ECE 445 FA24
Project Proposal

Team 20: A Better Yogurt-Maker

Team

Betty Nguyen (bcnguye2@illinois.edu)
Zitao Wu (zitaowu2@illinois.edu)
Chidambara Anagani (canaga2@illinois.edu)

TA

Angquan Yu

10/03/2024

1 Introduction

1.A Problem

Making yogurt at home is a popular and healthy hobby. However, many homecooks struggle with producing consistent tasting yogurt. Yogurt is fermented using lactic acid bacteria (LAB) that consume the sugars in milk and produce lactic acid. Yogurt is made at home using backslopping, a technique where a sample of a previous batch of yogurt is used as the starter for the new batch. However, using back slopping, we don't know the bacterial strain and amount of LAB in our starter batch.Thus it is impossible to fully predict the behavior of the yogurt during the fermentation process.

There are some smart yogurt fermentation devices on the market meant to address this issue. However, these devices only allow the user to set a timer and a temperature. It is important to monitor temperature, but time can be an unreliable indicator due to the high amount of variability in the starting culture. This can lead to differing fermentation rates, or even a dead batch of yogurt. In addition, typical yogurt fermentation times range from 8 to 24 hours which makes it difficult for a home cook to constantly monitor their yogurt. If yogurt is left to ferment too long, it will become too sour and thick. Because of all these factors, there would be significant demand for a real-time monitoring device for yogurt fermentation.

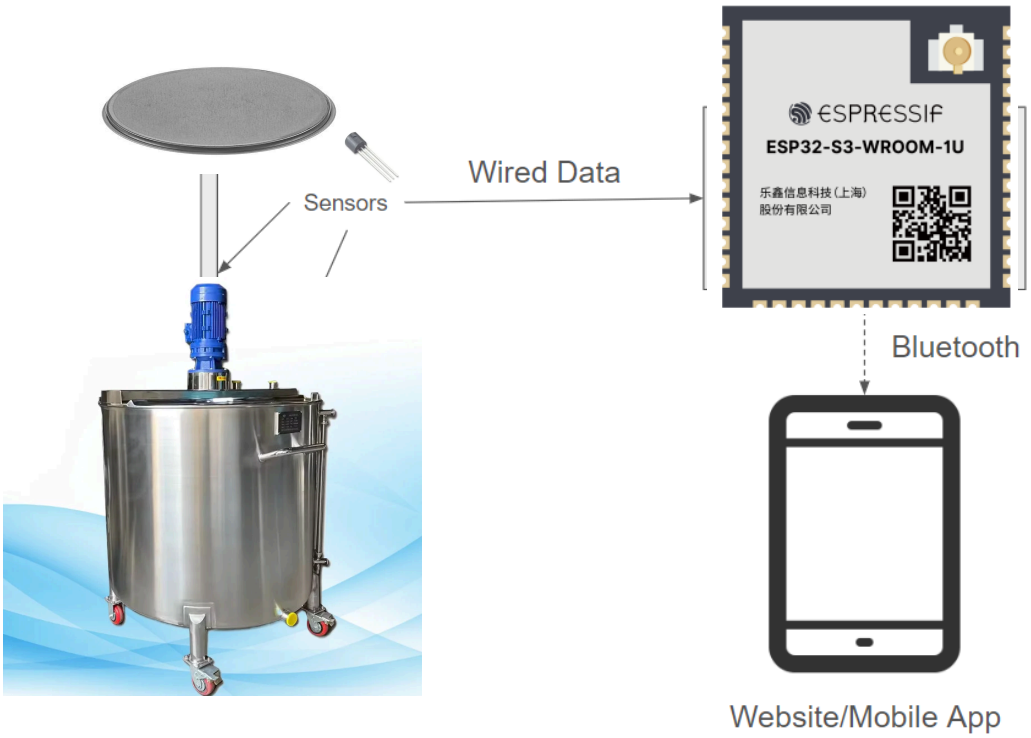
1.B Solution

Our solution is to design a real time monitoring device that uses sensors to collect data from the yogurt and transmit the information to a mobile interface. The quantities our sensors will need to measure are: taste, texture, and temperature. Temperature will be measured with a temperature sensor and taste with a pH sensor. We will approximate the texture of yogurt by finding its viscosity. Our design uses a custom rod and paddle that will measure the torque required in a rotation and use this data to approximate viscosity. The torque will be approximated by the current draw in a DC motor. All sensors will be mounted to the lid or sides of the fermentation container for user convenience and ease of use. Our texture and pH sensors will have to directly touch the yogurt to function.

All measurements taken by the sensors will be sent to a microcontroller for processing. The microcontroller extracts useful information from the sensor data and sends the data, via wifi, to a mobile interface, such as a website. This will make it easier for users to monitor their yogurt when traveling outside of their home. Our device will not tell the user when to stop the fermentation process

since multiple users may have differing taste preferences. Rather, the goal of our device is to provide the user with enough information to make their own informed decision.

