

# Coffee Barista

B.E.Sc Electrical and Computer Engineering

ECE 2242, Principles of Design

Jadyn Powell || 251175024

Dilpreet Bhambra || 251158848

Professor Lyudmil Marinov

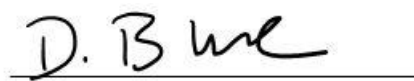
March 2022

## Declaration Statement

The project, report, and all other work included is our own, except where otherwise referenced.



Jadyn Powell



Dilpreet Bhambra

## Abstract

The purpose of this project was to design, test, and develop a circuit for an automated Coffee Barista using an Arduino Uno microcontroller board and a wide variety of sensors and actuators. The design had to include a minimum of four sensors and three actuators, and was to be completed within five weeks, excluding the initial ideation process. A Printed Circuit Board (PCB) was also designed, manufactured, and tested, to simplify the circuitry of the Coffee Barista. The Coffee Barista is an automated milk and sugar dispenser, with quantities that are specified by the user using pushbuttons and includes an LCD screen that displays the specified quantities. Two servo motors are used to dispense the milk/sugar. Once the amount of milk/sugar has been selected, the motors will be used to open their respective reservoirs for a specific amount of time based on the quantity selected. An ultrasonic sensor is used to detect the presence of a cup and will prevent any milk and/or sugar from being dispensed if there is no cup detected. The device also monitors the quantity of milk within the storage container using a weight sensor and alerts the user via a Bluetooth connection to their phone as to when the milk is running low. This same alert is displayed on the LCD screen and the servo motors are prohibited from turning, preventing any milk dispensing until the container is refilled. The temperature of the milk is also monitored using a temperature probe sensor, and another alert is sent to the user, via the same method, when the milk temperature rises above a safe storage level. Again, the same alert is displayed on the LCD screen and the servo motors are prohibited from turning, preventing the user from dispensing any milk that does not fall within the USDA standards for safe milk storage. After the five-week timeline, the circuit for the Coffee Barista was fully functional, and all sensors and actuators operated as expected. The PCB that was designed, was also printed and tested, and proved to be fully functional. The final results of the Coffee Barista were compared to the constraints determined at the beginning of the project, and it was deemed successful.

## Acknowledgments

Thank you to Professor Lyudmil Marinov, TA Thakshanth Uthayakumar, and TA Liwei Wang for their help, input, and guidance throughout this project. Thank you to the Electrical and Computer Engineering Department at Western University for providing some of the components necessary to create this circuit.

## Table of Contents

List of Figures.....	4
Glossary .....	5
Introduction.....	6
Problem Definition.....	6
Aims and Objectives .....	6
Structure .....	6
Literature Review .....	7
Design Constraints and Standards .....	8
Design Constraints .....	8
Design Attributes .....	8
Three Project Design Ideas .....	8
Weighting of Customer Needs .....	9
Decision Matrix .....	10
Design Standards .....	11
Design.....	11
Design Complexity .....	11
TinkerCad.....	15
Pushbuttons .....	16
Servo Motors .....	16
Ultrasonic Sensor .....	17
LCD Screen and I2C Serial Adaptor .....	19
Temperature Probe Sensor .....	20
Weight Sensor.....	21
Bluetooth Module.....	25
Schematic and PCB.....	26

<b>Implementation .....</b>	<b>27</b>
<b>Prototype.....</b>	<b>27</b>
<b>Pushbuttons and LCD Screen.....</b>	<b>29</b>
<b>Servo Motors .....</b>	<b>29</b>
<b>Ultrasonic Sensor .....</b>	<b>30</b>
<b>Temperature Probe Sensor and Bluetooth Module .....</b>	<b>31</b>
<b>Weight Sensor.....</b>	<b>32</b>
<b>Technical Testing and Debugging Procedure.....</b>	<b>33</b>
<b>Issue 1: Bluetooth Module not Connecting to Phone .....</b>	<b>33</b>
<b>Issue 2: Notification not Sending to Phone .....</b>	<b>34</b>
<b>Issue 3: Temperature Sensor Probe Not Sending Alert via Bluetooth Module.....</b>	<b>39</b>
<b>Issue 4: Code Persistently Not Uploading to Arduino Uno Board .....</b>	<b>42</b>
<b>Issue 5: Weight Sensor Not Reading Values.....</b>	<b>43</b>
<b>Issue 6: Weight Sensor Only Reading Value of Zero.....</b>	<b>46</b>
<b>Issue 7: Weight Sensor Values Not Accurate .....</b>	<b>47</b>
<b>Issue 8: Weight Sensor Values Not Sending Alert via Bluetooth Module.....</b>	<b>50</b>
<b>Issue 9: Bluetooth Module not Sending Alert.....</b>	<b>50</b>
<b>Issue 10: Weight Sensor Only Reading Value of Zero .....</b>	<b>51</b>
<b>Results and Evaluation .....</b>	<b>51</b>
<b>Future Work.....</b>	<b>52</b>
<b>Conclusion .....</b>	<b>52</b>
<b>Appendix.....</b>	<b>54</b>
<b>User Manual .....</b>	<b>54</b>

## **List of Figures**

Figure 1: Newco Coffee Companion. Adapted from [3]. .....	7
Figure 2: Analytical Hierarchy Process of Customer Needs .....	10
Figure 3: Decision Matrix of Three Alternative Ideas.....	10
Figure 4: Circuit Block Diagram .....	12
Figure 5: Algorithm of Coffee Barista.....	13
Figure 6: Prototype Design .....	14
Figure 7: Coffee Barista Simulation .....	15
Figure 8: LCD-I2C Milk Method Code .....	16
Figure 9: Initializing Servo Motor .....	17
Figure 10: if-else-if statement.....	17
Figure 11: Ultrasonic Sensor Working Principle. Adapted from [5]......	18

Figure 12: Initializing the Ultrasonic sensor.....	18
Figure 13: Ultrasonic Sensor Void Setup Code.....	18
Figure 14: Ultrasonic Sensor Main Body Code.....	18
Figure 15: Ultrasonic Sensor Main Body Code.....	19
Figure 16: LCD-I2C Initialization Code and Variables.....	19
Figure 17: Temperature Sensor Initialization Code and Variables.....	20
Figure 18: Temperature Sensor Void Setup Code.....	20
Figure 19: Temperature Sensor Main Body Code.....	20
Figure 20: Weight Pads and Amplifier Schematic. Adapted from [5]......	21
Figure 21: Portion of HX711 Calibration code returning the Calibration Factor.....	22
Figure 22: Weight Sensor Initialization code.....	22
Figure 23: Weight sensor void setup code.....	23
Figure 24: Weight sensor main body code .....	24
Figure 25: Dabble Setup Code.....	25
Figure 26: Dabble Main Body Code.....	25
Figure 27: Arduino Shield Schematic.....	26
Figure 28: Arduino Shield PCB.....	27
Figure 29: Prototype Build.....	28
Figure 30: Prototype Circuitry.....	28
Figure 31: Algorithm for pushbuttons, servos, lcd, ultrasonic.....	31
Figure 32: Algorithm of temperature sensor and Bluetooth module.....	32
Figure 33: Algorithm for weight sensor and Bluetooth module .....	33
Figure 34: Altered Bluetooth Debugging Code.....	34
Figure 35: Altered Bluetooth Notification Debugging .....	35
Figure 36: Second Altered Bluetooth Notification Code.....	36
Figure 37: Altered Bluetooth Pin Code .....	37
Figure 38: Example Bluetooth Notification Code .....	38
Figure 39: Altered Library Bluetooth Notification Code .....	39
Figure 40: Altered Temperature Sensor Notification Code.....	40
Figure 41: Temp Cast as Float Temperature Sensor Debugging Code.....	41
Figure 42: Altered Weight Sensor Debugging Code .....	44
Figure 43: Alternate Weight Sensor Library .....	45
Figure 44: Example Weight Sensor Code.....	45
Figure 45: Portion of Calibration Factor Code .....	47
Figure 46: Correct Calibration Factor Code .....	48
Figure 47: Tare added to Weight Sensor Code .....	49
Figure 48: Altered Weight Threshold Value .....	50
Figure 49: Design Constraint Evaluation.....	51
Figure 50: Customer Constraint Evaluation.....	51

## Glossary

### Arduino Peltier:

A thermoelectric cooler, which can be used as a heater or a cooler when a current is applied [1].

United States Department of Agriculture (USDA):

Department of the United States government which provides leadership on food, agriculture, natural resources, rural development, nutrition, and related issues based on public policy, the best available science, and effective management [2].

LCD: Liquid Crystal Display

## **Introduction**

### **Problem Definition**

Many people rely on and look forward to starting their morning off with a fresh cup of hot coffee. However, it can be difficult to get the correct proportions of milk/sugar as per an individual's preference and testing the coffee poses a risk of burning one's mouth/tongue since the coffee is hot. By the time the coffee has cooled to a drinkable temperature, it may be too late to add additional milk/sugar.

Creating a system to automatically dispense the correct proportions of milk/sugar which can be personalized to each person's individual preferences will alleviate this issue and risk.

### **Aims and Objectives**

The purpose of this project was to design, test, and develop a circuit for an automated Coffee Barista using an Arduino Uno microcontroller board and a wide variety of sensors and actuators. The design had to include a minimum of four sensors and three actuators, and was to be completed within five weeks, excluding the initial ideation process.

### **Structure**

This report will further examine the above problem, and upon an in-depth analysis of customer needs a final decision will be made regarding the project. It will detail the preliminary planning process of the project design, followed by a thorough examination of the circuit, and each of the sensor and actuators. The implementation of the circuit, testing/debugging process, and the final Printed Circuit Board (PCB) will also be detailed. A final review of the project will also be conducted, while commenting on any future work identified for the Coffee Barista.

## Literature Review

Developing a system which automatically dispenses the correct proportions of milk/sugar, which can be personalized to each person's individual preferences, will alleviate this issue and risk of not being able to reliably get the correct proportions of milk/sugar and the risk of burning one's mouth/tongue while testing hot coffee.

There are pre-existing milk/sugar dispensers for coffee, such as the Newco Coffee Companion, seen in Figure 1 below.



Figure 1: Newco Coffee Companion. Adapted from [3].

These are intended for large scale, industrial use which does not solve the issue for users at home and in smaller locations. Since the storage size of the milk and sugar can be changed to fit individual needs, whether it is a much smaller scale for at home use, or much larger scale for use in a cafeteria, the Coffee Barista is able to reach a wide target audience.

Many of the existing designs also do not have any way of ensuring no mess is made, other than a catch basin which the cup is placed on top of. The Coffee Barista is designed with an ultrasonic sensor, which will prevent any milk and/or sugar from being dispensed if there is no cup detected.

## **Design Constraints and Standards**

### **Design Constraints**

It was necessary to identify the constraints of the project in order to begin effectively working and move beyond the initial ideation process. Constraints are things that the project idea must have in order to be considered for the final decision. These are also things that may be used to evaluate the project at the end of the term to determine if the project was successful or not. The constraints were determined to be design complexity, feasibility, implementation, and cost. Did the project design idea meet the minimum required four sensors and three actuators, was the project appropriate given the time constraints and knowledge limits, was the project idea able to be effectively tested to debug and determine effectiveness of the circuit design and construction, and would the project have additional costs associated with the required materials that were within an acceptable range given the funds accessible? In terms of constraints for the customers of the project design, it must be a convenient solution to the problem, it must be easy to use, easy to maintain, and it must be compatible with any additional existing things customers will use with the product (i.e. a coffee cup with the Coffee Barista).

### **Design Attributes**

It was also necessary to identify the attributes of the project. Attributes are things that the project would ideally have but will not hinder the success of it if they are not present. The attributes were determined to be whether the project was appealing, easily maintainable, and multifunctional. Was the project visually appealing, easy to maintain when implemented, and did the project have multiple uses?

