a comprehensive examination of the complexity of implementing a slicer for this type of 3D printer and what its additional capabilities and limitations are will be the content of this master thesis.

1.1 Objective

This work aims to gain a complete understanding of the modified 4-axis 3D printer and the corresponding slicing strategy. The following areas are the main focus of this work.

1. Elaborate the geometrical capabilities and limitations of the 4-axes printer modification by FullControl,
2. showcase the challenges of implementing a non-planar slicer, and
3. investigate, how curved and tilted layers influence the mechanical properties of a 3D printed object.

In order to achieve these goals, recent work about non-planar 3D printing is investigated, and a non-planar slicer that provides a user-friendly usage of the modified 4-axes 3D printer is implemented.

1.2 Structure of this work

In the following sections the process of FFF 3D printing is explained along with an examination of the current state of the art for non-planar 3D printing. These topics are covered in Section 2. The conversion of a Prusa MK3S+ Cartesian 3D printer, introduced by FullControl, is explained, and the process is described in Section 3. In order to be able to use the modified 4-axis printer, a slicer is needed that generates the G-code for the printer. The concept, implementation and challenges of the 4-axis slicer is covered in Section 4. As mentioned above, thorough testing of the capabilities and limitations of the printer and its slicer is the main goal of this thesis. This is the content of Section 5. In Section 6 the results and quality of this thesis are discussed, followed by the conclusion in Section 7 and the prospects for further research in Section 8.

2 State of the art

This work aims to investigate the 4-axis 3D printer modification by FullControl. In order to do so, the basic concepts of 3D printing and slicing have to be understood, as well as the current state of the art in 3D printing of planar and non-planar printing on 3-axis and 4-axis 3D printers. The following sections explain the Cartesian printer design that the original printer, used in this work, has, followed by an illustrated explanation of the concepts of slicing and FFF 3D printing. After that, the relevant papers and slicing strategies are examined for applicability to this work.

2.1 Conventional FFF 3D printing

FFF is an additive manufacturing technique in which filament is extracted and placed on a print bed or a previously extracted filament line layer by layer [6]. This leads to a printed object as the result (see Figure 1). The conventional procedure is to stack planar layers parallel to the xy-plane in the z direction, as shown in Figure 1b [7]. There are several aspects that must be met to meet the additive manufacturing criteria according to Mwema [8].

1. Computer aided design (CAD) software must be used for designing a 3D model,
2. a slicer must generate a toolpath, and
3. a machine must produce a physical product that matches the 3D model designed.

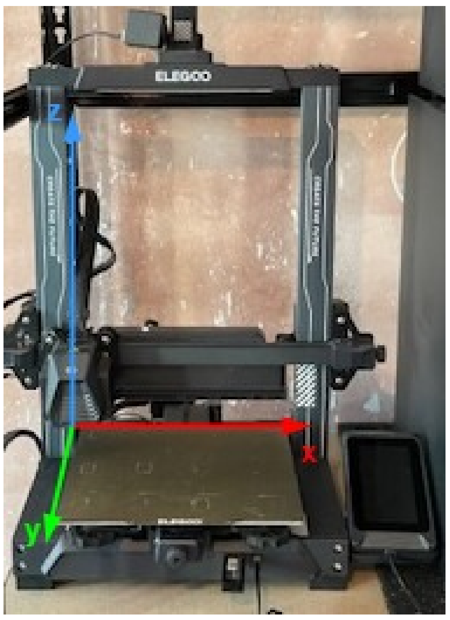
In the next sections, the basic concepts of FFF 3D printing and slicing are presented, as well as the 3D printer that is used for the 4-axis conversion.

2.1.1 Cartesian 3D printer

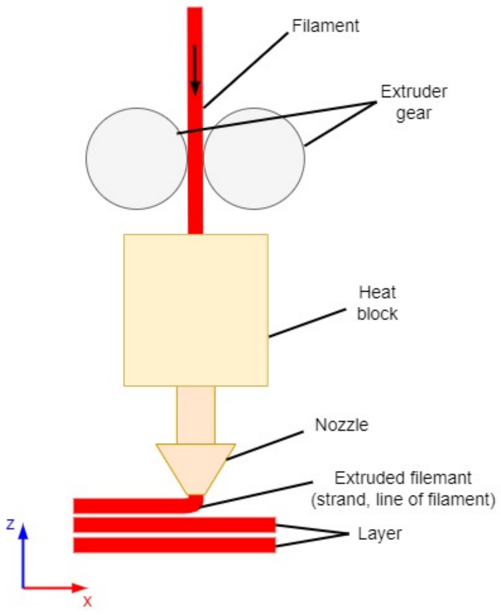
For this work the Prusa MK3S+ 3D printer (Prusa Research a.s. Partyzánská 188/7A, 17000 Prag 7, Tschechische Republik) was used. The printer has a Cartesian axis design in which the axes are orthogonal to each other [9]. The three axes are located as follows. The x-axis moves the print head from left to right. The z-axis is perpendicular to the build plate and moves the print head up and down. The y-axis is detached from the print head and moves the print bed back and forth. Another Cartesian 3D printer with the corresponding axes can be seen in Figure 1a. The standard nozzle that comes with the printer is a 0.4 mm brass nozzle, which will also be used in this work.

2.1.2 Planar 3D printing and slicing

FFF 3D printing is one technique in the field of AM, in which the goal is to create a physical object out of a digital 3D model file [6]. With conventional planar 3D



(a) All three axes of a standard Cartesian 3D printer (Elegoo Neptune 4).



(b) Schema of filament extrusion.

Figure 1: Basics of FFF 3D printing.

printing, an object is built in the direction of z. The print bed forms the xy-plane, which is also the plane on which each layer is printed. During printing, the layers are printed in ascending order from bottom to top. The most commonly used file format for CAD models is standard tessellation language (STL), which represents the 3D CAD model with triangular facets [6]. Therefore, in the STL file only the outer surface of the model is described. However, when designing a solid 3D model, not only the shell should be printed, but a solid physical object should be the result, which, in the example of 3D printing, is realized by a slicer [3]. The slicer is software that converts the STL file into G-code. The G-code is a data format defined in ISO 6983 [10]. The .gcode file is defined consisting of blocks of words, each word being an instruction for the machine, followed by arguments, if needed [11]. For example, "G1 X10 Y20 Z30 E10" is the command to move the tool head to the coordinate (10,20,30) and extrude 10 mm of filament, in the example of a 3D printer. The slicer converts the STL file into G-code in a way it cuts the model into horizontal slices that are printed in ascending order [6]. Each slice is printed with lines of filament that are extruded by the nozzle. Different types of lines allow the right settings to achieve the desired goal, such as optical appearance or mechanical stability [12]. For conventional FFF 3D printing with planar layers, there are already many slicer programs. Some of them are free or even open-source, like Slic3r [13], Cura [14] or PrusaSlicer [15]. Common settings in these programs are layer heights of 0.2 mm and a line width of 0.4 mm. Therefore, the filaments strands are in an ellipsoid shape, as shown in Figure 2

This method has some major constraints: Overhangs can only be printed to angles of 45°-60° with acceptable visual quality [16]. Otherwise, support structures are needed that are printed underneath those surfaces. Furthermore, the visual appear-

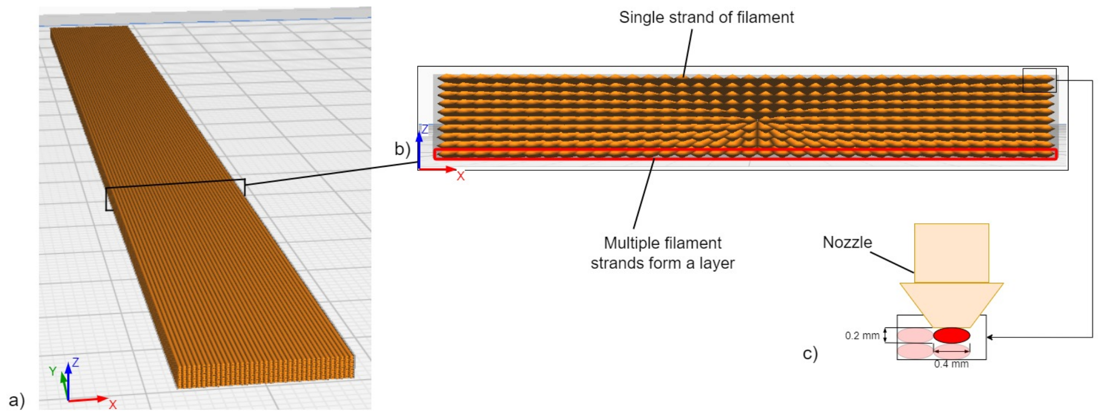

Single strand of filament
b)
Multiple filament strands form a layer
Nozzle
0.2 mm
0.4 mm
c)
a)
X
Y
Z