1.C Visual Aid



1.D High-Level Requirements List

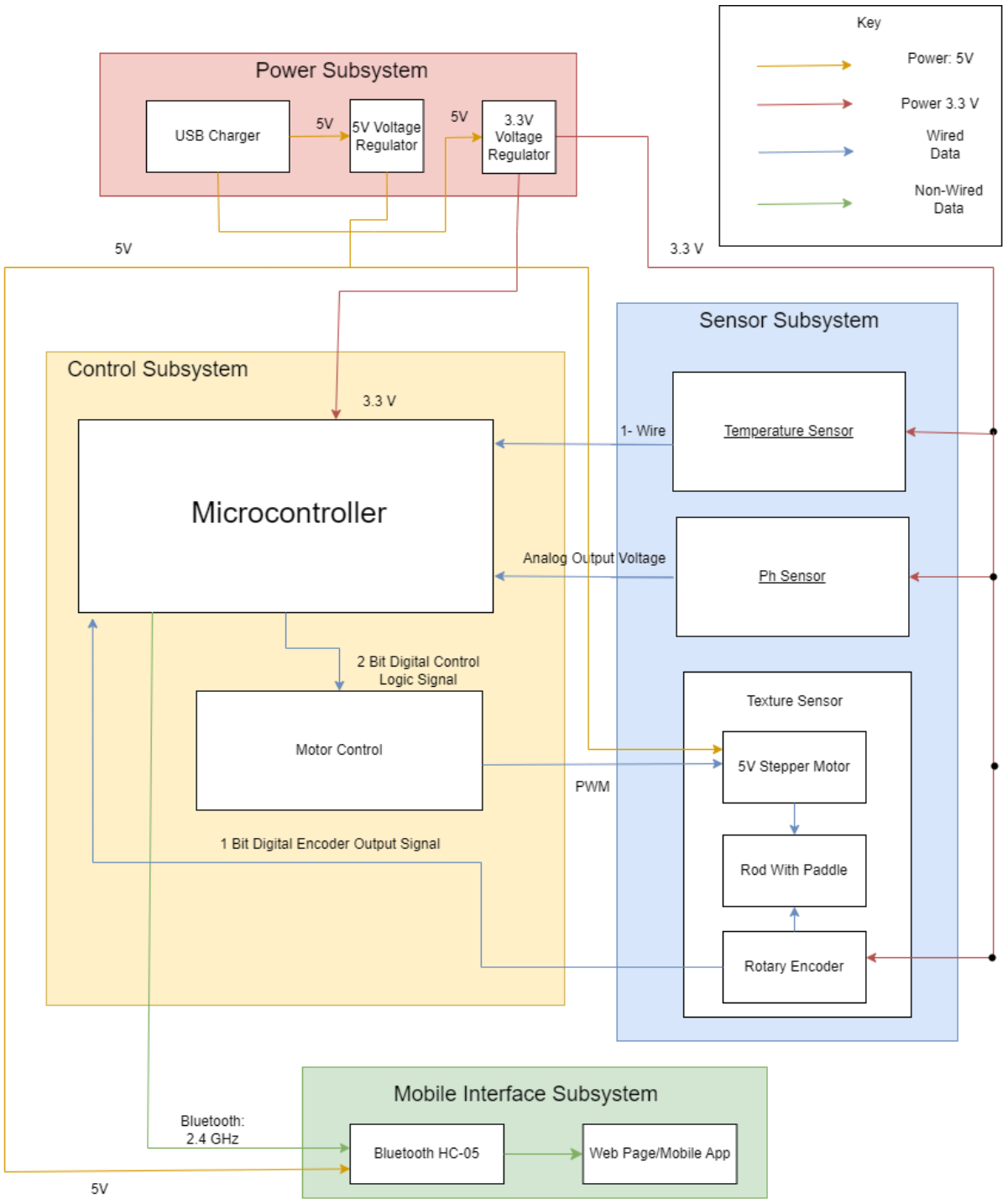
- **Ease of Use/User Interface:** We want our user interface to be easy to use and clear. The mobile application must display the yogurt pH, viscosity, and temperature in a way that is easily digestible and also provide clear actionable feedback on the yogurt fermentation process. Updates to and from the mobile app should be responsive with little latency (less than 1 second).

- **Durability:** The entire device should be able to go through a fermentation cycle (8 to 24 hours) without needing to be recharged.

- **Real Time Monitoring and Alerts:** The designed app will show the progress of the fermentation (updating the readings and charts at least once every 10 minutes) while providing any necessary alerts to the user if any of the defined conditions are out of range. All sensor readings should be accurate with a +/- 5% error range.

2 Design

2.A Block Diagram



2.B Subsystem Overview

- **Power Subsystem**

  The power subsystem will power all the electronic components in this project for the entire duration of the yogurt fermentation process. This will be made possible by using a USB cable which will provide a reliable 5V source from either a USB power adapter or a standard laptop USB port. USB power was chosen over battery power for consistent voltage, higher efficiency, and low-maintenance reasons. A voltage regulator will then be used to step down this voltage to the required 3.3V to power the ESP32 microcontroller, motor, and other sensors.

- **Sensor Subsystem**

  The sensor subsystem is responsible for detecting and calculating the necessary measurements for the pH, temperature, and viscosity of the yogurt in real-time. The three main sensors that make up this subsystem are the DFRobot Gravity Analog pH Sensor [4], DS18B20 Digital Thermometer [5] for temperature, and a DC motor with encoder that powers a rotating spindle allowing for viscosity calculations. For more information regarding the sensor subsystem, see the sensor subsystem design section below.

- **Control Subsystem**

  The control subsystem is composed mostly of the ESP32-S3-WROOM [3] microcontroller and a motor driver integrated circuit. The ESP32 microcontroller is responsible for reading and processing all of the data picked up by the sensor subsystem. The processed data is then sent wirelessly to a mobile application where users can monitor the fermentation process and receive any alerts if any preset conditions are out of range. The motor driver IC will serve as an interface between the DC motor and the microcontroller ensuring necessary voltage and current.

- **Mobile Interface Subsystem**

  The mobile interface subsystem will be an application built on Android or locally hosted on the microcontroller and will display basic sensor readings, graphs, and real-time alerts. An alert would be triggered if any measurement (temperature or pH) is outside of the defined acceptable ranges.

2.C Subsystem Requirements

- **Power Subsystem**
    - The power subsystem must supply 3.3 ± 10% Volts to all connected components under a max load of 800mA.
    - The power subsystem must be able to supply continuous power over a maximum time period of 24 hours.

Requirement	Verification
• The power system needs to supply 3.3± 10% Voltages to most of the components, while it needs 5V to supply the bluetooth and motor.	• Run the motor at different speeds, to represent the changing load. • Use a voltmeter to check whether the voltage is in the range. May need parallel connections between the power system and modules. • Since we are using a USB charger, it means we would have a stable 4.5V-5.5V power, and can use the voltmeter to verify it again.

