Grocery Store Checkout System

Authors: Brian Chhour, Shubhi Jain, Simon Xu
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of computing the fastest checkout line in a small grocery store and displaying the result to users in less than 5 seconds with at least 90% prediction accuracy. Cameras are used to capture information regarding each checkout line, which will be sent to our software running on a laptop. For processing the camera data, YOLO is the computer vision algorithm of choice for motion detection and object recognition. Our CV algorithms are designed to run in parallel to improve scalability. Our primary goal is to improve the checkout experience for shoppers by expediting the process.**

***Index Terms*—Computer Vision, Motion Detection, Object Detection, YOLO**

1 INTRODUCTION

When going grocery shopping, customers generally want to minimize the time that they are in the store. Oftentimes, waiting in checkout lines and subsequently being checked out is a major bottleneck that increases the time spent inside of a store. Currently, the strategy for the majority of shoppers is to observe the number of people in a line and then choose the line with the least amount of people. However, this method of choosing a line can backfire: a line with less people can take longer to check out depending on the speed of the cashier working in that line and the total number of items shared between the people waiting in that line. As a simple example, a line with two people could take longer to check out than a line with three people if the two people have more items, if the cashier in that line is working at a slower rate, or both.

Our solution aims to be a system that will analyze the aforementioned factors in checkout time (the number of people, items, and cashier work rate) to recommend to a user the checkout line that takes the least amount of time possible. In order to achieve our goal, we plan to set up 2 cameras in each checkout line, one facing out towards the shoppers and their carts, and another facing the cashier. These cameras will feed footage to our computer vision algorithms, which should determine information about the number of shoppers, the fullness of their shopping carts, and the speed the cashier is working at. From there, we will use that information to estimate the time a checkout line should take and compare with our estimate for all other checkout lines to determine which is fastest. Finally, our result will be displayed on an LCD display that the users can see. Our system will generate significantly more accurate predictions of the fastest checkout line than a user could due to 2 main reasons:

1. Our system takes into consideration the cashier work rate, which would be difficult for a person to discern
2. We aim to approximate the number of items in a checkout line, and since we will have cameras in each line, we can do all this simultaneously, whereas a person would need to look at all the lines one at a time

2 USE-CASE REQUIREMENTS

Our system expedites the grocery shopping process, which hopefully alleviates some of the frustrations that might arise from having to wait in lines. A user should be able to find the fastest checkout line in less than 5 seconds, without having to analyze the lines themselves. This requirement was determined mainly from personal experience and briefly surveying people around us, as people generally decide which line they want to go to in order to check out relatively quickly. They are more likely to want to use the system as long as it decides which line to enter faster than they are able to decide. They should approach the checkout area, see a display that has the current fastest checkout time, and decide to go to that line.

2.1 Business Satisfaction

Our system increases the speed at which customers are able to check out of the store, and this could allow for stores to become more popular due to customer satisfaction with checkout times. With increased popularity, a given store using our system could increase revenue and by extension profit. Also, with faster checkout times, a store might not need as many cashiers to man the counters, which would also increase store profits due to cost cutting.

2.2 Ethical Implications

With regard to public health, a more pleasant shopping experience could encourage people to shop for their own groceries more frequently, which tends to be healthier than eating at restaurants. This also benefits public welfare, since our system aims to make the basic need of buying groceries more convenient. Where safety is concerned, the use of cameras in our system could be concerning for the customers. However, the only data we aim to collect about people is the number of them (and not who is) in a line at a given time, and we will not be storing any significant amount of camera footage. With respect to the cameras in our systems that observe how fast cashiers work, they only point towards the hand level, and so no faces are being recorded and used for data processing or facial recognition purposes. Furthermore, people are already monitored in

grocery stores through security cameras, so our additional cameras will not be capturing any information that would not already be known for security purposes.

2.3 Consumer Focus

Individuals are bound to have a better shopping experience where they are able to save time at the grocery store. People are more likely to return back to the grocery store and make more visits if they have a positive shopping experience, almost fostering a sense of community at the grocery store. Many demographics of people don't get a lot of social interaction, such as the elderly, and a positive grocery store experience provides an opportunity to foster that sense of community and increase social interaction between such groups. Integrating a system that decreases the time it takes to get to the checkout counter also decreases the overall time a customer spends at the store, which is more time to spend elsewhere that can be directed to bettering society.