Figure 2: Basic slicing with a sliced 3D model with the lines of filament displayed (a), the cross section where the shape of the strands and layers are visible (b), and a schematic drawing of the shape of an extruded line of filament with the measures (c).

ance of a printed model depends on the surfaces it has, because planar layers can become visible when flat surfaces are printed almost parallel to the xy-plane [3]. With respect to the mechanical strength of a print, the orientation on the build plate is crucial. The bonds between the filament lines and especially between the layers are the regions that fail first under load [17] [12].

2.2 Non-planar slicing and printing

Besides the advances in planar 3D printing, the field of 4 or 5-axis 3D printing is also being investigated. With respect to the 4-axis modification that is used in this work, several papers are examined with regard to the slicing strategy and comparisons to planar 3D printing. The relevant papers in the field of 4-axis 3D printing are presented below.

2.2.1 Review of non-planar printing and slicing

In Nayyeri et al. [3] planar and non-planar slicing and printing strategies are reviewed and compared. The researchers looked into 47 slicing algorithms from various papers with planar slicing methods starting from 1995 and non-planar slicing starting in 2011. The paper presents several benefits that non-planar slicing has over planar slicing: #1 Higher surface quality through reduced staircase effect. The review examines several algorithms for different types of 3D printers and finds that non-planar slicing can reduce the staircase effect. #2 The paper also looked at the effects of the non-planar layer on mechanical properties and found that in some cases the non-planar slicing leads to a 57% increase in strength. This is due to the ability of non-planar 3D printing to print shapes more freely as one continuous fiber. Printed strands are stronger than inter-layer bonds, which can be avoided with non-planar 3D printing. #3 Another benefit of non-planar slicing according to the paper is the elimination, or reduced need, of support structures, due to the fact that non-planar 3D printers have more freedom of movement. This can lead to shorter production times, less postprocessing and according to Wüthrich et al. [16] also to increased surface quality. #4 Non-planar layers can lead to faster print times, although it is not always the case. The review paper concludes that non-planar slicing strategies are still in an early stage. The biggest gap between planar and non-planar slicing is a missing universal approach to non-planar slicing. Furthermore, the applicability to arbitrary models or structures is not always given.

2.2.2 Neural Slicer for Multi-Axis 3D Printing

An advanced non-planar neural network-based slicer for multi-axes 3D printers is presented in Liu et al. [18]. This work improves the S^3-Slicer by Zhan et al. [19], which is a comprehensive non-planar slicer that focuses on optimizing fabrication objectives such as support-free, stress-reinforced, and quality-enhanced prints. In order to optimize for those objectives, Neural Slicer uses a neural network based computational pipeline. The authors conducted physical tests on printed models to see whether the optimized layer orientations lead to stronger parts. The results showed that a model sliced with the Neural Slicer can withstand double the forces of the same model sliced with the S^3-Slicer. In the paper on the S^3-Slicer, this has already been shown to produce prints with higher breaking force. Both papers implement an impressive approach to non-planar slicer and provide valuable insights on how non-planar 3D printing can solve various manufacturing objectives that are impossible with planar printing. However, the solution provided by the authors is rather complex and therefore out of the scope of this work.

2.2.3 RotBot

Another modification that converts a 3-axis 3D printer to a 4-axis 3D printer is the rotating print head (RotBot) of Wüthrich et al. [16] with the RotBotSlicer presented by Wüthrich et al. [20]. Unlike the modification by FullControl, the additional axis used in this paper rotates around the z-axis with the nozzle installed at a fixed 45° angle. The RotBotSlicer uses the following strategy to create the G-code for the RotBot, which was inspired by Couped et al. [21].

1. If the resolution of the faces in the STL is too low, so that the transformation would not lead to smooth surfaces, the STL file is refined by subdividing the faces.
2. The triangular faces of the STL file of the 3D model are transformed into a (inverse) cone shape, depending on the slicing direction.
3. The transformed model is sliced with a regular planar slicer, e.g. Cura.
4. The inverse transformation function is applied to the G-code, resulting in the original shape of the 3D model.

The findings of this paper are that overhangs of 90° angle can be printed without support structures and with good visual quality of the downside surfaces. The fact that the nozzle has a fixed tilt angle makes the print result quite predictable. Therefore, simple premises for G-code modification can be defined to increase the quality of the outcome. This is not the case for the variable tilt angle of the modification that I use.

2.2.4 CurviSlicer

In Etienne et al. [22], the CurviSlicer is introduced. A non-planar slicer for 3-axis 3D printers, which reduces planar slicing artifacts like the staircase effect, and consequently produce smoother surface finishes. The goal is to print curved surface layers that should lead to stronger and visually smoother parts without the staircase effect. In order to achieve that, the authors combine planar layers and curved layers in a way that they start by printing planar layers and only print the top layers curved. Then, their approach is to progressively curve and thicken the layers to avoid a sudden change in the layer type. The curved layers are obtained by following a procedure similar to the strategy of Wüthrich et al.: The input is a 3D mesh Ω as a STL file. The mesh is then deformed using a transformation or mapping function $M(\Omega)$. The transformed mesh is then sliced with a conventional planar slicer to generate the toolpath T. After slicing, the original shape of the input model is restored by applying the inverse mapping on the toolpath $M^{-1}(T)$. Collision avoidance is also crucial for non-planar layers. Etienne et al. define an inverse cone-shaped area with the tip of the cone being the nozzle. The premise to avoid collisions is that no previously printed strand is not allowed to enter this area.

2.2.5 Other non-planar slicers

Other non-planar slicer projects like the Slicer4RTN (slicer for rotating tilted nozzle) by Rene K. Mueller [23] or the Radial non-planar slicer for the Core R-Theta Printer by Joshua Bird [24] also utilize the above mentioned strategy. I will especially use some of the code base from Joshua Bird because he implements a user-friendly code base in Python.

To the best of my knowledge and research, existing slicer programs like Cura or PrusaSlicer do not implement any features for non-planar slicing.

3 Converting the Prusa MK3S+ to a 4-axis printer

The modification of a standard 3-axis Cartesian 3D printer to a 4-axis 3D printer is the basis of this work. FullControl provided an open source manual about the modification they made on the Prusa MK3S+ [25, 4]. This modification adds the b-axis to the printer, which rotates the print head around the y-axis. Several parts are added and changed with this modification. In the following sections, the modification process is explained and the differences between a regular Prusa MK3S+ and the modified version are explained.

3.1 FullControl modification

The printer modification adds several new parts to the printer, as well as replacing existing parts. Due to limited availability in Germany, not all parts are exactly the ones used in the FullControl manual. The differences are explained later. The following parts are needed for the modification: Duet 3 Mini 5+ [26] and Duet 3 Mini 2+ Dual Stepper Expansion Board [27] by Duet3D (Duet3D Ltd Unit 74, Workspace House 28-29 Maxwell Rd Peterborough PE2 7JE, United Kingdom) have the same specifications as those used in the manual. The Duet 3 Mini 5+ is a control board that replaces the existing control board on the Prusa printer because it does not support an additional axis. Due to unavailability in the European Union, the stepper motor used in this work is a slightly different motor than the one used by FullControl, but with almost the same dimensions and specifications. Instead of the Nema 11, which is used by FullControl, a Joy-it NEMA 17 - 07GM (SIMAC Electronics GmbH Pascalstr. 8 47506 Neukirchen-Vluyn, Germany) is used, purchased at Conrad [28]. According to Duet documentation [29] the higher power consumption of 1.68 A is supported by the Duet 3 Mini 5+ control board. In addition, some mounting brackets have to be printed, which are used to mount the fourth axis to the x-axis. FullControl does not provide information about the recommended printing parameters for printing the brackets. I chose to print the brackets in ABS with four walls and 50% infill, which should be strong enough to hold the stepper motor and the print head. Finally, some bolts, nuts, and tools are needed to fit everything.

