3D Printing Robotic Arm

A Major Qualifying Project Report:
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

_____
Mark Swanson

_____
Will Spurgeon

_____
Taylor Vass

_____
Monika Danielewicz

Date: March 25, 2016

_____
Professor Torbjorn Bergstrom
Project Advisor

_____
Professor Michael J. Ciaraldi
Project Advisor

Abstract

In this paper we show the process our group went through to make a FANUC LR-MATE 200*i*B or similar robot arm capable of producing 3D printed parts. It explains the design, construction, and testing of our extruder end of arm effector. We also show the process of creating a translator for G-code taken from an open source slicing software. Our translator takes this code and turns it into instructions for the LR-MATE 200*i*B. Ultimately, we were able to successfully and consistently print. Our paper ends with suggestions for future groups looking to continue the project.

Table of Contents

Table of Figures

1. Introduction

1.1. Objective

Our objective was to use a six-axis robot arm as a fused deposition modeling 3D printer.

1.2. Rationale

Our project focused on the possibilities of using an industrial robotic arm to 3D print. There are a number of advantages this project could present, such as more flexibility or a larger range. Several projects of this nature already exist, though this platform is only beginning to be explored. To the best of our knowledge it has not previously been attempted by undergraduate WPI students.

1.3. State of The Art

Three-dimensional printing is a form of additive manufacturing that builds up material layer by layer to form a part, instead of removing material like most traditional manufacturing processing. There are five main categories of 3D printing: light polymerization, powder bed (laser sintering, laser melting, electron beam melting, binder jetting, and material jetting), extrusion, electron beam freeform fabrication, and laminated object manufacturing[1,2,3]. The most common method of 3D printing is fused deposition modeling (FDM), which we decided to use for this project.

In the extrusion process, the CAD model of the part is first sliced into layers and then translated into instructions that the printer understands. The material is then fed into the hot end which melts it, allowing it to be extruded onto the printing surface. The material solidifies and is built up layer by layer until the part is formed. To improve part quality, the printing surface can be heated. This helps to ensure that the material does not catch and get pulled up during the printing process. The two most common materials used in this process are plastics called Acrylonitrile Butadiene Styrene (ABS) and Polylactic Acid (PLA). The material our printer uses is PLA.

Robotic arms are computer-controlled machines made of segments connected with joints actuated by motors, hydraulics, or pneumatics. They are often used in manufacturing for repetitive and/or heavy labor. Robotic arms sometimes use a variety of sensors to determine position, velocity, and acceleration. More complex robots use more advanced sensors such as cameras or pressure sensors to observe their environment and learn their tasks. Less advanced arms, like the one we used for our project, merely know the position of their joints, allowing them to determine the position of the end effector. These arms come in many shapes, sizes, and degrees of freedom – the complexity depends on the purpose.

Robotic arms have previously been used for various 3D printing projects. Two notable examples of plastic extrusion with robotic arms were created by Dutch engineer Jasper Menger, who uses his printer to create large-scale plastic objects [4,5], and Dutch designer Dirk Vander Kooij, whose printing process is very similar but instead concentrates more on the artistic applications of 3D-printing [6]. Other plastic extrusion projects put more focus on the freedom that using a robot arm offers, creating freeform structures that seem to defy gravity. The company Mataerial is one such venture [7], as well as the spider web-slinging KUKA arm developed by students at the College of Architecture and Urban Planning at Tongji University [8]. As for other materials, 3D-printing robot arms have been developed to create structures of stone [9], ceramic [10,11], and metal [12]. Branching away from extrusion and powder bed methods, one company utilizes the light polymerization method of injecting light-curing resin [13]. These applications are all innovative ways of combining the use of robotic arms with 3D-printing technologies. It is still a field that is just beginning to be explored. Our project was meant as a small-scale proof of concept on an older model robot arm, and as a possible starting point to explore these applications more thoroughly.

2. Methods

In order to complete the project objectives, we moved through the following steps. A detailed breakdown of each of the steps is located in the Results and Discussion section.

2.1 Decide Which System Components Should be Purchased and Which Should be Built by Our Team

We researched the components that make up a 3D printing extruder. We determined which components we would purchase and which we would build based upon: part complexity, cost of commercial equivalent, and feasibility of manufacturing.

2.2 Iteratively Design, Manufacture, and Test the Custom-Built Components

We utilized an iterative design approach to make the custom components of our assembly. We would first brainstorm ideas and discuss feasibility. From this discussion we would choose a design to manufacture, we then created CAD models of the parts, those CAD files would then be adjusted for manufacturability and then realized. Next we would test their effectiveness and adjust or remake the designs as necessary, beginning the process again. Each part mentioned in this section went through this design process.

2.2.1 Mounting Angle

For the process of deciding a mounting angle, there was no true prototyping done, however, there was much discussion on feasibility and simplicity.

2.2.2 Mounting System

The mounting system was designed through many paper designs trying to determine ways to overcome the challenges posed by the system. Two were prototyped. They were designed in CAD and iterated, or changed completely based on the changing needs of the project.

2.2.3 Idler System

Most of the process in designing this system came down to theoretical analysis. The design was ultimately chosen based on the parameters of ease of initial feed, and ability to hold the material in place allowing it to be fed through the hot end.

2.2.4 Extruder Assembly

Designing the extruder assembly was done largely by changing the enclosure design to fit the needs as they changed. The extruder assembly must effectively attach the idler, driving mechanism, and electronics to the robot. Designs were adjusted for manufacturing feasibility and effectiveness of extrusion.

2.2.5 Driving Mechanism

The driving mechanism needed to be capable of driving the filament through the hot end without damaging the filament in the process. Additionally, it had to effectively constrain the movement of the filament during the extrusion process.

2.3 Decide how to best program the FANUC robot

Before a decision was made, we familiarized ourselves with the robot. Once we familiarized with as much of the system as possible, we analyzed the pros and cons of the various programming techniques we found in our research. With this analysis done, we chose the simplest method that allowed us to meet our objectives.

2.4 Design the robot control system

Most 3D printers are controlled using G-code created using a slicing algorithm. The robot cannot execute G-code, so we brainstormed various solutions to the components that the robot control system required. Once prototype designs and plans for the individual components were completed, we analyzed the system as a whole. We ensured that each of the individual components could meet their requirements using a centralized power supply.

2.4.1 G-code Production

We researched popular slicing programs before arriving at one that was reliable and free to use.

2.4.2 G-code Translation

The G-Code Translation program was written iteratively. New program features were written then thoroughly tested before additional features were begun.

2.4.3 Program Compilation and Simulation

New robot programs were imported into the FANUC software and compiled before being exported and saved to a Compact Flash card.

2.4.4 Extruder Control

We brainstormed ways of easily and quickly ensuring precise temperature control of the hot end. We quickly began implementing our solution as soon as we found one that was cheap and versatile.

2.4.5 Coordinate Frame Configuration

The coordinate frames were configured using the information contained in the robot's documentation. Quick tests were performed to ensure that the configuration was correct.

2.4.6 Implement the Robot Control System

We purposefully designed the control system to be composed of smaller discrete systems. Because of this, we were able to design, implement, and test each component individually and in parallel before bringing them together and performing full system tests.

2.5 Assemble and Install Heated Bed

PLA, the type of material we planned on printing with, has a tendency to curl and become unstuck from non-heated surfaces. A heated bed was necessary to optimize print quality. After research into custom-made heated beds, we decided to follow a tutorial for the heat distribution components of the heated bed that we then adapted to suit our needs. The design and assembly of the base to hold the heat distribution components was also done by an iterative process: brainstorming a solution, creating a prototype, and altering it based on the issues confronted.

2.5.1 Design of Heated Bed Base

The height of the base was required to be easily adjustable by hand. This was to simplify the setup process and give us an additional fail-safe in case of a crash. We also brainstormed ways to ensure that the heated bed base would not break if the robot were to accidentally drive the hot end into the heated bed.

2.5.2 Assembly of Heated Bed

Because the heated bed was not a major focus of the project, we determined that designing a bed from scratch would take too long and take resources away from the rest of the project. The team conducted research online and was able to find a well-documented construction tutorial.

2.5.2.1 Installation

The hot end needed to be able to reach as much of the heated bed as possible. We performed an analysis of the robot's work envelope and found an optimal location for the heated bed.

2.5.2.2 Power

For the sake of simplicity, we chose a heated bed design that could easily be connected to a commercial power source without the need for any current-limiting or control circuits.

2.6 Test all parts of the system and print parts

We thoroughly tested the individual components of the system before attempting any prints. With the sub-systems tested, we chose a small CAD model to test print. We performed several tests, while changing certain settings between attempts. We also printed the model using a MakerBot Replicator 2X in order to compare our results with those of commercially-available 3D printers.

3. Results and Discussion

The results of the methods of design and analysis discussed in the above section are detailed below, along with the rationale behind each decision. We discuss buy or build decisions, extruder assembly design and manufacturing, system design, heated bed assembly, test prints, the overall project outcome, and suggestions for future MQPs.

3.1. Buy or Build Decisions

Part	Hot End	Stepper motor	Hobbed Shaft Collar	Idler	Extruder Assembly
Decision	Bought	Bought	Built	Built	Built

Table 1: Build or buy decision matrix for extruder parts

An extruder consists of a hot end, which heats up the filament and extrudes out of an aperture of the chosen size, and the driving mechanism, which forces the filament through the hot end. These parts must have their movements tightly constrained by an assembly to prevent the filament flow from being disrupted. Before the design phase could begin, we first needed to decide which parts needed to be built, and which parts needed to be bought. Ultimately, we bought the hot end, stepper motor, and other electronic components, and we built the extruder assembly, idler, and hobbed shaft collar.

We first determined that the assembly needed to be built. It would need to be securely attached to the robot arm and optimally extrude the material onto our heated bed. No such commercial product exists, so we needed to build it. The hot end, conversely, needed to be purchased. The temperature would need to be precisely controlled, and none of us have an extensive knowledge of Electrical Engineering. MakerBot, the company that made Washburn's 3D printer, sells the hot end used in their own printers. The temperature dissipation is well controlled, and the nozzle is a precise 0.35mm. Additionally, this hot end comes with a built-in thermistor. It is a better hot end than we could have made, and the choice to purchase this item saved us several weeks of work.

The driving mechanism was partly built and partly bought. The actual driving was done by a standard Nema 17 stepper motor, which we elected to buy. It is a cheap and easily available commercial product that we do not possess the knowledge to build. The force on the

filament is supplied on one side by a hobbed or knurled shaft collar, and by some kind of idler on the other. The idler was to be custom built so that it could easily be actuated when attached to the extruder assembly. The shaft collar is a complicated and necessary part of the assembly, but all of the commercially available options were determined to be too expensive. As such, it too, was determined to be built.

3.2 Design, Manufacture, and Test

3.2.1 Mounting Angle

The first goal in the design process was to determine the manner in which we would mount the extruder assembly. Our first step was to decide in which orientation it was to be mounted. We determined that there were three viable options: in line, perpendicular, or angled. We ultimately chose an offset in-line mounting method.

3.2.1.1 In-Line

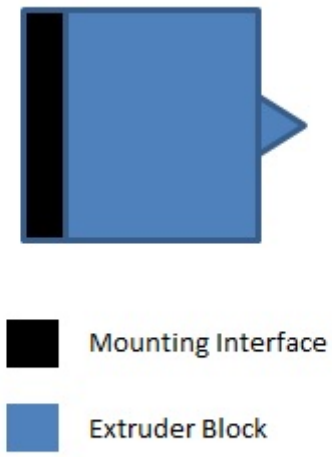

Figure 1: In Line Mounting Angle

This design provided the far simplest tool offset, as well as a relatively consistent work envelope. The major drawback to the in line mounting is the feed of the material. When testing, the material tends to snap when bent sharply, therefore requiring the extruder assembly to be pushed farther out from the mounting plate on the robot.

3.2.1.2 Perpendicular

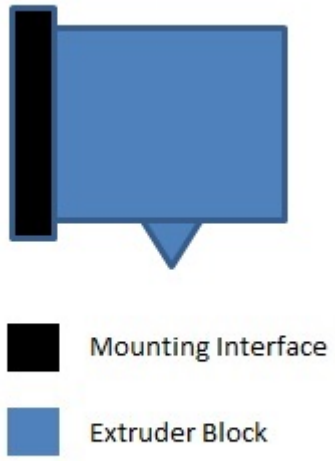

Figure 2: Perpendicular Mounting Angle

While this design was intended to help with the ease of feeding the material, the filament would still suffer a sharp turn, reiterating the issue of the material potentially snapping. This configuration would also severely limit our work envelope.

3.2.1.3 Angled

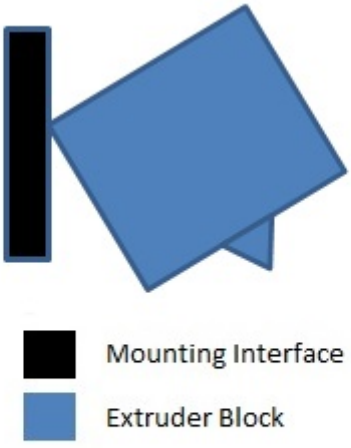

Figure 3: Angled Mounting

Mounting the extruder assembly at an angle was believed to be a good compromise between the two earlier approaches. We assumed that this would allow for an ease of feeding and maximum use of work angle. This design was scrapped due to concerns for manufacturing feasibility and sturdiness of mounting.

3.2.2 Mounting System

After scrapping the first three designs, we decided to move on to a different approach. We took the idea of not mounting the extruder assembly flush to the robot mounting plate and decided to work from there. We felt that it would help to mitigate the filament feeding problems, as well as increase our work envelope.

3.2.2.1 Extended Mounting Plate

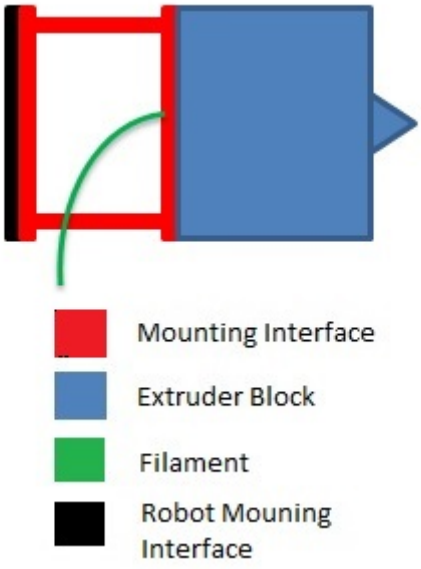

Figure 4: Extended Mounting Plate

The intent behind this design was to extend the mounting plate outward in order to allow for inline mounting. By extending the plate outward it would allowed the feed to come in at a shallower angle, reducing the chance of filament failure. This, however, was thrown out due to concerns of stability and ease of manufacturability.

3.2.2.2 Concentric Pipe Mounting

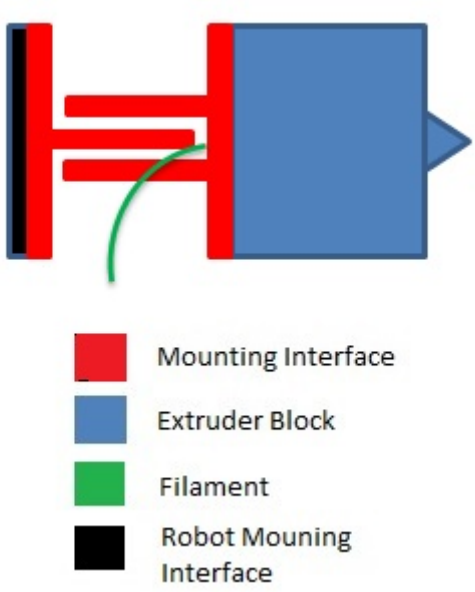

Figure 5: Concentric Pipe Mounting

This design intended to alleviate the problems in the small tool mounting interface. This was to be done by mounting a rod or pipe to the interface, attaching a slightly larger pipe to the extruder assembly, and then placing that over the piece mounted to the robot. There were concerns with the extruder assembly rotating and ultimately this design was scrapped because it was overly complex.

