Zipcart

Ricardo Henriquez, EE Ryan P. Lagasse, CSE Jonathan Azevedo, CSE

***Abstract*— Checkout aisles in grocery stores are inefficient. There are better ways to process transactions that don't involve waiting in long lines for a cashier. We propose Zipcart: a system that employs computer vision techniques to keep track of the items in an order as a customer shops. It employs a centralized data store on public cloud infrastructure that contains order and item information. A smartphone interface is used by shoppers to audit their order and the balance of their selected items**

I. Introduction

Of all parts of the grocery shopping experience, the checkout process is the most needlessly long and frustrating. Once shoppers are done selecting the items they wish to purchase, they queue up for service from cashier attendants. This entails iteratively aligning the barcodes of items with a scanner, bagging the groceries, and navigating the point of sale (POS) terminal's interface to process payment.

Maintaining order information and payment processing are both tasks of the digital system. The cashier's part in this is helping the system identify each item in the order and selecting the appropriate means for payment. Using computer vision techniques, this first task could also be automated, as a system can be trained to recognize the items that shoppers select through passive observation. The second task could be entrusted to the shopper, so long as the supported methods are digital-friendly (nowadays, even checks can be processed by digital systems using computer vision).

There are many competing solutions in this space. Amazon Go [1] uses a large amount of cameras embedded in their stores to track what customers take out of the store, charging them accordingly. This approach is very costly and requires the type of infrastructure and technical expertise that only a company like Amazon has. Peapod [2] is one of a class of websites that offer grocery delivery as a service. Consumers avoid trips to the store at the expense of service charges. Additionally, articulating *which* items (out of a similar class of items) can be difficult, given the user interface and things like sales and coupons for shoppers to consider. Both solutions require large changes to the landscape of the grocery store. For Amazon Go, it's the addition of cameras and sophisticated infrastructure to process video feeds. Sites like Peapod either need to employ warehouses for groceries or alter the layout of stores to make it more efficient for "grocery pickers" to sweep through the landscape and fulfill orders. Zipcart necessitates retrofitting shopping carts – not the store. This keeps the customer shopping experience relatively unchanged and makes the cost of renovation less expensive in terms of cost and time.

II. Design

A. Overview

Our project seeks to present a more efficient alternative to the checkout aisle. It starts with a system embedded onto every shopping cart, with a camera mounted onto it. That camera identifies items as they enter or exit the cart by reading their barcodes. Once the system reads the barcode, it passes that information onto a service running in public cloud infrastructure (e.g. Amazon Web Services), which keeps track of orders and item information in a database. The shopper will use an interface on their smartphones (e.g. an Android application) to audit the order balance and items selected as they shop. For the future of our project, this would enable us to process payments through services such as Google / Samsung / Apple Pay, Venmo, and PayPal.

There are some interesting challenges that arise when trying to utilize computer vision for this purpose. First, we need to scan the barcodes of items as they enter the cart. Since we do not require shoppers to put items into the basket in a specific way, they enter the cart at unpredictable angles and speeds, and yet we still must be able to read the barcodes with a high rate of accuracy. Furthermore, the barcodes must be read in a timely manner. Computer vision algorithms are highly resource-intensive. The system running these algorithms has the heavy workload of processing image frames and detecting not only the barcode but direction of each item for the duration of each shopping trip. This must all be done nearly in real-time, as large latencies may confuse or upset shoppers who expect to see the current state of their orders on the interface as soon as possible.

As we are not acting as store facilitators, we do not maintain our own database of items and their relevant information, yet we want our system to be able to scan and account for any item with a barcode. As such, another one of our challenges involves getting this information from an external source quickly. When the web service happens upon an item it has no information on, it must request for and cache it in a manner that complies with the real-time constraints of our system. Lastly, our systems must operate over the course of entire business days. To avoid

R. Henriquez Jr. <ricardohenri@umass.edu>
R. Lagasse <rplagasse@umass.edu>
J. Azevedo <jazevedo@umass.edu>

requiring maintainers to charge carts throughout the day, we utilize the mechanical energy exerted by shoppers pushing the cart to power our system. This necessitates the design of an efficient subsystem that can generate an adequate amount of power to sustain our system with the mechanical potential it sees.