The clearance of the nozzle is an important factor for the degree of freedom added by the modification, as discussed later in Section 5.1. To increase clearance, a longer nozzle or long tip nozzle can be used. For this work, the standard M6 brazz nozzle that comes with the Prusa MK3S + is used. It is sufficient to showcase the theory on printed parts and also reduces the cost and time for the modification.

The original Prusa MK3S+ runs on a proprietary Prusa-built Marlin firmware built on proprietary [30]. However, since the modification of the printer, which is presented in the next section, uses the Reprap firmware, only this is explained here. It supports 32-bit processors and is designed for 3D printers [31].

3.2 Converting the 3D printer to a 4-axes printer

As mentioned in the previous section, in this work a different stepper motor is used. For this reason, the printed parts need to be adjusted in their dimensions to fit the dimensions of the stepper motor. To mitigate the chances of the parts getting loose and leaving artifacts on the printed object due to vibration, the dimensions are chosen in a way that they have almost no tolerance and some force had to be applied to get the parts on the stepper motor. The parts are printed in ABS and with 50% infill and four perimeters.

The setup of the Duet control board is well documented in the FullControl Google Colab notebook [25] and Duet3D documentation on how to connect the board to the computer [32] and install and update the firmware [33]. The latest version of the Reprap firmware that I update the board to is 3.5.4.

During the process of modifying a Prusa MK3S+ 3D printer for integration with a Duet 3 Mini 5+ control board, several mechanical and electrical adjustments were required to ensure proper function. The modification began with adapting the printer's wiring. This involved crimping the wires to attach them to the appropriate connectors compatible with the Duet 3 Mini 5+ board. Each wire had to be manually prepared and fitted with the correct terminal so that it could be inserted directly into the new board, ensuring stable and reliable electrical connections (see Figure 3a). Not all original functions can be used with the Duet control board. For example, the display is no longer operable.

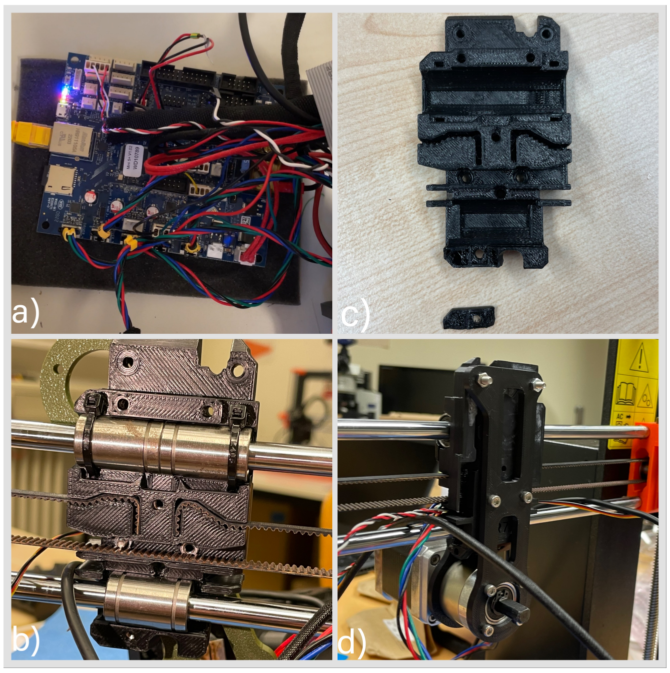


Figure 3: (a) Wired Duet 3 Mini 5+ control board. (b) X-carriage part on the x-axis. (c) Broken X-carriage printer part. (d) b-axis installed on the x-axis with printed modification parts.

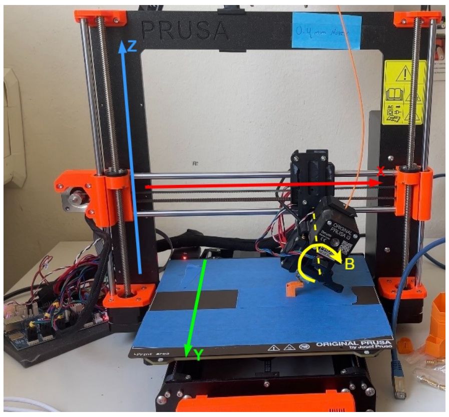
A significant mechanical issue occurred when trying to mount the new stepper motor. The rear motor bracket was discovered to be too short due to the larger body diameter of the replacement motor (36 mm), which caused it to collide with the X-carriage part of the printer. To accommodate this, the rear bracket had to be lengthened by 7 mm. Consequently, the front bracket also required the same extension in order to maintain the proper horizontal alignment of the motor. However, while attempting to install the extended front bracket, the X-carriage component broke, likely due to the stress of the new assembly. This necessitated reprinting of the X-carriage. Thanks to the open-source approach of Prusa, all printer parts are available online. See Figure 3b,3c,3d for the adjusted printed parts and the broken X-carriage.

Further modifications were carried out on the printer, including adjustments to the extruder bracket. The bracket was found to have not been modified correctly, as the motor shaft could not fit through the designated opening. This required further redesign of the bracket to ensure proper alignment and clearance for the motor components. The fully assembled 4-axis printer can be seen in Figure 4a.

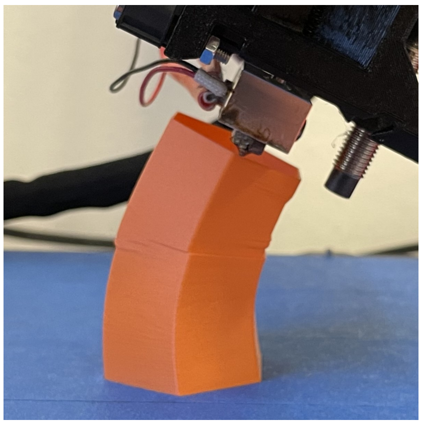
After completing the necessary hardware adjustments and setting up the firmware, the printer was successfully powered up and the web interface was reachable via the local IP address 192.168.2.4. The four axes (X, Y, Z, and B) responded to movement commands. However, the b-axis exhibited inaccurate movement: when instructed to rotate 45 degrees, it moved approximately 80 degrees instead. This indicated a calibration issue in the stepper motor settings, specifically the steps per millimeter parameter. This is due to the different gear ratio.

To correct this, the printer's firmware was accessed and modified via the RepRap

Web Interface. The config.g file contains critical parameters, including network settings, axis-to-driver mapping, heater and end-stop definitions, and stepper motor configurations. The steps per degree for the b-axis were adjusted to 170 steps/mm, instead of the 238.68 steps/mm used by FullControl, to compensate for the over-rotation. During the calibration process, several errors surfaced. One recurring issue was a stall warning: "Driver 1 stalled at Z height 0.00", which was traced to incorrect z-axis speed settings in both the homex.g and homey.g scripts. These were corrected by reducing the Z speed to 50%, resolving the stalling errors. Another issue occurred when attempting to extrude, with the error message "attempting to extrude with no tool selected." This was resolved by adding the command T0 to the config.g file, ensuring that the printer recognizes and selects the only available extruder [34]. A test print was initiated using a model from FullControl G-code, with a nozzle tip axis offset of 46 mm. The first 1.5% of the print was completed successfully, confirming that the system was partially operational. However, it became evident that the b-axis stepper motor was wobbling during operation, leading to inconsistent and unreliable print results (see Figure 4b). This mechanical instability presented a challenge, especially as replacing the motor was not feasible at the time. To mitigate this issue, a provisional solution was implemented using rubber bands. By attaching the bands to pull the print head from opposite sides, a counteracting force was introduced to stabilize the head. It was important to use rubber bands of the correct length; shorter bands were necessary to maintain consistent tension throughout the movement of the print head, as longer bands would stretch too much and lose effectiveness. This mechanical workaround allowed for some stabilization of the b-axis and continued experimentation. However, small artifacts can still be seen when the print head shifts from one side to the other. To completely eliminate this problem, a different stepper motor would need to be installed.



