Final Report

Final system specifications

The idea of the project is to build a drinker mixer machine that allows users to create and order drink recipes using an iOS app that communicates with a server that sends the order to an Arduino which then controls four pumps to dispense liquid into a cup.

The user can use the phone app save, edit, and delete custom recipes and order drinks based off these recipes. When a user wants to make an order, he/she taps their phone to an NFC tag that allows the app to send an order to a server. The Arduino Uno R3 uses a WiFi module that reads orders from the server and the Arduino parses the data to control the appropriate peristaltic pumps based on the ingredients in the order. During drink order processing, the order details are shown through a TFT display and prompts the user to place a cup under the nozzle. The cup is detected using a ultrasonic sensor. The volume of liquid dispensed will be controlled by the microcontroller via timers. Additionally, the TFT display shows user data such as total amount consumed.

Key progress from previous week

Over the previous week, we soldered all of our components onto our PCB and built the final chassis. Building the final chassis included laser cutting out top, bottom, and sides of our base. In addition, we 3D printed the neck and head. We also updated the phone app to support adding, saving, editing, and deleting recipes. Finally, we integrated all of the components with the chassis.

Open problems

The biggest open problem we have is not being able to accurately dispense when the motors are on. We suspect that this is due to interference with the Arduino through a common ground between the motors, power supply, and Arduino. When the relays are not connected to the common ground, the timing of the motors are accurate. Our efforts to resolve this would be to separate the grounds of the 15V power supply, buck converter and Arduino.

Currently, the output of the flow sensors are not accurate when the motors are running, we the motors are disconnected, the flow sensors perform as expected. We found there was interference from the common ground between the motors and the Arduino. We tried using tinfoil to act as a radiant shield for the signal wire but that did not work well.

We are limited to sending a payload of 64 bytes from the WiFi module to the serial buffer of the Arduino. We tried to read the payload as the WiFi module was sending it but were unable to find a consistent, working solution.

Testing & Results

- Accuracy tests were conducted for several volume amounts for water and juice.
- Timing and flow sensor methods were tested (timing was more accurate)
- Surface tension was achieved for 1 oz to 1 cup for various combinations of water volumes
- We confirmed pumping accuracy within 0.1 oz for water
- We used Postman to send HTTP requests to test the server side processing.

Robustness

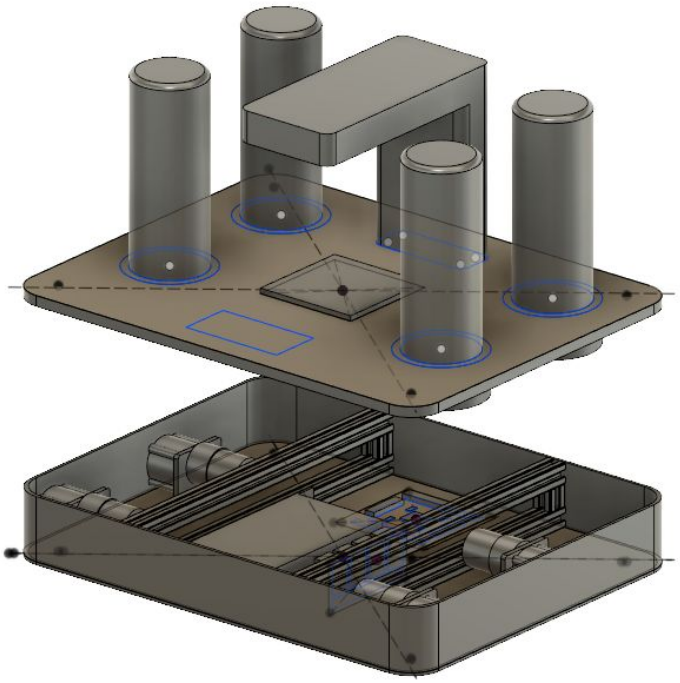
The phone app checks for invalid data in volume field for recipes and any recipe name with ASCII characters is supported. The app converts spaces to underscores in order to send valid HTTP requests. Currently, the machine does not keep track of source volume so it does not detect when the source is empty or when the user orders a drink that has more volume than the source. Also, if a recipe has an ingredient an ingredient not in stock, the app currently does not check for that. This can be easily implemented by fetching from the database which items are in stock and check with the order when the user hits send.

