

5-Axis 3D Printing

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
in
Computer Science
Robotics Engineering
Mechanical Engineering
By

Ethan Coeytaux
Cameron Crook
Alexandre Pauwels
Hugh Whelan

Advisors:
Diran Apelian
George Heineman
Craig Putnam
Gabor Sarkozy

Worcester Polytechnic Institute
April 27, 2017

Abstract

The goal of this project was to create a hardware and software platform for performing 5-axis additive manufacturing. The 5-axis 3D printer translated in the x-, y-, and z-axes, and rotated in the a- and b-axes. Although the hardware printed using fused deposition modeling (FDM), the platform was adaptable to a variety of deposition systems such as cold spray. The software converted a 3D model into 5-axis instructions for the printer, with no further user input. This software made 5-axis 3D printing, previously a niche research area, accessible to anyone with a 3D object file.

Acknowledgements

A special thanks to all of the following:

Diran Apelian for funding our project in addition to your entrepreneurial guidance and mentorship.

Craig Putnam for your assistance with the robotics portion of the project.

Gabor Sarkozy and George Heineman for your guidance and encouragement with the computer science portion of the project.

Todd Keiller for providing us the opportunity to participate in the Accelerate WPI Program as well as your entrepreneurial guidance.

Sarah Mahan for connecting us with the WPI Tech Advisory Network (TAN).

Contents

1	Introduction	4
2	Background	7
2.1	Additive Manufacturing	7
2.2	3-axis vs. 5-axis 3D printing	8
2.3	Computer 3D model representations	10
2.4	3-axis slicing algorithms	10
2.5	5-axis slicing algorithms	11
2.6	5-axis 3D printer	14
3	Methodology	16
3.1	Overview	16
3.2	Software	16
3.2.1	Volume Decomposition	17
3.2.2	Build Direction	25
3.2.3	Build Sequencing	30
3.2.4	Collision Detection	31
3.2.5	5-axis G-code Generation	33
3.3	Printer	34
3.3.1	Mechanical Design	34
3.3.2	Nozzle Gantry	37
3.3.3	Printer Calibration	37
3.3.4	Forward Kinematics	39
3.3.5	Inverse Kinematics	44
3.3.6	Nozzle and Heated Bed	48
3.4	Embedded Design	56
3.4.1	Control Loop	56
3.4.2	Closed-Loop Feedback	56
3.4.3	Stepper Operation	57
3.4.4	Microcontroller Selection	58
3.4.5	Power	59
3.4.6	Circuitry & Wiring	60
3.4.7	Path Planning	62

4	Results & Discussion	68
4.1	Software Development	68
4.1.1	Volume Decomposition	68
4.1.2	Build Orientation	69
4.1.3	Sub-volume Sequencing	70
4.1.4	Collision Detection	70
4.1.5	5-axis G-code generation	70
4.2	Printer	70
4.3	Embedded Design	72
4.3.1	Circuitry & Hardware	72
4.3.2	Embedded software	73
5	Conclusion & Future Work	75
5.0.1	Continuous 5-axis Printing	75
5.0.2	Advanced Collision Detection	75
5.0.3	Advanced Sub-Volume Sequencing	75
5.0.4	Software Optimizations	76
5.0.5	Servos	76

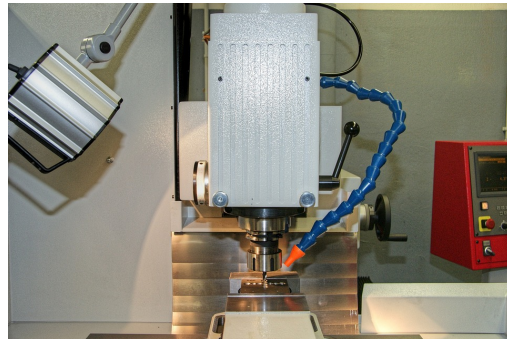
Chapter 1

Introduction

Additive manufacturing (AM) is relatively new to the manufacturing space, but has revolutionized the way in which objects are created. Previously, the popular method of manufacturing a part or product out of a solid block of material was to remove material until only the desired geometry remains. This method is known as subtractive machining (SM) and can be seen in the form of lathes or computer numerically controlled (CNC) mills, as shown in Figure 1.1. In contrast, AM works by depositing material to build up the shape of the object. This is typically done in a layer-by-layer fashion called layered manufacturing (LM). The object is "sliced" from its base up to its highest extreme, creating a sequence of planar layers which will be deposited independently, one after the other. A consumer-grade layered manufacturing machine, or 3D printer, is depicted in figure 1.2a. AM has many benefits over SM, such as efficiency in low volume production and prototyping and a simplified process[12]. There are many disadvantages, however, to LM, particularly with consumer 3D printers.



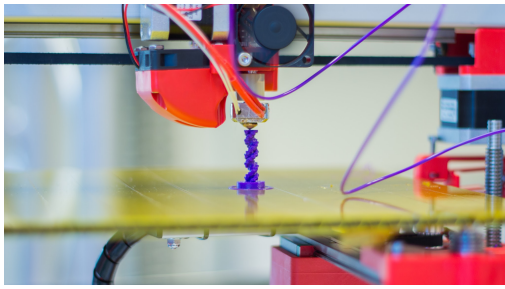
(a) Lathe



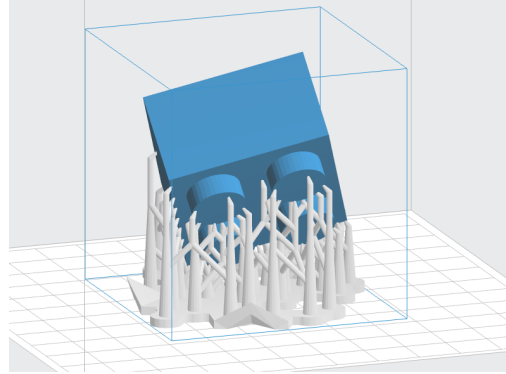
(b) CNC Mill

Figure 1.1: (a) shows a lathe cutting down a metal rod spinning very quickly to shape. (b) shows a CNC mill using a drill to remove material from a block of metal.

One disadvantage of LM is that in order for material to be deposited, there must be previously deposited material directly underneath it. As a result, objects with overhanging features cannot



(a) 3D printer



(b) The support structures for a Lego brick built at an odd angle

Figure 1.2: 1.2a shows a 3D printer depositing the material for an object layer by layer[11] 1.2b shows the support structures for a Lego brick built at an odd angle.[18]

be printed unless support structures are also printed beneath the overhang, as shown in Figure 1.2b. Printing support structures increases the amount of material used, print time, and post-print processing, which usually involves the manual removal of the support structures from the final part [3].

One way to minimize the need for support structures is the implementation of a 5-axis 3D printer, in which either the nozzle or the base platform has two additional rotational axes. These axes allow the printer to deposit material in different orientations. The potential impact of a 5-axis 3D printer on AM can be compared to the impact of a similar advancement in the SM space, with which the industry jumped from 3-axis to 5-axis machines. To further improve 5-axis SM, researchers such as Mahadevan Balasubramaniam published research on the automatic toolpath generation for the process. These substantial advancements also reduced manufacturing time and cost, and allowed for the manufacturing of previously impossible geometries.

This paper describes the design and implementation of an original 5-axis 3D printing system. This system includes hardware for a 3D printer with 5 degrees of freedom, as well as software to facilitate in the use of hardware and remove the need for skilled operators. We have developed a complete software package which automates the toolpath generation for a 5-axis 3D printer instructions for 3D models. By detecting simple overhanging features of a 3D model, the software makes use of the benefits of the 5-axis printer by generating instructions to print these overhangs at the appropriate orientations to reduce support structures. We also designed and built the 5-axis 3D printer hardware compatible with the aforementioned software. Our 3D printer hardware used a standard stationary material extruder coupled with a base plate operating in the three Cartesian axes, similar to a traditional 3D printer. Our printer hardware adds two additional rotational axes, B and A , to the build plate. These additional axes provide the capability to rotate the object being printed, and thereby print some overhang features without support structures. This closed system of both software and hardware allows for simple, automated 5-axis printing.

This project began as a entrepreneurial endeavor, targeting the market for a unique 5-axis printer. We participated in the Accelerate WPI program, an entrepreneurial program funded by WPI, to develop a value proposition and find customer segments. Interviews with potential cus-

tomers, industry professionals, and other 3D printing start-ups helped to identify consumer pain points with existing 3D printers. These pain points included low build quality, time consuming prints, steep learning curves, and unreliability. Based on these insights into the 3D printing market, we pivoted our value proposition from a 5-axis 3D printer to an easy-to-use, intuitive, and reliable 3D printer. We also presented our product and progress to the WPI Board of Trustees, the Dean of Engineering Advisory Board, and to WPI's Tech Advisory Network.

Chapter 2

Background

2.1 Additive Manufacturing

Additive manufacturing (AM) is a manufacturing technique that is growing rapidly and expanding the scope of manufacturing. Commonly referred to as 3D printing, AM was first proposed in the 1980's as an alternative to the traditional fabrication process, in which useless material was removed from a solid block until the desired shape was obtained. This process, also known as subtractive manufacturing (SM), uses tools such as lathes and milling machines to create the desired shape [17]. In contrast, AM involves the deposition of planar layers to build a predefined 3-dimensional shape, and as such is also referred to as layered manufacturing (LM) [2]. The LM process has exploded into many different areas of manufacturing, with over 30,000 3D-printing-related patents filed in the United States alone. Not only is AM an important manufacturing process for many industries, it is also extremely affordable, enabling home inventors to expand their manufacturing capabilities. The rapid growth, wide range of applicability, and recent affordability all suggest that we are at the beginning of a manufacturing revolution [17].

Additive manufacturing provides many benefits over traditional manufacturing techniques. Perhaps one of the most well known and widely taken advantage of benefits of additive manufacturing is the speed and ease at which a part can be manufactured. AM helps to bridge the gap between a computer model of a shape, and the physical part itself. In other words, AM provides an extremely fast method of manufacturing a part from an arbitrary computer model, allowing the manufacturer to avoid the many iterative stages that are typically involved in manufacturing. Using traditional manufacturing, even simple changes or complexities in a design can dramatically increase the number of stages that must be carried out to manufacture the end product. AM has been widely referred to as rapid prototyping (RP) as a result of these advantages. The process has primarily been used to speed up the prototyping of parts, which would often hinder other phases of product development [14].

In addition to the fast pace at which AM can bring a computer model into the physical world, AM can also substantially reduce the cost of manufacturing by reducing the number of processes and resources required for manufacturing. Using traditional manufacturing techniques may require many different processes, which might include molding, CNC machining, hand carving, etc. Each process requires different, often expensive resources, as well as expertise about the process. Depending on the computer model to be manufactured, all of these processes may be required at once.

In contrast, using AM to manufacture a computer model typically requires fewer steps, using a single process regardless of geometric complexity. In fact, the higher the geometric complexity, the greater the advantage AM has over CNC machining [14]. The cost benefits are another benefit that make AM great for rapid prototyping. However, with recent developments in the process AM has proven to be a useful manufacturing technique in many more areas than just rapid prototyping.

AM is no longer simply a fast process for prototype manufacturing. As technologies have developed and additive manufacturing has been explored more thoroughly, the process has begun to move from a prototype-focused process to one that can be used to manufacture parts fit for end use. This is a result of advances in accuracy and material properties of 3D printing. Parts manufactured using AM are now used for a variety of applications, including medical prosthetics, aerospace applications, and automotive applications [14].

2.2 3-axis vs. 5-axis 3D printing

Traditionally, 3D printers have three degrees of freedom, allowing the nozzle to move in the 3 axes defined by the 3D Cartesian coordinate system: x , y , and z [22] as illustrated in Figure 2.1. These printers build from the bottom up, creating the bottom layer of the object on a flat platform, then moving in the positive z direction to build the next layer. This means the z -axis value of the nozzle of the printer only moves in a positive direction once printing begins. A setback of this layer by layer technique is that all extruded material must have previously extruded material below it to prevent it from falling. A feature which does not have material directly below it is referred to as an overhang. 3D printers are typically capable of printing overhangs which extend from existing material below a given angle with no support structures. This maximum angle is usually 45 degrees from the build direction but varies depending on the material. The weight and adhesive properties of the printing material are two factors which impact the maximum overhang angle. For overhangs that extend from existing material at a angle greater than this maximum angle, support structures must be built below the overhang to prevent it from falling. These support structures require extra material and printing time, and must be manually removed from the object after printing is completed, which is a tedious process and can damage the surface quality of the object. Figure 2.2 shows two different overhangs: (a), which is printable without the use of support structures, and (b) which requires support structures to be printed.

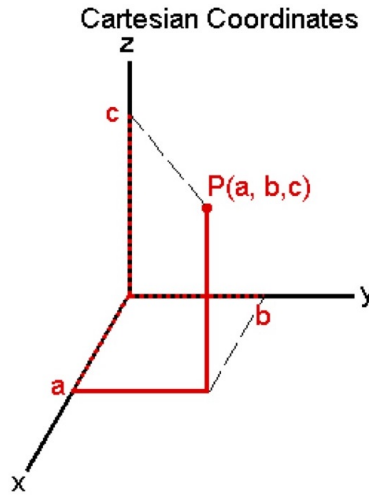


Figure 2.1: A 3 dimensional coordinate system, with axes X, Y, and Z. The X-coordinate of point P is a, the Y-coordinate is b, and the Z-coordinate is c [21].

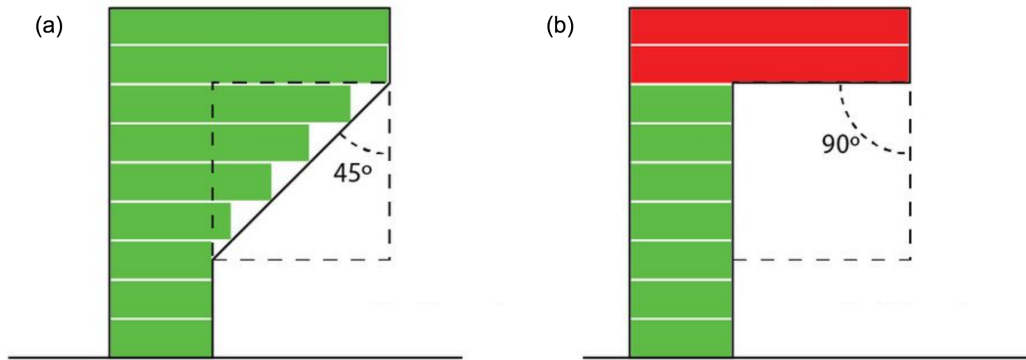


Figure 2.2: Two different overhangs: (a) is a printable without the need for support structures, and (b) requires support structures to support the overhanging feature, which is at an angle of 90 degrees

One way to minimize the need for support structures is the implementation of a 5-axis 3D printer, in which either the nozzle or the base platform has two additional rotational axes. These axes allow the printer to deposit material in more than one direction. Figure 2.3 represents the approach taken to print a part with an overhang using rotational axes on the base platform of the

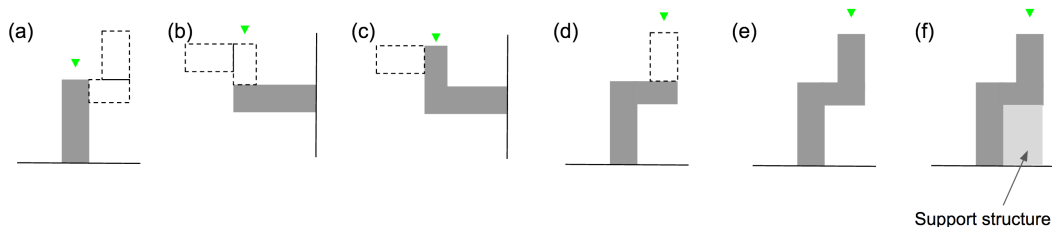


Figure 2.3: The basic overview of the steps involved in printing a simple part with an overhang. In order to print the part represented in (e), the build platform rotates along with any previously deposited material. This allows overhangs to be deposited onto previously deposited material, such as from (b) to (c). (f) shows the output of a traditional 3-axis 3D printer, which deposits support structures to support overhanging features.

printer. In Figures 2.3(b) and (c), the base platform rotates the part, allowing the overhang to be deposited. The part must then be rotated again to print the rest of the object. Figure 2.3(f) shows what final printed object would look like without 5-axes, using support material to support the overhang.

2.3 Computer 3D model representations

AM typically uses a 3D model of the geometry, which is then converted into instructions for the printer hardware. Meshes are a common representation of 3D objects, in which each face of a finite set of faces is discretely represented [leon]. This finite set of faces represents a 3D model or a collection of disjoint 3D models. Because of this format, there are no curved surfaces. Instead, "curves" are represented as many small faces, with the size depending on the resolution of the StereoLithography (STL) file. An STL file, created by 3D Systems Inc., is a common file type for representing meshes in many 3D printing softwares [24]. An STL file represents a 3D object as a collection of unidirectional triangular faces, meaning a face which has only one orientation. Each face represents face of the mesh, and is represented with a normal vector and three vertices, each represented with three numbers in Cartesian form.

2.4 3-axis slicing algorithms

In order to 3D print an object using 3-axis AM, a mesh of the object is converted into layer by layer instructions for the printer. Thanks to software both open-source and proprietary, this process has been automated into a single step [17]. To generate instructions for the printer hardware, slicing software divides the object into horizontal flat slices which lie perpendicular to the build direction, which is always the positive z -axis. Each printer has a defined layer height, which is the thickness of the material being extruded from the nozzle. Moving positively along the build direction by increments of the layer height, a slicing algorithm divides the shape into slices until the entire height of the object has been sliced [2]. The slices can then be used to generate a toolpath, or a pattern for the material deposition of an entire 3 dimensional object [3]. Open-source consumer

slicing software packages such as Cura [20] and Slic3r [19] also automate the generation of support structures in order to resolve complications caused by overhangs described in Chapter 2.5.

Existing 3-axis slicing software provides solutions to many different problems that 5-axis 3D printing has in common. Although 3-axis slicing and toolpath algorithms are very relevant to 5-axis slicing and toolpath algorithms, this paper does not explore these algorithms in detail. Instead, the 3-axis toolpath software is used without modification to directly solve sub-problems of 5-axis toolpath generation, as described in Chapter 3.2. These algorithms have been studied extensively.

Of more interest is the ability to generate support structures, which involves the detection of overhanging features as well. One way to detect overhanging faces is to simply take each face of the object and measure its normal relative to the build direction. If that angle is greater than some cutoff (typically 45 degrees), then it's considered an overhang face and is marked for support [13]. The second part, generation of the support structure shape, is much more nuanced. A variety of solutions have been developed to solve this problem, ranging from cellular support structures [9], meant to remove as much volume as possible from the structure, to tree-like structures which branch out from each other to create far-reaching supports with minimal material [13]. The detection of overhanging features on an object is discussed in Chapter 2.5 as it relates to 5-axis slicing algorithms.

2.5 5-axis slicing algorithms

Adding two or more rotational axes to traditional three axes 3D printers enables the ability of overhang deposition without support structures. The rotational axes, which are implemented on the printer as either nozzle or base platform movement axes [16], allow for better surface quality, reduced print time, and minimal post-processing, which usually involves removing the support structures from the final part [3]. In order to generate the printer toolpaths for 5-axis printing, however, a team of researchers is usually required to customize controls for the printer for specific geometries [16]. Unlike 3-axis 3D printing, 5-axis printing has no software package which makes slicing and toolpath generation a single automated step. In this paper, a description of the software implementation and hardware necessary to perform such 5-axis 3D printing in a single step is provided.

Despite the lack of existing 5-axis slicing and toolpath generation software, there has been a number of publications describing algorithms needed to create such software. The problem description is relatively constant throughout the previous research on the topic: "How much of a part should be made in a particular direction and why?" [3, p.129].

In order to answer this question, it is important to first understand the different factors which the software must take into account in order to generate a printer toolpath. These factors include the collision of the nozzle with the printed piece, the support of the printed material such that it does not collapse under its own weight, and the minimization of support structure use. By optimizing the toolpath according to these factors, a 5-axis printer using the generated toolpath from the software can accomplish the previously listed benefits of 5-axis over 3-axis 3D printing.

The steps that must be taken in an algorithm which slices and generates a toolpath for an object are common to various publications which attempt to outline such an algorithm. These are the core steps which were used to design our algorithm and software, listed below [3] [16] [6].

- Decomposing the volume into sub-volumes
- Determining the build orientation for sub-volumes

- Sequencing the sub-volumes

Volume Decomposition

The first objective in 5-axis slicing and path planning is the identification of overhang surface features which cannot be built without rotation or support structures. In "Slicing Algorithms for Multi-Axis 3-D Metal Printing of Overhangs", Lee and Jee observe that the detection of undercut/overhang features, when given an STL file is "a simple set theory between two adjacent sliced layers" [16],p. 5141. The upper slice, S_k^T , and the lower slice, S_k^B , are compared to determine the topology relationship. Only four different relationships can occur, which are shown in Figure 2.4. Using the relationship between two layers, we can begin to deconstruct the volume into separate sub-volumes. By separating each layer into buildable and unbuildable parts, we can sum all unbuildable parts of all layers to find the unbuildable sub-volume [16]. The buildable parts of all layers can also be summed to determine the buildable volume.

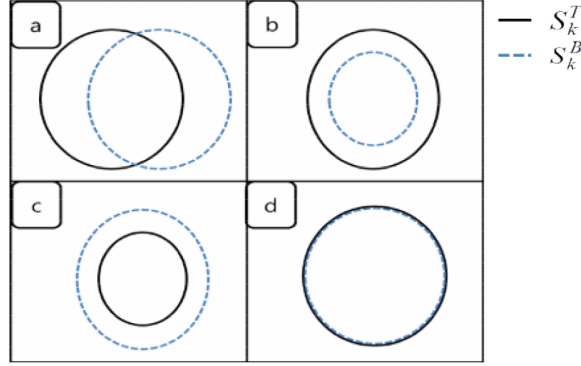


Figure 2.4: Only four topology relationships are possible between two adjacent layers: (a) a partial overlap, (b) a complete inclusion such that the bottom slice is completely contained in the top slice, (c) a complete inclusion such that the top slice is completely contained in the bottom slice, and (d) a complete overlap[16]

There are various other approaches that are taken to this problem which are outlined in other publications. In "A Slicing Procedure for 5-Axis LaserAided DMD Process," Sundaram and Choi propose the use of surface interrogation to detect overhangs. This approach, which compares the surface normals to the z-axis, examines all points on the surface of the object, marking any angle which exceeds the maximum overhang as an unbuildable surface. Once the lowest overhang point is determined, An intersection graph is created by intersecting a plane horizontal to the z axis at said lowest point with the object. By sweeping this intersection graph in the build direction, buildable and unbuildable sub-volumes can be identified [6].

