Bread Starter Monitor - Report

CASA0016 - Making, Designing & Building Connected Sensor Systems

Patrick Whyte

2021

UCL - Connected Environments MSc

Introduction

This project's aim was to design a connected sensor system that senses a chosen environment. I chose to create a device to sense a particularly specific environment, one that is accustomed to constant changes while also requiring stable climate conditions.

The device developed for this project is a sourdough bread starter monitor. The device collects data on vital environmental conditions for bread starter health (temperature, humidity and CO_2 levels) and presents this information clearly to the user. This report aims to walk through the design, development, experimentation and deployment processes for the bread starter monitor device. I intend to explain what I have learnt, and reflect on what can be done to improve any future iterations.

Design

Inspiration:

During the pandemic, like many others, I started a new hobby of baking my own sourdough bread. In order to bake sourdough bread, you must first create a sourdough starter. Sourdough starter is essentially a colony of active yeast and good bacteria that is cultivated by fermenting a combination of flour and water. The yeast and bacteria feed on the carbohydrates present in the flour to produce lactic and acetic acid (which provides great flavour and nutritional value to the bread) and expel carbon dioxide as a result. *[1]*

Sourdough starter must be constantly fed with new water and flour, and is very sensitive to changes to its environment. When not kept in optimal conditions, it can easily die out. As the yeast and bacteria reproduce, the starter grows in size. Once it has grown large enough, a portion of the starter can be removed and be used to bake bread. When kept in the right conditions, sourdough starters can remain alive for centuries. This is why I decided to create the bread starter monitor to help me keep the starter in its optimal conditions and clearly visualise its activity to know when it needs to be fed.

Existing Solutions:

In order to get ideas on how I wanted to develop this project, I did some research to find similar devices that aim to monitor sourdough starter. Breadwinner is one such device: it tracks the starter using an infrared sensor to monitor how much it has risen while also

measuring the temperature. This data is then visualised for the user in an app on their phone. *[2]*

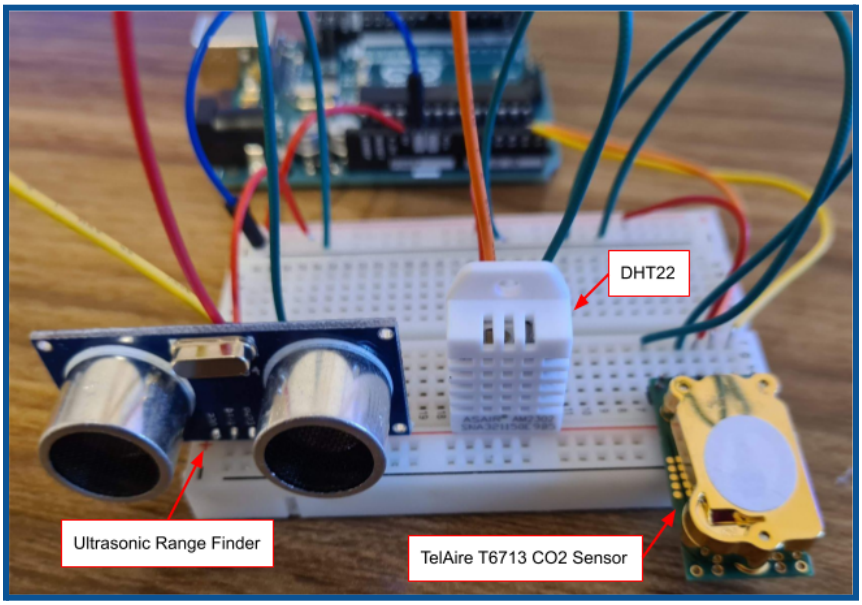
Defining The Problem:

The main external factors that can impact the health of sourdough starter are temperature and humidity. The temperature should be kept between 22-28 degrees Celsius, while humidity should remain between 60-80%.*[4]* In order to know when to feed the starter, we must keep track of its activity. By monitoring the CO_2 levels given off by the starter, we can have a good idea of how active the starter is. Lower levels of CO_2 indicates less activity which means the sourdough most likely needs to be given more water and flour. It is also important to keep track of how much the sourdough starter has grown in order to know when to cultivate it and ensure it does not get out of control.

Development

The Sensors:

The figure below shows the initial prototype used to test the sensors for the device.