3.2.2.3 Holding Mount

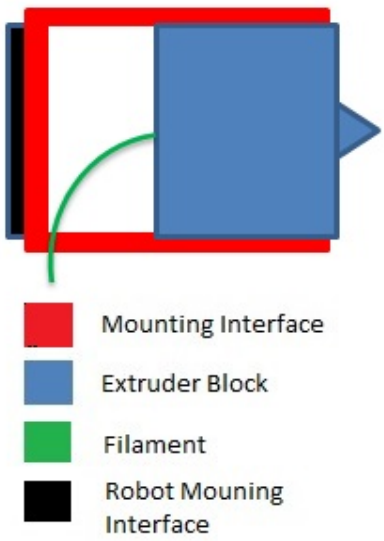

Figure 6: Holding Mount

We believed this design would allow the extruder assembly to be easily mounted at an angle. This design was ruled out because it required more material and ultimately offered no real benefits as we determined that an angled mounting system was sub-optimal. The filament would still be subjected to a sharp turn before entering the hot end.

3.2.2.4 Offset Mount

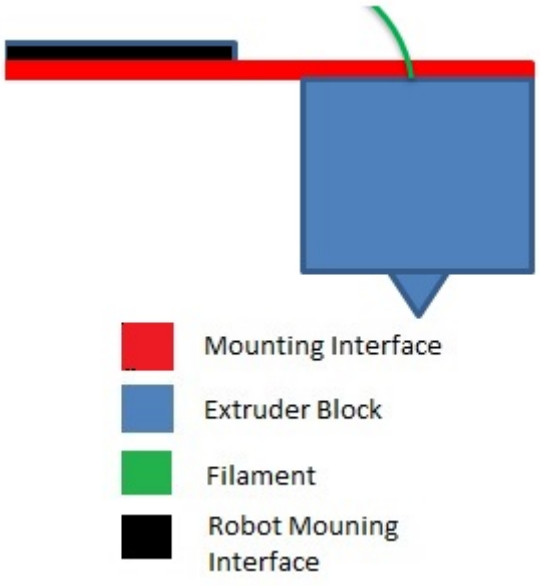

Figure 7: Offset Mounting

This is the design we finally chose due to its simplicity and lower material cost. By extending the roof of the assembly in order to mount directly onto the robot, the filament was able to continue on a natural arc over the robot's wrist. While this design did limit its range of motion, we were still able to achieve an acceptable work envelope.

3.2.3 Idler Design

When printing, an idler is required to keep the material pressed against the driving piece on the stepper motor. It must keep the material secure, though still allow for the material to be fed into the extruder before the beginning of the print. The challenge in the design was managing a balance between the snug fit and the ease of initial feed.

3.2.3.1 Fixed Bearing

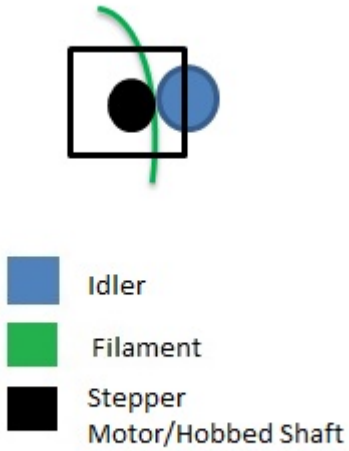

Figure 8: Fixed Bearing Hold

This design is the easiest to manufacture because it requires fewer moving parts. We feared that the friction caused by the filament moving against the stationary idler would slow the rate of extrusion, ultimately leading to lower quality parts. Without some way of moving the idler from the shaft collar, we feared loading filament would be unnecessarily difficult.

3.2.3.2 Tension Spring Bar

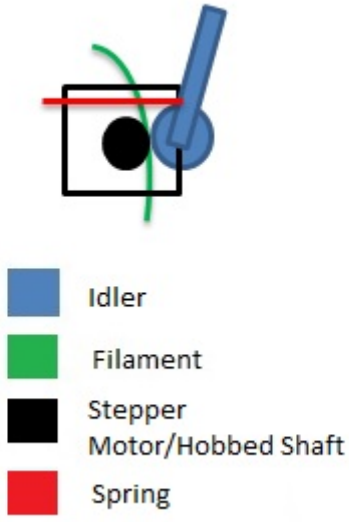

Figure 9: Tension Spring Levered Hold

This design allows the idler to be moved for filament loading. Applying pressure to the end of the idler assembly would cause the other end to swing away for the hobbed shaft collar. The wheel actually in contact with the filament was designed to spin when in contact with the driven shaft collar. There were concerns with this design as the spring would have to cross over the filament and attach to the far wall of the extruder assembly.

3.2.3.3 Compression Spring Bar

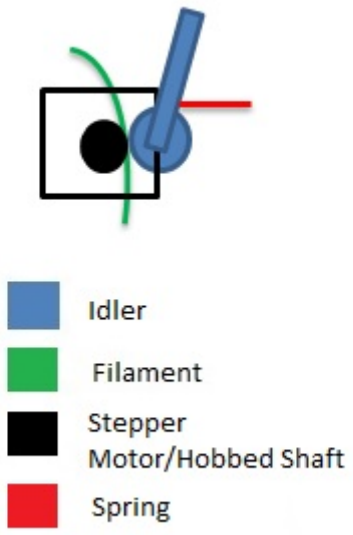

Figure 10: Compression Spring Levered Hold

This design alleviates the problem of the spring crossing over the filament. While this was the design we ultimately chose, we had significant issues applying enough pressure with the springs we had access to.

3.2.3.4 Idler Manufacturing

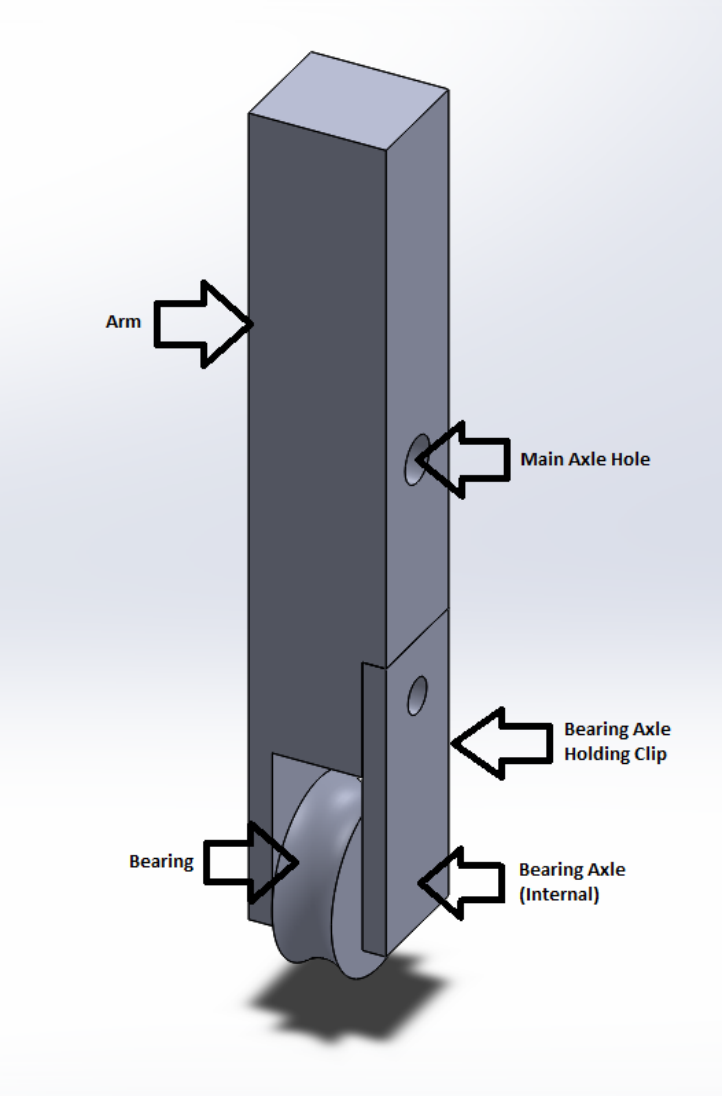

Figure 11: Full Idler Design

The idler originally consisted of five components: the arm, the main axle, the bearing, the bearing axle, and the bearing axle holding clip. The bearing is the piece that is actually in contact with the filament, and is meant to rotate as the filament is driven downwards. It rotates around the bearing axle, which originally sat in a pocket machined into the arm on one side, and in a pocket built into the 3D printed bearing axle holding clip. The clip was made using the MakerBot Replicator 2X. The entire assembly rotates around the main axle. A hole in the right side of the idler allows the end of a spring to sit in it and provide pressure against the filament. Instead of machining the bearing, as originally intended, a rubber-edged bearing was procured.

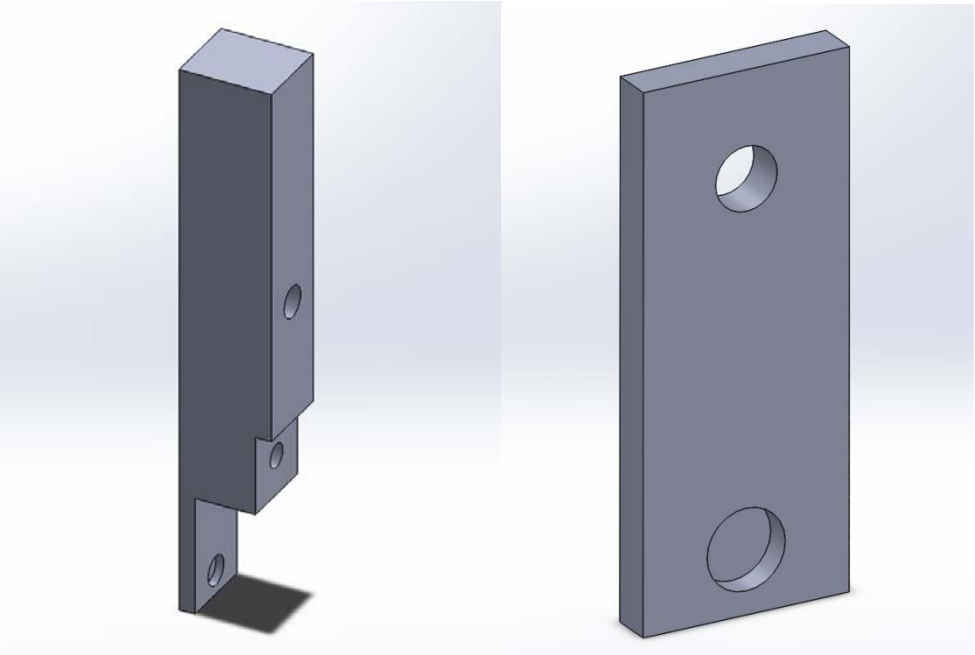

Figure 12: Idler Lever Function

3.2.4 Extruder Assembly

The assembly itself needed to contain the hot end, the stepper motor, and the idler system. Initially, we intended the Arduino to be located in the extruder assembly as well. We chose to place the Arduino below the heated bed where there is more room. We originally had concerns with the hot end overheating our system, but in practice the system remained cool.

3.2.4.1 Iteration 1: Testing Manufactured Components

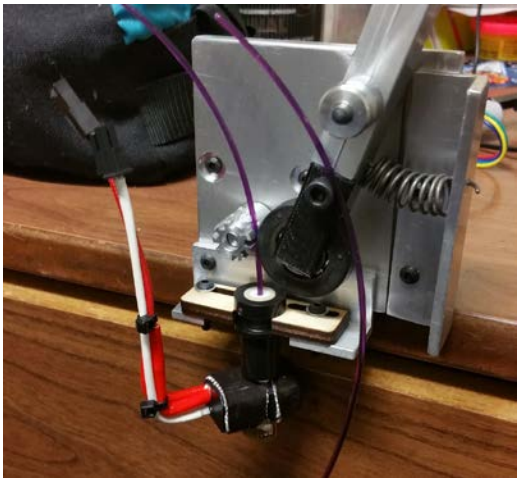

Figure 13: Initial Test Construction

This initial extruder assembly was built to test the effectiveness of the idler-shaft collar method of extrusion. It consists of a single mounting plate to which the other components are bolted. The stepper motor is attached in the bottom left of the plate, with the idler pivoting off the top right. An L-bracket is bolted to the left of the plate so that the idler's spring can be compressed. Bolted to the bottom edge of the plate is another L-bracket with an open pocket and two slots machined into it to allow for the laser-cut plywood hot-end-holding piece to be positioned optimally. However, the bearing axle was not capable of applying enough pressure, so it had to be replaced. The axle pocket in the idler arm was drilled out and the bearing axle replaced with a 3/4in 10-24 bolt. The bearing axle clip was removed entirely. With this new, stronger axle we were able to successfully extrude material with this assembly. We decided to build on this design.

3.2.4.2 Iteration 2

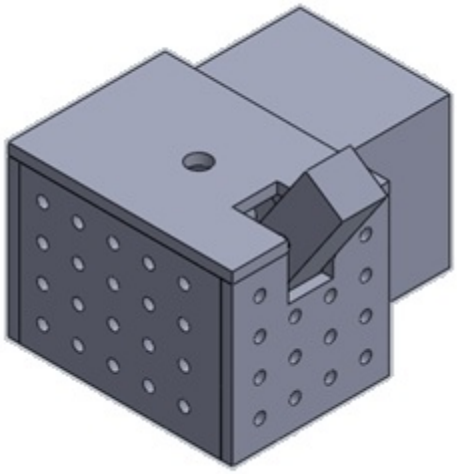

Figure 14: Partially Enclosed Design

Initially, the design of the EOAT extruder assembly had the stepper motor and Arduino attached to the outside of the back plate, with the idler pivoting off the top right. In order to prevent heating, the front, left, and right face were perforated. The hot end would be attached to the bottom plate. In the top plate there was a hole to feed the material. It was suggested that the front plate be hinged to allow access to the inside of the assembly for maintenance and filament loading, but this was determined to be unnecessary due to how rarely this would be needed.

3.2.4.3 Iteration 3

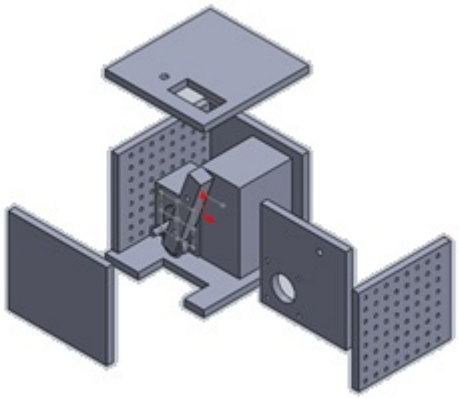

Figure 15: Fully Enclosed Design

The third iteration added a second section to the back of the extruder assembly in order to better contain the Arduino and stepper motor. Nothing else was changed in this iteration.

3.2.4.4 Final Iteration

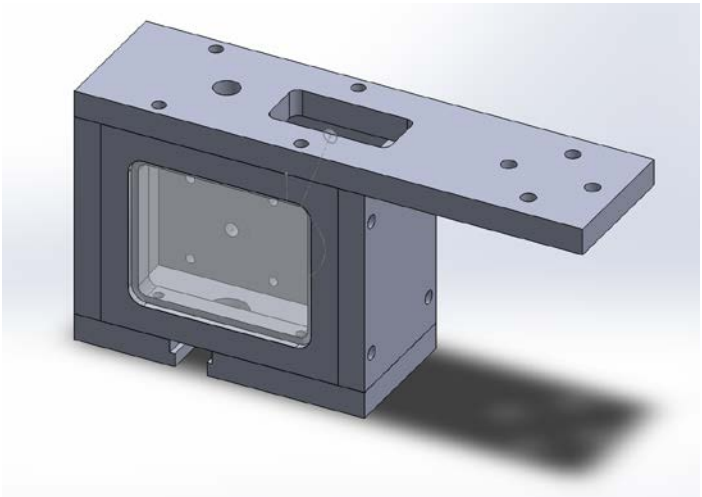

Figure 16: Final Assembly Design

In this final design, the back section was removed. This was to reduce weight and unnecessary material costs. After testing temperatures from the hot end we found that there was not a need for the ventilation, and decided to remove the perforation to avoid having to

either waste material making it, or to waste money purchasing it. In the bottom panel a slot was added to precisely line up the hot end with the filament groove on the hobbed shaft collar. An acrylic window was added to the front panel both to make it easier to load filament, and to better display the internal mechanism. It is held on be very short bolts that can be removed in case emergency access is needed. The entire assembly is held together by 16 10-24 bolts, two on each joined edge.

