GNG 1103
Design Project User and Product Manual

Climate Sensor Module

Submitted by:

Sensor Society, Group 10

Benjamin Hotte, 300207427

Evan Lacroix, 300194699

Alison Kamikazi, 300073548

Supathira Uthayakumar, 300167100

Gabriel Krausert, 300213672

April 11, 2021

University of Ottawa

Abstract

This document is a user product manual that describes how to operate Sensor Society's final version of the climate sensor module and includes pertinent information for future developers to continue to improve the product. The climate sensor module is a sensor system that has been created to monitor environmental conditions including temperature and humidity of a package onboard the JAMZ delivery drone to ensure the good standing of the package upon receipt by the customer.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Glossary

Table 1. Acronyms

Acronym	Definition
ABS	Acrylonitrile Butadiene Styrene
BOM	Bill of Materials
CAD	Computer Aided Design
COM	Communication
CSM	Climate sensor module
IDE	Integrated Development Environment
UPM	User and Product Manual
USB	Universal Serial Bus

Table 2. Glossary

Term	Definition
I2C	A serial communication protocol to allow multiple devices to communicate
Mux	Multiplexer
SCL	Serial clock line (carries the clock signal to the other components)
SDA	Serial Data line (line used to send and receive data between the master and slave)
Si7021	A temperature and humidity sensor

1.0 Introduction

This User and Product Manual (UPM) provides the necessary information for the client JAMZ and/or for other students to effectively use the Climate Sensor Module (CSM) and for prototype documentation. The information specified in this document will allow anyone to use, recreate or build on to our design for improvements if needed. This document supports the following objectives:

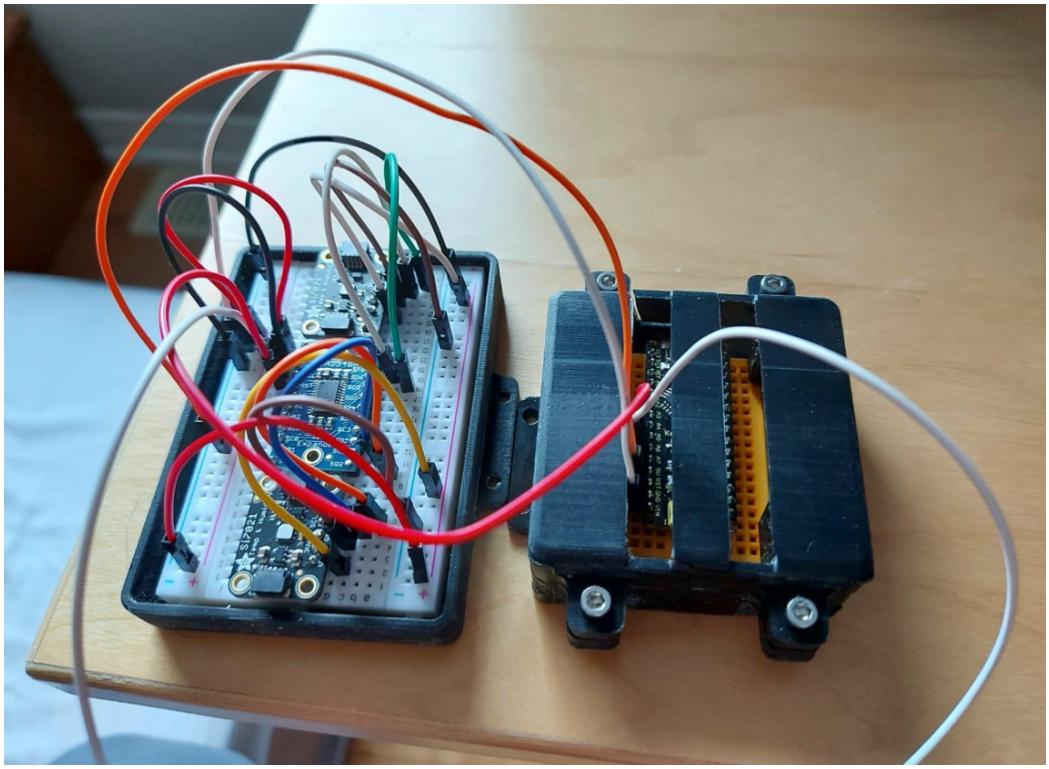
- Provide an overview of the design and features of the product
- Prototype development and analysis
- Troubleshooting and maintenance of the product
- Identify the required resources needed to reproduce the product
- Describe the methods and strategies used to test the product

2.0 Overview

JAMZ, our client, needed a system that is able to transfer reliable and accurate data on a package's climate condition from the drone to the drone operator. A safe, reliable and cost efficient climate sensor module was developed according to the client's requirements and a final prototype was built. From the problem statement "Continuous exchange of ***accurate data*** with the operator about the package's climate, ultimately determining the ***good standing*** of the buyer's package", a final prototype was developed which provided a solution to each design specification.

The ultimate objective is to determine the temperature of the package and if it is in good condition for the customer in order for the content to be consumed or not. To satisfy this requirement, sensor selection and sensor placement on the drone were key factors in the design. Based on our client's needs, the drone's operating condition was stated to be temperatures ranging from -15 to 40 °C and for the sensor to relay a ± 0.5 °C of accuracy when taking the package's temperature. The sensor needs to relay a package humidity between 50 and 55% RH as well. Our choice sensor, the Adafruit Si7021 Temperature + Humidity Sensor, is a highly reliable sensor with a great range of temperatures (-10 to +85 °C) and provides an accurate reading of ± 0.4 °C rather than ± 1 °C. It also has a ± 3% relative humidity measurements with a range of 0–80% RH. Another crucial part to our system is the use of two temperature sensors to average the temperature readings from the package to provide more accurate data and redundancy. This minimizes the errors that might occur in readings and provides more reliability.

Figure 1. Final climate sensor prototype inside its 3D printed enclosure



The final prototype relays a TRUE or FALSE statement to the operator based on a range of temperatures pre-set as the client does not need the temperature readings every second. If the sensors give a reading outside of the accepted ranges, the drone operator will receive information that the package may be compromised and not in good condition. Protective casings were also implemented into our design and were 3D printed to fit and shield our components. The design consists of two Si7021 temperature and humidity sensors attached to a multiplexer and an Arduino Nano, all assembled inside the 3D printed enclosure. The whole system is to be placed inside a closed box attached to the drone in order to get the best temperature reading of its content and ensure its condition.

