FINAL REPORT

Team May15-03

Shawn LaGrotta lagrotta@iastate.edu
Matt Eckes mweckes@iastate.edu
Jacob Mayer jdmayer@iastate.edu
Trevor Boone tdboone@iastate.edu
Jacob Schulz jschulz@iastate.edu

TABLE OF CONTENTS

1 LIST OF FIGURES

2 DEFINITIONS

Term	Description
HDD	Hard Disk Drive
HDD Jig	Device used to hold the HDD in place
PCBA	Printed Circuit Board Assembly
Oscilloscope	Device used to view electrical signals
LCD	Liquid-Crystal Display
LED	Light Emitting Diode
HTTP	Hypertext Transfer Protocol
G-Code	Numerical control programming language used in the controlling automated machine tools
Gerber Files	The standard used by printed circuit board industry software to describe the printed circuit board images: copper layer, solder mask, legend, etc.
Drill Files	Defines hole locations, tool numbers and finished hole sizes (X&Y coordinates) for PCBA design
Arduino	A single board microcontroller
Raspberry Pi	A single board computer
RAMPS	RepRap Arduino Mega Pololu Shield
UI	User Interface
GUI	Graphical User Interface
Stepper Motor	Motors which operate only in discrete increments of rotation
Carriage	The moving middle assembly on the x-axis of a RepRap which holds the probe
Via	Electrical connection between layers on a circuit board
GPIO	General Purpose Input/Output

3 PROJECT DESIGN

The design approach is laid out in this section. The overall system design and network structure including the individual components are described in this section. The reasoning behind each design choice will be explained.

3.1 SYSTEM OVERVIEW

The user will manually place the 2.5” HDD into the secure jig and then plug in the HDD development electrical harness. The user will then access a control web page through the HGST Intranet using a web browser. Once on the page, the user will next open a drill file in the web application, the web application will send the file to the Raspberry Pi web server, and the file will be parsed on the server. Parsing the drill file will find all via locations and generate a list with each location.

Next, the server will return the populated list of via locations to the web application. The web application will plot each via location onto a HDD template on the GUI. Once the test points are displayed, the user can then proceed to click and select a via to test. On selection, the web

application will send a message to the web server to move the probe to that via and test it. The server will then take the coordinates of the selected via and transform them according to our calibration method. The transformed coordinates will be used to generate the corresponding G-code commands that move the probe to the selected via. These G-code commands are then sent from the web server to the Arduino. The Arduino converts the commands into the corresponding waveform and then sends the signal to the RAMPS 1.4 motor board. Based on the waveform, the RAMPS board sends power to the stepper motor such that it is moved appropriately.

Throughout this process, the Raspberry Pi will activate status LEDs that indicate the state of the robot. For example, a red LED will be activated while the robot is in motion. Additionally, the current coordinates of the oscilloscope probe are displayed on the LCD by the Raspberry Pi.

Below are images that summarize the above description and overall design of the project. The first image is a block diagram that describes how each major component in our design from a communication viewpoint. The following images show concept sketches that visually describes our concept and shows the major components of our design from a user and system viewpoint.

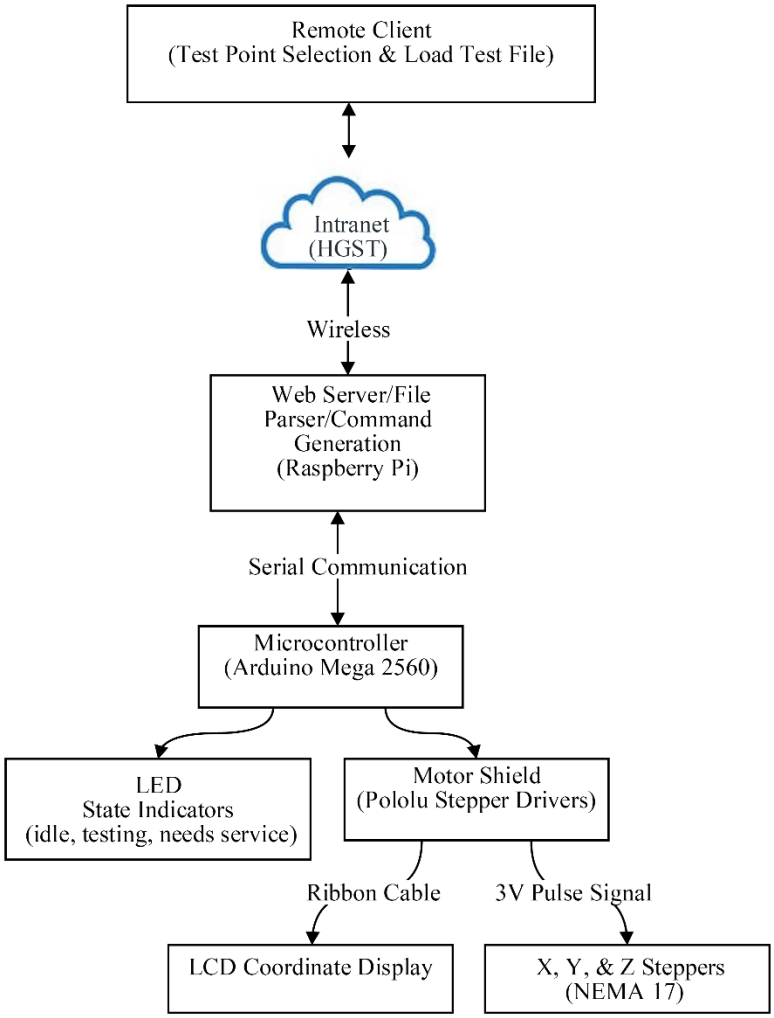

Figure 1: System Diagram Block Overview

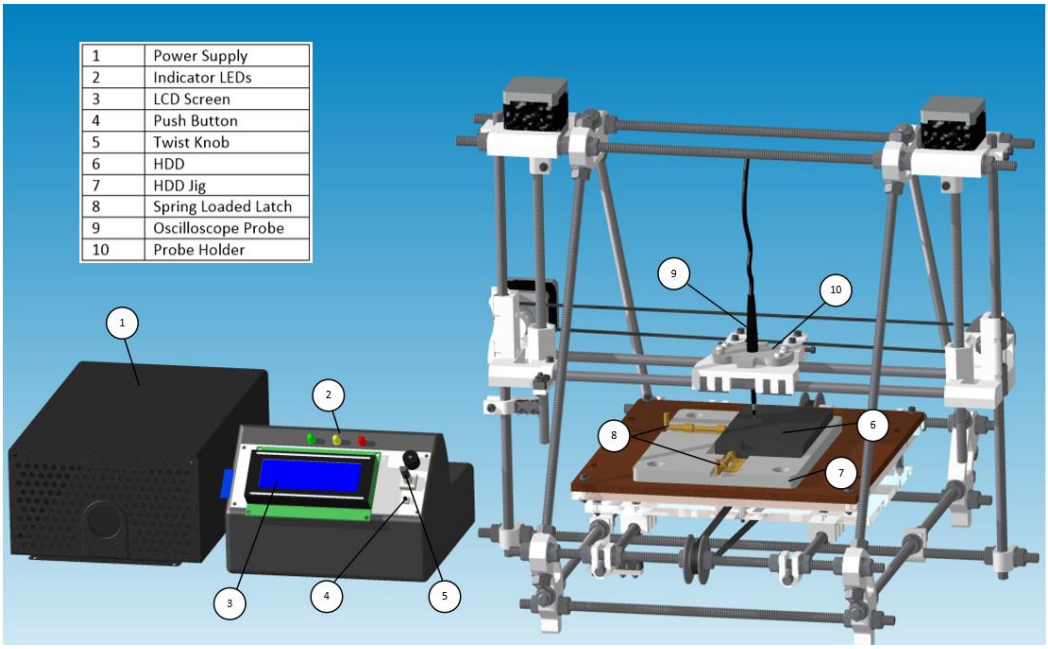

Figure 2 Concept Sketch

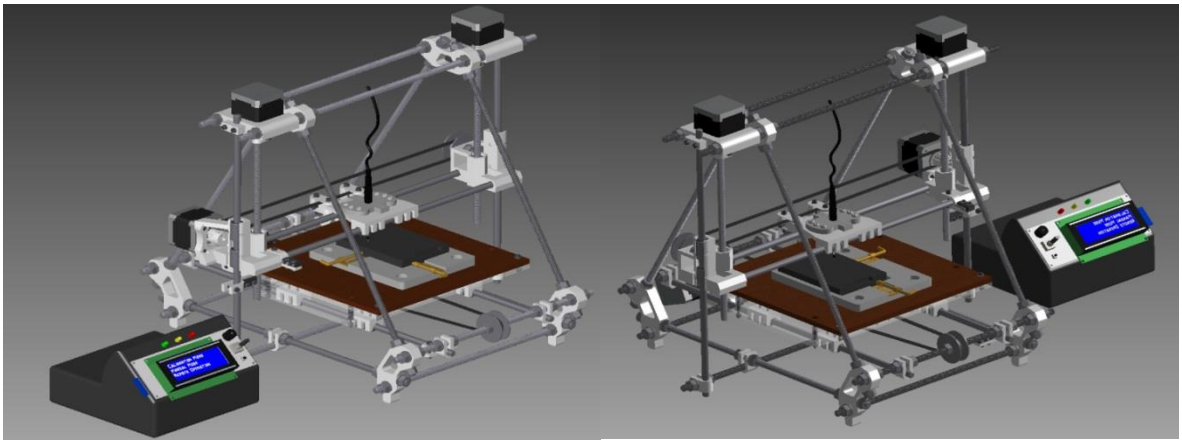

Figure 3: Images of robot LH and RH sides

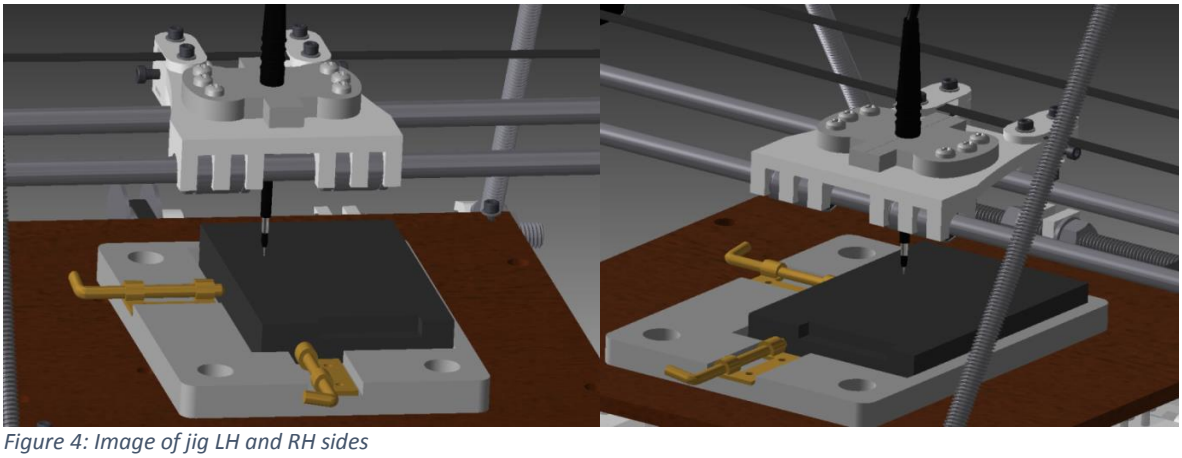

Figure 4: Image of jig LH and RH sides

3.2 HARDWARE

The hardware section includes all components for communication, computation, data processing and motor control. This includes the web server, microcontroller, motor driver, and LCD display.

3.2.1 Raspberry Pi

The Raspberry Pi (Pi) will be the web server that communicates with both the web application and the robot; it is the proxy for the two components. It will be responsible for managing and handling commands from the client, in addition to generating and communicating G-codes to Arduino. The Pi will communicate with the web application over the internet within the HGST intranet. It will connect to the HGST intranet wirelessly via Wi-Fi provided by a dongle that is plugged into a USB port on the Pi. Communication between the Pi and the Arduino is over USB. Power to the Pi is converted from wall power to a 5V 2A DC signal provided over micro-USB.

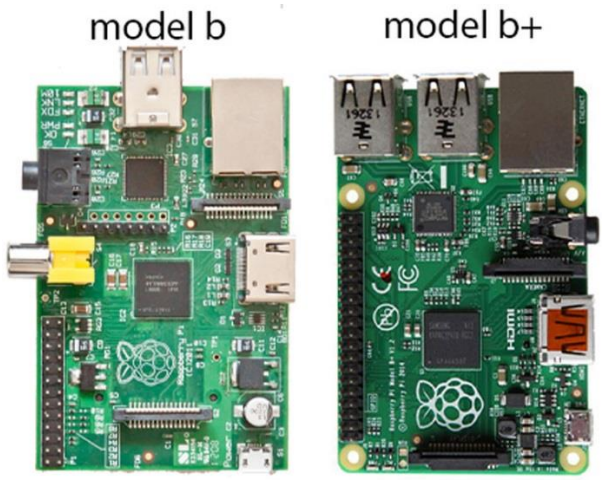

Figure 5: Two models of the Raspberry Pi. Model B is used in this project.

3.2.2 Arduino with RAMPS Motor Shield

The motors on the robot will be controlled by an Arduino Mega 2560 with a RAMPS 1.4 motor shield connected to it through the GPIO pins. The RAMPS board will have stepper drivers to provide the current that is needed to drive the stepper motors for the X, Y, and Z axis. The RAMPS motor can support up to six stepper drivers. The Arduino is responsible for converting G-code from the web server to the waveforms that will correctly move the stepper motors. Power to the Arduino is provided over USB. Power to the motor shield is provided by an external power supply that is capable of providing 360 watts.

Figure 6: Arduino Mega 2560 with and without RAMPS 1.4 shield on top

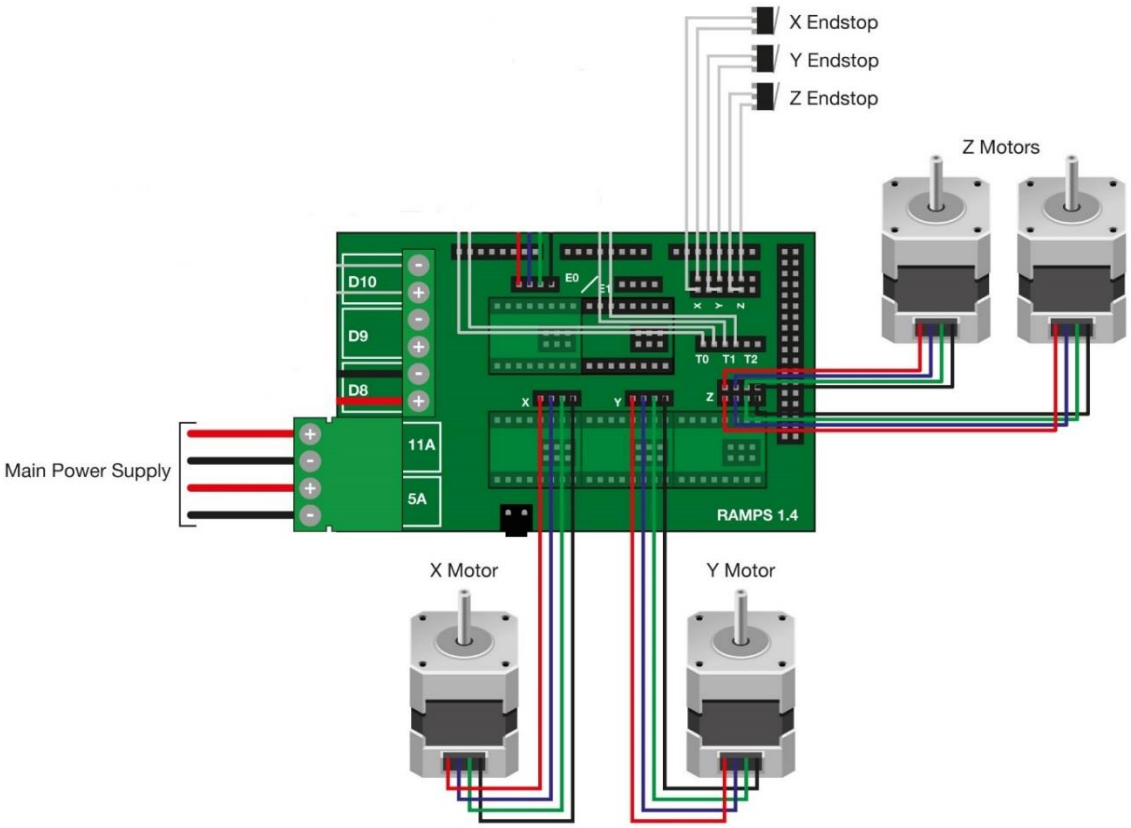

Figure 7 Ramps 1.4 Wiring Schematic

3.2.3 LCD

A Reprap Discount SmartController LCD will be connected to the Arduino for the local user. The LCD comes with a combination push button and rotary encoder as well as a small speaker. The LCD will be used to display error messages and the current coordinates of the probe. The combination push button and rotary encoder will expose an input method for controlling the robot. The local user will use this control input for manual calibration. The LCD is connected to the Arduino through the GPIO pins on the motor shield, which are connected to the GPIO pins on the Arduino. Power is supplied to the LCD by the Arduino.