3.2.5 Design of Driving Mechanism

In order for the stepper motor to actually be able to drive the filament through the hot end, a hobbed or knurled shaft collar needed to be designed. This piece needed to be capable of gripping onto filament running tangentially to the cylindrical face, keeping the flow at a consistent spot. The hobbing would need to be aggressive enough to prevent the filament from slipping against it as the stepper motor ran, yet dull enough to not damage the filament. Damaged filament can lead to inconsistencies or even gaps in the extrusion. All designs have a center through hole with a diameter of 0.2 in (5.08mm), allowing for our 5mm stepper motor drive shaft, and a 6-32 tapped set screw hole, to keep the drive shaft in place. They have an outer diameter of 0.5in and a length of 0.7in.

3.2.5.1 Iteration 1

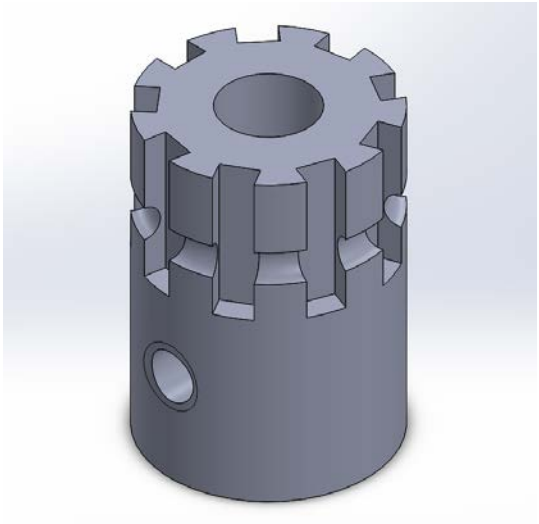

Figure 17: Filament Driver First Prototype

The preliminary design was based off of a gear. It has nine equally sized, equally spaced grooves around the cylindrical face. The grooves themselves are all tangent to the radius. These grooves were machined using live tooling on a Haas ST30 SSY. Traveling around the

cylindrical face at 0.15in from the front face of the shaft collar is another groove meant for the filament to sit in. The groove has a diameter of 0.07in (1.778mm) which gives a snug fit to our 1.75mm ABS filament. This is to constrain the movement of the filament, forcing it to consistently feed into the hot end. In this design, the filament groove is set into the material at a depth equal to the groove's radius: 0.035mm. As such, only half of the thickness of the filament was held in the groove. As a result, this design had a tendency to allow the filament to jump out of the groove. Additionally, there were concerns that the nine tangential grooves would merely rub against the filament without actually driving it due to their angle with respect to both the filament and themselves.

3.2.5.2 Iteration 2

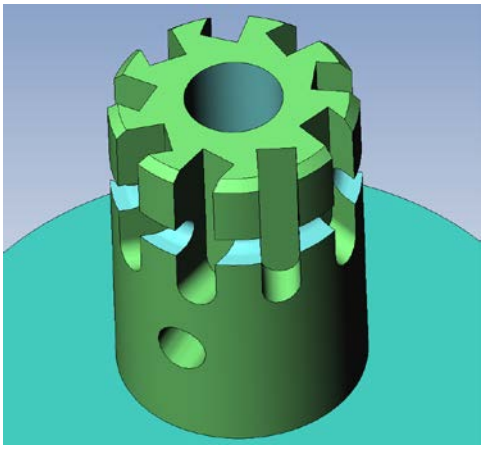

Figure 18: Filament Driver Final

The final shaft collar design has two major differences from the preliminary design. The first difference is the depth of the filament groove. The original depth of this groove had been 0.035in, but in this design it is 0.05in. This increase in depth helps to keep the filament from jumping out of the groove, but it is still less than the diameter of the filament. This allows the idler to still transmit its pressure directly to the filament instead of just to the shaft collar. The second difference is the angle at which the nine grooves make with the radius. They have all been tilted clockwise by 30°. This increased angle is to reduce slip against the filament. Additionally, the final shaft collar design has a chamfer around the front edge. This was added for aesthetic reasons.

3.3 Robot Familiarization

FANUC's proprietary TPP programming language included all of the functionality we required for this project. We explored the possibility of passing instructions to the robot through the built-in serial port. However, due to the severe lack of documentation, we decided that pursuing the serial connection would take too much time away from other parts of the project. Instead, we focused on loading programs directly onto the robot controller in their entirety. This ran the risk of running out of available memory on the device. After testing, we determined that we were able to print suitably sized models even with this limitation.

3.4 Initial System Design

The ultimate goal of this project was to allow a user to produce a 3D printed part from a standard CAD file using our setup. Several systems were created in order to automate the process of producing the physical part. Our system design included methods for converting a CAD model to runnable robot code, controlling the temperature and extrusion rate of the hot end, and executing printing programs on the robot. The workflow that a user moves through in order to complete the process is explained below.

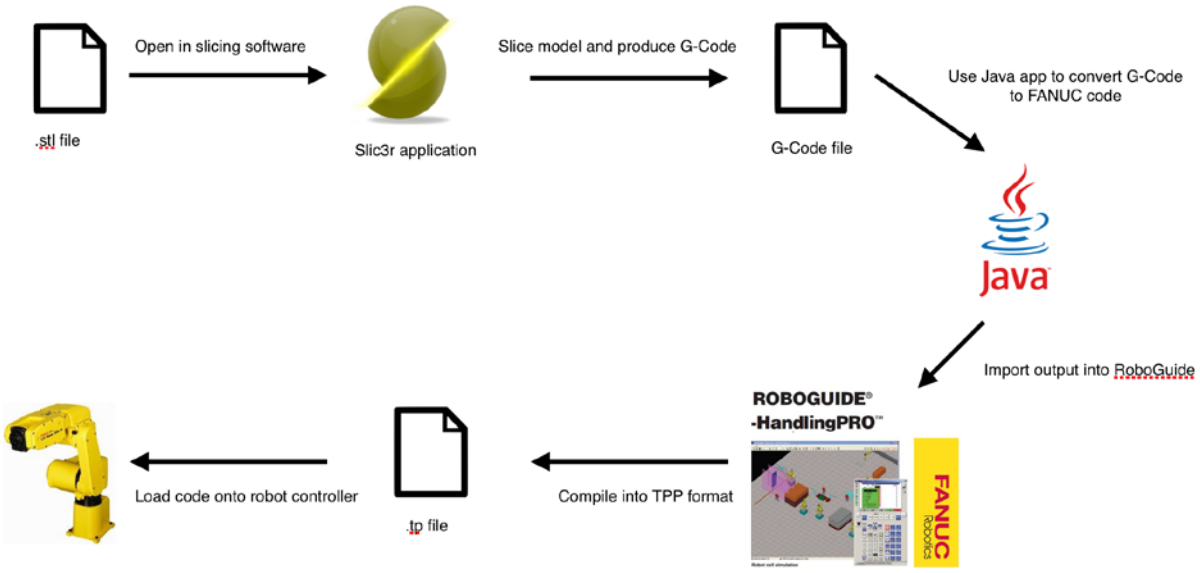

Figure 19: An overview of the process for converting a CAD model into motion instructions for the FANUC robot.

3.4.1 G-code Production

G-code is a popular language for Computer-numerical control machines and other automated machine tools. It is often used to describe the motions and actions of 3D printers. Many freely-available programs exist that convert CAD files into G-code motion instructions. These programs work by “slicing” the part into layers and producing the instructions that produce the part layer by layer. The G-code file produced by the slicing program can then be loaded directly into a 3D printer and run.

We chose to use the program Slic3r. Slic3r is a free and open source slicing program used throughout the 3D printing community. Slic3r allows users to customize the desired characteristics of the G-Code output and it includes an autospeed feature. Autospeed is described in detail below in the Extruder Control section.

3.4.2 G-code Translation

The FANUC robot controller we used for this project cannot directly read G-code. Instead, it primarily uses a proprietary programming method known as Teach Pendant Programming. 3D slicing programs universally output G-Code, not TPP code. We decided to create a Java program that would automatically produce TPP code from G-Code.

The robot controller includes a hand-held operator panel known as the Teach Pendant. Operators use the Pendant to jog the robot into desired positions and “teach” the robot where to move one instruction at a time. Because even small 3D print jobs involve thousands of motion instruction, it is impractical to teach the motions by hand. Instead, it was necessary to translate each motion instruction described in the G-code program into an equivalent instruction in the TPP language.

By far the most common G-code instructions used in 3D printing are the G0/G1, G2, and G3 instructions. These correspond to linear motions, clockwise arcs, and counterclockwise arcs respectively. The TPP language provides these instructions in a different syntax. The original G-code file was parsed instruction-by-instruction and the corresponding TPP instructions were generated and written to a file. Once a complete TPP file has been produced, it can be compiled and loaded onto the robot controller.

3.4.3 Program Compilation and Simulation

Before a TPP program can be executed by the robot, it must be compiled using FANUC's RoboGuide software. The TPP program text was saved to a .ls file. This file would then be opened in RoboGuide and saved as a .tp file. This file would finally be saved to a compact flash card which would be loaded into the robot controller. With the card installed in the controller, the power could be turned on and the program would then be run.

In order to ensure that the robot program had been successfully generated, we tested all newly made programs within the RoboGuide simulator before the program was run on the physical robot. RoboGuide provides a 3D representation of the robot arm and workspace. Our programs were loaded onto this virtual robot and run. If an error was present, it would become immediately apparent as all of the motions of the robot are drawn on the screen and the virtual robot controller reports any errors in the running of the program. Simulating allowed us to catch potentially dangerous robot programming errors before the physical robot was ever powered up. This saved both time and money as an error could easily cause the real-world robot to jog into a solid object.

3.4.4 Extruder Control

The filament is driven into the heating element by a stepper motor controlled by an Arduino Uno. We chose to use an Arduino because of its low price, ease-of-use, and abundance of online support documents and forums. The Arduino is connected to an AdaFruit Motor Shield 2.3, which provides power to the stepper motor.

Originally, we planned to have the extruder's extrusion rate able to change throughout the execution of the program. This would have allowed us more flexibility. In order for this to be possible, the FANUC robot controller would need to be capable of passing extrusion rate values to the Arduino. While this is surely possible, we concluded that it was more important to perfect the G-Code translation system. We developed a simplified method of extruder control instead.

Because Slic3r includes an auto-speed feature, we could be sure that each program required the extruder to operate at exactly one speed. Before the program begins, the Arduino is configured with this speed. The FANUC robot controller can now send simple ON/OFF signals to the Arduino extruder system. When the Arduino receives an ON signal, it extrudes at the given rate. If an OFF signal is sent, the Arduino simply stops extrusion.

The Arduino Uno also controlled the temperature of the hot end. The extruder's built in thermistor changes resistance as a function of the hot end's temperature. A voltage divider was used to produce a signal that could be read in by one of the Arduino's analog inputs. Power to the hot end is passed through a relay controlled by the Arduino. If the Arduino reads that the hot end is getting too hot, it switches power off until the hot end has cooled down.

3.4.5 Coordinate Frame Configuration

The robot's User and Tool Frame offsets were configured so that the extruder's "zero" point was located on the build surface, at the correct height. In this case, User Frame 8 held this information. The programs that we passed to the robot controller required that the robot have its active User Frame set to frame 8 before execution could begin. This ensures that a printing program cannot be accidently started if the robot is left with the wrong User Frame active.

3.4.6 Robot Operation

The basic process for printing a part is as follows:

1. Turn on power to the heated bed and hot end.
2. Ensure that User Frame 8 is set as the active User Frame Number.
3. After both components reach their ideal temperature, begin execution of printing program and turn on power to the stepper motor shield.
4. Adjust program speed (if necessary) to ensure that enough filament is being laid down by the extruder.
5. When the program completes, turn off all power to the electronics and carefully remove the newly printed part from the build surface.

3.5 Heated Bed Assembly and Installation

A heated bed was necessary to properly print the piece, and considering the workspace provided, a custom one had to be built. This included the actual assembly for heat distribution along with the base that it would be mounted on.

The design of the heated bed for this project was taken from an online tutorial posted by Jeremie Francois [14]. The tutorial detailed several failed attempts and explained the process of creating his final successful bed. The other tutorials for DIY heated beds that we found had

similar components and advice. We chose this tutorial because it focuses on the construction of a heated bed that is larger than the standard PCB (Printed Circuit Board) heated beds used for 3D printing. This was necessary for our build in order to utilize as much of the robot's work envelope as possible.

3.5.1 Design of Heated Bed Base

Like other 3D printers, the "zero" point of the hot end needed to be adjusted prior to each print in order to guarantee a clean print. Changing the coordinate frame configuration of the robot was a tedious process, and ultimately we concluded that it would be easier to adjust the base of the heated bed. For this reason, and to ensure that the heated bed made as much use of the robot's work envelope as it could, a variable height heated bed base was designed. The reasons for this are contained in the Installation section below. The base plate was made to be 14x14", since that would provide more than enough stability to an 18x18" heated bed without using too much material.

3.6.1.1 Support Iteration 1

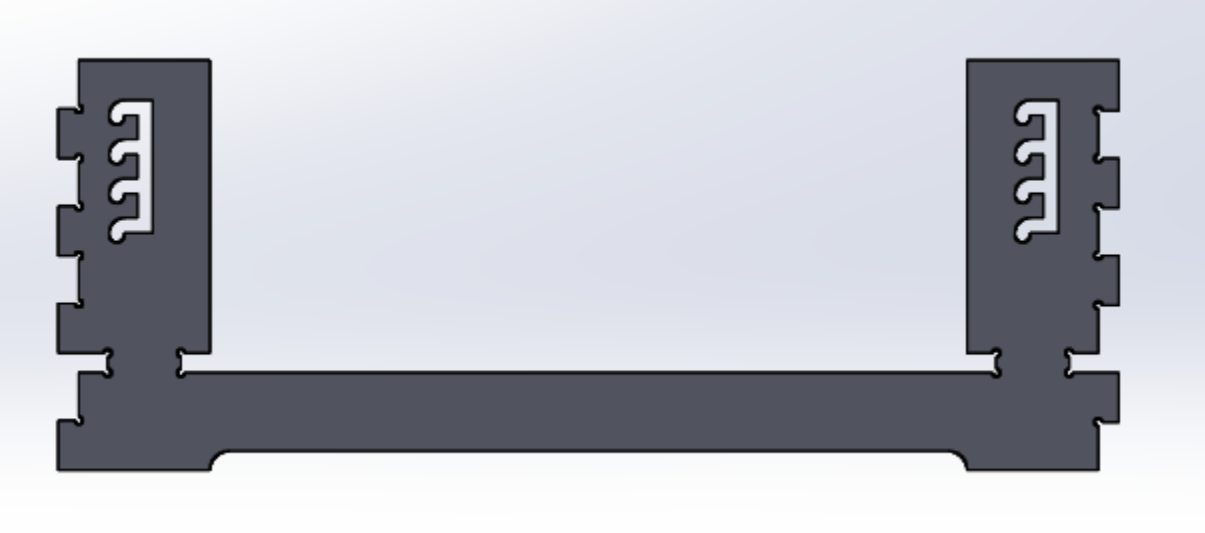

Figure 20: Heated bed support structure initial design

The supports were originally designed to be interlocking corners with long struts, cut from 0.25" plywood. The slots would allow for an assembly with four supporting bolts on the side to be lifted up and shifted around from level to level. However, the plywood used for the support was too thick and when cut with the laser cutter the edge became charred and crumbled.

