# Investigation of Additive Manufacturing Using Industrial Robots

Matt Heuser
University of Texas at Arlington
Department of Mechanical Engineering
May 2016

Supervising Professor:
Panos Shiakolas, Ph.D.

Submitted in partial fulfillment of the requirements for
the Master of Science degree at the University of Texas at Arlington

**Acknowledgments**

I would like to thank my supervising professor, Panos Shiakolas, Ph.D., whose encouragement, guidance, and support was indispensable in this research.

I would like to thank Pranesh Aswath, Ph.D. and Tre Welch, Ph.D. for serving as committee members at my thesis defense.

I am grateful for the help and support received from my colleagues from the MARS research team especially Christopher Abrego, Apoorv Patwardhan, Prashanth Ravi, and Tushar Saini.

I would like to thank Kermit Beird and Sam Williams from the UTA machine shop for machining the mechanical parts necessary to make this research possible.

I would also like to thank Christopher McMurrough, Ph.D. for his assistance with stepper motors and drivers.

Finally, I would like to thank Hyejin Moon, Ph.D for lending the use of a microscope for measurement and imaging purposes.

**Abstract**

Conventional additive manufacturing platforms are controlled using G-code, a numerical control language used to define the process parameters and movement of the print head. First, a digital model of the desired shape is created using 3-D computer graphics software. Then the model is analyzed by software tools that generate the necessary G-code instructions to be used by a 3-D printer to produce the desired object.

The goal of this research is to lay the foundation and develop the necessary tools to 3-D print custom objects based on the viscous extrusion process using industrial robots. The first task is to process a set of G-code instructions and execute the coded operations with an industrial robot. This requires the coordination of the motion of the robot and the viscous extruder actuation.

In this research, an Adept Python robot controlled using the Adept V+ language was employed along with two stepper-based custom viscous extrusion modules controlled by an Arduino board. A methodology was developed to transform the a set of G-code instructions to the V+ language while considering the coordinated motion between the robot and the extruder module. A customized stepper motor control circuit and software were developed to control the extruder motors. The developed software and hardware tools were successfully implemented and demonstrated using viscous extrusion modules to fabricate a multi-material and multi-layer scaffold.

# Contents

# List of Figures

# 1 Introduction

Conventional 3-D printers are programmed using a numerical control language called G-code. The general methodology used for these printers is to create a 3-D model of the object to be printed using software modeling tools such as SolidWorks [11]. The model is then exported to a mesh format called Standard Tessellation Language (STL). The STL file is then analyzed using a software package such as Slic3r to generate the tool paths to perform the necessary additive manufacturing operations to construct the object [10]. These tool paths are obtained as the output from Slic3r and are readable by the 3-D printer firmware.

## 1.1 Importance

The purpose of this work is to investigate the possibility of performing additive manufacturing tasks using industrial robots. A plunger-based viscous extrusion module was used as the end-effector of the robot in order to perform controlled viscous extrusion tasks. This work was demonstrated using a three-axis Cartesian Adept robot. However, the control language used by this particular robot is shared by all Adept robots, including six-degree-of-freedom robots that provide arbitrary orientation capability. Robots made by other manufacturers have different syntax requirements but are similar in their general programming capabilities. As such, it is possible to use the tools developed in this research with other industrial robotic platforms with minimal modifications.

The large workspace of an industrial robot allows for the fabrication of larger structures than those which are possible with a conventional 3-D printer. It also allows for the dispensation viscous materials on a pre-existing large structure. The arbitrary orientation capabilities of a six-degree-of-freedom robot make it possible to follow the contours of an existing object.

## 1.2 Literature Survey

A research group at ETH Zurich investigated the robotic fabrication process on a large scale in 2013. They used an articulated six-degree-of-freedom robot to fabricate the concrete forms necessary to construct high-rise buildings. They also experimented with robot fabrication of polymer-based concrete reinforcement structures. [14]

In 2012, research was conducted by a group at Loughborough University, investigating construction-scale additive manufacturing processes. Using a Cartesian robot, they were able to fabricate structures by extruding the cement itself. Their work was demonstrated using a 9-20 mm nozzle with a 6-25 mm layer height. [18]

Another research group investigated mobile robotic fabrication on construction sites. This group utilized an ABB robotic unit mounted on a mobile platform. The particular robot used was a six-degree-of-freedom articulated robot. The goal of their work was to make it easier for construction workers to use robots to perform repetitive tasks. A pick-and-place methodology was used for assembly in such applications as the construction of block retaining walls. [15]

A British automotive manufacturer, Aston Martin, used industrial robots for the adhesive bonding of car body parts. Specifically, a six-degree-of-freedom robot was used to apply the adhesives. The robot used for this application on the Aston Martin DB9 was a Kawasaki ZX130L. [20]

In 2013, research was conducted at the Massachusetts Institute of Technology using a multi-functional robotic platform for digital design and fabrication. These scientists utilized a KUKA six-degree-of-freedom robot to explore the integration of additive, formative, and subtractive fabrication methods. They substituted quick connect disconnect (QCD) modules on the end-effector of the robot to accommodate the tools necessary for each type of fabrication. [17]

A research group at the National University of Singapore in 2002 conducted re-

search in fabrication using a robotic dispensing system. They used a Cartesian robot with a pneumatic dispenser to extrude a viscous material. They were able to create bio-compatible scaffolds useful for cell seeding using hydroxyapatite (HA). [12]

Scientists at Dankook University in 2009 also experimented with bio-compatible scaffolds for bone tissue engineering. They also used a Cartesian robot to dispense viscous material. Using a nozzle diameter of 0.520 mm, they successfully fabricated 3-D porous scaffolds from a solution of polycaprolactone (PCL) and hydroxyapatite (HA). [16]

Research has been conducted in 2013 at the Michigan Technological University in additive manufacturing using a metallic material. The purpose of this research was to produce a low-cost open-source metal 3-D printer. This was accomplished using a gas-metal arc welder as the end-effector of a RepRap deltabot 3-D printing platform. They were able to successfully demonstrate a metal additive manufacturing platform which could be built for under $2000. [13]

Scientists at the University of Technology and Life Sciences in Poland have conducted research in robotic surface finishing. Their experiments were performed using a six-degree-of-freedom ABB robotic platform. They began with a CAD model of the surface of the object to be finished and used software called IRBCAM to convert G-code tool paths to the ABB robot language for subtractive manufacturing [5]. [19]

Research has also been conducted in robotic surface finishing at the University of Texas at Arlington in 1999. This research group developed a CAD-based robot path and process planning environment for surface finishing called RobSurf. This system was capable of generating a CAD model of an object through a measuring device and reverse engineering techniques. The native robot code to perform the task was then generated from the model. This research was demonstrated using metal and fiberglass composite surfaces. [21]

## 1.3 Methodology

A common approach used for 3-D printing is to create a digital model of the object to be printed using various software tools. Then, a software preprocessor is used to convert the model into a series of commands that the printer follows to produce the physical object. These commands contain information about the speed and coordinates for Cartesian motion, as well as process parameters such as the amount of filament to be extruded and the extruder temperature for polymeric extrusion. The printer firmware then coordinates the motion of the platform with the filament feed rate to produce the desired paths.

G-code is a language widely used for sending instructions to a 3-D printer. The firmware for most conventional 3-D printers is capable of interpreting G-code for their operations. Many software tools exist, such as SolidWorks or FreeCAD, to create a solid model of a desired object. Other software tools, such as Slic3r, are then used to analyze the model and produce a set of G-code commands that can be used to print the object.

The Adept Python is a three-degree-of-freedom Cartesian industrial robot. It is programmed and controlled using a proprietary programming language called V+ which was developed by Adept [6]. The Python controller is unable to interpret the G-code used by most 3-D printers.

In order for the Python to perform the functions of a 3-D printer, the G-code script must be translated to the V+ language. It is necessary to create a program that automates this porting process. For this purpose, a *Python* script was created which parses each G-code command and produces equivalent commands in the V+ language. These commands interface with external hardware that controls the print head in a coordinated fashion.

# 2 Tools Employed

Many hardware and software tools were employed to perform viscous extrusion operations. The robot itself provided the motion platform for the operations while the plungers were actuated by stepper motors. Proprietary software provided by Adept was used to control the robot, while an Arduino with an open source library was used for stepper motor control. Communication for these two platforms was performed through the digital outputs from the robot controller. For larger programs, the Trivial File Transfer Protocol was utilized to load programs onto the robot controller.

## 2.1 Hardware Tools Employed

### 2.1.1 Robot Platform

The robot platform used to verify and demonstrate this work was the Adept Python shown in figure 1. The Python can achieve a maximum speed of 1450 mm/sec. It has a workspace of 800 mm by 300 mm by 100 mm and a repeatability of $\pm$ 0.01 mm.

The Python was chosen as the demonstration platform for this work because it is a relatively safe environment. This safe environment is especially important during the development phase of the transformations in order to avoid damage to the robot if an unsafe command is generated. The only way for the robot to be damaged is a collision with the bed. This robot cannot collide with itself as is possible with a six-degree-of-freedom robot. The Python shares a control language with all other Adept robots. This makes it possible to transfer the tools created for the Python to a six-degree-of-freedom robot such as the Adept Viper.

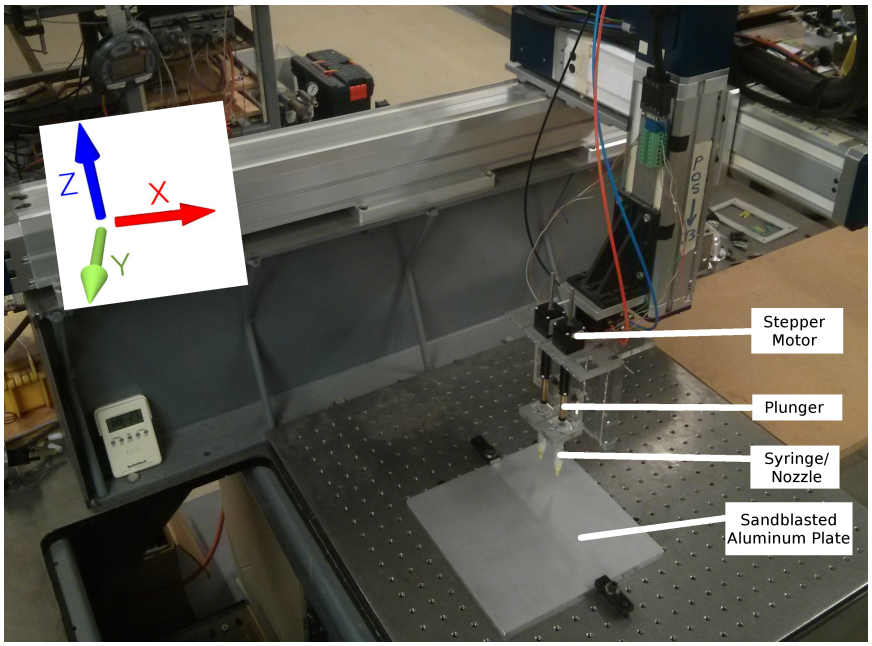



Figure 1: Adept Python with viscous extrusion tools attached

### 2.1.2 Robot Controller

The robot controller used was the Adept SmartController CX. This particular controller offers eight optically isolated digital outputs each allowing for two states. This allows for $(2!)^8$ or $256$ possible combinations of states. The voltage output of the digital signals is 0 or 24 V. It also allows for ethernet communication using TCP/IP and includes a Trivial File Transfer Protocol (TFTP) server.