(a) The fully assembled 4-axis 3D printer with the four axes X, Y, Z, and B.



(b) Visible artifacts due to wobbling of the print head.

Figure 4: Modified printer with four axes and tilted nozzle.

3.3 Comparison: 3D Prusa MK3S+ and 4D Prusa MK3S+

The most obvious difference is the fourth axis that was added. The print head can now rotate along the y-axis which adds a degree of freedom to the printer. To what extent freedom can be utilized is discussed in Section 5.1. The modified printer is still able to print in the conventional planar style. Regarding technical differences, between the original Prusa MK3S+ and the modified version, there are a few drawbacks with the modification. The part cooling fan is removed to gain more clearance for the nozzle, to print at greater angles. The display is not connected due to missing Duet support. The automated bed leveling sensor is also removed to gain more clearance. Features such as automated bed leveling or automated determination of the z-offset are no longer available. The printer has to be accessed through the Web interface by connecting to the computer via Ethernet. The printer mod still misses a reliable homing routine, since the homing of the b-axis, which marks the $0°$ angle, has to be determined by eye. In the following sections of this work, the pros and cons of the b-axis are examined further. That is, the additional freedom for printed structures, limitations when printing more complex structures, the complexity and challenges for implementing a slicing pipeline, and how the different printing styles influence some mechanical properties of the prints. The key findings, in which the modified 4-axis version differs from the standard MK3S+ are

1. The 4-axes design instead of 3-axes,
2. Probably additional freedom for printing,
3. Missing part cooling fan,
4. The display is not connected anymore, and
5. No automated bed leveling.

4 B-axis slicing pipeline

In order to be able to use the modified 3D printer without writing or programming the G-code for every model that should be printed, a slicer program is needed. Full-Control offers a test file, which is a G-code for a bent tube. Currently, there is no slicer for this 3D printer modification with a b-axis. The following sections explain the concept and implementation of the B-axis slicing pipeline, as well as challenges for generating the G-code for this type of 3D printer.

4.1 Requirements

Before developing a concept for a slicing strategy, the requirements for this project must be determined. Several requirements can be derived from the objectives of this work, mentioned in Section 1.1.

The objective #1 is to elaborate on the geometric capabilities and limitations. To be able to do so, a comprehensive slicer is needed that is capable of creating the G-Code for many different, better yet, arbitrary 3D models. Furthermore, different settings for slicing are needed, to be able to handle a large variety of shapes.

The objective #2 is to give insight into what is needed for a non-planar slicer. This does not impose explicit requirements, since this can be fulfilled by investigating the program in the end. A non-functional requirement that makes the investigation in the end easier and clearer could be formulated so as to have a modular code that is easy to review.

The objective #3 is to investigate the mechanical properties of the 3D printed parts. Here, almost the same requirements can be derived as for the first objective. The slicer needs to be capable of handling multiple different forms and provide some settings that allow printing with different print head orientations. In addition, consistent print quality must be ensured between prints because multiple specimens are printed for one test case.

Following requirements can be concluded:

R1 The b-axis is utilized.

R2 Different types of non-horizontal layers are supported.

R3 It is possible to make various settings.

R4 Versatile models can be handled.

R5 The slicer has a modular architecture.

R6 The slicer produces consistent printing qualities.

4.2 Concept for implementing the B-axis slicing pipeline

As mentioned in section 2.1.2 several slicer programs already exist for planar slicing. What they do not support are non-planar 3D printers or a slicing strategy for non-planar printing. In Section 2.2, some implementations were presented to utilize different non-planar 3D printer designs. In Section 3 I explained the 4-axis modification on which this work is based. This work differs from the other 4-axes projects by the type of the fourth axis. The RotBot is a 4-axis printer with a rotating tilted nozzle that is mounted at a fixed angle of 45° and rotates around the z-axis. The R-Theta 3D printer is completely different, as it uses polar coordinates and does not have a Cartesian design. The CurviSlicer is designed to be used on standard Cartesian 3-axis 3D printers, not utilizing the additional freedom of a fourth axis. Since none of the nonplanar slicers supports the 4-axes design used in this work, an additional slicer has to be implemented. In the following, I will examine different strategies and concepts for implementing the slicer and determine the most suitable options for this use case.

The first consideration to make is where the slicer should be implemented. One way could be to implement a completely new toolpath generator (slicer). This would incorporate not only the challenges of using the b-axis but also additional challenges for path finding, differentiation between line types such as perimeters, infill, top and bottom shell, and more. Modern slicers solve those problems already and come with handy settings for the user to tune and optimize their prints. Having this in mind, developing a whole new slicer is out of the scope of this work and would just add unnecessary effort. With open-source projects like Cura, Slic3r, or PrusaSlicer it is possible to implement additional features right in the slicer. This could increase the simplicity of the usage for the user, but it would also limit the feature to only be used in a specific slicer. Additionally, the programming language and project structure would need to be adopted when implemented in an existing slicer. A third approach is to implement a wrapper that is built around an existing slicer, which would also limit the usage to one specific slicer.

Projects like the RotBotSlicer [20], the CurviSlicer [22], or the Radial non-planar slicer for the The Core R-Theta Printer [24] implemented their slicer with a pipeline approach. The STL file is preprocessed first and then the modified STL file is sliced with a standard planar slicer. The created G-code is then modified again in a post-processing step. This pipeline approach has several advantages.

1. Multiple slicers can be tested to validate their applicability for this use case, and future 4-axis printer projects can use a different slicer.
2. Python can be used, which is widely used and popular [35].
3. The complexity of the implementation is kept appropriately for this work, as the feature does not have to be integrated into existing graphical user interface (GUI).
4. An open-source notepad can be created to make the slicer more accessible.

For those reasons, the pipeline approach is used in this work, for which I inherit the concepts and partly even the code from the non-planar slicers, mentioned before. This incorporates three steps. In step #1, the preprocessing of the STL file, which contains two essential sub-steps: First, refinement of the STL file, and second, transformation of the STL file.

Step #2 is the planar slicing of the transformed STL file, where several settings must be set, to get a reliable G-code for the 4-axis printer.

Step #3 is the post-processing of the G-code, which contains two substeps that are completed after another. First, the inverse transformation is applied on the G-code, generated by the planar slicer. Then, the *b*-axis is added to the G-code and the values for the other three axes are recalculated, depending on the value of B.

All three main steps, as well as their substeps, are visualized in Figure 5 are explained in detail in the following sections.

The *B*-axis slicing pipeline is implemented as a Jupyter Notebook in Python. The library `stl` is used to read and modify STL files. The library `pygcode` is used to read and modify G-code files. `Numpy` is used to handle and manipulate large data sets.