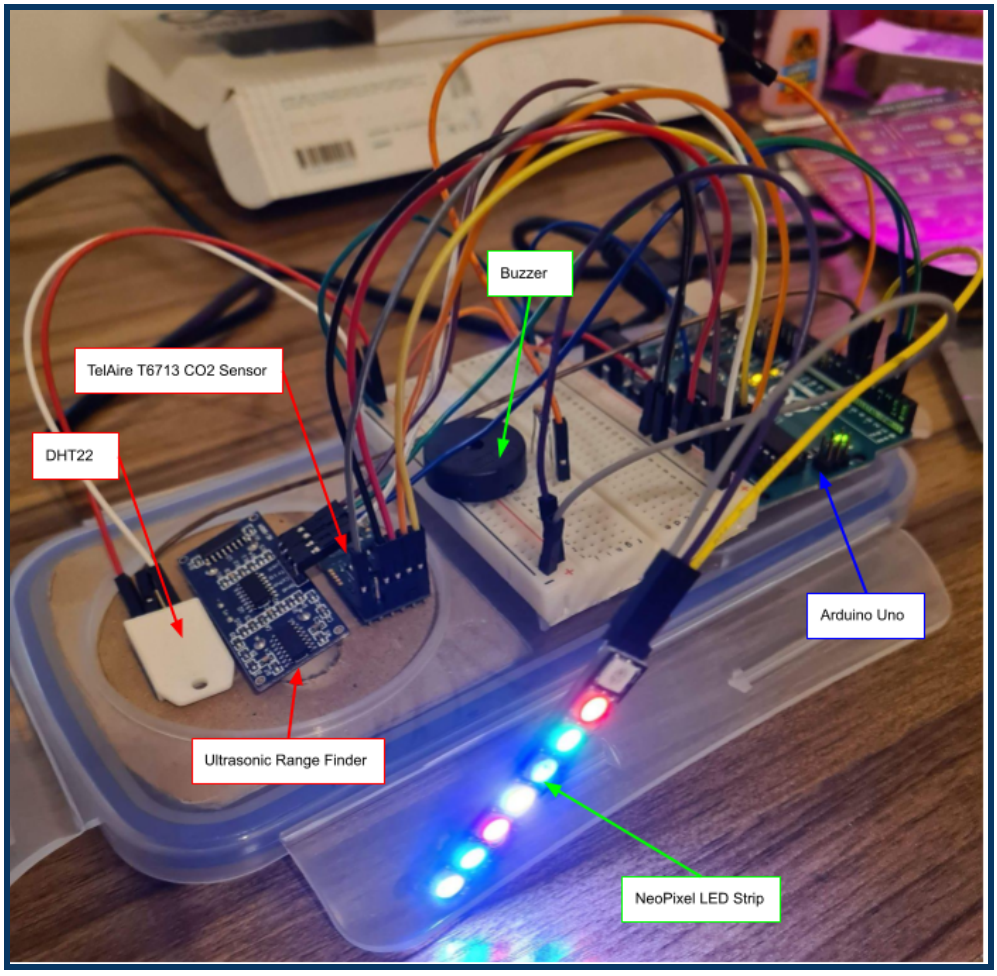
- To measure the temperature and humidity levels, I used the DHT22 sensor.
- To monitor the carbon dioxide levels, I used the TelAire T6713 CO_2 Sensor.
- An ultrasonic range finder was used to monitor how much the starter has risen.

The Actuators:

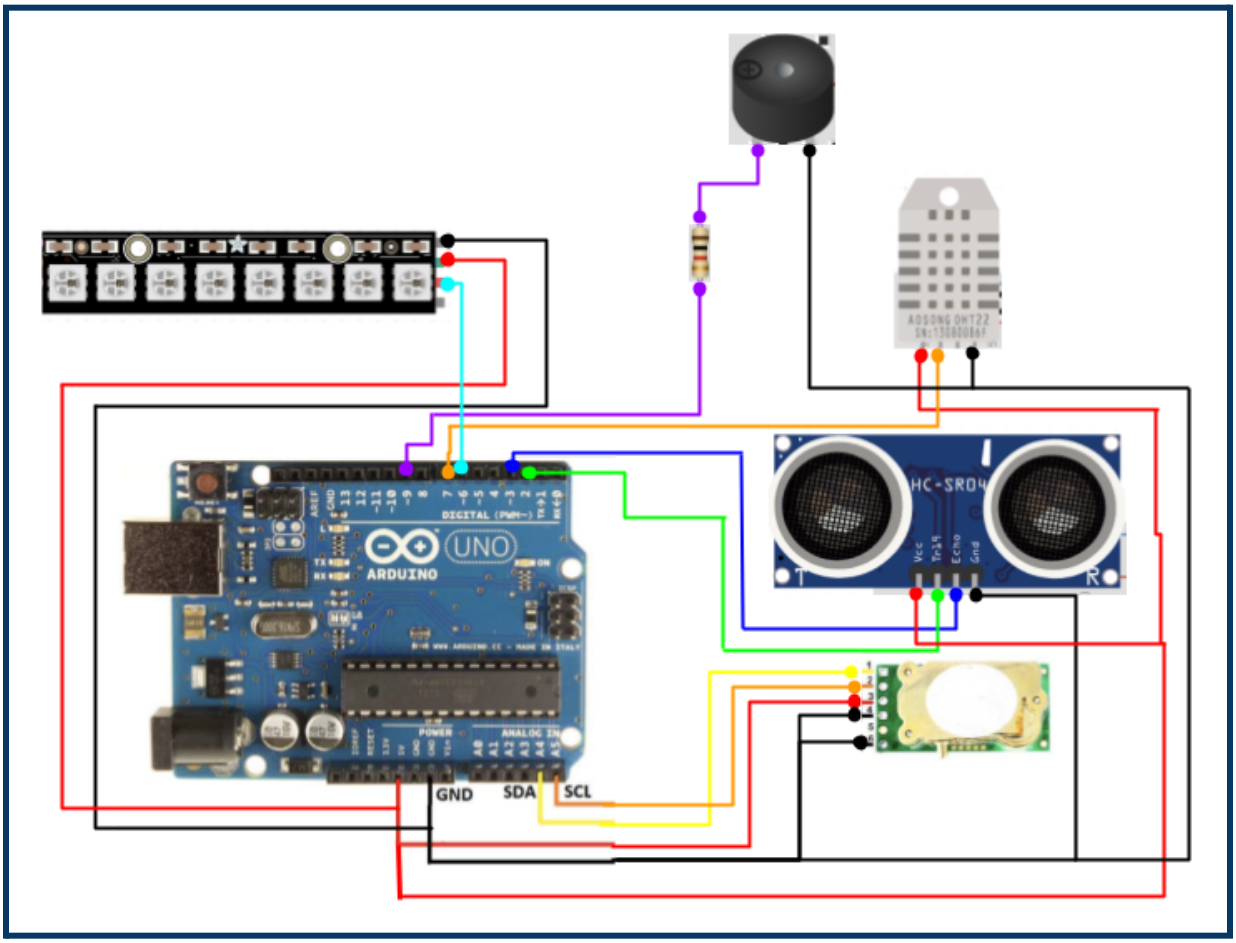
In order for the data gathered by the sensors to be useful to the user, it had to be displayed in a simple and clear way. I wanted to create a passive display that could present the relevant information at a glance without the user having to use an app. To do this, I used a NeoPixel LED strip by adafruit to display temperature, humidity and CO_2 activity. A buzzer was used to alert the user when the starter has risen to an optimal height to be cultivated and warn the user if it has risen too much.

Creating the Prototype:

In order to control the sensors and actuators, I used an Arduino Uno microcontroller. The figure below shows the first functional prototype with all the sensors and actuators wired to the Arduino Uno.



Circuit Diagram:



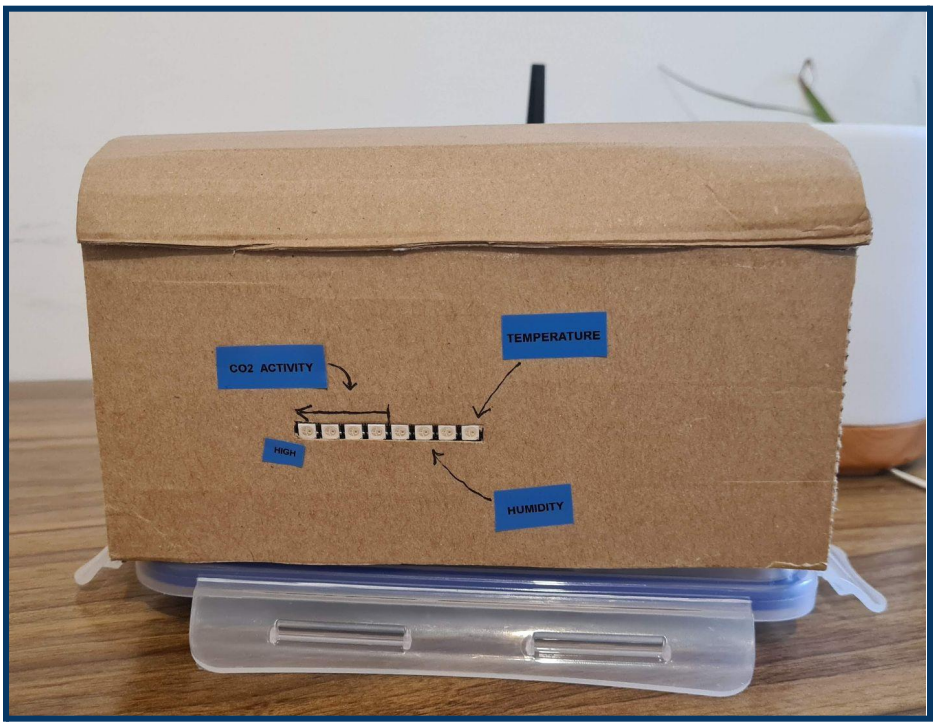
Creating the Enclosure:

In order to protect the circuitry and provide a housing for the LEDs to be displayed, I created an enclosure for the device that sits on top of the box containing the sourdough starter. The enclosure was fashioned out of cardboard and was shaped into the form of a loaf of bread to add some character to the device. The bread shape of the enclosure was specifically designed to provide enough space for the wires and electronics kept inside. Cardboard was chosen as a fast medium to create the enclosure that could be moved around and manipulated easily.

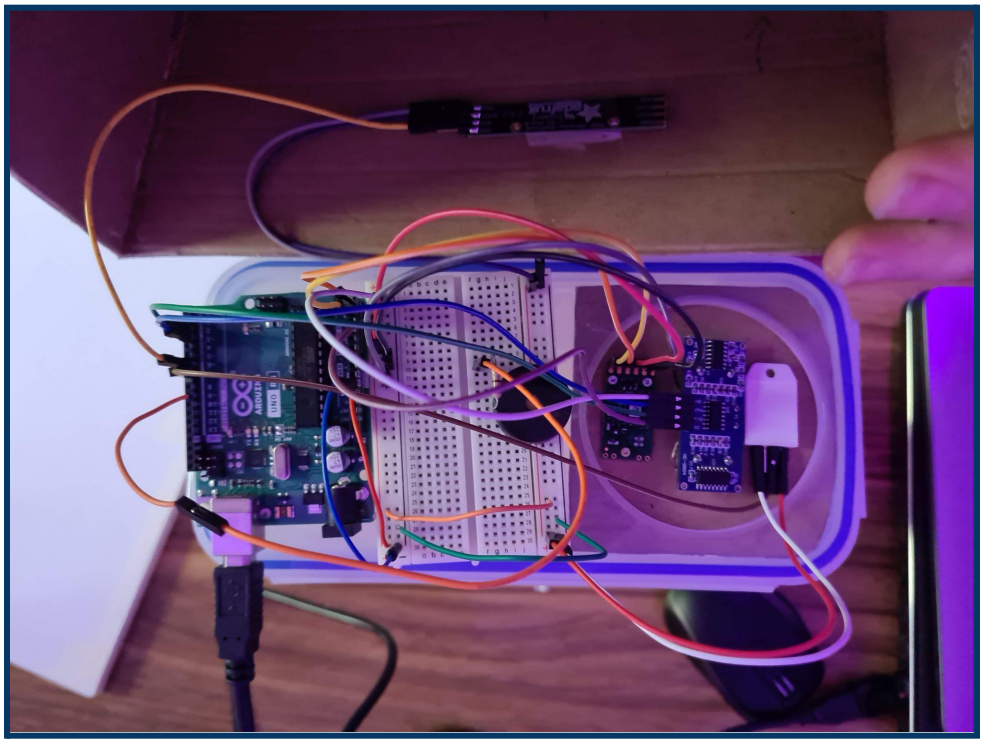
The figures below present the enclosure and how it was made.

