Final Report

Food Cabinet Project

Upgrading and Repairing Food Cabinet

Group Members

Justin Wei, Noah Labbe, Megan Wong, Henry McCarthy

MAE4220/4221/5220

TABLE OF CONTENTS

EXECUTIVE SUMMARY....4
1 SOCIAL CONTEXT....5
1.1 Why Are We Doing This?....5
1.2 Who Cares?....5
1.3 Beyond Tompkins County....6
1.4 Community Partnership....6
2 REVIEW OF RELATED WORK....6
2.1 Design Critiques....6
2.2 Design Improvements....7
3 PROJECT SCOPE....7
3.1 Vision....7
3.2 Design Objective....7
3.2.1 Mechanical Design....7
3.2.2 Electrical Design....8
3.3 IoT System Under Consideration....8
4 TECHNOLOGICAL DEVELOPMENT....9
4.1 Mechanical Development....9
4.1.1 Weatherproofing the Roof....9
4.1.2 Door Sensor....9
4.1.3 Camera Installation....9
4.2 Electrical Development....10
4.2.1 Prototype....10
4.2.2 Testing & Results....10
4.2.3 Final Design....10
4.3 Justification for Sensor Selection....11
4.4 Wireless Communication and System Integration....12
4.4.1 Sensor Integration and Sending Data....12
4.4.2 Power Management....12
4.4.3 Storing Data....13
5 PLANS FOR NEXT STEPS....13
5.1 Future Work....13
5.2 Barriers....13
5.3 Recommendations....14
APPENDIX A. Cabinet Images....15
APPENDIX B. Bill of Materials....21
APPENDIX C. Transmitting Data....22
C.1 TTN....22
C.2 Google Sheet....23
APPENDIX D. Code....24

D.1 Food Cabinet Code 24
D.1.1 TwoCameraFinalCode.ino 24
D.1.2 Catena_Fram32k.cpp 34
D.1.3 Catena_Fram32k.h 38
D.1.4 Ece4950_Mcci_Fram.cpp 40
D.1.5 Ece4950_Mcci_Fram.h 46
D.1.4 keys.h 49
D.2 Decoder 50
D.3 Google Apps Script [2] 50
RESOURCES 54

EXECUTIVE SUMMARY

Our mission is to confront the issue of food availability for the less fortunate in Tompkins County using IoT technology. We aim to improve upon the previous food cabinet design by collaborating with Mutual Aid Tompkins.

Our vision for the cabinet is a simple yet effective system that is safe and easy to use for those in need of food. We designed our product to collect important information while maintaining security along the many who use it. Challenges with the previous designs include cabinets deteriorating due to weather, users damaging the electronics in the system due to a fear of being recorded, and lack of data access for the people at Mutual Aid Tompkins.

Our weatherproofing improvements to the previous design include: weather-treated plywood and roof shingles. This solves the deterioration problems of previous designs, where the plywood roof was warped and compromised the electronic's safety from harsh weather. Our IoT solution is a camera-oriented system that collects data from each shelf and sends it to Mutual Aid. This data is shown as a percentage of shelf space showing, for example, 100% is an empty shelf and 0% is a completely full shelf. This eliminates any issues with the weight sensors such as faulty wiring, misplaced shelves altering readings, or asymmetric food placements revealing misleading data. Also, a large variety of food items are placed in the cabinet, all having very different weights. This can lead to a cabinet reading which seems to be full but may have a few lighter items. Or a cabinet reading to be almost empty when it is filled with much lighter food items like bread or chips.

While this method is much more accurate for showing capacity, there are some limitations. We had to change the cabinet from a three shelf system to a two shelf system in order to compensate for the limited field of view from the cameras. In order to maintain complete anonymity, our system is designed to only take pictures once the door is closed. This eliminates the possibility of anyone being recorded. Even if the camera were to take a picture when the door is open, the angle would only show the person’s hands inside the food cabinet. We have also included wire channels in order to hide the wires connecting the electrical components. These wires have been a target of vandalism and hiding them will increase the lifetime of the system.

Going forward, the most important development needed is an easy access gateway between the IoT system and Mutual Aid. As of now, we are sending the data to Mutual Aid and finding a location with a gateway they can access is currently under consideration. Also, finding a better way to ensure the doors are closed when the cabinet is not in use is another major issue. With our design, the door being closed is extremely important to the cabinet's functionality.

One idea for future improvements is combining components from the previous design to our prototype. This way a weight and image of each shelf can be taken, giving Mutual Aid Tompkins two sets of valuable information about the capacity of each cabinet. We are open and encouraging other potential changes in order to make the food cabinet as accessible, safe, and efficient as possible.

1 SOCIAL CONTEXT

1.1 Why Are We Doing This?

Homelessness and food insecurity are issues that affect both the community directly around Cornell University and the state of New York as a whole. According to the non-profit “Feeding America”, 10% of people in Tompkins County experienced food insecurity, and a staggering 18% of children experienced food insecurity in 2018 [1]. In addition to the people facing food insecurity, there is also a large homeless population in Tompkins County, specifically around the city of Ithaca. Homeless individuals in upstate New York have trouble finding regular meals due to the rough winters and limited aid from New York State. To combat this, Mutual Aid Tompkins County organizes a food pantry service, where people can anonymously donate food and other necessities to people who are in need.

The food pantries that Mutual Aid Tompkins County organizes are enormously successful and are maintained through various Facebook groups that the community organizes. People in the Facebook group post announcements about when and where a donation is made, which food pantries require more supplies, and if an issue comes about with one of the pantries. As great as this method of communication is, the information about the food pantry is not readily available, easily accessible, or consistently accurate.

Our desired impact on this system is to improve communication of the food pantry with the donors and beneficiaries. With this better method of communication between the donor beneficiaries and cabinet, many parts of the process could be made more effective. The beneficiaries can know when a donation happens so that they don’t need to manually check, they can know how much food was donated, and they can expect more reliable updates as to when these donations will be deposited. These changes would make them less likely to be food insecure. Our impact can reach the donors as well, allowing them to know when the food cabinet is empty, the most common times that beneficiaries come to the cabinet, and how much food is currently available. This would allow for them to make more effective decisions about when and how much to donate, and it would lead to less food insecurity in the recipients.

1.2 Who Cares?

The list of people who care about the effectiveness of the food cabinets extends from the locals that use them, all the way up to the elected New York State representatives. The donors and beneficiaries of the food pantry care about our project because it will make the process of using the food pantry more convenient and effective. The donors of the food pantry are all volunteers who spend their own time and money to help less fortunate people. Streamlining this process by allowing them to remotely access information about the pantry will allow them to spend less time and effort, while also increasing their effectiveness as volunteers. This could potentially draw more donors, as the information about the food pantry will be more readily accessible, and the process of donating will be more clear. The beneficiaries of the food pantry will also care about our contribution, as the donations will arrive at more effective times and the pantry will be maintained better. With the added potential that this will draw more donors, they will also receive more donations.

In addition to these two groups, the elected representatives that are in charge of Ithaca, Tompkins County, and New York State will also care about this innovation as it will provide improvements to the quality of life of those who reside in the areas that they are in charge of, boosting their statistics and making them look good as elected leaders.

1.3 Beyond Tompkins County

According to Feeding America [1], 1 in 5 children in the United States face hunger and are unsure of where their next meal will come from. On top of that, children of African American and Latino communities are twice as likely to face food insecurity. Children who suffer from child hunger are more likely to have poor academic performance, increased health problems, and developmental differences in both their body and mind. These historically disadvantaged communities can be positively impacted from a reliable source of donated food not only in the areas directly relating to hunger and hunger statistics, but also in branching out impacts in the already mentioned areas that relate to childhood hunger. More reliable nutrition sources for these disadvantaged children would lead to better education outcomes and better development. If our solution is effective in the Tompkins County implementation, it can be widely implemented easily on a larger scale and could potentially help decrease the disparities between the different communities that exist in the United States.

1.4 Community Partnership

We are partnering with Mutual Aid Tompkins [4], which is a volunteer-run organization that aims to provide services and resources for those in need within the Tompkins County Community. Our main point of contact is Theresa Fulton, a volunteer with Mutual Aid and the Director of the US division of NSIP, LLC. Theresa worked with the IoT team last year and has extensive experience in overseeing the food cabinet system. Through a Zoom meeting, Theresa walked us through all of the suggested improvements for the food cabinet and the current state of the project. Theresa is currently in possession of the prior team's food cabinet, which was severely damaged due to weather and vandalism. We will continue to be in contact with Theresa throughout the next months as we start to implement our solutions to the food cabinet. She will also aid us with finding a location for the food cabinet, once we repair it to working order.

2 REVIEW OF RELATED WORK

2.1 Design Critiques

Looking at last year's report [3], the previous designs often encountered similar problems, mostly with vandalism and weather. A major problem was the location and appearance of the wires inside the cabinet body. People could open the door and directly access the wires which are essential for the system to function properly. This led to wires being ripped out of the system as well as some other essential electronics. Another issue is keeping the cabinet doors shut when nobody is using them. The two-door design that most of the cabinets have are more prone to being left open, which would cause food to come into contact with the weather and even local wildlife. The design we will work on has one door, which is more effective. However, there still is no system in place to shut the door if it were left open. The materials used for the walls and roof of the design are showing major signs of wear from the weather and

use. The plywood is splitting which can lead to structural problems. While the electronics and code for the design are functional, there is no efficient way to receive the data measured over time. People in charge of filling the cabinets have no way to predict when the cabinets need to be replaced. Also, the weight sensors are unreliable because they are difficult to maintain due to the higher number of electronics to wire up and maintain (as opposed to the cameras).

2.2 Design Improvements

To deal with the vandalization, wire channels, and a boarded-up lockbox will be used to house all essential electrical components for the system. A key will be needed to access any of the major electrical components of the system. The roof will be replaced with a more durable and weather-resistant material that can withstand harsh weather conditions. The sealant will be used to fill any cracks in the joints of the cabinet body to keep the food dry during any rain or snow. A door close sensor will be installed to ensure the door is closed when not in use, and the data taken from the cameras will be sent to the volunteers working with the food cabinets, consisting of the shelf capacity percentages.

3 PROJECT SCOPE

3.1 Vision

Our vision for the food cabinet project is to improve upon the previous designs, specifically with anti-vandalism and weatherproofing. Additionally, we want this food cabinet to send more reliable sensor data back to the organizers at Mutual Aid Tompkins so they can better understand the impact of the food cabinets within the county. The previous design's food sensors had inconsistent readings, so we will also improve upon the reliability of the data that is sent from the cabinet.

3.2 Design Objective

3.2.1 Mechanical Design

This semester, we will be improving the mechanical design of last year's food cabinet and switching from weight sensors to a camera for identifying food cabinet capacity. One of our main goals is to prevent vandalism, as our community partner has informed us that any exposed electronics and wires would likely be ripped out. To combat this, we will add wire channels covering the wires so that they will be out of sight and inaccessible to those who don't need to access the electronics. Part of the camera installation will include a door close sensor, to allow for the camera to re-update the data every time that the door is shut. Any exposed electronics will be hidden or embedded into the shelves such that they cannot be ripped out without power tools.

Other structural improvements include further weatherproofing of the cabinet, as the current cabinet's roof has warped a great amount due to water and weather damage. We will replace the current roof with a new panel of treated plywood to prevent weather damage and add shingles on top of the panel for further protection.