3 ARCHITECTURE

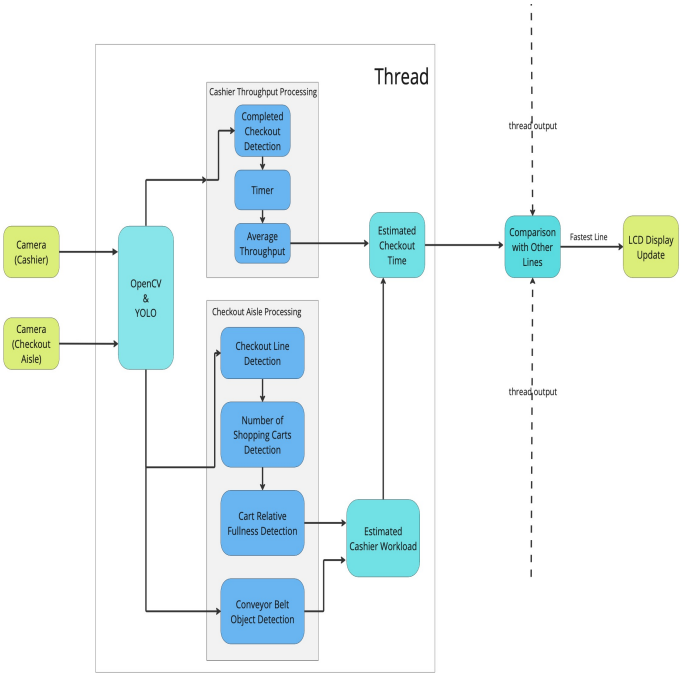

Figure 1: Software Architecture.

The three main components for our system are the cameras at the checkout lines (see Fig. 2 for the layout), a laptop running all of our algorithms, and an LCD display to inform users of our recommended checkout line. Our system begins with receiving input data from two cameras for each checkout line in the store. Then, it will process the camera footage using OpenCV, and send data to three different software modules: data from the camera facing the cashier will be sent to a software module that will calculate the throughput of the cashier to determine how fast they are working. This will be done by using YOLO to see when an item goes from the beginning of the checkout area to the bagging area, and keeping a running average of the time that it takes between items leaving the checking area and reaching the bagging area. The data from the second camera in the checkout line will be sent to the second and third modules, which will find the number of people lined up for the corresponding checkout line, and keep track of how full item carts are and the number of items on the conveyor belt in the checkout line. Information from all three of these software modules will be subsequently inputted into a local database. This local database will store all computed data from each thread, which runs our main software module that calculates relevant data for each checkout line. Our local database will then have for each checkout line: the throughput of the cashier, the number of people lined up, and the number of items on the conveyor belt and how full the carts are of the people behind the conveyor belt. Using this information, our system can then compute estimated times for how much time it will take for a cashier to finish checking out all the current customers that are lined up in their checkout line, which will then be displayed on an LCD display.

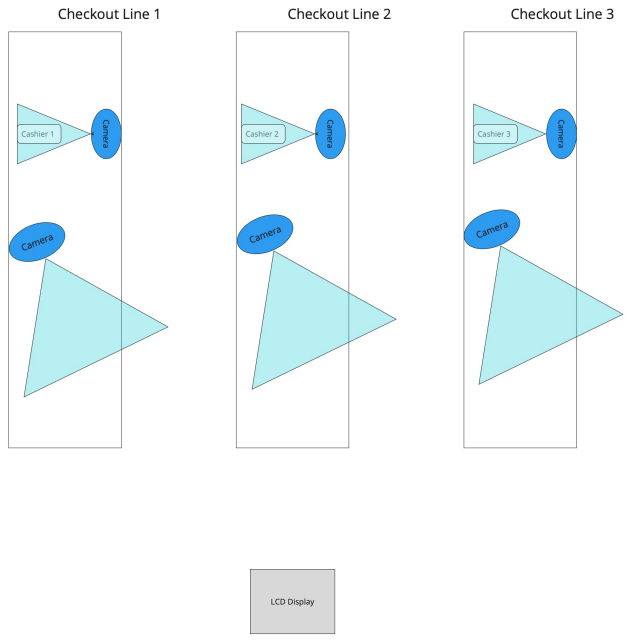

Figure 2: Physical Camera Layout

The physical layout of our system has two cameras per checkout line: one will view the cashier, and the other will view everything else: the items on the conveyor belt, the people waiting in line behind that conveyer belt, and their shopping carts and/or baskets.

