Central Connecticut State University
Department of Manufacturing and Construction Management
Robotics and Mechatronics Engineering Program

ROBO 497 – Capstone Senior Project

PREPARED IN PARTIAL FULFILLMENT OF THE
REQUIREMENT FOR THE DEGREE OF
BACHELOR OF SCIENCE
IN
ROBOTICS AND MECHATRONICS ENGINEERING TECHNOLOGY

3D-Printing with Fanuc Robot Arm Final Report

Pedro Urbina

Advisor(s): Professor Ravindra Thamma, Ph.D.

August 18th, 2019

Abstract

In this project, a Fanuc LR Mate 200iD 7L robot arm was equipped and programmed to function as a 3D printer. It was developed by integrating the robot arm system with a generic 3D printer extrusion assembly, which acted as the robot's end effector, and an Arduino Uno microcontroller. The extrusion assembly, or print head, included a stepper motor, hot end, nozzle, cooling fan, and feeder. The hot end and stepper motor, as well as a heated build plate, were controlled by the microcontroller. Mounting hardware was designed in Siemens NX12 and 3D-printed with the lab's 3D printer.

Producing a part starts by obtaining a 3D model in STL format. The file is passed through multiple file conversion programs until it is in Fanuc's TP format. It is then downloaded to the robot controller so it can be executed. Due to the robot controller's low available memory, only very small parts could be printed. Given more memory, a setup like this could potentially produce much bigger parts than a common gantry-style 3D printer, thanks to the robot arm's large work envelope. The stepper motor system used in this project was unable to simultaneously operate at low speed and high resolution. As a result, the print head either extruded material too quickly or slowly but inconsistently. In the end, the system was able to produce small, simple parts that resembled the desired geometry. With some minor equipment upgrades and improved code, the system should be able to produce acceptable parts.

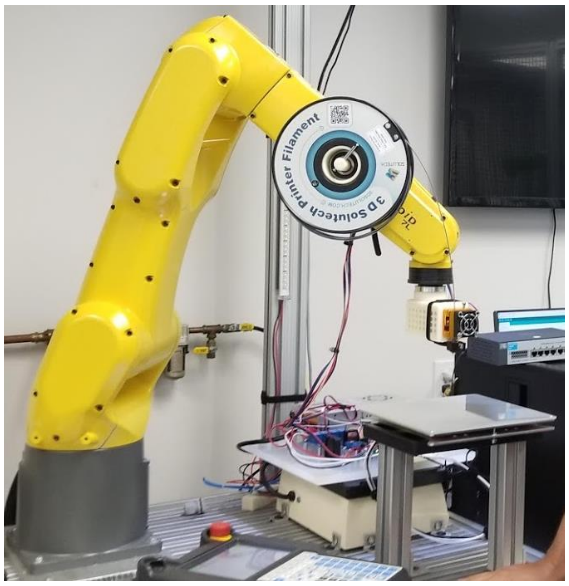
Figure 1
Full View of Prototype

Table of Contents

Contents

Tables and Figures

Acknowledgment

Thanks to my advisor, Ravindra Thamma, Ph.D., for going above and beyond to provide access to the supplies and resources I needed and for coming in at odd hours to get me as much time in the lab as I needed.

I. Problem Definition

3D printing, a type of additive manufacturing, has sharply increased in popularity over the past several years as a method for rapid prototyping and final production of custom parts, tooling, and fixtures. By far, the most common type of 3D printer used today features a print head mounted on an X-Y gantry which deposits material on a build plate that moves in Z to build the part layer by layer. One drawback of this system is that the size of the part is limited by the size of the 3D printer's enclosure. If the part to be built is larger than what the 3D printer can accommodate, the part will have to be built in pieces then assembled, or the 3D printer will need to be replaced with a bigger one. Although the cost to buy a 3D printer has reduced significantly in recent years, it remains a significant investment; and if large parts need to be built, a significant investment in floor space will also be required.

Robot arms are a commonplace in manufacturing plants and are valued for their versatility, accuracy, and repeatability in performing repetitive tasks. Additionally, and quite relevant to the application in question, they have a large work envelope and small footprint by design. Is it possible and practical to equip and program a robot arm to function as a 3D printer for producing large parts? This project seeks to answer that question.

II. Engineering Process

The goal is to design and integrate tooling and electronics to a robot arm to 3D-print large parts. The system should be able to produce parts from an STL file from any typical CAD program, just like a typical 3D printer.

The general idea is to mount a material extruder to the end of the robot arm and to program the robot to move along the entire volume of the desired part while material is extruded. In an ideal process, the systems controlling the extruder and robot's movements will communicate so that the right amount of material is extruded to match the speed of the robot and, therefore, produce a detailed part.

2.1 Robot Arm Selection

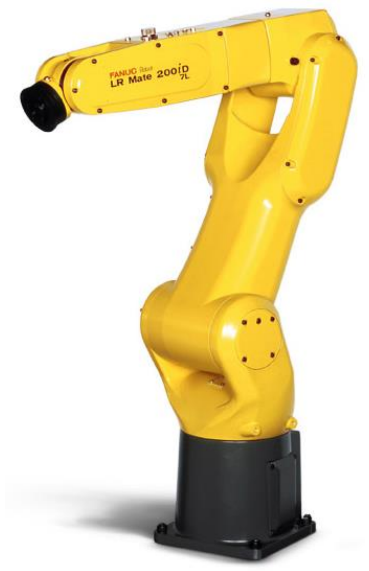

Figure 2
Fanuc LR Mate 200iD 7L

Certainly, the most important component of the system is also the basis of the project, the robot arm. Fortunately, several different models of Fanuc 6-axis robot arms are available in the CCSU engineering lab. All the robots operate with the same software, so the only configuration to choose was the size of the robot, which defines the range and load that it can handle. The choice was narrowed down to two models, the LR Mate 200iD 4S and the LR Mate 200iD 7L. The 4S model has a reach of 550mm and load capacity of 4kg, while the 7L has a reach of 911mm and load capacity of 7kg. The 7L was chosen simply for its extended reach and load capacity. This would allow more workspace to experiment with the design of the other components and reduce the risk of compromising accuracy when mounting a potentially heavy end effector.

2.2 Extruder Assembly Selection

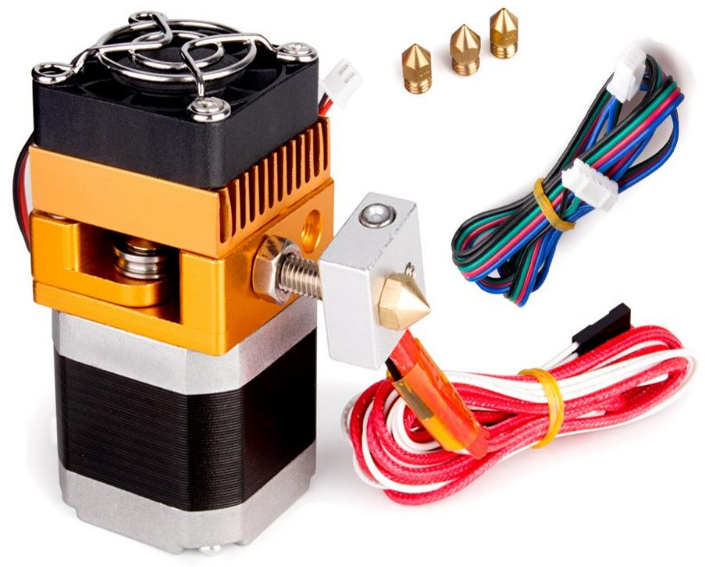

Figure 3
MK8 Extruder Hot End Kit

The material extruder, or print head, is the second most important component of the system. Material extrusion will require four main components: spool, motor, feeder, hot end, and nozzle. The spool carries the material, which comes in filament form, and rotates freely as filament is pulled into the extruder. The motor pulls the filament with enough force to push it through the extruder, rotate the spool, and at a consistent speed to provide an appropriate flow rate of material. The feeder pins the filament against the motor shaft and guides it into the hot end and nozzle. As the filament passes through the hot end, it is quickly heated and melted. Finally, the melted material is pushed through the nozzle, concentrating it into a thin layer so it can produce fine details in the part to be made.

