IntelliStorage

Jason Kim, Siyuan Li, Yuma Matsuoka

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— For individuals who do not have enough time to organize their food storage space at home. While many commercial grocery-tracking systems exist, many require a subscription service, or are compatible with one store (Costco, Whole Foods). On top of this, rarely do these systems have a built-in system of keeping track of expiration dates. IntelliStorage provides a unified, convenient system that keeps track of groceries at home and suggests items to use before their expiration date.**

***Index Terms*— Optical Character Recognition, Universal Product Code Barcode, Distributed Consensus, Image Acquisition, openCV, pyTesseract, Raspberry Pi, RAID, Message Passing, User Interface**

I. Introduction

In this fast-paced world, efficient management of household tasks remains a significant challenge, particularly when it comes to keeping track of groceries—a task that is both time-consuming and often overlooked until it results in waste. IntelliStorage is an innovative solution designed to simplify the way individuals and families manage their groceries. Intellistorage aims to provide a convenient method for organizing and monitoring groceries at home, specifically targeting those who struggle with time management and keeping track of their groceries.

Intellistorage comprises a main module and several scanner modules, all interconnected via Wi-Fi. This setup allows users to scan items before storing them with important information, including product details and expiration dates, displayed on screen. This not only streamlines the organization process but also significantly reduces food waste by alerting users to upcoming expirations. Other technologies such as the Costco and Whole Foods application offer partial solutions by giving a list of items purchased. These technologies however fail to consider that customers shop at multiple stores. They also fail to consider the fact that these items are all made unique and have different expiration dates. IntelliStorage is able to fill these voids with an universal tracking tool for items and their expiration dates.

Our project's goals are to minimize the time and effort required for grocery management and to reduce food waste by providing timely reminders about expiration dates. Ultimately, IntelliStorage is not just about organizing groceries, it's about organizing lives, one scan at a time.

I. Use-Case Requirements

The following use-case requirements have been set to ensure our project meets the needs of its users. These requirements are necessary for delivering a seamless experience for the user.

A. Item Registration & Tracking

All modules within the IntelliStorage system must avoid the need for extensive wiring. This ensures ease of installation and flexibility in module placement around the home.

B. Rapid Information Processing

The system must be capable of storing information within the time window of scanning two items. This efficiency is crucial for users who are looking to save time while organizing their groceries.

C. Fast Display Response

Upon scanning, the item's information and expiration date must be displayed fast. This quick feedback is essential for a smooth and user-friendly experience, allowing for immediate action or corrections if necessary.

D. Daily Expiration Alerts

IntelliStorage must generate and provide users with a daily report of items nearing their expiration dates. This feature is vital for reducing food waste and ensuring users can consume or utilize products before they spoil.

E. Quick Setup

The initial setup process for the IntelliStorage system, including the main module and any scanner modules, should not exceed 5 minutes. This requirement addresses the user's need for a solution that is not only effective but also easy to implement.

F. Scanning

The scanner modules must have a wide scanning angle and a wide scanning distance range. These specifications ensure that users can easily scan items of various sizes and shapes without having to specially accommodate the item.

G. Touch Screen Interface

The display must be large enough to ensure the readability of displayed information. It must be touchscreen as other input devices, such as a keyboard or mouse, would take up too much space and inconvenience the user.

II. Architecture and/or Principle of Operation

Our design did not have any architectural changes from the design report. In order to meet the user requirements and operate efficiently, our system will consist of scanner modules

and a central module communicating with each other through Wi-Fi as depicted in Fig. 1.

The central module forms the core of IntelliStorage, equipped with a computational unit and a touchscreen display. Its primary function is to manage and store the database, maintaining it with the new data sent from scanner modules. This module will be responsible for analyzing the stored data to generate the daily recommendation of items to use.

Each scanner module has its computational unit and touchscreen display. In order to scan items in, it is also equipped with a barcode scanner to capture the barcode, a camera to capture the expiration date, and a temperature sensor to categorize the storage space's temperature. The barcode scanner and camera will be integrated together to efficiently capture both pieces of data at once. The temperature sensor will categorize the storage space as hot or cold as item health and expiration date are highly correlated with ambient temperature.

Fig. 2. breaks down the communication of data happening intra and inter modules. At the front-end of the system, item data will be parsed and stored locally on the nodes. A database look-up will happen for the item code to find out its name, and an image will acquire the expiration date. This set of data will also be displayed for the user to view and confirm. If the data looks wrong, the user also has the power to edit it. This data will be communicated to the back-end of the system, where the central computer will store it in a database. It will also take the aggregate of the item data and create a priority list of what items should be used. Many factors will be considered in this algorithm, from the purchase date, expiration date, to even the temperature of the storage unit.

