Where's the Milk?

Project Contributors: Lucky Wavai, Allen Ding, and Takshsheel Goswami

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—For the average shopper who is not able to find the items in their local stores that they desire, the employees that are constantly showing customers the aisles where things belong only to find them out of stock, the businesses losing customers to the ease of online shopping and retail, it seems that the traditional brick and mortar shopping experience needs to be reevaluated. We hope to create a component of the next generation of the in-person experience that will provide real time information to customers and businesses alike. More specifically, we aim to build a platform that incorporates computer vision for inventory management of grocery store aisle shelves.**

***Index Terms*—AJAX (Asynchronous JavaScript and XML), Background Subtraction, BRISK (Binary Robust Invariant Scalable Keypoints), Brute Force Matching, Feature Detection, Client, Flask Micro Web Framework, FLANN (Fast Library for Approximate Nearest Neighbors) Matching, HTTP Response, HTTP Requests, Lowe's Ratio Theorem, OpenCV, ORB (Oriented FAST and Rotated BRIEF), RPI (Raspberry pi), Server, SIFT (Scale Invariant Feature Transform), SQLite3, SURF (Speeded Up Robust Features), Web App (Web application), Wireless Communication**

I. Introduction

These days, more and more people are turning from in-person shopping to e-commerce alternatives for ease compared to that of traditional brick and mortar stores. According to Fortune, "UBS forecast that some 80,000 retail stores, or roughly 9% of the current total, will shut their doors permanently by 2026. The forecast is based on the assumption that e-commerce as a percentage of total retail sales jumps to 27%—up from 18% now" [2]. There seems to be a trend of local brick and mortar stores closing which also inevitably results in lost jobs nationwide. One potential way to combat this is to provide customers with a reason to do more of their shopping locally.

Current solutions lack real time analysis and/or require heavy infrastructure. For instance, the typical inventory management systems involve scanning products when received at the backdoor, then keeping track of what gets sold at the register. These processes have no comprehensive management of items while in the store such as information about the presence of items on shelves or misplaced products. [3]

Some more advanced solutions involve robotics in grocery stores such as the Marty Robot. Similar technologies maneuver around the store sensing for the desired information such as spills or hazards, but the downsides include high cost of such robotics hardware (i.e a single Marty Robot costs about $35,000), take up aisle space, and lack the ability to perform comprehensive real time analysis as robots do not scan all aisles of the store simultaneously. [4]

We hope to work towards a new generation of brick and mortar stores with real time automated inventory management systems, navigation systems, and more. Specifically, this semester, we will attempt to take steps towards that store of the future by developing a platform which incorporates computer vision technology to transmit real time data of an aisle status to an application for store operators.

II. Use-Case Requirements

Our primary goal is to build a platform that accurately detects and provides real time data on aisle and object states to a web-interface. Therefore, our primary focus is on accuracy and time/latency..

One use case requirement is to provide a system with at minimum 75% accuracy in detecting the presence of all of the expected store items on aisle shelves. The qualifier of "expected" is to differentiate from items that are on the shelf that were displaced from another aisle. For our primary goal, such items will not be a requirement, but we hope to tackle this area in a possible reach scenario. This use case requirement will be analyzed in our testing phase called the Presence Test.

Another use case requirement is to provide a system with at least 80% accuracy in detecting activity within the aisle, specifically the motion / activity of people within the aisle. This use case requirement will be analyzed in our testing phase called the Motion Test.

Another use case requirement is for all of the information to be processed, transmitted, and displayed to the user device within 15 seconds of detected motion in the aisle i.e. someone entering and leaving an aisle.

III. Architecture and/or Principle of Operation

We are building our platform primarily using the Python programming language with some exception i.e. the web application front end. On top of Python, we are using various libraries packaged under the conda-forge openCV requirements package. The primary libraries of note are openCV, numpy. We will use the MOG2 class within openCV for background subtraction. During the design review we were unsure of which feature extraction method we would use i.e. SIFT, SURF, ORB, or BRIEF, but after testing and implementation we settled on using SIFT with openCV for feature extraction. See Fig 3. Python with the addition of these libraries provides an efficient development environment for our detection use case requirements as well as backend technology and communication between devices for the rest of our system.

The system will process in a manner similar to a finite state process. Initially, the system is in an idle state awaiting aisle activity. After a person is detected within the aisle the system

enters the motion state. Once there is no longer any motion detected, the system enters the item detection state where the presence of all items will be updated. Once processing is complete, the system resets to the idle state. See Fig 2

On the user-facing side, the web application is built upon the Flask framework, backed by a SQLite3 database. The website gives users the ability to create and populate aisle(s) with item(s) as if they were creating their stores. Additionally, these aisles can be viewed in a format that presents aisle information, specifically, the presence of each item on the shelf that will continuously update as changes are being made real-time via the other two subsystems.

Fig. 1. *Overall system*. In reference to the above image, depicted is a demonstration of our overall system. In the background observe the shelf of items, placed at a distance of 5 ft from the wireless system (ARDUCAM connected to a raspberry pi4). This image does not show the height of the camera which is supposed to be 3 ft above the ground. Next to the wireless system is the web application user interface running on a Macbook Pro, displaying the status of items on the shelf for a user created aisle.

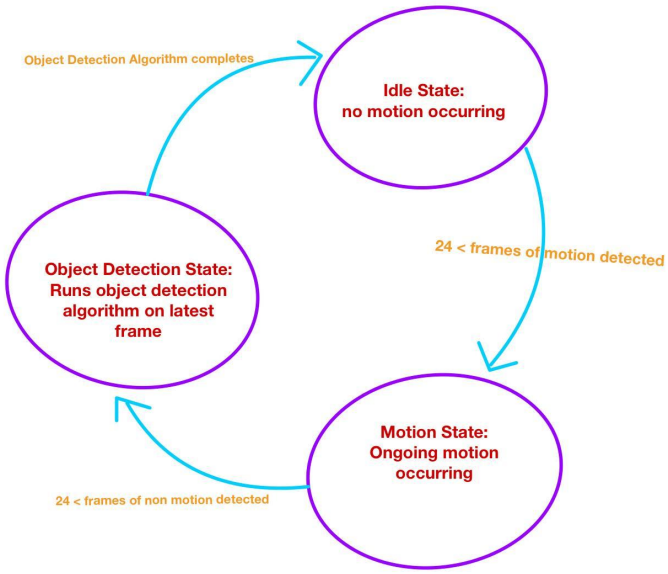

Fig. 2. This is the overall finite state-esque control system for the motion detection and object detection system on each camera module. When a camera module is assigned to an aisle it begins in an idle state. Both the Idle state and the Motion State run the motion detection algorithm indefinitely. The Idle state transitions to the motion state when motion is detected. Once motion ceases, the control transitions to the object detection algorithm which detects the presence of items on the shelf then outputs results to the web app. When that portion completes, control transitions back to the Idle State

