Lego 3D Printer

December 3rd, 2019

Group 8-36
Chealsie Bains 20852602 (MTE 100 & GENE 121)
Addesh Bhagwandeen 20761425 (GENE 121)
Owen Brake 20845391 (MTE 100 & GENE 121)
David Sun 20851667 (MTE 100 & GENE 121)

Summary

This report details the production and final design of the 3D printer robot, including both its mechanical and software elements. It details the scope of the design, from file selection and the various objects that can be printed to its main operations and shutdown procedures. Criteria related to the object print are explored, as well as criteria that were laid out that the design should be able to meet. A full breakdown of the mechanical and software aspects of the design is discussed, including but not limited to the x-y and z-actuators and the button pressing mechanism responsible for controlling the extruder releasing the filament. The various software elements of the program are explored, including their implementation and testing. Finally, a verification of the final design was carried out to determine it met both the criteria and constraints, as well as recommendations to pursue in future iterations of such a device.

Acknowledgments

A thanks to Kenneth van Roon for selling Lego rack gears used on the three-axis motors.

Contents

Summary i
Acknowledgments ii
Contents iii
Table of Figures vi
Table of Tables vii
Introduction 1
Scope 1
  Startup 1
    File Selection 1
  Calibration 3
  Main Operation 4
    File Parsing 4
    File Operation 5
  Shutdown 5
    Emergency Shutdown 5
    Standard Shutdown 6
Constraints and Criteria 7
  Criteria 7
  Constraints 7
Mechanical Design and Implementation 8
  Overall Design Description: 8
    X-Y Actuators 9
    Design Alternatives 10
  Z Actuator 13
    Design Alternatives 14
  Baseplate 15
    Design Alternatives 16
  Button Pressing Mechanism 17
Software Design and Implementation 17
  Structure 17
    UI 18

File I/O 18
Actuation...19
Error Handling...21
Demonstration Task List 21
Testing 22
Unit Testing...22
Integration Testing 23
Program Decisions 23
Changes...24
Significant Issues 25
Inaccuracy of Motion Control..25
Loading Files Automatically 25
Constant File Array 26
Formatting Errors when Loading Files..26
Verification 26
Project Plan 26
Mechanical 26
Software 27
Conclusions 27
Recommendations 27
Mechanical 27
Heated Base 27
Thinner Diameter Filaments..28
Continuous Refill...28
Gear Ratios..28
Finer Tooth Rack Gear..28
Software 28
Control Algorithms 28
G-Code Commands 28
References 29
Appendix A 30
Robot Code 30
Appendix B 42

Original Flow Charts 42
Finished Project Flow Charts 45
Appendix C 50
Project Plan 50
Appendix D 51
File Driver Code 51
Rectangular Prism 51
Triangular Prism..52
Shutdown Test 53

Table of Figures

Figure 1. EV3 brick with center button highlighted in red, left and right buttons highlighted in blue [2]........ 1
Figure 2. Rectangular prism file screen........ 2
Figure 3. Rectangular prism printed by Lego 3D printer........ 2
Figure 4. Triangular prism file screen........ 3
Figure 5. Triangular prism printed by 3D printer........ 3
Figure 6. Shutdown test file screen........ 3
Figure 7. Printer with calibration sensors highlighted in blue........ 4
Figure 8. EV3 brick with up button highlighted in green [2]........ 6
Figure 9. User thrown error screen........ 6
Figure 10. Standard shutdown screen........ 7
Figure 11. Side view of Y actuator........ 9
Figure 12. X actuator mounted to rack gears........ 10
Figure 13. 3D printer with 3 axis movement on the extruder [1]........ 12
Figure 14. 3D printer alternative design with unified X-Y axis [1]........ 13
Figure 15. Z actuator mounted in the printer........ 14
Figure 16. Delta 3D printer [3]........ 15
Figure 17. Failed prints........ 16
Figure 18. Pen button actuator........ 17
Figure 19. Original high-level flow chart........ 42
Figure 20. Original calibration module flow chart........ 43
Figure 21. Original extrusion module flow chart........ 44
Figure 22. Original button module flow chart........ 45
Figure 23. Finished project high level flow chart........ 46
Figure 24. Finished project calibration module flow chart........ 47
Figure 25. Finished project extrusion module flow chart........ 48
Figure 26. Finished project button module flow chart........ 49
Figure 27. Original project plan GANTT Chart........ 50

Table of Tables

Table 1. Project Constraints [1] 8
Table 2. X, Y Actuator Design Alternatives..11
Table 3. Baseplate alternatives 16
Table 4. UI functions 18
Table 5. File I/O functions 19
Table 6. Actuation functions 20
Table 7. Error handling functions...21
Table 8. List of Unit tests performed..22
Table 9. Integration tests performed..23
Table 10. Software design decisions..24
Table 11. Changes to program structure 25

Introduction

As mentioned in the preliminary report “Our project, the 3D printer, will allow people to manufacture their own unique products quickly, cheaply, and conveniently. 3D printers are machines that receive digital file inputs and autonomously manufacture 3D objects … This technology possesses a variety of uses in industry, from rapid-prototyping fire-retardant plastics in the aerospace engineering field to creating customized parts for medical devices … In 2014, NASA even used 3D printing technology to email the digital file for a socket wrench for use on the ISS … It is evident that 3D printing technology has stakeholders in numerous fields, because it makes manufacturing faster, lowers labour costs, and can be used in any location.” [1].

Scope

The actual printer operation very closely matches the task list described in the scope from the preliminary design report. Just as planned in the preliminary report the robot consists of “the start-up, the main operation, and the shutdown” [1].

Startup

File Selection

When the robot is initialized the user is provided with a list of files to select from. The user can scroll left and right, then select the object using the center button as shown in Figure 1.

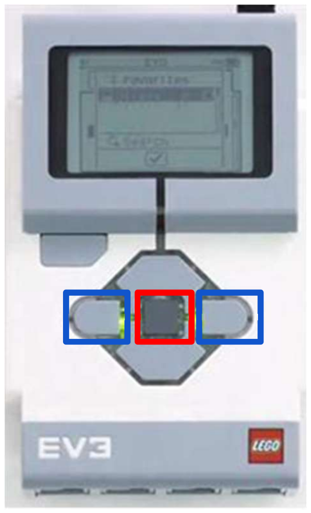


Figure 1. EV3 brick with center button highlighted in red, left and right buttons highlighted in blue [2]

There are three files to select from, Rect. Prism, Tri. Prism and Shutdown Test. For each file, there is a title and a description underneath it.

Rectangular Prism

The rectangular prism file screen is shown in Figure 2. The rectangular prism file produces a rectangular prism vertically of dimensions 22mm x 22mm x 22mm as shown in Figure 3. The rectangular prism was chosen due to the relative simplicity of the shape and its compatibility with cartesian 3D printers, which excel in printing straight lines.

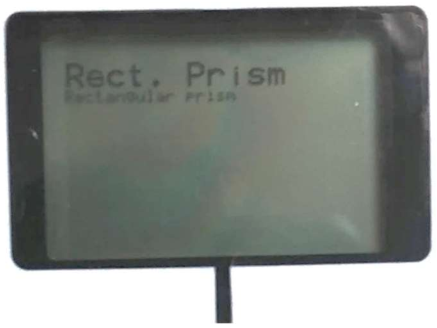


Figure 2. Rectangular prism file screen

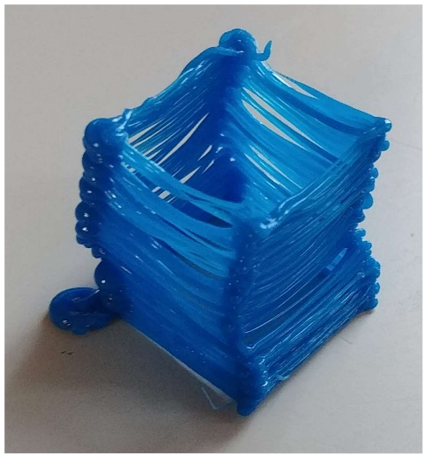


Figure 3. Rectangular prism printed by Lego 3D printer

Triangular Prism

The triangular prism file screen is shown in Figure 4. The triangular prism file produces a triangular prism vertically of dimensions 22mm x 22mm x 3mm as shown in Figure 5. The triangular prism was written to demonstrate the speed ratios and motion control algorithm.

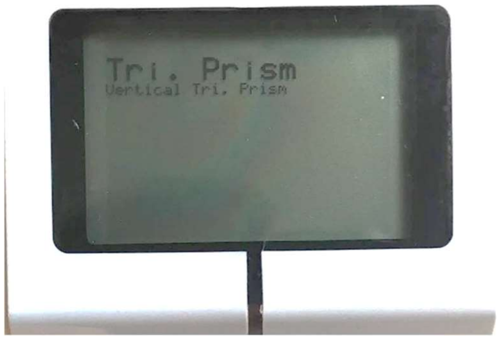

Tri. Prism
Vertical Tri. Prism

Figure 4. Triangular prism file screen

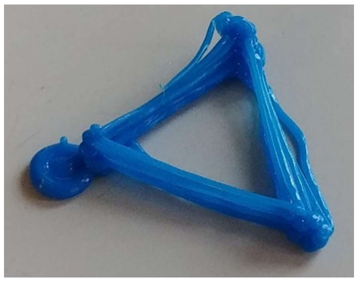


Figure 5. Triangular prism printed by 3D printer

Shutdown Test

The shutdown test file screen is shown in Figure 6. The shutdown test file is a file with an identifier to shut down the file and perform no operations. This was implemented for debugging as well as a quick demonstration of the shutdown phase.

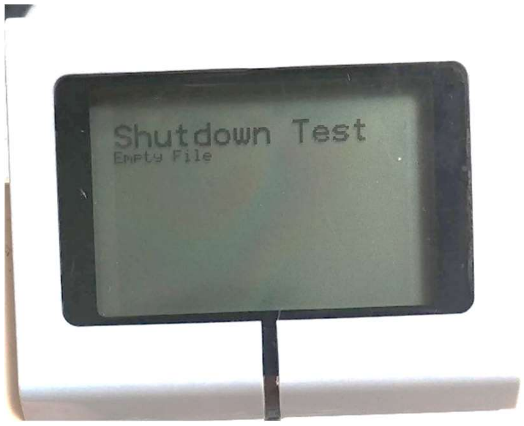

Shutdown Test
Empty File

Figure 6. Shutdown test file screen

Calibration

In this phase, the x, y, and z actuators moved towards the extreme positions in their axis until the respective touch sensors were hit as seen in. This was done to make each print consistent because the motor encoders would always start with reference to an absolute position.

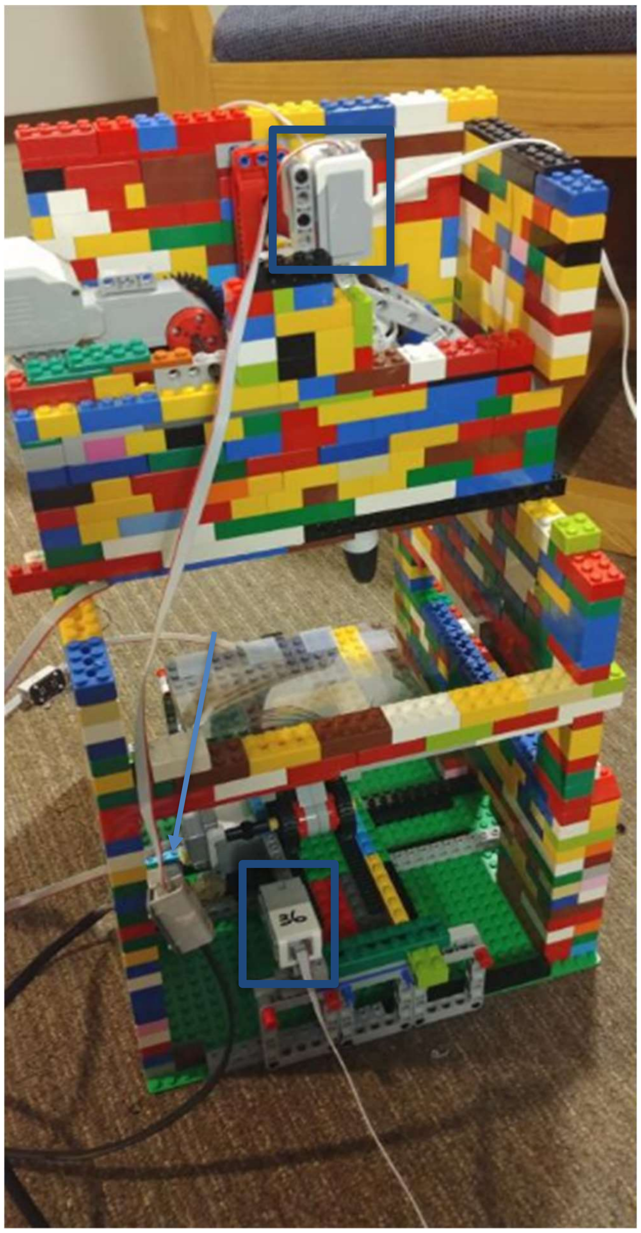


Figure 7. Printer with calibration sensors highlighted in blue

Main Operation

File Parsing

The major difference between the preliminary design report and the final working product was the order in which File Parsing occurs. In the preliminary design report file, parsing occurs before calibration [1]. In the final product, each operation is loaded in as it happens. There are four operations that the file can load, a coordinate to translate to in the X, Y direction, a coordinate to translate to in the z-direction, toggling of the extrusion and exiting of the program.

