Refill Dispensary

By: Michael Blyakhman (mlb11), Jackson Wiessing (jtw6), Lyla Zegelstein (lrz2)

Team 18

Final Report for ECE 445, Senior Design, Fall 2022

TA: Jason Paximadas

Professor: Arne Fliflet

7 December, 2022

Abstract

We designed a machine where customers can come refill their containers with the liquid of choice. Our machine holds two different types of liquids. The customer places their container on the scale and spins the potentiometer located under the liquid of their choice and then presses the corresponding button to start dispensing. The customer can see the directions on how to operate the machine as well as the weight increase during the dispensing process. There are 3 LEDs on the outside that further communicate the state of the machine to the user as either ready, dispensing, or out of order.

Contents

1. Introduction

Plastic waste is a massive issue world-wide, particularly as it pertains to the packaging of food and other household goods. The United States Environmental Protection Agency estimates that in 2018, 14 million tons of plastic was consumed for packaging in the USA with about seventy percent of that ending up in landfills[1]. Plastic waste is detrimental to the environment as it doesn't decompose naturally on human time scales. End-user plastic waste is often unnecessary as consumers own containers capable of being reused.

We built a vending machine that dispenses precise quantities of goods (between 10 g and 5 kg) into reusable containers. The goal of the machine is to get users to keep bringing the same container back. In addition to plastic waste, our machine tackles the issue of waste in general as users are no longer forced to get quantities larger than desired.

1.1 Subsystem Overview

The machine is broken into 4 parts: User Interface, Dispensing, Control Unit and Power subsystems. The User Interface is where the user places the order which then gets communicated to the Control Unit. The Control Unit will monitor the Dispensing subsystems which executes the order. Finally, the Power subsystem will be supplying all other subsystems the necessary power type to function.

1.2 High-level requirements

1. The user is able to choose an item and quantity to get dispensed via buttons and rotary potentiometers. Before placing the order, the user will have some indication of the amount ordered within a tolerance of 10 grams for weights between .1 kg and 2kg.
2. The machine dispenses two different types of low-viscosity liquids and is able to perform 2 orders in succession.
3. The machine can deliver another order properly after 1 of the liquids is switched out or refilled.

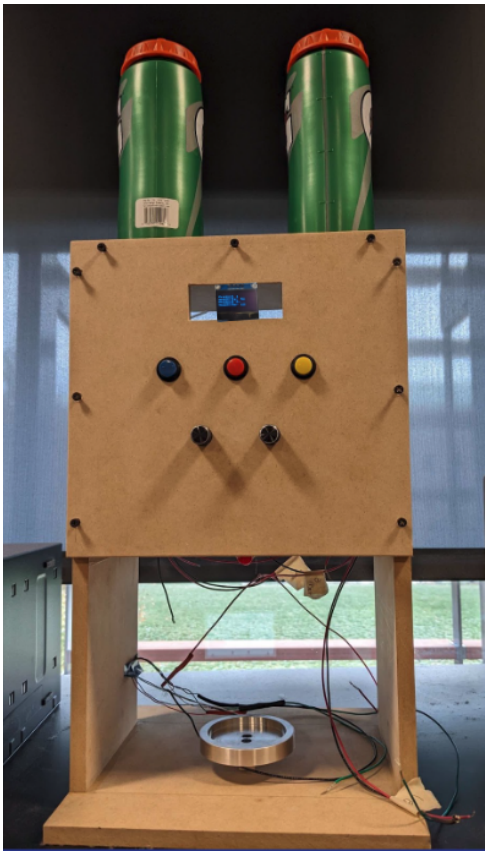


Figure 1: The Refill Dispensary

2 Design

2.1 Block Diagram

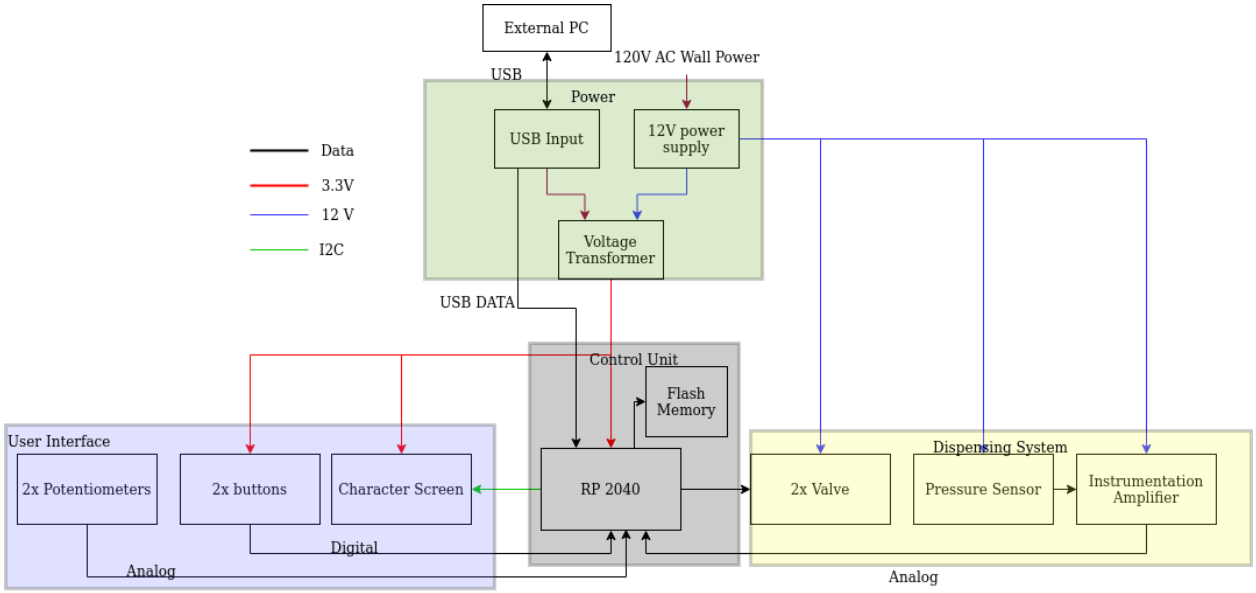


Figure 2: Block diagram for the vending machine

The block diagram has 4 main parts. The user interface interacts with the RP2040 by signaling button presses and potentiometer changes. The RP2040 then will display the number from the potentiometer onto the character screen of the user interface. The dispensing system valves are controlled by the RP2040. Additionally, the pressure sensor in the dispensing system will notify the RP2040 when to start and stop dispensing. The flash memory will be used to store program instructions and data about the containers. Finally, the power subsystem will supply all other subsystems with the proper power type.

