AUTOMATED BOBA STATION

By

Hunter Huynh

Timothy Ko

Jordan Wu

Final Report for ECE 445, Senior Design, Spring 2019

TA: John Kan

1 May 2019

Project No. 49

1 Introduction

Boba, a popular drink among millennials, is still largely made manually, thus shops still require many employees to keep up with demand. As such, the prices are still largely dictated by the manual labor. In addition, with the large variety of recipes, human workers are prone to make mistakes. Finally, taste consistency is hard to achieve leaving drinks sometimes oversweet. Unlike coffee, the Boba making process is hard to automate as it requires handling and dispensing of both solid and liquid ingredients.

1.1 Background

Boba drinks come in a multitude of configurations. However, it generally consists of just solids and liquids. In general, the process of creating a Boba drink consists of first adding the toppings (solids) in a cup, pouring in tea and milk (milk tea may already be mixed together with a specific concentration of milk and tea), then adding a user specified amount of sugar and ice (such as 0%, 25%, 50%, 75%, or 100% of the normal amount) [1] [2].

Because the creation of Boba drinks is currently all manual and based off percentages, there is no unified concentration of each component of the drink. For example, 25% sugar level varies employee to employee as it is up to them to gauge the amounts. The concentration of milk and tea varies as well as the amount of toppings is in a drink. If done automatically, drinks will follow the recipe every single time, increasing taste consistency and customer satisfaction.

1.2 Objectives

Our goal is to develop an automated Boba station having multiple modular dispensers. The dispenser mechanism must be accurate to ensure taste consistency. The modular design would allow the customer (shop owner) to customize the number and type of dispensers themselves, permitting a wide permutation of drinks and leading to a lower cost in the long run. Finally, the system will be controlled through an easy to use web interface that allows for recipe customization. In addition, this web interface can also keep track of ingredients used, thereby reducing food waste caused by overbuying ingredients

1.3 High-level Requirements

- (MET) Have at least two dispensers, one for tapioca pearls and one for the milk tea.
- (MET) The station must be able to dispense a user-specified amount of milk tea and tapioca pearls with no more than ±10% error in mass.
    - In our final performance, our liquid dispensing is accurate to 1 gram. In typical scenarios, this would be less than 1% error in mass. Our solid dispensing controlled by serving counts rather than by gram due to design of dispenser.
- (MET) Have a web interface to control the amount of liquids/solids dispensed in each dispenser.

2 Design

This automated Boba station will have two dispensers: one for tapioca pearls and one for milk tea. We will be using a “revolving door” to dispense the tapioca pearls and a solenoid valve to dispense the milk tea. An employee can manually refill the containers holding the milk tea and tapioca pearls. In a commercial setting, there will then be a sloped gutter the dispensers will dispense to, which direct the ingredients into a cup placed at the bottom. The cup will be on a small raised platform with a load sensor underneath, which is connected to its own microcontroller.

Each dispenser and sensor module will be powered by its own AC-DC transformer and controlled by its own microcontroller module. All microcontrollers will have a Wi-Fi capability and host a simple web server listening for instructions. A web server will be controlling the microcontrollers via HTTP and serve a web user interface to control the amount of liquid/solid dispensed and when to begin the Boba making process.

The modular and wireless design also allows for the customer to place dispensers wherever convenient without needing to run data cables. The wireless nature also supports automatic over-the-air software updates for updating or adding new features such as making dispensing more efficient. (Similar to how Tesla can update their cars software functionalities automatically)

