GNG1103

Design Project User Manual

Team 19 – OctoPi User Manual

Submitted by:

Team OctoPi, Group 19

Senthura Yogarajan, 300077498

Dayu Liu, 300063931

Tom Lin, 300076426

Kian Mozafarian, 300138481

Alexander Steeves, 300085657

Date: December 7, 2019

University of Ottawa

Abstract

This project is being executed because we want to solve problems regarding the automation of STEM. We have empathized with our clients by becoming users in their space and listening to their needs. We then defined a problem statement in which we highlighted the key problems faced by our clients. We have ideated to assist in brainstorming possible designs and solutions. We created an initial prototype in which it deemed successful based on our criteria to evaluate it. Our second prototype was successful as well. Our final prototype incorporated elements from both the first and second prototype and is successful in its applications. Our project has been evaluated by our clients, judges, and users of Makerspace and they approve of our product. Limitations out of control exist in which we are unable to integrate our product into the current system of Makerspace. However, we plan on working with CEED staff, our clients, and the University of Ottawa in integrating our product to automate the 3D printers in Makerspace and the STEM building. This document is a user manual in which we will guide people who would like to design something like our product to automate 3D printers or enhance the existing product. The document will explain the major problems faced and how they were overcome to prevent the reader from making the same mistakes. This will be a user manual to help the reader to achieve the same results we have from our project.

Table of Contents

Abstract ii

Table of Contents iii

List of Figures iv

List of Tables vi

List of Acronyms vii

1. Introduction 1

2. How the Prototype is Made 9

2.1. Mechanical 22

2.1.1. BOM (Bill of Materials) 22

2.1.2. Equipment list 23

2.1.3. Instructions 23

3. How to Use the Prototype 24

3.1. Setting up link between printer and instance: 24

3.2. Uploading prints: 27

3.3. Finding Key Features and Making Changes in OctoPrint: 31

3.4. Starting python for HTTP Get Requests: 34

3.5. Using Ross Dashboard to Monitor Prints: 36

4. How to Maintain the Prototype 40

5. Conclusions and Recommendations for Future Work 43

6. Bibliography 45

APPENDIX: Design Files 46

List of Figures

Figure 1: Our UI Created with Dashboard 3

Figure 2: OctoPrint Web Interface 4

Figure 3: Raspberry Pi Enclosed in Case 6

Figure 4: Four 3D Printers Wirelessly Printing Simutaneously 7

Figure 5: Ross Video Dashboard UI Parameters 10

Figure 6: UI Code for Printer Status 11

Figure 7: UI Code for Printer Status 12

Figure 8: UI Code for Printer Status 13

Figure 9: UI Code for Printer Status 14

Figure 10: UI Code for Printer Status 15

Figure 11: UI Code for Printer Status 15

Figure 12: UI Code for Complete Button 16

Figure 13: UI Code for Filament Status 17

Figure 14: UI Code for Makerspace Timer 18

Figure 15: UI Code for Makerspace Timer 18

Figure 16: UI Code for Makerspace Timer 19

Figure 17: UI Code for Makerspace Timer 19

Figure 18: UI Code for Makerspace Timer 20

Figure 19: How the Raspberry Pi is Powered On 24

Figure 20: Rear View of Ultimaker 2+ Indicating the USB Port 25

Figure 21: Enabling the Hotspot on the Computer 26

Figure 22: Connection Section of OctoPrint's Web Interface 26
Figure 23: State Section of OctoPrint's Web Interface 27
Figure 24: Files Section of OctoPrint's Web Interface 28
Figure 25: Load and Print Button 29
Figure 26: OctoPrint State Change 29
Figure 27: Adjusting and Monitoring Temperature 30
Figure 28 Selection tool bar 31
Figure 29: Settings Tab 31
Figure 30: Filament Manager Web Interface 32
Figure 31: Filament Manager Details 33
Figure 32: Python Code OctoPrint Variables 34
Figure 33: Python Code Plugin Variables 35
Figure 34: Notepad Response File in JSON Format 35
Figure 35: Dashboard with Printers in Operational State 38
Figure 36: Dashboard Displaying Multiple States of Multiple Printers 39
Figure 37: Dashboard Displaying Paused Prints and an OctoPrint Error in One of the Printers 39
Figure 38: Dashboard Displaying Multiple Printers Printing Simultaneously 40
Figure 39: Client/User Testing Product 42
Figure 40: Sample of Our Test Results (Slic3r vs Cura) 42

List of Tables

Table 1: Ranked Client Needs Based on Design Criteria 1

Table 2: BOM 22

Table 3: State of Printers Displayed in the UI and their Meaning 36

Table 4: Features of the UI 37

List of Acronyms

Acronym	Definition
UI	User Interface

1. Introduction

Welcome to OctoPi. OctoPi is a prototype created by group 19 in order to eliminate the troubles with SD cards and automate 3D printing in Makerspace. This document serves as a user manual to explain how we have created OctoPi so that you can recreate it and further expand its capabilities. The task of our GNG1103 class was to create a product that helped enhance the STEM building in a positive way, focusing on Makerspace and Brunsfield. After speaking with our clients, CEED representatives that worked in these spaces, we created a list of client needs as seen in Table 1. After ranking the list of client's needs, we decided that the most important needs were to reduce the amount of SD cards lost, decrease the time spent to start a print, and enhance the current system with the automation of 3D printers.

Table 1: Ranked Client Needs Based on Design Criteria

#	Needs	Design criteria
1	Allowing for usage of tools (e.g. 3D printers) to be used in a more organized and faster manner	• Less time wasted when using Makerspace tools (mins) • Decreased time in vacancy of tools (mins)
2	A way to let students know what machines are being used before entering the Maker Space and Brunsfield Center.	• Decreased time in vacancy of tools mins)

3	Tracking system for signed out tools such as SD cards allowing for items to be returned and not lost.	• Decrease in amount of SD cards lost (# of SD cards lost)
4	A way to keep track of materials/tools in stock especially in the Brunsfield Center.	• Decrease in amount of questions pertaining to stock of materials/tools (# of questions) • Decrease in the number of tools misplaced or lost (# of tools lost/misplaced)
5	A better way to help users navigate STEM	• Decrease in lost users (surveying users)
6	A tool or system indicate users to pack up/clean up before the rooms are closed	• Decrease in the time it takes to close (mins)
7	A functional security system for Makerspace. Makerspace currently doesn't have cameras set up for security.	• Decrease in theft and damage (# of things lost or damaged)

1. Ross Video Dashboard:

Ross video Dashboard is a company that sponsored our GNG1103 class, asking us to incorporate their Dashboard software in our product. We utilised their software to create a UI that displays 3D print details of all printers attached to the pi. The final prototype of our Dashboard UI can be seen in Figure 1. Our UI displays the status of the printers, the time left and time spent printing for a print, the status of completion, and other key details such as a progress bar, the file name, and the filament remaining. Our idea incorporated the Dashboard to be displayed on a large monitor in Makerspace, so that users can see the status of the printers before walking into the space or before approaching the 3D printers. A possible continuation of OctoPi would be to display our UI to a website so that Makerspace users can observe the status of the printers online before having to enter Makerspace.

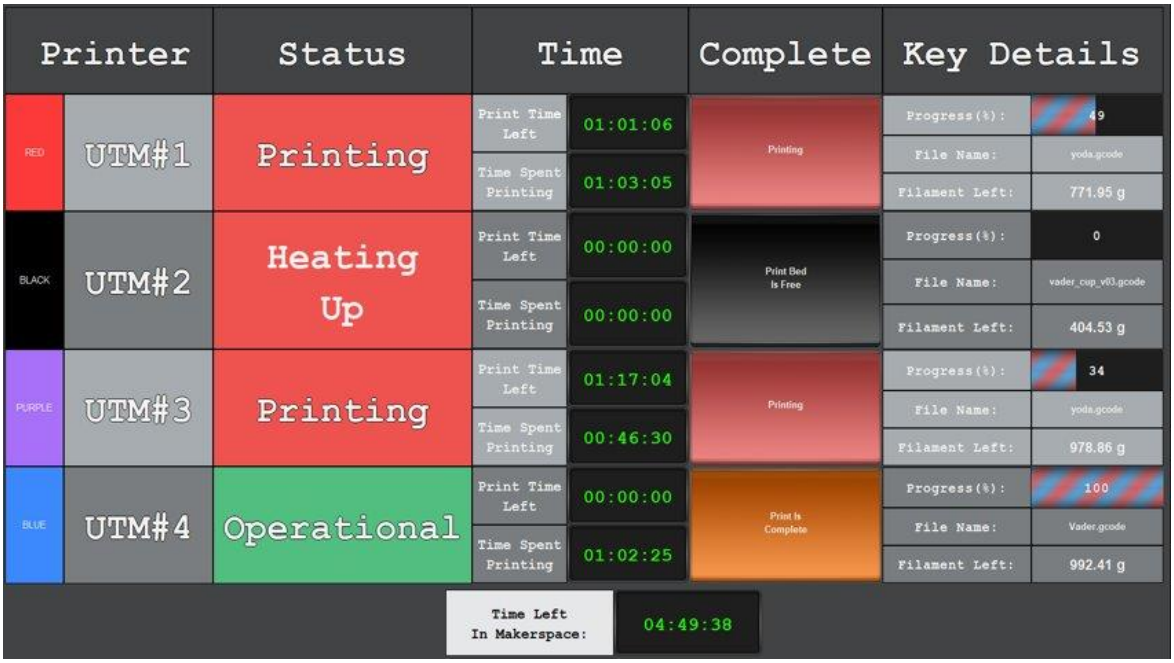

Figure 1: Our UI Created with Dashboard

2. OctoPrint Software:

OctoPrint is an open source software used to wirelessly upload g-code. We use the software to connect to 3D printers via a raspberry pi. OctoPrint monitors the states of the 3D print and displays the temperatures of the print head (the red line) and the print bed (the blue line) of the 3D printer as seen in Figure 2. We use HTTP GET requests to get this information and display it in our UI. This will be explained later in the document. OctoPrint can be downloaded from: https://OctoPrint.org/download/.