To meet the expectations of each of our stakeholders, we set the following project requirements:

1. Recognize barcode as item is placed in cart
2. Detect when item is removed from cart
3. Display item list and current balances
4. Detect unscanned items to prevent theft
5. Sustain power for a full business day

To create this design, we set a number of assumptions about our project. These are our specifications:

1. One item entered or removed per two-second interval
2. Barcode surface must be reasonably flat
3. Maximum system latency of four seconds
4. Eighteen hours of continuous operation

Residing in the appendix is an enhanced block diagram made up of two figures that illustrate the topology of our system in addition to the function of each component and the relationships between them.

B. Power

We want our system to be self-sufficient in order to avoid having to plug it into a wall outlet – having to do so multiple times over the course of a day would be impractical. Our approach is to take advantage of the mechanical motion of the shopping cart and convert that into electrical energy which in turn will charge our lithium ion battery. The project requirements, state that our system must operate for a full business day, which means the power generated must be greater than the power consumed. Figure 1 shows the schematic diagram of our power circuit. The diagram is separated into four pieces which are the stepper motor, full-wave rectifier, voltage regulator, and the Adafruit Powerboost 1000C [3].

The first piece is the Vexta PX245-02B-C8 stepper motor which is two-phase and rated each at 6V and 0.8A. We decided to use a stepper motor because they work best at low speeds and in our application, the average walking speed of a customer is 3mph which equates to roughly 200RPM. Heading into MDR, we only use one stepper motor as a proof of concept, which plays the role of converting mechanical energy into electrical energy. The power generated by a stepper motor is proportional to the rotational speed of the motor shaft in the form of AC.

The second piece is converting the AC generated into DC and we do this through a full-wave rectifier made up of 1N5818 schottky diodes [4]. Our stepper motor has two phases, each phase must be rectified which is shown in figure 1. We added a 100uF capacitor at the output of the rectifier to help reduce voltage variations.

This rectified voltage then feeds into the Pololu S7V7F5 voltage regulator [5] which takes the input voltage between the acceptable voltage range of 2.7V-11V and efficiently converts it into a stabilized 5V output. We chose this regulator because it has the capability to step-up or step-down the input voltage, which makes it an ideal choice in our application since the voltage the stepper motor generates varies dependent on speed. This regulator can source up to 1.6A which gives us plenty of headway, since we do not expect to generate more than 1A. Figure 2 shows the efficiency of this regulator sourcing current at different input voltages. In our case, the efficiency depends on how much voltage we generate at the input which is dependent on the speed of the shopping cart.

The last piece is the Adafruit Powerboost circuit, which has several necessary features. It has a built-in load-sharing battery charger circuit which allows the Raspberry Pi to run while charging the lithium-ion batteries. It also features a built-in battery protection circuit which is necessary when charging lithium-ion batteries to protect from overcharging them. This circuit can recharge at a max rate of 1A and allows the battery to output more than 1A if required.

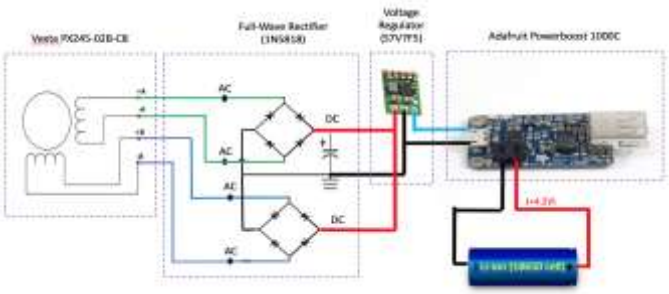

Figure I: Power circuit diagram

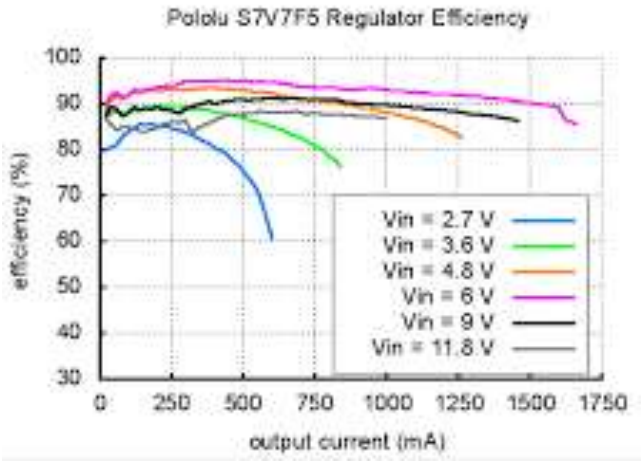

Figure II: Pololu Voltage Regulator Efficiency [3]

To build this power circuit, we had to leverage the information we learned in ECE 211 and 212 to understand the general rules of putting together a circuit and the role each element plays. ECE323 and ECE324 equipped us with the knowledge to understand full-wave rectifiers and voltage regulators.