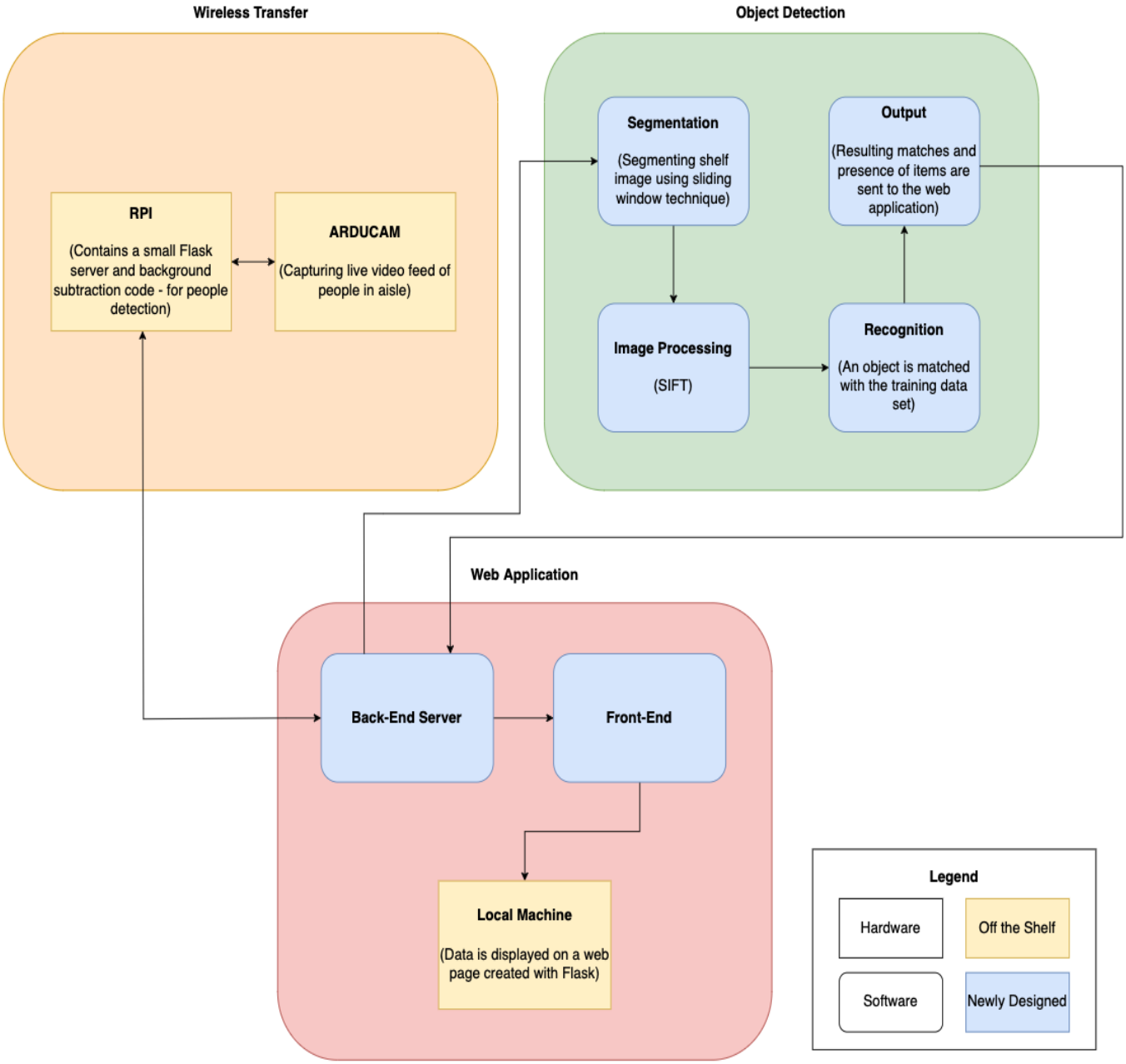

Fig. 3. *System Architecture*

IV. Design Requirements

For this system, we have 3 primary subsystems, namely the motion detection, the object detection, and the web application. The main use case requirements are speed and accuracy for our whole system. Our wireless camera system would capture frames and process both motion detection and object detection locally. Our camera should be able to capture clear images of people and items on shelves 5 feet away, and must be compatible for use with a raspberry pi. The rpi controls the camera, motion detection, object detection, and a flask server. The server sends a rendered image of the shelf section showing where items are located along with a dictionary of items mapped to booleans representing their presence. The rpi serves that data to the web app for the user interface. The process from detecting motion to updating the web app user interface should be within the 15 second user requirements. The transfer from the rpi to the web app should not take longer than 1-2 seconds (preferably near instantaneous) so we have 13-14 seconds to perform image processing and detection.

For our image processing, we have 3 primary processes/algorithms that we would need to execute at any point. Namely, motion detection (background subtraction), segmentation (sliding window), and feature extraction (SIFT). Motion detection/background subtraction is not counted towards our 15 seconds since it is always running and serves as the initial trigger for the rest of the actions. However, the time from the trigger to object detection is counted towards the 15 seconds as the initial event. That time should be near instantaneous at best and 1 second at worst. Through research, we determined that our feature extraction portion could take at the least 3-6 seconds. In combination with the segmentation, we aim to complete a cycle of detection in 10-11 seconds.

Besides the timing requirements, we need to have a high degree of accuracy as well. Our motion detection algorithm must have over 80% accuracy in determining motion in the aisle. We also aim to have at least a 75% accuracy with the feature extraction and matching with SIFT. Once this processed data has been collected, we need to present it in a viable format. That is where our web application comes in.