2.1 Block Diagram

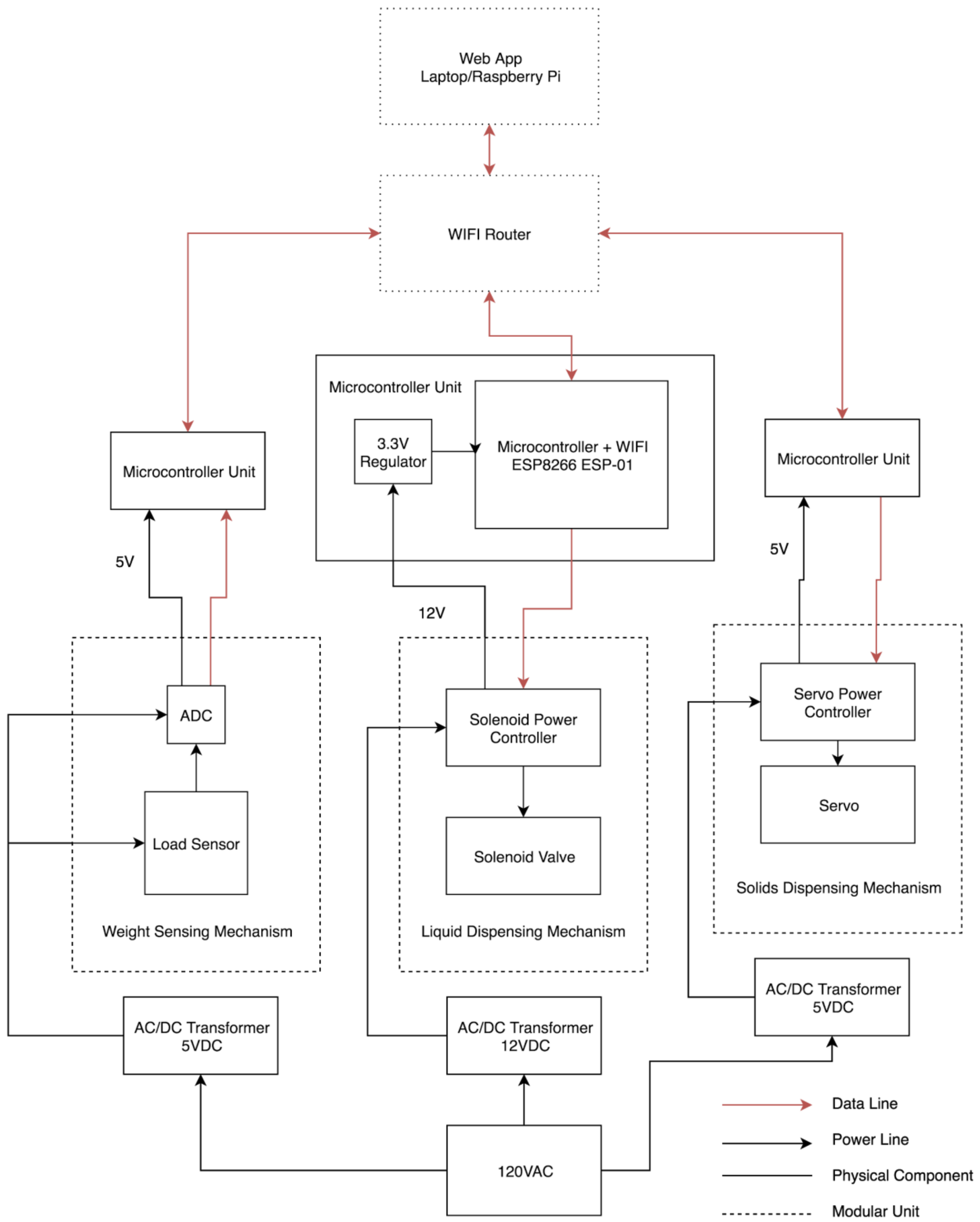


Figure 1. Block Diagram

2.2 Physical Design

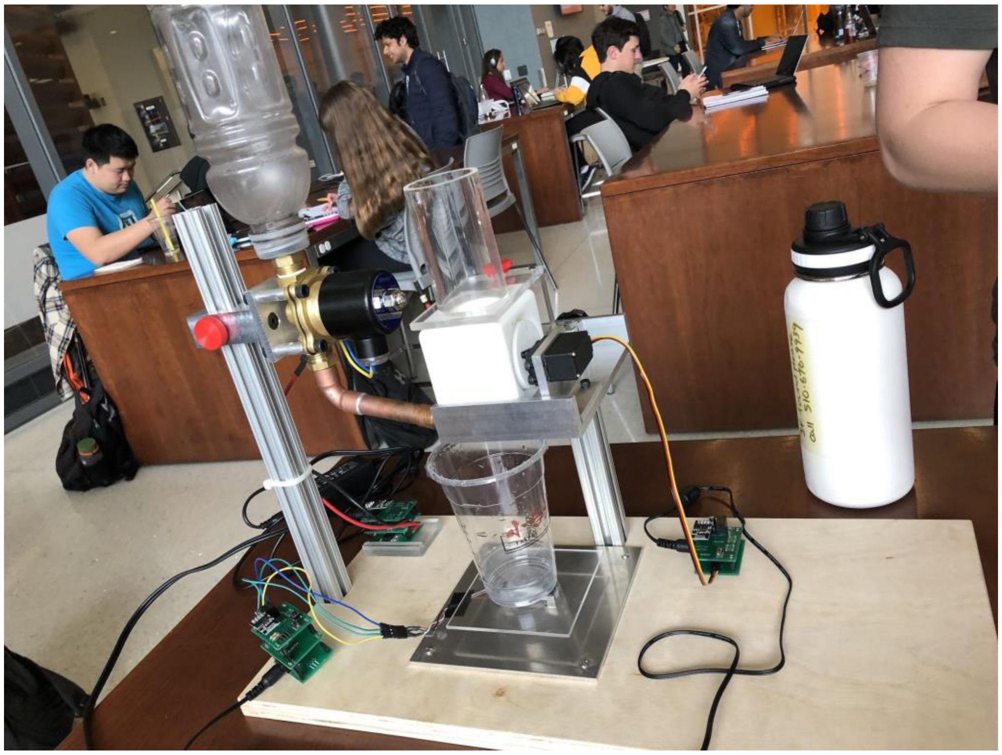


Figure 2. Physical Design of the machine

2.3 Liquid Dispensing Module

The liquid dispensing module is a core part of our automated Boba station since it dispenses the necessary liquids (milk and tea) to make a Boba drink. It consists of a solenoid valve and the circuitry required to open and close the valve given a GPIO pin voltage. As it relates to the rest of the components, the liquid dispensing module will be directly connected to a microcontroller module, which provides the aforementioned GPIO voltage.

2.3.1 Design Procedure

The first step was to decide between using a pump or a valve. Since we wanted high accuracy, a pump was the best option as it could be opened and closed with much less latency than a pump. Since pressurizing the liquid would introduce additional components and complexities, we also narrowed our search for valves to those that are gravity fed.

There were two paths to create a circuit for driving our 12V 3A solenoid valve. One was to use transistors, and the other was to use an electromechanical relay. We decided against a relay since it introduces additional latency due to its mechanical design and would still need transistor circuitry for control.

We initially followed the solenoid valve datasheet in our design by using a single TIP120 Darlington NPN transistor, and during breadboard testing the components operated as expected. However, we forgot to verify operation over a longer period. It is only after PCB assembly and testing, we noticed the abnormally high power dissipation being asked of the transistor due to the high V_{CE} characteristics of the TIP120[3]. We then re-designed the circuit using two RFP30N06LE N-channel MOSFETs, shown in **Figure 3**, due their low V_{DS} characteristics, which alleviated the power dissipation problem. The reason we needed two was that the 3.3V GPIO signal was unable to saturate the RFP30N06LE, so another was used as a gate driver.

We used the following equations in determining required voltages and power dissipation:

$$V = IR$$

$$P = IV$$

2.3.2 Design Details

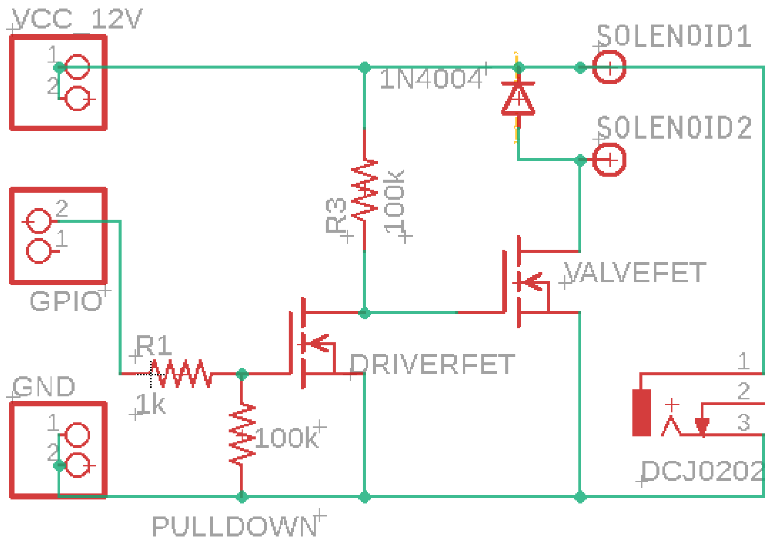


Figure 3. Liquid Dispenser Driver

Due to the undefined characteristics of a floating MOSFET gate, a pulldown resistor is required on the gate of the MOSFET labeled DRIVER in Figure 3. As the gate of the MOSFET also acts as a capacitor, there needs to be a resistor between the GPIO pin and the gate to limit instantaneous current flow and prevent damage to the microcontroller. In addition, a resistor is needed to prevent a short when the DriverFET is opened. This resistor also needs to keep ValveFET above saturation voltage while DriverFET is closed. Finally, a flyback diode was added across the solenoid terminals to eliminate inductive flyback from the solenoid.

2.3.2.1 Calculations

Microcontroller Specifications [4] :

$$I_{GPIOMax} = 12mA$$

RFP30N06LE N-Channel MOSFET Specifications [5] :

$$V_{GSThresh} = 2V$$

$$V_{GSSat} = 5V$$

$$V_{GSMax} = 10V$$

Solenoid Valve Specifications [6] :

$$I_{Solenoid12V} = 3A$$

$$6V < V_{SolenoidON} < 12V$$

Calculations:

$$I_{GPIO} = \frac{V_{GPIO} - V_{GSDriverFET}}{R_1}$$

$$I_{GPIO} = \frac{3.3V}{1k\Omega} = 3.3mA < I_{GPIOMax}$$

$$V_{GSDriverFETON} = 3.3V * \frac{1k\Omega}{1k\Omega + 100k\Omega} = 3.26V$$

$$V_{GSThresh} < 3.26V < V_{GSMax}$$

For $R3$, we just plugged in various resistors to satisfy the following equation:

$$V_{GSSat} < V_{GSValveFET} < V_{GSMax}$$

We found $100k\Omega$ to work well.

From **Table 1** in the verification section, we can calculate the power dissipation across the MOSFETs and ensure safe operations.

$$P_{ValveFET} = IV = 3A * 1V = 3W$$

$$P_{DriverFET} = IV = 5.8V * \frac{12V - 5.8V}{100k\Omega} = .359mW$$

2.4 Solid Dispensing Module

The solid dispensing module is responsible for dispensing solid toppings. We chose to build around dispensing tapioca pearls as it is the most popular topping. However, the system can be adapted to dispense other types of toppings with modification to the mechanical portion.