Final vs. Initial Specifications:

- **Server**: The server used to process HTTP requests is a Python Flask server (https://stark-beach-45459.herokuapp.com/). A MongoDB database stores user recipes, amount consumed, ingredients, and orders. Our goal was to make are endpoints work correctly, and there are currently no bugs we have found. The following endpoints were created:
    - GET /order
    - POST /order
    - POST /user
    - GET /ingredients
    - GET /recipes
    - POST /recipes
    - DELETE /recipes

- **Mobile App**: we created an iOS app that the user interfaces with to create, edit, delete and send custom recipes as drink orders. If the user wants to add a new recipe, they start from a template recipe that contains all of the ingredients in stock and they can change the volumes. This was our goal initially and were able to achieve it. One significant change we made involved using the phone as a reader vs. as a tag. Initially, the idea was to use an NFC reader to read the phone's RFID tag and send data through NFC. We found it was not possible to do this with our phones (it could be done with a Nexus 5). We switched to using the phone as a reader and send data through WiFi. This turned out to be easier to facilitate communication between the phone, server, and Arduino.

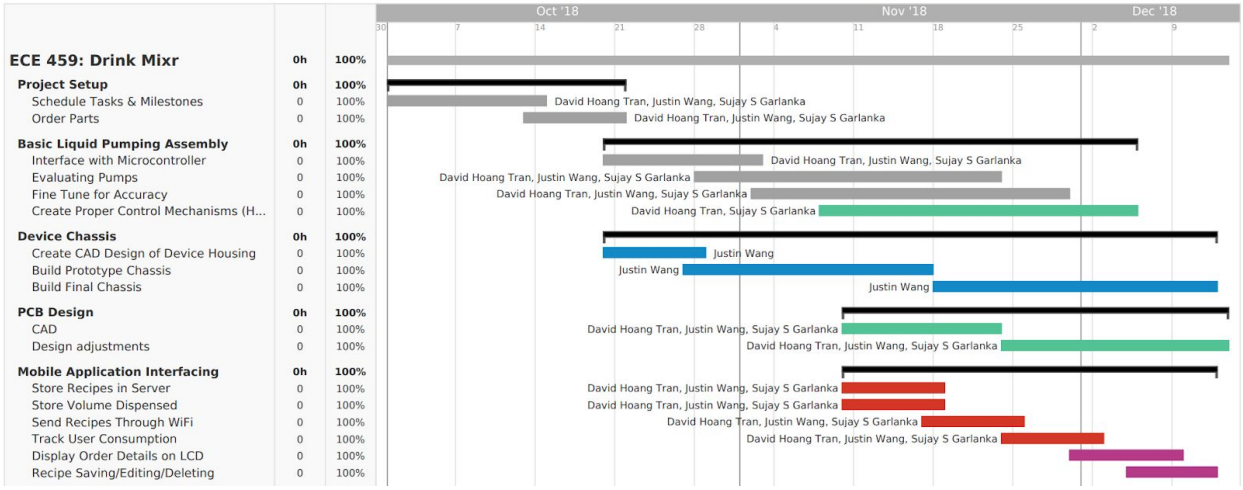
- **PCB**: our PCB is used to integrate our Arduino with its peripherals in a compact, robust fashion. We used header pins to mount the Arduino, WiFi module, and buck converter. We soldered the rest of the components via wires. Ideally, we would want to recreate our own Arduino but our general goal was to integrate without a breadboard and this goal was met.
- **Chassis:** the chassis was designed to house all electrical components in an enclosed basement chamber that doubles as the main frame of the device. The peristaltic pumps reside in this basement as well, with their tubing routed through the hollow neck of the device. To house liquid ingredients, there are four plastic cylindrical reservoirs that flank the center dispenser console. These containers are supported by 3D-printed support frames that are integrated into the basement chamber. A beverage plate resides at the center of the device that serves as an outlet for any leaks and drippage that occurs after dispensing.