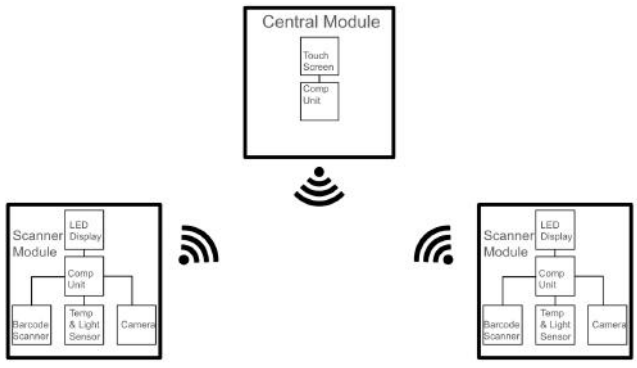

Fig. 1. Overall System Setup

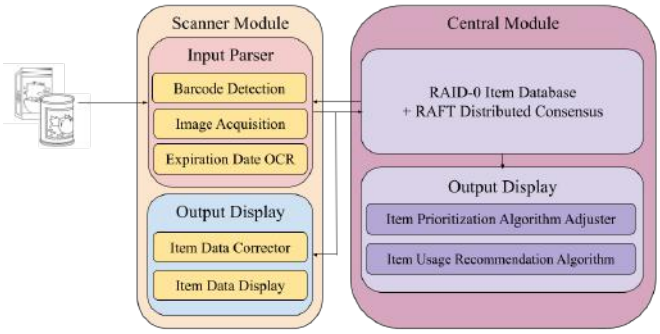

Fig. 2. Block Diagram of Data Flow

I. Design Requirements

A. Item Registration & Tracking

Our primary objective was to ensure precise data registration for users. We determined that the optimal scanning conditions included a 30-degree angle and a scanning distance of 15-25 cm, based on manual tests measuring typical barcode scanning distances and angles. Furthermore, we aimed for data acquisition accuracy of 98% from the barcode scanner and 90% from the camera. We decided on 90% accuracy because the state-of-the-art models that already exist for OCR (optical character recognition) perform with around 90-98% accuracy. Given the project's scope, achieving this minimum threshold could result in approximately 10% of items needing manual user correction.

Additionally, we aim for the data to be read and registered by the system within 4 seconds. This timeframe is based on the notion that users would want to see the item registered before they are ready to scan a new item. We simulated this motion and timed the duration it took to scan and grab another item, which was on average 4 seconds. This would also allow the user to correct any mistakes in input before the new item was grabbed.

B. Scaling

To fulfill our use case, we required our system to be scalable for each user's needs. By taking a survey of our member's pantries, we decided that roughly 40 unique items were in our pantry, which resulted in our requirement. As we are busy students with not enough time, this would fit the use case and our consumption needs are baseline. Additionally, we on average had 3 storage spaces per house, which led to the number of scanner module requirements. In order to have perceived zero lag between synchronizing data scanned on a local module with another module, we want at maximum 10 seconds for data synchronization. This is due to roughly the time it takes minimum to get from one storage space to another in a house.

Additionally, we do not want scanned data items to be lost.

Regardless of a scanner module or the central computer going down, we would like the system to preserve the data scanned for use. It would be inconvenient and would deter the user from using the system if all items had to be scanned after a module got disconnected. This is why one of the design requirements along with scaling is the preservation of data – data integrity and consistency between the modules.

C. *Usability*

Lastly, we wanted data to be stored and accessed easily for the user every time. We want to display item information within 500ms of the user's query, regardless of which machine it was first scanned on. Any longer would be substantial lag for multiple queries, frustrating the user and deterring them from the platform.

We also want a daily report of items to use, which can be customized by the user's preference. As something such as a recommendation is subjective, quantifying this as a requirement is quite difficult. However, we will want the algorithm to perform as expected in the edge case – if a feature is the only feature to be considered, the algorithm should return items in order with respect to that feature value, and that value only.

Additionally, we would like the setup of the system to not be a hurdle for the user. We thought that qualitatively, 5 minutes would allow for system complexity while not being too high of a hurdle for the user to use.

These design requirements are summarized in Fig. 3.

Requirement *A* (Item Registration & Tracking)	Requirement *B* (Scaling)	Requirement *C* (Usability)
>90% read-in accuracy	**40 items** per storage space	Store information within **1 sec** of scanning
30 degree scanning angle	**3 storage spaces** per network	Display info within **500 ms**
15-25 cm scanning distance	**10 sec** Synchronization	Daily report of expiring item
4 sec registration time	Data integrity	**5 min** setup node

Fig. 3. Summary of the Design Requirements

III. Design Trade Studies

A. *Camera Selection*

In our process of selecting a camera, we looked for one that was specialized at auto-focusing on medium-distance targets. This requirement stemmed off of some real-life testing as none of us had experience with camera technology. To better understand, we borrowed a few cameras from the inventory such as the TedGem 1080P WebCam. Although this camera worked for previous OCR projects [1], we realized soon enough that this did not fit our use case. For our 15-25 cm scanning distance use case, the captured images were blurry. As seen in Fig. 4, the characters were not detected even when put under Google Lens, a industry-level OCR algorithm as seen in [2]. We realized the cause was due to the camera's focus distance being 45cm, farther away than the use-case requirement. This was expected as it was a webcam, which is made to focus in on large details such as a face. Consequently, we didn't want a fixed focus camera as this would require the user to put the scanning object at a fixed distance from the camera. This would inhibit the barcode scanning process as different barcode sizes require different scanning distances.

Fig. 4. OCR Distance test with TedGem 1080P WebCam.

In order to meet the accessibility requirement, the camera is also very easy to install and attach to the module. Many alternatives such as the ArduCam line of products required RPi knowledge to assemble, something we could not guarantee the user had. It also needs to be inexpensive, as lowering the cost would lower the hurdle to entry for users. Therefore we decided that a connection of a USB would be the lowest barrier, as hardware installation would be simply inserting the USB into the port of the Raspberry Pi (RPi).

Ultimately, we settled on the NexiGo N60 1080P WebCam. Unlike the TedGem WebCam, it was able to auto-focus on targets reasonably, as seen in Fig. 5. Notably, Google Lens OCR was able to detect characters correctly in two of the three cases as well [2]. This, coupled with the low cost made it a logical choice for satisfying our requirements (Fig. 12).