File Operation

The Extrusion step from the preliminary design report was relabeled to File Operation in our final design [1]. The step was renamed because this step consists of operations on the 3D printer including but not limited to extrusion. In this phase, the operations are performed as described in the File Parsing step.

Translation

The system utilizes rudimentary motion control to navigate to each coordinate position when translating as described in the Software Design component.

Toggle Extrude

When the toggle extrude option is read in then the robot simply actuates the pen motor to press the extrude button on the pen and release after a defined set of time to toggle the extrusion.

Exit

The exiting of program operation causes the safe shutdown of the file operation phase. This operation was implemented to certify the end of the file, is helpful for debugging and allows the shutdown test file to be implemented.

After each file operation, the robot returns to the File Parsing stage until the end of the file is reached or the exit program operation occurs.

Shutdown

Emergency Shutdown

The emergency shutdown mode is activated upon pressing the up button as shown in Figure 8. Upon entering the emergency shutdown, All the motors on the robot are stopped and a display saying, “User Exit” and “Closing in 5s” is shown to the user as shown in Figure 9. The program then closes after 5 seconds.

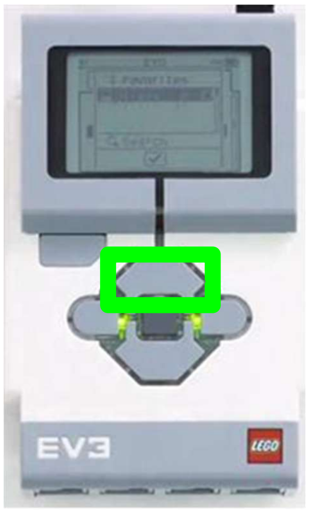


Figure 8. EV3 brick with up button highlighted in green [2]

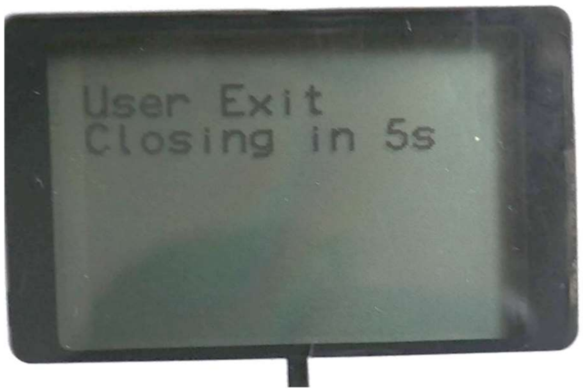
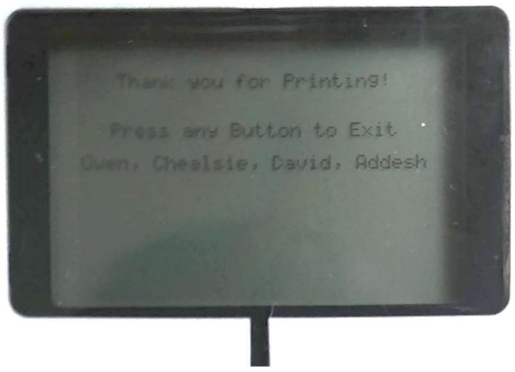

User Exit
Closing in 5s

Figure 9. User thrown error screen

The emergency shutdown tasks replaced the reset task from the preliminary design report [1]. The emergency shutdown task was added as it is important for the safety of the user and to prevent damage to the project if any errors occur.

Standard Shutdown

In standard shutdown, the robot moves all the motors to the calibration position. The EV3 brick then displays to the user a message asking the user to press any button to exit the program as shown in Figure 10. The program then exits on the press of any button.



```
Thank you for Printing!

Press any Button to Exit
Owen, Chealsie, David, Addesh
```

Figure 10. Standard shutdown screen

Constraints and Criteria

Criteria

With regards to the criteria outlined in the previous report, one significant change from the list was to remove the criteria regarding a simpler design, as at this point a design has already been chosen, namely the Cartesian Fused Deposition Modeling (FDM) design for 3D printers.
Once this design was chosen, the following list of criteria previously established in earlier reports remains unchanged:

- Print Quality - This is based on the accuracy of the object with respect to the dimensions of the digital file. A higher print quality is defined by a closer representation of the 3D model.
- Print Speed - This criterion measures the time spent on an object to be printed. A high print speed means less time for the object to be printed.
- Print Volume - The maximum volume of the object being printed. Systems with high print volumes are capable of producing large parts which in turn means they can be used to make more complex objects.

Constraints

Constraints from the previous report remain unchanged as shown in Table 1.

Table 1. Project Constraints [1]

Constraint	Justification	Method of Testing
The 3D printed object must not stick to the platform.	3D printers utilize filament which when heated in the printing process can act like glue. Should the object be stuck to the surface, removal may cause damage to its structure.	Printing multiple test objects and attempting to remove them can help determine how they stick to the surface of the plate.
The distance between any two design features in the input file cannot be smaller than 1.75mm.	The filament diameter is 1.75mm, therefore it cannot make features smaller than this.	Implementing error checking in the code to display an error when two design features less than 1.75mm apart are detected and to not print the object.

One key constraint that was helpful was that the object must not stick. Resolving this required an alteration of the physical design. Previously, cardboard was used as the surface of the baseplate. However, the object sometimes stuck to this type of surface causing an unclean finish. To prevent this a plastic covering was used instead, allowing the object to be safely removed without damaging the print or the baseplate. Additionally, it was found through testing that printing some filament in one spot at the start ensured that the print would stick to the baseplate during extrusion. Thus this was implemented into the final software design.

Print quality and print speed were important in the final project. The print volume was initially thought to be a source of improvement but due to lack of components, the print volume could not be improved. Thus print speed and accuracy were focused on as there were numerous ways in which we could improve them.

Mechanical Design and Implementation

Overall Design Description:

Our printer was comprised of 3 main design features, the actuators which allowed us to move along a 3D coordinate system, the baseplate on which we printed, and the button pressing mechanism which was used to turn the 3D pen on and off. The actuators can further be split into the x-y actuator which had the same methods of gearing and sensor placement, and the z-actuator. In total, we used 4 motors, (1 for each actuator and one for the button pressing), 3 touch sensors, and a variety of Lego to build the outer structure and baseplates.

X-Y Actuators

The x and y actuators were made by combining linear gears (also known as rack gears) shown with rotational gears. We started by attaching two 20-tooth gears and two 36-tooth gears to two large motors as seen in Figure 11. This size was chosen because when the motor was placed flat on the floor, the platform lying on top would be parallel to the ground.

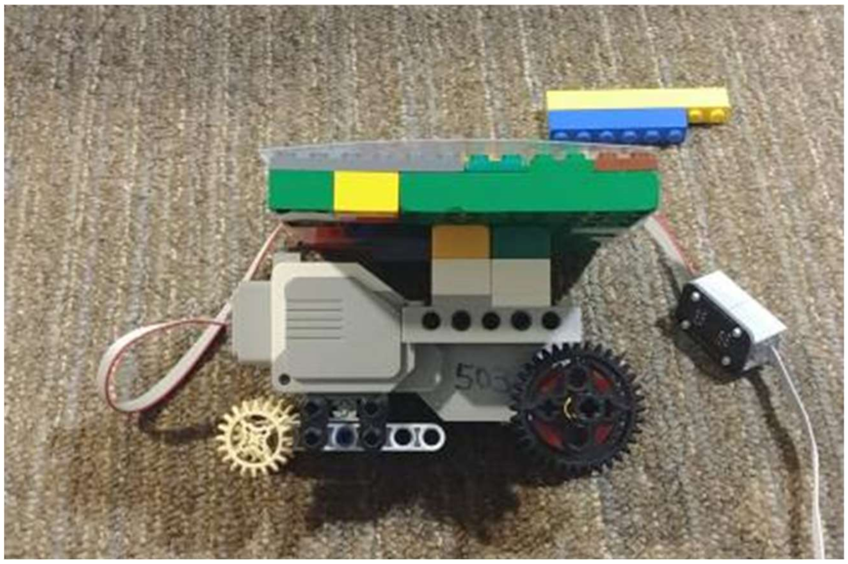


Figure 11. Side view of Y actuator

We then placed 2 rows of linear gears on the bottom of the plate so the motor could “drive” back and forth. This essentially created one axis of movement. The second axis was created by attaching a long platform on top of the bottom motor and adding rack gears along as is seen in Figure 12.

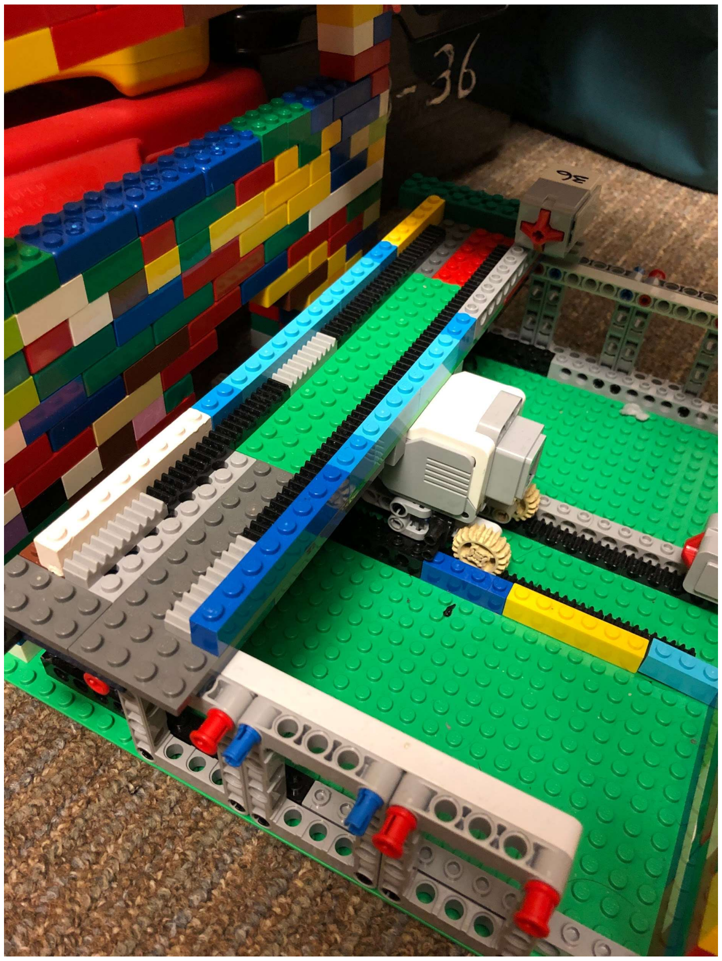


Figure 12. X actuator mounted to rack gears

Finally, the second motor was placed on top of this platform to account for 2 degrees of motion. Touch sensors were placed at two ends of each axis as highlighted in Figure 7. to allow for calibration.

Design Alternatives

There were some design alternatives that were considered when making and designing the X, Y actuators as shown in Table 2.

Table 2. X, Y Actuator Design Alternatives

Design Alternatives	Pros	Cons
Having the pen be actuated in the X-Y direction rather than the baseplate as seen in Figure 13.	Would create faster prints theoretically because the pen is lighter than the base and therefore can be actuated at a faster speed. Would provide a greater print range because the pen is smaller than the baseplate, and therefore has a greater range of motion within the rectangular structure than the design used	Complex to build because having motors suspended in the air and trying to create motion in 3 directions at once is harder than separating the x-y and z actuators. Maybe less precise with the materials which we were working with, because overshoot in any one direction would affect the outcome in all 3 axes.
Having the X-Y axis in one plane using slides rather than stacking the two motors as seen in Figure 14.	Requires less Lego to build and as a result provides a bigger print range in terms of height. This is because less vertical space is taken up by the x and y axes.	Harder to build as it would require the use of slides. Since we were using the Lego motors rather than Tetrix, it would be difficult to obtain appropriate Lego slides.