2.2 Power

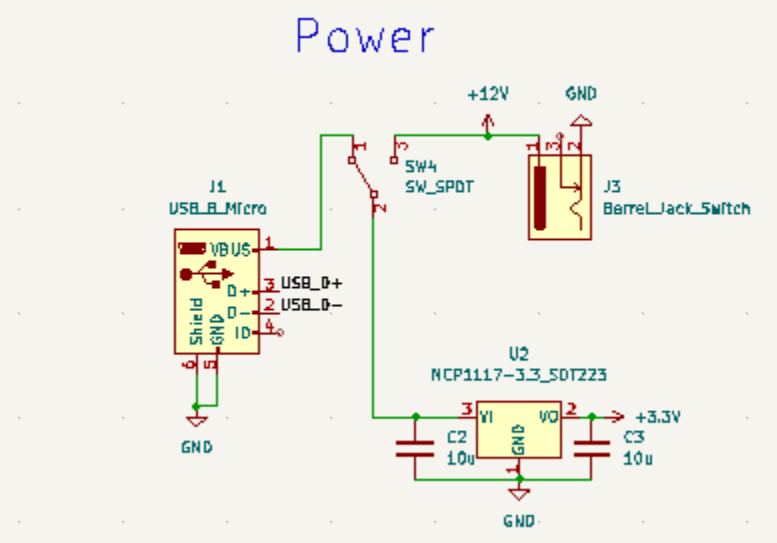


Figure 3: Power Subsystem

The power system consists of several parts: a barrel jack for the 12V line, a 3.3V voltage regulator, two 10uF capacitors, and a switch to be able to switch from the 5V USB power line to the 12V line from the barrel jack. The USB power input is necessary to initially flash the device, but the 12V input is intended as the main power input during normal usage. The high voltage components(e.g. solenoid valves) are only powered from the 12V line, but the digital components can be powered from either the USB or the 12V power supply filtered through the voltage regulator.

2.3 Dispensing System

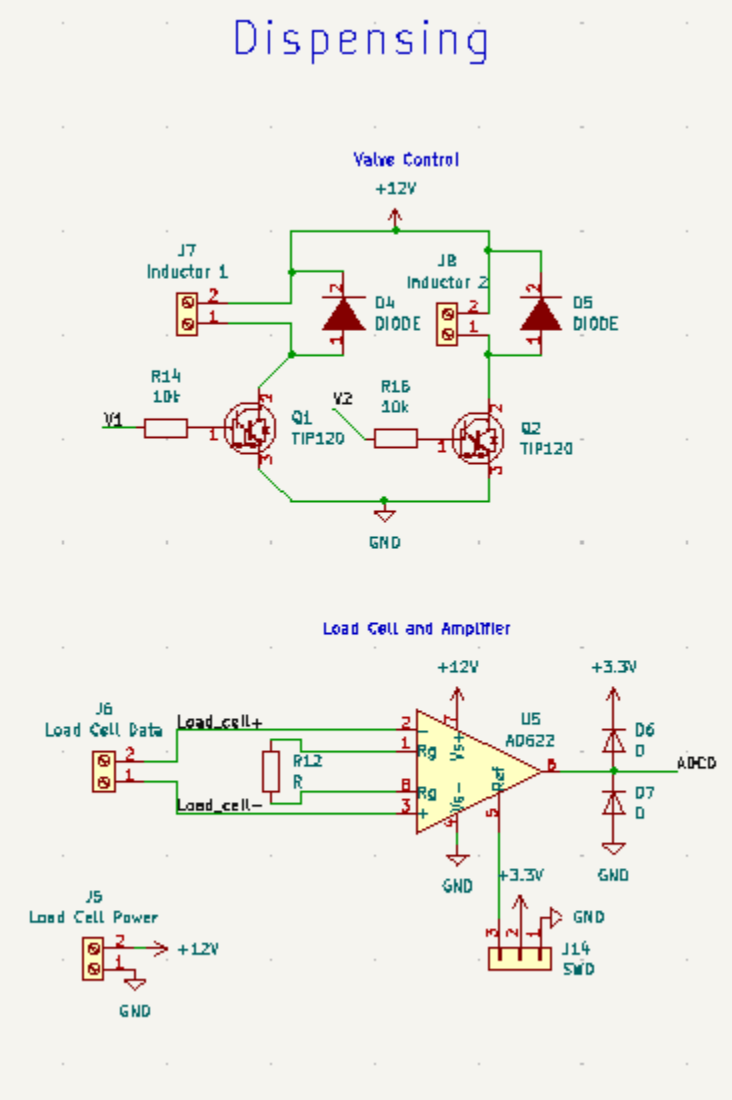


Figure 4: Dispensing Schematic

The dispensing system consists of two halves: solenoid valves and a load cell. The load cell is composed of a strain gauge (essentially a resistor whose value changes as it is bent) attached across a Wheatstone bridge. This generates a small differential voltage that is then fed through an instrumentation amplifier. The solenoid valves are simple 12V 400mA valves that are by default closed, each of which is connected to a Darlington BJT transistor and a flyback diode.

Originally, we planned to feed the load cell values through the AD622 instrumentation amplifier which would allow the value to be read by an ADC port of the microcontroller. We added in an analog to digital converter (ADC) protection circuit at the output of the amplifier to ensure we didn't overload the ADC on the microcontroller as the instrumentation amplifier required 12V while the microcontroller could only handle 3.3 V. After unit testing this section of the design on the breadboard, it was clear that something was wrong and it wasn't going to work. Instead, we came up with another way to get the load cell values over to the microcontroller.