2.4.1 Design Process

We originally thought about using an Archimedes screw mechanism in moving and dispensing the pearls. However, we were unable to find a pre-made mechanical solution that was viable in terms of cost and building one from scratch would be too difficult as it requires advanced 3D modeling techniques that none of us were capable of.

Our final design was inspired by cereal dispensers in the school cafeterias, which allows for dispensing of solids via a rotating-door mechanism.

2.4.2 Design Details

The final design calls for a rotating tube with a hole sitting inside a block. In the “closed” state, the toppings drop into hole in the tube, which is facing up at 0 degree. To dispense, a PWM signal is sent to the servo, and the servo rotates 180-degree so the hole faces down, and the toppings fall out into the cup. After a few seconds, another PWM signal is sent to the servo to rotate back to 0 degree.

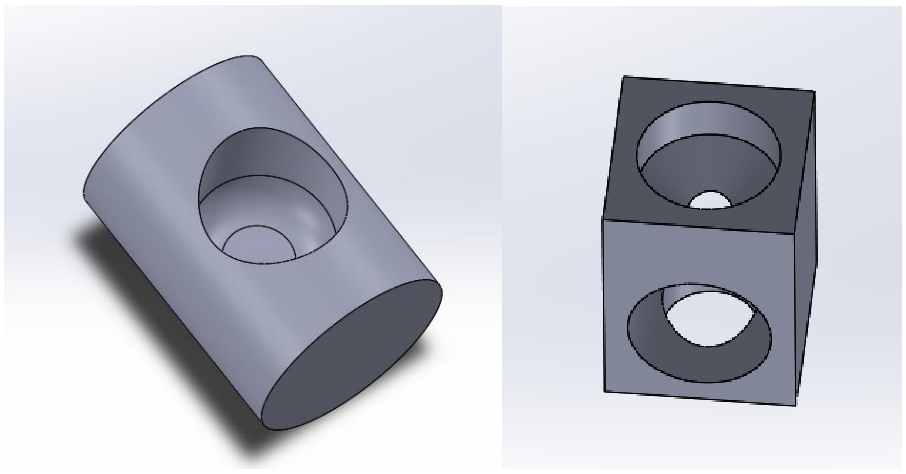


Figure 4. Solid Dispensing Mechanism Parts

The size of the mechanism was crucial as every gram of material affects the final cost for 3D printing. The small size of the final mechanism also helped simplify the circuitry. The lower torque required by the smaller size allows for use of a 5V servo, which it turns allows for control by a 3.3V PWM signals directly from the microcontroller [7].

2.4.2.1 Calculations

To calculate the required PWM duty cycle for 180-degree rotation, we followed **Figure 5**.

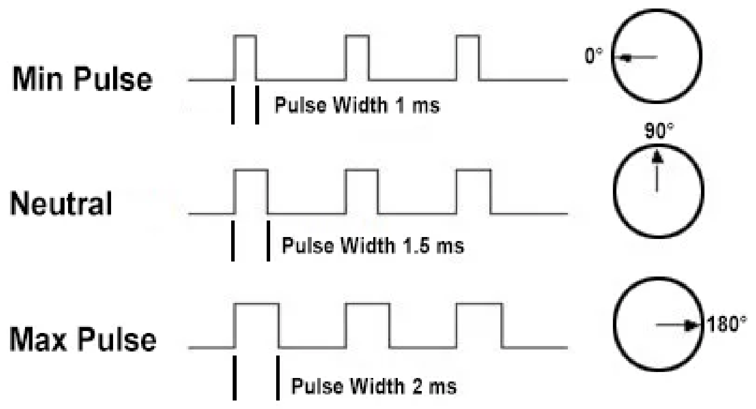


Figure 5. PWM signal vs position [8]

Given a 50Hz signal, as recommended by various sources, and a duty cycle out of 1024.

$$minDuty = \frac{1024 * 1ms}{20ms} = 51 \qquad maxDuty = \frac{1024 * 2ms}{20ms} = 102$$

2.5 Weight Sensing Module

2.5.1 Design Procedure

The purpose of this module is to measure the weight of liquids and solids already dispensed in the cup. We chose the TAL221 mini load cell (500g, straight bar) as the weight sensor for its ability to accurately measure weight up to 500 g.

Since the load cell alone outputs a tiny (< 8 mV) voltage, it would be nice if we could amplify it and convert it into a digital signal. Thus, we have an Amplifier/ADC board that does exactly that. The board uses an HX711 chip since it is a precise 24-bit ADC designed for use in applications such as weight sensing [9]. The design of the board schematic, based on the block diagram on the first page of the HX711 datasheet [9], provides a simplified interface to the HX711.

2.5.2 Design Details

2.5.2.1 Load Cell

We mounted the load cell under the platform the cup sits on. As verified, when a constant voltage is applied to its red/black input wires, the load cell outputs a small voltage that increases as the weight on the load cell increases. From the verification **Figure 12**, the linear model of the load cell is:

$$voltage = 6.85 * 10^{-3} * weight + 3.79$$

where weight is assumed to be no more than 500 g. Thus, the output voltage is no more than 0.00715 V. We found that the Pearson correlation coefficient is 1.0, so a linear model is an extremely good fit.

2.5.2.2 Load Cell Amplifier/ADC

In the schematic of the Amplifier/ADC board in **Figure 6**, the RED, BLK, WHT, and GREEN pins are connected directly to the load cell. The VDD, VCC, and GND pins are connected to 3.3 V, 5.0 V, and ground respectively. Finally, the CLK and DATA pins are connected to the microcontroller's GPIO pins.

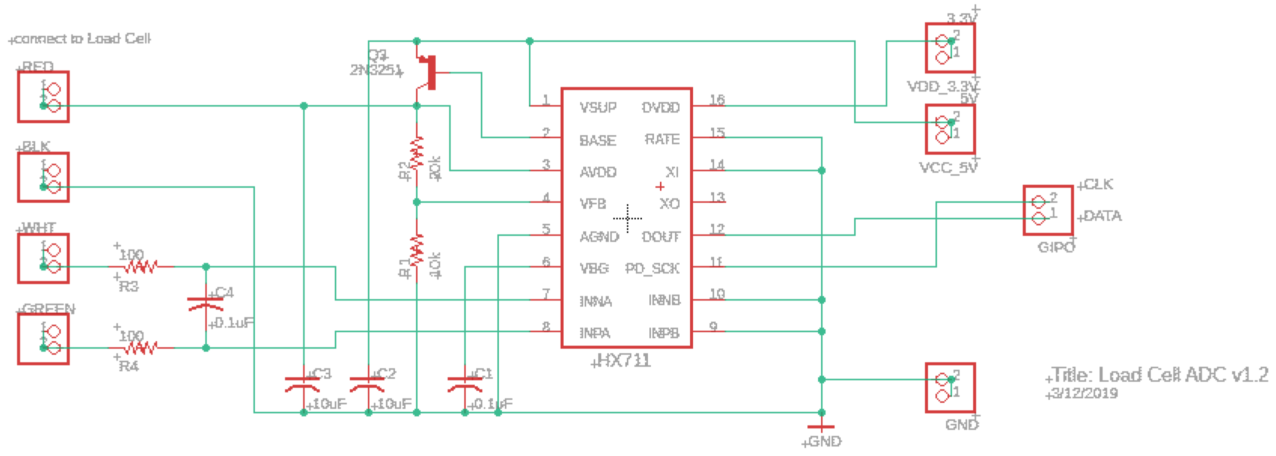


Figure 6. Load Cell Amplifier/ADC Schematic

2.5.2.3 Weight Sensing Software

The HX711 has a digital value corresponding to weight, but how does the microcontroller obtain it? The microcontroller must be programmed to follow the HX711's serial interface, which is as follows. The HX711 can be powered down by changing CLK from low to high and having it be high for > 60 us. It can be powered up by returning CLK to low. To read a digital value, wait until DATA is low. Then, apply 25 pulses to CLK where each pulse is high for < 50 us. Each pulse shifts out one bit from DATA, starting with the most significant bit, until all bits are shifted out. The 25th pulse causes DATA to go back to high until another digital value is ready to be read.