The entire assembly, except for the spool, will need to be mounted at the end of the robot arm, in such a way that the nozzle can easily reach the build surface. One option is to design and build a custom extruder assembly to best fit the robot’s mounting plate; however, several compact extruder assemblies are available on the market at low cost. The MK8 Extruder Hot End Kit (shown below) includes a NEMA 17 stepper motor, feeder, hot end, multiple nozzles with varying diameters, thermistor to monitor hot end temperature, and a cooling fan. It is compatible with PLA, one of the most commonly used 3D printing materials.

2.3 Build Surface Selection

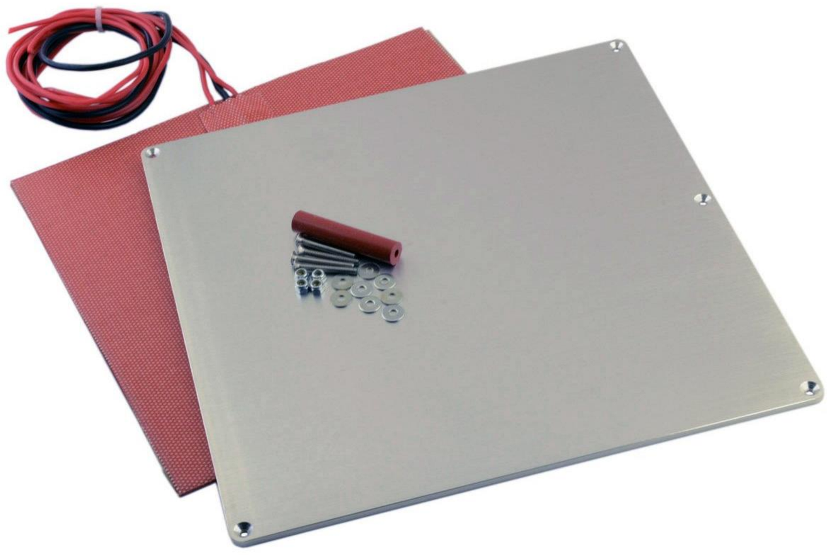

Figure 4
12V 200W Heated Aluminum Build Plate (220mm X 220mm)

Another very important component necessary for 3D printing is the build surface. At a minimum, the build surface should be flat, smooth, and easy for the build material to adhere to. A more effective build plate is also heated. This helps the first few layers, the most important in any print, to lay as flat as possible on the build surface. For the application desired in this project, the build surface should be as large as possible; but for the purpose of prototyping, a small (200mm x 200mm), heated aluminum build plate was selected. It includes a thermistor to monitor and control the temperature. A glass plate was placed on top of the aluminum plate, as it would be easier to remove the produced part.

After running a few test prints on the glass surface, it was discovered that the PLA was not adhering consistently. It was replaced with a “heat bed platform sticker sheet”. This is a textured plastic sheet that self-adheres to the aluminum plate. It resulted in much better adhesion of the PLA onto the build surface.

2.4 Logic Controller Selection

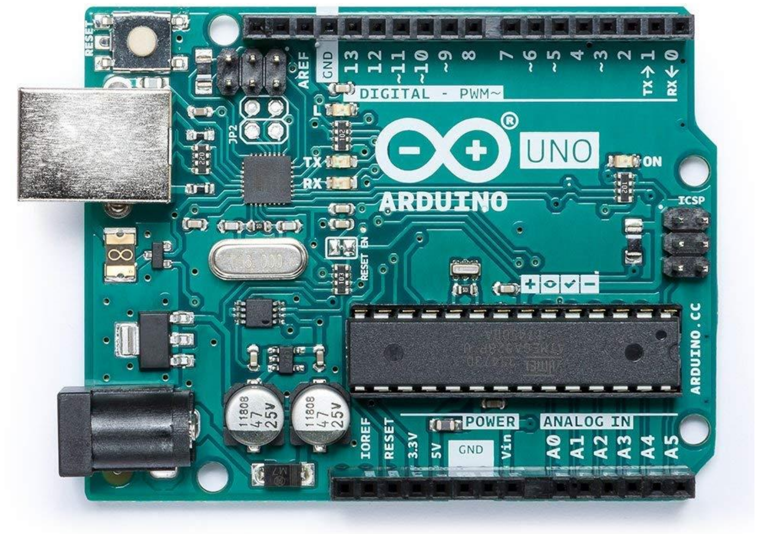

Figure 5
Arduino Uno Microcontroller

A microcontroller is necessary to give instructions to the extruder’s stepper motor, hot end, and the heating element of the build plate. The robot arm is equipped with its own controller, but it runs on Fanuc’s proprietary Teach Pendant programming language. It is optimized for programming the robot’s motions and inputs and outputs for accessories; however, it doesn’t contain libraries and commands to control something like a stepper motor easily. Any typical microcontroller will be enough to control these accessories. The Arduino Uno microcontroller was selected for its simple user interface and vast availability of libraries and open-source code for the functions needed in this project.

The Arduino Uno board features an ATmega328P processor with a clock speed of 16 MHz, 32 KB of flash memory, 15 digital input/output pins, and 6 analog input pins. It can execute up to 20 million instructions per second.

2.5 Power Supply Selection

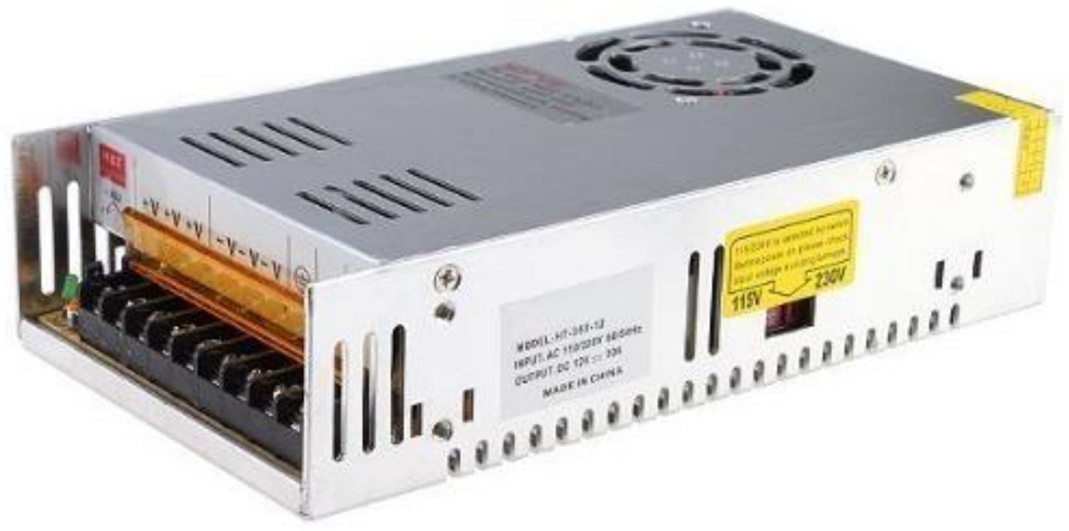

Figure 6
12V 30A DC Universal Regulated Switching Power Supply

The heating elements and the extruder's stepper motor require much more power than the 20 mA that can be supplied by the microcontroller. All the components to be powered require 12V DC. To select an appropriate supply, the current requirement at maximum load of each component was simply summed. Referring to each component's respective manufacturer's specifications, the current requirements are as follows:

Component	**Maximum Current Requirement (Amps)**
Extruder Stepper Motor	3.0
Extruder Hot End	3.3
Build Plate	16.7
Total	**23.0**

Figure 7
Current Requirements of Major Components

The power supply purchased provides up to 30 amps of DC at 12V and has multiple screw terminals to easily connect multiple components. It did not include a power cable or a standardized connector for a power cable, so an AC power cable designed for a desktop computer was stripped, fitted with ferrules, and connected to the screw terminals on the power supply.