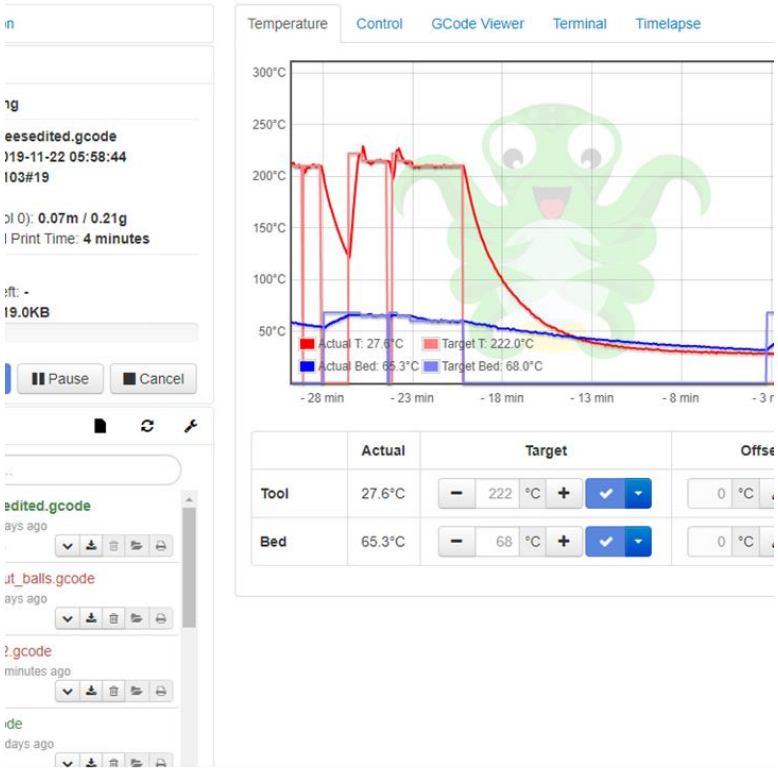

Figure 2: OctoPrint Web Interface

3. The Raspberry Pi:

The raspberry pi is an essential part of our prototype as it connects OctoPrint to the 3D printers. One raspberry pi alone can host up to four printers as it only has four USB ports. After benchmarking, we discovered that up to nine printers can be monitored through one raspberry pi by using a USB hub. We did not pursue this because the project didn't allocate enough funds to purchase that many USB to printer cables and a USB hub. OctoPrint can wirelessly send g-code to the raspberry pi when the OctoPrint instance and the pi are on the same network. This was done by hosting a hotspot on a computer that the pi will always connect to. To add, OctoPrint also allows you to slice the STL files within the OctoPrint web interface to be then sent straight to the printer. However, this requires you to upload slicing profiles to OctoPrint and we found that slicing in different software's such as Cura and Slic3r provided much more freedom of customization. Figure 3 and Figure 4 show the raspberry pi and its current capabilities without a USB hub.

Figure 3: Raspberry Pi Enclosed in Case

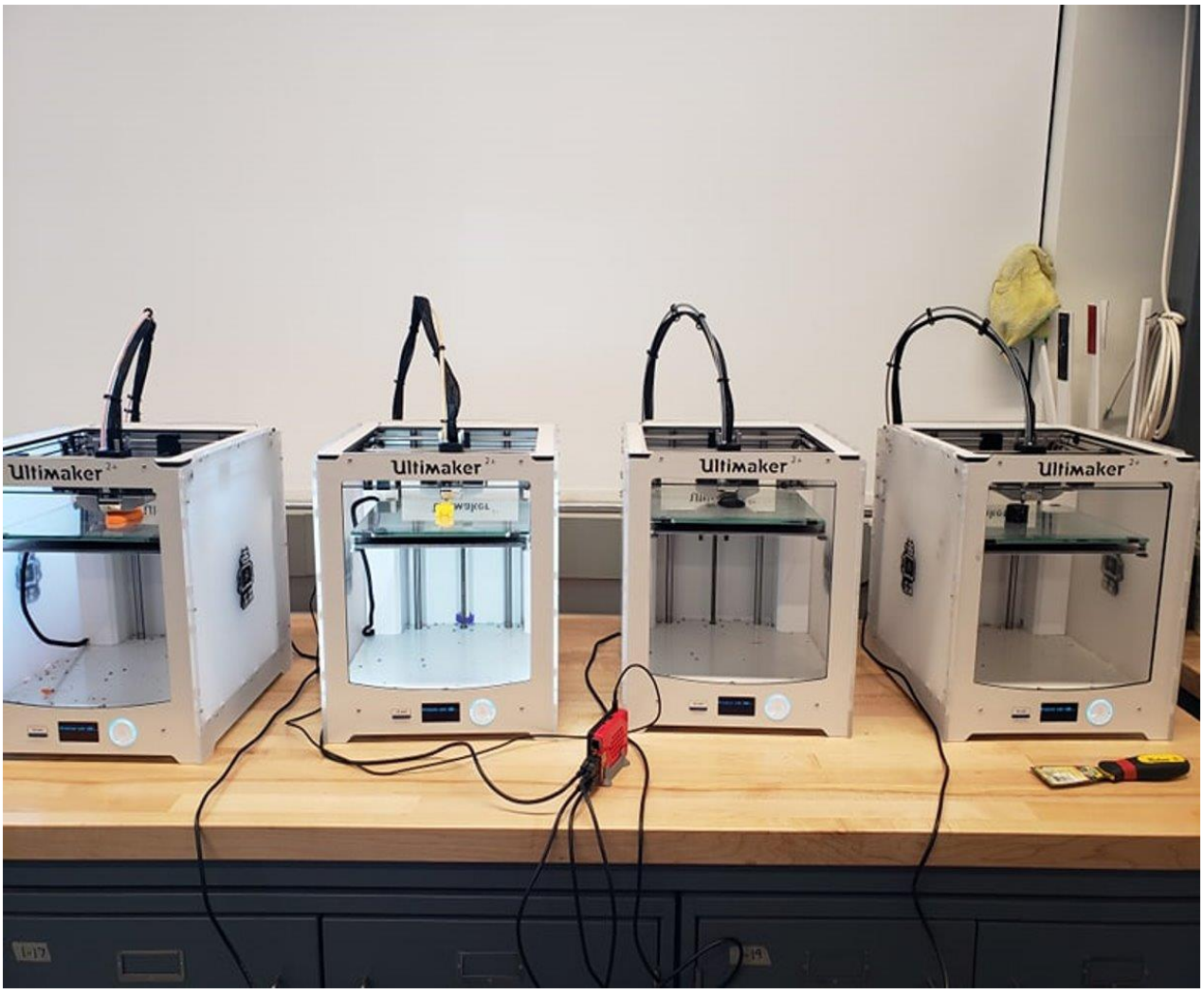

Figure 4: Four 3D Printers Wirelessly Printing Simutaneously

Our prototype aims to eliminate SD cards from 3D printing in Makerspace, control printers wirelessly, decrease set up time, and to automate printers. This is what sets our product apart from others. This is important because currently in Makerspace SD cards are often stolen, misplaced or lost which causes the Makerspace staff extra money to replace. The current printing system is also flawed as it requires people to take SD cards, use them for specific printers and put them back. This does not always happen as some users put them in the wrong spot or misplace

them leading to a disorganised and slow system. The primary users of our product are Makerspace users who use 3D printers and Makerspace staff.

Our product functions to wirelessly upload g-code to 3D printers and monitor the status of multiple 3D printers as well as key details such as filament remaining per printer. This is important for the user because they will be able to identify which printer has the colour of filament the user would like to initiate their print on. Not only that, our product indicates the user and staff how much filament is left in each printer. This is useful information because the user will know if their print will finish without failing due to lack of filament. Not to mention, this feature is important to the staff of Makerspace because our product will give them an accurate measurement of how much filament is left in each printer so the staff can reload new filament spools when necessary without having to reach behind the printers to check. Our product can give users accurate real time key details about each printer which enhances their user experience in Makerspace and eliminates their wasted time looking for available printers and initiating prints with the current system in place.

2. How the Prototype is Made

The software of our prototype consists of three major sections. The Ross video Dashboard (status UI), the raspberry pi, and the OctoPrint (print UI). The 3 sections are all connected. For example, if the status were to change on OctoPrint, then the Dashboard UI will display data accordingly.

Ross Dashboard User Interface:

Dashboard is the tool that was used to make a graphical interface for the data collected from OctoPrint. The data shown in OctoPrint may be hard for some users to understand or even see, so we used Dashboard to make a clean interface for users to see how long their print is, which printers are free, the status of the print, the progress of the print being done, and how much filament is left. To add, only the user with access to OctoPrint can check the status of the printer. However, by making the Dashboard UI, it allows other users to check the status as well. A monochromatic colour palette was used to assist users in easily navigating the UI. Labels have embedded text to make the UI informative for colour blind users and for user's who find text more easily to comprehend compared to colours. There are many open source GUI creator applications that we did not test due to constraints of the project.

