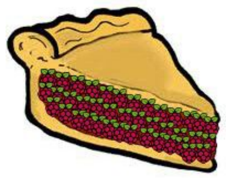
PiBakers

Group 7

"Fast, Cheap, 3D Printing"

12/5/14

Maxwell Miller
Ray Cload
William Corlett



Executive Summary

Additive manufacturing systems, more commonly known as 3D printing, are no longer just a tool in the hands of the aerospace, auto, and defense industries. 3D printing is rapidly making its way to desks inside the home or classroom. The reason for this growth is the push by smaller companies, start-ups, and hobbyists to get prices down and interest up (http://goo.gl/E2mekv) .

The Pi Bakers aim to take the spirit of this movement and lower the prices even more, while adding the power of the cloud to their printer. The ability to both reduce costs and add functionality comes from tapping another new piece of engineering hardware that has a similar cult-like following to 3D printing: the Raspberry Pi.

The Raspberry Pi’s cheap point of entry, $35, not only undercut the standard 3D Printing company’s custom build controller boards, but also have a robust networking capability. The Pi’s ability to seamlessly connect to the internet allow your printer to join “the internet of things” and suddenly such things as starting prints from your smart phone become possible.

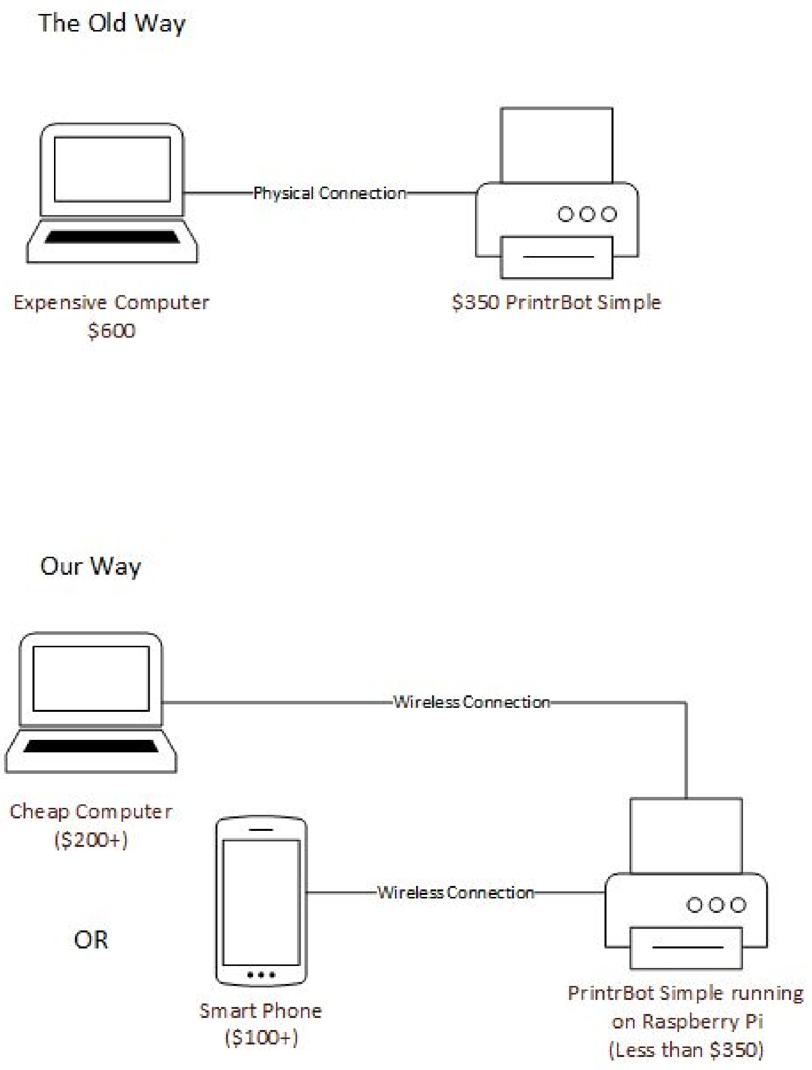
This project aims to take one of the most reasonably priced 3D printers on the market, the Printrbot Simple, replace it’s onboard controller with a Raspberry Pi, and create an open source software suite, on the Pi and online, to allow for Pi controlled prints directly from 3D models or pre-sliced code.

Our Motivation

A major part of the culture and spirit of the open source 3D printing movement is the development of low cost machines that can be put together and distributed to as many people as possible.

The aim of the Pi Bakers is to lower the initial cost of entry to the exciting world of 3D printing (See figure 1). This will be done in the spirit of the open source community via design and creation of a custom circuit board, along with tapping the computational power of the Raspberry Pi computer.

Figure 1: The Old Way vs. Our Way

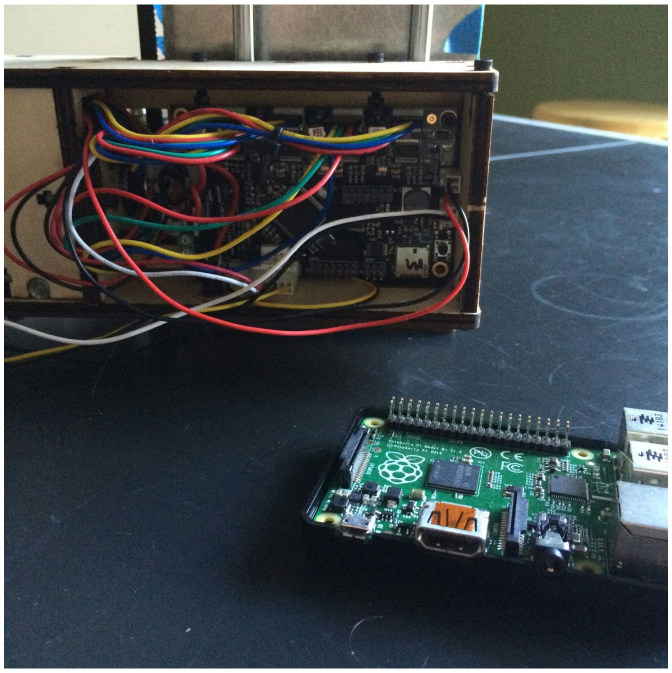


Our proposed cost savings comes from multiple sources. Cheaper 3D printers, like the Printrbot Simple, rely on the horsepower of a moderately powerful computer. With our solution the user can use a significantly less powerful PC, or even a smartphone to interface with their printer. These hardware intensive calculations of slicing potential 3D models can be done in the cloud.

Today’s 3D printing market is dominated by devices controlled either by variations on the Arduino Mega microcontroller with an interface known as RAMPS (RepRap Arduino Mega Pololu Shield, see figure 3) that costs around $70. These come with a decently sloped learning curve to assemble and maintain. A custom build controller board made by 3D printing companies for their particular devices, like Printrbot’s Printrboard (see figure 4), cost around $120. Even though these custom boards are significantly easier to use, they still need to be attached directly to a computer via USB or Bluetooth.

By replacing with printer controller boards (PCBs) with a Raspberry Pi one can remove both the computer and RAMPS board, cheapening the entire process to begin a print (see figure 2). The Raspberry Pi, itself a small, cheap ARM based computer, allows for more advanced options not available to the simplistic microcontrollers shipped by 3D printing companies. Most exciting is LAN/WLAN networking to allow for connection to the cloud.

Figure 2: The PrintrBoard and Pi



Costs can be lowered and functionality increased by swapping the PrintrBoard (back) for a Raspberry Pi (foreground).

Figure 3: RAMPS Shield

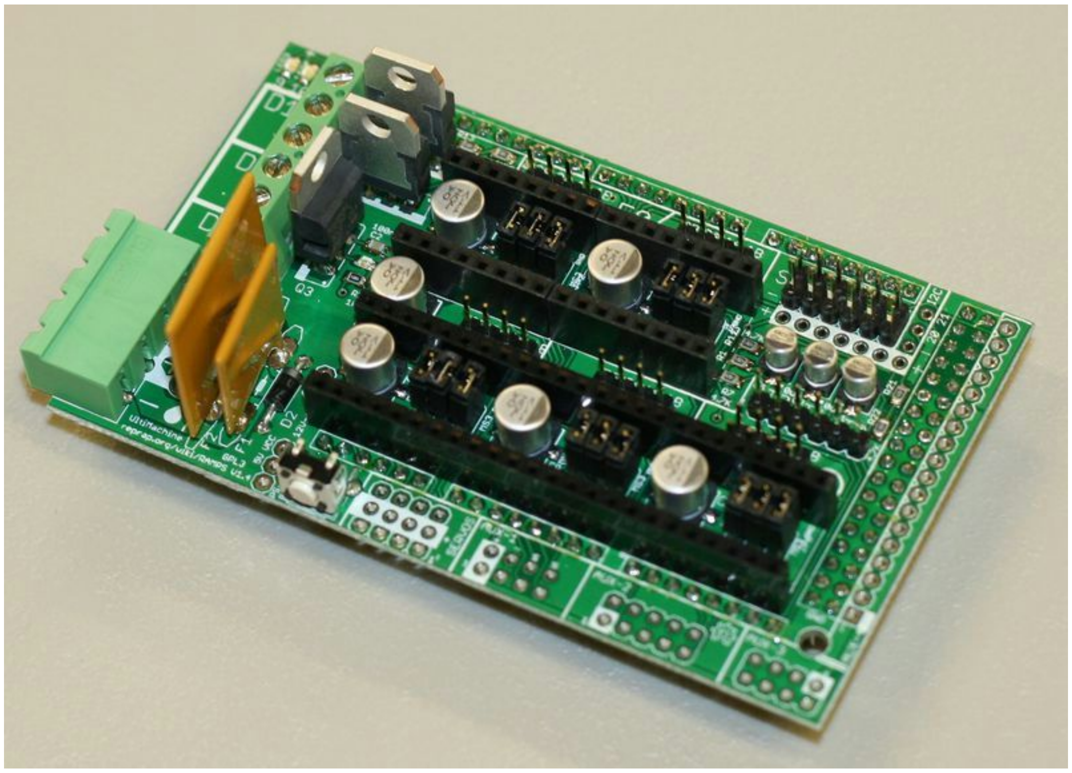
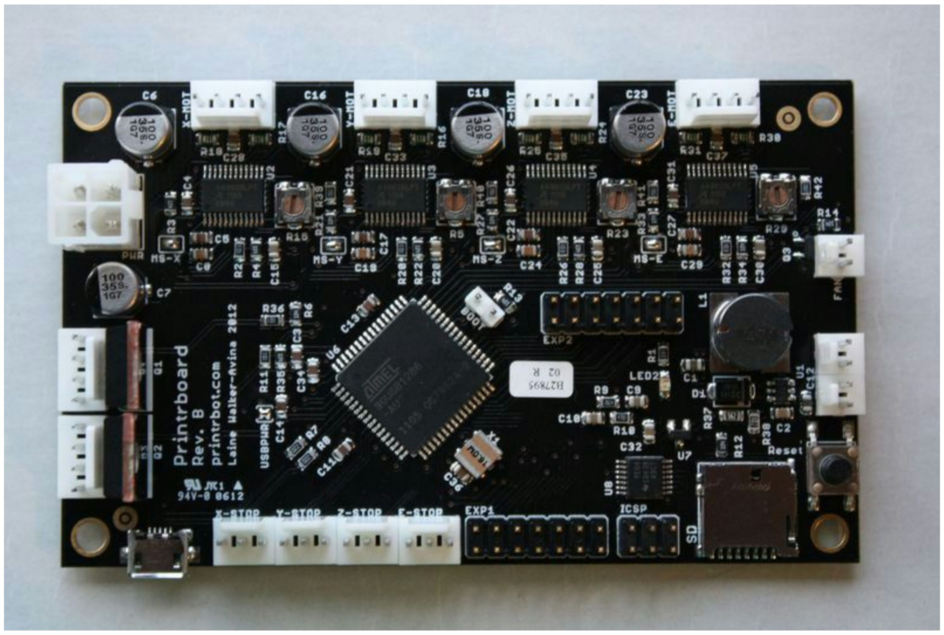


Figure 4: Printrboard 3D printer controller by Printrbot.



Requirements

The Raspberry Pi shall completely replace the control electronics onboard the 3D printer

Our Raspberry Pi will be capable of performing all of the duties the control electronics fulfil in order to successfully print a 3D model.

The Raspberry Pi shall be able to store and print complete 3D models without having a computer connected

The user should be able to print a 3D model without a physically connected computer. Once a print job has started, no connection should be needed (the instructions to print should be cached locally on the Raspberry Pi)

The user shall have the ability to upload a model to a remote server for slicing and transmission to the Raspberry Pi

A remote server will be set up to accept 3D models in the .stl, .obj, or .amf formats and perform all required steps necessary to have the model be printed on the 3D printer.

The user shall be able to begin a print by interfacing directly with the Raspberry Pi or by interfacing with an intermediate server.

If the user has set up an intermediate server to interface with, they should be able to connect either to that intermediate server, or to the Raspberry Pi directly in order to print a 3D model.

The intermediate server shall be able to slice 3D models and transmit them to the Raspberry Pi

The intermediate server should take a 3D model and produce a G-code output which will be transmitted to the Raspberry Pi.

Both the intermediate server and the Raspberry Pi shall have a functional web interface for uploading files

A web interface should be hosted on both the Raspberry Pi and the intermediate web interface to provide a consistent and easy means of printing no matter the device the user decides to connect to.

The intermediate server shall allow printing to multiple, user configurable, printers

The user should be able to add printers to the intermediate server through the web interface and easily direct models to be printed on specific printers (which must be running our software).

Specifications

The Raspberry Pi shall be able to replace the control electronics onboard the 3D printer entirely

A circuit board will be designed which will plug into the GPIO pins on the Raspberry Pi to interface with the following:

1. The stepper drivers on a cartesian style (one motor for each axis and one for the extruder) 3D printer at at least 30mm / second (this is the default speed of our test printer)

2. Read input from the temperature sensors on the 3D printer (thermistor) to allow for the temperature regulation of the hot end.

3. Read 3 endstops (one for each axis) that will allow the printer to automatically find the home (0,0,0) point (this is a must have for cartesian 3D printers).

4. Provide power via standard ATX PC power supply to power all of printer components and a USB port to power the Raspberry Pi

The Raspberry Pi shall be able to store and print 3D models without having a computer connected after the initial slicing and data transmission.

The Raspberry Pi must be able to:

1. Store the entire sliced model in the form of G-code on the Raspberry Pi.

2. Create an interpreter to convert G-code into GPIO signals which will communicate with the 3D printer.

The user shall be able to wirelessly transmit a 3D model to the Raspberry Pi

The following transmission mechanisms will be implemented:

1. Directly to the printer as a 3D model, where the Raspberry Pi will transmit the model to a remote server for slicing, and receive back G-code to use to start the print job (due to the low processing power of the Pi, slicing the model on the Pi is somewhat impractical).

2. Directly to the Raspberry Pi as G-code, where the Raspberry Pi will start the print job directly.

3. Directly to the remote server, which will slice the model and pass it on to the Raspberry Pi

The ability to upload a model to a remote server for slicing and transmission to the Raspberry Pi

A virtual machine will be used to store the 3D model and perform the slicing of the model. The virtual machine will transfer the sliced model to the Raspberry Pi using Node.js.

Stretch Goals

The user shall be able to pause a print job

The web interface should provide functionality to allow the user to see that a print job is running, and allow them to pause the printing process and resume the process at the same point later.

The user shall be able to manually control the printer

The user should be able to perform predefined operations on the printer from the web interface, including:

1. Move the heated nozzle away from the print (to check for defects or change the filament).

2. Perform a calibration check of the printer.

Modify software to interface with Delta style 3D printers

Popular Delta-style 3D Printers, such as the Rostock, should be supported by the interpreter, giving much more flexibility to which printers our project is capable of working with.

Implement automatic bed leveling with inductive proximity sensors and force sensitive resistors

Automated bed leveling should be implemented for printers which include this functionality. This bed leveling should be done before any print job begins, in case some environment change occurred since the last time the printer printed. This is done by replacing one of the endstops, specifically the one for the Z axis, with some other kind of sensor, that will give you an approximation of the hotends proximity to the print surface. Inductive proximity sensors do this by mounting a probe near the hotend that can detect a metal surface, then probing multiple points on the bed and interpret that data as a Z offset to use while printing. Force sensitive resistors are normally mounted under the build plate at various different positions. When the hot end presses down on the build plate, it creates pressure on the force sensitive resistor which causes its resistance to go up, when the resistance gets high enough to trigger the binary endstop detector it

Add multiple extruder support

Add support for a second hot end and extruder assembly. This would allow the use of different materials in a single print without requiring user intervention. This includes different colored plastics as well as different kinds of materials.