For our web application we need a speedy framework specifically for display of data. We aren't performing much computation in this step, so our framework need not be extremely comprehensive (eg: django) so we wil veer towards Flask which is lightweight and faster for the specific purposes of receiving and displaying data. For this web application, we'd also need a database that could store processed image data and also work in conjecture with the python image processing that we shall perform. With the current timing, this should leave us about 2–3 seconds to spare from our use-case requirements, while meeting and/or exceeding the accuracy standards involved.

V. Design Trade Studies

The following are some design trades we made before and during implementation along with the reasonings behind them:

A. Raspberry Pi 4 vs Jetson and other alternatives

Raspberry Pi is an affordable microprocessor / computer device that is about the size of a wallet or credit card. The pi module is programmable, and can be used as a controller for other devices. Due to its form factor it can serve as a gateway to IOT device like applications. These specifications make it an ideal candidate for the camera subsystem of our project.

Other options could include using our own laptops or computers, devices from the arduino, or a device such as the jetson nano. One reason laptops and jetsons are not reasonable to meet our project goals is in part because they are expensive devices that provide more computing power than we need. Therefore, a cheaper alternative is desirable. Another reason laptops are not desirable is that it is too large to incorporate on store shelves. The arduino is a cheaper option, but it is also a limited option. The arduino is essentially just a controller for other devices. We want to attempt some onboard processing using the raspberry pi's and transmission that would otherwise require additional complexities if we used an arduino.

One drawback/issue is the limited supply of raspberry pi's in the market. We will implement the local wireless system on a MacBook Pro (16-inch, 2019) with a 2.6 GHz 6-Core Intel Core i7 processor, 32 GB 2667 MHz DDR4 of RAM, and AMD Radeon Pro 5500M 8 GB graphics. Then, we will attempt to port everything over to a raspberry pi

B. Raspberry Pi FB2 Camera - Arducam vs other alternatives

Raspberry Pi cameras can take good quality images and are fully programmable. Another benefit with using the raspberry pi camera is integration with the raspberry pi device itself.

We will avoid the risk of hardware incompatibility that may come with certain webcam or other camera devices. We will also avoid the risk of cameras that require more complexity to control or may be limited in functionality altogether for the purposes of our project.

One drawback of the Arducam is that the current lens we have is a fisheye lens. Fisheye lenses cause some distortion in footage. This is addressed in the risk mitigation portion of this report. In the meantime we will use a Logitech HD Pro Webcam C920 in conjunction to the laptop. Then, we will attempt to transition when porting to the raspberry pi

C. Raspberry Pi Power Supply

One goal is to make the system as unobtrusive as possible within the use case environment. Therefore, we are using a separate power supply for the camera subsystem.

The alternatives were battery power or connecting to a laptop or computer device. The laptop or computer device connection defeats the purpose of using a raspberry pi in place of a traditional large computational device to begin with i.e. bulk and cost. The battery powered option would require constant replacements. We want our system to be as set and forget as possible to increase the ease of installation and maintenance

D. SIFT, SURF, ORB or BRISK - Object Detection

SIFT stands for Scale Invariant Feature Transform. That means SIFT can be used to detect objects with a degree of invariance or degree of tolerance to objects that are scaled differently or in different orientations. For our use case, that could mean items that are further back in the shelves, items that are toppled over or replaced in an un-orderly fashion by people. Therefore, to meet accuracies of above 75% for object presence detection, we would benefit from such an algorithm

One drawback to SIFT is that it is an algorithm that incurs a lot of latency. From research paper [9], it is evident that among similarly peer reviewed alternatives, SIFT is orders of magnitudes slower in total processing time. For instance it averages about double the processing time of SURF.

However, among the alternatives, SIFT time and again out performs in terms of matching accuracy. Since these algorithms run up to about 3-5 seconds in research papers [5], [6], and [9] , and our use case requirement is 15 seconds of total processing time we hope to remain compliant to user expectation and deliver a high accuracy robust system.

Nevertheless, SURF is twice as fast as SIFT, and BRISK and ORB are on average faster than SURF according to research paper [9].

We also performed our own in-house testing to compare SIFT with ORB. We found that SIFT outperformed ORB using our own matcher and filtering algorithms – FLANN and Lowe's Ratio Theorem – (as in Table 1). From research paper [9], we know that BRISK performs around the same as ORB.

E. FLANN and Lowe's Ratio Theorem

After extracting the features using SIFT, we will need to implement a matching algorithm to determine if new images match items in our stored trained images of items.

In order to draw clearer matches we will use FLANN (Fast Library for Approximate Nearest Neighbors) rather than Brute Force Matching.

To further improve accuracy, we need to filter the matches to only include matches that are sufficiently different from one another i.e. distinct to filter out noise and focus only on good matched points. To do this we will use David Lowe's Ratio Theorem as mentioned in paper [13].

However, the Lowe's Ratio Theorem itself requires the adjustment of a threshold for the distance between matched points, and a threshold for the number of total good points to be considered a detected object. This will be determined in testing and validation to achieve the best results

F. MOG Background Subtraction

We decided to use background subtraction for detecting motion within the aisles, more specifically MOG2 background subtraction. Background subtraction is a simpler process of object detection than other methods such as HOG or YOLO because it is essentially just a difference of two images rather than an algorithm to generate and match keypoints and descriptors.

In addition, compared to other background subtraction alternatives such as MOG or GMG, MOG2 performs 3 times faster than MOG and 10 times faster than GMG according to research paper [7]. MOG2 is also capable of upwards of 89% accuracy which exceeds our use case requirement of 80%. [7]

TABLE I. SIFT VS ORB

Items	SIFT		ORB	
	Max Num Correct Matches	***Max Num Incorrect Matches***	***Max Num Correct Matches***	***Max Num Incorrect Matches***
apple	0	7	1	0
bread	10	1	0	0
broccoli	0	1	0	0
cereal	133	0	0	1
chips	10	3	1	0
eggs	0	1	0	0
lemon	0	2	0	0
milk	1	1	0	0
orange	0	5	0	0
oreos	75	1	2	0
peanut butter	57	26	0	0
potato	0	1	0	0
soda	10	1	0	1
tomato	0	3	0	0
case of water	0	1	0	0

a.