2.6 Stepper Motor Controller

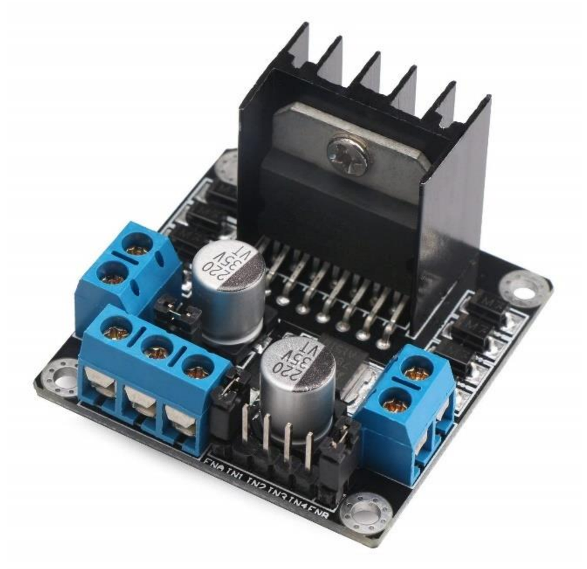

Figure 8
L298N Stepper Drive Board

Stepper motors function by rotating in discrete angular increments called steps. To achieve each step, the coils in the motor are energized at different times and polarities. Therefore, the stepper motor will not work by simply connecting it directly to DC source. A stepper driver, like the one shown above, connects to the DC source and “chops” the current among the four outputs based on the logic sequence received from the microcontroller.

2.7 Build Material Selection

Figure 9
PLA Filament Spool

The print head being used for this project can reach temperatures high enough to be compatible with any type of 3D printing material. Polylactic Acid (PLA) was selected for this project for its ease of use, biodegradability, and low cost.

2.8 Print Head Mounting Bracket Design

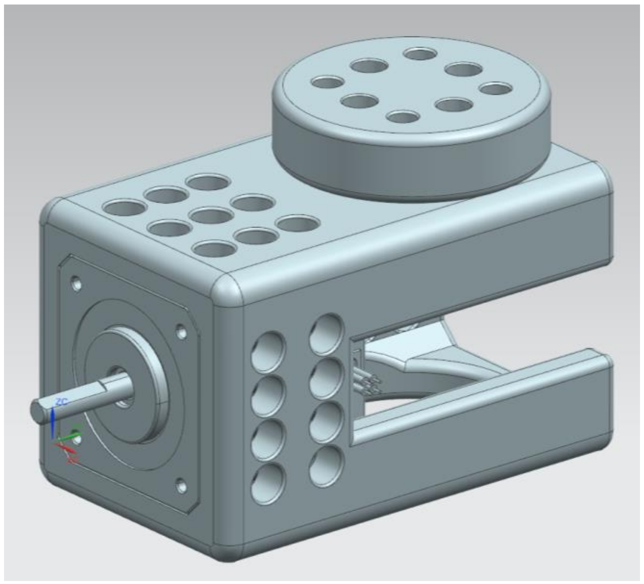

Figure 10
Print Head Mounting Bracket

Because the printhead being used was designed for a specific 3D printer, it naturally did not come with any mounting hardware that could be used to secure it to the robot; nor was there such a product available off-the-shelf. It was decided to design and build mounting hardware in the engineering lab.

The first consideration was the material and manufacturing process to use. The most convenient option would be to 3D print the parts needed here in the lab; but there was concern that the extruder assembly would generate too much heat during use and melt the plastic. After testing the stepper motor at length, however, it was discovered that it only became warm to the touch. Acrylonitrile Butadiene Styrene (ABS), the material used in the lab's 3D printer, starts becoming malleable above 105° C – much higher than the temperatures the stepper motor is expected to reach. It was decided to 3D print the mounting bracket.

The print head should be mounted so that the nozzle is oriented in the same direction as the end of the robot to reduce complexity when programming the robot's movements. Also, the filament entry point and connector ports on the extruder assembly should be unobstructed by the robot's links. The print head was reassembled so that all the electrical connections could be run on one side of the robot and so the filament could be fed in from above.

The bracket was designed with a slot for the stepper motor to be inserted. The geometry of the stepper motor prevents it from being pushed too far into the slot. Then, the rest of the extruder is assembled onto the end of the motor. The geometry of the extruder hardware prevents the whole assembly from sliding back through the slot. The slot itself was designed with a tight tolerance to prevent the motor from moving around. Counter-bored holes are included to mount the bracket to the robot's faceplate. Holes were added wherever possible for heat dissipation.

The resulting fixture holds the print head rigidly in the desired orientation and provides proper clearance for connections and filament. The only issue is the stepper motor slot is too tight, meaning that the motor had to be forced in. Disassembly may require an amount of force that could damage the motor or the bracket.

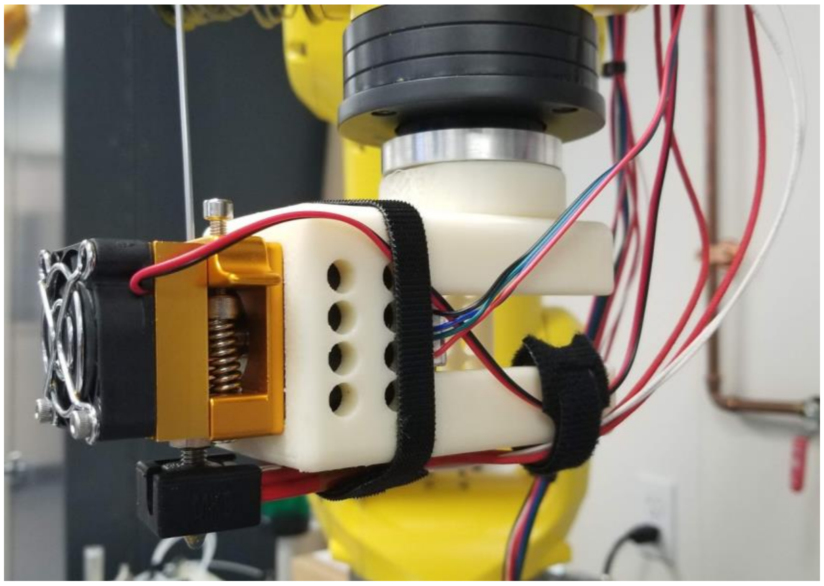
Figure 11
Mounted Print Head

2.9 Filament Spool Holder Design

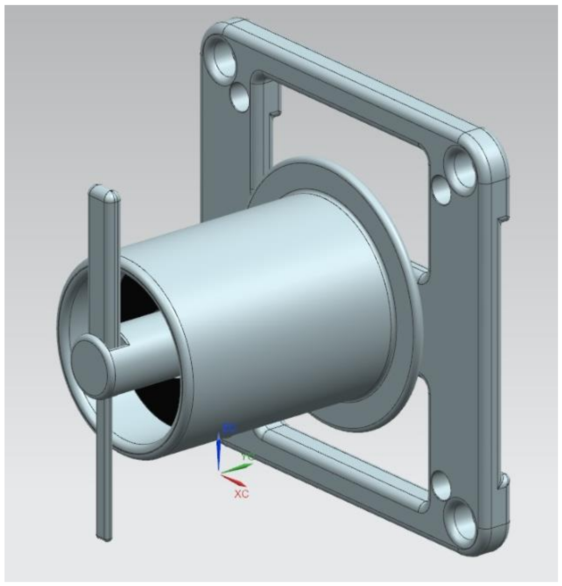
Figure 12
Filament Spool Holder

The filament spool needs to be mounted in a way that the filament can easily be fed into the extruder. It should also give as little resistance as possible, so the stepper motor does not struggle to pull the filament in and to not disrupt the flow of material to the print.

The desire was to mount the spool as close to the extruder as possible. The link between joints four and five on the robot has enough space for the spool and is close to the end of the robot, so that was the best place to mount the spool. A mounting bracket for this application was not available off-the-shelf, so it was designed. To simplify the process and save time, this bracket would be 3D printed as well.

The bracket was designed to conform to the shape of the robot arm and with holes to tie it down with zip ties. The axle was designed to fit with a bearing that was available in the lab. The spool is simply placed onto the axle and a pin is placed at the end of the axle to prevent the spool from falling off during operation.