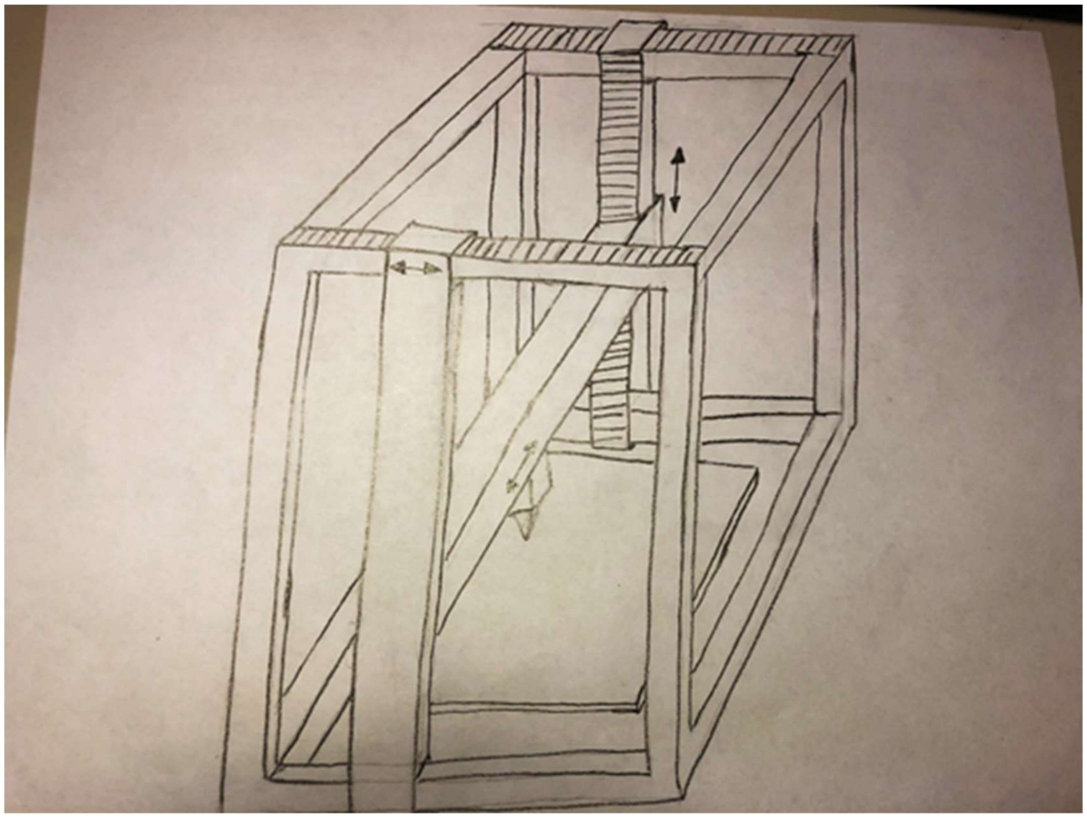


Figure 13. 3D printer with 3 axis movement on the extruder [1]

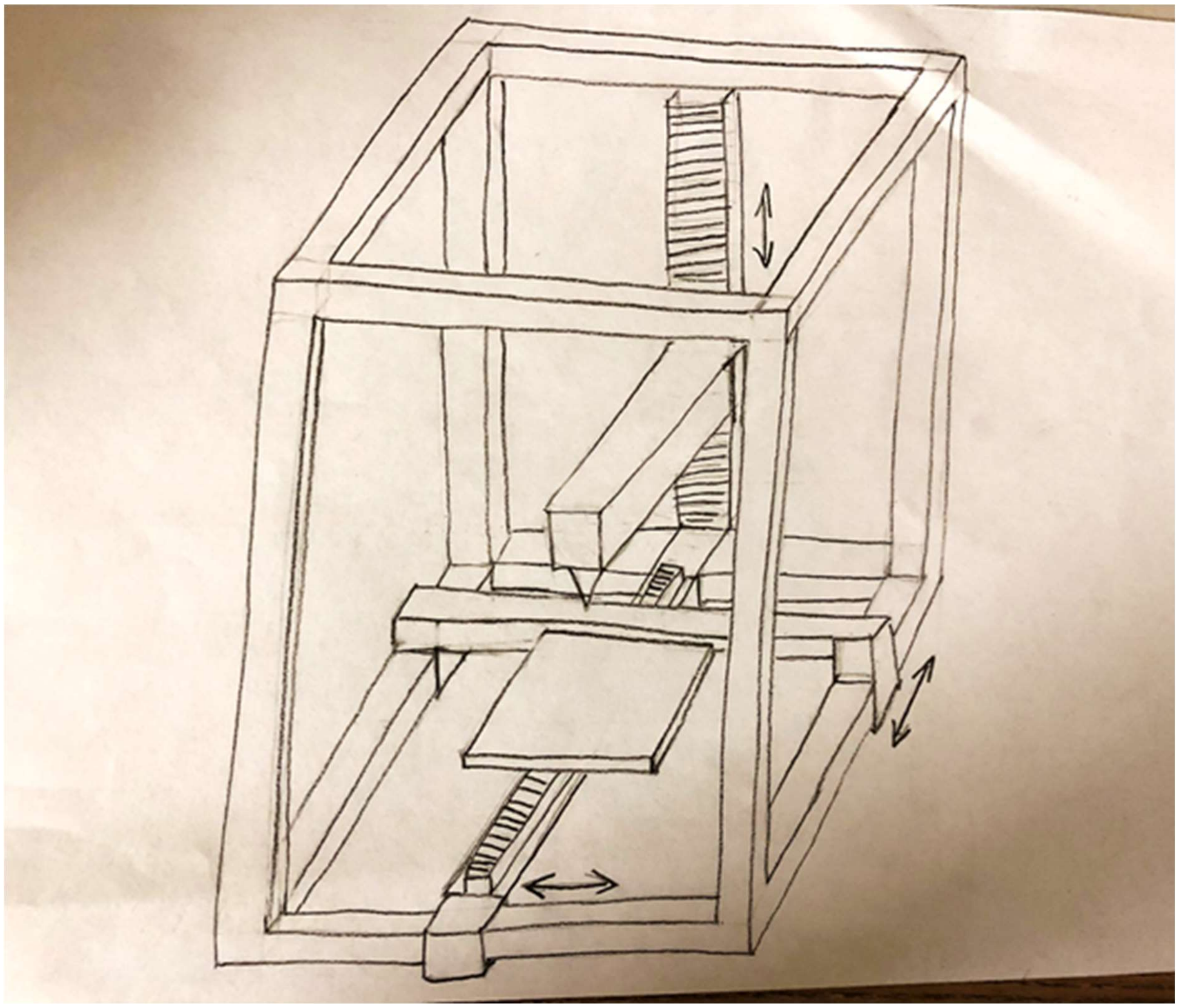


Figure 14. 3D printer alternative design with unified X-Y axis [1]

In the end, the design that we used was chosen for simplicity [1]. We were given a time constraint, so it was more important to have a working project by the end than having the best version of a 3D printer. We also concluded that we could sacrifice print volume or print time for a simpler design because we could still complete the minimum success criteria from the interim report (to create basic shapes) with the simpler design.

Z Actuator

The z-actuator was made by using a rack gear and slide attached to a motor with a gear ratio. This differed from the X-Y actuator because in this case the rack gear was attached to the pen and moved whereas the rotational gears remained in a stationary position as seen in Figure 15.

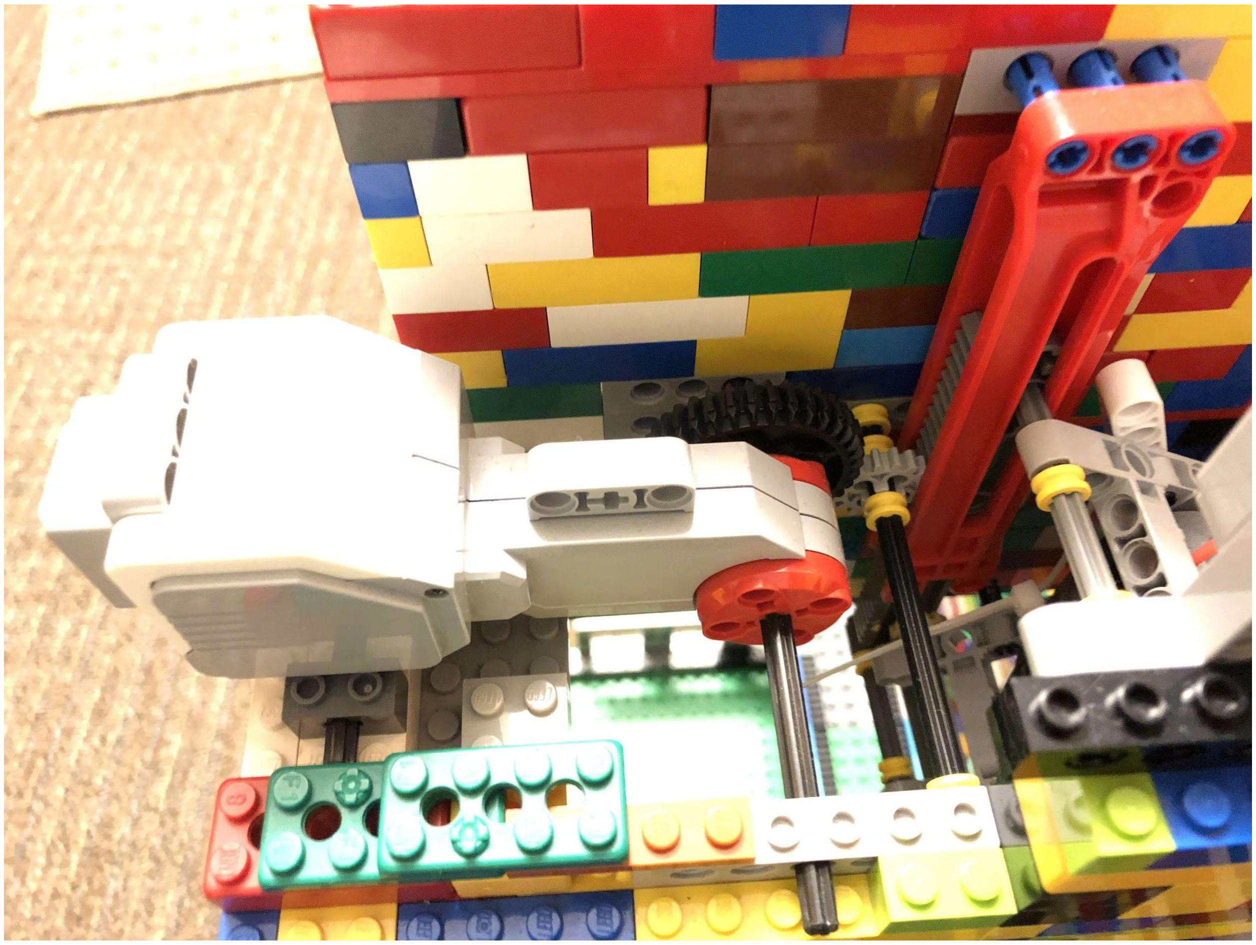


Figure 15. Z actuator mounted in the printer

The gear ratio was made because no single gear could span perfectly from the mounted motor to the rack gear. Similar to the X, Y actuators, there was a touch sensor located at the top to sense when the pen had reached the extreme in the upwards direction.

Design Alternatives

Given that we were using Lego motors, there are not many alternative methods of creating this system. In industry, some printers called delta printers in which 3 arms are affixed to vertical supports in a triangular configuration and move up and down in coordination to create movement in all 3 axes as seen in Figure 16. This design, however, would be too complex to create using Lego parts.

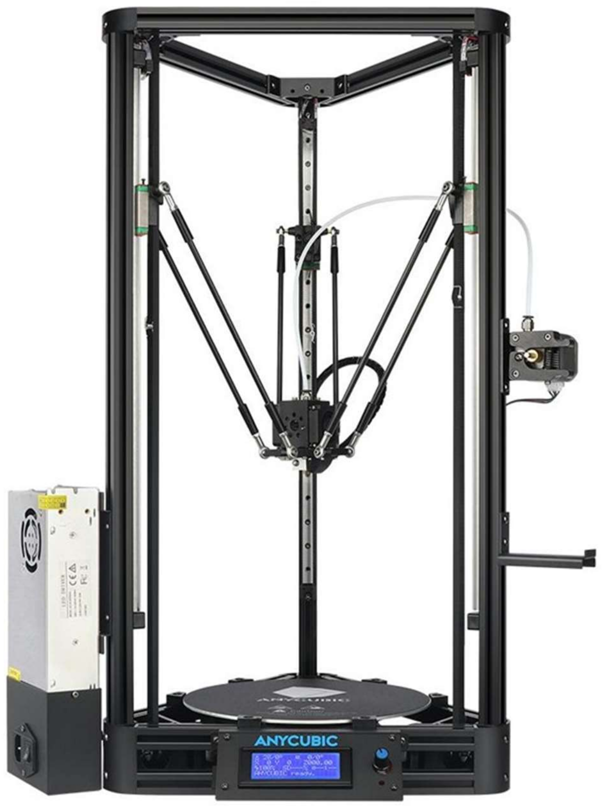


Figure 16. Delta 3D printer [3]

Baseplate

The baseplate was made using a large flat Lego piece with a plastic sheet sitting on top. This was chosen to avoid the filament from sticking to the plate while having a sturdy and flat surface to print on. We faced some problems with this design during testing however, because the surface became too slippery, making it difficult for the filament to stick on. When this failed to happen, prints like the one seen in Figure 17 occurred. To account for this, we started the filament extrusion and paused in one spot for some time to let the filament stick, and then began going through the movements. This created the blob seen in the top right corner of the green printed square.

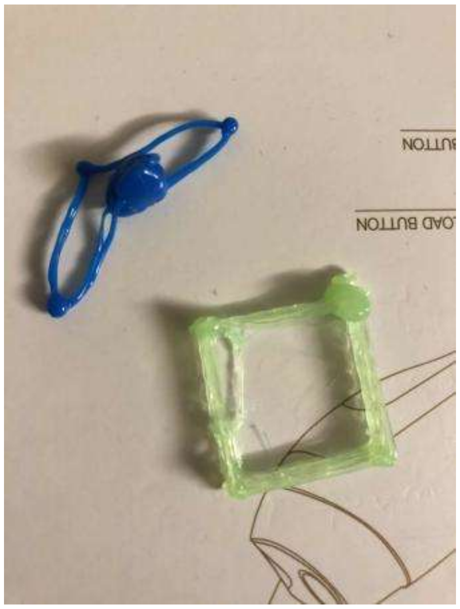


Figure 17. Failed prints

Design Alternatives

There were some alternatives that were considered when choosing the base plate as shown in Table 3.