Implement any other G-code operations that were previously omitted

Implement any G-code instructions which were deemed unnecessary at first to make the platform more robust and capable of handling the widest range of use cases.

Allow for Real-Time Monitoring of the Printer's Status

If the printer is currently printing, add real-time (constantly updated) information of information such as the extrusion and bed temperatures, head location, and whatever else may aid the user in determining if their print job is being successfully carried out.

Research and Investigations

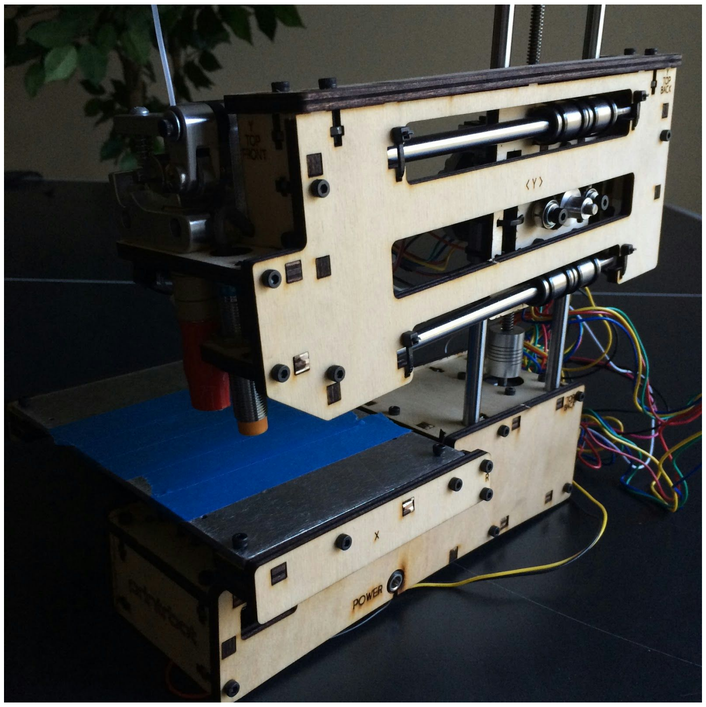
The Printrbot Simple

Initially funded via a Kickstarter campaign, the Printrbot aimed to be “a 3D Printer Kit anyone can build”. The Printrbot Simple (see figure 5), unassembled, can be purchased at the extremely competitive price of $350 (http://printrbot.com/). The Printrbot is interfaced via USB, and while has a custom board, does not come with it’s own proprietary software but relies on open source software.

The Printrbot is part of the RepRap Project. A movement in the 3D printing community that aims to produce free and open source hardware designs whose full spec is released under the GNU General Public License (http://reprap.org/wiki/Main_Page).

The series of Printrbots ranges from their top of the line Printrbot Metal for $999, down to their wooden, unassembled Printrbot Simple (http://printrbot.com/). The Simple is what was selected for this project for both it’s open source nature and low cost of entry.

Figure 5: The Printrbot Simple Makers Edition 1405



Above is the Pi Baker's second fully constructed Printrbot Simple

The Raspberry Pi B+

The small "credit-card" sized computer called the Raspberry Pi was created by the UK based charity *Raspberry Pi Foundation*. Citing the rise of "the situation where computers had become so expensive and arcane that programming experimentation on them had to be forbidden by parents", the Foundation wished to create a cheap platform "like those old home computers" that allowed learning from even the command line (http://www.raspberrypi.org/about/).

Not only does the open source nature of the Pi fit nicely with the spirit of our project, more importantly does so the price. Some models of the Raspberry Pi can be purchased for as low as $20, though for our purposes the "high end" $35 model, the B+ (see figure 6 below), was the most compelling. Not only did the B+ introduce 40 GPIO pins for general usage, but also the comparatively muscular 512MB of RAM versus the 256MB of the older, cheaper models.

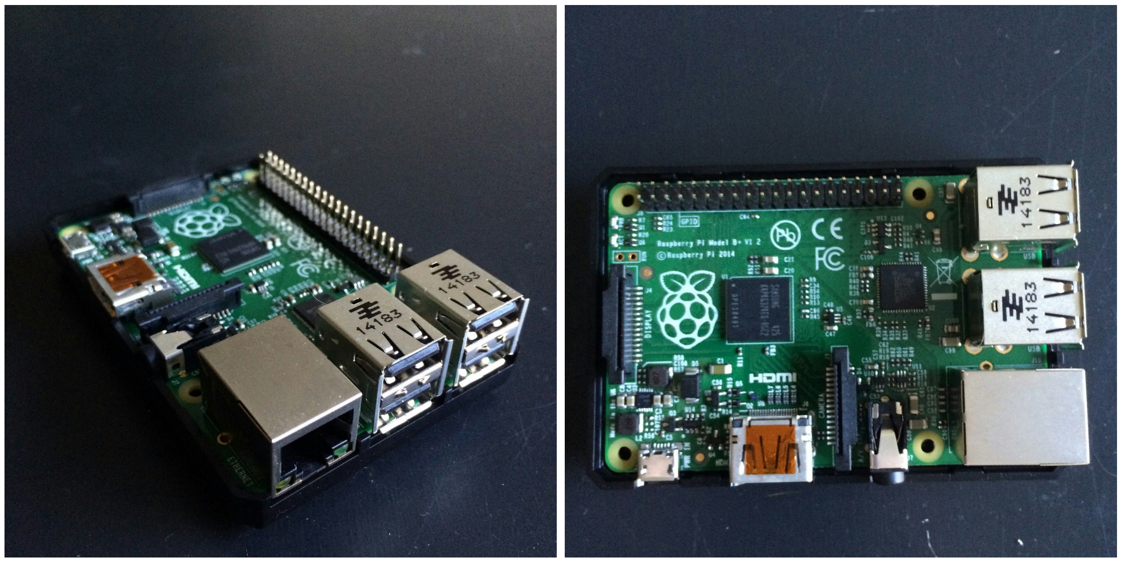
During the development of our initial prototypes the Foundation released a new revision of the Raspberry Pi hardware, the A+. While managing to lower their cost again down the original Pi's $20, as well as retain the same number of GPIO pins as the B+ model, the memory was again slashed to 256MB. Furthermore, the A+ had only a singular USB port compared to the B+'s four, and no ethernet port. Considering the fact that our system requires network access, this would require the only input device to be a wireless adapter. A major goal of this project is to be able to run your print off the Pi, so running your Pi "headless" by using secure shell (SSH) into your singular USB port A+, we want the user to be able to have more options directly interfacing with their Pi.

While bring down the price-point of entry into 3D printing is a major goal of this project, the extra $15 dollars to double the amount of onboard RAM and quadruple the number of USB ports was deemed worth the cost. See Table 1 below for a breakdown in the various models of the Raspberry Pi.

Table 1: Model Comparisons

	RAM	GPIO Pins	Price
Model A (2012)	256MB	26	$20
Model A+ (Nov 2014)	256MB	40	$20
Model B (2012)	512MB	26	$35
Model B+ (July 2014)	512MB	40	$35

Figure 6: The Raspberry Pi Model B+



The Model B+ with 40 GPIO Pins and 4 USB Ports

Despite the Model B+'s generous USB ports and RAM, its 700MHz ARM processor fails to be speedly enough for what we consider to be our objective of "fast" and cheap 3D printing. Instead we will be "off-shoring" the the more muscular computation of slicing the 3D model to the cloud.

Raspbian

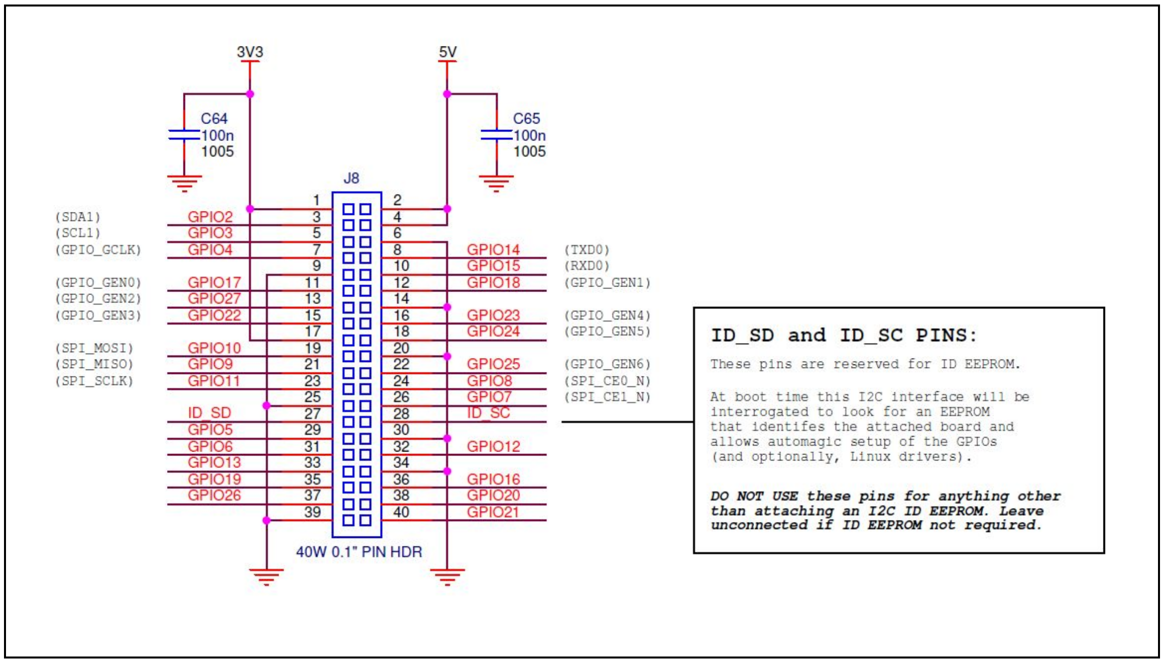
The Raspberry Pi is powered by an ARM processor and therefore primarily uses Linux kernels. The Raspberry Pi Foundation provides an installation manager, NOOBS (New Out Of the Box Software), that provide a few flavors of Linux distributions and media centers. We have chosen to go with **Raspbian**, a debian based kernal maintained and recommended for use by the Foundation (http://www.raspbian.org/). Boasting that it comes 35,000 packages and software bundles pre-compiled upon installation, Raspbian provides both ready “out of the box” networking tools and packages to access to the Raspberry Pi’s GPIO pins to allow for smooth development.

GPIO Pins

The Raspberry Pi’s General Purpose Input/Output (GPIO) pins allow the small computer to reach out and interface with a legion of other electronics (http://www.raspberrypi.org/documentation/usage/gpio/). Each available pin can be used as a switch to do simple tricks like turning on LEDs to more advanced exercises, such as signaling a stepper driver to a stepper motor. It is not just because of the low price of the Pi that it was selected for this project, but it’s GPIO pins to allow for physical computing.

The Raspberry Pi B+ has the expanded pin set as seen below in Figure 7.

Figure 7: Mode B+ Expanded GPIO pins



ID_SD and ID_SC PINS:

These pins are reserved for ID EEPROM.

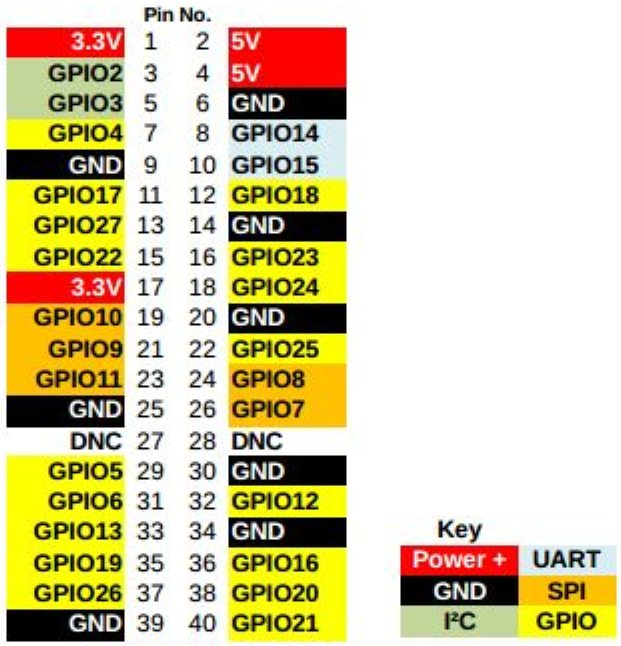
At boot time this I2C interface will be interrogated to look for an EEPROM that identifes the attached board and allows automagic setup of the GPIOs (and optionally, Linux drivers).

DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.

Released by Creative Commons license by The Raspberry Pi Foundation

Due to the number of stepper drivers the pins will be controlling, the expanded pin set of the B+ was needed. This is not because one needs 40 pins, but one needs the smaller amount of “free” general purpose pins that are not involved in power, ground, or serial ports and interfaces. It is preferred to use these “general purpose” pins before accessing serial pins. Ground and power are, of course, off limits. See figure 8 below for individual pin usage in the model B+.

Figure 8: B+ GPIO Legend



	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Key

Power +	UART
GND	SPI
I^{2}C	GPIO

Included with permission from raspi.tv/rpi-gpio

G-Code

Initially developed at the Massachusetts Institute of Technology as a tool to aid in “three-axis continuous path control”, G-code is commonly still used today in machine manufacturing (http://museum.mit.edu/150/86). G-code’s main functionality focuses on informing machine tools of a desired x,y,z toolpath to achieve the machine’s particular goal. G-code has numerous commands due to its wide usage from machines that cut or finish materials to newer “additive” machines like 3D printers.

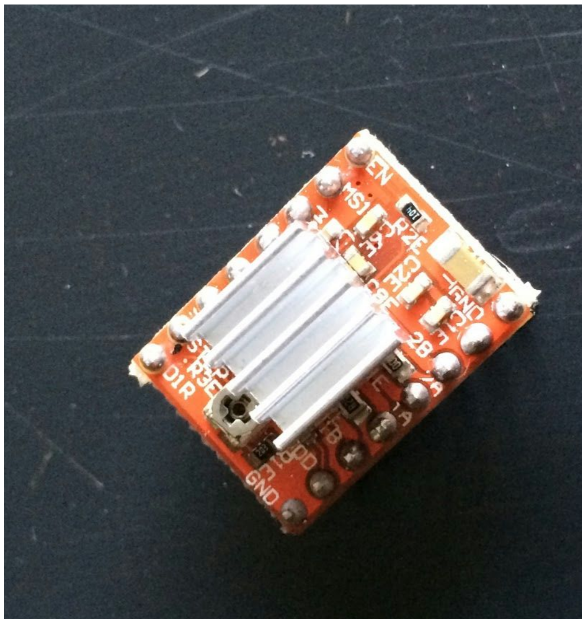
The particular open source version G-code this project will focus on is “Marlin”. Marlin is the variant of G-Code commands that popular open source software like Slic3r generate from 3D models. Like the Printrbot hardware itself, Marlin is a RepRap open source version of G-code, and releases itself under the GNU GPL. Marlin boasts a feature set that, unlike some of its competitors, can handle printing arcs “with full velocity” and folder support for nested files (http://reprap.org/wiki/Marlin).

But even a full one-to-one functionality of Marlin is not needed for achieving cheap 3D printing with the Printrbot Simple. The Simple’s lack of advanced features such as a heated bed multiple extruders make a portion of Marlin out of scope for the base requirements of this project (See Appendix G-Code Commands).

Stepper Drivers

While powerful, the GPIO pins on the Raspberry Pi are not high enough voltage to power a printer’s stepper motors directly. Instead the pins will communicate to a A4988 stepper driver (see figure 9) connected to another power supply to drive the motors. These tend to be priced from $4 to $10 depending on the vender -- again aiming to keep costs low.

Figure 9: The A4988 Stepper Driver



Similar to wiring up the GPIO to the A4988 is a similar setup shown in figure 10 below, but with an Arduino, to move a stepper motor. The Pi Baker configuration will be wired up with four stepper drivers.