List of Parameters:

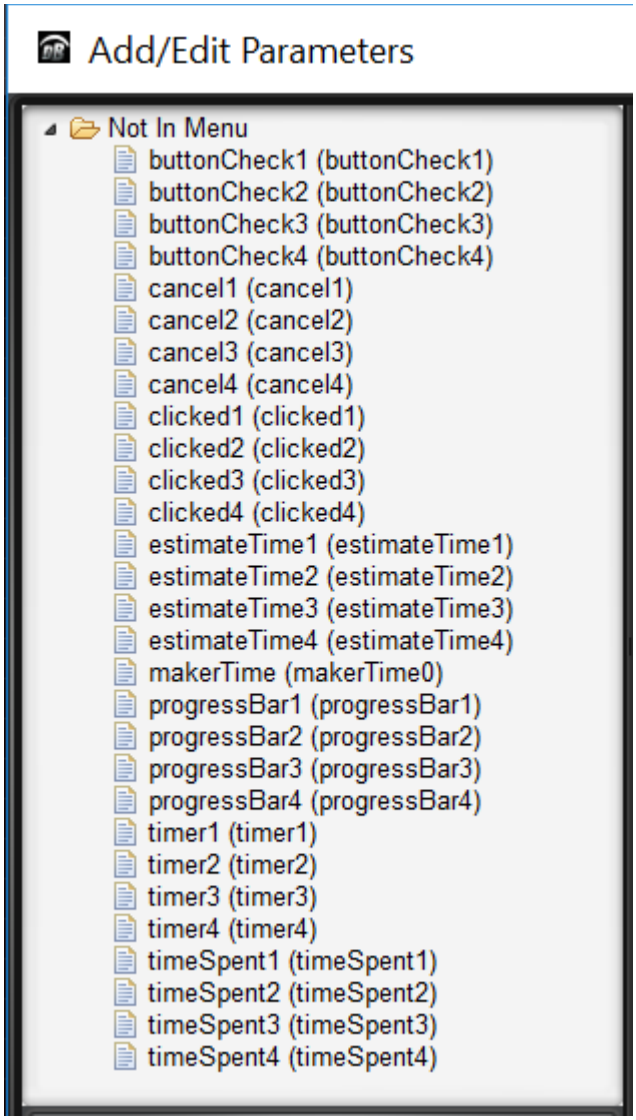

Figure 5: Ross Video Dashboard UI Parameters

Printer Status:

```
var statusFile = ogscript.getFile("C:/Users/Senthura/Desktop/Response Files for Printers/UTM#1.json") // Getting response file from file path
var fileInputParser = ogscript.createFileInput(statusFile); //reading the status from the file
var fileSize = fileInputParser.getSize(); //getting the size of the file
var myUTF = fileInputParser.readString(fileSize); //parsing through the file
ogscript.debug(myUTF); //debug script to check if the file is loaded
var json = JSON.parse(myUTF); //json object file used to to read each value in the repsonse file

var status = json.state; //status
var percent = json.progress.completion;//completion percent
var time = json.progress.printTimeLeft; //print time left
var pTime = json.progress.printTime; //time spent printing
var name = json.job.file.name; //file name

ogscript.rename("printTimeLeft1","Print Time"+"\n"+"Left"); //Renaming labels that span multiple lines
ogscript.rename("timeSpentPrinting1","Time Spent"+"\n"+"Printing");
```

Figure 6: UI Code for Printer Status

The code in Figure 6 shows how the response file is accessed and parsed through. Variables are given values based on the parsed response file. Some text in the labels require to be spanned for multiple lines and cannot be done in the label setting in Dashboard.

```
if(time!=null){//checking time left conditions
   var hours = parseInt(time/3600);
   var mins = parseInt((time-(3600*hours))/60);
   var sec = time-(3600*hours)-(mins*60);
   if(hours<10){ //formatting the timer label
      if(mins<10){
         if(sec<10){
            var display = "0"+hours+":0"+mins+":0"+sec;
         }
         else{
            var display = "0"+hours+":0"+mins+":"+sec;
         }
      }
      else{
         if(sec<10){
            var display = "0"+hours+":"+mins+":0"+sec;
         }
         else{
            var display = "0"+hours+":"+mins+":"+sec;
         }
      }
   }
   else{
      if(mins<10){
         if(sec<10){
            var display = ""+hours+":0"+mins+":0"+sec;
         }
         else{
            var display = ""+hours+":0"+mins+":"+sec;
         }
      }
      else{
         if(sec<10){
            var display = ""+hours+":"+mins+":0"+sec;
         }
         else{
            var display = ""+hours+":"+mins+":"+sec;
         }
      }

   }
   params.setValue('timer1', 0, display);
}
else{
   params.setValue('timer1', 0, "00:00:00");
}
```

Figure 7: UI Code for Printer Status

Figure 7 shows the code for how the time left for the print to be completed is converted from milliseconds, to hours, minutes, and seconds. Then these values are then formatted to be displayed on a label on the UI.

```
if(pTime!=null){//checking time spent conditions
   var phours = parseInt(pTime/3600);
   var pmins = parseInt((pTime-(3600*phours))/60);
   var psec = pTime-(3600*phours)-(pmins*60);
   if(phours<10){
      if(pmins<10){
         if(psec<10){
            var pdisplay = "0"+phours+":0"+pmins+":0"+psec;
         }
         else{
            var pdisplay = "0"+phours+":0"+pmins+":"+psec;
         }
      }
      else{
         if(psec<10){
            var pdisplay = "0"+phours+":"+pmins+":0"+psec;
         }
         else{
            var pdisplay = "0"+phours+":"+pmins+":"+psec;
         }
      }
   }
   else{
      if(pmins<10){
         if(psec<10){
            var pdisplay = ""+phours+":0"+pmins+":0"+psec;
         }
         else{
            var pdisplay = ""+phours+":0"+pmins+":"+psec;
         }
      }
      else{
         if(psec<10){
            var pdisplay = ""+phours+":"+pmins+":0"+psec;
         }
         else{
            var pdisplay = ""+phours+":"+pmins+":"+psec;
         }
      }

   }
   params.setValue('timeSpent1', 0, pdisplay);
}
else{
   params.setValue('timeSpent1', 0, "00:00:00");
```

Figure 8: UI Code for Printer Status

Figure 8 shows the code for how the time spent printing is converted from milliseconds, to hours, minutes, and seconds. These values are then formatted to be displayed on a label on the UI.

```
params.setValue('progressBar1', 0, percent); //////percent

if(status=="Offline"){//finish button conditions
   ogscript.rename("complete1",status);
   ogscript.setStyle("complete1", "bg#000000");
}
if(status=="Cancelling"){
   params.setValue("clicked1", 0, "False");
   ogscript.rename("complete1","Cancelling");
   ogscript.setStyle("complete1", "bg#FF1744");
   params.setValue("cancel1", 0, "True");
}
if(status=="Operational" && params.getValue("cancel1", 0)=="False"){
      params.setValue("buttonCheck1", 0, 0);
      ogscript.setStyle("complete1", "bg#000000");
      ogscript.rename("complete1","Print Bed"+"\n"+"Is Free");
}
if(status=="Operational" && params.getValue("cancel1", 0)=="True"){
   ogscript.setStyle("complete1", "bg#FF6D00");
   ogscript.rename("complete1","Clear"+"\n"+"Print Bed");
}
if(status=="Printing"){
   params.setValue("clicked1", 0, "False");
   ogscript.rename("complete1","Printing");
   ogscript.setStyle("complete1", "bg#EF5350");
}

if(params.getValue("buttonCheck1", 0)==0 && params.getValue("clicked1", 0)=="False" && time==0 && percent==100 && status=="Operational"){
   ogscript.setStyle("complete1", "bg#FF6D00");
   ogscript.rename("complete1","Print Is"+"\n"+"Complete");
   params.setValue("buttonCheck1", 0, 1);
}

if(name!=null){//checking file name
   ogscript.rename("file1", name);
}
else{
   ogscript.rename("file1", "No File Uploaded");
}
```

Figure 9: UI Code for Printer Status

Figure 9 shows the values given to the progress bar and the file name. Also, the conditions for the complete button are checked. The style and function of the button are decided accordingly as to what state the printer is in.

```
if(status!=null){//status conditions
   if(status!="Operational" || status!="Printing" || status!="Pasuing" || status!="Paused" || status!="Cancelling" || status!="Error" || status!="Offline"){
      ogscript.rename("status1", "OctoPrint"+"\n"+"State");
      ogscript.setStyle("status1", "bg#000000");
   }
}
if(status!=null){
   if(status=="Operational"){
      ogscript.rename("status1", status);
      ogscript.setStyle("status1", "bg#52BE80");
   }
   if(status=="Printing"){
      ogscript.rename("status1", status);
      ogscript.setStyle("status1", "bg#EF5350");
   }
   if(status=="Starting"){
      ogscript.rename("status1", "Heating"+"\n"+"Up");
      ogscript.setStyle("status1", "bg#EF5350");
   }
   if(status=="Pausing"){
      ogscript.rename("status1", status);
      ogscript.setStyle("status1", "bg#26A69A");
   }
   if(status=="Paused"){
      ogscript.rename("status1", status);
      ogscript.setStyle("status1", "bg#26A69A");
   }
   if(status=="Cancelling"){
      ogscript.rename("status1", status);
      ogscript.setStyle("status1", "bg#FF1744");
   }
   if(status=="Error"){
      ogscript.rename("status1", status);
      ogscript.setStyle("status1", "bg#FFFF00");
   }
   if(status=="Offline"){
      ogscript.rename("status1", "Printer"+"\n"+"Is OFF");
      ogscript.setStyle("status1", "bg#000000");
   }
   if(status=="Detecting baudrate"){
```

Figure 10: UI Code for Printer Status

```
   }
   if(status=="Detecting baudrate"){
      ogscript.rename("status1", "Detecting"+"\n"+"baudrate");
      ogscript.setStyle("status1", "bg#000000");
   }
   if(status=="Finishing"){
      ogscript.rename("status1", "Finishing");
      ogscript.setStyle("status1", "bg#000000");
   }
}
else{
   ogscript.rename("status1", "OctoPrint"+"\n"+"Disonnected");
   ogscript.setStyle("status1", "bg#000000");
}
```

Figure 11: UI Code for Printer Status

Figure 10 and Figure 11 show how the status label change style and value based on the state of the printer acquired from OctoPrint.

The above figures show the code meant for all the values for one printer in the UI. This code is repeated accordingly to the number of printers that is connected to the raspberry pi.