G. Flask vs alternatives

Flask is a lightweight web framework that is developed using python. Since the rest of our system is primarily built using python this reduces complexity. [10]

An alternative we initially considered was the Django Web framework. Django has more configuration requirements and features that go beyond the scope of what we need for the user interface of our project. The focus of our system is not in the web application portion, therefore a lightweight option with configurable features / components such as Flask is desirable. [10]

One drawback however is that Flask does not have a built-in administrative tool which comes with frameworks such as Django. Therefore the development portion will lack in the ease of creating and managing instances of models and dummy data for presentation and testing [10]

H. SQLite Database for Object Detection

SQLite is a lightweight database system that supports the ACID principle. This makes SQLie ideal for embedded systems which would serve our project well if we are able to process the object detection on the camera sub system and still mee processing time requirements for our system. The

liteweight, configuration free, self-contained , and transactional nature of SQLite enables such implementation. SQLite also works with python thus reducing the complexity of our overall system. [11]

The ubiquitous nature of an ACID based database schema is crucial for an inventory management system that is focussed on localized aisle items. Therefore desirability of partitioning at the cost of consistency is not an issue. In addition, we will not require high traffic requests for the object given the specifics of application for our object detection. [11]

VI. System Implementation

Our project can be seen as four submodules integrated together. These four submodules are:

A. Motion Detection
B. Object Detection
C. Web Application
D. Wireless Integration

Note: See Figure 1 and Figure 2 above for reference throughout this section.

Before we delve into each submodule, we will discuss how the aforementioned parts will be integrated together. To start, we will have a continuous video feed monitoring the aisle for human activity. Once a person is detected in an aisle (using openCV, specifically background subtraction), the RPI/Laptop that was used with people detection will run object detection using the latest frame from the motion detection. The object detection algorithm examines the image and sends an output detailing what items were on the shelf along with a rendered image with bounding boxes around the items that were found. Processing and running our people/object detection algorithms will be tested in two ways: on the RPI and on our laptop. The RPI is the final desired implementation, but given the supply of raspberry pi and the increased complexity of them, we will implement it first on a laptop with the USB camera as described in section V. Design Trade Studies. The detection output will be transmitted through a flask server to the web app. Our output will be tested in potentially two ways: continuously running AJAX from our web application to update data and synchronous HTTP responses and requests. Once the data has reached our web application, we will add the data to our database, which will trigger our app to send the updated data to the frontend for the user to view. Now, let's examine each submodule in detail.

A. Motion Detection

Our first subsystem is our motion detection subsystem which must detect motion in the aisle until the aisle is clear, and then report to the object detection subsystem. For our camera we're going to be using the Logitech Webcam then eventually replace it with the ARDUCAM for both the motion as well as the object detection algorithm. We will eventually connect to this arducam via a RPi, but begin with the laptop and Webcam. Once we capture footage, we run a background subtraction algorithm using MOG2 for our implementation since it was the most reliable one; especially since other algorithms like KNN specialize in finding motion in a few pixels while our bodies in the frame take up a substantial amount of space. Our background subtraction algorithm works by finding differences between a mobile foreground image and a stationary background image, and we record that there is motion in the aisle if we can see motion in the frame differences. The detection uses a finite state-esque control to trigger the object detection. First state is the idle state when there is no motion, next state is the state where there is motion. The final state is the object detection state that occurs on the transition from the motion state to the idle state, obviously motion must be detected within the system, and we do so when at least 24 frames have motion detected. Once in the motion state, nothing occurs until no motion is detected which occurs after at least 24 frames have been detected without motion (as in Fig 2 above).

B. Object Detection

For object detection, we will traverse the image given using a sliding window algorithm to yield segments that are of size similar to our desired grocery items. Then we will extract key points and descriptors from the image using SIFT, and compare those points with our training data set. The training data set will be SIFT extracted keypoints and descriptors of images we acquired online and take ourselves mapped to our desired grocery items. We will use FLANN to match the key points, and Lowe's Ratio to compute a number of good matches between each window and one of our desired grocery items. If the number of good matches exceeds a threshold we will mark the object as present and store the window's location on the shelf. At the end, we will use the presence information and the locations where found to modify the original shelf image and draw labeled bounding boxes of detected images. Both the rendered detected image and the presence information will be served to the web app via a local flask server.

C. Web Application

For the web application, we will be using the Flask web framework with SQLite3 as our database. The motivation behind choosing these two were mainly because of functionality and speed. For our use cases, Flask provides more than enough (i.e. user authentication, forms/form validation, database compatibility, HTML templates, etc). At its core, Flask is very bare bones and we add modules to it as we need. This lightweight service compared to other services (i.e. Django) is much faster. Flask may not be as comprehensive as other web frameworks, but for our use case, it's perfect. As for SQLite3, just like Flask, it's very lightweight. To meet our user requirements, we are going to need fast access and management with our database. SQLite can handle up to 281 Terabytes, which far exceeds our project scope [12]. Once the data from the computer vision algorithms is finished computing, we will process that data and store it in

our database in a table where the name of the product, the existence of the product, and a unique identifier will be located. After this, we will trigger a call to the frontend where we will upload the new data to the webpage. The frontend will be designed through five main pages: login/registration, inventory management, homepage, and items presence. For login/registration, that is self-explanatory, but for the homepage, it'll have various tiles on the page (similar to a grid view). The tiles represent an aisle in the store and when clicked, the user will be redirected to a page that has all of the items in that aisle shown as present or not (item presence page). In order to populate those aisles, we have a page where the user can create aisles and assign items to them (i.e. building the layout of the store). This will be the inventory management page. The frontend will be styled with Bootstrap, HTML, CSS, and JavaScript.

D. Standalone/Miscellaneous (Foot Traffic Counter)

A 4th, standalone system was also made that unfortunately we couldn't integrate in time. This subsystem was completed rather late into the project, and we didn't have the time to use it in our final design. However, we developed a foot traffic counter with the goal of having it perform 2 functions. First, the counter would keep track of how many people walked into frame. To do this, initially we were interested in centroid tracking and contour detection, however, on testing, we realized that these worked well for systems where the people were over 10 feet away from the camera, and these systems couldn't detect people who were 3-5 feet away very accurately. For our system which has a 5 ft wide aisle, we need something that works for a shorter distance. Then I switched my idea to a face detection algorithm using Haar cascades. This worked rather well until we realized another issue. There's no guarantee that the shoppers would face the camera. Thus, we switched from a face detection Haar cascade to an upper body haar cascade. Now, no matter which side the customer faces, we have got a detecting body. The second thing we wanted our foot traffic counter to do, was to signal to the object detection system when the frame was clear for running the object detection system. To do this, we checked for when the current number of people in our frame were 0, and signal.