4 DESIGN REQUIREMENTS

Latency for processing the frames from camera footage in the backend should take less than 500 milliseconds. Our system needs to compute a result within 5 seconds, and processing the frames from the camera footage should not take up much of that time budget. Furthermore, processing the frames is a minimal portion of the total work in computing a result, so it should not take too much time to do this regardless. However, this does take 10% of the time budget because it is nontrivial to process the multiple frames per second being input by the camera.

The computation time for our computer vision algorithms should take less than 3.75 seconds total. This is the bulk of our system implementation and where the most work is being done. There are multiple algorithms that will run in parallel and this generates a significant amount of overhead. Therefore, it should take a majority of the time budget from our 5 second deadline in computing a fastest checkout line recommendation.

Furthermore, the time to update the LCD display after computing a fastest checkout line recommendation should take less than 250 milliseconds. Updating the LCD display is the simplest part of our system implementation and therefore should take the least amount of our time budget.

Our system, most critically, must produce a result that is accurate at least 90% of the time. An accurate result is defined as a user getting to the checkout counter faster than if they were to join any other line, which also means that a user approaching the checkout area after that user also must not get to the checkout counter before them. The entire purpose of our system is to find the fastest checkout line and allow a given user to get checked out as soon as possible, so accuracy is by consequence the most vital design requirement of our system. Without accuracy, our system is effectively useless.

For our algorithm that finds the average throughput of the cashier, we must have a maximum margin of error of 10%. This margin of error is simply defined as the percentage difference between our estimated cashier throughput and the real cashier throughput. Our system has cameras that monitor each cashier as they scan items for checkout, and these cameras will have a clear view of all the items (with the rare case that they are stacked on each other), and will therefore be able to figure out when an item is being checked out with a high degree of accuracy, since a cashier generally scans one item at a time when checking someone out. Based on this average, we are able to determine how much time the entire line will take to get checked out, which is extremely important to our use case. This increases the importance of having an accurate computation of the camera stream.

For our algorithm that determines the fullness of the carts, we must have a maximum margin of error of 20%. For this metric, percentage is relative to shopping cart capacity, so an estimate of 20% fullness when true fullness is 10% would be within our margin of error, despite our estimate being double the true fullness. Our system has cameras that view the carts in a line, but cameras might not be able to capture a very detailed view of each cart. Additionally, current edge detection algorithms may not be able to perfectly measure and report the amount of area the items take up in the cart, so a high accuracy can not be guaranteed. Therefore, our error requirement is more lax in this case because there are many ways that the camera view can lead to errors in detecting how full a cart is. For example, someone could have a cart where the side closest to the camera has many items, but the other side of the cart isn't filled at all. With edge detection, it is entirely possible that our system would predict that the cart is filled more than it is in this case. While we have a large margin of error, this won't impact our system's overall accuracy too much because we only need our system to observe a general trend of how long the line will take. As such, one cart's estimate being off by 20% will be averaged out by other carts in a line, likely leading to less overall error than 20%. We don't plan on displaying any quantifiable data that we collect to the users, so small inaccuracies here will likely go unnoticed by them.

Lastly, the algorithm that counts the number of shopping carts lined up must have an accuracy of at least 95% (where accuracy is simply detecting the right number of carts). This requirement is the most stringent for two reasons. Firstly, incorrect detection of the number of carts is going to heavily impact our estimated time. Secondly, we need to be able to detect a cart first before detecting the fullness of a cart, so successfully detecting cart fullness is predicated on accurately detecting carts.

5 DESIGN TRADE STUDIES

5.1 Item Counting Methods

One approach we had was to detect the number of items in every user's cart using a sensor, such as an ultrasonic sensor. The idea was that every time a user dropped an item into a cart, it would update the count in a database that would then be used to determine the number of items in a checkout line once the carts were checking out and standing in line. The benefit to using sensors over computer vision was mainly in accuracy. Sensors would be more accurate than computer vision because issues like items being stacked on top of each other or being behind other items would not be relevant with sensors being used. However, we found this solution to not be scalable as it would require multiple sensors and bluetooth connectivity to communicate the number of items to the overall system and process the overall time the line would take. We would need multiple sensors to be attached to every cart, and this would be prone to damage as users could easily and accidentally hit the sensors. Building an apparatus to attach to each cart and mounting it properly, making sure it doesn't obstruct a customer in any way, as well as making sure it isn't fragile quickly becomes too expensive as the number of carts and baskets in a store increase, making this approach cost

inefficient and inconvenient.