4.2.1 Refinement of the STL faces

The concept of the refinement of the triangular faces in the STL file is inherited from the RotBotSlicer by Wüthrich et al. [20]. When creating a 3D model and exporting it as a STL file, flat surfaces are described by as few triangular faces as possible to reduce file size. This has no impact on the print if the STL file is printed as is. However, with the strategy of transforming the STL file in a pre-processing step, the resolution of the model plays a crucial role. When flat surfaces are transformed with

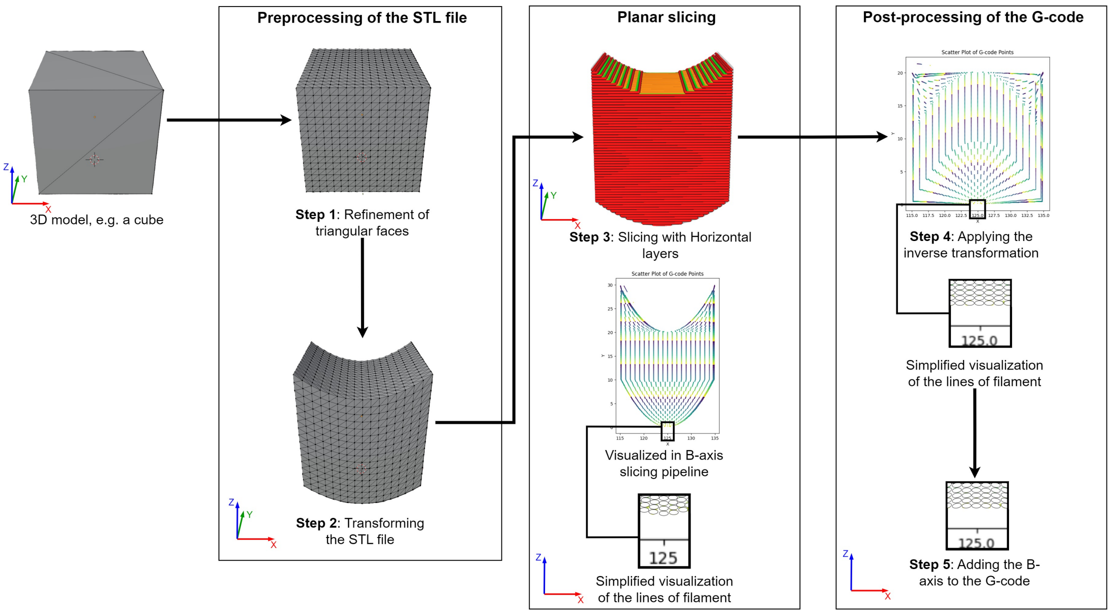
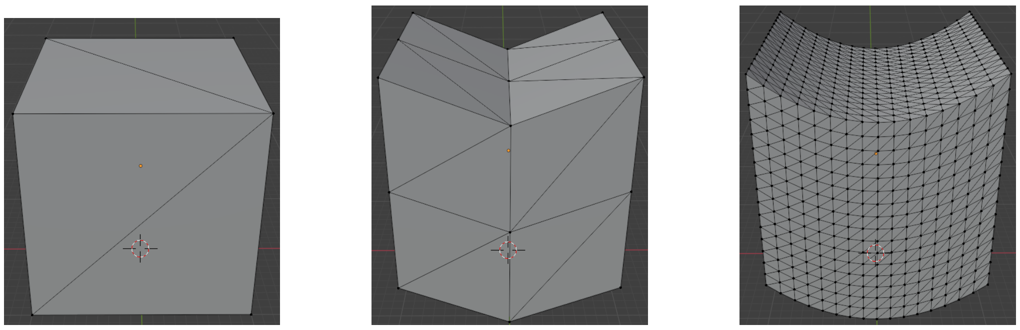

Preprocessing of the STL file
Planar slicing
Post-processing of the G-code
3D model, e.g. a cube
Step 1: Refinement of triangular faces
Step 2: Transforming the STL file
Step 3: Slicing with Horizontal layers
Scatter Plot of G-code Points
Visualized in B-axis slicing pipeline
125
Simplified visualization of the lines of filament
Step 4: Applying the inverse transformation
125.0
Simplified visualization of the lines of filament
Step 5: Adding the B-axis to the G-code
Z
Y
X

Figure 5: Concept and processing steps of the *B*-axis slicing pipeline.

low face resolution, an angular surface is the result, as can be seen in Figure 6. For refinement, I use the code from the RotBotSlicer from Wüthrich et al. [20], which divides each triangle into four smaller triangles per iteration, by bisecting its sides and connecting the new points. As mentioned by Wüthrich et al., this method divides every triangle, even the ones that are already small, which increases the computation time unnecessarily. However, since I will focus mainly on simpler geometrical models, this drawback is acceptable to me.



(a) 0 refinement iterations.

(b) 1 refinement iteration.

(c) 4 refinement iterations.

Figure 6: Transformed STL file of a cube with different number of iterations of dividing the faces.

4.2.2 Transform the STL file

The transformation of the STL file is a key step for non-planar slicing. This step maps the 3D model into non-planar space, depending on the type of layer. A visualization of the transformation can be seen in Step 2 in Figure 5. The section 4.2 briefly explained the strategy for slicing with transformation and inverse transformation, used in this work. The concept of transforming the STL file is inherited from the CurviSlicer by Etienne et al. [22], the RotBotSlicer by Wüthrich et al. [20], the Slicer4RTN by Rene K. Mueller [23], and the Radial non-planar slicer by Bird [24]. In this section, the strategy is explained in more detail. There are essentially two types of transformation that can be utilized to add freedom to the prints, which are inward bent (concave) layers and outward bent (convex) layers [16]. The advantages of each type are examined in Section 5.1. Other transformations can mix inward and outward bent layers. In the following work, I will focus on just inward or outward bent layers and not a combination of both, because they are the basic shapes. Any other transformation type will be ignored in this work, because they may just combine the capabilities of inward- and outward-bent transformation.

For the transformation, different functions can be used to achieve different types of layers. For example, layers can be transformed so that they are printed as a curve. Another type of layer can be straight with an angle. Both variants, as well as the outward and inward bent transformations, are shown in Figure 7. The function for the transformation of the curved layer is shown in Equation 1, which is a quadratic function. The function for angled straight layers with an incorporated tipping point is shown in Equation 2. Both functions modify the value of the z-coordinate and leave the x anyy value as is. That is, because the x and y-coordinates of the layers are the same as in the planar layers. Only the z-coordinate determines the type of layer. This can also be seen in Figure 7. The value of z is transformed depending on the x-coordinate. The scalar s is used to define the degree of deformation. The higher the value of s, the steeper the angles of the nozzle will be. The scalar c is used to define the type of transformation either being inward ($c = -1$) or outward ($c = 1$). In Equation 1 the z value is dependent on the quadratic value of x. Furthermore, an offset a for x can be applied that shifts the tipping point along the x-axis (see Figure 8). The term $1/x_{max}$ is added to limit the quadratic transformation function to add a maximum value of $x_{max} + 3a$, where x_{max} is the maximum distance from the tipping to the left-most or right-most point of the object along the x-axis. For the absolute transformation function, the absolute value of the distance to the center point along the x-axis is used with an offset.

$$T : \mathbb{R}^3 \to \mathbb{R}^3 : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto T(x,y,z) = \begin{pmatrix} x \\ y \\ z + s * c * \frac{1}{width_x/2+|a|} * (x+a)^2 \end{pmatrix} \tag{1}$$

$$T : \mathbb{R}^3 \to \mathbb{R}^3 : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto T(x,y,z) = \begin{pmatrix} x \\ y \\ z + s * c * |x+a| \end{pmatrix} \tag{2}$$

Before applying the function on the 3D model, the model is centered along the x axis. With this, the x-coordinate represents the distance to the center of the model. The function is then applied to every point P of every triangle of the STL file Ω, resulting in transformed model $T(\Omega)$. In Figure 8, a schematic front view on a cube 3D model

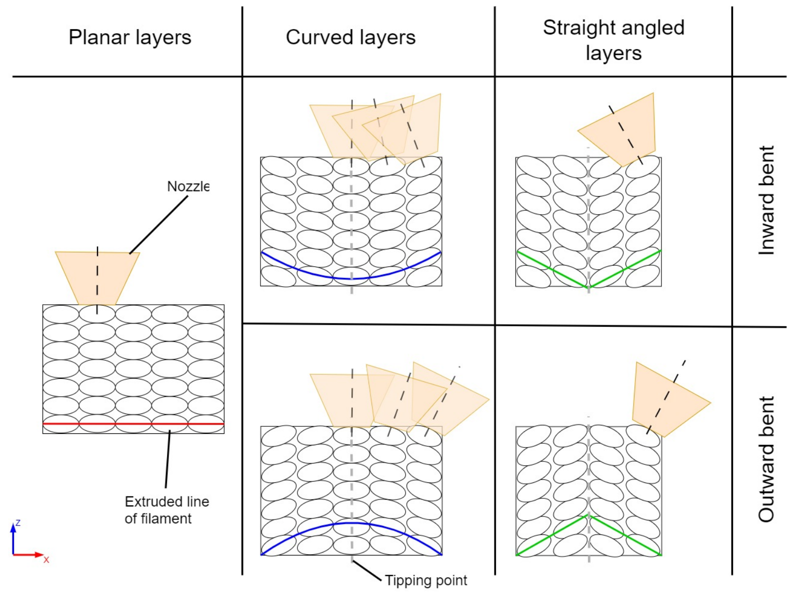


Figure 7: Different types of layers.

in orange with the corner points $\Omega = [A, B, C, D]$ is shown. The purple line is a quadratic transformation function that is applied on the orange model, resulting in the transformed model $T(\Omega) = [T(A), T(B), T(C), T(D)]$ in green. The left image shows the transformation without a given offset. The right image shows the transformation with an offset of +4 along the x-axis. Both images have the model already centered along the x-axis. A 3D view of a transformed model can be seen in Figures 6c and 5.