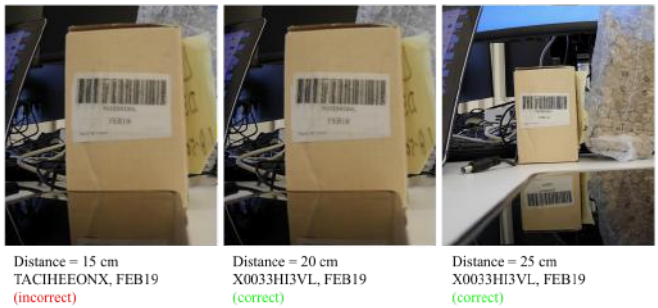

Fig. 5. OCR Distance test with NexiGo N60 1080P WebCam.

B. *Data Redundancy Type Selection*

In our process to guarantee data integrity of the system, we looked for a system that would allow for multiple node failures while guaranteeing a shared state across the multiple

nodes. We would want a system that agrees on a shared state as the state of all the storage space's items should be agreed upon. Even when one of the scanner modules fails, we would want this state to persist as it contains all of the scanned data. Users will be deterred if all the items in their house need to be scanned when one node gets disconnected from power. We also want restarted nodes to agree with this pre-established consensus.

An initial solution was having a central computer with sections of disk and memory partitioned off for each scanner module's items. If the system worked perfectly, it should hypothetically be tolerant to scanner module failures as the global state can be passed to the alive scanner module as the agreed state. However, if the central module fails and stays down, we cannot guarantee that an item registered on scanner A will have corrupt data when scanned on scanner B.

In order to mitigate loss from the central computer failing, we have decided to store parity blocks on each of the scanner modules along with each data block, similar to a RAID-5 setup. This is where data is striped across all blocks and a parity block for each row of blocks is scattered. A parity block aggregates `N` blocks across the `N` scanner modules in a fashion that allows a block to be recovered given the `N-1` other blocks. This process of creating and retrieving blocks from the parity block is called erasure coding.

If a node fails and one of the data blocks is effectively gone, we can recover it by erasure coding all the other blocks and the parity. This setup alone may suggest that a block will not be recovered if two nodes go down. However, with the existence of the central computer being one of the nodes, we can always guarantee that we will have enough updated blocks to retrieve the missing block. In summary, this setup is akin to a RAID-5 setup across each of the scanner nodes with a RAID-0 setup on the central computer.

C. Data Consistency Method

In order to keep the data on each of these modules to be consistent, we need a data consistency algorithm. Through discussion, it was decided that the RAFT consensus algorithm will be employed. In this algorithm developed by Diego Ongaro and John Ousterhout, a leader node decides a consensus for the follower nodes [3]. If a leader node goes down, then an election is held to elect a new leader. Data is sent on messages across nodes on an `epochTimer` interval, with the data field being empty if there's nothing as a heartbeat message.

This algorithm fits our case as we are able to customize many facets of the algorithm, such as the timer for message broadcasts. This is important as our use case differs from the normal case of distributed systems where there is constantly non-negligible workload. In this use-case, there are times when no items are scanned (no new data generated) and times when items are rapidly scanned into a storage space.

In order to minimize unnecessary actions while guaranteeing consistency, we have planned to customize the `epochTimer` to be 2 seconds during a scanning window and 1 minute otherwise. Due to the nature of the application, scanning modules will most likely be active for less than an hour per day. This means that during the remaining time, there is not much updated data to be communicated, making a setup with the epoch timer being 2 seconds to send 41,400 messages over the inactive 23 hours (1). This is meaningless as even if a node dies, a re-election and a leader needed for appending to the shared state is not needed immediately. This is unnecessary computation and wastes power. By reducing this threshold down to 1 minute, we are able to reduce this message count to 1,380, a far more reasonable number (1). Re-election in this case would only take 5 minutes after a node goes down, which is fine as there is no urgency when the system is not being used. If the system is needed, a scanner will be activated and the timer will be set to 2 seconds, reducing this time as well. Such customizations would not be possible on another distributed consensus algorithm such as Paxos or Byzantine Fault Tolerance. These other distributed consensus algorithms, especially Paxos also allow gaps in their commit logs, which would make it very difficult to debug correctly. Raft guarantees continuity on the logs, which would make it easier to trace through.

$$Messages\ Sent\ = (message\ rate)\ *\ (time) \qquad (1)$$

Another benefit of using RAFT is that it is very programmer-friendly, creating logs on scanner modules for debugging. This allows us to efficiently allocate time to other components of the system.

D. Database Format

There are two different formats for a database that could be used – a relational database and a key-value store. A relational database such as SQL would benefit from being able to easily find relations in a database, a positive for the recommendation algorithm. However, it is very hard to replicate a relational database across multiple modules. This is because the relations across sets of data need to be stored as well, complicating the distributed consensus protocol [4]. On the other hand, a key-value store is what the name implies, it stores a key and value. Although computation is required to find relations between the values or the keys, it offers an easier method of replication. Since the stored data is only keys and values, simply sending these values to other modules would suffice for establishing a shared state and dataset. Since data communication and state replication happens more often than an aggregation attempt for the recommendation algorithm, we decided to have a key-value store instead.

E. Camera Flashing Interval

The flashing interval for the camera was initially at 0.5 seconds. This was a pre-set value, as we believed during the planning phase that this would be sufficient. Through comprehensive testing, we realized that this interval could be reduced up to 0.1 seconds, with a tradeoff. Although the camera was able to capture more images with this reduced shutter speed, due to the delay of the OCR having to process

many more pictures, the faster speed was not effective. Therefore we chose an interval which was in between the two points and decided on 0.3 seconds, which took images every 0.3 seconds. This was also decided due to the design requirement, where we would like the time between scans to be less than 4 seconds. Having a flashing rate of 0.1 seconds resulted in processing time of minutes, and 0.5 seconds gave blurry pictures that resulted in no confidence for many of the items we tried to scan in. 0.3 seconds gave a good accuracy while having processing time around 3 seconds. This also ensured that the image buffer wasn't overloaded and overheated the computer, while also allowing for better quality of images and image processing.