3.2.2 Electrical Design

In terms of electrical design, we will be replacing the weight sensors implemented by last year's group with cameras. These cameras will take photos of the cabinet and determine the shelf capacity by calculating the number of pixels present that match the shelf's color. There will also be LED backlights placed on the roof of each shelf to ensure proper lighting for when photos are taken, and a magnetic door sensor implemented to signal to the camera when a photo should be taken. Using the cameras instead of the weight sensors will provide greater accuracy in determining how full the shelf is, so volunteers will be able to restock the cabinets more efficiently. Initially, Theresa was hesitant to jump on our design due to concerns about anonymity, but after laying out the pros and cons of both systems she was in favor of the new method.

Additionally, we want to reorganize the circuit and make it more permanent so that it's easier to access. This will allow the circuit to sit inside the cabinet more neatly, since the previous circuit was messily placed inside the electronics container at the top of the cabinet, making it hard to understand the wiring. To do so, we will solder all components in place, making the connection much more secure and easier to observe.

3.3 IoT System Under Consideration

During a meeting with Theresa, we pushed for a design that uses cameras instead of weight sensors to measure the food capacity of the food. Initially, she was hesitant to accept this new method due to the potential breach of anonymity between the cabinet and the people taking food. However, we overcame this issue once we developed a method in which no person will be photographed. Our system would only capture pictures once the door is closed, ensuring privacy for any user. Also, the camera is only able to see the shelf. Therefore if a picture were to be taken of someone grabbing food, it would only be their hands in the picture. To calculate food cabinet capacity, we painted the shelves black and determined the percentage of black pixels in the image to get a capacity metric. According to last year's design, the cabinet was powered by a rechargeable LiPo battery pack that was connected to a 3.5W solar panel to recharge the batteries for consistent power. Additionally, the Feather M0 should have a built-in voltage reference we can use to measure the battery percentage of the lithium-ion battery. A Feather M0 board will receive this data from the sensors and send data over the TTN.

A gateway will be deployed in a local establishment that is willing to host it (someone's house, store, cafe, etc) that is within range for the Feather M0 to communicate over LoraWan, and the data packets can be sent over the TTN network. We can create a decoder to decode the uploaded sensor information stored in the data packets, and the data from TTN will be automatically populated in a Google sheet document with the timestamp and food capacity information for both shelves. Theresa can then upload the report to her Facebook group for Mutual Aid Tompkins so her associates can assess the cabinets' statuses. There will also be no cameras monitoring the cabinets to ensure privacy and anonymity. A diagram of the overall system is illustrated in Figure 1.

4 TECHNOLOGICAL DEVELOPMENT

4.1 Mechanical Development

4.1.1 Weatherproofing the Roof

As stated in our Design Objectives, we have taken further measures to weatherproof the cabinet. When we first got the cabinet, the roof was in very poor condition. The sides were all warped, making it unusable. To ensure that this would not happen in the future, we placed shingles on top of the roof, paired with weather-treated wood for the structure (Fig. 2). Upon initial investigation, we found that the previous groups that worked on the cabinet had not used weather-treated wood, and made it out of a flimsy material that was easy to vandalize. As a result, we made sure to order wood that was hard to mistreat and warp due to weather.

To embed the solar panel into the roof, we used a similar approach to last year's group. We created a rectangular hole within the center of the roof to precisely accommodate the solar panel, ensuring a secure and snug fit to prevent theft. Additionally, we drilled a hole in the rectangle for the wire of the solar panel to channel through to the electronics section of the cabinet.

4.1.2 Door Sensor

The next mechanical development that we had with the door was the placement and installation of the magnetic read sensor, which tells the IoT system when the door is shut. A mission goal of this project is to keep most of the electronics hidden so that the users of the cabinet are not scared of being recorded and are not prone to vandalize the machine. With this under consideration, we had to make sure that the sensor drew the smallest amount of attention to itself as possible. To do this, we installed it midway down the door, on the side closest to the hinges, and routed its wires along the bottom of the top shelf to avoid detection (Fig. 11).

4.1.3 Camera Installation

Due to issues with the previous group's sensors, we developed a different process in order to collect the data Mutual Aid needed. Instead of using weight sensors, we decided to use a camera to detect the amount of food in the cabinet. Our partner was skeptical at first due to the fear of breaching the anonymity of people donating and withdrawing from the cabinet. In order to overcome this issue, we chose a design in which the camera only collects pictures of the shelf, where the camera would be mounted to the top of the shelf, facing down. Moreover, the camera is programmed to take pictures only when the door is shut. Therefore, no pictures will be taken of anybody who chooses to give or take from the cabinet. Theresa was in favor of this idea after a brief explanation and was excited to move forward with it.

Another mechanical development we had was due to the constraints of the cameras. The cameras had a limited field of view (FOV), and with the shelving configuration that was previously installed, there was no way to have the cameras effectively see the entire shelf area. To fix this problem, we took out the two previously installed shelves and installed only one. This one shelf would allow for two compartments, both with high enough ceilings to allow for the camera's field of view to reach the entire area (Fig. 3).

The final mechanical development that we had was the installation of the camera mount. To create the requirements for the camera mount, we had to consider its visibility from the outside, its field of view and the connection wires. Our solution with all of these requirements was a mount that was sub flush with the top of the top shelf, with a circularly cut wooden piece to hold the camera (Fig. 9). This height allowed its field of view to see the entire shelf area and hold the camera in place. Once the cameras were mounted, we added wire channels throughout the cabinet to hide the wires and hopefully decrease the potential for vandalism (Fig. 10).

4.2 Electrical Development

4.2.1 Prototype

To provide proof of concept for Theresa, we agreed to test the cameras on one shelf. To determine the shelf color, we researched the most commonly found colors on food packaging [5] and tested the camera's detection abilities of various colors. We ultimately decided that black would be the most reliable and easiest to implement, as it contains all 0 values on the RGB color spectrum. This way, we can establish if an item is placed in the cabinet if the camera detects any pixels where the R, G, or B values are above 0.

The setup for our prototype is illustrated in Figure 4. As stated above, we drilled a hole at the top of the shelf for the camera to look down through, and taped the LED on the top of the inside of the cabinet. Using a battery to power the LED temporarily, we ran our code to see what percentage of black pixels were detected by the camera with a fully empty cabinet. Then, we placed various items inside the cabinet to test whether or not the percentage decreased.

4.2.2 Testing & Results

In testing our prototype, we found that we had to increase the threshold of the RGB565 values, as the "black" color the camera was seeing was not a true black. In order to define the correct threshold for the most accurate results, we tested many different items of varying colors inside the cabinet, tuning the threshold so that it correctly detected the relative number of items present. We found that the camera detected "black" the best when the threshold was set to $R \leq 18$, $G \leq 36$, and $B \leq 18$.

For our preliminary prototype, we obtained the results shown in Figure 6 with the items placed in the cabinet shown in Figure 5. With the fully empty cabinet, we were able to tune the threshold for detecting black pixels so that 99.98% of the pixels were seen as black. When the first object, a box of screws, was placed inside, the black pixel percentage went down to about 84%, indicating that the camera was able to sense the object inside. Furthermore, when a second object, a can of Celsius, was placed inside, the black pixel percentage decreased even more, to about 79%. This demonstrated the success of our camera setup, which we were able to show to Theresa, who was excited about our results and agreed to move forward with implementing the cameras on both shelves.

4.2.3 Final Design

After developing a fully working prototype, we were able to integrate the second camera into our design. Since the bottom shelf had no space above it to hide the camera (the top shelf had the electronics section above it), we had to mount the camera in a different way in order to make it flush with the surface. To do

so, we first mounted the camera onto a small piece of wood (see Fig. 12), then fitted it into a hole drilled into the shelf, similarly to the top shelf. We then painted the wood black to blend in with the rest of the shelf (see Fig. 13).

Once the second camera and LED were embedded, we were able to update our script to account for both so that a picture of each shelf was taken. We then conducted similar testing to our single shelf prototype to make sure the thresholds still gave accurate results. This time, we decided to use actual food items to test the accuracy of the capacity measurements (Fig. 14), and updated the data sent to TTN to incorporate information about both shelves. As one of our goals was to be able to send cabinet data to our community partner, we also created a Google Sheet where the TTN data would be uploaded too and interpreted for viewing.

4.3 Justification for Sensor Selection

Theresa initially was hesitant to be on board with our idea to replace the weight sensors because she was concerned that it would violate anonymity. However, after we proved that our design is easy to build/maintain, anonymous, and superior to the weight sensors using the following pros and cons chart, she agreed with implementing the camera into the food cabinet.

Device	**Pros**	**Cons**
Weight Sensors	● Cheaper ● Full anonymity	● Shelves warp due to unevenness from sensors ● Can't analyze capacity very accurately because some items might be much heavier than others ● More wires and electronics → higher chance of vandalism, harder to build/maintain, higher chance of something breaking
Cameras	● Estimate shelf capacity based on surface area coverage ● Easier to build and maintain, only 3 cameras and 3 lights wired up compared to the 12 load cells	● Potentially breaches anonymity ○ Can mitigate this by taking pictures only when the door is closed by using an interlock condition ○ If pictures did happen to be taken when the door was open, the camera is only angled from above and will only see hands ○ If somehow an image of a face was taken, the camera resolution is so low that the face would be unrecognizable (160x120 pixels scaled down by a factor of 4, so 40x30, usually needs at least 320x240 to detect facial features)

The camera is an optic device where the lens (which can be adjusted to increase magnification or FOV) of the camera refracts light and converges it, creating an inverted image onto the camera sensor. The camera sensor then converts that light into an electrical charge within each of the semiconductor materials on the pixels on the sensor. Finally, this charge is converted into electrical signals to be processed as digital image data.

We chose to use the ArduCam Mini 2MP Plus - OV2640 SPI Camera Module because it communicates over SPI and I2C, which is supported by the FeatherM0 board.

This sensor is appropriate for our design because it has an FPGA onboard that can handle real-time data processing such as image preprocessing, data compression, and format conversion, offloading the number of processing tasks of the microcontroller on the FeatherM0 board. This is critical because it enables us to inexpensively take images of the cabinet and perform low-level image processing with a limited amount of processing power and memory. Since we are taking 160x120 images and scaling them down to 40x30 by sampling every 4th pixel, only 2.4KB of data is stored on the RAM of the FeatherM0 board, where it is erased and replaced after each loop. This sensor also has 60 degrees of FOV, so we should be able to fit at least two shelves for the current food cabinet, but the height of the cabinet may need to increase to accommodate more shelves.

4.4 Wireless Communication and System Integration

4.4.1 Sensor Integration and Sending Data

In order to detect food capacity in the cabinet, the SPI camera takes a 160x120 pixel image of the shelf storing any food items. This picture is taken 10 seconds after the door's magnetic read sensor recognizes that the door has been opened and closed and will make sure to only take pictures if the door has been closed for these 10 seconds. This ensures that the door cannot be spammed with data, if someone continuously opens and closes it, for example. Additionally, we have connected the I2C FRAM to the Feather board, which is important in our design because it allows us to store our data and retain it even when power is lost. This also allows the cabinet to resume communication over LoRaWan without having to rejoin the network each time power is lost.

With the Feather board, the image data will be downsampled from 38.4KB to 2.4KB and then stored on the onboard 32KB RAM. The onboard processor will do a calculation to determine how much of the surface is visible. This calculation will output a percentage of the cabinet bottom that is visible and will be sent to the gateway and uploaded to the TTN App via LoRaWAN (see Fig. 15). With the TTN decoder (Appendix D.2), it is able to interpret the data from bytes into readable numbers that represent the capacity of each shelf as a percentage.

4.4.2 Power Management

In order to power the system, a solar panel will charge a rechargeable 3.7V lithium-ion polymer battery, which will be responsible for powering all electronics. The electronics will only wake to take an image and upload the data, then they return to a deep sleep. This greatly reduces the amount of power used by the electronics, as it is only powered on when pictures need to be taken.