3.6.1.2 Support Iteration 2

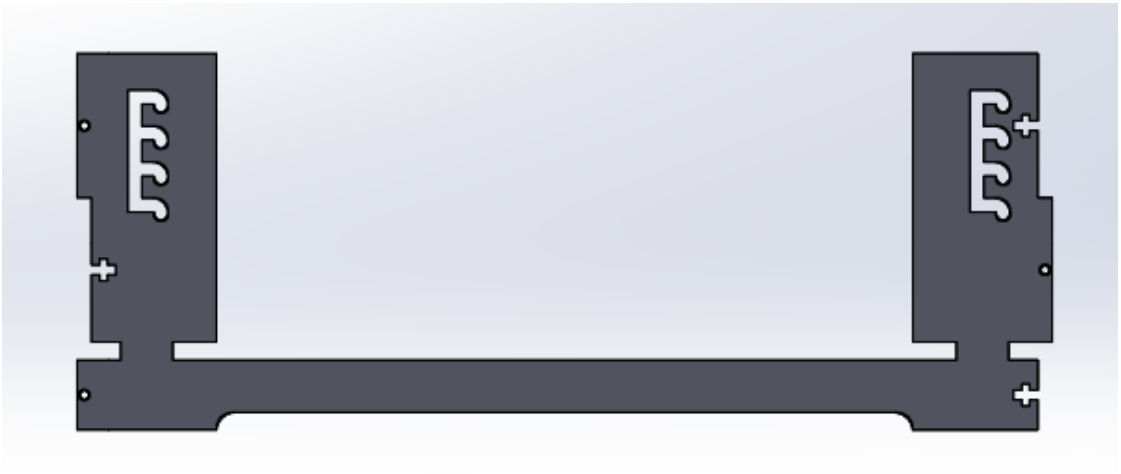

Figure 21: Heated bed support structure with slots

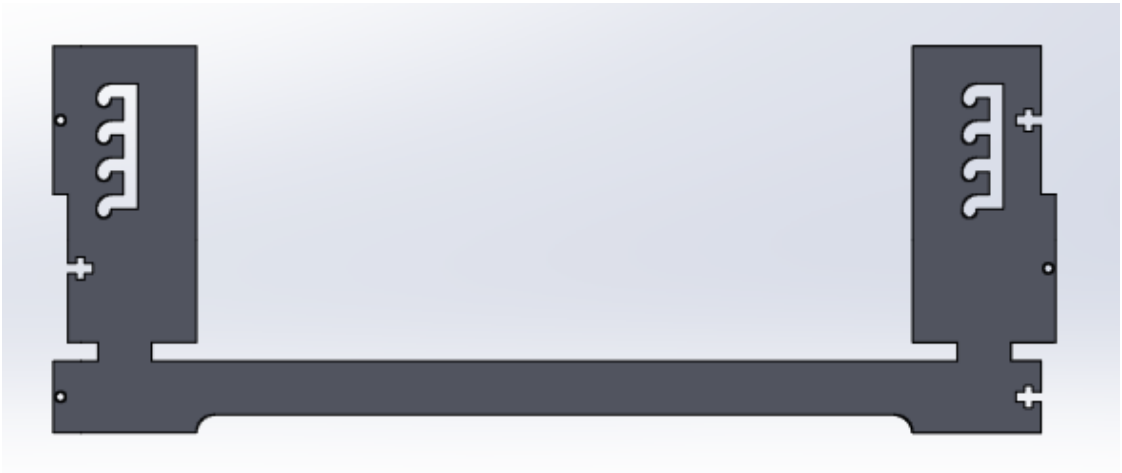

Figure 22: Heated bed support structure with reversed slots

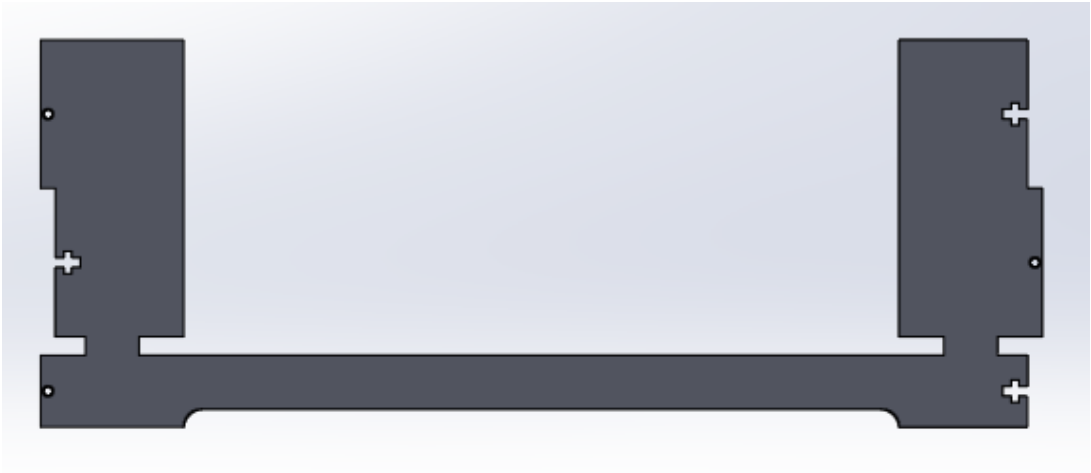

Figure 23: Heated bed support structure

The final design uses 0.22” acrylic instead of plywood, and has fewer interlocking corners to allow for greater tolerances in the cutting. The holes fit 6-32x1/2” bolts, and the cross-shaped holes fit the corresponding nuts, securing the four support pieces tightly together. With the final assembly, the height from the bottom of the stand to the top of the glass printing surface varies between 7.44, 7.95, 8.47, and 8.98 inches. There are three different support pieces - two with corresponding sliding slots placed opposite each other and connected with the third, plain support, of which there are two. The drawings for these pieces can be found in Appendix A: Drawings.

Figure 24: Sliding slot in final assembly

3.6.1.3 Top and Bottom

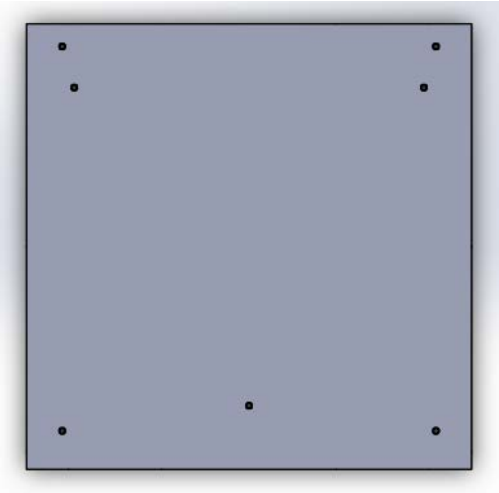

Figure 25: Heated bed top design

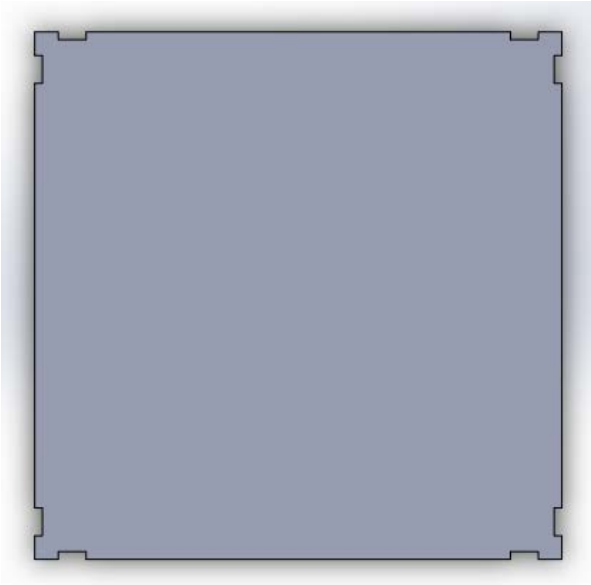

Figure 26: Heated bed bottom design

Plywood was still used for the horizontal top and bottom wooden components since the tolerances did not have to be as precise, and the top was simply a square piece drilled through with holes for bolts.

3.6.1.4 L-Brackets

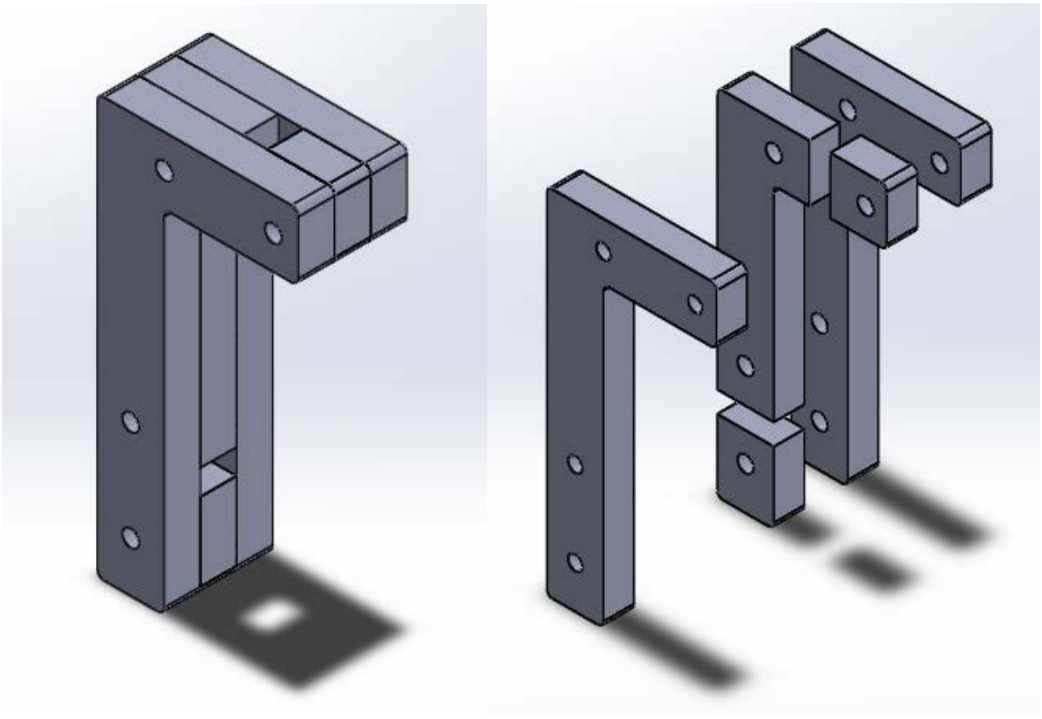

Figure 27: Heated bed L brackets CAD

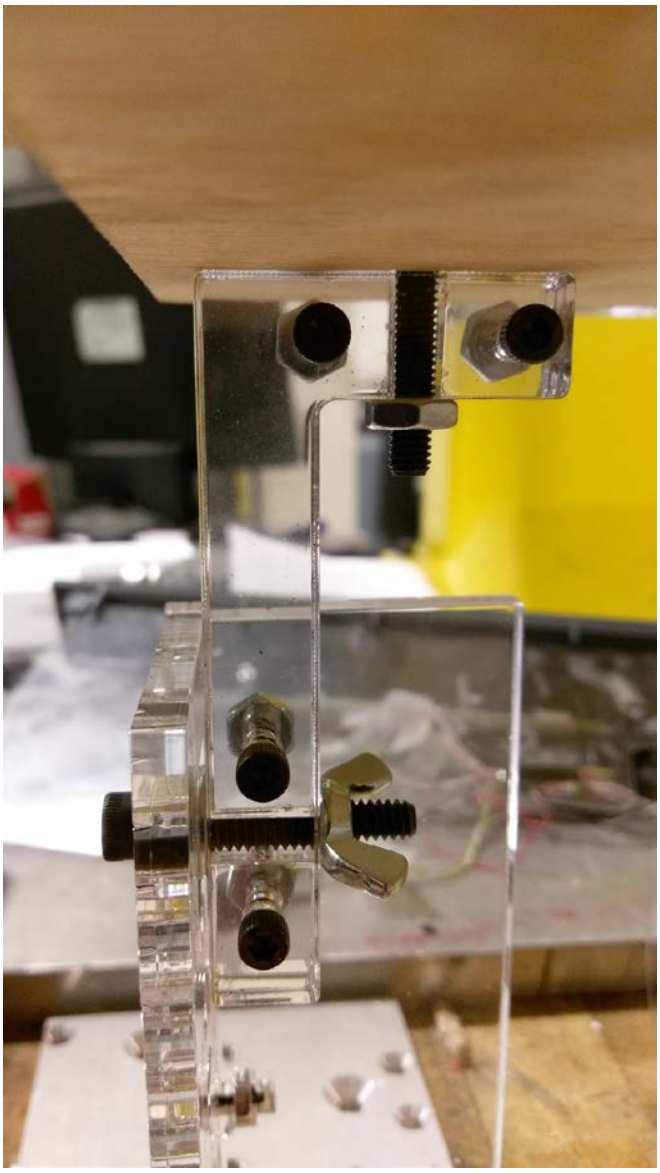

Figure 28: Heated bed L brackets

The top piece of the stand that would connect to the actual heated components was mounted onto four custom L-shaped brackets composed of five individual 0.22” acrylic pieces bolted together with 6-32x1” bolts. The rectangular hole on the horizontal part of the bracket was bolted to the underside of the top through the four outermost holes in the top with 10-24x1-¼” bolts. The rectangular hole on the vertical part of the bracket was used to attach to the supports with a 10-24x1-¼” bolt designed to loosen with wingnuts so it could slide through the slots on the support pieces.