5.2 Computer Vision Algorithms

We have evaluated several different computer vision algorithms to use in our system, and we finally decided on YOLO due to its ease of adaptability with OpenCV as well as its speed at identifying different items in a single layer. Another algorithm we were considering is faster R-CNN. A benefit of R-CNN over YOLO would be its ability to handle larger amounts of object classes. However, there is a large time delay in the proposition of different objects, which is one of our key design use cases to count the number of different objects. Furthermore, R-CNN is much more resource intensive, due to the need of a larger dataset to train the algorithm. Another algorithm that exists is called Histogram of Oriented Gradients. This algorithm is much more accurate than edge detection, since it uses magnitude and angle in its computations. Unfortunately, this algorithm is too time consuming for our use case, as part of our use case is that our system updates in less than 5 seconds. YOLO is a faster algorithm since it detects objects in a single pass through an image, making it a more suitable algorithm for our needs.

5.3 Processor

One of our original plans was to offload the computer vision computation onto an FPGA to speed up the process. We planned to implement this hardware acceleration using OpenCL, a framework that allows for converting C++ code into a synthesizable RTL design. The main advantage to using an FPGA is that computer vision algorithms are highly parallelizable, so we would be able to achieve significant speedup by parallelizing the algorithms on an FPGA. Our planned approach was to take each of the software modules, convert each of the portions that required speedup to OpenCL friendly code, and then subsequently convert and synthesize the resulting RTL implementation. The reason why we didn't further pursue using an FPGA for hardware acceleration lies within our use case: there is no need to use an FPGA to meet our timing requirements, as our timing requirements are quite lenient and a laptop's CPU would suffice. As such, there is no need to incorporate an FPGA into our design, as the benefit it provides would be minor relative to the time it takes to develop. To focus on our user use cases, we will be computing our software on a laptop.

5.4 User Analysis

Another previous design feature we were considering was inspired by TSA lines. The idea was to section off an area for the user to enter before they receive the decision for which checkout line to enter. This design The purpose of this sectioned off area was to have a camera that would analyze the user's cart before the user entered any line, and that data would then be stored in the database to use in evaluating the lines once the user enters a line. We found that this design was not very beneficial for our use case as users would not be motivated to enter a designated area and wait for the system to capture their data before they go check out. If anything, this might increase the time and create a greater inconvenience for the user when trying to the checkout from the grocery store, driving them away from visiting the grocery store again.

5.5 Cart Item Counting

Before our current planned approach of determining the relative fullness of a cart, we were planning on differentiating between items in a cart based on gaps and borders between them. We could then calculate the time to scan the entire cart by multiplying the estimated item count by the throughput of the cashier. However, a huge flaw with this method was that because of our limitations with the number of cameras per checkout line, the camera that faces the lined up individuals for checkout cannot detect the number of items in a cart with high enough accuracy. Many things can throw off the estimation: for example, items can be behind other items, and there could be items stacked on top of each other, making the implementation for estimating the item count extremely difficult and unreliable. A camera facing a cart on one side that has many items blocking the other items from the line of sight of the camera will not be able to detect at all the other items, which would lead to a very noticeable hit to our system's accuracy. Therefore, we decided to change this approach and look for relative fullness of carts instead.

6 SYSTEM IMPLEMENTATION

The stack will have multiple modules in communication with each other as well as the system database.

6.1 Camera 1

We will have one camera dedicated to capturing the cashier's throughput, which will observe the time it takes for a user to finish checking out. To determine how long a single user takes to check out, we will use a timer and computer vision. Once a previous user is done checking out, which will be identified by when the user is a certain distance away from the checkout counter using YOLO, the next user's timer will start. Once that user is done and also identified as finished checking out, the time it took for the user will be recorded. Based on the data we already have on the relative fullness (see the section under Camera 2 for a definition of relative fullness) of the cart, we can determine the average time it takes for a unit of relative fullness to be checked out (Which is simply relative fullness divided by the time taken). The calculated cashier's speed for that user will be used to update the rolling average of the cashier throughput at that checkout counter.