Figure 8: A RepRap Discount SmartController LCD

3.2.4 LED Status Indicators

LEDs will be connected to the Raspberry Pi to indicate the operational status of the robot. There are three colors: green, yellow, and red. Only one LED is lit at a time. When the green LED is lit, the robot has not been used within five minutes, and so the robot is not in use. If any commands are sent to the server from the web application, the activity is noted and the yellow LED is lit to caution nearby users. Finally, when the robot is powered and moving, the red LED is lit, indicating that local uses should not presently interact with the robot. The LEDs are powered, controlled by, and connected to the Raspberry Pi. The LEDs are mounted in an easily visible location below the LCD.

3.3 MECHANICAL DESIGN

The mechanical design is laid out in this section. Of the listed components, some are externally designed/ fabricated components and others are custom designed components created by our team. The need for the company to have readily available parts was considered in the design process.

3.3.1 3D Printer

In order to achieve the accuracy and precision needed to complete the task and still manage to stay within budget we are leveraging existing 3D printing technology. The robot is able to control the probe in all three Cartesian axes with high accuracy. A third-party 3D printer design was chosen for the robot, as opposed to custom-designed components, due to the group's limited knowledge of mechanical design. The RepRap Prusa Mendel i2 was chosen because it is fully open source, meets the accuracy requirements, falls within the given budget, and has the most third party support out of all 3D printing platforms.

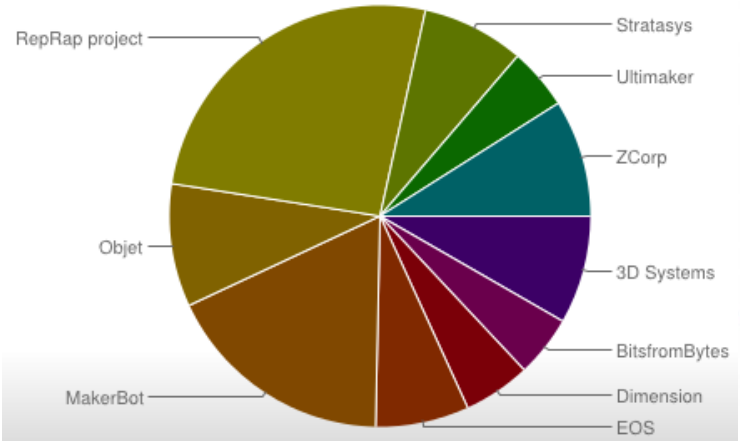

Figure 9 3D Printing Platform Use

Source: Moilanen, J. & Vadén, T.: Manufacturing in motion: first survey on the 3D printing community, Statistical Studies of Peer Production

Figure 10 RepRap Prusa i2

Source: NWRepRap.com, 2013

3.3.2 HDD Jig

The jig will hold 2.5” HDD in the recessed rectangular area of the jig. Spring barrel latches are used to apply pressure on the HDD from the bottom side (where the harness connects) and the side not touching the recessed wall. The purpose of the latches is to hold the HDD in one corner of the recessed rectangle and prevent any movement when testing. The whole jig is fastened down to the X-axis platform using three M8 bolts. The jig was fabricated out of 3/8” aluminum stock which will allow proper grounding of the oscilloscope to the HDD chassis.

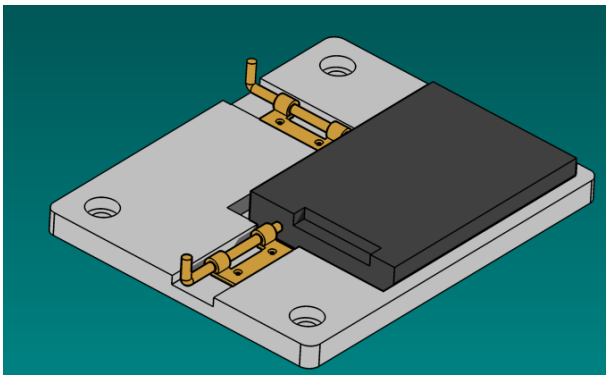

Figure 11 HDD Jig Design

Figure 12: Spring Barrel Latch

3.3.3 Probe Holder

The probe holder securely holds the probe during testing. The holder itself is fastened to the 3D printer carriage. To avoid damaging the probes and the HDD, the probe is equipped with a spring loaded tip. The holder was fabricated via 3D printing out of ABS plastic.

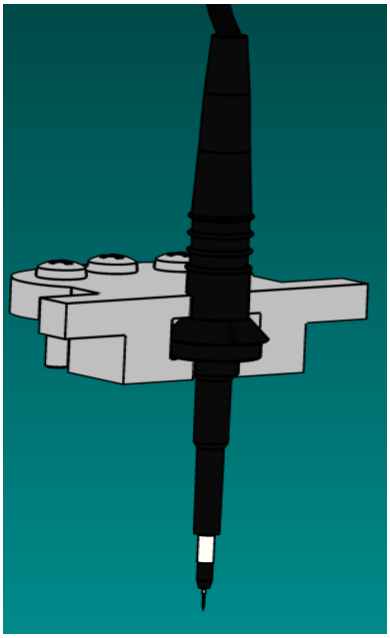

Figure 13: Probe Holder

3.4 SOFTWARE

This section describes the design of the software. A diagram of the high-level structure is provided below, which is comprised of several stages between the interface presented to the client and control of robot's hardware. Rationale for the inclusion and modification of each module is discussed in addition to their details. Each module could have existing software and/or custom pieces.

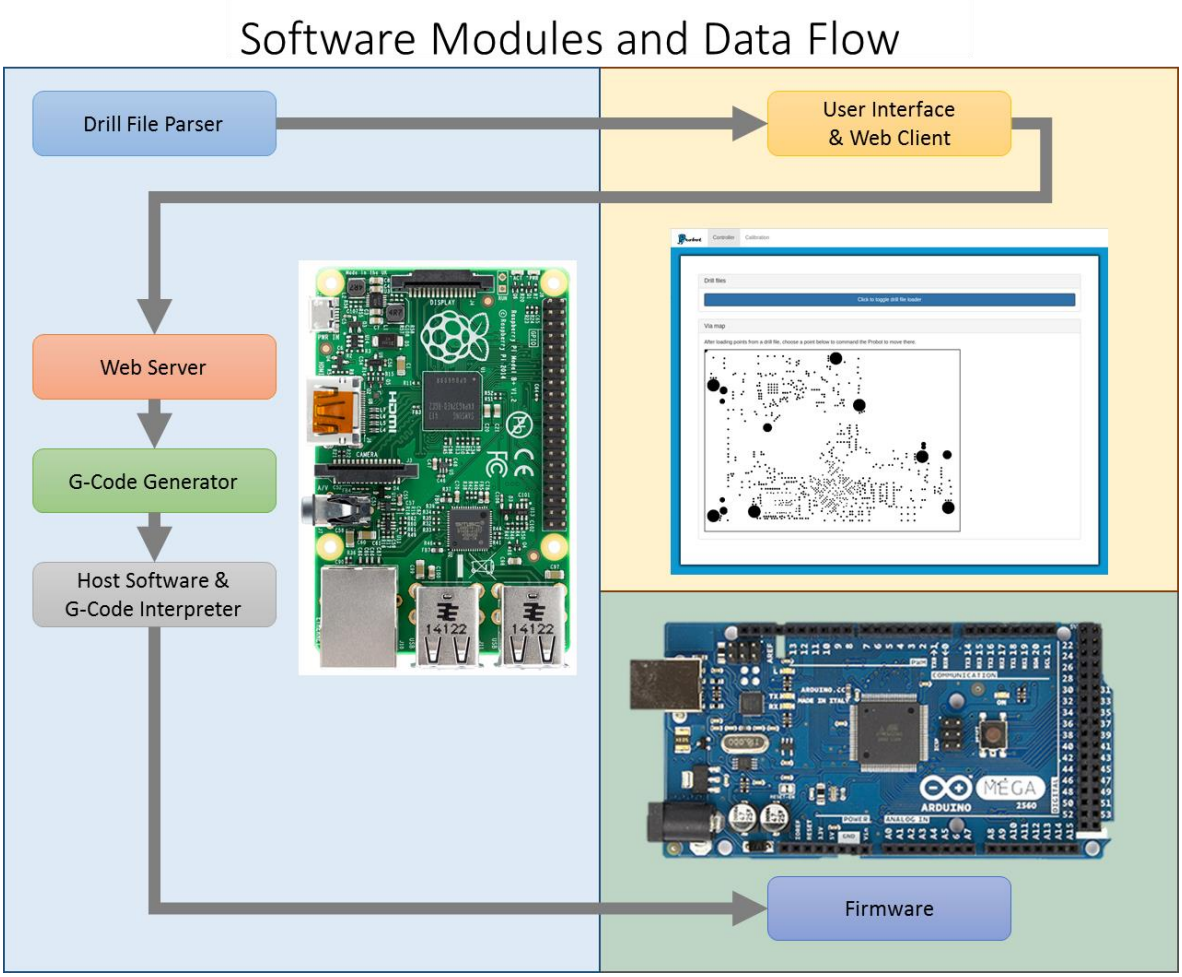

Figure 14: Software Data Flow Block Diagram

3.4.1 Interface

The user interface for the remote user is a web application. It is served off of a computer that is connected by a serial connection to the hardware controller. A web interface has an advantage over native apps in the fact that there is already rich ecosystem of tools to help develop web apps. This allows for a solution that is cross platform and does not require the user to install any extra software on their computer. The web applications uses HTML/CSS for layout and styling of the GUI. The functionality of the GUI, such as loading, displaying, and selecting points, are implemented using JavaScript.

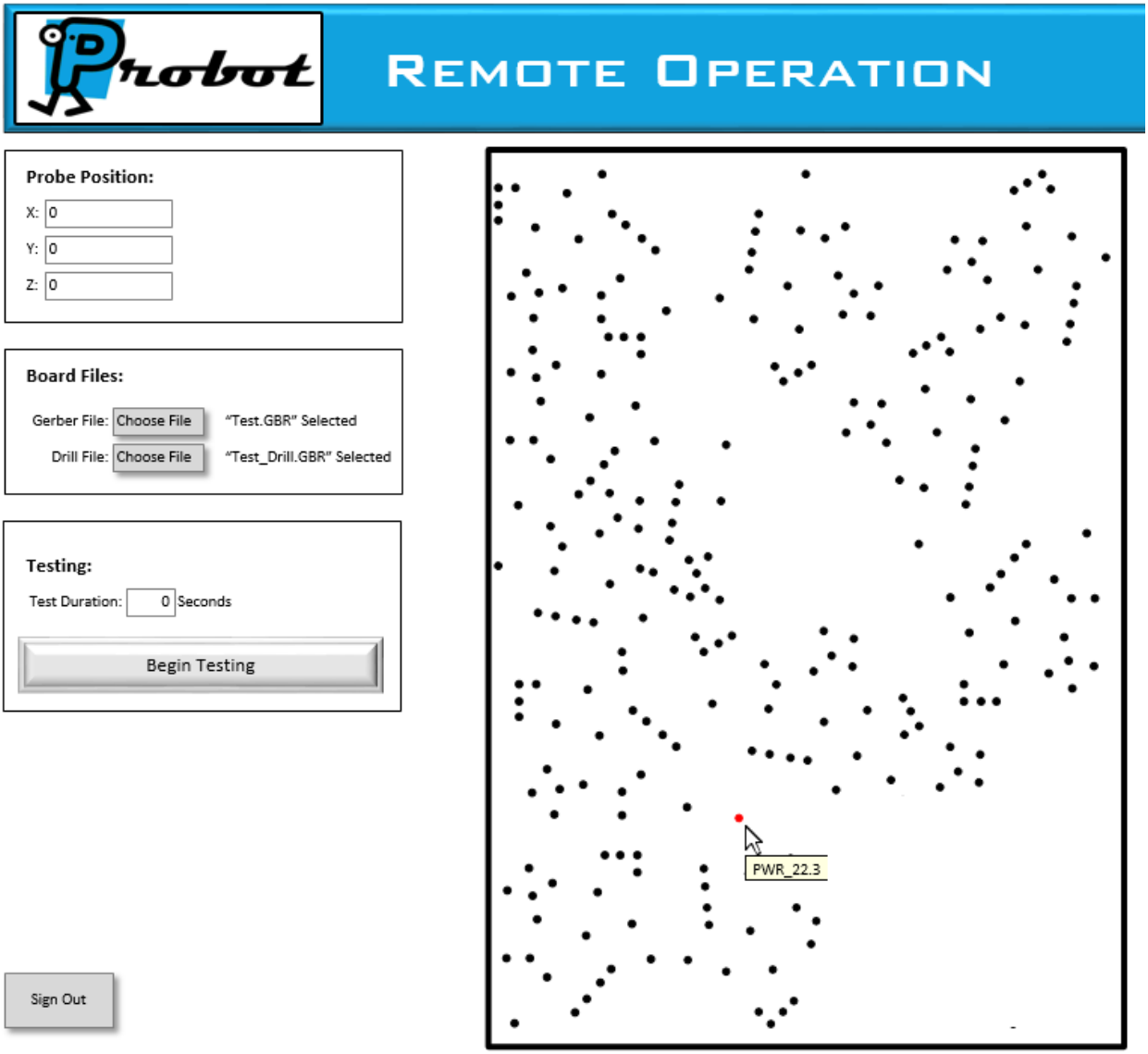

Figure 15: GUI Mockup

3.4.2 Server Design

To serve a web application, we needed a web server to serve the webpages. We decided to have the backend of the web server in Python because it allowed us access to existing libraries such as the Tornado web server library, which was chosen because it is lightweight and allows for asynchronous handling of requests. It is also designed for small projects and allowed us to efficiently develop the relatively small backend needed for this project.

3.4.2.1 File Storage/Parsing

To parse a drill file, the user will use the web application to upload a file to the server. The file is stored on the server. When requested, the file is parsed on the server and the points are sent to the web application. The code to do this is written in Python, and the points are transmitted to the web application in a JSON format. We included file parsing into our program because it removes the need for our users to manually get the coordinates of test points from another program. Drill files are stored on the server so that it is not necessary to upload the files each time it is needed, and also so files can be shared between users.

3.4.2.2 Long Polling

In a traditional client-server system, clients can always post data to a server but the server can only send data back when the clients request it. However, because operations on the machine will take some time to complete, and there could be multiple clients connected to the server at the same time we needed an asynchronous way to send messages between the server and the client. We chose long polling as the solution for this problem. Long polling is a technique where the client will keep sending requests to the server, but the server will only respond to them if there is new data. By doing this, a connection to the server is always open and the server can send messages back to the client at any time. This asynchronous messaging system also makes it easier to have more clients as you can ensure that every client gets the same message. Long polling was implemented by our team in Python.

3.4.3 Client Interface Design

The overall goal of the GUI is to make it simple to choose a point on the HDD to test. To accomplish this task, our implementation has two major parts: the files management and the selection of test points. Both of these functions are implemented in JavaScript combined with asynchronous requests to the server.

3.4.3.1 Point Selection

Point selection is the main functional component of the GUI and was therefore one of our top priorities. To facilitate this, we plotted selectable points on a 2d representation of the HDD circuit board. Once a point is selected the client will send the point in a post request to the server and the server will execute the move accordingly. With the points being sometimes just a few pixels across it is sometimes difficult to click exactly on a point so we select the closest point to where the click was. To find the closest point efficiently we partitioned the points into a grid corresponding to 5mm by 5mm sections of the hard drive and we then do a breadth first search from the clicked grid square until a point is found and all unsearched squares are too far away to contain a closer point.