3.6.1.5 Heated Bed Stand

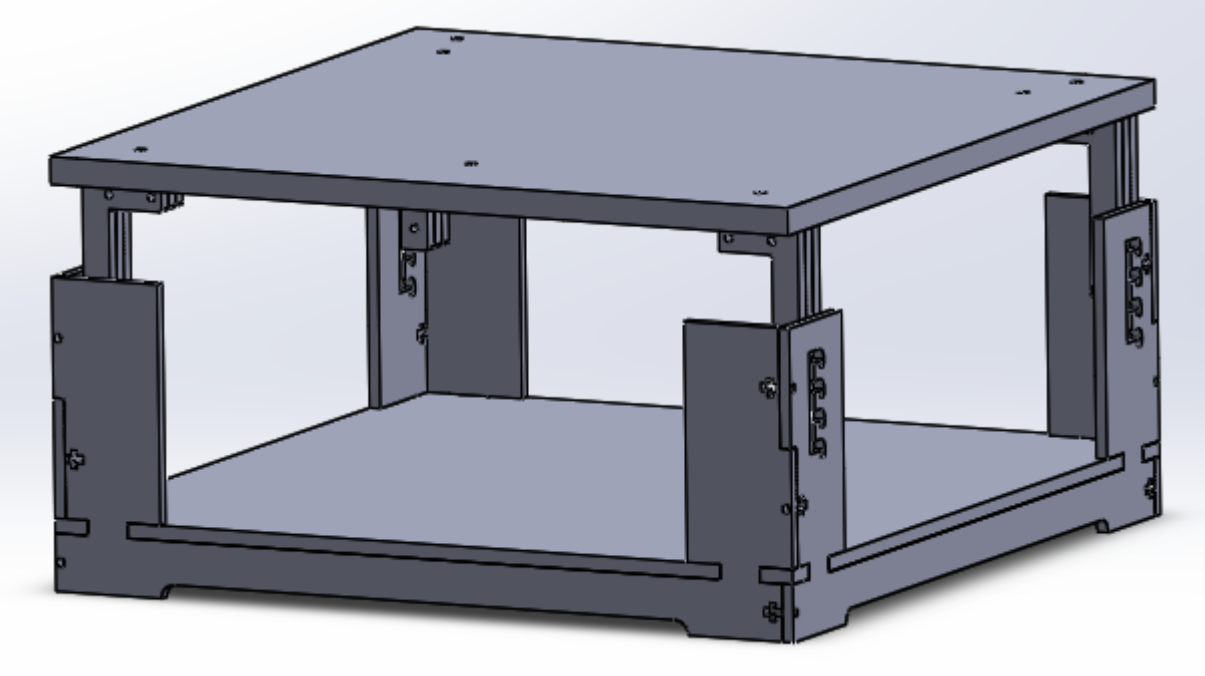

Figure 29: Heated bed design

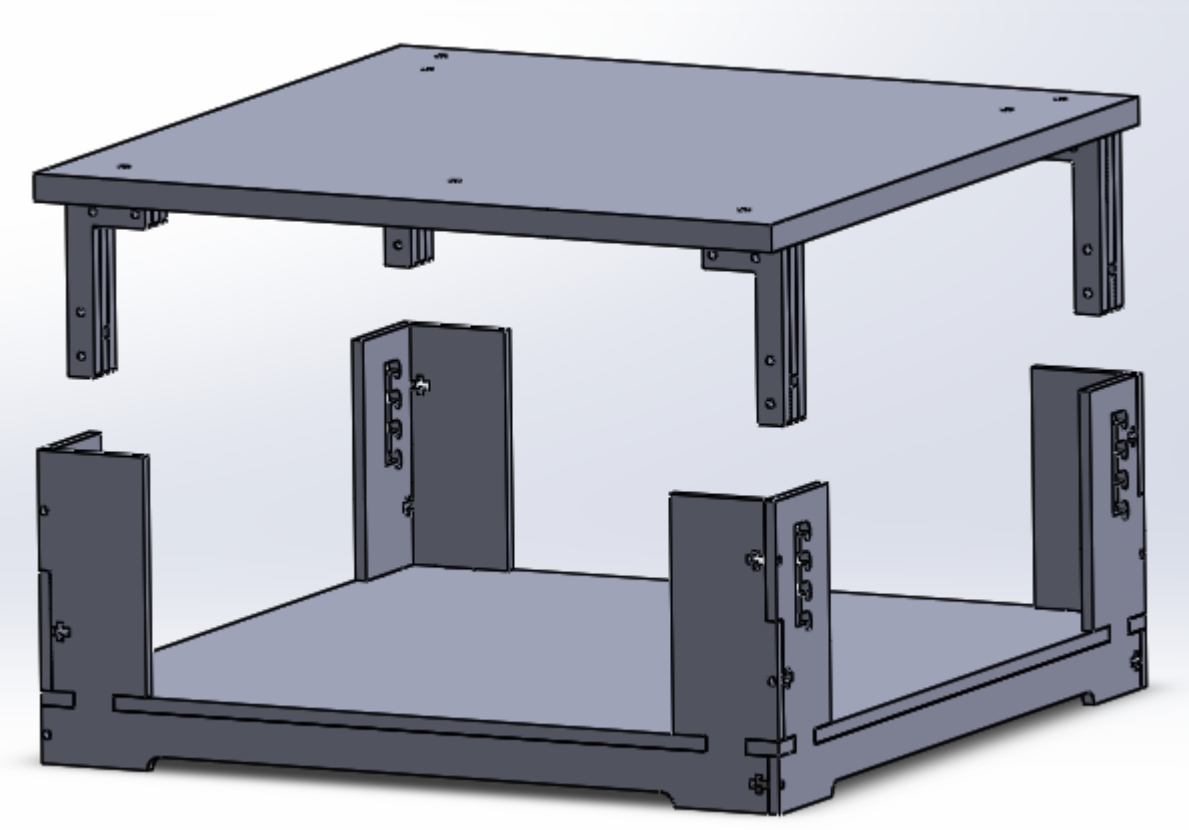

Figure 30: Heated bed design with separated top

This whole assembly was designed to keep the heat-distribution part of the heated bed at the correct level. The slots on the supports allows for the entire upper assembly to be lifted and moved to different heights.

3.5.2 Assembly of Heated Bed

The assembly of the actual heat-distribution components of the heated bed came almost directly from the tutorial, though the dimensions were modified to fit our needs. An 18x18x0.1875” sheet of aluminum was drilled with three 0.25” holes for three-inch countersunk bolts, enabling a flat surface for the glass to sit on.

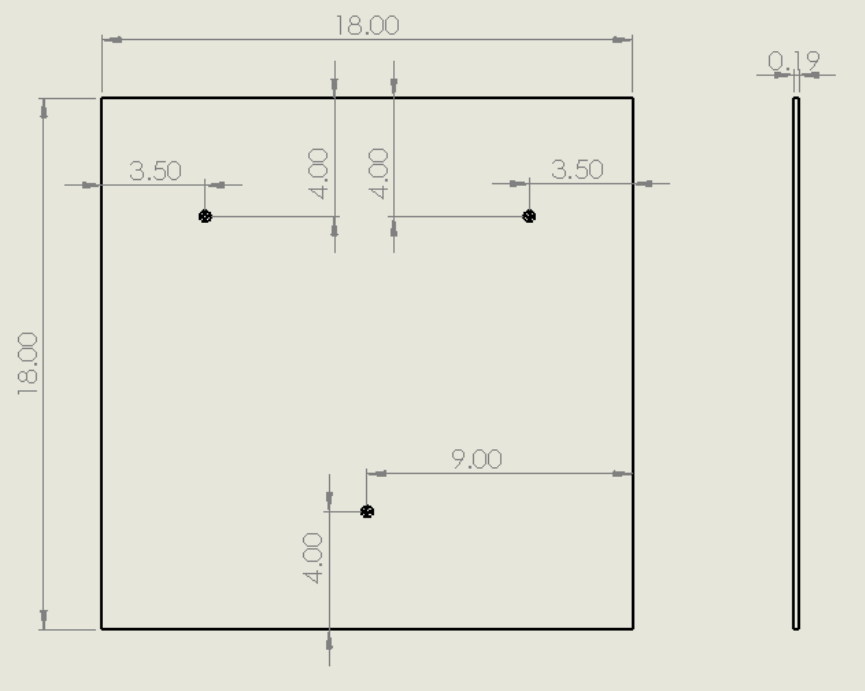

Figure 31: Measurements of holes on aluminum

An 18x18x0.25” tempered glass pane was placed directly on top of the aluminum and attached with eight binder clips. An MK2b Dual Power 12/24V printed circuit board (PCB) HeatBed was taped directly to the underside of the aluminum using kapton tape. An 18x18x0.125” sheet of plywood was attached underneath to ensure that the PCB was isolated and that its heat went primarily into the aluminum. To make sure the PCB was pressed firmly against the aluminum, a large rubber gasket was placed between the PCB and the plywood, with two hex nuts also in between on the three main bolts in order to ensure equal spacing.

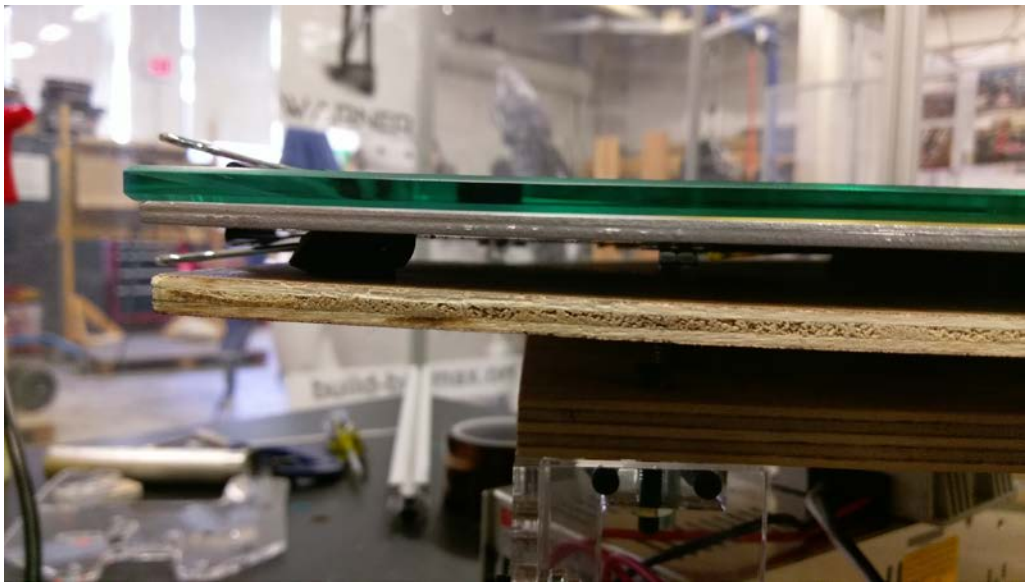

Figure 32: Placement of layers of heated bed

This entire assembly was then mounted onto the top wooden part of the stand using the three bolts and adding in springs in between. Although the tutorial used a rubber pivot for the leveling mechanism of the heated bed, we used springs, as is traditional in 3D printing, since they were much easier to locate and acquire.

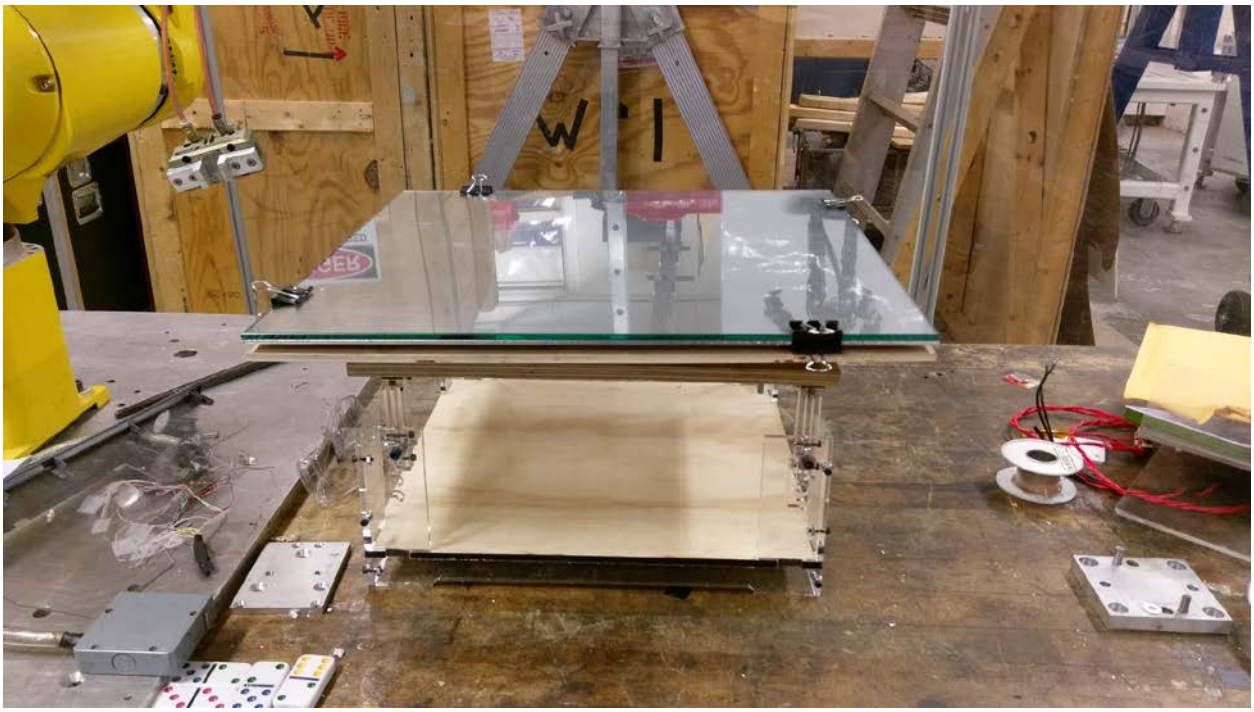

Figure 33: Final construction of heated bed

3.5.3 Installation

Since a square heated bed was the simplest and most efficient to construct, we used the work envelope of the robot arm to determine the position and height of the heated bed that would grant us the optimum amount of space to print in.

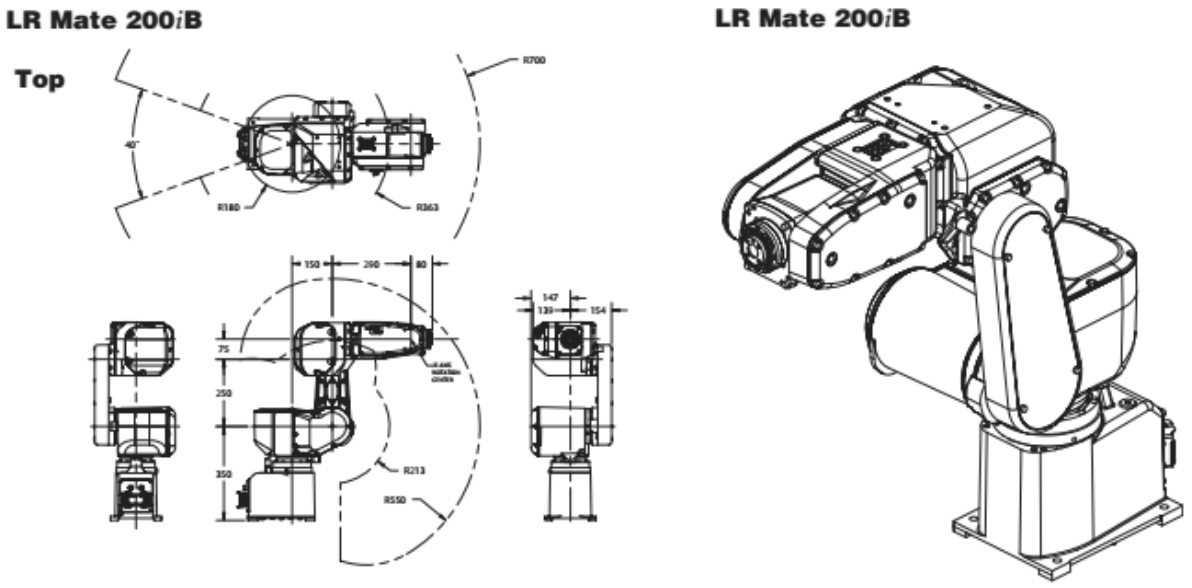

Figure 34: Fanuc LR Mate 200iB work envelope

The size of the heated bed was determined by the span of the widest part of the envelope, plus a few extra inches for good measure. The plane that was determined to grant us the most building space as possible was approximately 6-7” vertically from the plane of the base of the robot. However, at the time of these calculations, the size of the final extruder assembly that was to be mounted onto the tip of the robot arm was unknown, since it hadn’t been fully designed yet. Also, the robot was sitting on a thick aluminum base, and it was unknown whether that would be its final resting place or not. Therefore, an adjustable-level heated bed stand was built to take these uncertainties into account.

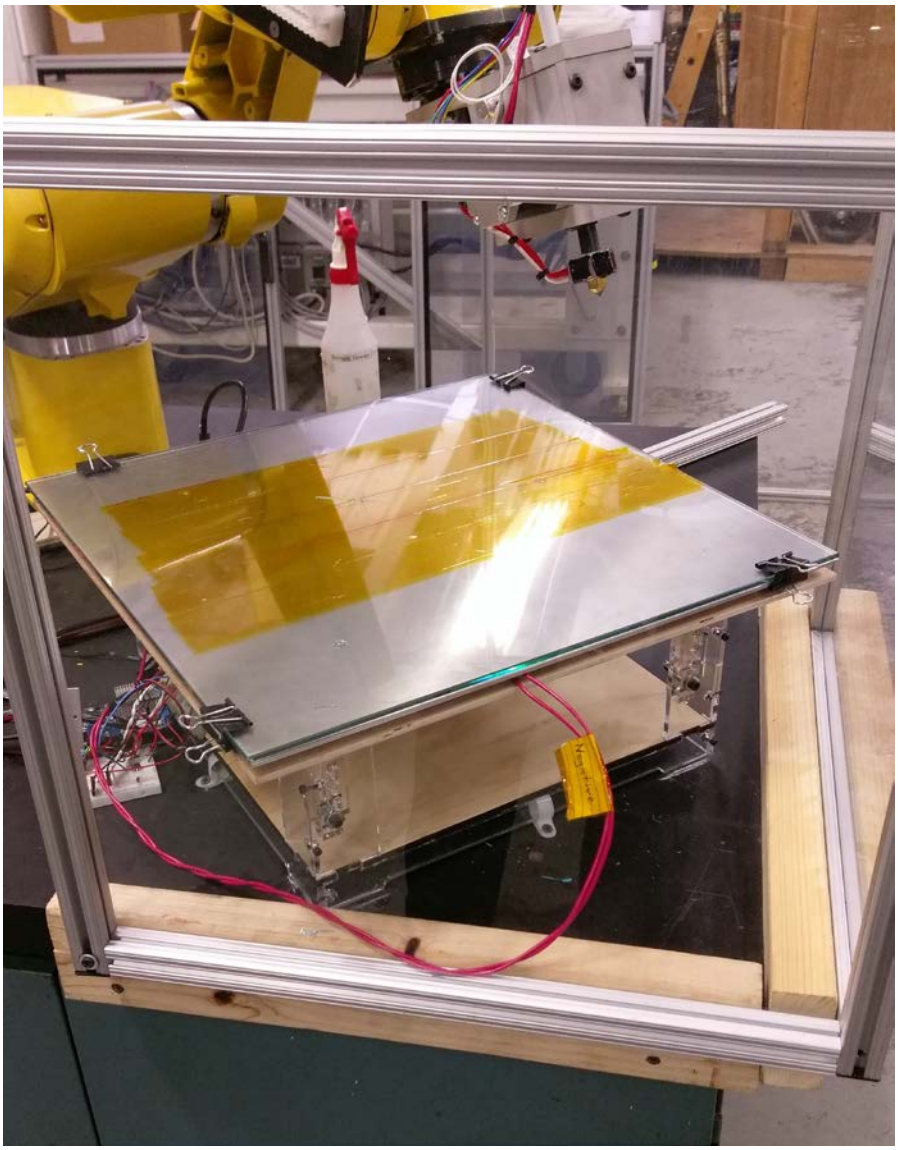

Figure 35: Final setup of heated bed

3.5.4 Power

The heated bed is directly powered by our 24V source. The heated bed does not require any sort of temperature control or current limiting. It takes about ten minutes for the bed to become hot enough for printing.