Table 3. Baseplate alternatives

Design Alternative	Pros	Cons
Using cardboard rather than plastic for the top layer of the baseplate - This was used during some of our testing phases before we switched to plastic.	The prints consistently stuck onto the base plate at the start and did not slide during printing.	When removing the finished product from the plate, some cardboard residue would sometimes remain.
Using glass	Sturdy so would not require Lego bottom making the entire structure lighter and therefore more efficient. Most likely would not have the plastic filament stick.	Can be dangerous to work with and is a more expensive option. The filament may still slide on this surface, these properties have not been tested in our setting.

The plastic top design was chosen in our case because we were required to meet our design criteria “The object does not stick to the platform”. When using the cardboard this could sometimes occur, so we did not want to take that risk. The glass was not used because it would be harder to obtain, more expensive, and potentially cause a safety hazard.

Button Pressing Mechanism

The 3D pen was turned on and off by clicking one button. To do this, we attached our small motor to the same rack gear the pen was attached to as shown in Figure 18. This ensured that as the pen moved up and down, it could be turned on and off at any point.

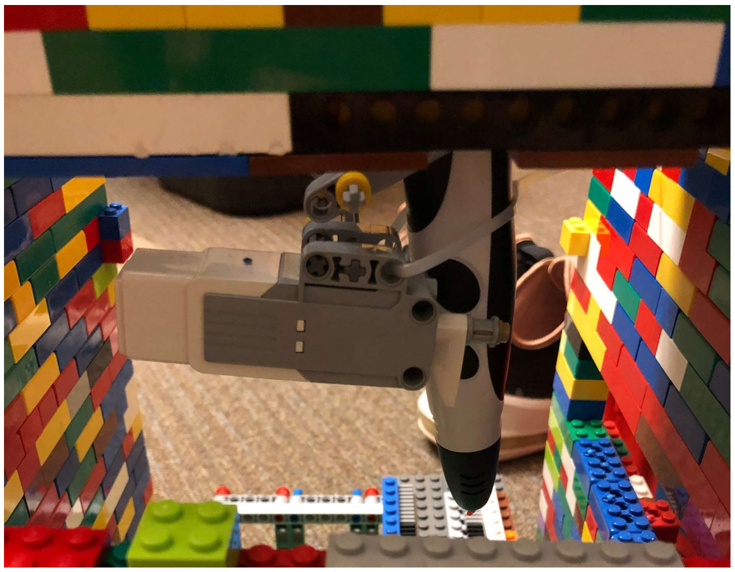


Figure 18. Pen button actuator

A challenge for this design was that there was no second mounting point for the axle in the motor. This could not be done on the Lego wall as the motor moved up and down with the pen, and there was very little room left on the rack gear to accomplish a second mounting point. In the demo, we did not experience any trouble with the axle falling out because we used the shortest possible axle. A design alternative could have been found had we planned the structure holding the pen in one spot better. If we had planned the mounting spots at an earlier time the motor could have been mounted with increased stability, however, the button pressing mechanism never once failed both during demo and testing, so we chose to stay with our design.

Software Design and Implementation

Structure

The program structure can be broken up into 4 different groups: UI, File I/O, Actuation, Error Handling. These 4 groups were broken up due to their distinct differences and ability to program these concurrently. Each of these groups can be programmed to a great extent without a great dependence on each other. All source code and functions referenced are found in Appendix A. All flow charts can be found in Appendix B.

UI

The UI block consists of the different functions which print information to the EV3 brick. There are some other UI actions throughout the code but listed in Table 4 are the main UI functions.

Table 4. UI functions

Function Name	Parameters	Return type	Description	Author
display_utility	File array	integer	Manages the UI for the file manager. The user can scroll left or right and select a file to load. This is the “Select Object” block in Figure 23	David
shutdown	N/A	void	Handles the shutdown operation and in particular the shutdown UI. Outputs final message and waits for a button click to exit program. This represents the larger block of “return to calibration position” in Figure 23.	Chealsie

File I/O

The File I/O block of the code is a significant part of the 3D printer program. The file I/O block manages all the operations related to loading, parsing and managing the operations from the file. Table 5 describes the different functions which make up the File I/O block.

Table 5. File I/O functions

Function Name	Parameters	Return type	Description	Author
load_run_operations	character, position_struct struct	integer	Loops through each operation in a file and handles that operation. This is the function that manages much of the extrusion module as shown in Figure 25. Return type was for debugging purposes.	Owen
get_instruction	character	integer	A simple function that returns the action type given a character from the printing file. This is essentially the identifier manager, assigning values to the characters in Figure 25.	Addesh
write_file	N/A	void	This was a test/driver function to insert or update a file into the brick at initialization. This is not included in the flow charts as it is not part of the main operation. The commands for each object which were inserted into write_file can be found in Appendix D.	Addesh
load_files	File array	void	A simple function which loads the files into the file list array	David

The file struct type is used in some of the File I/O functions and is an important feature of the code. The file struct type contains 3 properties: file name, title and file description. The file struct acts as a descriptor for each file loaded onto the EV3 brick. There is an array of file structs which describes all the files the robot can read and act on.

Actuation

The actuation block is the main feature of the program. The actuation block consists of all the major motor movements of the printer and all the actions in the physical extrusion. The functions relating to the actuation of the 3D printer are shown in Table 6.

Table 6. Actuation functions

Function Name	Parameters	Return Type	Description	Author
move_operation	position_struct struct, integer, integer	integer	This function takes an x and a y coordinate as integers. This function then moves the base of the robot to the x,y coordinate specified. The function uses bang-bang algorithm and also uses speed ratios to handle moving at angles. This is the function for the “Travel to X,Y coordinate” block in Figure 25.	Owen
move_z_axis	position_struct struct, integer	void	This function takes a z-axis and uses bang bang algorithm to raise or lower the pen to a specific z coordinate position. This is the function for the “Move pen along Z axis” block in Figure 25.	Addesh
calibrate_axis	N/A	void	This function handles the entire calibration module in Figure 24. It moves all the motors to their calibration positions.	Owen
actuate_pen	N/A	void	The actuate_pen function toggles the pen either turning it on or off. This is represented in “Turn Pen On/Off” in Figure 25.	Addesh
reset_encoders	N/A	void	Simple function to reset all the motor encoders in the program.	David
load_position_struct	N/A	void	Initializes the position struct	Chealsie
run_program	string	void	Acts as an extension to the main tasks, calling the necessary functions to initialize the program and start the printing process.	Addesh

The position_struct is a struct to store the ideal position of the printer. This was important in debugging when determining the drift or error in the printer actuation and viewing this error helped identify the flaws in the program. The position_struct was also going to be important for more advanced motion control and a better 3D coordinate system but due to time constraints this was not possible but the position_struct object was no deprecated for easy debugging and future extensibility.

Error Handling

Error handling is an important feature of the 3D printer. Error handling is important for identifying issues in the software, debugging and ensuring safety when failures occur. There are various error checking protocols throughout the program but the functions in Table 7. are the main error functions.

Table 7. Error handling functions

Function Name	Parameters	Return type	Description	Authors
error_check	N/A	N/A	Waits for user press of the top button and shuts down all processes. Turns off all motors. Is the button module function as shown in Figure 26.	David
non_user_error	integer	void	Takes in an integer parameter describing the reason for error being thrown. This is the general non user thrown error handler. Stops program and displays an error to the user.	Chealsie

Demonstration Task List

The task list remains relatively the same as the tasks we had planned. Only minor changes were made in particular the specific objects that were produced. The robot was turned on and the rectangular prism file was selected through the file menu UI. The object was produced, and the standard shutdown was demonstrated. Then the robot was restarted, and the triangular prism file was selected. The triangular prism object was shown being created and the up button was pressed to demonstrate the user error thrown.

Testing

The testing protocol for the project could be broken up into unit testing and integration testing.

Unit Testing

Most of the major functions were tested independently using main as a driver function.

Table 8. describes the unit testing protocol for the major functions. Many of the smaller functions were tested by using the debug stream to view their output.

Table 8. List of Unit tests performed

Function Name	Reason for Test	Test Evaluation
write_file	It is important to check that the files are being properly written.	In the write_file function different characters and floats were written to a file. The write_file function was then modified to also read the subsequent file and the output was sent to the debug stream. The debug stream was then read on ROBOTC and the file data was verified to match the input provided
move_operation	Testing the accuracy of the movement mechanism in the x,y axis is important as this controls one of the main features of the printer.	The base plate was moved to a certain position and the move_operation function was called. The distance travelled by the base plate was then measured to verify its accuracy.
move_z_axis	Testing the accuracy of the z axis movement is important as just like the x,y movement it is critical to the operation of the printer	The pen was moved to some middle position and the distance between the ground and the tip of the pen was measured. The pen was then moved by some z distance and the distance measured again. The change in the distance was verified to be as expected.
calibrate_axis	The calibration step is important as if calibration is not performed correctly then the accuracy of the rest of the print will be damaged.	The pen and base plate were placed in different positions and the calibrate_axis function was called to test that the calibrate function still works in different positions. The calibration phase was determined to be finished by a DebugStreamLine call.
actuate_pen	Proper actuation of the pen is important as if the	The actuate pen function was called numerous times in main to ensure that no matter if the pen is

	pen doesn't turn on the printer doesn't work. It is also important for safety reasons.	on or not the pen will still toggle.
display_utility	The program cannot function if the user doesn't get past the UI screen.	The display_utility and load_files function was called in main and the return value was outputted to DebugStreamLine. The tester then selected the different files and viewed the output in the debug stream window to ensure the file was properly selected.

Integration Testing

Integration testing is important to ensure that all the functions work not only separately but also when put together. The entries in Table 9. describe some of the different integration tests performed on the 3D printer.

Table 9. Integration tests performed

Function Names	Reason for Test	Test Evaluation
write_file, load_run_operations	These two functions make up the bulk of the file I/O	A file was written with the different identifiers then the file was read in using load_run_operations. The values read in through load_run_operations were then outputted to the debug stream. The debug stream was compared to the expected input of the file to ensure that the different actions were being properly written and properly read.
write_file, load_run_operations, move_operation, actuate_pen, move_z_axis	All these functions consist of the main operation of the 3D printer, the file I/O and the actuation	Just as in the above test a file was written and read. The actions described in the test file were tested and they were verified to have moved or actuated in the correct manner as described.

Program Decisions

The information in Table 10. describes the design decisions made in the 3D printer program, including the trade-offs and reasoning.

Table 10. Software design decisions

Decision	Reasoning	Tradeoffs
Bang-Bang as the motion control algorithm	Given the time constraints, the bang bang algorithm was chosen for motion control. Bang-bang is quick to implement and requires little tuning.	Inaccurate, high chance of over or undershoot.
Calibration Setup	The calibration step is very important as it guarantees a common starting point for the extruder. The program will not know where the extruder starts without the calibration step.	Calibration takes significant time before the program start.
Absolute Positioning System	Navigating to different positions in the 3D printer's space using absolute positions rather than relative positions is beneficial. It means that the printer can somewhat correct for errors if one movement is off.	Relative positioning system will have a cumulative error but it is consistent while absolute position may cause the shape to warp.
Separating the X,Y movement and Z movement operations	Through testing it was found moving the Z axis at the same time as the X,Y axis caused the filament to lose contact with the existing shape. Thus the Z movement and X,Y movement had to be separated and cannot occur concurrently.	Excess filament is added on the corners of shapes while the pen moves up in the Z direction.

Changes

Much of the overall program structure remains unchanged from the original plan. There are some small differences in the flow charts as shown in Table 11.

Table 11. Changes to program structure

Change	Reasoning
Removed arc and straight line command as in Figure 21. Replaced with stop, move in X, Y and move in Z axis commands as in Figure 25.	Due to time constraints, the arc command was scrapped. The X, Y movement and the Z movement functions were separated as explained in Table 10.
Lowered timeout for calibration from 45 seconds in Figure 20. to 4.5 seconds in Figure 24.	The calibration stage took very little time to complete in practice, 45 seconds was too long to be an acceptable timeout. This was lowered by a factor of 10 to a more acceptable time.
Removed refill filament button from Figure 22	Refill filament button was scrapped due to time constraints.
Made button module a new, concurrent task in Figure 23. rather than consecutive in the extrusion module from Figure 21.	The button module was ineffective running consecutively as emergency shutdown operation only works once the operations had finished thus making it semi concurrent was more effective.

Significant Issues

There were many issues in the production of the software, these issues and their solutions are described below.

Inaccuracy of Motion Control

Various parts of the motion control functions were tuned to get the most accurate and most efficient movement. The speeds were tuned to be fast enough to reach the target but slow enough so as not to overshoot too much. The position tolerance was tuned so that the printer reached a close enough position but did not oscillate around the desired position, never reaching it.

Loading Files Automatically

The traditional and best method for handling the file array would be too scan the brick for all the files and load them, but this was difficult to implement and would take time to load in the file descriptors. Thus it was decided that the array of file should be defined in the program.

Constant File Array

The proper method to store these files in an array would be to use a constant array defined before the main code. There were significant issues initializing a constant struct array with all the files' data. Thus the load_files function was added and the array was no longer constant and each function that required it had to have the function passed into it.