4.4.3 Storing Data

The data sent to TTN is stored on a Google Spreadsheet (Fig. 17) using a webhook integration (Fig. 16) and Google Apps Script (Appendix C.2). The information recorded includes the timestamp of when the data was sent, and the percentages of the top and bottom shelves. Each time data is sent to the spreadsheet, a new row is appended with this information. Moreover, to further interpret the data, we have the spreadsheet calculate the average of the two shelf capacities as a representation of the total capacity of the cabinet. Additionally, we added a column for the "status" of the cabinet based on the total capacity, where a capacity over 50% is considered "Good," 20%-50% is considered "Ok," and under 20% is considered "Low." To demonstrate the accuracy of these values, we filled the cabinet with what we determined to be Low, Ok, and Good amounts (Fig. 7), and obtained the results shown in Figure 8.

5 PLANS FOR NEXT STEPS

5.1 Future Work

To improve on our design, the next group may need to upgrade the camera sensor or design a custom adapter for attaching a fisheye lens onto the OV2640 so it can see the entire shelf with less vertical distance between the shelf and the camera since our design only supports two shelves using the old cabinet. Another option would be to make the food cabinet taller.

To make the food capacity data more accurate, perhaps a two-camera setup can be implemented on each shelf to perform stereoscopic imaging, enabling a volumetric measurement rather than a surface area measurement. This may be more computationally expensive than our current design, and something stronger than the FeatherM0 board might be necessary to pursue this design. Another option could be to combine the camera setup with the weight sensors from the year before and develop a method to calibrate both sensors to calculate the food capacity with greater accuracy.

Additionally, the self-closing hinges may need to be improved since the food cabinet door can be swung open by the wind and animals may try to get it. This can impact the power consumption of the system since it will set off the system to take images of the cabinet more times than necessary.

5.2 Barriers

In terms of regulatory barriers, there are potential legal issues with placing a camera in public. For instance, having the camera in a hidden location could pose some problems, especially if people do not know about it. For instance, there are surveillance laws in NY state that state that it is legal to place surveillance cameras as long as they are not in private locations such as changing areas and bathrooms. There are even stricter laws on audio recordings, as there needs to be one-party consent. In our instance, we are only taking images so this does not apply to us. This issue may pose a problem if we expand beyond Tompkins County and NY state, as different states may have different laws. We also need to make sure the food cabinets are within the vicinity of an establishment that is willing to host the gateway.

For economic barriers, we may face operational costs as more food cabinets get deployed. It may get difficult to maintain and stock the food cabinets if there are not enough people taking care of them, as it

costs money to stock the cabinets, build the cabinets, and maintain them if they break. This may get difficult for Mutual Aid Tompkins, as they are simply peers helping each other out and they do not get funding from a central organization.

5.3 Recommendations

As stated earlier, in order to make our design more scalable for mass deployment, it would be beneficial to find a different camera or lens adapter that has a wider field of view. This would allow for the camera system to be implemented in the three-shelf configuration of the cabinet, as all of the other existing cabinets currently have three shelves.

Another recommendation is to develop a more modular way to mount the cameras on the shelves. Our current design utilizes pieces of wood stacked together to level the camera and mount it onto the shelf with screws. However, this is not very reproducible, as it requires multiple components and could differ across cabinets. A potential solution to this problem could be to design a 3D-printed mount that fits into the hole drilled into the top of the shelves. This would be a relatively cost-effective and much less labor-intensive method at mounting the cameras. Moreover, it could be uniform for all cabinets.

Mass-producing a circuit board will make production and maintenance much easier. We encountered many connection issues when using a breadboard and converting to a more permanent board was essential to the consistent performance of the electronics inside the food cabinet.

APPENDIX A. Cabinet Images

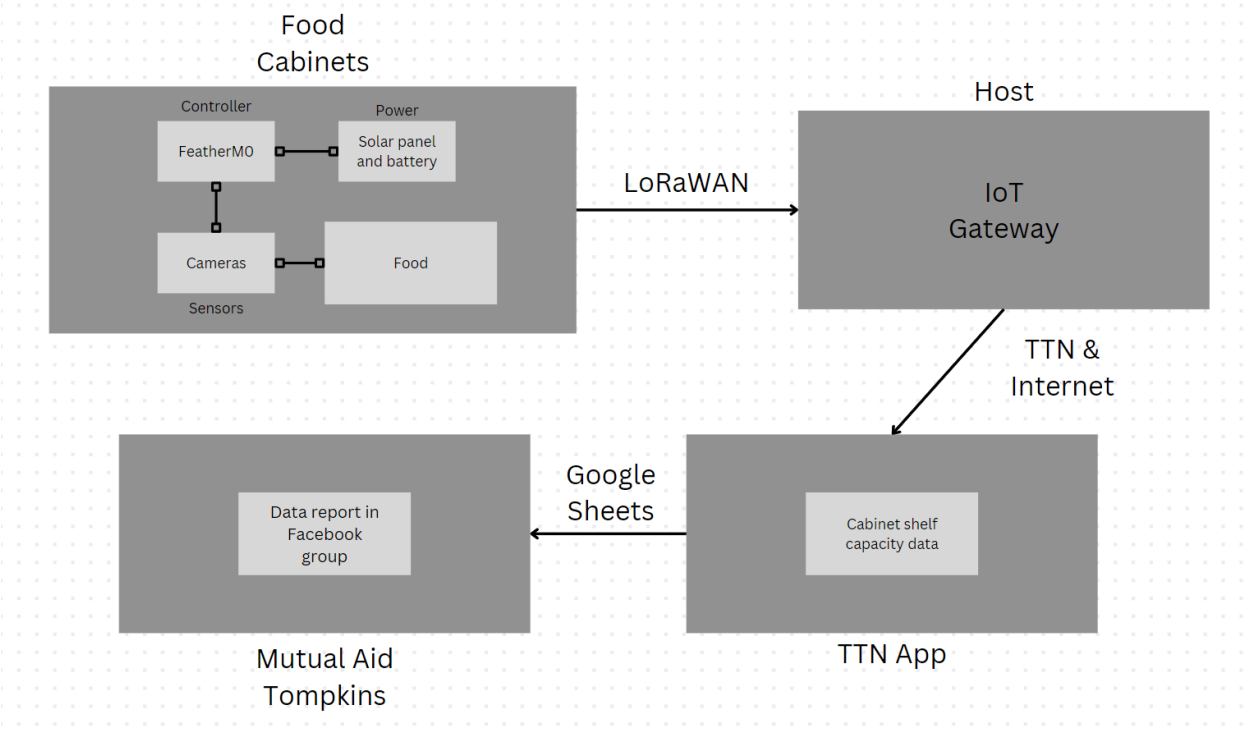


Figure 1. *IoT System Diagram.*

Figure 2. *Roof shingles and solar panel mount.*

Figure 3. *Two shelf configuration.*

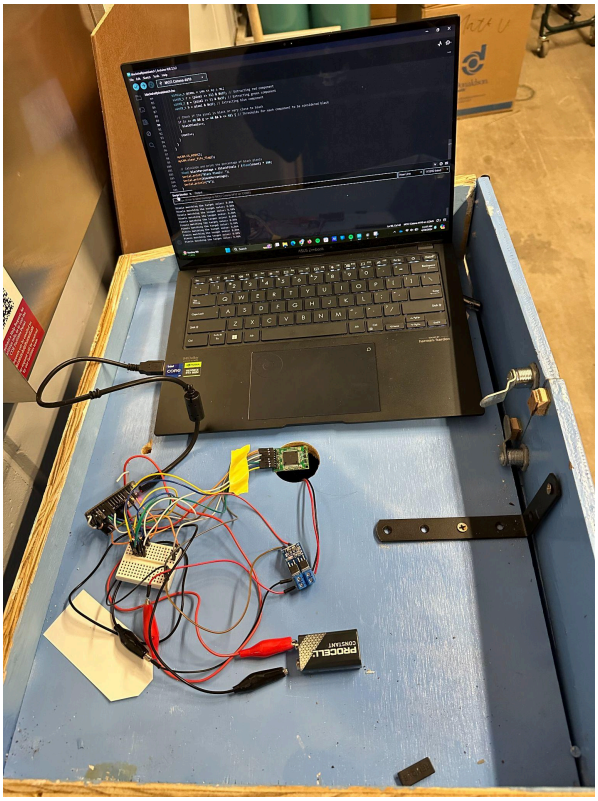


Figure 4. *Prototype setup.*

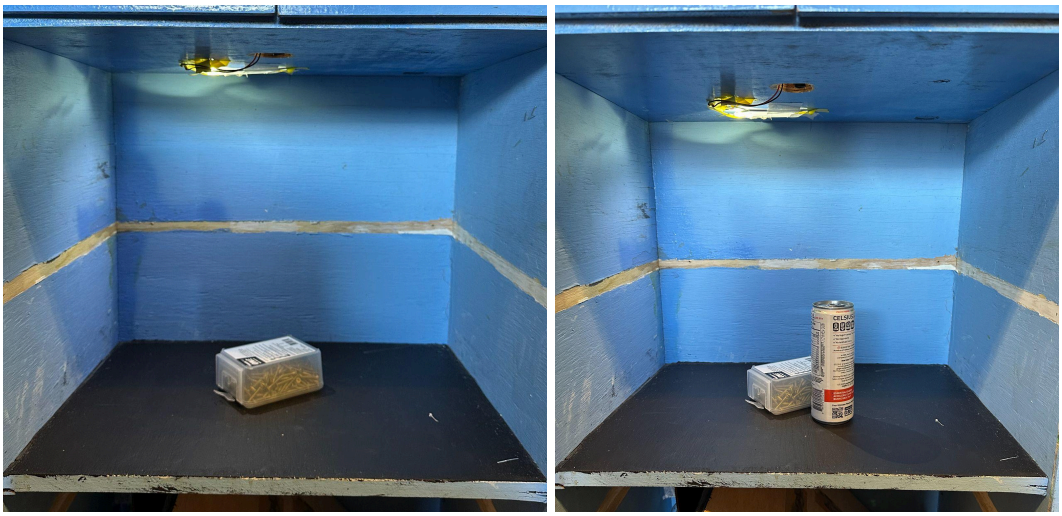


Figure 5. *Cabinet view of prototype setup.*

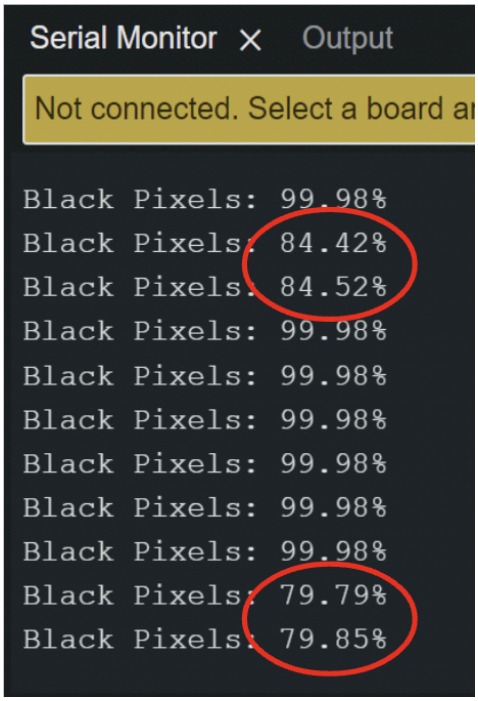


Figure 6. *Preliminary capacity results.*

***Figure 7**. Testing for Low (left), Ok (middle), and Good (right) food capacities.*

2024-05-15T20:55:50.865680355Z	4.917	1.750	3.333335876	Low
2024-05-15T20:59:19.735385783Z	27.792	32.125	29.95833206	Ok
2024-05-15T21:12:53.574761254Z	56.000	65.125	60.5625	Good