### **Three Project Design Ideas**

Three ideas were generated from the initial ideation process, a climate-controlled greenhouse, a fish tank monitor, and the Coffee Barista. The climate-controlled greenhouse monitored humidity, temperature, light, and soil moisture and was capable of automatically watering plants and turning on/off UV lights. This design would use four sensors, a temperature sensor, a soil moisture sensor, a humidity sensor and a photoresistor. It would also use three actuators, a UV LED, a water pump, and an LCD display. The estimated cost of this project would be approximately \$60, as it would be necessary to purchase a soil moisture sensor, humidity sensor, UV LED, and water pump.



The fish tank monitor was a tank water level, temperature, and quality monitor with automated feeding two times per day. It would use four sensors, a proximity sensor, temperature sensor, water quality sensor and a clock, and two actuators, LCD screen and DC motor. The estimated cost of this project would be approximately \$20, as it would be necessary to purchase a water quality sensor.

The Coffee Barista design idea was an automated milk and sugar dispenser, with quantities that are specified by the user using pushbuttons and includes an LCD screen that displays the specified quantities. Two servo motors are used to dispense the milk/sugar. Once the amount of milk/sugar has been selected, the motors would be used to open their respective reservoirs for a specific amount of time based on the quantity selected. An ultrasonic sensor would be used to detect the presence of a cup and will prevent any milk and/or sugar from being dispensed if there is no cup detected. The device would also monitor the quantity of milk within the storage container using a weight sensor and alerts the user via a Bluetooth connection to their phone as to when the milk is running low. The temperature of the milk would also be monitored using a temperature probe sensor, and another alert is sent to the user, via the same method, when the milk temperature rises above a safe storage level. This design would use four sensors, a temperature probe sensor, a weight sensor, an ultrasonic sensor, and two pushbuttons. It would also use three actuators, an LCD screen, two servo motors and the Bluetooth module. The approximate cost of this project would be approximately \$18, as it would be necessary to purchase the temperature probe sensor and weight sensor.

### **Weighting of Customer Needs**

Weighting of customer needs is a key factor to consider when developing the project. Without customers, there is no need to pursue the project design. The Analytical Hierarchy Process (AHP) was used to compare the customer needs of convenience, usability, maintainability, and compatibility, as seen in Figure 2: *Analytical Hierarchy Process of Customer Needs*. Convenience refers to how convenient of a solution the product is for the problem, usability refers to how easy the product is to use, maintainability refers to how easy the product is to maintain, and compatibility refers to how easy it is to use with any other additional existing things customers will use with the product (i.e. a coffee cup with the Coffee Barista). When using the AHP method, the left column of customer needs is compared to the row of customer needs, with the number

representing have much more or less the column need is to the row need. For example, Convenience is compared to usability and has a value of 1.5. This means that the convenience of the product is 1.5 times more important to customers than the usability of the product.

	Convenience	Usability	Maintainability	Compatibility	Total	Weighting
Convenience	1.0	1.5	1.2	2.4	6.10	0.338889
Usability	0.7	1.0	1.3	1.7	4.70	0.261111
Maintainability	0.8	0.8	1.0	2.1	4.70	0.261111
Compatibility	0.4	0.6	0.5	1.0	2.50	0.138889

Figure 2: Analytical Hierarchy Process of Customer Needs

From the AHP figure, it is clear that the convenience need is most important, followed by usability tied with maintainability, and finally compatibility. This means that the most important customer need of the chosen project is clearly solving the problem, followed by ease of use and maintainability, and finally compatibility with other devices.

### Decision Matrix

The three project design ideas, the greenhouse, the fish tank, and the Coffee Barista, were each individually evaluated against the project constraints in a decision matrix, as seen in Figure 3. This decision matrix is on a scale of 1-3, with 3 being the most desirable score, and 1 being the least desirable score.

	Design Complexity	Feasible	Implementable	Cost	SCORE
Greenhouse	3	2	1	1	7
Fish Tank Monitor	1	3	3	2	9
Coffee Barista	3	3	2	3	11

Scale 1-3

Figure 3: Decision Matrix of Three Alternative Ideas

It is clear from the decision matrix that the best project idea to consider is the Coffee Barista. This project also aligns with the weighting of customer needs, as the Coffee Barista is a clear and effective solution to the problem. It is also easy to use, easy to maintain and is compatible with almost any coffee cup.

## **Design Standards**

Since the Coffee Barista will be storing milk to dispense for coffee, it must be kept at acceptable standards. According to the United States Department of Agriculture (USDA), cows' milk must be kept at a temperature below 40°F, or 4.44°C [3] and is safe to store for approximately three weeks [4]. The design of the product must take steps to ensure that these standards are met.

## **Design**

### **Design Complexity**

The Coffee Barista consisted of four sensors and three actuators. It made use of the DS18B20 temperature probe sensor, the HX711 weight sensor, the ultrasonic sensor, and two pushbuttons. The three actuators used were the LCD screen, two servo motors, and the HC-06 Bluetooth module. The general function of each of these sensors and actuators can be seen in Figure 4 below.

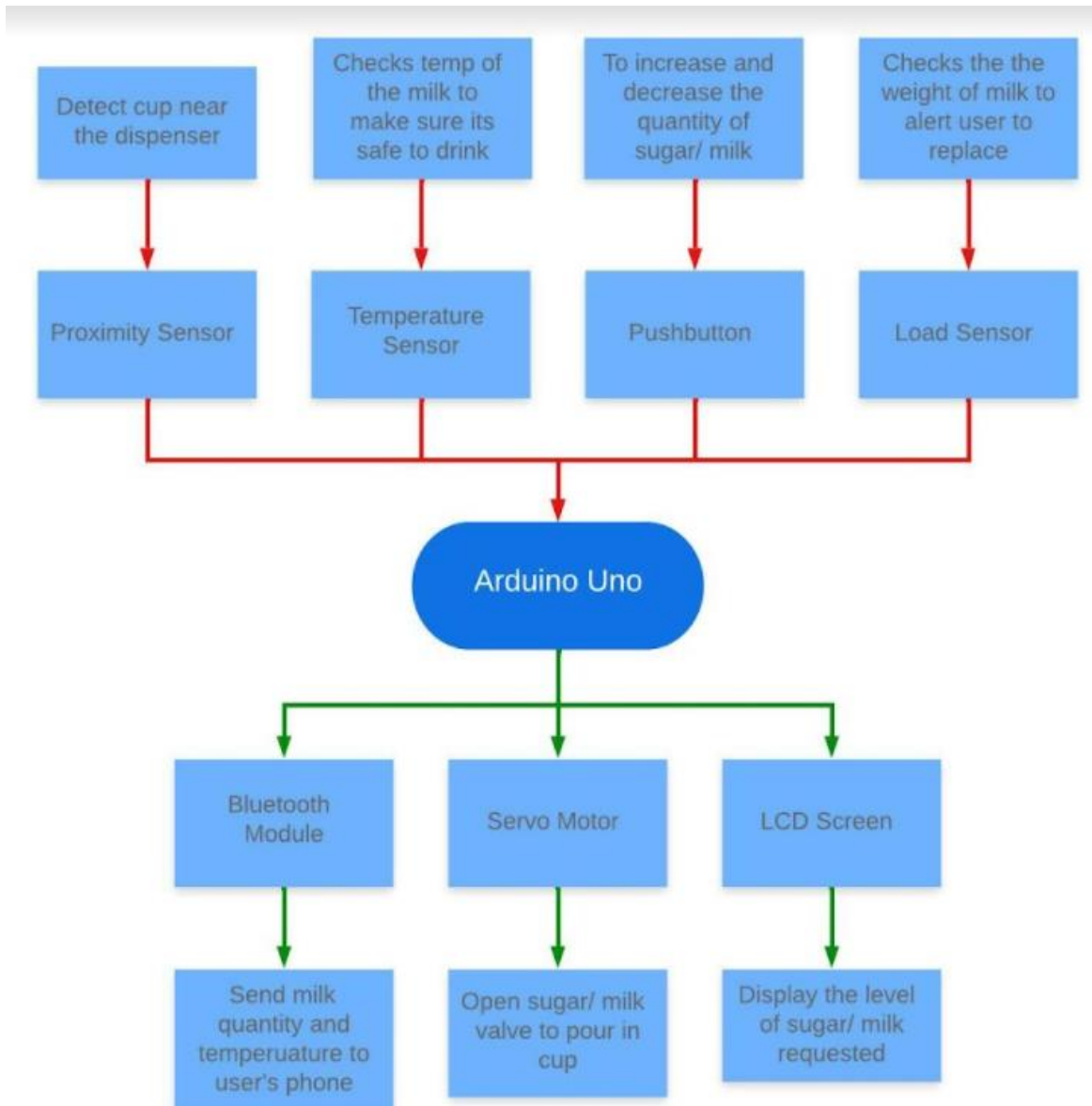


Figure 4: Circuit Block Diagram

These sensors and actuators followed the algorithm seen below in Figure 5.

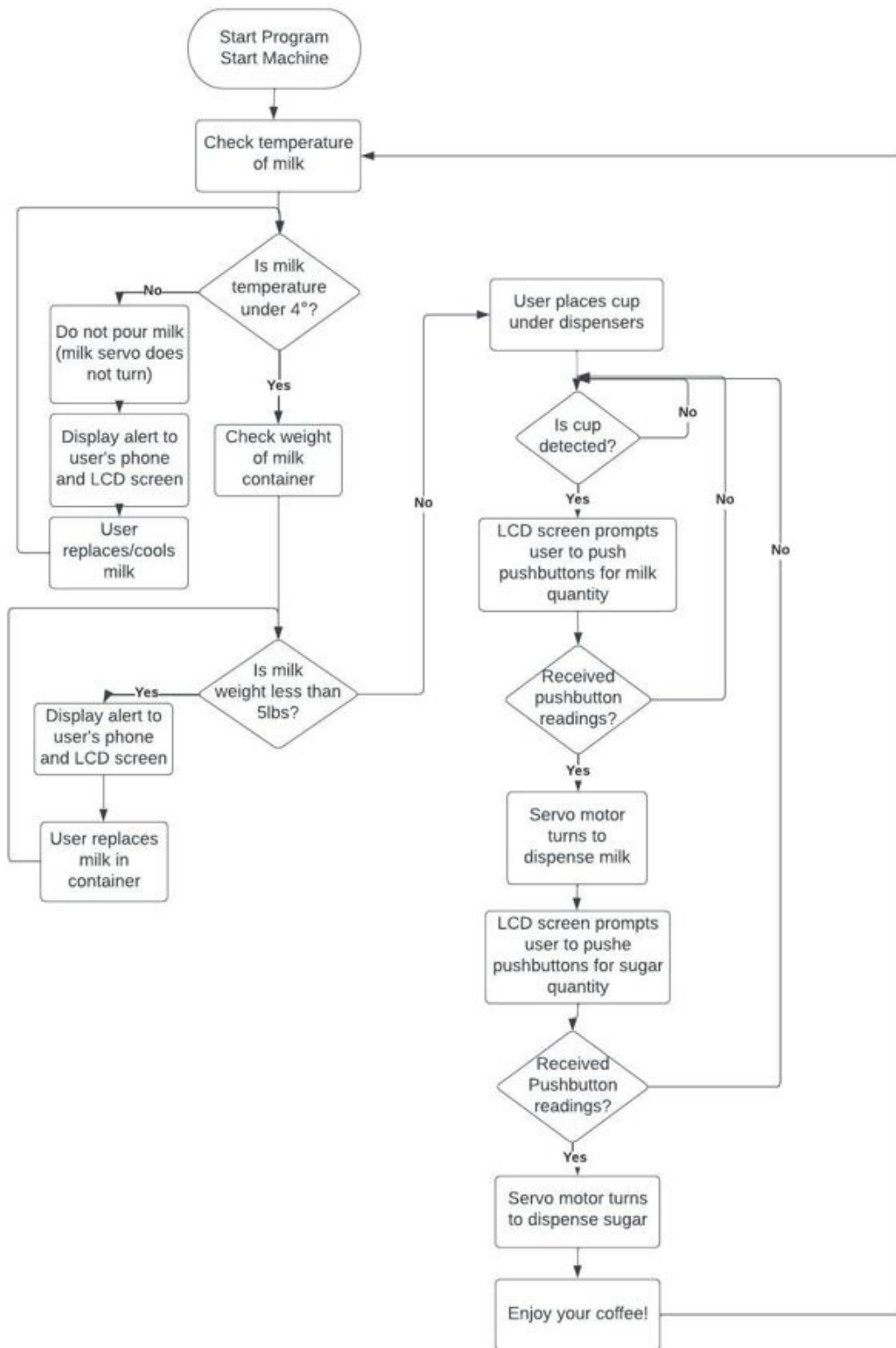


Figure 5: Algorithm of Coffee Barista

A prototype was also designed as seen below in Figure 6. This design did not include an explicit method for dispensing the milk/sugar, however every other component of the circuit was realized.