The single most important aspect of the entire power circuit is the stepper motor. None of the courses I have taken covered stepper motors being used as generators so that is still a grey

area and must spend more time researching.

For demo day, we constructed a separate circuit not shown in the power diagram for testing purposes. This circuit was designed to simulate the movement of a shopping cart. We used an Arduino Micro, LM298N motor driver, and an extra stepper motor to drive the primary stepper motor. We coded the Arduino to rotate at a maximum and constant speed of 200RPM. This experiment will help us test different stepper motors and reliably measure the amount of power produced at varying speed.

The results we are primarily focused on is the amount of power produced by the stepper motor. As mentioned, when the motor shaft is rotating, current is produced. The rotational speed of the shaft is proportional to the amount of generating current. After assembling the entire circuit and presenting the load of the battery and impedance of the Adafruit Powerboost 1000C circuit, we drive the primary generator motor with the Arduino simulation motor at maximum speed. Figure 3 is a plot that shows the relationship between I-vs-RPM we presented during MDR. This plot indicates that we produced a maximum of 180mA at 200RPM using only one motor which equates to roughly 1W generated.

Our goal heading into CDR is to increase the amount of power generated to increase the charging rate of the battery. We plan to do this by wiring four stepper motors (one stepper motor for each wheel on a shopping cart) in parallel which should ideally quadruple the current. If four motors in parallel is not feasible then we must consider different motors with larger ratings per phase. It is worth mentioning that the Raspberry Pi will consume 2.5W/h on standby and 4W/h with all peripherals plugged in. We obtained these values by taking measurements with a KILL-A-WATT power strip. Overall, to consider our system being self-sufficient means the power generated must be greater than the power consumed.

Figure III: Vexta PX245-02B-C8 (I-vs-RPM)

C. Optics & Detection

The detection subsystem is the main portion of our embedded system, which reads the barcodes of items as they enter the cart. The detection subsystem consists of two parts: optics and software. The optical part includes a camera placed on the far side of the cart facing inwards in addition to mirrors aligned around the perimeter facing inwards as well. Once an item enters the system and is seen by the camera, the detection algorithm processes the video input, using image processing techniques to locate the barcode in the image and extract the UPC code from it.

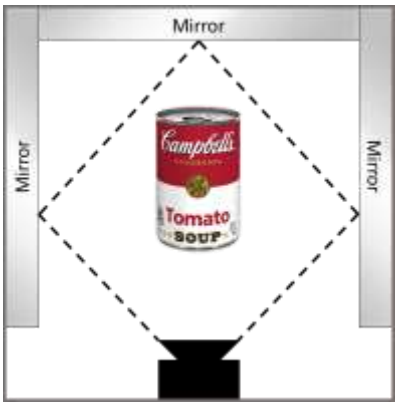

Figure IV: Camera and Mirror arrangement

The optics system is the most important part of the detection subsystem as it is the greatest bottleneck. The camera must be able to capture the barcode in its frame no matter orientation or distance in respect to the camera. Cameras have a limited field-of-view (FOV), which the barcode must be within to have a chance of being recognized. For this reason, we use mirrors to reflect the item back into the camera's FOV allowing us to extend our angle of detection around the barcode instead of being limited to placing objects flat in the direct line of sight of the camera. The camera must also have a high enough resolution to detect the spacing between the lines of the barcode. Since it has to do this at variable lenses it must also have an adjustable focus. Another requirement for the camera is to be compatible with our software libraries, which all cameras connecting to the Raspberry Pis CSI interface should be. When using the Raspberry Pi there are limited camera alternatives for its CSI interface, because of it we chose the best one we saw fit which was the Kuman 5MP camera [12] with adjustable focus that can stream 1080p at 30 frames-per-second (FPS).

For implementing computer vision, we use two libraries jointly. The first being OpenCV [13] an extensive and powerful open source computer vision library which provides us with a plethora of visual processing tools. The second is PyZBar [14] - the library we use jointly with OpenCV to detect barcodes entering our video frame and to extract the UPC code to send to our database.