The following is the Python code that transforms the STL file. The `max_x_distance` is the half-width of the object. The variable `points` contains all the triples for the position of each corner of each triangle of the STL file.

```
f = lambda x: s * c * 1/(max_x_distance+x_offset) * (x+
    x_offset)**2
transform = (lambda x, y, z: numpy.array([x, y, z + f(x)]))
points_transformed = list(map(transform, points[:, 0],
    points[:, 1], points[:, 2]))
```

4.2.3 Planar slicing

In this step, the transformed STL file is sliced with a conventional planar slicer. I use Cura because it produces less boilerplate G-code, as e.g. PrusaSlicer. In order to ensure proper functioning of the 4-axis printer, several settings need to be made. First, I recommend setting the standard Prusa MK3S+ as printer profile. Then the maximum z-height has to be adjusted to a higher value to be able to fit the transformed models in the build area. I also recommend removing the start and end G-code in the machine settings and lowering the x width to 230 mm and the y depth to 190 mm, since the homing routine by FullControl for those axes cuts 20 mm to be sure not to hit the end stops while printing. For the slicer settings, the following values worked.

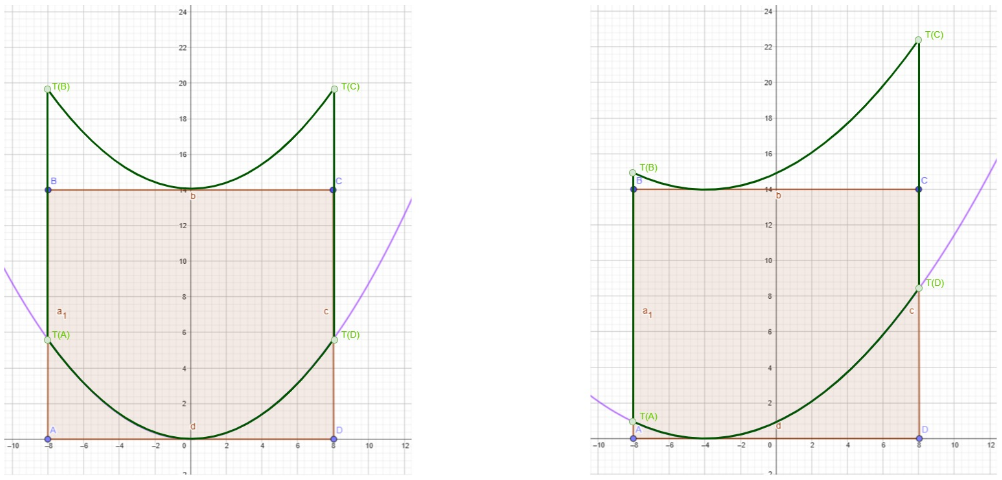


Figure 8: Schematic front view on a cube model with the original model (orange), a quadratic transformation function (purple) without an offset (left) and with an offset of +4 (right), and the transformed model (green).

- Layer height = 0.2 mm
- Line width = 0.4 mm
- Wall speed = 30 mm/s
- Infill speed = 40 mm/s
- Travel speed = 120 mm/s
- Relative extrusion = On

For structural settings like number of walls or infill density and infill pattern, the printer is capable of handling different settings. Those settings should be set according to the requirements of the printed object and the printability constraints.

The result of the planar slicing is a G-code file, which is further post-processed by the *B*-axis slicing pipeline in the next step.

4.2.4 Applying the inverse transformation on the G-code

The first step of the post-processing is reading the G-code to be able to further process it. While reading the G-code file, the commands are checked for long print movements. If a line longer than 0.5 mm is extruded along the x-axis, this command is split into segments of 0.5 mm length. This is a strategy by Wüthrich at al. [20] and Bird [24], both of whom split long nozzle movements in any direction. For the b-axis printer, this is only necessary along the x-axis, because it can only print non-planar along the x-axis. For the implementation of this step, I use the code from Bird [24], which can be found on GitHub [36], and modified it to fit this use case. The segmentation is necessary because there is no G-code command that tells the nozzle to move from one point to another whilst describing a curve. So multiple points along

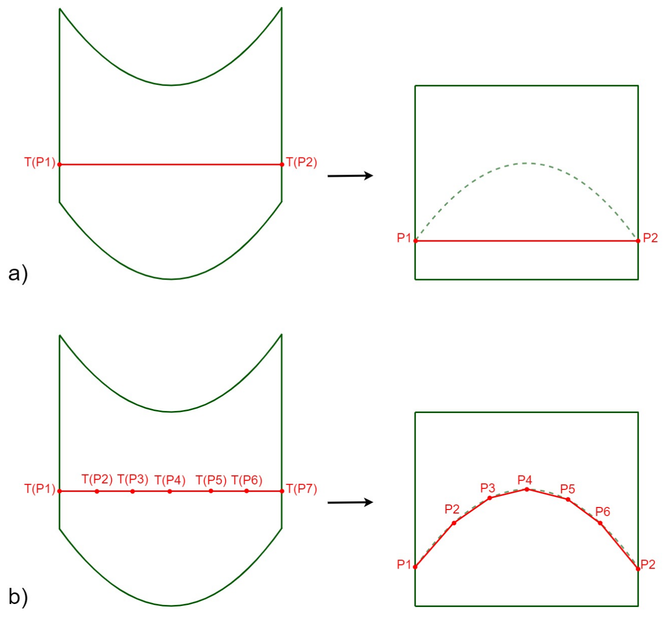


Figure 9: Schematic visualization of the problem of applying the inverse transformation on long nozzle movements (a) and the segmentation of long nozzle movements into smaller sections (b). This image is inspired by the Figure 9 of Wüthrich et al. [20].

the curve, which describe the non-planar layer have to be added and thereby control the printer, and especially the print head, in every position. See Figure 9 for an illustration of the problem.

The next step is to reverse the transformation. This is done by the inverse transformation T^{-1}, which basically just subtracts the value that was previously added by the transformation function. The inverse quadratic transformation function is shown in equation 3 and the inverse transformation for the absolute function is shown in equation 4. A visualization of this step can be seen in step 4 in Figure 5.

$$T^{-1} : \mathbb{R}^3 \to \mathbb{R}^3 : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto T^{-1}(x, y, z) = \begin{pmatrix} x \\ y \\ z - s * c * \frac{1}{width_x/2 + |a|} * 2 * (x + a) \end{pmatrix} \tag{3}$$

$$T^{-1} : \mathbb{R}^3 \to \mathbb{R}^3 : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto T^{-1}(x, y, z) = \begin{pmatrix} x \\ y \\ z - s * c \end{pmatrix} \tag{4}$$

In the B-axis slicing pipeline, the inverse transformation is applied by subtracting the value that was added to z at the transformation. The `position` array contains the coordinates for every G-code command. The function `f` is the same as in the previous code block where the STL file was transformed.