Figure 10: An A4988 wired to a Nema 17 Stepper Motor

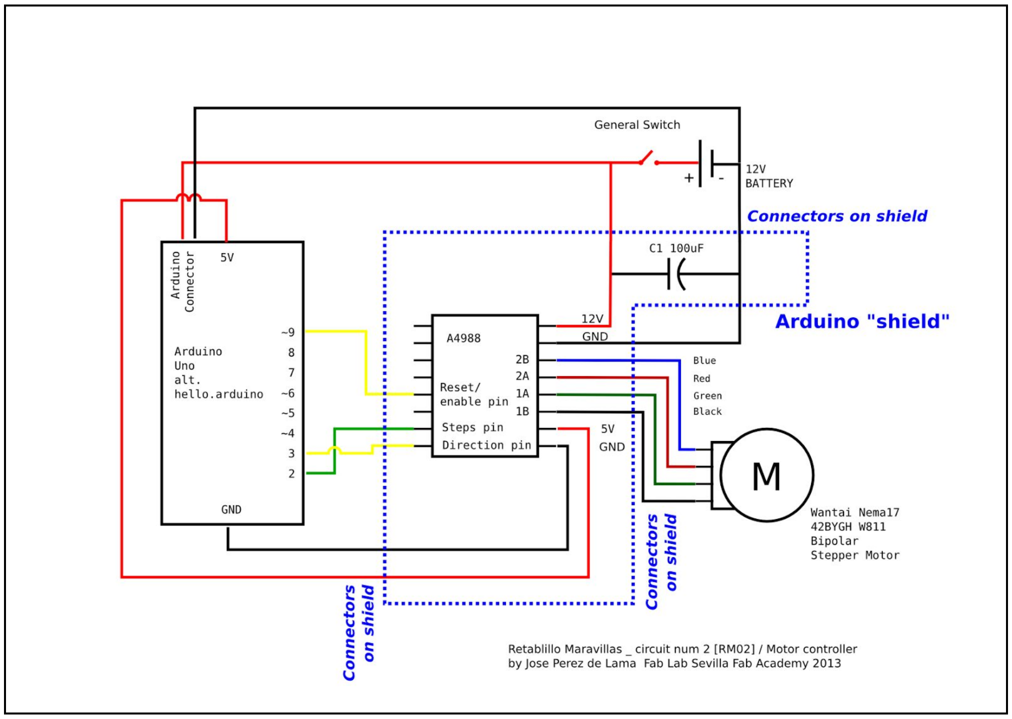


Figure used with permission via the Creative Commons Attribution-Share License (http://academy.cba.mit.edu/2013/students/perezdelama.jose/final_project_04_electronics_jpl.html)

Slic3r

Slic3r is one of the leading open source G-code generators for 3D printers. Not only does its open nature fit nicely with ethos of this project, but Slic3r is also fast, claiming to be up to 100x faster than other open solutions like Skienforge (http://slic3r.org/).

The idea of “slicing” a 3D object is the logical foundation for 3D printing to work. Slicing programs, like Slic3r, cut the fed model into horizontal slices, generates toolpaths, and estimates the necessary amount of “additive material” to be extruded.

Node.js

The question of what language to write the server in was another significant step in solidifying our project plan. At this point in time, so many pre-built systems exist for managing a web server, there was no question that it would be built on one of these pre-built platforms, but with so many options, the choice of which one can be difficult.

PHP

PHP is probably the “classic” way of writing a web server. You create php files which, when accessed by a remote user, perform some action on the server and return a valid .html file the web browser can parse. This could have worked for our system, but in order to use PHP you need to run a web server which is capable of parsing and running PHP files, like a apache or nginx. This would have added complexity and overhead that our project did not need, detracting from PHP as a viable candidate to base our server on.

Furthermore, in the open source community these days PHP is looked down upon for design decisions the creator made in the early days which lead to many “gotcha” moments while programming in it. Several newcomers have made great headway in replacing PHP in startups and open source development.

Go

Go is a programming language developed by Google and announced in 2009. In it’s short life it has come to dominate Google’s in house codebase, especially for web server programming.

The Go standard library includes a number of networking packages which facilitate easy web server programming, including opening ports and for listening on a device and assigning handler functions for those ports. Go is also an extremely fast language, being very close to C in it’s design (Ken Thompson, one the of developers of C, played a prominent role in the design of Go), fulfilling that desire of ours.

Really, Go has everything we needed and wanted as far a featureset goes, and the initial prototype of the system was written in Go, but we believe that it might still be a bit out of the norm, and that its lack of a package manager or community involvement hampered its popularity and made another option seem far more likely to help our project succeed.

Node.js

Node.js also dates back to 2009, but seems to have taken off as the most popular platform for developing modern web apps and APIs. What makes Node.js unique is that it is not a new language which attempted to “do networking better.” It is a platform in which you write Javascript which runs on Chrome’s V8 Javascript engine and interacts with the node core which is written in native code.

Node.js’s biggest asset though, and what Go did not have, is a package manager which allows developers to provide useful functionality, and which facilitates users to easily download these packages.

To install, for instance, the express web framework that we used in our project, the user need only type:

```
npm install express
```

and the package manager will identify which version of express to install based on your project’s package file, which describes which versions of node.js you will be supporting. and it will download the necessary files into your project’s directory. Once installed into your project’s directory that package can be used by that project anywhere it exists (each user does not have to download the packages on their machine, they are distributed with the project), making it extremely simple to use for the user.

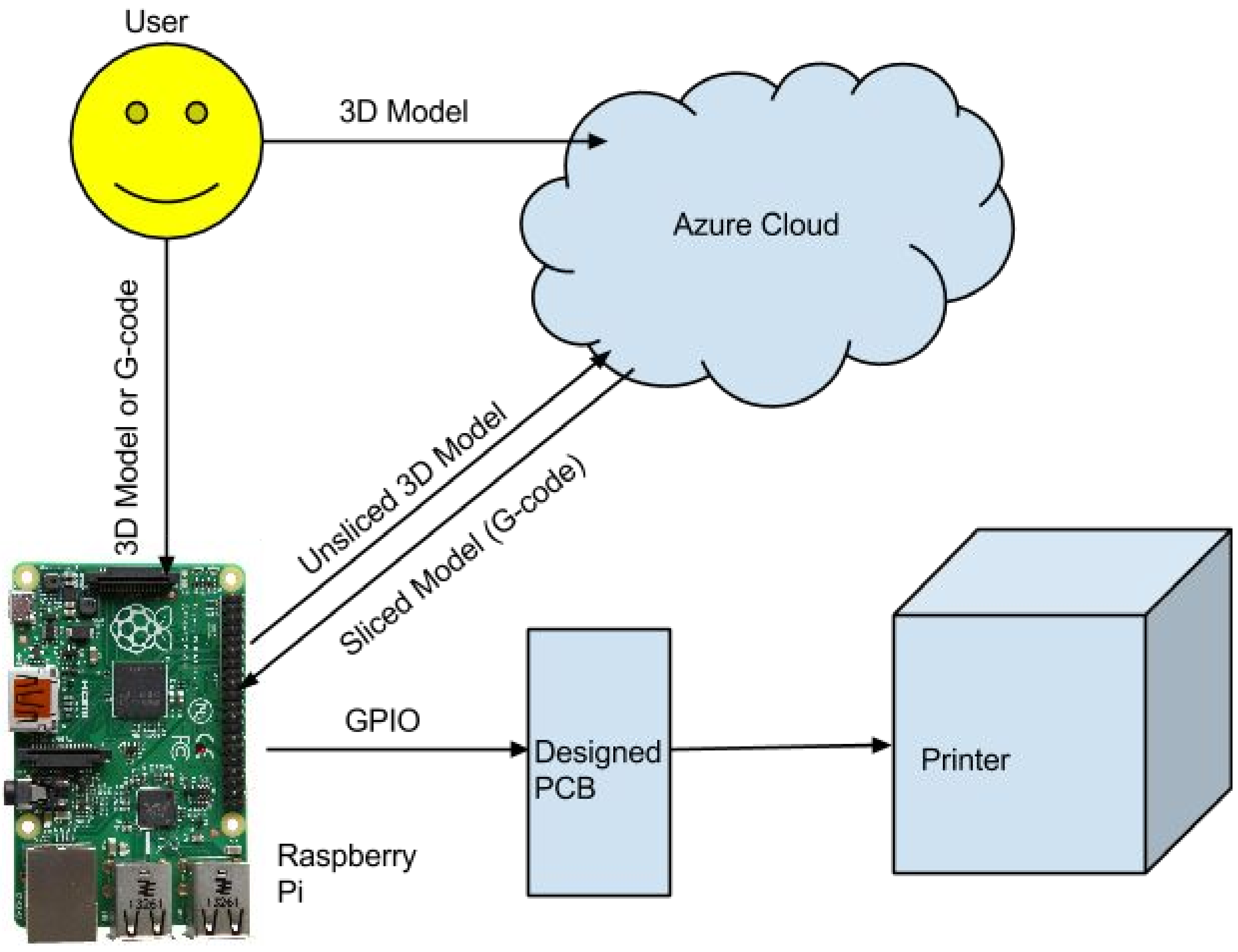
Node.js comes with a very limited standard library to keep it lightweight and allow it to support the greatest range of different applications. Community packages more than made up for this lack of pre-built functionality, and reimplementing the server in Node.js with all of the functionality of the Go server was not too painful. The extensive package library should make development going forward much easier though.

Design Summary

High Level Design

At the highest level, the intent of this project is to remove the necessity of having both a microcontroller on the 3D printer and a computer connected to it, and instead wrap a small computer into the printer itself. This allows 3D printers to come closer to a stand alone appliance, something that could be anywhere in your house and not just wherever you happen to keep your PC. In the process we are also eliminating some of the cost of the electronics normally associated with a 3D printer, instead replacing them with a simple PCB built to bridge the gap between the Raspberry Pi and the machinery of the 3D printer.

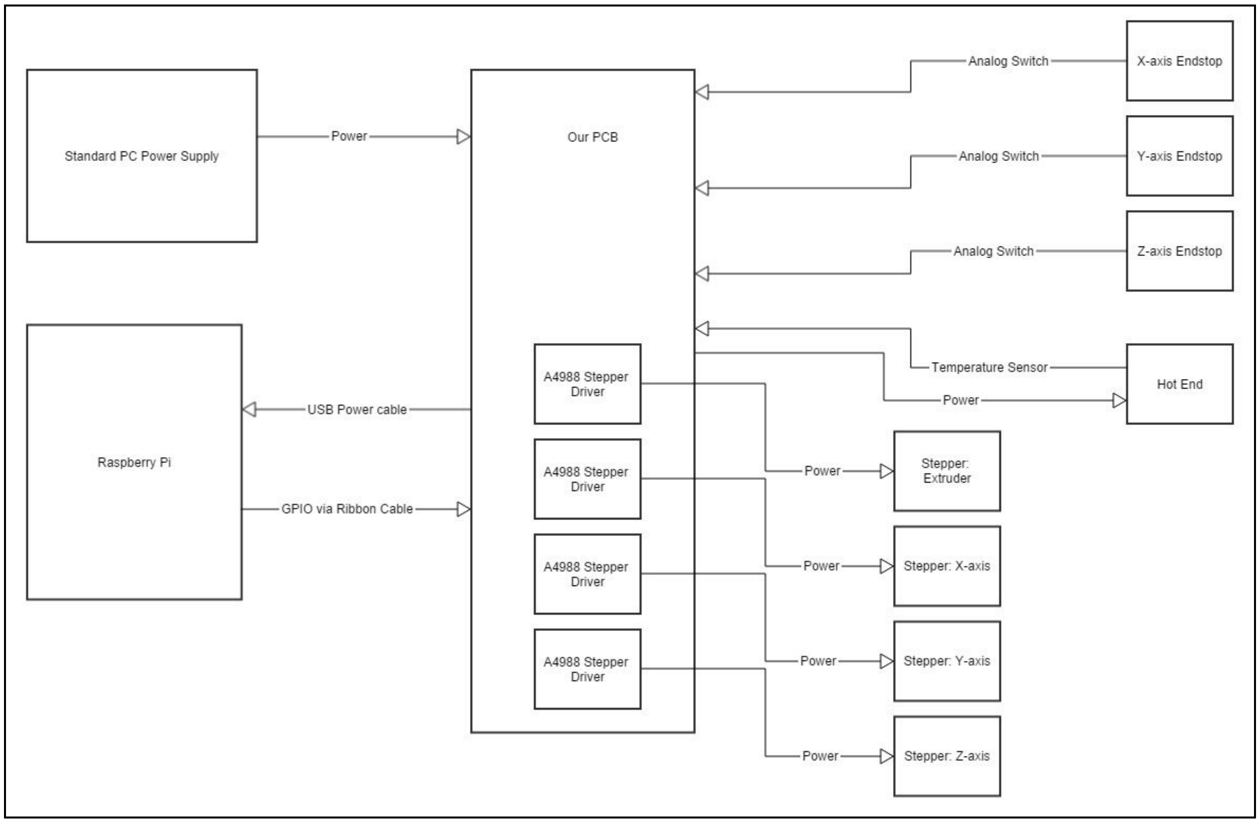
Figure 11: The High Level Design



Hardware Block Diagram

Below in figure X is our hardware design block laying out the Pi Baker's basic plans for the Raspberry Pi, our custom printer controller board, and its connection to power, stepper drivers, motors, and endstops.

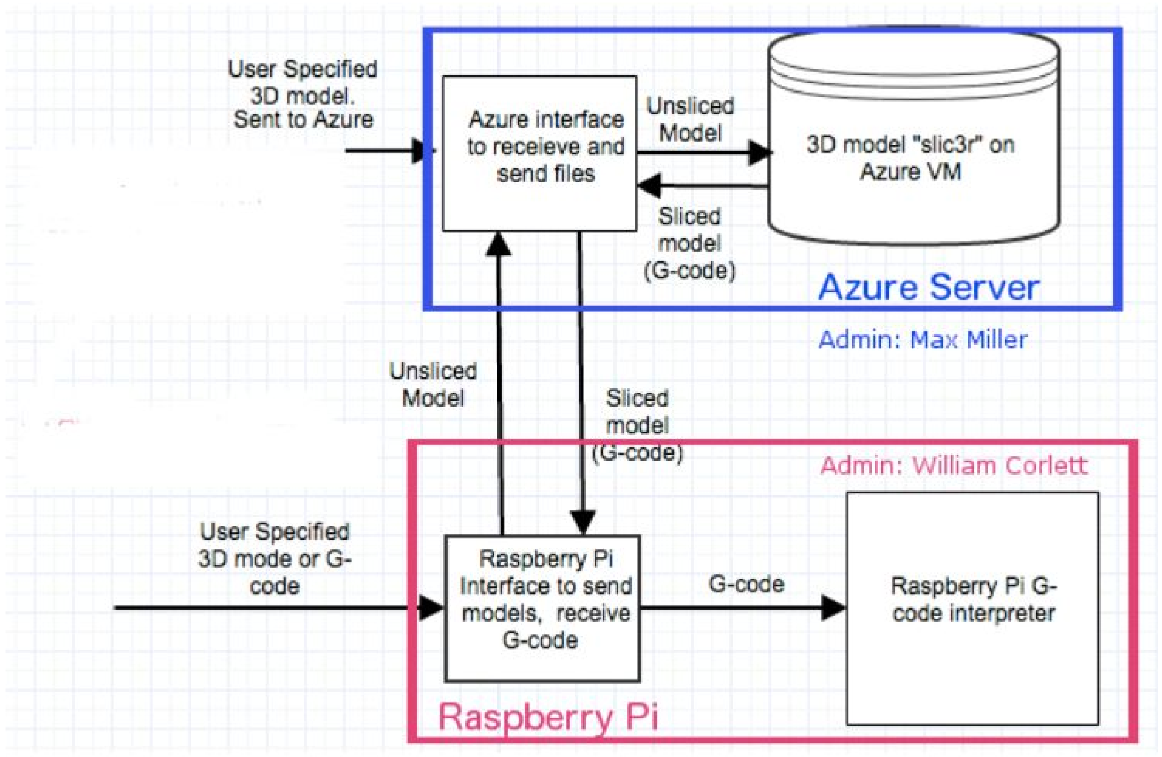
Figure X: Hardware Design Diagram



Software Diagram

Below in figure X is the Pi Baker's software diagram, displaying the connections of the open source networking and local software connections between the Raspberry PI, the Azure server, and then back to the Pi.