Complete Button:

```
//Reading the response file and parsing through the data
var statusFile = ogscript.getFile("C:/Users/Senthura/Desktop/Response Files for Printers/UTM#1.json")
var fileInputParser = ogscript.createFileInput(statusFile);
var fileSize = fileInputParser.getSize();
var myUTF = fileInputParser.readString(fileSize);
ogscript.debug(myUTF);
var json = JSON.parse(myUTF);

//assigning values from the response file to variables
var status = json.state;
var percent =json.progress.completion;
var time = json.progress.printTimeLeft;

//checking conditions as to when the button can be clicked
//assigning tasks for the button to set styles and text based on conditions for the button to be clicked
if(params.getValue("buttonCheck1", 0)==1 && params.getValue("clicked1", 0)=="False" && time==0 && percent==100 && status=="Operational"){//finish button conditions
   ogscript.setStyle("complete1", "bg#000000");
   ogscript.rename("complete1","Print Bed"+"\n"+"Is Free");
   params.setValue("buttonCheck1", 0, 0);
   params.setValue("clicked1", 0, "True");
}
if(params.getValue("cancel1",0)=="True"){
   ogscript.setStyle("complete1", "bg#000000");
   ogscript.rename("complete1","Print Bed"+"\n"+"Is Free");
   params.setValue("cancel1", 0, "False");
}
```

Figure 12: UI Code for Complete Button

The code in Figure 12 shows how the conditions are set to allow the button to be clicked. The code also demonstrates what task is executed by the button based on the clickable conditions. The code above is repeated for the number of printers that are displayed in the UI.

Filament Status:

```
var statusFile = ogscript.getFile("C:/Users/Senthura/Desktop/Response Files for Printers/F#1.json") // Getting response file from file path
var fileInputParser = ogscript.createFileInput(statusFile); //reading the status from the file
var fileSize = fileInputParser.getSize(); //getting the size of the file
var myUTF = fileInputParser.readString(fileSize); //parsing through the file
ogscript.debug(myUTF); //debug script to check if the file is loaded
var json = JSON.parse(myUTF); //json object file used to to read each value in the repsonse file

var usedF= json.spools[0].used; //checking how much filament is used
var leftF = (1000-usedF).toFixed(2); //checking how much filament is left

ogscript.rename("filament1",leftF+" g"); //displaying the filament left
```

Figure 13: UI Code for Filament Status

The code in Figure 13 shows how the response file is accessed and parsed through for the filament plugin. Variables are given values based on the parsed response file. The calculation is done to determine how much filament is left based on how much filament is used. The amount of filament left is formatted and then displayed in the respective label. This code above is repeated for all the printers that the user requires to acquire the filament information from.

Time Left in Makerspace:

```
ogscript.rename("timeLeft","Time Left"+"\n"+"In Makerspace:"); //renaming label that spans multiple lines

var date = new Date(); //instance od current date

var startTime = "8:30:00".split(':'); //value of start time as a string which is split into an array
var endTime = "20:30:00".split(':'); //value of end time as a string which is split into an array

var startDate = new Date().setHours(startTime[0], startTime[1], startTime[2], 0); //creating start date and updating start time
var endDate = new Date().setHours(endTime[0], endTime[1], endTime[2], 0); //creating end date and updating end time

var endMins = parseInt(endTime[1]); //value for end time minutes
```

Figure 14: UI Code for Makerspace Timer

The code in Figure 14 shows how the input start and end times are formatted into date objects.

```
if(date.getDay()==1 || date.getDay()==2 || date.getDay()==3 || date.getDay()==4 || date.getDay()==5 || date.getDay()==0){ //check if Makerspace is open of this day(Mon, Tues, Wed, Thurs, Fri, Sund)
    if(date>=startDate && date<=endDate){ //check if current time is between the start and end times
        var countDown = endDate-date; //check the difference between the end time and the current time

        //Get values for the differnece in time in hours, minutes, and seconds
        var h = parseInt((countDown)/3.6e+6);
        var m = parseInt((countDown-(h*3.6e+6))/60000);
        var s = parseInt((countDown-(h*3.6e+6)-(m*60000))/1000);

        //get values for the current hour, minutes, and seconds
        var currentHour = date.getHours();
        var currentMin = date.getMinutes();
        var currentSec = date.getSeconds();

        //set values for the hours, minutes, and seconds that will be diplsayed
        var displayH = h;
        var displayM = m;
        var displayS = s;
```

Figure 15: UI Code for Makerspace Timer

The code in Figure 15 shows variables defined for the amount of time left before the end time is reached and what values are to be displayed in the timer labels.

```
31
32      //check special conditions to format the timer
33      if(displayH==0){
34         if(endMins!=0){
35            var displayM =Math.abs(parseInt((endDate-date)/60000)-(parseInt((endDate-date)/3.6e+6))*60);
36         }
37         else{
38            var displayM = 59 - currentMin;
39         }
40      }
41      else{
42         if(endMins==0){
43            var displayM = 59 - currentMin;
44         }
45         else{
46            var displayM = Math.abs(parseInt((endDate-date)/60000)-(parseInt((endDate-date)/3.6e+6))*60);
47         }
48      }
49
```

Figure 16: UI Code for Makerspace Timer

The code in Figure 16 shows how the amount of time left in Makerspace is formatted differently depending on the countdown conditions.

```
49
50      //Format the timer
51      if(displayH<10){
52         if(displayM<10){
53            if(displayS<10){
54               var edisplay = "0"+displayH+":0"+displayM+":0"+displayS;
55            }
56            else{
57               var edisplay = "0"+displayH+":0"+displayM+":"+displayS;
58            }
59          }
60          else{
61            if(displayS<10){
62               var edisplay = "0"+displayH+":"+displayM+":0"+displayS;
63            }
64            else{
65               var edisplay = "0"+displayH+":"+displayM+":"+displayS;
66            }
67         }
68      }
69      else{
```

Figure 17: UI Code for Makerspace Timer

```
68          }
69          else{
70              if(displayM<10){
71                  if(displayS<10){
72                      var edisplay = ""+displayH+":0"+displayM+":0"+displayS;
73                  }
74                  else{
75                      var edisplay = ""+displayH+":0"+displayM+":"+displayS;
76                  }
77              }
78              else{
79                  if(displayS<10){
80                      var edisplay = ""+displayH+":"+displayM+":0"+displayS;
81                  }
82                  else{
83                      var edisplay = ""+displayH+":"+displayM+":"+displayS;
84                  }
85              }
86          }
87          params.setValue('makerTime0', 0, edisplay);
88      }
89      else{
90          params.setValue('makerTime0', 0, "00:00:00");
91      }
92  }
93  else{
94      params.setValue('makerTime0', 0, "00:00:00");
95  }
```

Figure 18: UI Code for Makerspace Timer

The code in Figure 17 and Figure 18 show how the remaining time is formatted for the timer label.

Installing the OctoPrint Software:

Instructions on how to install OctoPrint can be found in the link provided below. The software is compatible with Mac, Linux, and Windows. https://OctoPrint.org/download/

Setting up The Raspberry Pi Software:

To SSH into the raspberry pi to make multiple instances of OctoPrint and to set a static IP address a software called PuTTY was used. It can be downloaded from: https://www.putty.org/

Setting Up OctoPrint and Making Multiple Instances:

https://www.youtube.com/watch?v=7Saa1HpLRoM

The video describes step by step on how to set up OctoPrint and how to configure the network file to add a network which the raspberry pi can connect to. The video describes how to SSH into the raspberry pi and create multiple instances.

Setting Up a Static IP Address:

https://www.youtube.com/watch?v=LCJtUbRlIXE

The video describes step by step on how to SSH into the raspberry pi and set up a static IP address for the raspberry pi, so it does not change over time.

Hotspot/Network:

In order to wirelessly send g-code from OctoPrint to the raspberry pi we set up a hotspot called GNG1103#19. The hotspot credentials were entered into the network file of the raspberry pi.

2.1. Mechanical

2.1.1. BOM (Bill of Materials)

Table 2: BOM

Part Name	Picture of Template	Description/ Specification	Quantity	Supplier	Cost (CAD)
Raspberry Pi		N/A	1	Amazon	$55.97
Micro SD		16 GB	1	Amazon	$9.30
Printer Cable		6ft/1.8m	1	Amazon	$4.43
Printer Cable		6ft/1.8m	3	Amazon	$4.60 (per cable)

3D Printed Case + Stand		ABS/PLA	1	Thingiverse	$0
Wall plug		3 Amp/5Volt	1	N/A Similar Product On Amazon	$13.99
TOTAL COST(GST =3.59)					**$101.08**

2.1.2. Equipment list

No equipment is explicitly needed to build our prototype. A case for the raspberry pi can either be purchased or 3D printed. A metal file is recommended to file rough edges of the print to ensure that the raspberry pi case fits correctly. The remaining parts found in the BOM are key parts in making the product function properly. The printer cables are used to make a connection from the pi to the 3D printer. We tested with Ultimaker 2+. The user must first ensure that their printer is supported by OctoPrint, which they can find out at: https://github.com/foosel/OctoPrint/wiki/Supported-Printers.

2.1.3. Instructions

The only physical assembly required is for the raspberry pi case. We used a case and stand that was premade on Thingiverse, a website consisting of 3D designed models ready to download. The case was printed and assembled.

3. How to Use the Prototype

3.1. Setting up link between printer and instance:

Before setting up the link, the user must ensure that the pi is connected to a power outlet. This is done by plugging the three-amp power brick into a wall outlet and attaching a USB to micro USB from the power brick to the raspberry pi. This process can be seen below in Figure 19.

Figure 19: How the Raspberry Pi is Powered On

To let the raspberry pi connect to a Wi-Fi network, we used a hotspot hosted on a laptop. Figure 21 shows the hotspot we used. One at a time, connect the pi to a 3D printer via the printer cable, ensuring that OctoPrint recognizes the printer. Figure 20 shows where to connect the printer cable. Next, click connect on the OctoPrint web interface once the printer is recognized. This will

connect the instance of the web interface to the 3D printer. Figure 22 shows where in OctoPrint to find the connect button. Repeat this step to connect each printer to each instance of OctoPrint.