With this, we can get a digital value from the microcontroller corresponding to the weight. From verification, a linear fit of the digital value vs. the weight is:

$$value = 2966 * weight + 361$$

where the weight is in grams. Therefore, to have the weight sensing software output the weight in grams, we divide the digital value by 2966.

2.6 Microcontroller Module

The microcontroller module handles communication with the main server and control of the previously mentioned modules through GPIO. As most microcontrollers run at 3.3V, we will need some voltage regulator to step down from the 12V or 5V power from the power supply. In addition, the microcontroller module will also need the passive components required for controlling and programming the microcontroller chip.

2.6.1 Design Process

We first needed to decide on which microcontroller chip to use. We needed Wi-Fi capabilities to allow the main server can contact each individual microcontroller through HTTP requests. Initially, we looked at using the ATmega328 chip as the main microcontroller running the logic along with a slave Wi-Fi module. However, it seemed quite pointless as the ESP8266 WIFI module was much more powerful in terms of functionality than the ATmega chip.

For power delivery, we decided to go with the TSR 1-2433 as building our own buck converter would not have been more cost effective.

2.6.2 Design Details

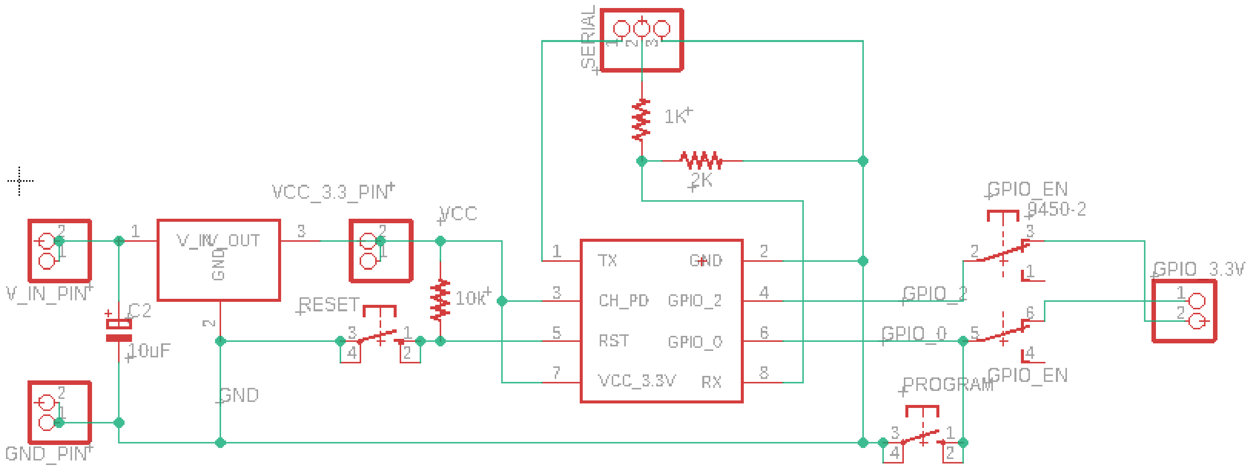


Figure 7. Microcontroller Module Schematic

In our microcontroller module, we also needed various passive components for programming and control of the ESP chip. For example, there is a step-down circuitry at the top of **Figure 7** near the pins labeled SERIAL. This is used for stepping down to 3.3V the 5V logic level from a USB to Serial chip in an Arduino Uno board. To the right, near the GPIO pins, is a DPDT slide switch, to disconnect the GPIO pins. This is used to prevent undefined logic levels in the other modules from preventing the ESP chip from booting correctly, as the GPIO pins also serve to put the ESP chip into bootloader mode.

2.6.2.1 Software

We flashed the MicroPython firmware on our microcontrollers [10]. This allows for programming over WIFI as well, which eliminated the need for serial programming after flashing.

To further simplify programming, all the microcontrollers, regardless of what module they are connected to, run the same code. They are then configured to do certain tasks by the main server. This modularity also allows us to swap the microcontrollers onto different modules, in the case one break, without having to load new software.

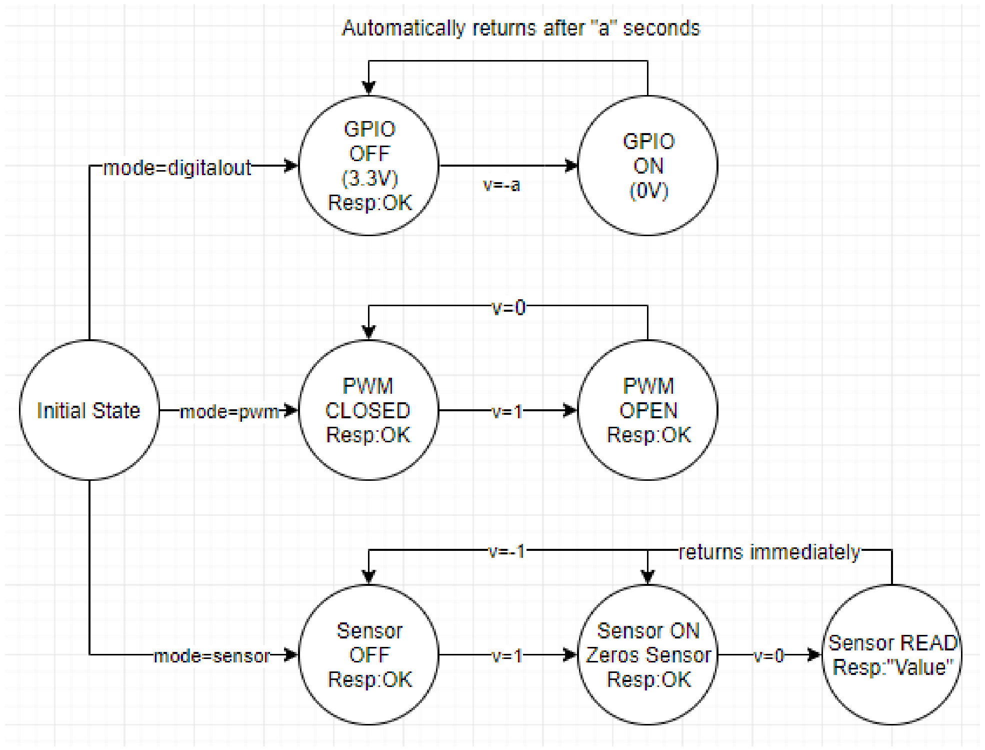


Figure 8. Microcontroller State Diagram

2.7 Web App / Server

The software routes information between the main server, the microcontrollers controlling the dispensers, and the web interface. All routing and information of transfer will be done via HTTP and thus TCP to ensure accurate retrieval of information and instructions.