2.1 Cautions & Warnings

The module also contains many small pieces, which is a choking hazard for young children. This being said, please keep the module away from children under the age of 16 and away from animals. If pieces ingested or module causes burns, please seek immediate medical help.

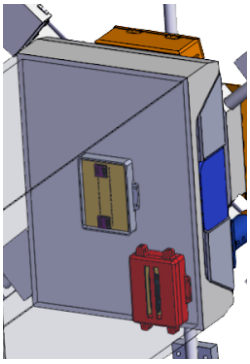
3.0 Getting started

In order to have to have the sensor system to be complete and fully functional, please read and follow the instructions below:

Step 1 (Installation of the System):

- Grab 8 M3 x 0.5mm screws in order to drill and fasten the sensors, as well as the arduino casing on the underside of the top of the box. (**Place the sensors anywhere on the bottom side of the box. Preferably centred at an equal length away from all sides of the box for optimal accuracy)**.
- Preferably install the arduino and it's casing in any corner on the top of the box. Making sure it is 1.5 inches away from the corners it is being placed in.
- See attached photo for visual clarification.

Figure 2. Proposed placement of climate module system for installation



Step 2 (Connecting the Arduino nano to the Raspberry pi):

- To connect the arduino nano to the onboard raspberry pi you will need a mini-USB to USB 2.0 cable. Plug the USB end into one of the Raspberry Pi 's USB 2.0 ports, and the mini USB end into the Arduino nano's mini-USB port.

Step 3 (Configuration of the sensors)

- To ensure that the system is working properly, put the sensors in a room temperature environment (20℃) to analyze ambient conditions.
- When accessing your raspberry pi IDE a statement reading **"Temperature is NOT in desired range"** will appear. The same statement will appear for humidity.

With the installation process complete, here is the process of how the system works when properly installed on the drone:

Step 1:

- The drone's power supply is turned on hence turning on our sensor system.

Step 2:

- The sensors will begin to take the temperature and humidity of the box or whatever environment they are placed in.

Step 3:

- An order is placed via JAMZ and the drone starts to take its route to deliver the package.

Step 4:

- The sensor data will travel from the arduino nano to the raspberry pi, where if the desired temperature of the client is outside of the range they wanted, an alert will appear letting the operator know that the package climate is out of the desired range.

Step 5:

- Depending on how many orders the drone has to deliver the sensors will always be running. When the sensor system is no longer needed it should just be disconnected from the drone's power supply manually or just turn off the drone completely. Doing either of these will completely shut the system down.

3.1 Set-up Considerations

To complete the installation and configuration process of the sensor system some external tools will be required to assemble it. See the attached list for guidance:

Table 3. Pre-requisite tools and equipment for installation of the climate sensor.

Tool/Device	**Purpose of Tool/Device**
x8 M3 x 0.5mm thread phillips head screws. **Figure 3.1** 	To fasten the sensors and the arduino casing to the drone.
x1 Power drill **Figure 3.2** 	To tighten the screws in place. **(Phillips head drill bit needed)**
x1 Mini usb to usb cable **Figure 3.3** 	Connects the arduino nano to the raspberry pi.

3.2 User Access Considerations

There are two users of this physical system; the client who is buying and installing the sensor system. There is also the drone operator who will be using our system. The drone operator should ideally have some experience and knowledge with code more specifically C++, in order to fully understand the data they are receiving and how our system works. The only restriction put in place is in the code. The operator will only get an alert/statement when the temperature has risen too high or dropped too low. Rather than having endless amounts of data displaying the temperature of the package every one second. The user does not need to interact much with the actual system, since it runs automatically. All the user needs to do is monitor the alerts our system is providing. Ultimately making this an easy job for the user/client.

3.3 Accessing the System

Since the arduino nano is connected to the power supply and the raspberry pi, once the drone is turned on, the sensor system will also be turned on automatically. In terms of accessing the data being produced from the sensors. The operator will need to open the onboard raspberry pi's IDE Once they have opened the IDE and the drone's power supply is turned on the sensors will begin to relay their information back to the drone operator. There is no user ID or login system put in place for our system. If JAMZ wanted to implement this function they could.

3.4 System Organization & Navigation

What is unique about our system is that it is very organized and easy to navigate. The sensor's data is automatically transmitted to the drone operator when needed with no intervention. All the drone operator has to do is access the Arduino IDE serial monitor to see all of the code which is displayed in a very organized way. The code also has comments on certain pieces of code to indicate what that function does. This makes it easy for the operator to understand what is happening when the sensors are on and running.

3.5 Exiting the System

To exit the system or turn it off, the arduino can either be disconnected from the drone's power supply or completely turn off the drone's power supply which will shut off all power going to the nano hence turning off the sensors. Once the nano and the sensors are shut off, the drone operator can just close the IDE window since there will be no data relaying to them anymore.

4.0 Using the System

The module's system comes as is ready for use, with temperature and humidity cutoff values obtained from testing with coffee in package simulations. As such, if the user desires only the basics and wants to know when their coffee is in good humidity and temperature conditions, the system does not need any changes whatsoever. However, if the user desires to use the module for other foods or a combination of foods, the users must utilize the below functions to alter the code and wiring for their needs. Additional testing with other foods may be required. The module comes with five premade functions to facilitate the use and alteration of the module. These functions are explained below.

4.1 Si7021 Defogger

This feature comes from the Adafruit Si7021 imported library, it allows the user to enable or disable the onboard Si7021 heater. This heater is used to defog the sensor's temperature reading by producing heat for a brief period of time to clear condensation buildup on the sensor. It is best used for when the humidity reaches values of 80%. However, this functionality requires the user to have a basic knowledge of C++ coding or rather Arduino IDE coding. From another perspective, the issue with activating the onboard heater function is that once the function is on, the temperature readings will fluctuate from three to five degrees Celsius until the function is turned off. There is a work around by implementing the heater function only on certain criterias and have them on a set amount of time, but this requires the user to have good knowledge of C++ code to do this 'workaround'.

```
bool enableHeater = false;
```

4.2 Lower and upper bounds (Threshold values)

The code's easy readability and basic code allows the user to easily change the temperature's and the humidity's upper and lower bounds depending on the desired ranges. For instance, if one was to test the good standing or coffee, while the other wanted to test ice cream, the bounds would definitely have to be altered to fit the right criterias. Also, changing the bounds does not require the user to have any knowledge of codes (hence it is just changing a number to another), however, it would be best that the user knows basic code for a smoother implementation. It is also crucial to note that temperature readings are in degrees Celsius, and humidity readings are in percent relative humidity.