Holes were precisely cut out of cardboard to allow the sensors to poke into the box in order to take readings and be properly secured in place.

Small details were added to make the enclosure as similar to a loaf of bread as possible.



A label maker was used to create clear legible labels for the side display.



Tape was used to secure the NeoPixel LED strip to the inside of the enclosure.

The Code:

The program code for this project was developed in C++ using the Arduino sketch IDE. The following libraries are required for the project.

- Wire.h
- DHT.h and DHT_U.h
- Adafruit_NeoPixel.h

Setup:

```
void setup() {
  Wire.begin ();

  // Start serial connection for debugging
  Serial.begin(9600);

  // start DHT sensor
  pinMode(DHTPin, INPUT);
  dht.begin();

  // Set up range finder
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input

  // Set buzzer - pin 9 as an output
  pinMode(buzzerPin, OUTPUT);

  // start NeoPixel LEDs
  pixels.begin();

  // Get inital CO2 reading for base value and level values for range
  baseCO2Value = readCO2();

  // Tweak this range so it makes sense
  CO2Level1 = baseCO2Value + 500;
  CO2Level2 = baseCO2Value + 1000;
  CO2Level3 = baseCO2Value + 3000;
  CO2Level4 = baseCO2Value + 5000;

  Serial.println("Bread Starter Monitor");
}
```

Once the libraries have been defined and the global variables are established, the setup method essentially ensures that each of the sensors are activated, and defines their input and output pins. A serial connection is established mainly used for debugging, providing real time monitoring of the sensor data. Finally, an initial CO_2 reading is made to establish a base value for the CO_2 in normal condition. This is required as the initial CO_2 reading can widely vary depending on the environment the sourdough starter monitor is in. The base CO_2 value is also used to calibrate the CO_2 activity threshold levels which will be used later on for the display. The CO_2 activity levels were selected after experimentation. However, these values may need to be changed depending on the size of the container used.

Main Loop:

```
void loop() {

  Serial.print("Base CO2 Value: ");
  Serial.println(baseCO2Value);

  // Get new readings from sensors
  readCO2();
  readTemperature();
  readHumidity();
  readDistance();

  // Update the LEDs using the new values collected from the sensors
  updateTemperatureLED();
  updateHumidityLED();
  updateCO2LED();
  pixels.show();

  // Check the starter has not risen too high
  checkDistance();

  delay(500);
}
```

The main loop gathers the data from each of the sensors then updates the LEDs to reflect these changes and/or activates the buzzer if needed. This process repeats every 500 milliseconds.

Reading CO_2 Value:

```
int readCO2()
{
  // start I2C
  Wire.beginTransmission(ADDR_6713);
  Wire.write(0x04); Wire.write(0x13); Wire.write(0x8B); Wire.write(0x00); Wire.write(0x01);
  // end transmission
  Wire.endTransmission();
  // read report of current gas measurement in ppm
  delay(2000);
  Wire.requestFrom(ADDR_6713, 4);    // request 4 bytes from slave device
  data[0] = Wire.read();
  data[1] = Wire.read();
  data[2] = Wire.read();
  data[3] = Wire.read();
    Serial.print("Func code: "); Serial.print(data[0],HEX);
    Serial.print(" byte count: "); Serial.println(data[1],HEX);
    Serial.print("MSB: 0x");  Serial.print(data[2],HEX); Serial.print("  ");
    Serial.print("LSB: 0x");  Serial.print(data[3],HEX); Serial.print("  ");
  CO2Value = ((data[2] * 0xFF ) + data[3]);
  Serial.print("CO2 Value: ");
  Serial.println(CO2Value);
  return CO2Value;
}
```

This function comes from the sample source code for the TelAire T6713 CO_2 Sensor found in documentation for the sensor. It communicates with the Arduino over the i2c protocol, and is used to collect a CO_2 reading in parts per million (ppm). The documentation and original sample source code can be found under reference 3. *[3]*

Temperature and Humidity:

Once the temperature and humidity values have been taken by the DHT22 sensor, they are used to update the LEDs on the display.

```
void updateTemperatureLED()
{
  if (Temperature < 22) {
    // Too cold - Set LED to Cyan
    pixels.setPixelColor(tempLED, 0, 37, 37);
  } else if (Temperature > 28) {
    // Too hot - Set LED to Orange
    pixels.setPixelColor(tempLED, 45, 20, 0);
  } else {
    // Temperature is in optimal range - Set LED to Green
    pixels.setPixelColor(tempLED, 0, 37, 0);
  }
}

// This may need tweaking too
void updateHumidityLED()
{
  if (Humidity > 80) {
    // Too humid - Set LED to Cyan
    pixels.setPixelColor(humLED, 0, 37, 37);
  } else if (Humidity < 60) {
    // Too dry - Set LED to Orange
    pixels.setPixelColor(humLED, 45, 20, 0);
  } else {
    // Humidity is in optimal range - Set LED to Green
    pixels.setPixelColor(humLED, 0, 37, 0);
  }
}
```

If the temperature is too cold or the humidity is too high, their respective LEDs will turn cyan. If the temperature is too hot or the humidity is too dry, their respective LEDs will turn orange. When these values are in their optimal range the LEDs turn green.

CO_2 Activity Range:

```
void updateCO2LED()
{
  // Reset the range
  pixels.setPixelColor(CO2LED1, 0, 0, 0);
  pixels.setPixelColor(CO2LED2, 0, 0, 0);
  pixels.setPixelColor(CO2LED3, 0, 0, 0);
  pixels.setPixelColor(CO2LED4, 0, 0, 0);

  if (CO2Value <= baseCO2Value) {
    // Not active - Set LED to Yellow
    pixels.setPixelColor(CO2LED1, 37, 37, 0);
  } else if (CO2Value > baseCO2Value && CO2Value < CO2Level1) {
    // Mildly active - Set LED to Orange
    pixels.setPixelColor(CO2LED1, 45, 20, 0);
  }

  // If the current C02 level is greater or equal to the threshold CO2 value
  // for a level we set that LED to Red
  if (CO2Value >= CO2Level1) {
     pixels.setPixelColor(CO2LED1, 37, 0, 0);
  }
  if (CO2Value >= CO2Level2) {
     pixels.setPixelColor(CO2LED2, 37, 0, 0);
  }
  if (CO2Value >= CO2Level3) {
     pixels.setPixelColor(CO2LED3, 37, 0, 0);
  }
  if (CO2Value >= CO2Level4) {
     pixels.setPixelColor(CO2LED4, 37, 0, 0);
  }
}
```

The CO_2 activity is displayed over 4 LEDs as a range from least to most active. Each iteration of the loop, after collecting the new value from the CO_2 sensor, the LEDs are reset. Then, if no change is seen from the base CO_2 value, the first LED shows up as yellow to indicate no activity. If the sensor picks up a small increase in CO_2, that LED turns orange. When the CO_2 value surpasses the different threshold levels established in the setup method, the four LEDs light up red one after another. By doing this, it shows up as a scale from least to most active that the user can clearly comprehend with a glance.

Range Finder and Buzzer:

```
void checkDistance(){

  if (Distance <= 6){
    //Emergency about to overflow - Sound alarm!
    for (int i = 0; i <= 10; i++) {
      tone(buzzerPin, 900);
      delay(50);
      noTone(buzzerPin); // Stop sound
      delay(50);
    }
  } else if (Distance <= 10){
    // Getting close - Warning sound; two beebs
    tone(buzzerPin, 700);
    delay(100);
    noTone(buzzerPin); // Stop sound
    delay(100);
    tone(buzzerPin, 800);
    delay(100);
    noTone(buzzerPin); // Stop sound
  }
}
```

The ultrasonic range finder measures the distance from the top of the box to where the starter has currently risen to. If the starter gets within 10cm of the top of the box, the buzzer activates by sounding a warning sound of two beeps increasing in pitch to indicate that the starter should be cultivated. If the starter continues to rise past this and gets within 6cm of the sensor then an alarm is activated, causing the buzzer to make 10 rapid high pitch beeps each iteration of the loop. This is to get the attention of the user and ensure that the sensors do not come into contact with the starter, which could potentially ruin the device.