Fig. 4: Sending a signal "clear" when no customers in frame

VII. Test, Verification and Validation

Seeing as our project has 3 primary sections we shall divide our testing of project requirements into the 3 parts as we have discussed in our block diagram, where we test the computer vision portion, wireless transmission, and the web application as separate entities and individually test those. After successful testing of these separate components, we shall test the integration of our components with full system tests after our full implementation.

A. Motion Test - Results for the Wireless System

For our Motion Test, we did not have a working subsystem in time for thorough testing. From research paper [7], we found that background subtraction methods can reach accuracies up to about 90%with averages around 80%. Therefore, our goal is to be able to detect the presence of items on shelves above 80% accuracy. Since our motion detection will be a video feed as opposed to the image capture used for item presence detection, our testing process will be different. We will run multiple tests of different motion states: (1) steady state - a person standing in an aisle simulating browsing items, (2) a person moving through the aisle to simulate traffic / motion through the aisle, (3) no people in the aisle.

What we instead tested for was the minimum frames to use to determine that items may have been moved on the shelf in order to cause transitions in the control system. We timed how fast we could take to enter the camera, frame, and modify the shelf i.e. take objects from the shelf. We determined that the length of time was at least 3-4 seconds which when using our Logitech Webcam and motion detection algorithm was about 24 frames. Therefore we set that as the minimum frame count for transitions in control as shown in Fig 2 above.

B. Presence Test – Results for the Object Detection System

For our Presence Test, tested the shelf item detection accuracy. From research paper [5], SIFT, ORB, and SURF reached accuracies up to about 50-70% after undergoing several comprehensive tests. From research paper [6], SIFT reached accuracies up to 100% in some unit tests. Therefore, our goal is to be able to detect the presence of items on shelves above 75% accuracy by using either SIFT, SURF, or BRIEF feature matching algorithms. Our data set used multiple images containing one or more of the intended shelf items for the following items: apple, loaf of bread, broccoli head, cereal box, bag of chips, egg carton, lemon, milk, orange, oreos, peanut butter, potato, soda, tomato, case of water, oil, protein powder, oats, paper towel, tuna can, box of brownie mix, box of ziploc bags, container of chia seeds. We split the data set into 25% testing and 75% training groups (not training as in that of a machine learning algorithm but just for explanation sake). We compared the training set to the test set to benchmark accuracy ratings for our project, and to help us select which algorithm to proceed with. From testing We found that SIFT was the better option for our use case (as in Table 1 above)

From the testing to determine to use SIFT, we also found

that some items were not optimal for our project i.e. the product and decided to remove them from our interest list. We considered other designs i.e. adding a color detection algorithm, but did not implement it within the semester.

We also used the metrics to tweak parameters i.e. the filtering threshold for Lowe's Ratio. As shown in Fig 5 below , we found a Lowe's ratio between 0.70 and 0.80 yielded the best results with consideration of correct matching, items that were missed altogether, and mismatching i.e. false positives.

In total we were able to receive consistent results of at least 70-80% when all of the parameters were set in our apartment testing environment.

However, as found when we demoed in the CMU gymnasium, our system was severely hindered by the change in ambient light. We had to teak parameters on the spot to achieve presentable results. A future consideration would be to create an image processing process that set's the test image to a certain parameters of contrast and light exposure

Lowe's Ratio Thresh	Present Objects Correctly Detected (INT)	Present Objects Not Detected (INT)	False Positives (INT)	List of present items not detected	List of present items false positive
0.00	0	11	0	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels	N / A
0.05	0	11	0	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels	N / A
0.10	0	11	0	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels	N / A
0.15	0	11	0	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels	N / A
0.20	0	11	0	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels	N / A
0.25	0	11	0	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels	N / A
0.30	0	11	0	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels	N / A
0.35	4	7	0	protein-powder, oats, milk, chia-seeds, peanun-butter, tuna, vegetable-oil	N / A
0.40	4	7	0	protein-powder, oats, milk, chia-seeds, peanun-butter, tuna, vegetable-oil	N / A
0.45	4	7	0	protein-powder, oats, milk, chia-seeds, peanun-butter, tuna, vegetable-oil	N / A
0.50	4	7	0	protein-powder, oats, milk, chia-seeds, peanun-butter, tuna, vegetable-oil	N / A
0.55	6	5	0	protein-powder, milk, chia-seeds, peanun-butter, vegetable-oil	N / A
0.60	7	4	1	milk, chia-seeds, peanun-butter, vegetable-oil	tuna
0.65	7	4	1	milk, chia-seeds, peanun-butter, vegetable-oil	brownie
0.70	6	5	7	protein-powder, oats, milk, peanun-butter, vegetable-oil	protein-powder, milk, chia-seeds, brownies, tuna, ziplocs
0.75	5	6	10	protein-powder, oats, milk, chia-seeds, peanun-butter, tuna	protein-powder, oats, milk, peanun-butter, brownies, tuna, ziplocs, vegetable-oil
0.80	8	0	17	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels
0.85	11	0	35	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels
0.90	11	0	49	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels
0.95	11	0	96	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels
1.00	11	0	127	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil	protein-powder, oats, milk, cereal, chia-seeds, peanun-butter, brownies, tuna, ziplocs, vegetable-oil, paper-towels

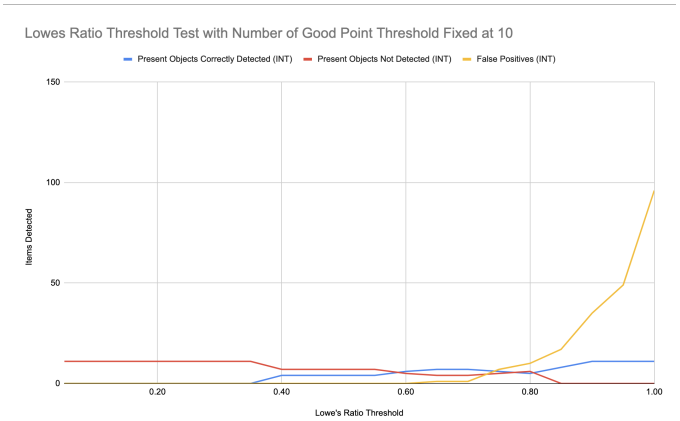

Fig. 5. This table and graph shows the number of matches produced as the Lowe's Ratio threshold was increased from 0.00 to 1.00. The blue line is the number of good correct matches. The red line is the number of missed items i.e. items that were present but not detected. The yellow line represents the number of false positive matches, i.e. matches to incorrect items