The result allows the spool with spin freely and the bracket is secured firmly onto the robot. The pin, however, was too thin and quickly broke when handled. Two metal screws are inserted into the pin-slot instead.

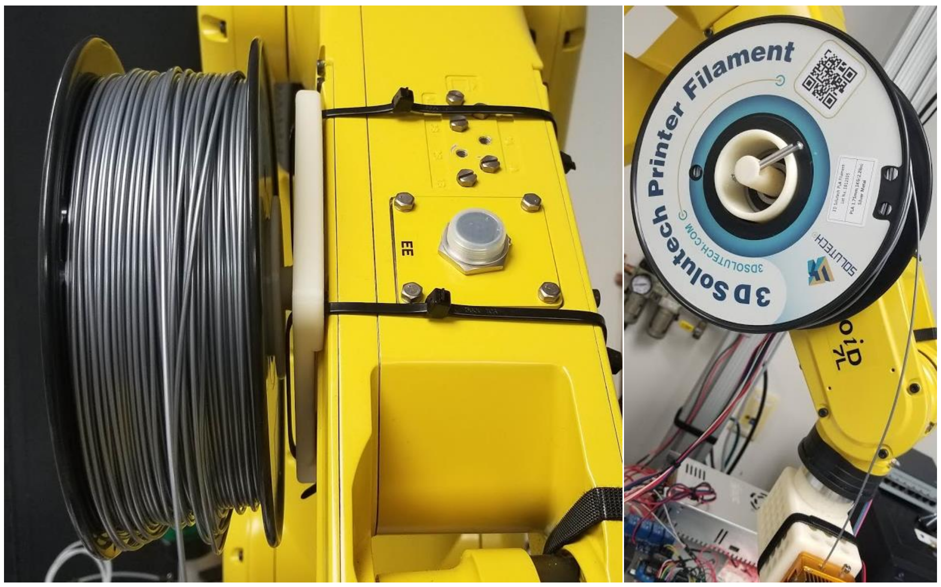

Figure 13
Mounted Filament Spool

2.10 Programming Robot Movements

One of the criteria for this project is to be able to produce parts from CAD models. The robot will need to be programmed to move through the entire geometry of the desired part precisely and quickly. The robot being used does not come with a feature in its software to command the robot to move through the entire volume of a part. Furthermore, to produce even a small part, the robot will need to execute hundreds or thousands of motions, so manually programming is completely impractical. A third-party solution is needed.

In a typical 3D printing process, the CAD model is saved into an STL file, then converted to a G-Code file by the 3D printer's software. G-Code essentially "slices" the 3D model into many thin 2D layers and gives the 3D printer the motion instructions for each layer. This type of programming would be a good place to start for this project; however, the robot controller cannot read G-Code directly. Fortunately, a Java program to convert G-Code to Fanuc's Teach Pendant code already existed and was available for free. There are also many free programs available to convert STL to G-Code. The process to get the desired TP code for the robot worked as follows:

1. Obtain a 3D model and save it as an STL file using any standard CAD software.
2. Use any preferred "slicing" software to convert the STL file to G-Code.
3. Type the G-Code filename into the appropriate line in the G-Code-to-TP conversion Java program.
4. Compile the Java program in Windows command prompt or any Java compiler.
5. Run the Java program. A LS file is produced.
6. Upload the LS file into Fanuc's ROBOGUIDE software and build it as a TP file.
7. Download the TP file to a flash drive, then to the robot controller.
8. Run the program on the robot.

2.11 Programming Extruder Stepper Motor

The extruder's stepper motor was to be controlled by the Arduino microcontroller. The Arduino software makes it simple to program a stepper motor. It includes a command library that allows the programmer to write simple instructions to tell the motor the direction, speed, and number of steps to move. Nonetheless, there are two major issues with the stepper motor design.

The first major issue is regarding how to tell the stepper motor when to run and when to stop, as well as how fast to run. In a typical 3D printing system, the G-Code program provides these instructions. In this system, however, the G-Code instructions are being loaded to the robot controller and there isn't a simple method to pass the instructions from the robot controller to the Arduino. There was not enough time to program this functionality, so the Arduino was programmed to have the stepper motor start when a button is pressed and then run at a constant speed. When the user is ready to print, he or she will press this button as soon as they start the robot's TP program. The obvious consequence is that either too much or too little material is extruded at different stages throughout the print.

The second major issue concerns the speed of the motor. To optimize the process outlined above, the motor needs to run at a constant speed and the speed should be low to not extrude too much material during the print. The stepper driver used provides a minimum step angle of 1.8°. To reduce the speed, the program can simply create a delay between each step. The delay necessary to prevent too much material from being extruded with this equipment is around one second. This means that material flow is "jerky", resulting in uneven features in the printed part. The solution would be to use a higher-resolution stepper driver with smaller step angles. The motor could then be programmed with lower delays between steps and still achieve the low speed needed. Another solution would be to use gears to reduce the output speed, but this would mean redesigning the extruder assembly.

2.12 Programming Heating Elements

To control the current going to the heating elements, a set of digital relays are used. The main circuit to the heating element is normally open. When a digital output from the microcontroller goes high, the main circuit is closed and current to the heating element heats it up. Both heating elements came with thermistors to monitor the temperature. A voltage divider circuit allows the microcontroller to measure the change in resistance in the thermistor. The relationship between resistance and temperature for the 100K NTC type thermistors used in both heating elements are available in tables on the internet. The microcontroller program handles the calculations to interpret resistance as temperature.

The method used to regulate temperature is hysteresis. The microcontroller simply turns off the relay when temperature gets too high and turns on the relay when temperature gets too low. To accommodate for overshoot, the maximum and minimum temperatures in the program are set well before the actual desired minimum and maximum temperatures. PID control is available in the Arduino libraries, which would result in steadier and more predictable temperatures, but would take significantly more time to set up. Hysteresis was good enough for the prototype.

III. Design Constraints

The goal is to design a 3D-printing process that can build very large or long parts while maintaining a relatively small footprint when not in use. It should be possible to produce parts from a CAD model, just like with a typical 3D printer.

The project will have a modest budget and preferably make use of the equipment and materials already available in the engineering lab.

The timeline is to begin work in mid-May 2019 and to have a working prototype in two months.

IV. Costs

Part #	Part Name	Description	Qty	Units	Unit Cost	Cost
1	MK8 Extruder Hot End Kit	Stepper motor, extruder, hot end, 4 nozzles, 1.75mm PLA compatible	2	assy.	$26.98	$53.96
2	3D Printer PLA Filament	PLA, 1.75mm, 2.2lbs, blue	3	spool	$16.99	$50.97
3	3D Printer PLA Filament	PLA, 1.75mm, 2.2lbs, white, glow in the dark	3	spool	$17.99	$53.97
4	3D Printer PLA Filament	PLA, 1.75mm, 2.2lbs, silver	3	spool	$15.99	$47.97
5	Power Supply	12V, 30A, universal regulated switching power supply	1	unit	$18.95	$18.95
6	Arduino Uno	Micro-controller for extruder	1	unit	$20.69	$20.69
7	L298N Motor Controller	Stepper motor drive controller board	1	unit	$7.01	$7.01
8	MK8 Extruder Nozzles	Extra nozzles	1	10-pk	$8.99	$8.99
9	Nozzle Cleaning Kit	33-piece kit	2	kit	$10.99	$21.98
10	Heat Bed and Build Plate	220mmx220mm, 12V, 200W, aluminum	1	kit	$44.99	$44.99
11	Heat Bed Platform Sticker Sheet	200mmx200mm	2	4-pk	$25.99	$51.98
	Total		**20**			**$381.46**

Figure 14: Bill of Materials

V. Project Deliverables

At the time of this writing, a partially functioning prototype is deliverable. In its current state, a user can perform the following general steps:

1. Load TP program for the desired part to the robot.
2. Press ON button on microcontroller to automatically heat hot end build plate to the appropriate temperature.
3. When heating elements are ready, start the TP program and press button on microcontroller to start extrusion.