Singh and Dutta improve on this technique in "Multi-Direction Slicing for Layered Manufacturing" in order to resolve a substantial error that Sundaram and Choi overlooked in their algorithm. Singh and Dutta use silhouette edges, as opposed to all surfaces with an unbuildable normal, to determine which surfaces are buildable [3]. Despite this improvement, both algorithms require a

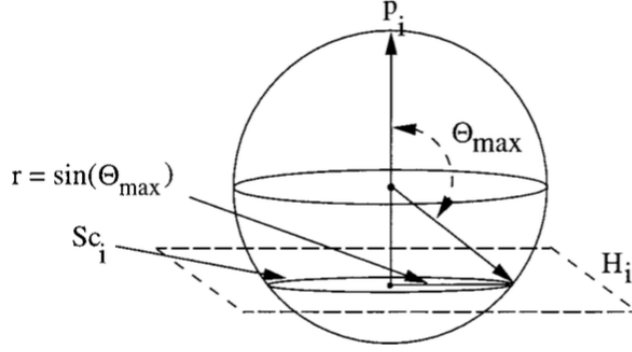


Figure 2.5: Representation of a build map where all values corresponding to a normal p_i are removed from the build map. The plane H_i represents the area of the build map to be removed.[3]

3-dimensional sweep of a two-dimensional shape, which is a complex algorithm. In addition, the volume decomposition may need to be recursively performed when a identified connected sub-volume cannot be build in any single build direction. For this reason, in addition to the existence of a previously implemented overhang detection algorithm loosely based on Sundaram and Choi's technique in the open source Cura Engine, we focused on Sundaram and Choi's research for volume decomposition.

Sub-volume Orientation

Once sub-volume decomposition is complete, a build direction must be assigned for each sub-volume which cannot be built along the original build direction. Singh and Dutta propose a concept of a "build map" to represent a set of valid build directions, with a valid build direction defined as a orientation at which the entire sub-volume can be built, layer by layer, without support structures or further decomposing the sub-volume into more sub-volumes [3]. It is possible for a build map to be empty for a given sub-volume, in which case the sub-volume may need to be decomposed further or support structures may be required [3]. A build map is represented as a filled portion of a sphere's surface, where any vector that has its origin at the sphere's origin and passes through the filled portion of the sphere's surface is a valid build direction, shown in Figure 2.5.

The build map starts out as a representation of a sphere's surface. Then, for every face on the sub-volume, a certain portion of the sphere is subtracted from the build map. The portion subtracted is the surface created by every point where the angle between the normal of the face and the point is greater than $\pi - \theta_{max}$. Once this has been done for each face on the sub-volume, any portion of the sphere's surface that remains is a valid build direction. While this provides a way of determining a valid orientation, it does not guarantee that a given valid orientation will not cause the nozzle to collide with other parts of the object while constructing the sub-volume. Collision detection must be done separately, and with proper sequencing each sub-volume can be built at its optimal orientation. Once again, a sequence of sub-volumes that causes no collisions may not exist, in which case sub-volumes may need to be built at different orientations or support structures may be needed [3].

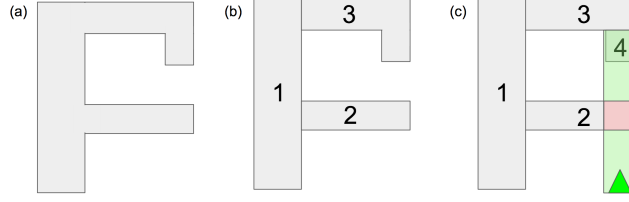


Figure 2.6: The side view of an example of an invalid sequencing computed using the process described in [6]. (a) shows the figure that is being constructed using a 5-axis 3D printer. (b) Shows the initial sequencing after the first volume decomposition. (c) shows the sequencing of sub-volumes after all sub-volumes are decomposed. In order to print sub-volume 4 of (c), the print nozzle must move in the area indicated by the green box. A collision occurs when the nozzle collides with sub-volume 2, indicated by a red square.

Sub-volume Sequencing

The order in which sub-volumes are printed is determined by two constraints. First, a base for which to print the sub-volume upon must exist. That is, if sub-volume a is being printed using sub-volume b as a base, sub-volume b must be printed before sub-volume a . Second, collisions between the nozzle and layers that have already been deposited must be avoided [3]. To solve this problem, Sundaram and Choi propose simply extruding sub-volumes in order of their heights [6]. This does not, however, take into account any sub-volumes which must be further subdivided. An example in which this technique does not provide a valid build sequence is shown in Figure 2.6. Figure 2.6a shows a two-dimensional representation of the three-dimensional object which is being printed. Figure 2.6c shows the object divided into its four sub-volumes as described in Section 3.2.1. Using Sundaram and Choi’s solution, sub-volume’s 3 and 4 will be printed first, resulting in a collision with sub-volume 2 when printing sub-volume 4.

Path planning

Slicing and path planning are the final steps in generating instructions for the 3D printer. Slicing refers to the division of an object into equal thickness parts along the build direction. Unlike 3-axis 3D printing, this build direction is not constant in 5-axis printing. Path planning refers to the material deposition patterns within each slice [3]. Volume slicing for 3-axis printers has been extensively studied, as well as implemented in consumer software packages such as Cura and Slic3r. The division of a volume into sub-volumes which can be printed completely along one build direction can be used to produce a full 5-axis slicing and toolpath for an object. Due to the fact that 3-axis slicing and path planning has been extensively studied and implemented, this paper assumes the ability to generate these toolpaths. Section 3.2 describes further how to generate a 5-axis toolpath with this information.

2.6 5-axis 3D printer

Only a few examples of 5-axis 3D printing have been created and made public, such as DMG MORI’s LASERTEC 65 3D, TWI’s Laser Metal Deposition Printer, and Øyvind Kallevik Grutle’s

Pentarod [10] [23] [15]. These systems however, lack a robust software that will generate G-code for any arbitrary STL file. The G-code used for test objects with Grutle's Pentarod were taken from CNC machine G-code and altered by hand to work the 5-axis printer [15]p. 56. Despite their limitations, these systems were the foundation for our 3D printer. As the LASERTEX 65 3D is a proprietary system, not much information was able to be retrieved about how the system is built and works. However, Grutle's Pentarod design is published in a Master's Thesis paper from the University of Oslo and is available academically.

There are three types of configurations a 5-axis 3D printer can use: table/table, head/table, and head/head [15]. In a table/table configuration, the build platform of the printer rotates in the two additional axes, A and B, while the nozzle stays fixed at a given orientation, still able to move in the three Cartesian coordinate system axes. In a head/table configuration, the table rotates in one additional axis, either A or B, and the nozzle can rotate in the other axis, still providing the same degrees of freedom. In a head/head configuration, the build platform remains stationary, while the nozzle rotates in both rotational axes in addition to the Cartesian coordinate system axes [15] p. 10-11. While all configurations provide the same degrees of freedom, they each have their own distinct strengths and weaknesses. These differences include issues with the weight of the object being printed, accuracy, and material deposition [15] p. 24.

Chapter 3

Methodology

3.1 Overview

This chapter describes the design and implementation of an original 5-axis 3D printing system. This system includes hardware for a 3D printer with 5 degrees of freedom, as well as software to facilitate in the use of hardware and remove the need for skilled operators. We have developed a complete software package which automates the toolpath generation for a 5-axis 3D printer instructions for a limited set of 3D models. By detecting simple overhanging features of a 3D model, the software made use of the benefits of a 5-axis printer by generating instructions to print these overhangs at the appropriate orientations to reduce support structures. We also designed and built the 5-axis 3D printer hardware compatible with the aforementioned software. Our 3D printer hardware used a standard stationary material extruder coupled with a base plate operating in the three Cartesian axes, similar to a traditional 3D printer. Our printer hardware adds two additional rotational axes, B and A , to the build plate. These additional axes provide the capability to rotate the object being printed, and thereby print some overhang features without support structures. This closed system of both software and hardware allows for simple, automated 5-axis printing.

3.2 Software

In order to create a user friendly 5-axis 3D printer, a software package is necessary to allow the user to simply input a 3D object file and generate instruction for the 5-axis printer. In this chapter, we discuss the algorithms developed and implemented to solve the four main sub-problems which compose the overall task of generating these 5-axis printer instructions for an arbitrary 3D model. First, volume decomposition breaks the 3D object into smaller 3D "sub-volumes," such that all overhanging volumes which would normally require support structures are disjoint. Then, sub-volume orientation determines which orientation each sub-volume can be built in. Sub-volume sequencing outputs an sequence of sub-volumes which allow them to be built with no collisions or complications. Finally, path planning uses the output of these algorithms to generate the tool path for the 5-axis 3D printer.

3.2.1 Volume Decomposition

In order to minimize the use of support structures by varying the orientation at which an overhanging feature is built, the volume of each overhanging feature must be separated from the original volume so that it can be processed independently, as described in the following sections. By separating all overhanging volumes from the original volume, the result is one buildable sub-volume which can be built entirely without support structures, and a collection of sub-volumes which must be built at different orientations to avoid the use of support structures. Several factors must be accounted for in performing this decomposition, which are:

1. Mesh integrity
2. Overhang bases
3. Speed of computation

The integrity of the mesh refers to maintaining a mesh, such that each triangular face is surrounded by three other faces, sharing exactly one edge with each surrounding face. This issue frequently arises when performing mesh operations to disconnect sub-volumes that are identified as overhangs from the buildable volume of object. In order to make any given cut on a mesh, which separates the mesh into two disjoint meshes, any existing face intersecting the cut must be cut by a line which is determined by projecting the plane of the cut onto the face. As shown in Figure 3.1a and Figure 3.1b, the ideal cut frequently results in the need to split a triangle into a triangle and a quadrilateral. As a quadrilateral face is not compatible with a mesh representation, additional calculations must be performed to maintain the integrity of the mesh and divide the quadrilateral into two triangular faces, as seen in Figure 3.1b.

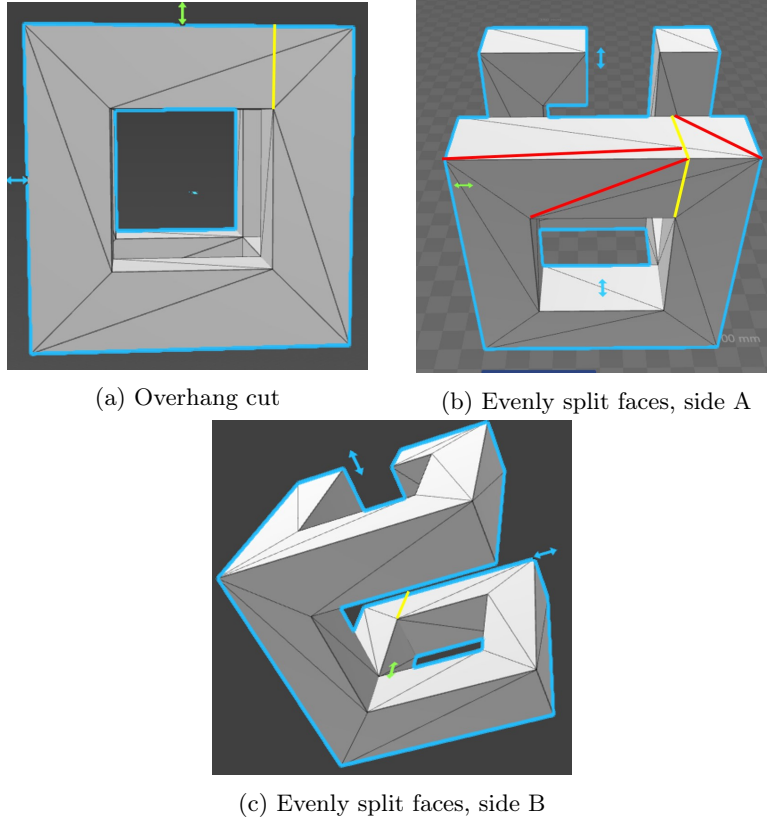


Figure 3.1: (a) shows how cutting the overhang clean through would result in open faces. (b) and (c) both show how the faces cut in two (yellow) would be further split to maintain mesh integrity (red).

Overhang bases are also an important factor in build decomposition. Any sub-volume which is determined to be an overhang must be printed onto a surface on previously deposited material. If this surface is not generally flat, it may present difficulties for layer adhesion and nozzle collisions. It is therefore important that the volume decomposition algorithm takes into account the simplicity of overhang bases when determining the optimal cut to separate an overhanging volume from the base volume.

Although solutions to the mesh integrity and overhang base problems exist, the difficulty is in performing them in a reasonable time-frame for the user. Since overhangs may themselves have overhangs, this algorithm is recursive and must run as many times as there are total overhang volumes. Each time the algorithm runs, it must iterate through all of the faces in the buildable and unbuildable meshes. This gives a $O(N^2)$ run time, where N is the number of faces on the mesh to be printed. In order to mitigate the effects of this run-time, special attention is paid to the number of individually buildable overhangs in the input mesh. The algorithm will run recursively through sub-meshes for as many sub-meshes as are identified, which translates directly to the number of overhangs.

The completed algorithm was split up into several steps:

1. Identifying transition faces
2. Finding the cut along a transition face
3. Splitting the transition face smoothly
4. Recursing through the overhang base faces
5. Assembling the final sub-mesh graph

The algorithm begins by going through all sliced layers of the object. For each layer, it iterates through all the faces which contributed segments to the layer and identifies which ones belong to both an unbuildable mesh (overhang) and the buildable mesh; these are known as transition faces. Each transition face is split into two or more sub-faces in order to create a smooth cut between the meshes, and then its neighbors are checked to find more transition faces. The algorithm recurses through neighbors of transition faces until it returns back to the original transition face, essentially cutting off the entire unbuildable mesh from the buildable mesh. The overhang mesh is added to the graph of meshes, with the buildable mesh as its parent. Once all unbuildable meshes have been discovered, each one has the algorithm recursively called on it to discover any further overhangs.

Identifying Transition Faces

To identify transition faces, the algorithm begins by slicing the mesh in its designated build direction, as a standard 3-axis slicer would. Each layer is a 2D polygon representing one slice of the object that will be printed. Two types of layers are used in this portion of the algorithm: the current layer and the comparison layer. Understanding what each of these layers does is crucial to the identification process.

The comparison layer essentially represents the boundary for any faces which should be considered buildable. When a face is flattened by simply eliminating the z-component of its three vertices, the resulting vertices can be checked to see if they fall inside or outside the comparison polygon. This can be seen in Figure 3.2 A face with all three of its vertices inside or on the boundary, of the comparison polygon is considered a buildable face. Any face with all three of its vertices outside the comparison polygon is part of an overhang. Finally, any face with some vertices inside and some vertices outside the polygon is a transition face, meaning it connects the buildable mesh with the unbuildable mesh.

This layer is used in conjunction with the current layer to identify all buildable and unbuildable components of a volume. The current layer is simply the slice of the object that is currently being processed. Each face that contributed to a segment of the current layer is flattened and checked for membership to the buildable, unbuildable, or transition faces using the comparison layer.

By iterating through all layers of the object and using the comparison layer and current layer, all transition faces can be identified. The algorithm begins by setting the first layer of the object as the initial comparison layer, and the second layer as the current layer. Each face in the current layer is iterated through and checked to see if it is a transition face. If it is, the face is passed into a separate function which identifies the cut points where the face will be split into sub-faces for buildable and unbuildable meshes.

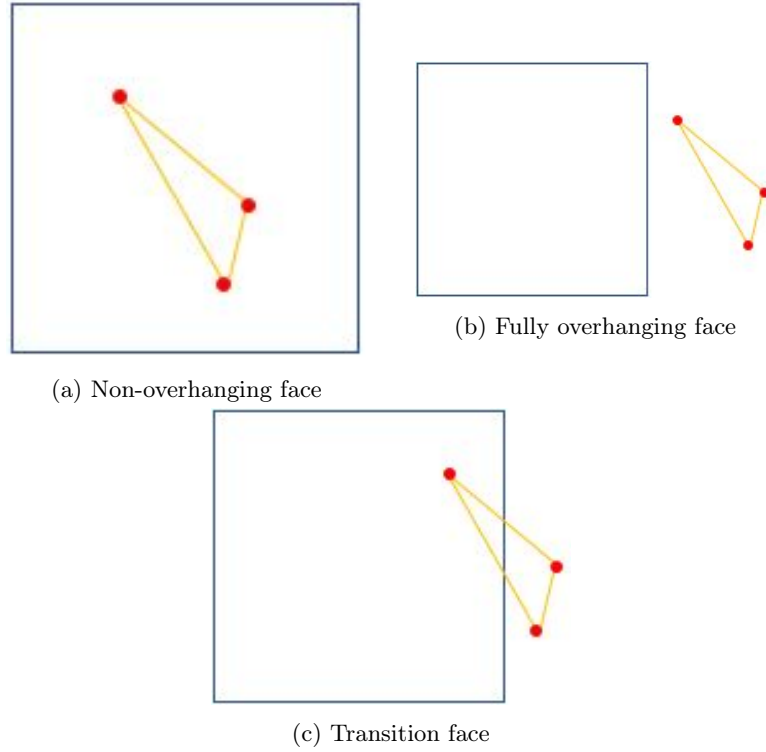


Figure 3.2: All three cases encountered when comparing a face with the comparison polygon.

Identifying Split Points of Each Transition Face

Identifying how to cut a face into sub-faces, each of which is either entirely buildable or unbuildable, is dependent on the concept of flattening 3D polygons. Although the current layer is already a 2D shape, the transition face which must be split exists in 3D. Thankfully, the cut is assumed to always be parallel with the build direction, and this simplification allows for the use of the comparison polygon itself to make the cut.

First, the transition face is flattened by eliminating the z-component of each of its vertices. The comparison polygon is then taken and subtracted from the flattened transition face. What remains is the portion of the face which belongs exclusively to the unbuildable mesh. This can be seen in Figure 3.3a. Each point of the resulting shape is checked to see if it lies on the edge of the comparison polygon. If the point does, then the point is considered a split point: a location on an edge of the face where the cut will occur. This can be seen in Figure 3.3b.

Two possible cases exist for these split points. Either the face has one split point, or it has two split points. The former occurs only when the face shares exactly one vertex with the comparison polygon, and nothing else. In that case, the subtraction removes none of the face's area and the only common point is the vertex; no further splitting need occur, and the algorithm proceeds with neighbor checking. If there are two split points and each split point corresponds to a unique vertex, the face remains intact and is separated from a neighboring faces. In all other cases where there

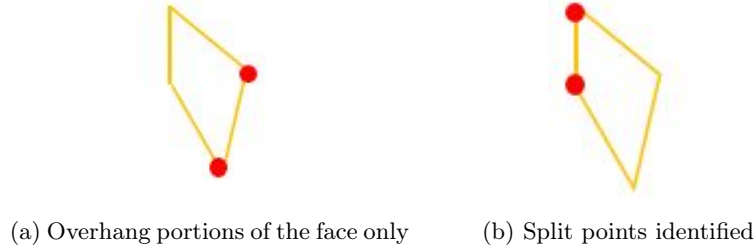


Figure 3.3: (a) shows what remains of the face after subtracting the portion of it inside the comparison polygon. (b) shows the points where the face was cut.

are two split points the face will have to be split into two or more faces. All cases can be seen in Figure 3.5.

Furthermore, there is another complication to this part of the algorithm. It is possible that one face may be part of multiple different overhangs. In this case, the face would be split in different ways along multiple sets of split points. An important observation, however, allows this problem to be resolved fairly simply. If a face belongs to multiple overhangs, the subtraction of the comparison polygon from the face will always result in as many distinct polygons as there are overhangs. This concept is seen in Figure 3.4. Each of these polygons can be iterated through individually and the split points for each found. After which the entire set of split points is returned.



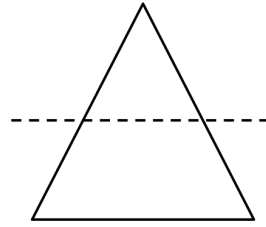
Figure 3.4: (a) shows a face which spans two separate overhangs. (b) shows how the subtraction by the comparison polygon requires two sets of split points to be found.

Splitting a Transition Face

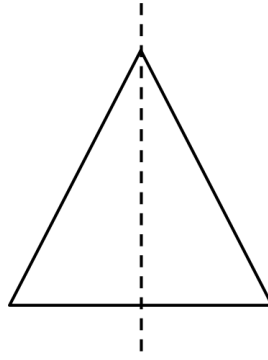
Once a face has been identified as being part of an overhang, the face must be split so that the part of the face that is an overhang is put into a separate mesh and the other part of the face remains connected to the original mesh. This sometimes involves creating new vertices and sometimes new faces inside the mesh, explained in more detail in the following paragraph. Inside the mesh data structure, each vertex and face must update it's adjacent vertices and faces to correctly represent the new mesh. A face on the edge of a split cannot be connected to the corresponding face on the opposite side of the split or the meshes will stay connected. After each face is split, the same procedure is applied to the neighboring faces that share the split line. Each time a face is split it creates the new vertices and faces if needed and updates it's adjacency lists.

We identified four possible cases that could happen when splitting a face with a straight line.

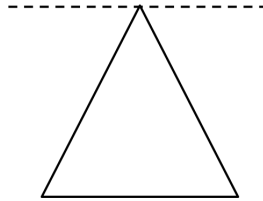
In the first case, the line goes through two separate edges of the face. In the second case, the line goes through one edge and the opposite vertex. In the third case, the line goes through exactly one vertex. Finally, in the last case, the line goes through two vertices, therefore running along the edge of the face. A diagram of the different cases can be seen below.



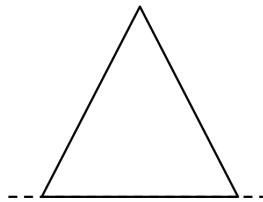
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 3.5: The four different cases encountered when splitting a mesh face.

For each case the vertices are labeled with x , y , and z , where x is equal to either 0, 1, or 2, and $y = (x + 1) \bmod 2$, and $z = (x + 2) \bmod 2$, meaning $\{x, y, z\}$ can be equivalent to either $\{0, 1, 2\}$,

$\{1, 2, 0\}$, or $\{2, 0, 1\}$. The edges are identified as the x_{th} edge being the edge created by vertices x and y , the y_{th} edge created by y and z , and the z_{th} edge being the edge created by vertices z and x . This way, regardless of the orientation of the face the correct indices of adjacent vertices and faces can be called variably, allowing us to treat them the same. For example, if two faces were a case 3, but one has the line going through vertex 0 and the other going through vertex 1, x could be set to 0 and 1 respectively, and the same code could handle both cases. In cases 1 and 2, new faces have to be created, which are labelled alphabetically in the order they are created, with the A_{th} face being the original face only with it's adjacent vertices and faces updated. The following figures show the labelling of the different cases as well as a chart showing which cases can follow a given case when iterating through the faces.