2.7.1 Software Design Procedure

The software logic for the main server evolved as we continually tested with the entire dispensing procedure and finding ways to improve its efficiency. This was what put all the modules together and ran the entire dispensing sequence. Thus, we had to do this last, after all the modules and microcontroller software were usable. We were all familiar with Python, which is why we used it along with the Flask Web Framework.

We first manually found all the IPs of each individual microcontroller and then tested each module by sending http requests from our web server. We then were able to accurately measure the rate of liquid flowing through the solenoid valve when it was open. Thus, through trial and error we were able to find that opening the valve for 0.1 seconds was around 10g or 10mL. For solid dispensing, we tried out different waiting times for the tapioca pearls to drop down into the hole in the tube. Throughout this process, we also realized the microcontroller wasn't powerful enough for the main server to constantly send http requests to the one controlling the load sensor, since it crashed once we did. Thus, this changed how we interfaced with microcontrollers such that we would make as little requests as possible, which required updates to the microcontroller software. Specific examples included sending the liquid dispenser a time of how long to open instead of separate commands of "open" and "close" and the use of "ticks" which were rounds in the main server logic that occurred every second, so we wouldn't overload the microcontrollers.

2.6.2 Software Design Details

2.6.2.1 High Level Architecture

A main web server, running on a personal computer, will serve the user-friendly web User interface, which will receive user input in the configuring the amount of liquids and solids to dispense. A web interface will interact with the main server through its REST API endpoints, which provides an easy interface to controlling the different modules including a mobile app for the future.

When the main server boots up, it will automatically search through the entire network to find the IPs of the given microcontrollers and then configure them for their respective modules through setting their `mode`, which was described in the microcontroller section. The user can automatically do this again, in cases including when one of the microcontrollers wasn't on at the time the main server booted up or the user needed to reboot one of the microcontrollers.

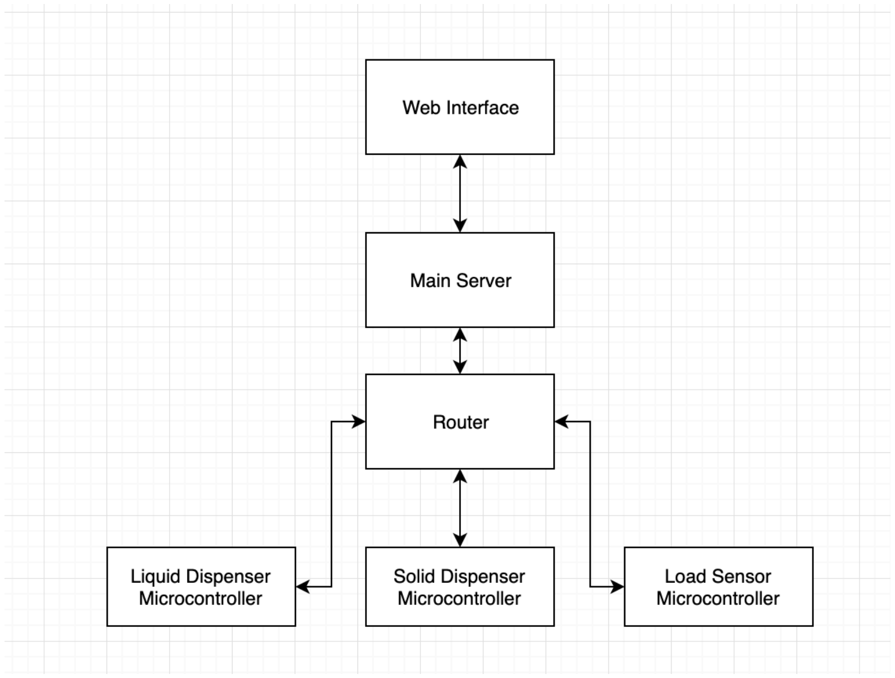
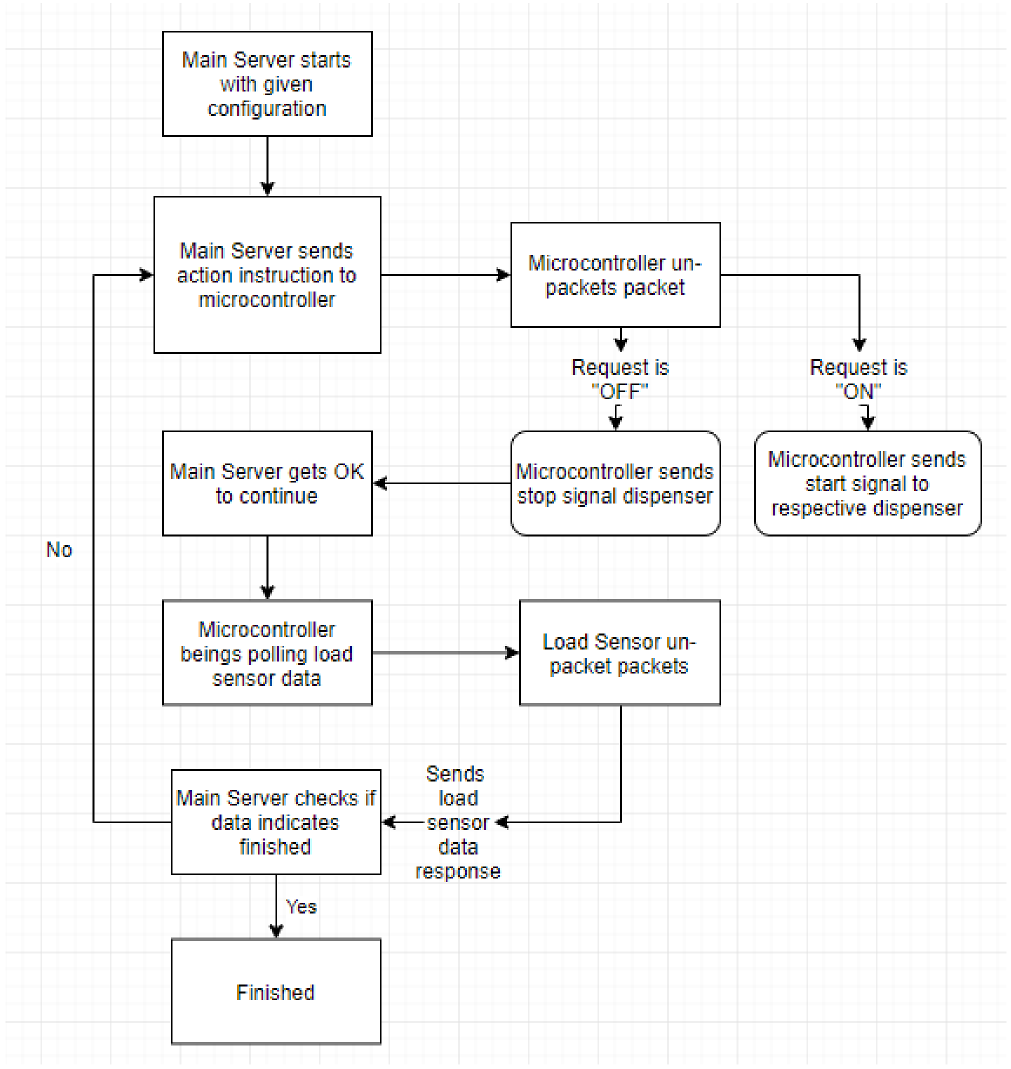

Web Interface
Main Server
Router
Liquid Dispenser Microcontroller
Solid Dispenser Microcontroller
Load Sensor Microcontroller

Figure 9. Software High Level Architecture

2.7.2.3 Routing and Dispensing Logic

Figure 10 describes the network transfer logic when a user starts the boba making process, beginning from the main server receiving that request (which includes the amount of milk tea and boba to dispense). Based on the configurations, it begins to give instructions to the respective microcontroller, starting with the solid dispenser module. This order - dispensing solids before liquids - prevents splashing.