6.2 Camera 2

Another camera will be used to determine the length of the checkout line, as well as used to determine how "full" each cart is. Each cart will be saved as an object as part of a queue in the database for the relative checkout counter. Each cart object will store the relative fullness of the cart, and based on how long it takes on average for each unit of fullness to be checked out, the software will compute the estimated time that cart will take in the line. Based on the queue in the database for that specific checkout counter, we can observe when a cart hits the top of the queue, and based on when the cart is done checking out and what relative fullness the camera and system had observed for that cart, we can update our estimate for how much time it would take for each unit of fullness to get checked out, and therefore update our system's accuracy for projected time.

Relative Fullness - Our system will measure this metric using edge detection to determine the general area that the items in the basket cover, and look at the edges of the basket itself to determine what the max area of the basket can have. From those two measurements, we can then proportion and estimate how much volume the items in the basket are taking, and have a measurement of "relative fullness" to use for our basket. We will know which lines to observe based on the detection of where the cart exists via YOLO, and from there be able to filter out the edges that are observed. Figure 4 below shows how edge detection can help create a bounding box (blue box) around the items in the cart, from which it could be compared to the top edges of the shopping cart to determine the relative fullness of the cart.

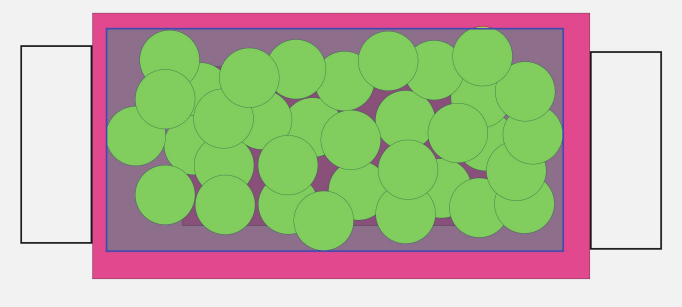

Figure 3: Shopping Cart with Bounding Box

Line Detection: A large part of our system also relies on our camera being able to determine where the checkout line is and whether a cart is in that line or not. If we are able to identify the checkout counter and the line of carts closest to the counter, we can start to form the concept of a line there, and based on the direction of said line, we can start to observe a line following out of the counter.

6.3 Output

Once the data from the cameras is processed, another module will look at the data and compute the total estimated time the checkout line will take. For each checkout line, we will have the cashier throughput, the number of people in line, and the relative fullness of the carts/baskets. Based on the relative fullness of a cart, we can create a metric to determine approximately how much time it would take for a cashier to finish scanning someone's cart, which then allows for the calculation of the total amount of time to finish checking out all of the people currently waiting in line.

$$TotalTime = CashierAvg \times \sum(RelativeFullnessOfEachCart \quad (1)$$

Total Time = cashier avg time * sum(relative fullness of each cart) Once this number is computed for each checkout line, the line with the lowest projected time will be determined as the result. This result will then be displayed on a separate window on the laptop, which will be connected to an LCD display via an HDMI cord, that will also display the output.

7 TEST & VALIDATION

7.1 Cashier Throughput

The most basic method for validating the cashier throughput calculating module is to have a camera pointing at a table, and lifting items from one end to another and seeing if the throughput is calculated correctly. We plan to manually time when the tester lifts and puts down the items, and then calculating the error of the throughput calculation module will indicate the accuracy of this module. We will repeat this process on different items and plan on testing 5-10 times for each item in order to get a more complete understanding of this module's accuracy.

7.2 Item and Line Identification

We plan to test our item counting module and line identification module using manual tests where we will set various scenarios of different numbers of people entering lines with varying carts. We will start with a small number of lines and total number of items in carts/baskets, and scale our tests, starting with two lines. We also plan on having tests where a person/people are standing in the field of view of the camera, but aren't actually lined up to checkout. This will allow us to check for correctness in our line detection module that counts the number of carts lined up. It is critical to accurately tell how many carts/baskets are lined up to check out because an inaccurate calculation on the number of carts/baskets in a line can drastically affect the correctness of the entire system when integrated. We will aggregate results from multiple different line configurations where there are extra people standing but not lined up in frame, and configurations where there are only people lined up in frame. After aggregating these results, we will calculate the error of the line detection module.