### 2.1.3 Stepper Motors

In order to dispense the viscous material in a controlled manner, a stepper motor, a plunger, and a syringe were mounted as the end-effector of the robot. A stepper motor was used to actuate the plunger which pushed the material out of the syringe. The stepper motors used were Haydon Kerk 28H47-2.1-925 linear actuators shown in figure 2 [1]. These motors have a resolution of 0.003175 mm/step (0.000125 in/step) and a maximum thrust of 11.3398 kg (25 lbs).

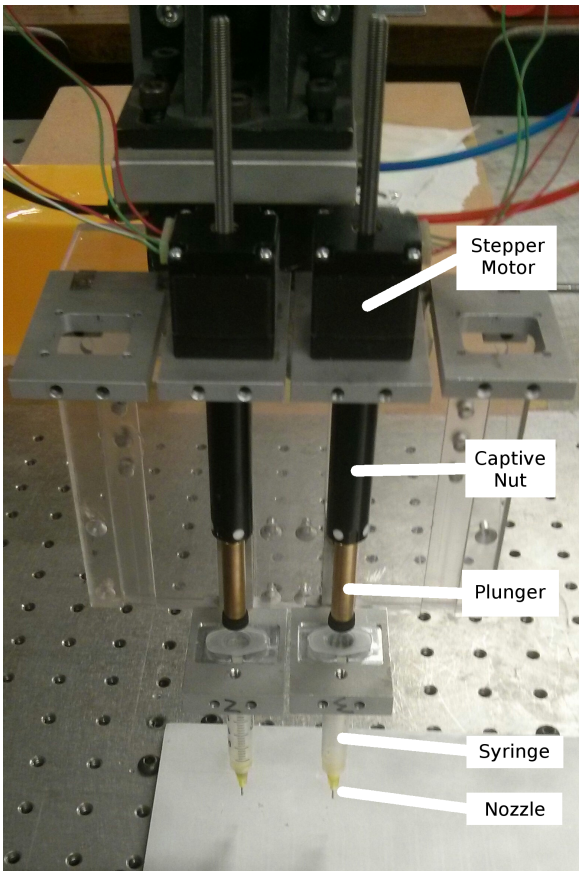



Figure 2: Haydon Kerk stepper motors with plungers attached

### 2.1.4 Stepper Motor Control

The stepper motors are controlled using an Arduino Mega [3]. The Arduino receives input from the SmartController as a digital signal which corresponds to the motor and speed to be used. These signals are controlled using the V+ robot language and are turned on and off at the beginning and end of each extrusion. The motors are driven by Cooldrv drv8825 stepper motor drivers placed on a RAMPS 1.4 shield for Arduino shown in figure 3 [7].

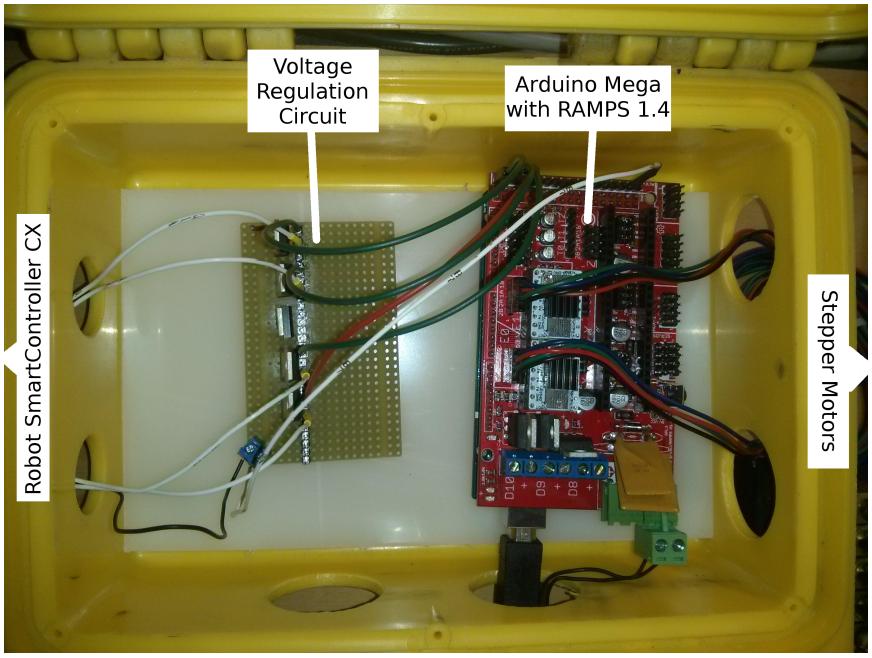



Figure 3: Left: voltage regulation circuit. Right: Arduino Mega with RAMPS 1.4

### 2.1.5 Voltage Regulation Circuit

The SmartController has eight two-state digital output signals for a total of 256 combinations. These digital signals output 24 V in their HIGH state. The voltage of the LOW signal value depends on the load placed on the circuit. The Arduino firmware recognizes voltages above 3 V to indicate a HIGH state and voltages below 3 V to indicate a LOW state. The maximum input voltage supported by the Arduino board is five volts, so the voltage from the SmartController must be stepped down.

A circuit using NTE 960 voltage regulators was used to step each 24 V volt output from the Smart Controller down to the 5 V needed by the Arduino board. This circuit is shown in figure 3. The voltage regulator also provided the circuit load necessary to reduce the voltage for the LOW signal value. With the voltage regulator providing the circuit load, the LOW state voltage is approximately 1 V.

The number of possible speeds is related to the number of available channels for signals and the number of states possible for a single channel. For purposes of the

Adept SmartController, there are eight digital output signals each with two possible states. The number of possible speeds is given by the factorial of the number of states to the power of the number of channels, or $(states!)^{channels}$.

## 2.2 Software Tools Employed

### 2.2.1 *Python* Programming Language

The *Python* programming language was employed as the base language to develop the translation algorithm used to transform G-code commands to V+ and Arduino code. *Python* is a high-level general-purpose interpreted programming language. It was selected because it contains tools which aid in the reading and parsing of text files. *Python* also allows for easy manipulation of list structures.

### 2.2.2 Stepper Motor Control

The AccelStepper library for Arduino was utilized to program and control the motion of the motor [2]. This library allows the speed of the stepper motor to be defined in steps per second and sends the necessary voltage to the stepper motor driver in order to achieve this speed.

The G-code to V+ translator generates an Arduino program in addition to the translated V+ program. The Arduino program responds to changes based on the states of the input pins on the Arduino board. The program will change the speed of the stepper motors based on the combination of states of the input pins.

Since the necessary speeds are determined by the translator program prior to execution time, they can be uploaded with the firmware to the Arduino board. The necessary speed can then be selected through the digital input pins on the Arduino board. The values for the input pins on the Arduino board are received from the digital outputs on the Adept SmartController. These outputs are controlled through

Monitor Commands given by the V+ program.

### 2.2.3 Automated Control Environment

The Automated Control Environment (ACE) is a software program used to program and control the robot. A screenshot of the software interface is shown in figure 4. The left column is the workspace explorer used to store and retrieve programs. The middle column is the code editor. The right column is used for task status control. The Adept ACE software also includes an interface for manual control of the robot and an interface for manual control of the digital output signals.

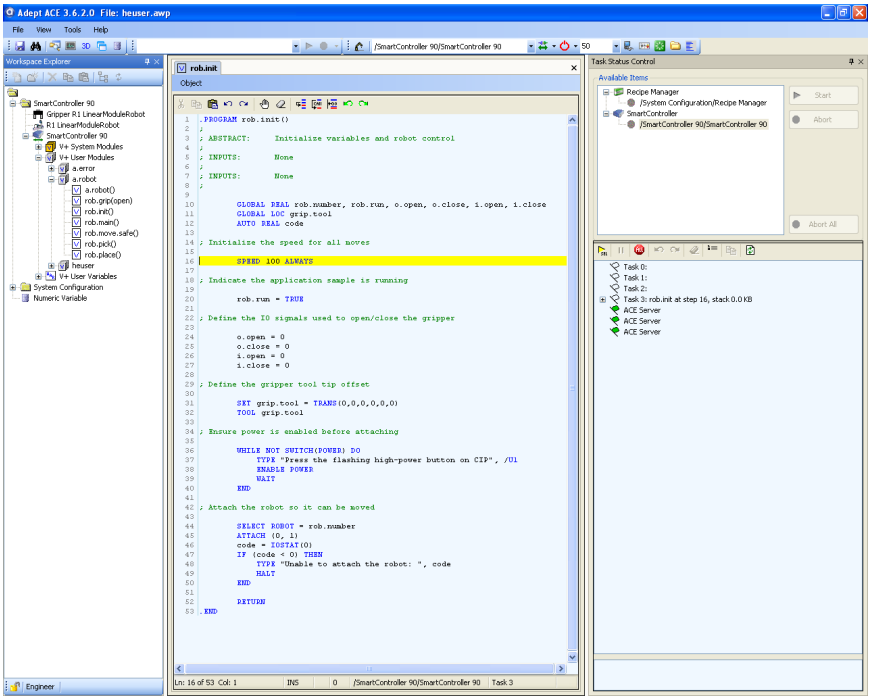



Figure 4: Screenshot of the Adept ACE software interface

### 2.2.4 Trivial File Transfer Protocol

For small programs of a few hundred lines or less, the lines of code can be transferred to the controller directly through the Adept ACE software interface. However, as the number of lines increases the amount of time required by ACE to acquire these

lines increases exponentially.

G-code programs used for 3-D printing are typically quite large, on the order of thousands of lines. Pasting the equivalent V+ program directly into ACE becomes impractical. For instance, the G-code file necessary to fabricate the representation of the Empire State Building shown in figure 5 is 1562 lines. The V+ program generated from this G-code file is 6108 lines long.

A G-code command for additive manufacturing may contain information for the extrusion speed, platform speed, and destination coordinates all within a single line. In the V+ language, each of these parameters is specified by a separate command. Hence, the V+ files generated are typically four to five times as long as the original G-code file.

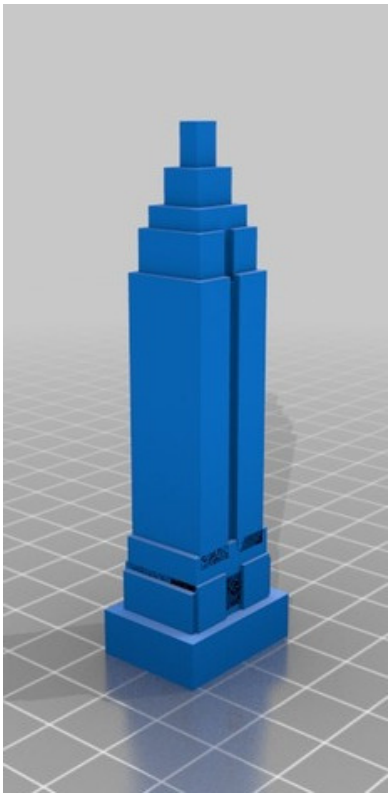


Figure 5: Empire State Building model [9]. Object height: 7.6 cm.

Robots equipped with an Adept SmartController allow for other methods of communication outside the Adept ACE environment. The SmartController is capable of

communicating with other devices on the same network through the Trivial File Transfer Protocol (TFTP). A TFTP server running on a device on the network facilitates access to the files on that device through its the TCP/IP protocol. TFTP server software is provided by Adept and is accessible through Monitor Commands. Monitor Commands are special V+ codes used to access disk files, execute programs, and display system status. [8]

The TFTP software makes it is possible to load V+ programs stored on other devices directly onto the SmartController. This capability bypasses the step of cutting and pasting the code into the ACE software editor before transferring it onto the SmartController.