Detection Procedure

1. Request an order to be made in the database
2. While true:
    a. Detect barcode in video stream from optics system
    b. Extract UPC code from barcode using PyZBar
    c. Sends UPC code to cloud service

Techniques from CMPSCI 503: Embedded Systems will greatly help in designing and building this system. In 503 we used computer vision using OpenCV to track lanes for a self-driving robot. Jonathan took this course and learned

implementation and debugging of real-time embedded computer vision systems. The experience acquired from this course in terms of knowledge on OpenCV tools can be used to better implement code to detect barcodes with minimal delay and processing power. Skills in debugging are always helpful in overcoming issues and reducing development time.

To continue with this system, we will have to learn better techniques to reduce delays and extend the visibility with our camera. This will mostly be in terms of better implementations of CV and possibly machine learning to supplement the system to improve performance.

The system must feel natural and require little user supervision. To test the system, we must run timed tests in which we see how fast or slow we can enter items in a cart at any given interval without requiring multiple user scans per items. By determining the interval and speed at which we can place items through our system we can compare this to a natural pace of an average shopper at a grocery store to determine whether the system is sufficient enough to not interfere with the users shopping experience

D. Cloud Infrastructure

The web service for this project takes barcodes from the embedded system as input and uses them to manage both orders and item information in a database. To build this service, we leveraged the offerings on Amazon Web Services' public cloud infrastructure [6]. Web requests made to the service and responses from it pass through the API Gateway, which handles and routes this information with the context it is given. Our system logic operates on Lambda [7], a "serverless" computational platform. It has a Python 3 runtime just like our embedded system. Lambda manages DynamoDB [8] (our non-relational database platform) and makes requests to an external barcode API. To interact with DynamoDB, we use a specially-designed object mapping library called Bloop [9]. At the moment, we are utilizing an API called Barcode Lookup [10], although our design allows us to change vendors with minimal effort.

Concepts from *Software Intensive Engineering* course were beneficial in the process of designing this subsystem. In order meet our latency specification, we evaluated a number of ways to tackle this problem. Leveraging public cloud infrastructure was a glaringly obvious solution, but what remained to be seen was the manner in which we would instrument platforms to meet our requirement. Through experimentation, in consideration with what we had learned, we determined an optimal solution for our needs. We also found the mindset of writing good software – gained from various assignments – to be useful in our implementation. Taking care to handle cases for errors and exceptions, considering performance and scalability, testing code well, and writing good documentation were all practices that improved the development process of this subsystem.

In order to test this subsystem, we performed a great deal of integration testing. This involved having the system perform mock actions to the database and manually verifying its correctness. In getting this system ready for our Midway Design Review demonstration, we accrued a good deal of technical debt. What is really lacking from the subsystem that would better ensure its effectiveness is unit tests and local integration testing. We plan to implement this using a unit testing library called nose [11] and using a local and self-contained version of DynamoDB.

When we perform our integration tests on AWS infrastructure, it returns the runtime of the code in milliseconds. By performing numerous trials, we determined the average runtime of our system for both a cache-hit and cache-miss (when we are required to get info from the barcode API), finding that the durations of both fit well within our specification. On cache-misses, the expected runtime of our system (updating both the item information cache and order table) is 2.446 seconds.

E. User Interface

The User Interface is the portion of our system which will actively communicate with the shopper to relay important information like whether an item was successfully scanned. It will also allow the user to view and manage their orders while keeping track of malicious activity.

The feedback system consists of two meters of RGB 60 LED Dotstar LEDs [15] placed along the inner, top perimeter of the cart. This will flash green or red depending on the status of a scan.

We also chose to create an Android application as it is the most popular OS worldwide and the easiest to work with. Having the ability to easily download and access the Android Studio IDE gives us freedom and flexibility to take the app where we want to. The Android application enables the user to view their balance and the list of items currently in their cart nearly in real time. In addition, it provides us with the opportunity to integrate payment processing into the system.

Software Intensive Engineering will prove to be very helpful in this part of the project as well as we need to create clean, concise, sustainable code. Creating an application can be a hard process and bad code practice will only server to delay us more and cause more issues. It is very important that for this part of the system as their will be multiple programmers working on it that we must write clean, well commented, functional code and practice good software development techniques like correct usage of version control to complete this subsystem.