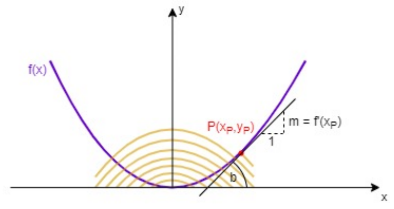
```
distances_to_x_center = numpy.array(positions[:,0]-x_center)
    .reshape(-1, 1)
translate_upwards = numpy.hstack([numpy.zeros((len(positions
    ), 2)), f(distances_to_x_center).reshape(-1, 1)])
new_positions = positions - translate_upwards
```

4.2.5 Adding the b-axis to the G-code

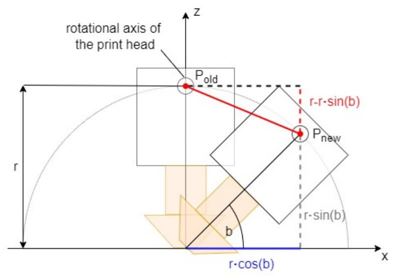
To this point in the process, the G-code describes non-planar or tilted planar layer (or a horizontal layer if the curvature strength c is set to 0). However, the current G-code considers only the x-, y-, and z-axes, which means that the nozzle only points down (0°). In this step, the angle of the nozzle is calculated and the value of the b-axis is added to the G-code. With the tilt of the nozzle, the z and x coordinates also have to be changed accordingly to compensate for rotation.

Before calculating b, the x coordinates need to be centered again. This is because the slicer produces G-code with coordinates for the printing location on the print bed. The b-axis is then calculated by taking the arc-tan of the slope of the transformation function at a given point on the x-axis and transforming it from a radial value to a degree. This results in the formula $b = 180 * \arctan(f'(x))/\pi$. A visualization can be seen in Figure 10a. The Python code to calculate b is shown below.

```
x_distance_to_center = position[0] - x_center
f_derivative = lambda x: s * c * 1/(max_x_distance+x_offset)
    * 2*(x+x_offset)
b = numpy.rad2deg(numpy.arctan(f_derivative(
    x_distance_to_center)))
```



(a) From transformation function to angle b of the nozzle.



(b) Influence of the tilt angle of the print head on the x and z coordinates.

Figure 10: Calculation of the b-axis value and adjustment of the x and z coordinates.

With the b-axis calculated, the coordinates for x and z also need to be adjusted. This is because the rotational axis of the print head is not at the tip of the nozzle. When the b-axis rotates, the actual position of the nozzle changes without recalculating x and z. The distance from the rotational axis to the tip of the nozzle is the offset of the tip of the nozzle. This also defines the radius r for the rotational circle, described by the print head rotating around a point $P(x, y, z)$ (see Figure 10b). The new x-coordinate of the print head is calculated using the formula 5. The new z-coordinate of the print head is calculated using the formula 6.

$$x_{new} = x_{old} + r * cos(b) \tag{5}$$

$$z_{new} = z_{old} - (r - r * \sin b) \tag{6}$$

In Section 4.3.1, the angle of the b-axis is adjusted to prevent collision. This is done right after the b-axis is calculated and before the x and z coordinates are adjusted. It

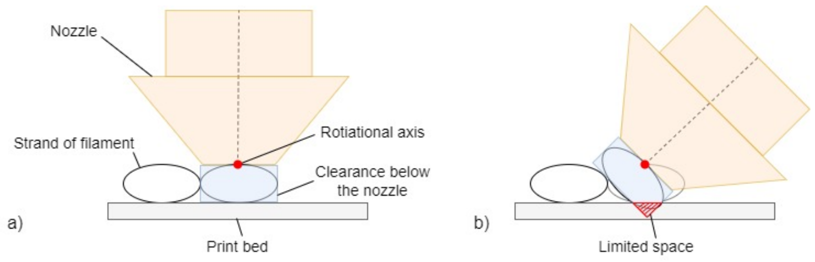


Figure 11: Change of the space below the nozzle due to rotation. Print area below the nozzle in horizontal prints (a) and with a tilted nozzle (b).

is crucial to understand how the nozzle behaves when tilted and what the rotational axis is, which the nozzle rotates around. If a line of filament is extruded on a first horizontal layer with a layer height of 0.2 mm, the height of the nozzle in the G-code is set to "Z0.2". Therefore, if the nozzle is rotated, the *B*-axis slicing pipeline rotates it around the tip of the nozzle. This changes the clearance space below the nozzle. This is visualized in Figure 11. Therefore, the extrusion rate has to be adjusted. This is explained in Section 4.3.2. After that is done, the G-code can be saved as a .gcode file and sent to the printer.

4.3 Challenges

Modern slicers such as Cura or PrusaSlicer have advanced features and predefined profiles for the most popular 3D printers. This allows the user to easily slice a 3D model and to be sure that the print will finish with at least acceptable quality. However, for the *B*-axis slicing pipeline, this is not the case since a whole new printer design is used. That reintroduces challenges that have already been solved for conventional planar slicing. The first challenge is collision avoidance, which is discussed in Section 4.3.1. With the tilt of the nozzle, the space below the nozzle changes. Therefore, the extrusion rate must be adjusted, which is explained in Section 4.3.2. Further challenges such as positioning the *z*-axis and the *b*-axis are covered in Section 4.3.3.

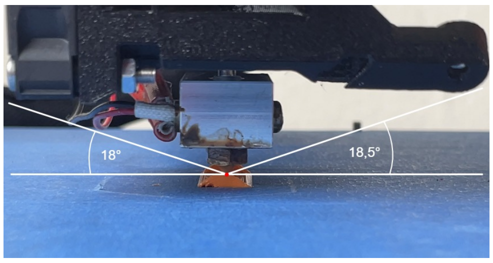
4.3.1 Collision avoidance

Collision avoidance plays a crucial role in ensuring a successful print. In planar 3D printing, collision avoidance is already done by taking care that the nozzle is higher than the latest printed layer and leveling the nozzle before printing. With the additional fourth axis, the complexity for collision avoidance increases, because the nozzle can now tilt and vary the *z* height within a layer. The assumption that the nozzle is always above the printed part, as is in planar 3D printing, is no longer sufficient, e.g. when printing sideways. Therefore, further mechanisms have to be defined.

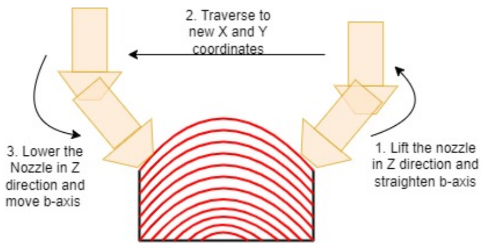
1) Avoid that the nozzle crashes into the print bed. For that, a maximum tilt angle is chosen, depending on the Z height. When the nozzle prints the first layer, the maximum angle can be 18°, which I determined with the clearance of the print head (see Figure 12a). The maximum angle of 90° can be used at a height of 50 mm. Depending on the height, the angle of the print head can increase linearly from 18° to

90° over a distance of 50 mm. The Python code for the clipping of the *b*-axis is shown below. The variable `position[2]` is the *z*-height of the current G-code command.

```
clip_height = 50
if(position[2] < clip_height):
    if(b < 0):
        b = numpy.max([b, -18-position[2]/clip_height*72])
    else:
        b = numpy.min([b, 18+position[2]/clip_height*72])
```



(a) Maximum tilt angle due to the clearance below the print head.


2. Traverse to new X and Y coordinates
3. Lower the Nozzle in Z direction and move b-axis
1. Lift the nozzle in Z direction and straighten b-axis

(b) Collision avoidance when traveling from one side to the other.

Figure 12: Collision avoidance.

2) The second principle is to prevent the nozzle from crashing into the printed object on travel moves (G0 command). When the transformed STL file is sliced with a planar slicer, the resulting G-code can contain travel moves from one point, diagonally traversing the print, to another point. This is no problem with planar 3D printing, but with non-planar 3D printing, the nozzle could crash into the print. To prevent this from happening, the nozzle first needs to be lifted, then moved to the target position on the *xy*-plane, and then moved to the target position (see Figure 12b). The code for this part is quite long, which is why I will not display it here.