Experimentation

This device proved to be very difficult to experiment. One problem discovered through experimentation was the CO_2 activity not decreasing over time after not feeding the starter. I expected the CO_2 range to follow a bell curve that initially, after feeding it, would increase. As the carbohydrates are used up and the yeast and bacteria become less active the CO_2 range would fall back down from the peak and return to a lower value. This was not the case during my initial experiment. This was happening because the container used for the device is essentially air tight. Thus, there was nowhere for the CO_2

gas to expel, causing the values to be constantly high even when the starter was no longer active. In order to work around this problem, I had to change how I interacted with the device. The first modification I made was to increase the final CO_2 threshold value to 5000ppm above the base CO_2 value. Now when the CO_2 range is full, it indicates that the box has become completely saturated with CO_2, meaning that the starter has converted most if not all the carbohydrates into gas. When it reaches this level, I know to feed the starter by removing the lid as normal, but also to give it enough time to expel the gas stored in the box before putting the lid back on. This way when the lid is put back on the cycle resets and I can continue to monitor the starter.

The initial threshold CO_2 values for the CO_2 activity range went through many changes before I managed to find values that represented the activity of the sourdough reliably. To find these values, I performed multiple experiments and would start with a fresh new batch of sourdough starter each time. For each test, I made sure to use the same volume of water and flour and would take note of the CO_2 values around the times it needed to be fed or cultivated. Finally, once these kinks were worked out I could start monitoring a fresh new batch of sourdough starter. The figure below shows the device monitoring my sourdough starter.

The device worked as expected going through multiple feeding cycles before I cultivated some of it. I used that starter to bake a loaf of sourdough bread which I thoroughly enjoyed.

Reflection

After finally going through the whole process from starter to loaf, I have identified several areas where the device could be improved. Firstly, given that the device required a wired connection to my laptop to work, it made the device very cumbersome to use for extended periods of time. Ideally, I would like to make a smaller overall device that could be powered with its own battery and fit onto a standard container such as a mason jar. I feel this project would greatly benefit from more time dedicated to gathering data over longer periods of time. This data could be used to create more general CO_2 threshold values and a more optimal rising size. This data could also be visualised as a graph allowing the user to compare different techniques and starting conditions. The enclosure could be improved upon, potentially utilising a 3D printer to create a more visually recognisable loaf of bread with a clearer display for the LEDs.

One concern that has to be considered with a project involving food is its safety. If this device were to be developed further, more research must go into the effect the sensors and equipment may have on the starter. It should also be considered whether the solution developed during this project is over engineered. Perhaps the best users for this device are bread baking beginners that would benefit from a device guiding them on the process of taking care of bread starter.

An unexpected positive outcome that came from the device was the enjoyment I had from occasionally glancing at the display. Watching it slowly become more active over time, and taking care of it when it would get too cold or dry made the experience feel as if I was truly taking care of something alive, something I forget when I usually keep a bread starter.

In conclusion, although the device had limited utility in more real world scenarios, it showed great potential for future iterations. This project has given me a much deeper understanding of how to design and build a connected sensor system. I feel I have learnt some valuable lessons from my mistakes and widened my skills in circuitry that will no doubt help me with my future projects.

References:

1. https://www.breadandbasil.nyc/sourdough/sourdough-starter
2. https://www.theverge.com/2021/3/19/22340817/breadwinner-smart-sourdough-starter-tracking-wifi-gadget-bread-yeast
3. http://co2meters.com/Documentation/AppNotes/AN161-T6713-arduino-i2c.pdf
4. https://truesourdough.com/best-temperature-for-proofing-sourdough-full-guide-how-to/
5. https://www.mouser.co.uk/ProductDetail/Amphenol-Advanced-Sensors/T6713-5k?qs=N10xTRV9dj2DLVnP5YzNbg%3D%3D
6. https://www.mouser.co.uk/datasheet/2/18/AAS-920-634F-Telaire-T6713-Series-100417-web-1315857.pdf
7. https://www.mouser.com/catalog/additional/Amphenol_CO2-SingleDual-Channel-022216-web.pdf
8. https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf
9. https://www.dovesfarm.co.uk/hints-tips/bread-making/guide-to-sourdough-making
10. https://www.arduino.cc/reference/en/
11. https://forum.arduino.cc/t/maximize-buzzer-loudness/132861

Link to the GitHub Repository:

https://github.com/wezpez/bread-starter-monitor