3.6 System Tests

3.6.1 Test Print #1

Because our system was modular, we were able to start by testing basic functionality before doing a full systems test. Because the robot must be able to accurately follow motion instructions, we tested that functionality first. The robot was loaded with a small test program and the extruder was turned on in order to determine how well the material stuck to the heated bed.

The first test print was a test of the extruder and code functionality. The heated bed was not leveled, and the hot end's temperature was not controlled. As a result, this print did not fully achieve the 5mm 4-sided pyramid shape we were hoping for. However, the extruder was able to extrude, and the robot successfully followed our instructions. To better compare our results we also printed this same file using Washburn Shops' MakerBot Replicator 2X. To visually differentiate the pyramids, our robot printed using blue filament, while the MakerBot printed in red.

Figure 36: Initial Test Print. Red pyramid printed by MakerBot, blue by our system

Having leveled our plate and completed the extruder temperature control circuit, we next attempted to print the same pyramid scaled up by a factor of four. Again, our robot printed using blue filament, while the MakerBot printed in red.

3.6.2 Test Print #2

With the preliminary tests completed, we performed several full system tests. The conditions of the tests were identical so that we could determine how consistent the full printing system was.

Our robot successfully printed what is very clearly a pyramid shape. However the quality is not nearly as high as the MakerBot part. We printed this file six times to determine repeatability. While there were differences between each print, they were consistent overall in terms of size and approximate shape. The robot was able to print each pyramid without interruption or intervention on our part.

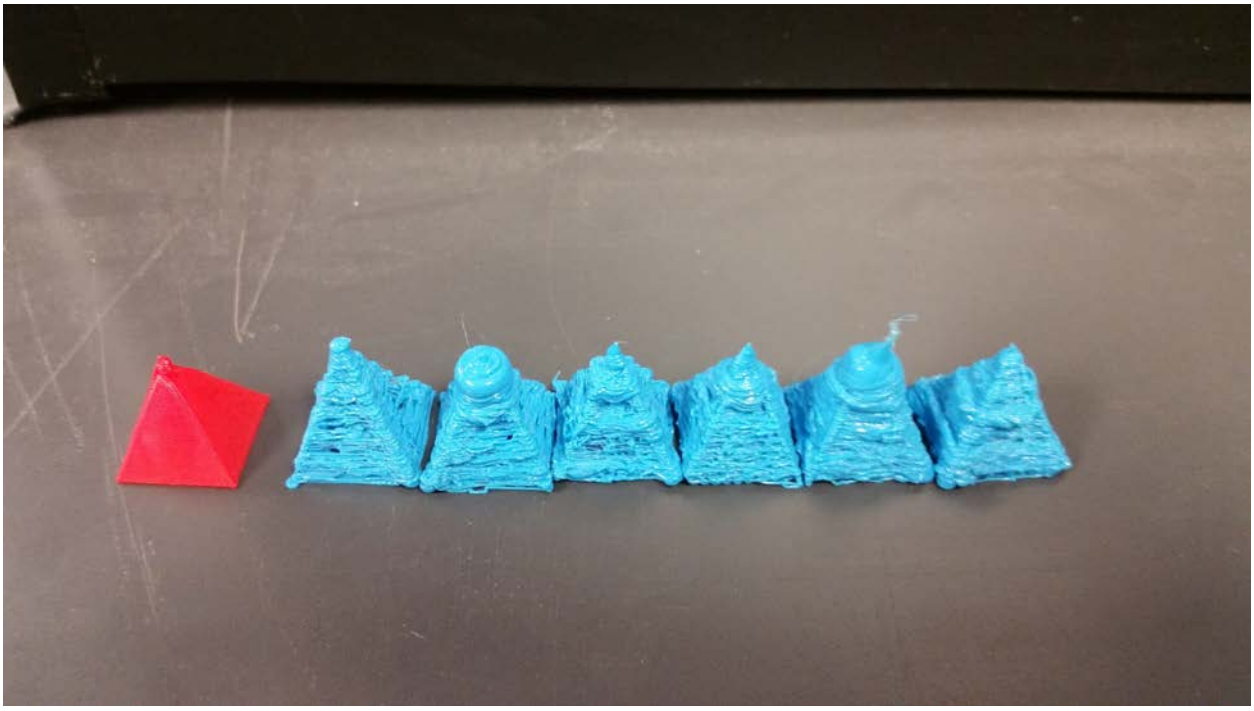

Figure 37: Pyramid Test Prints. Red pyramid printed by MakerBot, blue by our system.

3.7 Discussion of Project Outcomes

Ultimately, the speed of extrusion was responsible for the quality of our parts. The robot program assumes that the hot end extrudes material at a constant rate. However, due to the design of our hobbed shaft collar, the flow rate of material out of the hot end is not within tolerances. We were unable to test whether this would be a problem until it was attached to the robot. To the human eye, the extrusion rate was smooth enough, but in practice it was not. In order to get closer to the necessary extrusion rate more pressure from the idler was needed. Increasing this pressure caused the idler wheel to stop spinning, which increased friction against our material, and further slowed the extrusion rate. Again, this problem did not become apparent until we attempted to print.

Additionally, we came across hardware limitations that prevented us from printing objects much larger than the pyramids. The current memory capacity of the FANUC System R-J3IB Mate controller is less than 1MB. The controller also could not load programs much longer than 500 instructions, even if there was enough hard disk space. Because of this limitation, the G-Code translation program was made to automatically split large programs into sub-routines that were 500 motion instructions long. These subroutines were then called in sequence. This is one of several problems future projects should look to solve.

3.8 Suggestions for Future MQPs

The main goal of future projects should be to refine the extrusion method. As stated earlier, the idler is not driven, and the force required to keep the filament extruding does not allow it to spin freely. This slows down the rate of extrusion. The shaft collar itself also could stand to be improved upon. If the surfaces that connect with the filament were closer together, but still not perfectly circular, it might allow the extrusion to be more constant. Two identical shaft collars, spinning in the opposite direction on either side of the filament would likely supply the necessary constant rate of extrusion. Once this is accomplished, future projects should attempt to find the optimal speed at which to move the robot through its instructions. These changes should vastly improve the robot's print quality.

Future projects should also attempt to find ways of either expanding the robot controller's memory, or finding a way to pass instruction to the robot in real-time through the controller's serial communication interface. This could potentially allow for much larger and more complex prints than even a professional 3D printing product like the MakerBot Replicator

2X. The increased maneuverability theoretically lets it print on angled pre-existing surfaces. This system could be much more flexible than commercial 3D printers.

4. Conclusions

- The rate of extrusion was responsible for the quality of our parts.
- Hardware limitations prevented us from printing large or complicated parts.
- Future MQPs should work to refine the driving mechanism and find the optimal speed at which to move the robot.
- Future MQPs should work to improve the memory of the controller or figure out a method of passing instructions during the print.

References

[1] "Types of 3D Printers or 3D Printing Technologies Overview." *3D Printing from Scratch*. N.p., n.d. Web. <http://3dprintingfromscratch.com/common/types-of-3d-printers-or-3d-printing-technologies-overview/#ebm>.

[2] "3D Printing Processes: The Free Beginner's Guide." *3D Printing Industry*. N.p., n.d. Web. <http://3dprintingindustry.com/3d-printing-basics-free-beginners-guide/processes/>.

[3] "What Is 3D Printing?" *3D Printing*. N.p., n.d. Web. <http://3dprinting.com/what-is-3d-printing/>.

[4] Krassenstein, Brian. "Dutch Engineer Jasper Menger Presents Incredible Robotic Arm 3D Printer With 26 Foot Reach." *3DPrint.com*. 3DR Holdings, 23 Mar. 2015. Web. <http://3dprint.com/52978/robotic-arm-3d-printer/>.

[5] "Over 3D Robot Printing." *3D Robot Printing*. N.p., n.d. Web. <http://www.3d-robotprinting.com/Home/Overons>.

[6] "About." *Dirk Vander Kooij*. N.p., n.d. Web. <http://www.dirkvanderkooij.com/pages/about>.

[7] "Anti-Gravity Object Modeling." Mataerial, n.d. Web. <http://www.mataerial.com/#description>.

[8] Shi, Ji. "ROBOTIC EXTRUSION(6-Axis KUKA+ABS 3D Printing)." Behance, 7 Jan. 2015. Web. <https://www.behance.net/gallery/22536831/ROBOTIC-EXTRUSION%286-Axis-KUKAABS-3D-Printing%29>.

[9] Chalcraft, Emilie. "Stone Spray Robot by Anna Kulik, Inder Shergill and Petr Novikov." Dezeen, 22 Aug. 2012. Web. <http://www.dezeen.com/2012/08/22/stone-spray-robot-by-anna-kulik-inder-shergill-and-petr-novikov/>.

[10] Friedman, Jared. "Woven Clay." *Jared Friedman: Architecture Soup for the Nerdy Soul*. N.p., Dec. 2013. Web. <http://jaredfriedman.org/Woven-Clay>.

[11] "Fabclay." *IAAC Blog*. Institute for Advanced Architecture of Catalonia, 30 May 2012. Web. <http://www.iaacblog.com/programs/fabclay-2/>.

[12] "MX3D Metal." MX3D, n.d. Web. <http://mx3d.com/projects/metal/>.

[13] "Suspended Depositions." NSTRMNT, n.d. Web. <http://nstrmnt.com/#/suspended-depositions/>.

[14] Francois, Jeremie. "Homemade Heated Bed." *3D Printer Improvements*. N.p., 24 Oct. 2012. Web. <http://www.tridimake.com/2012/10/homemade-heated-bed.html>.