***Figure 8**. Google Sheet results for Low, Ok, and Good food capacities based on items in Figure 7.*

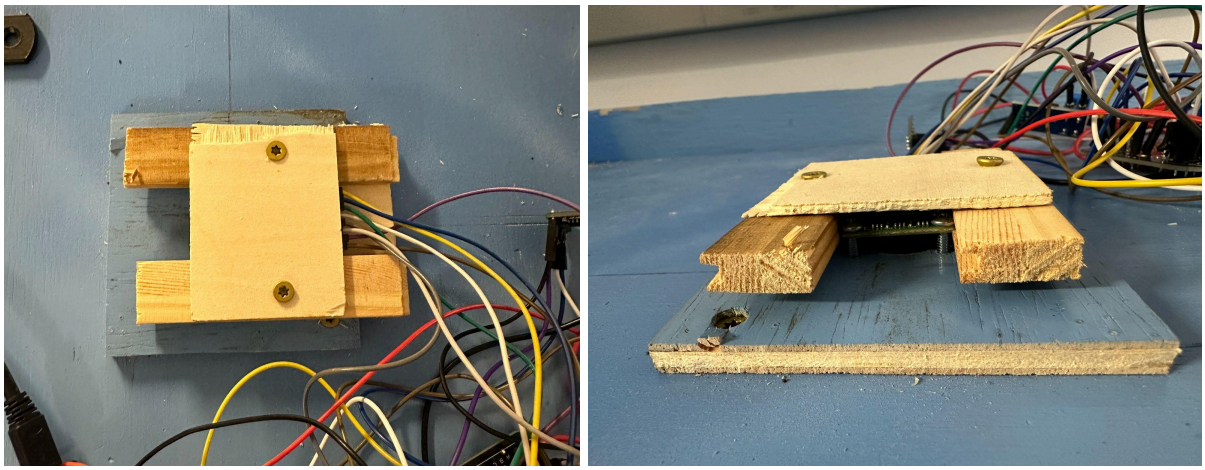


***Figure 9**. Shelf 1 camera fixture.*

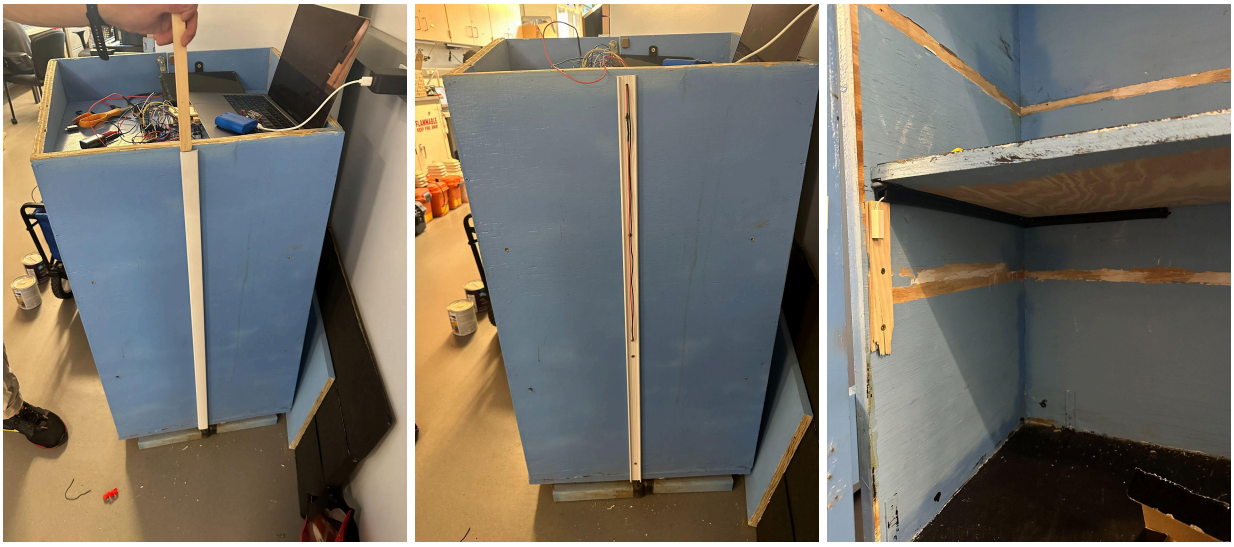


Figure 10. *Wire channels.*

Figure 11. *Magnetic door sensor.*

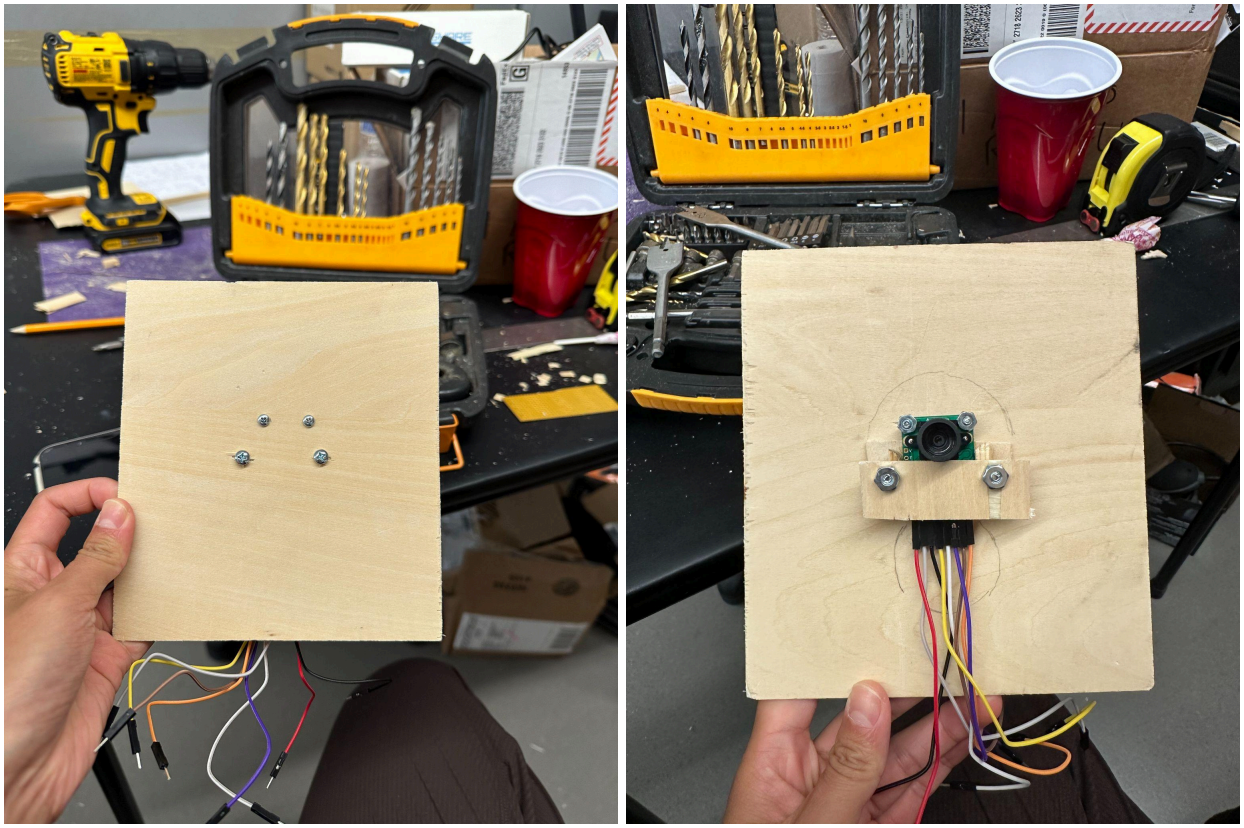


Figure 12. *Shelf 2 camera fixture.*

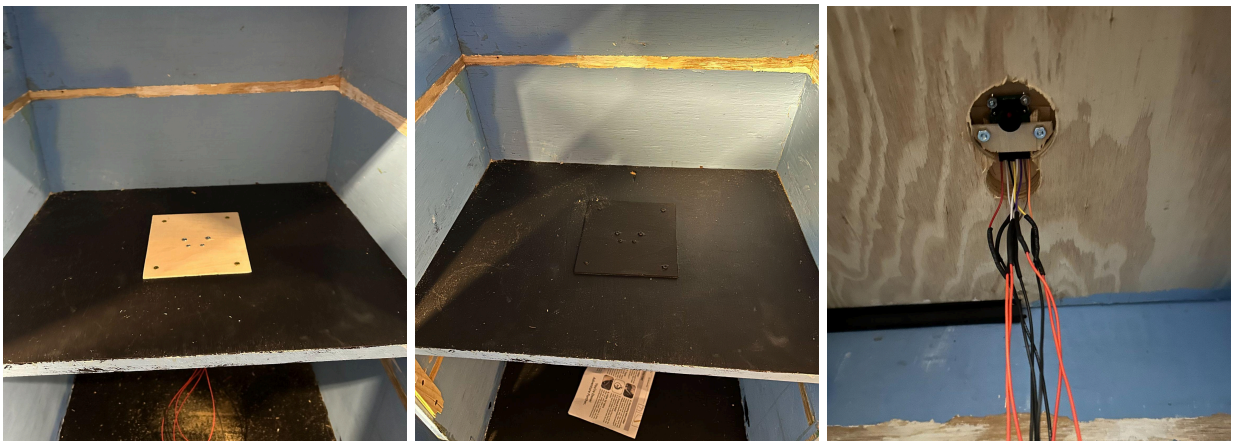


Figure 13. *Shelf 2 camera fixture mounting.*

Figure 14. *Food used during testing.*

APPENDIX B. Bill of Materials

Item	Quantity	Cost ($)	Total Cost ($)
1 Watt LEDs	2	0.75	7.49
SPI Cameras	2	25.99	51.98
Wire Channel	1	13	13
Treated Plywood	1	31.38	31.38
Magnetic Door Contact Sensor	1	3.95	3.95
Wire Channel 2	1	21	21
Shingle roll	1	27.98	27.98
½” Wood Screws	1 (pack)	3.98	3.98

¾” Wood Screws	1 (pack)	3.98	3.98
Matt Black Paint	1 (1 gal can)	19.93	19.93
Primer	1 (can)	17.43	17.43
Sandpaper	1 (sheet)	6.98	6.98
Paintbrush 3 Pack	1	1	11.98
		Total	**221.06**