Figure X: Software Design Diagram


User Specified 3D model. Sent to Azure
Azure interface to receieve and send files
Unsliced Model
3D model "slic3r" on Azure VM
Sliced model (G-code)
Azure Server
Admin: Max Miller
Unsliced Model
Sliced model (G-code)
Admin: William Corlett
User Specified 3D mode or G-code
Raspberry Pi Interface to send models, receive G-code
G-code
Raspberry Pi G-code interpreter
Raspberry Pi

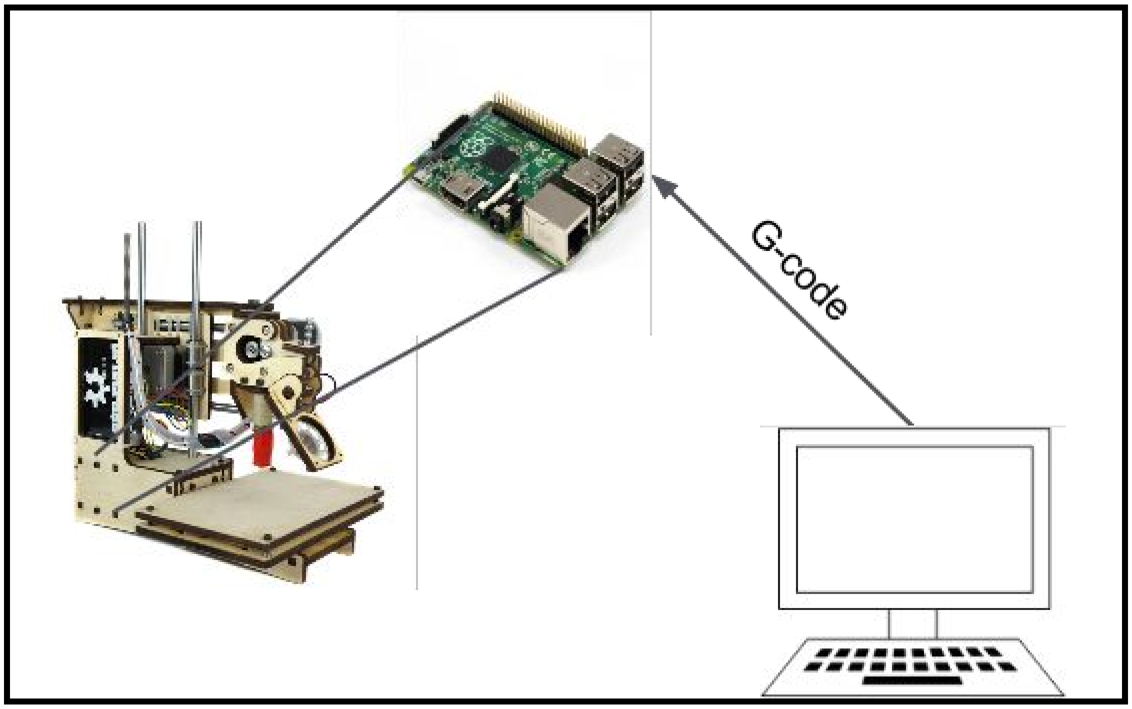
Printing

3D models should be able to be printed either locally or remotely. All printing should be done wirelessly through a web interface to remove the need to install drivers and to allow the user to print from wherever they are.

Printing Locally

Printing locally would be the case where the user has a pre-sliced model, G-code, already in their possession and ready to print (see figure 12).

Figure 12: The Local Print



Printrbot image provided by 32b.it and Raspberry Pi image provided by the Raspberry Pi foundation

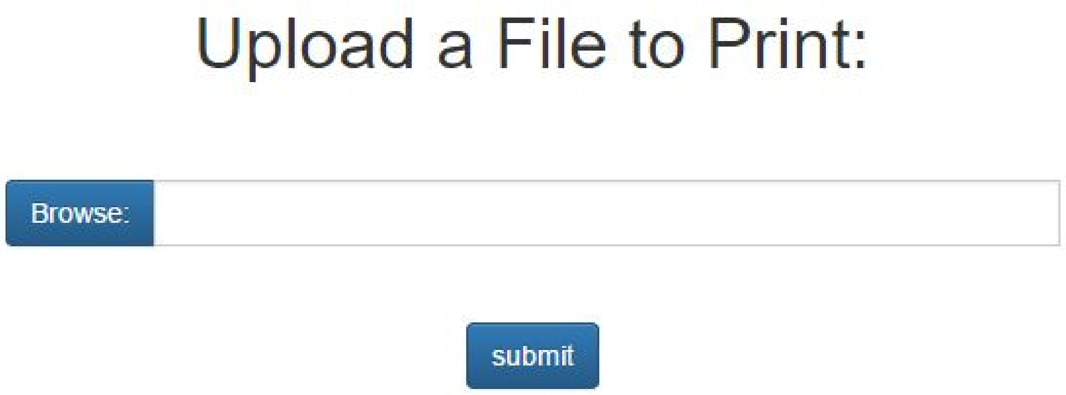
Summary:

The user will slice the 3D model themselves, using the software they choose, to create a set of instructions (G-code) for the 3D printer to follow. The user will send that file to the printer over the local network, and the printer will begin printing.

This will be the primary method of printing as it requires no configuration on the part of the Raspberry Pi, and you do not need any additional hardware.

Optionally, the user can add an intermediate server to use when a 3D model (non-sliced) is uploaded to the Raspberry Pi. If such a server is supplied, the software will allow model uploads, have the other server slice them, and then print as usual.

The Interface:



Browse Button:
Opens a system dialog which allows the user to browse for a file to upload.

Submit Button:
Initiates the upload process, which will kick off the rest of the print job.

Steps taken to configure the printer for local wireless printing:

1. ssh into the printer using the default login credentials
2. Download the printer software
3. Run the server using the command “nodejs app.js”
4. Find your printer’s local IP address by running “ip addr” and looking for the “inet” field under “wlan0”. This is the IP address you will connect to through your web browser to print.

Optionally, if the user wants to use another server for slicing:

5. edit the file “config” in the “data” directory to include the line

```
use-slicing-server *ip address or url of slicing server*
```

Steps taken to print locally:

1. Slice your 3D model using slicing software such as Slic3r (the recommended slicer)

2. Connect to your server through a web browser at the address *your ip*:8080/pi, where *your ip* is the IP address you found earlier. It will look like 192.168.1.2:8080/pi.

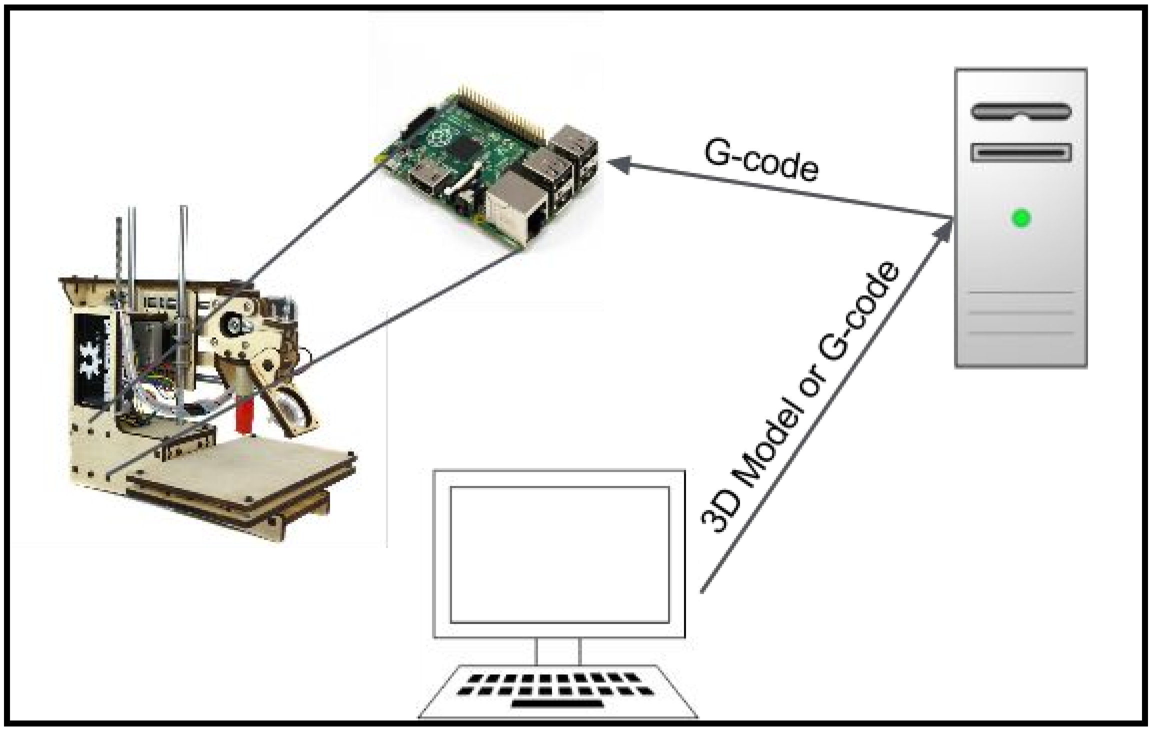
3. Select your G-code (or model, if a slicing server is supplied) file and submit to upload it.

4. The printer will begin printing immediately if a gcode file is supplied, or when slicing has completed, if it is a 3D model.

Printing Remotely

The case where the user has a 3D model they wish to print, but first must be sliced (see figure 13).

Figure 13: The Unsliced Model to Print



Printrbot image provided by 32b.it and Raspberry Pi image provided by the Raspberry Pi foundation

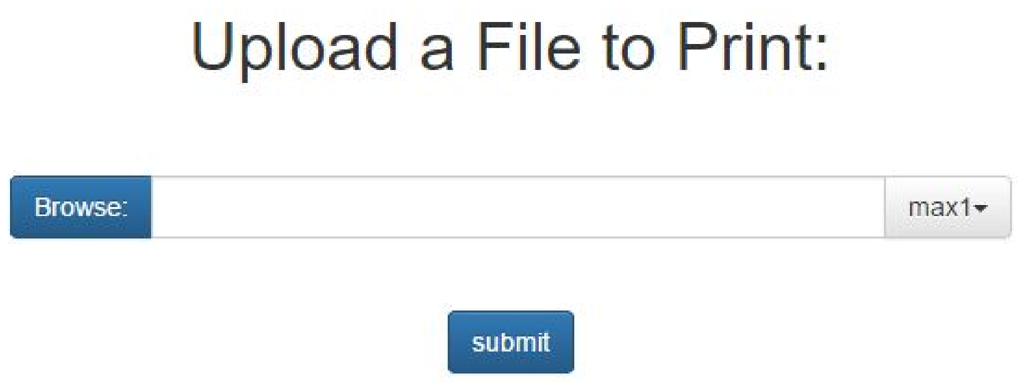
Summary:

The user will be able to set up an intermediary server which will allow printing of non-sliced 3D models of several standard formats (usually .stl, but also .obj and .amf files) as well as allowing printing to several different printers through one unified interface.

G-code files can be uploaded along with non-sliced 3D models to allow users to slice the model themselves if the settings the server provides are not sufficient.

Note: This requires an external linux server, not included with the printer.

The Interface:



Browse Button:

Opens a system dialog which allows the user to browse for a file to upload.

Dropdown List:

Contains three types of list elements:

1. A Printer which has been registered with the service

2. The “Add Printer” element, which opens a dialog with “Name” and “IP Address” fields

3. The “Remove Printer” element, which opens a dialog with a list of currently registered printers, and the ability to select and delete them.

The first printer in the list will be shown in the label of the dropdown menu at first, with the currently selected printer replacing it if the user selects a different printer.

If no printers are registered with the service, “Add Printer” will be shown as the dropdown label.

Submit Button:

Initiates the upload process, which will send the G-code to the printer and kick off the rest of the print job.

Steps taken to configure printing remotely:

1. Follow the above steps to configure the server on the printer for local printing.
2. If your server is not on your local network, configure your router to forward the port you want to connect to (usually 8080) to your printer, located at the internal ip address you discovered earlier.
3. SSH into the server using your credentials.
4. Download the printer software onto the server.
5. Change the line “server no” to “server yes” in config file located in the data folder.
6. Visit the web interface on the server
   a. If the server is on the local network, this can be found the exact same way as the printer.
   b. If the server is publicly accessible, you can find your public IP address by running “curl ident.me” on the server.
7. Visit the web interface at your *server ip*/pi
8. Click the dropdown and “Add a Printer”
9. Add a name, and the IP address of your printer, and port number. The IP address will look like 192.168.1.2:8080 if your server is on your local network, the numbers will be significantly different if it is a remote server.

Steps taken to print remotely:

1. Navigate to the server web interface via a web browser (the same way as in the configuration instructions)
2. Select a file to print, and choose a printer from the dropdown menu. If the printer you want to print to not available, select “Add a Printer”.
3. Use the configuration options provided to customize the way the slicing software operates, if a 3D model is chosen.
4. Click submit to send the 3D model to the server, the selected printer will begin printing when the server finishes slicing it and sending it to the printer.

Software Used:

Node.js

Node.js was used to write the web server used to to upload files and manage the server-side tasks which need to be performed on uploaded files.

Node.js is a platform which allows the building of network applications in javascript. Node.js was chosen because of the large repository of pre-built packages that are available for it which provide enormous power, including the express web framework, which we have made extensive use of.

The express web framework provides an api which allows for the monitoring of certain ports on the host machine, and registering of handler functions for requests made to those ports, and specifically to certain directory accesses on those ports.

For example, a handler function could be made which will be called when somebody accesses http://www.mywebsite.com/pi and a different handler function for when http://www.mywebsite.com/otherpi is called. These handler functions are given the full http request, and can do anything they want with it. Express also allows the developer to differentiate between different kinds of requests (GET, POST, PUT, DELETE) and handle them differently.

These features make express with node.js an ideal platform to run on top of, as they provide all of the necessary functionality to do what we need them to do, while not requiring something as heavy as a fully-featured web server running all the time.

Other node.js packages used include:

Busboy - Used to parse form data being submitted. In our case, the upload form and configuration options.

File System - Used to interact with the filesystem. We used this to write uploaded files to disk and to clean up old files to avoid clutter in the system.

Child Process - Used to create and manage child processes. This was used to invoke slic3r on the server, and the interpreter on the printer.

Bootstrap

Bootstrap is a cross-platform HTML, CSS, and Javascript platform which is used for creating interfaces which are consistent across browsers and devices, allowing the web developer to only develop a UI once and have it be automatically compatible with the browsers almost any user could want to use.

None of us are web designers, so, even if developing the site requires a bit more work (to structure it in a way bootstrap understands), the time-savings and UI benefit we get more than make up for it.

Slic3r

Slic3r is a cross platform G-code generator that fully supports the g-code standard the Printrbot Simple, and by extension our project, will support. Slic3r provides extensive customization options and is actively developed and supported.

We chose to use Slic3r as our G-code generator on the server because it is free and open source, and allows for running / configuration through the command line and saved configuration files, so it can be invoked in an automated way.

Hardware Used

Wifi Dongle

The printer must be accessible on the local network at least for the wireless interface to be accessible. These are available for less than $10, and are extremely small, making them very convenient.

The alternative would be connecting the Raspberry Pi to a router with an ethernet cord. The Model B+ has an ethernet port, meaning the only cost here would be an inexpensive ethernet cord. The downside, though, would be that the 3D printer would have to be located within close proximity to the wireless router, which is not usually a possibility, as routers tend to be located in a central area of households to maximize the signal received throughout the house, and 3D printers tend to be kept in back rooms and hackerspaces people set up in their houses.

Remote Web Server

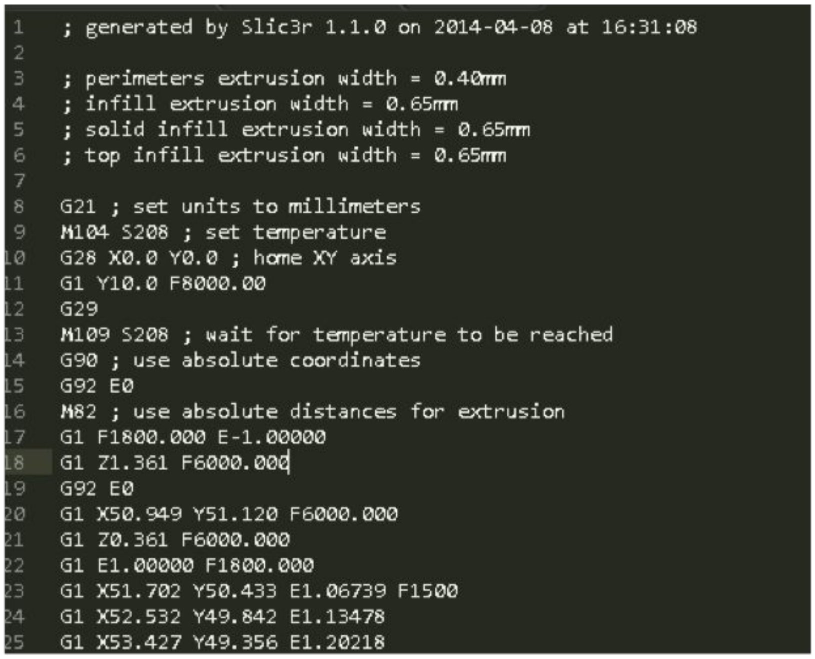
We used Azure to host our remote server, but any service which provides general-purpose linux virtual machines would work just as well.