3.4.4 G-Code Generator

In the 3D printing world, where the RepRap originates, G-Code is usually generated by a separate utility. The input to this utility is a file (usually a STL file) which describes a dimensioned 3D model. This file is typically created by an engineer using CAD software. Using this file, the model would be translated into a long list of G-Code commands for the printer.

Our project, however, does not get its input from a 3D model. Instead, we generate the G-code ourselves. The list of G-codes are generated dynamically based on commands send by the client and additional processing on the server. To generate the G-codes, our team used a combination of custom code and an existing library called MeCode. The custom code handles the project-specific modifications to the original printer, like backlash support and calibration. MeCode handles translation of high level motion commands, such as “move from point a to point b”, into a series of G-Code instructions. In addition, MeCode also tracks the robot’s position over time.

3.4.5 Host Software & G-Code Interpreter

Once G-Code instructions have been generated, they are submitted to another module on the server called “host software”. The job of the host software is to provide machine configuration (sometimes through a GUI), manage a G-code queue, and supervise jobs while they execute. The host software is also responsible for communicating with the microcontroller over a serial connection. This functionality is necessary in order to send G-Code to the firmware and read the statuses of hardware sensors. The different choices of host software are given below. We chose to use Pronterface as our host software because of its modern support, helpful utilities, and compatibility with the firmware. It lives on and communicates with the server and is written in Python.

Name	Description
Pronterface	Pronterface is the visual host by Kliment. It features an intuitive user interface, a 3D model to G-Code translator, and track separate G-Code groups.
RepSnapper	Fast, barebones host program. It has a fast, simple G-Code generator and focuses on providing easy customization.
Replicator G	Well-rounded compared to the others. Also has a 3D model to G-Code translator built in as well. Designed primarily for 3D printers, so many of its special features aren’t of use for this project.
Repetier	Has a simple interface to two different 3D model to G-Code translators (which have different strengths depending on the type of model).

3.4.6 Firmware

The firmware module has the responsibility to interpret G-Code and control the hardware. It is tailored to the specifications of the hardware and is able to take the abstract G-Code instructions sent from the host software, incorporate calibration details specific to the hardware, and generate the necessary electrical signals for manipulating the hardware (e.g., moving motors). Again, there are several options of firmware available for the RepRap 3D printer. The choices are listed below. We chose Marlin because of its calibration abilities, customization, accuracy, and safety features.

Name	Description
Sprinter	Simple to setup and start using. Simple, first-time calibration. Support for most electronics and has acceleration control.
Marlin	A feature-extension to sprinter with more control over calibration, motor control, safety features, and better customization. The downside is that it is more complex to set up.
Teacup	Smallest firmware size, so it can fit on smaller Arduinos than any other firmware. Has no dependencies with Arduino libraries. However, these properties mean a reduction in features.
Repetier	Built alongside its host controller sibling for an all-in-one solution. However, is incompatible with other host controllers. Communication between host and firmware is more robust and reliable.

3.5 CALIBRATION

With 3D printing, calibration of the machine is simplified because the models can start anywhere within the platform and only the relative changes in position matter. With our project, we will be securing and grounding a HDD to the platform, so we need to be able to consistently match the origin of our machine with the origin of the HDD PCBA. Without this, the machine will fail to read the contacts on the HDD correctly. This requires a more complex and robust form of calibration.

We took advantage of the PCBA board manufacturing accuracy to define an accurate origin to base a coordinate system off of. Through manually moving the probe to a specific point and then identifying to robot which point the probe is really at, the software will be able to compute calibration coefficients using the data from the board files. Any via can act as the origin, however, an ideal point is a small via close to the center of the HDD. The accuracy of test point coordinates is increased using a small via because they have more significant digits documented using the board file locations. In addition any variation unaccounted for in aligning the two coordinate systems will be expounded the further the probe is from the origin.

3.6 STANDARDS

3.6.1 User Interface

This user interface utilizes a myriad of web standards that are standard to websites. It is built off of the combination of HTML, CSS, and JavaScript that make up the backbone of the current web. HTML is used to create all of the objects that are seen and used by the client while the layout and style of these clients is specified by the CSS. While HTML and CSS are used to specify the view we used JavaScript for user interactions with our webpage and also interactions with the server. All of these web technologies were passed in files from the server to the client by utilizing standard HTTP over TCP/IP.

3.6.2 G-code

To communicate with the firmware we sent it dynamically generated G-code. G-code is a standard language widely used in industry for controlling different machine tools. By using G-code, we can use the same software systems to communicate with any firmware or robot that understands it. Additionally, by using the standard we benefit from the design, knowledge, thorough testing, feature-rich tools, and support that revolve around G-code.

4 IMPLEMENTATION DETAILS

4.1 FINAL PRODUCT

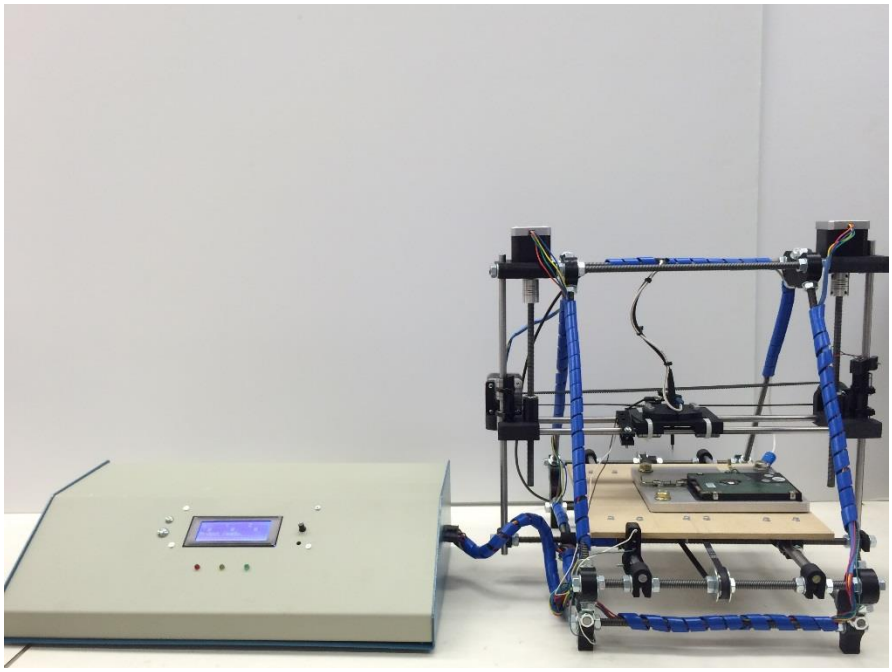

Figure 16: An Overview of the Final Probot System

Figure 17: A Close Up on the Enclosure containing the server, controller, LCD, and power supply.

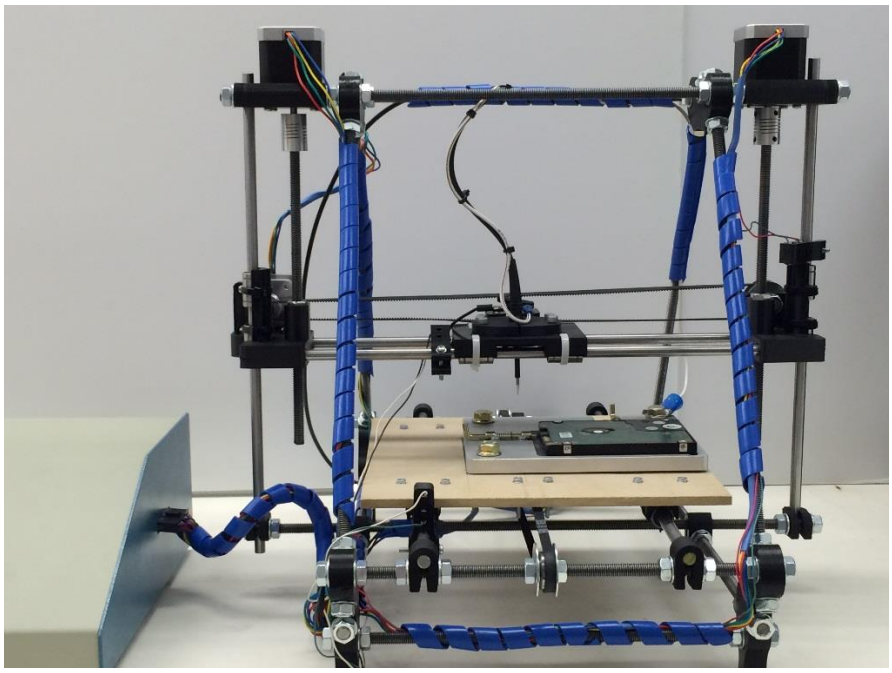

Figure 18: A Close Up on the Robot

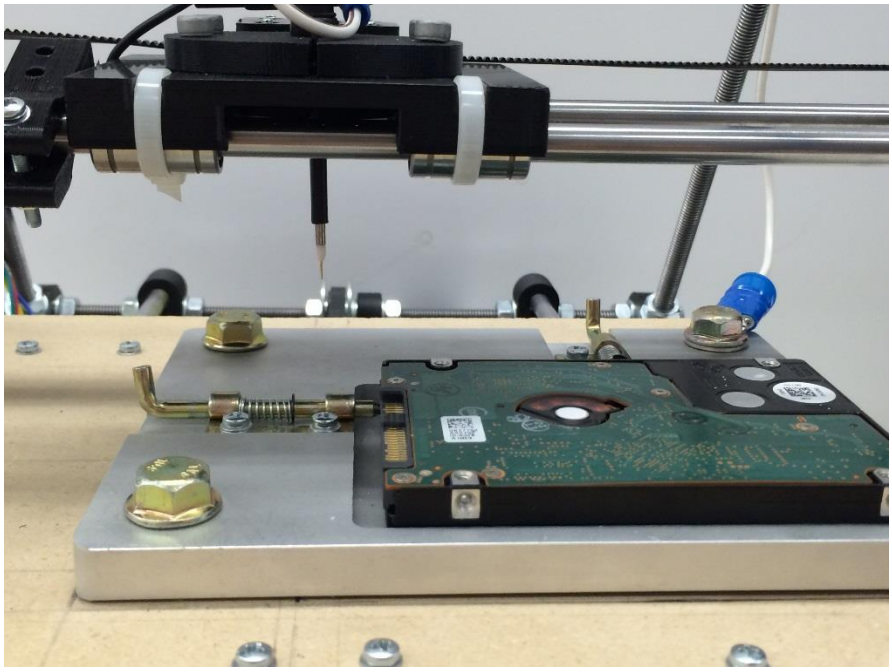

Figure 19: A Close Up on the HDD Jig and Probe Holder

4.2 ENGINEERING DRAWINGS

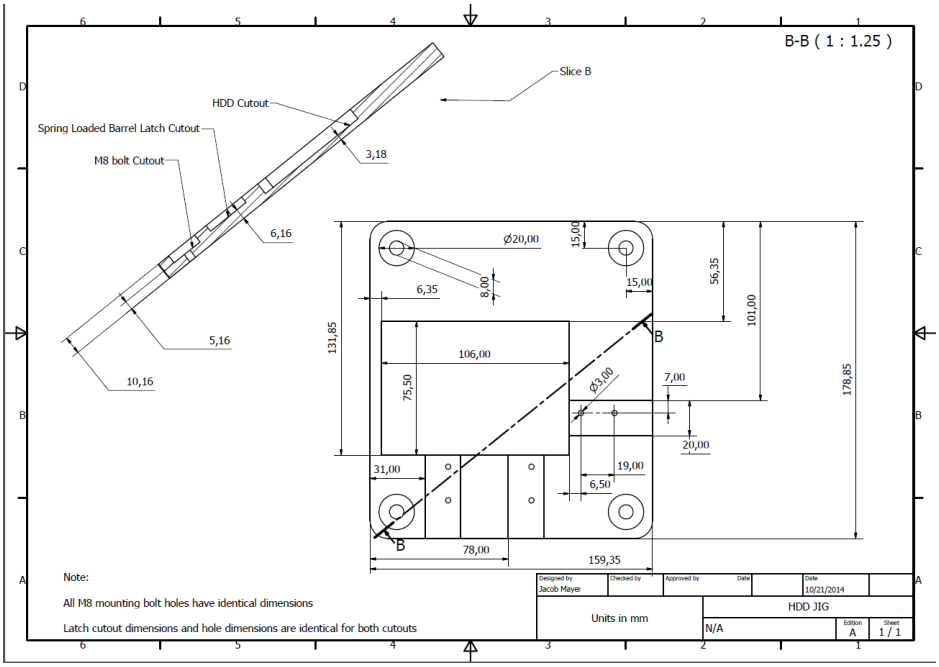

Figure 20: Engineering Drawing for Jig

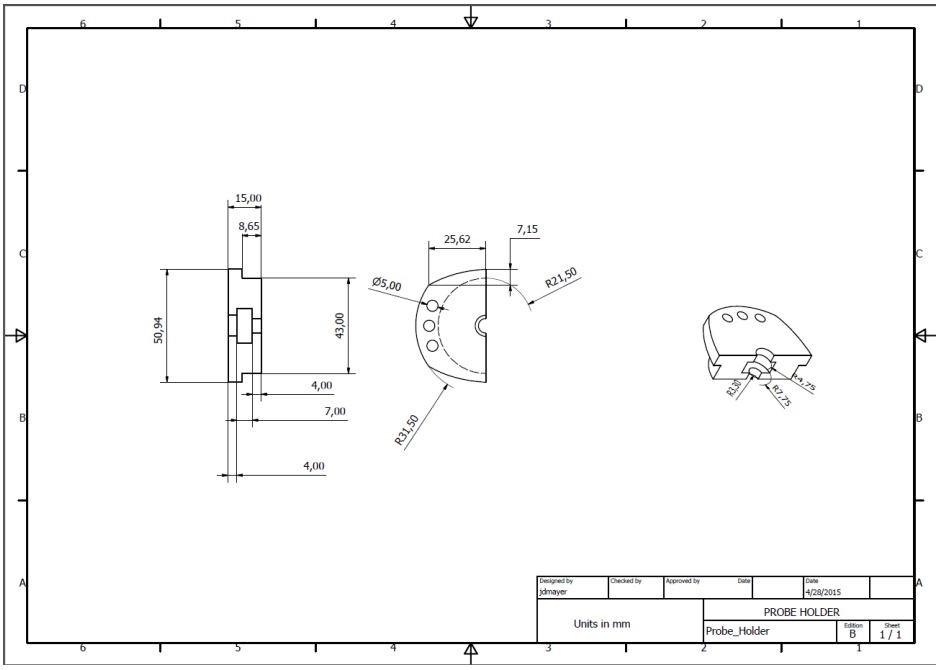

Figure 21: Engineering Drawing for Probe Holder

5 TESTING AND VALIDATION

5.1 SOFTWARE COMMUNICATION

Without communication between the client and the server, the user's command will never make it to the robot. To test that the communication module is working correctly, first a connection will be obtained and verified. From there, the client and server agree to go into test mode in which each client command will be sent to the server, the server will note the command, and return a successful status to the client. This will ensure that communication between the two is correct and that the server can handle any requests the client is capable of sending.

5.2 FILE PARSING GERBER FILES

The success of this project lies in the ability to accurately locate test vias on a HDD. If we are unable to accurately locate the test points with regard to a reference point then the whole system fails. Due to the importance of this step, we must be able to show that the GUI accurately represents the test vias on the HDD. After the software generates a representation of the HDD with the test points located, the coordinates will be compared to the locations given in both the board file and the measured values.

5.3 MOTOR CONTROLS

Testing of motor controls is an important step in ensure all components are working properly. Before significant time is spent in mounting the motors to the robotic system, we first would like to check that they are operating correctly. RepRap software has an application, Pintrun, that allows for motor commands to be sent to the Arduino, and then to the specified motor (X, Y, or Z). For this testing procedure, the motor control components (Arduino, RAMPS, 3 Nema 17 stepper motors) will be connected. Commands, sent through Pintrun, will control each individually and test for clockwise and counter clockwise motion. This test is simply to ensure that all motor controls components are working correctly and the Arduino motor pins are correctly identified.

5.4 X, Y, AND Z MOTION

Accuracy and precision in the X, Y, and Z direction is an absolute must for this project. The robotic system needs to extremely reliablein order to be useful. For this reason, extensive testing will be done to ensure both accuracy and precision. For this testing procedure, we will give the robot a command to travel in one cardinal direction. The movement will then be accurately measured using calipers. The actual moving distance will be compared to the expected value. To get a high degree of statistical importance a large sample size will be taken. This procedure will be done at a variety of lengths as well as compared against different speed ratings. Once the accuracy and precision are documented, we will be able to publish an optimum speed rating that will allow for accurate movement with limited traversing time.