- **Sensor Subsystem**

Requirement	Verification
• Taking the temperature sensor data as input, the sensor subsystem must convert the 1-Wire Protocol data to Fahrenheit values with an accuracy of ±5%. The sensor subsystem must also send an alarm if the ambient temperature exceeds 115 °F	• Turn on the heater and take readings using our temperature sensor and an external thermometer. Start at 80°F • Incrementally increase the fahrenheit by five degrees and record measurements from the temperature sensor and external thermometer. • Compare the data readings from the external thermometer and our temperature sensor and confirm they are within ±5% of one another. • At 115°F, record the boolean for temperature alarm from the board microcontroller via serial debugging. Confirm that the boolean changed from 0 to 1 once the alarm is triggered.
• Taking the rotary encoder data as input, the sensor subsystem must be able to determine the RPM of the spindle with an accuracy of ±5%	• Initially, make sure the spindle is at rest. Record the RPM measured from the microcontroller using serial debugging. Confirm that the RPM is 0 when the spindle is at rest. • Manually spin the spindle. Record the RPM value from the microcontroller using serial debugging. Confirm the RPM is greater than 0. • Test the spindle alone (no yogurt or other medium causing interference). Set the steps per second (SPS) and number

	of revolutions per step (RPS) of the stepper motor by changing the programming code for the microcontroller. ● Theoretical RPM can be calculated using: $Rpm = \frac{SPS * RVPS}{60}$ ● Compare the measured RPM with the theoretical RPM and confirm that the values are within ±5% of one another.
● All sensors and sensor components that come in contact with the fermentation container must be able to withstand a maximum temperature of 115 °F	● Start the heater at 100 °F and turn up the temperature in increments of 5 °F every five minutes. ● Record the output of the pH, temperature, and rpm sensors from the serial debugging port of the microcontroller. ● Confirm that the data is reliably accurate (to within ±5% error) under these operating conditions.

Sensor System Design

We must be able to extract the needed information from each sensor. Below are the algorithms proposed to decode the information from each sensor.

pH Sensing

Our pH sensor has the simplest protocol to extract information. To convert the voltage to a temperature we can use the formula: $pH = (-5.6548 * voltage) + 15.509$. One concern with our pH sensor is temperature compensation. As temperature increases, pH decreases. The sensor we chose can not perform temperature compensation but is cheaper. Additionally, in an ideal fermentation environment, the temperature should be kept constant and we can manually scale our result to adjust for temperature if needed.

Temperature Sensing

For the temperature sensor, we chose a digital 1-wire thermometer sensor. We chose this sensor because of its efficiency. The temperature sensor requires only one pin for communication with the microcontroller. In addition, power can be supplied parasitically using the communication line, eliminating the need for an external power source.

We will be using the typical hardware configuration of the temperature sensor (using parasitic power) as described by the data sheet.