Our new method involved using an HX711 amplifier which had the 24-bit ADC built in. We used a MicroPython HX711 library to tare the scale and analyze the values coming from it [2]. This allowed the 2^24 or roughly sixteen million unique values that the ADC was capable of differentiating. A further advantage was that we could power this amplifier from the 3.3V line. However, this amplifier complicated our design because it did not conform to a standard data transfer protocol. We were successfully able to read values from the load cell this way, however, it was too late to make a spot for the HX711 on the printed circuit board.

2.4 User Interface

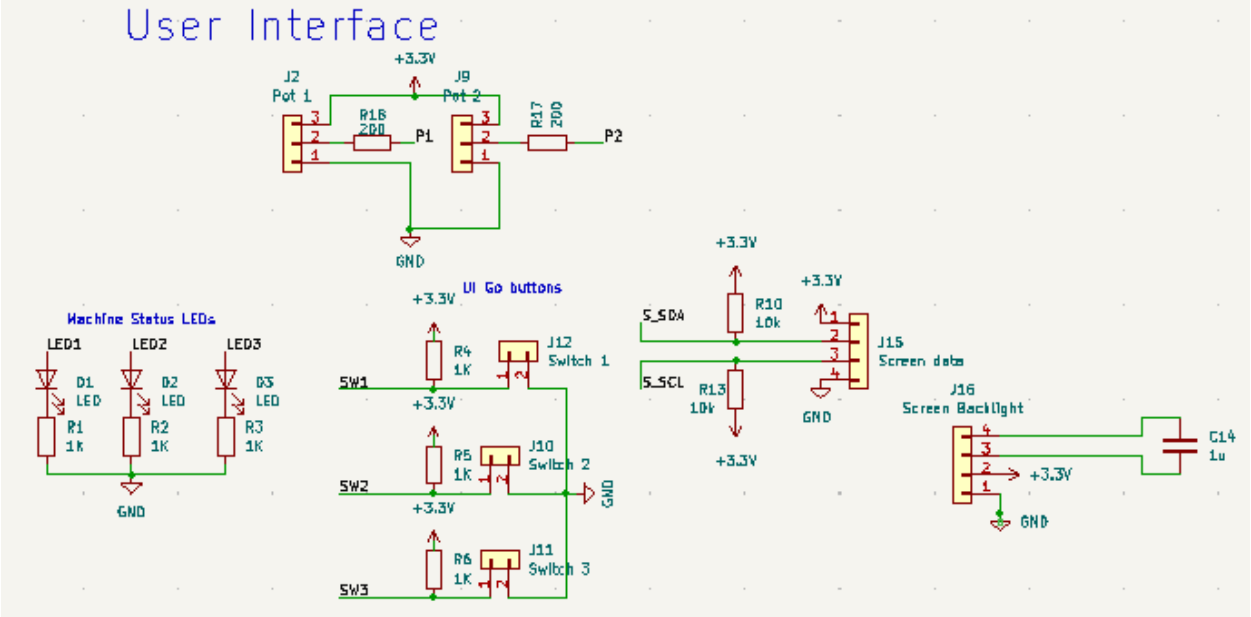


Figure 5: User Interface Schematic

The user interface consists of two potentiometers, three push-buttons, three LEDs and a screen. Two buttons are located underneath the bottles which lets the user know which button to press to dispense an item. The potentiometers are each located underneath the buttons to signify that turning it controls the amount to be dispensed for that particular bottle. As the potentiometers are turned the value gets sent to the microcontroller which sends the proper value to the screen to update the amount to be dispensed in grams. There is a red button in the middle of the user interface which resets the machine after a restocking takes place or overflow gets cleaned up. This button sends the machine back to the beginning where it's waiting for a user to place an order. There are three LEDs on the side of the machine which indicate to the user if the machine is ready for an order (green), the machine is processing an order (orange), or if the machine needs maintenance (red).

Originally, we planned to use the NHD-C0220BIZ screen from Newhaven Display as we thought it would interface well with the I2C while consuming little power. Unfortunately, that was not the case. After carefully building out the circuit specified in the datasheet and turning on power, the screen would heat up immediately and no text could be

displayed. We decided to switch out the screen for an OLED LCD Display Board Module I2C IIC SSD1306. This screen was easier to use as there were only 4 pins instead of 12 and the screen worked well with the Machine, SSD1306, and OLED libraries in MicroPython[3]. The screen communicates the amount to be dispensed to the user, live-time updates of the amount dispensed, the final amount dispensed, and if the machine needs maintenance.