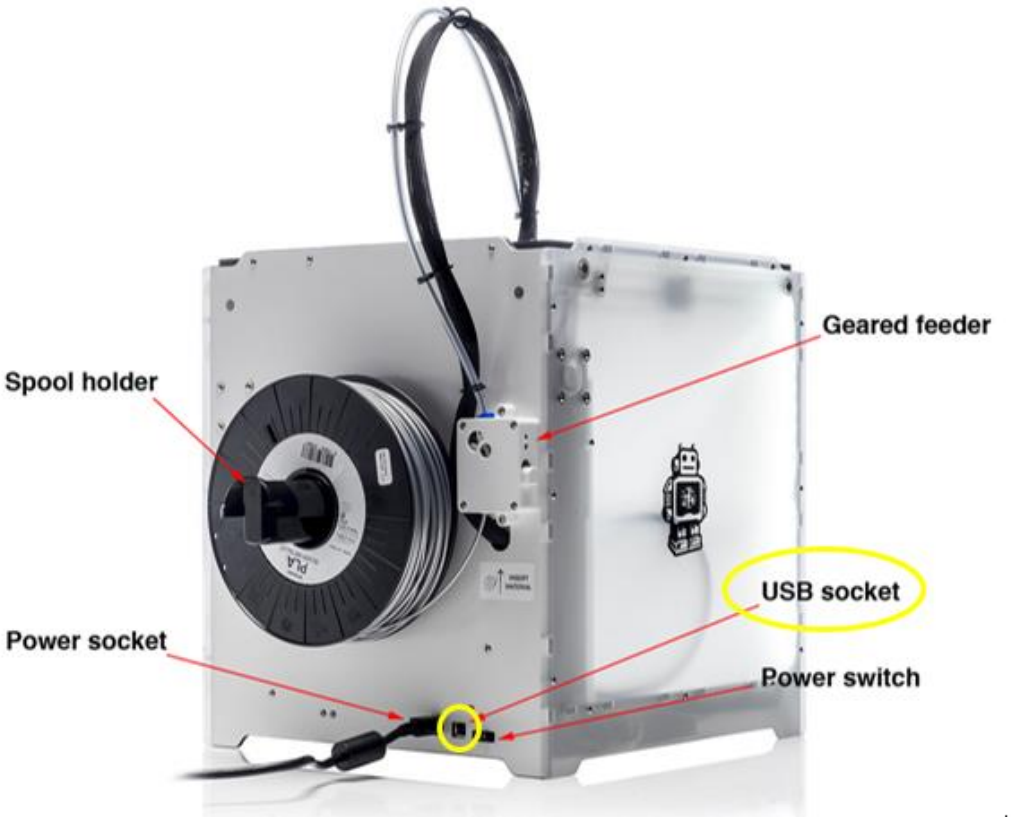

Figure 20: Rear View of Ultimaker 2+ Indicating the USB Port

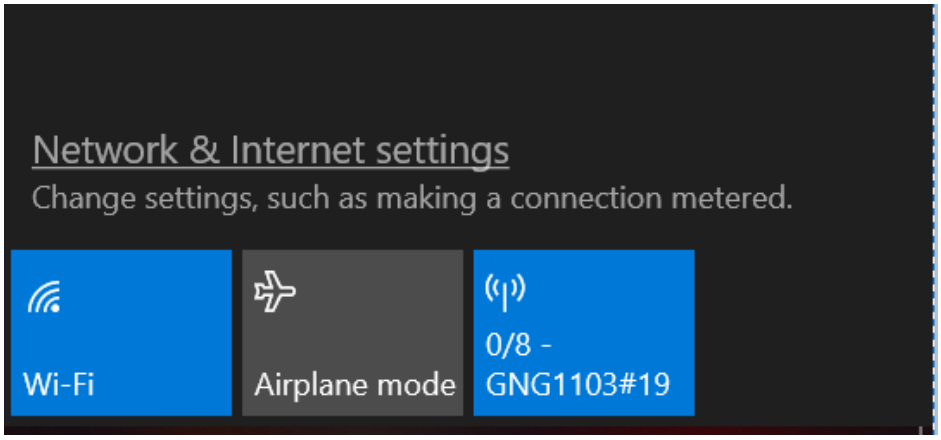

Figure 21: Enabling the Hotspot on the Computer

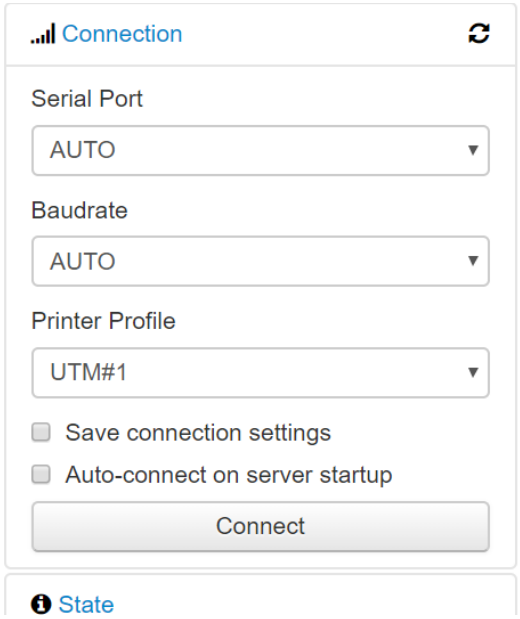

Figure 22: Connection Section of OctoPrint's Web Interface

To make sure that this step is done correctly, OctoPrint should display Operational in the status section. Now the printer is connected and ready to use. Figure 23 shows this.

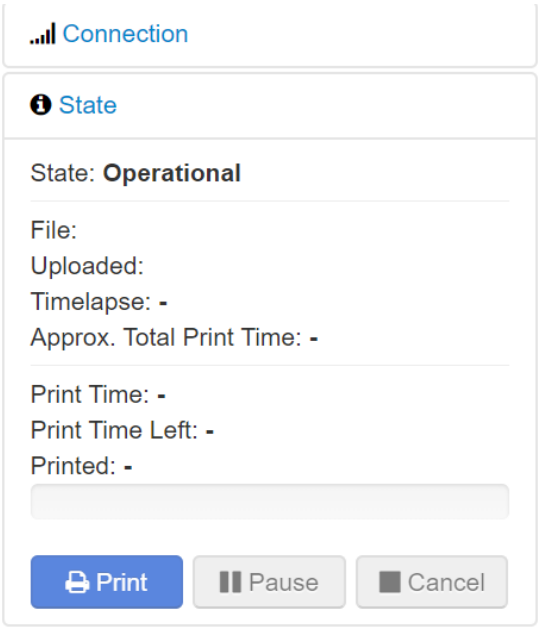

Figure 23: State Section of OctoPrint's Web Interface

3.2. Uploading prints:

In order to upload a print, one must first obtain a g-code of their desired print. We recommend using either Ultimaker Cura or Slic3r as they are both free software. From our testing, Slic3r has produced better quality prints then Cura.

Link to download Cura: https://ultimaker.com/software/ultimaker-cura

Link to download Slic3r: https://slic3r.org/download/

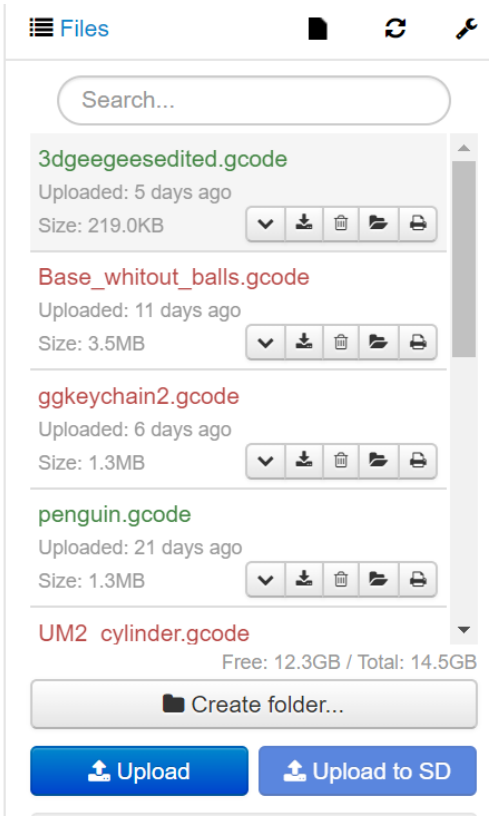

Figure 24: Files Section of OctoPrint's Web Interface

Once the web interface of OctoPrint is up and running, one can search for their g-code by clicking on the “Files” button on the left side of the OctoPrint UI as seen in the top left of Figure 24. A file can be uploaded by clicking upload and searching for your file in documents or by dragging the desired g-code file into the files section of OctoPrint. After this, click the load and print button to upload the g-code and start the print as seen in Figure 25. You can also load your g-code first and then click the “Print” button to initiate the print.

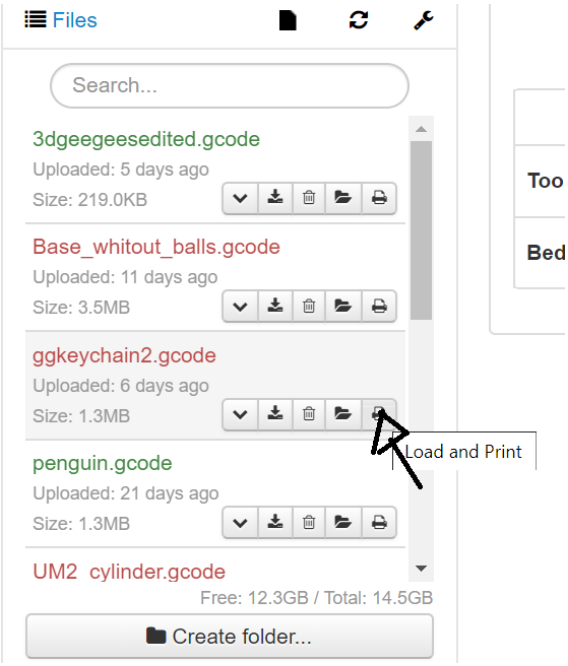

Figure 25: Load and Print Button

Once clicked, the status on OctoPrint should change to “Starting” and then to “Printing” once the desired temperatures of the bed and nozzle are reached.