Appendices

Appendix A: Drawings

Heated Bed

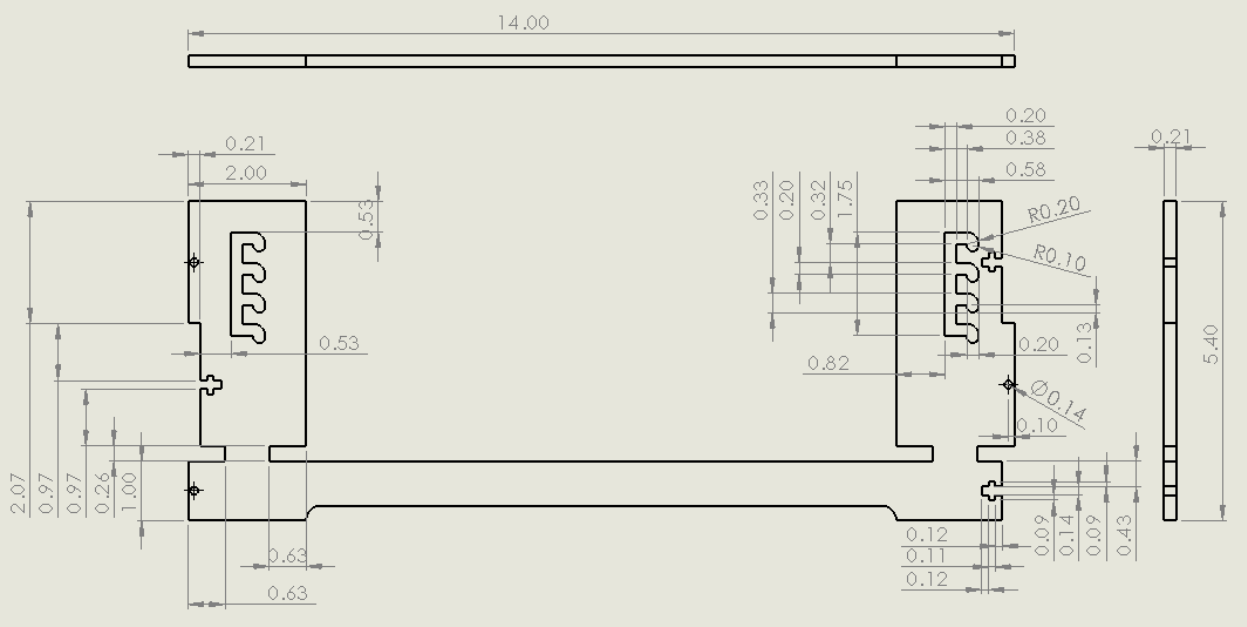

Figure 38: Support measurements

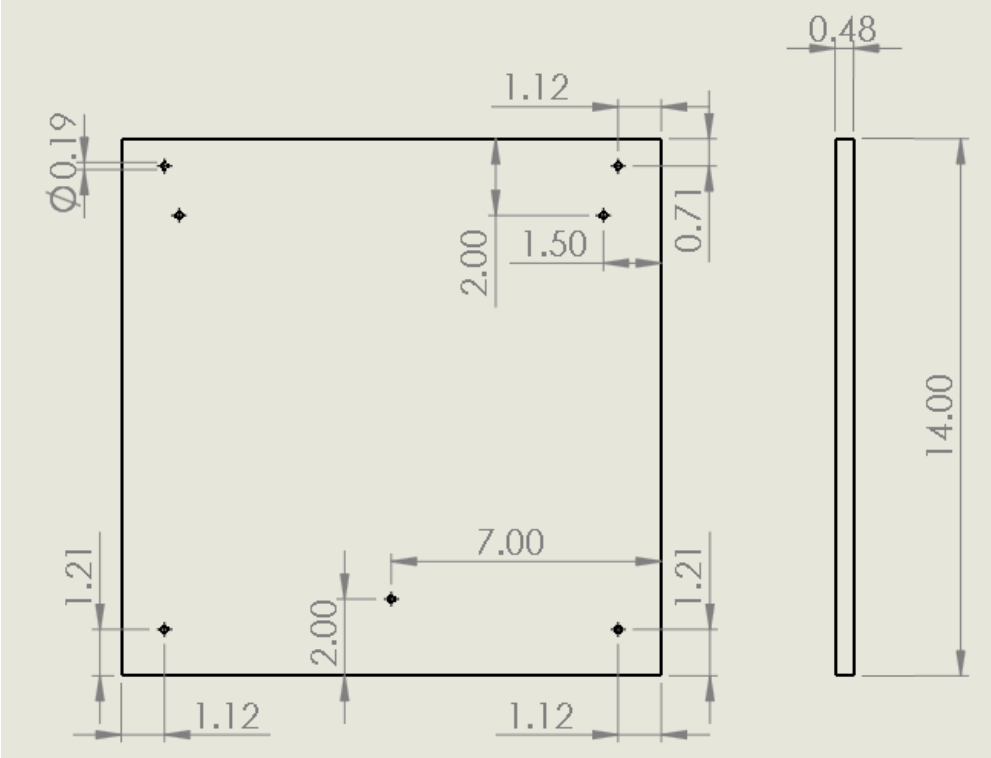

Figure 39: Top measurements

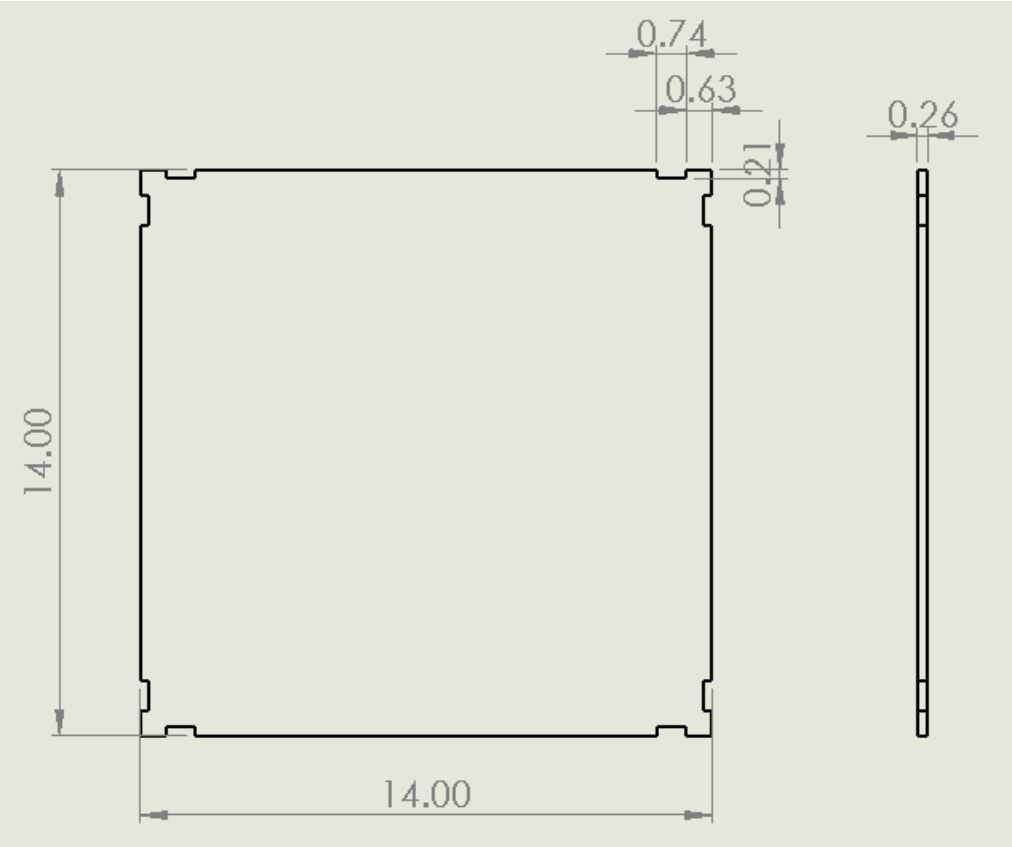

Figure 40: Bottom measurements

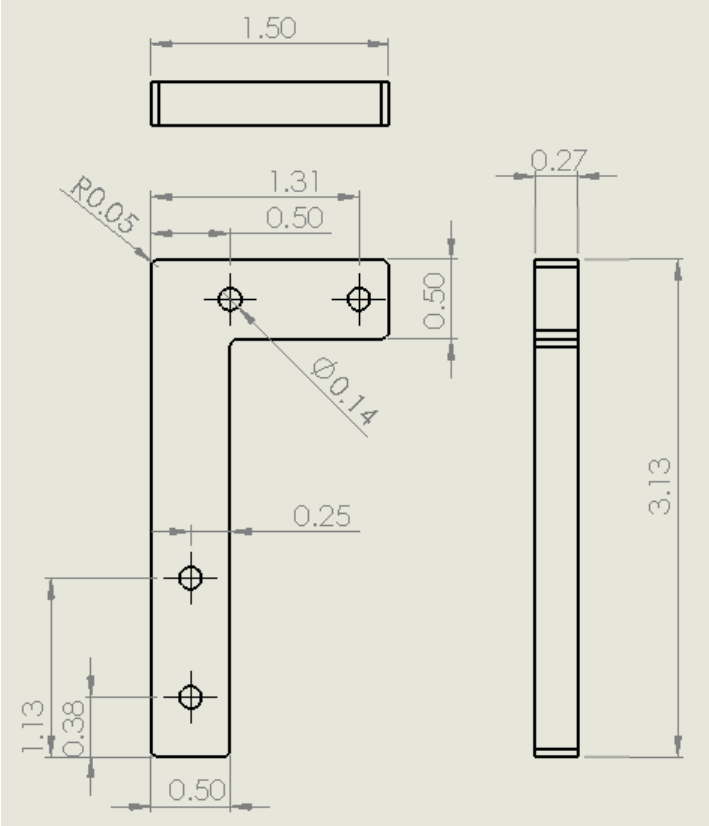

Figure 41: L-bracket part 1 measurements

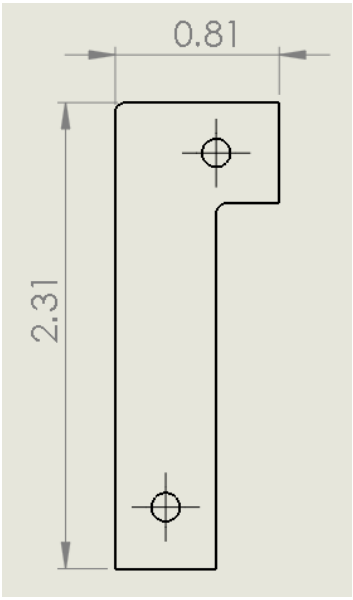

Figure 42: L-bracket part 2 measurements

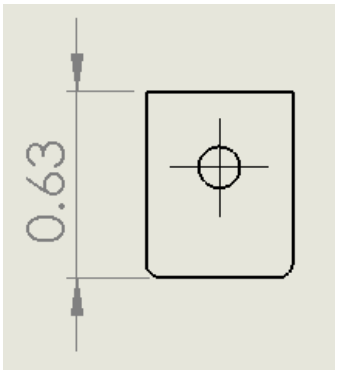

Figure 43: L-bracket part 3 measurements

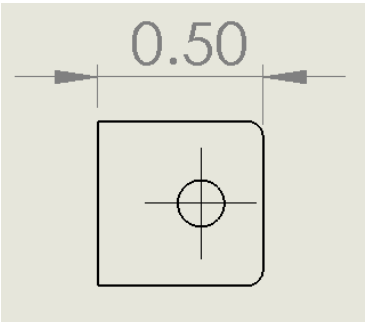

Figure 44: L-bracket part 4 measurements

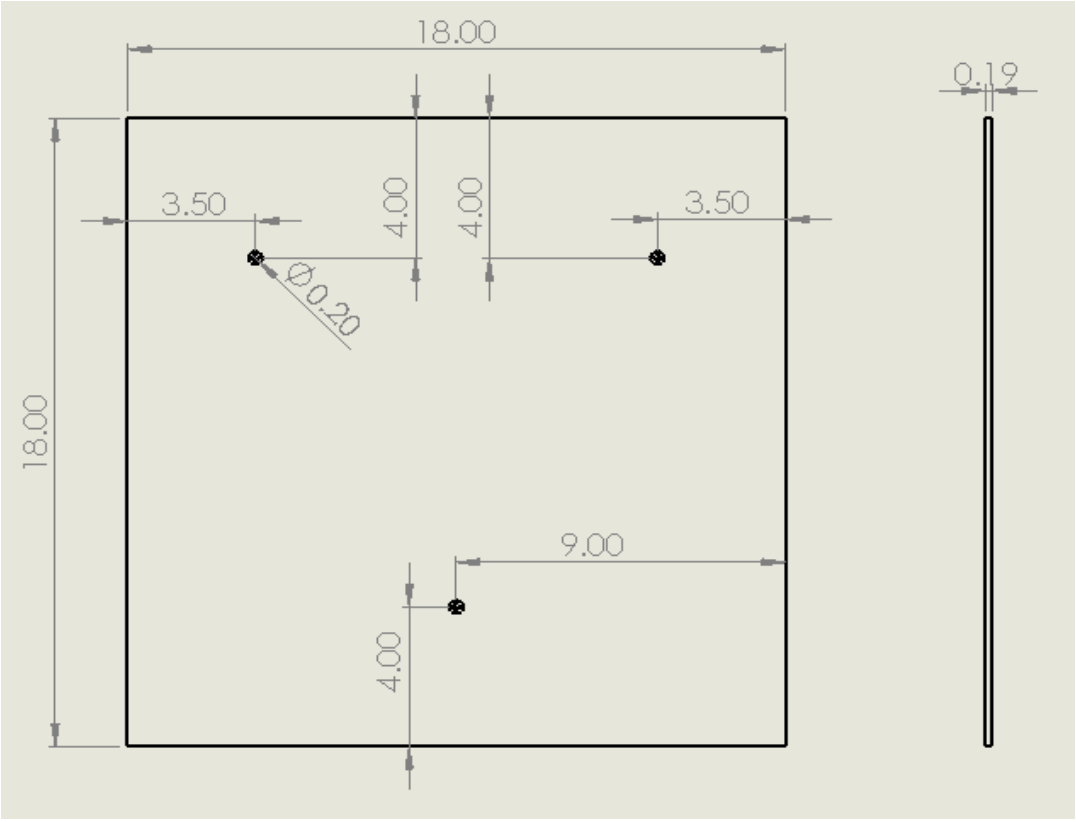

Figure 45: Aluminum measurements

Appendix B: Operator's Manual

The basic process for printing a part is as follows:

1. Turn on power to the heated bed and hot end.
2. Ensure that User Frame 8 is set as the active User Frame Number.
3. After both components reach their ideal temperature, begin execution of printing program and turn on power to the stepper motor shield.
4. Adjust program speed (if necessary) to ensure that enough filament is being laid down by the extruder.
5. When the program completes, turn off all power to the electronics and carefully remove the newly printed part from the build surface.

Appendix C: G-Code Translation and Extruder Control Software

main.java

```
import java.io.*;
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;



public class main{

   public static final String programName = "PBig";
   public static final String inputFileName = "CalibrationPyramid(20mm).gcode";
   public static Parser parser = new Parser();

   /**
    * Opens a file containing gCode.
    * @param path
    * @return
    * @throws IOException
    */
   public static Object[] OpenFile(String path) throws IOException {
       FileReader reader = new FileReader(path);
       BufferedReader buffReader = new BufferedReader(reader);

       return buffReader.lines().toArray();
   }
```

```
    public static void main(String args[]) {
        try {
            Object[] fileContents = OpenFile("src/" + inputFileName);
            int length = fileContents.length;

            PrintWriter writer = new PrintWriter(programName + ".ls", "UTF-8");
            PrintWriter posWriter = new PrintWriter("posOutput.txt", "UTF-8");
            PrintWriter movWriter = new PrintWriter("movOutput.txt", "UTF-8");
            //writer.println("1: PR[1] = P[1];");
            //writer.println("2: UFRAME[8] = PR[1];");
            //writer.println("3: UFRAME_NUM = 8;");

            for (int i = 0; i < length; i++) {
                //System.out.println(fileContents[i]);
                if (i % 500 == 0) {
                    //Periodically write data to file.
                    System.out.println("Working on " + (i+1) + " of " + length);
                    movWriter.print(parser.movements);
                    posWriter.print(parser.positions);
                    parser.movements = "";
                    parser.positions = "";
                }
                parser.parseLine(fileContents[i].toString());
            }
            movWriter.print(parser.movements);
            posWriter.print(parser.positions);
            parser.movements = "";
            parser.positions = "";

            movWriter.close();
            posWriter.close();

            Object[] movContents = OpenFile("movOutput.txt");

            System.out.println("Length: " + movContents.length);
            int movLength = movContents.length;
            int num = 0;
            ArrayList<String> endPos = new ArrayList<String>();

            writer.println("/PROG " + programName);
            writer.println("/ATTR \n/MN");

            for (int i = 0; i < movLength; i++) {
                /*if(i % 1000 == 0){
                    Pattern pointPattern = Pattern.compile("(.*)(P\\[(\\d+)\\])(.*)");
                    Matcher pointMatcher = pointPattern.matcher((String)
movContents[i]);

                    if(pointMatcher.matches()){
                        endPos.add(pointMatcher.group(3));
                    }
                    writer = new PrintWriter(programName + num + ".txt", "UTF-8");
```

```
                writer.println("/PROG " + programName + num);
                writer.println("/ATTR \n/MN");
                num++;
            }*/

            //Write motion instructions to file.
            synchronized (fileContents) {
                writer.println((String) movContents[i]);
            }
        }

        writer.println("/POS");

        //Set up the robot frame this program will use.
        writer.println("P[1]{ \n" +
                "  GP1:\n" +
                "  UF:F, UT:F,\n" +
                "  CONFIG: 'N U T, , 0, 0',\n" +
                "   X = -900.0mm, Y = 0.0mm, Z = 0.0mm, W = -180.000 deg, P = 0.000
deg, R = 0.000 deg \n" +
                " };");
        Object[] posContents = OpenFile("posOutput.txt");
        int posLength = posContents.length;
        num = 0;
        for (int i = 0; i < posLength; i++) {
            /*if(!endPos.isEmpty()) {
                if (i % Integer.parseInt(endPos.get(0)) == 0) {
                    //writer = new PrintWriter(new FileOutputStream(new
File(programName + num + ".txt"), true /* append = true *///));
                    //Write the position data to file.
                    //synchronized (fileContents) {
                        writer.println((String) posContents[i]);
                    //}
                    //endPos.remove(0);
                //}
            //}
        }

        writer.println("/END.");
        posWriter.close();
        movWriter.close();
        writer.close();

        FileSplitter splitter = new FileSplitter(programName, movContents,
posContents);
        splitter.split();

    }catch (IOException error){
        System.out.println("Could not open file. " + error);
    }

  }
}
```

Parser.java

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
* Created by Will on 10/29/15.
*/
public class Parser {

   boolean extruderOn = false;
   int lineNumber = 1; //Starting line of motion instructions.