IV. System Implementation

A. *Integrated Scanner Module*

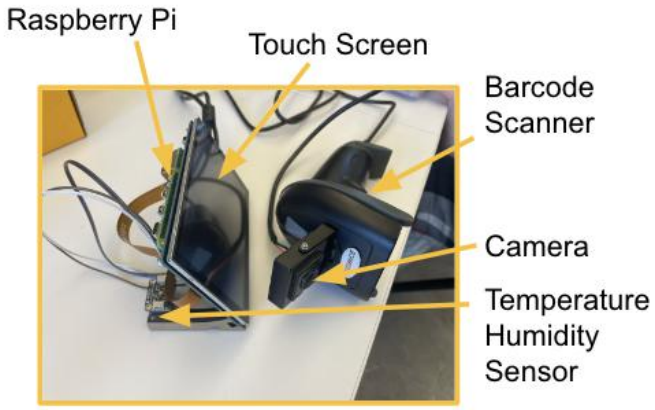

Fig. 6. Complete Integrated Scanner Module

Fig. 6. is the picture of the complete scanner module, which contains the barcode scanner, camera, temperature sensor, and the Raspberry Pi (RPi). The barcode scanner will act as the trigger for both the barcode and the camera. Although the frames captured by the camera around the trigger time will be used to find the expiration date, the camera will be capturing images continuously. Specifically, the frames captured by the camera will be stored in a 40 picture buffer that will continually be updated first-in first-out (FIFO) style. When the barcode scanner acquires a barcode, this buffer will be saved and OCR will be performed on each of these images. To improve the processing time, each picture will be spun off into its own thread for OCR processing. These results will be pooled together and the image with the highest confidence value will be used as the overall expiration date.

The mounted camera is an Arducam 4K 8MP IMX219 camera, which has the following specifications.

- 3280 x 2464 pixel resolution
- Diagonal=72 / Horizontal=60 / Vertical=47° FOV
- Focus Range = 40mm - ∞

Notably, this camera has a relatively high resolution with an optimal close-range focus ability. This camera is different from the NexiGo Webcam, which was mentioned in our design report and was planned to use. After testing, we found that its smallest focus distance was too large for our design. It could only focus at around 200mm or further, and it gave a photo without enough clarity for small texts. So we switched to the Arducam, which gives the smallest focus distance of 40mm, and it perfectly fits our needs.

The camera faces the same direction as the barcode scanner. The scanner will notify the user if it has scanned the barcode, and the expiration date as well with a different tone. It will be ensured that the user will not have any problems or discomfort in getting all of the information necessary for each scan. The RPi will be connected to a power source, and will not be on the scanner itself. This is to limit the movement of the computational unit and thus possible ways it could fail from damage. It is to be mounted on a wall, where the user will be able to interact with it via the touchscreen display. The temperature humidity sensor will be connected to the RPi via GPIO pins and will measure the temperature and humidity of the storage space. In order for the best accuracy, the temperature sensor should either be in the storage space or in close proximity.

A simple and easy to use user interface (UI) is provided, where the user will be able to see what they have scanned, what the expiration date of the item is, and check if correct information has been entered. If the user sees that incorrect information has been entered, they can manually overwrite the necessary information with the touchscreen. The user is also able to see all of the items that are in the storage space, and decide which item to use. When the user wants to take an item out of the storage space, the user simply has to scan the item and the corresponding information will be updated.

B. *Barcode and Temperature Humidity Sensors*

Fig. 7. Adafruit BME280 sensor [5]

The barcode scanner will read and output a serial number, which is not useful by itself. This serial number will be looked up in a large commercial produce item database to understand what it actually is. The database we will be using is the upcitemdb, which is an universal product code (UPC) item database. Every time an item is scanned, its serial code will be looked up in the upcitemdb and its result will be stored and displayed for the user to confirm. If the item is not found in

this database, we will still allow the user to manually override and manually input the item details.

The temperature humidity sensor constantly monitors the environment of the storage system. The sensor's readings can help ensure that the storage conditions remain within the safe range for the products stored because temperature is a critical factor in preserving the condition of perishable goods. If the temperature or humidity is too high or too low for the item, the system will let the user know so that the user can adjust accordingly.

C. *Optical Character Recognition Model*

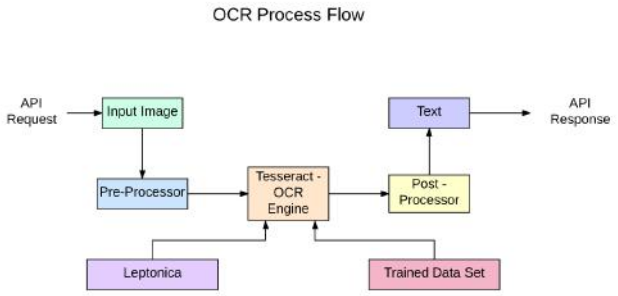

Fig. 8. OCR Process Flow [6]

The optical character recognition model is what identifies and logs all of the expiration dates from the item to the database. It is of most importance that the model is accurate and returns the output promptly. The process is divided into three stages: Pre-processing, OCR engine, and Post-processing. For pre-processing, libraries including Numpy and OpenCV are used. The pre-processing goal makes the input image clear and makes the dates as distinguishable as possible. In order to pre-process these inputs, we are employing multiple techniques, including

1. Resizing the image
2. Enhancing the image contrast
3. Adjusting color to grayscale
4. Analyzing image contours to filter out non-text regions

The OCR engine is the core component that interprets the visual information of the text from the given pre-processed images. We are utilizing libraries including Pytesseract, which is a wrapper for Google's Tesseract-OCR Engine, Tensorflow and OpenCV. The OCR engine is able to analyze the structure of the text and recognize the characters. In order to improve the engine's performance, we are training it on specific fonts and formats that are often used for the expiration dates of products. Lastly for post-processing, the expiration date and confidence score are exported as these two pieces of information are crucial for determining the final value of the expiration date.