We will need to perform intensive integration tests covering the entire spectrum from optics and detection to the cloud and back in order to verify its effectiveness. The main goal of such an experiment would be to ensure full system functionality and minimal end-to-end latency. We will have to run both physical and software tests by first adding time logging to important functions in our code that communicate with different systems, the purpose of this would be to later determine the greatest bottlenecks in latency. We would then scan at least ten items, with duplicates and new items in batch, at our defined natural interval of 2 seconds. At this point, our system should be automatically pushing data to our application and we should be able to see the list of items, quantities, and the balance.

III. Project Management

#	Deliverable	Status
1	Detect barcode around front-face of camera perspective	Works up to 4”
2	Update cloud database with product information	Works, meeting specification
3	Successful integration of feedback system	Fully operational
4	Demonstrate power generation using stepper motor	Generates 180mA at 5V

Table I: Status of MDR deliverables

Our team consists of two Computer Systems majors (Ryan and Jonathan) and one Electrical major (Ricardo). As a result, we split the deliverables amongst the group by major, with the CSEs taking deliverables one through three and Ricardo taking deliverable four. As a small team of three, we had agreed that each of us would work primarily alone in order to make progress, and stay in close communication with the team with status and updates. This would allow us to each be flexible with our schedules and not have to meet as a group to make progress, while also allowing for the opportunity to meet when needed to collaborate or solicit help.

Ricardo worked on the power subsystem and his deliverable mostly alone and with great success. When issues arose or he needed validation for his work before continuing onwards, he arranged meetings with Professor Robert Jackson to talk about the circuits and electronic components of his design. In creating his demo, Ryan assisted him by loaning an Arduino and helping him out with the code.

Ryan also worked mostly unsupervised, completing the second deliverable in its entirety and that of deliverables one and three with some assistance from Jonathan. This included writing the detection code and integrating AWS for the MDR demo. Ryan also completed the website in its entirety. Getting the feedback system operational involved creating a circuit and writing code, both of which Jonathan helped to debug.

Ryan and Ricardo set up the basic layout for the mock shopping cart, which involved finding a suitable cardboard box, installing mirrors, and testing out detection with a stationary camera. Jonathan further expanded this by routing holes for cables and mounting both the camera and feedback system LEDs, making the mock system suitable for the demo.

IV. Conclusion

Team Zipcart is on schedule to complete our necessary goals. We have completed each of our MDR deliverables and demonstrated a functioning, minimal viable product for each part of our system. Each piece works at a generally acceptable state and is ready for enhancement going forward.

As we move closer to CDR, we have big hurdles to cross in terms of new system capabilities to develop but we are on track to completing these goals. We plan to greatly improve upon barcode detection which is currently the largest issue in our system, by employing improved cameras, image processing techniques, and machine learning algorithms. Power generated by the motors will also enhanced in order to increase output current and meet the power requirements for our system. We will also improve upon our user interface by incorporating a user Android application that can be used for real-time order monitoring.

If all goals are met by CDR, the team aims to optimize the User Interface’s ease-of-use and aesthetic appeal, in addition to tackling the problem of theft detection.

Acknowledgment

Team Zipcart would like to thank our advisor, Professor Wolf, for making the time out of his busy schedule to meet with us weekly and provide us with invaluable advice. We also want to thank our evaluators, Professor Krishna and Professor Aksamija, for critiquing our project and supplying us with thoughtful feedback. We would also like to thank Fran Caron, Professor Hollot, and Shira Epstein for their valued assistance.

References

[1] Amazon.com, Inc., *Amazon Go.* [Online]. Available: https://www.amazon.com/b?node=16008589011

[2] Peapod, LLC. *Peapod*. [Online]. Available: https://www.peapod.com/

[3] ada, l. (2018). *Adafruit Powerboost 1000C*. [online] Cdn-learn.adafruit.com. Available at: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-powerboost-1000c-load-share-usb-charge-boost.pdf [Accessed 20 Dec. 2018].

[4] Vishay.com. (2018). *1N5818*. [online] Available at: https://www.vishay.com/docs/88525/1n5817.pdf [Accessed 20 Dec. 2018].

[5] Pololu.com. (2018). *Pololu 5V Step-Up/Step-Down Voltage Regulator S7V7F5*. [online] Available at: https://www.pololu.com/product/2119 [Accessed 20 Dec. 2018].

[6] Amazon Web Services, Inc., *What is AWS*. [Online]. Available: https://aws.amazon.com/what-is-aws/.