C. *Results for the Web Application*

Recall that our design specification for the web application revolves around user experience and security. To achieve this, we ensured that at each step of the development process we succeeded in the following: User authentication/password encryption, cross-checking/cleaning form inputs, no HTTP errors, limiting visibility of pages based on user credentials, ease of access/UX on each page, and more. Namely, the cross-checking/cleaning form inputs was achieved by comparing item values to our global data set of items that only the developers see. This way, users using the inspect tool on browsers or by hacking the form inputs cannot successfully have those items be stored in aisles as they will not be inside our global set of items.

VIII. Project Management

A. *Schedule*

Our scheduling changed by a large amount from the design to the final project report. We had initially allocated the last three weeks of the semester (up until the final presentation) for integration and testing, but we did not get to integration stages until roughly the final presentation week. From there, integration took much longer than expected and we eventually could not resolve our bugs in time. Additionally, there were miscommunications about how the design/functionality of the website should have been during the final week of the semester, and we ended up running out of time to implement all of the changes we agreed upon during those last few days.

B. *Team Member Responsibilities*

Please refer to the below paragraphs that detail the responsibilities and tasks that each team member completed over the course of the semester.

Lucky was in charge of the object detection subsystem and he was able to research and test different object detection

algorithms, as well putting SIFT (the algorithm that he went forward with) into practice. He wrote a completely modular codebase to be used easily with just a few function calls that Allen was able to test with his website during the integration stages. Eventually, due to integration problems with the RPI, Lucky integrated the motion detection code with his existing object detection code. Additionally, Lucky would take charge in coming up with overall design ideas. During integration time, he assisted Allen in trying to get integration working with the RPI and he made the motion detection code more modular. Lastly, this project idea stemmed from an initial idea that he had.

Allen was in charge of developing and designing the web application, he set up the RPI/ARDUCAM with the background subtraction code, and had initial integration in place with the website and the wireless system. Together with Lucky, they tested the wireless system and further implemented it from the original background subtraction code they used as a resource online to account for the sensitivity of detecting moving objects. Additionally, Allen and the rest of the team would bounce ideas off each other and eventually, Allen and Lucky would have further talks in design and implementation.

Lucky and Allen also spearheaded the presentations, which included creating the slides. Additionally, the both of them wrote nearly the entirety of both the design/final reports, the poster, and the video.

Takshsheel was originally in charge of the wireless system, but that trailed off mid-semester. He developed a people counter with the existing background subtraction code he found online that was not tested fully and was not involved in our final demo/video, or for much of the semester.

C. Bill of Materials and Budget

Please see Table II below

In our final demo, we did not end up using either RPI or ARDUCAM, but rather we used a wired USB camera that we had prior to this course and the Macbook pro as described in V. Design trade Studies due to issues when transitioning to the wireless system and properly relaying HTTP responses and requests between the wireless camera system flask server and the main flask web app. We also had lighting issues in the gymnasium that we did not account for in testing.

D. Risk Management

The main risks that our team faced dealt within the realm of design and each person's schedule.

To combat design issues (i.e. how many cameras should we use? What is the best method of integration? What is the best object detection algorithm?), we focused on trying to make our processes as fast as possible to reach our ussr requirements, while also simplifying the integration process.

For the number of cameras, we took into account the ethics talk that we had and condensed the number of cameras from two to one (i.e. both motion and object detection would be run on the same camera) thereby halving the overall camera usage in a fully implemented version of our system

As for integration, an example of mitigation was how Lucky wrote his object detection code. He wrote it in a modular and object-oriented manner, so that Allen would be able to seamlessly import the necessary functions. This step cut down integration since it was like using a python module / package. Additionally, we were unsure of how to connect our wireless system with the web application. To mitigate this, we realized that the simplest form of triggering the website to run object detection was to send over an image of the shelf after someone has walked through. We achieved this through HTTP requests, although we quickly realized that communication from the website back to the RPI did not go as smoothly as the reverse direction. Therefore we p[ivoted to creating a flask server on the RPI, but did not finish a working implementation on time.

As for the choice of the object detection algorithm, Lucky had many to choose from: SIFT, SURF, ORB, BRISK, etc. To mitigate the risk of which one would be best suited for our purposes, he ran a series of tests to compare the accuracy and runtimes of each program. Once Lucky determined that SIFT was the best algorithm out of the ones he analyzed for our project, he further improved on the algorithm by implementing a sliding-window technique to segment items to be analyzed on a shelf, as well as other features.

In terms of scheduling, we underestimated the amount of time that it would take to integrate, along with design changes along the way with the website and wireless system. This coupled with other assignments and exams from other classes proved to show faultiness in our time management. To mitigate this, we had placed in a week to two weeks maximum of slack time, but we had ended up taking more than that especially when hitting unforeseen hurdles towards the end as we integrated and ported to the RPI's. This portion was not handled in the best manner and showed us clearly how for future projects, adhering to a schedule is extremely critical to the project's success. Communication also became an issue as teammate(s) went on long series of unresponsiveness when they felt overwhelmed.

IX. Ethical Issues

As with any project, ethical issues will arise and as engineers, it is our job to plan to mitigate those issues during the development process.

The main issue involves the camera in our setup. As of right now, we have one camera that is used for both motion and object detection. This issue is inevitable because our project

depends on the use of a camera. A possible case that would spark trouble would be if a user is tampering with the camera (i.e. using it for inappropriate purposes). To mitigate this, we could potentially have the camera placed on a robotic stand or something similar. When motion detection is running, the stand could be titled towards the lower part of the shelf, meaning that the camera would only be scanning below the knees. The task of motion detection would still be fulfilled, but the risk of invasion of privacy would decrease. Afterwards, the stand would move up so that that camera would be in position to take a picture of the opposite shelf for object detection. Lastly, the camera would go back into its original position.