5.5 Transition Time

A non-functional requirement the client expects is that the movement from one test point to another will take no more than 60 seconds. The accuracy of movement trumps this speed requirement, but if we can show that the robot can move quickly from one test point to another, the robots efficiency will greatly increase. For this test, we will create a random list of test points the robot must travel to. The time it takes the robot to move from one test point to another will be measured. Once a large sample size is taken the average will be published.

5.6 Locating Test Points

The final test of the system is a complete test of the devices capabilities. A HDD will be placed in the HDD jig. The HDD Gerber files will be parsed and the GUI will indicate where the test points are located. The user will then select randomly a test point to measure. The robot's performance will then be measured on the basis of locating the test point and making sound electrical contact with the test via. We will do a large sample size in order to simulate the debugging process a HGST employee may encounter.

5.7 Risks

- **Calibration:** It is crucial that the system stay calibrated as the user is likely to not be around the machine to fix the problem. Additionally, small calibration errors (< 1mm) cannot be tolerated as it would exceed minimum feature separation. However, our team has no experience with calibration methods and we don't know how well our equipment will cooperate in regards to both being calibrated and staying calibrated.
- **Accuracy:** The machine has to be accurate within less than 1 mm. This means that our algorithms, motors, and construction must meet strict tolerances that might be difficult to obtain.
- **Customization:** The foundational bulk of the work is going to be purchased in order to best utilize our time and improve the quality of our final solution. However, in order to build exactly what we need, there is some custom mechanical work that needs to be done. Due to the lack of practical mechanical engineering experience on our team, it is possible that problems will arise with our design that we did not foresee.

6 TESTING RESULTS

Transition time and locating test point accuracy were combined into one overall test that simulates how the robotic system is designed to work. A script was written to send eighty randomly generated test points to the server. The robot then transitions to each test point and makes contact with the via. The probe was connected to an oscilloscope so the signal can be viewed. A binary pass/fail was recorded to gauge whether or not the probe was able to make good electrical contact with the test via. The time to complete this test was recorded so that an average transition time between test points could be calculated. The results are summarized in the table below.

Trial	Number that made contact	Points tested	Accuracy	Time	Average time between points
1	67	80	84 %	21 m 30 s	16.1 s
2	69	80	86 %	21 m 25 s	16 s
3	71	80	89 %	21 m 05 s	15.9 s

The location of the vias tested were recorded and mapped out in order to view where on the HDD the robot had difficulty with accuracy. The figure below shows the region in which HDD test vias were missed.

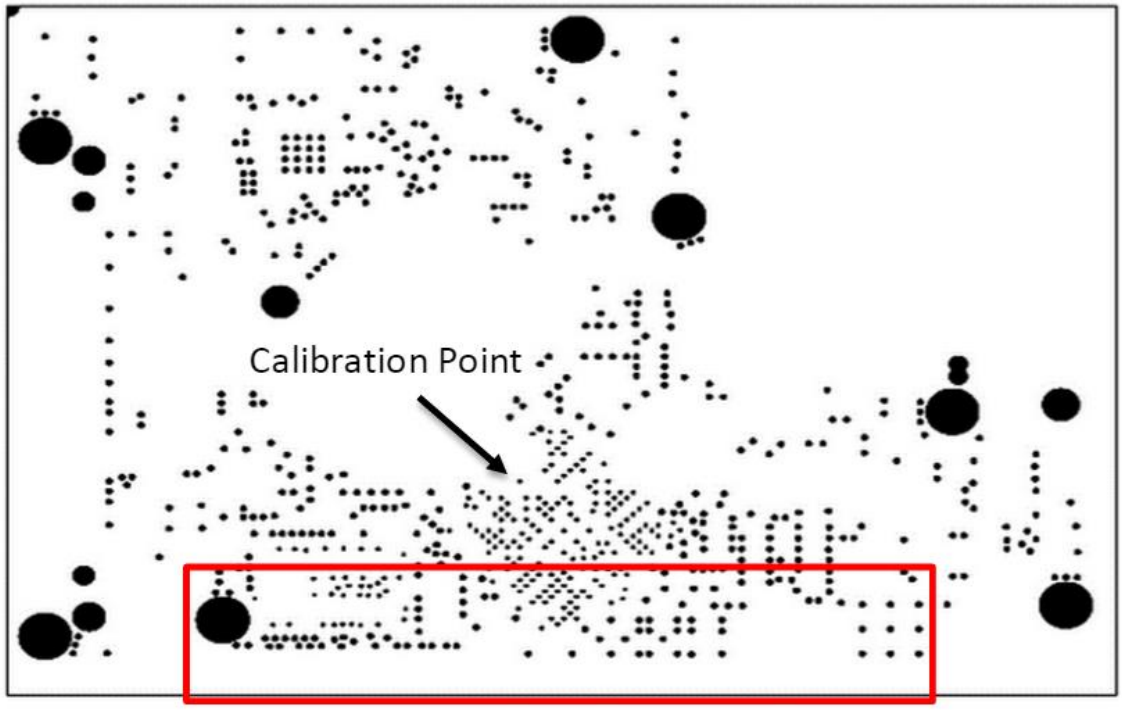

Figure 22: Missed Test Point Map

From the figure, we determined that a systematic error must exist in order for all the missed points to be localized on the HDD. Further investigation showed that the HDD jig has a design flaw that could account for the inaccuracy. The HDD jig design has bolts in three corners instead of all four. The lack of a bolt on this side of the jig causes the HDD to be elevated off of the robots platform. A new jig was designed, however due to a lack of time we did not manufacture and test this hypothesis. Our hope is that with a new jig the robot will be near 100% accuracy.

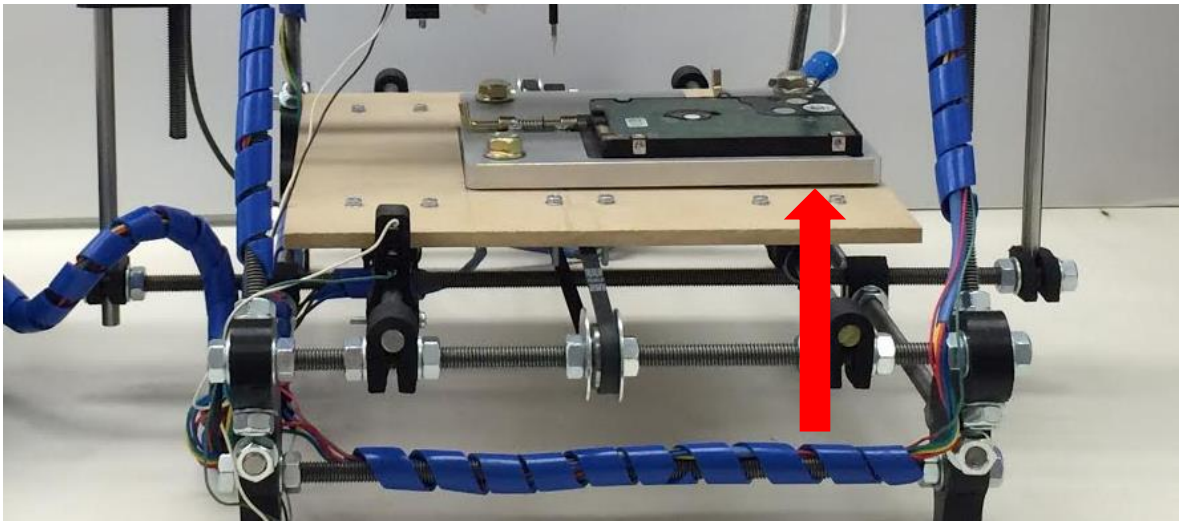

Figure 23: HDD Placement Error

7 APPENDIX I OPERATION MANUAL

7.1 SETTING UP THE SERVER

7.1.1 Outline:

1. Install Required Programs
    a. Install python
    b. Install git
    c. Install pip
2. Downloading code
    a. Clone git repository and submodules
    b. Install requirements
    c. Install submodules
    d. Adding necessary files

7.1.2 Installing Required Programs

The below commands will install the required programs.

```
pi@probot-pi:~/Desktop$ sudo apt-get install python2.7
pi@probot-pi:~/Desktop$ sudo apt-get install git
pi@probot-pi:~/Desktop$ sudo apt-get install python-pip
```

Python should already be on the Pi and the other two commands will download git (this will download our source code) and pip (which is used to download python packages). Note: The sudo password is necessary for the download of packages as directories used by apt-get require administrator privileges.

Often in installation users get the below error which most commonly occurs when the command was not run as root (i.e. the user didn't use the sudo command).

```
pi@probot-pi:~/Desktop$ apt-get install python2.7
E: Could not open lock file /var/lib/dpkg/lock - open (13: Permission denied)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you root?
```

7.1.3 Installing the Source Code

The below command will download the source into the directory specified by the second argument (the italicized argument is the name of the directory to create where the underlined portion is the name of the repository in which the code resides.

pi@probot-pi:~/Desktop$ **git clone --recursive repository *directory***

For example this command could look like this:

pi@probot-pi:~/Desktop$ **git clone --recursive https://github.com/TrevorBoone/May-15-03.git *Probot***

After the repository is cloned then navigate into the newly created directory

pi@probot-pi:~/Desktop$ **cd *Probot***

Once you are in the repository then the dependencies need to be downloaded. Pip install will install the dependencies from the python package index and the other two commands will install the submodules.

```
pi@probot-pi:~/Desktop/Probot$ sudo pip install -r requirements.pip
pi@probot-pi:~/Desktop/Probot$ sudo python mecode/setup.py install
pi@probot-pi:~/Desktop/Probot$ sudo python Printrun/setup.py install
```

All of the required packages should be installed and the installation is almost complete. Attempt to start the server with the below command:

pi@probot-pi:~/Desktop/Probot$ **sudo python WebMain.py**

It should output an error where the last line looks something similar to:

IOError: [Errno 2] No such file or directory: '/usr/local/lib/python2.7/dist-packages/mecode-0.2.1-py2.7.egg/mecode/header.txt'

Lastly we need to create this file. Simply take the filename (it should be the same as the above filename on most systems) and run the following commad:

pi@probot-pi:~/Desktop/Probot$ **sudo touch *filename***

So using the above error as an example it would be

pi@probot-pi:~/Desktop/Probot$ **sudo touch /usr/local/lib/python2.7/dist-packages/mecode-0.2.1-py2.7.egg/mecode/header.txt'**

As you can see this creates a file called header.txt in the mecode directory. We also need to create a footer file. This can be done by replacing 'header.txt' with 'footer.txt' in the above command. Using the example this leads to the command:

pi@probot-pi:~/Desktop/Probot$ **sudo touch /usr/local/lib/python2.7/dist-packages/mecode-0.2.1-py2.7.egg/mecode/footer.txt'**

From here the server should be fully configured and able to be run using the command:

pi@probot-pi:~/Desktop/Probot$ **sudo python WebMain.py**

7.2 Generating A Drill File

1. Open Cadence Allegro PCB Designer.
2. Open the board file corresponding to the HDD you wish to test. Once the board file is imported it should look similar to *Figure 2*.

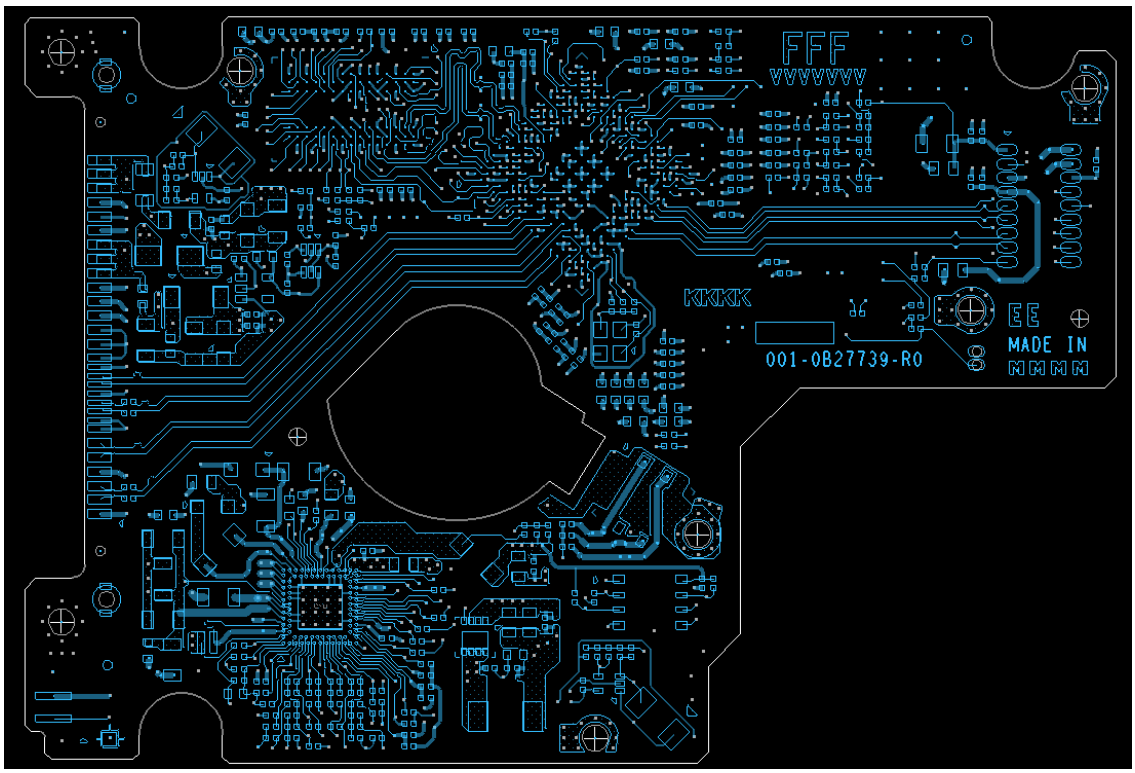

Figure 24 Imported board file in Cadence Allegro PCB Designer.

3. Navigate to the “Manufacturing” tab in the top tool bar and then select “Artwork.” Then click on the “General Parameters” tab and the following window should be displayed.

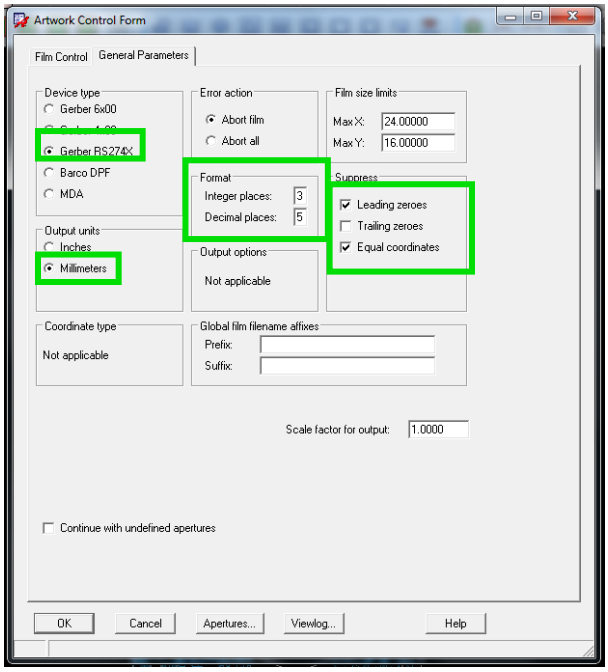

Figure 25: Artwork - General Parameters - pop up window

4. Ensure your options match the highlighted options shown in *Figure 3.*
   a. Millimeters
   b. 3 Integer Places and 5 Decimal Places
   c. Gerber RS274X
   d. Leading Zeros and Equal Coordinates
5. Next select “OK” when you are completed.
6. Next navigate to the “Manufacturing” tab in the top tool bar and then select “NC.” Then select the “NC Drill” option, the following window should be displayed.