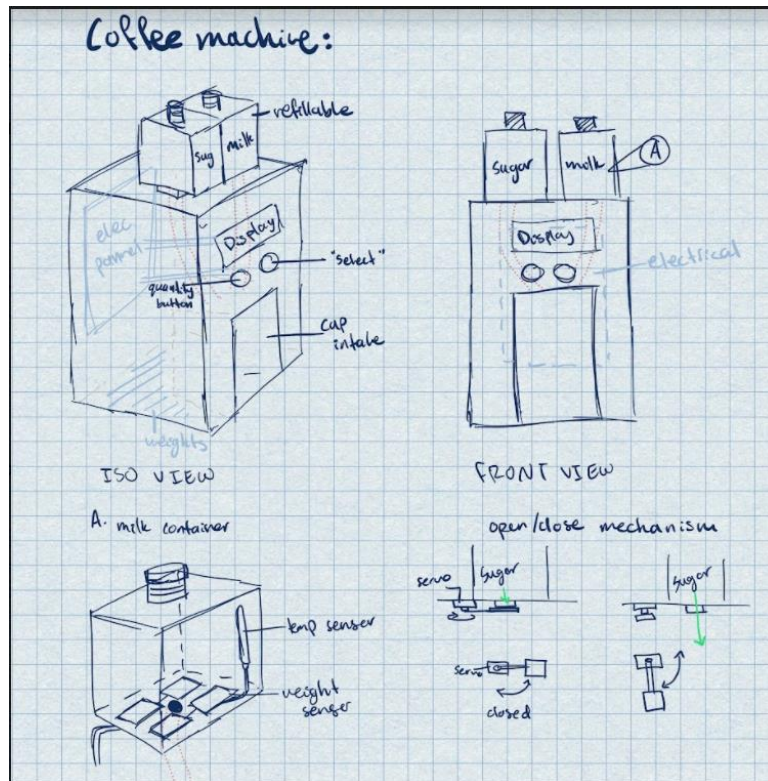


Figure 6: Prototype Design

## TinkerCad

Based on the above block diagram and algorithm, a TinkerCad simulation was created, as seen in Figure 77 below.

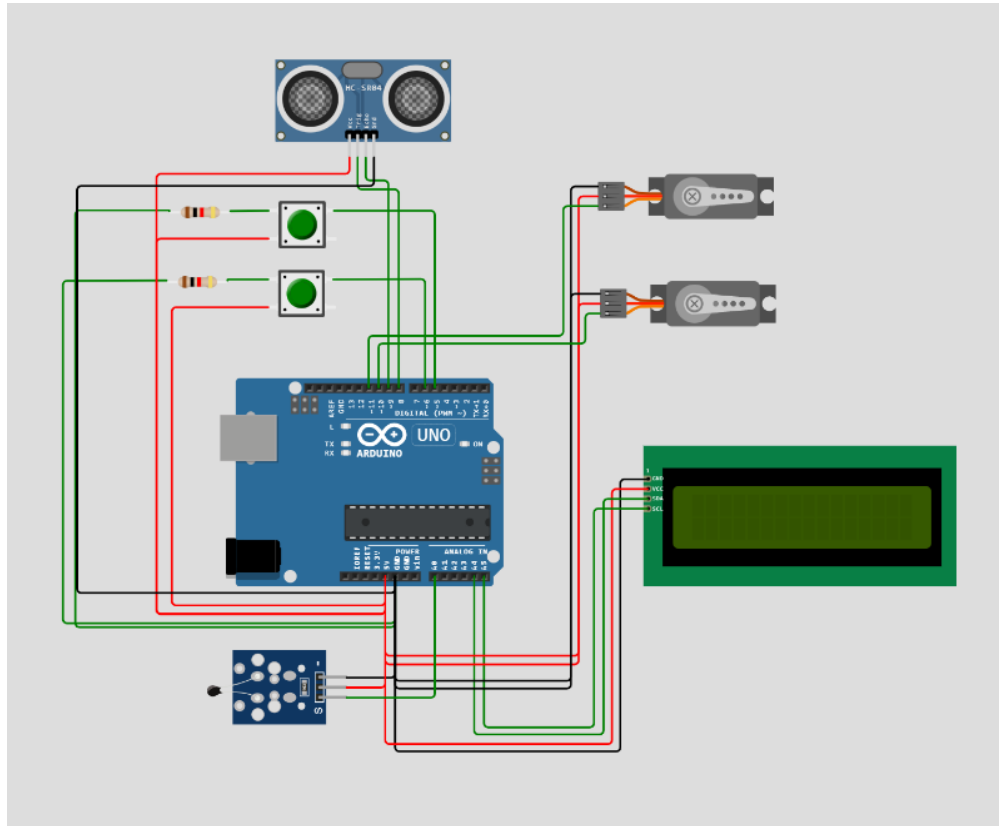


Figure 7: Coffee Barista Simulation

Due to the limitations of the simulation program, only a portion of the sensors and actuators were able to be included in the circuit. The two pushbuttons, LCD screen, and the ultrasonic sensor were implemented almost identically to the physical circuit, with slight modifications to what is displayed on the LCD screen. The weight sensor, HC-06 Bluetooth module, and the DS18B20 temperature probe sensor were not available on the simulation program. The temperature probe was simulated with a similar temperature sensor, however there were nothing similar to use as the weight sensor or the Bluetooth module. The main functions of the Coffee Barista were able to be replicated, with the simulated circuit only missing the weight sensor and the Bluetooth module, since the majority of the circuit operation is dependent on the on the user's specified milk and sugar quantity.

## Pushbuttons

The pushbuttons used in this project were for making personalized selections of milk and sugar quantities. This actuator was the easiest to implement within this circuit as there were no additional libraries required within the Arduino code. The circuit consisted of two pushbuttons and required a 10k $\Omega$  for each pushbutton. The red pushbutton was used to make the desired quantity selections which means the more times the user pressed the button, the more the quantity increases. The green pushbutton was used to select the quantity, so upon pressing it the appropriate servo would turn 90° for the appropriate amount of time, as seen in Figure 8.

```
    lcd.setCursor(1, 0);  
    lcd.print("Milk Quantity:");  
    // milk input  
    while (count <= 5) {  
        if (digitalRead(buttonRed) == HIGH) {  
            count++;  
            delay(250);  
        }  
        else if (digitalRead(buttonGreen) == HIGH) {  
            milkVal = count;  
            break;  
        }  
        lcd.setCursor(0, 1);  
        lcd.print(count);  
    }  
    count = 0;  
    lcd.clear();
```

Figure 8: LCD-I2C Milk Method Code

## Servo Motors

This project consisted of two mini servo motors. Each servo motor was responsible for dispensing milk or sugar into the coffee cup. The user inputted their milk/sugar preference using the pushbuttons, and the servo motors would turn 90° for the appropriate amount of time. The more quantities the user selected, the longer the servo motor would stay in the “open” position, before returning back to its original position. The servo motors have a GND wire, VCC wire, and a Signal pin which was connected to the pulse width modulation pin (PWM) on the Arduino. The servo motor required one additional library, the Servo.h library. Initializing the servo motor was very simple, as shown in Figure 9.



```

#include <Servo.h>

// servo pin
const int servoPin1 = 10;
const int servoPin2 = 11;
Servo servo_1;
Servo servo_2;

```

Figure 9: Initializing Servo Motor

The servo motors go through an if-else-if statements to determine how long to hold the “open” position (hold the 90° position) according to the user’s preferred input, also seen in Figure 10 below.

```

if (milkVal == 0) {
    servo_1.write (0);
}
else if (milkVal == 1) {
    servo_1.write (90);
    delay(250);
    servo_1.write (0);
}
else if (milkVal == 2) {
    servo_1.write (90);
    delay(500);
    servo_1.write (0);
}

```

Figure 10: if-else-if statement

## Ultrasonic Sensor

The HC-SR04 ultrasonic sensor was used to detect if a coffee mug is placed under the dispensers. The ultrasonic sensor has a GND, VCC, TRIG, and ECHO pin, it also does not require any additional libraries to operate. As shown in Figure 11, the ultrasonic sensor measures distance by sending sound waves in the direction of a cup, while a timer is started. The cup then reflects the sound waves back to the ultrasonic sensor. The receiver picks up the reflected sound waves and stops the timer. The distance travelled is determined by dividing the time it takes for the wave to return by the speed of sound.

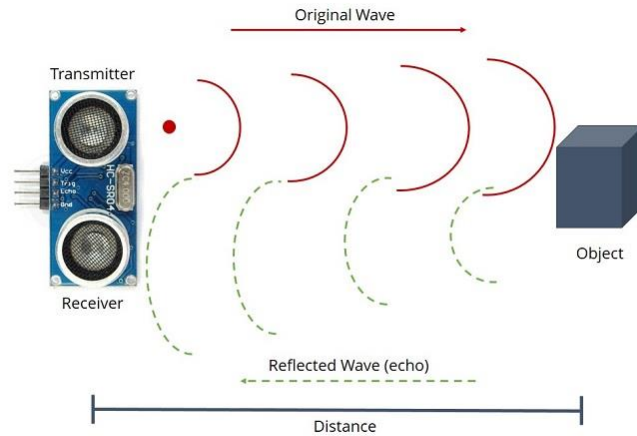


Figure 11: Ultrasonic Sensor Working Principle. Adapted from [5].

The ultrasonic sensor is initialized using the pins from the Arduino board, seen in Figure 12.

```
//ultrasonic sensor
const int TRIG_PIN = 8;
const int ECHO_PIN = 9;

const int DISTANCE_THRESHOLD = 5;
```

Figure 12: Initializing the Ultrasonic sensor

The code within the void setup was declaring the TRIG pin as OUTPUT and the ECHO pin as INPUT, as seen below in Figure 13.

```
//ultrasonic sensor
pinMode(TRIG_PIN, OUTPUT);
pinMode(ECHO_PIN, INPUT);
```

Figure 13: Ultrasonic Sensor Void Setup Code

The main body code is demonstrated below in Figure 14a, this sends/receives the sound waves and calculates the distance between the cup.

```
// generate 10-microsecond pulse to TRIG pin
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);

// measure duration of pulse from ECHO pin
duration_us = pulseIn(ECHO_PIN, HIGH);
// calculate the distance
distance_cm = 0.017 * duration_us;
```

Figure 14: Ultrasonic Sensor Main Body Code

In Figure 15, an if-else-if statement was included to check if a cup is placed under the dispensers. If the cup is not detected, then the user will not be allowed to input any quantity of milk or sugar.

```
//if cup is placed within 5cm of ultrasonic, print okay, let servos turn
if (distance_cm < DISTANCE_THRESHOLD){
  lcd.setCursor(0, 1);
  lcd.print("Okay");
  lcd.clear();
  milkServo();
  sugarServo();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Thank you");
  delay(500);
}
//if no cup detected, print place cup and dont let servos turn
else if (distance_cm > DISTANCE_THRESHOLD){
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Place cup");
  servo_1.write (0);
  servo_2.write (0);
}
```

Figure 15: Ultrasonic Sensor Main Body Code

## LCD Screen and I2C Serial Adaptor

The LCD screen is used to prompt and display the quantity of milk and sugar the user selected. Originally, the LCD screen required 8-16 pins to display information. Fortunately, the I2C Serial Adaptor reduced the number of pins required to 4, including the GND, VCC, SCL and SDA pins. The SCL and SDA pins connect to pins A4 and A5, respectively. The LCD Screen and I2C Serial Adaptor (LCD-I2C) required one additional library, the LiquidCrystal\_I2C.h library. In Figure 16, the LCD-I2C is initialized.