We aim to achieve less than 15% average error for the calculated results in this module. Furthermore, for item detection, we want to test how accurate our edge detection algorithm is with estimating the fullness of a cart/item. With a camera pointed at a line of people, we will look at the computed results for the fullness of the carts/baskets and then calculate the difference between our estimation and the actual volume of the carts/baskets, since the basket and cart total volume is standardized in stores, as a grocery store generally does not carry two different types of carts or baskets. We aim for a maximum margin of error of 20%, because there are many difficulties in getting accurate measurements of cart fullness due to stacked items or unevenness in item spread in a cart (a cart with many items on one side that the camera can see, but the other side of the cart is empty). Repeating these tests multiple times will help us get a more complete understanding of the accuracy of the relative fullness module.

7.3 Computation Speed of Integrated System

When looking at our integrated system, we want to make sure that it meets our design requirement of computing a result after processing camera footage in less than 5 seconds. We can do this programmatically by starting a timer when initially taking in camera footage, and then stopping the timer right before displaying the results on the LCD display. We want to aim for the timer result to be less than 5 seconds, and we will repeat this process 15 times in order to make sure that our system can compute results in less than 5 seconds consistently.

7.4 Accuracy of Integrated System

After our system is completely integrated, we plan on testing the system by having real-time testing in Scotty's Market, a grocery store located on campus in Forbes and Beeler apartments. We will mount our cameras in our described physical layout in Figure 2, and have our system compute consistently which line should be taken to a given user. The most vital component of our system is the accuracy of the computed result - we want a given user to be able to check out of the store faster than the person behind them. Therefore, we can observe people that use our system, and see how many times the users are able to check out before people behind them. The individuals behind the user of interest described above can use the system as well to get a recommended checkout line, but our system should not be recommending a checkout line that allows a later user to get out of the store faster than a user from the minute prior. We will repeatedly observe the usage of our system and calculate an accuracy measure of the integrated system after 20-30 trials, and we aim for at least 90% accuracy.

8 PROJECT MANAGEMENT

8.1 Schedule

By the week of 3/11, we hope to have the individual software modules completed or close to finished, and an integrated product by 3/26. Testing is the more vital part of our project, and we hope to finish integration testing by 4/5. The schedule is shown in Fig. 4.

8.2 Team Member Responsibilities

We split the responsibilities for each team member evenly, considering the difficulty of each relative subtask. Simon and Brian will be working on the line detection software module, and Simon will then test that module. Brian will work on the cashier throughput calculation module, and test that module. Shubhi will work on the cart relative fullness calculation module, and test that module. Everyone will work on combining the software modules, integrating the system, and integration testing.

8.3 Bill of Materials and Budget

See Table 1 on the next page for our system's budget.

8.4 Risk Mitigation Plans

Our most significant risk right now is getting permission from Scotty's Market to set up our system for integrated testing. Without permission, our testing will be much more difficult to accomplish. For risk mitigation, if we are unable to get permission to test our system at Scotty's Market, we plan on using a room on campus, setting up multiple lines using long tables, and doing grocery shopping simulations with baskets there. We will vary the line length, number of people in a line, and the throughput of those who are acting as cashiers. In doing this, we hope to at least simulate a relatively similar environment to a grocery store.

Another risk we foresaw when initially designing this system was the accuracy of the system to be able to identify how full each user's shopping cart would be. We decided to mitigate this risk by changing our metric from the number of items in the shopping cart to the relative fullness of the shopping cart using edge detection. However, even trying to detect relative fullness could prove to be difficult, considering that we are only using 1 camera to collect information about shopping carts. If the angle we put this camera at does not give us definitive enough data to make an acceptably accurate estimate, we plan to put one camera capturing a side view, and another camera capturing a top-down view. Combining these two cameras, we hope to be able to improve our measurements of fullness.