D. *Database*

The database on the central computer is a key-value store with the key being the item barcode and the value representing the captured image, expiration date, scanning date, and other important information. This is stored on the disk of the RPi 5 central computer. The scope of the database is extended further as each of the scanner modules stores a partition of the data. Updated data is communicated with the central database as well as the other scanner modules as each of the scanner modules also contains a parity block as seen in Fig. 9. This parity block aggregates the data being stored on the other scanners and is used to recover data if another node goes down.

With the addition of the temperature sensor, we packed more information in the initialization phase of the Scanner Modules. On startup, the scanner module will read the temperature and humidity data from the sensor and will send it to the central module. The central module will keep track of these values across the scanner modules and will note which one has the highest and lowest temperature values. When items are scanned in, we assume that fresh produce does not have a barcode. For these manual override cases, the node it is scanned on will check its temperature and see if it is the lowest value out of all the modules. If it isn't the lowest, it will ask the user to bring it to the lowest temperature module, and will automatically register it on that scanner.

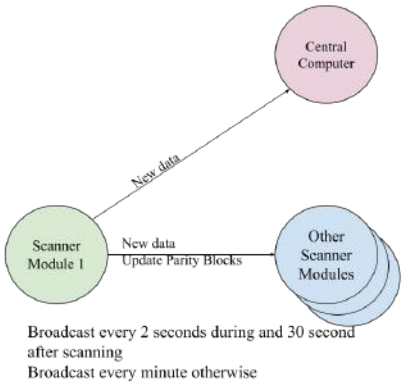

Fig. 9. Module Data Broadcasting Mechanism

E. *Recommendation Algorithm*

The recommendation algorithm takes in the key-value store and outputs an ordered list of items to use. This algorithm is implemented in Python, with weights for each of the numerical features in the database. These features include the initial scan date, expiration date, and temperature of the storage unit. To adapt to the user's needs, these weights are able to be controlled by a slider with values from 0 to 100, which characterizes how important each one is. 0 indicates to not consider that feature entirely while 100 indicates that the sorting should happen solely based on this feature. The blending of this data normalizes the input features and weights them with their respective values to get the item's recommendation heuristic value (2).

In order to reduce latency of computing these values,

differential sorts were performed periodically, every minute. This resulted in partitions of items that were sorted, which allowed the system to not have to sort the entire list at a given request. This reduced the computation time that needed to be put upfront that a user experienced when pressing the recommendation button.

$$H(item) = \sum_{f \in features} w_f p(f) \quad (2)$$

$p(f) \in [0,1]$, the scaled value of a feature f

w_f, the user input weight for a feature f

Note that $\sum_{f=features} w_f = 1$

V. Test, Verification and Validation

Fig. 10 shows a summary of the results from the testing conducted, along with possible cause if a metric was not met.

A. Results for Item Registration & Tracking

Our use case was to scan items of all shapes and sizes into a storage space, so we tried to replicate this test in order to test the registration and tracking requirement. In order to standardize a testing metric though, we created fake items with cardboard boxes and printed expiration date & barcode labels to replicate real life items. The item contents were taken inspiration from our own pantry's contents.

For the read-in accuracy sub-requirement, we replicated the real use case by having team members try to scan items in naturally, which turned out to be around 10cm at a 0 degree angle to the normal of the item plane. This test was performed 60 times, 30 times per person to replicate different user's styles to judge the OCR and barcode recognition accuracy.

For this requirement, we were not able to meet the goal of 90% read-in accuracy, as we were only able to accurately scan in 44 of the 60 items, resulting in 73%. We believe this was due to the form factor of the product, where the camera was mounted on top of the barcode scanner. Even if a hand is held steadily, it might be oscillating very slightly which results in a motion blur. With the camera's focusing time, this resulted in streaks in the image, making it very difficult for the items to be detected. This often resulted in batches of camera photos not having any detected expiration date, which resulted in an overall confidence of -1. The expiration date detection was the bottleneck for the overall accuracy of the system. Notably, all 60/60 items had a correct barcode read-in and item lookup.

We tested the remainder of the sub-requirements by performing this test at the boundary conditions noted for the 60 items. For example, we put a protractor up and scanned items in at an angle ranging from 0 to 45 degrees. We measured the distance from the scanner to the item and scanned items at distances from 5 cm to 30 cm. We tried to overwhelm the system by scanning items in consecutively, rapidly at intervals of 6 seconds all the way down to 3 seconds. A requirement was met if the barcode and expiration date acquisition accuracy was not lower than the first test. These boundary conditions were to allow the user to scan items in various different positions, not constraining the user.