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 20, 4);
```

Figure 16: LCD-I2C Initialization Code and Variables

While using the LCD-I2C, the code requires “lcd.xxx” to instruct the screen what to do. This can be seen in Figure 8, where the LCD-I2C prints the “Milk Quantity” prompt in the first row and the user’s desired quantity is printed to the second row.

## Temperature Probe Sensor

The DS18B20 temperature probe sensor was chosen to monitor the temperature of the milk since it is necessary to ensure it stays at a safe temperature. It was easy to implement and liquid proof, making it suitable for this application. This probe required two different libraries, the Wire.h library and the DallasTemperature.h library. The circuit for the probe was quite simple to build and only required one 4.6kΩ resistor. The GND wire of the temperature probe was connected to the GND of the Arduino, the VCC wire was connected to 5V from the Arduino, and the DQ wire was connected to pin A0 of the Arduino. The initialization of the probe was very straight forward, as seen below in **Error! Reference source not found.17**.

Figure 17: Temperature Sensor Initialization Code and Variables

The code within the void setup was one simple line of code, as seen below in Figure 18.

```
tempSensor.begin();    // initialize the sensor
```

Figure 18: Temperature Sensor Void Setup Code

The code within the main body of the code was, once again, very straightforward as seen below in Figure 19.

```
tempSensor.requestTemperatures();    // send the command to get temperatures
tempCelsius = (float)tempSensor.getTempCByIndex(0);    // read temperature in Celsius

while (tempCelsius > 4){
    Terminal.print("Milk is too warm");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Milk is too warm");
    delay(1000);
}
```

Figure 19: Temperature Sensor Main Body Code

The code above sends the command to retrieve the temperature in Celsius. That value was then compared to the safe storage temperature of cows' milk as stated by the USDA, 4°C. If the temperature that was read was greater than 4°C, a notification was sent to the phone via the Bluetooth module and the LCD screen, and the servo motors were not turned, ensuring that no users were at risk.

### Weight Sensor

The weight sensor chosen for the project was the HX711 weight sensor. This was the most challenging sensor to integrate into the circuit design, as it required a lot of code and preliminary work. It used the HX711\_ADC.h library and required multiple integers throughout. The circuit construction itself was also particularly challenging, as it was necessary to connect the four weight pads and the amplifier in a specific manner, seen schematically in Figure20.

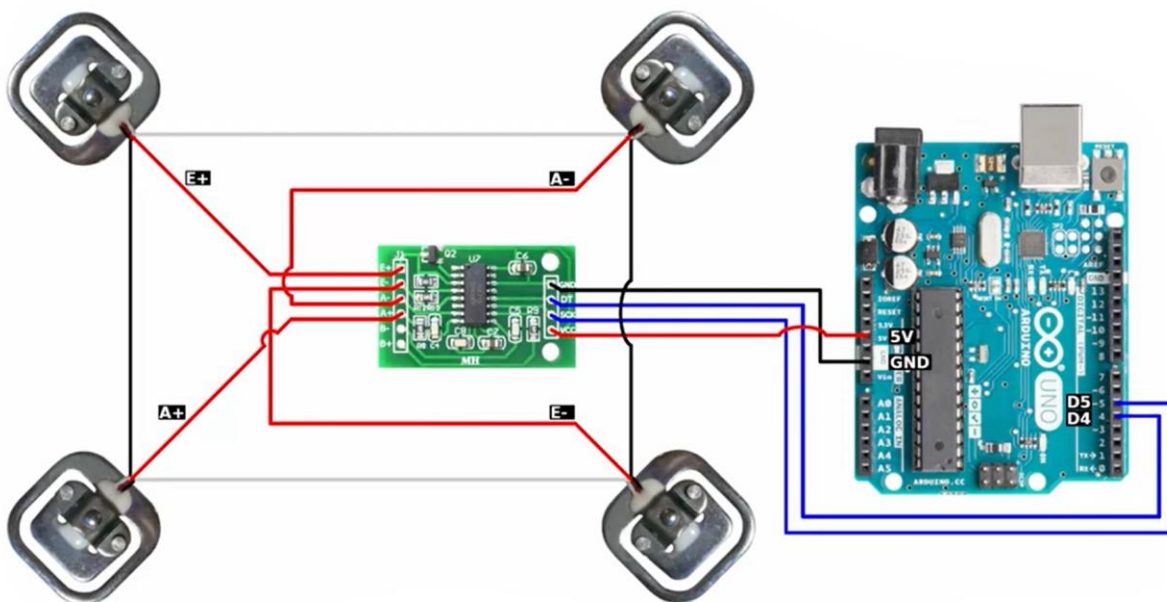


Figure 20: Weight Pads and Amplifier Schematic. Adapted from [5]

Each weight pad had 3 wires which had to be connected, and some of those connections had to be connected to the amplifier. This made creating this circuit very difficult, as it was challenging to ensure a good connection was made between wires. To add to the difficulty, the wires did not fit into the amplifier, so it was necessary to include additional circuitry to make this connection

happen. Once the circuit was built, it had to be calibrated using the Calibration code from the HX711\_ADC.h library. This code was very complex and hard to understand, however following the instructions was fairly straightforward. Running the code, placing a known weight on the scale and entering it into the serial monitor resulted in a specific calibration factor, as seen in Figure 211721.

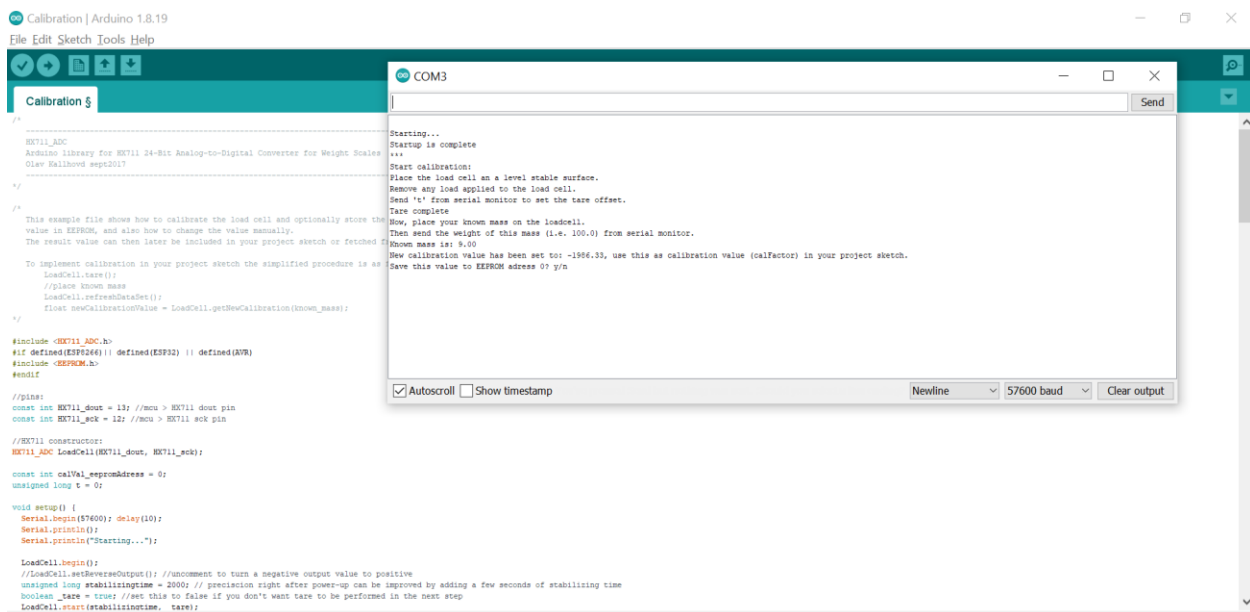


Figure 2117: Portion of HX711 Calibration code returning the Calibration Factor

This calibration factor was then declared in the actual project code, in order to obtain the most accurate reading possible. The code used for initializing the weight sensor is seen below in Figure 22.

```

//weight sensor
const int HX711_dout = 13; //mcu > HX711 dout pin
const int HX711_sck = 12; //mcu > HX711 sck pin

HX711_ADC LoadCell(HX711_dout, HX711_sck);

const int calVal_eepromAdress = 0;
unsigned long t = 0;
  
```

Figure 22: Weight Sensor Initialization code

The above code was used to assign the pins used on the Arduino for the weight sensor and create the load cell.

```

LoadCell.begin();
float calibrationValue; // calibration value (see example file "Calibration.ino")
calibrationValue = -1986.33; // uncomment this if you want to set the calibration value in the sketch

unsigned long stabilizingtime = 2000; // preciscion right after power-up can be improved by adding a few seconds of stabilizing time
boolean _tare = true; //set this to false if you don't want tare to be performed in the next step
LoadCell.start(stabilizingtime, _tare);
if (LoadCell.getTareTimeoutFlag()) {
    Serial.println("Timeout, check MCU>HX711 wiring and pin designations");
    while (1);
}
else {
    LoadCell.setCalFactor(calibrationValue); // set calibration value (float)
    Serial.println("Startup is complete");
}

```

Figure 23: Weight sensor void setup code

This code, Figure 23, was used in the void setup to initialize the weight sensor. It declares the calibration factor calculated from the example code and tares the scale. This means that each time the code was reuploaded to the Arduino board, the scale was set to zero, even if there was weight on it. This had to be taken into account when setting the threshold weight in the body of the code. Below, in Figure 1824, is the body of the code for the weight sensor.

```

//weight sensor
Dabble.processInput();
static boolean newDataReady = 0;
const int serialPrintInterval = 0; //increase value to slow down serial print activity

// check for new data/start next conversion:
if (LoadCell.update()) newDataReady = true;

// get smoothed value from the dataset:
if (newDataReady) {
    if (millis() > t + serialPrintInterval) {
        float i = LoadCell.getData();
        Serial.print("Load_cell output val: ");
        Serial.println(i);

        while (i<-1){
            Terminal.print("Milk is low");
            Terminal.print(i);
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Milk is low");
            delay(1000);
            float i = LoadCell.getData();
        }
        newDataReady = 0;
        t = millis();
    }
}

// receive command from serial terminal, send 't' to initiate tare operation:
if (Serial.available() > 0) {
    char inByte = Serial.read();
    if (inByte == 't') LoadCell.tareNoDelay();
}

// check if last tare operation is complete:
if (LoadCell.getTareStatus() == true) {
    Serial.println("Tare complete");
}

```

Figure 184: Weight sensor main body code

It can be seen that the threshold weight is set to -1. As mentioned earlier, the scale executes the tare operation every time the code is uploaded, meaning that even though weight from the milk storage container is already on the scale, it is set to zero. The negative threshold accounts for the fact that as more milk is used from the storage container, the scale will read a more negative value. The values that are printed to the serial monitor helped during the debugging/demonstration process to understand what was happening in the code and when. These are not necessary in code that would be provided with the product when supplied to the customer.



## Bluetooth Module

The HC-06 Bluetooth module was used to send alerts to the user's phone, based on the results of certain sensors. If the temperature of the milk reached too high or if the milk ran low in the storage container the alerts were sent. The HC-06 had multiple compatible libraries to choose from, however the Dabble library was chosen for the project. Since the Dabble library was used, the RX and TX pins of the Bluetooth module had to be connected to pins 3 and 2 of the Arduino respectively. This was because the functions within the Dabble library set these values, meaning they cannot be easily changed by the programmer. Using Dabble greatly simplified the code, as implementing the library was very easy. It also had an app to download on the phone, which helped alleviate issues which occurred when trying to connect the phone to the Bluetooth. The largest challenge arose simply from connection errors between the phone app and the Bluetooth module. Troubleshooting these connections issues were simplified with the Dabble app, as there was an icon to indicate when a successful connection was made. The setup code was quite simple, as seen below in Figure 1925.