9 RELATED WORK

There is a project where a smart grocery store device is built using computer vision algorithms to detect and iden-

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
2K Webcam	V11	NURUOM	6	$26.99	$161.94
7-port USB hub	HU4133	vienon	1	$9.99	$9.99
LCD Display	M700	AISHICHEN	1	$42.99	$42.99
					$214.92

tify specific grocery store items and store them in an SD card, then display the scanned items and their prices on a user-friendly interface. [1]

This project mainly focuses on making a device that will give normal grocery store carts more functionalities like the Amazon Go smart shopping carts. Therefore, the fundamental difference between our system and this project would be the focus on streamlining checkout speed via looking at how full a cart is versus the focus on recognizing specific grocery items and showing all of the items in a checkout cart on a display screen. Our system will not focus on scanning individual items, and takes into account more so the fullness of a cart rather than the specific items inside of it.

10 SUMMARY

We hope to create a system that allows users to check out of a store faster by analyzing data that a user trying to check out of a grocery store wouldn't be able to see right away using footage from cameras. Our system will take into consideration the speed of a cashier in each checkout line. It will also take into account the amount of people in a line, more specifically the number of groups of people, since multiple people could be sharing the same cart. It will also take into account the relative fullness of each cart belonging to people that are lined up to check out, in order to provide a more accurate prediction on how much time it will take for a cashier to get through the line. The predicted fastest checkout line will then be displayed for a user to see (and follow). The upcoming challenge with implementing our system's components is mainly implementing an algorithm that will find the relative fullness of a cart and then subsequently assigning a metric to the cart in order to estimate how fast it would take to check the cart out. Relative fullness algorithms can be inaccurate due to many factors, and so assigning a metric that will smooth out these errors is the most vital and challenging part of implementation.

Glossary of Acronyms

- FPGA - Field Programmable Gate Array
- OpenCL - Open Computing Language
- OpenCV - Open-Source Computer Vision Library
- R-CNN - Region-Based Convolutional Neural Network
- RTL - Register Transfer Level
- YOLO - You Only Look Once (algorithm)

References

[1] Kutluhan Aktar. *Smart Grocery Cart Using Computer Vision - OpenMV Cam H7*. URL: `https://docs.edgeimpulse.com/experts/image-projects/smart-grocery-cart-with-computer-vision-openmv-cam-h7#description`. (accessed: February 28th, 2023).

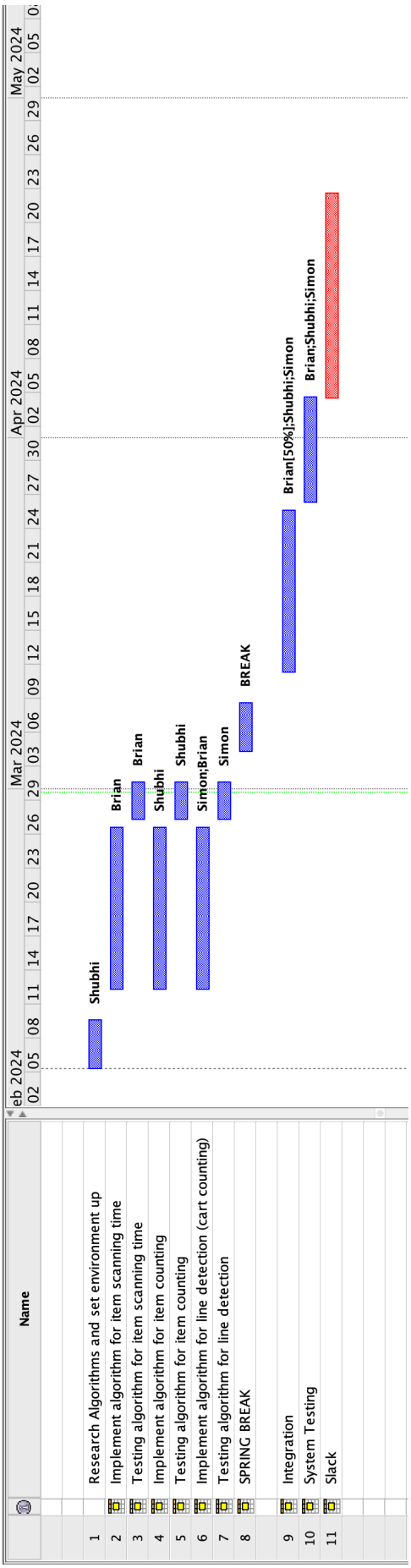

Figure 4: Gantt Chart