Azure was chosen because we had free credit to host on Azure, and since the software is not tied to the service at all there was no need to be picky about providers. Azure provided more than enough in terms of horsepower and bandwidth to implement our project on.

On-Board the Raspberry Pi: Interpreting

Either received from the Azure server or fed directly to the Pi, a sliced model ready for printing comes as a G-Code file (see figure 14). As the Pi will be replacing the standard PCB, it will then interpret the G-Code and send the appropriate commands to x,y,z and extruder motors and read from the thermistor to adjust voltage sent to the heater block (See Appendix G-Code Commands).

Figure 14: G-code Example



```
; generated by Slic3r 1.1.0 on 2014-04-08 at 16:31:08

; perimeters extrusion width = 0.40mm
; infill extrusion width = 0.65mm
; solid infill extrusion width = 0.65mm
; top infill extrusion width = 0.65mm

G21 ; set units to millimeters
M104 S208 ; set temperature
G28 X0.0 Y0.0 ; home XY axis
G1 Y10.0 F8000.00
G29
M109 S208 ; wait for temperature to be reached
G90 ; use absolute coordinates
G92 E0
M82 ; use absolute distances for extrusion
G1 F1800.000 E-1.00000
G1 Z1.361 F6000.000
G92 E0
G1 X50.949 Y51.120 F6000.000
G1 Z0.361 F6000.000
G1 E1.00000 F1800.000
G1 X51.702 Y50.433 E1.06739 F1500
G1 X52.532 Y49.842 E1.13478
G1 X53.427 Y49.356 E1.20218
```

Above is the opening lines of the G-Code of a Printrbot Fan Shroud, open and released to the community, which presents numerous examples of common G-code commands.

Python, GPIO, and G-Code

The interpreter is written in python due to Raspbian's RPi.GPIO library; a easily configurable and powerful read-write package to access and read input and output from the Pi's GPIO pins. Since 2012 RPi.GPIO has come installed in Raspbian's release images, so every new or up to date installation of the OS will not have to track down any dependencies to run our interpreter.

Inputs and outputs in the GPIO library are boolean values of GPIO.HIGH or GPIO.LOW depending on the state of it being fed 3.3V of power or not. Due to the low voltage through the pins, merely the state of true or false will be delivered to the A4988 stepper driver (see fig. 4) which will then drive the Nema 17 Stepper Motors.

Many G-Code commands for 3D printing follow a standard pattern. For example:

```
G1 X51.7 Y50.4 E1.0 F1500
```

G1 informs the printer that this command is for a "controlled move", and to move towards the XY coordinate (51.7, 50.4), and while doing so, extrude 1.0 mm of plastic. This particular command is also setting the feed rate to 1500mm/m. The unit of mm would have been toggled by a previous G-code command.

The commands the Raspberry Pi will be able to interpret will focus on "G" commands, which focus on machine motion, or "M" commands, which fall in the miscellaneous category. For the purposes of getting clean prints from the Printrbot Simple, M commands will focus on fan control, temperature control, errors and emergencies.

Commandline line arguments

The only information the interpreter needs to be fed in the command line call is the G-code file name. All other variables like temperature, print speeds and other live print options are addressed in the G-code itself.

Upon Receiving a G-Code file

The process of interpreting a G-code file once received directly from the user or the server is the same:

1. Validate file
2. Parse through possible comments, freshly sliced G-code from Slic3r comes with pre-comments.
3. Set internal variable switches according to opening M and G commands, such as home axis point, temperature, and units.
4. Begin parse through of each instruction procedurally.
5. At end of G-code instruction file perform GPIO voltage cleanup, even if parse error occured in file.

Build Plan

Prototyping

Hardware

The hardware prototyping intends to create a low cost, fully functional interface to be placed between the Raspberry Pi and the components of the 3D printer. To do this, each part must be tested independently and a basic framework of use needs to be built to allow the software layer to utilize the hardware interface with as little hassle as possible. For example, the stepper drivers will need a function to drive them a given number of steps, a function to calculate the distance moved by those steps, and a function to change the direction of the stepper motor. Once those are written and tested, the process of coding the interpreter should be immensely simplified.

So far the hardware prototyping is completed for the most important part of the hardware, the stepper motor controls, which need to be controlled with the utmost accuracy for the final 3D printer to function correctly. Since the endstop switches are simple binary switches very little testing was needed to prototype them, so they are as good as finished as well. This only leaves the most dangerous component, the hotend temperature control. With multiple options awaiting testing that could be useful in different situations, a decision is going to need to be made on which method to implement in the final product. More research and test data using a controlled environment, such as reading the data from a thermistor attached to a live hot end controlled by industry standard hardware, will need to be collected before a final decision can be reached.

Interpreter

The interpreter aims to take progress done on the hardware prototyping progress using the A4988 stepper drivers and expand it out to full functionality of the x, y, z, and e motors. Interpreted commands are custom built depending on their variance; turning a fan off differs from moving the print head back to a home axis point. More closely related G commands are easier to build together.

Work is methodical as the cost of failure involves awaiting another hardware shipment if wiring and voltage connections are not carefully watched. Best practices such as clearing voltage from the GPIO pins after running particular commands must be observed. The python can be rewritten and recompiled quickly but rewiring a fried bread board takes longer. Measuring twice and cutting once is the ethos in building the commands.

Web Interface

The networking software for the project is being iterated on and prototyped as it is built. Due to the low cost of failure (really, only time spent), this software is not thoroughly prototyped before being developed.

Before features are added and integrated into the project, proof of concept implementations are always done, and features are researched enough to ensure that the operation is certainly possible before there is a major commitment to it. This more rapid, less meticulously planned out, design processes seems to be the best fit for this type of system.

Final Implementation

Hardware

The final hardware implementation will be done as a PCB with the final circuit built in and soldered. Due to the relative cost, compared to prototyping on a breadboard or similar, of having a fully functional PCB created and the parts permanently soldered in place it would be preferable to leave those stages until we are sure it is functional and has all of the properties we need to fully run our 3D printer.

The intention is to have this done early next semester, most of the research has already been done and the only thing left is to prototype each component and test their viability.

Interpreter

Implementation of the interpreter has been largely “ramp up”. Learning python was a quick process, as was starting the foray into physical computing via the Raspberry Pi’s GPIO pins. Sending proper commands through to motors is currently in an in-house “emulation” phase, by switching different LEDS on and off to represent different motors, as we await the delivery of both a power supply and extra Nema 17 stepper motors.

Upon delivery of the external supply, expected by the end of 2014, final implementation will commence and unit testing of commands will begin.

Web Interface

Implementation of the final version of the web interface and software pipeline (managing the printing process step by step) is already underway, and has been throughout the semester. Features are tackled one by one, in no particular order, unless there is a dependance on something being complete before it.
Currently the system is only able to as far as getting the G-code onto the Raspberry Pi, and that will be the case until the interpreter and printer implementations become more developed.

This implementation plan has the drawback of there being a lot more work to do when a major shift occurs (like the transition from Go to Node.js) but at the same time it helps us get our heads wrapped around exactly what needs to happen, and how should it be implemented. The underlying architecture has evolved, it has never been scrapped entirely, and our ideas about exactly what this system should look like would be far less developed without this process.

The interconnective between the Raspberry Pi and the server should be completed by the start of classes in January, while the configuration mechanism (whose final design is not yet solidified) will wait until the interactions between the server and Raspberry Pi are fully ironed out and known before they are developed. After these steps are completed, the configuration options we will need to support will be chosen and implemented in the web interface, paying specific attention to security, as it will likely be our largest potential vector of vulnerability. As soon as the printer is able to print, and the interpreter is functional, work on solidifying the software pipeline and managing the print job can begin (actually invoking the interpreter, waiting until it’s finished, cleaning up, and keeping track of its status along the way).

While all of this is happening, research into the stretch goals is happening, and we will determine which of them, if any, we will attempt to implement. These will be more of a group effort than the rest of the web interface, as many of the stretch goals involve interacting with the printer and extracting information.

Test Plan

Testing Environment

Two members of the team have Printrbot Simple 3D printers, and can run end-to-end tests of the system. They are also responsible for running the hardware tests. The server software tests can be run by anybody in the group.

The Raspberry Pi units which testing will occur on will be running version 7 of Raspbian Linux a Debian derivative operating system, as it is the recommended operating system for the Raspberry Pi, and it comes preloaded with all of the software needed with the exception of Node.js. The remote web server is running Ubuntu Linux, version 14.04.

The version of Node.js being supported is only the latest stable release, version 4.x, so testing does not need to occur on systems running other versions of Node.js.

Stopping Criteria

If blocking errors are discovered during testing resulting in the system being unusable, the rest of the team should be contacted for triage. Bringing the system back into a functional form will take priority.

Non-blocking errors, such as visual glitches, should be documented and brought up during the next team meeting, or the team member responsible for that aspect of the system should be notified.

Description of Individual Test Cases

Individual Component Test Cases

Test 1: Uploading A 3D model to the intermediate server

Test Objective

To ensure 3D models can be uploaded to the intermediate server and are sliced.

Test Description

The tester will upload a valid .stl file to the intermediate server following these steps:

1. Access the intermediate server at *ip address or url of server*/pi.html
2. Click “Browse” to choose a file to upload
3. Browse to a valid .stl formatted 3D model and click “ok”
4. Click “Submit”

Expected Results

A .gcode file has been created in the models/ directory of the web server, and the .stl file has been disposed of.

Test 2: Uploading A G-code file to the intermediate server

Test Objective

To ensure G-code files can be uploaded to the intermediate server and are accepted as they are.

Test Description

The tester will upload a valid .gcode file to the intermediate server following these steps:

1. Access the intermediate server at *ip address or url of server*/pi.html
2. Click “Browse” to choose a file to upload
3. Browse to, and select, a valid .gcode file, and click “ok”
4. Click “Submit”

Expected Results

The .gcode file will be saved to the /models directory of the web server and not modified at all by the server.

Test 3: Uploading A G-code file to the Raspberry Pi

Test Objective

To ensure G-code files can be uploaded directly to the Raspberry Pi and be saved correctly.

Test Description

The tester will upload a valid .gcode file to the Raspberry Pi following these steps:

1. Access the web interface hosted on the Raspberry Pi by connecting to *ip address of Raspberry Pi*/pi.html
2. Click “Browse” to choose a file to upload
3. Browse to, and select, a valid .gcode file, and click “ok”
4. Click “Submit”

Expected Results

A .gcode file has been created in the models/ directory.

Test 4: Uploading A 3D model to a Raspberry Pi with no slicing server

Test Objective

To ensure 3D models are rejected by the Raspberry Pi if there is no slicing server specified.

Test Description

The tester will upload a valid .stl file to the Raspberry Pi following these steps:

1. Access the web interface hosted on the Raspberry Pi by connecting to *ip address of Raspberry Pi*:8080/pi.html
2. Click “Browse” to choose a file to upload
3. Browse to, and select, a valid .gcode file, and click “ok”
4. Click “Submit”

Expected Results

The Raspberry Pi does not accept the 3D model, and nothing is stored on the Raspberry Pi.

Test 5: Add multiple printers to the web interface on the intermediate server

Test Objective

To ensure the user is able to add multiple different 3D printers to the web interface on the intermediate server.

Test Description

The tester will add two different printers to the web interface of the intermediate server by following these steps:

1. Access the web interface hosted on the Raspberry Pi by connecting to *ip address of Raspberry Pi*/pi.html
2. Click the dropdown box and select “Add Printer”
3. Type in the printer’s name and IP address
4. Click “Okay”
5. Repeat these steps to add an additional printer
6. Refresh the page

Expected Results

The names of both servers which were added should be listed in the dropdown list still, and they should remain through sessions.

Test 6: Remove a printer from the web interface on the intermediate server

Test Objective

To ensure the user is able to remove printers from the web interface on the intermediate server.

Test Description

The tester will remove a printer from the web interface

1. Access the web interface hosted on the Raspberry Pi by connecting to *ip address of Raspberry Pi*/pi.html
2. Click the dropdown box and select “Add Printer”
3. Type in the printer’s name and IP address
4. Click “Okay”
5. Repeat these steps to add an additional printer
6. Refresh the page
7. Click the dropdown box and click “Remove Printer”
8. Remove the first printer which was added
9. Refresh the page

Expected Results

The name of only the second printer should be displayed in the dropdown list in the web interface of the intermediate server.

Test 7: Feed appropriate movement “G” commands to Interpreter

Test Objective

Ensure the interpreter is parsing x, y, z and e motors correctly.

Test Description

The tester will create and run a dummy print focused on “G” movement commands, testing:

G0: a “rapid” movement

G1: a controlled movement to set coordinate
G4: Dwell/Sleep command for time before resuming movement
G28: Bring all Axis “Home”
G29: Detailed Z probe (3 points)
G30: Single Z probe at current XY location
G90: use absolute coordinates
G92: Set current position to coordinates given

Expected Results

Smooth logical movement from expected coordinates to next command. Certain commands like G4 must to tested in context, with another movement command afterwards to ensure the “dwell” has ended.

Test 8: Feed appropriate miscellaneous “M” commands to Interpreter

Test Objective

Ensure intended M commands both have desired effects, and correctly affect dependent G commands.

Test Description

The tester will create and run a dummy print focused on “M”, and “M-dependent” commands, testing:

M0: Unconditional stop
M1: For our purposes, same as M0
M17: Power/ Enable all motors
M18: Disable all motors
M82: set E to default
M85: set inactivity shutdown timer with a parameter
M92: set axis steps
M104: set extruder target temp
M105: read current temp
M106: Fan ON
M107: Fan OFF
M109: Set temperature and wait with parameter
M112: Emergency stop
M115: display message

M119: output endstop status to serial port
M140: set bed target temperature
M190: wait for bed temp to reach target temperature
M200 set filament diameter
M201: set max acceleration for *print* moves
M202: set max acceleration for *travel* moves
M203: set max feedrate for your machine
M204: set default acceleration
M205: advanced settings (see appendix)
M206: set additional homing offset
M207: set retract length
G10: retract filament according to settings of M207
M208: set recovery of unretract length
G11: retract recover filament according to settings of M208
M209: enable auto retract detect
M220: set speed override percentage
M221: set extrude factor override percentage
M302: allow cold extrudes
M400: finish all moves
M500: store all parameters in EEPROM
M501: read all parameters in EEPROM
M503: print current settings from memory (not EEPROM)

Expected Results

Have commands create desired effects (i.e. have the fan turn on, off). Have dependent commands like G10/G11 correctly depending on settings of M commands.

Test 9: Verify consistent control of hotend temperature

Test Objective

To verify consistent temperature results from the data read from the thermistor by the Raspberry Pi

Test Description

By attaching two thermistors to the same heated block, one attached to our test system and one attached to commercially available equipment meant to drive 3D printers, we can verify that our system will be able to read the data we need quickly enough for our software to react.

Expected Results

Plotted data from both thermistors should be relatively close in both output and frequency.

End-to-end Test Cases

Test 1: Printing a STL-formatted 3D model remotely

Test Objective

To ensure that remote 3D printing works as intended.

Test Description

The tester will, from a location outside of the 3D printer’s local network, upload a .stl model to the intermediate server.

1. Access the web interface hosted on the intermediate server at *ip address or url*/pi.html
2. Click “Browse” to choose a file to upload
3. Browse to, and select, a valid .stl file, and click “ok”
4. Click “Submit”

Expected Result

The model should be sliced properly, sent to the Raspberry Pi, and the print job should run, with the result being a valid physical replication of the model. At the end of the process, no files should remain on either the server or Raspberry Pi.

Test 2: Printing a G-code file remotely

Test Objective

To ensure that remote 3D printing works as intended when a G-code file is provided.

Test Description

The tester will, from a location outside of the 3D printer’s local network, connect to the remote server via the web interface and upload a valid .gcode file to the intermediate server following these steps:

1. Access the web interface hosted on the intermediate server by connecting to *ip address or url of server*/pi.html
2. Click “Browse” to choose a file to upload
3. Browse to, and select, a valid .gcode file, and click “ok”
4. Click “Submit”

Expected Result

The model should be forwarded to the Raspberry Pi, and the print job should run, with the result being a valid physical replication of the model. There should be no remaining files left on the server after completion as a result of this operation (the /models directory should be empty).

Test 3: Printing a G-code file locally

Test Objective

To ensure that local 3D printing works as intended.