Hardware limitations and programming gaps have resulted in a few major faults. The robot and print head do not provide feedback to each other in the current state; therefore, the amount of material extruded is usually incorrect. This results in a low-quality finished part. Another opportunity for improvement is to streamline the process of producing a TP file to load to the robot. As it stands, the user must manually perform several intermediate steps to convert the original STL file to the required TP file and the process is time-consuming and unintuitive.

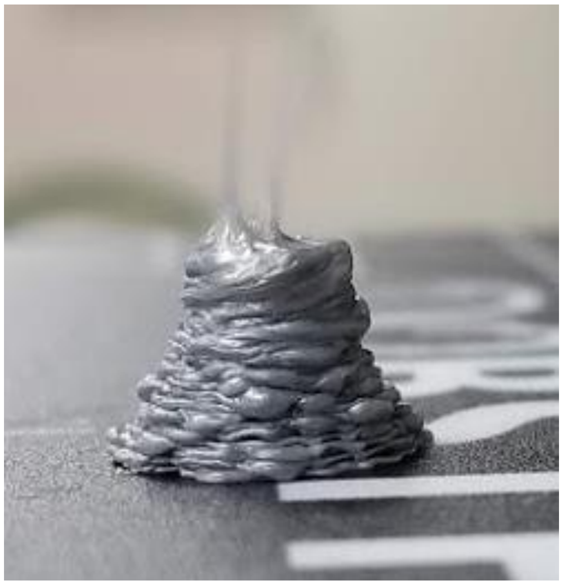

Figure 15
Produced Part: Cone

VI. Conclusion/Summary

As stated earlier in the report, one major limitation of most 3D printers on the market is build volume, meaning they cannot produce parts with large dimensions. The two main alternatives are to print the part in pieces and then assemble, or to manufacture it by other means. Assembling the part can introduce stress points and increase the complexity of the part. Manufacturing the part by other means can often mean long turnaround times and increased cost, which is usually detrimental to prototyping.

The objective of this project was to test the concept of 3D printing with a robot arm. One of the major benefits is that robot arms already have a large work envelope which can translate to a large build volume, while also maintaining the high accuracy and repeatability needed for 3D printing. Another benefit is the cost savings from not having to build or purchase an entire machine if a robot arm is already available in the facility. A user would just need to attach the required accessories, which can be removed if the robot is needed to perform other jobs.

To carry out the project at low cost and time consumption, the available resources in the university's engineering labs were exploited as much as possible. A rough prototype was produced using simple and cheap electronics, an easy-to-use Arduino microcontroller, and 3D-printed mounting assemblies. While the quality of parts produced by the prototype leave much to be desired, it should be noted that it was all put together in two months by just one senior undergraduate. With modest additional investment of time and money, the prototype can be improved to produce parts with comparable quality to that of a professional 3D printer.

VII. Future Work

Future students looking to improve on this project should first focus on implementing communication between the robot controller and the microcontroller. The microcontroller should indicate to the robot controller when the heating elements are at operating temperature so the print can begin automatically. The robot controller should indicate to the microcontroller when and how much material to extrude.

Another major improvement would be to develop a program to automate the process of converting a G-Code file to a TP file.

VIII. Bibliography

3D Hubs, 2018, “What is 3D Printing? The definitive guide”, https://www.3dhubs.com/guides/3d-printing/, Jul. 6th, 2019, 3D Hubs

Swanson, Spurgeon, Vass, and Danielewicz, 2016, “3D Printing Robotic Arm”, https://web.wpi.edu/Pubs/E-project/Available/E-project-032516-143806/unrestricted/3D_Printing_Robot_Arm_MQP_Report_3-24-25_v2.pdf, April 20th, 2019, Worcester Polytechnic Institute

Ada, 2012, “Thermistor”, https://learn.adafruit.com/thermistor/overview, June 15th, 2019, Adafruit Industries