## 2.3 Algorithm Limitations

After the G-code file has been parsed, all extruding speeds that were used are explicitly written into the Arduino program and assigned a unique combination of input signals. This strategy allows the desired speed to be selected by the robot controller. In addition to the speed, the G1 command also specifies the amount of material to be extruded with the ”E” parameter. The method used does not allow the amount of material to be explicitly selected by the robot controller. Instead, the amount of material extruded is determined based on time. The extrusion is started and stopped at the beginning and end of each movement command. For G1 commands where material is extruded but no motion occurs, the time that it will take to extrude the specified amount of material is calculated. Then, the robot controller will turn on the extruder for that amount of time.

Conventional 3-D printers use polymeric filament that is solid at room temperature. The filament is then heated in the extruder until a sufficient viscosity has been reached and then pushed through a nozzle. The goal of this research is to develop the tools necessary to perform viscous extrusion using Adept robots. Therefore, the

research was conducted for material that is sufficiently viscous at room temperature. Using material that has sufficient viscosity to be extruded at room temperature eliminates the need for temperature control. Consequently, the translation algorithm written does not include support for heater or fan control.

# 3 Methodology Development

Conventional 3-D printers are designed to be able to interpret G-code files to perform the operations necessary to fabricate an object. Industrial robots are not designed specifically to implement 3-D printing techniques and thus are not capable of interpreting G-code commands. Each industrial robot manufacturer has its own language for control of its respective robotic platforms. Adept robots are programmed using the V+ language.

In order to use the Adept Python robot to perform additive manufacturing tasks, a translation algorithm was developed and implemented to interpret a G-code file and produce a set of equivalent V+ commands. Since the stepper motors were controlled separately from the robot, the algorithm also had to produce an Arduino code for stepper motor control and coordinate the robotic motion platform with the Arduino control platform.

Figure 6 shows a visual representation of the methodology used in this research. The first three blocks are common to the conventional 3-D printing methodology. The translation algorithm block represents the software tools developed in this research. The robot platform and controller are used in place of a conventional 3-D printing motion platform.

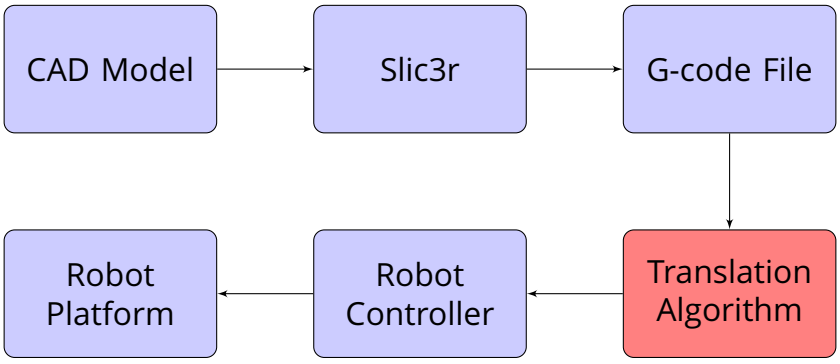



Figure 6: Process diagram

Figure 7 shows a visual representation of the tasks performed by the translation algorithm. A G-code file is taken as the input to the algorithm. The file is analyzed and the necessary robot operations are output in a V+ file. An Arduino file is also output for control of the extruders. The robot and the extruder platforms are coordinated through digital signals sent by the robot controller to the Arduino. These signals are controlled through the V+ code.

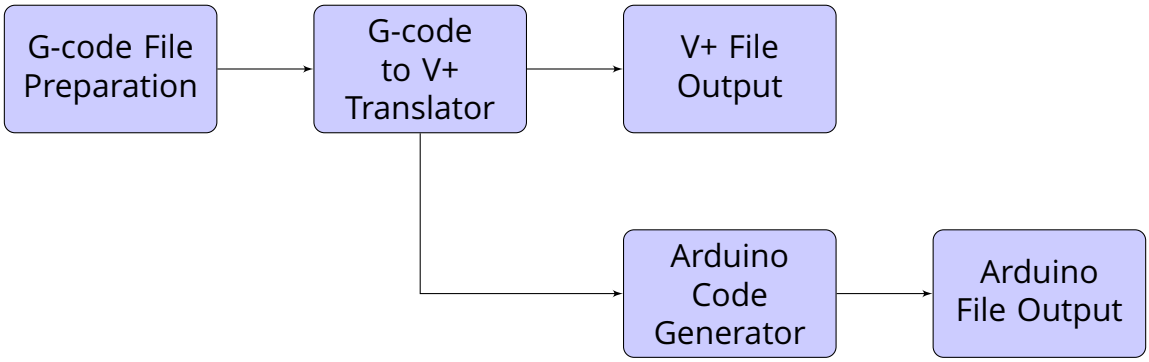



Figure 7: Translation algorithm

## 3.1 Algorithm Features

The purpose of this algorithm is to enable Adept robots to 3-D print structures out of viscous material from CAD models. Software is available to convert CAD models to a set of G-code commands but, to the best of our knowledge, no software exists in the public domain that will convert G-code to V+ for additive manufacturing purposes. The algorithm developed in this research will convert a set of G-code commands to a V+ program that an Adept robot is able to execute.

Since the SmartController does not have the ability to drive a stepper motor, an Arduino was used alongside the controller for this purpose. The controller can then communicate with the Arduino through digital outputs. The algorithm will automatically generate the Arduino code necessary to facilitate this communication.

The algorithm includes support for multiple extruders. There is no software limi-

tation to the number of extruders that can be used. The number of extruders is only limited by the number of digital outputs available on the robot controller.

Sometimes it is necessary or convenient to modify the location of the structure to be printed after the G-code has already been generated. For instance, the location of the object along the Z-axis may need to be altered depending on the nozzle length used. The algorithm provides a method to easily accommodate location changes for the final product in 3-D space. Location changes can be addressed through X, Y, and Z offset variables defined in the user-defined parameters section of the algorithm. If no change is desired, then these should be set to zero.

Another feature of the algorithm is that bed limits can be defined. Variables in the user-defined parameters section define the minimum and maximum values of all three axes. The algorithm displays a warning with information about which limits have been exceeded and by how much. Often this warning can be corrected by making an adjustment to the location of the object using the X, Y, or Z offset variables.

The total number of digital output signals configured for use on the controller may be defined using the max_signals variable. If the input file requires more than the available number of signals, a warning will be displayed once the algorithm execution is completed.

## 3.2 User-Defined Parameters

The first lines of the translation algorithm contain the variables that are defined by the user prior to run-time. These parameters include the directory path on the hard drive where the G-code file to be translated is located, an origin specific to the workspace of the robot being used, and the boundaries of the robot workspace. They also include the total number of signals available for use on the controller and the maximum RPM supported by the stepper motor.

Prior to execution time, maximum and minimum limits for each axis can be de-

fined to describe the robot workspace. Each V+ command is evaluated as it is generated to determine whether the destination point lies within the defined workspace. Each coordinate is individually compared to the maximum and minimum values defined for its respective X, Y, or Z axis.

If any coordinate lies outside the limits for its axis, the execution is not interrupted, but the count of the number of commands outside the workspace is increased by one. Separate counts are kept for the number exceeding the maximum and the number below the minimum for each axis.

The coordinates of the user-defined origin are used to reposition the object to be printed within the workspace. This can also be achieved using Slic3r; however, the proposed method provides user control after the G-code file has been generated.

The available input pins are defined in a list named ”inputs”. This list contains the pin numbers of pins on the Arduino board that will be used. The workspace boundaries, number of signals available, and maximum stepper motor speed are compared with each V+ command generated. If any command is instructs the robot to move outside these limits, then a warning is displayed once the execution has completed. If the object fits within the robot workspace, this issue can be addressed by redefining the local origin and thus shifting the position of the object away from the borders.

## 3.3 Supported G-code Commands

In order to parse the G-code commands, each line from the G-code file is analyzed in sequence. The line is first tokenized so that each G-code parameter can be referenced individually. A conditional is used which calls a predefined function depending on which G-code is contained within the first token of the line.

### 3.3.1 G1 Xnnn Ynnn Znnn Ennn Fnnn

The G1 G-code indicates linear motion of the motion platform from its current location. The X, Y, and Z parameters correspond to the X, Y, and Z coordinates of the endpoint of the move in the Cartesian plane. The E parameter indicates the length of the filament that will be extruded throughout the motion. The difference between the current filament position and the next filament position gives the amount of filament to be extruded over the next movement. The F parameter indicates the linear speed in millimeters per minute and it is used to indicate the speed of both the filament and the head of the printer. A V+ command for this linear speed converted to millimeters per second is output when the F parameter is present.

When a G1 command is encountered, it is first scanned for the F parameter. If one is present, a V+ SPEED command is generated, and the value is stored for use with the next extrusion command. Then, the list of unique speeds encountered is scanned to determine whether it contains the new speed. If it does not, then this new required speed is added to the list.

Next, the G1 command is scanned for the E parameter. If an E parameter is located , then its value is compared to the last stored value. If the difference is greater than zero, then a V+ SIGNAL command is generated to activate the stepper motor.

If multiple extruders are being used, the SIGNAL command is generated by using the binary representation of the tool number, followed by the binary representation of the list-index of the speed in the unique speeds list. These two binary numbers are concatenated to form a list of signals. The position of each ”1” in the list corresponds to a signal number to be activated. If a single extruder is being used then the tool number is excluded.

Once the signal numbers have been extracted, a V+ SIGNAL command is generated to engage those signals. However, a WAIT command must be issued before every SIGNAL command to prevent the controller from engaging the signals before

the previous MOVE command has been completed.

A V+ command for movement is output only if one or more of the X, Y, or Z parameters is present. Since the V+ command for motion incorporates the X, Y, and Z coordinates all in a single command, the command cannot be generated until all G-code parameters have been analyzed. This is accomplished by placing an indicator variable set to False prior to the conditional. This variable is then set to true whenever an X, Y, or Z parameter is present. Once all parameters have been analyzed, a V+ command is generated which includes the values from all Cartesian variables present in the G1 command.

The MOVE command immediately follows the SIGNAL command. The algorithm then scans the G1 command for X, Y, and Z parameters. Any Cartesian parameters present are used to create a V+ MOVE command. The SIGNAL command is utilized to turn off the signals in use, preceded by another WAIT command.

If no Cartesian parameters are present but the E parameter is present, then the robot will extrude the given amount of filament in place. This is accomplished by taking the length of filament to be extruded divided by the current speed setting to obtain the time of extrusion. Next, a WAIT command is generated for the calculated amount of time, followed by the SIGNAL off command.

### 3.3.2 G4 Pnnn Snnn

The G4 G-code signals a pause in the current operation. It may be followed by a ”P” parameter or an ”S” parameter. The ”P” parameter indicates the amount of time to wait in milliseconds while the ”S” parameter indicates the time to wait in seconds. Since the V+ WAIT command only accepts seconds, lengths of time specified using the ”P” parameter are converted to seconds before the V+ command is generated.