Test Description

The tester will, from a location inside of the 3D printer’s local network, connect to the Raspberry Pi via the web interface and upload a valid .gcode file following these steps:

1. Access the web interface hosted on the Raspberry Pi by connecting to *ip address of Raspberry Pi*:8080/pi.html
2. Click “Browse” to choose a file to upload

3. Browse to, and select, a valid .gcode file, and click “ok”
4. Click “Submit”

Expected Result

The print job should run with the .gcode file as the source of instructions, with the result being a valid physical replication of the 3D model. This should not result in any files remaining on the server (/models directory of the project should be empty).

Test 4: Printing an invalid G-code file locally

Test Objective

To ensure that the interpreter correctly identifies issues with G-code files.

Test Description

The tester will, from a location inside of the 3D printer’s local network, connect to the Raspberry Pi via the web interface and upload an invalid .gcode file, which contains unsupported instructions following the following steps:

1. Access the web interface hosted on the Raspberry Pi by connecting to *ip address of Raspberry Pi*:8080/pi.html
2. Click “Browse” to choose a file to upload
3. Browse to, and select, an invalid .gcode file, and click “ok”
4. Click “Submit”

Expected Result

The print job should fail immediately, before running any of the instructions in the G-code file. The /models directory of the project should be empty.

Test 5: Printing different 3D models to different 3D printers at the same time

Test Objective

To ensure that printing to different printers works as intended for users with multiple printers.

Test Description

The tester will, from a location outside either 3D printer's local network, select a 3D model and the first printer to print to, then submit. Immediately after (before slicing finishes on the first), select a different 3D model to print and the second printer as the target, then submit. The following steps detail exactly what needs to be done:

1. Access the web interface hosted on the intermediate server by connecting to *ip address or url of */pi.html
2. Add two different 3D printers to the web interface, if there are not two different printers already registered (select dropdown list -> Add Printer)
3. Click "Browse" to choose a file to upload
4. Browse to, and select, a valid .stl model, and click "ok"
5. Select the first printer to print to from the drop down list
6. Click "Submit"
7. Click "Browse" to choose another file to upload
8. Browse to, and select, a different valid .stl model, and click "ok"
9. Select a different printer from the drop down list
10. Click "Submit" to print to the other printer

Expected Result

Both models should be sliced and sent to their respective 3D printers, each correctly printing the 3D model that was selected for it. Neither the server nor the Raspberry Pi's the printers are running on should have anything in their /model directory at the completion of both print jobs.

Evaluation Plan

Completeness

The project's completeness will be determined based off of how closely our delivered project matches our set of requirements.

The project will be deemed complete when it

- Is able to completely replace the control electronics onboard the 3D printer
- Is able to print 3D models with no computer (other than the Raspberry Pi) attached to it
- Allows users to print to it remotely
- Allows users to print 3D models, instead of necessarily supplying G-code directly
- Correctly delegates slicing work to an intermediate server, instead of attempting to do it onboard the Raspberry Pi
- Allows users to connect to either the Raspberry Pi or the Intermediate server (depending on their desires) through a web interface
- Allows users to configure multiple printers through the web interface, and print to any of them arbitrarily

If time permits, further work which will be done on the project subsequent to it being deemed complete shall be

- Allowing the user to pause print jobs
- Allowing the user to perform predefined specialized operations on the printer, such as moving the heated nozzle away from the print, or calibration
- Updating the interpreter to also work with delta-style 3D printers
- Implement automatic bed leveling
- Allowing the use of multiple extruders on a single 3D printer
- Implementing less important G-code functions which were left out
- Adding real-time information about the printer

Our project's completeness will not rely on these work items being completed, but we believe that they (at least some of them) are not unreasonable goals, and we would feel like our project is more complete and usable in a real-world scenario if it had a more complete implementation of the G-code standard, for instance, or more robust status monitoring in the web interface.

Usability

The usability of our system will be judged by its ability to perform in comparison to current standards. This would include:

Is is able to print most real-world 3D models?

Clearly, it the most important usability metric is the ability to print models that you would find in the real world. This is entirely dependant on the completeness of the G-code implementation on the part of the interpreter, assuming the hardware works as intended.

This is easily determined by attempting to print models directly from websites such as Thingiverse http://www.thingiverse.com/ (as in, letting the intermediate server perform the slicing with the options and configurations we've provided), as well as slicing the models ourselves with more advanced configurations, or perhaps with other commonly used slicing software, and ensuring that these models are processed correctly and the G-code is understood by the printer accurately.

Is the UI fully featured?

This is a bit more difficult to define. The user should be able to perform their daily printing tasks via the web interface that we provide, given they have a server they can use for slicing. Manual slicing should never be a requirement, it should only be an option in case a very specific use case is needed which is far enough from the norm that it was determined to be unnecessary to support when adding configuration options to the web interface.

Again, by printing 3D models which are downloaded from Thingiverse a wide selection of real-world 3D models will be tested. If they are able to be printed without error, the project's UI can be deemed sufficient.

How difficult is it to configure?

Our goal is to make printing easier. The last thing we want is a configuration nightmare which distracts from the contributions our project is making. To this end, we would like a working web interface running on the Raspberry Pi with zero configuration.

Of course, the user will have to be running in the same environment as us, as in they will need a Raspberry Pi Model B+ and a Printrbot Simple, but given this, the extent of what they should need to do to run our project is installing Node.js if it does not come preinstalled on their distro and running our program.

Configuration of the intermediate server is necessary though, as it needs to know where the Raspberry Pi is (the local server is physically connected to the computer, it definitely knows where to look for the GPIO pins). The Raspberry Pi will also need to be visible to the intermediate server, which means that the user will have to perform tasks such as port forwarding if the intermediate server is not on the same local network as the printer. These tasks are often challenging for inexperienced users, and simplifying this process as much as possible is a goal of ours.

The evaluation of this difficulty necessarily is subjective, but the desire for it is for there to be a configuration script which walks the user through the changes that may need to happen on the configuration file, and the user may have to do some networking to allow the server to see the printer.

Personnel

Raymond

Experience

Raymond Cload is a senior at UCF studying Computer Science whose hobbies include 3D modeling, 3D printing, and generally taking apart anything within arms reach. With this in mind, Raymond is in charge of the design of the electronics used to allow the Raspberry Pi to interface with the hardware of the 3D printer.

Personal Approach

I proposed this project because I wanted to contribute something to the 3D printing community which has helped me so much. I have owned a 3D printer for nearly 2 years now and in that time I have printed dozens of things, from a keychain to hold a few Excedrin to the framework of a second 3D printer, but most of what I have printed have been other people's designs and the few things that I did design myself tended to be too cobbled together for me to want to share.

While setting up a Raspberry Pi to replace a small file server I realized that with the new A+ and B+ models there should be enough GPIO pins to drive the motors and temperature sensors of a 3D printer right from the Pi, instead of tethering the current, rather expensive, electronics to a computer. This way, any hobbyist 3D printer should be able to be set up as a standalone machine, either with a monitor/keyboard/touchscreen to start and control prints or by connecting to a web interface over the internet.

Max

Experience

Maxwell is a senior at UCF studying Computer Science with a background in AI planning and automated software testing having done research in the field at the

University of North Texas, as well as operating system kernel development as an intern at Microsoft. He is in charge of the “cloud” aspect and software pipeline in this project.

Personal Approach

I was motivated to choose this project because I saw it as being, of all of the proposed projects, both the most relevant project to my future career, as well as having the least overlap with what I'm currently familiar with. I bring to the project my experience writing low level software, having interned with the OS kernel group at Microsoft, but I have no experience with hardware development and bridging the hardware/software divide. This resulted being a little more out of the loop over the summer whenever my coworkers would talk about their latest personal hardware hacks or hardware issues relating to the team's work.

Computer Science majors do not have any opportunities to work with physical hardware, so most CS students who do have hardware experience got that experience from personal projects. This opportunity seemed like a great way for me to jump in, since it is mostly a hardware project, with some software components, and the scope of the project seems reasonable for the amount of time we have.

I was also drawn to the project since the bulk of the work required is doing research and figuring out how to get past certain key issues, rather than writing and rewriting thousands of lines of code which do not add to my knowledge as much as they act as a necessary evil when you want to create something. With a project like this, we get to both learn a lot and make something useful.

Even though my work on this project lies entirely on the software side of the fence, I’ve made sure that I keep up with the hardware side. I’ve also experienced brand-new-to-Raspberry Pi ramp up the rest of the group has gone through, and I’ve played around quite a bit with my Pi and breadboard.

William

Experience

William is a senior in Computer Science with previous work experience at such places like Lockheed Martin and currently works at the Institute of Simulation and Training. William is in charge on the “intersection of hardware and software”: the

Raspberry Pi’s python interpreter and wiring up GPIO pins to stepper drivers and stepper motors.

Personal Approach

Choosing this project was an exercise in following a “gut-feeling”. Lots of the proposed projects involved Unity development, which is a skill-set I practice four days a week at work. Needless to say I recused myself from these projects – not only to seek personal challenges of exploring new topics, but to avoid mixing my work life with senior design and turning both into drudgery.

When the words “Raspberry Pi” and “3D printing” started being thrown around, there was an internal voice that said, “Oh, I want to work on that right now”. I feel this is important when committing to the senior design project; where the feeling of compulsion outweighed the feeling of obligation. Things like exploring the Raspberry Pi, or in-home 3D printing seemed like activities I would “get around to” at some nebulous far-flung date when the free time would make itself apparent. Pursuing this project has not just allowed me, but really forced me, to make time for this. Make time for exploring a really limited Linux machine. Make time to finally explore 3D printing, and make time to connect one’s code to a finished product and prototype sitting in front of you.

Unlike a lot of the senior design projects this one gives me the most feeling of “owning the whole stack” and understand what each piece does and hands-off to what, when, and how, and that’s really exciting.

Related Work

Hardware

When this project was proposed there was only one blip on the radar for such a project, a Reddit post by user Wallacoloo (which can be read http://goo.gl/GSgHI8) who managed to build a similar design to what we had in mind to drive his Delta style 3D printer via the Raspberry Pi and even open sourced his interpreter to drive the hardware, although no diagrams of the hardware itself. Since then several other similar projects have surfaced, but they are either lacking in documentation or, in the case of the failed indegogo campaign by Mungkie (see http://goo.gl/niydhI), are dead in the water. The largest differences between our project and those that have been unable to get off the ground is that we have multiple people working on the system rather than just one, and we have resources provided by the University to allow us to test hardware without the need to purchase expensive tools.

OctoPrint

The idea of controlling a 3D printer through a web interface is not new, in fact there is a very popular piece of software that facilitates this and provides a very powerful user interface, which is even specifically designed to run on the Raspberry Pi, called OctoPrint. There are key differences between the use case of our project and OctoPrint which necessitated the design of a new system for managing print jobs for our project.

OctoPrint is designed to run on a Raspberry Pi which is connected to a 3D printer's control electronics via USB. This is fundamentally different than the approach that we have, which is to remove those components of the printer entirely and replace them with the Raspberry Pi.

In addition, as OctoPrint was designed to be physically connected the the 3D printer is is interfacing with, the ability to interact with multiple printers was never added to OctoPrint, and the likelihood that the backend structure of the software was designed in a way that is fundamentally incompatible with printing to multiple printers was a big concern.

OctoPrint does contain some powerful features, such as the ability to visualize your G-code before you print (actually see what your model will look like before printing it), the ability to connect a webcam to monitor your printer remotely, and the ability to see real-time detailed information about the state of the printer and the print job.

We could conceivably incorporate G-code visualization the same way OctoPrint did, as they integrated a view which used the open source project gCodeVisualizer (http://gcode.ws/) to render the visualization.

OctoPrint is also more limited in that there is no way to print a 3D model with it directly, as it cannot perform slicing on the Raspberry Pi (for the same reasons we cannot), but it does not have the functionality of assigning the job to a more powerful server, which was one of the key aspects we desired of our project.

In the end, the Lack of these key features we needed kept us from using OctoPrint as our base software-wise, and instead we are writing the software from scratch.

Budget

There are no sponsors, so each member of the team is responsible for their own expenses.

Since we do not have a sponsor we have purchased our own Pi’s and accessories to develop on. Some of the items below were previously owned by team members, as such this isn’t so much a budget (Table 2) as it is a way to track the cost of everything necessary to develop the system.

Table 2: Budget

Item	Price	Number Purchased	Cost
Raspberry Pi model B+	$35	3	$105
Sunfounder Development kit	$33	3	$99
Pack of 5x A4988 Stepper Drivers	$13	1	$13
Printrbot Simple 3D printer	$350	2	$700
Azure Virtual Machine	$55	Monthly*	$55*
Analog to digital converter (MCP3002)	$2.30	3	$6.90
Raspberry Pi 40pin shrouded header	$0.95	3	$2.85
Raspberry Pi 40pin ribbon cable	$2.50	3	$7.50
N-Channel MOSFET 60V 30A	$0.95	6	$5.70
		TOTAL	$994.95

Milestones

General

Familiarize ourselves with Raspberry Pi

Going in to the project, most of us had never touched a Raspberry Pi before. There is a ramp-up period to where we are familiar enough working with the Raspberry Pi that we can produce meaningful work.

Allow Raspberry Pi to update status of the print

The Raspberry Pi should be able to keep track of its current status during a print, such as when it is not busy, printing, completed, or if there are any errors. This may include status indicated from various G-code commands, to error outs, to a print being finished.

Both the web interface on the server and on the Raspberry Pi should be able to update this global status. In the case of the server, it would be able to update the status granularly for each individual printer which is registered to it.

William

Build a Second Printrbot Simple Unit

Upon realizing the bottleneck of having one 3D printer to perform component tests and end-to-end test, a second 3D printer was acquired and constructed. This also aided in familiarity with the 3D printing process and RepRap design.

Determine which G-code operations MUST be implemented for the system to be viable

G-code operations have a gradient of usage, with some operations being extremely common, and others being rarely used. In addition, some G-code commands are not necessary to print a model, such as changing materials, and so are not necessary to implement for a functional system. The necessary G-code instructions to implement.

Familiarize self with Python

Raspbian's powerful RPi.GPIO package made the choice of language to write the interpreter in easy. Having never programming in python before, a small learning ramp up is required.

Complete subsection of "G" motion commands in Interpreter

G commands focus on fine tune step movement of the 4 motors of x,y,z and extrusion. This is the backbone of getting an object printed.

Complete subsection of "M" commands in Interpreter

M commands vary from setting variables that will remain for the length of the print, like temperature, to more binary variables like turning the fan on or off.

Functionality for interpreter to catch errors and elegantly stop the print

A possible unfamiliar G-code command or parsing error needs to be handled with both alerting the user and safely resetting the GPIO pins back to their grounded state.

Allow the Raspberry Pi to validate files

The Raspberry Pi must be able to validate .gcode files before beginning the print job, and aborting if there are any unsupported instructions or potentially malicious activity.

Ray

Design the interface board

An interface board must be designed to facilitate interactions with the 3D printer in order to reduce latency to levels small enough that accurate 3D models can be produced. The first major goal to that end is designing the interface board's layout and what will be included in it.

Create a working prototype of the interface board

Prototyping the interface board using a breadboard and inexpensive parts to ensure the design is valid and has the required functionality is a must, so that small errors can be corrected before a final board is printed.

Investigate hardware solutions to terminate printing in the case of temperature control malfunctions

There are concerns about the behavior of the printer if the temperature is increased past the safety limit. The printer is not capable, by itself, of terminating the print job on the event of the head overheating. We would like to look into potential hardware-based sensors which could perform this safety check, since the printer may be used when the user is not around to monitor the status of the print job.

Maxwell

Familiarize self with Node.js

Believing that Node.js would be a better bet than Go for writing the web server, I needed to quickly ramp up on and translate my work to Node.js.

Create a web interface for letting the user transfer files to the Raspberry Pi

A web must be developed which will run on the Raspberry Pi. Users must be able to connect to it and upload a .gcode file to print.

Create an intermediate server, capable of slicing 3D models

An intermediate server must be created which has a web interface running on it, and is capable of accepting 3D models (in the .stl format), slicing them, and sending them to the Raspberry Pi.

Initiate the Printing Process When G-code is Ready

When the final G-code is on the Raspberry Pi and ready to print, the server should kick off the printing process by initiating the interpreter.

Allow printing to multiple 3D printers from the intermediate server

The user should be able to register multiple printers and allow the user to choose which one print to.

Allow Raspberry Pi to accept models directly, and send them to the server for slicing

If a server is set up and configured, the Raspberry Pi should be configurable to send models to the remote server, have them sliced, and reuploaded to the Raspberry Pi.