4.2.1 Temperature

It is possible to alter the temperature bounds without altering the humidity range. To change the lower bound, simply change the number 25 from the code below to the desired lower bound, and to change the upper bound, simply change the number 30 from the code below to the desired temperature. Remember, if this statement is TRUE, the temperature is not in the desired range. For example, the code below signifies that the desired temperature is between 30 and 25.

```
if (averageTemp > 30 or averageTemp < 25){
  Serial.print("\t Temperature is NOT in desired range!\t");
  flag = false;
```

4.2.2 Humidity

It is also possible to alter the humidity range independently from the temperature. To change the lower bound and upper humidity bounds, simply change the 50 and 55 respectively from the code below to the desired brackets. Remark, the code returns a “NOT in the desired range” message if the humidity readings are outside these brackets, thus the statement is TRUE. For example, the code below shows a desired relative humidity of 50% to 55%, outside this range it sends the problem message.

```
if (averageHumidity > 55 or averageHumidity < 50){
   Serial.print("\n");
   Serial.print("\tHumidity is NOT in desired range!");
   flag = false;
```

4.3 2 Sensors

The greatest attribute to our code functionality is the use of 2 sensors. This allows a user with good knowledge of Arduino IDE coding to use both sensors to compare, average, and regresse read data. In other words, having 2 sensors onboard allows the user to verify the output data. This means that when one sensor breaks, thus reads bad data, the other sensor can compensate, giving the user two choices, either completely disregard the broken sensor until it can be checked, or average out the values to see if it is a temporary issue that could be caused by external factors (wind, overheating, sun). The code does not implement these fail safe criterias for the user. As such, the user must have a fair understanding of C++ coding. If the coding is not properly structured with if statements, the sensors may not read at all. As such, see below function, section 4.5, to see a fail-safe.

```
// Get humidity and temp from sensor 1
TCA9548A(2);
float humidity1 = sensor1.readHumidity();
float temp1 = sensor1.readTemperature();

// Get humidity and temp from sensor 2
TCA9548A(6);
float humidity2 = sensor2.readHumidity();
float temp2 = sensor2.readTemperature();

float averageTemp = (temp1 + temp2) / 2;
float averageHumidity = (humidity1 + humidity2)/2;
```

4.4 Mux

There is an 8x1 multiplexer on our module. As such, this allows the user to connect new sensors or other micro features depending on their desired use of the module. The 2 sensors take up 2 of the multiplexers input lines, leaving the user with 6 other lines to connect other material. Wiring to a multiplexer requires basic knowledge of electrical wiring and information transfer. It also requires the user to have basic knowledge of code, because altering or picking new mux bus numbers will have to be hand coded. For example, the below line of code is the mux choosing bus port 2. To facilitate mux coding, the wire.h library is imported in the nano's code. Remember that the microcontroller used in the module (arduino nano) uses 5v, thus connecting anything requiring much more or less voltage will lead to either frying the added material or simply not powering it.

```
TCA9548A(2);
```

4.5 Fail-Safe

As most of the features require the user to play around with the code, a fail-safe line has been added to prevent the lecture of null values with unfound sensors (see code paste below). This does not require the user to change anything. It is a simple fail safe from the Adafruit Library.

5.0 Troubleshooting & Support

In the case where our sensor system is not running properly here are a couple scenarios that may help the user troubleshoot and solve any errors and bugs in the system.

- Make sure that all the wires are connected properly and there are no loose wires.
- If there are any errors/bugs in the code, seek the multiple libraries we used to compile the code. See **6.1.3** for the different libraries used.
- If code is edited and you are unsure of which microcontroller to compile, unplug from source and plug back in to determine name of microcontroller.
- If further problems persist contact or reference arduino IDE's website (https://forum.arduino.cc/).
- If the raspberry pi is no longer receiving the data from the arduino, again make sure the USB to mini USB is not compromised or loose.

5.1 Error Messages or Behaviors

The code searches to initialize both sensors before the reading of any data. As such, if the wiring or the wrong bus adresse is given, the code will print: "Did not find Si7021 sensor #". If this code arrives, verify the wiring of the sensors to the mux, verify that all components share the same ground and relay power, a trick to do so is to connect the module to your computer and verify that each sensor's green light turns on, else the power is not being relayed properly. Another thing to verify is that the sensors are wired to the proper mux addresses and that the code reflects those addresses.

```
// Initialize sensor 1
 TCA9548A(2); //I2C bus number 0
 if (!sensor1.begin()) {
   Serial.println("Did not find Si7021 sensor 1!");
   while (true)
     ;
 }
 // Initialize sensor 2
 TCA9548A(6); //I2C bus number 1
 if (!sensor2.begin()) {
   Serial.println("Did not find Si7021 sensor 2!");
   while (true)
     ;
 }
```

The code also verifies the good standing of the package's humidity and temperature. For example, if the humidity or temperature are not in the desired ranges, the code prints the

statements : “Humidity is NOT in desired range1” and “Temperature is NOT in desired range”. If the ranges are not proper, see section 4.2 to learn how to change the bounds.

```
if (averageHumidity > 55 or averageHumidity < 50){
   Serial.print("\n");
   Serial.print("\tHumidity is NOT in desired range!");
   flag = false;
}
if (averageTemp > 30 or averageTemp < 25){
  Serial.print("\t Temperature is NOT in desired range!\t");
  flag = false;
```

5.2 Special Considerations

Mismatch of data: If the data from sensor 1 no longer matches the data from sensor 2 within your given tolerances, one or both of the sensors may be disregarded or averaged depending on the regression of the past data.

5.3 Maintenance

To keep the module up to date, it is crucial to do all Arduino IDE updates and library updates (Adafruit Si7021 library and the wire.h library). Another maintenance aspect is to activate both sensors minimally every three days and clear all dust build up before every use. It is also important to not overuse the module to not overheat the sensors or Nano board which may cause them to break.