Another potential concern stems from the fact that our system will most likely not be 100% accurate. As the store manager sees that items are not present, they may want to send someone to replenish the shelf. Ideally, this does not sound hazardous, but consider the flip side: the website displays false-positives generated from the object detection code. Should the manager trust the code more than the employees, the employees may face trouble for a “misstocked shelf” or for not stocking a shelf in time. To help mitigate this, besides improving the performance of the object detection, we could include a disclaimer in our product about the accuracy and also serve the website on multiple platforms. This way, employees running the aisles can have the same amount of time as the manager once they see that an item is out of stock on the shelf.

X. Related Work

Throughout the web, there are many *standalone* projects or products that implement human detection or object detection.

Currently, there are no projects or products that are using computer vision, wireless transfer, and a web application for the purpose of keeping track of store inventory.

From our research, there was a product that was similar to ours in the sense that it uses openCV to examine objects in an inventory system before it leaves the factory. This product is called the *Smart Factory Defect Detection System*. [8]

In summary, this product continuously monitors the products in the manufacturing process and searches for defects. For the detection algorithm, this product mainly uses YOLO with an CNN to find flaws (i.e. defects on metal surfaces, such as scratches, dents, etc). The size and location of the defect(s) are then reported. [8]

We observe here that there are underlying similarities not just in our software and hardware, but in the use case as well for inventory. While we are not looking for defects in a product, we are both examining inventory and using openCV to do so.

XI. Summary

In summation, our system met most of the user requirements, namely: object and motion detection accuracy rates. User experience tests for the web application were also met. However, due to issues with integration, we were unable to have all three subsystems running together to test the overall runtime requirement (i.e. <= 15 seconds). When we were testing a partially integrated system (Web application/object detection receiving an image from the wireless transfer system), we ran into issues with displaying/saving the image to the website. Speed was relatively fast though, we clocked in at around ~10 seconds before the image failed to upload to the website, meaning that the user should have been able to see the image in roughly ~11-12 seconds.

By ensuring that processed images were saved to the database and uploaded to the website properly, we believe that our base integration would have been complete. From there, we would have optimized our code and tested different methods of integration to potentially speed up our overall runtime.

A. Future work

Yes, as a team we have discussed progressing this project even further. After working together for a semester, we have a much better understanding of what the design requirements and vision of the project should be.

For motion detection, similarly to object detection, we would turn the code into a reusable library that other systems can just plug in easily. Additionally, we would ensure that the connection going to the website is flawless, and vice versa.

For object detection, we would recommend experimenting and adding additional detection pathways such as a color detection to enable detection of a wider array of items. Also, other segmentation techniques besides sliding windows might be faster and more accurate. One idea we recommend while still incorporating the sliding window is potentially a dynamic sliding window in which the window changes size as it passes over the test image to better fit the desired items.

As for the website, we would make it much more robust by giving the user the options to edit/delete aisles, improve the UI, allow users to request for new items to be analyzed, review forms that the user can send to the developers, implement AJAX, deploy the website, make this website accessible to phones and tablets, and more.

Overall, we have discussed including a customer component as our product is mainly for the business side right now. We would essentially build an app for the user to interface with. Similarly to the web application, users would be able to view the status of all items in all aisles in real-time. Additionally, they will be able to put in a shopping list (format of the list is

TBD) and a pop-up of the existence and location of each item would be displayed. Lastly, we would have a navigation system that given a list of items, would generate a shortest path algorithm that would minimize the distance traveled by the customer in the store. Other features would be included of course, but the aforementioned features would be some of the main ones.

B. Lessons Learned

For other student groups looking to address this application in future semesters, we would advise them to spend a dedicated 30-40 hours to research and design, as a group. Yes, they will end up writing block diagrams and presenting a number of times during the semester, and groups cannot prepare for everything, but having as much detail as possible, especially during the beginning, will be crucial. Additionally, we would recommend teams to start testing and integration at around the halfway point of the semester. We know that is early, but teams should aim to finish their MVP by mid-semester break and give themselves lots of slack time. We would recommend testing integration on a small level (i.e. sending HTTP signals to a server) and working their way up to more advanced topics like websockets. With multiple systems, coordination is key, and we recommend that each team meet up once a week for a detailed team status report. This in addition with the status report meetings with their assigned professor and TA will decrease chances of error in the design process.

One more thing that we have learnt is that instead of focusing on subsystems in the later part of the project, getting correct successful integration is far more important. For eg: instead of focusing on completing the counter, our time might have been better spent getting a more correct and fully integrated system together.

Glossary of Acronyms

AJAX - Asynchronous JavaScript and XML
BRISK - Binary Robust Invariant Scalable Keypoints
CNN - Convolutional Neural Network
FLANN - Fast Library for Approximate Nearest Neighbors
HOG - Histogram of Oriented Gradients feature descriptor
ORB - Oriented FAST and Rotated BRIEF
SIFT - Scale Invariant Feature Transform
SURF - Speeded Up Robust Features
YOLO - You Only Look Once feature descriptor