APPENDIX C. Transmitting Data

C.1 TTN

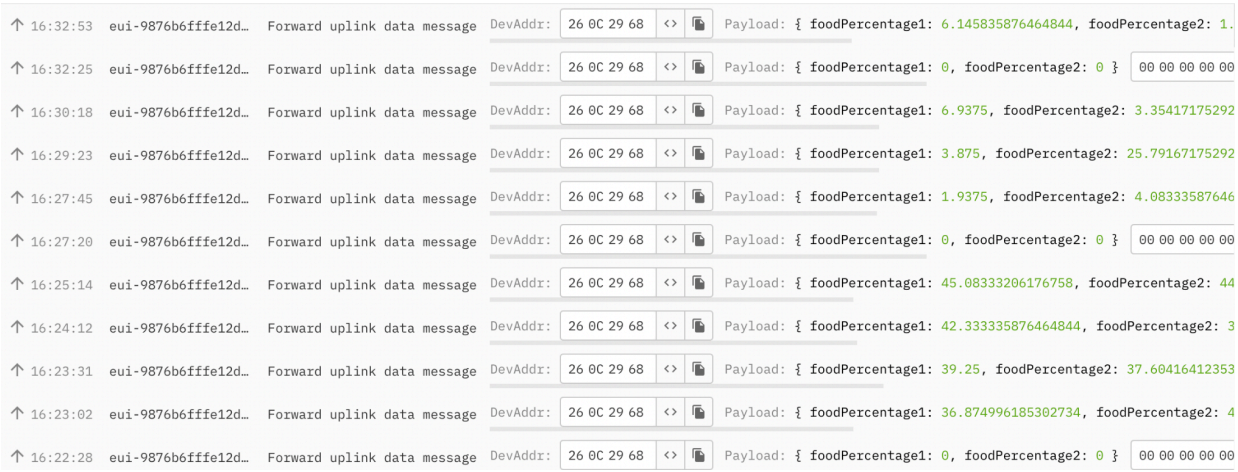


Figure 15. *Screenshot of data sent to TTN.*

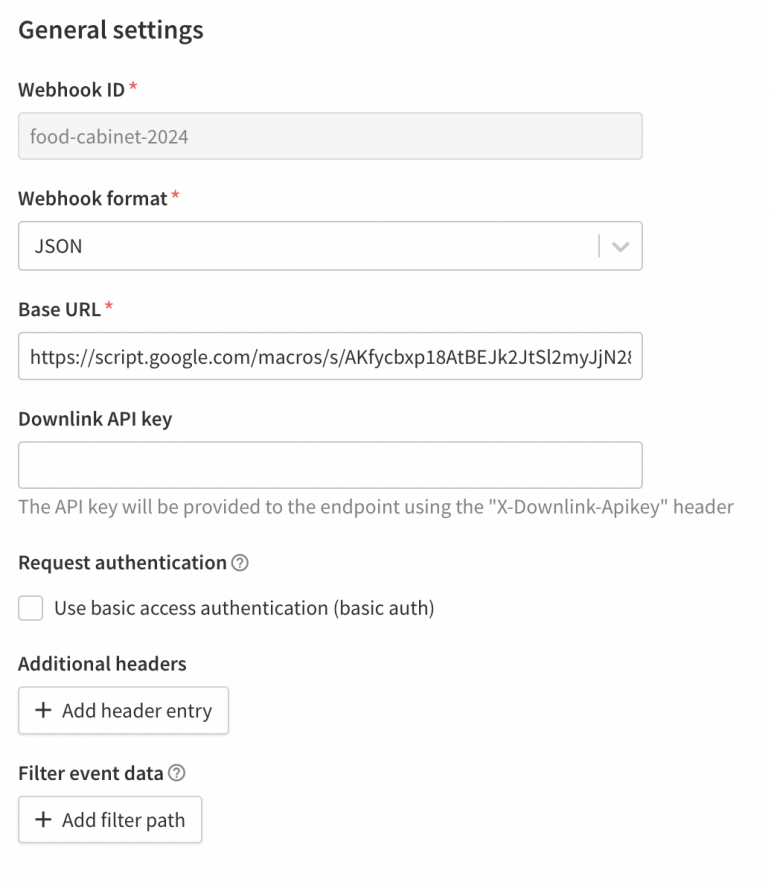
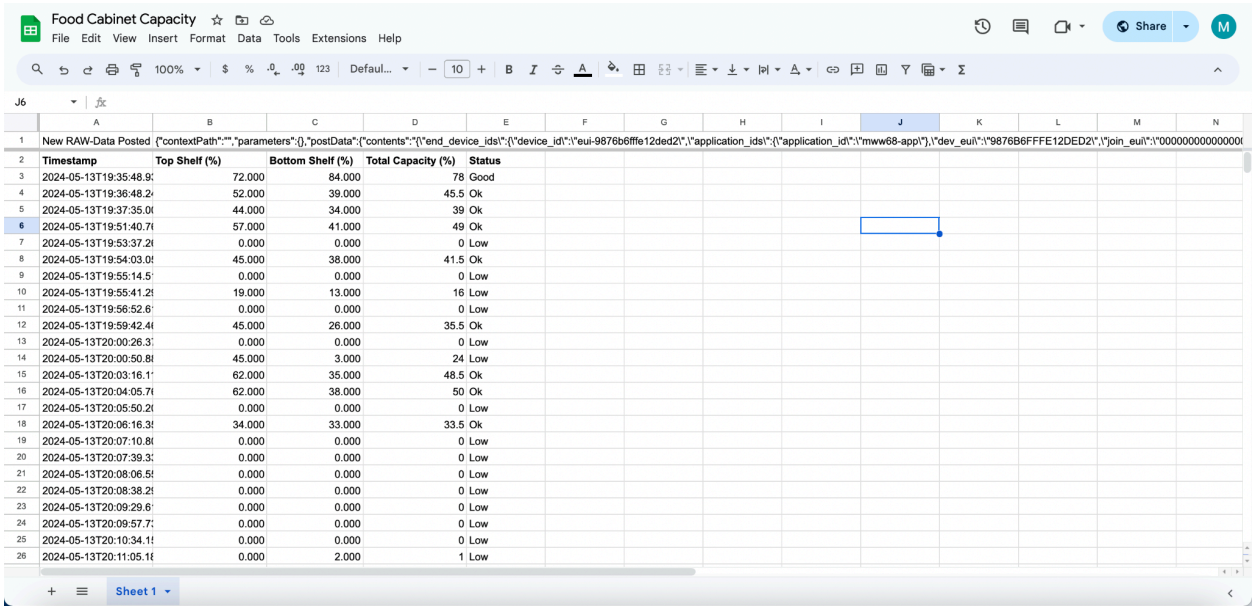


Figure 16. *TTN Webhook Integration.*

C.2 Google Sheet



	A	B	C	D	E
1	New RAW-Data Posted	{"contextPath":"","parameters":{},"postData":{"contents":"{\"end_device_ids\":{\"device_id\":\"eui-9876b6fffe12ded2\",\"application_ids\":{\"application_id\":\"mww68-app\"},\"dev_eui\":\"9876B6FFFE12DED2\",\"join_eui\":\"0000000000000000			
2	Timestamp	Top Shelf (%)	Bottom Shelf (%)	Total Capacity (%)	Status
3	2024-05-13T19:35:48.9	72.000	84.000	78	Good
4	2024-05-13T19:36:48.2	52.000	39.000	45.5	Ok
5	2024-05-13T19:37:35.0	44.000	34.000	39	Ok
6	2024-05-13T19:51:40.7	57.000	41.000	49	Ok
7	2024-05-13T19:53:37.2	0.000	0.000	0	Low
8	2024-05-13T19:54:03.0	45.000	38.000	41.5	Ok
9	2024-05-13T19:55:14.5	0.000	0.000	0	Low
10	2024-05-13T19:55:41.2	19.000	13.000	16	Low
11	2024-05-13T19:56:52.6	0.000	0.000	0	Low
12	2024-05-13T19:59:42.4	45.000	26.000	35.5	Ok
13	2024-05-13T20:00:26.3	0.000	0.000	0	Low
14	2024-05-13T20:00:50.8	45.000	3.000	24	Low
15	2024-05-13T20:03:16.1	62.000	35.000	48.5	Ok
16	2024-05-13T20:04:05.7	62.000	38.000	50	Ok
17	2024-05-13T20:05:50.2	0.000	0.000	0	Low
18	2024-05-13T20:06:16.3	34.000	33.000	33.5	Ok
19	2024-05-13T20:07:10.8	0.000	0.000	0	Low
20	2024-05-13T20:07:39.3	0.000	0.000	0	Low
21	2024-05-13T20:08:06.5	0.000	0.000	0	Low
22	2024-05-13T20:08:38.2	0.000	0.000	0	Low
23	2024-05-13T20:09:29.6	0.000	0.000	0	Low
24	2024-05-13T20:09:57.7	0.000	0.000	0	Low
25	2024-05-13T20:10:34.1	0.000	0.000	0	Low
26	2024-05-13T20:11:05.1	0.000	2.000	1	Low

Figure 17. *Screenshot of Google Sheet with capacity data recorded from TTN.*

Google Sheet Link:
https://docs.google.com/spreadsheets/d/1s6_laOgfUgFxFmQCI7TcXpX5-heR96bkFNSmK6Ip0xw/edit?usp=sharing

APPENDIX D. Code

D.1 Food Cabinet Code

The libraries used can be found at the top of Appendix D.1.1.
Link to code and libraries: https://github.com/JustinWei02/FoodCabinet