5.4 Support

Troubleshooting errors can be done online on the Arduino website. It displays clear wiring between parts and code snippets. If the sensors or Arduino Nano starts to produce smoke, unplug the main power line to the module and step away from the module until the smoke has stopped. If possible, use heat gloves to take the module outside away from any flammable materials. Do not let any children use or manipulate the module as it may disperse a small shock or produce heat that could potentially lead to first degree burns. Please keep the module away from pets and stored in a safe cool environment.

6.0 Product Documentation

The temperature sensor subsystem is responsible for the collection of temperature and/or humidity data. Data output should be accurate and continuous.The temperature targeted for the sensor to read is between -20 to +40 ºC and among all the sensor concepts designed only the Adafruit, the Thin Film RTD and the BME280 sensors are the only ones to measure below zero temperatures. The Thin Film RTD provides a way larger than needed range and the Adafruit has ± 0.4 ºC accuracy which is better than the accuracy of the BME280 sensor at ± 1.0 ºC. Though the BME280 sensor also provides a good range of temperatures it can measure, -40 to +85 ºC, the Adafruit Si7021's range of -10 to +85 ºC falls closer to the target temperature specified, thus making Adafruit Si7021 a better temperature sensor.

When comparing the humidity levels that each design concept can detect, we based it on RH levels. To begin with design 3 uses the NTC Thermistor which can not detect humidity levels. The DHT11 sensor used for design 1 is great for humidity readings between 20-80%, whereas the Adafruit Si7021 is great for humidity readings between 0-80%. Since both sensors detect similar RH levels the accuracy was referenced to determine the best sensor. The Adafruit Si7021 has a ±3% accuracy while the DHT11 sensor has a ±5% accuracy, thus making Adafruit Si7021 a better humidity sensor.

The sensor interfacing subsystem will be the interface between the environmental sensor and the drone's computational platform. This subsystem is responsible for powering the sensor and receiving the data from the sensor in serial communication. When choosing the best microcontroller our checklist consisted of: low weight and volume. During the design concept stage of the three microcontrollers: arduino nano, arduino uno, and arduino micro. Of the three the arduino nano had the lightest weight and was most compact in size.

When designing our protective casings we made sure to base everything around our problem statement and design criteria. This led to a very small, compact and simple design. Something that was easy to assemble and take apart to cause less hardship on the clients. We also took into account the weight, so we chose to print our cases using ABS plastic. ABS plastic is relatively inexpensive, has good impact resistance, and is also very durable. This makes our cases ideal for JAMZ since they are low in costs yet very effective for holding the sensors and the microcontroller in place, and also protecting them from getting damaged from loose objects. Another client requirement was to make any threading suitable for M3 screws which is visible in our casing.

The initial code of the module relied on one Si7021 sensor relaying constant data to the slave board, forwarded to the master board. However, this implementation gives no reference to the captured data by the sensor, as such, the sensor's readings cannot be verified or regressed to

create tendencies. Also, this code pulled a lot of memory from both the slave and master boards, because they would be always receiving data. To fix these issues, the next code implementation used two Si7021 sensors and used flag coding to only relay information if the measured item or food is outside of the desired bounds. However, an adafruit Si7021 has a specific built-in address, thus the code uses a 8x1 mux as a data selector to each sensor one at a time.

Wiring the module was mainly referenced from the arduino mux and sensor wiring aid website. The concept is simple, the microcontroller (nano) relays both clock and data lines (SCL, SDA) to the multiplexer, who then relays those lines by connecting the mux's SDX and SCX (where x is a number) to the sensors respective clock and data lines. As such, they all share the same clock line and have an open slave master data sender line open (I2C communication protocol). Then, the power wiring was done by wiring nano's ground pin (GND) to the breadboard's ground and linking all components to that same ground. In other words, all components of the module share the same ground. Then, the nano's 5v pin is linked to the board's power lines (+ lines) and each component connects the breadboard's power with their respective VIN pin. Note that the nano is put on a breadboard of its own.

6.1 Temperature/Humidity Sensor Subsystem of Prototype

6.1.1 BOM (Bill of Materials)

Table 4. Bill of materials for the temperature/humidity sensor subsystem.

Item	Purpose	Cost	Quantity	Link
Adafruit Si7021 Sensor	Temperature/Humidity Sensor	$11.28	2	https://www.adafruit.com/product/3251
TCA9548A 1-to-8 I2C Multiplexer Breakout	Enables system to select a data port	$8.76	1	https://www.adafruit.com/product/2717
Breadboard	For electrical connection between components	$10.00	1	https://makerstore.ca/shop/ols/products/breadboard
Wires (Male to Male)	For connection between electrical components	$1.99	1 package (40 jumper wires in each package)	https://elmwoodelectronics.ca/products/male-male-jumper-wires?variant=13911790420020&currency=CAD

Soldering Kit	Enables sensors to be placed directly into the breadboard	$18.88	1	https://www.amazon.ca/Soldering-Adjustable-Temperature-Welding-Tweezers/dp/B0882WCRCS/ref=pd_day0_2?pd_rd_w=aGcdv&pf_rd_p=a0f07c06-3bfe-427e-9527-5be8cea27b66&pf_rd_r=HGHFMS17R7XC3S8HRQW8&pd_rd_r=56fe807c-901a-435c-a5b2-8150c78d9852&pd_rd_wg=xSPLh&pd_rd_i=B0882WCRCS&psc=1

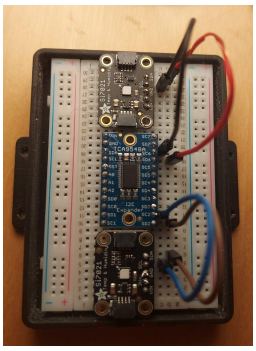
6.1.2 Equipment list

Table 5. Equipment list and quantities for the temperature/humidity subsystem.

Item	Quantity
Adafruit Si7021 Sensor	2
TCA9548A 1-to-8 I2C Multiplexer Breakout	1
Soldering Kit	1
Breadboard	1
Wires (Male to Male)	12

6.1.3 Instructions

Step 1: Solder the two temperature sensors and the multiplexer, using the soldering kit.