Formatting Errors when Loading Files

Whenever we tried to load actual typed out files onto the EV3 brick via the file transfer there would be issues with the file. When reading the files the program would read extraneous formatting data or just would not be able to load the characters properly. Thus the driver write_file function was created to insert and update files in initialization, thus resulting in perfectly compatible and well-defined files.

Verification

For the two constraints listed, were were able to successfully meet them for the final design. First, with regards to not having the object stick the platform, the platform was covered with a plastic layer that made removal of the object easier, as the heated filament sticks less to the plastic covering when compared to the surface itself, consisting of the LEGO bricks. Secondly, the other constraint was met through a software implementation that checked each instruction read to verify that the distance between any two print features were 1.75mm or more. It should be noted also that the objects available for printing were coded in such a way that there were minimal occurrences of this in the files, thus reducing the likelihood of there being such as error.

Project Plan

The project plan from the preliminary report as shown in Appendix C was followed fairly closely. The project plan can be split up into two divisions: the Mechanical and the Software development.

Mechanical

The mechanical systems portion of the plan was followed very closely. The tasks were divided as they are shown in Appendix C and those tasks were completed at or around the time of the specified deadlines. Though the initial design was complete following the deadlines, the initial mechanical system was proven to not be a finished product and constant refinements, repairs and improvements were made throughout the software development and testing process. The structured order for the mechanical systems felt necessary as the build process was very linear and could not occur concurrently so having people assigned to tasks and ensuring deadlines were met was important.

Software

The software portion of the project plan followed a more blurred timeline and task distribution then the mechanical systems. Rather than following the exact steps of the modules in Appendix C, the functions were created as needed and people assigned themselves to the production of those functions. So for example, if some function needed to be created then one of the members would claim that function and finish it by the end of the day and get it ready for unit testing. The precise and strict software development plan felt unnecessary in this case and that is why we scrapped it fairly early on. The more fluid, iterative program development system was most effective in our case.

Conclusions

The robot successfully solves the problem initially proposed, of having a 3D printer capable of printing objects that a user can upload and choose from. The final design chosen adequately meets the constraints outlined earlier in the report, where the printing process is able to maintain the criteria stated as well. Through the combination of a sound mechanical design and efficient software program the robot was able to complete its desired tasks. Key features of the mechanical aspects include having a full three axis of movement, an effective baseplate, as well as a button pressing mechanism that never failed during both tests and demo. The key aspects of the software program include the file I/O to allow users to use their own files, a clean user interface and an accurate motion control system. Though not without difficulty, through a sound project plan and thorough testing the robot was successful in its tasks.

Recommendations

Mechanical

Heated Base

Often when printing larger objects the filament on the bottom of the object would cool down and cause the object to be detached from the base. Utilizing a heated base would cause the filament to stay attached to the base until printing was done as well as causing the layers to merge better [4].

Thinner Diameter Filaments

The pen used filament with a diameter of 1.75mm. This large filament diameter placed a physical limit on the precision. Utilizing a better extruder and thinner filament would result in better more precise objects.

Continuous Refill

Occasionally while printing the printer would run out of filament. When it ran out the operation had to be stopped and the user had to manually replace the filament which took a lot of time. To improve reliability and maximum build size a method to store more filament or hot swap filament should be added.

Gear Ratios

The robot used a very simple gear ratio with a 36-tooth gear attached directly to the rack gears for the X, Y actuators. This meant that every degree turn of the motor meant a certain distance was travelled. As can be seen in Appendix A the power values of the motors are very low. Creating a more complex gear ratio to gear the motor for torque rather than speed would allow for greater control over the motor and lessen the impact of motor encoder drift and inaccuracies.

Finer Tooth Rack Gear

The size of the teeth of the rack gears enforce a ceiling for the precision of the 3D printer. With lower distance in between the teeth comes a greater precision the printer can have. Manufacturing custom gears by 3D printing or laser cutting would have allowed the printer to be more precise.

Software

Control Algorithms

The inaccuracy of the printer was a significant hindrance to the quality of the 3D objects produced. Improving the control algorithms for the X, Y and Z axis actuators would have improved the consistency and quality of the print. The current method being used is a simple Bang-Bang algorithm with simple speed ranges but as described in Table 10 there are significant drawbacks.

G-Code Commands

In the preliminary report, it was suggested that the parsing of G-Code commands would have made an appropriate extension allowing for cross-compatibility with modern 3D printers [1]. The 3D printer becomes impractical if standard 3D models have to be converted into this 3D printer’s file format.

References

[1] C. Bains, A. Bhagwandeen, O. Brake and D. Sun, "Lego 3D Printer Preliminary Design Report," University of Waterloo, Waterloo, 2019.

[2] O. C. Stone, "Connect to the EV3 brick," Lynda.com, [Online]. Available: https://www.lynda.com/Mindstorms-tutorials/Connect-EV3-brick/455331/482351-4.html. [Accessed 2 December 2019].

[3] Powerplant Online, "Impresora 3D Anycubic Kossel Plus Delta," Powerplant Online, [Online]. Available: https://www.powerplanetonline.com/3d-printer-anycubic-kossel-plus-delta-en. [Accessed 2 December 2019].

[4] E. Sommer, "What's the Best PLA Print & Bed Temperature," ALL3DP, [Online]. Available: https://all3dp.com/2/the-best-pla-print-temperature-how-to-achieve-it/. [Accessed 1 December 2019].

Appendix A

Robot Code

```
/*
Constant Definitions
and
Function Prototypes
*/
const tMotor X_MOTOR = motorA;
const tMotor Y_MOTOR = motorC;
const tMotor Z_MOTOR = motorD;
const tMotor PEN_MOTOR = motorB;

const tSensors X_TOUCH = (tSensors)S3;
const tSensors Y_TOUCH = (tSensors)S1;
const tSensors Z_TOUCH = (tSensors)S2;

//This position struct was nice for testing
//It has no effect on the current program but provided a theoretical basis
//for where the printer head should be
typedef struct{
    int x;
    int y;
    int z;
} position_struct;
//This struct is used for file descriptors, is initialized in load_files.
//
typedef struct{
    string file_name;
    string title;
    string description;
}file;


int load_run_operations(const char * filename, position_struct &
position_module);
int get_instruction(char action_char);
int move_operation(position_struct & position_module,int x,int y);
void load_position_struct(position_struct & pos);
void reset_encoders();
void calibrate_axis();
void write_file();
void actuate_pen();
void move_z_axis(position_struct & position_module, int z);
void load_files(file *file_list);
int display_utility(file *file_list);
void shutdown();
task error_check();
void non_user_error(int reason);

void run_program(string file_name);


//Update this every time we add a new file
const int NUM_FILES = 3;




/*
Main Task
*/
task main()
{
    startTask(error_check)
    write_file();
    file file_list[NUM_FILES];
    load_files(file_list);
    int file_index = display_utility(file_list);
    wait1Msec(500);
    run_program(file_list[file_index].file_name);
    shutdown();
}


/*
Driver Functions
*/

/*
Driver function, extension of main.
Goes through the operations and sets up everything.
*/
void run_program(string file_name){
    position_struct position_module;
    load_position_struct(position_module);
    calibrate_axis();
    reset_encoders();
    load_run_operations(file_name,position_module);
}

/*
File I/O and Loading
*/

/*
File write driver, is simply a function we use at initialization if we want
to load
more files
*/
void write_file(){
  long fileHandle;
  const char * filename = "shutdown";
  fileHandle = fileOpenWrite(filename);
    fileWriteChar(fileHandle,'A');
  fileClose(fileHandle);

}


/*
Loads file and interprets that data based on the filename
Then calls the necessary function depending on the identifier
*/
int load_run_operations(const char * filename,
position_struct & position_module){
    bool first_extrude = true;
    long fileHandle = fileOpenRead(filename);
    writeDebugStreamLine("File: %s",filename);
    //wait1Msec(200);
    char char_value = 0;
    while (fileReadChar(fileHandle, &char_value))
    {
       writeDebugStreamLine("Type: %c",char_value);
       int instruction_type = get_instruction(char_value);
       if(instruction_type == -1){
             non_user_error(1);
```

```
            return -1;
        }
        if(instruction_type == 1){
            short x = 0;
            short y = 0;
            if(fileReadShort(fileHandle,&x) &&
               fileReadShort(fileHandle,&y)){
                  writeDebugStreamLine("x: %d y: %d",x,y);
                  move_operation(position_module,x,y);
            }
        }
        else if(instruction_type == 2){
            actuate_pen();
            if(first_extrude == true){
                  wait1Msec(2000);
                  first_extrude = false;
            }
            wait1Msec(550);
        }
        else if(instruction_type == 3){
            short z = 0;
            if(fileReadShort(fileHandle,&z)){
                  writeDebugStreamLine("z: %d",z);
                  move_z_axis(position_module,z);
            }
        }
        else if(instruction_type == 4){
            fileClose(fileHandle);
            return 0;
        }
        wait1Msec(200);
    }

    fileClose(fileHandle);
    return 0;
}

//Simple manager function, takes in char and determines what action that is
int get_instruction(char action_char){
    const char move_command = 'M';
    const char pen_command = 'P';
    const char z_command = 'Z';
    const char empty_command = 'A';
    if(action_char == move_command){
       return 1;
    }
    else if(action_char == pen_command){
       return 2;
    }
    else if(action_char == z_command){
       return 3;
    }
    else if(action_char == empty_command){
       return 4;
    }
    else{
       return -1;
    }
}

/*
Movement Operations
*/

/*
Takes in position module struct to update it.
Travels along the x,y axis to the desired position
*/
/*
The position struct was going to be used here when implementing PID but time
constraints made this impossible.
*/
int move_operation(position_struct & position_module,int x,int y){
    eraseDisplay();
    displayTextLine(2,"Printing file");

    time1[T1] = 0;
    const int POSITION_TOLERANCE = 3;
    const int standard_motor_power = 6;
    int x_distance = x - (-nMotorEncoder[X_MOTOR]);
    int y_distance = y - nMotorEncoder[Y_MOTOR];
    float x_multiplier = 1;
    float y_multiplier = 1;
    if(abs(x_distance) < 20 || abs(y_distance) < 20){
       x_multiplier = 1;
       y_multiplier = 1;
    }
    else if(abs(x_distance) > abs(y_distance)){
       y_multiplier = fabs((float)y_distance/(float)x_distance);
    }
    else{
       x_multiplier = fabs((float)x_distance/(float)y_distance);
    }
    while(abs(x_distance) > POSITION_TOLERANCE ||
          abs(y_distance) > POSITION_TOLERANCE){

       if(abs(x_distance) > POSITION_TOLERANCE){
             motor[X_MOTOR] = (float)-standard_motor_power *
                              (float)sgn(x_distance) *
                              (float)(x_multiplier);
       }
       else{
             motor[X_MOTOR] = 0;
       }
       if(abs(y_distance) > POSITION_TOLERANCE){
             motor[Y_MOTOR] = (float)standard_motor_power *
                              (float)sgn(y_distance) *
                              y_multiplier;
       }
       else{
             motor[Y_MOTOR] = 0;
       }

       writeDebugStreamLine("X:%d,Y:%d",nMotorEncoder[X_MOTOR],
                                        nMotorEncoder[Y_MOTOR]);
       writeDebugStreamLine("delta X:%d,delta Y:%d",x_distance,y_distance);
       writeDebugStreamLine("motor X:%d,motor Y:%d",
                             motor[X_MOTOR],
                             motor[Y_MOTOR]);

       wait1Msec(50);
       if(time1[T1] > 5000){
             non_user_error(3);
       }
       x_distance = x - (-nMotorEncoder[X_MOTOR]);
       y_distance = y - nMotorEncoder[Y_MOTOR];
    }
    motor[X_MOTOR] = 0;
    motor[Y_MOTOR] = 0;
```

```
    position_module.x = x;
    position_module.y = y;
    writeDebugStreamLine("Finished Move");
    return 0;
}
/*
Takes in the position struct to update it
Travels along z axis to desired position
*/
void move_z_axis(position_struct & position_module, int z){
    time1[T1] = 0;
    const int POSITION_TOLERANCE = 2;
    int speed = 5;
    int z_distance = z - nMotorEncoder[Z_MOTOR];
    int last_z = z_distance;
    bool steady = false;
    while(abs(z_distance) > POSITION_TOLERANCE || steady == false){
       z_distance = z - nMotorEncoder[Z_MOTOR];
       if(abs(last_z - z_distance) < 2){
             steady = true;
       }
       else{
             steady = false;
       }
       if(abs(z_distance) < 50){
             speed = 3;
       }
       else{
             speed = 4;
       }
       if(time1[T1] > 5500){
             non_user_error(3);
       }
       motor[Z_MOTOR] = speed * sgn(z_distance);
       last_z = z_distance;
       wait1Msec(75);
    }
    motor[Z_MOTOR] = 0;
    position_module.z = z;
}

/*
```

```
Sends motors to their extremes until their calibration sensor is clicked
*/
void calibrate_axis(){
    eraseDisplay();
    displayTextLine(2,"Calibrating...");

    time1[T1] = 0;
    bool x_pressed = false;
    bool y_pressed = false;
    bool z_pressed = false;
    while(!x_pressed || !y_pressed || !z_pressed){
       if(!x_pressed){
             motor[X_MOTOR] = 15;
       }
       else{
             motor[X_MOTOR] = 0;
       }
       if(!y_pressed){
             motor[Y_MOTOR] = -15;
       }
       else{
             motor[Y_MOTOR] = 0;
       }
       if(!z_pressed){
             motor[Z_MOTOR] = 25;
       }
       else{
             motor[Z_MOTOR] = 0;
       }

       if(SensorValue[X_TOUCH] == 1){
             x_pressed = true;
       }
       if(SensorValue[Y_TOUCH] == 1){
             y_pressed = true;
       }
       if(SensorValue[Z_TOUCH] == 1){
             z_pressed = true;
       }
       if(time1[T1] > 4500){
             non_user_error(2);
       }
    }
    motor[X_MOTOR] = 0;
    motor[Y_MOTOR] = 0;
    motor[Z_MOTOR] = 0;
}

/*
Simple function to actuate pen and click the button
*/
void actuate_pen()
{
      const int POW = -10;
      const int SWITCH = -90;
      nMotorEncoder[PEN_MOTOR] = 0;

      motor[PEN_MOTOR] = POW;
      while(nMotorEncoder[PEN_MOTOR] < SWITCH)
      {}

      wait1Msec(1200);

      motor[PEN_MOTOR] = -POW;
      while(nMotorEncoder[PEN_MOTOR] < 0)
      {}

      motor[PEN_MOTOR] = 0;
}


/*
Simple Misc Functions
*/

//Initializes position_struct to default values
void load_position_struct(position_struct & pos){
    pos.x = 0;
    pos.y = 0;
    pos.z = 0;
}
//Resets all motor encoders
void reset_encoders(){
    nMotorEncoder[X_MOTOR] = 0;
    nMotorEncoder[Y_MOTOR] = 0;
    nMotorEncoder[Z_MOTOR] = 0;
    nMotorEncoder[PEN_MOTOR] = 0;
}
//Initializes all the predefined files, these file names and descriptions
//are inserted manually
void load_files(file *file_list){
    file_list[0].file_name = "move_test";
    file_list[0].title = "Rect. Prism";
    file_list[0].description = "This is a rectangular prism";
    file_list[1].file_name = "triangle";
    file_list[1].title = "Tri. Prism";
    file_list[1].description = "Vertical Tri. Prism";
    file_list[2].file_name="shutdown";
    file_list[2].title="Shutdown Test";
    file_list[2].description="Empty File";

}
//Is the menu system for right after initialization
//User selects file here
int display_utility(file *file_list){
    int index = 0;
    eraseDisplay();
    displayBigTextLine(2,file_list[index].title);
    displayTextLine(4,file_list[index].description);

    while(!getButtonPress(buttonEnter)){
       if(getButtonPress(buttonLeft) || getButtonPress(buttonRight)){
             if(getButtonPress(buttonLeft)){
                   index--;
             }
             else if(getButtonPress(buttonRight)){
                   index++;
             }
             if(index < 0){
                   index = 0;
             }
             if(index >= NUM_FILES){
                   index = NUM_FILES-1;
             }
             eraseDisplay();
             displayBigTextLine(2,file_list[index].title);
             displayTextLine(4,file_list[index].description);
             wait1Msec(500);
       }
    }
    return index;
}
//Standard shutdown operation
void shutdown(){
    calibrate_axis();
    eraseDisplay();
    displayCenteredTextLine(2,"Thank you for Printing!");

    displayCenteredTextLine(5,"Press any Button to Exit");

    displayCenteredTextLine(7,"Owen, Chealsie, David, Addesh");
    while(!getButtonPress(buttonAny));
}

//Constantly running error_check task
//Kind of run concurrently but of course the EV3 is single threaded
//So it's not perfect concurrency, but it is good enough
task error_check(){
    while(!getButtonPress(buttonUp)){
    }
    while(getButtonPress(buttonUp));
       stopTask(main);
       motor[X_MOTOR] = motor[Y_MOTOR] = motor[Z_MOTOR]
                      = motor[PEN_MOTOR] = 0;
       eraseDisplay();
       displayBigTextLine(2,"User Exit");
       displayBigTextLine(4,"Closing in 5s");
       wait1Msec(5000);
}
//Non_user_error handler, shutdown the operation and displays to the user the
error
void non_user_error(int reason){
       motor[X_MOTOR] = motor[Y_MOTOR]
                      = motor[Z_MOTOR] = motor[PEN_MOTOR] = 0;
       eraseDisplay();
       displayBigTextLine(2,"Program Exit");
       displayBigTextLine(4,"Closing in 5s");
       if(reason == 1){
             displayCenteredTextLine(6,"File Read Error");
       }
       else if(reason == 2){
             displayCenteredTextLine(6,"Calibration Error");
```

```
        }
        else if(reason == 3){
              displayCenteredTextLine(6,"Move Timeout Error");
        }

        wait1Msec(5000);
        stopTask(error_check);
        stopTask(main);


}
```