### 3.3.3 G28 X Y Z

The G28 G-code moves the extruder to the origin. The G28 G-code can be used by itself without any parameters. In this case, all axes of the extruder are moved to the origin. If one or more parameters are present, then only the axes explicitly specified are moved to the origin. The G28 command is handled similarly to the G1 command except that the origin coordinates are used instead of coordinates specified in the command. If coordinates are specified in the G28 command, they are ignored.

When a G28 command is encountered, it is scanned for Cartesian parameters. If none are present, then a V+ MOVE command is generated using the coordinates of the user-defined origin. If at least one Cartesian coordinate is present, then a V+ MOVE command will be generated using the user-defined origin, only for the axes present. Coordinates specified after the axis label are ignored.

### 3.3.4 G92 Xnnn Ynnn Znnn Ennn

The G92 G-code resets the location of the origin. Similar to the G28 command, it can be used without any parameters. In this case, the coordinates of the current location of the extruder are used. Otherwise, only the axes which are explicitly specified in the command are changed. Unlike G28, the coordinates following the axis label are not ignored. They can be used to specify a new origin other than the current position.

When a G92 command is encountered, it is scanned for Cartesian parameters as well as for the filament position parameter, E. If no parameters are present then the origin is reset to the current position in Cartesian space and the filament position is reset to zero. However, if one or more parameters are present, only those origins explicitly stated will be affected. Any values given with the parameter names are ignored. The G92 command only resets the location of the origin; it does not cause

any physical motion to occur.

### 3.3.5 Tnnn

The tool change command, T, is followed by the tool number. It is only used with G-code files that use more than one tool. The T command has one token; the remainder of the line is ignored. When a T command is encountered, the current tool is updated with its value and the tool number is added to the list of unique tools.

## 3.4 Custom G-code

The Slic3r software allows for custom G-code to be added to its output in pre-defined places. Additionally, Slic3r provides some placeholder variables that can be used to reference Slic3r settings.

The Start G-code is placed at the beginning of each G-code file generated by Slic3r. These G-codes define an initialization procedure that is executed prior to object fabrication. The Start G-code used in this work is shown in figure 8.

```
G28 ; home all axes
G1 Z5 F5000 ; lift nozzle
T0 ; First tool
G92 E0 ; Reset extruder
G1 E300 F600 ; Build pressure
```

Figure 8: Start G-code

At this point, no tool has been defined by Slic3r. Therefore, the [next_extruder] placeholder is not valid in the Start G-code. If a tool other than T0 is used first, then the tool change command must be adjusted accordingly. The Tool change G-code used in this work is shown in figure 9.

```
G28 X Y ; home X and Y axes
G92 E0 ; Reset extruder
G1 E-150 F600 ; Retract
G4 S30 ; Wait 30 seconds
T[next_extruder]
G92 E0 ; Reset extruder
G1 E300 F600 ; Build pressure
```

Figure 9: Tool change G-code

Before every tool change command, the X and Y axes of the robot are returned to the home position and the plunger is retracted to relieve the pressure in the syringe. A pause of thirty seconds is given to allow any excess material to be extruded due to syringe pressurization. Then, the tool change command with the [next_extruder] placeholder is used and the plunger is advanced in order to begin building pressure within the syringe.

Lastly, the End G-code is appended by Slic3r to the end of the generated G-code file. The Tool change G-code used in this work is shown in figure 10. Once the print has completed, all axes of the robot are returned to their home position.

```
G28 ; home all axes
```

Figure 10: End G-code

## 3.5 G-code File Preparation

The translation algorithm first reads the G-code file into memory. Then, each line of code is stored as an item in a list. This allows for iteration over the lines individually. With each line of code stored as a list item, the return character and the newline character are removed from the end of each line.

Semicolons indicate comments in both G-code and V+ languages. They can be

full-line comments where the semicolon begins the line, or partial line comments where the semicolon is placed after a command. In both cases, the semicolon and any characters following it are ignored by the interpreter. Using regular expressions, all comments are removed by the algorithm.

Blank lines are also ignored by the algorithm. Any full-line comments which are removed leave a blank line in their place. The index of each empty list item is recorded for later deletion. Deleting an item from a *Python* list causes each subsequent index to be subtracted by one to fill the gap. By not immediately deleting empty list items, the indices of subsequent list items are not disturbed.

Once all indices of blank lines have been recorded, they are deleted from the list in reverse order. In this way, only the indices of list items that have already been treated are changed. Removing blank lines, after comments have been removed, treats both pre-existing blank lines and those left by full-line comments.

## 3.6 G-code to V+ Translator

Once the G-code file has been stored in memory, each line is analyzed individually. Each line is broken into separate tokens, using spaces as the delimiter. If the first token is G1, G28, G92, or begins with a T, the appropriate function is called to handle that line.

Some of the G-code that Slic3r outputs only applies to temperature controlled environments. The M105 and M109 commands set the temperature of the extruder, while the M106 and M107 commands activate and deactivate the fan. Since a heated extruder and fans are not being used in this work, these codes are ignored by the translator. However, they could be easily added if needed.

Other G-codes set conditions that are the default behavior for Adept robots. The G21 command sets the units to millimeters, the G90 code sets it to absolute positioning, and the M82 command sets the extruder to absolute mode. These are all default

settings of the robot and are ignored by the algorithm.

The M84 code stops the idle hold on all axes and the extruder. This code is only recommended for use between or after print jobs to resolve noise issues on some 3-D printers. These issues are not present on Adept robots and so the M84 code is ignored as well.

If the algorithm encounters any G-codes not defined in the program, the program will halt and a V+ code will not be output. An error message containing the name of the unrecognized G-code will be displayed. In order for the algorithm to complete successfully, there must be a procedure defined for every G-code encountered.

## 3.7 V+ File Output

The base name of the V+ file to be written is the same as the name of the G-code file being ported. The ".gcode" extension is swapped for the ".v2" extension used by the Adept ACE software. The output file is placed in a subdirectory of the location of the G-code file. The directory name is the same as the name of the program. If no directory with that name exists, then one will be created. If a directory with that name does exist, each file within that directory that has the same name as one of the output files will be overwritten.

Every V+ program begins with ".PROGRAM", followed by the program name, followed by "()". The same name is used for the program name as was used for the file name excluding the ".v2" extension. These three strings are concatenated and written to the output file as the first line. The comment template created by the ACE software at the beginning of each V+ program is replicated and placed in the output file after the program name line.

Once all of the header lines have been created and written to the file, each translated V+ line is written individually to the output file. Finally, every V+ program must end with ".END", followed by a newline. The new line must be present after the ".END"

in order for the controller to load the program successfully. If it is not present then the LOAD Monitor Command will fail.

Once the output file has been written, the program will display the total number of unique speeds encountered. If the number of signals used in the code exceeds the pre-defined maximum, a warning will be displayed. Similarly, if any of the V+ commands generated instruct the robot to move outside the pre-defined boundaries of the robot workspace, a warning will be displayed indicating which boundaries were exceeded and the maximum distance by which each was exceeded.

## 3.8 Arduino Code Generator

The signals received by the Arduino board consist of two groups: the signals used to select the tool and the the signals used to select the speed of the extruder motor. As defined in the V+ code generator section, the tool signals will occupy the first group of pins. The number of pins used for tool selection is given by the number of digits in the binary representation of the total number of speeds.

Once the number of pins used for both tool and speed selection is calculated, they are added to obtain the total number of pins that will be used by the algorithm. The list of available input pins is then truncated at this number to contain only the pin numbers of the pins that will be used.

If no tools are defined in the G-code, then the list of tools will be an empty list. When this is the case, tool zero is added to the empty list of tools for the purposes of the Arduino code only. The Arduino program will default to using tool zero when no tools are defined in the G-code.

The head of the Arduino file contains definitions that will be used by both the setup function and the loop function. Here, the AccelStepper library is included, followed by the assignment of variable names to each pin number [2]. Finally, each motor is defined using the AccelStepper class.

The AccelStepper library requires an input speed to be defined in steps/sec. It also supports controlled acceleration and deceleration of the motor while reaching its required speed. However, this custom acceleration functionality was not used in this work since the required stepper motor speeds are very low.

Next, the setup function is created for the Arduino code. The setup function contains commands that will be executed only once at start time. In this function, each pin that will be used is configured as an INPUT or an OUTPUT pin. The motors are enabled by writing a LOW value to their respective enable pins.

Finally, the loop function is created. Commands contained within the loop function will be executed repeatedly in sequence for an indefinite amount of time. The first task in the loop function is to read the state of each pin. Then a conditional is created for the ”stop” case. If all the speed pins are in the LOW state, then the motor will not move, regardless of the state of the motor pins.

Next, a conditional is created for each motor and speed combination. The signals for each tool in the list of tools are calculated by taking the list index of the tool and converting it to binary format. The binary number is then reversed so that the first signal is on the left side, as the controller signals are numbered. Then the right side of the number is padded with zeros until its length is equal to the length of the longest tool number. The same procedure is implemented for each speed in the list of speeds. Each tool number is then paired with each speed number to create a combination of signals for every tool and speed combination in the G-code file.

A conditional statement is generated for each number combination. A ”0” represents a LOW state and a ”1” represents a HIGH state on that pin. Within the conditional, a command is generated which selects the appropriate motor and sets it to the corresponding speed. Then a command is generated to move the motor one in the specified direction. After each step, the algorithm returns to the beginning and reads the state of each pin again. In this way, the motor will never move more than

one step without re-verifying that the pin states are unchanged.

The G-code F parameter specifies the speed to be used in mm/min. However, the AccelStepper library only accepts speed commands in steps/sec. The stepper motor moves 0.003175 mm/step (0.000125 in/step). Using this information a volume conservation calculation is performed to obtain the linear speed required for the plunger (equation 1) followed by a unit conversion calculation to convert the units to steps per second (equation 2).

$$speed_{plunger} = speed_{nozzle} \times \frac{radius^2_{nozzle}}{radius^2_{syringe}} \tag{1}$$

$$\frac{steps}{second} = \frac{millimeter}{second} \times \frac{inch}{millimeter} \times \frac{steps}{inch} \tag{2}$$

## 3.9 Arduino File Output

The file name for the Arduino program file is created by adding the ".ino" extension onto the program base name. The directory to which the Arduino file will be written is the same directory used for the V+ file output. If any of the stepper motor speed commands exceed the predefined maximum, then a warning will be displayed after the output file has been written.

## 3.10 Sample G-code Translation

A sample line of G-code, shown in figure 11, was created for the purpose of demonstrating the capabilities of the translation algorithm. The V+ code and Arduino code generated for this particular line of G-code are shown in figures 12 and 13 respectively.

```
G1 X600 Y150 Z-60 E0.1 F600.000
```

Figure 11: Sample G-code input

First the speed of the robot motion platform is defined by converting the speed in mm/min given in the G-code to mm/sec. Then, a WAIT command is generated before every SIGNAL command to ensure the that the execution of the previous command has reached completion. The digital signal corresponding to the desired tool and speed defined in the Arduino is enabled. Finally, a MOVE command is generated to move the robot to the destination coordinates defined by the X, Y, and Z parameters in the G-code command. Once the motion has has completed, the stepper motor actuator is disabled through a second SIGNAL command.

The STATE(2) function shown in figure 12 returns information about the current or previous robot motion. The ”== 2” corresponds to the robot motion being stopped at a planned location.

```
.PROGRAM example();
        SPEED 10.0 MMPS ALWAYS
        WAIT STATE(2) == 2
        SIGNAL 1
        MOVE TRANS(600.0,150.0,-60.0,0,180,0)
        WAIT STATE(2) == 2
        SIGNAL -1
.END
```

Figure 12: Sample V+ output

The translation algorithm also outputs an Arduino program necessary to coordinate the stepper motor actuation with the motion of the robot platform. The Arduino code created from the sample G-code in figure 11 is shown in figure 13.