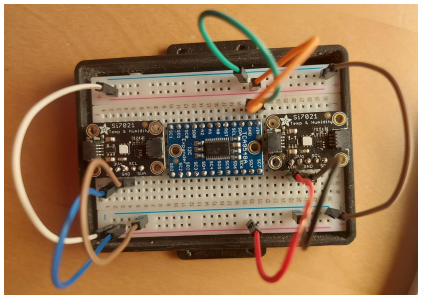
Step 2: Connect data line (SDA) from sensors to lines (ex.SD2) on the multiplexer that correspond to the data line.

Step 3: Connect clock line (SCL) from sensors to lines (ex.SC2) on the multiplexer that correspond to the clock line.

Step 4: Both power lines need to be connected with a wire.

Step 5: Both ground lines need to be connected with a wire.

Step 6: Place both sensors and multiplexer on the breadboard, and connect parts to ground and power (5V).



6.2 Sensor Interfacing Subsystem of Prototype

6.2.1 BOM (Bill of Materials)

Table 6. Bill of materials for the sensor interfacing subsystem.

Item	Purpose	Cost	Quantity	Link
Arduino Nano	Microcontroller	$12.99	1	KEYESTUDIO Nano Atmega328P CH340 Micro Controller Board + USB Cable for Arduino: Amazon.ca: Electronics
Breadboard	For electrical connections between components	$2.46	1	https://www.amazon.ca/Taidacent-Solderless-Breadboard-Experimental-Platform/dp/B081YTBWDZ/ref=sr_1_26?dchild=1&keywords=yellow+breadboard&qid=1618082418&sr=8-26
Soldering Kit	Enables sensors to be placed directly into the breadboard	$18.88	1	https://www.amazon.ca/Soldering-Adjustable-Temperature-Welding-Tweezers/dp/B0882WCRCS/ref=pd_day0_2?pd_rd_w=aGcdv&pf_rd_p=a0f07c06-3bfe-427e-9527-5be8cea27b66&pf_rd_r=HGHFMS17R7XC3S8HRQW8&pd_rd_r=56fe807c-901a-435c-a5b2-

				8150c78d9852&pd_rd_wg=xSPLh&pd_rd_i=B0882WCRCS&psc=1
Wires (Male to Male)	For connection between electrical components	$1.99	1 (40 jumper wires in each package)	https://elmwoodelectronics.ca/products/male-male-jumper-wires?variant=31581987864628&currency=CAD

6.2.2 Equipment list

Table 7. Equipment list and quantities for the sensor interfacing subsystem.

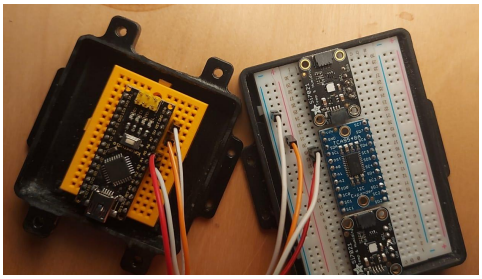
Item	Quantity
Arduino Nano	1
Breadboard	1
Wires (Male to Male)	4 single wires
Soldering Kit	1

6.2.3 Instructions

Step 1: Solder the arduino nano to its pins, enabling for direct attachment on the breadboard.

Step 2: Attach arduino nano to breadboard by placing pins to fit in holes on the breadboard.

Step 3: Connect Arduino nano's 5V to power on the second breadboard.

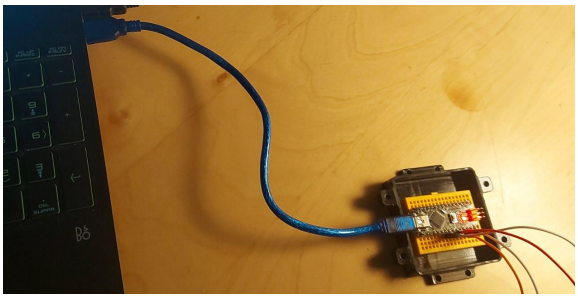


Step 4: Arduino nano's pin A4 (can be used as SDA pin) connects to the multiplexer's SDA pin.

Step 5: Arduino nano's A5(can be used as SCL) connects to the multiplexer's clock line.

Step 6: Arduino nano's ground is linked to breadboard ground.

Step 7: Connect USB cable to power source.



6.3 Software Subsystem of Prototype

6.3.1 BOM (Bill of Materials)

Table 8. Bill of materials for the software subsystem.

Item	Purpose	Cost	Quantity	Link
Arduino IDE	Software for code compilation	$0		https://www.arduino.cc/en/software
Adafruit_Si7021 library	Collection of precompiled routines for code	$0		

6.3.2 Equipment list

Table 9. Equipment list and quantities necessary for completing software subsystem.

Item	Quantity
Arduino IDE	
Adafruit_Si7021 library	
Temperature/Humidity Sensor Subsystem	
Adafruit Si7021 Sensor	2
TCA9548A 1-to-8 I2C Multiplexer Breakout	1
Soldering Kit	1
Breadboard	1
Wires (Male to Male)	12
Code Intersurface Subsystem	
Arduino Nano	1
Breadboard	1
Wires (Male to Male)	4 single wires
Soldering Kit	1

6.3.3 Instructions

Step 1: Use of Adafruit_Si7021 library to receive output data from sensors.

Step 2: Create code that takes the average of the data output by both sensors.

Step 3: Create if/else statement on code that displays warning statement (flagging system).

Step 4: Run code for errors.

Step 5: Compile code on to arduino nano.

6.4 CAD Casing Subsystem of Prototype

6.4.1 BOM (Bill of Materials)

Table 10. Bill of materials for 3D printed housing subsystem.

Item	Purpose	Cost	Quantity	Link
AutoCAD	To design system case for printing	$0		https://www.autodesk.ca/en/products/autocad/free-trial
3D-printer	To 3D print system case	$274.95	1	https://3dprintingcanada.com/products/creality-ld-002r?variant=31825817141317&ab_version=B&gclid=CjwKCAjwvMqDBhB8EiwA2iSmPPaK5F_3amtcSiHuQo767tiuwHGBy5iyZQEs-c9xd-CiGY0Q56Ky4RoCziYQAvD_BwE
M3 Screws	To screw casing shut	$0.62	4 bag (3 piece per bag)	https://www.homedepot.com/p/Everbilt-M3-0-5-x-10-mm-Plain-Metric-Socket-C

				ap-Screw-3-Piece-per-Bag-803188/204808024

6.4.2 Equipment list

Table 11. Equipment list and quantities necessary for 3D printed housing subsystem.