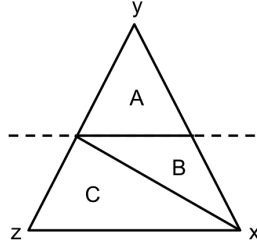


Figure 3.6: Case 1 labeled

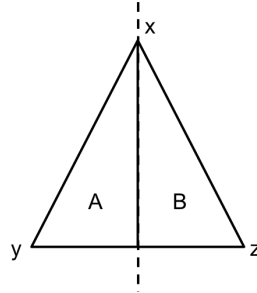


Figure 3.7: Case 2 labeled

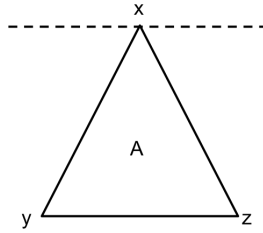


Figure 3.8: Case 3 labeled

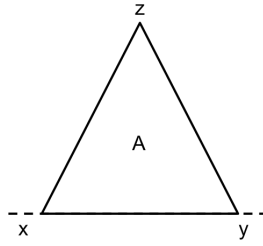


Figure 3.9: Case 4 labeled

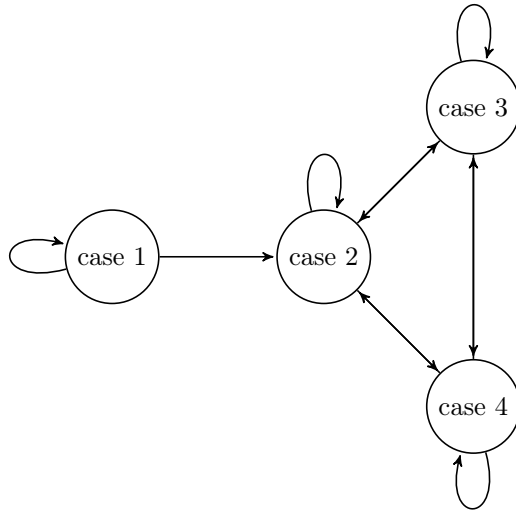


Figure 3.10: Graph of possible cases

Representing Sub-volumes in a Graph

Volume decomposition constructs a directed graph of the sub-volume's in order to store information about relationships between them. The need for this graph is described further in 3.2.3. A sub-volume is represented as a node in the graph, which also contains information about the build direction, as well as all direct descendants of the node. A directed edge from node A to node B represents that sub-volume A must be printed before sub-volume B. Upon completion of volume decomposition, a new node is created for the identified buildable sub-volumes. Each disjoint overhang sub-volume is also a new node, and stored as a descendant of the buildable node. Volume decomposition is recursively run on the descendant nodes, until all nodes in the graph represent a sub-volume which is completely buildable in one build direction.

3.2.2 Build Direction

To determine the optimal orientation at which to deposit a sub-volume, the algorithm described by Singh and Dutta and overviewed in Section 2.5 is used. This algorithm uses the surface of a sphere to represent viable orientations for which to build a sub-volume. We represent the surface of the sphere as a series of discrete points, specifically 3,240,000 ((360 degrees \cdot 0.1 b-axis precision) \cdot (90 degrees \cdot 0.1 a-axis precision)) points, in spherical coordinates in the form of (r, θ, ϕ) . In spherical coordinates, r represents the distance of the point from the origin, θ represents the angle from the positive x-axis of the projection of the vector onto the xy plane, and finally ϕ represents the angle of the vector from the positive z-axis, shown in Figure 3.11. This precision matches the limitations of the hardware, as each rotational axis on the printer hardware can only rotate to a tenth of a degree, as described in Section 3.3.1. In spherical coordinates we can represent every point on the semi-sphere with the integer points in the rectangle in the form of $(1, x, y)$. This means we substitute θ with the x coordinate of the rectangle and ϕ with the y coordinate of the rectangle. Because it is a semi-sphere we can use a constant, in this case 1, as a constant for r .

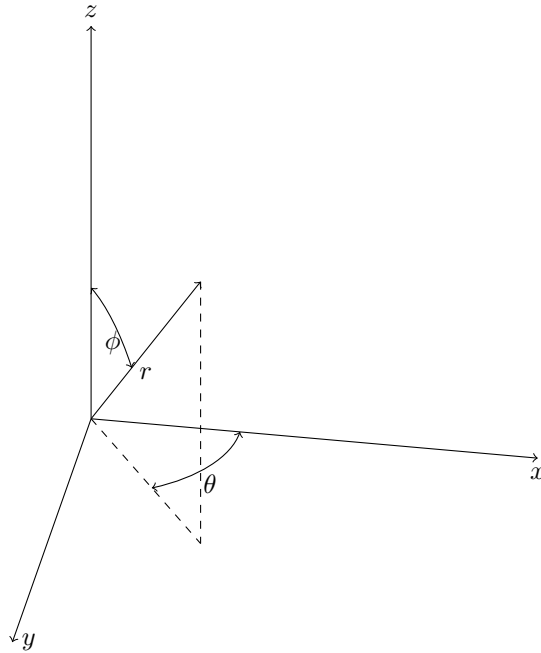


Figure 3.11: Spherical coordinate diagram

Using this representation of a build map, we can remove sections of the build map that are determined to be invalid build directions, using the formula described in Section 2.5. To remove these points from the 2D rectangle, for each face we remove every integer point inside the ellipse with the following equation, where θ and ϕ are the spherical coordinates of the normal of the face and θ_{max} is the max overhang angle.

$$\left(\frac{(x - \theta) \times \sin \phi}{\theta_{max}}\right)^2 + \left(\frac{y - \phi}{\theta_{max}}\right)^2 = 1 \quad (3.1)$$

Because the rectangle is representing a sphere, the ellipse must wrap around the rectangle horizontally. This means if the right edge of the ellipse is out of bounds of the rectangle, whatever section is out of bounds was removed from the left side of the rectangle. By removing every point within the ellipse created by equation 3.1 for each face, we are left with the valid points (if any) that represent the directions at which the sub-volume can be built. The following pseudo code outlines creating the build map given a mesh.

Algorithm 1 Construct Build Map

```

1: procedure CONSTRUCT-BUILD-MAP( $F$ )  $\triangleright F$  is a set of faces on a mesh
2:    $map \leftarrow \{x \in \mathbb{N}, y \in \mathbb{N} \mid x \leq \text{number of discrete b-axis points}, y \leq$ 
    $\text{number of discrete a-axis points}\}$ 
3:   for all  $f \in F$  do
4:      $center_\theta \leftarrow f.normal.\theta$ 
5:      $center_\phi \leftarrow f.normal.\phi + \pi$ 
6:      $ellipse \leftarrow \{x \in \mathbb{N}, y \in \mathbb{N} \mid (\frac{(x - center_\theta) \times \sin(center_\phi)}{\theta_{max}})^2 + (\frac{y - center_\phi}{\theta_{max}})^2 = 1\}$ 
7:      $map \leftarrow map \setminus ellipse$ 
8:   end for
9:   return( $map$ )
10: end procedure

```

Once the build map has had all invalid points removed, we need to find to optimal build direction at which the sub-volume can be built, assuming the build map has a positive area. Every valid orientation can be assigned a heuristic, and we want to find the orientation with the smallest heuristic, although any valid orientation is acceptable. The heuristic we use to determine optimal build quality is the average cusp height of the object if it were to be built at a given build direction. Cusp height is defined as the closest distance from the base of a layer to the expected surface of an object. A perfectly vertical object would theoretically have a cusp height of zero, while an object with a 45 degree angle would have a cusp height of $z_\tau \cdot \sqrt{2}$, where z_τ is the slice thickness. Figure 3.12 provides a diagram of cusp height. To determine the optimal build direction, we first start by finding any valid point on the build map to use as our starting point for a hill climbing search. To first find any valid point we start with the entire build map and cut it in half vertically. We choose whichever half of the build map still has an area greater than zero. If both halves have an area greater than zero it does not matter which half we choose. From there we cut the section in half horizontally and repeat the process, alternatively cutting the section vertically and horizontally, until we have focused onto a single point. The pseudo code outlining this process is described below.

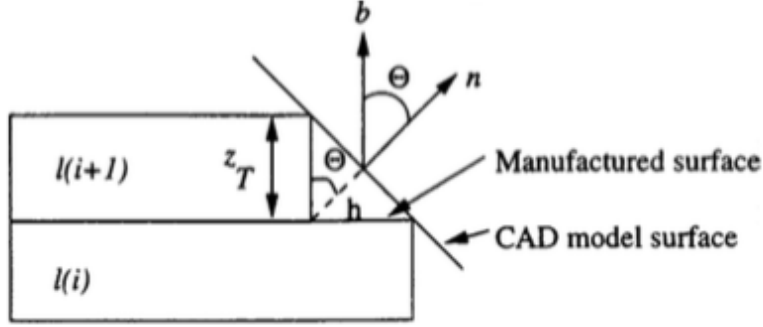


Figure 3.12: Cusp Height [3]

Algorithm 2 Find Valid Orientation

```

1: procedure FIND-VALID-ORIENTATION( $\text{map}, F$ )  $\triangleright$   $\text{map}$  is the build map,  $F$  is a set of faces on
   a mesh
2:   if  $\text{map.area} = 0$  then
3:     return( $\text{null}$ )
4:   end if
5:    $S \leftarrow \text{map}$ 
6:   while  $S.\text{size} \neq 1$  do
7:      $h1, h2 \leftarrow \text{SPLIT}(S)$   $\triangleright$   $\text{SPLIT}(S)$  returns two halves of search
8:     if  $h1.\text{area} > 0$  then
9:        $S \leftarrow h1$ 
10:    else
11:       $S \leftarrow h2$ 
12:    end if
13:  end while
14:  return( $\text{GET-ELEMENT}(S, 1)$ )  $\triangleright$  returns the first (and only) element of  $S$ 
15: end procedure

```

Once a valid direction has been determined, we perform a hill climbing search to find the optimal valid orientation to guaranteed we find a local minimum. One problem we encountered is that the search may determine the edge of a build map to be a local minimum, if its direct neighbor is actually considered to be an invalid direction, with a heuristic of infinity. To fix this problem, when a neighboring point has a lower heuristic, it is added to a queue. Each point on the queue then performs its own hill climbing search until it reaches a local minimum. From each local minimum found, the minimum with the lowest value is chosen. Algorithm 4 describes the hill climbing algorithm.

To weigh a given build direction for a sub volume, we first check if the build direction is in the build map for that sub-volume. If it is not, then the heuristic will be infinity, to show that the direction is not feasible. If not, we calculate the heuristic with the equation below, for a sub-volume with n faces.

$$\frac{\sum_{i=1}^n (z_T |face[i].normal \bullet build_direction|) \cdot face[i].area}{\sum_{i=1}^n face[i].area} \quad (3.2)$$

Algorithm 3 Weigh Build Direction

```

1: procedure WEIGH-BUILD-DIRECTION(map, v, F)    ▷ map is the build map, v is a proposed
   vector, F is a set of faces on a mesh
2:   if v ∉ map then
3:     return(∞)
4:   end if
5:   cusHeight ← 0
6:   area ← 0
7:   for all f ∈ F do
8:     cusHeight ← cusHeight + (zτ × |v · f.normal| × f.area)
9:     area ← area + f.area
10:  end for
11:  cusHeight ←  $\frac{cusHeight}{area}$ 
12:  return(cusHeight)
13: end procedure

```

Algorithm 4 Find Optimal Orientation

```
1: procedure FIND-OPTIMAL-ORIENTATION(map, F)
2:    $V \leftarrow \emptyset$  ▷ all local minimums
3:    $dx \leftarrow (map.width/4, 0)$ 
4:    $dy \leftarrow map.height/4$ 
5:    $Q \leftarrow \{\text{FIND-VALID-ORIENTATION}(map, F)\}$  ▷ queue used for depth-first search
6:   while  $Q \neq \emptyset$  do
7:      $v \leftarrow \text{POP}(Q)$  ▷ returns first element of Q and removes it from set
8:      $h \leftarrow \text{WEIGH-BUILD-DIRECTION}(map, v, F)$  ▷ get heuristics of all directions
9:      $n \leftarrow \text{WEIGH-BUILD-DIRECTION}(map, v + dy, F)$ 
10:     $s \leftarrow \text{WEIGH-BUILD-DIRECTION}(map, v - dy, F)$ 
11:     $e \leftarrow \text{WEIGH-BUILD-DIRECTION}(map, v + dx, F)$ 
12:     $w \leftarrow \text{WEIGH-BUILD-DIRECTION}(map, v - dx, F)$ 
13:    if  $vh < \text{MIN}(n, s, e, w)$  then
14:      if  $dx = 1 \wedge dy = 1$  then
15:         $V \leftarrow V \cup \{v\}$  ▷ local minimum has been found
16:      else
17:         $Q \leftarrow Q \cup \{v\}$  ▷ search needs to be narrowed so add v back to queue
18:      end if
19:    else
20:      if  $n < h$  then
21:         $Q \leftarrow Q \cup \{v + dy\}$ 
22:      end if
23:      if  $s < h$  then
24:         $Q \leftarrow Q \cup \{v - dy\}$ 
25:      end if
26:      if  $e < h$  then
27:         $Q \leftarrow Q \cup \{v + dx\}$ 
28:      end if
29:      if  $w < h$  then
30:         $Q \leftarrow Q \cup \{v - dx\}$ 
31:      end if
32:    end if
33:     $dx \leftarrow dx/2$  ▷ narrow search
34:     $dy \leftarrow dy/2$ 
35:  end while
36:  return( $\text{MIN}(V)$ )
37: end procedure
```

The following figures show an example of a single build map visualized in MATLAB as both a 2-dimensional rectangle and a 3-dimensional semi-sphere. In the 2-dimensional representation, the z value represents the heuristic, with invalid directions are given a heuristic of negative one for readability. On the sphere however, the heuristic is represented with the radius, with invalid directions given a heuristic of zero.

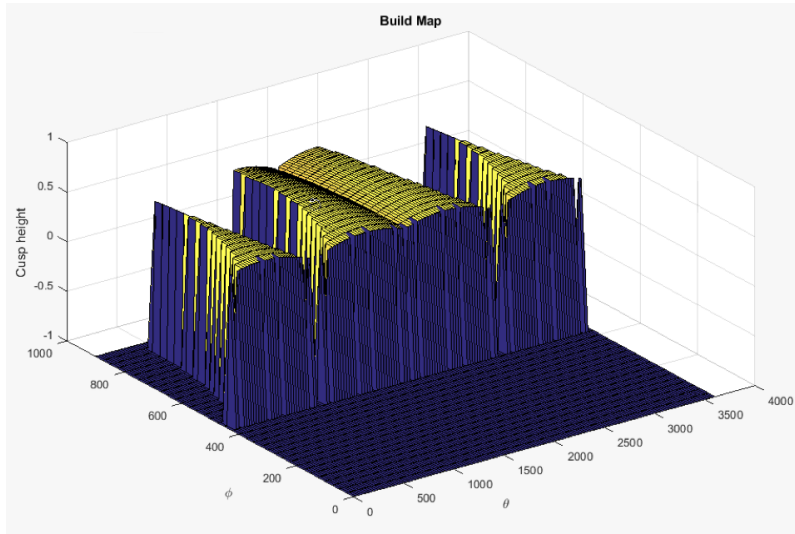


Figure 3.13: Build map 2D representation

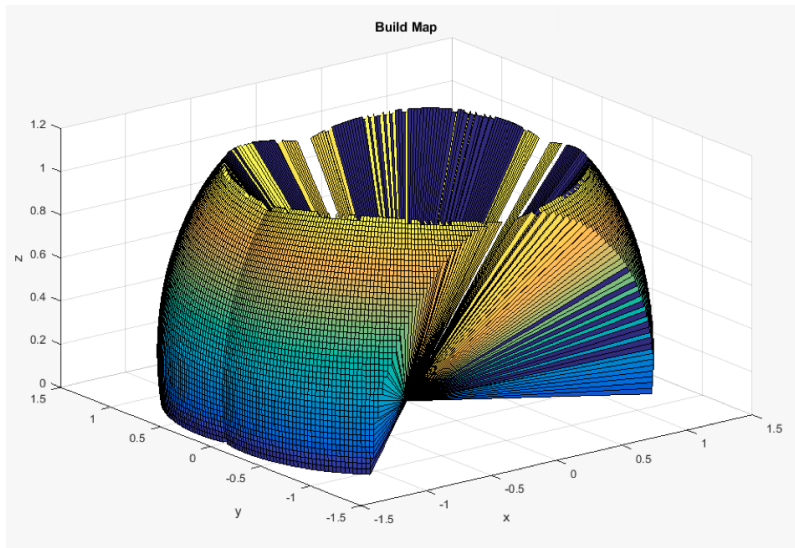


Figure 3.14: Build map 3D representation

3.2.3 Build Sequencing

Once an object is divided into printable sub-volumes, it is necessary to sort these sub-volumes in the order in which they will be printed. Two constraints determine the order of the sub-volume printing. Firstly, all but one sub-volume (the base sub-volume) must be printed on a previously printed sub-volume as opposed to the build plate. Therefore, the sub-volume acting as a base must

be printed first. Secondly, all sub-volumes must be printed in a way that the material-extruding nozzle does not collide with parts of the object which have already been built.

3D printing with 5 degrees of freedom introduces the possibility that, while printing an object, the nozzle will collide with parts of the object that have previously been printed. This is a result of the varying build directions relative to the orientation of the object being printed. These collisions must be avoided at all costs because a collision would result in a failed print. This complex problem of anticipating all possible collisions when printing material can be discretized by recognizing that, since the build direction does not change relative to the sub-volume orientation while printing this sub-volume, collisions cannot occur within a sub-volume. This is the same reason that collisions cannot occur on a 3-axis 3D printer. Therefore, the nozzle can only collide with a sub-volume which is not the sub-volume being currently printed.

These constraints on the sequence of the printing of sub-volumes motivates the use of a directed graph to represent the printing sequence, described in Section 3.2.1. Using this graph, a simple topological sort will determine a valid print sequence of the sub-volumes. In the event of a graph which does not have a valid topological sort, traditional 3D printing can be used to print the entire object. In this case, traditional 3D printing eliminates the benefits of the 5-axis 3D printing. However, accommodations are made in the software to allow for future work in this area, discussed further in Chapter 5.

Implementing build sequencing requires building software which constructs the graph data structure described above, and performing a topological sort on this graph. To construct the graph, volume decomposition establishes the simple relationships between sub-volumes, indicating which sub-volumes are built upon which other sub-volumes. These relationships represent an incomplete sequence graph, with no collisions taken into account. 3.2.4 describes the implementation of a collision detection algorithm which can identify these constraints and add the appropriate edges into the graph.

3.2.4 Collision Detection

Collision detection must be performed for each sub-volume created by volume decomposition. This involves testing the sub-volume against other sub-volumes in the print. To minimize the number of collision detections performed, which involves many expensive polygon operations using the Clipper library, an oriented bounded box collision detection is first performed. Any collisions detected by the oriented bounded box (OBB) collision detection are then tested again with a more precise technique. The OBB collision detection is outlined in Algorithm 5. This algorithm creates an oriented bounding box for the subject sub-volume, and then extends the OBB in the direction of the build direction. This additional area of the OBB represents the space which the nozzle will occupy when printing the sub-volume. To detect collisions, an OBB for each other sub-volume in the print is created, then checked for intersections with the subject sub-volume. If the axis-aligned bounding box of a sub-volume is found to intersect with the OBB of the subject sub-volume, the ID of this sub-volume is added to the list of possible collision ID's.

Algorithm 5 OBB Collision Detection

```
1: procedure OBB-COLLISION-DETECTION(sequenceGraphNode) ▷  
   sequenceGraphNode is the node being printed on the semi-constructed sequence graph, which  
   has an empty set of collision edges ▷ returns the ID's of sub-volume nodes that may cause  
   collisions with the nozzle while printing sequenceGraphNode  
2:   ret  $\leftarrow \emptyset$  ▷ set of ID's of graph nodes which may cause collisions  
3:   sequenceGraph ▷ the entire Sequence Graph  
4:   transformation  $\leftarrow$  the sequenceGraphNode's transformation  
5:   printingOBB  $\leftarrow$  the OBB of sequenceGraphNode's mesh  
6:   printingOBB.EXTEND(transformation)  
7:   for all node  $\in$  sequenceGraph do  
8:     collisionAABB  $\leftarrow$  the AABB of the node's graph  
9:     if PRINTINGOBB.HIT(collisionAABB) then  
10:       ret  $\leftarrow$  ret  $\cup$  node.ID  
11:     end if  
12:   end for  
13:   return(ret)  
14: end procedure
```

OBB collision detection may find false positives, however it does not find any false negatives. Therefore, performing precise collision detection on the nodes found by OBB collision detection finds all collisions with no false positives or negatives. Once OBB collision detection is complete, precise collision detection is performed, ignoring all but the candidate nodes identified by OBB collision detection.

The precise collision detection algorithm, which detects collisions with no false positives or false negatives, works similarly to Algorithm 5 in that the sub-volume is extended upwards in its build direction, as to represent the area in which the printing nozzle will occupy. Once this volume has been extended, it must be checked for intersections with other sub-volumes which have been determined to be candidates by the OBB Collision detection, as described in Algorithm 6. To detect intersections, a slice is taken on the same plane for the subject sub-volume mesh and all candidate meshes. The slice of the subject mesh is unioned with all slices beneath it, to account for the upwards (in the positive build direction) expansion of the sub-volume, which represent the printer nozzle location. Then this modified slice is checked for intersection against all other slices along this plane. This is repeated with parallel planes along the build direction.

Algorithm 6 Precise Collision Detection

```
1: procedure PRECISE-COLLISION-DETECTION(sequenceGraphNode, candidates)  $\triangleright$   
   sequenceGraphNode is the node being printed on the semi-constructed sequence graph, which  
   has an empty set of collision edges  $\triangleright$  candidates is a collection of ID's of nodes which  
   sequenceGraphNode will be tested against  $\triangleright$  returns the ID's of sub-volume nodes that cause  
   collisions with the nozzle while printing sequenceGraphNode  
2:   ret  $\leftarrow \emptyset$   $\triangleright$  set of node IDs that collide with mesh  
3:   height  $\leftarrow 0$   
4:   while height < MAX-HEIGHT(sequenceGraphNode) do  $\triangleright$  continues until we are past all  
   meshes  
5:     S  $\leftarrow \emptyset$   $\triangleright$  slices for each mesh in M  
6:     for all c  $\in$  candidates do  
7:       tempSlice  $\leftarrow$  SLICE(m, Plane(height, buildDirection))  
8:       tempSlice.parent  $\leftarrow$  m  
9:       S  $\leftarrow S \cup \{tempSlice\}$   
10:    end for  
11:    slice  $\leftarrow$  slice  $\cup$  SLICE(mesh, height)  $\triangleright$  union slice with union of all previous slices  
12:    expandedSlice  $\leftarrow$  EXPAND(slice, rτ)  $\triangleright$  rτ is the radius of the nozzle  
13:    for all s  $\in$  S do  
14:      if INTERSECTS(expandedSlice, s) then  
15:        ret = ret  $\cup$  {s.parent}  
16:      end if  
17:    end for  
18:    height  $\leftarrow$  height + 1  
19:  end while  
20:  return(ret)  
21: end procedure
```

3.2.5 5-axis G-code Generation

Once overhanging volumes have been identified as sub-volumes, an optimal build direction has been determined for each sub-volume, and a valid sequence for the building of each sub-volume has been determined, the next step is to send instructions to control the printer. These instructions are sent in the form of G-code, a set of locations and instructions for every movement and action the printer must make. To generate these instructions, each sub-volume must be sliced perpendicular to the build orientation, such that each sub-volume is cut into flat layers along the axis of its build orientation.