Appendix B

Original Flow Charts

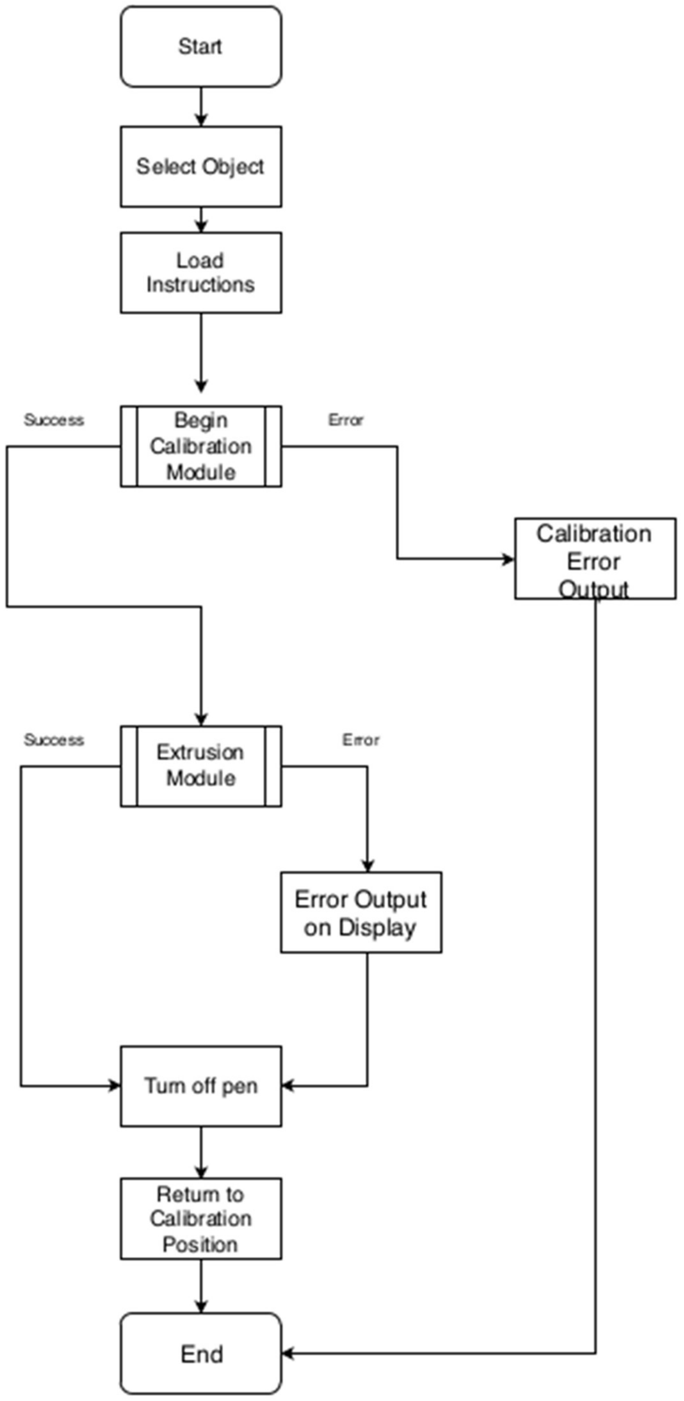


Figure 19. Original high-level flow chart

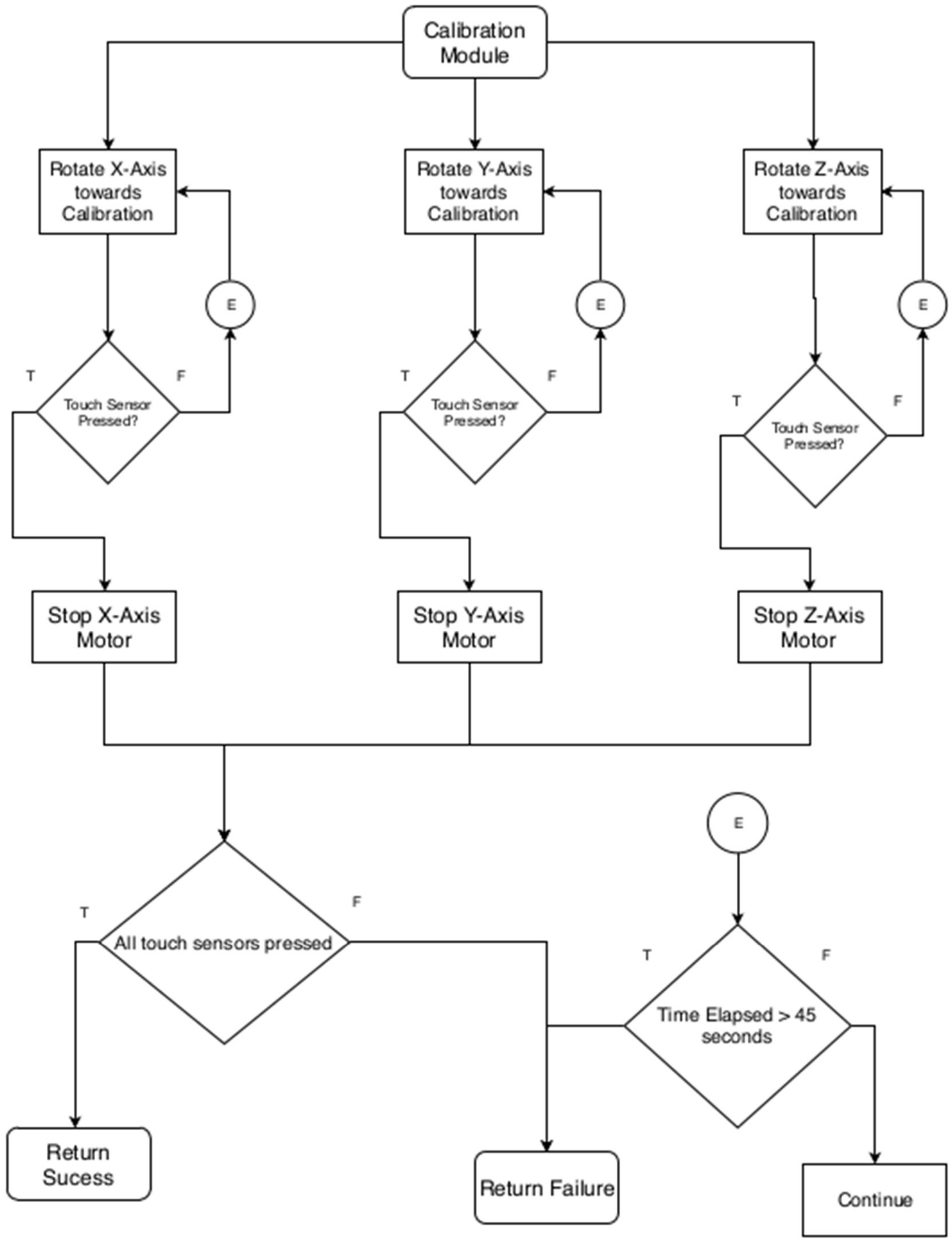


Figure 20. Original calibration module flow chart

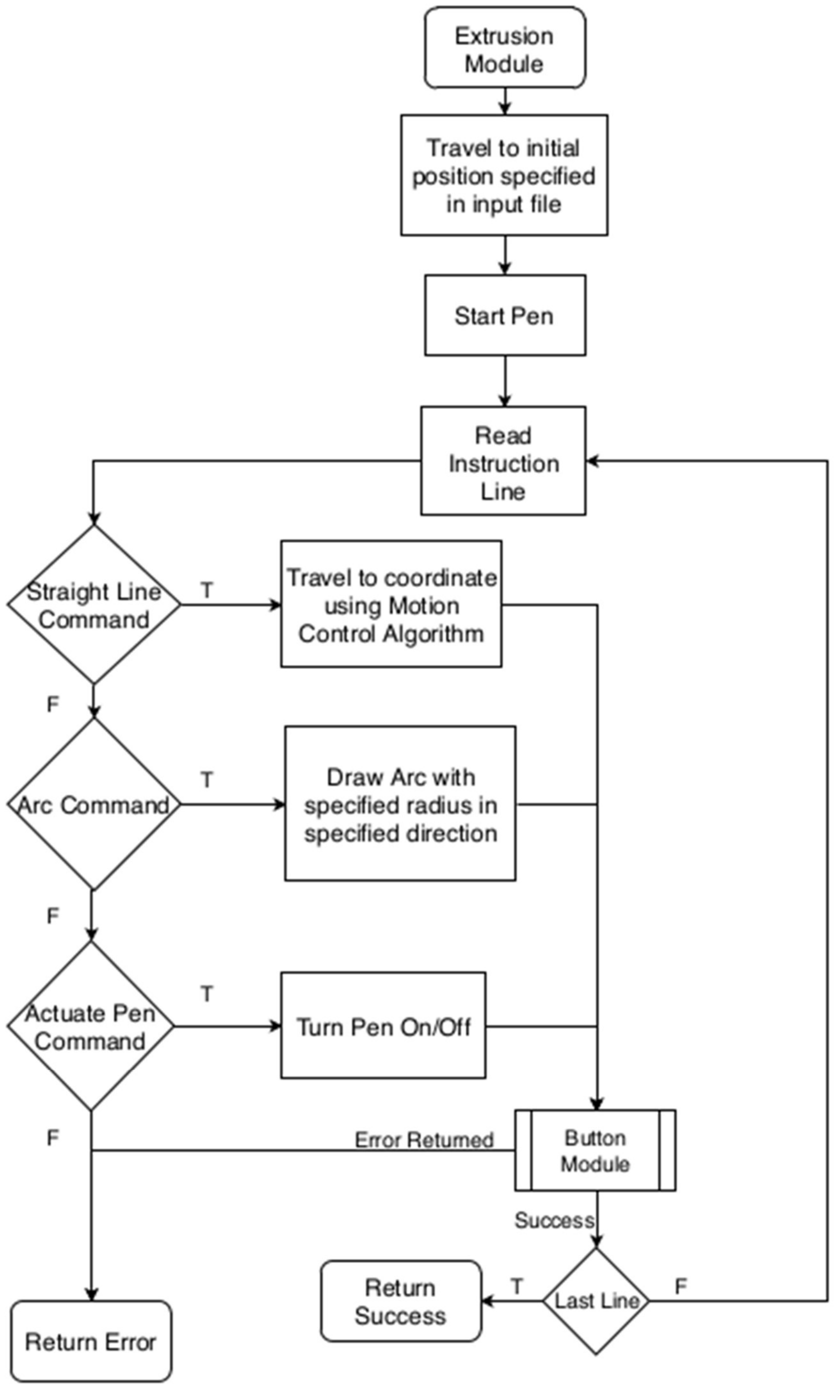


Figure 21. Original extrusion module flow chart

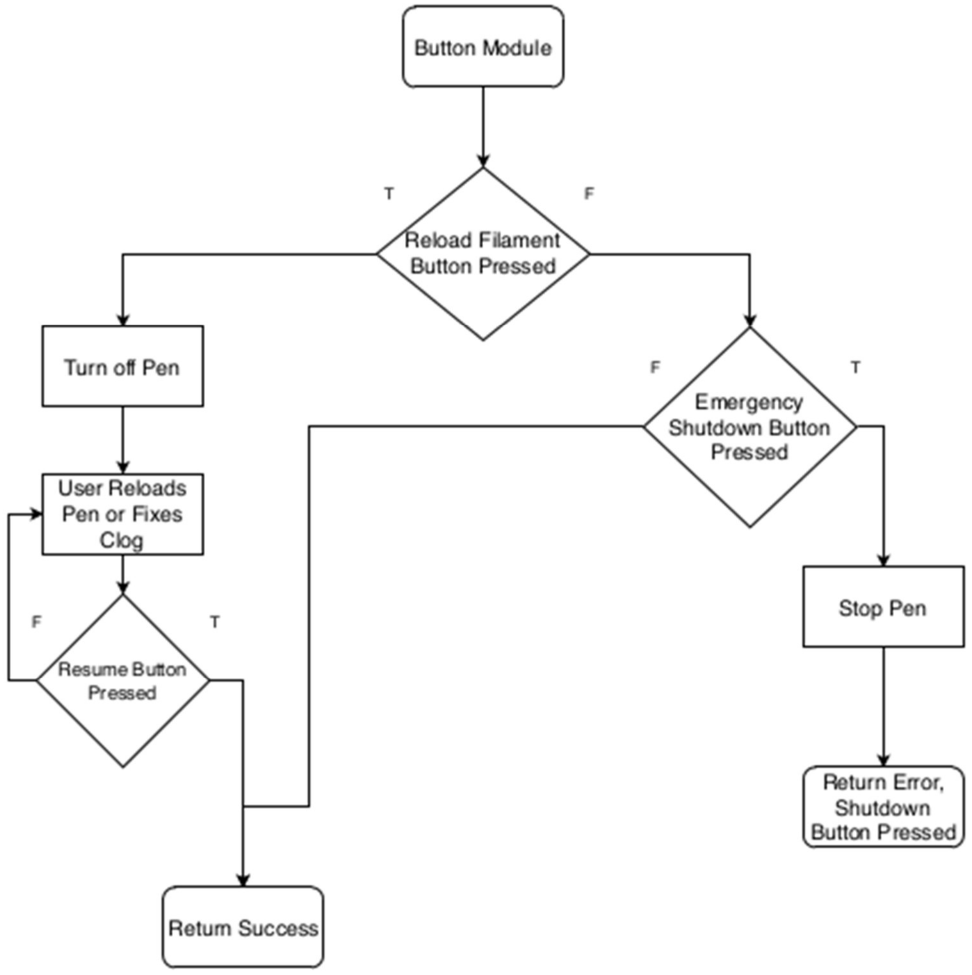

Button Module
T
F
Reload Filament Button Pressed
Turn off Pen
F
T
Emergency Shutdown Button Pressed
User Reloads Pen or Fixes Clog
Stop Pen
F
T
Resume Button Pressed
Return Error, Shutdown Button Pressed
Return Success