2.5 Control Unit

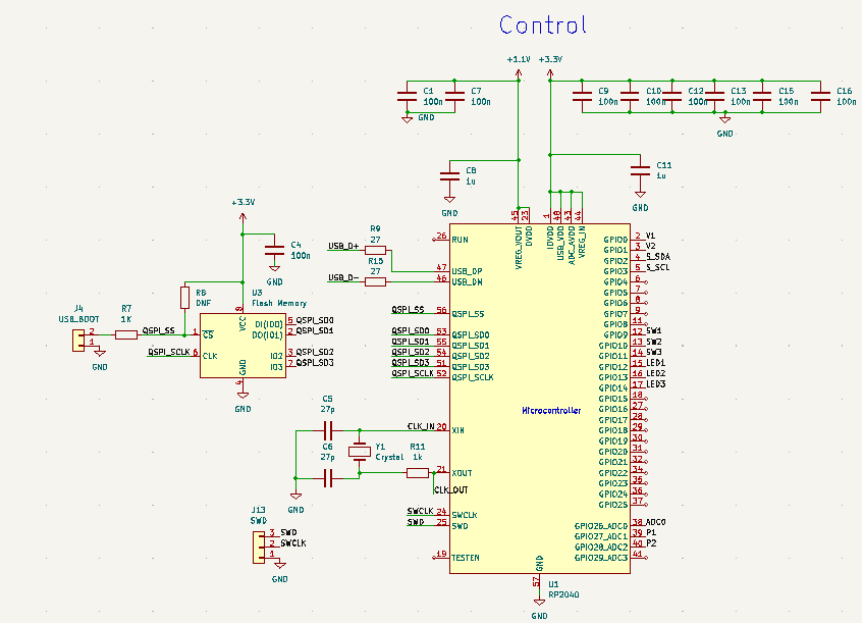


Figure 6: Control Unit Schematic

We used an RP2040 in combination with flash memory on the PCB. Our design is inspired by the one in the RP2040 Hardware design doc [4]. In order to load code to the RP2040, a USB connection was made to the computer, the code was downloaded and stored on the flash memory and then the program would continuously run. We chose the RP2040 as it was quite powerful for the price, which made it useful for scalability of the dispensing machines, and it is programmable using a variety of IDEs.
For testing purposes, we used a Raspberry Pi Pico on a breadboard which was cheap ($4), easily replaceable, and allowed us to test each subsystem before placing it on the PCB. Additionally, since the code relied on GPIO pins instead of physical pin numbers, we were able to interchange the code on the Pico for the RP2040 seamlessly.

3 Subsystem Results

$$\delta = \left| \frac{v_A - v_E}{v_E} \right| \cdot 100\%$$

Equation 1: Formula for calculating percent error

3.1 Power

Requirement: Power supply capable of generating 500mA on 3.3V line at +/- 0.2V
Verification: While the machine is running, we will measure the voltage at the voltage transformer, as well as measuring the current drawn under load.
Results: The power supply worked as intended. We were able to measure a very consistent 3.289V on the 3.3V line. However, we were unable to test the current capacity on this line as it would have been difficult to break out a series connection from the voltage regulator.

Requirement: Power supply capable of generating 1A on 12V line at +/- 0.5V
Verification: While the machine is running, we will measure the voltage at the voltage transformer, as well as measuring the current drawn under load.
Results: We were able to measure an appropriate voltage on the 12V line. However, again it would not have been feasible to measure the current as creating a series connection was difficult.

Line	Recorded Voltage
12V	12.0995 V
12V to 3.3V	3.28983 V
5V to 3.3V	3.28974 V

Table 1: Voltage values for Power

3.2 Dispensing System

Figure 7: Solenoid Valves

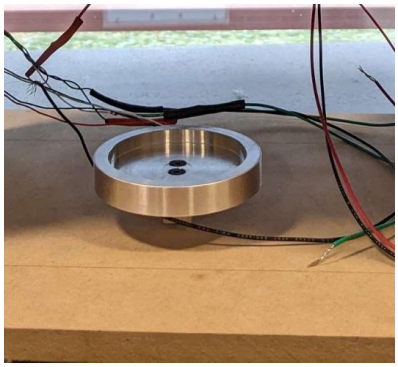


Figure 8: Load Cell + Cup Holder

Requirement: The load cell should be able to weigh items with a tolerance of 10 g for weights up to 5 kg.
Verification: We weighed several objects on our load cell and compared the values to a scale.

Trial #	Experimental Weight	Actual Weight	Percent Error
1	15.63 g	15.2 g	2.83 %
2	200.75 g	202.0 g	0.619 %

Table 2: Load Cell Accuracy Trials

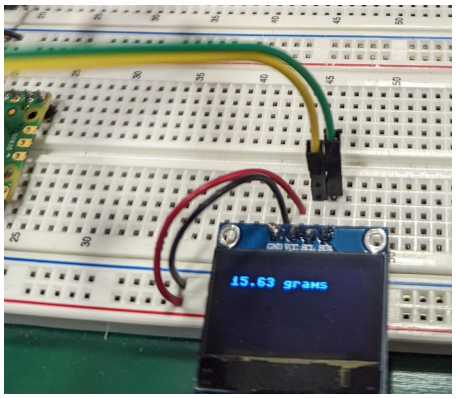


Figure 9: Trial 1 Experimental Results

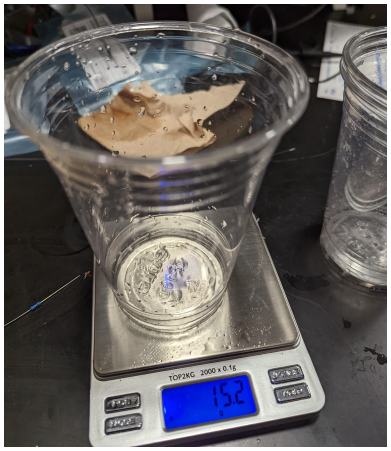


Figure 10: Trial 1 Actual Results

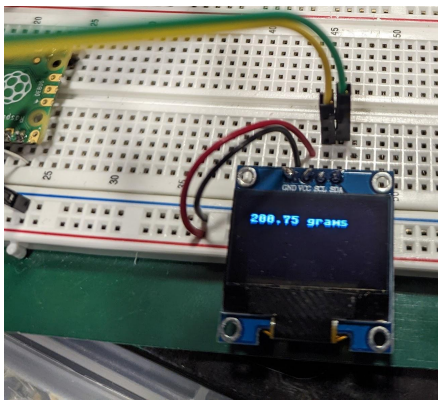


Figure 11: Trial 2 Experimental Results

Figure 12: Trial 2 Actual Results

Results: The load cell ended up being more precise than expected. The percent error ended up being very low.

Requirement: The value spun on the potentiometer is the amount that gets dispensed.
Verification: We selected several small values on the potentiometer and then waited for the amount to get dispensed and weighed the results. The small weights are due to the slow flow rate of the machine.

Trial #	Amount Requested	Actual Weight	Percent Error
1	12 g	13 g	8.33 %
2	13 g	15 g	15.4 %

Table 3: Dispensing System Accuracy Trials

Results: The user ends up getting slightly more than what they requested. However, this is acceptable as it is advantageous to the customer.

Requirement: Voltage across inductor should always be less than the specified maximum voltage (12V) of the BJT
Verification: Measure the voltage across the inductor before and after closing the BJT switch.

Component	Recorded Voltage (V)
Solenoid 1 Diode	11.158
Solenoid 2 Diode	11.176

Solenoid 1 Screw Terminal	11.318
Solenoid 2 Screw Terminal	11.202

Table 4: Voltage across the inductors

Results: The drop across the inductor is 0V when it is turned off. When it is on the drop is under 12V.