For the solid dispensing, the configuration the user passes in is the number of "ticks" or rounds the solid dispenser will turn. As shown in the microcontroller section, there is an on or off state which determines the servo's rotation. It will wait 8 seconds for the pearls to slide into the holder before turning 180 degrees drop the pearls into the cup. It then waits for another 6 seconds. That is one "tick" or round. The default number of ticks is 3. We purposely wait because the pearls take a while to slide into the dispenser and to fall out of it (our video shows how slow it is).



Main Server starts with given configuration
Main Server sends action instruction to microcontroller
Microcontroller un-packets packet
Request is "OFF"
Request is "ON"
Main Server gets OK to continue
Microcontroller sends stop signal dispenser
Microcontroller sends start signal to respective dispenser
No
Microcontroller beings polling load sensor data
Load Sensor un-packet packets
Main Server checks if data indicates finished
Sends load sensor data response
Yes
Finished

Figure 10. Routing and Dispensing Logic

The liquid dispensing logic is more complex. For every tick, we first send a request for the load. Based on the current load and the desired load, we find a proportional rate to which the valve opens for. Initially, the rate is at 0.05 for the first 5 ticks to start off slow, which prevents the cup from being knocked around. It then goes to the default rate of 0.1 seconds. And once the load crosses the threshold load (we set to 20mL below the desired load), the proportional rate is the following equation, where *target* is the desired load set by the user, *current* is the current load, *threshold* as the threshold load in which the rate should slow down is:

$$rate = (target - current + 1)/((threshold + 1) * 10)$$

If the user wants 10mL, it would still work. **Figure 11** shows the total open time of the valve and load for a desire load of 200mL or 200g. Each dot represents a tick. You can then see a smaller rate (smaller open time) at the beginning and when the load reaches 180g (the threshold load), the rate slows down based on the equation above. Appendix B has the raw data table.

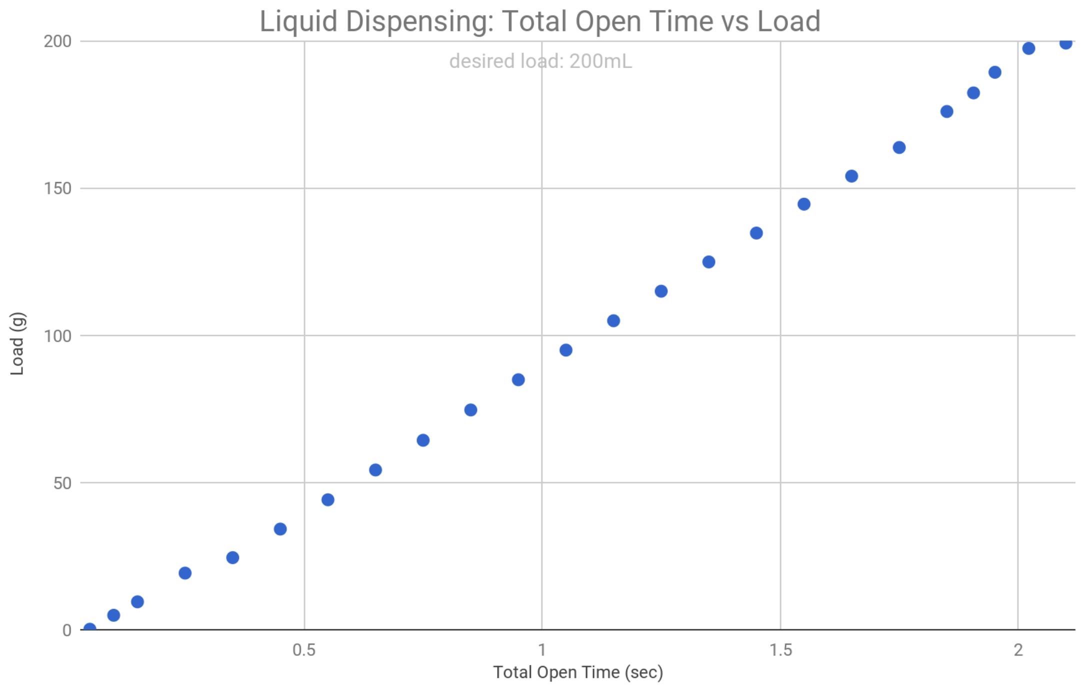


Figure 11. Liquid Dispensing: total open valve time vs load

To combat the deficiencies of networking (server crashing, network packet loss, etc), there is an upper-bound of 30 ticks (any cup bigger than the normal medium sized cup with 500mL doesn't require more than 30 ticks) on how long the liquid dispenser dispenses for and the microcontroller must return an acknowledge response back to the main server whenever it receives an instruction from it. If the request to the microcontroller times out, which usually means the microcontroller died, the entire process will stop through a python exception as a safety precaution. In addition, there is a STOP button on the interface that overrides and stops all actions.

3 Verification

3.1 Liquid Dispenser Module

These readings correspond to the final schematic in **Figure 2.** The voltage $V_{GSDriverFET}$ is applied using a lab bench, and the solenoid valve is connected across the ports Solenoid1 and Solenoid2. As stated in the datasheet [6], any $V_{Valve} > 6V$ will open the valve.

Table 1. Multimeter Readings for Liquid Dispensing Driver Circuitry

$V_{GSDriverFET}$	$V_{GSValveFET}$	$V_{DSValveFET}$	V_{Valve}
0V	~5.8V	~1V	~11V
3.3V	0V	12V	0V

3.2 Solid Dispenser Module

Since there was no need to implement a level shifter in our final design, we did not need to verify the servo control circuitry. We connected the servo's control line directly to the microcontroller GPIO and applied the PWM signal calculated **2.4.2.1**. However, this did not allow the servo to rotate a full 180 degree. Below is the final duty cycle values that allowed for full 180 degree rotation.

$$minDuty = 35 \, maxDuty = 125$$

3.3 Weight Sensing Module

The load cell output increases as weight increases, as required. Further, the output increases linearly as weight increases.