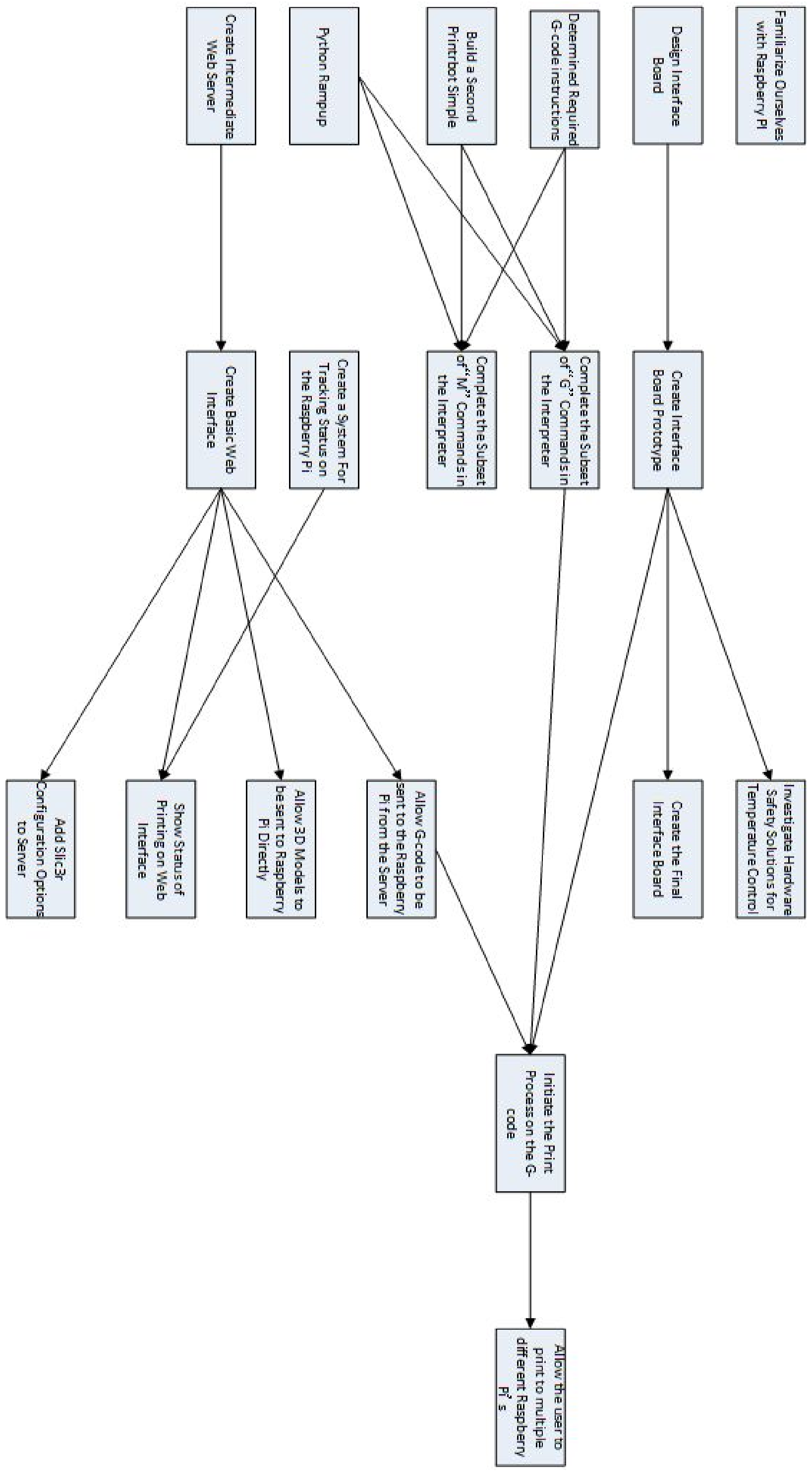
Add slic3r configuration options to the web interface on the intermediate server

Basic, important, configuration options should be able to be set in the web interface which will feed into slic3r, in case the user needs some specific slicing configuration options set, and is not on a computer with the prerequisite software to slice themselves.

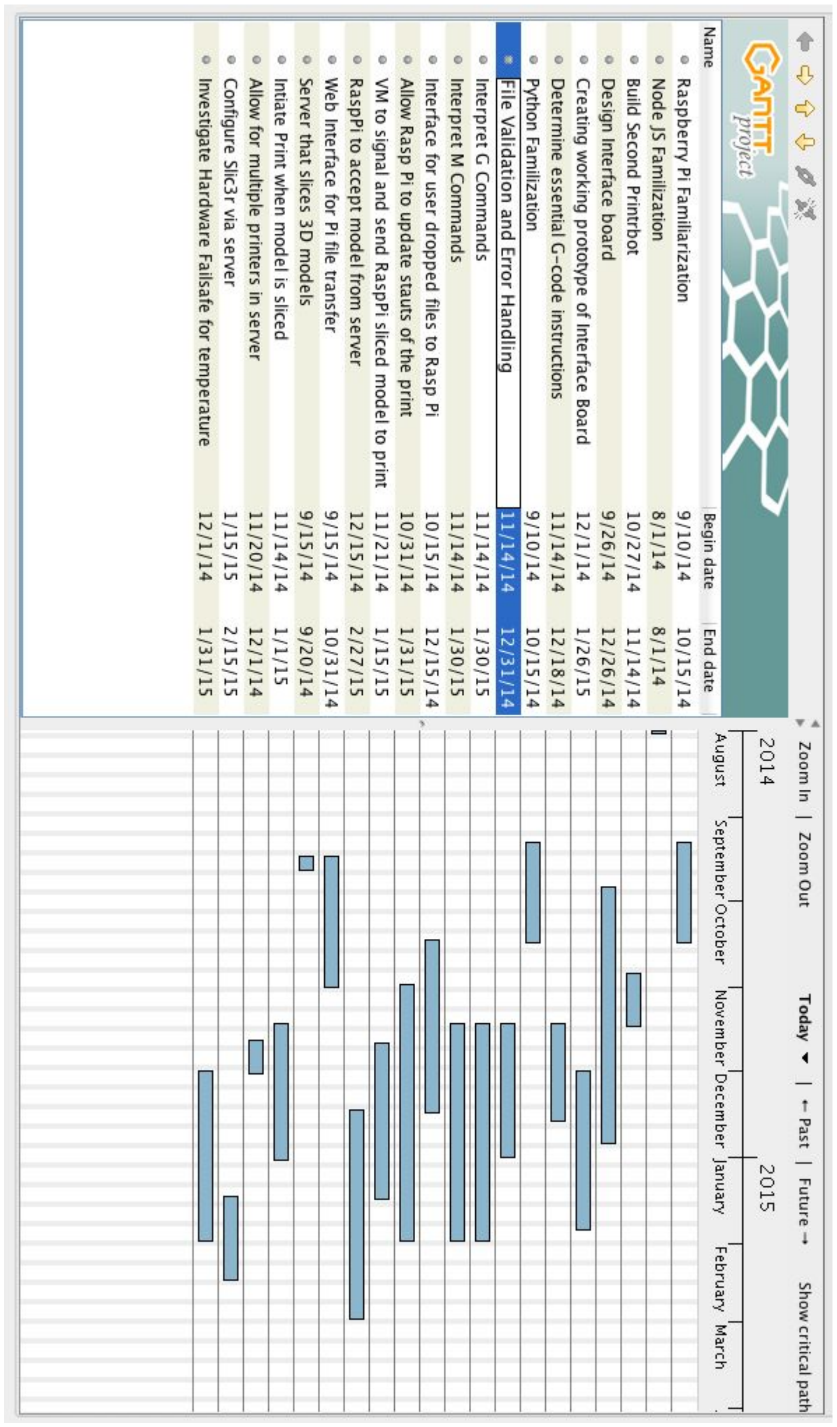
Indicate the status of the print job in the web interface

The web interface should give some indication of the status of the print job. It should indicate whether the model is being sliced, transferred to the Raspberry Pi, currently printing, and printed. Both the web interface on the Pi and the Server should receive the status notifications.

PERT Chart


Familiarize Ourselves with Raspberry PI
Design Interface Board
Determined Required G-code instructions
Build a Second Printrbot Simple
Python Rampup
Create Intermediate Web Server
Create Interface Board Prototype
Complete the Subset of "G" Commands in the Interpreter
Complete the Subset of "M" Commands in the Interpreter
Create a System For Tracking Status on the Raspberry Pi
Create Basic Web Interface
Investigate Hardware Safety Solutions for Temperature Control
Create the Final Interface Board
Allow G-code to be sent to the Raspberry Pi from the Server
Allow 3D Models to be sent to Raspberry Pi Directly
Show Status of Printing on Web Interface
Add Slic3r Configuration Options to Server
Initiate the Print Process on the G-code
Allow the user to print to multiple different Raspberry Pi's

Gantt Chart



Name	Begin date	End date
Raspberry Pi Familiarization	9/10/14	10/15/14
Node JS Familization	8/1/14	8/1/14
Build Second Printbot	10/27/14	11/14/14
Design Interface board	9/26/14	12/26/14
Creating working prototype of Interface Board	12/1/14	1/26/15
Determine essential G-code instructions	11/14/14	12/18/14
Python Familization	9/10/14	10/15/14
File Validation and Error Handling	11/14/14	12/31/14
Interpret G Commands	11/14/14	1/30/15
Interpret M Commands	11/14/14	1/30/15
Interface for user dropped files to Rasp Pi	10/15/14	12/15/14
Allow Rasp Pi to update stauts of the print	10/31/14	1/31/15
VM to signal and send RaspPi sliced model to print	11/21/14	1/15/15
RaspPi to accept model from server	12/15/14	2/27/15
Web Interface for Pi file transfer	9/15/14	10/31/14
Server that slices 3D models	9/15/14	9/20/14
Intiate Print when model is sliced	11/14/14	1/1/15
Allow for multiple printers in server	11/20/14	12/1/14
Configure Slic3r via server	1/15/15	2/15/15
Investigate Hardware Failsafe for temperature	12/1/14	1/31/15

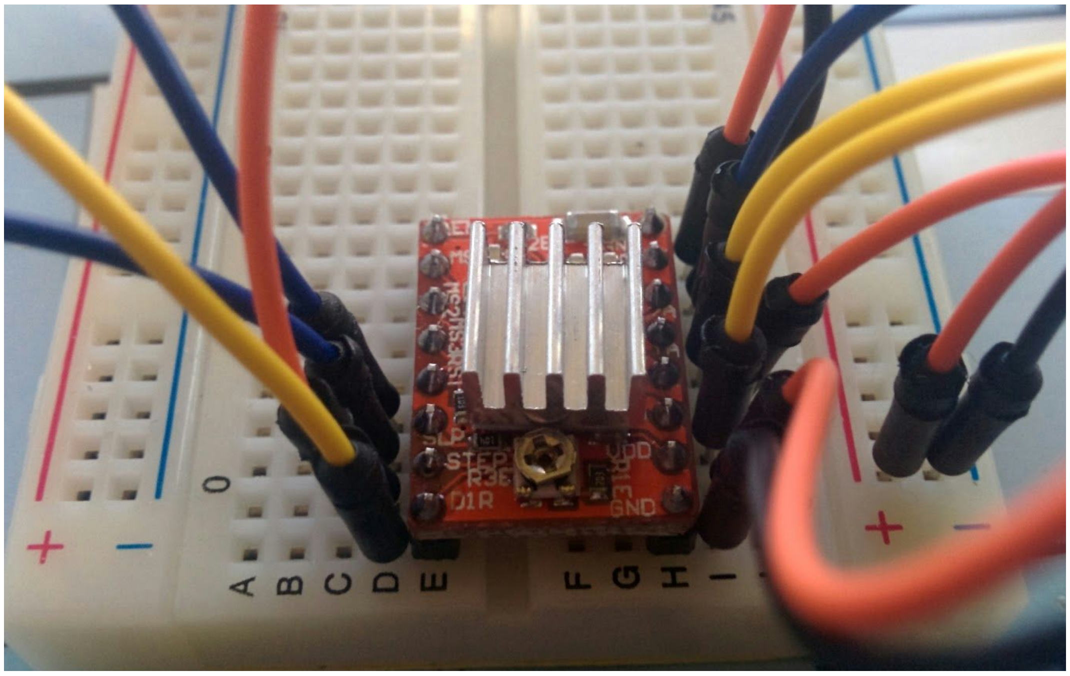
Summary & Conclusions

Hardware: Current Status

The important functionality of the printer has been investigated and has been divided into three major parts: Stepper driving, hot end temperature control, and endstop status.

As shown in figure 15 below, a successful prototype of the stepper motor control circuit utilizing A4988 stepper drivers has been designed and tested via a basic Python script made to verify the integrity of the circuit. This step in the hardware design process could be done with materials purchased or previously owned by group members, however for further implementation to allow multiple motors to be controlled simultaneously a more powerful power supply will need to be purchased or borrowed from the University. Each stepper motor uses approximately 400 milliamps of current from a 12 volt power supply via the VMOT input on the A4988 stepper driver. The power supply currently used in testing can only supply 1 amp, and as such could not power more than 2 stepper motors simultaneously.

Figure 15: Stepper driver prototype wiring on breadboard

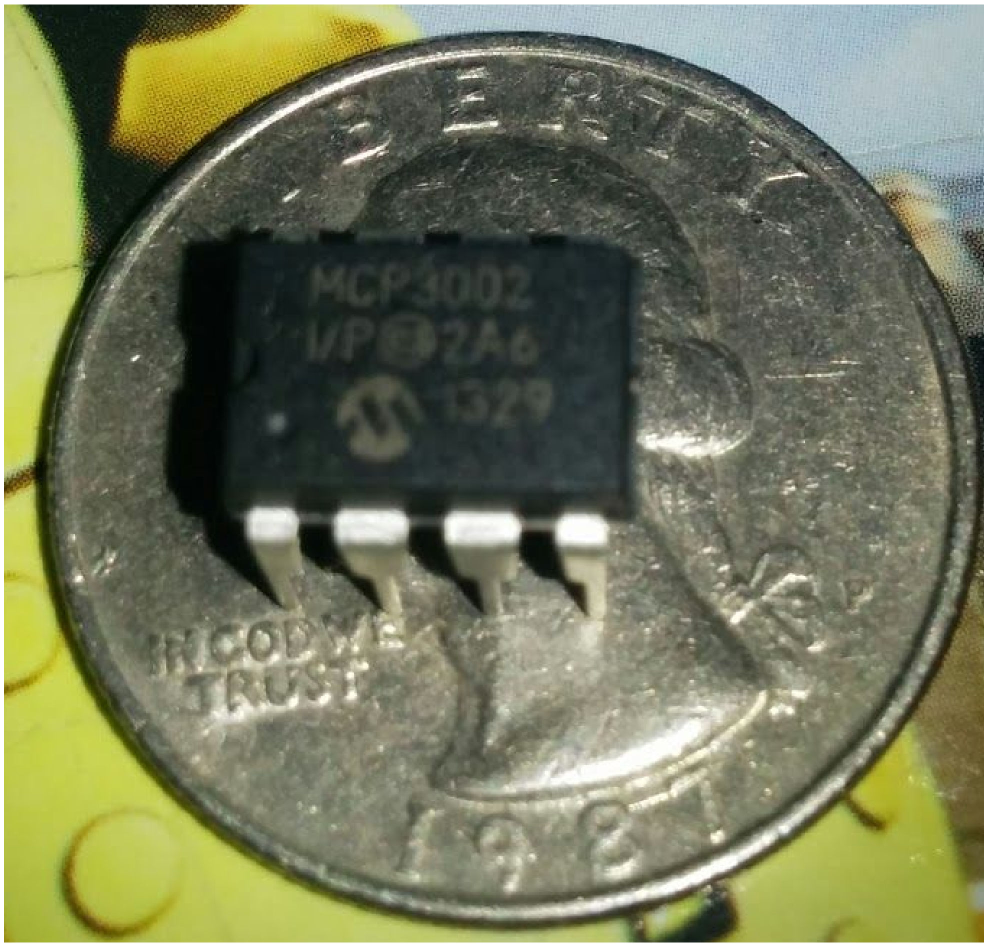


The hot end temperature control will require more investigation before testing in a prototype, in fact simple reading of a thermistor, an analog temperature sensor, via the digital inputs of the raspberry pi needs to be implemented first before any integration with the heating elements of the hotend can be done.