Item	Quantity
AutoCAD	
3D printer	1
M3 screws	12

6.4.3 Instructions

Step 1: Design casing for sensor interface and temperature/humidity sensor subsystem.

Step 2: 3D print design.

Step 3: Enclose the subsystem within cases.

Step 4: Screw casing shut with M3 screws.

6.5 Testing & Validation

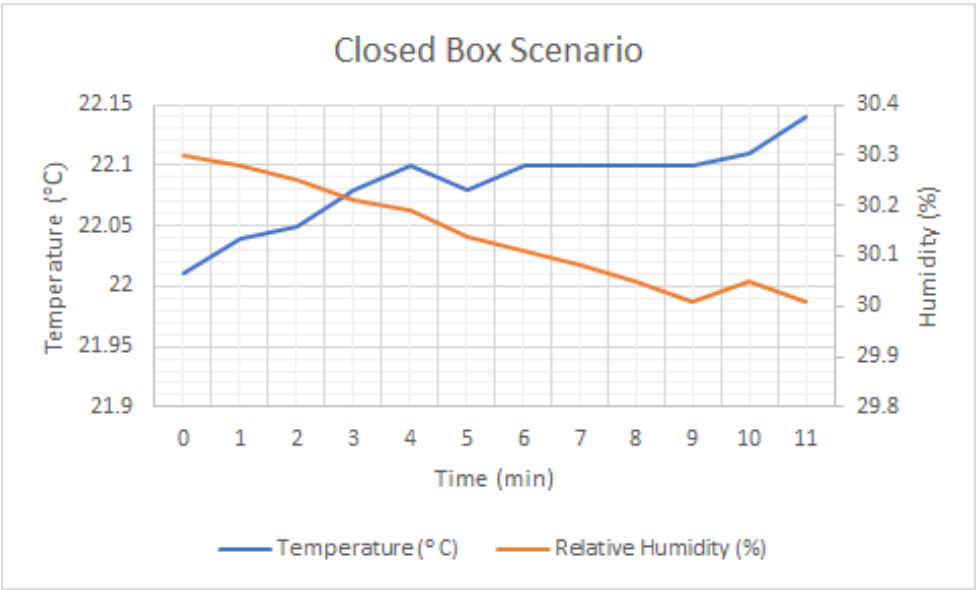
In order to test and validate the climate sensor prototype, the majority of testing was concerned with the data output from the sensors themselves. Additional testing was also conducted on the placement of the sensor and design of the sensor system housing.

The Adafruit Si7021 sensors were tested for accuracy, precision, robustness in measuring both temperature and humidity. Each individual sensor was first tested under ambient conditions, and then with a hair-dryer test, to see how the temperature and humidity values fluctuate from a source of dry and hot air.

For the ambient condition test, the Si7021 sensor was placed inside a closed cardboard box for 11 minutes. Overall, the temperature and humidity data was consistent over the measured time interval.

Figure 3. Sensor inside a closed cardboard box during the ambient conditions test.

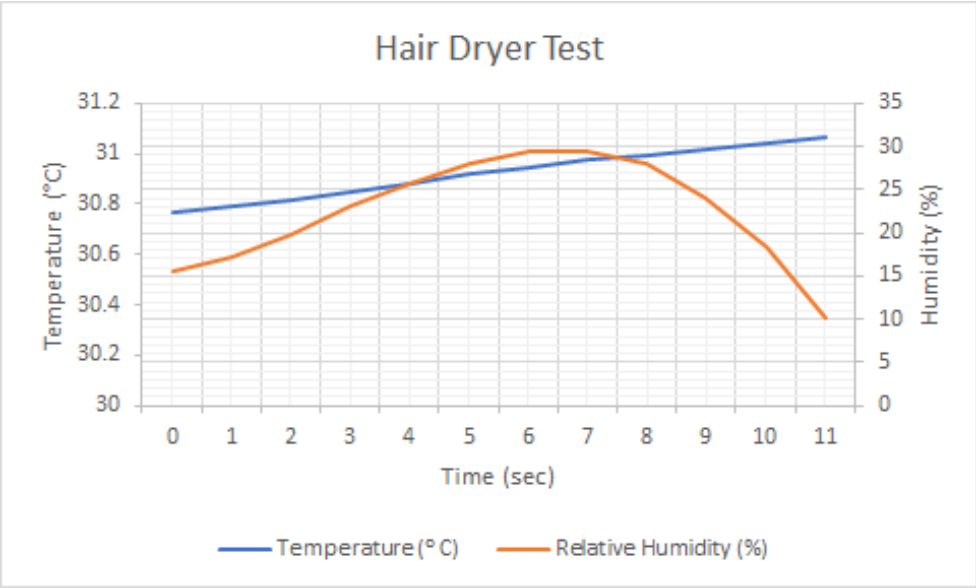
Figure 4. Temperature and humidity measured under ambient conditions for 11 minutes.



The next test performed was the hair dryer test, to see how the sensors would respond to a source of hot and dry air. The Si7021 sensor was first placed inside an open cardboard, with a hair dryer aimed into the box. The hair dryer was then turned on to measure the change in temperature and humidity inside the box.

Figure 5. Sensor inside an open cardboard box with a hair dryer during the hair dryer test.

Figure 6. Temperature and humidity measured during the hair dryer test.