There are several existing open source software packages that convert 3D models into 3D printer G-code by performing slice operations on the entire volume along the *z*-axis, and converting the sliced object into printer instructions. This software is suitable for use within the 5-axis slicer software, with each sub-volume being sliced and converted into G-code individually through the open source program. We used Ultimaker's Cura [20] to perform these operations, which have been extensively studied and implemented.

3.3 Printer

3.3.1 Mechanical Design

In order to enable the printer to reach any point and orientation in the build space, the printer required 5 independent stages, two rotational and three linear. The three linear stages comprise the Cartesian movements in X, Y, and Z while the two rotary stages, denoted as A and B, were responsible for changing the orientation of the nozzle. The key design criteria the printer had to satisfy were as follows:

- Relatively easy to manufacture
- Modular for future development with other printing processes such as cold spray
- Durable to withstand harsher printing processes
- Upgradeable
- Accurate to 10 microns

In many processes, such as cold spray, the nozzle assembly for depositing the material is unwieldy, complex, and heavy. This prompted the design of the stages to be centered on actuating the build plate while the nozzle remained stationary. Additionally, both the linear stages and rotary stages need to be in series respectively to enable the machine to reach any three dimensional orientation and position. This narrowed the stage configurations to two possible arrangements, either the rotary stages formed the base and the linear stages attached atop them or the linear stages formed the base with the rotary stages attached atop. The latter was chosen because it minimized the total amount of power needed to actuate the rotary stages. The printer was modeled in Solidworks first. The material of choice was aluminum 6061-T6 because of its easy machinability, ready availability, and low cost.

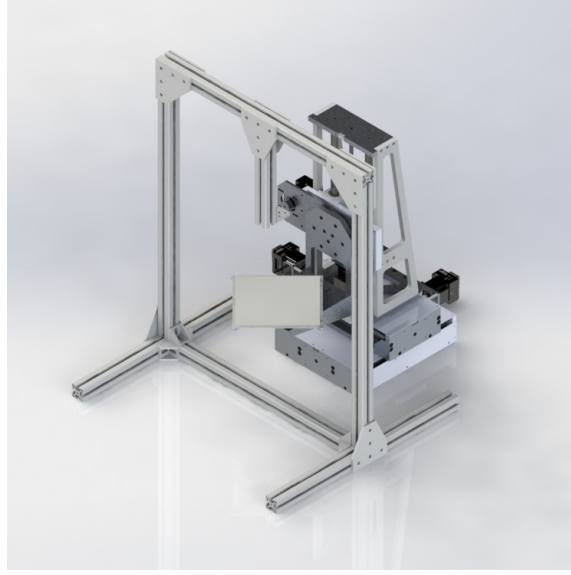


Figure 3.15: CAD Rendering of 5-axis 3D Printer

Linear Stages

Given the weight demands, each of the linear stages utilized a leadscrew driven by a stepper motor for increased torque when compared to belt driven stages. The critical design choices were the diameter of the leadscrew and pitch. Additionally, the linear stages in concert needed to be able to make precise movements of 10 microns. Standard stepper motors offer 1.8° angular accuracy with microstepping up to a division of 8 typically. Thus most stepper motors are capable of 0.225° step sizes. For the prototype of the printer, it was unlikely movement of the rotary stages and linear stages was possible due to the constraints with the slicer algorithm. Therefore the assumption was made that the rotary stages remain stationary during printing and the maximum resolution, or minimum distance, occurs when all linear stages are moving then the individual resolutions at minimum must be $\frac{10\mu\text{m}}{\sqrt{3}} \approx 5.773 \mu\text{m}$ according to Pythagorean Theorem. The critical pitch, largest possible pitch which satisfies the design criteria, can thus be calculated using $p_c = 5.773 \mu\text{m} \left(1600 \frac{\text{steps}}{\text{rev}}\right)$, which for a 0.225° step size evaluates to approximately 9.237 millimeters or 0.364 inches.

All three of the stages utilize stepper motors with coaxial encoders which allow for closed loop feedback. The ends plates of each stage have limit switches for detection of when the stages carriages have reached the end of their travel.

The torque required to turn the leadscrew was calculated with the following equations [5]. Note that the torque to lower the carriage on the stage only applies to the Z stage, because both the X and Y stages are arranged perpendicular to the pull of gravity and thus only require use the torque to raise. Additionally, this particular application requires precision movement and as such an ACME leadscrew will be used giving a thread angle of 29° .

$$T_{\text{raise}} = \frac{F d_m}{2} \left(\frac{l + \pi \mu d_m \sec \alpha}{\pi d_m - \mu l \sec \alpha} \right) \quad (3.3)$$

$$T_{\text{lower}} = \frac{F d_m}{2} \left(\frac{\pi \mu d_m \sec \alpha - l}{\pi d_m - \mu l \sec \alpha} \right) \quad (3.4)$$

In the above equations, F is the applied load on the leadscrew, d_m is the mean diameter, l is the lead, also called pitch, α is half the thread angle, and μ is the coefficient of friction for the leadscrew and the nut. Using the equations above, a 0.1 inch leadscrew pitch and the approximate weight for the Y stage to carry given the Solidworks model as well as a factor of safety of two accounting for 50 pounds then the torque required to turn the screw is approximately 47.5 oz-in given the coefficient of friction for polyacetal nut on steel is 0.25. With the above criteria a NEMA 23 175 oz-in stepper motor from Anaheim Automation was selected for two reasons, it provided a maximum speed for the given torque of 31.75 mm/s which is comparable to print speeds on many consumer printers for high resolution and it had an integrated driver.

The print volume was determined by comparing commercially available consumer grade 3D printers. In the future, it is desired that the printer be tested for build quality against comparable printers therefore a cubic build volume with 8 inch sides was chosen. In turn, this means that each stage needed to actuate 8 inches. This resulted in the X and Y stages having a total length of 12 inches, 4 inches of which accounted for the carriage and the remaining for travel. The Z stage had an additional half inch of length because it had to accommodate large linear bearings due to the size of the linear shafts but retained the same amount of total travel.

X & Y Stages

The X and Y stages formed the base of the printer. Each stage was comprised of several main components: the carriage, leadscrew, guide rails, end plates, and bottom plates. The carriage included a shallow U-shaped cross section. The pocket allowed for the addition of sheet metal plate to be placed over top and secured on the end plates. In conjunction with the sheet metal plates added to the sides, positive air pressure could be flown through the stage to prevent dirt and/or powder from collecting inside the stage and binding the moving components. In addition, the carriage had four linear bearings capable of 1° of misalignment, two for each guide rail. In the center affixed via an angle bracket was the anti-backlash nut. The anti-backlash nut was made from polyacetal and therefore self-lubricating and removed slop from the leadscrew and nut with a preloading spring. The guide rails were fully supported to minimize deflection and increase the levelness of the machine as opposed to unsupported horizontal rods. In addition, they were responsible for carrying the load of the stages atop them, freeing the leadscrew to only withstand the axial load and torque from actuation.

Z Stage

Unlike the X and Y stages, the Z Stage is mounted vertically. Though, guide rails would also be suitable in this orientation, they would require the addition of a back plate which would be heavy and significantly add to the overall weight of the assembly. Therefore, larger three quarter inch linear shafts of high carbon steel were used to minimize deflection of the build plate.

Rotary Stages

Both rotary stages are comprised of a slew ring, timing belt pulleys, and the stepper motors. The A rotary stage is affixed to the carriage of the Z-axis stage and rotates about the local Y-coordinate axis. Mounted perpendicularly to the A stage is the B stage which rotates about the local Z-axis. Together, any orientation of the nozzle is achievable in 3D space. In both cases, the standard stepper motor shaft size for a NEMA 23 motor was a quarter inch which limited the size of the timing belt pulleys that were compatible with that shaft size to 14 and 72 teeth as the minimum and maximum available on McMaster-Carr. This limitation set the upper limit of the prototypes angular resolution to 80 microns at a 4 inch radius given a gear ratio of approximately 5.143. Increasing the resolution of microstepping with an alternative driver capable of above 1600 steps per revolution is feasible as a retro fit in the future. Cost limitations and machine availability restricted the possibility of higher resolution microstepping drivers.

3.3.2 Nozzle Gantry

The nozzle is held by a separate gantry which is not physically connected to the printer. This is a safety precaution to prevent major machine damage if the nozzle was to collide with the build plate or printer. Instead of bending, the gantry will be pushed away. Additionally, the gantry holds the extruder motor and filament holder. The extruder motor is NEMA 17 stepper motor salvaged from a 3D printer kit with an accuracy of 0.225° when microstepping. The gantry was constructed from 80-20 aluminum extrusions for modularity and allowed for easy adjustment of the nozzle height. The hot end was affixed using angle bracket with a slotted clamping plate that snugly fit with the grooves in the hot end and prevented the it from moving vertically. Three touch servo touch probes were attached to the nozzle hot end to be used in the printers calibration which is detailed in the following section.

3.3.3 Printer Calibration

It is impossible to ensure perfect alignment of the nozzle normal to the build plate. As such, calibration is required to account for discrepancies and enable the printer to compensate. Here a simplistic approach is detailed although more advanced techniques could be employed later on. First, the assumption is made that the build plate surface is flat. However, its coordinate frame may be misaligned with respect the idealized coordinate frame in which the nozzle is perpendicular to the xy plane of the build plate.

To account for the build plate's orientation being misaligned two independent angles are required which have been defined as ρ and ψ , where ρ is the angle the normal of the misaligned build plane makes with the z unit axis of the idealized coordinate system when projected into the x-z plane of the parent coordinate system and ψ is the angle between the normal of the misaligned plane and the projection in the x-z plane. This can be seen in Figure 3.16.

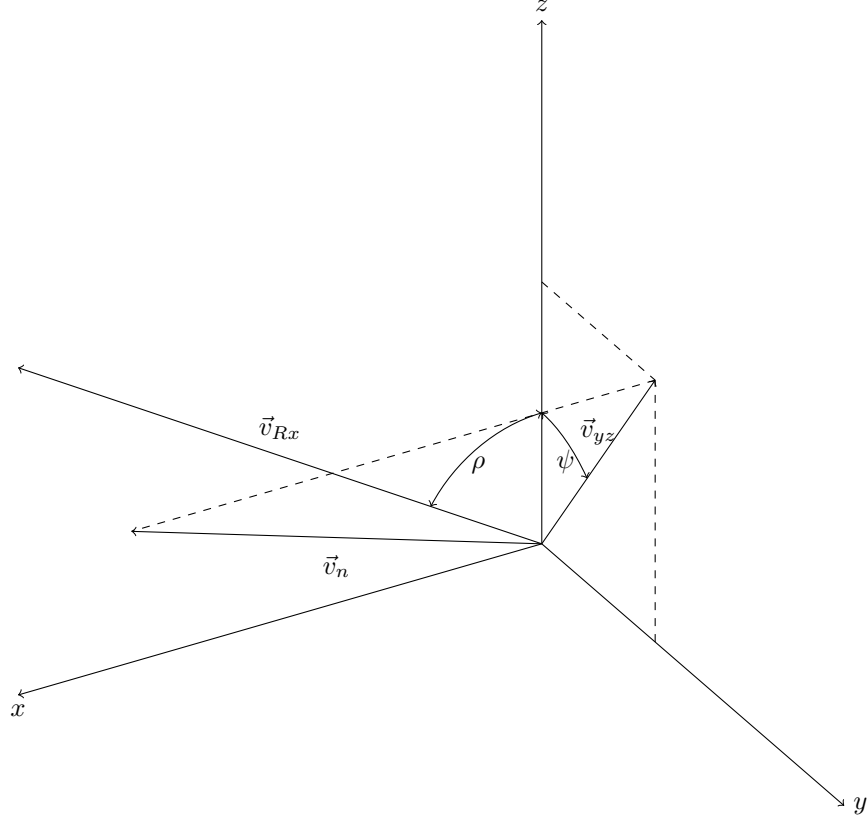


Figure 3.16: Build plate normal defined by ρ and ψ in stage coordinate system

Each of the angles will constitute a rotational coordinate transformation shown below in Equation 3.5.

$$\begin{aligned}
 R_y(\rho)R_x(\psi) &= \begin{bmatrix} \cos \rho & 0 & \sin \rho & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \rho & 0 & \cos \rho & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \psi & -\sin \psi & 0 \\ 0 & \sin \psi & \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \cos \rho & 0 & \sin \rho & 0 \\ \sin \psi \sin \rho & \cos \psi & -\cos \rho \sin \psi & 0 \\ -\cos \psi \sin \rho & \sin \psi & \cos \psi \cos \rho & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}
 \end{aligned}$$

To find ρ and ψ the first step is to probe the build surface with three points, measuring the point at which the nozzle makes contact with the surface and recording the relative coordinates. These three points constitute a plane, the normal of which represents the z unit vector of the local coordinate system. This normal can then be used to calculate the angles.

Given $A = (x_a, y_a, z_a)$, $B = (x_b, y_b, z_b)$, and $C = (x_c, y_c, z_c)$, two vectors can be constructed, \vec{AB} and \vec{BC} . The normal of these two vectors is found via their cross product.

$$\vec{AB} = (x_a - x_b)\hat{i} + (y_a - y_b)\hat{j} + z_{ab}\hat{k} = x_{ab}\hat{i} + y_{ab}\hat{j} + z_a\hat{k} \quad (3.6a)$$

$$\vec{BC} = (x_b - x_c)\hat{i} + (y_b - y_c)\hat{j} + (z_b - z_c)\hat{k} = x_{bc}\hat{i} + y_{bc}\hat{j} + z_{bc}\hat{k} \quad (3.6b)$$

$$\begin{aligned} \vec{v}_n = \vec{AB} \times \vec{BC} &= \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_{ab} & y_{ab} & z_{ab} \\ x_{bc} & y_{bc} & z_{bc} \end{vmatrix} \\ &= (y_{ab}z_{bc} - z_{ab}y_{bc})\hat{i} + (z_{ab}x_{bc} - x_{ab}z_{bc})\hat{j} + (x_{ab}y_{bc} - y_{ab}x_{bc})\hat{k} = x_n\hat{i} + y_n\hat{j} + z_n\hat{k} \end{aligned} \quad (3.7)$$

Now that the components for the normal vector of the build plane have been calculated in the parent coordinate system, it is possible to compute ψ using the dot product between the projection of the normal into the x-z plane and it.

$$\vec{v}_{yz} = y_n\hat{j} + z_n\hat{k} \quad (3.8)$$

$$\vec{v}_n \cdot \vec{v}_{yz} = |\vec{v}_n||\vec{v}_{yz}|\cos(\psi) \quad (3.9)$$

$$\psi = \cos^{-1} \left(\frac{\vec{v}_n \cdot \vec{v}_p}{|\vec{v}_n||\vec{v}_p|} \right) = \cos^{-1} \left(\frac{y_n^2 + z_n^2}{\sqrt{(x_n^2 + y_n^2 + z_n^2)(y_n^2 + z_n^2)}} \right) = \cos^{-1} \left(\sqrt{\frac{y_n^2 + z_n^2}{x_n^2 + y_n^2 + z_n^2}} \right) \quad (3.10)$$

The same process is used to compute ρ . However, the projection is dotted with the z axis for the parent coordinate system.

$$\vec{v}_{Rx} = x_n\hat{j} + \sqrt{y_n^2 + z_n^2}\hat{k} \quad (3.11)$$

$$\vec{v}_{Rx} \cdot \hat{z} = |\vec{v}_{Rx}||\hat{z}|\cos(\rho)$$

$$\rho = \cos^{-1} \left(\frac{\vec{v}_{Rx} \cdot \hat{z}}{|\vec{v}_{Rx}||\hat{z}|} \right) = \cos^{-1} \left(\frac{z_n\sqrt{y_n^2 + z_n^2}}{\sqrt{x_n^2 + y_n^2 + z_n^2}} \right) \quad (3.12)$$

3.3.4 Forward Kinematics

Unlike conventional 3D printers that traditionally operate with three orthogonal stages, X, Y, and Z, a 5 axis 3D printer has two additional rotary axes, A and B. Depending on their orientation at any point during printing, it is not necessarily true that the positions of the linear stages map directly to the position above the build plate. Therefore, coordinate transformation matrices are required to convert frame positions to build space positions, given θ and ϕ , the angles of the rotary axes A and B respectively.

Figure 3.17 shows the three coordinate frames of interest for the printer. For clarity, the individual joint transformations have been omitted. Note in the configuration of the printer's axes, S is a constant which is experimentally obtained at the start of a print and remains constant throughout the print. However, the position vectors s and $v_{build\ plate}$ change depending on the position the nozzle must be over the build plate.

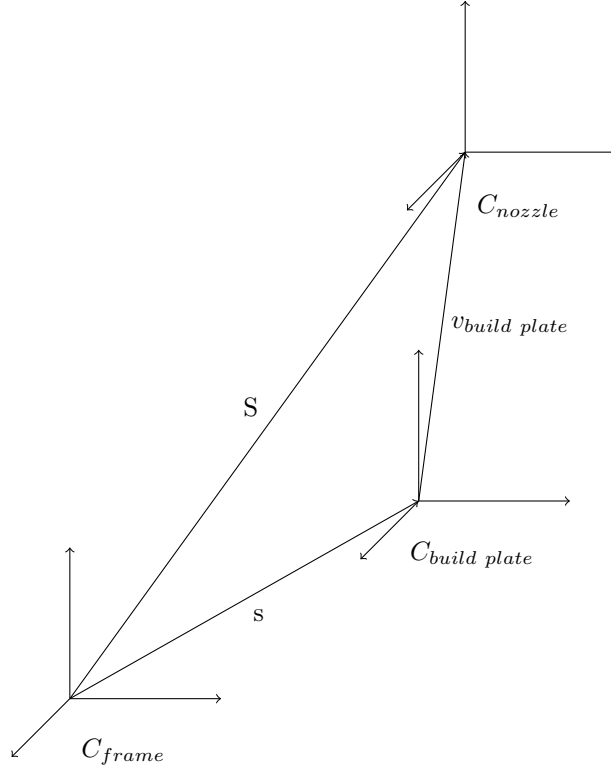


Figure 3.17: Coordinate Systems

Position

The first task is to develop a transform that maps vectors inside the nozzle coordinate frame to the printer's frame coordinate frame. This is done using homogeneous coordinate transformations. Equation 3.13 is comprised of the individual transformations between each of the stages of the printer and the calibration matrices from the previous section. Note T_{frame} is comprised of the x, y, and z positions of the stages because each of the stages are orthogonal and combining them into a single transform preserves some brevity. Each transform in their matrix representations are shown below.

$$H = T_{frame}(-T_{offset})R(\theta)_yT_{offset}R(\phi)_zR(\rho)_yR(\psi)_xT_{build\ plate} \quad (3.13)$$

$$\begin{aligned}
T_{frame} &= \begin{bmatrix} 1 & 0 & 0 & x_{frame} \\ 0 & 1 & 0 & y_{frame} \\ 0 & 0 & 1 & z_{frame} \\ 0 & 0 & 0 & 1 \end{bmatrix} & T_{offset} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -z_{offset} \\ 0 & 0 & 0 & 1 \end{bmatrix} & R(\theta)_y &= \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
R(\phi)_z &= \begin{bmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & R(\rho)_y &= \begin{bmatrix} \cos \rho & 0 & \sin \rho & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \rho & 0 & \cos \rho & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & R(\psi)_x &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \psi & -\sin \psi & 0 \\ 0 & \sin \psi & \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
T_{build\ plate} &= \begin{bmatrix} 1 & 0 & 0 & x_{build\ plate} \\ 0 & 1 & 0 & y_{build\ plate} \\ 0 & 0 & 1 & z_{build\ plate} \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

The homogeneous transformation matrix H takes the form shown in Equation 3.14a where R is a 3 by 3 rotational transformation and P is position vector [7].

$$H = \begin{bmatrix} R & P \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.14a)$$

$$R = \begin{bmatrix} c_\phi c_\rho c_\theta - s_\rho s_\theta & s_\psi (c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi & c_\psi (c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi \\ c_\rho s_\phi & c_\phi c_\psi + s_\phi s_\psi s_\rho & c_\psi s_\phi s_\rho - c_\phi s_\psi \\ -c_\theta s_\rho - c_\phi c_\rho s_\theta & s_\psi (c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta & c_\psi (c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta \end{bmatrix} \quad (3.14b)$$

$$P = \begin{bmatrix} x_{frame} - x_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) - z_{offset} s_\theta + y_{build\ plate}(s_\psi (c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\ \quad + z_{build\ plate}(c_\psi (c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi) \\ y_{frame} + y_{build\ plate}(c_\phi c_\psi + s_\phi s_\psi s_\rho) - z_{build\ plate}(c_\phi s_\psi - c_\psi s_\phi s_\rho) + x_{build\ plate} c_\rho s_\phi \\ z_{offset} + z_{frame} - x_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) - z_{offset} c_\theta + y_{build\ plate}(s_\psi (c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) \\ \quad + z_{build\ plate}(c_\psi (c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) \end{bmatrix} \quad (3.14c)$$

As was mentioned in the beginning of this section, S remains constant for the duration of the print but must be experimentally determined at the beginning of it based on known positions of the stages such as when the nozzle is located at the origin of the build plate. Equation 3.15 shows that S corresponds to P given that it always points to the origin of the nozzle.

$$S = H \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} P \\ 1 \end{bmatrix} = \text{constant} \quad (3.15)$$

Velocity

Although the forward kinematics for velocity will not be of much use because S is constant and therefore its derivative is 0, the equation $0 = \dot{P}$ will be of interest in the next section for finding the velocity of the stages and nozzle above the build plate. P is differentiated by first computing the Jacobian of P and then multiplying by the velocity joint space vector which includes the velocities for both the x y z of the frame and build plate coordinate systems as well as θ and ϕ . Note that ψ and ρ do not change during printing and are constants therefore they are excluded from the joint space vector.