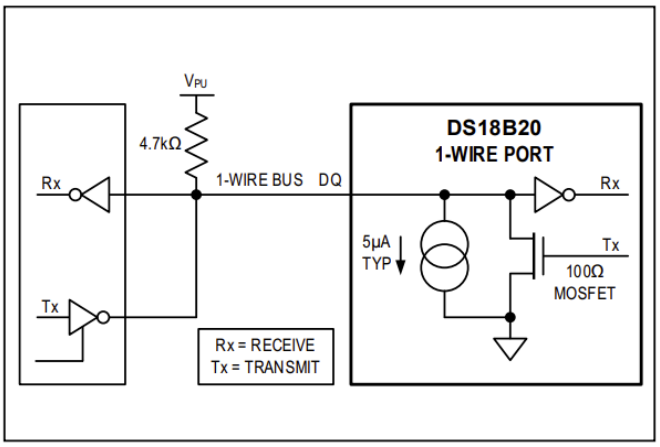


Figure 12. Hardware Configuration

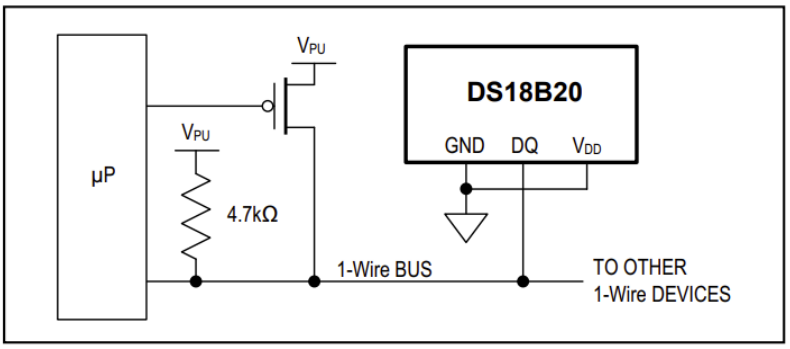


Figure 6. Supplying the Parasite-Powered DS18B20 During Temperature Conversions

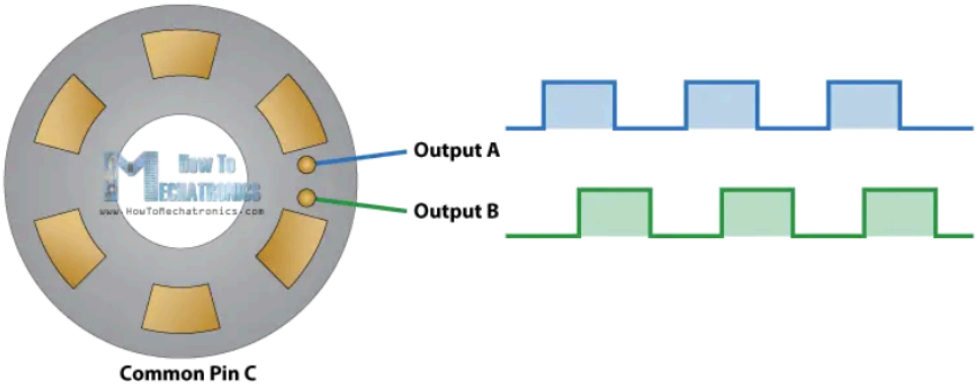
The communication protocol must start with an initialization sequence. This initialization sequence consists of a reset pulse and a presence pulse transmitted from the slave. After the initialization sequence is detected, the master can send a ROM command. A list of possible commands from the datasheet is included below. The master must then repeat this whenever a new measurement is taken. Below is an example from the data sheet of a read operation with one DS18B20 using parasitic power.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	4Eh	Master issues Write Scratchpad command.
Tx	3 data bytes	Master sends three data bytes to scratchpad (T_H, T_L, and config).
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	BEh	Master issues Read Scratchpad command.
Rx	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	48h	Master issues Copy Scratchpad command.
Tx	DQ line held high by strong pullup	Master applies strong pullup to DQ for at least 10ms while copy operation is in progress.

RPM Sensing

To measure RPM we will use a rotary encoder. The main advantage of a rotary encoder is simplicity and cost. We considered Magnetic and Hall Effect encoders but the use of a magnet could pose food safety concerns.

A rotary encoder is a disk with evenly spaced contact zones. The contact can be denoted as A, B, and a common pin C. When the disk rotates, A and B will come into contact with C and generate two square wave signals.



Since we do need to know the direction of rotation, we only need to work with one output. We can determine RPM using the formula $RPM = \frac{Frequency * 60}{Line\ Count}$. The line count is a mechanical property of the encoder, it represents the pulses per revolution of the encoder. This number can be found in the encoder data sheet. The frequency can be calculated by measuring the amount of pulses in a second. We can perform this operation in software as follows:

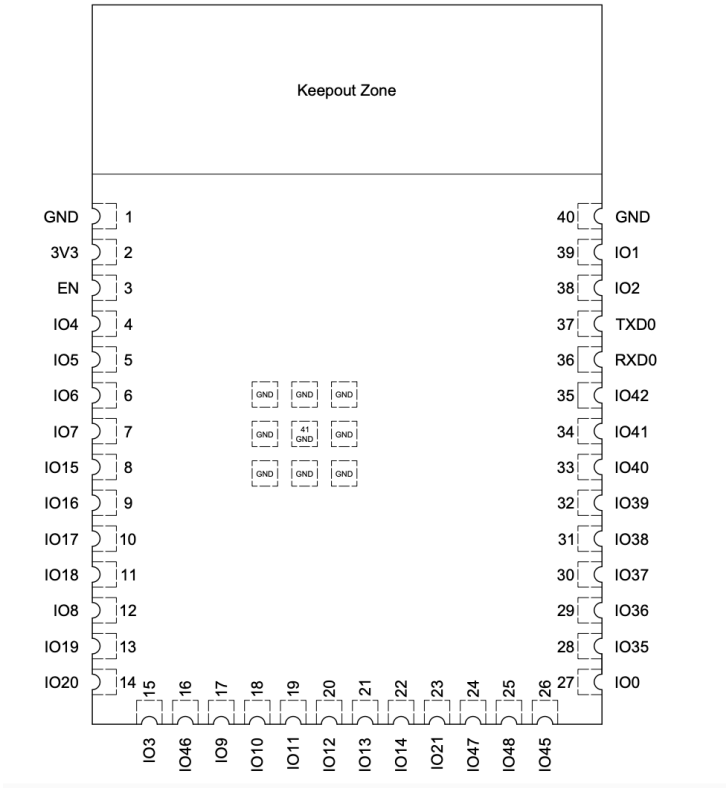
- Set a timer for 60 seconds on the microcontroller
- During those 60 seconds, increment the pulse counter everytime a pulse occurs.
- Multiply the amount of pulses by 60 seconds.
- Divide by the Line Count to get RPM.

Control Subsystem

- The control subsystem must transmit the received sensor data using Bluetooth, at a minimum rate of once every five minutes. Our plan for Bluetooth data transmission is described in the section below.
- Compute the average value temperature and pH sensor readings to reduce noise.
- Send a control signal to the motor to rotate ten rotations (or 3600 degrees) every ten minutes.
- Use the to classify the yogurt as either: runny, good, thick.

Requirement	Verification
• ESP32-S3 series, Xtensa dual-core 32-bit LX7 microprocessor, up to 420MHz, 384KB ROM, 512KB SRAM, [3]	• For the size of the ROM and SRAM, we can verify it through the IDE we are using, which would display the size of the loading data
• WIFI [3]	• WIFI component would not be used, we are using bluetooth
• Operating voltage: 3.0~3.6V[3]	• We would use a 5V voltage regulator connected with a USB charger to provide stable power • The voltage comes out of the power system will be tested first, then apply to the microcontroller
• Operating ambient temperature : 65C : -40 ~ 65C[3]	• We don't need to worry about the temperature dropping to -40C. • Instead, we need a thermometer to approximate the temperature of the microcontroller in case it is over-heat.
• When operating, it should be able to receive the data from each sensor, and transfer it to the bluetooth module.	• First, test on board using voltmeter and ampere meter to check whether the circuit works, then we should be able to use the IDE to debug and check the data transmission.

• It should be able to start/stop the sensors and data transmission at the time period we set.	• Set a small time period such as 10 seconds to check whether it is functioning, then set the desired time period such as 5 or 10 minutes.