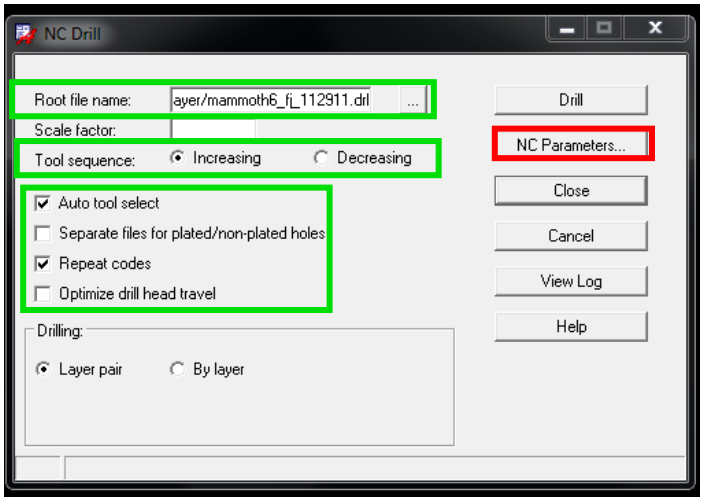

Figure 26: NC Drill pop up window

7. Ensure your options match the highlighted options shown in *Figure 4.*
    a. Auto Tool Select and Repeat codes
    b. Increasing Tool Sequence
    c. ***Important:*** You cannot export a drill file to a network drive. Make sure it is saved to your C:\ drive.
8. In the "NC Drill" window next select "NC Parameters. " The following window should be displayed.

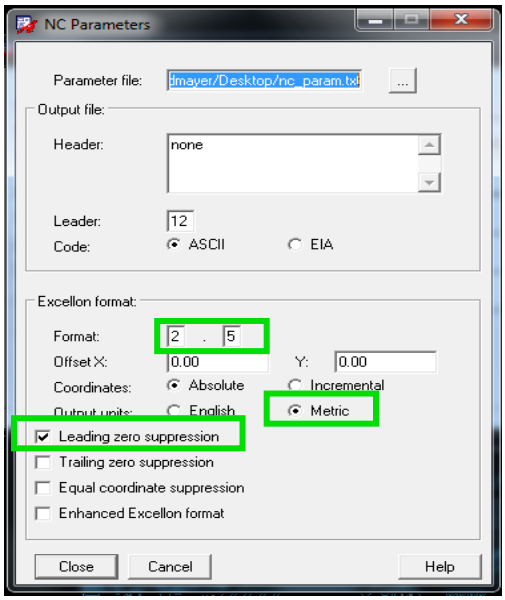

Figure 27: NC Parameters pop up window

9. Ensure your options match the highlighted options shown in *Figure 5.*
    a. 2,5 Format
    b. Metric Units
    c. Leading Zero Suppression
10. Next select "Close" when you are completed.
11. Will you will be returned to the NC Drill original window. Click on the "Drill" button and the drill file will be exported to wherever you chose on your C:/ drive.

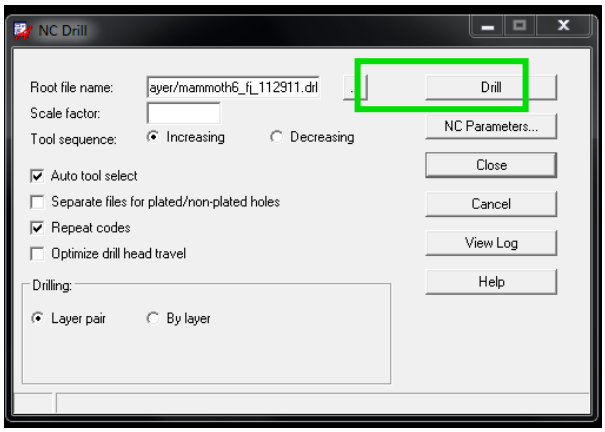

Figure 28: Generating the Drill file

7.3 CALIBRATION PROCEDURE

This procedure will cover how to calculate the rotational angle between the Hard disk drive (HDD) reference frame and the probes reference frame. Coordinates for the test points are provided by a drill file that references them to the corner of the HDD. When the HDD is placed into the jig the coordinates need to be translated into the robots reference frame in order for accurate placement of the probe tip. The following is a step by step guide for the coordinate translation.

1. Open Cadence and find the location of two test points (preferably far apart)

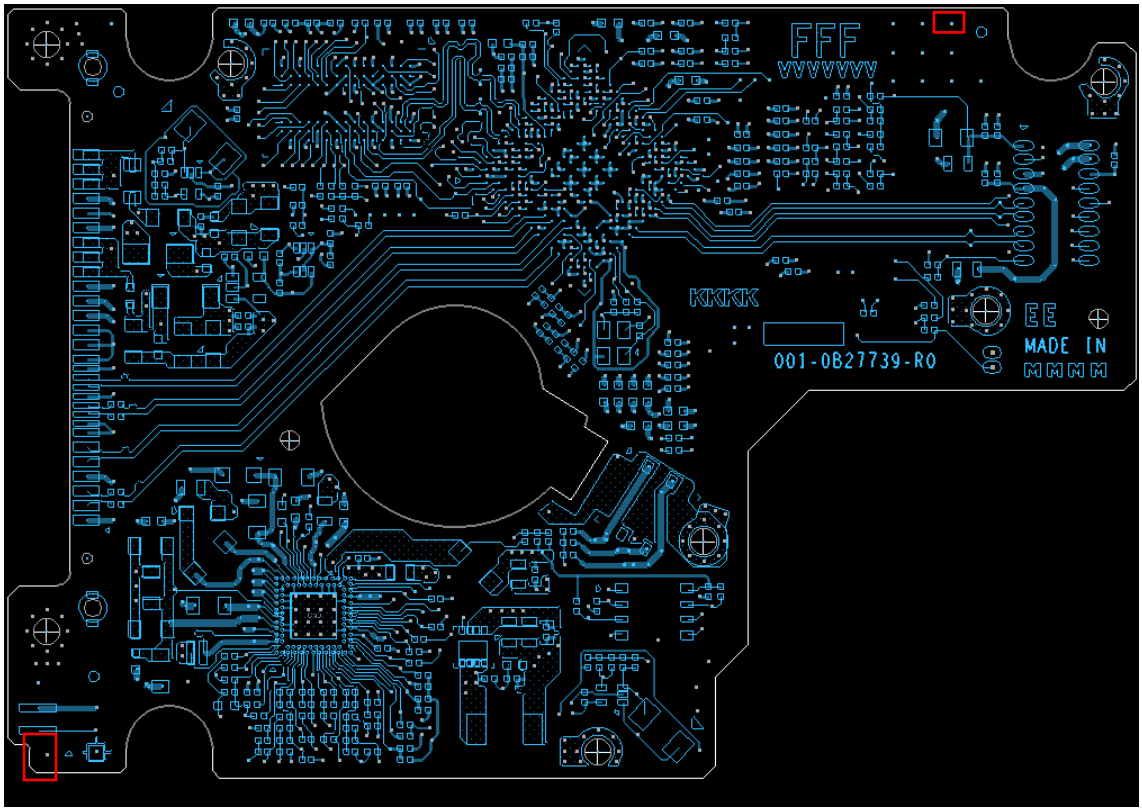

Figure 29: Picking Two Calibration Points

$$Pt1 = (3.302\ mm, 3.048\ mm) \quad Pt2 = (81.915\ mm, 66.6115\ mm)$$

2. Find the tangent between the two points.

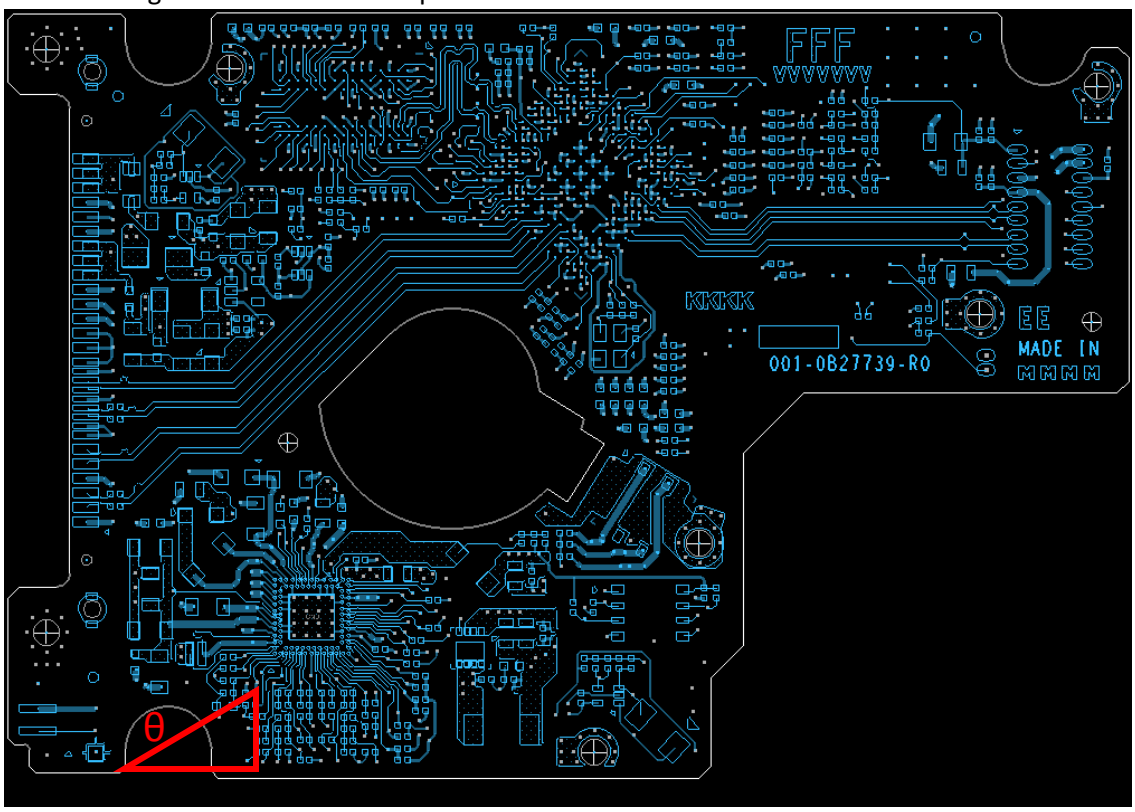

Figure 30: Finding Theta for Calibration

$$\theta_{HDD} = \tan^{-1}\left(\frac{y_1 - y_2}{x_1 - x_2}\right) = \tan^{-1}\left(\frac{81.915 - 3.302}{66.6115 - 3.048}\right) = 0.8909\, rad$$

3. Place the HDD into the HDD jig located on the robots platform.
4. Manually move the probe to the first test point and reset the LCD. This will set the tip location to (0,0).
5. Next manually move the probe tip to the second test point and note the location of the probe tip from the LCD display.

$$Pt1 = (0\, mm\, , 0\, mm)\;\; Pt2 = (65.21\, mm, 82.18\, mm)$$
$$\theta_{robot} = \tan^{-1}\left(\frac{x_1 - x_2}{y - y_2}\right) = \tan^{-1}\left(\frac{82.18 - 0}{65.21 - 0}\right) = 0.900\, rad$$

   a. Note the x and y are switched for the robot coordinate system.

6. Calculate the rotational angle

$$\theta_{rotation} = \theta_{HDD} - \theta_{robot} = 0.8909 - 0.900 = -0.009\, rad$$

The theta value can be implemented on the web site under the calibration tab.

8 APPENDIX IIALTERNATIVE DESIGNS

8.1 BED OF NAILS

This design uses a single, flat collection of pogo pins placed in pattern that matches the circuit board's vias. When it presses down onto the board, all pins come in contact with nodes. Individual pins are wired to another module that aggregates, measures, and gives the readings of the pins to the operator. This method has a long turnaround time for different layouts (programming and setup) and is very expensive.

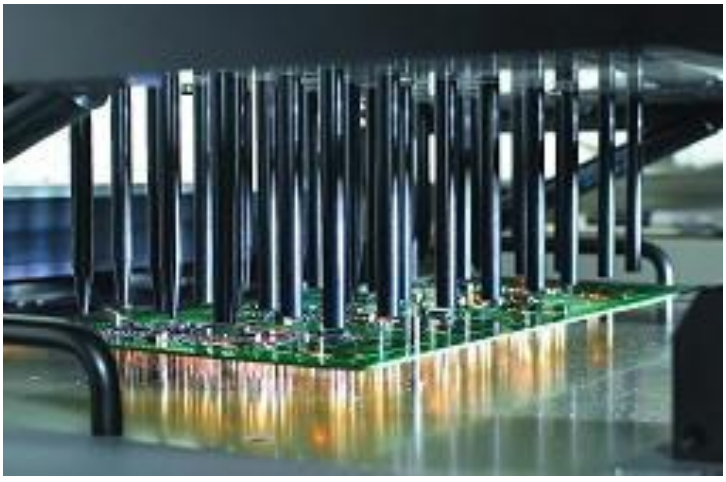

Figure 31: Bed of nails PCB tester

8.2 FLYING PROBE

This design uses a single pogo pin moved by an electromechanical system like a coordinate measurement machine. Simply moves the pin to the desired node on the circuit board and performs a measurement. It sacrifices the sheer performance of the bed-of-nails method for extremely quick turnaround (virtually no setup and no programming) and low cost. This was the basis of our design.

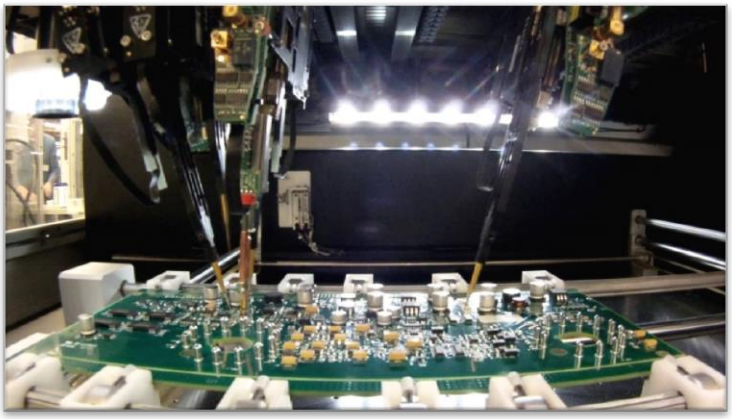

Figure 32 SPEA S2 Flying Probe Testers

8.2.1 Dual Probe

We thought about modifying our current design by allowing the HDD jig to rotate and adding another probe carriage on the existing rail. The rotation and translation of the jig can align any two vias of the hard drive with the probe axis. The two probes can then probe the two vias along that line. The probes would have to be angled in their holders in order to be able to probe neighboring vias. This would make it more difficult to be accurate, because the tip of the probe would move slightly horizontally as the spring was depressed. In the interest of time, we wanted to make sure our current design was working smoothly before adding the second probe.

8.2.2 Selective-Compliance-Articulated Robot Arms (SCARAs)

A version of flying probe that uses an articulating arm to move the probe to a via on the stationary circuit board. Allows for very quick movement from via to via and can be extended to multiple probes by adding more arms. The challenge here would be to translate Cartesian coordinates of the circuit board into angular coordinates of the robotic arm. Cartesian robots are easily reconfigurable and can use standardized parts, making them less expensive to obtain and maintain.

9 APPENDIX III OTHER CONSIDERATIONS

9.1 Z-AXIS DEBUGGING

A major issue that caused significant down time during the robot build was a malfunction in the z-axis motor. The symptoms of the motors was a binding, jerking motion, which caused significant amount of audible noise. The problem occurred after we completed a connection for all the motors into the enclosure. Continuity was established and did not seem to have a significantly higher resistance than the other working motors. At this point, we checked several different things including better z-motion alignment, separating the z-axis motors on two separate pinouts to provide higher current, and even applying grease to the linear rod for a smoother rotation. After approximately 30 hours comprised over a 2 week period we investigated the possibility of a bad motor. Instead of soldering the motor wires into the connection through the enclosure connector, we decided to go directly into the RAMPS pinout to save time. This fixed the issue, so we determined that the cause must have been a bad motor. Interestingly, when we went to solder the motor to the connection wire for the finished product the motor began to act up again. The conclusion we reached is that the culprit for the z axis motor malfunction was in fact the connector. We seemed so sure that this was not the problem due to continuity being established that we anchored ourselves to this belief. We learned that while debugging a system it is important to not rule anything out absolutely.

10 APPENDIX IV CODE

10.1 FIRMWARE

The firmware is a modification of the open source Marlin firmware.