$$0 = \dot{P} = J(q)\dot{q} \quad (3.16a)$$

$$\dot{q} = \begin{bmatrix} \dot{x}_{frame} \\ \dot{y}_{frame} \\ \dot{z}_{frame} \\ \dot{x}_{build\ plate} \\ \dot{y}_{build\ plate} \\ \dot{z}_{build\ plate} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} \quad (3.16b)$$

$$J(q) = \begin{bmatrix} 1 & 0 & 0 & j_1 & j_2 & j_3 & j_4 & j_5 \\ 0 & 1 & 0 & j_6 & j_7 & j_8 & j_9 & j_{10} \\ 0 & 0 & 1 & j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \quad (3.16c)$$

$$\begin{aligned} j_1 &= y_{build\ plate}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) - z_{offset}c_\theta - x_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) \\ &\quad + z_{build\ plate}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) \\ j_2 &= z_{build\ plate}(c_\phi c_\theta s_\psi - c_\psi c_\theta s_\phi s_\rho) - y_{build\ plate}(c_\phi c_\psi c_\theta + c_\theta s_\phi s_\psi s_\rho) - x_{build\ plate}c_\rho c_\theta s_\phi \\ j_3 &= c_\phi c_\rho c_\theta - s_\rho s_\theta \\ j_4 &= s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi \\ j_5 &= c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi \\ j_6 &= 0 \\ j_7 &= z_{build\ plate}(s_\phi s_\psi + c_\phi c_\psi s_\rho) - y_{build\ plate}(c_\psi s_\phi - c_\phi s_\psi s_\rho) + x_{build\ plate}c_\phi c_\rho \\ j_8 &= c_\rho s_\phi \\ j_9 &= c_\phi c_\psi + s_\phi s_\psi s_\rho \\ j_{10} &= c_\psi s_\phi s_\rho - c_\phi s_\psi \\ j_{11} &= x_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) + z_{offset}s_\theta - y_{build\ plate}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\ &\quad - z_{build\ plate}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi) \\ j_{12} &= y_{build\ plate}(c_\phi c_\psi s_\theta + s_\phi s_\psi s_\rho s_\theta) - z_{build\ plate}(c_\phi s_\psi s_\theta - c_\psi s_\phi s_\rho s_\theta) + x_{build\ plate}c_\rho s_\phi s_\theta \\ j_{13} &= -c_\theta s_\rho - c_\phi c_\rho s_\theta \\ j_{14} &= s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta \\ j_{15} &= c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta \end{aligned}$$

Acceleration

Now differentiating \dot{P} a second time yields the acceleration equations. Again the second derivative of S is 0, but now the chain rule must be applied to the Jacobian and joint space vector as

seen in Equations 3.17a, 3.17b, and 3.17c. Note that the acceleration joint space vector is now comprised of the acceleration for the angles of the rotaries, and the x, y, z of the build plate and frame. Additionally, the derivative of Jacobian results in the first 3 by 3 sub matrix being a set of zeroes, which indicates that the velocities of the frame have no bearing on the accelerations.

$$0 = \ddot{P} = \dot{J}(q)\dot{q} + J(q)\ddot{q} \quad (3.17a)$$

$$\ddot{q} = \begin{bmatrix} \ddot{x}_{frame} \\ \ddot{y}_{frame} \\ \ddot{z}_{frame} \\ \ddot{x}_{build\ plate} \\ \ddot{y}_{build\ plate} \\ \ddot{z}_{build\ plate} \\ \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} \quad (3.17b)$$

$$\dot{J}(q) = \begin{bmatrix} 0 & 0 & 0 & \dot{j}_1 & \dot{j}_2 & \dot{j}_3 & \dot{j}_4 & \dot{j}_5 \\ 0 & 0 & 0 & \dot{j}_6 & \dot{j}_7 & \dot{j}_8 & \dot{j}_9 & \dot{j}_{10} \\ 0 & 0 & 0 & \dot{j}_{11} & \dot{j}_{12} & \dot{j}_{13} & \dot{j}_{14} & \dot{j}_{15} \end{bmatrix} \quad (3.17c)$$

$$\begin{aligned}
\dot{j}_1 &= \dot{\theta}(x_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) + z_{offset} s_\theta - y_{build\ plate}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\
&\quad - z_{build\ plate}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi)) - \dot{x}_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) \\
&\quad + \dot{\phi}(y_{build\ plate}(c_\phi c_\psi s_\theta + s_\phi s_\psi s_\rho s_\theta) - z_{build\ plate}(c_\phi s_\psi s_\theta - c_\psi s_\phi s_\rho s_\theta) \\
&\quad + x_{build\ plate} c_\rho s_\phi s_\theta) + \dot{y}_{build\ plate}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) \\
&\quad + \dot{z}_{build\ plate}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) \\
\dot{j}_2 &= \dot{\theta}(y_{build\ plate}(c_\phi c_\psi s_\theta + s_\phi s_\psi s_\rho s_\theta) - z_{build\ plate}(c_\phi s_\psi s_\theta - c_\psi s_\phi s_\rho s_\theta) + x_{build\ plate} c_\rho s_\phi s_\theta) \\
&\quad - \dot{\phi}(z_{build\ plate}(c_\theta s_\phi s_\psi + c_\phi c_\psi c_\theta s_\rho) - y_{build\ plate}(c_\psi c_\theta s_\phi - c_\phi c_\theta s_\psi s_\rho) + x_{build\ plate} c_\phi c_\rho c_\theta) \\
&\quad - \dot{y}_{build\ plate}(c_\phi c_\psi c_\theta + c_\theta s_\phi s_\psi s_\rho) + \dot{z}_{build\ plate}(c_\phi c_\theta s_\psi - c_\psi c_\theta s_\phi s_\rho) - \dot{x}_{build\ plate} c_\rho c_\theta s_\phi \\
\dot{j}_3 &= -\dot{\theta}(c_\theta s_\rho + c_\phi c_\rho s_\theta) - \dot{\phi} c_\rho c_\theta s_\phi \\
\dot{j}_4 &= \dot{\theta}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) - \dot{\phi}(c_\phi c_\psi c_\theta + c_\theta s_\phi s_\psi s_\rho) \\
\dot{j}_5 &= \dot{\theta}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) + \dot{\phi}(c_\phi c_\theta s_\psi - c_\psi c_\theta s_\phi s_\rho) \\
\dot{j}_6 &= 0 \\
\dot{j}_7 &= \dot{z}_{build\ plate}(s_\phi s_\psi + c_\phi c_\psi s_\rho) - \dot{y}_{build\ plate}(c_\psi s_\phi - c_\phi s_\psi s_\rho) - \dot{\phi}(y_{build\ plate}(c_\phi c_\psi + s_\phi s_\psi s_\rho) \\
&\quad - z_{build\ plate}(c_\phi s_\psi - c_\psi s_\phi s_\rho) + x_{build\ plate} c_\rho s_\phi) + \dot{x}_{build\ plate} c_\phi c_\rho \\
\dot{j}_8 &= \dot{\phi} c_\phi c_\rho \\
\dot{j}_9 &= -\dot{\phi}(c_\psi s_\phi - c_\phi s_\psi s_\rho) \\
\dot{j}_{10} &= \dot{\phi}(s_\phi s_\psi + c_\phi c_\psi s_\rho) \\
\dot{j}_{11} &= \dot{x}_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) + \dot{\phi}(y_{build\ plate}(c_\phi c_\psi c_\theta + c_\theta s_\phi s_\psi s_\rho) - z_{build\ plate}(c_\phi c_\theta s_\psi - c_\psi c_\theta s_\phi s_\rho) \\
&\quad + x_{build\ plate} c_\rho c_\theta s_\phi) - \dot{y}_{build\ plate}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\
&\quad - \dot{z}_{build\ plate}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi) + \dot{\theta}(x_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) + z_{offset} c_\theta \\
&\quad - y_{build\ plate}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) - z_{build\ plate}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta)) \\
\dot{j}_{12} &= \dot{y}_{build\ plate}(c_\phi c_\psi s_\theta + s_\phi s_\psi s_\rho s_\theta) - \dot{z}_{build\ plate}(c_\phi s_\psi s_\theta - c_\psi s_\phi s_\rho s_\theta) \\
&\quad + \dot{\theta}(y_{build\ plate}(c_\phi c_\psi c_\theta + c_\theta s_\phi s_\psi s_\rho) - z_{build\ plate}(c_\phi c_\theta s_\psi - c_\psi c_\theta s_\phi s_\rho) \\
&\quad + x_{build\ plate} c_\rho c_\theta s_\phi) + \dot{\phi}(z_{build\ plate}(s_\phi s_\psi s_\theta + c_\phi c_\psi s_\rho s_\theta) \\
&\quad - y_{build\ plate}(c_\psi s_\phi s_\theta - c_\phi s_\psi s_\rho s_\theta) + x_{build\ plate} c_\phi c_\rho s_\theta) \\
&\quad + \dot{x}_{build\ plate} c_\rho s_\phi s_\theta \\
\dot{j}_{13} &= \dot{\theta}(s_\rho s_\theta - c_\phi c_\rho c_\theta) + \dot{\phi} c_\rho s_\phi s_\theta \\
\dot{j}_{14} &= \dot{\phi}(c_\phi c_\psi s_\theta + s_\phi s_\psi s_\rho s_\theta) - \dot{\theta}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\
\dot{j}_{15} &= -\dot{\phi}(c_\phi s_\psi s_\theta - c_\psi s_\phi s_\rho s_\theta) - \dot{\theta}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi)
\end{aligned}$$

3.3.5 Inverse Kinematics

Position of the Frame Given Nozzle Position

Using the equations derived in the previous section it is now possible to compute the joint positions (i.e. the positions of the individuals stages) given the desired position above the build plate. Recall that S is constant during printing and represents the position vector of the nozzle inside the coordinate system of the frame. Also note that the frame position happens to be a simple addition to the vector and therefore, it can be separated and the equation can be rearranged to give the frame position in terms of the build plate position and the rotary positions.

$$S = P$$

$$S = \begin{bmatrix} x_{frame} - x_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) - z_{offset} s_\theta + y_{build\ plate}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\ \quad + z_{build\ plate}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi) \\ y_{frame} + y_{build\ plate}(c_\phi c_\psi + s_\phi s_\psi s_\rho) - z_{build\ plate}(c_\phi s_\psi - c_\psi s_\phi s_\rho) + x_{build\ plate} c_\rho s_\phi \\ z_{offset} + z_{frame} - x_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) - z_{offset} c_\theta + y_{build\ plate}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) \\ \quad + z_{build\ plate}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) \end{bmatrix}$$

$$S = \begin{bmatrix} x_{frame} \\ y_{frame} \\ z_{frame} \end{bmatrix} + \begin{bmatrix} -x_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) - z_{offset} s_\theta + y_{build\ plate}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\ \quad + z_{build\ plate}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi) \\ y_{build\ plate}(c_\phi c_\psi + s_\phi s_\psi s_\rho) - z_{build\ plate}(c_\phi s_\psi - c_\psi s_\phi s_\rho) + x_{build\ plate} c_\rho s_\phi \\ z_{offset} - x_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) - z_{offset} c_\theta + y_{build\ plate}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) \\ \quad + z_{build\ plate}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) \end{bmatrix}$$

$$S = P_{frame} + \begin{bmatrix} -x_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) - z_{offset} s_\theta + y_{build\ plate}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\ \quad + z_{build\ plate}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi) \\ y_{build\ plate}(c_\phi c_\psi + s_\phi s_\psi s_\rho) - z_{build\ plate}(c_\phi s_\psi - c_\psi s_\phi s_\rho) + x_{build\ plate} c_\rho s_\phi \\ z_{offset} - x_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) - z_{offset} c_\theta + y_{build\ plate}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) \\ \quad + z_{build\ plate}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) \end{bmatrix}$$

$$P_{frame} = S - \begin{bmatrix} -x_{build\ plate}(s_\rho s_\theta - c_\phi c_\rho c_\theta) - z_{offset} s_\theta + y_{build\ plate}(s_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi) \\ \quad + z_{build\ plate}(c_\psi(c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi) \\ y_{build\ plate}(c_\phi c_\psi + s_\phi s_\psi s_\rho) - z_{build\ plate}(c_\phi s_\psi - c_\psi s_\phi s_\rho) + x_{build\ plate} c_\rho s_\phi \\ z_{offset} - x_{build\ plate}(c_\theta s_\rho + c_\phi c_\rho s_\theta) - z_{offset} c_\theta + y_{build\ plate}(s_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta) \\ \quad + z_{build\ plate}(c_\psi(c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta) \end{bmatrix}$$

Position of Nozzle Above Build Plate

In addition to knowing the position of the frame, it is also desirable to be able to confirm the position above the build plate given the frame positions so as to error correct for quantized movements of the stepper motors in otherwise unconstrained G-code positions provided by the slicer. Following the methodology above, the position above the build plate can be separated, although this time with slightly more difficulty.

$$S = \begin{bmatrix} x_{frame} \\ y_{frame} \\ z_{frame} \end{bmatrix} + \begin{bmatrix} -z_{offset} s_\theta \\ 0 \\ z_{offset}(1 - c_\theta) \end{bmatrix} + R_p \begin{bmatrix} x_{build\ plate} \\ y_{build\ plate} \\ z_{build\ plate} \end{bmatrix}$$

$$R_p = \begin{bmatrix} c_\phi c_\rho c_\theta - s_\rho s_\theta & s_\psi (c_\rho s_\theta + c_\phi c_\theta s_\rho) - c_\psi c_\theta s_\phi & c_\psi (c_\rho s_\theta + c_\phi c_\theta s_\rho) + c_\theta s_\phi s_\psi \\ c_\rho s_\phi & c_\phi c_\psi + s_\phi s_\psi s_\rho & c_\psi s_\phi s_\rho - c_\phi s_\psi \\ -c_\theta s_\rho - c_\phi c_\rho s_\theta & s_\psi (c_\rho c_\theta - c_\phi s_\rho s_\theta) + c_\psi s_\phi s_\theta & c_\psi (c_\rho c_\theta - c_\phi s_\rho s_\theta) - s_\phi s_\psi s_\theta \end{bmatrix}$$

$$P_{build\ plate} = R_p^{-1} \left(S - P_{frame} - \begin{bmatrix} -z_{offset} s_\theta \\ 0 \\ z_{offset} (1 - c_\theta) \end{bmatrix} \right)$$

$$R_p^{-1} = \begin{bmatrix} c_\phi c_\rho c_\theta - s_\rho s_\theta & c_\rho s_\phi & -c_\theta s_\rho - c_\phi c_\rho s_\theta \\ c_\rho s_\psi s_\theta - c_\psi c_\theta s_\phi + c_\phi c_\theta s_\psi s_\rho & c_\phi c_\psi + s_\phi s_\psi s_\rho & c_\rho c_\theta s_\psi + c_\psi s_\phi s_\theta - c_\phi s_\psi s_\rho s_\theta \\ c_\psi c_\rho s_\theta + c_\theta s_\phi s_\psi + c_\phi c_\psi c_\theta s_\rho & c_\psi s_\phi s_\rho - c_\phi s_\psi & c_\psi c_\rho c_\theta - s_\phi s_\psi s_\theta - c_\phi c_\psi s_\rho s_\theta \end{bmatrix}$$

Velocity of the Frame Given Nozzle Velocity

In the same manner as the computations of the position of the frame or nozzle above, the inverse kinematics for the velocity can be computed by rearranging and solving for the frame velocity and nozzle velocity in the build space.

$$0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_{frame} \\ \dot{y}_{frame} \\ \dot{z}_{frame} \end{bmatrix} + \begin{bmatrix} j_1 & j_2 & j_3 & j_4 & j_5 \\ j_6 & j_7 & j_8 & j_9 & j_{10} \\ j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \dot{x}_{build\ plate} \\ \dot{y}_{build\ plate} \\ \dot{z}_{build\ plate} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}$$

$$0 = \begin{bmatrix} \dot{x}_{frame} \\ \dot{y}_{frame} \\ \dot{z}_{frame} \end{bmatrix} + \begin{bmatrix} j_1 & j_2 & j_3 & j_4 & j_5 \\ j_6 & j_7 & j_8 & j_9 & j_{10} \\ j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \dot{x}_{build\ plate} \\ \dot{y}_{build\ plate} \\ \dot{z}_{build\ plate} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}$$

$$0 = V_{frame} + \begin{bmatrix} j_1 & j_2 & j_3 & j_4 & j_5 \\ j_6 & j_7 & j_8 & j_9 & j_{10} \\ j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \dot{x}_{build\ plate} \\ \dot{y}_{build\ plate} \\ \dot{z}_{build\ plate} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}$$

$$V_{frame} = - \begin{bmatrix} j_1 & j_2 & j_3 & j_4 & j_5 \\ j_6 & j_7 & j_8 & j_9 & j_{10} \\ j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \dot{x}_{build\ plate} \\ \dot{y}_{build\ plate} \\ \dot{z}_{build\ plate} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}$$

Velocity of Nozzle above Build Plate

Again the equation is rearranged and solved for the velocity of the nozzle inside the build plate coordinate system.

$$\begin{aligned}
0 &= V_{frame} + \begin{bmatrix} j_1 & j_2 & j_3 \\ j_6 & j_7 & j_8 \\ j_{11} & j_{12} & j_{13} \end{bmatrix} \begin{bmatrix} \dot{x}_{build\ plate} \\ \dot{y}_{build\ plate} \\ \dot{z}_{build\ plate} \end{bmatrix} + \begin{bmatrix} j_4 & j_5 \\ j_9 & j_{10} \\ j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \end{bmatrix} \\
0 &= V_{frame} + \begin{bmatrix} j_1 & j_2 & j_3 \\ j_6 & j_7 & j_8 \\ j_{11} & j_{12} & j_{13} \end{bmatrix} V_{build\ plate} + \begin{bmatrix} j_4 & j_5 \\ j_9 & j_{10} \\ j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \end{bmatrix} \\
V_{build\ plate} &= - \begin{bmatrix} j_1 & j_2 & j_3 \\ j_6 & j_7 & j_8 \\ j_{11} & j_{12} & j_{13} \end{bmatrix}^{-1} \left(V_{frame} + \begin{bmatrix} j_4 & j_5 \\ j_9 & j_{10} \\ j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \end{bmatrix} \right)
\end{aligned}$$

Acceleration of the Frame Given Nozzle Acceleration

Rearrangement of the acceleration equations, solving for the frame acceleration are shown below.

$$0 = \ddot{P} = \dot{J}(q)\dot{q} + J(q)\ddot{q}$$

$$A_{frame} = \begin{bmatrix} \ddot{x}_{frame} \\ \ddot{y}_{frame} \\ \ddot{z}_{frame} \end{bmatrix}$$

$$0 = \ddot{P} = \dot{J}(q)\dot{q} + \begin{bmatrix} 1 & 0 & 0 & j_1 & j_2 & j_3 & j_4 & j_5 \\ 0 & 1 & 0 & j_6 & j_7 & j_8 & j_9 & j_{10} \\ 0 & 0 & 1 & j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \ddot{q}$$

$$0 = \dot{J}(q)\dot{q} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{x}_{frame} \\ \ddot{y}_{frame} \\ \ddot{z}_{frame} \end{bmatrix} + \begin{bmatrix} j_1 & j_2 & j_3 & j_4 & j_5 \\ j_6 & j_7 & j_8 & j_9 & j_{10} \\ j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \ddot{x}_{build\ plate} \\ \ddot{y}_{build\ plate} \\ \ddot{z}_{build\ plate} \\ \ddot{\theta} \\ \ddot{\phi} \end{bmatrix}$$

$$0 = \dot{J}(q)\dot{q} + A_{frame} + \begin{bmatrix} j_1 & j_2 & j_3 & j_4 & j_5 \\ j_6 & j_7 & j_8 & j_9 & j_{10} \\ j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \ddot{x}_{build\ plate} \\ \ddot{y}_{build\ plate} \\ \ddot{z}_{build\ plate} \\ \ddot{\theta} \\ \ddot{\phi} \end{bmatrix}$$

$$A_{frame} = -\dot{J}(q)\dot{q} - \begin{bmatrix} j_1 & j_2 & j_3 & j_4 & j_5 \\ j_6 & j_7 & j_8 & j_9 & j_{10} \\ j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \end{bmatrix} \begin{bmatrix} \ddot{x}_{build\ plate} \\ \ddot{y}_{build\ plate} \\ \ddot{z}_{build\ plate} \\ \ddot{\theta} \\ \ddot{\phi} \end{bmatrix}$$

Acceleration of Nozzle Above Build Plate

Lastly, the acceleration is obtained inside the build plate coordinate system given the acceleration of the frame.

$$\begin{aligned}
0 &= \dot{J}(q)\dot{q} + A_{frame} + \begin{bmatrix} \dot{j}_1 & \dot{j}_2 & \dot{j}_3 \\ \dot{j}_6 & \dot{j}_7 & \dot{j}_8 \\ \dot{j}_{11} & \dot{j}_{12} & \dot{j}_{13} \end{bmatrix} \begin{bmatrix} \ddot{x}_{build\ plate} \\ \ddot{y}_{build\ plate} \\ \ddot{z}_{build\ plate} \end{bmatrix} + \begin{bmatrix} \dot{j}_4 & \dot{j}_5 \\ \dot{j}_9 & \dot{j}_{10} \\ \dot{j}_{14} & \dot{j}_{15} \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} \\
0 &= \dot{J}(q)\dot{q} + A_{frame} + \begin{bmatrix} \dot{j}_1 & \dot{j}_2 & \dot{j}_3 \\ \dot{j}_6 & \dot{j}_7 & \dot{j}_8 \\ \dot{j}_{11} & \dot{j}_{12} & \dot{j}_{13} \end{bmatrix} A_{build\ plate} + \begin{bmatrix} \dot{j}_4 & \dot{j}_5 \\ \dot{j}_9 & \dot{j}_{10} \\ \dot{j}_{14} & \dot{j}_{15} \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} \\
A_{build\ plate} &= - \begin{bmatrix} \dot{j}_1 & \dot{j}_2 & \dot{j}_3 \\ \dot{j}_6 & \dot{j}_7 & \dot{j}_8 \\ \dot{j}_{11} & \dot{j}_{12} & \dot{j}_{13} \end{bmatrix}^{-1} \left(\dot{J}(q)\dot{q} + A_{frame} + \begin{bmatrix} \dot{j}_4 & \dot{j}_5 \\ \dot{j}_9 & \dot{j}_{10} \\ \dot{j}_{14} & \dot{j}_{15} \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} \right)
\end{aligned}$$

3.3.6 Nozzle and Heated Bed

In order to extrude the plastic from the nozzle of the printer, the nozzle must be heated to a temperature such that plastic can flow under the force of the filament being pushed. For polylactic acid (PLA), the material of choice, this temperature is between 180 and 220 degrees Celsius [4].

In effect, this is a transient heat transfer problem and as such any reasonable assumptions will significantly decrease the complexity of solving the problem. One possible assumption is the lumped capacitance method. Simply stated, if the rate of convective heat transfer from the surface of the object is significantly less than the rate of conduction within the object, it is safe to assume that the object abides at one uniform temperature.

The Biot number is a dimensionless parameter in heat transfer that is a ratio of the convective to conductive heat transfer as seen in Equation 3.18, where h is the convective heat transfer coefficient, k is the thermal conductivity of the object, and L_c is the characteristic length. This length is simply the ratio of the volume V to surface area A_s .

$$B = \frac{hL_c}{k} \quad (3.18)$$

$$L_c = \frac{V}{A_s} \quad (3.19)$$

Therefore if Biot number is less than 0.1, lumped capacitance is a valid assumption. As seen in Table 3.1, the condition is satisfied for both the hot end and build plate.