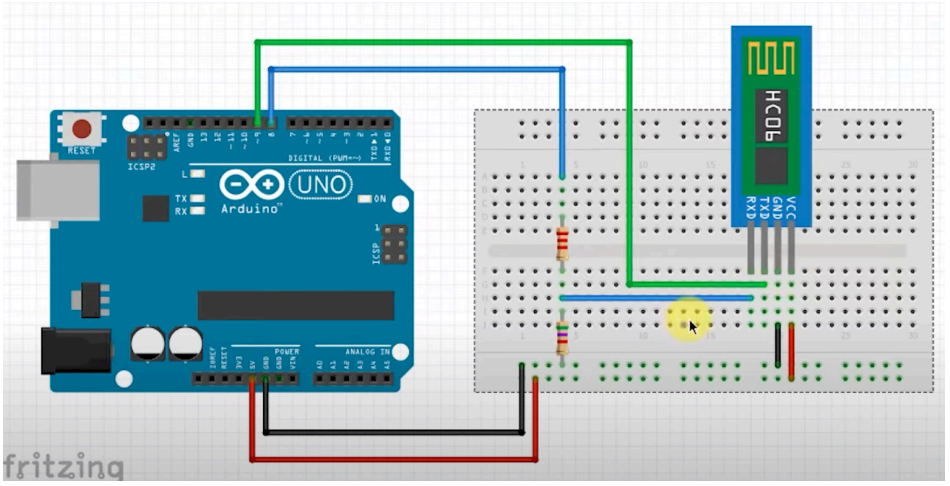
[3]

Name	No.	Type [a]	Function
GND	1	P	GND
3V3	2	P	Power supply
EN	3	I	High: on, enables the chip. Low: off, the chip powers off. Note: Do not leave the EN pin floating.
IO4	4	I/O/T	RTC_GPIO4, **GPIO4**, TOUCH4, ADC1_CH3
IO5	5	I/O/T	RTC_GPIO5, **GPIO5**, TOUCH5, ADC1_CH4
IO6	6	I/O/T	RTC_GPIO6, **GPIO6**, TOUCH6, ADC1_CH5
IO7	7	I/O/T	RTC_GPIO7, **GPIO7**, TOUCH7, ADC1_CH6
IO15	8	I/O/T	RTC_GPIO15, **GPIO15**, U0RTS, ADC2_CH4, XTAL_32K_P
IO16	9	I/O/T	RTC_GPIO16, **GPIO16**, U0CTS, ADC2_CH5, XTAL_32K_N
IO17	10	I/O/T	RTC_GPIO17, **GPIO17**, U1TXD, ADC2_CH6
IO18	11	I/O/T	RTC_GPIO18, **GPIO18**, U1RXD, ADC2_CH7, CLK_OUT3
IO8	12	I/O/T	RTC_GPIO8, **GPIO8**, TOUCH8, ADC1_CH7, SUBSPICS1
IO19	13	I/O/T	RTC_GPIO19, GPIO19, U1RTS, ADC2_CH8, CLK_OUT2, **USB_D-**
IO20	14	I/O/T	RTC_GPIO20, GPIO20, U1CTS, ADC2_CH9, CLK_OUT1, **USB_D+**
IO3	15	I/O/T	RTC_GPIO3, **GPIO3**, TOUCH3, ADC1_CH2
IO46	16	I/O/T	**GPIO46**
IO9	17	I/O/T	RTC_GPIO9, **GPIO9**, TOUCH9, ADC1_CH8, FSPIHD, SUBSPIHD
IO10	18	I/O/T	RTC_GPIO10, **GPIO10**, TOUCH10, ADC1_CH9, FSPICS0, FSPIIO4, SUBSPICS0
IO11	19	I/O/T	RTC_GPIO11, **GPIO11**, TOUCH11, ADC2_CH0, FSPID, FSPIIO5, SUBSPID
IO12	20	I/O/T	RTC_GPIO12, **GPIO12**, TOUCH12, ADC2_CH1, FSPICLK, FSPIIO6, SUBSPICLK
IO13	21	I/O/T	RTC_GPIO13, **GPIO13**, TOUCH13, ADC2_CH2, FSPIQ, FSPIIO7, SUBSPIQ
IO14	22	I/O/T	RTC_GPIO14, **GPIO14**, TOUCH14, ADC2_CH3, FSPIWP, FSPIDQS, SUBSPIWP
IO21	23	I/O/T	RTC_GPIO21, **GPIO21**
IO47 [c]	24	I/O/T	SPICLK_P_DIFF,**GPIO47**, SUBSPICLK_P_DIFF
IO48 [c]	25	I/O/T	SPICLK_N_DIFF,**GPIO48**, SUBSPICLK_N_DIFF
IO45	26	I/O/T	**GPIO45**
IO0	27	I/O/T	RTC_GPIO0, **GPIO0**
IO35 [b]	28	I/O/T	SPIIO6, **GPIO35**, FSPID, SUBSPID
IO36 [b]	29	I/O/T	SPIIO7, **GPIO36**, FSPICLK, SUBSPICLK
IO37 [b]	30	I/O/T	SPIDQS, **GPIO37**, FSPIQ, SUBSPIQ
IO38	31	I/O/T	**GPIO38**, FSPIWP, SUBSPIWP
IO39	32	I/O/T	**MTCK**, GPIO39, CLK_OUT3, SUBSPICS1
IO40	33	I/O/T	**MTDO**, GPIO40, CLK_OUT2
IO41	34	I/O/T	**MTDI**, GPIO41, CLK_OUT1
IO42	35	I/O/T	**MTMS**, GPIO42
RXD0	36	I/O/T	**U0RXD**, GPIO44, CLK_OUT2
TXD0	37	I/O/T	**U0TXD**, GPIO43, CLK_OUT1
IO2	38	I/O/T	RTC_GPIO2, **GPIO2**, TOUCH2, ADC1_CH1
IO1	39	I/O/T	RTC_GPIO1, **GPIO1**, TOUCH1, ADC1_CH0
GND	40	P	GND
EPAD	41	P	GND

[3]

- **Bluetooth/WIFI Usage**

  Since we only need to transfer a few data from several sensors, we do not need a high bandwidth transmission line such as WIFI, bluetooth will do the job. We plan to use HC-05 or HC-06 [6] which are both compatible with breadboard or PCB using arduino. It accepts voltage ranging from 3.6v to 6v, and there are a total of seven slave nodes we can use. After connecting the RXD and TXD pins to the source, we are able to do data transmitting.