References

[1] IEEE, *IEEE Author Center: Author tools*, Accessed on Jan 17, 2022, [Online]. Available: Source Link
[2] Wahba, Phil. “Another 80,000 Retail Stores Will Close by 2026, Says UBS.” Fortune, Fortune, 5 Apr. 2021, Accessed on February, 7, 2022, [Online]. Available: Source Link
[3] Campbell, Jeff. “How Do Grocery Stores Keep Track of Inventory?: The Grocery Store Guy %.” The Grocery Store Guy, 29 Aug. 2021, Accessed on February, 7, 2022, [Online]. Available: Source Link
[4] Gallucci, Nicole. “Marty the Robot: Non-Essential Worker.” Mashable, Mashable, 23 Nov. 2020, Accessed on February, 7, 2022, [Online]. Available: Source Link
[5] Karami, Ebrahim et al. “Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images, ” *Faculty of Engineering and Applied Sciences, Memorial University, Canada*, Accessed on February, 7, 2022, [Online]. Available: Source Link
[6] Sun, Yan Li et al.”Performance Analysis of SIFT Feature Extraction Algorithm in Application to Registration of SAR Image,” *Department of basic experiment, Naval Aeronautical and Astronautical University, Yantai264000,China*, [Online]. Available: Source Link
[7] Marcomini, L. A. et al.”A Comparison between Background Modeling Methods for Vehicle Segmentation in Highway Traffic Videos,” [Online]. Available: Source Link
[8] Park, Sang-Hyun, et al. “Deep Learning-Based Defect Detection for Sustainable Smart Manufacturing.” *MDPI*, Multidisciplinary Digital Publishing Institute, Accessed on February 25, 2022, [Online]. Available: Source Link.
[9] Tareen, Shaharyar, et al. “A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK,” *IEEE*, Accessed on March 1, 2022, [Online]. Available: Source Link
[10] “Flask vs Django in 2022: Which Framework to Choose?” Hackr.io, Accessed on March 2, 2022, [Online]. Available: Source Link.
[11] “Sqlite Reviews & Ratings 2022.” *TrustRadius*, Accessed on March 2, 2022, [Online]. Available: Source Link
[12] “Appropriate Uses For SQLite.” *Appropriate Uses for SQLite*, https://www.sqlite.org/whentouse.html#:~:text=SQLite%20supports%20databases%20up%20to,will%20support%20281%2Dterabyte%20files. Available: Source Link
[13] Kaplan, Avi “Interpreting the Ratio Criterion for Matching SIFT Descriptors” Technion - Computer Science Department - M.Sc. Thesis, 2016, Accessed on March 2, 2022, [Online]. Available: Source Link

	Task	Owner	Status	Start Date	Due Date	Notes
1	Initializing Web Application	Allen	Done	02/21/2022	02/23/2022	Initializing Front-End and Back-E...
2	RPI and ARDUCAM configuration	Allen	In Progress	02/21/2022	02/26/2022	Ensuring that the hardware (i.e. R...
3	Detecting one person	Takshsheel	In Progress	02/21/2022	02/27/2022	Allen and Lucky will assist
4	Matching one shelf item	Lucky	In Progress	02/21/2022	02/27/2022	Allen and Takshsheel will assist
5	Designing Front-End to MVP	Allen	In Progress	02/27/2022	03/05/2022	
6	Testing people detection (one person)	Takshsheel	In Progress	02/28/2022	03/02/2022	Allen and Lucky will assist
7	Testing object detection (one item)	Lucky	In Progress	02/28/2022	03/02/2022	Allen and Takshsheel will assist
8	Display people/object data onto web page	Allen	To Do	03/02/2022	03/05/2022	
9	Slack	Everyone	To Do	03/05/2022	03/12/2022	Things potentially won't be on sc...
10	Repeat steps 6-7 but with up to 5 objects	Lucky	To Do	03/12/2022	03/19/2022	Allen and Takshsheel will assist
11	Repeat steps 4-5 but with multiple people	Takshsheel	To Do	03/12/2022	03/19/2022	Allen and Lucky will assist
12	Repeat step 10 but with up to 10 objects	Lucky	To Do	03/20/2022	03/27/2022	Allen and Takshsheel will assist
13	Uploading all data to Web App and finalizing UI	Allen	To Do	03/27/2022	04/03/2022	
14	Testing complete people/object detecting and matching	Everyone	To Do	03/28/2022	04/04/2022	
15	Testing complete Web Application	Allen	To Do	04/03/2022	04/10/2022	
16	Testing complete integration	Everyone	To Do	04/11/2022	04/15/2022	
17	Final Presentation Prep	Everyone	To Do	04/16/2022	04/18/2022	
18	Slack	Everyone	To Do	04/19/2022	04/24/2022	
19	Final Video	Everyone	To Do	05/01/2022	05/07/2022	

Fig. 4. *Schedule: Milestones and Team Responsibilities*

TABLE II. BILL OF MATERIALS

	A	B	C	D	E	F	G
1	Product	Part/Model Number	Manufacturer	Description	Source	Cost	Notes
2	Raspberry Pi 4	Model B	Raspberry Pi	Board only - 8GM RAM	CMU ECE Department	$0	
3	Raspberry Pi Power Adapter	Raspberry Pi 4 Model B	Raspberry Pi	Power Supply	CMU ECE Department	$0	
4	Raspberry Pi USB-C Power Supply	Raspberry Pi 4 Model B	Raspberry Pi	Power Supply	CMU ECE Department	$0	
5	ARDUCAM	LN05101	ARDUCAM	120 Degree Ultra Wide Angle CS LENS for RPi HQ Camera - 3.2mm Focal Length with Manual Focus	CMU ECE Department	$0	
6	Raspberry Pi High Quality Camera	FB2	Bicool	Raspberry Pi HQ Camera Module for Raspberry Pi 4B/3B+/3B/2B/A+/Zero/W/Zero WH,12.3MP IMX477	Amazon	$88.80	
7	Raspberry Pi Camera Cable Set	B0177	ARDUCAM	Arducam for Raspberry Pi Camera Ribbon Flex Extension Cable Set (7Pcs), 5.9" 7.87" 11.8" 19.69" 39.37" for Raspberry Pi, 2.87" 5.91" for Pi Zero	Amazon	$10.59	
8	ARDUCAM Tripod	N/A	ARDUCAM	Arducam Tripod for Raspberry Pi HQ Camera, Mini Lightweight Portable Camera Tripod Stand	Amazon	$9.53	
9	Raspberry Pi 4	Model B	Canakit	Extreme Kit - 8GB RAM, 128GB Storage	Canakit	$214.90	
10	Samsung Mirco SD Card	MB-MC32D	Samsung	SAMSUNG 32GB EVO Plus Class 10 Micro SDHC with Adapter 80mb/s (MB-MC32DA/AM)	Amazon	$10.02	
11	Raspberry Pi 4 Micro HDMI Cable	N/A	Canakit	CanaKit Raspberry Pi 4 Micro HDMI Cable - 6 Feet (Pack of 2)	Amazon	$15.85	
12	Shelf	N/A	Amazon	Amazon Basics 5-Shelf Adjustable, Heavy Duty Storage Shelving Unit on 4" Wheel Casters, Metal Organizer Wire Rack, Chrome (30L x 14W x 64.75H)	Amazon	$71.41	
13	Logitech HD Pro Webcam	A25	Logitech	Logitech HD Pro Webcam C920, 1080p Widescreen Video Calling and Recording (960-000764)	Amazon	Free	(owned prior to class)
14					Total:	421.1	