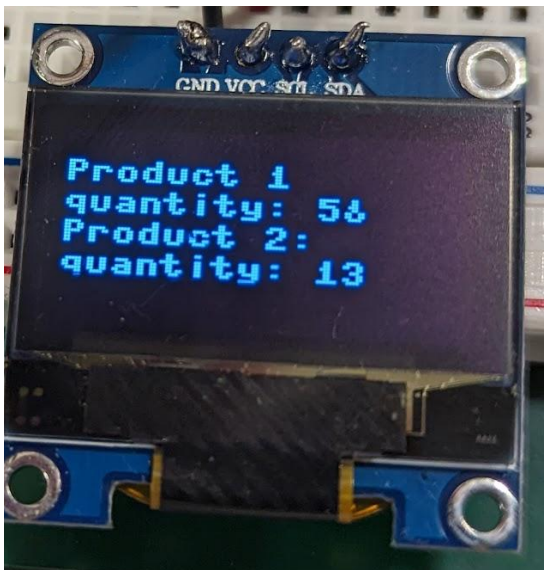
3.3 Control System

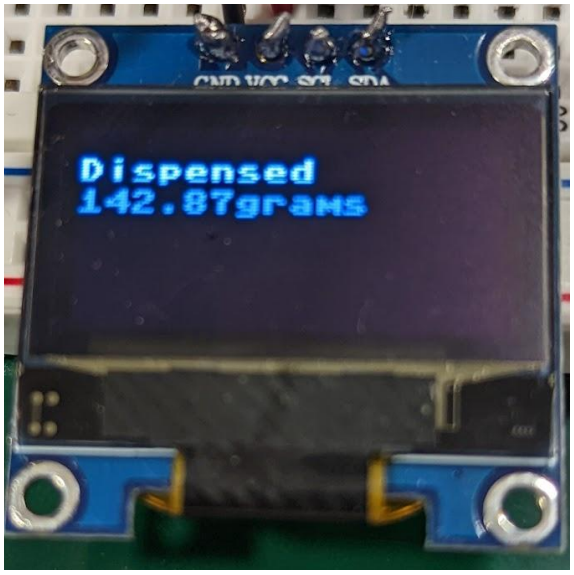
Requirement: The microcontroller's ADC should be precise enough to measure 250 unique values from the load cell.
Verification: To test this, we printed readings from the load cell with HX711 setup into the shell. We used our hands to apply variable force and watched the values go up.
Results: The Pico was able to read 65,536 different values, which goes far above this requirement.

Requirement: Microcontroller sends proper signals to other subsystems.
Verification: The correct text displays on the screen, the right item is chosen, machine status gets updated on the UI, and the proper item gets dispensed.
Results: As seen in the demo, we were able to dispense from both bottles by hitting the respective buttons and get results dispensed in the cup within the allowed tolerance. Additionally, the user saw the status LEDs change with the machine and screen text update throughout various parts of the process. Seeing the screen update can be seen in figures 9, 11, 13, and 14.

3.4 User Interface System

Requirement: The screen should use an I2C connection and be capable of displaying the item to be dispensed as well as its quantity with appropriate units.
Verification: Check that the correct text is displayed on the screen in 1 second after a button is pressed or a potentiometer is moved
Results: The screen displays different text for each state, as well as the amount selected and the amount dispensed, as seen in Figures 13 and 14. The screen is updated in real time as the amount selected and amount dispensed changes, with essentially no delay.





Figures 13 (left image) & 14 (right image): The left image shows the screen reflecting the potentiometer values during the selection state. The image on the right shows the actual quantity that got dispensed from an order.

Requirement: Status LEDS should match machine state within 1s of machine state changes.
Verification: Check the current machine state and see if the LED is the correct output. The amount of time required for the correct LED to turn on should be within 1 second of the state change.
Results: The LEDs changed almost instantaneously after a state change.

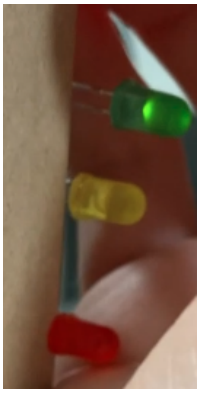


Figure 15: The Green LED is on - machine is ready for an order

Requirement: The machine should dispense the proper item.
Verification: Pressing a button will cause the machine to attempt to dispense the proper item.
Results: As seen in table 3 in section 3.2, after selecting 13 grams of a product to be dispensed, the machine dispensed 15 grams with a small amount of error.

4 Cost & Schedule

4.1 Cost Analysis

Description	Part	Manufacturer	Quantity	Extended Price	Link
Microcontroller	RP2040	Raspberry Pi	1	$1.00	Chicago Electronics
I2C Screen	SSD1306	AiTrip	1	$2.25	Amazon
Instrumentation amplifier/ADC	HX711	AiTrip	1	$6.63	Amazon
Red LED	HLMP3.301 - Red	Avago Technologies	1	$0.16	ECE Supply Center
Green LED	HLMP3507 - Green	Avago Technologies	1	$0.18	ECE Supply Center
Yellow LED	HLMP3401 - Yellow	Avago Technologies	1	$0.21	ECE Supply Center
Memory	IC FLASH 32MBIT SPI/QUAD 8SOIC	Winbond Electronics	1	$0.94	Digi-Key
Barrel Jack	Breadboard-friendly 2.1mm DC barrel jack	Adafruit	1	$0.95	Adafruit
Micro USB Plug	Micro USB Plug Female	Adafruit	1	$0.95	Adafruit
Voltage Regulator	NCP1117	OnSemi	1	$0.67	DigiKey
Push Buttons	0661273664773	Cyclewet	3	$6.29	Amazon
Rotary Potentiometer	CA-WH148-10KBK	TWTADE	2	$10.99	Amazon
Diode	IN4001	ON Semiconductor	4	$1.50	Adafruit
Transistor	TIP120	Fairchild	2	$2.50	Adafruit
100 nF Decoupling	1C20Z5U103M0	Sprague	4	$0.92	ECE Supply