```
Dabble.begin(9600);
```

Figure 195: Dabble Setup Code

The above line is the only line of code necessary to include outside of the main body of code, shown in Figure 2026.

```
//bluetooth and temp
Dabble.processInput();
tempSensor.requestTemperatures(); // send the command to get temperatures
tempCelsius = (float)tempSensor.getTempCByIndex(0); // read temperature in Celsius

while (tempCelsius > 4){
    Terminal.print("Milk is too warm");
    Terminal.print(tempCelsius);
    // ...
}
```

Figure 206: Dabble Main Body Code

The first function `Dabble.processInput();` must be called before including any additional commands for the Bluetooth module, to ensure a line of communication is made between the phone and the Bluetooth module. The following code, `Terminal.print();` is what controlled what was sent

to the phone. This same structure of code was used for both the temperature sensor and the weight sensor.

## Schematic and PCB

Based on the block diagram, Figure 4, and algorithm, Figure 5, a schematic was designed to create an Arduino shield, as seen in Figure 727 below. Using the schematic, the PCB was created to house the components used in the circuit. The PCB allowed for clean connections between each component and Arduino, as shown in Figure 28.

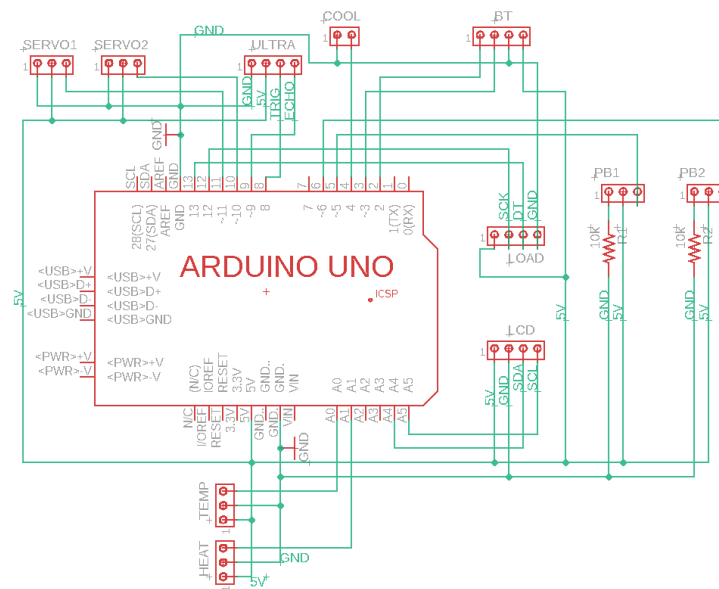


Figure 27: Arduino Shield Schematic

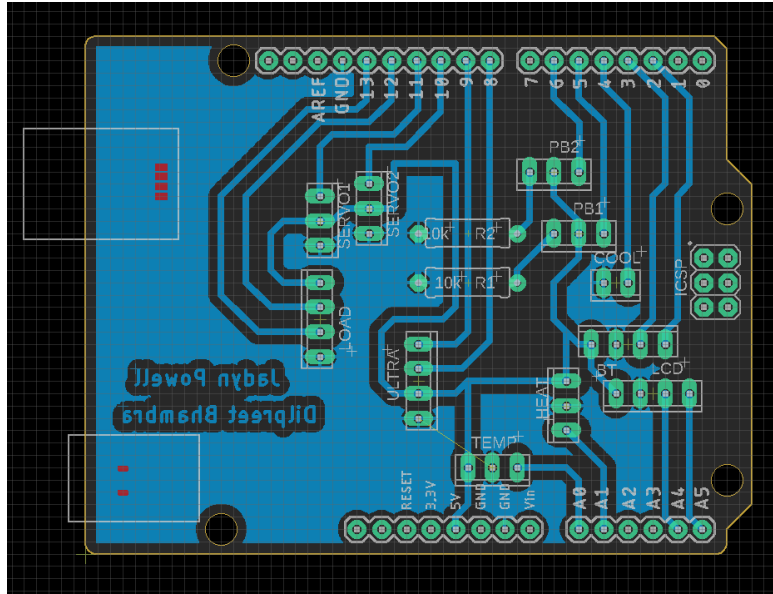


Figure 218: Arduino Shield PCB

Pin headers were used to allow for easy connect and disconnect of the circuit components. All ground pins of the Arduino are connected in order to maximize current flow. During the design process, it was ensured that no wires crossed, as this PCB was a single layer PCB. Multiple iterations of the design were necessary to eliminate crossed wires from the design. Placement of components was also taken into account during the design process as certain sensors/actuators, such as the Bluetooth module, were soldered directly onto the PCB board. Extra room had to be allocated to ensure a good fit on the board. The placement of the pins for the ultrasonic sensor was also intentional, as the proximity sensor had to be located under the LCD screen and pushbuttons. The pins allocated to the various sensors and actuators were specifically chosen to optimize wire placement on the PCB and minimize unwanted traces and unnecessary trace lengths.

## Implementation

### Prototype

A full prototype for the circuit, as seen in Figure 2229 and Figure30, was constructed following the original design in Figure 6.

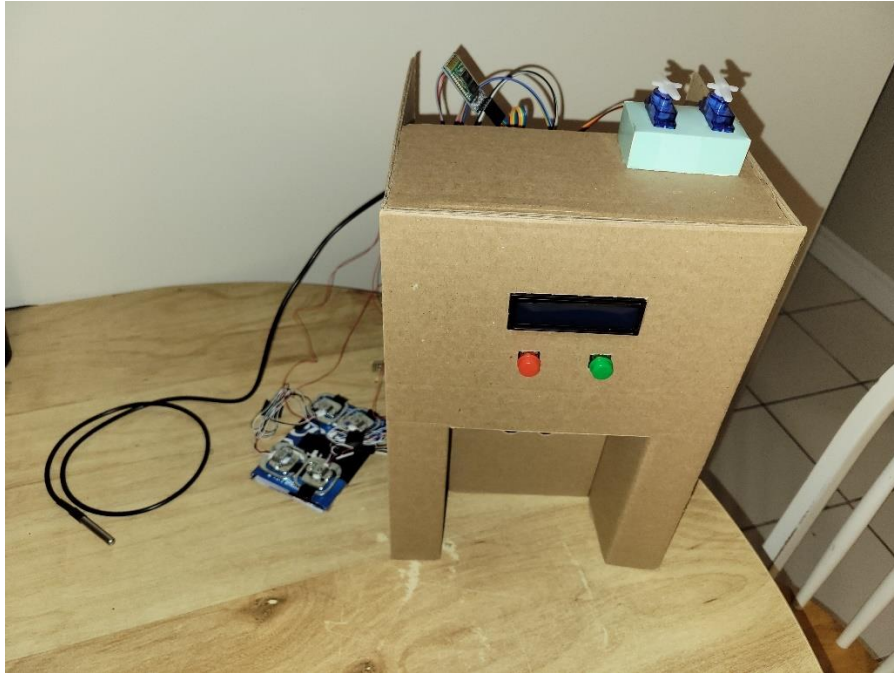


Figure 229: Prototype Build

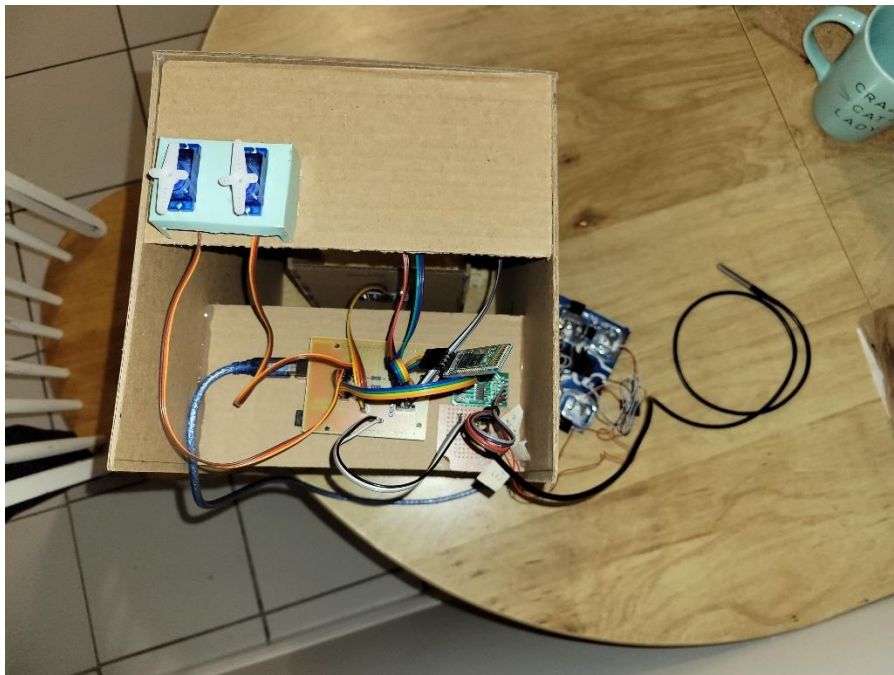


Figure 30: Prototype Circuitry

This incorporated all components of the circuit, leaving the only thing unrealized was the storage method of the milk and sugar. This allowed for a more accurate representation of the actual product, making testing much easier. It also allowed for the opportunity to interact with the product

in a very similar way which the actual users would, giving a much better understanding of how the design met the customer constraints described earlier.

### **Pushbuttons and LCD Screen**

The two pushbuttons and the LCD were the first sensors and actuators to be implemented. The majority of the circuit operation is dependent on the on the user's specified milk and sugar quantity, so it was necessary to have these sensors and actuators functioning first. The two pushbuttons were chosen since they are a very simple, intuitive way for the user to input their selected quantity. One button is used to increase the amount selected, and the other button is used to confirm the amount and dispense it. The LCD screen was chosen as was a very straightforward, direct way of communication information and instructions to the user. The LCD screen prompts the user to input their selected milk quantity, then their selected sugar quantity. It also serves as a convenient way for the user to see their selected quantity, so they are able to see their selected quantities in real time and make adjustments before dispensing the milk and sugar.

### **Servo Motors**

The operation of the servo motors was directly dependent on that of the two pushbuttons, making implementing them afterwards a very logical choice. The servo motors were chosen as they are responsible for controlling the dispensing mechanism and it was very easy to open and close a storage container lid by simply adjusting the turning angles, and durations of the servo motors within the code. One servo motor was used to control the milk storage container, and the other servo is used to control the sugar storage container. Each servo operated separately, one at a time, so milk was dispensed first, followed by sugar. They were dispensed by setting the servo motor to the "open" position (turned 180° from its original position), then returning it to the closed position (the original position). The duration that each servo motor stays open is dependent on the selected quantities of milk and sugar, so the higher quantity a user selects, the longer the servo motor will remain in the "open" position. Arbitrary, pre-set times were assigned for each quantity which dictated how long the servo remained "open" before closing again.

## Ultrasonic Sensor

The ultrasonic sensor was used to override control of the servo motors, pushbuttons, and LCD screen, making it necessary for the servo motors to be operational before the ultrasonic sensor was implemented. The ultrasonic sensor was used to ensure that the milk and sugar did not dispense if there was no coffee cup present. The use of the ultrasonic sensor was a very intuitive one, as setting a threshold distance and detecting whether something was present or not is a very common application of an ultrasonic sensor. If there was nothing detected within the specified threshold distance which was specified to be 5cm in the code, the servo motors were overwritten and were not permitted to turn to the “open” position. This ensured that no mess was made, and that nothing would dispense. Also, if there was no cup detected within 5cm of the sensor, no input was read by the pushbuttons and the LCD screen read “Place cup”. This was a clear, straightforward way to communicate instructions to the user and help ensure easy operation of the Coffee Barista.

Below, in Figure31, is the algorithm for the pushbuttons, LCD screen, servo motors and ultrasonic sensor.

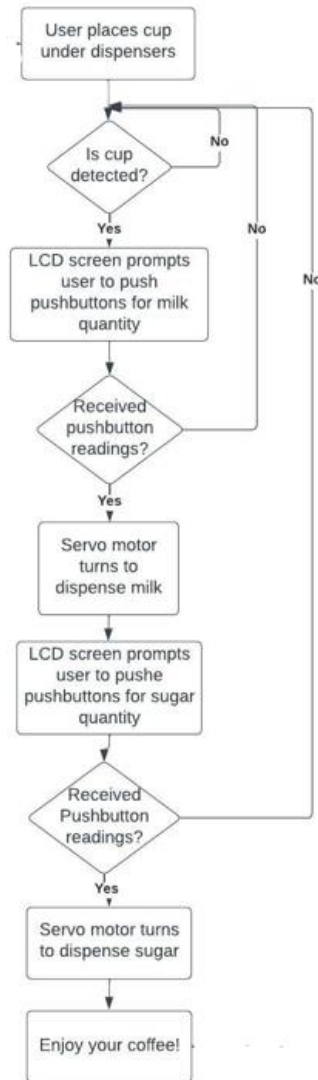


Figure 31: Algorithm for pushbuttons, servos, lcd, ultrasonic

### Temperature Probe Sensor and Bluetooth Module

Since the Coffee Barista was used to store cows' milk, it was necessary to ensure those storage standards met those declared by the USDA. Monitoring the temperature of the milk to ensure that it remained within the acceptable storage temperature was necessary. The DS18B20 temperature probe sensor was chosen to monitor the temperature of the milk as it was easy to work with and liquid proof, making it suitable for this application. It was also necessary to notify the user when the temperature of the milk rose above the safe storage level and using the Bluetooth module was a clear solution to this issue. Using the Bluetooth module ensured that the user received the alert

on their phone, making it highly likely that they will see it and be able to alleviate that issue. The Bluetooth module was compatible with multiple libraries, however the Dabble library was chosen for use. This was because the Dabble library was very easy to implement and use, had a phone app that was easily compatible, and coding was very straightforward. Below, in Figure 32, is the algorithm for the temperature sensor and the Bluetooth module.

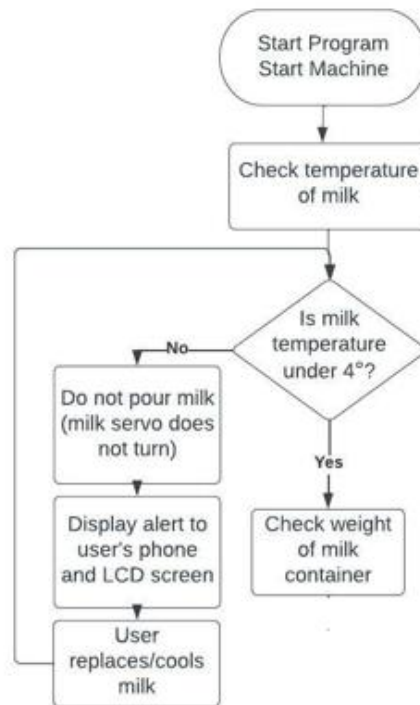


Figure 32: Algorithm of temperature sensor and Bluetooth module

## Weight Sensor

The final sensor integrated into the circuit was the HX711 weight sensor. This was because the weight sensor also required use of the Bluetooth module to send a notification to the user. The weight sensor was used to monitor the amount of milk in the storage container. It was chosen as it was easy to monitor the weight of the storage container, and if that weight dropped below a specified threshold, an alert was sent via the Bluetooth module. Again, using the Bluetooth module was a clear solution to this issue, as the user received the alert on their phone, making it very likely that they will see it and be able to alleviate that issue. The same Dabble library was used, for the



same above stated reasons. Below, in Figure 2333, is the algorithm for the weight sensor and Bluetooth module.

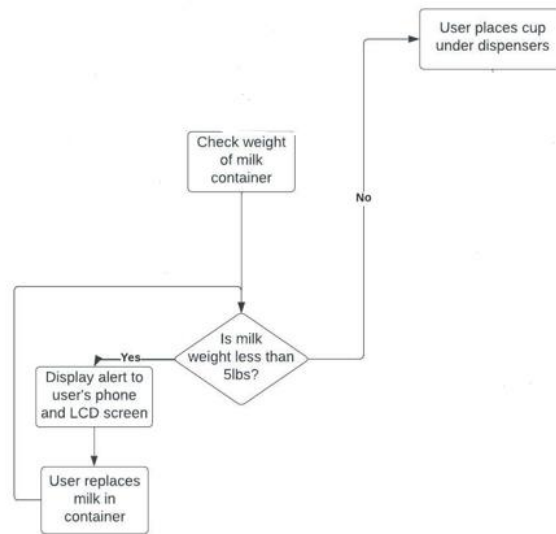


Figure 233: Algorithm for weight sensor and Bluetooth module

## Technical Testing and Debugging Procedure

The following documents the debugging process of the ten main issues faced while implementing the Coffee Barista. Each issue was documented upon discovery, followed by line-item notes of the suspected cause, debug test, and result of the test. This documentation was repeated until the issue was successfully resolved.

### Issue 1: Bluetooth Module not Connecting to Phone

**Suspected Cause:** Bluetooth module not connected to circuit correctly

#### Debug Test:

- Confirmed circuit diagram matched what we had built

**Results:** Issue persisted, no connection was made

**Suspected Cause:** Issue with code

### Debug Test:

- Rewrite portions of code in hopes of getting the phone to connect