For this requirement, we were able to hit all the goals. For scanning angle, we were able to get up to 35 degrees from the normal of the item with a scan-in accuracy of above 73%. For scanning distance, we were surprised to see that a lower distance bound of 5 cm was achieved, and the upper bound was also extended by 5 cm to a total of 30 cm. For scanning time, we were able to achieve 3 seconds on average for each scan while maintaining the scan-in accuracy, with the majority of the time taken being the OCR algorithm figuring out the expiration date.

B. Results for Scaling

Our use case was to have one module per storage space in the house, so we wanted to verify that this system could be scaled up. This section mainly tested the main module to make sure that data consensus and message passing did not inhibit the use case.

In order to test these sub-requirements, we set up three scanner modules and one central module. We scaled up the item scanning test across these modules and timing code was inserted to figure out the latencies of communication. Data consistency was monitored in a similar fashion. The modules were also moved around the room to replicate the real distances between these modules.

All the metrics for this test were able to be achieved. We were able to get 100 items into each scanner module without slowing down the message passing, which was far beyond the 40 item limit that we set out to achieve. We also noted that the biggest part of the decline in performance past this point was the heavy image files attached to the message, which was the best confidence expiration date picture. We were able to have this scanning setup work for the three storage spaces in a home, with no data being lost in between. This meant that data consistency was achieved via the central module, which had an append-only log of the items being put in. Additionally, we were able to display items scanned on other modules within 500ms of the item initially being scanned in on another module. This was able to beat our use case target by a large margin, again due to the broadcasting timing of the modules.

C. Results for Usability

Since this product revolves around the user being able to easily use the system, we wanted to make sure the interface latency did not affect the user. In order to test this requirement, we built on the test that was conducted for item registration. For the first sub-requirement, we inserted timing code to figure out how long it took for data to be stored in the overall system. Thes second sub-requirement was tested by interacting with the user interface and seeing how long it took for item recommendations to appear on this screen. We tested this metric with very low scanned item counts (1, 10) as well as high item scan counts (30, 40). The time requirement for setting up the node was tested at the very start before performing the scaling tests. This was tested by having a

team-member follow instructions to set up a module while timing this action.

We were able to achieve all the metrics in this section as well. Information was stored very fast, at on average 100ms after the initial scan OCR/UPC database lookups. The display information was also able to be displayed under 500ms, which mainly was achieved by performing differential sorts periodically, so that the recommendation result does not have to be recomputed fully for any call. The daily report was successfully fired off, with the UI displaying it at system clock time of 8AM. The setup time requirement was also met, as all three nodes were able to be set up under the 5 minute time limit.

Requirement #1 (Item Registration & Tracking)	Requirement #2 (Scaling)	Requirement #3 (Ease/Accessibility of Use)
Target: **>90%** read-in accuracy Result: **73%** accuracy Cause: Blurry saturated photos from shaky hand	Target: **40 items** per storage space Result: **>100 items**	Target: Store information within **1 sec** of scanning Result: **100ms**
Target: **30 degree** scanning angle Result: max **35 degrees**	Target: **3 storage spaces** per network Result: **Achieved**	Target: Display info within **500 ms** Result: **Achieved**
Target: **15-25 cm** scanning distance Actual: **5-30 cm**	Target: **10 sec** Synchronization Result: **500ms**	Target: Daily report of expiring item Result: **Achieved**
Target: **4 sec** registration time Actual: **3 sec**	Target: Data consistency Result: **Achieved**	Target: **<5 min** setup node Result: **4 min**

Legend: Green is meeting the requirement, Red is failing to meet the requirement

Fig. 10. Testing & Verification Result Summary

VI. Project Management

A. Schedule

The Gantt Chart attached as Fig. 11 lays out the task division and schedule for this project. One notable change has been the addition of the temperature sensor, which has cut down our slack and integration time. Otherwise, the schedule is relatively similar compared to previous iterations.

B. Team Member Responsibilities

There were three primary responsibilities for this project– image acquisition and processing, barcode acquisition and processing, and central software and database development. Siyuan was responsible for the barcode acquisition. Software integration was found to be time consuming, which Yuma took a large part of the responsibility. Jason was responsible for the image processing, which also took a lot of time to debug and optimize, and Yuma for the central software and database portion of the project. As Yuma also had experience in dealing with computer vision, he also supported Jason in the image field. Yuma also helped Jason with integrating the camera module, and provided optimization strategies for image acquisition and parsing. This included spinning off threads for each picture to send to the OCR module and writing in FIFO queues for image buffer freshness. Finally, Yuma and Jason worked together on the central software and integration towards the end as all these submodules inherently relied on the main module.

C. Bill of Materials and Budget

The Bill of Materials and Budget attached as Fig.12 lays out the materials purchased and their cost. Notably, we had to purchase temperature sensors as this feature was added, and we had to buy three more of each module set in order to test scalability. Our total came out to $536.78, which was still under the budget cap of $600.

D. Risk Management

The critical risk factors in our design were the ability to identify and recognize different types of expiration date labels on items with accuracy above 90%, and developing a good user interface so that the user can interact with the module effortlessly. While expiration dates themselves were a set of numbers, each product had varying locations and background colors, making it difficult for OCR algorithms to properly identify them. Additionally, expiration dates came in various different formats, some with the prefixes “EXP” or “GOOD BY.” Therefore, we dedicated ample time to researching, implementing, and testing an optimal OCR model. Specifically, we researched in depth on how to pre-process the image before inputting it into the model so that background colors and text format would have minimal impact on the accuracy. We also put in time to train the neural network for our OCR model. Specifically, unifying the date format as well as the background color have significantly decreased the number of errors that the OCR engine would produce. We were also not able to mitigate a few risks, as the

user interface development took a lot more time than expected. We had to eventually port some of the features such as manual overriding onto the terminal interface.