D.1.1 TwoCameraFinalCode.ino

This is the main code for running the camera and taking photos to calculate the percentage of food on the shelf, which is then sent to TTN.

```
#ifdef COMPILE_REGRESSION_TEST
#define FILLMEIN 0
#else
#define FILLMEIN (#Don't edit this stuff. Fill in the appropriate FILLMEIN values.)
#warning "You must fill in your keys with the right values from the TTN control panel"
#endif

#include "Catena_Fram32k.h"
#include <Arduino_LoRaWAN_ttn.h>
#include <lmic.h>
#include <hal/hal.h>
#include "keys.h"
#include <Wire.h>
#include <ArduCAM.h>
#include <SPI.h>
#include "memorysaver.h"
#include <ArduinoLowPower.h>

#define BMPIMAGEOFFSET 66

uint64_t lastTime = 0;
uint64_t timer = 0; //timer for when door is closed
uint32_t bufferLength = 8;
static uint8_t messageBuffer[8] = {0, 1, 2, 3, 4, 5, 6, 7};

struct __attribute__((__packed__)) Capacity{
 float foodPercentage1;
```

```
 float foodPercentage2;
};

Capacity myfoodAmount;

#ifdef __cplusplus
extern "C"{
#endif

void myStatusCallback(void * data, bool success){
 if(success)
   Serial.println("Succeeded!");
 else
   Serial.println("Failed!");
 }

#ifdef __cplusplus
}
#endif

class cMyLoRaWAN : public Arduino_LoRaWAN_ttn {

public:
   bool begin(const Arduino_LoRaWAN::lmic_pinmap& map);
   cMyLoRaWAN() {};
   using Super = Arduino_LoRaWAN_ttn;
   McciCatena::cFram32k theFram;

protected:
   // you'll need to provide implementations for each of the following.
   virtual bool GetOtaaProvisioningInfo(Arduino_LoRaWAN::OtaaProvisioningInfo*)
override;
   virtual void NetSaveSessionInfo(const SessionInfo &Info, const uint8_t *pExtraInfo,
size_t nExtraInfo) override;
   virtual void NetSaveSessionState(const SessionState &State) override;
   virtual bool NetGetSessionState(SessionState &State) override;
   virtual bool GetAbpProvisioningInfo(Arduino_LoRaWAN::AbpProvisioningInfo*)
override;

};

// set up the data structures.
```

```
cMyLoRaWAN myLoRaWAN {};

// The pinmap. This form is convenient if the LMIC library
// doesn't support your board and you don't want to add the
// configuration to the library (perhaps you're just testing).
// This pinmap matches the FeatherM0 LoRa. See the arduino-lmic
// docs for more info on how to set this up.
const cMyLoRaWAN::lmic_pinmap myPinMap = {
    .nss = 8,
    .rxtx = cMyLoRaWAN::lmic_pinmap::LMIC_UNUSED_PIN,
    .rst = 4,
    .dio = { 3, 6, cMyLoRaWAN::lmic_pinmap::LMIC_UNUSED_PIN },
    .rxtx_rx_active = 0,
    .rssi_cal = 0,
    .spi_freq = 8000000,
};

// Ensure the correct camera module is defined in your memorysaver.h
#if !(defined OV2640_MINI_2MP)
#error "Camera module not supported!"
#endif

// Define CS pins for two cameras
const int CS_CAM1 = 12; // CS pin for camera 1
const int CS_CAM2 = 10; // CS pin for camera 2
const int doorPin = 11; // pin for magnetic door sensor

ArduCAM myCAM1(OV2640, CS_CAM1); // Initialize camera 1
ArduCAM myCAM2(OV2640, CS_CAM2); // Initialize camera 2

int pwmPin = 13;

bool doorChanged = false; // Flag to track door state change
unsigned long doorChangeTime = 0; // Time of last door state change
const unsigned long doorTimeout = 10000; // Timeout in milliseconds (1 minute)
bool doorOpen=false; //is the door open

void setup() {
 Wire.begin();
  Serial.begin(115200);
 while (!Serial); // Wait for Serial port to connect.
```

```
pinMode(8, OUTPUT);
digitalWrite(8, HIGH);

pinMode(CS_CAM1, OUTPUT);
digitalWrite(CS_CAM1, HIGH);
pinMode(CS_CAM2, OUTPUT);
digitalWrite(CS_CAM2, HIGH);

pinMode(pwmPin, OUTPUT);

SPI.begin();
myCAM1.write_reg(ARDUCHIP_TEST1, 0x55);
myCAM2.write_reg(ARDUCHIP_TEST1, 0x55);

uint8_t temp1 = myCAM1.read_reg(ARDUCHIP_TEST1);
uint8_t temp2 = myCAM2.read_reg(ARDUCHIP_TEST1);

if (temp1 != 0x55 || temp2 != 0x55) {
  Serial.println("SPI Error with cameras!");
  while (1);
}

Serial.println("SPI Success with cameras");

Serial.begin(115200);
 {
    uint64_t lt = millis();
  while(!Serial && millis() - lt < 5000);
  }
  myLoRaWAN.begin(myPinMap);
  lastTime = millis();
  Serial.println("Serial begin");

  if(myLoRaWAN.IsProvisioned())
    Serial.println("Provisioned for something");
  else
    Serial.println("Not provisioned.");



  //Load data into the data structure.
  float foodPercentage1 = 0;
```

```
   float foodPercentage2 = 0;
   memcpy(&myfoodAmount.foodPercentage1, &foodPercentage1, 4);
   memcpy(&myfoodAmount.foodPercentage2, &foodPercentage2, 4);
   myLoRaWAN.SendBuffer((uint8_t *) &myfoodAmount, sizeof(myfoodAmount),
myStatusCallback, NULL, false, 1);

 // Initialize cameras
 myCAM1.set_format(BMP);
 myCAM1.InitCAM();
 myCAM1.OV2640_set_JPEG_size(OV2640_160x120);
 myCAM1.clear_fifo_flag();

 myCAM2.set_format(BMP);
 myCAM2.InitCAM();
 myCAM2.OV2640_set_JPEG_size(OV2640_160x120);
 myCAM2.clear_fifo_flag();

 pinMode(doorPin, INPUT_PULLUP); // Set door status pin as input with pull-up resistor
 LowPower.attachInterruptWakeup(doorPin, doorStateChanged, CHANGE); // Attach wakeup
interrupt to door status change

 Serial.println("System going to sleep...");
 LowPower.sleep();
 delay(100); // Give some time for serial to print
}

void loop() {
  // Check if door state changed
 myLoRaWAN.loop();

 if (doorChanged) {
   doorChanged = false;
   timer = 0; // restart the timer each time the door is opened or closed
   if (digitalRead(doorPin)==0){
     doorOpen=false;
     Serial.println("door is closed");
     timer = millis(); // start timer to track how long it's been since the door
closed
     }
   else{
     doorOpen=true;
     Serial.println("door is open");
```

```
    }
}
 // Only takes picture if it's been 10 seconds after door closed
if (timer != 0 && millis() - timer > 10000){
  // If the door is closed retrieve image and calculate black pixel
  if (!doorOpen) {
    //First Camera Image Capture
    Serial.println("Capturing Image1");

    analogWrite(pwmPin, 10); // Example: Turn on a device
    delay(5000); // Wait 5 seconds for the device to stabilize

    myCAM1.flush_fifo();
    myCAM1.clear_fifo_flag();
    myCAM1.start_capture();

    while (!myCAM1.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK));
    uint32_t length1 = myCAM1.read_fifo_length();
    if (length1 > MAX_FIFO_SIZE || length1 == 0) {
      myCAM1.clear_fifo_flag();
      return;
    }

    // Read BMP data
    myCAM1.CS_LOW();
    myCAM1.set_fifo_burst();
    uint8_t VH1, VL1;
    uint32_t count1 = 0, blackPixels1 = 0, totalPixels1 = 0;


    // Assuming BMP data begins after a specific header length
    for (int i = 0; i < BMPIMAGEOFFSET; i++) SPI.transfer(0x00); // Skip BMP header

    // Read every fourth pixel to simulate downscaling
    for (int i = 0; i < 160; i++) {
      for (int j = 0; j < 120; j++) {
        VH1 = SPI.transfer(0x00); // Higher byte
        VL1 = SPI.transfer(0x00); // Lower byte

        if ((totalPixels1++ % 4) == 0) { // Sample every fourth pixel
          // Convert two bytes to a single 16-bit pixel value (RGB565)
          uint16_t pixel = (VH1 << 8) | VL1;
```

```
            uint8_t r = (pixel >> 11) & 0x1F; // Extracting red component
            uint8_t g = (pixel >> 5) & 0x3F; // Extracting green component
            uint8_t b = pixel & 0x1F; // Extracting blue component


            // Check if the pixel is black or very close to black
            if (r <= 18 && g <= 36 && b <= 18) { // Thresholds for each component to be
considered black
              blackPixels1++;
            }
            count1++;
          }
        }
      }

      myCAM1.CS_HIGH();
      myCAM1.clear_fifo_flag();

      // Calculate and print the percentage of black pixels
      float blackPercentage1 = (blackPixels1 / (float)count1) * 100;
      Serial.print("Black Pixels1: ");
      Serial.print(blackPercentage1);
      Serial.println("%");


      analogWrite(pwmPin, 0); // Example: Turn off the device

      float foodPercentage1 = 100 - blackPercentage1;
      memcpy(&myfoodAmount.foodPercentage1, &foodPercentage1, 4);
      //myLoRaWAN.SendBuffer((uint8_t *) &myfoodAmount, sizeof(myfoodAmount),
myStatusCallback, NULL, false, 1);

      //Second Camera image capture
      delay(3000);
      Serial.println("Capturing Image2");

      analogWrite(pwmPin, 10); // Example: Turn on a device
      delay(5000); // Wait 5 seconds for the device to stabilize

      myCAM2.flush_fifo();
      myCAM2.clear_fifo_flag();
      myCAM2.start_capture();
```

```
  while (!myCAM2.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK));
  uint32_t length2 = myCAM2.read_fifo_length();
  if (length2 > MAX_FIFO_SIZE || length2 == 0) {
    myCAM2.clear_fifo_flag();
    return;
  }

  // Read BMP data
  myCAM2.CS_LOW();
  myCAM2.set_fifo_burst();
  uint8_t VH2, VL2;
  uint32_t count2 = 0, blackPixels2 = 0, totalPixels2 = 0;

  // Assuming BMP data begins after a specific header length
  for (int i = 0; i < BMPIMAGEOFFSET; i++) SPI.transfer(0x00); // Skip BMP header


  // Read every fourth pixel to simulate downscaling
  for (int i = 0; i < 160; i++) {
    for (int j = 0; j < 120; j++) {
      VH2 = SPI.transfer(0x00); // Higher byte
      VL2 = SPI.transfer(0x00); // Lower byte


      if ((totalPixels2++ % 4) == 0) { // Sample every fourth pixel
        // Convert two bytes to a single 16-bit pixel value (RGB565)
        uint16_t pixel = (VH2 << 8) | VL2;
        uint8_t r = (pixel >> 11) & 0x1F; // Extracting red component
        uint8_t g = (pixel >> 5) & 0x3F; // Extracting green component
        uint8_t b = pixel & 0x1F; // Extracting blue component


        // Check if the pixel is black or very close to black
        if (r <= 18 && g <= 36 && b <= 18) { // Thresholds for each component to be
considered black
          blackPixels2++;
        }
        count2++;
      }
    }
  }
```

```
      myCAM2.CS_HIGH();
      myCAM2.clear_fifo_flag();

      // Calculate and print the percentage of black pixels
      float blackPercentage2 = (blackPixels2 / (float)count2) * 100;
      Serial.print("Black Pixels2: ");
      Serial.print(blackPercentage2);
      Serial.println("%");

      analogWrite(pwmPin, 0); // Example: Turn off the device


      float foodPercentage2 = 100 - blackPercentage2;

      memcpy(&myfoodAmount.foodPercentage2, &foodPercentage2, 4);
      myLoRaWAN.SendBuffer((uint8_t *) &myfoodAmount, sizeof(myfoodAmount),
myStatusCallback, NULL, false, 1);
    }
    timer = 0; // reset timer

    Serial.println("System going to sleep...");
    LowPower.sleep();
  }
}

void doorStateChanged() {
 doorChanged = true;
 doorChangeTime = millis();
}

bool cMyLoRaWAN::begin(const Arduino_LoRaWAN::lmic_pinmap&  map){
    if(!theFram.begin()){
       Serial.println("Fram begin fail");
     }
     if(
     !theFram.initialize()){
       Serial.println("Fram not valid");
     }
     if (!Super::begin(map))
       return false;
     return true;
```

```
            }

bool
cMyLoRaWAN::GetOtaaProvisioningInfo(
   OtaaProvisioningInfo *pInfo
   ) {
     if (pInfo){
       memcpy_P(pInfo->AppEUI, APPEUI, 8);
       memcpy_P(pInfo->DevEUI, DEVEUI, 8);
       memcpy_P(pInfo->AppKey, APPKEY, 16);
     }
   return true;
}

void
cMyLoRaWAN::NetSaveSessionInfo(
   const SessionInfo &Info,
   const uint8_t *pExtraInfo,
   size_t nExtraInfo
   ) {
   // save Info somewhere.
   theFram.saveField(McciCatena::cFramStorage::kDevAddr, Info.V2.DevAddr);
   theFram.saveField(McciCatena::cFramStorage::kNetID, Info.V2.NetID);
   theFram.saveField(McciCatena::cFramStorage::kNwkSKey, Info.V2.NwkSKey);
   theFram.saveField(McciCatena::cFramStorage::kAppSKey, Info.V2.AppSKey);
}

void
cMyLoRaWAN::NetSaveSessionState(const SessionState &State) {
   // save State somwwhere. Note that it's often the same;
   // often only the frame counters change.
   theFram.saveField(McciCatena::cFramStorage::kLmicSessionState, State);
}

bool
cMyLoRaWAN::NetGetSessionState(SessionState &State) {
   // either fetch SessionState from somewhere and return true or...
   return theFram.getField(McciCatena::cFramStorage::kLmicSessionState, State);
}

bool
cMyLoRaWAN::GetAbpProvisioningInfo(Arduino_LoRaWAN::AbpProvisioningInfo* Info){
```

```
 //either get ABP provisioning info from somewhere and return true or...
 if (!Info) return false;//Library calls with null pointer sometimes
  SessionState temporaryState;
  if (!theFram.getField(McciCatena::cFramStorage::kLmicSessionState, temporaryState))
return false;
  if (!theFram.getField(McciCatena::cFramStorage::kNetID, Info->NetID)) return false;
  if (!theFram.getField(McciCatena::cFramStorage::kNwkSKey, Info->NwkSKey)) return
false;
  if (!theFram.getField(McciCatena::cFramStorage::kAppSKey, Info->AppSKey)) return
false;
  if (!theFram.getField(McciCatena::cFramStorage::kDevAddr, Info->DevAddr)) return
false;
  //Use the temporary SessionState to get important values from it.

  Info->FCntUp = temporaryState.V1.FCntUp;
  Info->FCntDown = temporaryState.V1.FCntDown;

  return true;
}
```