```
const int led = 2;

void setup() {
  Serial.begin(9600);
  pinMode(led, OUTPUT);
}

void loop() {
  char c = Serial.read();
  if(c == 'a'){
    digitalWrite(led, HIGH);
  }
  else if(c == 'b'){
    digitalWrite(led, LOW);
  }
  else{

  }
  delay(25);
}
```

Figure 244: Altered Bluetooth Debugging Code

**Results:** Issue persisted, no connection was made

**Suspected Cause:** TX and RX pins connected while uploading code

### Debug Test:

- Arduino Uno board was reset and the TX, RX pins were disconnected from the TX and RX pins of the Arduino Uno
- Attempted to connect phone with no code uploaded and TX/RX pins unplugged

**Results:** Issue resolved, connection was made

### Issue 2: Notification not Sending to Phone

**Suspected Cause:** Improper Bluetooth connection

### Debug Test:

- Confirmed phone was connected to Bluetooth module

**Results:** Issue persisted, no notification was sent

**Suspected Cause:** Error in code

### Debug Test:

- Code was modified to include different commands

```
#include <SoftwareSerial.h>
SoftwareSerial MyBlue(2, 3); // RX | TX
int flag = 0;
int pinTrig = 9;
int pinEcho = 10;
long duration;
int distance;

void setup()
{
  Serial.begin(9600);
  MyBlue.begin(9600);
  pinMode(pinTrig, OUTPUT);
  pinMode(pinEcho, INPUT);
  Serial.println("Ready to connect\nDefault password is 1234 or 000");
}

void loop()
{
  digitalWrite(pinTrig, LOW);
  delayMicroseconds(2);

  digitalWrite(pinTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(pinTrig, LOW);

  duration = pulseIn(pinEcho, HIGH);
  distance = (duration/2) / 29.1;
  if(distance >= 20) {
    MyBlue.print("Milk/Sugar is low");
  }

  delay(500);
}
```

Figure 255: Altered Bluetooth Notification Debugging

**Results:** Issue persisted, no test notification was sent

**Suspected Cause:** Error in code

## Debug Test:

- Code was modified, syntax was altered, and location of lines were changes

---

```
#include <SoftwareSerial.h>
#include <Wire.h>

SoftwareSerial hc06(3,2);
const int TRIG_PIN = 6; // Arduino pin connected to Ultrasonic Sensor's TRIG pin
const int ECHO_PIN = 7;

const int DISTANCE_THRESHOLD = 5;

float duration_us, distance_cm;

void setup(){
  //Initialize Serial Monitor
  Serial.begin(9600);
  // Serial.println("ENTER AT Commands:");
  //Initialize Bluetooth Serial Port
  hc06.begin(9600);
  pinMode(TRIG_PIN, OUTPUT); // set arduino pin to output mode
  pinMode(ECHO_PIN, INPUT);
}

void loop(){
  // generate 10-microsecond pulse to TRIG pin
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(ECHO_PIN, HIGH);
  // calculate the distance
  distance_cm = 0.017 * duration_us;

  if (distance_cm < DISTANCE_THRESHOLD){
    Serial.write("milk is low");
    hc06.print("low");
    delay(800);}
  else{
  }

  delay (25);
}
```

---

Figure 266: Second Altered Bluetooth Notification Code

**Results:** Issue persisted, no test notification was sent

**Suspected Cause:** Error in Bluetooth module placement

## Debug Test:

- Different pins were used for the Bluetooth module
- Code was modified, syntax was altered, and location of lines were changes

```

#include <SoftwareSerial.h>

SoftwareSerial BTSerial(10, 11);
const int TRIG_PIN = 6; // Arduino pin connected to Ultrasonic Sensor's TRIG pin
const int ECHO_PIN = 7;

const int DISTANCE_THRESHOLD = 5;

float duration_us, distance_cm;

void setup() {
  Serial.begin(9600);
  pinMode(TRIG_PIN, OUTPUT); // set arduino pin to output mode
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  char c = Serial.read();
  // generate 10-microsecond pulse to TRIG pin
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(ECHO_PIN, HIGH);
  // calculate the distance
  distance_cm = 0.017 * duration_us;

  if(c == '?' && distance_cm < DISTANCE_THRESHOLD)
  {Serial.write("milk is low");
  delay(800);}
  else if (c == '?' && distance_cm > DISTANCE_THRESHOLD) {
    Serial.write("milk is okay");
    delay(800);}
  else{

  }

  delay(25);
}

```

Figure 277: Altered Bluetooth Pin Code

**Results:** Issue persisted, no test notification was sent

**Suspected Cause:** Error in code

**Debug Test:**

- Code was abandoned and example code was used to see if library was working

```

#include <SoftwareSerial.h>

SoftwareSerial hc06(2,3);

void setup(){
  //Initialize Serial Monitor
  Serial.begin(9600);
  Serial.println("ENTER AT Commands:");
  //Initialize Bluetooth Serial Port
  hc06.begin(9600);
}

void loop(){
  //Write data from HC06 to Serial Monitor
  if (hc06.available()){
    Serial.write(hc06.read());
  }
  //Write from Serial Monitor to HC06
  if (Serial.available()){
    hc06.write("Hello");
    delay(1000);
  }
}

```

Figure 288: Example Bluetooth Notification Code

**Results:** Issue persisted, no test notification was sent

**Suspected Cause:** Error with chosen code library and phone app

#### Debug Test:

- Code library and app was switched to utilize Dabble
- Code was altered to conform to Dabble requirements

```

#include <Dabble.h>
#include <SoftwareSerial.h>
#include <Wire.h>

SoftwareSerial hc (3,2);
const int TRIG_PIN = 6; // Arduino pin connected to Ultrasonic Sensor's TRIG pin
const int ECHO_PIN = 7;

const int DISTANCE_THRESHOLD = 5;

float duration_us, distance_cm;

void setup() {
  // put your setup code here, to run once:
  Dabble.begin(9600);
  Serial.begin(9600);
  pinMode(TRIG_PIN, OUTPUT); // set arduino pin to output mode
  pinMode(ECHO_PIN, INPUT);
}

void loop() {

  Dabble.processInput();