Capacitor	50B				Center
Solenoid	Plastic Water Solenoid Valve - 12V - 1/2" Nominal	Adafruit	2	$13.90	Adafruit
12V Power Supply	ALITOVE DC 12V 5A Power Supply Adapter	ALITOVE	1	$12.99	Amazon
Load Cell	4541	Adafruit	1	$3.95	Adafruit
27 Ohm Resistor	CMP0805-FX-27R0ELF	Bourns	2	$0.44	Mouser
1K Ohm Resistor	810-MMZ1608R102ATD25	TDK	6	$0.24	Mouser
10k Ohm Resistor	603-RT0603DRE1010KL	YAGEO	2	$0.24	Mouser
27 pico Farad Capacitor	C0805X270J5GAC7800	Kemet	2	$0.68	Mouser
1uF Capacitor	810-C1608X5R1H105K	TDK	2	$0.14	Mouser
10uF Capacitor	CL32Y106KCVZNWE	Samsung	2	$2.24	Mouser
100nF Capacitor	810-C1608X7R1H104K	TDK	6	$0.24	Mouser
12MHz Oscillator	ABLS-12.000MHZ-B4-T	Abracon	1	$0.22	Mouser
Total Cost				$71.42	

Table 5: Parts List

In addition to these parts, there is also the fluctuating cost of a printed circuit board. On our second order, we were able to get five boards for twenty dollars. This comes out to a price of roughly four dollars per board. Furthermore, we used a considerable amount of wire and connectors - both JST and screw terminals. We estimate that we used less than ten dollars worth of connectors and wire.

The largest cost of course was labor. We estimated in our design document that the cost of our labor would be $34,942.50, as well as a fee from the machine shop on the order of $400.

This brings our final total cost to **$35,417.92**

4.2 Work Distribution

	Michael	Jackson	Lyla
9/19	Design Document	Design Document	Design Document
9/26	Design Document	Design Document	Design Document
10/3	Ordered components from online retailers and ECE supply center	Began PCB design	Talked to the machine shop and ordered components.
10/10	Started writing main loop code in Arduino	Finished PCB design	Started the Software in Arduino. Wrote code & tests for the user interface.
10/17	Finished main loop	Verified components and ensured they operated at stated voltages	Broke Picos trying to get software working.
10/24	Soldered on screw terminals, pin headers, and USB onto PCB, set up dispensing system	Soldered components onto PCB, tested power system	Broke more Picos while getting the potentiometer to work.
10/31	Soldered on SMD resistors and capacitors	Redesigned PCB with larger components and better placements.	Began working with the screen and realized we would need a completely new one in order to get this to work.
11/7	Switched the main loop to use more functions and states, added screen printing functionality using an Arduino library, added ability to calculate weight from an ADC. Decided to	Began working with software on PCB, attempted to get USB to work on PCB, but ran into many problems	All software parts for the user interface work with Pico except the screen as I was waiting for the new one to come in. Switched to using MicroPython instead of Arduino IDE because Arduino liquid crystal screen libraries weren't compatible with the Pico. Also realized

	switch from Arduino code to MicroPython. Added switch to power subsystem and tested it, swapped out RP2040 to fix voltage issue		that the instrumentation amplifier wasn't going to work so I ordered the HX711.
11/14	Added screen printing functionality with all errors and warnings to MicroPython code using a MicroPython library for the new screen. Soldered components and RP2040 onto redesigned PCB, tested power system, attempted to install software onto RP2040 using USB. Replaced some chips to get the USB connection working.	Calibration and finalizing PCB construction. Redid complete design on pico at last second as amplifier fried PCB.	Got the new screen to display text and the load cell to print values with the HX711.
11/21	Debugged main loop python code, made sure it compiled on a pico	Fall Break	Fall Break
11/28	Tested all subsystems together on PCB. The PCB broke, replacing all the chips didn't fix it. Switched to pico and got all subsystems to work on it instead for the demo.	Pivoted to pico when PCB failed. Constructed temporary 12V power distribution system. Final Demo	Pieced together all the code from unit tests until the machine finally worked!
12/5	Presentation/Report	Presentation/Report	Presentation/Report

Table 6: Work Schedule

5 Conclusion

Overall, this project was successful as we have completed all the requirements that we set for ourselves. Along the way we have encountered many challenges which have made us stronger engineers in the end. When we realized that some parts wouldn't work -such as the original screen, and AD622- we found other solutions to get the job done. The countless times we have spent resoldering parts of the board and Pico headers, have only made us faster and extremely familiar with the parts we were working with. The successes from this project came from unit testing and slowly putting pieces together. The day before the project was due, the PCB broke which was unfortunate as all subsystems were working on it as intended except the load cell part. Instead of giving up, we went back to using the Raspberry Pi Pico with the breadboard and were able to rebuild the circuit and have it working in a few hours. The breadboarded circuit was largely the same as the PCB version. If we were to do another PCB order, we would add two GPIO pins to be able to use the HX711. Based on the results from the Raspberry Pi Pico tests with the HX711, we are confident that this new PCB would work perfectly. In addition, to be able to rebuild the machine faster in the future, we would add more through-hole components to make soldering easier. We would also change the tubing with the valves to allow liquids to get dispensed in a more timely fashion. Although the machine was accurate, liquids came out way too slowly and made it very difficult to test dispensing large amounts. Lastly, we would swap out the RP2040 for a microcontroller with integrated flash to reduce the complexity of the design. When bad code got uploaded to the PCB, it would break both the RP2040 and the flash which made us spend time replacing two chips on the PCB instead of one. If we were to turn this project into a business, we would add in more risk mitigation by incorporating a draining system in the cup holder part.

5.1 Ethics:

Ethics are of paramount importance to our project. Our design aimed to reduce plastic waste in the environment, thereby complying with the sustainability clause of the IEEE code of ethics [5]. Dealing with products people put in their bodies only increases our

responsibility. We have done our best to reduce the chances of cross contamination, but the fatal flaw in our design is that both items get dispensed in a common funnel, where it is possible for the liquid to hit the sides. This increases the chances of cross contamination between different orders. If we had to do this project again, we would forgo the funnel in favor of a different component to isolate the liquids from each other. In order to give users a proper understanding of the implications of using the machine, we have added the following warning label.