The head of the Arduino code contains the include command necessary for use of the AccelStepper library, as well as the variable names assigned to the utilized

pins on the Arduino board. It also contains the AccelStepper motor definition commands. The motor definition command consists of ”AccelStepper”, followed by the name given to the motor being defined, followed by ”AccelStepper::DRIVER,” and the step and direction pin to which the motor driver is attached.

The setup function contains a set of commands which will be run once at startup time. First, the baud rate for communication over USB is defined. Configuration commands to set each pin to be utilized as an INPUT or an OUTPUT pin are also defined. Next, each motor being used is enabled by setting its enable pin to the LOW state. Finally, the maximum speed of the stepper motor is defined for each motor being used.

The loop function contains commands which will be continuously executed in sequence. The algorithm first parses the G-code file to create a list of all tool and speed combinations. Each combination is assigned a binary value and incorporated into the Arduino code. The Arduino interprets the digital input signals from the SmartController as a binary value corresponding to one of the pre-defined tool and speed combinations. A conditional for each combination is created which contains commands to cause the appropriate motor to move at the specified speed. The units of the stepper motor speed are in steps/sec.

```
#include <AccelStepper.h>

#define D1_SIGNAL_PIN      16

#define E0_STEP_PIN        26
#define E0_DIR_PIN         28
#define E0_ENABLE_PIN      24

AccelStepper E0(AccelStepper::DRIVER, E0_STEP_PIN, E0_DIR_PIN);

void setup()
{
  Serial.begin(9600);

  pinMode(D1_SIGNAL_PIN, INPUT);    // Speed Pin

  pinMode(E0_STEP_PIN, OUTPUT);
  pinMode(E0_DIR_PIN, OUTPUT);
  pinMode(E0_ENABLE_PIN, OUTPUT);

  digitalWrite(E0_ENABLE_PIN, LOW);

  E0.setMaxSpeed(1000);

}

void loop()
{
  int IN0 = digitalRead(D1_SIGNAL_PIN);

  if (IN0 == LOW) {
    // motor does not move
  }
  else if (IN0 == HIGH) {    // Extruder 0, Speed 1
    E0.setSpeed(8.51);
    E0.runSpeed();
  }
}
```

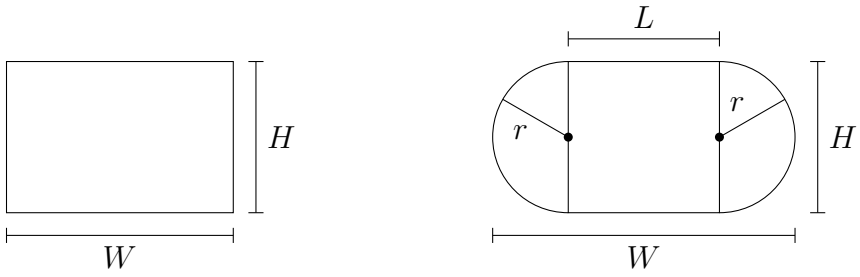
Figure 13: Sample Arduino output

## 3.11 Scaffold Pore Size Calculation

In tissue engineering applications, a three-dimensional porous scaffold is often constructed from a bio-compatible material. In order to construct a scaffold from a viscous material, the geometry of the cross-section of the extruded strand must be approximated. This approximation is used to calculate an infill density from a desired pore size.

The analysis is based on volume conservation of the extruded material and the assumed geometry of the deposited strand. The original geometry of an extruded strand as it exits the nozzle is circular with radius $R_0$ (diameter $D_0$). The area of this shape is given by the equation $A_0 = \pi R_0^2$.

The deposited strand could exhibit a rectangular profile or a composite geometry cross-section as shown in figure 14.




(a) Rectangular approximation (b) Composite approximation

Figure 14: Extruded strand geometry cross-section

Using the rectangular strand approximation shown in figure 14a, the width of the extruded strand is calculated and the result is shown in equation 3, where $n$ represents the fraction of the nozzle diameter used as the layer height. The steps taken to obtain this result are shown in appendix A.

$$W = \frac{\pi}{2n} R_0 \tag{3}$$

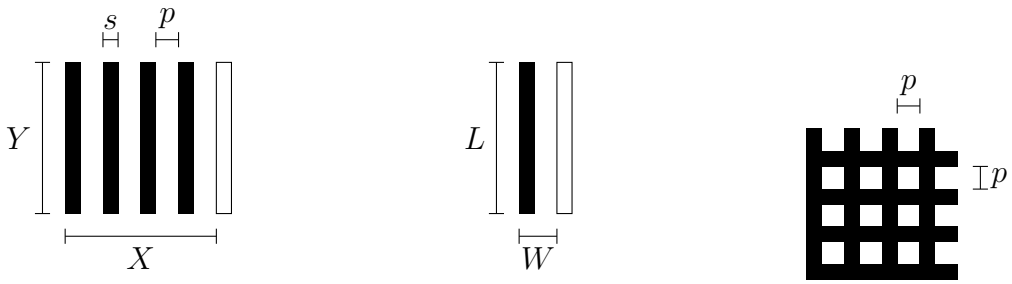
Another possible approximation is a composite shape composed of a rectangle with a semicircle on each end. A diagram of this approximation is shown in figure 14b. A conservation of volume calculation with this approximation gives the result shown in equation 4, where $n$ represents the fraction of the nozzle diameter used as the layer height. The steps taken to obtain this result are shown in appendix B.

$$W = \left( \frac{\pi - \pi n^2}{2n} + 2n \right) R_0 \tag{4}$$

The results shown in these equations provide a method to approximate the width of an extruded strand. Using only the nozzle-to-bed distance and the nozzle radius, an approximation of the width can be calculated.

Once the strand width has been approximated, the infill density necessary for a desired pore size must be calculated. Figure 15 shows a visualization of a set of extruded strands that form an object. Figure 15a shows a single layer of strands extruded side by side. The X and Y variables indicate the width and height of the layer. The solid rectangles indicate strands within the dimensions of the object; the outlined rectangles indicate strands outside the object dimensions shown for position information.

Figure 15b shows a subset of the strands required to form a full layer. $L$ represents the length of the strands, while W indicates the distance from the side of one strand to the same side of the next. Figure 15c shows two layers printed in opposite directions to form a scaffold. The pore size is indicated by $p$ in figures 15a and 15c, while $s$ in figure 15a represents the width of a strand.




(a) Single layer (b) Strand to pore ratio (c) Two layers

Figure 15: Schematic for infill density calculation

Equation set 5 shows a method of calculating the infill density necessary to give a desired pore size. The variable $D$ represents the infill density. The infill density and the strand width are given as inputs to Slic3r to produce a scaffold with the desired pore size.

$$\begin{aligned} D &= \frac{Area_{strands}}{Area_{total}} \\ &= \frac{sL}{LW} \\ &= \frac{s}{W} \\ W &= s + p \\ D &= \frac{s}{s+p} \end{aligned} \tag{5}$$

# 4 Methodology Verification

In order to verify the developed methodology, experiments were conducted using toothpaste as the viscous material. The multiple extruder capability was demonstrated by drawing letters using two different types of viscous material. Experiments were conducted to characterize the width of a single strand of extruded toothpaste. A three dimensional scaffold was also fabricated using different materials for each layer, for verification of the developed tools. In this research, toothpaste was used as the viscous material since it has similar viscosity to bio-gels and it is substantially less expensive.

## 4.1 Preparation

Before extrusion can be performed, the hardware must be prepared. The syringes must be fitted with the desired needle size and filled with the desired viscous material. Then the stepper motors are ”manually” activated, advancing the plungers until they make contact with the material and a small amount of material is extruded.

In order to accommodate varying needle lengths and bed positions, the position of the Z-axis of the robot at its lowest point must be obtained and provided to the porting algorithm before execution. The extruding hardware is adjustable in order to accommodate syringes and nozzles of varying lengths simultaneously. The location of the bed is obtained by releasing the brake on the Z-axis of the robot and manually lowering it until it comes into contact with the bed. Its Z-axis position is then recorded and adjusted in the porting algorithm through the Z_offset variable.

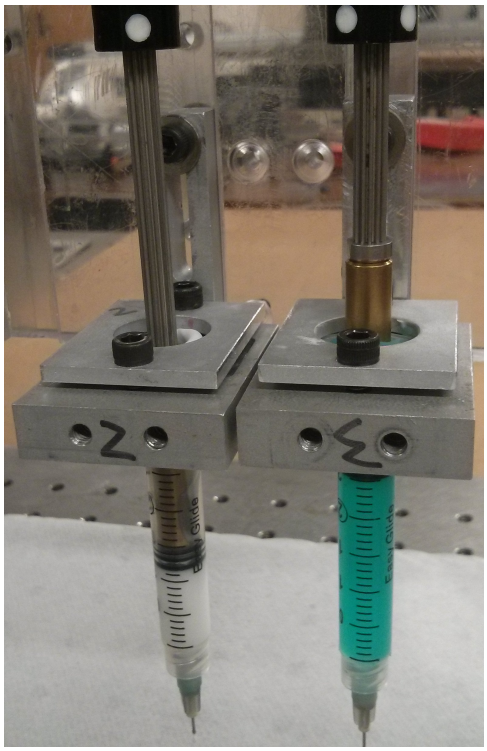


Figure 16: Preparation

## 4.2 Challenges Encountered

This section describes some of the challenges faced in this research and how these challenges were overcome. These challenges were discovered through experimentation and observation, thus some of the early experiments exhibited imperfections which were addressed in the later ones.

### 4.2.1 Bed Leveling

The bed was initially determined to be leveled within 0.05 mm (0.002 in) using a machinist's level. After initial experiments it became apparent that the nozzle to bed distance did not remain constant over the bed. When this was discovered, a dial gauge was used instead of a machinists level to ensure that the bed was parallel to the robot platform as shown in figure 17. The dial gauge was used to verify that

the bed was parallel by attaching it to the robot end-effector and then moving the end-effector along the X-axis and then the Y-axis axis. Once the bed was leveled, this procedure was repeated for the aluminum plate printing surface.

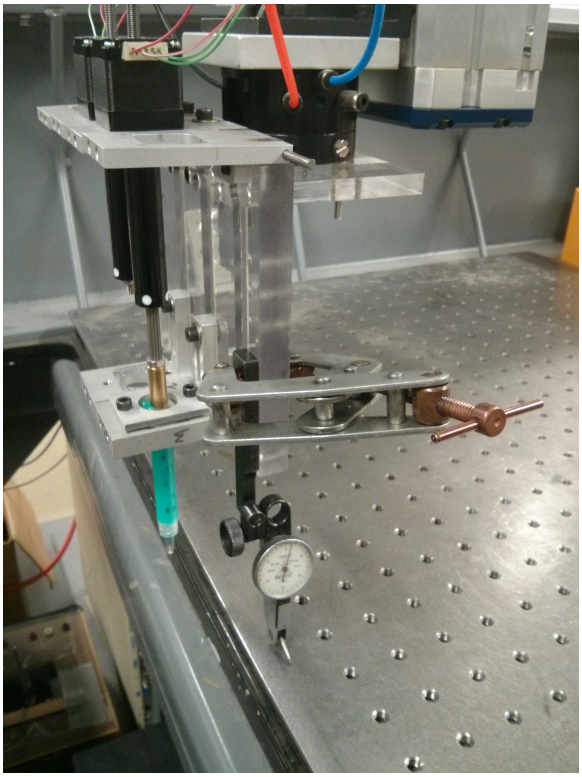


Figure 17: Dial gauge attached to the robot

The aluminum plate, used as the extruding surface, was determined to be warped such that the center was a few thousandths of an inch lower that the edges. This problem was addressed by clamping the sides of the plate to the robot bed during the printing process.