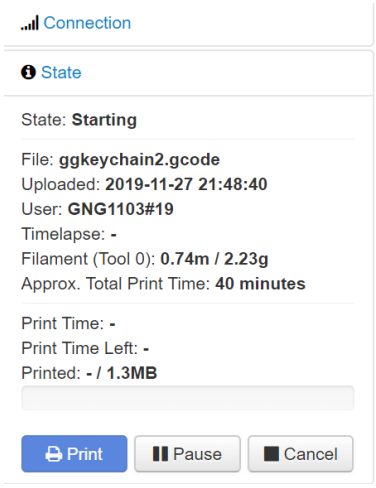

Figure 26: OctoPrint State Change

The temperature of the print bed and nozzle can be monitored and adjusted. This can be done on the right side of the web interface. The target temperature can be changed allowing the printer to start extruding filament when the actual temperatures have reached the target temperatures.

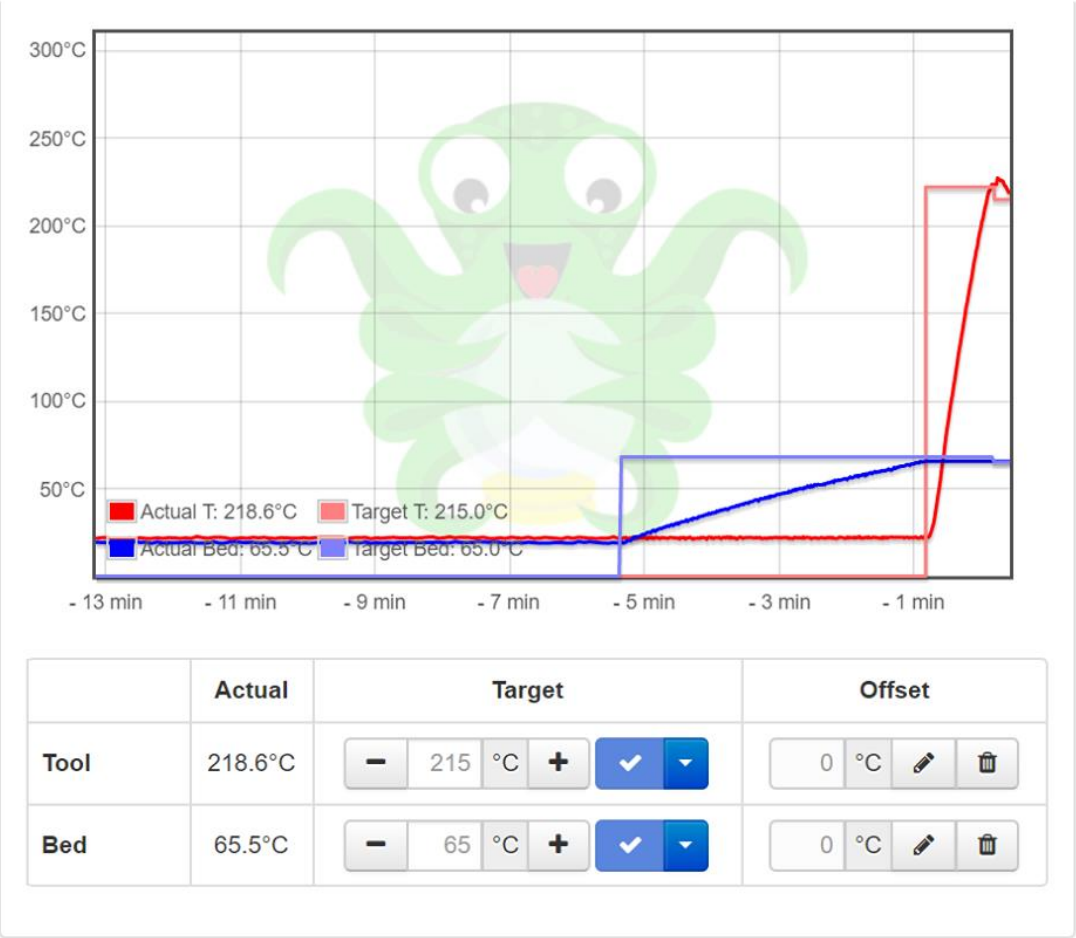

Figure 27: Adjusting and Monitoring Temperature

3.3. Finding Key Features and Making Changes in OctoPrint:

Clicking the buttons shown below will open tabs to different settings available in OctoPrint for customization. For example, clicking the wrench on the middle top of Figure 28 allows users to access the settings tab to make changes to the g-code scripts, plugins, print profiles, etc. The settings tab is shown in Figure 29.

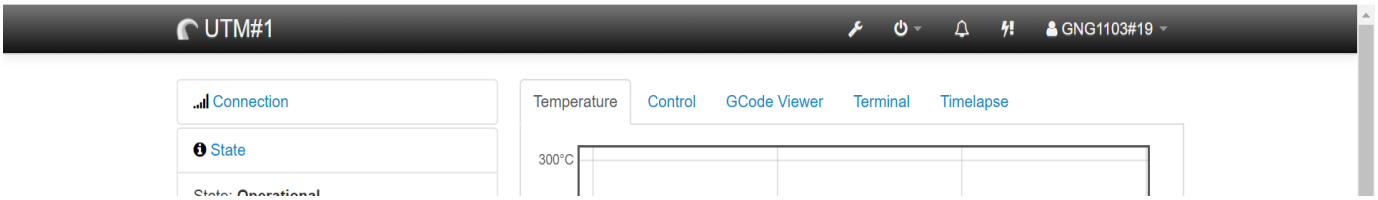

Figure 28 Selection tool bar

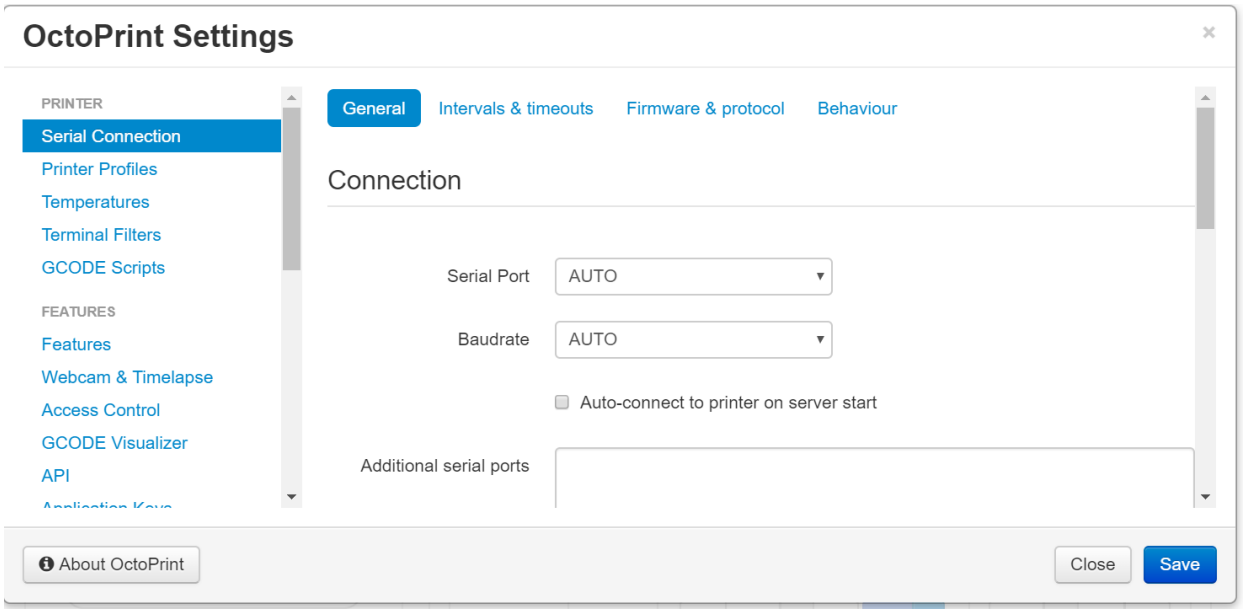

Figure 29: Settings Tab

The filament manager plugin must be installed first by searching the plugin through the settings. It can be viewed at the bottom of the OctoPrint web interface by locating the section “Filament Manager” as shown below.

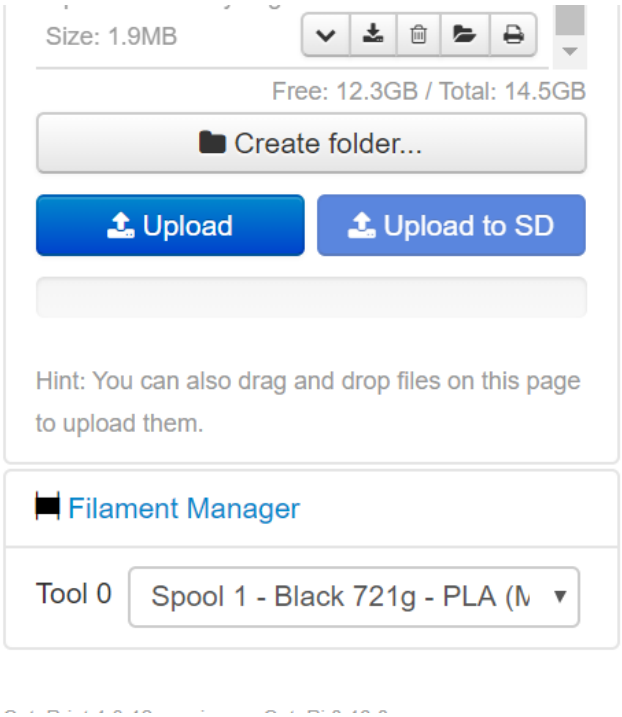

Figure 30: Filament Manager Web Interface

In the settings of the filament manager, new spools must be added with a spool profile. Values such as filament density, diameter, remaining weight of filament, and total weight of a new filament must be inputted. Once the spools are added, details of all the spools used can be checked in the settings of the filament manager as seen in Figure 31. Details displaying the weight remaining in grams, colour, and other features can be accessed. The weight can be changed by clicking the pencil icon which should be done whenever a new spool is loaded into the printer.