VII. Ethical Issues

The primary concern is that of item regionality. The current system relies on the universal product code (UPC) Barcode item database, which contrary to its name, is not an universal format. This format mainly is for items originating from the United States, which means the UPC database only contains items from the U.S. [7]. The rest of the world uses the European Article Number (EAN) barcode format which is the international standard for barcodes, containing thirteen digits. This stands as a large hurdle as individuals who consume international products, i.e. that of Japan or China, may not get the benefit of automatic barcode read-ins. This implies that all these international items would have to be manually overridden in the system, which adds time and complexity on to the user. A possible solution for this issue is cascading the database calls if an item is not found. For example, if an item lookup fails in the UPC item database, the EAN database for Japan can be called, then to Singapore, etc. This would add scan-in time, but the first database being called could be customized based on a user's geographical region or preference.

Another concern would be privacy. Inherently our system contains a lot of information about the user, from what products are bought, when they are bought, to where they are kept. If any malicious actor were to receive the information, this might be used to reveal or take advantage of an individual's dietary habits or health conditions. This concern can be mitigated in part by implementing good data encryption. For example, a diffie-hellman key exchange can happen in the initialization phase as well, which would allow for all modules to have a shared secret to encrypt messages. This still does not prevent middle-man attacks, which can be alleviated by adding a MAC address or another form of authenticating messages to verify they were sent from a module in the system.

VIII. Related Work

There are various past projects that have explored similar topics of OCR and barcode recognition. A previous 18-500 ECE Capstone conducted in Fall 2022 by Group A3, "Where's the Barcode." was a large inspiration for our project [1]. This project gave us inspiration for the hardware and methods available to us to capture a clear photo, with text able to be extracted. Unfortunately, since the use case regarding the camera distance was different across these two projects (discussed in section V), we had to find alternative hardware.

Another source of work we have used as inspiration was an article detailing the OCR process using known libraries such as OpenCV and Tesseract by Nanonets [6]. It went over the general process flow and dove into how each step of the process was handled. It also had some code snippets of introductory helper functions which would be adapted and improved based on the specifics of our design.

IX. Summary

Overall, we have demonstrated a solid proof of concept. We were able to meet most of the requirements set out by the Design Review, barring the accuracy of the system. This was in part due to the form factor of the system, where the camera is very susceptible to perturbations as it is mounted onto the barcode scanner. This is coupled with the relatively slow focusing speed of the ArduCam and the limitations of the OCR algorithm. A few possible improvements if given the time would be to train the OCR algorithm on more various expiration dates, buying a ArduCam that has a faster focusing speed, and possibly having a gyro below the camera to prevent it from jerking and creating streaks in the acquired images. This would inevitably be a tradeoff though as more/fancier components in the system would raise the cost and consequently the hurdle for a user to acquire the system.

During these fourteen weeks, we have learned a lot about the real troubles of integration. We learned this the hard way, as we were stuck with many separate but working components in the last few weeks, but they were not stringed together. If we were able to do this again, we would definitely have talked together about the code we were writing and made function signatures detailing what exactly would be input and output of these components.

Another lesson learned is that UI development is hard. All of us on the team have ample experience with developing systems on the back-end, but not really how these systems interact with the user. This resulted in UI development being a large hurdle. This ranged from learning how UI libraries work, to implementing hacky refactors to get components to be updated.

The final important lesson we learned is that we shouldn't try to overachieve. We tried to add on more functionalities to the system, such as the temperature sensor, which might make the system more impressive and user-friendly. This resulted in a period where we were trying to integrate a lot of things without realizing what our MVP was. Keeping sight of what the MVP is and trying to achieve it before doing anything else is a very important insight I learned throughout this process.

Glossary of Acronyms

CV - Computer Vision
EAN - European Article Code
FOV - Field of View
OCR – Optical Character Recognition
RPi – Raspberry Pi
UI - User Interface
UPC - Universal Product Code
USB - Universal Serial Bus

References

[1] Lee, Jiyeon , et al. “Where’s the Barcode.” Carnegie Mellon University, 2022, Accessed on Jan 31, 2024, [Online]. Available: https://course.ece.cmu.edu/~ece500/projects/f22-teama3/
[2] Google, Google Lens, https://lens.google/
[3] Ongaro, Diego, et al. “In Search of an Understandable Consensus Algorithm.” Stanford University, 2011, Accessed on Feb 28, 2024, [Online]. Available: https://raft.github.io/raft.pdf
[4] Patterson, David A., et al. "A Case for Redundant Arrays of Inexpensive Disks (RAID)." Carnegie Mellon University, 1988, Accessed on Feb 13, 2024, [Online]. Available: https://www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf.
[5] Adafruit. “Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor” Adafruit, 2023, Accessed on Feb 27, 2024, [Online]. Available: https://www.adafruit.com/product/2652
[6] Zelic, Filip, and Anuj Sable. “How to OCR with Tesseract in Python with Pytesseract and Opencv?” Nanonets, 2023, Accessed on Feb 16, 2024, [Online]. Available: https://nanonets.com/blog/ocr-with-tesseract/.
[7] Scandit. “Types of Barcodes: Choosing the right Barcode” Scandit, 2021, Accessed on April 15, 2024, [Online]. Available: https://www.scandit.com/resources/guides/types-of-barcodes-choosing-the-right-barcode/