bluetooth_terminal

```
#include "Arduino.h"
#include <SoftwareSerial.h>

const byte rxPin = 9;
const byte txPin = 8;
SoftwareSerial BTSerial(rxPin, txPin); // RX TX

void setup() {
  // define pin modes for tx, rx:
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  BTSerial.begin(9600);
  Serial.begin(9600);
}

String messageBuffer = "";
String message = "";

void loop() {

  while (BTSerial.available() > 0) {
    char data = (char) BTSerial.read();
    messageBuffer += data;
    if (data == ';'){
      message = messageBuffer;
      messageBuffer = "";
      Serial.print(message); // send to serial monitor
      message = "You sent " + message;
```

The first step is to assign pin numbers to RX and TX sides, then, the SoftwareSerial Object would represent our mobile device, and we set a connection between them. The PinMode would be able to set RX and TX to transmitter and receiver of data, then, it is important to set up the baud width of the transmission, in the screenshot it is 9600, we would adjust that depending on our own data. Eventually, there is a loop used for transmitting and receiving data. After the setup in arduino, we need a bluetooth terminal on the mobile device we are using. There are two metrics we are planning:

1. Directly download the bluetooth terminal that is available at the APPLE store, which allows us to transmit data.
2. Building a new app, which is able to show the data transmitted from the sensors, with a personal-designed UI interface by using the MIT app inventor. https://appinventor.mit.edu/ [7].

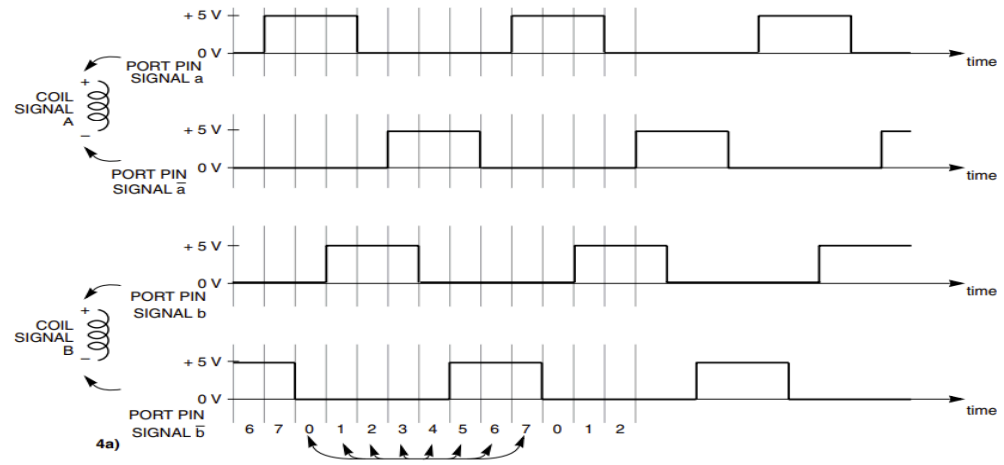
Credit of screenshots goes to: https://www.youtube.com/watch?v=NXlyo0goBrU.

Motor Design

We decided to use a stepper motor to power and control the rotating spindle. The main advantages of a stepper motor in our design would be that stepper motors are good for repeatable motions, smooth low-speed motion, and can be easily controlled.

A stepper motor works by alternating which stator phase coil is energized in sequence. This generates a magnetic field that moves the rotor in alignment. The stepper motor will be controlled using four Pulse Width Modulation (PWM) signals from the microcontroller. The PWM signals control when the stator coils are energized and the order they are energized in. For instance, phase A+ is energized then B+ then A-. Below is an example of a waveform that can control the motor. The speed the motor rotates is dependent on the frequency of the control waveform. Thus, we can carefully control the speed of the motor to minimize disturbance to the yogurt.

Waveforms that can Drive a Stepper Motor



Requirement	Verification
• +5V positive supply needs to be given to this pin for powering the module, operating voltage: 3.3V to 6V	• We can use voltmeter to check whether the voltage is in the permissible range
• Serial data is transmitted/received by module through the pin TXD and RXD at 9600 bps by default, 3.3V voltage	• Same idea, we can use voltmeter to check the voltage • For the bits per second,we could specify the bit transfer rate from each data sheet of several sensors, and the bps is changeable for this module.
• Operating temperature: -20C to 55C	• Since we are operating the module at room temperature, we should not have to worry about temperature tolerance.
• When operating normally, it should transfer the data from the microcontroller received by sensors to mobile devices.	• We would download a simple bluetooth terminal on the mobile devices to check if the data transfer is functioning. All data transferred should be displayed on the terminal including bits.

- **Mobile Interface Subsystem**
  - The mobile application must display temperature, pH, and viscosity information and be updated at least once every 10 minutes.
  - The app must alert users within 5 minutes if any temperature or pH reading deviates from the acceptable ranges via a push notification.
  - The user interface should be easily understandable to even those without a technical background.

Requirement	Verification
• The mobile application must display all metrics/plots (temperature, pH, and viscosity etc) and update the info at least once every 10 minutes.	• Use implemented timestamp log to check history of how often the updates took place and verify the minimum once per 10 minutes condition is met.
• User alerts must be sent within 5 minutes for any sensor data deviating from preset ranges	• Simulate out of range testing by temporarily setting the acceptable ranges to > 0 which would trigger alerts for each sensor • Use a timer to record the amount of time taken to get a push notification alert
• App should work on Android device with Bluetooth capabilities	• Take the Android device team has used for project and verify that all functionality is working

2.D Tolerance Analysis

The apparent viscosity of a yogurt varies from 1.8 to 3.8cp, and based on the fact that the container of yogurt would simply be a standard cylinder with radius of $r1$, and the radius of the stirrer would be $r2$. As we are using a simple model, the situation would be: a smaller cylinder stirrer with a larger cylinder container filled with yogurt.