IX. Code

9.1 Arduino Code

```
// ROBO 497 - Capstone Senior Project
// Spring 2019
// Pedro Urbina
// Advisor: Ravindra Thamma, Ph.D.

// EXTRUDER **********
#include <Stepper.h>
const int stepsPerRevolution = 200;
const int enApin = 12;
const int enBpin = 13;
int enAstate = 0;
int enBstate = 0;
// initialize the stepper library:
Stepper myStepper(stepsPerRevolution, 5, 4, 3, 2);
int stepCount = 0;        // number of steps the motor has taken
int extrudeStepCount = 10;
float motorSpeed = 20; //rpm
int extrudeDelay = 1000;

const int enableButton = 9;     // the number of the pushbutton pin
const int ledPin =  10;      // the number of the LED pin
int enableState = LOW;        // the current state of the output pin
int enableButtonState;        // variable for reading the pushbutton status
int lastEnableButtonState = HIGH;   // the previous reading from the input pin
long lastDebounceTime = 0;  // the last time the output pin was toggled
long debounceDelay = 200;    // the debounce time; increase if the output flickers

const int extrudeButton = 7;     // the number of the pushbutton pin
int extrudeButtonState;        // variable for reading the pushbutton status
int extrudeState = LOW;        // the current state of the output pin
int lastExtrudeButtonState = HIGH;   // the previous reading from the input pin

const int retractButton = 8;     // the number of the pushbutton pin
int retractButtonState = 0;        // variable for reading the pushbutton status

// HOT END **********
// which analog pin to connect
#define HOTEND_THERMISTORPIN A0
// resistance at 25 degrees C
#define HOTEND_THERMISTORNOMINAL 102900
// temp. for nominal resistance (almost always 25 C)
#define HOTEND_TEMPERATURENOMINAL 25
```

```
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define HOTEND_NUMSAMPLES 5
// The beta coefficient of the thermistor (usually 3000-4000)
#define HOTEND_BCOEFFICIENT 3950
// the value of the 'other' resistor
#define HOTEND_SERIESRESISTOR 99100
int hotend_samples[HOTEND_NUMSAMPLES];

unsigned int hotEndTargetTemp = 260;
unsigned int minHotEndTemp = 200;
unsigned int maxHotEndTemp = 280;
const int hotEndRelayPin = 6;
unsigned int hotEndTemp;

// HEATED BUILD PLATE **********
// which analog pin to connect
#define PLATE_THERMISTORPIN A1
// resistance at 25 degrees C
#define PLATE_THERMISTORNOMINAL 100000
// temp. for nominal resistance (almost always 25 C)
#define PLATE_TEMPERATURENOMINAL 25
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define PLATE_NUMSAMPLES 5
// The beta coefficient of the thermistor (usually 3000-4000)
#define PLATE_BCOEFFICIENT 3950
// the value of the 'other' resistor
#define PLATE_SERIESRESISTOR 100870
int plate_samples[PLATE_NUMSAMPLES];

unsigned int minPlateTemp = 60; //(value of the reading of your thermistor which is the lower bound of
your desired range)
unsigned int maxPlateTemp = 65; //(value of the reading of your thermistor which is the higher bound of
your desired range)
const int plateRelayPin = 11;
unsigned int plateTemp;

void setup() {

  Serial.begin(9600);

  // EXTRUDER **********
  pinMode(enApin,OUTPUT);
  pinMode(enBpin,OUTPUT);
  pinMode(enableButton, INPUT);
  pinMode(extrudeButton, INPUT);
  pinMode(retractButton, INPUT);
  pinMode(ledPin, OUTPUT);
  myStepper.setSpeed(motorSpeed);

  // THERMISTOR VOLTAGE REFERENCE **********
  analogReference(EXTERNAL);

  // HOT END **********
  pinMode(hotEndRelayPin,OUTPUT);
  digitalWrite(hotEndRelayPin,HIGH);

  // HEATED BUILD PLATE **********
  pinMode(plateRelayPin,OUTPUT);
  digitalWrite(plateRelayPin,HIGH);
}

void loop() {

  enable();

  if (enableState == HIGH) {
    hotend_thermistorReading();
    if(hotEndTemp > hotEndTargetTemp) {
        hotEndOff();
      }
      else if(hotEndTemp < hotEndTargetTemp) {
        hotEndOn();
      }

      plate_thermistorReading();
      if(plateTemp > maxPlateTemp) {
        plateOff();
      }
      else if(plateTemp < minPlateTemp) {
        plateOn();
      }

    if(hotEndTemp > minHotEndTemp && hotEndTemp < maxHotEndTemp && plateTemp > 55 &&
plateTemp < 70) {
      extrudeButtonFunction();
      retractButtonState = digitalRead(retractButton);
      if(extrudeState == HIGH) {
        extrude();
        delay(extrudeDelay);
      }
      else if(retractButtonState == HIGH) {
        retract();
      }
    }
  }
  else {
    hotEndOff();
    plateOff();
    extrudeState = LOW;
    disableMotor();
  }
}

void enableMotor() {
  digitalWrite(enApin,HIGH);
  digitalWrite(enBpin,HIGH);
}

void disableMotor() {
  digitalWrite(enApin,LOW);
  digitalWrite(enBpin,LOW);
}

void hotEndOn () {
  digitalWrite(hotEndRelayPin,LOW);
}

void hotEndOff () {
  digitalWrite(hotEndRelayPin,HIGH);
}

void plateOn () {
  digitalWrite(plateRelayPin,LOW);
}

void plateOff () {
  digitalWrite(plateRelayPin,HIGH);
}

void extrude() {
  enableMotor();
  myStepper.step(-extrudeStepCount);
  disableMotor();
}

void retract() {
  enableMotor();
  myStepper.step(extrudeStepCount);
  disableMotor();
}

void enable() {
  int reading = digitalRead(enableButton);
  if (reading == HIGH && lastEnableButtonState == LOW && millis() - lastDebounceTime >
debounceDelay) {
    if (enableState == HIGH)
      enableState = LOW;
    else
      enableState = HIGH;

    lastDebounceTime = millis();
  }
  digitalWrite(ledPin, enableState);
  lastEnableButtonState = reading;
}

void extrudeButtonFunction() {
  int reading = digitalRead(extrudeButton);
  if (reading == HIGH && lastExtrudeButtonState == LOW && millis() - lastDebounceTime >
debounceDelay) {
    if (extrudeState == HIGH)
      extrudeState = LOW;
    else
      extrudeState = HIGH;

    lastDebounceTime = millis();
  }
  lastExtrudeButtonState = reading;
}

void hotend_thermistorReading() {
  uint8_t i;
  float average;

  // take N samples in a row, with a slight delay
  for (i=0; i< HOTEND_NUMSAMPLES; i++) {
   hotend_samples[i] = analogRead(HOTEND_THERMISTORPIN);
   delay(10);
  }

  // average all the samples out
  average = 0;
  for (i=0; i< HOTEND_NUMSAMPLES; i++) {
     average += hotend_samples[i];
  }
  average /= HOTEND_NUMSAMPLES;

  //Serial.print("Average analog reading ");
  //Serial.println(average);
  // convert the value to resistance
  average = 1023 / average - 1;
  average = HOTEND_SERIESRESISTOR / average;
  //Serial.print("Thermistor resistance ");
  //Serial.println(average);

  float steinhart;
  steinhart = average / HOTEND_THERMISTORNOMINAL;     // (R/Ro)
  steinhart = log(steinhart);                  // ln(R/Ro)
  steinhart /= HOTEND_BCOEFFICIENT;                   // 1/B * ln(R/Ro)
  steinhart += 1.0 / (HOTEND_TEMPERATURENOMINAL + 273.15); // + (1/To)
  steinhart = 1.0 / steinhart;                 // Invert
  steinhart -= 273.15;                         // convert to C

  hotEndTemp = steinhart;

  Serial.print("Hot End Temperature ");
  Serial.print(hotEndTemp);
  Serial.print(" *C");
}

void plate_thermistorReading() {
  uint8_t i;
  float average;

  // take N samples in a row, with a slight delay
  for (i=0; i< PLATE_NUMSAMPLES; i++) {
   plate_samples[i] = analogRead(PLATE_THERMISTORPIN);
   delay(10);
  }

  // average all the samples out
  average = 0;
  for (i=0; i< PLATE_NUMSAMPLES; i++) {
     average += plate_samples[i];
  }
  average /= PLATE_NUMSAMPLES;

  //Serial.print("Average analog reading ");
  //Serial.println(average);

  // convert the value to resistance
  average = 1023 / average - 1;
  average = PLATE_SERIESRESISTOR / average;
  //Serial.print("Thermistor resistance ");
  //Serial.println(average);

  float steinhart;
  steinhart = average / PLATE_THERMISTORNOMINAL;     // (R/Ro)
  steinhart = log(steinhart);                  // ln(R/Ro)
  steinhart /= PLATE_BCOEFFICIENT;                   // 1/B * ln(R/Ro)
  steinhart += 1.0 / (PLATE_TEMPERATURENOMINAL + 273.15); // + (1/To)
  steinhart = 1.0 / steinhart;                 // Invert
  steinhart -= 273.15;                         // convert to C

  plateTemp = steinhart;

  Serial.print("\tPlate Temperature ");
  Serial.print(plateTemp);
  Serial.println(" *C");
}
```

9.2 Java Code (from bibliography entry 2)

Main Program

```
import java.io.*;
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class main{
public static final String programName = "PBig";
public static final String inputFileName = "cube.gcode";
public static Parser parser = new Parser();
/**
* Opens a file containing gCode.
* @param path
* @return
* @throws IOException
*/
public static Object[] OpenFile(String path) throws IOException {
FileReader reader = new FileReader(path);
BufferedReader buffReader = new BufferedReader(reader);
return buffReader.lines().toArray();
}
public static void main(String args[]) {
try {
Object[] fileContents = OpenFile("G-Code Files/" + inputFileName);
int length = fileContents.length;
PrintWriter writer = new PrintWriter(programName + ".ls", "UTF-8");
PrintWriter posWriter = new PrintWriter("posOutput.txt", "UTF-8");
PrintWriter movWriter = new PrintWriter("movOutput.txt", "UTF-8");
//writer.println("1: PR[1] = P[1];");
//writer.println("2: UFRAME[8] = PR[1];");
//writer.println("3: UFRAME_NUM = 8;");
for (int i = 0; i < length; i++) {
//System.out.println(fileContents[i]);
if (i % 500 == 0) {
//Periodically write data to file.
System.out.println("Working on " + (i+1) + " of " + length);
movWriter.print(parser.movements);
posWriter.print(parser.positions);
parser.movements = "";
parser.positions = "";
}
parser.parseLine(fileContents[i].toString());
}
movWriter.print(parser.movements);
posWriter.print(parser.positions);
parser.movements = "";
parser.positions = "";
movWriter.close();
posWriter.close();
Object[] movContents = OpenFile("movOutput.txt");
System.out.println("Length: " + movContents.length);
int movLength = movContents.length;
int num = 0;
ArrayList<String> endPos = new ArrayList<String>();
writer.println("/PROG " + programName);
writer.println("/ATTR \n/MN");
for (int i = 0; i < movLength; i++) {
/*if(i % 1000 == 0){
Pattern pointPattern = Pattern.compile("(.*)(P\\[(\\d+)\\])(.*)");
Matcher pointMatcher = pointPattern.matcher((String)
movContents[i]);
if(pointMatcher.matches()){
endPos.add(pointMatcher.group(3));
}
writer = new PrintWriter(programName + num + ".txt", "UTF-8");
55
writer.println("/PROG " + programName + num);
writer.println("/ATTR \n/MN");
num++;
}*/
//Write motion instructions to file.
synchronized (fileContents) {
writer.println((String) movContents[i]);
}
}
writer.println("/POS");
//Set up the robot frame this program will use.
writer.println("P[1]{ \n" +
" GP1:\n" +
" UF:F, UT:F,\n" +
" CONFIG: 'N U T, , 0, 0',\n" +
" X = -900.0mm, Y = 0.0mm, Z = 0.0mm, W = -180.000 deg, P = 0.000 deg, R = 0.000 deg \n" + "
};");
Object[] posContents = OpenFile("posOutput.txt");
int posLength = posContents.length;
num = 0;
```

```
for (int i = 0; i < posLength; i++) {
/*if(!endPos.isEmpty()) {
if (i % Integer.parseInt(endPos.get(0)) == 0) {
//writer = new PrintWriter(new FileOutputStream(new
File(programName + num + ".txt"), true /* append = true *///));
//Write the position data to file.
//synchronized (fileContents) {
writer.println((String) posContents[i]);
//}
//endPos.remove(0);
//}
//}
}
writer.println("/END.");
posWriter.close();
movWriter.close();
writer.close();
FileSplitter splitter = new FileSplitter(programName, movContents,
posContents);
splitter.split();
}catch (IOException error){
System.out.println("Could not open file. " + error);
}
}
}
```