   Pattern g1Patten = Pattern.compile("(G0 |G1 )(.*)");
   Pattern g2Pattern = Pattern.compile("(G2 )(.*)");
   Pattern g3Pattern = Pattern.compile("(G3 )(.*)");

   Pattern xPattern = Pattern.compile("(.*)X(-?)\\d+(\\.?)\\d*(.*)");
   Pattern yPattern = Pattern.compile("(.*)Y(-?)\\d+(\\.?)\\d*(.*)");
   Pattern zPattern = Pattern.compile("(.*)Z(-?)\\d+(\\.?)\\d*(.*)");
   Pattern ePattern = Pattern.compile("(.*)E(-?)\\d+(\\.?)\\d*(.*)");
   Pattern fPattern = Pattern.compile("(.*)F(-?)\\d+(\\.?)\\d*(.*)");

   Pattern iPattern = Pattern.compile("(.*)I(-?)\\d+(\\.?)\\d*(.*)");
   Pattern jPattern = Pattern.compile("(.*)J(-?)\\d+(\\.?)\\d*(.*)");

   public static float lastXPos = 0;
   public static float lastYPos = 0;
   public static float lastZPos = 0;
   public static int lastFeedrate = 0;
   public static float lastExtruderValue = 0;
   public static int pointCount = 2; //Point 1 sets the robot frame.

   public static int iterations = 0;
   public static int inRange = 0;

   float lastTestX = 0;
   float lastTestY = 0;

   public static String movements = "";
   public static String positions = "";

   /**
    * Determines if the input is a linear or arc motion. Ignores all other commands.
    * @param input
    */
   public void parseLine(String input){
       Matcher g1Matcher = g1Patten.matcher(input);
       Matcher g2Matcher = g2Pattern.matcher(input);
       Matcher g3Matcher = g3Pattern.matcher(input);
```

```
    if(g1Matcher.matches()){
        G1Parser(input);
    }else if (g2Matcher.matches()){
        arcParser(input, true);
    }else if(g3Matcher.matches()){
        arcParser(input, false);
    }
}

/**
 * Parse linear movement instruction.
 * @param input
 */
public void G1Parser(String input){
    float xPos = 0;
    float yPos = 0;
    float zPos = 0;
    float extruderVal = 0;
    int feedrate = 0;

    Matcher xMatcher = xPattern.matcher(input);
    Matcher yMatcher = yPattern.matcher(input);
    Matcher zMatcher = zPattern.matcher(input);
    Matcher eMatcher = ePattern.matcher(input);
    Matcher fMatcher = fPattern.matcher(input);
    if(xMatcher.matches()){
        int start = xMatcher.end(1)+1;
        int end = xMatcher.start(4);

        //System.out.println("Substring: " + input.substring(start, end));
        xPos = Float.parseFloat(input.substring(start, end));
        lastXPos = xPos;
    }else{
        xPos = lastXPos;
    }

    if(yMatcher.matches()){
        int start = yMatcher.end(1)+1;
        int end = yMatcher.start(4);

        yPos = Float.parseFloat(input.substring(start, end));
        lastYPos = yPos;
    }else{
        yPos = lastYPos;
    }

    if(zMatcher.matches()){
        int start = zMatcher.end(1)+1;
        int end = zMatcher.start(4);

        zPos = Float.parseFloat(input.substring(start, end));
        lastZPos = zPos;
    }else{
```

```
            zPos = lastZPos;
        }

        if(eMatcher.matches()){
            int start = eMatcher.end(1)+1;
            int end = eMatcher.start(3);

            extruderVal = Float.parseFloat(input.substring(start, end));
            lastExtruderValue = extruderVal;
        }else{
            extruderVal = 0;
        }

        if(fMatcher.matches()){
            int start = fMatcher.end(1)+1;
            int end = fMatcher.start(3);



            feedrate = (int) Float.parseFloat(input.substring(start, end));
            lastFeedrate = feedrate;
        }else{
            feedrate = lastFeedrate;
        }

        String posString;
        String movementString = "";

        //System.out.println("X: " + xPos);

        posString = "P[" + pointCount +"]{ \n GP1:\n UF:8, UT:F,\n CONFIG: 'N U T, , 0,
0',\n " +
                " X = "+Float.toString(xPos) + "mm," +
                " Y = "+Float.toString(yPos) + "mm," +
                " Z = "+Float.toString(zPos) + "mm," +
                " W = 180.000 deg, P = 0.000 deg, R = 0.000 deg \n };";

        if(extruderVal == 0){
            if(extruderOn) {
                movementString = lineNumber + ": DO[101]=OFF;\n";
                extruderOn = false;
                lineNumber++;
            }
        }else{
            if(!extruderOn){
                movementString = lineNumber + ": DO[101]=ON;\n";
                extruderOn = true;
                lineNumber++;
            }
        }

        movementString = movementString + lineNumber + ": L P[" + pointCount +"] " +
Integer.toString(feedrate) + "mm/sec FINE;";
```

```
        pointCount++;
        lineNumber++;

        movements = movements.concat(movementString + "\n");
        positions = positions.concat(posString + "\n");

        double distance = Math.sqrt(Math.pow((xPos-lastTestX), 2)+Math.pow((yPos-
lastTestY),2));
        double ratio = distance/(double) extruderVal;
        double dist = xPos-lastTestX;
        //System.out.println("Ratio: " + ratio + " " + inRange + "/" + iterations);

        if (ratio > 35 && ratio < 37){
            inRange++;
        }
        iterations++;
        lastTestX = lastXPos;
        lastTestY = lastYPos;
    }

    /**
     * Arc parser.
     * @param input
     * @param clockwise
     */
    public void arcParser(String input, boolean clockwise){
        float xPos = 0;
        float yPos = 0;
        float iPos = 0;
        float jPos = 0;

        float extruderVal = 0;
        int feedrate = 0;

        Matcher xMatcher = xPattern.matcher(input);
        Matcher yMatcher = yPattern.matcher(input);
        Matcher zMatcher = zPattern.matcher(input);
        Matcher iMatcher = iPattern.matcher(input);
        Matcher jMatcher = jPattern.matcher(input);
        Matcher eMatcher = ePattern.matcher(input);
        Matcher fMatcher = fPattern.matcher(input);



        if(xMatcher.matches()){
            int start = xMatcher.end(1)+1;
            int end = xMatcher.start(4);

            xPos = Float.parseFloat(input.substring(start, end));

        }else{
            xPos = lastXPos;
        }
```

```
if(yMatcher.matches()){
    int start = yMatcher.end(1)+1;
    int end = yMatcher.start(4);

    yPos = Float.parseFloat(input.substring(start, end));

}else{
    yPos = lastYPos;
}

if(iMatcher.matches()){
    int start = iMatcher.end(1)+1;
    int end = iMatcher.start(4);

    iPos = Float.parseFloat(input.substring(start, end));
}

if(jMatcher.matches()){
    int start = jMatcher.end(1)+1;
    int end = jMatcher.start(4);

    jPos = Float.parseFloat(input.substring(start, end));
}

if(eMatcher.matches()){
    int start = eMatcher.end(1)+1;
    int end = eMatcher.start(3);

    extruderVal = Float.parseFloat(input.substring(start, end));
    lastExtruderValue = extruderVal;
}else{
    extruderVal = 0;
}

if(fMatcher.matches()){
    int start = fMatcher.end(1)+1;
    int end = fMatcher.start(3);


    feedrate = (int) Float.parseFloat(input.substring(start, end));
    lastFeedrate = feedrate;
}else{
    feedrate = lastFeedrate;
}

String posString;
String movementString = "";

float thruXPos = 0;
float thruYPos = 0;
```

```
        Point radiusPoint = new Point(iPos, jPos);
        Point startPoint = new Point(lastXPos, lastYPos);
        Point endPoint = new Point(xPos, yPos);

        float radius = distance(lastXPos, lastYPos, iPos, jPos);

        float startTheta = radiusPoint.getAngle(startPoint);
        float endTheta = radiusPoint.getAngle(endPoint);

        System.out.println("Radius: " + radius);
        System.out.println("Start point: " + startPoint.x + ", " + startPoint.y + "End
point: " + endPoint.x + ", " + endPoint.y);
        System.out.println("Start theta: " + startTheta + "End theta: " + endTheta);

        float newTheta = 0;

        if(clockwise) {
            if (endTheta < startTheta) {
                //Does not reset theta
                newTheta = startTheta - ((startTheta - endTheta) / 2);
            } else {
                //Does reset theta
                newTheta = (startTheta + 360) - (((startTheta + 360) - endTheta) / 2);
            }
        }else{
            if (endTheta < startTheta) {
                //Does not reset theta
                newTheta = endTheta - ((startTheta - endTheta) / 2);
            } else {
                //Does reset theta
                newTheta = (endTheta + 360) - (((startTheta + 360) - endTheta) / 2);
            }
        }

        thruXPos = radius*(float)Math.cos((Math.toRadians(newTheta)));
        thruYPos = radius*(float)Math.sin(Math.toRadians(newTheta));

        //System.out.println("x: " + xPos + " y: " + yPos + " i: " + iPos + " j: " +
jPos);
        //System.out.println("X: " + thruXPos + " Y: " + thruYPos + " Radius: " +
radius + " New Theta: " + newTheta);

        assert distance(xPos, yPos, iPos, jPos) ==  distance(lastXPos, lastYPos, iPos,
jPos);
        int pointCount1 = pointCount + 1;

        posString = "P[" + pointCount +"]{ \n GP1:\n UF:8, UT:F,\n CONFIG: 'N U T, , 0,
0',\n " +
                " X = "+ String.format("%.4f", thruXPos) + "mm," +
                " Y = "+ String.format("%.4f", thruYPos) + "mm," +
                " Z = "+Float.toString(lastZPos) + "mm," +
```

```
                " W = 180.000 deg, P = 0.000 deg, R = 0.000 deg \n };\n" +

                "P[" + (pointCount1) +"]{ \n GP1:\n UF:8, UT:F,\n CONFIG: 'N U T, , 0,
0',\n " +
                " X = "+Float.toString(xPos) + "mm," +
                " Y = "+Float.toString(yPos) + "mm," +
                " Z = "+Float.toString(lastZPos) + "mm," +
                " W = 180.000 deg, P = 0.000 deg, R = 0.000 deg \n };";

        if(extruderVal == 0){
            if(extruderOn) {
                movementString = lineNumber + ": DO[101]=OFF;\n";
                extruderOn = false;
                lineNumber++;
            }
        }else{
            if(!extruderOn){
                movementString = lineNumber + ": DO[101]=ON;\n";
                extruderOn = true;
                lineNumber++;
            }
        }

        movementString = movementString + lineNumber + ": C P[" + pointCount +"] \n" +
"P["+ pointCount1 +"]" + Integer.toString(feedrate) + "mm/sec FINE;\n";
        pointCount += 2;
        lineNumber++;

        movements = movements.concat(movementString);
        positions = positions.concat(posString);

        lastXPos = xPos;
        lastYPos = yPos;

        double distance = Math.sqrt(Math.pow((xPos-lastTestX), 2)+Math.pow((yPos-
lastTestY),2));
        double ratio = distance/(double) extruderVal;
        double dist = xPos-lastTestX;

        //System.out.println("Ratio: " + ratio + " " + inRange + "/" + iterations);

    }

    /**
     * Returns the distance between the points (x,y) and (i,j)
     * @param x
     * @param y
     * @param i
     * @param j
     * @return
     */
    public float distance(float x, float y, float i, float j){
         double result = Math.sqrt((Math.pow(i-x, 2) +  Math.pow(j-y, 2)));
```

```
        return (float) result;
    }

    /**
     * Returns true if the input begins with "G1".
     * @param input
     * @return
     */
    public boolean matchesString(String input){
        String pattern = "G1(.*)";
        Pattern p = Pattern.compile(pattern);
        Matcher m = p.matcher(input);
        return m.matches();

    }

}
```

Point.java

```
/**
* Created by Will on 11/10/15.
*/
public class Point {

    float x;
    float y;

    Point(float x, float y){
        this.x = x;
        this.y = y;
    }
    /**
     * Returns degrees.
     * @param target
     * @return
     */
    public float getAngle(Point target) {
        float angle = (float) Math.toDegrees(Math.atan2(target.y - y, target.x - x));

        if(angle < 0){
            angle += 360;
        }

        return angle;
    }
}
```

FileSplitter.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Objects;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
* Created by Will on 12/10/15.
*/
public class FileSplitter {
   String programName;
   Object[] movements;
   Object[] positions;

   ArrayList<ArrayList<Object>> movementBlocks;
   ArrayList<ArrayList<Object>> posistionBlocks;

   FileSplitter(String programName, Object[] movements, Object[] positions) {
       this.programName = programName;
       this.movements = movements;
       this.positions = positions;
   }

   public void split() {
       ArrayList<String> endPos = new ArrayList<String>();

       movementBlocks = new ArrayList<ArrayList<Object>>();
       int num = -1;
       for (int i = 0; i < movements.length; i++) {
           if (i % 500 == 0) {
               Pattern pointPattern = Pattern.compile("(.*)(P\\[(\\d+)\\])(.*)");
               Matcher pointMatcher = pointPattern.matcher((String) movements[i]);

               if (pointMatcher.matches() && i != 0) {
                   endPos.add(pointMatcher.group(3));
               }
               movementBlocks.add(new ArrayList<Object>());
               num++;
           }
           movementBlocks.get(num).add(movements[i]);
       }

       //Add newline characters to the end of the movement strings.
       String[] movementString = new String[movementBlocks.size()];
       for (int i = 0; i < movementBlocks.size(); i++) {
           movementString[i] = new String("");
           for (int j = 0; j < movementBlocks.get(i).size(); j++) {
```

```
                movementString[i] =
movementString[i].concat(movementBlocks.get(i).get(j) + "\n");
                //System.out.println(movementString[i].toString());
            }
        }

        num = 0;
        for (int i = 0; i < movementString.length; i++) {
            PrintWriter writer = null;
            try {
                //Write the position data to file.
                writer = new PrintWriter(new FileOutputStream(new File(programName +
num + ".ls"), true));

                writer.print("/PROG "  + programName + num +
                        "\n/ATTR\n" +
                        "/MN\n");
                num++;
                //System.out.print("Printing: " + movementString[i]);
                //writer.print(movementString[0]);
                writer.write(movementString[i]);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
            writer.close();

        }
        String newString = "";
        for (int i = 0; i < positions.length; i++) {
            newString = newString + ((String) positions[i]);
        }

        positions = newString.split(";");

        posistionBlocks = new ArrayList<ArrayList<Object>>();
        int posNum = 0;
        posistionBlocks.add(new ArrayList<Object>());
        for (int i = 0; i < positions.length; i++) {
            if (endPos.size() <= posNum) {
                posistionBlocks.get(posNum).add(positions[i] + ";");
            } else {
                if (i == (Integer.parseInt(endPos.get(posNum)))-2) {
                    posistionBlocks.add(new ArrayList<Object>());
                    System.out.println("End Pos: " +
(Integer.parseInt(endPos.get(posNum))-2));
                    posNum++;
                }
                posistionBlocks.get(posNum).add(positions[i] + ";");
            }
        }
        System.out.print("Block Length: " + posistionBlocks.size());
        String[] positionString = new String[posistionBlocks.size()];
        //System.out.println("\nThis is here: " +
movementBlocks.get(0).get(7).toString());
```

```
        for (int i = 0; i < posistionBlocks.size(); i++) {
            positionString[i] = new String("");
            for (int j = 0; j < posistionBlocks.get(i).size(); j++) {
                positionString[i] =
positionString[i].concat(posistionBlocks.get(i).get(j) + "\n");
                //System.out.println(movementString[i].toString());
            }
        }

        num = 0;
        System.out.println("Length: " + positions.length);



        //System.out.print("First pos: " + positions[1].toString());
        for (int i = 0; i < positionString.length; i++) {
            PrintWriter writer = null;
            try {
                //Write the position data to file.
                writer = new PrintWriter(new FileOutputStream(new File(programName +
num + ".ls"), true));
                num++;

                //System.out.print("Printing: " + movementString[i]);
                //writer.print(movementString[0]);
                writer.print("/POS\n");
                writer.write(positionString[i]);
                writer.write("/END.");
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
            writer.close();

            /*for (int k = 0; k < endPos.size(); k++){
                System.out.println("End pos: " + endPos.get(k));
            }*/
        }

    }

    public static int getLineCount(String text) {
        return text.split("[\n|\r]").length;
    }

}
```

MQP_Printer_Controller.ino

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
//#include "utility/Adafruit_PWMServoDriver.h"
```

```
#define EXTRUDER_SPEED 90 // 60 rpm
#define EXTRUDER_PIN 5
#define HOT_END_PIN 2
#define THERMISTOR_PIN A0

#define THERMISTOR_CUTOFF 1020



// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// to motor port #2 (M3 and M4)
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 1);



void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);           // set up Serial library at 9600 bps
  AFMS.begin();  // create with the default frequency 1.6KHz
  //AFMS.begin(1000);  // OR with a different frequency, say 1KHz

  pinMode(EXTRUDER_PIN, INPUT);
  pinMode(HOT_END_PIN, OUTPUT);
  pinMode(THERMISTOR_PIN, INPUT);

  myMotor->setSpeed(EXTRUDER_SPEED);
}

void loop() {
  myMotor->step(3, FORWARD, INTERLEAVE);
    if(analogRead(THERMISTOR_PIN) > THERMISTOR_CUTOFF){
      digitalWrite(HOT_END_PIN, LOW);
    }else{
      digitalWrite(HOT_END_PIN, HIGH);
    }
  /*
    if(digitalRead(EXTRUDER_PIN) == HIGH){
      myMotor->step(2, FORWARD, INTERLEAVE);
    }else{
      myMotor->step(0, FORWARD, INTERLEAVE);
    }*/
    int sensorValue = analogRead(A0);
    // print out the value you read:
    Serial.print(sensorValue);
    Serial.print("     ");
    Serial.println(digitalRead(HOT_END_PIN));

    delay(20);
}
```