There are several possible strategies that can be used to implement the temperature sensors we need. The first would be to use an RC charging circuit, where you measure the resistance in a circuit by feeding a known voltage through a capacitor, a known resistor, and the unknown resistor (in our case, this is the thermistor that varies its resistance based on its temperature) and measuring the amount of time it takes for the Raspberry Pi's GPIO pin to register a HIGH voltage. You then divide the time it took by the capacitance of the capacitor to derive the total resistance of the circuit, and subtract out your known resistance. The biggest advantage to this method is that it only takes one GPIO pin on the Raspberry Pi to read a single temperature sensor. The downside to this method is that it may take a not insignificant amount of time to get the reading from the circuit and then you have to wait at least that long again for the circuit to discharge completely. It may also not be as accurate as other methods, however this will need to be tested further before a decision is made.

Another possible strategy is to utilize an analog to digital converter such as the MCP3002 pictured below (Figure 16). These analog to digital converters do just what their name suggests, converts the analog variation in resistance to digital information that can be read by the Raspberry Pi and converted back to resistance in the program logic. The biggest advantage to this setup is that the data should be very accurate, to within a few tenths of a degree celsius, and can be read often without needing to wait. The biggest disadvantage here is that the MCP3002 requires 4 of our 17 GPIO pins to operate. This can be somewhat mitigated if we decide to work towards our stretch goal of heated bed or multiple extruder support, because the MCP3002 has two channels for input, thus bringing it down to 2 GPIO pins per temperature sensor.

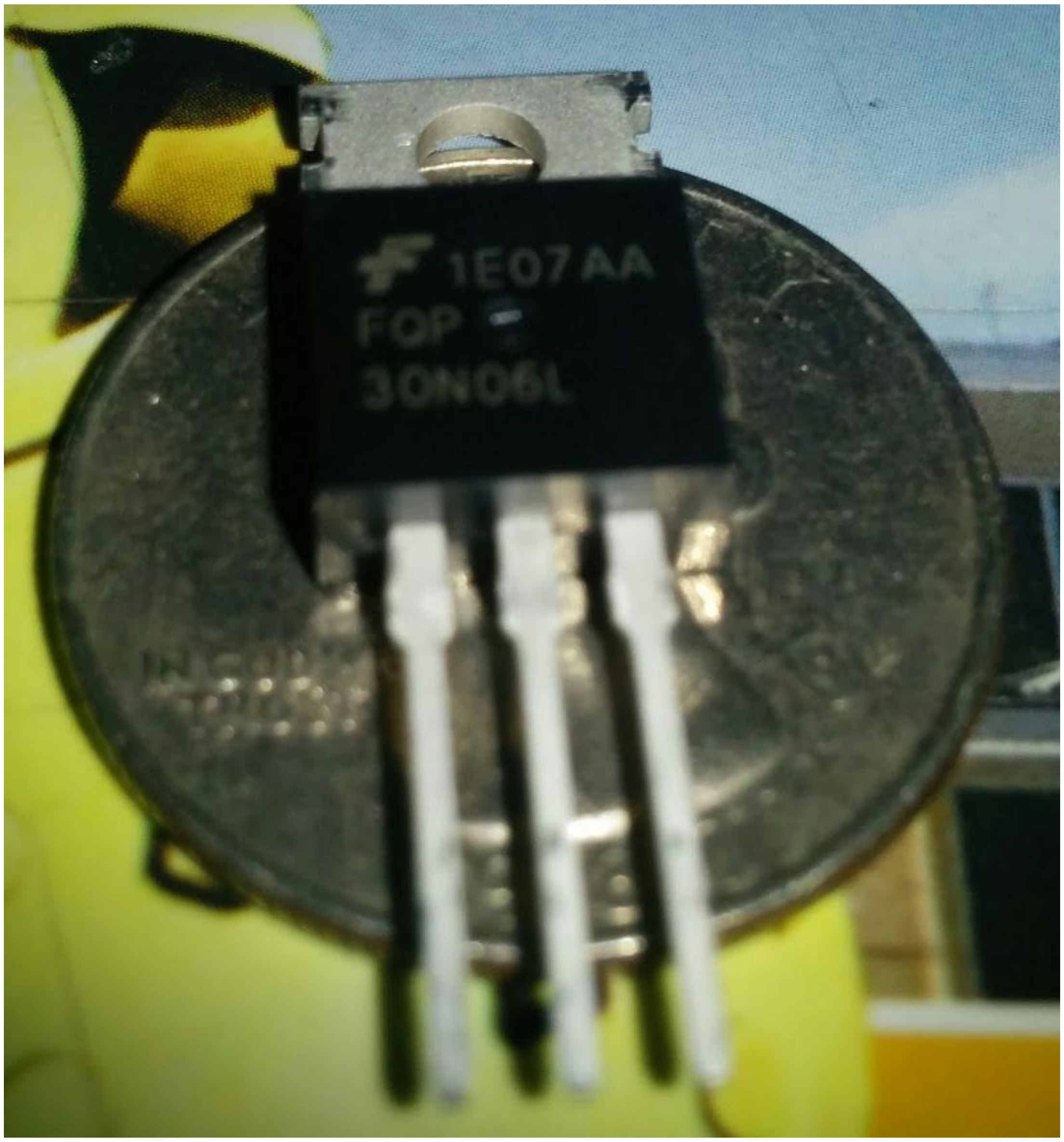
Figure 16: MCP3002 atop a quarter for scale



Once the resistance from the sensor is read by the Raspberry Pi it is converted into celsius by comparing the resistance in ohms to values in a table provided by the manufacturer of the thermistor. The thermistor found in the Ubis hotends found is the EPCOS 100K 8304 thermistor, which is listed as choice #1 in the Marlin firmware we are basing our interpreter off of. In table Y in the appendix you can see how the digital input the Raspberry Pi takes in compares to the temperature it represents. Note that while these sensors are rated to -55C, only the positive ranges are represented, as anything below 0C is obviously useless when attempting to melt materials.

The way the hot end is actually heated is by putting high current through a resistor embedded in a block of metal at the end of the hot end. When the current goes through the resistor it heats up and melts the plastic, and due to the way most thermoplastics set you want to make sure that the plastic does not have enough time to cool down until it is laid down by the printer, or at least out of the nozzle. To do this, when a temperature fluctuation is detected a single GPIO pin is set to HIGH, this pin is attached to an N-channel MOSFET (Figure 17) which allows a high current from the power supply to be switched on and off by the GPIO.

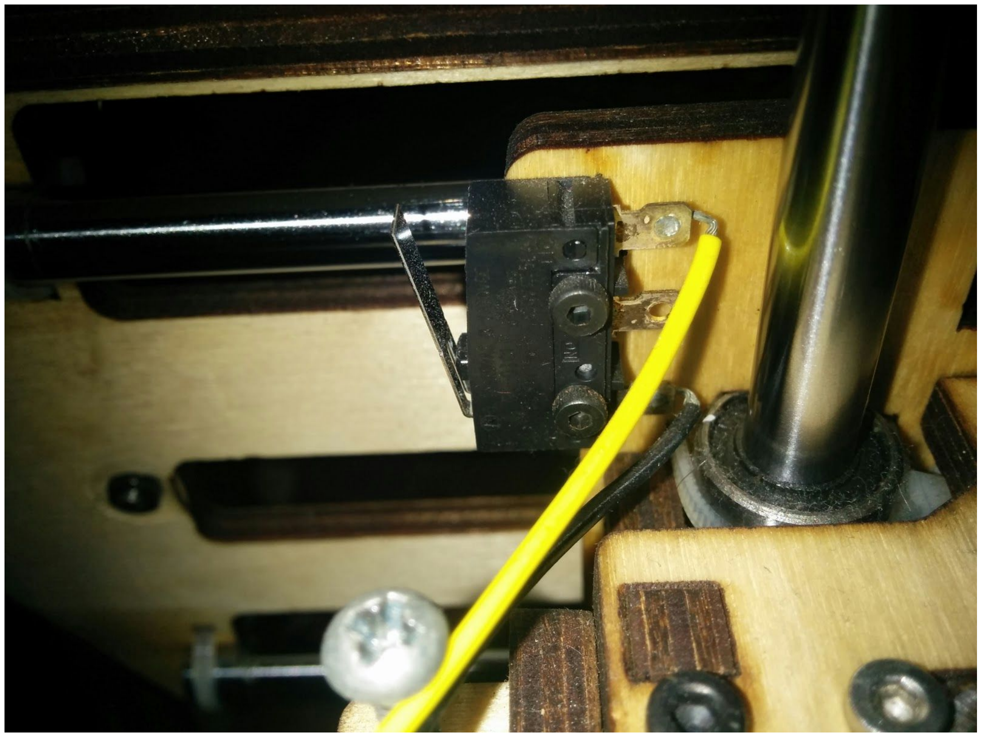
Figure 17: N-Channel Mofset



There is an issue with this setup, if something happens and the application crashes without disabling the heater core, or if the thermistor breaks mid print and begins reporting a lower temperature than it really is, it is quite possible that you can get what is referred to as ‘thermal runaway’, where the hot end of the printer heats up so much that it melts is self and can often cause fires. Fortunately the second issue has a software fix, simply set a maximum time to allow it to ‘heat’ without there being a significant change before giving up and halting the print, this could be as little as 5 seconds and you could easily prevent most of these issues, and this is in fact how most printers do this. The first issue, however, is much harder to solve and is a common problem among 3D printers. The only real safe way to do this is to have a second thermistor wired to your heating element and to an entirely different appliance that can cut off all power to the system. I do not think that this is within the scope of this project, but it may be worth investigating separately.

Finally we have to be able to read the status of the endstops (Figure 18), which are switches that are triggered when the printer reaches the ‘0’ point on any of its three axis. These switches are really only important in one of two situations, before the print starts so the printer has a repeatable starting point of (0, 0, 0) and during a print if they are triggered and none of the axis are at 0. Because of this we can simply wire all 3 of these endstops in parallel, when you check the axis at the start of a print you do one at a time, then move away from the trigger before testing the next one, and if any are triggered mid print you make an emergency stop.

Figure 18: Mechanical endstop used on Printrbot Simple



Interpreter: Current Status

The interpreter can currently read in G-code files with minimal validation, just a file type check. Current parsing happens at runtime without pre-checking the code for possible unknown commands. The command list is currently incomplete, with the focus on variable declaration M commands and simple G movement commands.

While awaiting a delivery of more Nema Stepper motors and power supply to test wiring design, a small emulation system to test which motor to command when, and for how long is being emulated “safely” and vastly cheaper with an array of LEDs.

Important functionality is attached to the progress of both the hardware and network progress. The interpreter will occasionally inform the server of the of the print progress, and specific wiring and checking needs to be done with establishing the temperature setting.

Network Functionality: Current Status

A web server has been developed which is configurable to be run as either a Raspberry Pi host or the intermediate server. This configuration is stored in a .config file located in the data directory.

The Raspberry Pi host interface contains a file upload form (a browse button and text field) to allow the user to select a file on their local machine to be uploaded to the Raspberry Pi, and a Submit button, which allows the user initiate the upload process. Upon being uploaded, the file is stored in the /model directory of the project. The web interface is styled using the Bootstrap framework.

The Intermediate server interface contains the same elements the Raspberry Pi interface hold, with the addition of a drop down list. The drop down list is populated after the page load using jQuery's AJAX capabilities. A file on the server stores the list of names and IP addresses. This list is transmitted in JSON between the server and the client browser, which uses the information to populate the dropdown list. The same mechanism can be used to write data back to the file, but the interface has not been developed to allow adding printers via the web interface yet--they are currently being added by manually editing this file.

The user is currently only able to upload .gcode files to the Raspberry Pi (the work has not been done yet to forward non-.gcode files to the intermediate server for slicing, if one is configured) but the user is able to upload .gcode, .stl, .amf, or .obj files to the intermediate server. All but .gcode files are sliced first, then they are sent to the Raspberry Pi, which downloads them and saves them in it's /models directory.

Once the other mechanisms in the pipeline, such as the interpreter, are complete, the server will just need to invoke them when ready, but as of now the extent of the process is the saving a .gcode file onto the Raspberry Pi.

The "intermediate server" is an ubuntu virtual machine running on Microsoft's Azure web service. It is publicly accessible, and capable of interacting with the Raspberry Pi.

Appendix

G-Code Commands

Necessary Commands to Interpret

G0	Rapid move
G1	Coordinated Movement X Y Z E
G4	Dwell S or P //this is a sleep command, seconds(S) or milliseconds(P)
G10	retract filament according to settings of M207
G11	retract recover filament according to settings of M208
G28	Home all Axis
G29	Detailed Z-Probe, probes the bed at 3 points. You must be at the home position for this to work correctly.
G30	Single Z Probe, probes bed at current XY location.
G90	Use Absolute Coordinates
G92	Set current position to coordinates given
M0	Unconditional stop - Wait for user to press a button on the LCD (Only if ULTRA_LCD is enabled)
M1	Same as M0
M17	Enable/Power all stepper motors
M18	Disable all stepper motors; same as M84 used.
M82	Set E codes absolute (default)
M84	Disable steppers until next move, or use S to specify an inactivity timeout, after which the steppers will be disabled. S0 to disable the timeout.

M85	Set inactivity shutdown timer with parameter S. To disable set zero (default)
M92	Set axis_steps_per_unit - same syntax as G92
M104	Set extruder target temp
M105	Read current temp
M106	Fan on
M107	Fan off
M109	Set Extruder Temperature and Wait
M112	Emergency stop
M115	Capabilities string
M117	display message
M119	Output Endstop status to serial port
M140	Set bed target temp
M190	Wait for bed temperature to reach target temp
M200	set filament diameter and set E axis units to cubic millimeters (use S0 to set back to millimeters).
M201	Set max acceleration in units/s^2 for print moves (M201 X1000 Y1000)
M202	Set max acceleration in units/s^2 for travel moves (M202 X1000 Y1000) Unused in Marlin!!
M203	Set maximum feedrate that your machine can sustain (M203 X200 Y200 Z300 E10000) in mm/sec
M204	Set default acceleration: S normal moves T filament only moves (M204 S3000 T7000) im mm/sec^2 also sets minimum segment time in ms (B20000) to prevent buffer underruns and M20 minimum feedrate
M205	advanced settings: minimum travel speed S=while printing T=travel only, B=minimum segment time X= maximum xy jerk, Z=maximum Z jerk, E=maximum E jerk

M206	set additional homing offset
M207	set retract length S[positive mm] F[feedrate mm/min] Z[additional zlift/hop], stays in mm regardless of M200 setting
M208	set recover=unretract length S[positive mm surplus to the M207 S*] F[feedrate mm/min]
M209	S enable automatic retract detect if the slicer did not support G10/11: every normal extrude-only move will be classified as retract depending on the direction.
M220	set speed factor override percentage
M221	set extrude factor override percentage
M302	Allow cold extrudes
M400	Finish all moves
M500	stores parameters in EEPROM
M501	reads parameters from EEPROM (if you need reset them after you changed them temporarily).
M503	print the current settings (from memory not from eeprom)

Optional Commands

M540	Use S[0|1] to enable or disable the stop SD card print on endstop hit (requires ABORT_ON_ENDSTOP_HIT_FEATURE_ENABLED)
M600	Pause for filament change X[pos] Y[pos] Z[relative lift] E[initial retract] L[later retract distance for removal]
M999	Restart after being stopped by error

EPCOS 100k 8304 Thermistor temperature table

Digitized Counts	Temperature (Celsius)
23	300
25	295
27	290
28	285
31	280
33	275
35	270
38	265
41	260
44	255
48	250
52	245
56	240
61	235
66	230
71	225
78	220
84	215
92	210
100	205
109	200
120	195

131	190
143	185
156	180
171	175
187	170
205	165
224	160
245	155
268	150
293	145
320	140
348	135
379	130
411	125
445	120
480	115
516	110
553	105
591	100
628	95
665	90
702	85
737	80
770	75
801	70

830	65
857	60
881	55
903	50
922	45
939	40
954	35
966	30
977	25
985	20
993	15
999	10
1004	5
1008	0

Licenses & Permissions

Node.js

Referenced 12/1/2014 from:
https://raw.githubusercontent.com/joyent/node/v0.10.32/LICENSE

Node's license follows:

====

====

Connect for Node.js

Referenced 12/1/2014 from:
https://github.com/senchalabs/connect/blob/master/LICENSE

MIT License

Bootstrap

Referenced on 12/3/2014 from:
https://github.com/twbs/bootstrap/blob/master/LICENSE

MIT License

Slic3r

Referenced 12/1/2014 from:
https://github.com/alexrj/Slic3r

GNU Affero General Public License, version 3

Model B+ GPIO Pins

“RaspPi.Tv RPi.GPIO Cheat Sheet”,
http://raspi.tv/download/RPi.GPIO-Cheat-Sheet.pdf