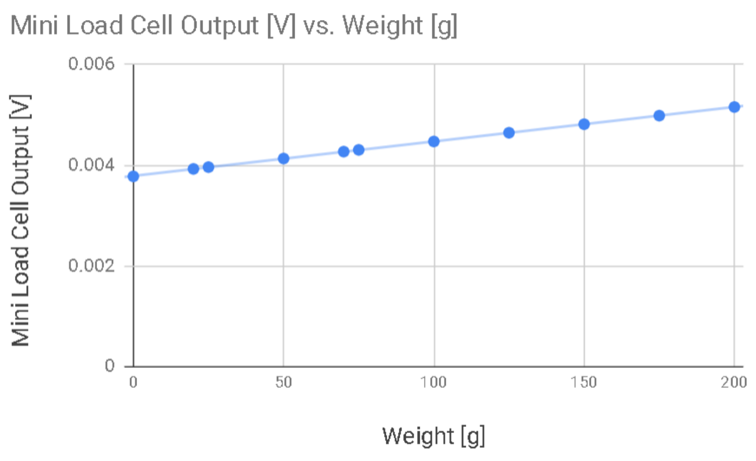


Figure 12. Multimeter Readings for Load Cell

4 Costs and Schedule

4.1 Cost Analysis

4.1.1 Labor

We assume a reasonable salary of \$50 / hour. Further, we estimate that each group member will work an average of 12 hours a week for 15 weeks. Therefore, the estimated labor cost is

$$\frac{\$50}{hr} * 2.5 * (\frac{12\ hrs}{1\ week} * 15\ weeks) * (3\ partners) = \$67{,}500$$

We also estimate a salary of \$50 / hour for the machine shop, who worked for 5 weeks. Therefore, the machine shop labor cost is

$$\frac{\$50}{hr} * 2.5 * (\frac{12\ hrs}{1\ week} * 5\ weeks) * (1\ worker) = \$7{,}500$$

4.1.2 Parts

Part	**Cost**
Liquid Dispensing Mechanism	**\$42.58**
Brass Liquid Solenoid Valve - 12V (Digikey; 1528-1280-ND)	\$24.95
RFP30N06LE N-Channel MOSFET (Amazon)	\$0.99
DCJ0202 Barrel Jack (Addicore)	\$0.65
1N4004 Diode (Addicore)	\$0.15
Chanzon 12V 5A 60W AC DC Power Supply	\$15.84
Solid Dispensing Mechanism	**\$162.91**
Towerpro SG92 Servo (Digikey; 1528-1076-ND)	\$5.95
BSS138 - SMD (Digikey; BSS138CT-ND)	\$0.31
DCJ0202 Barrel Jack (Addicore)	\$0.65
120VAC to 5V 2A DC Power Converter (Amazon)	\$6.00
3D Printed Solid Dispenser	\$150

Weight Sensing Mechanism	**$27.85**
Load Cell - 500g (Digikey; 1568-1899-ND)	$11.25
Load Cell Amplifier HX711 (Digikey; 1568-1436-ND)	$9.95
DCJ0202 Barrel Jack (Addicore)	$0.65
120VAC to 5V 2A DC Power Converter (Amazon)	$6.00
Microcontroller x 3	**$32.10**
ESP8266 ver. ESP-01 (Addicore)	$3.95
Traco Power TSR 1-2433 (Digikey; 1951-2742-ND)	$6.14
6x6x5mm Pushbuttons (Addicore)	$0.10
DPDT DIP Slide Switches (Ebay)	$0.40
10uF Electrolytic Capacitors (Addicore)	$0.11
Non-Electrical Components	**$30.00**
General Structure	$30.00
Total	**$265.44**

4.2.3 Grand Total

$$Grand\ total\ cost\ = labor + parts = \mathbf{\$75{,}265.44}$$

5 Conclusions

5.1 Accomplishments

We were successful in satisfying all our high-level requirements and requirements & verification for all of our modules. We had two dispensers, one for liquid and one for the solids (tapioca pearls) and had a web interface to allow the user to customize the desired amount of liquid/solid to dispense. In addition, we were able to dispense the liquid within ±1mL, which surpassed our requirement of ±10% error in mass tremendously and were able to get within 1mL when trying to just dispense 10mL. We were also able to have a modular design, with the same microcontroller software and board for the liquid, solid, and load modules, proving the easy extensibility. Finally, we were able to fully dispense a milk tea with tapioca pearl drink, from plugging in the machine, to starting the server, to starting the sequence through the web dashboard, to the finished drink.

The design of the solid dispenser could be better though. Some issues we encountered were the high cost, leaks due to loose tolerances, tapioca pearls clumping up, and pearls blocking each other. These issues could be alleviated through better mechanical design, which was not our forte.

5.3 Ethical Concerns

While working on this project, we abided by the IEEE Code of Ethics and the ACM Code of Ethics in their entirety. For this project, it is important to commit ourselves to #1 of the IEEE Code of Ethics: "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment"[11].

Throughout the process, we have followed the safety precautions mentioned in our Design Document. For instance, we will use certified transformers to convert 120 V AC to 5 V DC and 12 V DC. In addition, tapioca pearls, are made of tapioca starch and can be safety kept at room temperature for 10 hours[12]. Boba may be coated with honey, which is itself antibacterial because of its high osmolarity (concentration), high acidity, and presence of hydrogen peroxide[13]. On the other hand, room temperature milk tea can only be safety kept at room temperature for 2 hours[14]. We were careful to let any person drink any drink dispensed by this dispenser, even if it the pearls or milk tea weren't expired, just as a safety precaution. [Citation error]

5.4 Acknowledgements

Our team would like to thank our teaching assistant John Kan for his continuous guidance on this project throughout the semester. In addition, we would like to thank the ECE445 Professors for organizing this course. We hope to continue the project and commercialize if the opportunities arise.

References

[1] "Everything You Need to Know About Bubble Tea," *Sous Chef*. [Online]. Available: https://www.souschef.co.uk/blogs/the-bureau-of-taste/everything-you-need-to-know-about-bubble-tea. [Accessed: 01-May-2019].

[2] "Milk Tea& Iced Tea," *Sunny's BoBa*. [Online]. Available: http://sunnysboba.com/milk-tea--iced-tea.html. [Accessed: 01-May-2019].

[3] "TIP120 Datasheet." [Online]. Available: https://cdn-shop.adafruit.com/datasheets/TIP120.pdf. [Accessed: 01-May-2019].

[4] P. Pieter, "A Beginner's Guide to the ESP8266." [Online]. Available: https://tttapa.github.io/ESP8266/Chap04%20-%20Microcontroller.html. [Accessed: 01-May-2019].

[5] "RFP30N06LE Datasheet." [Online]. Available: https://www.sparkfun.com/datasheets/Components/General/RFP30N06LE.pdf. [Accessed: 01-May-2019].

[6] "Brass Liquid Solenoid Valve - 12V - 1/2 NPS." [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/996_Web.pdf. [Accessed: 01-May-2019].

[7] "HS-311 Servo." [Online]. Available: https://www.servocity.com/hs-311-servo. [Accessed: 01-May-2019].

[8] "How Does a Servo Work?" [Online]. Available: https://www.servocity.com/how-does-a-servo-work. [Accessed: 01-May-2019].

[9] "24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales." [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf. [Accessed: 01-May-2019].

[10] "MicroPython tutorial for ESP8266 — MicroPython 1.10 documentation." [Online]. Available: https://docs.micropython.org/en/latest/esp8266/tutorial/. [Accessed: 01-May-2019].

[11] "IEEE Code of Ethics." [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 01-May-2019].

[12] "FAQ." [Online]. Available: https://www.bubbleteasupply.com/faq/. [Accessed: 01-May-2019].

[13] S. M. Manisha Deb Mandal, "Honey: its medicinal property and antibacterial activity," *Asian Pac. J. Trop. Biomed.*, vol. 1, no. 2, p. 154, Apr. 2011.

[14] “Danger Zone.” [Online]. Available: https://www.fsis.usda.gov/wps/portal/fsis/topics/food-safety-education/get-answers/food-safety-fact-sheets/safe-food-handling/danger-zone-40-f-140-f/CT_Index. [Accessed: 01-May-2019].