Component	$k \left(\frac{W}{m \cdot K} \right)$	$V \text{ (m}^3\text{)}$	$A_s \text{ (m}^2\text{)}$	$L_c \text{ (m)}$	$h \left(\frac{W}{m^2 \cdot K} \right)$	Bi
Hot End	Aluminum (205)	2.75×10^{-6}	9.38×10^{-4}	0.00294	0.0273	3.916×10^{-7}
Build Plate	Glass (1.05)	1.29×10^{-4}	0.0451	0.00284	0.389	0.00105

Table 3.1: Constants and calculations of Biot number for hot end and build plate

With the lumped capacitance method validated, the heat transfer between the surrounding air and the hot end when no plastic is flowing or build plate can be described as a first order differential

equation during start up, where h is the convection coefficient, A is the surface area, T_∞ is the ambient temperature, T is the desired temperature of the nozzle, u is the input power from a resistive heating element, c is the specific heat of the nozzle, m is the mass of the nozzle, W is the volume of the nozzle, and ρ is the density of the nozzle.

$$mc \frac{dT}{dt} + \sum hA(T - T_\infty) = u \quad (3.20)$$

$$m = \rho W$$

$$u = P = \frac{V^2}{R}$$

$$\rho W c \frac{dT}{dt} + \sum hA(T - T_\infty) = \frac{V^2}{R} \quad (3.21)$$

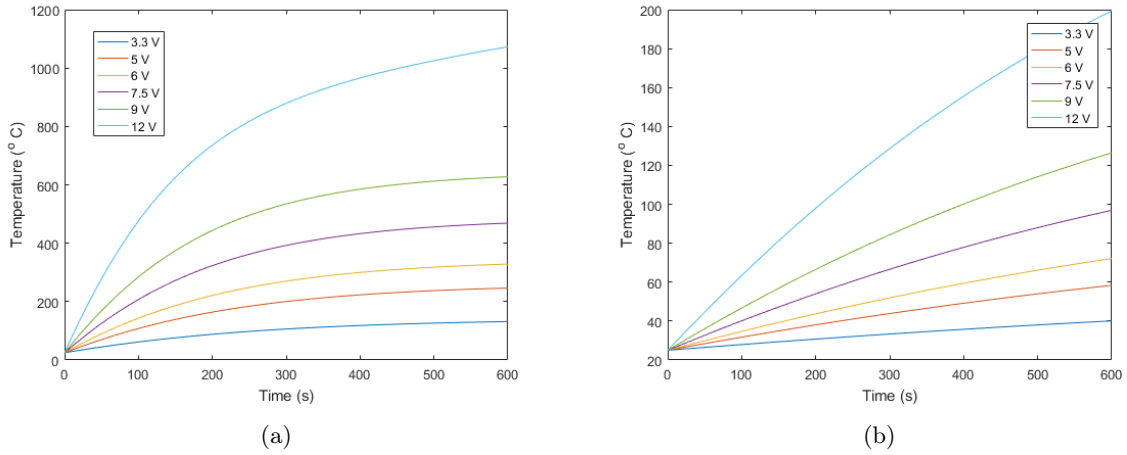


Figure 3.18: Plot of temperature vs time during transient state given voltage for (a) the hot end and (b) the build plate

During printing the heat transfer can be approximated by a steady state assumption, because the nozzle is fixed and perturbations to the temperature and movement of the ambient air is minimal. However, in this case plastic is flowing through the nozzle and thus an extra term is added to account for heat lost to the plastic flow.

$$\sum hA(T - T_\infty) + \dot{m}_{\text{plastic}} C_{\text{plastic}}(T - T_\infty) = \frac{V^2}{R} \quad (3.22)$$

$$A_{\text{nozzle}} = \frac{\pi d_{\text{nozzle}}^2}{4} \quad (3.23)$$

$$\dot{m}_{\text{plastic}} = \rho_{\text{plastic}} A_{\text{nozzle}} v_{\text{nozzle}} \quad (3.24)$$

$$V = \sqrt{R(T - T_{\infty}) \left(\sum hA + \dot{m}_{\text{plastic}} C_{\text{plastic}} \right)} \quad (3.25)$$

When the plastic enters the nozzle it will be at the temperature of the ambient air, T_{∞} and should be heated to the nozzle temperature upon exit, T . In actuality, the heating of the plastic can be considered insignificant compared to heat lost from convection considering the volume of plastic extruded per second is on the order of 10^{-9} . Therefore, this term is dropped and the following equation is resolved.

$$V = \sqrt{R \left(\sum hA \right) (T - T_{\infty})} \quad (3.26)$$

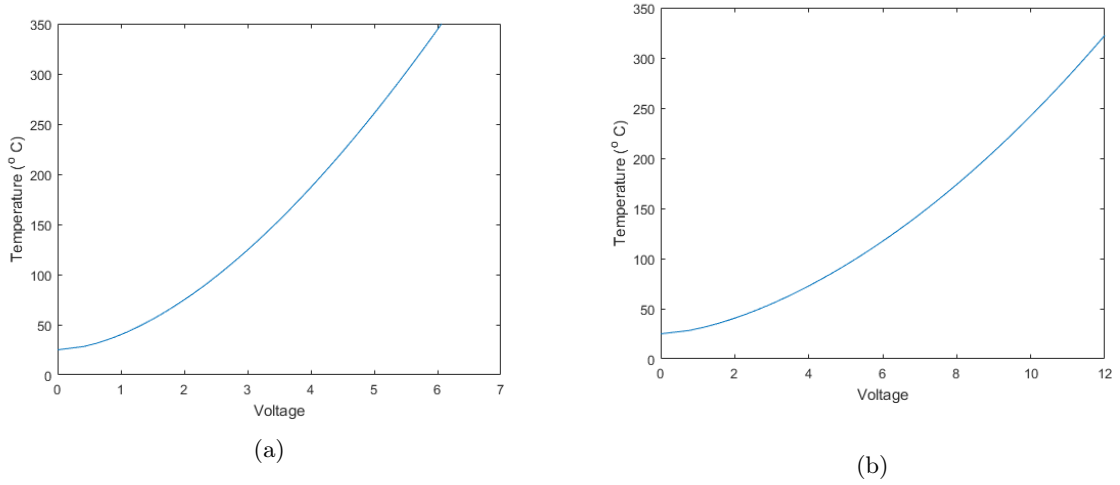


Figure 3.19: Plot of input voltage vs steady state temperature for printer (a) hot end and (b) build plate

In all of the aforementioned equations, the convection coefficient was used to determine the various quantities of interest. This is performed using Nusselt number correlations [1]. In all, three dimensionless parameters are required, the Nusselt, Rayleigh, and Prandtl numbers which can be seen below as Nu , Ra , and Pr respectively. In the following equations, v is the kinematic viscosity of the fluid, g is the acceleration due to gravity, k is the thermal conductivity of the fluid, T_{∞} is the ambient temperature, T is the surface temperature, C_p is the specific heat of the fluid, β is the volumetric expansion coefficient of the fluid and ρ is the fluid density. In the case of the hot end, it is assumed to be a box with the upper surface missing, because the top is covered by the input region of the filament. The build plate, a glass plate atop a heater, is simply a horizontal surface.

$$Ra = \frac{g\beta(T - T_{\infty})L^3Pr}{v^2}$$

$$Pr = \frac{v}{\alpha}$$

$$\alpha = \frac{k}{C_p \rho}$$

$$Nu = \frac{hL}{k}$$

$$h = \frac{kNu}{L}$$

$$L = \frac{V}{A}$$

$$\bar{C}_l = 0.515$$

$$C_t^H = 0.14$$

$$C_t^V = 0.103$$

Nusselt Number Correlations for Heated Vertical Flat Plate with Constant Surface Temperature

$$Nu^T = \bar{C}_l Ra^{1/4} \quad (3.27a)$$

$$Nu_l = \frac{2.8}{\ln(1 + 2.8/Nu^T)} \quad (3.27b)$$

$$Nu_t = C_t^V Ra^{1/3} \quad (3.27c)$$

$$Nu = [(Nu_l)^m + (Nu_t)^m]^{1/m}, \quad m \approx 6 \quad (3.27d)$$

Nusselt Number Correlations for Heated Horizontal Upward-Facing Flat Plate

$$Nu^T = 0.835 \bar{C}_l Ra^{1/4} \quad (3.28a)$$

$$Nu_l = \frac{1.4}{\ln(1 + 1.4/Nu^T)} \quad (3.28b)$$

$$Nu_t = C_t^H Ra^{1/3} \quad (3.28c)$$

$$Nu = [(Nu_l)^m + (Nu_t)^m]^{1/m}, \quad m = 10 \quad (3.28d)$$

Nusselt Number Correlations for Heated Horizontal Downward-Facing Flat Plate

$$Nu \approx Nu_l = \frac{0.527(Ra^{1/5})}{(1 + (1.9/Pr)^{9/10})^{2/9}} \quad (3.29)$$

Temperature Sensing

A negative temperature coefficient (NTC) 3950 Thermistor was used to measure the temperature of the hot end and build plate. As the temperature increases the resistance drops. In turn, the voltage drop across the resistor changes and is measured with a voltage divider connected to the analog input of the microcontroller. A circuit diagram of the voltage divider is shown in Figure 3.20.

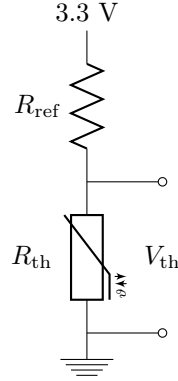


Figure 3.20: Thermistor circuit

The equation, shown in Equation 3.30, for the voltage drop across the thermistor, R_{th} given the value of the reference resistor, R_{ref} , is

$$V_{\text{th}} = \frac{R_{\text{th}}}{R_{\text{th}} + R_{\text{ref}}} V_{\text{in}} \quad (3.30)$$

This equation is then solved for the thermistor's resistance as in Equation 3.31, where the voltage drop across the thermistor is now the input.

$$R_{\text{th}} = \frac{R_{\text{ref}}}{\frac{V_{\text{in}}}{V_{\text{th}}} - 1} \quad (3.31)$$

Lastly, the temperature of the resistor was calculated using the β parameter equation, Equation 3.32, which is essentially an alternative form of the Steinhart-Hart Equation, where R_0 is the resistance of thermistor at a temperature T_0 , and β is the given parameter for the type of thermistor. Note the temperatures are in absolute units, either Kelvin or Rankine. For the particular thermistor used, the β parameter was 3950 K, T_0 was 298 K, and R_0 was 10 K Ω .

$$T = \frac{1}{\frac{1}{T_0} + \frac{1}{\beta} \ln\left(\frac{R_{\text{th}}}{R_0}\right)} \quad (3.32)$$

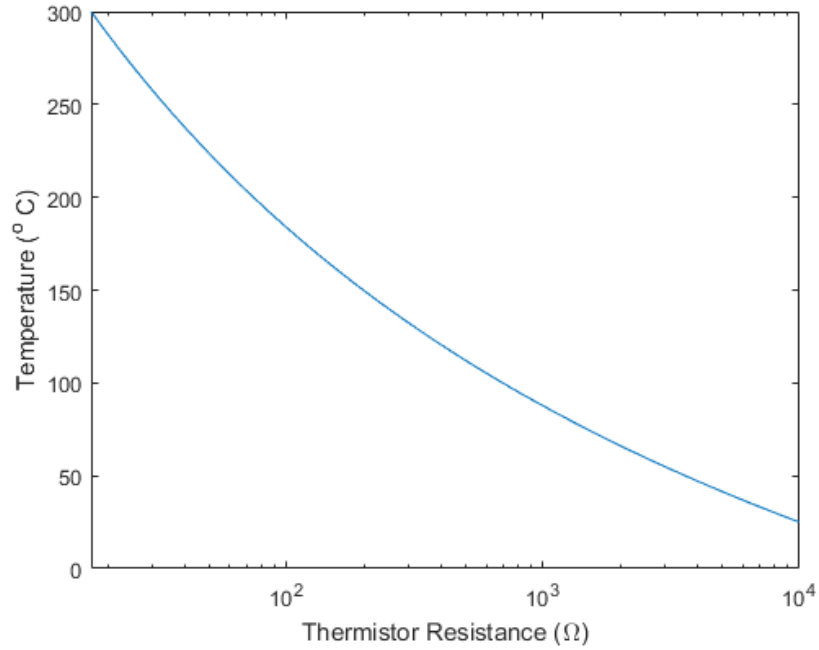


Figure 3.21: Plot of thermistor resistance vs temperature using β parameter equation

The reference resistor determines the precision of the temperature measurement. In this case plus or minus one degree is more than sensitive enough. However, it was possible to increase the temperature measurement accuracy to 0.15°C using the correct choice of resistor, $1500\ \Omega$. Figure 3.22 shows the sensitivity in ADC counts per 0.15°C given the value of the reference resistor at 40 and 300 degrees Celsius, the bounds of sensing requirements. Any resistor between the intersection of the blue and red curves with the yellow line is suitable. In particular, the sensitivity of a $1500\ \Omega$ reference resistor is plotted in Figure 3.23 between 0 and 300 degrees Celsius. In both cases the analog-to-digital converter (ADC) has a resolution of 13 bits.

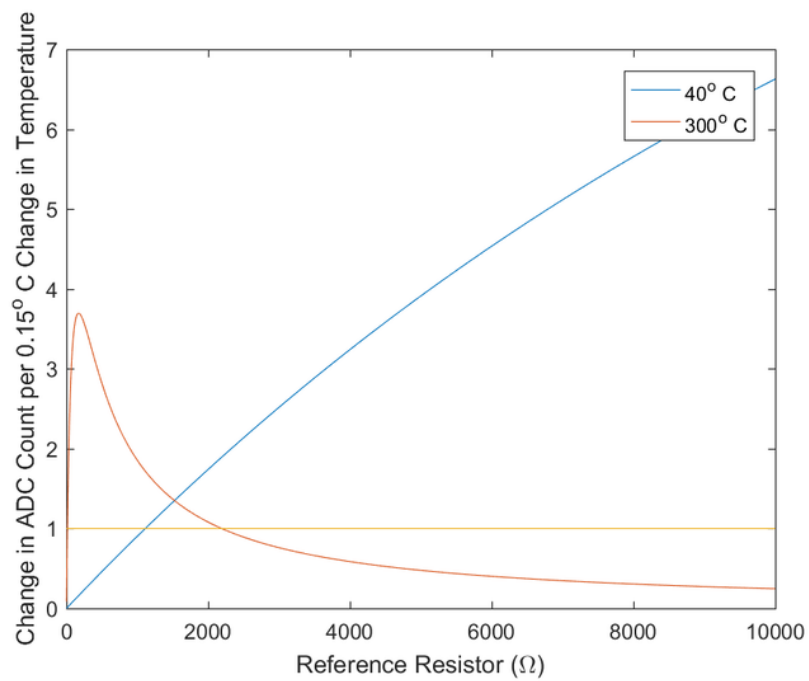


Figure 3.22: Sensitivity of resistor as a function of the reference resistor

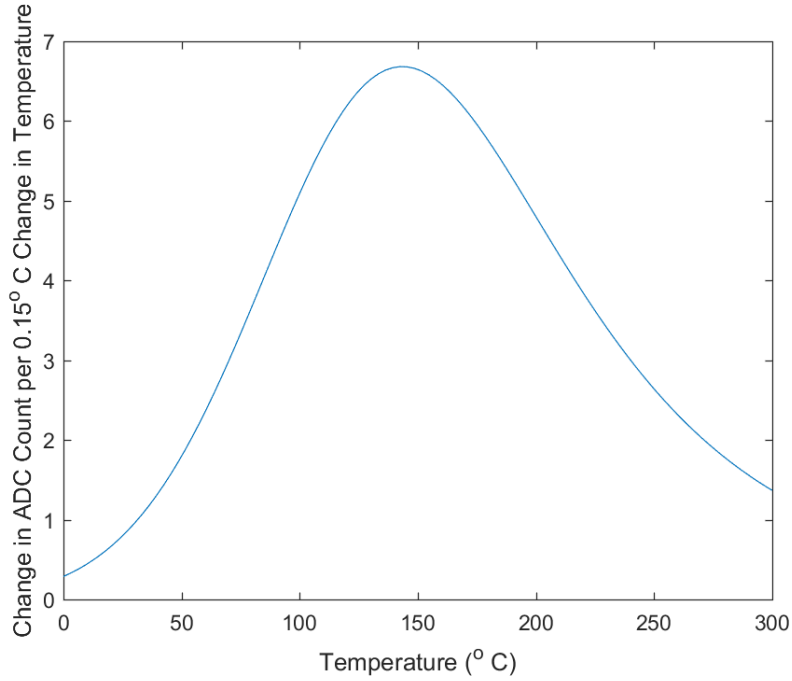


Figure 3.23: Sensitivity as a function of the temperature for 1500 Ω reference resistor

Thermal Control

The current is much too large for the heating elements to be driven directly by the microcontroller. Therefore a transistor is required to control flow of current given an input signal from the microcontroller. Furthermore, it is possible to regulate the average amount of power flowing through the heating element using a pulse width modulation (PWM) signal to the gate of the transistor. Given a sufficiently high frequency the average voltage can be determined by Equation 3.33, where V is the supply voltage and D is the duty cycle. In this particular application a N-channel logic level enhancement mode metal-oxide-semiconductor field-effect transistor (MOSFET), PHP79NQ08LT, was used as seen in Figure 3.24.

$$V_{AVG} = DV \quad (3.33)$$

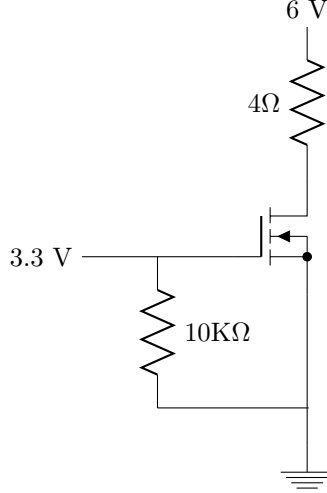


Figure 3.24: Temperature control circuit

3.4 Embedded Design

3.4.1 Control Loop

The max resolution desired was 10 microns. Thus this was the shortest path the printer was intended to travel. If the rotary stages were assumed to be at fixed angles and all three of the linear stages were actuated at their max possible speeds then it was possible to calculate the time interval required between control loops. Note that assuming the rotary stages were fixed simplifies calculations and was reasonable since the stages needed to travel slower while the rotary stages were moving to obtain the same linear speed due to the radius.

The motors max RPM at load was 23.75 rps and the stepper motors had a max resolution of 1600 steps/revolution. Thus the number of steps per second followed as $1600 \frac{\text{steps}}{\text{revolution}} \times 23.75 \frac{\text{revolutions}}{\text{second}} = 38000 \frac{\text{steps}}{\text{second}}$. If all of the stages moved at their max speed, $v_{max} = v_x = v_y = v_z$, which was identical then the magnitude of the velocity vector followed as $v = \sqrt{v_x^2 + v_y^2 + v_z^2} = \sqrt{3v_{max}^2} = \sqrt{3}v_{max}$.

Therefore, it followed that the time, t , to travel 10 microns at the max velocity was given by $t = \frac{10\mu\text{m}}{\sqrt{3}v_{max}}$. The max velocity was equal to the pitch of the leadscrew, 0.1 in or 2.54 mm, and the max RPS of the motor, 23.75 RPS. Thus $v_{max} = 2.54 \frac{\text{mm}}{\text{revolution}} \times 23.75 \frac{\text{revolution}}{\text{s}} = 60.325 \frac{\text{mm}}{\text{s}}$. From the previous equation for the time interval, it can be shown as $t = \frac{10\mu\text{m}}{60.325 \frac{\text{mm}}{\text{s}} \sqrt{3}} \approx 95.71\mu\text{s}$. This represents the period between control loops, thus the required minimum frequency is $f = \frac{1}{T} = \frac{1}{95.71\mu\text{s}} = 10.449 \text{ kHz}$. The actual frequency selected for the control loop was 20 kHz, as modern microcontroller speeds could easily exceed 10.449 kHz and guaranteed a 10 micron resolution.

3.4.2 Closed-Loop Feedback

In order to receive positional feedback from the stepper motors to adjust for error, motors were selected which came with precision rotary encoders. The motors were Anaheim Automation's

23MD106D model, which came with an integrated driver and a 1000-count rotary encoder. Using full quadrature encoding, this resulted in 4000 steps per rotation, or a 0.09 degree precision.

In order to read from up to five rotary encoders at once, we chose to use LS7166 encoder counter ICs. This freed the microcontroller from having to run an interrupt service routine to handle encoder counting which, using full quadrature encoding, could occur very frequently for five steppers. These ICs use an 8-bit parallel port for communication, and using the timing characteristics provided in the datasheet, it was possible to estimate the minimum amount of time required to retrieve a count value from an IC.

For a read, the LS7166 requires 110 *ns* between activation of the read pin and reading 8-bits of valid data. Additionally, it needed 30 *ns* between deactivation of the read pin and return to normal tri-state on the data-bus. Each single-bit pin read takes one instruction, which at 180 *MHz* takes 5.56 *ns*. Since this was a 24-bit encoder, this sequence had to occur three times. We added the individual components in order to get a final read-time value: $3(110\text{ ns} + 8(5.56\text{ ns}) + 30\text{ ns}) = 553.44\text{ ns}$. We were reading up to five encoders at once, which meant a full read of all stage positions took 2.767 μs .

3.4.3 Stepper Operation

In order to operate the stepper motors, the integrated drivers required a clock signal to time when to perform a tick on the motor shaft. The driver also included pins which enabled or disabled the motor, selected the stepping (with options for full, half, quarter, or eighth stepping), and selected either clockwise or counter-clockwise rotation.

In order to achieve our desired minimum resolution of 10 microns, we opted to permanently set the motors at eighth stepping. This gave 1600 steps per revolution, or 0.225 degrees per tick. The selected lead screws travelled 2.54mm per rotation, which meant that at eighth stepping our motors achieved $1.5875 \frac{\text{microns}}{\text{revolution}}$.

To time the motors, we initially opted to perform bit-banging using a timer interrupt on the microcontroller. The stepper control loop ran at 100 *kHz*, separately from the master control loop, and the counters it used to determine when to pull the clock line high or low were updated constantly in the background at a much faster rate. Although excellent performance was achieved when the microcontroller was only operating motors, this timer approach failed when incorporating parsing G-code from a file. The file reads took too long, delaying the execution of the control loop and therefore the updating of the stepper motor counters.

To overcome this issue, the timer approach was replaced by a hardware solution. The Analog Devices AD9833 chip generated square waves from 0 *Hz* to 12.5 *MHz* which was more than enough to cover our stepper's maximum range of 0 *Hz* to 40 *kHz*, obtained by multiplying 1600 steps per revolution by the 25 *RPS* maximum velocity.

The chip communicated using Serial Peripheral Interface (SPI) at maximum speeds of 40MHz. In order to change the frequency of the output, 64 bits of data were written to the SPI bus in total. Additionally, it took eight clock cycles at 25 *MHz* before the frequency changed to the new value. All in all, this meant the response time between sending a speed command and the new speed being reflected in the stepper was $8(40\text{ ns}) + 64(25\text{ ns}) = 1.92\text{ }\mu\text{s}$. For five steppers, this meant updating all five speeds took 9.6 μs .