D.1.2 Catena_Fram32k.cpp

```
/* Catena_Fram2k.cpp  Fri Mar 17 2017 19:32:14 tmm */

/*

Module:  Catena_Fram2k.cpp

Function:
 The persistent-storage provider for the Fram on Catena

Version:
 V0.5.0  Fri Mar 17 2017 19:32:14 tmm  Edit level 1

Copyright notice:
 This file copyright (C) 2017 by

   MCCI Corporation
   3520 Krums Corners Road
   Ithaca, NY  14850

 An unpublished work.  All rights reserved.
```

```
 This file is proprietary information, and may not be disclosed or
 copied without the prior permission of MCCI Corporation.

Author:
 Terry Moore, MCCI Corporation March 2017

Revision history:
  0.5.0  Fri Mar 17 2017 19:32:14  tmm
 Module created.

*/

#include "Catena_Fram32k.h"

#include "Catena_FramStorage.h"
//#include "Catena_Log.h"

using namespace McciCatena;

/****************************************************************************\
|
| read-only data
|
\****************************************************************************/


/****************************************************************************\
|
| the methods
|
\****************************************************************************/

/*

Name: McciCatena::cFram2k::begin()

Function:
 Prepare to use the FRAM

Definition:
 public: virtual bool
```

```
   McciCatena::cFram2k::begin()

Description:
 Initialize an cFram2K object prior to operation.

Returns:
 true for success, false for failure.

*/

/* virtual public */
bool
McciCatena::cFram32k::begin()
 {
       if (! this->Super::begin() ||
     ! this->m_hw.begin(0, &Wire))
   return false;

 return true;
 }
/*

Name: McciCatena::cFram2K::read()

Function:
 Read a string of bytes from the FRAM

Definition:
 public: virtual size_t
   McciCatena::cFram2k::read(
     cFramStorage::Offset uOffset,
     uint8_t *pBuffer,
     size_t nBuffer
     ) override;

Description:
 nBuffer bytes are read from the FRAM device, starting at uOffset.

Returns:
 number of bytes read.

*/
```

```
size_t
MccCatena::cFram32k::read(
 cFramStorage::Offset uOffset,
 uint8_t *pBuffer,
 size_t nBuffer
 )
 {
 return this->m_hw.read(uOffset, pBuffer, nBuffer);
 }
/*

Name: MccCatena::cFram2K::write()

Function:
 Write a sequence of bytes to the FRAM.

Definition:
 public: virtual bool
   MccCatena::cFram2k::write(
     cFramStorage::Offset uOffset,
     const uint8_t *pBuffer,
     size_t nBuffer
     ) override;

Description:
 The sequence of bytes at [pBuffer, pBuffer+nBuffer) is written
 to the FRAM at offset uOffset.

Returns:
 true for success.

Notes:
       At the moment, the lower level write can't return failure status,
       so this routine always returns true.
*/

bool
MccCatena::cFram32k::write(
 cFramStorage::Offset uOffset,
 const uint8_t *pBuffer,
 size_t nBuffer
```

```
 )
 {
 this->m_hw.write(uOffset, pBuffer, nBuffer);

       return true;
 }
```

D.1.3 Catena_Fram32k.h

```
/* Catena_Fram2k.h  Sun Mar 12 2017 17:47:52 tmm */

/*

Module:  Catena_Fram2k.h

Function:
 class McciCatena::cFram2k

Version:
 V0.5.0  Sun Mar 12 2017 17:47:52 tmm  Edit level 1

Copyright notice:
 This file copyright (C) 2017 by

   MCCI Corporation
   3520 Krums Corners Road
   Ithaca, NY  14850

 An unpublished work.  All rights reserved.

 This file is proprietary information, and may not be disclosed or
 copied without the prior permission of MCCI Corporation.

Author:
 Terry Moore, MCCI Corporation March 2017

Revision history:
  0.5.0  Sun Mar 12 2017 17:47:52  tmm
 Module created.

*/

#ifndef _CATENA_FRAM32K_H_    /* prevent multiple includes */
```

```
#define _CATENA_FRAM32K_H_

#pragma once

#ifndef _CATENA_FRAM_H_
# include "Catena_Fram.h"
#endif

#ifndef _MCCI_FRAM_I2C_H_
//# include <MCCI_FRAM_I2C.h>
#include "Ece4950_Mcci_Fram.h"
#endif

/****************************************************************************\
|
| The contents
|
\****************************************************************************/


namespace McciCatena {


class cFram32k : public cFram
 {
protected:
 using Super = cFram;

public:
 cFram32k() { this->m_fReady = false; };
 virtual ~cFram32k() {};

       // begin working with the FRAM
 virtual bool begin() override;

       // read from the store
       virtual size_t read(
               cFramStorage::Offset uOffset, uint8_t *pBuffer, size_t nBuffer
               ) override;

       // write to the store
       virtual bool write(
```

```
                cFramStorage::Offset uOffset, const uint8_t *pBuffer, size_t nBuffer
                ) override;

        virtual cFramStorage::Offset getsize() const override
                {
                return 32768;
                };


protected:
private:
 MCCI_FRAM_I2C m_hw;
 };

}; // namespace McciCatena

/**** end of Catena_Fram2k.h ****/
#endif /* _CATENA_FRAM2K_H_ */
```

D.1.4 Ece4950_Mcci_Fram.cpp

```
/**************************************************************************/
/*!
   @file     G_MCCI_FRAM_I2C.cpp
   @author   KTOWN (Adafruit Industries)
   @license  BSD (see license.txt)

   Driver for the Adafruit I2C FRAM breakout.

   Adafruit invests time and resources providing this open source code,
   please support Adafruit and open-source hardware by purchasing
   products from Adafruit!

   @section  HISTORY

   v1.0 - First release
   v1.1 - Adapt for FM24CL16B

*/
/**************************************************************************/
//#include <avr/pgmspace.h>
//#include <util/delay.h>
#include <stdlib.h>
```

```
#include <math.h>

#include "Ece4950_Mcci_Fram.h"

/*========================================================================*/
/*                            CONSTRUCTORS                                */
/*========================================================================*/

/**************************************************************************/
/*!
   Constructor
*/
/**************************************************************************/
G_MCCI_FRAM_I2C::G_MCCI_FRAM_I2C(void)
{
 this->m_framInitialized = false;
}

void G_MCCI_FRAM_I2C::prepIO(void) const
 {
 // this->m_pWire->setClock(1000000);
 }

uint8_t G_MCCI_FRAM_I2C::getI2cAddr(uint16_t framAddr) const
 {
 //return this->m_i2c_addr + ((framAddr >> 8) & 0x7);
 return this->m_i2c_addr;
 }

/*========================================================================*/
/*                           PUBLIC FUNCTIONS                             */
/*========================================================================*/

/**************************************************************************/
/*!
   Initializes I2C and configures the chip (call this function before
   doing anything else)
*/
/**************************************************************************/
boolean G_MCCI_FRAM_I2C::begin(uint8_t addr, TwoWire *pWire)
{
 /* scrub and save the address */
```

```
 if (addr == 0) // address of 0 is never valid on i2c.
 addr = MB85RC::kDefaultAddress;

 this->m_i2c_addr = addr & ~0x7;
 this->m_pWire = pWire;

 pWire->begin();
  /* Make sure we're actually connected ... do a begin/end on each address */
 this->prepIO();

 pWire->beginTransmission(this->m_i2c_addr);
 uint8_t uError = pWire->endTransmission();

 if (uError != 0)
   // device didn't ack
   return false;

 /* Everything seems to be properly initialised and connected */
 this->m_framInitialized = true;

 return true;
}

/**************************************************************************/
/*!
   @brief  Writes a byte at the specific FRAM address

   @params[in] framAddr
               The address to write to in FRAM memory (reduced modulo 2k)
   @params[in] value
               The 8-bit value to write at framAddr
*/
/**************************************************************************/
void G_MCCI_FRAM_I2C::write8 (uint16_t framAddr, uint8_t value)
{
 const uint8_t i2c_addr = this->getI2cAddr(framAddr);

 this->prepIO();
 this->m_pWire->beginTransmission(i2c_addr);
 this->m_pWire->write(framAddr >>8);
 this->m_pWire->write(framAddr & 0xFF);
 this->m_pWire->write(value);
```

```
 this->m_pWire->endTransmission();
}

/**************************************************************************/
/*!
   @brief  Writes a buffer to the specific FRAM address

   @params[in] framAddr
               The address to write to in FRAM memory (reduced mod 2k)
   @params[in] pBuffer
   Address of data in memory.
   @params[in] nBuffer
               number of bytes to write.
*/
/**************************************************************************/
void G_MCCI_FRAM_I2C::write(
 uint16_t framAddr,
 const uint8_t *pBuffer,
 size_t nBuffer
 )
{
 const uint8_t i2c_addr = this->getI2cAddr(framAddr);

 this->prepIO();

 this->m_pWire->beginTransmission(i2c_addr);
 this->m_pWire->write(framAddr >>8);
 this->m_pWire->write(framAddr & 0xFF);
 this->m_pWire->write(pBuffer, nBuffer);
 this->m_pWire->endTransmission();
}

/**************************************************************************/
/*!
   @brief  Reads an 8 bit value from the specified FRAM address

   @params[in] i2cAddr
               The I2C address of the FRAM memory chip (1010+A2+A1+A0)
   @params[in] framAddr
               The 16-bit address to read from in FRAM memory

   @returns    The 8-bit value retrieved at framAddr
```

```
*/
/**************************************************************************/
uint8_t G_MCCI_FRAM_I2C::read8 (uint16_t framAddr)
{
 const uint8_t i2c_addr = this->getI2cAddr(framAddr);

 this->prepIO();
 this->m_pWire->beginTransmission(i2c_addr);
 this->m_pWire->write(framAddr >>8);
 this->m_pWire->write(framAddr & 0xFF);
 this->m_pWire->endTransmission();

 this->m_pWire->requestFrom(i2c_addr, (uint8_t)1);
 while (! this->m_pWire->available())
 /* loop */;

 return this->m_pWire->read();
}

/**************************************************************************/
/*!
   @brief  Reads a buffer from the specified FRAM address

   @params[in] framAddr
               The 16-bit address to read from in FRAM memory
   @params[in] pBuffer
   The base address of the buffer to be filled
   @params[in] nBuffer
   Number of bytes to read.

   @returns    The 8-bit value retrieved at framAddr
*/
/**************************************************************************/
uint8_t G_MCCI_FRAM_I2C::read (uint16_t framAddr, uint8_t *pBuffer, size_t nBuffer)
{
 uint8_t const i2c_addr = this->getI2cAddr(framAddr);

 this->prepIO();
 this->m_pWire->beginTransmission(i2c_addr);
 this->m_pWire->write(framAddr >>8);
 this->m_pWire->write(framAddr & 0xFF);
 this->m_pWire->endTransmission();
```

```
 // we can only read 255 bytes at a time.
 if (nBuffer > 0xFF)
 nBuffer = 0xFF;

 this->m_pWire->requestFrom(i2c_addr, (uint8_t) nBuffer);
 uint8_t const save_nBuffer = nBuffer;

 while (this->m_pWire->available() && nBuffer > 0)
 {
 *pBuffer++ = this->m_pWire->read();
 --nBuffer;
 }

 return save_nBuffer - nBuffer;
}

/*****************************************************************************/
/*!
   @brief  Reads the Manufacturer ID and the Product ID frm the IC

   @params[out]  manufacturerID
                 The 12-bit manufacturer ID (Fujitsu = 0x00A)
   @params[out]  productID
                 The memory density (bytes 11..8) and proprietary
                 Product ID fields (bytes 7..0). Should be 0x510 for
                 the MB85RC256V.
*/
/*****************************************************************************/
bool G_MCCI_FRAM_I2C::getDeviceID(DeviceInfo& Info)
{
 uint8_t a[3] = { 0, 0, 0 };
 uint8_t results;
  this->m_pWire->beginTransmission(MB85RC::kSlaveID >> 1);
 this->m_pWire->write(this->m_i2c_addr << 1);
 results = this->m_pWire->endTransmission(false);

 this->m_pWire->requestFrom(MB85RC::kSlaveID >> 1, 3);
 a[0] = this->m_pWire->read();
 a[1] = this->m_pWire->read();
 a[2] = this->m_pWire->read();
```

```
 /* Shift values to separate manuf and prod IDs */
 /* See p.10 of
http://www.fujitsu.com/downloads/MICRO/fsa/pdf/products/memory/fram/MB85RC256V-DS501-0
0017-3v0-E.pdf */
 Info.uMfg = (a[0] << 4) + (a[1]  >> 4);
 Info.uProduct = ((a[1] & 0x0F) << 8) + a[2];

 return true;
}
```