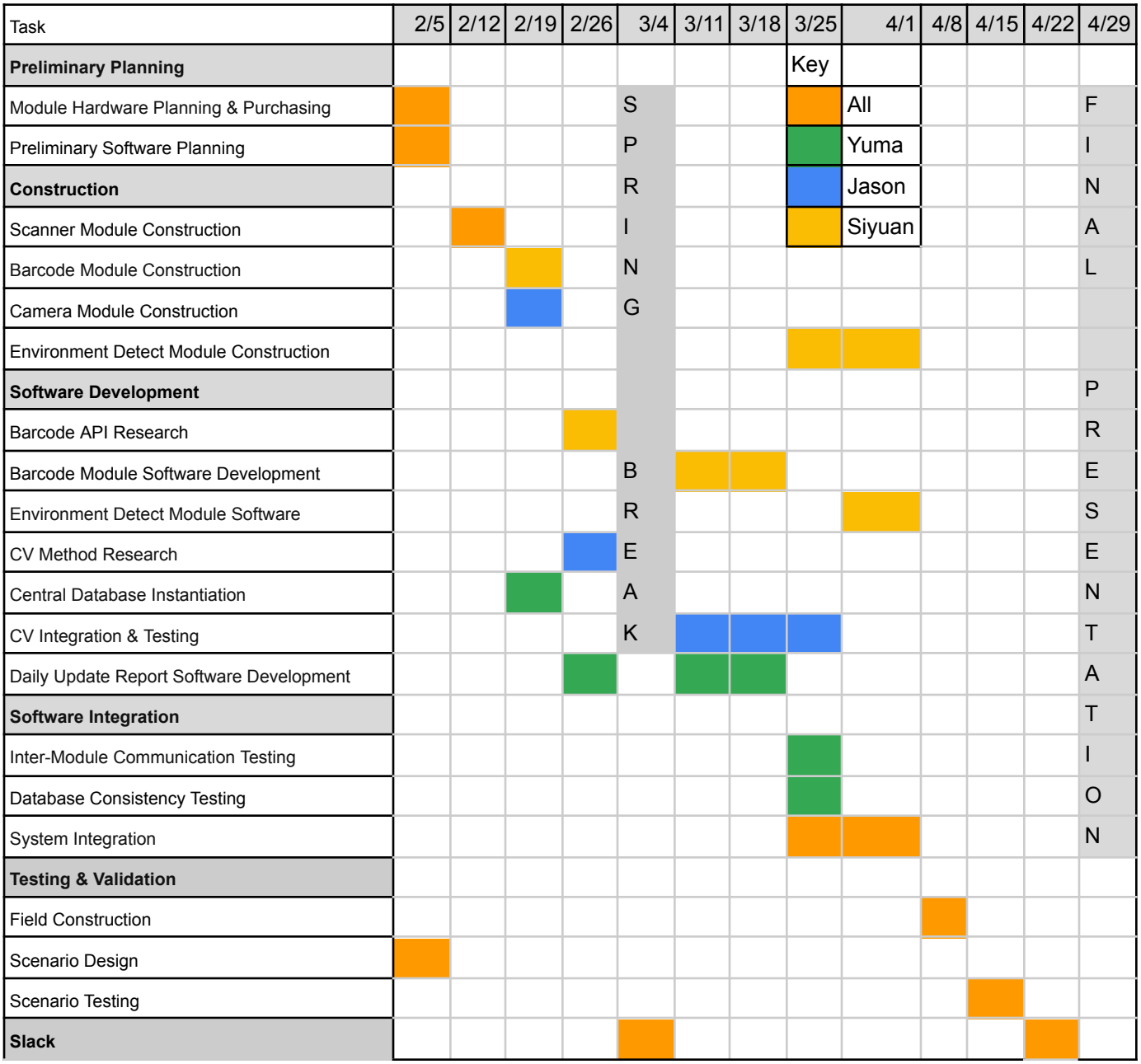

Fig. 11. Gantt Chart

Description #	Manufacturer	Quantity	Cost per Unit	Total Cost	Notes
NexiGo N60 1080P Webcam	NexiGo	1	$29.99	$29.99	Did not use due to design tradeoff
FREENOVE 5 Inch Touchscreen Monitor	FREENOVE	2	$39.90	$79.80	
iPistBit 5 Inch Touchscreen Monitor	iPistBit	1	$38.99	$38.99	Recent addition to make more modules and test scalability
iPistBit 7 Inch Touchscreen Monitor	iPistBit	1	$48.99	$48.99	Recent addition to make more modules and test scalability
Arducam IMX219 USB Camera Module	Arducam	3	$38.60	$115.80	Recent addition (x2) to make more modules and test scalability
CanaKit Raspberry Pi 4 Starter Kit	CanaKit	1	$109.99	$109.99	Recent addition to make more modules and test scalability
WoneNice USB Laser Barcode Scanner	WoneNice	3	$22.79	$68.37	Recent addition (x2) to make more modules and test scalability
Adafruit BME280 Sensor	Adafruit	3	$14.95	$44.85	Recent addition (x2) due to temperature module
Arducam Camera	-	1	$0.00	$0.00	Cost not factored -- taken from ECE Storage
USB WebCam	-	1	$0.00	$0.00	Cost not factored -- taken from ECE Storage
Raspberry Pi 5 Kit	-	2	$0.00	$0.00	Cost not factored -- taken from ECE Storage
Raspberry Pi 4 Kit	-	1	$0.00	$0.00	Cost not factored -- taken from ECE Storage
Computer Keyboard	-	1	$0.00	$0.00	Cost not factored -- taken from ECE Storage
Computer Mouse	-	1	$0.00	$0.00	Cost not factored -- taken from ECE Storage
		Grand Total		$536.78	

Fig. 12. Bill of Materials and Cost