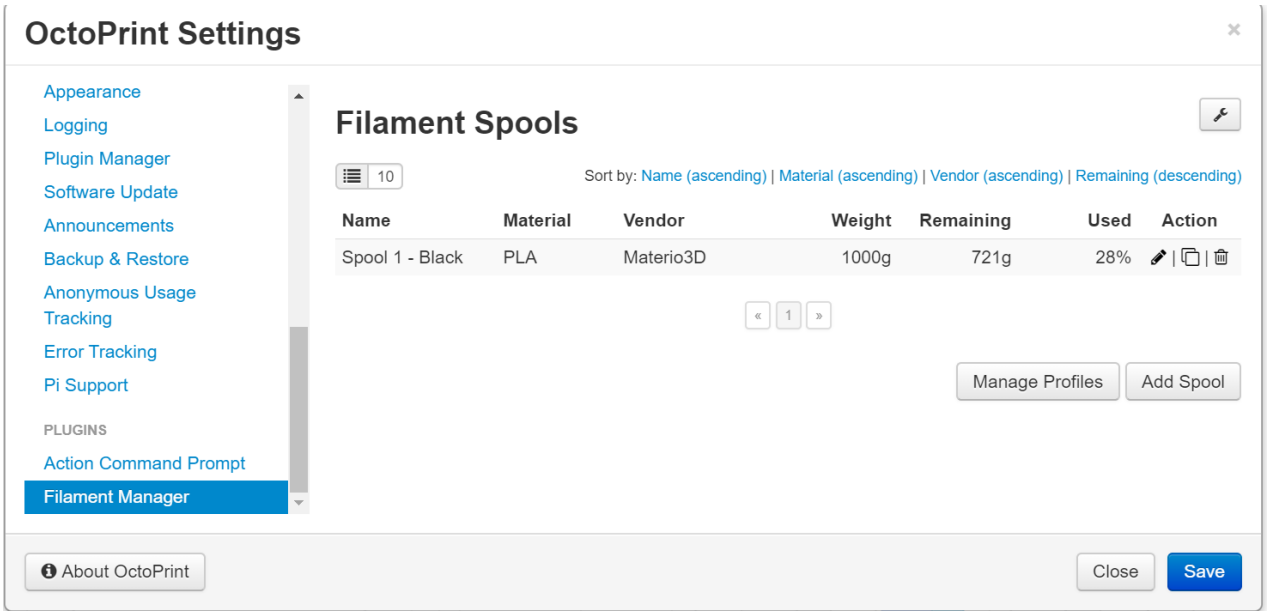

Figure 31: Filament Manager Details

3.4. Starting python for HTTP Get Requests:

Ensure that the correct URL, api-key and the file path wanted are correctly written into python code for each specific instance as show in Figure 32. This must be done for the filament plugin as well as shown in Figure 33 or the code will crash.

```
url1 = "http://192.168.137.199/api/job"
key1 = "B9DAC2A167DB4103A101C9336E5779E9"
file1 = "c:/Users/Bob/Documents/gng 1103/Project/UTM1.json"

url2 = "http://192.168.137.199:5001/api/job"
key2 = "2465E8A4F0E84DF98747E4E74BFDA051"
file2 = "c:/Users/Bob/Documents/gng 1103/Project/UTM2.json"

url3 = "http://192.168.137.199:5002/api/job"
key3 = "B17EA1D3C91844838CD7F78C1DC81084"
file3 = "c:/Users/Bob/Documents/gng 1103/Project/UTM3.json"

url4 = "http://192.168.137.199:5003/api/job"
key4 = "1C4A6752583F43F6A33D15F1D7E8D88F"
file4 = "c:/Users/Bob/Documents/gng 1103/Project/UTM4.json"
```

Figure 32: Python Code OctoPrint Variables

```
url5 = "http://192.168.137.199/plugin/filamentmanager/spools"
key5 = "B9DAC2A167DB4103A101C9336E5779E9"
file5 = "c:/Users/Bob/Documents/gng 1103/Project/F1.json"

url6 = "http://192.168.137.199:5001/plugin/filamentmanager/spools"
key6 = "2465E8A4F0E84DF98747E4E74BFDA051"
file6 = "c:/Users/Bob/Documents/gng 1103/Project/F2.json"

url7 = "http://192.168.137.199:5002/plugin/filamentmanager/spools"
key7 = "B17EA1D3C91844838CD7F78C1DC81084"
file7 = "c:/Users/Bob/Documents/gng 1103/Project/F3.json"

url8 = "http://192.168.137.199:5003/plugin/filamentmanager/spools"
key8 = "1C4A6752583F43F6A33D15F1D7E8D88F"
file8 = "c:/Users/Bob/Documents/gng 1103/Project/F4.json"
```

Figure 33: Python Code Plugin Variables

Run the python program and ensure the code works by seeing the code print "**Loaded and saved**" after each of the function calls completes. To debug, open the file in the file path written and ensure that the data being written into the notepad file is correct.

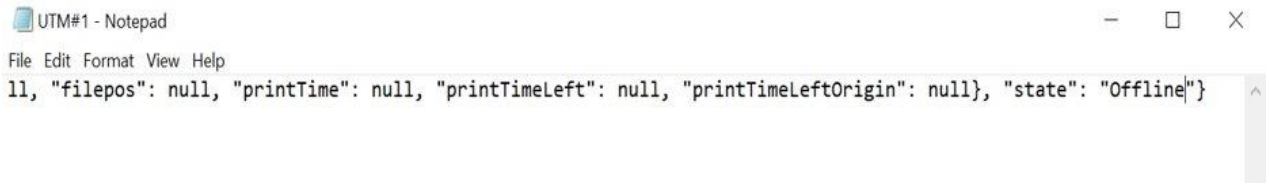

Figure 34: Notepad Response File in JSON Format

3.5. Using Ross Dashboard to Monitor Prints:

The printer status can be monitored with Ross Dashboard. The UI has many different states which will be explained here. The figures below will show what the Dashboard looks like in live time.

Table 3: State of Printers Displayed in the UI and their Meaning

State	Definition
Printer is OFF	OctoPrint is connected, but the printer is off
Error	Some error occurred with OctoPrint
Operational (no Orange button)	Printer is free to use
Operational (with Orange button)	Printer is not free to use. There is something on the print bed
Heating up	A print has been started on the printer. The printer nozzle and bed are heating up. The filament has not been extruded yet.
Printing	The printer is extruding filament and printing
Cancelling	The print job is being cancelled
Finishing	The print job is complete and the OctoPrint status is transitioning from printing to operational
Detecting baud rate	Detecting at what rate OctoPrint communicates with the printer.

OctoPrint State/Error	All other states in OctoPrint that are not significant

Table 4: Features of the UI

Features	Description
Print Time Left	How much time is remaining in the print (hours, minutes, and seconds)
Time Spent Printing	How long the print has gone (hours, minutes, and seconds)
Progress (%)	The percent completion of the print
File Name	The name of the file being printed
Filament Left	The amount of filament left in grams
Complete Button	Orange button that appears when a print job is cancelled or complete. This button indicates to the user that the print bed needs to be cleared before another print job can be started.