Altogether, a complete cycle of reading all five stepper positions and setting all five stepper speeds took 12.367 μs . Considering that our control loop ran at 20 *kHz*, which meant one run every 50 μs , the time cost it took to use external solutions for timing and counting the steppers

rather than doing it onboard was well within the bounds of operation.

3.4.4 Microcontroller Selection

In order to successfully deposit an object onto the build plate, the printer's microcontroller needed to be able to fulfill several high-level requirements. These requirements were for it to be capable of:

- Operate six stepper motors (one for each stage and one for extrusion).
- Receive closed-loop positional feedback from the stages.
- Read and parse G-code from an SD card.
- Compute inverse kinematics from the given G-code.
- Extrude filament at a constant rate.
- Modulate a PWM signal for temperature control of the nozzle and print bed.

The control loop of the printer operated at 20 *kHz*, but also needed to be capable of buffering and processing inverse kinematics live from inputted G-code. After reviewing several options currently available, including the Arduino Mega, Intel Galileo, Raspberry Pi, and Beaglebone Black, the Teensy 3.6 was selected to operate the printer. The Teensy had the following features which made it advantageous:

- Used a 180 *MHz* Cortex M4 ARM processor which more than fulfilled the processing power requirements.
- Was designed to be integrated into a breadboard or protoboard, which made development easier.
- Had a built in SD-card module with high-speed SDIO support for parsing G-code.
- Was compatible with the Arduino library which streamlined development.
- Had 39 digital I/O pins to operate the steppers and receive feedback.
- Had hardware support for multiple USART and SPI ports, which allowed for easy USB debugging while operating the printer and could max out the 40MHz communication limit of the frequency generators.
- Had an active support and knowledge base.

One of the disadvantages of the Teensy was its logic levels. All Teensy digital I/O pins wrote or read at +3.3V TTL, and were not +5V tolerant. On the other hand, the stepper motor drivers, encoders, and LS7166 encoder counter chips all operated at +5V TTL. The frequency generators operated at either +3.3V or +5V, and outputted a wave anywhere in that range as well. In order to accommodate these differences, several bi-directional level shifters were purchased in order to translate TTL levels from +3.3V to +5V and back again.

3.4.5 Power

In order to power the printer, four different power sources were used. The first was the +3.3V power from the Teensy, which went directly to the level shifters responsible for converting the Teensy logic levels to +5V logic levels. The Teensy itself was powered via USB.

The second source of power was +5V from a standard lab power supply. This supply powered the +5V end of the level shifters, the rotary encoders, the encoder counter chips, and the frequency generators. Together, the +3.3V and +5V power sources drew on the order of mA in order to power the various logic circuits.

The third source of power was the 24.0V, 13.0A power source dedicated to powering the stepper motors. Each motor consumed at most 2.0A and therefore at full stall all five motors consumed a total of 10.0A.

The fourth and final source of power was a 12.0V, 21.0A power supply. This supply was exclusively responsible for powering the heating elements in the nozzle and build plate. The nozzle consumed at most 1.5A of power and the build plate 10.0A, creating a total maximum consumption of 11.5A.

Figure 3.25 shows how the various power levels were distributed to the printer systems.

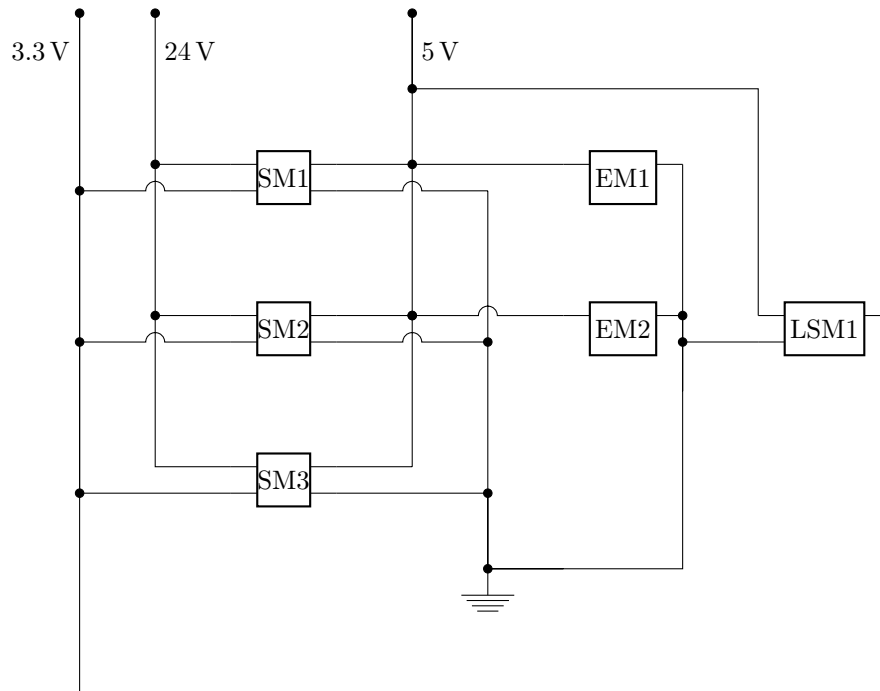


Figure 3.25: Distribution of power systems. SM are stepper modules, EM are encoder modules, and LSM is the level shift module.

3.4.6 Circuitry & Wiring

Several steps were taken to minimize wiring complexity and streamline connecting and disconnecting the actual printer hardware from the embedded logic. The modifications made to the various electrical elements were meant to fulfill the following requirements:

1. Given a severe printer malfunction where the connection between a motor and the embedded logic is strained, the connection broke away before damaging any equipment.
2. All I/O lines between a specific piece of printer hardware and the embedded logic were easily removable and replaceable without rewiring individual pins.
3. Electrical components were assembled in individual modules for easy future replacement in case of failure or upgrade.

In order to achieve items 1 and 2, DB15 cables were used to package all digital I/O coming from the printer hardware. Each stepper motor used the following pins:

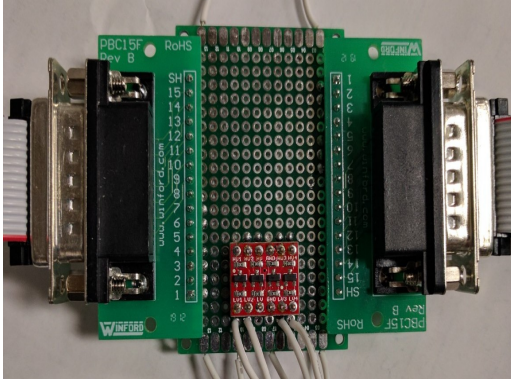
- Seven driver pins
- Five encoder pins
- Two limit switch pins

As such, each motor required fourteen pins for full operation. A DB15 screw terminal was sticky-taped to the body of the each stepper motor, with each of the fourteen leads going into one of the screw terminals. A DB15 ribbon cable was then plugged in to the terminal, with the other plugged into a printer stepper module. In this way, the DB15 cables were free to disconnect from either the motors or stepper modules without damaging the fragile pins on either.

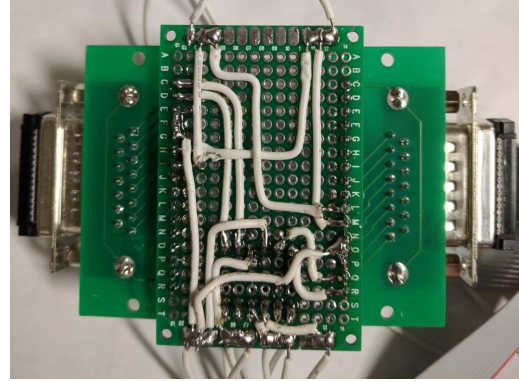
The printer stepper modules were made in order to interface two steppers with the microcontroller in an easy to use, replaceable module. Each stepper module consisted of a protoboard with: two DB15 terminals with protoboard-compatible pins, a level-shifter with four conversion points, +24V power, +3.3V power, +5V power, and wires for connecting the clock, direction, and enable pins of both steppers, and the A and B channels of both encoders. The clock cable went directly to the output of a frequency generator running on +5V. The enable and direction pins were plugged into the level shifter +5V pins. The A and B pins were plugged into the A and B inputs on the LS7166 encoder counters. Three of these modules were built and mounted on a board to wire the printer. An example module is shown in Figure 3.26.

Accompanying the stepper modules were two additional stand-alone modules: the encoder module and the level-shifter module. The encoder module was composed of three LS7166 encoder counting chips on a single protoboard, with their common pins tied together. These pins included all communications pins:

- RD pin
- WR pin
- CD pin
- Parallel pins 0-7



(a) Top view of the stepper module

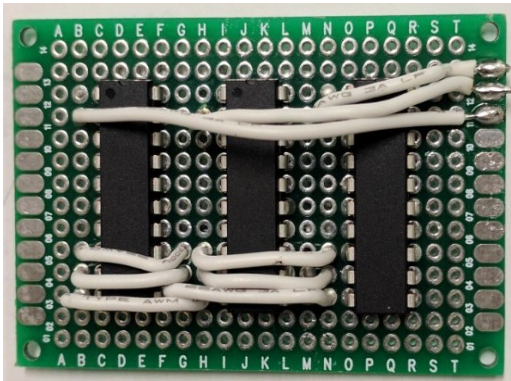


(b) Bottom view of the stepper module

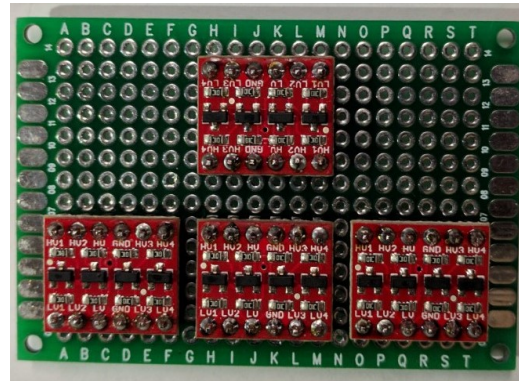
Figure 3.26: A stepper module used to convert control signals from the microcontroller to two separate steppers.

The only pins which required their own dedicated line to the microcontroller were the chip select pins, required to select which encoder chip would receive parallel communications from the controller.

The common pins and chip select pins were then wired to the level-shifter module. The level shifter module was composed of four level-shifters, each with four conversion points. This provided a total of 16 conversion points, which exactly matched the sixteen pins required to operate all five encoder counting modules: 11 communications pins and five chip select pins. In turn, the +3.3V side of the level shifter conversion points were wired to various digital I/O pins on the controller. The encoder and level shifter modules are shown in Figure 3.27.



(a) Top view of the encoder module



(b) Top view of the level shifter module

Figure 3.27: The stepper motor rotary encoder channels are wired to each encoder counter chip which communicates to the microcontroller through the level shifter module.

3.4.7 Path Planning

Path planning was by far the most algorithmically intensive portion of the embedded development. The path planning algorithm needed to perform the following steps, all in time for the printer control loop running at 20 kHz to pick up velocity instructions without interruption:

1. Parse and buffer G-code commands from a G-code file stored on an SD card.
2. From the commands, splice together the travel points into a smooth travel path using splines.
3. Run inverse kinematics on the desired feed rates to compute individual stage velocities.
4. Buffer the velocity instructions so that the control loop can pick them up and process them as the printer deposits material.

To accomplish this, the path smoother utilized a double-buffer system along with a 15-point look-ahead function to safely plan a path for the printer's build plate. The double-buffer systems served to initially buffer characters from a file and parse them into unprocessed G-code commands, and then buffer the parsed G-code into velocity commands for the control loop to send to the steppers. The control loops pulled from the velocity command buffer, while both buffers were updated in the background between control loop cycles. The look-ahead function was used to ensure that the nozzle's target feed rate was capable of safely traversing through future path segments.

The path smoothing was divided into x primary segments:

1. Corner smoothing
2. Segment smoothing
3. Feed-rate protection

Corner Smoothing

The printer instructions were specified in terms of straight line segments in G-code. Unfortunately, in their raw state the printer would be unable to follow the path without completely decelerating to a stop at every point. This significantly increases the time to traverse the path. An alternative is to blend individual segments together with a spline which was constrained by a maximum chord error to ensure that the path was followed to within an allowed tolerance as shown in Figure 3.28 as e_{max} .

Additionally, constraints were required to simplify the spline processing into something manageable as a dynamic computation on the microcontroller. It happened that two constraints significantly reduced the complexity of computing the spline. First, the magnitude of the entry and exit velocities to the spline were restricted to be equal. Second, the start and end points of the spline were restricted to be equidistant along their respective line segments from the shared point, p_m .

Inside the G-code the slicer specified a desired speed, F , which became the upper bound of the printers movement. Though it tried to traverse the path at this speed limitations imposed by the ends of path, tight corners, and chord error may have required the printer to slow down.

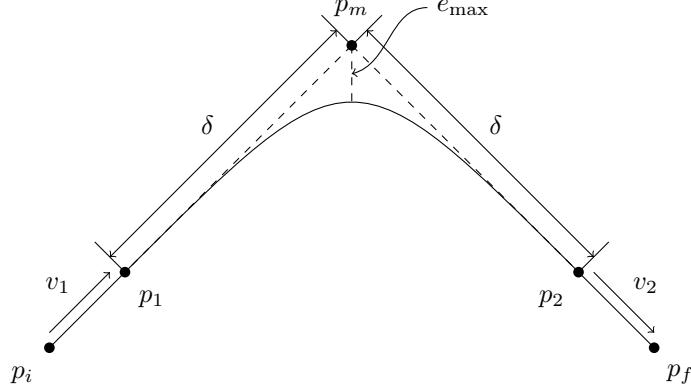


Figure 3.28: Diagram of corner smoothing using spline

A benefit of these constraints was that the equations of motion for the linear stages, x , y , and z , were reduced to second order polynomials, the coefficients of which represented stage acceleration, initial velocity, and initial position. Boundary conditions were required to determine the actual acceleration coefficient. It was assumed that the time required to traverse the spline was equal to the time to traverse the straight line distances between p_1 , p_m , and p_2 . Thus the time at the exit of the spline was $t_f = \frac{2\delta}{F}$ where δ was the offset to the entry and exit of the spline from p_m and F was the feed rate. Note that both F and δ were scalar and the coefficient a for each of the stages was computed using Equation 3.34.

$$a = \frac{v_2 - v_1}{t_f} \quad (3.34)$$

$$x_2 = \frac{1}{2}a_x t^2 + v_{x_1} t + x_1 \quad (3.35a)$$

$$y_2 = \frac{1}{2}a_y t^2 + v_{y_1} t + y_1 \quad (3.35b)$$

$$z_2 = \frac{1}{2}a_z t^2 + v_{z_1} t + z_1 \quad (3.35c)$$

$$e_{max} = \sqrt{(x_m - x_e)^2 + (y_m - x_e)^2 + (z_m - x_e)^2} \quad (3.36)$$

Under theses constraints the max error, e_{max} , always occurred at half the time required to traverse the spline or $t = \frac{\delta}{F}$. Then using a simple distance formula for the point p_m and p_e the max error position at the aforementioned time, the max error was computed. If this error was ever exceeded, then the max velocity was reduced across the buffer to allow the spline to be traversed without violating the error constraint. This smoothing operation was performed on the buffer which produced a path now conjoined at the point by splines as in Figure 3.29.

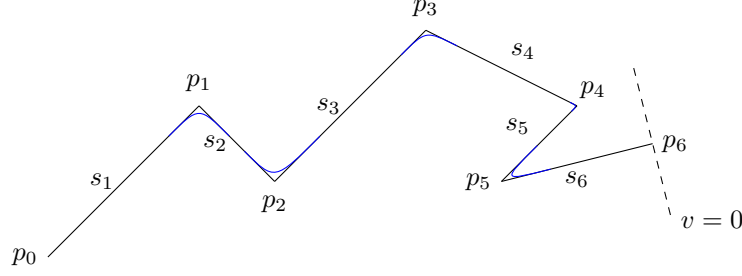


Figure 3.29: Example path with smoothing of corners by second order parametric Spline

Segment smoothing

Although a large portion of the path smoothing involved connecting two separate path segments, a significant amount of optimization was performed on the straight-line segments which connected spline entry and exit points. These optimizations allowed the printer to more quickly deposit a layer, as it aimed to always reach a maximum feed rate when connecting two points with a straight line. This meant the build plate accelerated to a specified maximum velocity and would then decelerate to the target velocity at the target destination.

In order to achieve this, an S-curve was computed and discretized to generate timed velocity instructions for the steppers. The goal of the S-curve was to allow the printer to determine, given a distance to travel in a straight line, how to travel that distance as fast as possible, constrained by the entry and exit velocities of the straight-line segment.

To begin, the printer was given a rate at which each stepper accelerated at. Using a set acceleration and the entry and exit velocities, the maximum velocity which could be accelerated to while still reaching the final velocity and travelling the required distance was computed. The equation to compute this maximum velocity was derived and is shown below. d_1 and d_2 represent the distance travelled when accelerating to maximum velocity from the entry velocity, and the distance travelled when decelerating to target velocity from the maximum velocity. These were added to get l , the total distance travelled. The variables were rearranged to isolate v_m .

$$d_1 = \frac{v_m^2 - v_i^2}{2a_1} \quad (3.37a)$$

$$d_2 = \frac{v_f^2 - v_m^2}{2a_2} \quad (3.37b)$$

$$a_2 = -a_1 \quad (3.37c)$$

$$l = d_1 + d_2 \quad (3.37d)$$

$$l = \frac{v_m^2 - v_i^2}{2a_1} + \frac{v_f^2 - v_m^2}{2a_2} \quad (3.37e)$$

$$l = \frac{v_m^2 - v_i^2}{2a_1} + \frac{-v_f^2 + v_m^2}{2a_1} \quad (3.37f)$$

$$l = \frac{1}{2a_1}(2v_m^2 - v_i^2 - v_f^2) \quad (3.37g)$$

$$v_m^2 = \frac{2a_1l + v_i^2 + v_f^2}{2} \quad (3.37h)$$

$$v_m = \sqrt{a_1l + \frac{v_i^2}{2} + \frac{v_f^2}{2}} \quad (3.37i)$$

This value was then capped by the computed maximum velocity of the steppers. The maximum velocity of each stepper was set by the manufacturer's data-sheet at 25 *RPS*; however, only the linear stages were considered for this, as the printer was not expected to both rotate and translate at the same time. The resulting magnitude of the maximum feed rate vector was computed to be 60.325 *mm/s*. This meant that the printer would max out the angular velocity of any one stepper motor to 25*RPS* and adjust any others down accordingly.

Using this maximum feed rate, every possible S-curve case was identified and handled in code to optimize straight-line paths. In total, there were ten specific cases. However, of these ten, there were only four unique cases, and the rest were variations of these cases. Each case is shown in Figure 3.30.

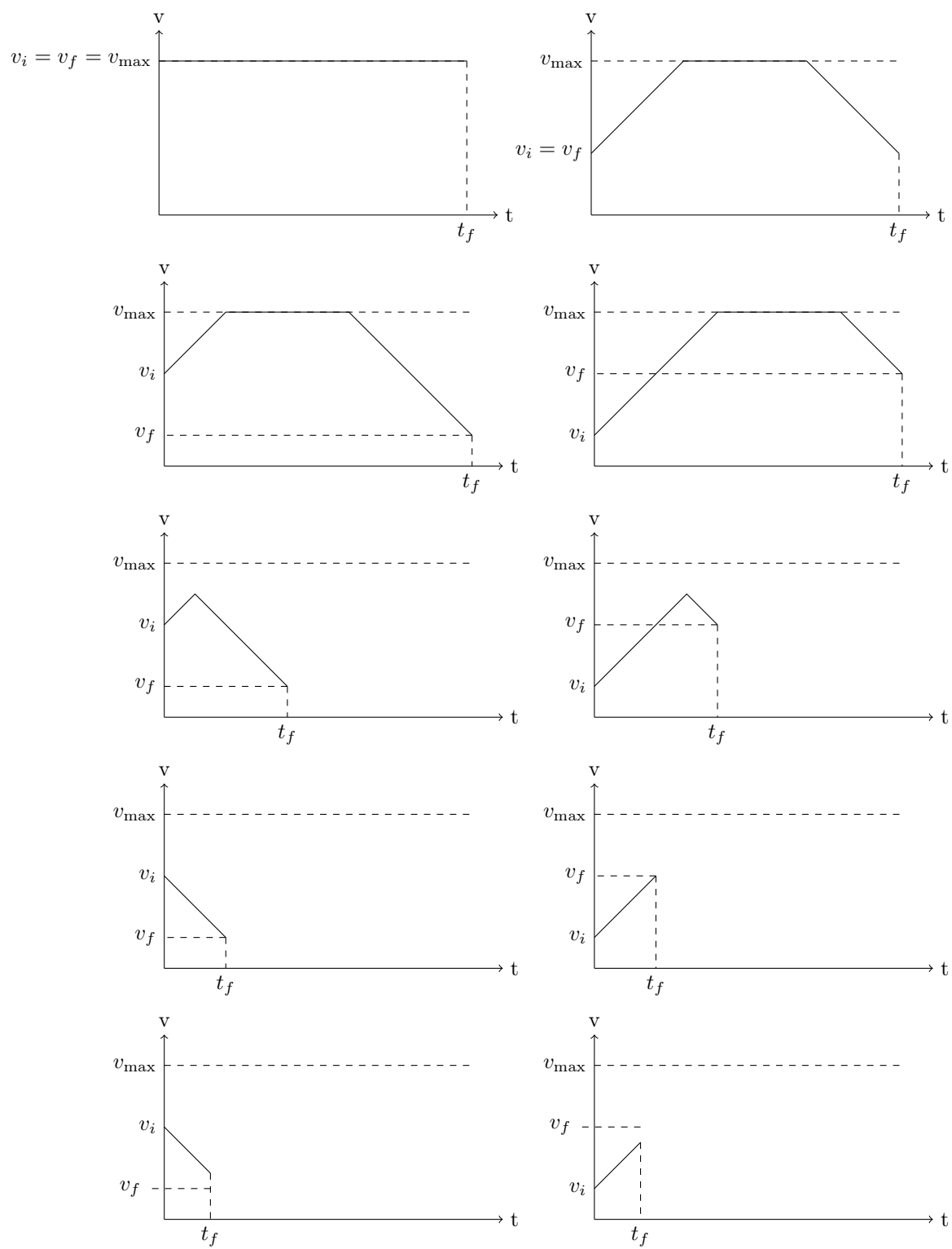


Figure 3.30: All possible S-curve cases

The S-curves defined three separate velocities to bound the velocity commands: v_i , v_f , v_{max} , and t_f . v_i represents the magnitude of the velocity at the beginning of the straight-line segment, which was the exit point of the previous spline. v_f represents the magnitude of the velocity at which the nozzle should be when reaching the end of the straight-line segment, which was the entry point of the next spline. v_{max} is the magnitude of the maximum feed rate, specified above. Finally, t_f is the time it took to reach the end of the line segment.