Appendix A

Table 1. System Requirements and Verifications

Requirement	Verification	Verification Status
Power Supply: 120 VAC to 12 VDC Transformer 1. Converts 120 VAC to 12 VDC.	**Power Supply: 120 VAC to 12 VDC Transformer** 1A. Connect the input of the power supply to a wall outlet outputting 120 VAC. 1B. Measure the output voltage using an oscilloscope. Check that the output voltage is in the range 12 V +/- 5%.	1. Y
Power Supply: 120 VAC to 5 VDC Transformer 1. Converts 120 VAC to 5 V DC.	**Power Supply: 120 VAC to 5 VDC Transformer** 1A. Connect the input of the power supply to a wall outlet outputting 120 V AC. 1B. Measure the output voltage using an oscilloscope. Check that the output voltage is in the range 5 V +/- 5%.	1. Y
Liquid Dispensing Mechanism 1. The dispensing mechanism must be able to dispense 10 mL +/- 5% milk tea in 0.1 second of opening.	**Liquid Dispensing Mechanism** 1A. Connect it to a microcontroller and open it for 0.1 second. 1B. Measure whether the amount of dispensed material is accurate.	1. Y
Liquid Dispensing Mechanism: Solenoid Valve 1. Valve must open when 12 V +/- 5% is applied. 2. Valve must be gravity fed (allow flow of liquid by gravity alone when open), as no	**Liquid Dispensing Mechanism: Solenoid Valve** 1A. Apply 12V DC to the valve's input. 1B. Ensure that the valve opens fully within 1 second of verification 1A.	1. Y 2. Y

other liquid pressurizing mechanism is in place.	2. During verifications 1A and 1B, place a liquid container above the valve and attach it. Ensure that liquid both enters and exits the valve.	
Liquid Dispensing Mechanism: Solenoid Control 1. Must be able to output 12 V +/- 5% and 3 A +/- 5% to the solenoid valve.	**Liquid Dispensing Mechanism: Solenoid Control** 1A. Apply 3.3V to transistor base. 1B. Measure the output voltage and current using an oscilloscope. Check that the output voltage is in the range 12 V +/- 5% and check that the output current is in the range 3 A +/- 5%.	1. Y
Solid Dispensing Mechanism 1. The solid dispensing mechanism must dispense at least 1 tapioca pearls during 1 second of opening.	**Solid Dispensing Mechanism** 1A. Connect it to a microcontroller and open it for one second. 1B. Measure the amount of dispensed pearls.	1. Y
Solid Dispensing Mechanism: Servo and Door 1. Servo must provide enough torque to rotate the tube. 2. Servo must be able to rotate 180 degrees. 3. Door must be 1” wide.	**Solid Dispensing Mechanism: Servo and Door** 1A. Connect the servo to the PVC tube and apply a 1ms pulse of 3.3 V DC to the servo. Ensure that the PVC tube rotates easily. 2A. Repeat verification 1A, but with a pulse of 2.4ms. Ensure that the PVC tube rotates any amount greater than or equal to 180 degrees. 3A. Measure the door with ruler and verify whether it’s 1” wide.	1. Y 2. Y 3. Y
Weight Sensing Mechanism: Load Sensor 1. Sensor must be able to produce a signal < 1 V that increases as the weight on the cell increases.	**Weight Sensing Mechanism: Load Sensor** 1A. Rigidly attach the load cell to two plates. 1B. Connect the load cell’s red wire to 5 V and its black wire to ground. 1C. Connect the load cell’s green and white wires to an oscilloscope.	1. Y

	1D. Apply increasing weight on top of the load cell. The oscilloscope reading should increase as the weight on the cell increases.	
Weight Sensing Mechanism: Load Sensor ADC 1. Must be able to read voltages less than 0.5 V from a load cell and convert it to a digital value that a microcontroller can read.	**Weight Sensing Mechanism: Load Sensor** 1A. Connect the ADC circuit to the load cell and to a microcontroller. 1B. Apply 25 pulses to pin CLK. The microcontroller should be able to obtain a number from the DATA pin that increases as the weight on the load cell increases.	1. Y
Control Unit: Microcontroller 1. Microcontroller connected to network and can communicate with laptop. 2. Set a HIGH signal (3.3 V) on GPIO 2 when a Request “v=1” is sent to the microcontroller. Respond “OK”. 3. Set a LOW signal (0 V) on GPIO 2 when a Request“v=0” is sent to the microcontroller. Respond “OK”. 4. Set a PWM signal MIN (pulse width 1 ms) for 2 seconds, then off on GPIO 2 when Request “PWMv=0” is received. Respond “OK”. 5. Set a PWM signal MAX (pulse width 2 ms) for 2 seconds, then off on GPIO 2 when Request “PWMv=1” is received. Respond “OK”. 6. Read value from GPIO 2 when Request “SENSOR,v=0” is received. Respond with value.	**Control Unit: Microcontroller** 1A. Ping microcontroller IP from laptop. B. Verify response 2A. Send “v=1” to microcontroller from laptop B. Probe GPIO 2. Check for 3.3 V 3A. Send “v=0” to microcontroller from laptop B. Probe GPIO 2. Check for 0 V 4A. Send “PWM,v=1” to microcontroller from laptop B. Probe GPIO 2 with oscilloscope. Check for 3.3V PWM with duty cycle 50% at freq = 50hz. 5A. Send “PWM,v=0” to microcontroller from laptop B. Probe GPIO 2 with oscilloscope. Check for 3.3 V PWM with pulse width 13% at freq = 50hz. 6A. Send “v=0” to microcontroller from laptop B. Verify response value is proportional to weight on sensor by varying weight.	1. Y 2. Y 3. Y 4. Y 5. Y 6. Y

Control Unit: Control Unit Power Delivery 1. Must supply 3.3 V DC at 1 A +/- 5%	**Control Unit: Control Unit Power Delivery** 1A. Attach V_In and GND to test bench power supply, vary input voltage between 4.5 V to 12.5 V. B. Probe V_Out with oscilloscope and insure stable 3.3 V output. C. Attach microcontroller, ensure stable 3.3 V under load.	1. Y
Control Unit: Web Server 1. Must be able to send HTTP requests through port 80.	**Control Unit: Web Server** 1A. Run server, check if it’s using port 80. Send a request to the server and it should send back a response	1. Y

Appendix B

Liquid Dispensing Logic Data for a desired load of 200g with final load of 299.22g. Tick is the round. Rate is the amount of time the solenoid valve is open for.

Tick	Rate (sec)	Total Time (sec)	Load (g)
0	0.05	0.05	0.194
1	0.05	0.1	4.93
2	0.05	0.15	9.5
3	0.1	0.25	19.25
4	0.1	0.35	24.5
5	0.1	0.45	34.2
6	0.1	0.55	44.15
7	0.1	0.65	54.27
8	0.1	0.75	64.35
9	0.1	0.85	74.65
10	0.1	0.95	84.91
11	0.1	1.05	94.97
12	0.1	1.15	104.94
13	0.1	1.25	114.97
14	0.1	1.35	124.91
15	0.1	1.45	134.71
16	0.1	1.55	144.53
17	0.1	1.65	154.05
18	0.1	1.75	163.77
19	0.1	1.85	175.99

20	0.056	1.906	182.305
21	0.045	1.951	189.31
22	0.071	2.022	197.411
Final		2.1	199.22