Safety considerations

The tubing is food safe and the sealant used at the bottom of the sources is also food safe. Given more time, we would fully isolate the electrical components from the food components like the tubing.

Final Gantt Chart



Final vs Initial Gantt chart

- Deviations: We added more subtasks for the phone app in order to organize developing the recipe customization. We needed to extend the deadline for testing & fine tuning flow meters since we wanted to use the temperature flow meters for liquids of different viscosity. We also extended the deadline for final chassis because it took longer than expected due to 3D print times and laser cutting issues.

Final budget

- Total spent: $520
- Cost to reproduce: $421

Technical problems encountered (and how were they overcome)

- Payload parsing: The server sends an order to the Arduino as a string of 64 bytes. An example payload would be *$O:robot-456.3:water_0-0-14.575:water_1-1-5.0:water_2-2-2.7:$.* The format of the payload is "O:*user_name-amount drank-ingredient-motor number-dispense time (sec):[additional ingredient entries]".* Here, *robot* is the name of the user and *456.3* is the total about consumed by the user. *water_0* is the name of the ingredient, *0* is the motor number, and *14.575* is the dispense time of this ingredient in seconds. We parse the input by splitting by colons and dashes.
- Pumping accuracy with motors powered on: we ran into an issue with motor accuracy whenever we connected the motors to the 15V power supply. We believed the issue was caused by interference with the Arduino through a common ground between the motor and Arduino grounds.
- Parallel motor controls: we solved the problem of running motors simultaneously by keeping track of when a motor has been turned on. This was done in the Arduino code

by storing the timestamp at which a motor turned on and iteratively checking each motor's start time with the current timestamp.
- App functionality: a technical problem we resolve involved keeping the app up to date whenever the user creates/edits/deletes a recipe. We added a save button so that the user can either discard unsaved changes or save them. Whenever a user deletes a recipe, the page automatically returns to the home screen with the recipe deleted. Making sure the home screen has a correct list always was a challenge due to HTTP requests taking longer than page switching. We need to make sure the app waits for the request to complete or refresh the page automatically once it completes.

Rev 2.0

- Use H-bridges for reverse flow in order to save liquid in the tube and prevent contamination.
- Transfer bigger recipes by partitioning payload or be able to read payload as it's being sent from WiFi module to the serial buffer.
- Not use a T-junction and use four flow sensors, which enables four motors to be run in parallel.

Member contributions

- Sujay
    - Integrated Wifi module, TFT display and relays with the Arduino
    - Helped design PCB
    - Set up Python Flask server for the API
    - Developed NFC functionality of the mobile app
- Justin
    - Developed a prototype and final chassis
    - Design of the liquid reservoir pipeline
    - Component design and selection
    - Tested thermal mass flow rate sensors
- David
    - Developed the PCB schematic and layout using Eagle.
    - Developed interface and functionality of the mobile app allowing users to create, edit, delete, and send orders.
    - Helped test NFC for the original plan of reading RFID tags in phones
    - Helped test accuracy of pumps and flow sensor signalling

Components:

power supply	14.99
4 channel relay	7.86
4 pumps	123.58
1 tube	9.99
2 flow meters	45.48
arduino uno	10
5V regulator	9.99
variable buck converter	7.99
TFT LCD	31.28
wifi module (huzzah)	16.95
4 VOSS bottles	5
PING	10
PCB	27.88
Chassis parts	100
h-bridges	30
RC522	5.49
large flow meter	19.64
expensive flow meters (2)	74
DIYall ESP8266	6.99
Additional tubing	20
4 wifi modules (Maker focus)	13.99

ESP8266 802.11 (Adafruit)	14.93