### 4.2.2 Surface Material

Several different surface materials were tested. Paper was evaluated first because it was inexpensive and easy to obtain. The paper was secured to the robot bed with masking tape but it was difficult to ensure that the paper remained flat against the

bed. The paper tended to buckle, causing small deviations in its height. These deviations in height caused the width of the extruded strands to be inconsistent.

A polymer plate, a Teflon plate, and a brush-finished aluminum plate were also evaluated. Each of these materials exhibited issues with the adhesion of the toothpaste to the surface. Finally, an aluminum plate with a sandblasted surface finish was determined to provide adequate toothpaste adhesion.

### 4.2.3 Adhesion to Surface Material

When using paper as the surface material, the toothpaste strands occasionally appeared to have non-uniform width. This issue was exacerbated by extruding on brushed aluminum as shown in figure 18. This non-uniform strand width was determined to be caused by inadequate bed adhesion.

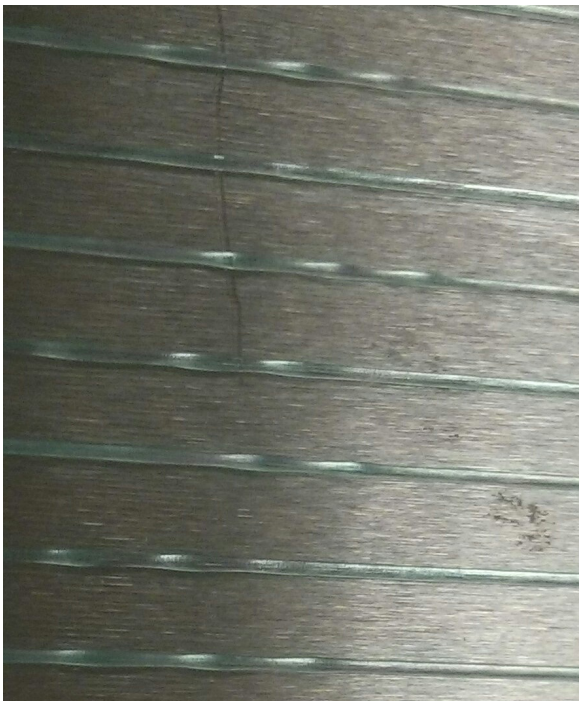


Figure 18: Non-uniform strand width on a brushed aluminum surface

The aluminum plate used as a print bed in figure 18 had a unidirectional brushed satin finish. The bed adhesion issue was more prevalent when printing along the

brushed grain than when printing across it.

Sandblasting the surface of the aluminum plate to roughen the material was determined to increase bed adhesion. Once this method was implemented, the bed adhesion issue was resolved.

### 4.2.4 Toothpaste Variety

Three different toothpastes were experimented with in this work: Aim, Close-Up, and Pepsodent. All three of these brands had small pockets of air in them which occasionally caused discontinuities in the extruded strands. However, Close-Up had much larger air pockets than the other two causing the discontinuities to occur more often and be more pronounced.

## 4.3 Experimental Results

### 4.3.1 CAD-modeled Letters

A sample demonstration model was created using FreeCAD. The model uses two different types of toothpaste to demonstrate its multiple extruder capabilities. Figure 19 shows the model used with a letter height of 3 cm. The results are shown in figure 20.

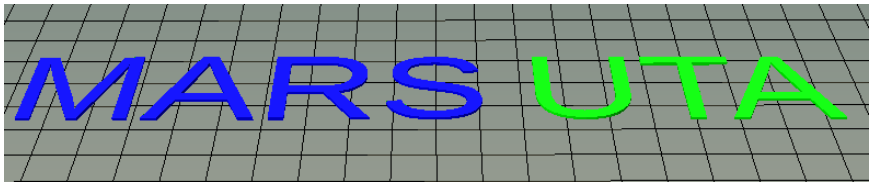



Figure 19: Model of letters (letter height: 3 cm)

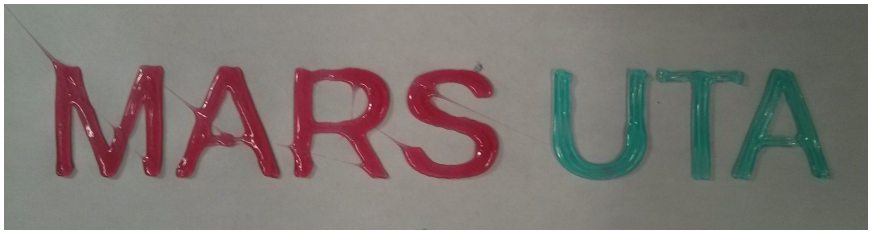



Figure 20: Demonstration of multiple extruders (approximate letter height: 3 cm)

### 4.3.2 Single Strand Width

A set of experiments was performed to determine the effects of two controllable factors on the width of the extruded line: the speed of the robot and the diameter of the nozzle. For each factor, three unique levels were defined, thus requiring a total of $3^2$ or 9 combinations for a full factorial analysis. Each combination was replicated three times for a total of 27 runs. The order of these runs was randomized using Design Expert, a statistical software package used for design of experiments [4].

The speed factor was defined to be 8, 10, and 12 mm/sec on the three levels respectively. The nozzle diameter factor was defined to be 0.468, 0.566, and 0.650 mm respectively. These speeds and diameters were chosen based on preliminary experiments and experience with a conventional 3-D printer. The layer height used was equal to the nozzle diameter to ensure minimum shape deformation of the toothpaste while still providing adequate bed adhesion.

Three measurements were obtained for each strand at $1/4$, $1/2$, and $3/4$ of the length of the extruded line. These measurements were then averaged to obtain the width response. The results of these experiments are shown in appendix C.

Images were captured using a microscope within 120 minutes of printing to minimize shape deformation of the toothpaste due to evaporation. A sample microscope image is shown in figure 21. The ambient temperature and humidity remained constant throughout the experiment at 68° F and 27% respectively.

The microscope used to obtain the images and measurements used in these ex-

periments was a Nikon Eclipse LV150. The objective lens used with this microscope was a Nikon L Plan 2.5x/0.075. The laboratory imaging software used to capture the images was NIS Elements version 4.13.04.

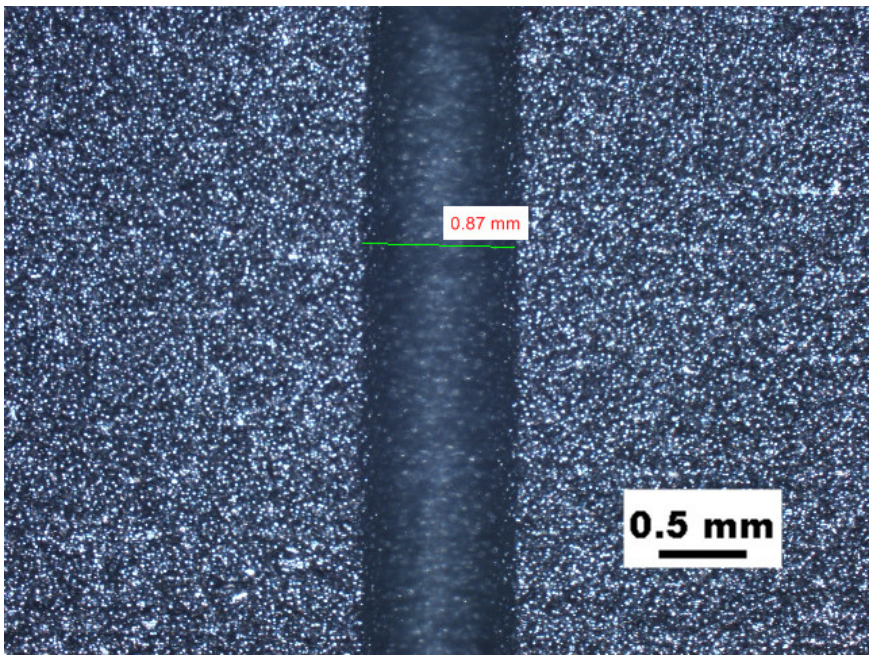



Figure 21: Measurement of strand width

Design Expert was used to analyze the data and produce a plot for the interaction of the two factors. The Analysis of Variance (ANOVA) given by the Design Expert software indicated that the results of this experiment were ”not significant” within the range of factors tested. The software indicates that the data is ”not significant” when the amount of error in the data is great enough that the results may be unreliable.

Even though the software indicates a high amount of error, the data can still be used to extract trends created by the parameters. The results, however, should be used with caution. An interaction plot for the data obtained is shown in figure 22. This plot indicates that the speed has a small effect on the width, as the lines exhibit a negative slope. The negative slope indicates that with increased speed, the strand width will decrease. The slope of the line is consistent among the three nozzle diameters.

The sources of error in this experiment could have included the accuracy of the bed leveling and the flatness of the surface material. Prior to the experiment, the working region was determined to be parallel to the robot within 0.76 mm (0.003 in). The aluminum plate which was placed on top of the bed was determined to have a crown toward its center. The edges of the plate were clamped to the bed prior to the experiment to address this issue. However, the clamps had to be removed between each experiment to analyze with the microscope. Some error could have been introduced each time the clamps were removed and replaced.

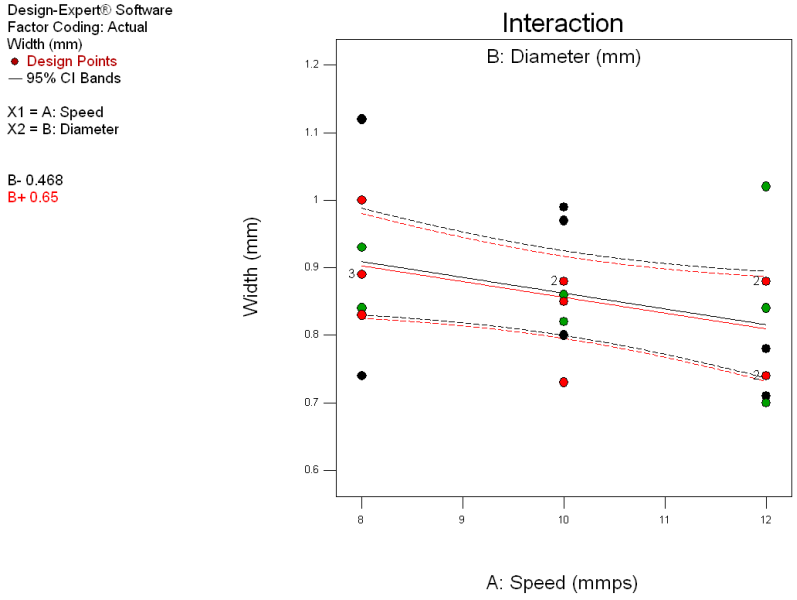



Figure 22: Interaction plot

### 4.3.3 Three Layer Scaffold

Another set of experiments was performed to construct a three-layer scaffold. The purpose of this experiment was to demonstrate the ability of the developed methodology and translation algorithm to extrude multiple layers using multiple viscous materials. A second purpose was to collect data about the upper limit of the infill density using toothpaste as the viscous material. When the infill density is set too high, indi-

vidual strands become indistinguishable from one one another.