  // generate 10-microsecond pulse to TRIG pin
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(ECHO_PIN, HIGH);
  // calculate the distance
  distance_cm = 0.017 * duration_us;

  if (distance_cm < DISTANCE_THRESHOLD){
    Terminal.print("Milk is low");

    delay(800);}
  else{

  }
}

```

Figure 299: Altered Library Bluetooth Notification Code

**Results:** Issue resolved, notification was sent with the ultrasonic sensor

### Issue 3: Temperature Sensor Probe Not Sending Alert via Bluetooth Module

**Suspected Cause:** Improper connection with phone and Bluetooth module

#### Debug Test:

- Bluetooth and phone pairing was disconnected, and Bluetooth module was unplugged.
- Bluetooth was reinserted and reconnected to phone

**Results:** Issue persisted

**Suspected Cause:** Improper code syntax

**Debug Test:**

- Code was rewritten using different syntax

---

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Dabble.h>
#include <SoftwareSerial.h>
#include <Wire.h>

SoftwareSerial hc (3,2);

const int SENSOR_PIN = A0; // Arduino pin connected to DS18B20 sensor's DQ pin

OneWire oneWire(SENSOR_PIN); // setup a oneWire instance
DallasTemperature tempSensor(&oneWire); // pass oneWire to DallasTemperature library

float tempCelsius; // temperature in Celsius
float tempFahrenheit; // temperature in Fahrenheit

void setup()
{
  Dabble.begin(9600);
  Serial.begin(9600); // initialize serial
  tempSensor.begin(); // initialize the sensor
}

void loop()
{
  tempSensor.requestTemperatures(); // send the command to get temperatures
  tempCelsius = tempSensor.getTempCByIndex(0); // read temperature in Celsius
  tempFahrenheit = (float)tempCelsius * 9 / 5 + 32; // convert Celsius to Fahrenheit
  Dabble.processInput();

  // Terminal.print("Temperature: ");
  Terminal.print(tempCelsius);
  Serial.print("Temperature: ");
  Serial.print(tempCelsius); // print the temperature in Celsius
  Serial.print("°C");
  Serial.print(" ~ "); // separator between Celsius and Fahrenheit
  Serial.print(tempFahrenheit); // print the temperature in Fahrenheit
  Serial.println("°F");

  delay(500);
}
```

Figure 40: Altered Temperature Sensor Notification Code

- New code was uploaded to Arduino Uno board and tested

**Results:** Issue persisted, no Bluetooth alert was sent



**Suspected Cause:** Improper variable types used

**Debug Test:**

- Temperature value was casted as a Float

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Dabble.h>
#include <SoftwareSerial.h>
#include <Wire.h>

SoftwareSerial hc (3,2);

const int SENSOR_PIN = A0; // Arduino pin connected to DS18B20 sensor's DQ pin

OneWire oneWire(SENSOR_PIN); // setup a oneWire instance
DallasTemperature tempSensor(&oneWire); // pass oneWire to DallasTemperature library

float tempCelsius; // temperature in Celsius
float tempFahrenheit; // temperature in Fahrenheit

void setup()
{
  Dabble.begin(9600);
  Serial.begin(9600); // initialize serial
  tempSensor.begin(); // initialize the sensor
}

void loop()
{
  Dabble.processInput();
  tempSensor.requestTemperatures(); // send the command to get temperatures
  tempCelsius = (float)tempSensor.getTempCByIndex(0); // read temperature in Celsius
  tempFahrenheit = (float)tempCelsius * 9 / 5 + 32; // convert Celsius to Fahrenheit

  // Terminal.print("Temperature: ");
  Terminal.print(tempCelsius);
  Serial.print("Temperature: ");
  Serial.print(tempCelsius); // print the temperature in Celsius
  Serial.print("°C");
  Serial.print(" ~ "); // separator between Celsius and Fahrenheit
  Serial.print(tempFahrenheit); // print the temperature in Fahrenheit
  Serial.println("°F");

  delay(500);
}
```

Figure 41: Temp Cast as Float Temperature Sensor Debugging Code

- Code was uploaded to Arduino Uno board and tested

**Results:** Issue resolved, Bluetooth alert was sent

#### **Issue 4: Code Persistently Not Uploading to Arduino Uno Board**

**Suspected Cause:** Improper connection to board

**Debug Test:**

- Arduino Uno Board was rest, unplugged from laptop and replugged in

**Results:** Issue persisted, code would not upload and would return an access denied error

**Suspected Cause:** Issues with Arduino app settings

**Debug Test:**

- Comm Port was set to COMM 3 in accordance with the Arduino Uno board
- Reuploaded code

**Results:** Issue persisted, code would not upload and would return an access denied error

**Suspected Cause:** Faulty laptop

**Debug Test:**

- The other group member attempted to connect their laptop to the Arduino Uno board and upload the same code

**Results:** Issue persisted, code would not upload and would return an access denied error

**Suspected Cause:** Faulty board

**Debug Test:**

- Board was switched out for other group members Arduino Uno board provided in the ECE2242 lab kit

**Results:** Issue persisted, code would not upload and would return an access denied error

**Suspected Cause:** Faulty board

**Debug Test:**

- Board was switched out for a group members personal Arduino Uno board
- Appropriate settings were changed within the Arduino app

**Results:** Issue persisted, code would not upload and would return an access denied error

**Suspected Cause:** Faulty board

**Debug Test:**

- Board was switched out for a group members personal Arduino Mega board
- Appropriate settings were changed within the Arduino app

**Results:** Issue persisted, code would not upload and would return an access denied error

**Suspected Cause:** Faulty cable

**Debug Test:**

- Board was switched back to original Arduino Uno board
- USB AB cable was switched for other group members cable that was included in the ECE2242 lab kit

**Results:** Issue resolved; code successfully uploaded to the Arduino Uno board

### **Issue 5: Weight Sensor Not Reading Values**

**Suspected Cause:** Error with code

**Debug Test:**

- Lines of code were rearranged

---

```

#include <HX711.h>
#include "HX711.h"

|
const int LOADCELL_DOUT_PIN = 2;
const int LOADCELL_SCK_PIN = 3;

HX711 scale;//(DOUT, CLK);

void setup(){
  Serial.begin(9600);
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  scale.set_scale(206.76); //This value is obtained by using the SparkFun_HX711_Calibration sketch
  // scale.tare(); //Assuming there is no weight on the scale at start up, reset the scale to 0
}

void loop() {
  float weight = (scale.get_units());
  Serial.print("Reading: ");
  Serial.print(scale.get_units(), 5);
  // Serial.print(weight); //scale.get_units() returns a float
  Serial.print(" lbs"); //You can change this to kg but you'll need to refactor the calibration_factor
  Serial.println();

  int max_weight = 2 ;
  //int red = (weight * ( 255 / max_weight)) ;
  // int green = 255 + (weight * (-255 / max_weight));

  if (weight > max_weight) {
    Serial.println("Milk is low");
  }
}

```

Figure 4230: Altered Weight Sensor Debugging Code

**Results:** Issue persisted, weight sensor would not read values

**Suspected Cause:** Error with chosen library

**Debug Test:**

- After further research a new library was downloaded and used

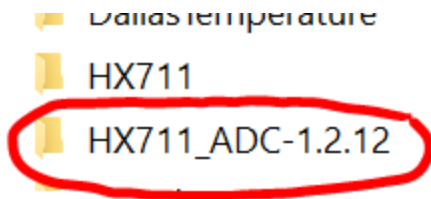


Figure 313: Alternate Weight Sensor Library

**Results:** Issue persisted, weight sensor would not read values

**Suspected Cause:** Error with code

**Debug Test:**

- Example code template from new library was used

```
'
#include "HX711.h"

// HX711 circuit wiring
const int LOADCELL_DOUT_PIN = 2;
const int LOADCELL_SCK_PIN = 3;

HX711 scale;

void setup() {
  Serial.begin(57600);
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
}

void loop() {
  if (scale.is_ready()) {
    long reading = scale.read();
    Serial.print("HX711 reading: ");
    Serial.println(reading);
  } else {
    Serial.println("HX711 not found.");
  }
  delay(1000);
}
```

Figure 324: Example Weight Sensor Code

**Results:** Issue persisted, weight sensor would not read values

**Suspected Cause:** Error with circuit construction

**Debug Test:**

- Circuit was deconstructed and rebuilt, following YouTube Video listed below:

“Arduino Scale with HX711 and 50kg Bathroom Scale Load Cells | Step by Step Guide”  
[5].

**Results:** Issue resolved; weight was reading but only zeros

#### **Issue 6: Weight Sensor Only Reading Value of Zero**

**Suspected Cause:** No calibration factor was declared

#### **Debug Test:**

- Without a calibration factor, the weight sensor will continuously read zero, since the internal calculations done within the library files require a calibration factor
- Calibration factor example code was run, using code below
- New calibration factor was obtained using example code provided in library and applied to the code

```

void calibrate() {
  Serial.println("****");
  Serial.println("Start calibration:");
  Serial.println("Place the load cell on a level stable surface.");
  Serial.println("Remove any load applied to the load cell.");
  Serial.println("Send 't' from serial monitor to set the tare offset.");

  boolean _resume = false;
  while (_resume == false) {
    LoadCell.update();
    if (Serial.available() > 0) {
      if (Serial.available() > 0) {
        char inByte = Serial.read();
        if (inByte == 't') LoadCell.tareNoDelay();
      }
    }
    if (LoadCell.getTareStatus() == true) {
      Serial.println("Tare complete");
      _resume = true;
    }
  }

  Serial.println("Now, place your known mass on the loadcell.");
  Serial.println("Then send the weight of this mass (i.e. 100.0) from serial monitor.");

  float known_mass = 0;
  _resume = false;
  while (_resume == false) {
    LoadCell.update();
    if (Serial.available() > 0) {
      known_mass = Serial.parseFloat();
      if (known_mass != 0) {
        Serial.print("Known mass is: ");
        Serial.println(known_mass);
        _resume = true;
      }
    }
  }

  LoadCell.refreshDataSet(); //refresh the dataset to be sure that the known mass is measured correct
  float newCalibrationValue = LoadCell.getNewCalibration(known_mass); //get the new calibration value

  Serial.print("New calibration value has been set to: ");
  Serial.print(newCalibrationValue);
}

```

---

Figure 335: Portion of Calibration Factor Code

**Results:** Issue resolved, but weight was were very inaccurate.

## Issue 7: Weight Sensor Values Not Accurate

**Suspected Cause:** Wrong Calibration Factor

**Debug Test:**

- Calibration factor example code was rerun, using code below and known weight of 9lbs
- New calibration factor was obtained and applied to the code

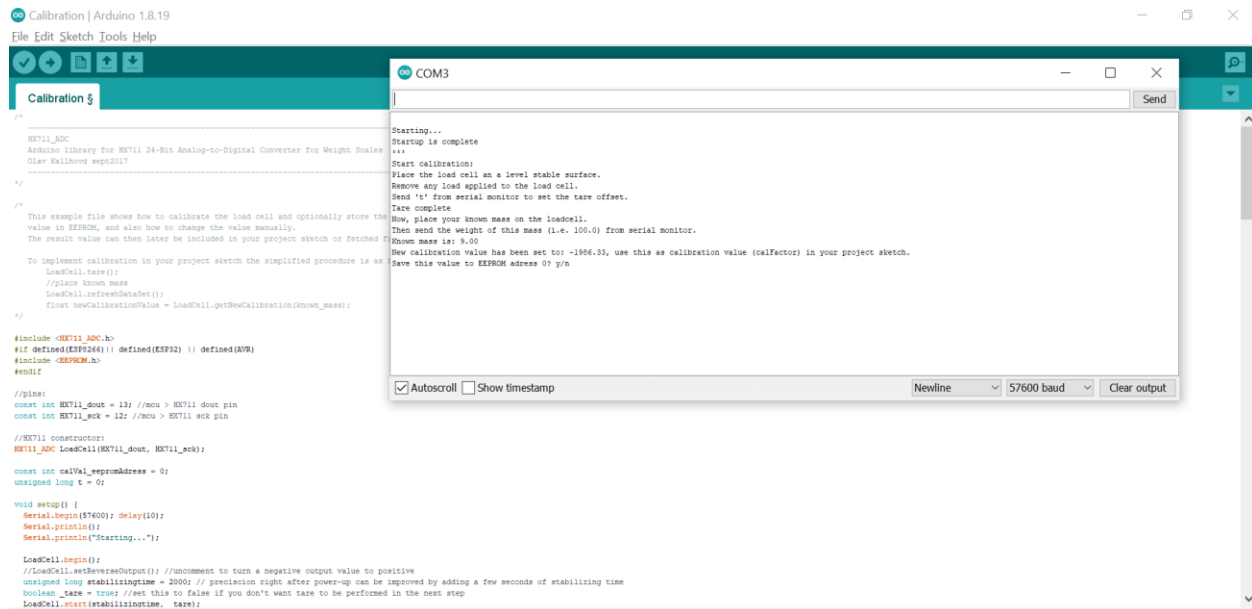


Figure 346: Correct Calibration Factor Code

**Results:** Error persisted, weight measurements were not accurate

**Suspected Cause:** Error with code

**Debug Test:**

- Tare function was added to code to zero scale upon uploading to Arduino Board