The first four cases starting from the top represented when the feed rate could actually reach v_{max} . The first was a special case where all velocities were equal, which was the simplest case to handle. The rest were variations of accelerating up to v_{max} and then back down to the target velocity.

The last six cases all occurred when the nozzle did not have the time to accelerate to v_{max} . The last two in specific were error cases, where the maximum feed rate was not properly computed and led to a spline failure. This ended the path smoothing.

Once the entry, maximum, and exit velocities were set, the path smoother discretized the S-curve into velocity instructions. Each individual portion of the S-curve was translated to a velocity instruction with a given acceleration for a set time.

Feed rate protection

In order to avoid running into the bottom two cases of the S-curve graphs, a 15-point look-ahead function was used to provide feed rate protection. This look-ahead computed the maximum feed rate possible for each stage when traversing a corner, and the entry feed rate possible at the last segment assuming that the final point must reach a velocity of 0 m/s . This ensured that:

1. All corners were traversed.
2. If the last point required the printer to stop, it was able to.

To compute the maximum feed rate of each stage in a corner, the following equation was used:

$$F = |a * \delta * d_1 * d_2 * \frac{1}{-s_1 * d_2 - d_1 * s_2}| \quad (3.38)$$

Here, a is the maximum acceleration of each stage. δ is half the length of the shortest path segment. d_1 and d_2 are the lengths of each segment. Finally, s_1 and s_2 are the vectors representing each line segment.

This equation gave the maximum feed rate in a corner for each stage. This was run on every corner in the 15-point buffer and the lowest component velocity was chosen, limiting all stage velocities to that minimum. This feed rate was dynamically computed as the printer continued processing G-code points.

Chapter 4

Results & Discussion

The goal of the project was to use 5-axis 3D printer hardware to reduce the time and wasted material of a 3D print, by reducing the use of support structures. To measure success, various other goals and success metrics were established at the beginning of the project for software development, mechanical engineering, and embedded design. This section evaluates accomplishments and setbacks in each area.

4.1 Software Development

To determine success in the software development portion of this project, we set one fundamental goal at the beginning of the project, to develop a complete software pipeline which generates 5-axis printer instructions from a 3D model in the form of an STL file. This pipeline can be broken down into five steps, which are described in Section 3.2.1. These steps are:

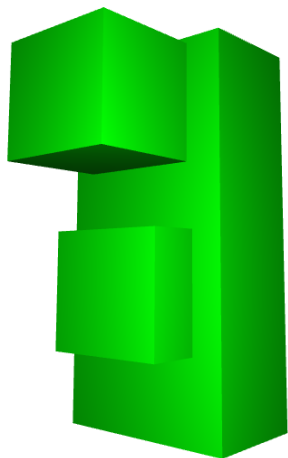
- Volume decomposition
- Build orientation
- Sub-volume sequencing
- Collision detection
- 5-axis G-code generation

Many of these steps were fully implemented into a final successful pipeline from STL file to 5-axis G-code. Time constraints meant that optimizations to the algorithms and their implementations were sometimes foregone, and in few cases that some features were minimally implemented to complete the pipeline.

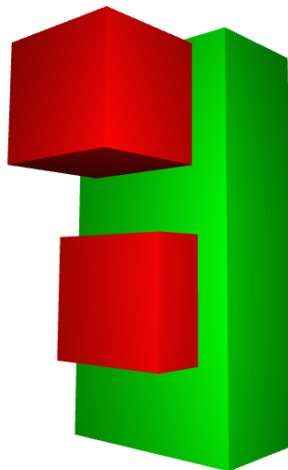
4.1.1 Volume Decomposition

Implementation of volume decomposition required the modification of mesh representations of 3D shapes, such that faces on overhang volumes and the base volumes are disjoint. Once the mesh has been modified to this extent, it must then be broken into separate meshes. The final

implementation of volume decomposition deterministically decomposed some test 3D models into one mesh representing the base volume, and multiple meshes representing overhanging volumes. An example of a successfully decomposed test model is shown in Figure 4.1. For all successful mesh modifications of 3D volumes with overhangs, the mesh was successfully broken into correct mesh representations of the object features. The output of each successful volume decomposition was verified by analyzing each 3D volume. There were no cases in which the volume decomposition implementation identified and output more than one 3D shape when there were no overhanging features on the original 3D shape.



(a) Example 3D Model



(b) Decomposed Example 3D Model

Figure 4.1: (a) shows an example 3D model to test our software one. (b) shows the example 3D model after volume decomposition, with identified overhangs in red.

4.1.2 Build Orientation

Build orientation was successfully implemented such that, with all tested 3D shapes, at least one valid build orientation were found when at least one existed, and no valid build orientation were found when it was not possible to print a 3D shape in 3-axes. These tests were validated with extensive testing on a library of diverse STL files. Upon determining the heuristic of each valid build orientations, the build orientation algorithm performs a hill-climbing search to find the lowest non-zero heuristic. Due to the nature of hill-climbing, this is prone to being caught in a local minimum, in which the hill-climbing search will find a local minimum instead of the global minimum, as described in Russel and Norvig's "Artificial Intelligence A Modern Approach" [8]. This issue with hill-climbing was overcome using simulated annealing [8]. Extensive testing was performed, analyzing by sight the orientations determined by the combined hill-climbing and simulated annealing methods, to determine that this method performs reliably.

4.1.3 Sub-volume Sequencing

Sub-volume sequencing was implemented to the extent that the sequence determined guaranteed that all overhanging features were printed after the base on which they must be printed. This was determined using the sequence graph described in Section 3.2.1. Many advanced sequence algorithms remain to be designed and implemented, to optimize the printing of 3D shapes which would normally result in collisions between the nozzle and previously deposited material. These were not implemented due to time constraints during the project.

4.1.4 Collision Detection

Collision detection is intended to prevent the nozzle colliding with previously deposited material after the base plate is rotated. The algorithm described in 3.2.4 was not implemented due to time constraints. Instead we only used test models that we knew would not cause a nozzle collision.

4.1.5 5-axis G-code generation

To generate 5-axis G-code to be used on the 5-axis printer hardware, 3-axis G-code was combined into a single G-code file, separated by rotational movements to reorient the object such that next sub-volume can be built at its determined build orientation. A G-code parser was implemented to parse 3-axis G-code files generated by Cura Engine for each sub-volume. This parser generates a new G-code file with the 5-axis G-code, the final output of the software.

4.2 Printer

The printer was easily manufactured in part due to its readily available materials and simple design. However, the price of machine shop contracted parts was considerably higher than expected and delayed the overall construction of the printer because of the addition of time required to machine parts that could not be contracted due to cost. In addition, the parts that were water jet cut were found to be significantly cheaper. As a result, future redesigns and iterations of the printer will likely be comprised of parts capable of being water jet cut with minimal post machining requirements. This further aided the reduction of weight, because ordinarily time consuming milling steps for weight reducing pockets did not add to the total cost or manufacturing time for water jet cut parts. All of the linear stages performed exactly as intended. Each stage's accuracy was confirmed using a ten thousandth dial indicator, however error in actuation of the stages away and back was nearly zero as expected because the resolution of the dial indicator was approximately 12 microns while the stages had a resolution of 2 microns. Higher precision measurement tools were prohibitively expensive to be purchased on the budget of the project.

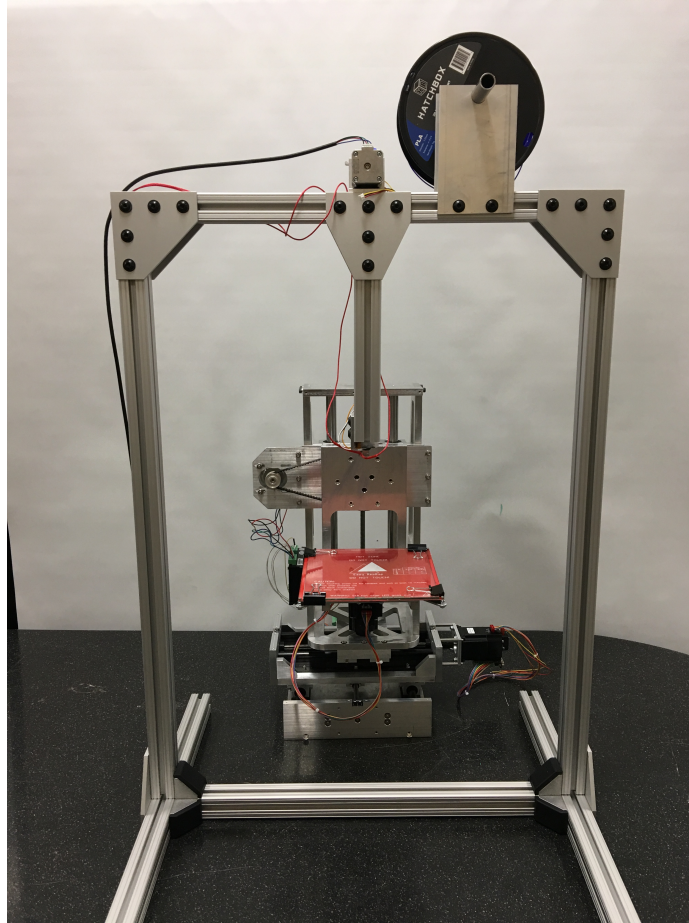


Figure 4.2: Image of fully assembled printer

Unfortunately, the rotary stages from the initial design were not able to be actuated by the 175 oz-in stepper motors due to unexpected friction addition from the slew rings and additional weight from the steel timing belt pulleys on both stages. However, the replacement of the motor with a 425 oz-in stepper motor solved the problem and both rotary stages performed as expected. Additionally, slop in the slew rings contributed to a slight downward angle. It appears that the plane bearing's polyacetal center is not stiff enough to withstand the moment. The issue would not prevent printing but is a consideration in possible sources of error. Fortunately, this possible error can be mitigated by the printer calibration, however, a future improvement would be two fold: reduce the weight of the B stage as well as replace the slew ring with a heavier duty full metal one.

4.3 Embedded Design

4.3.1 Circuitry & Hardware

Although the modular design of the circuitry and the consolidation of stepper cabling did aid in improving the electrical design of the printer, several issues were encountered when running the electronics as described. These issues ranged from less critical ones involving cable management, to problems affecting proper operation of the printer. The components affected were as follows:

1. Steppers
2. DB15 terminals
3. Frequency generators

Steppers

The first issue arose when first assembling the rotary stages. During mechanical assembly of the printer, no aluminum belt-drive gears of the appropriate size were found, and those were therefore abandoned in favor of steel gears. The additional weight of these gears meant the A-stage stepper motor was no longer capable of producing enough torque to fully rotate the build plate 90 degrees.

Several potential solutions to this problem were explored. One was to re-machine the vertical component of the build plate to include a counter-weight that would ease the torque requirements on the stepper. The second option was to re-adjust stepper motor selection to more closely match torque requirements. In this case, this meant replacing the A-stage stepper motor with a larger stepper in the same NEMA category, and replacing the B-stage stepper with a smaller, lighter option as it only needed to rotate the relatively light glass build plate, easing the torque requirements on the A-stage. The second option was selected and this proved adequate to properly operate both rotational stages.

Although the steppers worked very well when operated individually, issues began to arise when all were running at the same time. In those conditions, the motors would dump too much back-EMF in the clock lines such that the signals became unstable and the motors skipped or operated improperly. This was especially visible in the A-stage which would fail to produce enough torque to fully rotate the build plate 90 degrees. As it was also the largest motor, it consumed the most power and dumped the most back-EMF into the clock lines. Although it was not expected that the rotational and translational axes operate at the same time during printing of a part, the printer was expected to be capable of doing so in order to support future, more advanced software operations.

Several solutions were brainstormed to solve this issues as well. The first was to implement a low-pass filter on the clock line to filter out high-frequency noise. The second was to more clearly separate the individual logic lines to avoid cross-talk. This was accomplished by establishing separate analog and digital ground lines, and making sure each high-power, analog, and digital ground was properly isolated. The third potential way to improve noise would be to physically separate the logic lines from the power lines when running wires to the stepper motors. In its current implementation, the high-voltage, high-current lines shared spots on the same ribbon cable as the logic lines. Inductive interference from the high-powered lines potentially added noise to clock signals.

DB15 terminals

The original intentions for using DB15 ribbon cables and terminals were two-fold: streamline and clean-up the wiring between logic and mechanical actuators, and ensure communications lines break before pulling enough that an actuator or microcontroller is damaged. While the DB15 cables improved wiring, they performed poorly as a safety factor. The DB15 connector fit very snugly inside of the terminal and required quite a bit of force for removal, and typically had to be pulled in one specific direction for proper removal. The worry was that the terminal would pull on and damage the protoboard before the cable would disconnect. Thankfully, the installation of limit switches made it extremely unlikely that the printer would ever be able to extend the steppers outside the range of the cables.

Frequency generators

Although the frequency generators performed well as a variable clock source at our current resolution and control loop frequencies, some issues remained with their use. For one, control of the steppers could only be done by way of sending clock signal frequencies to the frequency generators, and precise movement had to be timed rather than counted out by number of steps. This meant that the microcontroller was incapable of step by step control of the stages. This was an issue both in terms of controlling precision of the stages, and providing manual control to the user. Although 12 micron accuracy was confirmed via use of a dial indicator, further raising this resolution or step-by-step control would require the microcontroller to itself time control of the steppers.

4.3.2 Embedded software

Given an arbitrary set of discrete path points, the algorithm was capable of computing and smoothing the path into discrete velocity points. Unfortunately, due to the aforementioned electrical issues, the path planning was not tested using an actual object toolpath. The clock signals going into the stepper motors during operation proved too noisy to properly test toolpaths. However, the path planning algorithm was implemented and tested via MATLAB simulation. This can be seen in Figure 4.3.

Some issues were encountered when implementing the G-code processing on the microcontroller. Namely, the process of reading from a file was lengthy in comparison to the processing speed of the microcontroller. Although the buffer systems for separately processing file characters and velocity instructions worked to reduce the impact, the operations performed by the microcontroller had to be carefully balanced and timed to ensure the steppers would have continuous path smoothing. An improvement on this would be to add an additional microcontroller to the embedded logic. The responsibilities of running the steppers and processing instructions would be split off to two different processors. The instructions controller would be connected to the stepper controller via a high-speed communications port to transfer instructions.

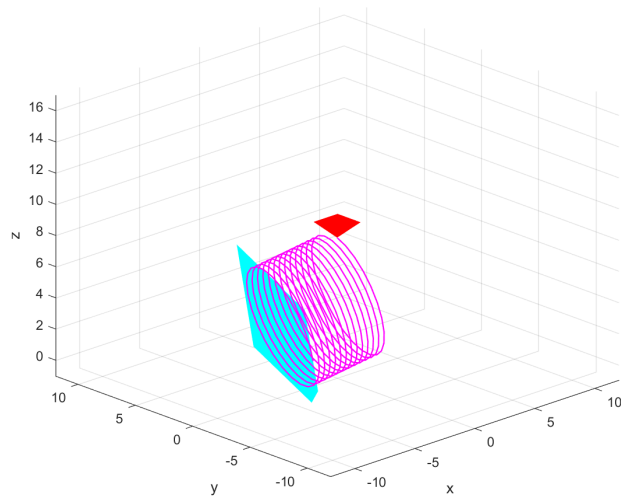


Figure 4.3: Path smoothing algorithm simulation for printing a cylindrical base. The square in blue is the build plate, the purple represents extruded material, and the nozzle is the red inverted square pyramid.

Chapter 5

Conclusion & Future Work

The goal of the project was to develop a hardware and software platform capable of 5-axis 3D printing. Given the time scale of the project and the size of the team, many hurdles were overcome despite those constraints. A prototype printer was constructed in addition to a software package capable of generating 5-axis G-code instructions for the printer. Although unexpected setbacks in the electronics of the printer ultimately prevented printing test pieces, independent validation of the sub-components through either simulation or testing gave credence to the capabilities of the platform. However, some additional improvements need to be made in order to make the printer capable of printing which are detailed below in addition to optimizations and general improvements.

5.0.1 Continuous 5-axis Printing

Currently our printer uses an approach that separates the 3-axis printing from the rotational axes, meaning material is never extruded while the rotational axes are in use, and once they have stopped rotating printing is treated as 3-axis printing. To fully make use of the 5 axes, our printer would extrude material while using the linear and rotational axes simultaneously. This would allow for even better build quality and potentially faster print times.

5.0.2 Advanced Collision Detection

Collision detecting for 5-axis 3D printing is an issue with very little background research available, as it is not an issue that arises in 3-axis printing. While we believe our for collision detection does not produce any false negatives in collision detection (saying there is not a collision while in reality there is one), there are many opportunities for false positives, which can lead to longer print times or worse build quality. For example, when rotating the object, we move the build plate to its lowest z value before rotating to avoid hitting the nozzle. It is possible for the software to calculate the minimum distance the build plate needs to move to ensure the nozzle does not collide with the already printed object.

5.0.3 Advanced Sub-Volume Sequencing

There are many possible objects that have two or more sub-volumes that cannot be printed at their optimal orientation due to collisions with the nozzle. At this stage our software does not

seek a solution to this problem and instead reverts to traditional 3-axis printing if such a cycle is detected. We hope to implement algorithms to find solutions to collision cycles, either by finding different orientations for the sub-volumes or generating support structures in different orientations.

5.0.4 Software Optimizations

There are numerous ways to optimize our software to run faster, although some are more impactful than others. Some significant optimizations we plan to make include multi-threading, memoization (storing results of an possibly expensive operation so the operation only needs to be called once), and improving time complexity of our algorithms.

5.0.5 Servos

A future improvement that could be retro fitted to the existing prototype of the printer are servos as an alternative to stepper motors. A similarly sized servo motor, at least in terms of current, would have far more torque. In addition, jerk limited control would be advantageous over acceleration limited control that is used with the stepper motors. Lastly, a dc motor with accompanying encoder would be significantly cheaper than a stepper motor.

Bibliography

- [1] Warren M Rohsenow, James P Hartnett, and Ejup N Ganic. “Handbook of heat transfer fundamentals”. In: *New York, McGraw-Hill Book Co., 1985, 1440 p. No individual items are abstracted in this volume.* (1985).
- [2] Prashant Kulkarni, Anne Marsan, and Debasish Dutta. “A review of process planning techniques in layered manufacturing”. English. In: *Rapid Prototyping Journal* 6.1 (Mar. 2000), pp. 18–35. DOI: 10.1108/13552540010309859. URL: <http://search.proquest.com/docview/214037039>.
- [3] Prabhjot Singh and Debasish Dutta. “Multi-Direction Slicing for Layered Manufacturing”. English. In: *Journal of Computing and Information Science in Engineering* 1.2 (2001), p. 129. DOI: 10.1115/1.1375816.
- [4] Yodthong Baimark and Robert Molloy. “Synthesis and characterization of poly (L-lactide-co- ϵ -caprolactone) copolymers: Effects of stannous octoate initiator and diethylene glycol coinitiator concentrations”. In: *Science Asia* 30 (2004), pp. 327–334.
- [5] Joseph E Shigley, Charles R Mischke, and Richard G Budynas. *Mechanical engineering design*. McGraw-Hill, 2004.
- [6] R. Sundaram and J. Choi. “A Slicing Procedure for 5-Axis LaserAided DMD Process”. English. In: *Journal of Manufacturing Science and Engineering* 126.3 (2004), p. 632. DOI: 10.1115/1.1763180.
- [7] John J Craig. *Introduction to robotics: mechanics and control*. Vol. 3. Pearson Prentice Hall Upper Saddle River, 2005.
- [8] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. English. 3rd ed. 2010. Upper Saddle River, New Jersey: Prentice Hall, 2010.
- [9] G. Strano et al. “A new approach to the design and optimisation of support structures in additive manufacturing”. English. In: *The International Journal of Advanced Manufacturing Technology* 66.9 (June 2013), pp. 1247–1254. DOI: 10.1007/s00170-012-4403-x.
- [10] James Anderton. *3D Printing and 5-Axis Machining Combined in One Machine > ENGINEERING.com*. Oct. 2014. URL: <http://www.engineering.com/advancedmanufacturing/articleid/8778/3d-printing-and-5-axis-machining-combined-in-one-machine.aspx>.
- [11] Jonathan Juursema. *A photo of the printing head of a FELIX 3D Printer in action*. 2014. URL: https://commons.wikimedia.org/wiki/File:Felix_3D_Printer_-_Printing_Head.JPG.

- [12] “‘Subtractive’ Industries Are Coming to Terms with Additive Manufacturing”. English. In: (Nov. 2014).
- [13] J. Vanek, J. A. G. Galicia, and B. Benes. “Clever Support: Efficient Support Structure Generation for Digital Fabrication”. English. In: *Computer Graphics Forum* 33.5 (Aug. 2014), pp. 117–125. DOI: 10.1111/cgf.12437. URL: <http://onlinelibrary.wiley.com/doi/10.1111/cgf.12437/abstract>.
- [14] Ian Gibson, David Rosen, and Brent Stucker. *Additive Manufacturing Technologies*. English. 2nd ed. 2015. New York, NY: Springer New York, 2015. DOI: 10.1007/978-1-4939-2113-3.
- [15] Øyvind Kallevik Grutle. *5-axis 3D Printer*. Aug. 2015. URL: http://www.robotikk.com/student/projects/OyvindKallevikGrutle3Dprint_5ax_ELDAT/Grutle-master.pdf.
- [16] Kyubok Lee and Haeseong Jee. “Slicing algorithms for multi-axis 3-D metal printing of overhangs”. English. In: *Journal of Mechanical Science and Technology* 29.12 (Dec. 2015), pp. 5139–5144. DOI: 10.1007/s12206-015-1113-y.
- [17] A. Savini and G. G. Savini. “A short history of 3D printing, a technological revolution just started”. English. In: IEEE, 2015, pp. 1–8. DOI: 10.1109/HISTELCON.2015.7307314. URL: <http://ieeexplore.ieee.org/document/7307314>.
- [18] Pranjal Singh. *Graphic showing support structures for a lego block*. 2015. URL: https://commons.wikimedia.org/wiki/File:Supports_in_3D_printing.png.
- [19] Alessandro Ranellucci. *Slic3r*. Version 1.2.9. Apr. 19, 2017. URL: <http://slic3r.org/>.
- [20] Ultimaker. *Cura*. Version 2.5. Apr. 19, 2017. URL: <https://ultimaker.com/en/products/cura-software>.
- [21] URL: <http://electron9.phys.utk.edu/vectors/images/p20.gif>.
- [22] *3D Coordinate System*. URL: <http://electron9.phys.utk.edu/vectors/3dcoordinates.htm>.
- [23] *Revolutionary development cuts manufacturing times using Laser Metal Deposition - Case Study 583*. URL: <http://www.twi-global.com/news-events/case-studies/revolutionary-development-cuts-manufacturing-times-using-laser-metal-deposition-583/>.
- [24] *What Is An STL File?* URL: <https://www.3dsystems.com/quickparts/learning-center/what-is-stl-file>.