Figure 22. Original button module flow chart

Finished Project Flow Charts

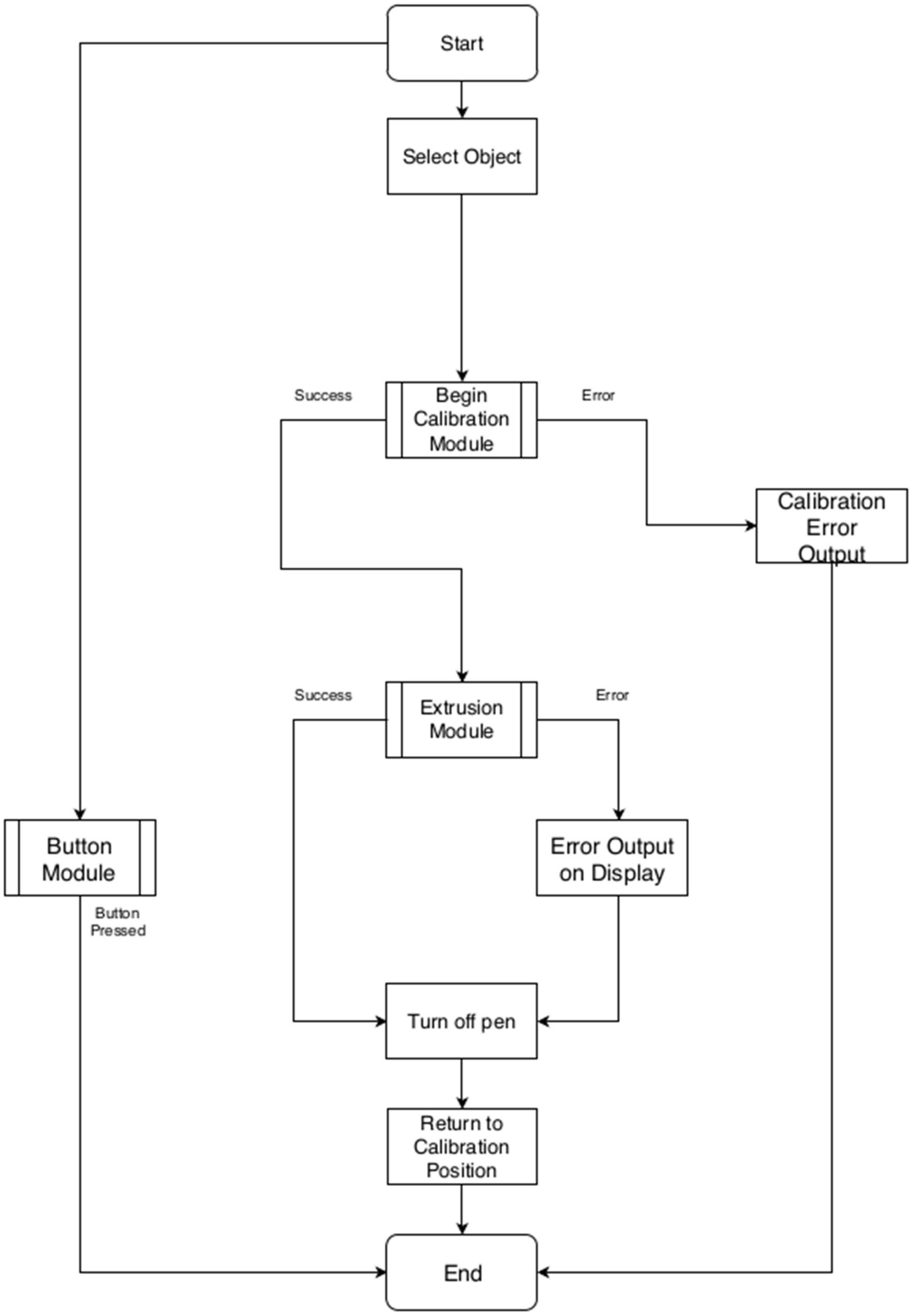


Figure 23. Finished project high level flow chart

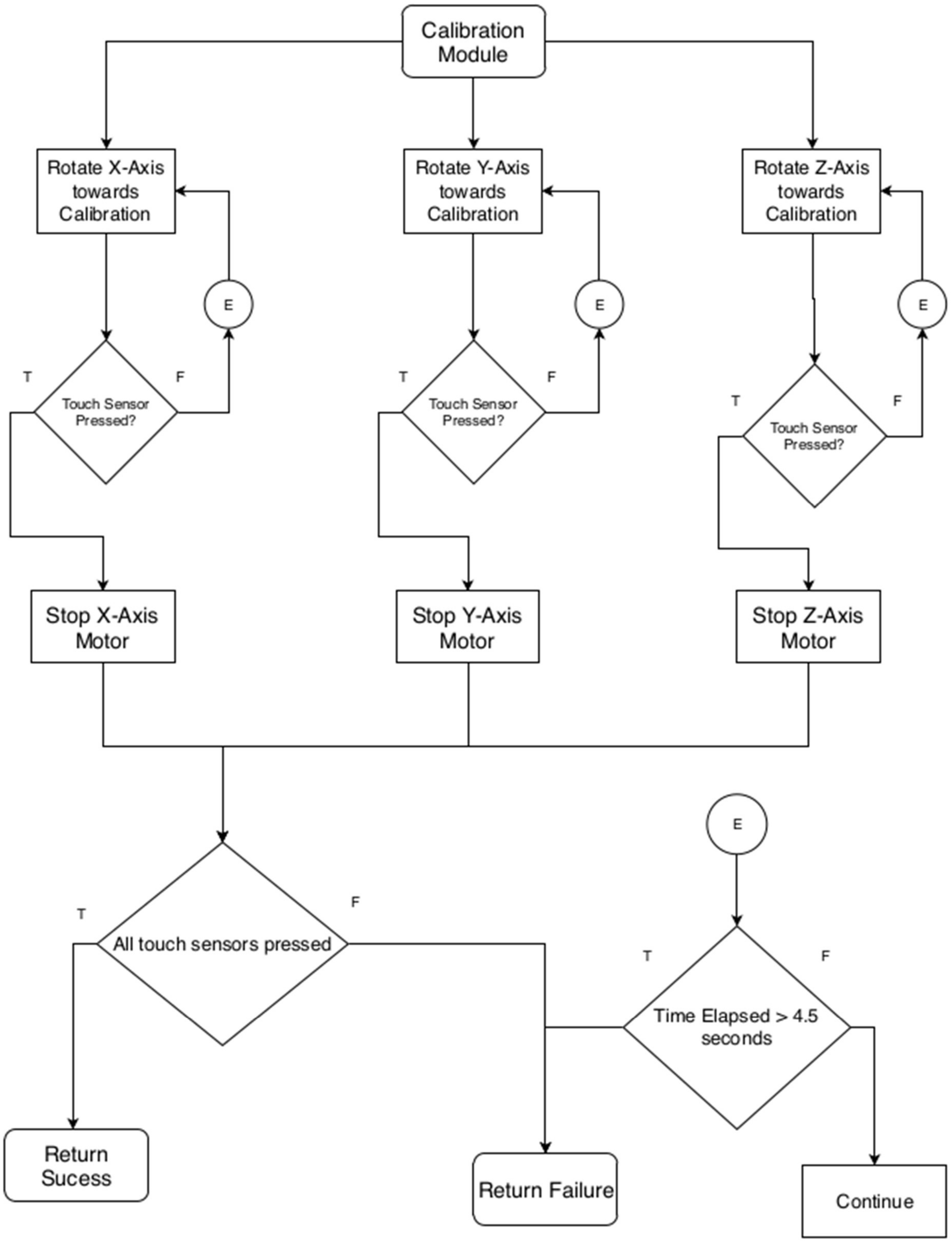


Figure 24. Finished project calibration module flow chart

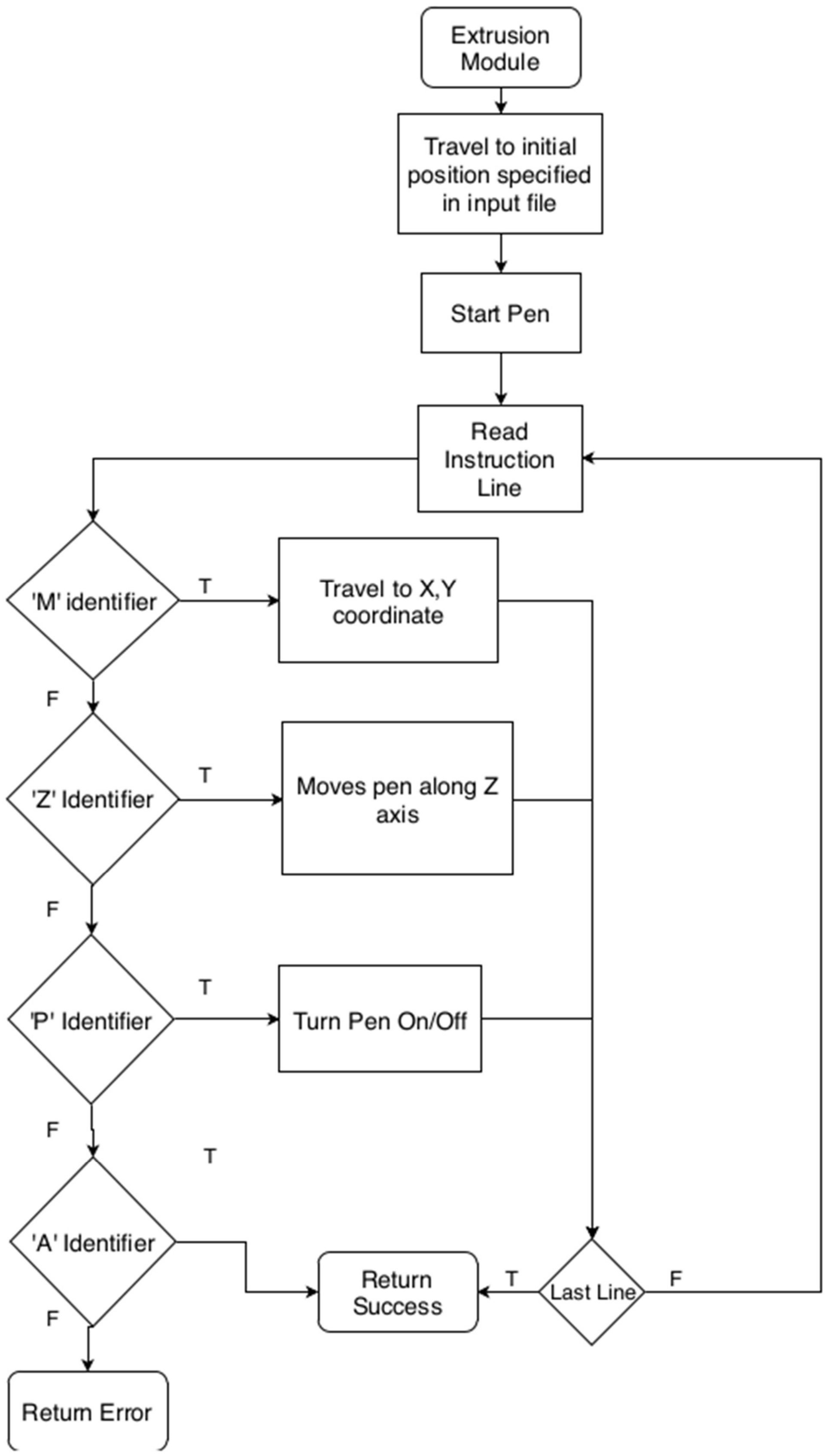


Figure 25. Finished project extrusion module flow chart

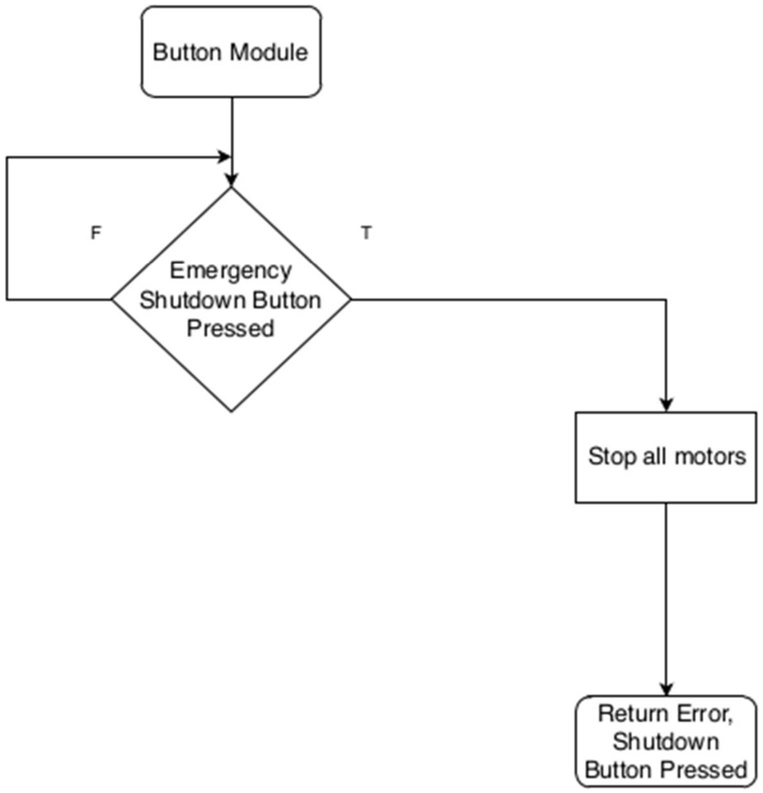
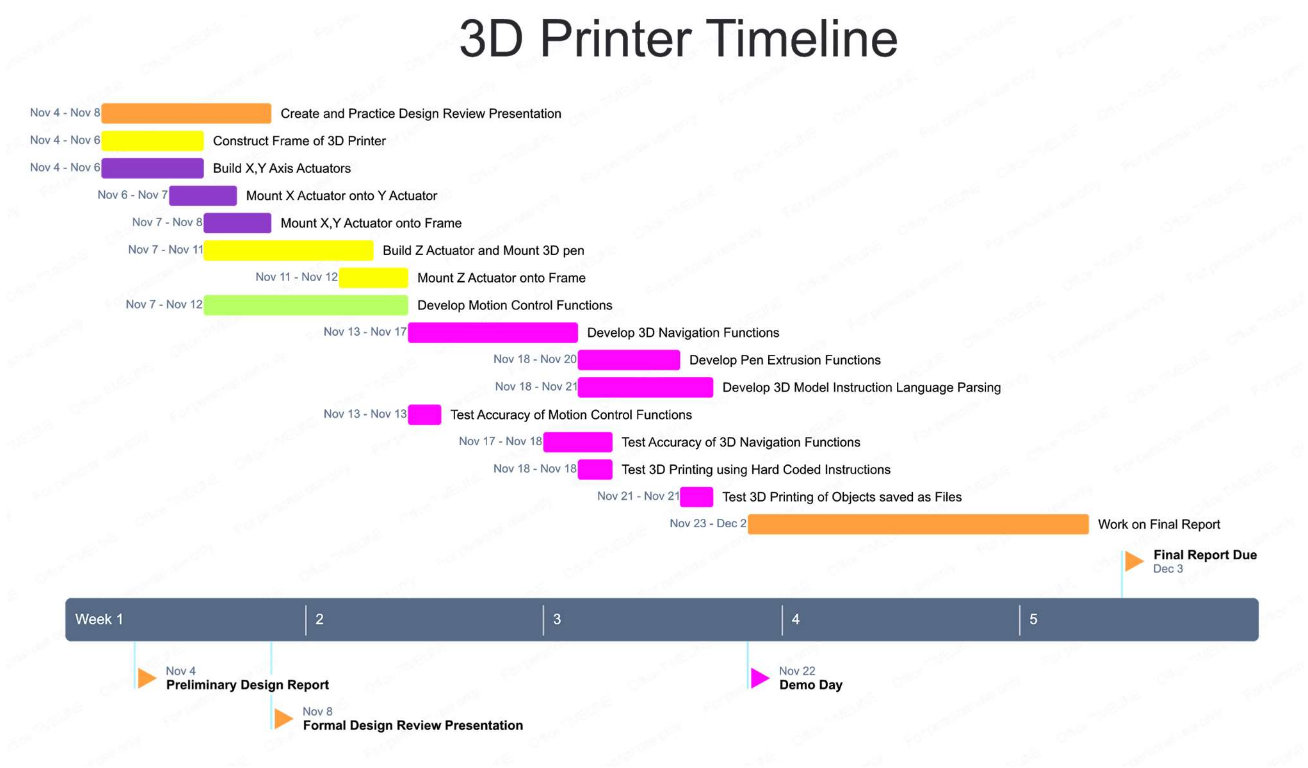

Button Module
F
T
Emergency Shutdown Button Pressed
Stop all motors
Return Error, Shutdown Button Pressed

Figure 26. Finished project button module flow chart

Appendix C

Project Plan



🟨	Chealsie
🟩	Addesh
🟪	Owen and David
🟧	Owen, David and Chealsie
🟪	Owen, David, Chealsie and Addesh

Figure 27. Original project plan GANTT Chart

Appendix D

File Driver Code

Rectangular Prism

```
//initialize file writer
long fileHandle;
const char * filename = "move_test";
fileHandle = fileOpenWrite(filename);
//Lower pen to standard position
fileWriteChar(fileHandle,'Z');
fileWriteShort(fileHandle,(short)-153);
//Move onto the plate, to the initial x,y positon
fileWriteChar(fileHandle,'M');
fileWriteShort(fileHandle,(short)75);
fileWriteShort(fileHandle,(short)0);
//Turn on pen
fileWriteChar(fileHandle,'P');
//draw first layer
fileWriteChar(fileHandle,'M');
fileWriteShort(fileHandle,(short)150);
fileWriteShort(fileHandle,(short)0);
fileWriteChar(fileHandle,'M');
fileWriteShort(fileHandle,(short)150);
fileWriteShort(fileHandle,(short)75);
fileWriteChar(fileHandle,'M');
fileWriteShort(fileHandle,(short)75);
fileWriteShort(fileHandle,(short)75);
fileWriteChar(fileHandle,'M');
fileWriteShort(fileHandle,(short)75);
fileWriteShort(fileHandle,(short)0);
//Increase pen for next layer
fileWriteChar(fileHandle,'Z');
fileWriteShort(fileHandle,(short)-145);
for(int i = 0; i < 10;i++){
      if(i % 3 == 0){
            //Every 3 loops raise pen
            fileWriteChar(fileHandle,'Z');
            fileWriteShort(fileHandle,(short)-135+5*(i/3));
      }
      //print each layer of rectangular prism
      fileWriteChar(fileHandle,'M');