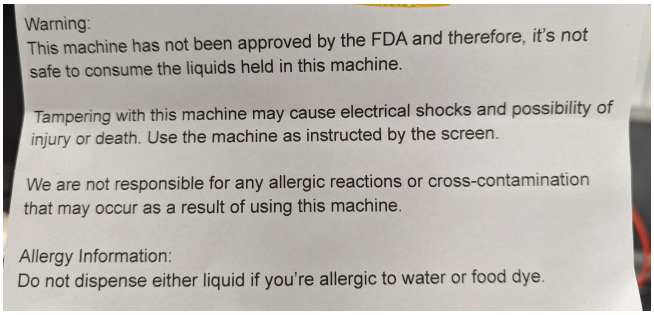


Figure 16: Warning Label

5.2 Safety:

Since the machine lacks a Food Sanitation Certification and Food Handler Training, we will not permit anyone to consume anything coming out of the machine. Our machine has a status bar which indicates to the user whether it may be used or not. On the exterior of the machine, warnings and allergy information are posted. Additionally, in our design, we made sure that food and wires didn't not mix. We also have a reset button that ignores all orders until the machine can be properly serviced. If we were to design this machine again, we would make it so people could bump into the machine without the fear of wires mixing with liquids. To do this, we would hollow out the walls of the machines and have all wires properly sealed in there. Lastly, we would include an emergency shut-off button that kills the power in the machine.

6 References

[1] “Containers and Packaging: Product-Specific Data,” *EPA*. [Online]. Available: https://www.epa.gov/facts-and-figures-about-materials-waste-and-recycling/containers-and-packaging-product-specific#PlasticC&P. [Accessed: 14-Sep-2022]

[2] S. Piskunov, “micropython-hx711,” *GitHub*. [Online].Available: https://github.com/SergeyPiskunov/micropython-hx711 [accessed Dec. 07, 2022].

[3] Stlehmann, “Stlehmann/micropython-SSD1306: A fork of the driver for SSD1306 displays to make it installable via UPIP,” *GitHub*. [Online].Available: https://github.com/stlehmann/micropython-ssd1306. [Accessed: 07-Dec-2022].

[4] “Hardware Design with rp2040,” *Raspberry Pi Foundation*. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf. [Accessed: 25-Sep-2022].

[5] “IEEE code of Ethics,” *IEEE*. [Online]. Available:https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 14-Sep-2022].