3) For concave printing, only the curved layers are printable. With tilted planar layers, printed as an inward bent shape, the nozzle would collide with the other side, respectively. Curved layers can only be printed with a relatively low deformation. This depends on the height and width of the printed object. Since most of the following tests are printed with outward bent layers, I am not implementing a collision avoidance algorithm for inward bent transformations.

4.3.2 Extrusion rate

The extrusion rate is a big challenge when it comes to curved slicing. The problem with a tilted nozzle is that it is not trivial to determine the exact position and space of each strand that is printed. Overlapping strands cause blobs on the print that can be dragged by the nozzle and lead to irregularities. It can be seen in figure 13 that with the regular extrusion rate of the planar slicer, there are irregularities that result in poor print quality. Two parameters could be the cause for that.

1) The extrusion rate. The irregularities indicate that the extrusion rate is too high, leading to the filament bulging out.

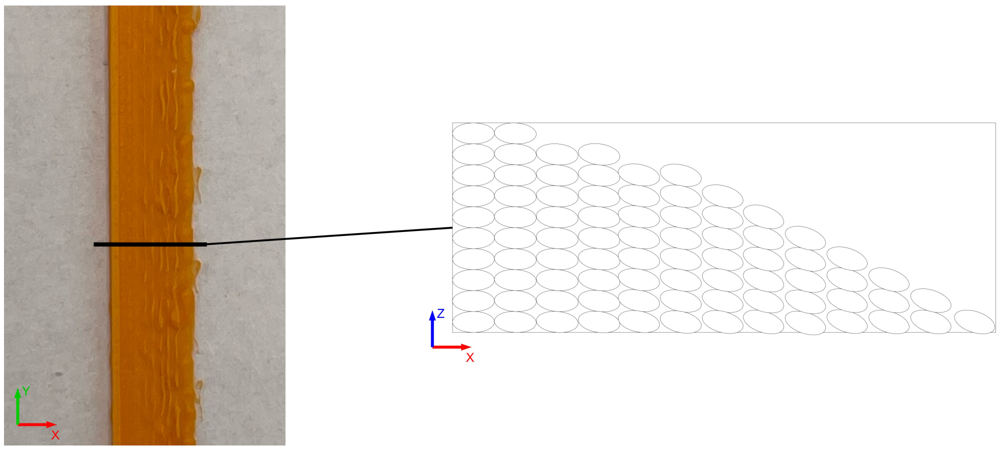
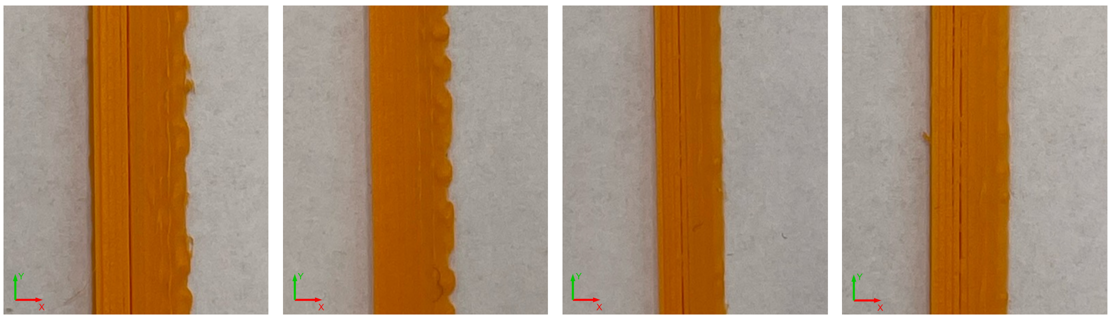


Figure 13: Over-extrusion causes blobs on the print and results in a bad print quality with 100 % extrusion rate and +0 mm z-offset.

2) Leveling of the nozzle. On the right-hand side of the printed sampled in Figure 13, it can be seen that the bulging is especially observed in the lower layers. The reason for this could be that because of the tilt angle of the nozzle, the space for extrusion is less than that with a straight nozzle.

To verify if these two parameters are the cause for the irregularities, I am printing four additional parts with the exact same settings and geometries, except for the extrusion rate and the z-offset. For the first part, I am changing the extrusion rate to 90 % of the original rate (see Figure 14a). In the second test, I am changing the z-offset to +0.2 mm (see Figure 14b). The third test print has both the extrusion rate at 90 % and the z-offset at +0.2 mm (see Figure 14c). The fourth test prints with 87 % extrusion rate and +0.2 mm z-offset (see Figure 14d).



(a) Extrusion rate 90 %, z-offset +0 mm. (b) Extrusion rate 100 %, z-offset +0.2 mm. (c) Extrusion rate 90 %, z-offset +0.2 mm. (d) Extrusion rate 87 %, z-offset +0.2 mm.

Figure 14: Test samples to find the cause for over-extrusion.

It can be seen that the reduction in the extrusion rate already has a positive impact on print quality. However, the z-offset has the greatest impact on the quality of the print and, combined with a reduced extrusion rate (see Figure 14d), can produce prints of acceptable quality. However, with reduced extrusion rates, the opposite effect occurs, which is called under-extrusion. In the images, it can be seen that the samples with lower extrusion rates show small air gaps between the strands. Lower extrusion rates lead to smaller line widths, which lead to gaps between the lines.

To prevent under-extrusion, an algorithm is needed that adjusts the extrusion rate. It can be derived from the above findings that the original extrusion rate is suitable for higher layers. So, the extrusion rate must be dynamically adjusted depending on the height of the nozzle. To verify this approach, I will test a simple algorithm that linearly increases the extrusion rate depending on the current nozzle height. Artifacts of over-extrusion occur in layers below 1 mm. In the above test series, an extrusion rate of 87% seems to be suitable for the first layer (nozzle height 0). To be sure to avoid over-extrusion, I will choose an extrusion rate of 85% for the first layer. From then on, the algorithm linearly increases the extrusion rate to 100% in layer 6 with a height of 1.2 mm. The *z*-offset is set to +0.2 mm. With these settings, the result is not satisfactory. Over-extrusion is still happening, as can be seen in Figure 15a. So I will adjust the algorithm to start the extrusion rate for the lower layers at 80%. The over-extrusion still occurs at 80% in the lower layer, as can be seen in Figure 15b. I set the *z*-offset higher, so that I don't have to lower the extrusion rate again and retain more of the original amount of filament. The Python code for the extrusion adaption is shown below. The variable `position[2]` is the value for the *z*-axis and `point["extrusion"]` is the value of the *e*-axis of the current G-code command.

```
extrusion_scale = 1
    if point["extrusion"] is not None and position[2] == 0:
        extrusion_scale = 0.85
    if point["extrusion"] is not None and position[2] < 1.2:
        extrusion_scale = 0.80 + 0.2 * (position[2]/1.2)
    point["extrusion"] = point["extrusion"]*extrusion_scale
        if point["extrusion"] is not None else None
```

With a *z*-offset of +0.3 mm, the over-extrusion shows minor artifacts and the print quality is sufficient (see Figure 15c). In addition, the problem of under-extrusion is eliminated, which can be clearly seen on the images. The gaps between the strands no longer exist. The question to answer is one of material loss from the adjusted extrusion rate. This is examined in Section 5.

4.3.3 Further challenges

Another challenge is manual calibration of the *z*-axis and the *b*-axis. With modification, the PINDA sensor (Prusa INDuction Auto-Leveling sensor), which is used for automated calibration of the print bed [37], is removed. This means that the offset of the *z*-axis has to be determined by hand. This is done by placing a paper piece between the nozzle and the print bed and lowering the nozzle until the piece of paper can no longer be moved back and forth. Then the command "G92 Z0" has to be sent to the printer, which tells the printer that the current position defines the height z 0. This method is not accurate and can lead to differences in print quality, as discussed in Section 5.2.1. Due to the limited time of this work, I did not find a way to automate this process again.

Even more prone to errors and inconsistent is the homing of the *b*-axis. The homing (command "G92 B0") sets the 0° position for the *b*-axis, in which the nozzle points