A model was constructed from three rectangular blocks, stacked vertically. The height of each block corresponded to the nozzle diameter being implemented. The bottom block was placed in the center of the workspace and had dimensions of 3 cm by 3 cm. The middle block was placed on top of one side of the bottom block and was half the area of the first layer. The top block was placed on top of one end of the middle block and was half the area of the second layer. In this fashion, all three layers were visible from a top-down perspective for characterization purposes. The model is shown in figure 23.

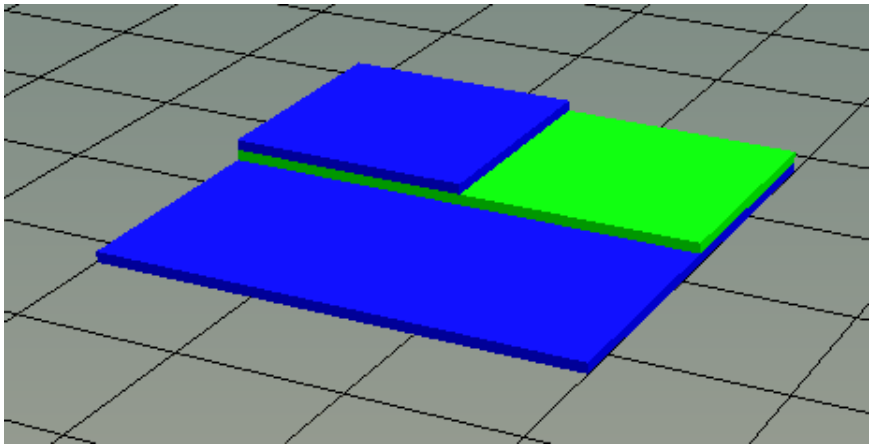
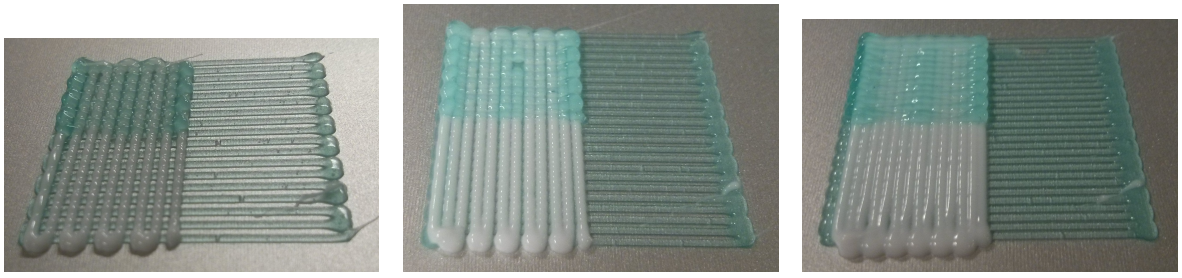


Figure 23: Scaffold model. Object dimensions: 3 cm by 3 cm.

Figure 24 shows the experiment performed with a nozzle diameter of 0.650 mm and speed of 10 mm/sec with infill densities of 30%, 40%, and 50%. A maximum of 50% was used since, at this infill density, the strands extruded by a 0.650 mm nozzle congeal into a single object as observed in figure 24c.

The first layer of each scaffold was constructed using Aim (green) toothpaste, the second layer was constructed using Pepsodent (white) toothpaste. The third layer was constructed using Aim toothpaste again. In this fashion, the multiple material capability was demonstrated while maintaining distinguishability between layers.

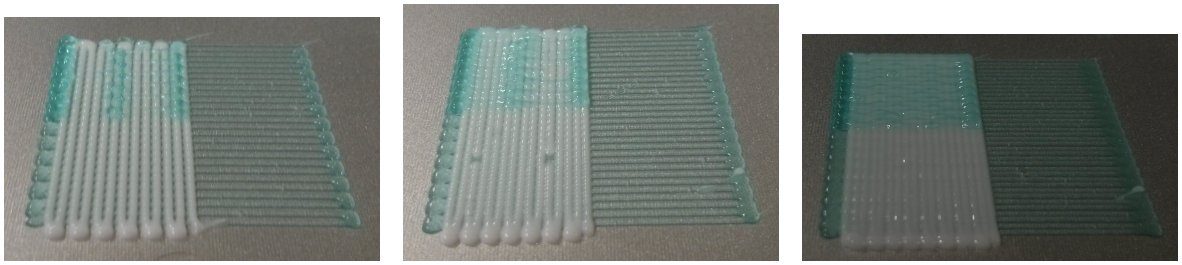


(a) 30% infill (b) 40% infill (c) 50% infill

Figure 24: Scaffolds with nozzle diameter 0.650 mm. Object dimensions: 3 cm by 3 cm.

Figure 25 shows the experiment performed with a nozzle diameter of 0.468 mm and a speed of 10 mm/sec with infill densities of 50%, 60%, and 70%. A maximum of 70% was used since, at this infill density, the strands extruded by a 0.468 mm nozzle congeal into a single object, as observed in figure 25c.

The congealing occurs at different infill densities for different nozzle diameters because of the different settling characteristics of the different sized strands. The strand extruded with the 0.650 mm nozzle diameter will be the tallest of the three and thus have the most weight being placed on its surface contact area. This causes the strand to settle differently than it would with a smaller geometry.

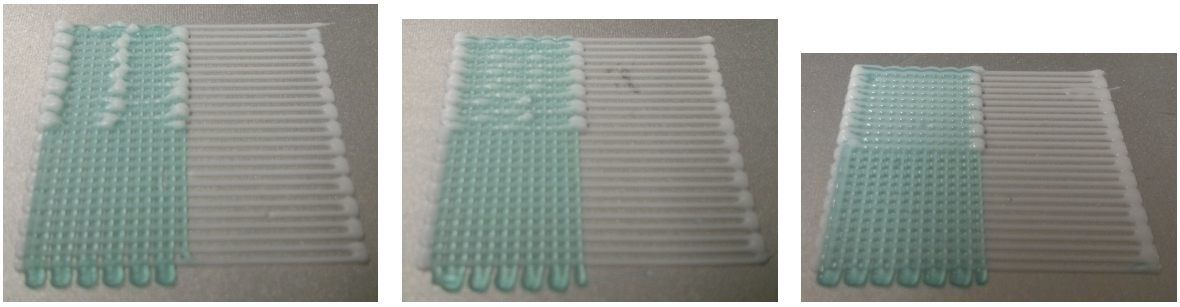


(a) 50% infill (b) 60% infill (c) 70% infill

Figure 25: Scaffolds with nozzle diameter 0.468 mm. Object dimensions: 3 cm by 3 cm.

Figure 26 shows the experiment performed with a nozzle diameter of 0.468 mm, an infill density of 50%, and speed of 10 mm/sec, with layer heights of 100%, 90%, and 80% of the nozzle diameter. At 100% and 90% layer heights, the third layer of

the scaffold displays an issue with adhesion to the second layer, as shown in figures 26a and 26b. This occurs because the toothpaste settles and spreads slightly as it is being extruded. At 80% layer height, adequate surface adhesion to the previous layer is achieved as shown in figure 26c.



(a) 100% layer height (b) 90% layer height (c) 80% layer height

Figure 26: Scaffolds with nozzle diameter 0.468 mm and infill density 50%. Object dimensions: 3 cm by 3 cm.

In this research, stepper motors were used in order to push the toothpaste through the syringe and out of the nozzle. Since the toothpaste is a viscous material, it has some compressibility, and a certain amount of pressure must be built up in the syringe before toothpaste will come out of the nozzle. This means that there will always be a small delay between the time that the stepper motor begins moving and the time that the toothpaste begins extruding.

This issue was addressed by adding custom G-code to Slic3r which requires the robot to return to the home position at every tool change command. During this pause, the active syringe is depressurized by retracting the plunger by a small amount. Subsequently, the next tool is activated and a small amount of toothpaste is extruded in order to build the pressure in the syringe.

Between extrude commands, the motor is deactivated but the syringe remains pressurized. Consequently, a small amount of toothpaste continues to be extruded. To minimize the effects of this issue, the travel speed is set to a very high value in Slic3r. This minimizes the amount of time where the syringe is idle between extrude

commands.

At the end of each strand included in the scaffold, it is observed that there is a small amount of excess material which causes the ends of the strands to appear rounded. However, as shown in figure 27, the ends of the strands should be square. This happens because the robot decelerates and comes to a stop at the end of each strand before changing direction and accelerating again at the beginning of the next strand. During this small amount of time, the robot is operating at a slower speed. The stepper motors are turned off once the robot comes to a full stop, but due to the compressibility of the toothpaste, the syringe remains pressurized. This pressure causes some build-up of excess material.

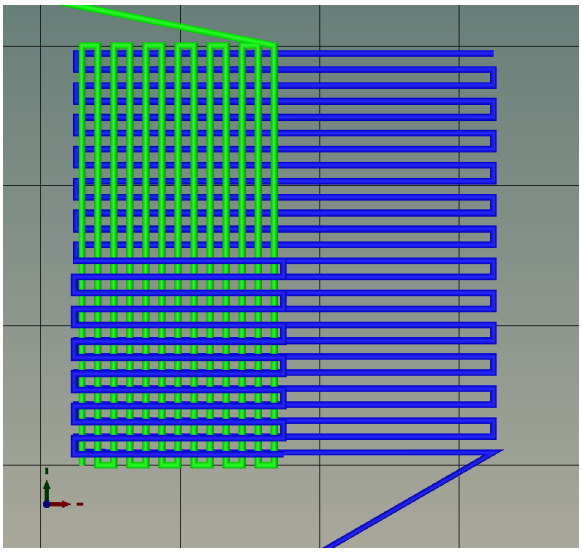


Figure 27: Tool path visualization CAD model

For long pauses, the robot is returned to its home position, where the plunger is retracted to allow the syringe to depressurize. In order to repressurize the syringe at the home position, a small amount of material is extruded. This method of controlling the extrusion of excess material is not feasible to implement at every direction change, as this would greatly increase printing time.

The scaffold with nozzle diameter 0.468 mm, infill density of 50% and layer height of 80%, shown in figure 26c, was imaged under a microscope. The first layer (vertical

strands) was constructed using Pepsodent toothpaste and the second layer (horizontal strands) was constructed using Aim toothpaste. The measurements taken from the section of the scaffold with two layers exposed are shown in figure 28.