Figure 35: Dashboard with Printers in Operational State

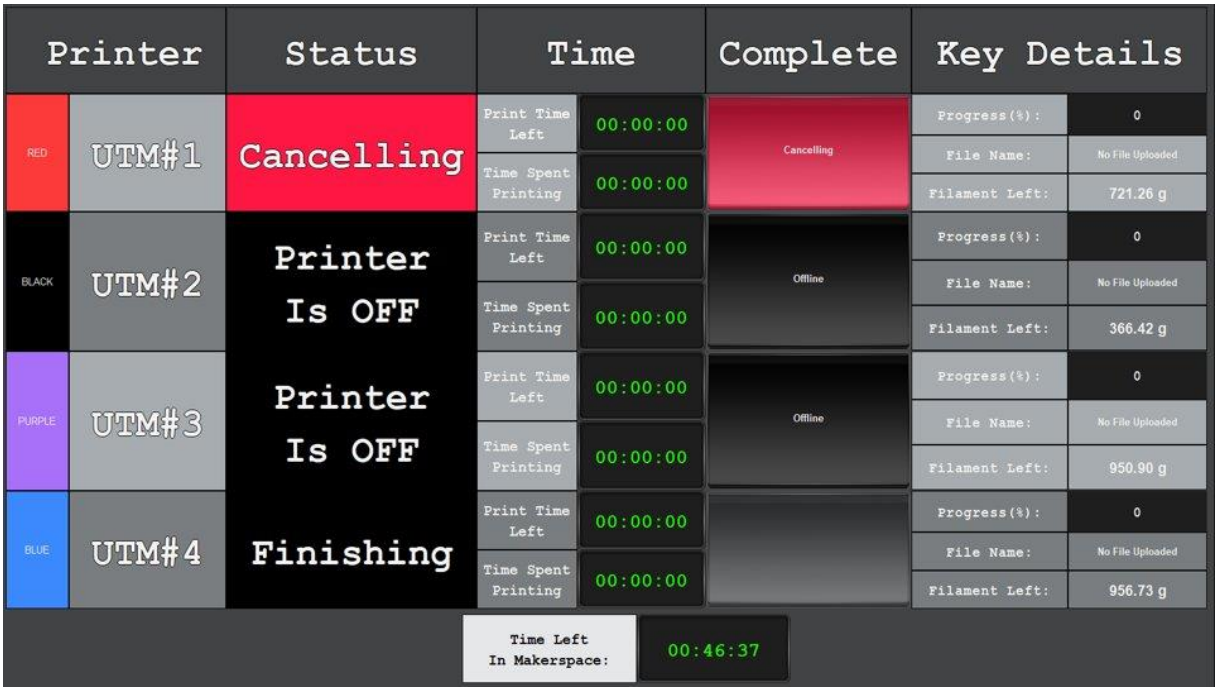

Figure 36: Dashboard Displaying Multiple States of Multiple Printers

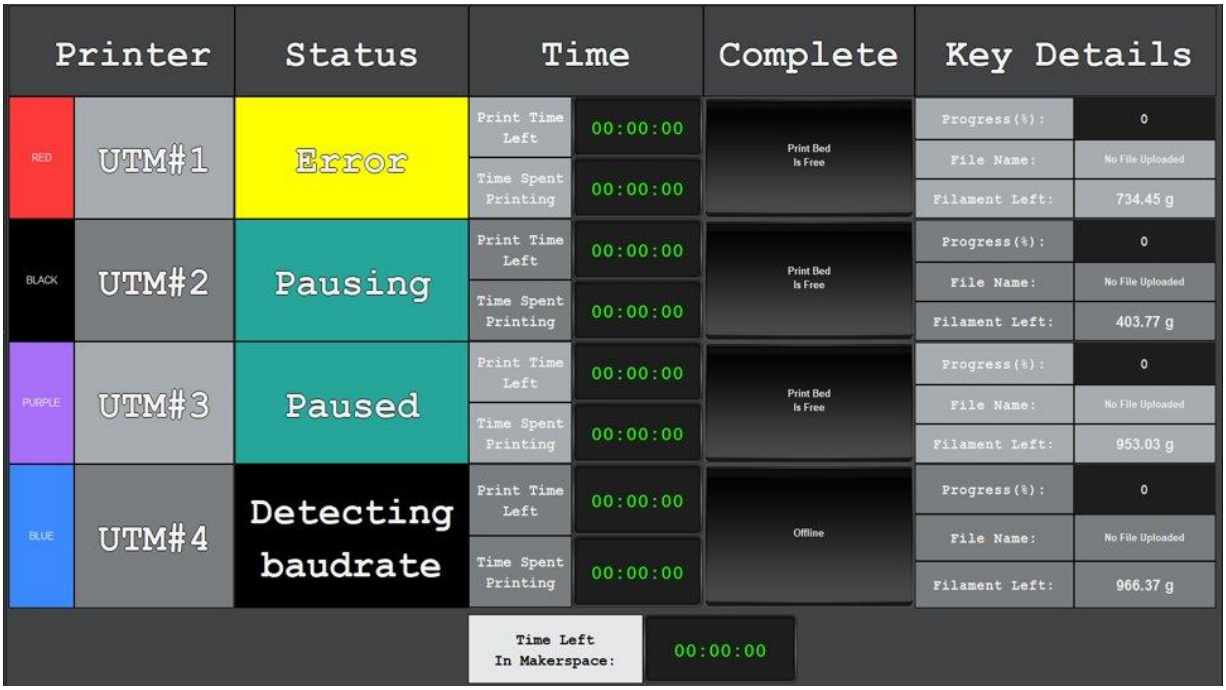

Figure 37: Dashboard Displaying Paused Prints and an OctoPrint Error in One of the Printers

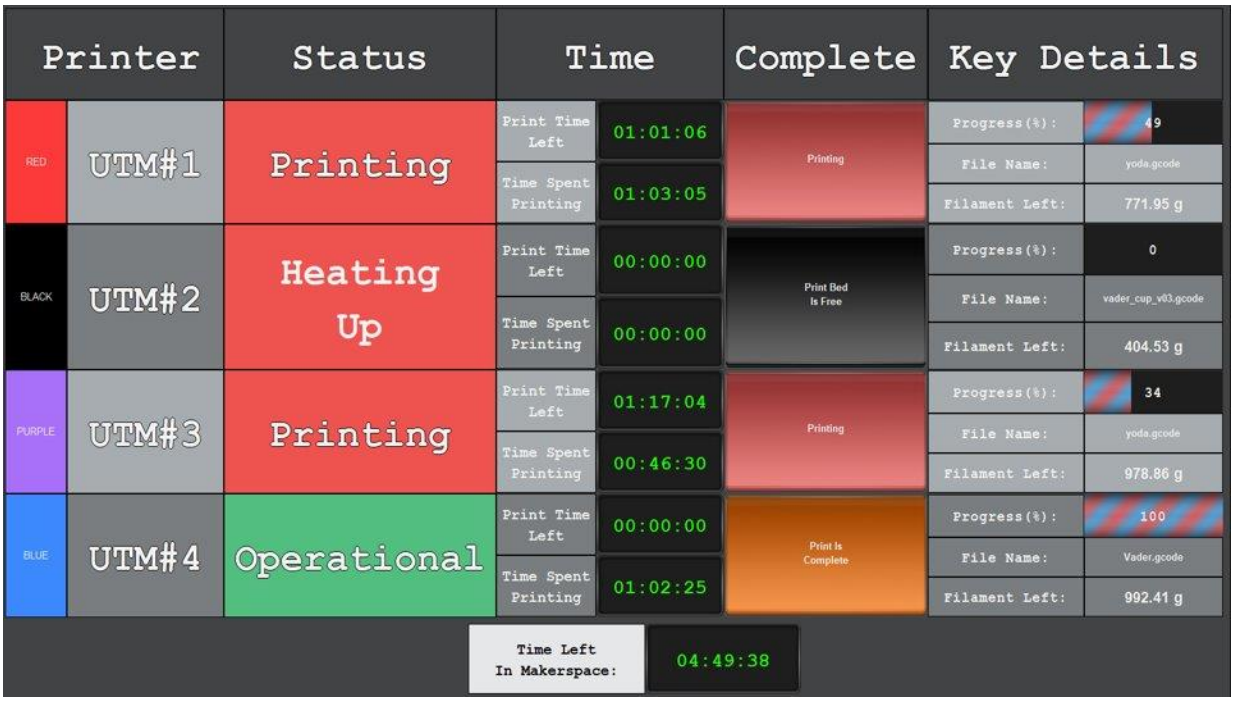

Figure 38: Dashboard Displaying Multiple Printers Printing Simultaneously

To edit the source code of the UI, in the Ross Video Dashboard software, press Ctrl+G to enter edit mode and click the part where you want to edit.

4. How to Maintain the Prototype

Main Points After Testing

- The slicing software that provided consistent high-quality prints was Slic3r. When slicing prints with Cura, the successful print percentage dropped nearly 50%.
- If the raspberry pi is to disconnect by powering off during a print, the print will not continue.
- If the raspberry pi was to lose internet connection, the print will continue, and the status can be updated once a connection is reestablished.

- If the raspberry pi is still connected during a print, but the printer is shut off, the print will not continue if the printer is turned on again
- The current system is capable of printing 4 prints at once with a 100% success rate for all printers. The limitations for the current system include the budget and computational power of the raspberry pi.
- If an SD card is to start the print with the raspberry pi connected, the print will still work, however, OctoPrint will not be able to read all the values such as print time left. However, some data such as the print time left will be displayed on the printer's LCD.
- If the print cable were to be disconnected and then reconnected the printer will need to restart the entire print.

Maintenance of Product

- Check the filament section in Dashboard to ensure that the filament doesn't run out in the printer. This should be done once a week for every printer. When the spool is replaced, the staff member should go into OctoPrint to update the weight of the filament.
- Ensure that OctoPrint is updated
- A few times per day, make sure that the instance of OctoPrint doesn't disconnect by checking the Dashboard, the python code, and the web interface.

Figure 39: Client/User Testing Product

Figure 40: Sample of Our Test Results (Slic3r vs Cura)

5. Conclusions and Recommendations for Future Work

The results of our project have pleased our whole team. After 3 months, we were finally able to control 4 printers and display the status live and simultaneously on Dashboard. All our features added on Dashboard work how we wanted such as the filament left, time left etc. Also, the Ross Video members and the client were satisfied by our group's prototype. With our product, we were able to succeed and receive 1st Place in the STEM Automation category of Design Day Fall 2019 hosted by the University of Ottawa.

Thus far, we have gained a lot of experience in many ways throughout the design project. Moving forward we would like to test on other 3D printers in Makerspace. Meanwhile, we would let the customers and clients test out our product further to collect more data and feedback before finalizing our product. For future implementations, we plan on adding features such as cameras to monitor the prints and send email notification to people whose prints have finished. Monitoring live can also be done with OctoPrint.

Suggestions from some users include making a system for remote printing from anywhere on campus and from the user's computer with each user having their own username and password. However, after talking to CEED staff about this, they preferred that the users came into the space to print and that a lot of things can go wrong such as a buildup of print in queue.

Some lessons that were learned working throughout the term that could also be beneficial to future GNG 1103 students taking the course is to start the design as soon as possible. While the course topics are covered at a steady pace that can be followed, using the same timeline to finish the design project is not advised because there will be less time to finish the final design with a lot of pressure. Furthermore, the project design tasks should be split up amongst group members, so

no one has overlapping work done for a phase in the project. The delegation of tasks should be done as soon as possible as well to ensure a good pace and organization.

As for anyone who would like to enhance OctoPi, some advice is to familiarize yourself with ogscript allowing you to code in Dashboard. Not only that, knowing multiple languages e.g. python, JavaScript, etc. will also be beneficial when working on this project as the project is primarily software based. Finally, work with 3D printers and possibly be a user in Makerspace to get a feeling of what it's like to 3D print in a busy environment.

Overall, our group had lots of fun working with the 3D printers and if you are starting from scratch or continuing the project, you should have a good time as well.

6. Bibliography

Nematic!, director. *How to Assign Static IP Address to Raspberry Pi!* 6 Apr. 2019, www.youtube.com/watch?v=LCJtUbRlIXE.

Riley, Chris, director. *Multiple 3D Printers All At Once - Raspberry Pi - Octoprint - Chris's Basement*. 11 Nov. 2017, www.youtube.com/watch?v=7Saa1HpLRoM.

7. APPENDIX: Design Files

The documents to our project including the full Dashboard, python code, user manual, and final deliverables can be found in the link listed below. Here, users will be able to download the files and continue what we started.

Link: https://makerepo.com/syoga104/gng1103a19automated-3d-printing-with-OctoPi