Parser Program

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/**
* Created by Will on 10/29/15.
*/
public class Parser {
boolean extruderOn = false;
int lineNumber = 1; //Starting line of motion instructions.
Pattern g1Patten = Pattern.compile("(G0 |G1 )(.*)");
Pattern g2Pattern = Pattern.compile("(G2 )(.*)");
Pattern g3Pattern = Pattern.compile("(G3 )(.*)");
Pattern xPattern = Pattern.compile("(.*)X(-?)\\d+(\\.?)\\d*(.*)");
Pattern yPattern = Pattern.compile("(.*)Y(-?)\\d+(\\.?)\\d*(.*)");
Pattern zPattern = Pattern.compile("(.*)Z(-?)\\d+(\\.?)\\d*(.*)");
Pattern ePattern = Pattern.compile("(.*)E(-?)\\d+(\\.?)\\d*(.*)");
Pattern fPattern = Pattern.compile("(.*)F(-?)\\d+(\\.?)\\d*(.*)");
Pattern iPattern = Pattern.compile("(.*)I(-?)\\d+(\\.?)\\d*(.*)");
Pattern jPattern = Pattern.compile("(.*)J(-?)\\d+(\\.?)\\d*(.*)");
public static float lastXPos = 0;
public static float lastYPos = 0;
public static float lastZPos = 0;
public static int lastFeedrate = 0;
public static float lastExtruderValue = 0;
public static int pointCount = 2; //Point 1 sets the robot frame.
public static int iterations = 0;
public static int inRange = 0;
float lastTestX = 0;
float lastTestY = 0;
public static String movements = "";
public static String positions = "";
/**
* Determines if the input is a linear or arc motion. Ignores all other commands.
* @param input
*/
public void parseLine(String input){
Matcher g1Matcher = g1Patten.matcher(input);
Matcher g2Matcher = g2Pattern.matcher(input);
Matcher g3Matcher = g3Pattern.matcher(input);
if(g1Matcher.matches()){
G1Parser(input);
}else if (g2Matcher.matches()){
arcParser(input, true);
}else if(g3Matcher.matches()){
arcParser(input, false);
}
}
/**
* Parse linear movement instruction.
* @param input
*/
public void G1Parser(String input){
float xPos = 0;
float yPos = 0;
float zPos = 0;
float extruderVal = 0;
int feedrate = 0;
Matcher xMatcher = xPattern.matcher(input);
Matcher yMatcher = yPattern.matcher(input);
Matcher zMatcher = zPattern.matcher(input);
Matcher eMatcher = ePattern.matcher(input);
Matcher fMatcher = fPattern.matcher(input);
if(xMatcher.matches()){
int start = xMatcher.end(1)+1;
int end = xMatcher.start(4);
//System.out.println("Substring: " + input.substring(start, end));
xPos = Float.parseFloat(input.substring(start, end));
lastXPos = xPos;
}else{
xPos = lastXPos;
}
if(yMatcher.matches()){
int start = yMatcher.end(1)+1;
int end = yMatcher.start(4);
yPos = Float.parseFloat(input.substring(start, end));
lastYPos = yPos;
}else{
yPos = lastYPos;
}
if(zMatcher.matches()){
int start = zMatcher.end(1)+1;
int end = zMatcher.start(4);
zPos = Float.parseFloat(input.substring(start, end));
lastZPos = zPos;
}else{
zPos = lastZPos;
}
if(eMatcher.matches()){
int start = eMatcher.end(1)+1;
int end = eMatcher.start(3);
extruderVal = Float.parseFloat(input.substring(start, end));
lastExtruderValue = extruderVal;
}else{
extruderVal = 0;
}
if(fMatcher.matches()){
int start = fMatcher.end(1)+1;
int end = fMatcher.start(3);
feedrate = (int) Float.parseFloat(input.substring(start, end));
lastFeedrate = feedrate;
}else{
feedrate = lastFeedrate;
}
String posString;
String movementString = "";
```

```
//System.out.println("X: " + xPos);
posString = "P[" + pointCount +"]{ \n GP1:\n UF:8, UT:F,\n CONFIG: 'N U T, , 0, 0',\n " + " X =
"+Float.toString(xPos) + "mm," + " Y = "+Float.toString(yPos) + "mm," + " Z =
"+Float.toString(zPos) + "mm," + " W = 180.000 deg, P = 0.000 deg, R = 0.000 deg \n };";
if(extruderVal == 0){
if(extruderOn) {
movementString = lineNumber + ": DO[101]=OFF;\n";
extruderOn = false;
lineNumber++;
}
}else{
if(!extruderOn){
movementString = lineNumber + ": DO[101]=ON;\n";
extruderOn = true;
lineNumber++;
}
}
movementString = movementString + lineNumber + ": L P[" + pointCount +"] " +
Integer.toString(feedrate) + "mm/sec FINE;";
pointCount++;
lineNumber++;
movements = movements.concat(movementString + "\n");
positions = positions.concat(posString + "\n");
double distance = Math.sqrt(Math.pow((xPos-lastTestX), 2)+Math.pow((yPos-lastTestY),2));
double ratio = distance/(double) extruderVal;
double dist = xPos-lastTestX;
//System.out.println("Ratio: " + ratio + " " + inRange + "/" + iterations);
if (ratio > 35 && ratio < 37){
inRange++;
}
iterations++;
lastTestX = lastXPos;
lastTestY = lastYPos;
}
/**
* Arc parser.
* @param input
* @param clockwise
*/
public void arcParser(String input, boolean clockwise){
float xPos = 0;
float yPos = 0;
float iPos = 0;
float jPos = 0;
float extruderVal = 0;
int feedrate = 0;
Matcher xMatcher = xPattern.matcher(input);
Matcher yMatcher = yPattern.matcher(input);
Matcher zMatcher = zPattern.matcher(input);
Matcher iMatcher = iPattern.matcher(input);
Matcher jMatcher = jPattern.matcher(input);
Matcher eMatcher = ePattern.matcher(input);
Matcher fMatcher = fPattern.matcher(input);
if(xMatcher.matches()){
int start = xMatcher.end(1)+1;
int end = xMatcher.start(4);
xPos = Float.parseFloat(input.substring(start, end));
}else{
xPos = lastXPos;
}
if(yMatcher.matches()){
int start = yMatcher.end(1)+1;
int end = yMatcher.start(4);
yPos = Float.parseFloat(input.substring(start, end));
}else{
yPos = lastYPos;
}
if(iMatcher.matches()){
int start = iMatcher.end(1)+1;
int end = iMatcher.start(4);
iPos = Float.parseFloat(input.substring(start, end));
}
if(jMatcher.matches()){
int start = jMatcher.end(1)+1;
int end = jMatcher.start(4);
jPos = Float.parseFloat(input.substring(start, end));
}
if(eMatcher.matches()){
int start = eMatcher.end(1)+1;
int end = eMatcher.start(3);
extruderVal = Float.parseFloat(input.substring(start, end));
lastExtruderValue = extruderVal;
}else{
extruderVal = 0;
}
if(fMatcher.matches()){
int start = fMatcher.end(1)+1;
int end = fMatcher.start(3);
feedrate = (int) Float.parseFloat(input.substring(start, end));
lastFeedrate = feedrate;
}else{
feedrate = lastFeedrate;
}
String posString;
String movementString = "";
float thruXPos = 0;
float thruYPos = 0;
Point radiusPoint = new Point(iPos, jPos);
Point startPoint = new Point(lastXPos, lastYPos);
Point endPoint = new Point(xPos, yPos);
float radius = distance(lastXPos, lastYPos, iPos, jPos);
float startTheta = radiusPoint.getAngle(startPoint);
float endTheta = radiusPoint.getAngle(endPoint);
System.out.println("Radius: " + radius);
System.out.println("Start point: " + startPoint.x + ", " + startPoint.y + "End point: " + endPoint.x
+ ", " + endPoint.y);
System.out.println("Start theta: " + startTheta + "End theta: " + endTheta);
float newTheta = 0;
if(clockwise) {
if (endTheta < startTheta) {
//Does not reset theta
newTheta = startTheta - ((startTheta - endTheta) / 2);
} else {
//Does reset theta
newTheta = (startTheta + 360) - (((startTheta + 360) - endTheta) / 2);
}
}else{
if (endTheta < startTheta) {
//Does not reset theta
newTheta = endTheta - ((startTheta - endTheta) / 2);
} else {
//Does reset theta
newTheta = (endTheta + 360) - (((startTheta + 360) - endTheta) / 2);
}
}
thruXPos = radius*(float)Math.cos((Math.toRadians(newTheta)));
thruYPos = radius*(float)Math.sin(Math.toRadians(newTheta));
//System.out.println("x: " + xPos + " y: " + yPos + " i: " + iPos + " j: " + jPos);
//System.out.println("X: " + thruXPos + " Y: " + thruYPos + " Radius: " + radius + " New Theta: " +
newTheta);
assert distance(xPos, yPos, iPos, jPos) == distance(lastXPos, lastYPos, iPos, jPos);
int pointCount1 = pointCount + 1;
```