10.1.1 Configuration.h

10.1.1.1 Hardware endstops

Change the following:

```
const bool X_MIN_ENDSTOP_INVERTING = true;
const bool Y_MIN_ENDSTOP_INVERTING = true;
const bool Z_MIN_ENDSTOP_INVERTING = true;
const bool X_MAX_ENDSTOP_INVERTING = true;
const bool Y_MAX_ENDSTOP_INVERTING = true;
const bool Z_MAX_ENDSTOP_INVERTING = true;
```

To the updated code below (all false):

```
const bool X_MIN_ENDSTOP_INVERTING = false;
const bool Y_MIN_ENDSTOP_INVERTING = false;
const bool Z_MIN_ENDSTOP_INVERTING = false;
const bool X_MAX_ENDSTOP_INVERTING = false;
const bool Y_MAX_ENDSTOP_INVERTING = false;
const bool Z_MAX_ENDSTOP_INVERTING = false;
```

10.1.1.2 Probot Configuration

Add the following towards the end (it works in the Additional Features section):

```
#define PROBOT
#ifdef PROBOT

  #define STRING_CONFIG_H_AUTHOR "Shawn LaGrotta and Jacob Schulz" // Who made the changes.

  #define DISABLE_MAX_ENDSTOPS
  #define min_software_endstops false // If true, axis won't move to coordinates less than
HOME_POS.

  #define HOMING_FEEDRATE {50*60, 50*60, 1*60, 0}  // set the homing speeds (mm/min)

  // default steps per unit for Ultimaker //78.7402,78.7402 //z=200*16/1.25 for prusai2
  #define DEFAULT_AXIS_STEPS_PER_UNIT   {100,100,2560,760*1.1}
  #define DEFAULT_MAX_FEEDRATE          {125, 125, 1, 25}    // (mm/sec)

  // X, Y, Z, E maximum start speed for accelerated moves. E default values are good for
  Skeinforge 40+, for older versions raise them a lot.
  #define DEFAULT_MAX_ACCELERATION      {9000,9000,0.01,10000}

  // X, Y, Z and E max acceleration in mm/s^2 for printing moves
  #define DEFAULT_ACCELERATION          500

  // X, Y, Z and E max acceleration in mm/s^2 for retracts
  #define DEFAULT_RETRACT_ACCELERATION  500

  #define DEFAULT_XYJERK                5.0    // (mm/sec)
  #define DEFAULT_ZJERK                 0.1     // (mm/sec)

  #define BAUDRATE 115200

  #define MOTHERBOARD BOARD_RAMPS_13_EEB//=34
  #define REPRAP_DISCOUNT_SMART_CONTROLLER
```

```
 #define CUSTOM_MENDEL_NAME "Probot"

 #define TEMP_SENSOR_0 0 //-1
 #define TEMP_SENSOR_1 0 //-1

 #define PROBOT_CALIBRATION_LEFT_BOTTOM 0
 #define PROBOT_CALIBRATION_RIGHT_TOP   1
 #define PROBOT_CALIBRATION_NUM_VIAS    2

 #define ENCODER_PULSES_PER_STEP 8//4

 #define ENCODER_STEPS_PER_MENU_ITEM 1

#endif
```

10.1.2 language_en.h

At the end of language_en.h, add (before `#endif // LANGUAGE_EN_H`):

```
#ifdef PROBOT
    /*Calibration*/
    #define MSG_PROBOT_CALIBRATE             "Calibration"
    #define MSG_PROBOT_CALIBRATE_LEFT_BOTTOM "Bottom-left Via"
    #define MSG_PROBOT_CALIBRATE_RIGHT_TOP   "Top-right Via"
    #define MSG_PROBOT_CALIBRATE_SAVE_POINT  "Save Point"
    #define MSG_PROBOT_CALIBRATE_DONE        "Calibration Done."
    /*Movement*/
    #define MSG_PROBOT_MOVE_AXIS             MSG_MOVE_AXIS // manual mode
    #define MSG_PROBOT_MOVE_X                MSG_MOVE_X
    #define MSG_PROBOT_MOVE_Y                MSG_MOVE_Y
    #define MSG_PROBOT_MOVE_Z                MSG_MOVE_Z

    //print this like status message(whole width)
    #define MSG_PROBOT_MOVE_TURN             "Turn Knob to move"
    //print this like status message(whole width)
    #define MSG_PROBOT_MOVE_DONE             "Press Knob to return”
    /*Remote*/
    #define MSG_PROBOT_REMOTE                "Enter Remote Mode"
    #define MSG_PROBOT_REMOTE_STATUS         "In Remote Operation"
    #define MSG_PROBOT_REMOTE_EXIT           "Exit Remote Mode"
    #define MSG_PROBOT_REMOTE_EXIT_QUERY     MSG_PROBOT_REMOTE_EXIT "?"
    #define MSG_PROBOT_REMOTE_EXIT_NO        "No, Stay."
    #define MSG_PROBOT_REMOTE_EXIT_YES       "Yes, Leave."

    #define MSG_MOVE_001MM                       "Move 0.01mm"
#endif // PROBOT
```

10.1.3 Marlin_main.cpp

10.1.3.1 Homing

Under the #else of the #ifdef DELTA of G28 implementation, either before or after the first `#if Z_HOME_DIR > 0` block, insert the following:

```
#ifdef PROBOT
if((home_all_axis) || (code_seen(axis_codes[Z_AXIS]))) {
  HOMEAXIS(Z);
}
#endif
```

And surround all remaining HOMEAXIS(Z); in the G28 like this

```
#ifndef PROBOT
HOMEAXIS(Z);
#endif
```

10.1.3.2 Calibration points

If we are getting calibration points from firmware, put this at the end of M114 implementation:

```
#ifdef PROBOT
      for (int i = 0; i < PROBOT_CALIBRATION_NUM_VIAS; i++){
        SERIAL_PROTOCOLPGM("PROBOT Cal ");
        SERIAL_PROTOCOL(i);
        SERIAL_PROTOCOLPGM(" X:");
        SERIAL_PROTOCOL(probot_calibration_vias[i][X_AXIS]);
        SERIAL_PROTOCOLPGM(" Y:");
        SERIAL_PROTOCOL(probot_calibration_vias[i][Y_AXIS]);
        SERIAL_PROTOCOLPGM(" Z:");
        SERIAL_PROTOCOL(probot_calibration_vias[i][Z_AXIS]);
      }
      SERIAL_PROTOCOLLN("");
      SERIAL_PROTOCOLLN("");
#endif //PROBOT
```

10.1.4 ultralcd.cpp

10.1.4.1 Probot LCD Menus, etc.

Code added in the menu function declarations (under the 'Different menus' heading)

```
#ifdef PROBOT
int probot_calibration_point = 0;
float probot_calibration_vias[PROBOT_CALIBRATION_NUM_VIAS][NUM_AXIS];
static void lcd_probot_move_back(int prev_menu);
static void null_function();
static void lcd_probot_calibration_save_point();
static void lcd_probot_calibration_right_top();
static void lcd_probot_calibration_left_bottom();
static void lcd_probot_calibration_menu();
static void lcd_probot_move_menu();
static void lcd_move_menu_001mm();
menuFunc_t probotMoveMenu = lcd_probot_move_menu;
#endif
```

10.1.4.2 Modification of lcd_main_menu()

In the definition of the lcd_main_menu() function, after the 'back' MENU_ITEM, insert

```
#ifdef PROBOT
   //MENU_ITEM(submenu, MSG_PROBOT_CALIBRATE, lcd_probot_calibration_menu);
   MENU_ITEM(submenu, MSG_PROBOT_MOVE_AXIS, lcd_probot_move_menu);
   //MENU_ITEM(submenu, MSG_PROBOT_REMOTE, lcd_probot_remote_menu);
   MENU_ITEM(gcode, MSG_AUTO_HOME, PSTR("G28"));
   MENU_ITEM(gcode, MSG_DISABLE_STEPPERS, PSTR("M84"));
#else // ! PROBOT
```

Then, right before the END_MENU() of the lcd_main_menu() function, insert:

```
#endif //PROBOT
```

10.1.4.3 Modification of _lcd_move()

In the _lcd_move() function, change:

```
  if (lcdDrawUpdate) lcd_implementation_drawedit(name, ftostr31(current_position[axis]));
```

to:

```
  #ifdef PROBOT
  if (lcdDrawUpdate) lcd_implementation_drawedit(name, ftostr32(current_position[axis]));
  #else //not PROBOT
  if (lcdDrawUpdate) lcd_implementation_drawedit(name, ftostr31(current_position[axis]));
  #endif //PROBOT
```

and change:

```
  if (LCD_CLICKED) lcd_goto_menu(lcd_move_menu_axis);
```

to:

```
   if (LCD_CLICKED)
 #ifdef PROBOT
     lcd_goto_menu(probotMoveMenu);
 #else
     lcd_goto_menu(lcd_move_menu_axis);
 #endif
```

10.1.4.4 Modification of lcd_move_menu_axis()

In the lcd_move_menu_axis() function, change:

```
        MENU_ITEM(submenu, MSG_MOVE_E, lcd_move_e);
```

to:

```
        #ifndef PROBOT
        MENU_ITEM(submenu, MSG_MOVE_E, lcd_move_e);
        #endif //not PROBOT
```

10.1.4.5 New function: lcd_move_menu_001mm()

After the lcd_move_menu_01mm() function, insert:

```
 #ifdef PROBOT
 static void lcd_move_menu_001mm()
 {
     move_menu_scale = 0.01;
     lcd_move_menu_axis();
 }
 #endif //PROBOT
```

10.1.4.6 New functions

After the lcd_move_menu() function, insert:

```
 #ifdef PROBOT

 static void lcd_probot_move_x()
 {
   prevEncoderPosition = encoderPosition;
   _lcd_move(PSTR("X"), X_AXIS, X_MIN_POS, X_MAX_POS);
 }
 static void lcd_probot_move_y()
 {
   prevEncoderPosition = encoderPosition;
   _lcd_move(PSTR("Y"), Y_AXIS, Y_MIN_POS, Y_MAX_POS);
 }
 static void lcd_probot_move_z()
 {
   prevEncoderPosition = encoderPosition;
   _lcd_move(PSTR("Z"), Z_AXIS, Z_MIN_POS, Z_MAX_POS);
 }
 static void null_function()
 {
 }
 static void lcd_probot_calibration_save_point()
 {
   //TODO-PROBOT: save the calibrated point and check (somewhere) that calibration is
completed
   for (int i = 0; i < NUM_AXIS; i++){
     probot_calibration_vias[probot_calibration_point][i] = current_position[i];
   }
```

```
}
static void lcd_probot_calibration_right_top()
{
  probotMoveMenu = lcd_probot_calibration_right_top;
  probot_calibration_point = PROBOT_CALIBRATION_RIGHT_TOP;
  START_MENU();
  MENU_ITEM(back, MSG_PROBOT_CALIBRATE, lcd_probot_calibration_menu);
  MENU_ITEM(function, MSG_PROBOT_CALIBRATE_RIGHT_TOP, null_function);

    MENU_ITEM(submenu, MSG_MOVE_10MM, lcd_move_menu_10mm);
    MENU_ITEM(submenu, MSG_MOVE_1MM, lcd_move_menu_1mm);
    MENU_ITEM(submenu, MSG_MOVE_01MM, lcd_move_menu_01mm);
  MENU_ITEM(function, MSG_PROBOT_CALIBRATE_SAVE_POINT, lcd_probot_calibration_save_point);
  END_MENU();
}
static void lcd_probot_calibration_left_bottom()
{
  probotMoveMenu = lcd_probot_calibration_left_bottom;
  probot_calibration_point = PROBOT_CALIBRATION_LEFT_BOTTOM;
  START_MENU();
  MENU_ITEM(back, MSG_PROBOT_CALIBRATE, lcd_probot_calibration_menu);
  MENU_ITEM(function, MSG_PROBOT_CALIBRATE_LEFT_BOTTOM, null_function);

    MENU_ITEM(submenu, MSG_MOVE_10MM, lcd_move_menu_10mm);
    MENU_ITEM(submenu, MSG_MOVE_1MM, lcd_move_menu_1mm);
    MENU_ITEM(submenu, MSG_MOVE_01MM, lcd_move_menu_01mm);
  MENU_ITEM(function, MSG_PROBOT_CALIBRATE_SAVE_POINT, lcd_probot_calibration_save_point);
  END_MENU();
}
static void lcd_probot_calibration_menu()
{
  START_MENU();
  MENU_ITEM(back, MSG_MAIN, lcd_main_menu);
  MENU_ITEM(submenu, MSG_PROBOT_CALIBRATE_LEFT_BOTTOM,
lcd_probot_calibration_left_bottom);
  MENU_ITEM(submenu, MSG_PROBOT_CALIBRATE_RIGHT_TOP, lcd_probot_calibration_right_top);
  END_MENU();
}
static void lcd_probot_move_menu()
{
  probotMoveMenu = lcd_probot_move_menu;
    START_MENU();
    MENU_ITEM(back, MSG_MAIN, lcd_main_menu);

    MENU_ITEM(submenu, MSG_MOVE_10MM, lcd_move_menu_10mm);
    MENU_ITEM(submenu, MSG_MOVE_1MM, lcd_move_menu_1mm);
    MENU_ITEM(submenu, MSG_MOVE_01MM, lcd_move_menu_01mm);
    MENU_ITEM(submenu, MSG_MOVE_001MM, lcd_move_menu_001mm);
    END_MENU();
}

#endif //PROBOT
```

10.1.5 ultralcd.h

At the top of the file, after the include(s), insert:

```
#ifdef PROBOT
  extern float probot_calibration_vias[PROBOT_CALIBRATION_NUM_VIAS][NUM_AXIS];
  #define PROBOT_POINT_LEFT_BOTTOM 1
  #define PROBOT_POINT RIGHT_TOP   2
#endif //PROBOT
```

10.1.6 ultralcd_implementation_hitachi_HD44780.h

In the lcd_implementation_status_screen() function:

At the very beginning, insert:

```
#ifndef PROBOT
```

Before the last line (should be lcd.print(lcd_status_message);), insert:

```
#else //PROBOT
lcd.setCursor(0,0);
    lcd.print('X');
    lcd.setCursor(7,0);
    lcd.print('Y');
    lcd.setCursor(14, 0);
    lcd.print('Z');
    lcd.setCursor(0,1);
    lcd.print(ftostr32sp(current_position[X_AXIS]));
    lcd.print(' ');
    lcd.print(ftostr32sp(current_position[Y_AXIS]));
    lcd.print(' ');
    lcd.print(ftostr32sp(current_position[Z_AXIS] + 0.00001));
    lcd.setCursor(0,2);
#endif // PROBOT
```

10.2 WEB APPLICATION

10.2.1 controller.html

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" type="text/css"
href="static/bootstrap/css/bootstrap.min.css" />
    <link rel="stylesheet" type="text/css" href="static/index.css"/>

    <link rel="icon" href="static/media/icon.png">
    <title>Probot | Controller</title>
</head>
<body>
<nav class="navbar navbar-default">
    <div class="container-fluid">
        <!-- Brand and toggle get grouped for better mobile display -->
        <div class="navbar-header">
            <button type="button"
                    class="navbar-toggle collapsed"
                    data-toggle="collapse"
                    data-target="#navbar-data">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
```

```
                </button>
                <a class="navbar-brand" href="controller">
                    <img alt="logo" src="static/media/logo-transparent.png" />
                </a>
            </div>
            <!-- Collect the nav links, forms, and other content for toggling -->
            <div class="collapse navbar-collapse" id="navbar-data">
                <ul class="nav navbar-nav">
                    <li class="active">
                        <a href="#">Controller
                                    <span class="sr-only">(current)</span>
                        </a>
                    </li>
                    <li>
                        <a href="calibration">Calibration</a>
                    </li>
                </ul>
            </div><!-- /.navbar-collapse -->
        </div><!-- /.container-fluid -->
    </nav>
    <div class="container">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h2 class="panel-title">Drill files</h2>
            </div>
            <div class="panel-body">
                <button class="btn btn-primary btn-block"
                        id="collapse-button"
                        type="button"
                        data-toggle="collapse"
                        data-target="#collapse-container"
                        aria-expanded="false"
                        aria-controls="collapse-container"
                >
                        Click to toggle drill file loader
                </button>
                <div class="collapse" id="collapse-container">
                    <br/>
                    <p>Loaded drill files are listed below.</p>
                    <div id="viewer">
                        <form class="fileForm">
                          <input class="file" type="file" />
                        </form>
                        <div class="list-group" id="files">
                          <!-- Populated by javascript -->
                        </div>
                    </div>
```