- Radius of stirrer : $r2$.
- Radius of cylinder container: $r1$.
- Angular velocity of stirrer connected to a rotor: ω.
- Viscosity of yogurt: η

After defining all parameters that we need for calculations, we can use the formula in fluid dynamics to calculate the torque that we need to stir the yogurt.

$$\tau = \frac{2\pi * \eta * L * r_2^3 * \omega}{r_1 - r_2}$$

For a normal DC motor, the RMP is around 1000 to 5000, which means the angular velocity is around 104.72 to 523.60 rad/s, with the average number of 314.16 rad/s.

The next step is calculating the work done for the rod paddle to go for a full circle, i.e. 360 degree.

$$W = \tau * \theta$$

, where W stands for the work done, τ is the torque, and θ is the degree that the rod is spinning. Also, for a normal DC motor, the resistance would be relatively small, around 10 to 100 ohms, since it allows more current to flow through. And we know the equation that:

$$W = \frac{U^2}{R}, U = \sqrt{W * R}.$$

Since this is the tolerance analysis, we will obtain the maximum voltage that the motor needs to spin a full cycle in the fluid, with the maximum approximation of other parameters such as the radius of the rod and container.

$$W = 2\pi * \frac{2\pi * 0.0038 * 0.1 * 0.01^3 * 523.6}{0.06 - 0.01} = 1.57e - 4 \text{ Joules}$$

And for the voltage, we get:

$$U = \sqrt{W * 100ohms} = 0.016 \text{ V for each full circle}$$

Therefore, the 3.3V motor should fulfill our purpose well without any malfunctioning in voltage.

For temperature tolerance, typical yogurt-making temperature is around 105 to 112 F, so any type of stirring rod , sensors, and glass cylinder container would be able to tolerate this temperature.

pH Probe

- Probe Type: Laboratory Grade
- Detection Range: 0~14
- Temperature Range: 5~60°C
- Zero Point: 7±0.5
- Response Time: <2min
- Internal Resistance: <250MΩ
- Probe Life: >0.5 years (depending on the frequency of use)
- Cable Length: 100cm

3 Cost and Schedule

3.A Cost Analysis

The total cost for the parts in the table below is $59.51 and then we will add on a 5% charge for shipping and 10% charge for IL state tax bringing the total part purchase cost to $68.73. For labor, we expect a salary of $40.00 per hour and a total of 30 hours as a team will be spent per week for 8 weeks. The total cost of labor comes out to $9,600, tax not included. Adding the purchase costs and labor, we have a total cost of $9,668.73 for the completion of this project.

Description	Manufacturer	Quantity	Price	Link
DFRobot Gravity Analog pH Sensor	DFRobot	1	$39.50	Link
DS18B20 Digital Thermometer	Analog	1	$6.93	Link
ESP32-S3-WROOM-1	Espressif System	1	$3.48	Link
Small Reduction Stepper Motor - 5VDC 32-Step 1/16 Gearing	Adafruit	1	$4.95	Link
LED	SunLED	1	$0.36	Link
USB power cable	EAHSUCC	1	$4.29	Link
Packs of Resistors	ECE 445 Lab	1	$0.00	ECE445 Lab supplies
Packs of Capacitors	ECE 445 Lab	1	$0.00	ECE445 Lab supplies
Button	ECE 445 Lab	1	$0.00	ECE445 Lab supplies
Glass Cylinder Container	ECE Machine Shop	1	$0.00	ECE Machine Shop
Spin Rod	ECE Machine Shop	1	$0.00	ECE Machine Shop
Enclosure	ECE Machine Shop	1	$0.00	ECE Machine Shop

3.B Schedule

	Goals	Roles for each Team Member
Week of 10/7	● Design Review with Instructor and TAs ● Review and order parts ● Begin Designing PCB	● Betty ○ Presentation Prep ○ Begin designing PCB in KiCad ● Chidambara ○ Presentation prep ○ Research good PCB design skills on YouTube ● Alex ○ start learning PCB and KiCAD.
Week of 10/14	● Discuss and finalize design with machine shop ● Continue working on PCB design ● PCB Order 10/14 & Pass Audit ● Research communication protocols: ● Successfully power and communicate with microcontroller using esp32 dev board	● Betty ○ Finish and team review KiCAD design ○ Test power subsystems and communication with microcontroller using development board ● Chidambara ○ Work with team on machine shop request ○ Start testing code on microcontroller ○ Research on building an Android app and sending sample data over Bluetooth ● Alex ○ learning communication protocols ○ testing bluetooth module. ○ Research on software development on Android.
Week of 10/21	● Assemble sensors and work on getting data from pH and temperature sensors. ● PCB Order 10/21 Deadline	● Betty ○ Assemble sensors on development board and ensure sensors working correctly ○ Test and debug dev board set up ○ Research motor control software

	• Research and work on motor control software.
	• Work on microcontroller to app bluetooth communication
	• Begin work on the mobile app
		• Chidambara
		  ○ Replace sample data with real sensor data and verify proper operation
		  ○ Start working on basic app layout
		• Alex
		  ○ testing data transmission among sensors, microcontroller and bluetooth.
		  ○ start programming the bluetooth module.

Week of 10/28	• Work on RPM sensing
	• Work and test control software
	• Continue on the mobile app.
		• Betty
		  ○ Try to get fully functional sensor on dev board
		  ○ Check if the microcontroller works on the PCB. If not, revise for 11/11 PCB order deadline
		• Chidambara
		  ○ Build plots on app using sensor data
		  ○ Research Android push notification processes for critical alerts
		• Alex
		  ○ Start calculating and debugging on sensors' feedback.
		  ○ Summarizing the accuracy and uncertainty.