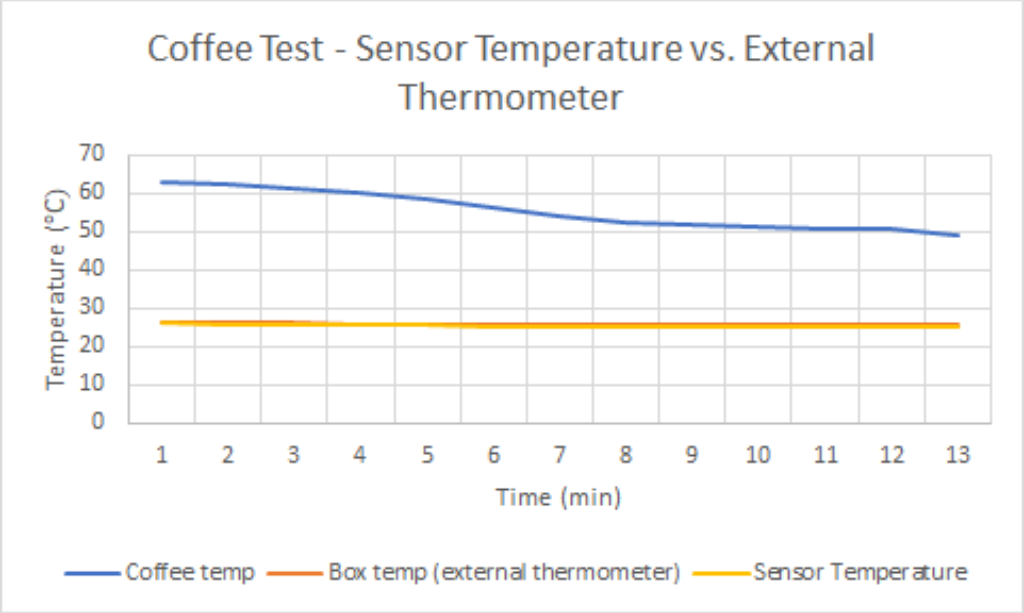


With the hair dryer turned on, the measured temperature increased and the measured humidity decreased over the measured time interval of 11 seconds. This showed that the sensor will respond as expected to temperature and humidity shifts in the environment it is placed in.

The final test conducted with a single sensor was the coffee test. In this test the Si7021 sensor was placed inside a cardboard box with an open cup of coffee and an external thermometer in the box, as well as a thermometer in the coffee, in order to measure the difference between the temperature of the box as measured by the Si7021 sensor, and the temperature of the coffee within the box as measured with an external thermometer.

Figure 7. Coffee test with a single sensor placed in a box with coffee and an external thermometer

Figure 8. Temperature and humidity measured during the coffee test from the coffee, the external thermometer, and the Si7021 sensor.



The single sensor coffee test showed that the temperature of the box as measured by the Si7021 sensor matched closely with the temperature of the box as measured by an external thermometer. The coffee test also showed how the temperature of the box may respond to the temperature of the cup of coffee.

Figure 9. Plot of temperature readings for both sensors and their average temperature during two coffee tests

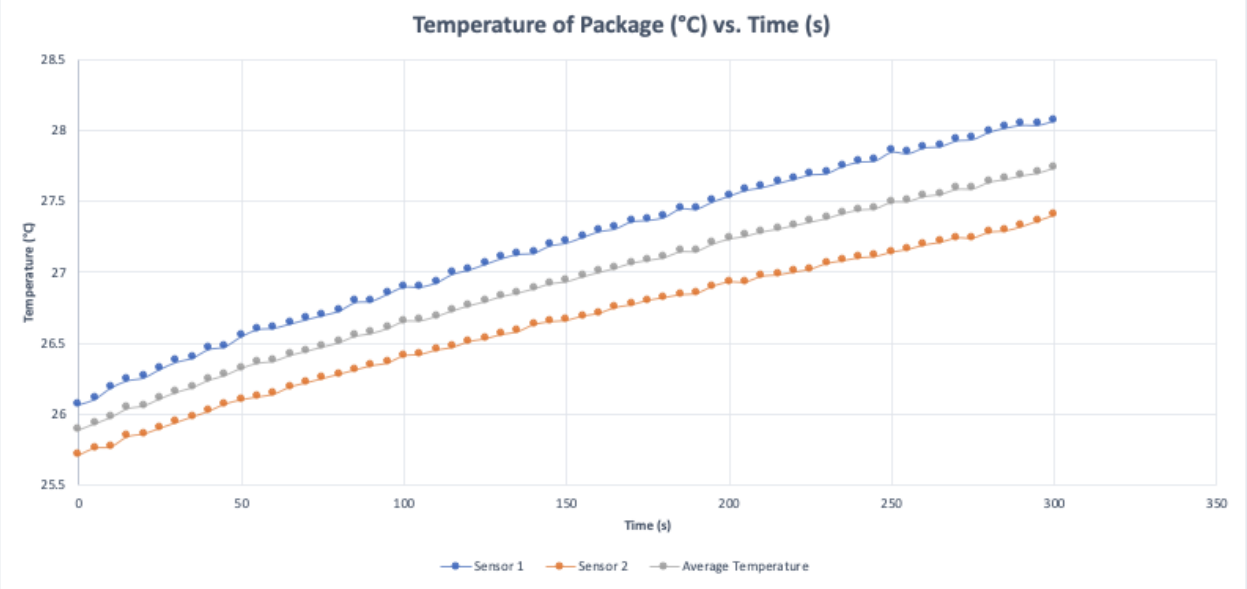
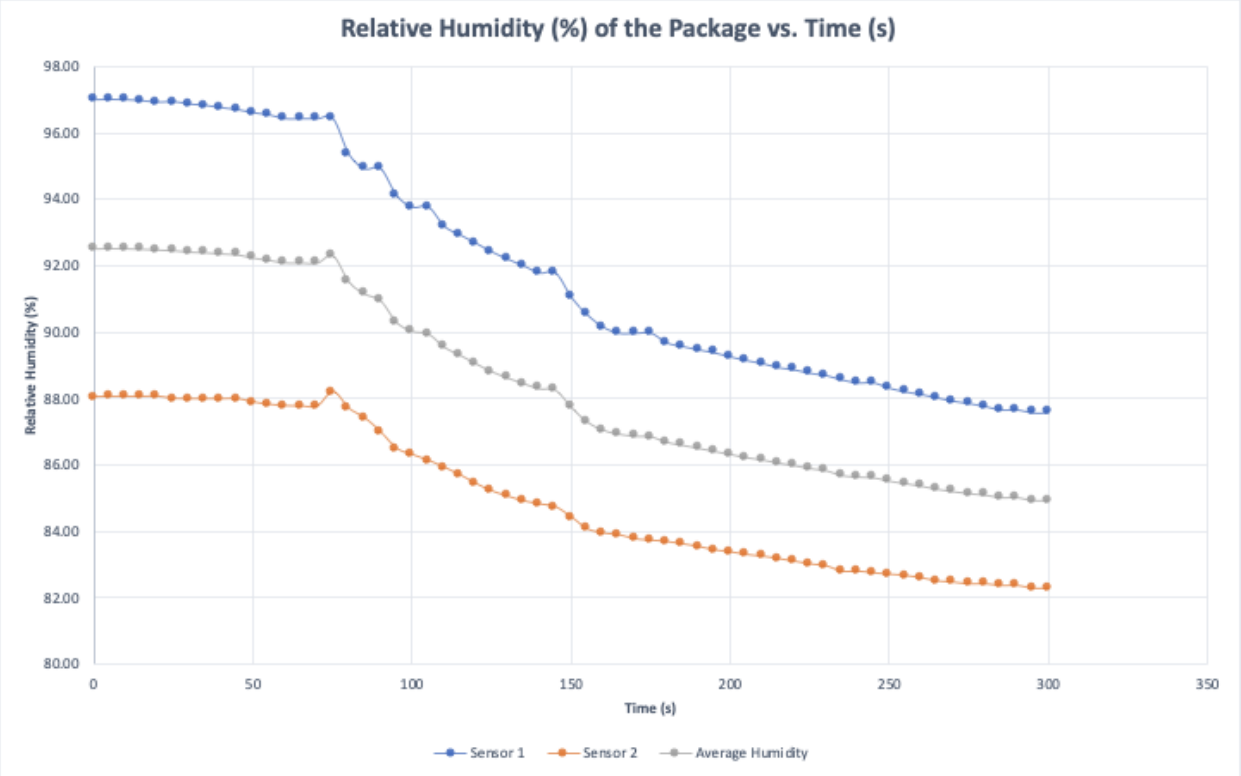