```
            </div>
        </div>
    </div>

    <div class="panel panel-default">
        <div class="panel-heading">
            <h2 class="panel-title">Via map</h2>
        </div>
        <div class="panel-body">
            <p>
                After loading points from a drill file, choose a point below to
command the Probot to move there.
            </p>
            <!--<svg id="draw" width=716 height=510
preserveAspectRatio="xMidYMin"></svg>-->
            <svg id="draw" viewbox="0, 0, 100, 70"
preserveAspectRatio="xMidYMin"></svg>
        </div>
    </div>
</div>
</body>
<script src="static/vendor/jquery-1.11.1.min.js"></script>
<script src="static/vendor/jquery.form.min.js"></script>
<script src="static/vendor/snap.svg-min.js"></script>
<script src="static/bootstrap/js/bootstrap.min.js"></script>
<script src="static/controller.js"></script>
</html>
```

10.2.2 controller.js

```
var svg = document.getElementById("draw");
var PROBOT = {};
PROBOT.files = $("#files");
/*------------ UPDATER BLOCK ------------*/
/**
 * This updater handles the long polling messaging system and
 * will update the ui when there are new messages
 */
PROBOT.updater = {
   fails: 0,
   cursor: 0,
   handlers: {},

   go: function(){
   $.get("update", {cursor: PROBOT.updater.cursor}).done(function(data) {
      PROBOT.updater.fails = 0;
      PROBOT.updater.cursor = data.messages.cursor+1;
```

```
      for (var key in data.messages) {
        if (data.messages.hasOwnProperty(key) &&
PROBOT.updater.handlers.hasOwnProperty(key)) {
          PROBOT.updater.handlers[key](data.messages[key]);
        }
      }

      console.log(data.messages.cursor+1);
    }).always(function() {
      console.log(PROBOT.updater.fails + " " + PROBOT.updater.cursor);
      if(PROBOT.updater.fails < 5)
        PROBOT.updater.go();
      else
        setTimeout(PROBOT.updater.go, 5000);
    }).fail(function() {
      PROBOT.updater.fails++;
    });
  },

  addHandler: function(name, handler) {
    this.handlers[name] = handler
  }
};

// data should be an array of filenames
PROBOT.updater.addHandler("file", function(data) {
  //files.html("<br>Files: <br>");
  for(var i in data) {
    filename = data[i];
    PROBOT.files.append(
        "<div class=\"filebox\">" +
            "<a  href=\"#\" " +
                 "id=\"file_" + filename + "\" " +
                 "class=\"list-group-item\" " +
                ">" +
                   "<div id=\"file_"+filename+"\" class=\"row\">" +
                       "<div id=\"file_"+filename+"\" class=\"col-xs-10\">" +
                         "<p id=\"file_"+filename+"\">"+filename+"</p>" +
                    "</div>" +
                         "<div class=\"col-xs-2\">" +
                        "<input type=\"button\" " +
                           "class=\"delete btn btn-primary \" " +
                           "id=\"delete_" + filename + "\" " +
                           "value=\"Delete\" " +
                           "autocomplete=\"off\" " +
                         "/>" +
                        "</div>" +
```

```
                "</div>" +
              "</a>"+
            "</div>"
          );
   $("#file_"+filename.replace(".", "\\.")).click(fileHandler);
   $("#delete_"+filename.replace(".", "\\.")).click(deleteHandler);
  }
}); // end of PROBOT.updater definition
var fileHandler = function(event) { // click handler for load files
 var id = event.target.id;
 var filename = id.substr("file_".length);
 loadFile(filename);
 $("#collapse-button").click();
};
var deleteHandler = function(event) { // click handler for delete button
 var id = event.target.id;
 var filename = id.substr("delete_".length);
 $.post("delete", {"name": filename} );
};
var loadFile = function(filename) { // helper function to load files
 console.log(filename);
 $.get("/select", {"name": filename})
  .done(function(data) {
   var points = data["points"]
   for(var key in points)
   {
    if (points.hasOwnProperty(key))
    {
     var val = points[key];
     var point;
     for(var i = 0; i < val["holes"].length; i++)
     {
      point = val["holes"][i];
     }
    }
   }

   for(var key in points)
   {
    if (points.hasOwnProperty(key))
    {
     var val = points[key];
     if(!val["holes"])
      continue;
     var point;
     for(var i = 0; i < val["holes"].length; i++)
     {
```

```
            point = val["holes"][i];
            canvas.addTestPoint(point["x"], point["y"], val["diameter"]);
          }
        }
      }
    })
};
/*------------ END OF UPDATER BLOCK ------------*/

/*------------ FILE SELECTOR BLOCK ------------*/

var uploaderHandler = function() { // uploads selected file
 var file = $(".file")[0].files[0];
 var name = file.name
 var reader = new FileReader();
 reader.readAsText(file, 'UTF-8');
 reader.onload = function(event) {
  var result = event.target.result;
  $.post('upload', { data: result, name: name });
 };
}
$(".file").change(uploaderHandler);

/*------------ END OF FILE SELECTOR BLOCK ------------*/


/*------------ GRAPHICS BLOCK ------------*/

// Definition of graphics class
/**
 * Class for drawing and selecting things on the svg
 */
var Graphic = function(svg) {
 var snap = Snap(svg)
 var POINT_RADIUS = 3;
 var testPoints = [];
 var sorted = false;
 var SQUARE_WIDTH = 10;
 var pointGrid = [];
 for(var col = 0; col <= 100 / SQUARE_WIDTH; col ++) {
  pointGrid[col] = []
  for(var row = 0; row <= 70 / SQUARE_WIDTH; row ++) {
   pointGrid[col][row] = [];
  }
 }
```

```
    /**
     * adds a selectable test point to this canvas object
     */
    this.addTestPoint = function(x,y, radius){
      var point = {x:x, y:y, circle:snap.circle(x, y, radius)};
      testPoints.push(point);
      pointGrid[ Math.floor(x/SQUARE_WIDTH)][ Math.floor(y/SQUARE_WIDTH)].push(point);
    }


    var distSquaredHelper = function(x1,y1,x2,y2) {
      return (x1-x2) * (x1-x2) + (y1-y2) * (y1-y2);
    }

    var getName = function(point) {
      return point.x + "," + point.y;
    }

    var pushIfValid = function(point, arr, visited) {
      if(point.x < 0 || point.y < 0) return;
      if(point.x >= pointGrid.length || point.y >= pointGrid[point.x].length) return;
      if(visited[getName(point)] === true) return;
      visited[getName(point)] = true;
      arr.push(point);
    }

    /**
     *  finds closest point to the x and y
     **/
    var findClosest = function(x,y) {

      var min = Number.MAX_VALUE;
      var minpoint = undefined;

      var visitedSet = {};
      var curGen = [];
      var nextGen = [];

      var visitedSet = {};

      pushIfValid({x: Math.floor(x/SQUARE_WIDTH), y:Math.floor(y/SQUARE_WIDTH)},
curGen, visitedSet)
      while(curGen.length > 0) {
        cur = curGen.pop();

        // checking to see whether it is possible to
        if(Math.sqrt(distSquaredHelper((cur.x +.5) * SQUARE_WIDTH, (cur.y +.5) *
```

```
SQUARE_WIDTH, x,y)) <= Math.sqrt(min) + SQUARE_WIDTH * .708 )
      {
       for(var i = 0; i < pointGrid[cur.x][cur.y].length; i++) {
        var circle = pointGrid[cur.x][cur.y][i].circle;
        var distSqr = distSquaredHelper(circle.getBBox().cx, circle.getBBox().cy, x,
y);
        if( distSqr < min )
        {
         minpoint = pointGrid[cur.x][cur.y][i];
         min = distSqr
        }
       }
       console.log("square = (" + cur.x + ", " + cur.y + ")" + "Min: " + min);
       pushIfValid({x:cur.x+1, y:cur.y}, nextGen, visitedSet);
       pushIfValid({x:cur.x-1, y:cur.y}, nextGen, visitedSet);
       pushIfValid({x:cur.x,   y:cur.y+1}, nextGen, visitedSet);
       pushIfValid({x:cur.x,   y:cur.y-1}, nextGen, visitedSet);
      }
      if(curGen.length == 0)
      {
       curGen = nextGen;
       nextGen = [];
      }
     }
     console.log("Point = (" + minpoint.x + ", " + minpoint.y + ")");
     return minpoint;
    }

    var transformPxToPt = function(x,y) {
     var pt = svg.createSVGPoint();
     pt.x = x;
     pt.y = y;
     return pt.matrixTransform(svg.getCTM().inverse());
    };

    this.selectPx = function(x,y) {  // need to do transformation
     var transformed = transformPxToPt(x,y);
     var point = findClosest(transformed.x,transformed.y);
     return point;
    }



    this.addClickHandler = function(handler) {
     snap.click(handler);
    }
```

```
    };
    // initialize board drawing
    var canvas = new Graphic(svg);

    var onClick = function(evt) { // click handler for svg
     var point = canvas.selectPx(evt.offsetX, evt.offsetY);
     $.post("move", {x:point.x, y:point.y});
    };
    canvas.addClickHandler(onClick);

    PROBOT.graphic = canvas;



    /*------------ END OF GRAPHICS BLOCK ------------*/




window.onload = function() {
  PROBOT.updater.go();
};
```

10.2.3 calibration.html

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet"
          type="text/css"
          href="static/bootstrap/css/bootstrap.min.css"
    />
    <link rel="stylesheet"
          type="text/css"
          href="static/index.css"
    />

    <link rel="icon" href="static/media/icon.png">
    <title>Probot | Calibration</title>
</head>
<body>
<nav class="navbar navbar-default">
    <div class="container-fluid">
```

```
        <!-- Brand and toggle get grouped for better mobile display -->
        <div class="navbar-header">
            <button type="button"
                    class="navbar-toggle collapsed"
                    data-toggle="collapse"
                    data-target="#navbar-data">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="controller">
                <img alt="logo" src="static/media/logo-transparent.png" />
            </a>
        </div>
        <!-- Collect the nav links, forms, and other content for toggling -->
        <div class="collapse navbar-collapse" id="navbar-data">
            <ul class="nav navbar-nav">
                <li>
                    <a href="controller">Controller</a>
                </li>
                <li class="active">
                    <a href="#">Calibration
                                <span class="sr-only">(current)</span>
                    </a>
                </li>
            </ul>
        </div><!-- /.navbar-collapse -->
    </div><!-- /.container-fluid -->
</nav>
<div class="container">
    <h1>Calibration</h1>
    <p>Follow the steps below to calibrate your Probot.</p>
    <div class="panel panel-default">
        <div class="panel-heading">
            <h2 class="panel-title">Initiate Calibration Mode</h2>
        </div>
        <div class="panel-body">
            <p>
            Initiating calibration mode will have the following effects. First, the
Probot will wait for any in-progress or queued actions to finished. Second, additional
actions to the Probot will be blocked. Third, the Probot will move back to the origin
related to that of the last calibration. Fourth and finally, the Probot motors will be
turned off.
            </p>
            <button type="button"
                    id="calibration-start-button"
```

```
                    data-starting-text="Starting..."
                    data-inprogress-text="In progress"
                    class="btn btn-primary"
                    autocomplete="off">
                    Start
                </button>
            </div>
        </div>
        <div class="panel panel-default">
            <div class="panel-heading">
                <h2 class="panel-title">Choose a reference point</h2>
            </div>
            <div class="panel-body">
                <p>
                Choose the point that will serve as the new origin or reference point for
the Probot. This point is typically the top-left most point for  the board design that
is being tested. If placement of the test board is consistent enough between
replacements, a common reference point may be given. The Probot uses this point to
reach any other point on the board design that is being tested. Please give the x and y
coordinate of the reference point below, in millimeters, as presented by design
software or show on the "controller" page.
                </p>
                X-ref-offset: <input type="text" id="x-calibration"> mm<br/>
                Y-ref-offset: <input type="text" id="y-calibration"> mm<br/>
            </div>
        </div>
        <div class="panel panel-default">
            <div class="panel-heading">
                <h2 class="panel-title">Determine the offset angle of the mount</h2>
            </div>
            <div class="panel-body">
                <p>
                When the test board is placed, the board is commonly rotated when compared
to the axes of the Probot. Even minor rotations can produce large enough error to miss
points on the test board. Find the angle in radians and enter it below.
                </p>
                Angle-offset: <input type="text" id="angle-calibration"> rads<br/>
            </div>
        </div>

        <div class="panel panel-default">
            <div class="panel-heading">
                <h2 class="panel-title">Move the probe into position</h2>
            </div>
            <div class="panel-body">
                <p>
                Manually move the probe on the Probot to the point on the test board. This
```

```
can be done with either the LCD or by hand. Typically, it is easier to move the x and y
axes by hand, and use the LCD for the z-axis. When positioning the z-axis, carefully
move the probe so that it is touching the board only as much as needed to make solid
contact with the via.
            </p>
        </div>
    </div>

    <div class="panel panel-default">
        <div class="panel-heading">
            <h2 class="panel-title">Submit calibration</h2>
        </div>
        <div class="panel-body">
            <p>
            Submit the new calibration values as given above. This action is permanent
until the next calibration.
            </p>
            <form>
                <button type="button"
                        id="calibration-submit-button"
                        data-submitting-text="Submitting..."
                        data-submitted-text="Submitted!"
                        class="btn btn-primary"
                        autocomplete="off">
                        Submit
                </button><br/>
            </form>
        </div>
    </div>
</div>
</body>
<script src="static/vendor/jquery-1.11.1.min.js"></script>
<script src="static/vendor/jquery.form.min.js"></script>
<script src="static/vendor/snap.svg-min.js"></script>
<script src="static/bootstrap/js/bootstrap.min.js"></script>
<script src="static/calibration.js"></script>
</html>
```

10.2.4 calibration.js

```
window.onload = function() {
    // Logic for start button states on calibration page
    $('#calibration-start-button').on('click', function () {
        var $btn = $(this).button('starting');
        // Do stuff
        $btn.button('inprogress');
        $('#calibration-submit-button').button('reset');
```

```
    });
    // Logic for submit button states on calibration page
    $('#calibration-submit-button').on('click', function () {
        var $btn = $(this).button('submitting');
        var x = $('#x-calibration').val();
        var y = $('#y-calibration').val();
        var a = $('#angle-calibration').val();
        $.post("calibrate", { x:x, y:y, a:a });
        // Do stuff
        $btn.button('submitted');
        $('#calibration-start-button').button('reset');
    });
};
```

10.2.5 index.css

```
body {
    background-color: #0c97d1;
}
h1 {
    font-weight: bold;
}
.navbar.navbar-default {
    border-radius: 0px;
    background-color: white;
}
.container {
    margin-top: 40px;
    margin-bottom: 30px;
    margin-left: auto;
    margin-right: auto;
    padding: 50px;
    box-shadow: 0px 0px 10px 3px black;
    background-color: white;
}
/*
.panel-default>.panel-heading {
    color: white;
    background-color: #0c97d1;
}
*/
.btn-primary {
    background-color: #0c97d1;
}
.navbar-brand img {
    margin: 0px;
    height: 30px;
```

```
}
#files {
    margin-top: 5px;
}
#draw {
    border: 1px solid black;
    max-height: 500px;
    max-width: 716px;
}
```