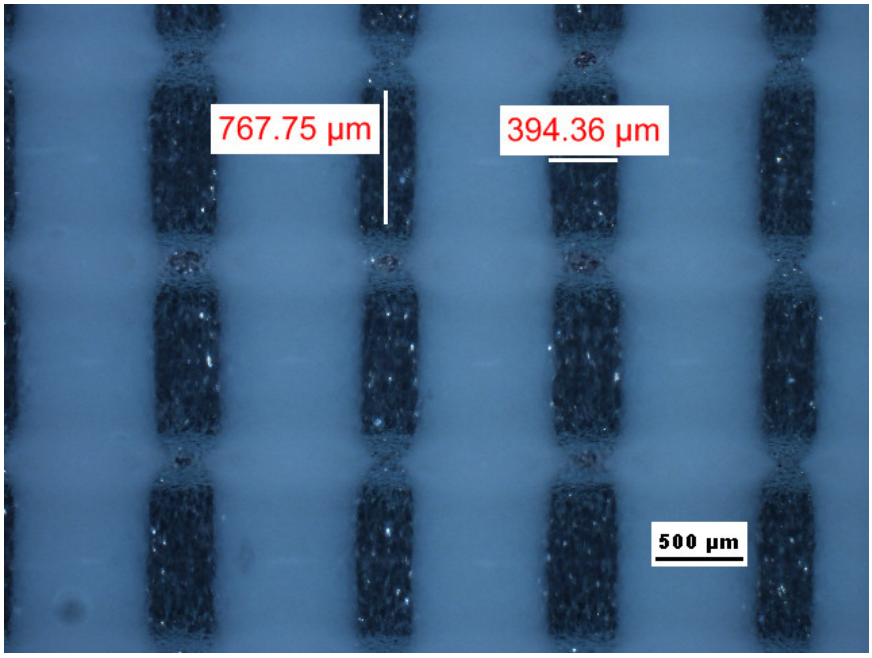



Figure 28: Measurement of scaffold pore size

Both layers were printed with the same infill density so, in theory, the pore dimensions should be equal. However, as shown in figure 28, this does not match the results of this experiment. The experimental results show a rectangular pore size.

There are multiple possible reasons for this discrepancy. Each layer of the scaffold is supported by the previous layers of the toothpaste material. With each subsequent layer, additional weight is placed upon the layers below, thus compressing these layers. The strands of the layers continue to spread as the weight increases. Thus, the height of the strands of the lower layers will be smaller than the height of the strands of the upper layers because they are supporting a larger portion of the weight of the structure.

The first layer (vertical strands) was printed directly on the sandblasted aluminum surface. The second layer (horizontal strands), however, was printed on top of the

first layer. This difference may have influenced the settling characteristics of the layers.

Another possible influence could have been the effects of the different brands of toothpaste used for these layers. Each toothpaste brand has a slightly different viscosity and thus different settling characteristics.

In order to verify the results of the analysis performed in the methodology section, the parameters used in this experiment were substituted into the final equations obtained from the previous calculations. Substituting the nozzle diameter of 0.468 mm and the layer height of 80% of the nozzle diameter into the rectangular approximation, the strand width approximation shown in equation 6 is obtained.

$$W = \frac{\pi}{2n}R = \frac{\pi}{2 \times 0.8} \cdot \frac{0.468mm}{2} = 0.459mm \tag{6}$$

Making the same substitutions in the equation for the composite approximation, the strand width approximation shown in equation 7 is obtained.

$$W = \left(\frac{\pi - \pi n^2}{2n} + 2n\right) R = \left(\frac{\pi - \pi(0.8)^2}{2 \times 0.8} + 2 \times 0.8\right) \frac{0.468mm}{2} = 0.539mm \tag{7}$$

Using both of these width approximation values, each paired with an infill density of 50%, two different pore size approximations are obtained. The pore size obtained using the rectangular approximation is 0.459 mm and the pore size obtained using the composite approximation is 0.539 mm. Since the infill density was defined to be 50%, the pore size obtained is expected to be equal to the strand width used.

Comparing these pore size results to the measured values obtained through the microscope in figure 28, it is observed that both of the approximated values are between the pore dimensions measured of 0.394 mm for the Pepsodent strands and

0.768 mm for the Aim strands. However, without being able to observe the actual cross-section of the strands, this approximation method can only be useful as a guide for predicting expected pore sizes.

# 5 Conclusions

In order to be able to perform additive manufacturing tasks using industrial robots, a methodology was developed and implemented, including an algorithm to convert additive manufacturing code to a language used by industrial robots. This algorithm was written to coordinate the motion platform of the robot with the actuation of plunger based syringes for extrusion.

In addition, a voltage regulation circuit was created to facilitate the communication of the robot controller with the stepper motor controller. A process was developed to accommodate the large file sizes used in additive manufacturing processes.

Multiple extruder capability was demonstrated through an experiment with letter fabrication. The extrusion process was characterized through experiments with strand widths and scaffold pore sizes.

## 5.1 Recommendations for Future Work

In the future, it would be beneficial to expand the algorithm functionality to include other 3-D printing modalities such as heated extrusion and photo-polymerization. The algorithm could also be adapted to include support for multiple simultaneous nozzle diameters.

A better approximation of the strand width should be obtained by observing the cross-section of the strand without disturbing its geometry. It is recommended to examine the strands with the aid of a profilometer. Additional experiments should be performed to better understand the flow of the viscous material in a scaffold.

If it was possible to access the low-level machine commands used by the robot, it would also be beneficial to fine-tune the coordinated movement of the robot with the syringe actuation by anticipating the acceleration and deceleration of the robot and adjusting the stepper motor speed accordingly.

# A Rectangular Geometry Approximation

$$
\begin{aligned}
H &= 2R_0 n \\
A_1 &= WH \\
W &= \frac{A_1}{H} \\
A_1 &= A_0 \\
W &= \frac{A_0}{H} \\
&= \frac{\pi}{2n} R_0
\end{aligned}
\tag{8}
$$

# B Composite Geometry Approximation

$$
\begin{aligned}
W &= L + 2r \\
A_2 &= LH + \pi r^2 \\
L &= \frac{A_2 - \pi r^2}{H} \\
W &= \frac{A_2 - \pi r^2}{H} + H \\
r &= \frac{H}{2} \\
W &= \frac{A_2 - \pi (\frac{H}{2})^2}{H} + H \\
&= \frac{4A_2 - \pi H^2}{4H} + H \\
H &= 2R_0 n \\
W &= \frac{4A_2 - \pi (2R_0 n)^2}{4(2R_0 n)} + 2R_0 n \\
&= \frac{A_2 - \pi R_0^2 n^2}{2R_0 n} + 2R_0 n \\
A_2 &= A_0 \\
W &= \frac{A_0 - \pi R_0^2 n^2}{2R_0 n} + 2R_0 n \\
A_0 &= \pi R_0^2 \\
W &= \frac{\pi R_0^2 - \pi R_0^2 n^2}{2R_0 n} + 2R_0 n \\
&= \frac{\pi R_0 - \pi R_0 n^2}{2n} + 2R_0 n \\
&= \left( \frac{\pi - \pi n^2}{2n} + 2n \right) R_0
\end{aligned}
\tag{9}
$$

# C Strand Width Experiment Results

| Run | Speed (mm/sec) | Diameter (mm) | Average Width (mm) |
|---|---|---|---|
| 1 | 12 | 0.468 | 0.74 |
| 2 | 12 | 0.566 | 0.84 |
| 3 | 8 | 0.468 | 0.89 |
| 4 | 12 | 0.650 | 0.74 |
| 5 | 12 | 0.468 | 0.71 |
| 6 | 12 | 0.566 | 0.70 |
| 7 | 8 | 0.468 | 0.74 |
| 8 | 10 | 0.650 | 0.73 |
| 9 | 10 | 0.468 | 0.80 |
| 10 | 12 | 0.468 | 0.78 |
| 11 | 8 | 0.566 | 0.93 |
| 12 | 8 | 0.650 | 0.89 |
| 13 | 10 | 0.468 | 0.97 |
| 14 | 10 | 0.566 | 0.82 |
| 15 | 12 | 0.650 | 0.88 |
| 16 | 10 | 0.650 | 0.85 |
| 17 | 8 | 0.650 | 0.83 |
| 18 | 8 | 0.566 | 0.84 |
| 19 | 12 | 0.566 | 1.02 |
| 20 | 10 | 0.566 | 0.86 |
| 21 | 8 | 0.566 | 0.89 |
| 22 | 8 | 0.468 | 1.12 |
| 23 | 12 | 0.650 | 0.88 |
| 24 | 8 | 0.650 | 1.00 |
| 25 | 10 | 0.566 | 0.88 |
| 26 | 10 | 0.468 | 0.99 |
| 27 | 10 | 0.650 | 0.88 |

# References


[1] 28000 series size 11 stepper motor linear actuator. http://haydonkerk.com/LinearActuatorProducts/StepperMotorLinearActuators/LinearActuatorsHybrid/Size11LinearActuator/tabid/75/Default.aspx. Accessed: 2016-02-02.

[2] Accelstepper. http://www.airspayce.com/mikem/arduino/AccelStepper/. Accessed: 2016-02-02.

[3] Arduino uno. https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed: 2016-02-02.

[4] Design-expert software version 10. http://www.statease.com/dx10.html. Accessed: 2016-04-13.

[5] Irbcam: Cad/cam for industrial robots. http://www.irbcam.com/. Accessed: 2016-05-02.

[6] Omron adept technologies, inc. http://www.adept.com. Accessed: 2016-04-12.

[7] Stepper motor driver. http://opensourceecology.org/wiki/Stepper_Motor_Driver. Accessed: 2016-02-02.

[8] Using the v+ command line. http://www1.adept.com/main/ke/data/V%20Plus/V%20OS%20User/Basics3.html. Accessed: 2016-02-02.

[9] Empire state building. https://www.thingiverse.com/thing:952373, Accessed: 2016-04-11.

[10] Slic3r: G-code generator for 3d printers. http://slic3r.org/, Accessed: 2016-04-22.

[11] Solidworks. http://www.solidworks.com/, Accessed: 2016-04-22.

[12] T.H Ang, F.S.A Sultana, D.W Hutmacher, Y.S Wong, J.Y.H Fuh, X.M Mo, H.T Loh, E Burdet, and S.H Teoh. Fabrication of 3d chitosanhydroxyapatite scaffolds using a robotic dispensing system. *Materials Science and Engineering: C*, 20(12):35 – 42, 2002.

[13] G. C. Anzalone, C. Zhang, B. Wijnen, P. G. Sanders, and J. M. Pearce. A low-cost open-source metal 3-d printer. *IEEE Access*, 1:803–810, 2013.

[14] Norman Hack, Willi Lauer, Silke Langenberg, Fabio Gramazio, and Matthias Kohler. Overcoming repetition: Robotic fabrication processes at a large scale. *International Journal of Architectural Computing*, 11(3):285–300, 2013.

[15] V. Helm, S. Ercan, F. Gramazio, and M. Kohler. Mobile robotic fabrication on construction sites: Dimrob. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4335–4341, Oct 2012.

[16] Seok-Jung Hong, Ishik Jeong, Kyung-Tae Noh, Hye-Sun Yu, Gil-Su Lee, and Hae-Won Kim. Robotic dispensing of composite scaffolds and in vitro responses of bone marrow stromal cells. *Journal of Materials Science: Materials in Medicine*, 20(9):1955–1962, 2009.

[17] Steven Keating and Neri Oxman. Compound fabrication: A multi-functional robotic platform for digital design and fabrication. *Robotics and Computer-Integrated Manufacturing*, 29(6):439 – 448, 2013.

[18] S. Lim, R.A. Buswell, T.T. Le, S.A. Austin, A.G.F. Gibb, and T. Thorpe. Developments in construction-scale additive manufacturing processes. *Automation in Construction*, 21:262 – 268, 2012.

[19] Tadeusz Mikołajczyk. Robot application to surface finish machining. *J Polish CIMAC*, 5(3), 2010.

[20] John Mortimer. Adhesive bonding of car body parts by industrial robot. *Industrial Robot: An International Journal*, 31(5):423–428, 2004.

[21] P. S. Shiakolas. Robsurf: A near real time olp system for robotic surface finishing. 1999.