```
posString = "P[" + pointCount +"]{ \n GP1:\n UF:8, UT:F,\n CONFIG: 'N U T, , 0, 0',\n " + " X = "+
String.format("%.4f", thruXPos) + "mm," + " Y = "+ String.format("%.4f", thruYPos) + "mm," + " Z
= "+Float.toString(lastZPos) + "mm," + " W = 180.000 deg, P = 0.000 deg, R = 0.000 deg \n };\n"
+ "P[" + (pointCount1) +"]{ \n GP1:\n UF:8, UT:F,\n CONFIG: 'N U T, , 0, 0',\n " + " X =
"+Float.toString(xPos) + "mm," + " Y = "+Float.toString(yPos) + "mm," + " Z =
"+Float.toString(lastZPos) + "mm," + " W = 180.000 deg, P = 0.000 deg, R = 0.000 deg \n };";
if(extruderVal == 0){
if(extruderOn) {
movementString = lineNumber + ": DO[101]=OFF;\n";
extruderOn = false;
lineNumber++;
}
}else{
if(!extruderOn){
movementString = lineNumber + ": DO[101]=ON;\n";
extruderOn = true;
lineNumber++;
}
}
movementString = movementString + lineNumber + ": C P[" + pointCount +"] \n" +
"P["+ pointCount1 +"]" + Integer.toString(feedrate) + "mm/sec FINE;\n";
pointCount += 2;
lineNumber++;
movements = movements.concat(movementString);
positions = positions.concat(posString);
lastXPos = xPos;
lastYPos = yPos;
double distance = Math.sqrt(Math.pow((xPos-lastTestX), 2)+Math.pow((yPos-lastTestY),2));
double ratio = distance/(double) extruderVal;
double dist = xPos-lastTestX;
//System.out.println("Ratio: " + ratio + " " + inRange + "/" + iterations);
}
/**
* Returns the distance between the points (x,y) and (i,j)
* @param x
* @param y
* @param i
* @param j
* @return
*/
public float distance(float x, float y, float i, float j){
double result = Math.sqrt((Math.pow(i-x, 2) + Math.pow(j-y, 2)));
return (float) result;
}
```

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

File Splitter Program

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Objects;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/**
* Created by Will on 12/10/15.
*/
public class FileSplitter {
String programName;
Object[] movements;
Object[] positions;
ArrayList<ArrayList<Object>> movementBlocks;
ArrayList<ArrayList<Object>> posistionBlocks;
FileSplitter(String programName, Object[] movements, Object[] positions) {
this.programName = programName;
this.movements = movements;
this.positions = positions;
}
public void split() {
ArrayList<String> endPos = new ArrayList<String>();
movementBlocks = new ArrayList<ArrayList<Object>>();
int num = -1;
for (int i = 0; i < movements.length; i++) {
if (i % 500 == 0) {
Pattern pointPattern = Pattern.compile("(.*)(P\\[(\\d+)\\])(.*)");
Matcher pointMatcher = pointPattern.matcher((String) movements[i]);
if (pointMatcher.matches() && i != 0) {
endPos.add(pointMatcher.group(3));
}
movementBlocks.add(new ArrayList<Object>());
num++;
}
movementBlocks.get(num).add(movements[i]);
}
//Add newline characters to the end of the movement strings.
String[] movementString = new String[movementBlocks.size()];
for (int i = 0; i < movementBlocks.size(); i++) {
movementString[i] = new String("");
for (int j = 0; j < movementBlocks.get(i).size(); j++) {
movementString[i] =
movementString[i].concat(movementBlocks.get(i).get(j) + "\n");
//System.out.println(movementString[i].toString());
}
}
num = 0;
for (int i = 0; i < movementString.length; i++) {
PrintWriter writer = null;
try {
//Write the position data to file.
writer = new PrintWriter(new FileOutputStream(new File(programName +
num + ".ls"), true));
writer.print("/PROG " + programName + num +
"\n/ATTR\n" +
"/MN\n");
num++;
//System.out.print("Printing: " + movementString[i]);
//writer.print(movementString[0]);
writer.write(movementString[i]);
} catch (FileNotFoundException e) {
e.printStackTrace();
}
writer.close();
}
String newString = "";
for (int i = 0; i < positions.length; i++) {
newString = newString + ((String) positions[i]);
}
positions = newString.split(";");
posistionBlocks = new ArrayList<ArrayList<Object>>();
```

```
int posNum = 0;
posistionBlocks.add(new ArrayList<Object>());
for (int i = 0; i < positions.length; i++) {
if (endPos.size() <= posNum) {
posistionBlocks.get(posNum).add(positions[i] + ";");
} else {
if (i == (Integer.parseInt(endPos.get(posNum)))-2) {
posistionBlocks.add(new ArrayList<Object>());
System.out.println("End Pos: " +
(Integer.parseInt(endPos.get(posNum))-2));
posNum++;
}
posistionBlocks.get(posNum).add(positions[i] + ";");
}
}
System.out.print("Block Length: " + posistionBlocks.size());
String[] positionString = new String[posistionBlocks.size()];
//System.out.println("\nThis is here: " + movementBlocks.get(0).get(7).toString());
for (int i = 0; i < posistionBlocks.size(); i++) {
positionString[i] = new String("");
for (int j = 0; j < posistionBlocks.get(i).size(); j++) {
positionString[i] =
positionString[i].concat(posistionBlocks.get(i).get(j) + "\n");
//System.out.println(movementString[i].toString());
}
}
num = 0;
System.out.println("Length: " + positions.length);
//System.out.print("First pos: " + positions[1].toString());
for (int i = 0; i < positionString.length; i++) {
PrintWriter writer = null;
try {
//Write the position data to file.
writer = new PrintWriter(new FileOutputStream(new File(programName +
num + ".ls"), true));
num++;
//System.out.print("Printing: " + movementString[i]);
//writer.print(movementString[0]);
writer.print("/POS\n");
writer.write(positionString[i]);
writer.write("/END.");
} catch (FileNotFoundException e) {
e.printStackTrace();
}
writer.close();
/*for (int k = 0; k < endPos.size(); k++){
System.out.println("End pos: " + endPos.get(k));
}*/
}
}
public static int getLineCount(String text) {
return text.split("[\n|\r]").length;
}
}
```

Point Program

```
/**
* Created by Will on 11/10/15.
*/
public class Point {
float x;
float y;
Point(float x, float y){
this.x = x;
this.y = y;
}
/**
* Returns degrees.
* @param target
* @return
*/
public float getAngle(Point target) {
float angle = (float) Math.toDegrees(Math.atan2(target.y - y, target.x - x));
if(angle < 0){
angle += 360;
}
return angle;
}
}
```