10.3 SERVER

10.3.1 WebMain.py

```
import tornado.ioloop
import tornado.web
from tornado import gen
from tornado.template import Loader
from InteractionHandler import InteractionHandler
loader = Loader("html/")
# handles the interactions with the filesystem and the embedded system
# also handles the generation of messages to the client
interaction_handler = InteractionHandler()

class ControllerHandler(tornado.web.RequestHandler):
    """
    Generates html from the template 'controller.html' and writes it to the response
    """
    def get(self):
        self.write(loader.load("controller.html").generate())
        self.finish()
class CalibrationHandler(tornado.web.RequestHandler):
    """
    Generates html from the template 'calibration.html' and writes it to the
response
    """
    def get(self):
        self.write(loader.load("calibration.html").generate())
        self.finish()
class MessageHandler(tornado.web.RequestHandler):
    '''
    Handles the long polling messaging system
    '''
    @gen.coroutine
    def get(self):
        last_message = self.get_argument("cursor", 0)
        #print(last_message)
        self.future = interaction_handler.waitForMessages(last_message)
```

```
        messages = yield self.future
        if self.request.connection.stream.closed():
            interaction_handler.cancelWaiter(self.future)
            return
        #print(dict(messages=messages))
        self.write(dict(messages=messages))
        self.finish()

class FileUploadHandler(tornado.web.RequestHandler):
    """
    Writes file from post data into the filesystem (can only write to one directory)
    """
    def post(self):
        name = self.get_argument("name")
        data = self.get_argument("data")
        interaction_handler.addFile(name, data)
        self.finish()

class FileDeleteHandler(tornado.web.RequestHandler):
    """
    Deletes file from post data into the filesystem (can only delete files in one
directory)
    """
    def post(self):
        name = self.get_argument('name')
        success = interaction_handler.deleteFile(name)
        self.write({"success":success})
        self.finish()
class FileSelectHandler(tornado.web.RequestHandler):
    """
    Returns parsed data from selected file
    """
    def get(self):
        name = self.get_argument('name')
        points = interaction_handler.getFile(name)
        #print(points)
        self.write({"success": True, "points": points})
        self.finish()

    def post(self):
        self.get()
class MovementHandler(tornado.web.RequestHandler):
    """
    Sends a movement command to the robot
    """
    def get(self):
        x = float(self.get_argument("x"))
```

```
            y = float(self.get_argument("y"))
            success = interaction_handler.moveCommand(x, y)
            self.write({"success":success})
            self.finish()

        def post(self):
            self.get()

    class EchoHandler(tornado.web.RequestHandler):
        """
        Handler used for debugging purposes (not required for release)
        """
        def get(self):
            message = self.get_argument('echo', 'test message')
            interaction_handler.notify('echo', message)
            #print(interaction_handler.message_buffer.messages['cursor'])
            self.finish()

        def post(self):
            self.get()

    class CalibrateHandler(tornado.web.RequestHandler):
        """
        Sets the calibration parameters of the robot.
        """
        def get(self):
            x = float(self.get_argument("x"))
            y = float(self.get_argument("y"))
            a = float(self.get_argument("a"))
            success = interaction_handler.calibrateCommand(x, y, a)
            self.write({"success":success})
            self.finish()
        def post(self):
            self.get()
    # defines the handlers associated with different urls
    '''
    Urls that match a specified regular expression will be
    sent to the corresponding RequestHandler
    '''
    handlers = [
                (r'/(favicon.ico)', tornado.web.StaticFileHandler, {'path': '/'}),
                (r'/static/(.*)', tornado.web.StaticFileHandler, {'path': 'static/'}),
                (r'/(?:controller)?', ControllerHandler), # matches '/' or '/controller'
                (r'/calibration', CalibrationHandler),
                (r'/update', MessageHandler),
                (r'/delete', FileDeleteHandler),
                (r'/upload', FileUploadHandler),
```

```
            (r'/select', FileSelectHandler),
            (r'/move', MovementHandler),
            (r'/echo', EchoHandler),
            (r'/calibrate', CalibrateHandler)
]
application = tornado.web.Application(handlers)

def main():
    '''
    Starts the tornado webserver on port 8888
    Note that this will not return until the sever shuts down
    '''
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start() # blocks until shutdown

if __name__ == "__main__":
    main()
```

10.3.2 InteractionHandler.py

```
'''
Created on Jan 19, 2015

@author: Trevor Boone, Shawn LaGrotta
'''

from FileHandler import FileHandler
from CommandHandler import CommandHandler
import RPi.GPIO as gpio
import threading
from tornado import concurrent
import os

FILE_BUFFER = "file"
PROBE_BUFFER = "probe"
FILE_DIRECTORY = os.path.join(os.path.dirname(__file__), "UploadedFiles")

'''
Style note to teammates:

Information concerning the state of the system as a whole
should ONLY be passed to the client through a MessageBuffer
object. This means that any command that changes the state
of the system should NOT return information about the change
of state but instead send a notification of the state change
to the MessageBuffer object.

This means that information not about the state of the system
(for example the points contained in a file) can be passed
directly to the client on request but if the user asks to
delete a file the request  can return success or failure but
```

```
data on the actual change to the filesystem should go through
the MessageBuffer.
'''
class MessageBuffer(object):
    '''
    Class made to hold all the different type of messages
    '''
    def __init__(self):
        self.notify_lock = threading.Lock()
        self.messages = {}
        self.waiters = []
        self.messages['cursor'] = 0

    def getMessages(self, cursor):
        '''
        gets the messages held in this object
        Args:
            cursor (int) : the message number being queried for
        Returns:
            None if the current message number is less than the one being
            looked for. Else it returns a dictionary of the current messages.
        '''
        if self.messages['cursor'] >= cursor:
            return self.messages
        return None


    def addWaiter(self, future):
        self.waiters.append(future)


    def notify(self, name, message):
        '''
        adds a message to the message dictionary
        Args:
            name (string): The key to be changed or created in the message
dictionary.
            message (string): the message to put into that key
        '''
        self.notify_lock.acquire()
        self.messages['cursor'] += 1
        self.messages[name] = message
        for waiter in self.waiters:
            if not waiter.done():
                waiter.set_result(dict(self.messages))
        self.notify_lock.release()
```

```
        def cancelWaiter(self, future):
            '''
            stops this future from being notified of future messages
            '''
            self.waiters.remove(future)

    class StateHandler(object):
        def __init__(self):
            self.usetimer = threading.Timer(60*5, self.setGreen) # 5 minutes
            gpio.setwarnings(False)
            gpio.setmode(gpio.BCM)
            gpio.setup(26, gpio.OUT) # Green
            gpio.setup(17, gpio.OUT) # Red
            gpio.setup(16, gpio.OUT) # Yellow
            gpio.output(26, 1)
            gpio.output(17, 0)
            gpio.output(16, 0)

        def onActivity(self):
            '''
            turns on yellow LED and sets a timer to turn it off after 5 minutes.
            '''
            self.setYellow()
            self.usetimer.cancel()
            self.usetimer = threading.Timer(60*5, self.setGreen) # 5 minutes
            self.usetimer.start()

        def onMovement(self):
            '''
            turns on the red led
            '''
            self.setRed()

        def setGreen(self):
            gpio.output(26, 1)
            gpio.output(17, 0)
            gpio.output(16, 0)

        def setYellow(self):
            '''
            turns the yellow LED on and the others off
            '''
            gpio.output(26, 0)
            gpio.output(17, 0)
            gpio.output(16, 1)
```

```
    def setRed(self):
        '''
        turns the red LED on and the others off
        '''
        gpio.output(26, 0)
        gpio.output(17, 1)
        gpio.output(16, 0)


# Singleton class to handle the interactions of the webserver with the other systems
class InteractionHandler:
    '''
    Singleton class to handle the interactions of the webserver with the other
systems
    The methods in this class correspond to methods in other classes. (note: this
class
    probably could be removed. Originally more functionality was intended to be in
this
    class but during development most of the logic was moved to the member objects)
    '''
    def __init__(self):
        self.message_buffer = MessageBuffer()
        self.fileHandler = FileHandler(self.message_buffer, FILE_BUFFER,
FILE_DIRECTORY)
        self.commandHandler= CommandHandler(self.message_buffer, PROBE_BUFFER)
        self.stateHandler = StateHandler()

    def addFile(self, name, data):
        self.stateHandler.onActivity()
        return self.fileHandler.addFile(name, data)
    def deleteFile(self, name):
        self.stateHandler.onActivity()
        return self.fileHandler.deleteFile(name)
    def calibrateCommand(self, x, y, a):
        self.stateHandler.onActivity()
        return self.commandHandler.calibrateCommand(x, y, a)

    def moveCommand(self, x, y):
        self.stateHandler.onMovement()
        returnme = self.commandHandler.moveCommand(x, y)
        self.stateHandler.onActivity()
        return returnme

    def getFile(self, name):
        self.stateHandler.onActivity()
        return self.fileHandler.getFile(name)
```

```
    def waitForMessages(self, cursor):
        future = concurrent.Future()
        messages = self.message_buffer.getMessages(int(cursor))
        if messages is not None:
            future.set_result(messages)
        else:
            self.message_buffer.addWaiter(future)
        return future

    def cancelWaiter(self, future):
        return self.message_buffer.cancelWaiter(future)

    def notify(self, name, message):
        return self.message_buffer.notify(name, message)
```

10.3.3 FileHandler.py

```
'''
Created on Jan 19, 2015
@author: Trevor Boone
'''
import os
DEFAULT_SCALE = 1e-5
def parse(filename, scaler = DEFAULT_SCALE):
    '''
    returns a dictionary for each tool with points, and diameter
    points dict of x, y

    It is in the following format where T0 and T1 are
    tool name (javascript implementation at time of writing doesn't
    care about what the key values for the tools actually are)::
        {"T0" : {"diameter": 5.1, "points": [{'x': 0.0, 'y': 0.0}, {'x': 234.0, 'y':
15.0}]},
        {"T1" : ...
        }
    '''
    scale = lambda position: scaler * position

    f = open(filename, 'r')

    # parsing tools
    tools = {"None": {"diameter":1, "holes": []}}
    while(True):
        line = f.readline()
        if line[1] == "T":
            tool = line[1:line.find(" ")]
            diameter_start = line.find("=") + 2
```

```
                diameter_end = line.find(" ", diameter_start)
                diameter = float(line[diameter_start: diameter_end])
                tools[tool] = {"diameter": diameter, "holes": []}
            if line.find(";") == -1:
                break

        cur_tool = "None"

        line = f.readline().strip()
        x = 0
        y = 0

        # commands implemented (T#, R#[X# or Y#], X#Y#, [X# or Y#])
        while(line != ""):
            if line[0] == "T":
                cur_tool = line.strip()
            else:
                if line[0] == "R":
                    xpos = line.find("X")
                    ypos = line.find("Y")
                    iter_end = xpos if xpos >= 0 else ypos
                    iterations = int(line[1:iter_end])
                    if xpos >= 0:
                        num_end = ypos if ypos >=0 else 0
                        x_incr = float(line[xpos+1:num_end]) if num_end > xpos else
    float(line[xpos+1:])
                    else:
                        x_incr = 0
                    if ypos >= 0:
                        y_incr = float(line[ypos+1:])
                    else:
                        y_incr = 0
                    for i in range(iterations):
                        x += x_incr
                        y += y_incr
                        tools[cur_tool]['holes'].append({"x": scale(x), "y": scale(y)})
                else:
                    xpos = line.find("X")
                    ypos = line.find("Y")
                    #print(" ".join((xpos.__str__(), ypos.__str__())))
                    if xpos >= 0:
                        num_end = ypos if ypos >=0 else 0
                        x = float(line[xpos+1:num_end]) if num_end > xpos else
    float(line[xpos+1:])
                    if ypos >= 0:
                        y = float(line[ypos+1:])
                    tools[cur_tool]['holes'].append({"x": scale(x), "y": scale(y)})
```

```
            line = f.readline().strip()
        f.close()
        return tools


class FileHandler:
    '''
    handles the calls that affect the server filesystem
    '''
    def __init__(self, messager, name, directory):
        '''
        Constructs a FileHandler
        Args:
            messager (MessageBuffer): MessageBuffer that handles the passing of
messages to the webserver
            name (str): key in the message dictionary that this object will store
it's messages in
            directory (str): All of the files read, created, or deleted by this
object will be in this directory
        '''
        self.messager = messager
        self.name = name
        self.directory = directory
        self.messager.notify(name, self.listFiles())

    def listFiles(self):
        '''
        lists files in the current directory
        '''
        return os.listdir(self.directory)

    def addFile(self, name, data):
        '''
        adds a file to the directory specified in the constructor

        Args:
            name (str): name of the file to be created
            data (str): contents of the file to be writted
        '''
        path = os.path.join(self.directory, name)
        self.raiseIfInvalid(path)
        self.deleteFile(path)
        f = open(path, 'w')
        f.write(data)
        f.close()
        self.messager.notify(self.name, self.listFiles())
```

```
    def  deleteFile(self, name):
        '''
        deletes a file to the current directory
        Args:
            name (str): name of the file to be deleted
        '''
        path = os.path.join(self.directory, name)
        self.raiseIfInvalid(path)
        if os.path.exists(path) and os.path.isfile(path):
            os.remove(path)
            self.files = os.listdir(self.directory)
            self.messager.notify(self.name, self.listFiles())

    def getFile(self, name):
        '''
        returns a parsed version of a file in the current directory
        Returns:
            dict: see documenation for parse
        '''
        path = os.path.join(self.directory, name)
        self.raiseIfInvalid(path)
        return parse(path)
    def raiseIfInvalid(self, path):
        '''
        raises a name error if the file path isn't in the current directory
        '''
        if os.path.dirname(path) != self.directory:
            raise NameError("Invalid file name: file name has to resolve to same
directory")

if __name__ == "__main__":
    points = parse("drill.drl")

    print(points)
```

10.3.4 CommandHandler.py

```
'''
Created on Jan 19, 2015
@author: Trevor Boone, Shawn LaGrotta
'''
import os
import threading
import time
import math
from mecode import G
from printrun import gcoder
from printrun.printcore import printcore
```

```
def curposcb(line):
    print("Current pos " + line);
class CommandHandler:
    '''
    Handles the interactions between the server
    and the printer.
    '''

    def __init__(self, messager, message_name):
        '''
        Constructor for a CommandHanler
        Args:
            messager (MessageBuffer): The MessageBuffer to handle the messages
            message_name (str): The key to be used in the dictionary of messages to
be sent
        '''
        self.messager = messager
        self.message_name = message_name
        self.state = {'state': 'idle', 'x': 0, 'y': 0, 'z': 0,
'communications':'acknowledged'}
        self.thread = threading.Thread(target=self.__read)
        self.thread.start()
        self.xoffset = 0.0
        self.yoffset = 0.0
        self.aoffset = 0.0
        self.gcodes = InMemoryFile()
        self.mecode = G(outfile=self.gcodes, print_lines=True)
        self.printcore = printcore("/dev/ttyACM0", 115200)
        self.printcore.tempcb = curposcb
        self.printcore.loud = True
        time.sleep(2)

    def calibrateCommand(self, x, y, a):
        '''
        sets positional and angle offsets for the machine to account
        for translating between the hard drive's and the printer's
        coordinate systems
        '''
        print("Calibrating to point " + str(x) + ", " + str(y) + " " + "with angle "
+ str(a));
        self.xoffset = x
        self.yoffset = y
        self.aoffset = a
        self.printcore.send("G92 X0.0 Y0.0 Z0.0")
        self.mecode._update_current_position(mode='auto', x=0.0, y=0.0, z=0.0)
    def moveCommand(self, x, y):
        # compensate for the origin of the calibrated coordinate system
```

```
            x = x-self.xoffset
            y = y-self.yoffset
            # compensate for the rotation of the calibrated coordinate system
            # for 3d, see http://inside.mines.edu/fs_home/gmurray/ArbitraryAxisRotation/
            x = y*math.sin(self.aoffset) + x*math.cos(self.aoffset)
            y = y*math.cos(self.aoffset) - x*math.sin(self.aoffset)
            # x & y swapped to convert between robot axis and display axis on client
            self.mecode.relative()
            self.mecode.move(0, 0, -5)
            self.mecode.absolute()
            self.mecode.abs_move(y, x)
            self.mecode.relative()
            self.mecode.move(-1, -1, 0) # resolve x,y backlash
            self.mecode.move(1, 1, 0)
            self.mecode.move(0, 0, 5)
            self.mecode.teardown()
            gcodes = [i.strip() for i in self.gcodes.getvalue().split(os.linesep)]
            self.gcodes = InMemoryFile()
            self.mecode.out_fd = self.gcodes
            self.mecode.setup()
            # Send to printer
            print("Waiting for current print to finish")
            self.printcore.startprint(gcoder.LightGCode(gcodes))
            while(self.printcore.printing): # Wait until all commands sent to firmware
                print('.')
                time.sleep(1)
            time.sleep(15)
            print("Done")


        def __read(self):
            '''
            method that runs in other thread and
            reads input from serial port
            '''
            pass

    class InMemoryFile:
        '''
        A file like object that stores the written contents in memory
        '''
        def __init__(self):
            self.string_list = []

        def write(self, string):
            self.string_list.append(string)
```

```
def flush(self):
    '''
    does nothing in this object. Needs to be defined
    to satisfy the requirements as a file like object
    '''
    pass

def close(self):
    '''
    does nothing in this object. Needs to be defined
    to satisfy the requirements as a file like object
    '''
    pass

def getvalue(self):
    '''
    gets the contents written to this object
    '''
    return ''.join(self.string_list)

def writelines(self, lines):
    '''
    writes the lines to this object. Adds a newline to all but
    the last line.
    Args:
        lines (list) : The list of lines to be written to this object
    '''
    self.write('\n'.join(lines))
```