D.1.5 Ece4950_Mcci_Fram.h

```
/**************************************************************************/
/*!
   @file     MCCI_FRAM_I2C.h
   @author   KTOWN (Adafruit Industries)
   @author   Terry Moore (MCCI Corporation)

   @section LICENSE

   Software License Agreement (BSD License)

   Copyright (c) 2013, Adafruit Industries
   Portions copyright (c) 2017, MCCI Corporation
   All rights reserved.

   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met:
   1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
   2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
   3. Neither the name of the copyright holders nor the
   names of its contributors may be used to endorse or promote products
   derived from this software without specific prior written permission.

   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ''AS IS'' AND ANY
   EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
   WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
   DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY
   DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
   (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
```

```
   LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
   ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
   SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
/******************************************************************************/

#ifndef _G_MCCI_FRAM_I2C_H_
#define _G_MCCI_FRAM_I2C_H_

#pragma once

#if ARDUINO >= 100
#include <Arduino.h>
#else
#include <WProgram.h>
#endif

#include <Wire.h>
#include <cstdint>



class G_MCCI_FRAM_I2C {
public:
 G_MCCI_FRAM_I2C(void);

 // definitions for the MB85RC FRAM chip.
 class MB85RC
   {
 public:
   static constexpr std::uint8_t   kDefaultAddress = 0x50;
   static constexpr std::uint8_t   kSlaveID = 0xF8;
   };

 // create a version number for comparison
 static constexpr std::uint32_t
 makeVersion(
     std::uint8_t major, std::uint8_t minor, std::uint8_t patch, std::uint8_t local =
0
     )
     {
```

```
    return ((std::uint32_t)major << 24u) | ((std::uint32_t)minor << 16u) |
((std::uint32_t)patch << 8u) | (std::uint32_t)local;
    }

 // version of library, for use by clients in static_asserts -- set version by editing
here:
 static constexpr std::uint32_t getVersion() { return makeVersion(2,0,0,0); }

 // extract major number from version
 static constexpr std::uint8_t
 getMajor(std::uint32_t v)
    {
    return std::uint8_t(v >> 24u);
    }

 // extract minor number from version
 static constexpr std::uint8_t
 getMinor(std::uint32_t v)
    {
    return std::uint8_t(v >> 16u);
    }

 // extract patch number from version
 static constexpr std::uint8_t
 getPatch(std::uint32_t v)
    {
    return std::uint8_t(v >> 8u);
    }

 // extract local number from version
 static constexpr std::uint8_t
 getLocal(std::uint32_t v)
    {
    return std::uint8_t(v);
    }

 // set up and probe device
 boolean  begin(
   uint8_t addr = MB85RC::kDefaultAddress,
   TwoWire *pWire = &Wire
   );
```

```
 // write a single byte
 void     write8 (uint16_t framAddr, uint8_t value);

 // write a buffer
 void     write  (uint16_t framAddr, uint8_t const *pBuffer, size_t nBuffer);

 // read a single byte
 uint8_t  read8  (uint16_t framAddr);

 // read a buffer
 uint8_t   read   (uint16_t framAddr, uint8_t *pBuffer, size_t nBuffer);

 struct DeviceInfo
 {
 uint16_t uMfg;
 uint16_t uProduct;
 };

 bool getDeviceID(DeviceInfo& Info);

private:
 uint8_t m_i2c_addr;
 boolean m_framInitialized;
 TwoWire *m_pWire;

 void prepIO(void) const;
 uint8_t getI2cAddr(uint16_t framAddr) const;

};

typedef G_MCCI_FRAM_I2C MCCI_FRAM_I2C;

#endif
```

D.1.4 keys.h

```
#include "stdint.h"
#ifndef KEYS_H
#define KEYS_H
const uint8_t PROGMEM APPEUI[8] = {0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
const uint8_t PROGMEM DEVEUI[8]= {0xD2, 0xDE, 0x12, 0xFE, 0xFF, 0xB6, 0x76, 0x98};
const uint8_t PROGMEM APPKEY[16]= {0x5F, 0xCC, 0x10, 0x77, 0x31, 0x2D, 0xB8, 0x35,
0xFE, 0x7F, 0xC9, 0x39, 0xAA, 0x19, 0x47, 0xA4};
```

```
#endif
```

D.2 Decoder

```
function floatFromBytes(bytes) {
	// JavaScript bitwise operators yield a 32 bit integer, not a float.
	// Assume LSB (least significant byte first).
	var bits = bytes[3]<<24 | bytes[2]<<16 | bytes[1]<<8 | bytes[0];
	// >>> is unsigned right shift. Zeros are shifted in at all times from the left
	var sign = (bits>>>31 === 0) ? 1.0 : -1.0;
	var e = bits>>>23 & 0xff;
	// Can code explicit leading 0 with 0 exponent. Otherwise, there is a
	// leading 1 implied.
	// Since we have constructed our significand/mantissa like we constructed
	// our integers, it is a factor of 2^23 too large (decimal point all the way)
	// right.
	// So, we correct that when we build the final number.
	var m = (e === 0) ? (bits & 0x7fffff)<<1 : (bits & 0x7fffff) | 0x800000;
	var f = sign * m * Math.pow(2, e - 127-23);
	return f;
}

function decodeUplink(input) {
	//initialize an object to store output data
	var cap = {};
	cap.foodPercentage1 = floatFromBytes(input.bytes.slice(0,4));
	cap.foodPercentage2 = floatFromBytes(input.bytes.slice(4,8));
	//return the output data along with any errors or warnings
	return {
	data: cap,
	warnings: [],
	errors: []
	};
}
```

D.3 Google Apps Script [2]

```
//This Script is made for TTN V3 Stack.
// Idea is to export Data from TTN to a Google Script
//https://www.thethingsindustries.com/docs/getting-started/migrating/major-cha
nges/
function test(e){
```

```
 var sheet =
SpreadsheetApp.openById("1s6_laOgfUgFxFmQCI7TcXpX5-heR96bkFNSmK6Ip0xw");
 var firstSheet = sheet.getSheets()[0];
 sheet.setName("Food Cabinet Capacity");
}

function doPost(e) {
  Logger.log("I was called")
 if(typeof e !== 'undefined')
 Logger.log(e.parameter);

 var sheet =
SpreadsheetApp.openById("1s6_laOgfUgFxFmQCI7TcXpX5-heR96bkFNSmK6Ip0xw");
 var firstSheet = sheet.getSheets()[0];
 firstSheet.setName("Sheet 1");
 //1. Zeile//1. Line
 firstSheet.getRange(1, 1).setValue('New RAW-Data Posted from TTNetwork:');
 firstSheet.getRange(1, 2).setValue(JSON.stringify(e));
 firstSheet.getRange(2,1).setValue(['Timestamp']);
 firstSheet.getRange(2,2).setValue(['Top Shelf (%)']);
 firstSheet.getRange(2,3).setValue(['Bottom Shelf (%)']);
 firstSheet.getRange(2,4).setValue(['Total Capacity (%)']);
 firstSheet.getRange(2,5).setValue(['Status']);

  // 2. Zeile
 //firstSheet.appendRow([rawdata]);
 var text = JSON.stringify(e);
 if (text) {
   if (text.indexOf('contents') > -1) {
       var textcutted = text.slice(text.indexOf('contents')+11, text.length);
// cut everything useless to get only the payload data
       textcutted = textcutted.replace(/\\/g, "");//remove all "\" to generate
a proper Json that can be converted
```

```
   var textcutted2 = textcutted.slice(0, textcutted.indexOf('postData')-24);
// remove last letters to get a Json that can be converted
   var endetest = textcutted2.slice(textcutted2.length
-10,textcutted2.length);//just for debug: Shows end of Payload
   }
   else{
     firstSheet.appendRow(['there was' , 'no content', 'that could', 'be
extracted']);//in case something went wrong
     firstSheet.appendRow(['Text does not contain contents:' , text]);//print
text in case something went wrong
   }
 }
 else{
  firstSheet.appendRow(['Text that has been posted is empty:' , text]);//print
text in case something went wrong
 }
  var jsonData = JSON.parse(textcutted2); // parse the extracted Text in JSON
Converter
 if (jsonData.uplink_message.f_port < 5){
 var topShelf =
parseFloat(jsonData.uplink_message.decoded_payload.foodPercentage1,10);
 var bottomShelf =
parseFloat(jsonData.uplink_message.decoded_payload.foodPercentage2, 10);
 var totalCapacity = (topShelf + bottomShelf)/2;
 var status;

// define status of cabinet (above 75% = good, 25-75% = ok, below 25% = low)
 if (totalCapacity > 75){
   status = 'Good';
 }else if (totalCapacity < 25){
   status = 'Low';
 }else{
```

```
   status = 'Ok';
 }

 firstSheet.appendRow([jsonData.received_at, topShelf, bottomShelf,
totalCapacity, status]);
 }
 else {
  firstSheet.appendRow(['0' ,'0','0','NO PROPER Data sent']);
 }
  return ContentService.createTextOutput(JSON.stringify(e))
}

function doGet(e){
 test(e);
}

function myResponse() {

 var data = {
   "dev_id": "0123456",
   "port": 1,
   "confirmed": true, // choose if Message needs to be confirmed
   //"payload_raw" : "1QIDBA=="
   "payload_fields": {    // The JSON object to be encoded by your encoder
payload function
     "temp_aktuell": 35,
     "humidity": 20,
     "temp_max": 50,
     "temp_min": -20,
     "condition_code": 500,
     "muell": 1
   }
 };
```

```
 // Payload: Temp[1B] / Humidity [1B] / Temp_max [1B] /Temp_min [1B] /
Condition_Code[2B] / Muell[1B]
 // A2 93 B1 6B F4 01 80 das war der Inhalt


 var options =
  {
    'method' : 'post',
    'contentType': 'application/json',
    'payload': JSON.stringify(data)
  };
  var response = UrlFetchApp.fetch("FILLMEIN", options);
 //var response = UrlFetchApp.fetch('http://www.google.com/');
 Logger.log(response.getContentText());
 var Test = "0"
}
```

RESOURCES

[1] https://map.feedingamerica.org/county/2020/overall/new-york/county/tompkins
[2] https://github.com/Uspizig/Ttn-goooogle-script/blob/master/src/ttnScriptV3Stack.js
[3] Food Cabinet 2023 Report
[4] https://mutualaidtompkins.com/
[5] Food packaging colors