      fileWriteShort(fileHandle,(short)150);
      fileWriteShort(fileHandle,(short)0);
      fileWriteChar(fileHandle,'M');
      fileWriteShort(fileHandle,(short)150);
      fileWriteShort(fileHandle,(short)75);

      fileWriteChar(fileHandle,'M');
      fileWriteShort(fileHandle,(short)75);
      fileWriteShort(fileHandle,(short)75);
      fileWriteChar(fileHandle,'M');
      fileWriteShort(fileHandle,(short)75);
      fileWriteShort(fileHandle,(short)0);
}
//turn off pen
fileWriteChar(fileHandle,'P');
//close file
fileClose(fileHandle);
```

Triangular Prism

```
//initialize file writer
long fileHandle;
const char * filename = "triangle";
fileHandle = fileOpenWrite(filename);
//lower pen to initial position
fileWriteChar(fileHandle,'Z');
fileWriteShort(fileHandle,(short)-158);
//move x,y base to initial position
fileWriteChar(fileHandle,'M');
fileWriteShort(fileHandle,(short)75);
fileWriteShort(fileHandle,(short)0);
//Turn on penv
fileWriteChar(fileHandle,'P');

fileWriteChar(fileHandle,'M');
fileWriteShort(fileHandle,(short)150);
fileWriteShort(fileHandle,(short)0);
for(int i = 1;i <= 10;i++){
      //Draws triangle
      fileWriteChar(fileHandle,'M');
      fileWriteShort(fileHandle,(short)112);
      fileWriteShort(fileHandle,(short)75);
      fileWriteChar(fileHandle,'M');
      fileWriteShort(fileHandle,(short)75);
      fileWriteShort(fileHandle,(short)0);
      fileWriteChar(fileHandle,'M');
      fileWriteShort(fileHandle,(short)150);
      fileWriteShort(fileHandle,(short)0);
      if(i % 2 == 0){
            //Raise pen every second layer
            fileWriteChar(fileHandle,'Z');
            fileWriteShort(fileHandle,(short)-158+i*2*2);
      }

}
//Stop pen
fileWriteChar(fileHandle,'P');
//Close file
fileClose(fileHandle);
```

Shutdown Test

```
//Initialize file writer
long fileHandle;
const char * filename = "shutdown";
fileHandle = fileOpenWrite(filename);
//Only insert the exit character, we insert A
//Just to make sure the file is actually written to.
fileWriteChar(fileHandle,'A');
//Close file
fileClose(fileHandle);
```