7 Appendices

7.1 Software Used in Demo

```
from machine import Pin, I2C, ADC
from ssd1306 import SSD1306_I2C
from oled import Write, GFX, SSD1306_I2C
from oled.fonts import ubuntu_mono_15, ubuntu_mono_20
from hx711 import HX711
import utime

# CHECK ALL THE PINS AGAIN BEFORE RUNNING THE CODE

# screen setup
WIDTH = 128
HEIGHT = 64
i2c = I2C(1, sda = Pin(2), scl = Pin(3), freq = 200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
text = ''

# port declarations
pot_1 = machine.ADC(27)
pot_2 = machine.ADC(28)

button_1 = Pin(9, Pin.IN,Pin.PULL_UP)
button_2 = Pin(10, Pin.IN,Pin.PULL_UP)
reset_button = Pin(11, Pin.IN,Pin.PULL_UP)

green_led = Pin(12, Pin.OUT)
yellow_led = Pin(13, Pin.OUT)
red_led = Pin(14, Pin.OUT)

valve_1 = Pin(0, Pin.OUT)
valve_2 = Pin(1, Pin.OUT)

load_cell = ADC(26)
# ---------------

# This class is from https://github.com/endail/hx711-pico-c
class Scales(HX711):
   def __init__(self, d_out, pd_sck):
       super(Scales, self).__init__(d_out, pd_sck)
```

```
        self.offset = 0

    def reset(self):
        self.power_off()
        self.power_on()

    def tare(self):
        self.offset = self.read()

    def raw_value(self):
        return (self.read() - self.offset) / 4.057

    def stable_value(self, reads=10, delay_us=500):
        values = []
        for _ in range(reads):
            values.append(self.raw_value())
            #sleep_us(delay_us)
        return self._stabilizer(values)

    @staticmethod
    def _stabilizer(values, deviation=10):
        weights = []
        for prev in values:
            if prev == 0:
                weights.append(0)
            else:
                weights.append(sum([1 for current in values if abs(prev - current) /
(prev / 100) <= deviation]))
        return sorted(zip(values, weights), key=lambda x: x[1]).pop()[0]
# ------- end of HX711 class

# This function turns on and off the machine status LEDs on the user interface
def updateLEDS(green_status, yellow_status, red_status):
 green_led.value(green_status)
 yellow_led.value(yellow_status)
 red_led.value(red_status)

# This function changes the text on the screen to keep the user informed
def updateScreen(text, pot_1_state, pot_2_state, w = 0):
 if (w < 0):          # prevents noise from making values go negative
     w = w * -1
```

```
 if text == 'Select': # show both products, vals of potentiometers for both
   oled.fill(0)
   oled.text("Product 1 ", 0, 10)
   oled.text("quantity: " + str(pot_1_state), 0, 20)
   oled.text("Product 2: ", 0, 30)
   oled.text("quantity: " + str(pot_2_state), 0, 40)
   oled.show()
  elif text == 'Dispense': # called while the machine is in dispensing mode to
continuously update the screen with the current amount
   oled.fill(0)
   oled.text(str(w) + " grams", 5, 10)
   oled.show()
  elif text == 'SUCCESS': # Shows how much was dispensed as soon as machine is done
   oled.fill(0)
   oled.text("Dispensed ", 0, 10)
   oled.text(str(w) + "grams", 0, 20)
   oled.show()

 elif text == 'NormalA': # Shown after the button for product A is pressed
   oled.fill(0)
   oled.text("Selected ", 0, 10)
   oled.text("Product 1", 0, 20)
   oled.text("Quantity = ", 0, 30)
   oled.text(str(pot_1_state) + " grams", 0, 40)
   oled.show()

 elif text== 'NormalB': # Shown after the button for product B is pressed
   oled.fill(0)
   oled.text("Selected ", 0, 10)
   oled.text("Product 2", 0, 20)
   oled.text("Quantity = ", 0, 30)
   oled.text(str(pot_2_state) + " grams", 0, 40)
   oled.show()

 elif text == 'NoContainer': # Shown if there's no container detected in the cup
holder
   oled.fill(0)
   oled.text("Error: ", 0, 10)
   oled.text("Must Place", 0, 20)
   oled.text("Container!", 0, 30)
   oled.show()
```

```
 elif text == 'NoQuantity': # Shown if the user tries to display
   oled.fill(0)
   oled.text("Error: ", 0, 10)
   oled.text("Must Select", 0, 20)
   oled.text("Quantity!", 0, 30)
   oled.show()

 elif text == 'Reset': # Shown if the reset button is pressed
   oled.fill(0)
   oled.text("Maintence ", 0, 10)
   oled.text("Required", 0, 20)
   oled.text("Out of order!", 0, 30)
   oled.show()

 elif text == 'OVERFLOW': # shown if overflow has been detected
   oled.fill(0)
   oled.text("Overflow", 0, 10)
   oled.text("Detected", 0, 20)
   oled.show()

 elif text == 'OUTOFSTOCK': # shown if the item requested is out of stock
   oled.fill(0)
   oled.text("Item is ", 0, 10)
   oled.text("out of stock", 0, 20)
   oled.show()


# opens the respective valve
def openValve(v):
 if v == 1:
   valve_1.value(1)
 elif v == 2:
   valve_2.value(1)

# closes both valves
def closeValves():
 valve_1.value(0)
 valve_2.value(0)


# this function is called as soon as a valve opens.
# It constantly checks the weight of the container with the weight of the load cell
```

```
def fillUp(w):
    updateLEDS(0, 1, 0)                         # turns on yellow LED
    scales = Scales(d_out = 5, pd_sck = 6)      # prepares the scale
    scales.tare()
    count = 0
    prev_value = 0
    load_cell_val = round(scales.raw_value() / (2**22) * 30000, 2) # converts to grams
    utime.sleep(1)
    reset_state = not (reset_button.value())

    while (load_cell_val < (0.9 * w) and load_cell_val != w):
        reset_state = not (reset_button.value())
        if reset_state:     # if the reset button is pressed, enter the reset state
            resetState()

        load_cell_val = round(scales.raw_value() / (2**22) * 30000, 2)

        if (load_cell_val > (3 * w)):
            updateScreen('OVERFLOW', 0, 0, 0)
            updateLEDS(0, 1, 0)
            closeValves()
            return 'OVERFLOW'
        if (count == 50):
            updateScreen('OUTOFSTOCK', 0, 0, 0)
            updateLEDS(0, 1, 0)
            closeValves()
            return 'OUTOFSTOCK'

        if prev_value == load_cell_val: # used to determine if product is out of stock
            count = count + 1
        else:
            count = 0

        if load_cell_val < prev_value:  # preventing noise
            load_cell_val = prev_value
        else:
          updateScreen("Dispense", 0 , 0, load_cell_val)

        prev_value = load_cell_val
        utime.sleep(0.01)

    updateScreen('SUCCESS', 0, 0, round((scales.raw_value() / (2**22)) * 30000, 2))
```

```
    updateLEDS(1, 0, 0)                         # set green LED
    closeValves()
    return 'SUCCESS'

# sends the machine into a waiting state, in order to re-enter the main loop,
# the reset button must be hit again
def resetState():
    closeValves()
    utime.sleep(3)
    while True:
        reset_state = not (reset_button.value())
        if reset_state:
            refillDispensary()

def refillDispensary():
    res = ""
    button_1_state, button_2_state, reset_state, pot_1_state, pot_2_state = 0, 0, 0, 0,
0
    scales = Scales(d_out = 5, pd_sck = 6)
    scales.tare()
    updateScreen('Select', pot_1_state, pot_2_state, 0)
    updateLEDS(1, 0, 0) # green LED
    closeValves()

    while True:
        button_1_state = not (button_1.value()) # gets in UI values
        button_2_state = not (button_2.value())
        reset_state = not (reset_button.value())
        pot_1_state = pot_1.read_u16()
        pot_2_state = pot_2.read_u16()

        if pot_1_state > 50000:    # max value for load cell is 5 kg
            pot_1_state = 50000    # cap values that are larger
        elif pot_1_state < 400:    # anything less than 400 could have noise
            pot_1_state = 0

        if pot_2_state > 50000:
            pot_2_state = 50000
        elif pot_2_state < 400:
            pot_2_state = 0
         pot_1_state = int(pot_1_state / 50) # convert to grams
        pot_2_state = int(pot_2_state / 50)
```

```
        if (button_1_state):                # button 1 pressed
            if pot_1_state < .01:
                updateScreen('NoQuantity', pot_1_state, pot_2_state)
            else:
                updateScreen('NormalA', pot_1_state, pot_2_state)
                updateLEDS(0, 0, 1) # set orange LED
                valve_2.value(1)
                res = fillUp(pot_1_state)

            utime.sleep(10)

        elif (button_2_state):              # button 2 pressed
            if pot_2_state < .01:
                updateScreen('NoQuantity', pot_1_state, pot_2_state)
            else:
                updateLEDS(0, 0, 1) # set orange LED
                updateScreen('NormalB', pot_1_state, pot_2_state)
                valve_1.value(1)
                res = fillUp(pot_2_state)

            utime.sleep(10)

        elif (reset_state):                 # reset pressed
            updateLEDS(0, 1, 0) # set red LED
            updateScreen('Reset', pot_1_state, pot_2_state)
            valve_1.value(0)
            valve_2.value(0)
            resetState()
        else:
            updateScreen('Select', pot_1_state, pot_2_state)


refillDispensary() # starts the software for the refillDispensary
```