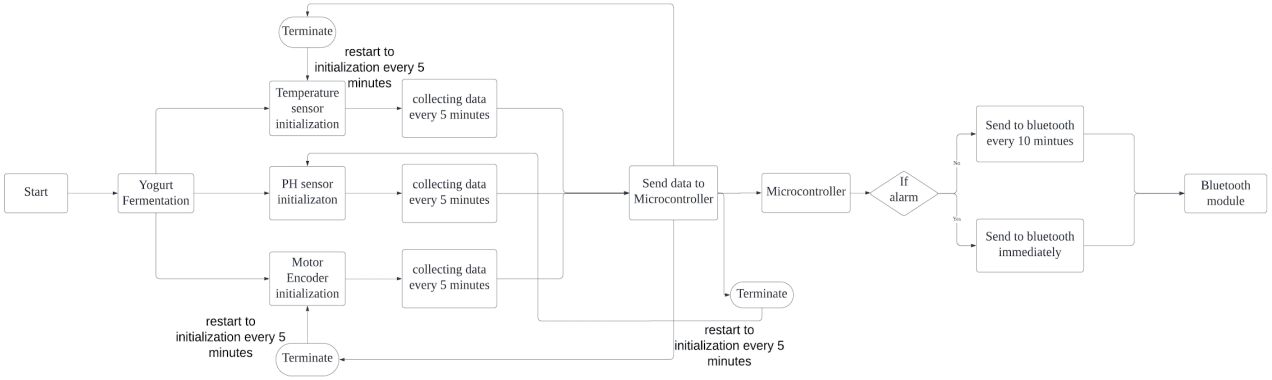
Week of 11/04	• PCB deadline.
	• Assemble and write software for control subsystem
	• Debugging
		• Betty
		  ○ PCB assembly and soldering
		  ○ Work on motor control hardware
		• Chidambara
		  ○ Write code for motor and debug
		  ○ Debugging app
		• Alex
		  ○ Moving components to personal-designed PCB and test Software side.

Week of 11/11	• Final: 11/11 Order Deadline and pass Audit (last minute PCB	• Betty
		  ○ Testing requirements and verification
		  ○ Debugging

	order) ● Control system software testing and debugging ● PCB Assembly and Soldering ● Debugging	● Chidambara ○ Verifying proper viscosity calculation using motor settings ○ Help with soldering ● Alex ○ Start soldering components onto the PCB based on the schematics.
Week of 11/18	● PCB and system Debugging ● Mock demo with TA	● Betty ○ Debugging ○ Present to TA ● Chidambara ○ Debug ○ Present to TA and incorporate feedback given ● Alex ○ Debug
Week of 11/25	● Fall Break	
Week of 12/02	● Fix Minor Existing Bugs ● Final Demo ● Work on final presentation ● Work on final paper	● Betty ○ Final Presentation Preparation ○ Debugging ● Chidambara ○ Debugging ○ Final Presentation Prep ● Alex ○ Final Presentation Preparation ○ Debugging

Week of 12/09	• Final Presentation
	• Work on final paper
		• Betty
		  ○ Work on Final Paper
		• Chidambara
		  ○ Final presentation prep
		  ○ Final Paper contributions
		• Alex
		  ○ Final presentation and paper

4 Flow Chart for Software



5 Ethics and Safety

As engineers developing a product, we must consider any ethical and safety issues that may arise because of our project. The IEEE Code of Ethics states that we must “hold paramount the safety, health, and welfare of the public” [1]. Our biggest concerns with this project would be health and safety related since our project involves food.

Food contamination is a major safety concern with our project. When possible we will avoid direct contact with the yogurt. However, some sensors such as the pH and texture sensor will need to be directly touching the yogurt in order to function properly. Thus, any materials used in the exterior of these sensors must contain non-toxic materials. In addition, the sensors and any part of our device that gets exposed to the yogurt needs to be able to withstand a cleaning and sanitation process. There are also some electrical safety concerns regarding the sensors and wiring that are directly exposed to the yogurt. These components must be able to withstand external temperatures of 112 F and be insulated from liquid to protect the user. We will refer to the NSF/ANSI 2 standards that cover proper sanitation and safety protocols for all equipment that comes into contact with food [2].

The IEEE Code also lays out ethical and professional guidelines we should act on as engineers. The code states that we must, “uphold the highest standards of integrity, responsible behavior, and ethical conduct in professional activities” [1]. As such, we will treat each other with respect, be non-discriminatory, and avoid injuring one another. We also agree to act with integrity by crediting any work we reference and avoid plagiarism.

5 Citations

[1] Institute of Electrical and Electronics Engineers. “IEEE Code of Ethics.” ieee.org. https://www.ieee.org/about/corporate/governance/p7-8.html (accessed Sep. 19, 2024).

[2] American National Standards Institute. “NSF/ANSI 2-2022: Food Handling Equipment.” ansi.org. https://blog.ansi.org/nsf-ansi-2-2022-food-equipment-standard/ (accessed Sep. 19, 2024).

[3] Espressif Systems. “ESP32-S3-WROOM-1 Datasheet” espressif.com. https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf (accessed Sep 19, 2024).

[4] Mouser Electronics. “Gravity Analog pH Sensor”, https://www.mouser.com/catalog/additional/DFRobot_ProductOverview_Gravity_Analog_pH_Sensor.pdf (accessed Sep. 19, 2024).

[5] Analog Devices. “DS18B20 - Programmable Resolution 1-Wire Digital Thermometer”, https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf (accessed Sep. 19, 2024).

[6] B. N. Bits, “Adding Bluetooth to Your Arduino Project with an HC-05 or HC-06 Bluetooth Module,” YouTube, https://www.youtube.com/watch?v=NXlyo0goBrU (accessed Sep. 19, 2024).

[7] “MIT App Inventor,” Massachusetts Institute of Technology, https://appinventor.mit.edu/ (accessed Sep. 19, 2024).

[8] Dejan, “How Rotary Encoder Works and How To Use It with Arduino,” *HowToMechatronics*, Jul. 25, 2016. https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/