Figure 10. Plot of humidity readings for both sensors and their average humidity during two coffee tests



For the purpose of this project and based on feedback received from our previous prototypes, we chose coffee as our testing product. This is because coffee could be considered as a worst case scenario of a food/drink type that is being shipped. The internal temperature of coffee is so large that monitoring its temperature loss would be easier than another type of food. Our program code was developed to fit the needs of shipping coffee and temperature ranges were determined of what is considered drinkable coffee.

7.0 Conclusions and Recommendations for Future Work

In this document, a detailed description of the user and product manual of our final design. Over the course of four months, we were able to build and present a solution to our client, JAMZ. This was done progressively on a weekly basis and we learned that each step builds onto the last and you can never have enough data. Based on previous work that was submitted each week and feedback received from the client, we were able to anticipate and learn from our errors and fix them. This helped us build a better design and learn a lot about Arduinos and how they function. Thorough testing of the entire sensor system is key as it helped us gather more information and make further improvements to our final design.

We were able to develop and present a comprehensive prototype to JAMZ but if more time had been provided, the code is one important aspect we could improve on and expand it. This is because for the scope of the project, the code was specific to one type of food, our test subject being coffee. The code would be built on to include other types of foods as well in order to allow multiple foods to be shipped. And to further improve our design, we would implement a way for the temperature readings collected from the sensor to be relayed to the customer as well in real time so they also have access to that information instead of just the operator.

8.0 Bibliography

Adafruit. A. (2019). Adafruit Unified Sensor. Arduino. Retrieved from https://www.arduino.cc/reference/en/libraries/adafruit-unified-sensor/

Cambell, Scott. C.S (2019). Basics of the I2C Communication Protocol. Circuit Basics. Retrieved from https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

Dherrada. (2020). Adafruit Si7021. GitHub. Retrieved from https://github.com/adafruit/Adafruit_Si7021

Lady Ada, LA (2020). Arduino Code. Adafruit. Retrieved from https://learn.adafruit.com/adafruit-si7021-temperature-plus-humidity-sensor/arduino-code

(n.a) (2018). Master Reader/Slave Sender. Arduino. Retrieved from https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterReader

(n.a) (2019). Buck Converter: Basics, Working, Design and Operation. Components 101. Retrieved from https://components101.com/articles/buck-converter-basics-working-design-and-operation

(n.a) (2019). Using the TCA9548A 1-to-8 I2C Multiplexer Breakout with Arduino. PMDWay. Retrieved from https://pmdway.com/blogs/product-guides-for-arduino/using-the-tca9548a-1-to-8-i2c-multiplexer-breakout-with-arduino

(n.a) (2020). C++ If … Else. w3schools. Retrieved from https://www.w3schools.com/cpp/cpp_conditions.asp

9.0 APPENDIX I: Design Files

Appendix Table 1. Referenced Documents

Document Name	Document Location and/or URL	Issuance Date
MakerRepo Project Page (Deliverables, CAD, code)	https://makerepo.com/elacr104/838.gng1103d10-sensor-society-	Apr 11, 2021
Deliverable A-J	See previous submissions and MakerRepo project page.	[January 24th 202, March 26th 2021]
GNG1103 Lectures	See course brightspace	[January 11th, April 14th 2021]
Wrike Project Tasks Chart	https://www.wrike.com/frontend/ganttchart/index.html?snapshotId=Jnp4IhJRrd277nGLu6AOeLsZkGFrVmc2%7CIE2DGNBVGEYTELSTGE3A	Apr 11, 2021

We were unable to attach the CAD files to this document because they are on solidworks and they cannot be transferred onto a google document.

Please see all previous deliverables, CAD files and code files on the MakerRepo project page.

10.0 APPENDIX II: Other Appendices

Adafruit_Si7021 Functions: Library Reference

Create the Adafruit_Si7021 object with:

Adafruit_Si7021 sensor = Adafruit_Si7021();

Initialize the sensor with:

Sensor.begin();

Returns True if the sensor was found and responded correctly and False if it was not found

Query the temperature in Celsius with:

Sensor.readTemperature()

Returns a floating point temperature

Query the humidity with:

Sensor.readHumidity()

Returns the humidity as a floating point value between 0 and 100 %humidity

To query sensor serial number:

Sensor.readSerialNumber()

To read out the 8 bytes unique ID Than you can access them from “sensor.sernum_a” and “sensor.sernum_b”