```

#include <HX711_ADC.h>

//pins:
const int HX711_dout = 12; //mcu > HX711 dout pin
const int HX711_sck = 13; //mcu > HX711 sck pin

//HX711 constructor:
HX711_ADC LoadCell(HX711_dout, HX711_sck);

const int calVal_eepromAddress = 0;
unsigned long t = 0;

void setup() {
  Serial.begin(57600); delay(10);
  Serial.println();
  Serial.println("Starting...");

  LoadCell.begin();
  float calibrationValue; // calibration value (see example file "Calibration.ino")
  calibrationValue = -8103.82; // uncomment this if you want to set the calibration value in the sketch

  unsigned long stabilizingtime = 2000; // preciscion right after power-up can be improved by adding a few seconds of stabilizing time
  boolean _tare = true; //set this to false if you don't want tare to be performed in the next step
  LoadCell.start(stabilizingtime, _tare);
  if (LoadCell.getTareTimeoutFlag()) {
    Serial.println("Timeout, check MCU>HX711 wiring and pin designations");
    while (1);
  }
  else {
    LoadCell.setCalFactor(calibrationValue); // set calibration value (float)
    Serial.println("Startup is complete");
  }
}

void loop() {
  static boolean newDataReady = 0;
  const int serialPrintInterval = 0; //increase value to slow down serial print activity

  // check for new data/start next conversion:
  if (LoadCell.update()) newDataReady = true;

  // get smoothed value from the dataset:
  if (newDataReady) {
    if (millis() > t + serialPrintInterval) {

      float i = LoadCell.getData();
      Serial.print("Load_cell output val: ");
      Serial.println(i);
      if (i<-2){
        Serial.print("Milk is low");
      }
      newDataReady = 0;
      t = millis();
    }
  }

  // receive command from serial terminal, send 't' to initiate tare operation:
  if (Serial.available() > 0) {
    char inByte = Serial.read();
    if (inByte == 't') LoadCell.tareNoDelay();
  }

  // check if last tare operation is complete:
  if (LoadCell.getTareStatus() == true) {
    Serial.println("Tare complete");
  }
}

```

Figure 357: Tare added to Weight Sensor Code

- This means that the scale will read zero with the weight on it, and will progressively read a more negative number as weight is removed
- This is not an issue, since the actual value should be accurate, just negative

**Results:** Issue resolved, weight was more accurately measured just with a negative value

### **Issue 8: Weight Sensor Values Not Sending Alert via Bluetooth Module**

**Suspected Cause:** Error with code

**Debug Test:**

- Code did not account for the fact that the weight sensor read the weight as negative values as weight was removed
- The code was modified to account for the negative weight reading by changing the threshold value

```
//weight sensor
Dabble.processInput();
static boolean newDataReady = 0;
const int serialPrintInterval = 0; //increase value to slow down serial print activity

// check for new data/start next conversion:
if (LoadCell.update()) newDataReady = true;

// get smoothed value from the dataset:
if (newDataReady) {
  if (millis() > t + serialPrintInterval) {
    float i = LoadCell.getData();
    Serial.print("Load_cell output val: ");
    Serial.println(i);

    if (i<-1){
      Terminal.print("Milk is low");
      delay(1000);
    }
    newDataReady = 0;
    t = millis();
  }
}
```

Figure 368: Altered Weight Threshold Value

**Results:** Issue resolved, alert was sent to the phone via the Bluetooth module when weight was removed

### **Issue 9: Bluetooth Module not Sending Alert**

**Suspected Cause:** Error with pin assignments after soldering PCB

**Debug Test:**

-confirmed connection of pins as per the designed PCB

**Results:** Issue resolved, alerts were sent via the Bluetooth module

### **Issue 10: Weight Sensor Only Reading Value of Zero**

**Suspected Cause:** Error with pin assignments after soldering PCB

#### **Debug Test:**

-confirmed connection of pins as per the designed PCB

**Results:** Issue resolved, weight sensor more accurately read weight

### **Results and Evaluation**

After successfully debugging all of the sensors and actuators, the Coffee Barista was fully functional and operated as expected. This meant that the project was a success. All of the project design constraints were met, as seen in Figure 37 below.

Design Constraint:	Design Complexity	Feasibility	Implementation	Cost
Was the constraint met? (Yes, No)	YES	YES	YES	YES

Figure 379: Design Constraint Evaluation

The Coffee Barista had four sensors and three actuators which were fully functional. The project was also clearly feasible given the time and knowledge restrictions, as it was successfully completed within the five-week timeline. Building the circuit and the prototype allowed for effective debugging, enabling all of the components of the circuit to function as expected. Finally, the project only cost an additional \$18, making for a very low additional cost.

The Coffee Barista successfully monitored the temperature of the milk storage, ensuring that it maintained the standard set by the USDA.

The customer constraints were also all met, as seen below in Figure.

Customer Constraint	Convenience	Usability	Maintainability	Compatibility
Was the constraint met? (Yes, No)	YES	YES	YES	YES

Figure 50: Customer Constraint Evaluation

The Coffee Barista was a convenient solution to the issue, as it automatically dispenses selected quantities of milk and sugar. It was also very user friendly, with only two pushbuttons as inputs and a simple Bluetooth connection. The milk and storage containers are the only things that would need to be maintained, and the prototype design accounted for varying coffee cup sizes, meeting both the maintainability and compatibility constraints.

## **Future Work**

When the circuit detects that the temperature of the milk is above a safe storage level, an alert is sent to the phone via the Bluetooth terminal. An idea that was unable to be realized was including a Peltier to cool the milk. Once the temperature of the milk was detected to be too high, not only would an alert be sent to the phone, but the Peltier would also automatically turn on in order to cool the milk.

Another unrealized idea was to have the coffee itself dispensed from the Coffee Barista. Rather than just dispensing milk and sugar into a cup of already made coffee, the machine itself would also brew and dispense coffee.

Including a weight sensor for the sugar, the same way one is included for the milk, was also unrealized, simply because the second set of weight sensors would not arrive within the project constraint time.

The chosen HX711 weight sensor functioned for its purpose, however the readings were pretty inaccurate, even after using the correct calibration factor. A different, more accurate, weight sensor may be implemented in place of the HX711 in the future.

Finally, including some method of stirring the coffee was also unrealized. The original idea was to use a stir stick attached to a DC motor, however, upon further examination, it was determined that this method was unhygienic, as the stir stick would have to be removed and washed after each use and would be difficult to implement within the time constraints of this project.

## **Conclusion**

The Coffee Barista was an automated milk and sugar dispenser, with quantities that are specified by the user using pushbuttons and included an LCD screen that displayed the specified quantities. It also monitored the temperature and quantity of the milk storage, to ensure that it stored safely

and did not run out. After the end of the five-week timeline, the Coffee Barista was fully functional, and all four sensors and three actuators operated as expected. The project was a success, as all of the project design constraints and customer constraints were met. All group members gained very valuable experience in the engineering design process, circuit design, and prototyping and debugging. Although the project was a success, there are multiple areas for future improvement, including but not limited to adding a cooling element, improving the accuracy of the weight sensor and adding a stirring mechanism.

## References

- [1] A. Atwa, "How to use a Peltier with Arduino," Medium, 16 September 2019. [Online]. Available: <https://ali-atwa.medium.com/how-to-use-a-peltier-with-arduino-a35b0d4e52c2#:~:text=The%20Peltier%2C%20or%20Thermoelectric%20coolers,Mosfet%20and%20a%2010K%20resistor.> . [Accessed 1 March 2022].
- [2] United States Department of Agriculture, "About the U.S. Department of Agriculture," United States Department of Agriculture, [Online]. Available: <https://www.usda.gov/our-agency/about-usda>. [Accessed 29 March 2022].
- [3] United States Department of Agriculture , "How long can you keep dairy products like yogurt, milk, and cheese in the refrigerator?," United States Department of Agriculture , 17 July 2019. [Online]. Available: <https://ask.usda.gov/s/article/How-long-can-you-keep-dairy-products-like-yogurt-milk-and-cheese-in-the-refrigerator#:~:text=Milk%20can%20be%20refrigerated%20seven,or%20go%20to%20FDA's%20web site..> [Accessed 6 March 2022].
- [4] Home and Garden Information Centre, "SAFE HANDLING OF MILK & DAIRY PRODUCTS," Home and Garden Information Centre, 8 March 2007. [Online]. Available: <https://hgic.clemson.edu/factsheet/safe-handling-of-milk-dairy-products/> . [Accessed 6 March 2022].
- [5] Indrek, "Arduino Scale with HX711 and 50kg Bathroom Scale Load Cells | Step by Step Guide.," YouTube, 17 April 2020. [Online]. Available: <https://www.youtube.com/watch?v=Lluf2egMioA> . [Accessed 20 March 2022].
- [6] Newco Enterprises Inc., "Newco Coffee Companion," Newco Enterprises Inc., [Online]. Available: <https://www.newcocoffee.com/product/newco-coffee-companion/>. [Accessed 3 March 2022].

## **Appendix**

### **User Manual**

- Plug in the device and wait for the “Place Cup” prompt
- To connect to Bluetooth
  - o Go onto Dabble App
  - o Click the plug icon in the top right corner of the interface next to the settings
  - o Scroll down in the pop-up window until you see “HC-06”
  - o Press ‘connect’ on the HC-06 line
  - o If properly connected, you should receive a notification
- Place your coffee cup under the dispensers
- After being prompted by the LCD, press the Red button 1-5 times until the desired milk quantity is reached
- Once the desired milk quantity is reached, press the Green button to confirm your selection
- Milk will be dispensed – please do not touch the device while this is happening
- Once milk is dispensed, you will be prompted the LCD to press the button again for desired sugar quantity
- Press the Red button 1-5 times until the desired sugar quantity is reached.
- Press the Green button to confirm the sugar quantity
- Do not touch the device while sugar is being dispensed
- Please remove the coffee cup once the LCD displays “Thank you” and enjoy your beverage.

### **Error Messages**

- Error messages are displayed on the Dabble app and LCD screen.
- To access Dabble error messages, from main screen go to:
  - o Terminal (symbol looks like a text box)
  - o Messages will pop up on the screen when conditions met
    - Note: Dabble terminal also displays real time data relevant to the user
- “Milk is too warm”
  - o Occurs when the milk temperature is greater than the maximum safe allowed temperature of 4 degrees Celsius
  - o Place the milk container back into the fridge or replace warm milk as appropriate
- “Milk is low”
  - o Occurs when the milk in the container is below the threshold weight
  - o Requires the user to refill the container

### **Troubleshooting**

- I can’t select milk/sugar quantity.

Make sure the pushbutton pins are securely connected

- I can't see the weight/ temperature on my phone.

Reconnect your phone to the Bluetooth module used in the circuit

- Nothing is displaying on the LCD screen.

Unplug and re-plug the power for the Coffee Barista