[7] Amazon Web Services, Inc., *AWS Lambda*. [Online]. Available: https://aws.amazon.com/lambda/.

[8] Amazon Web Services, Inc., *AWS DynamoDB*. [Online]. Available: https://aws.amazon.com/dynamodb/.

[9] numberoverzero, “Bloop: DynamoDB Modeling.” [Online]. Available: https://bloop.readthedocs.io/en/latest/

[10] “Barcode Lookup Homepage.” [Online]. Available: https://www.barcodelookup.com/

[11] J. Pellerin, “nose.” [Online]. Available: https://pypi.org/project/nose/

[12] “Kuman 5MP 1080p HD Camera Module for Raspberry Pi For Raspberry Pi 3 model B B A RPi 2 1 SC15.” [Online]. Available: http://www.kumantech.com/kuman-5mp-1080p-hd-camera-module-for-raspberry-pi-for-raspberry-pi-3-model-b-b-a-rpi-2-1-sc15_p0063.html.

[13] OpenCV library. [Online]. Available: https://opencv.org/.

[14] NaturalHistoryMuseum, “pyzbar,” *GitHub*, 09-Jun-2018. [Online]. Available: https://github.com/NaturalHistoryMuseum/pyzbar.

[15] iPixel LED Shiji Lighting “APA102 Data Sheet.” Adafruit. [Online].Available: https://cdn-shop.adafruit.com/datasheets/APA102.pdf

V. Appendix

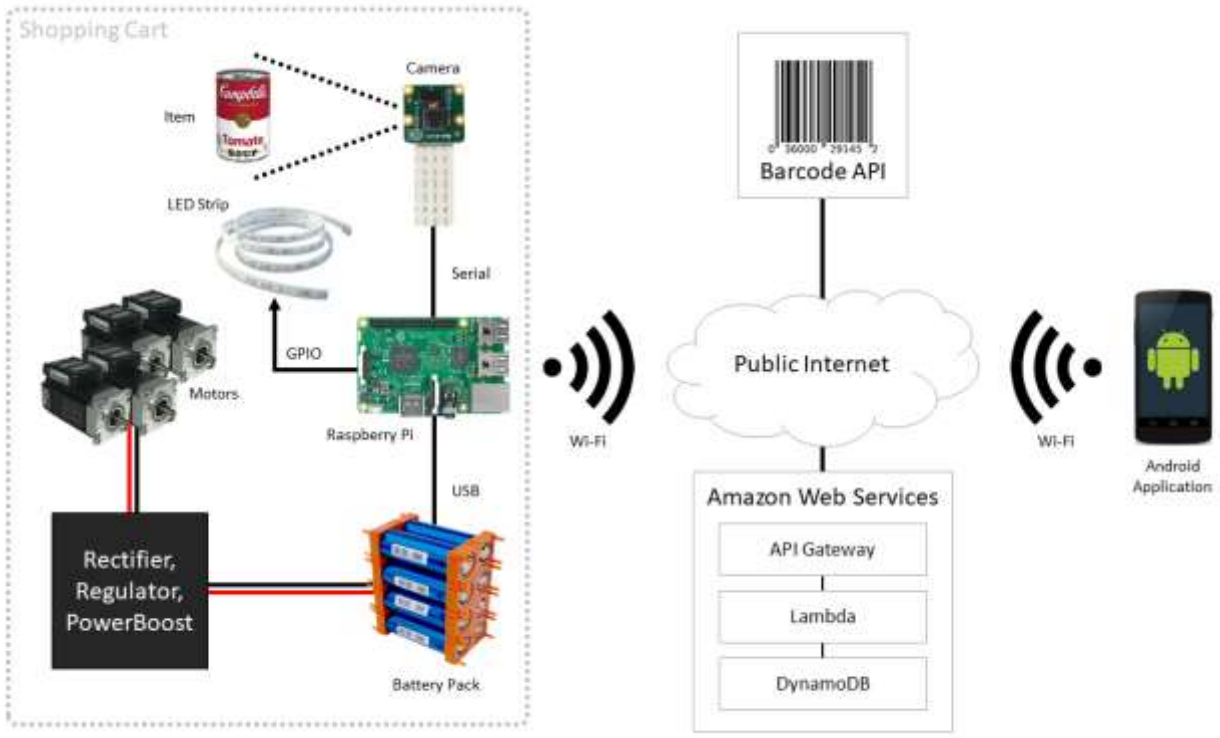

Figure V: System Topology

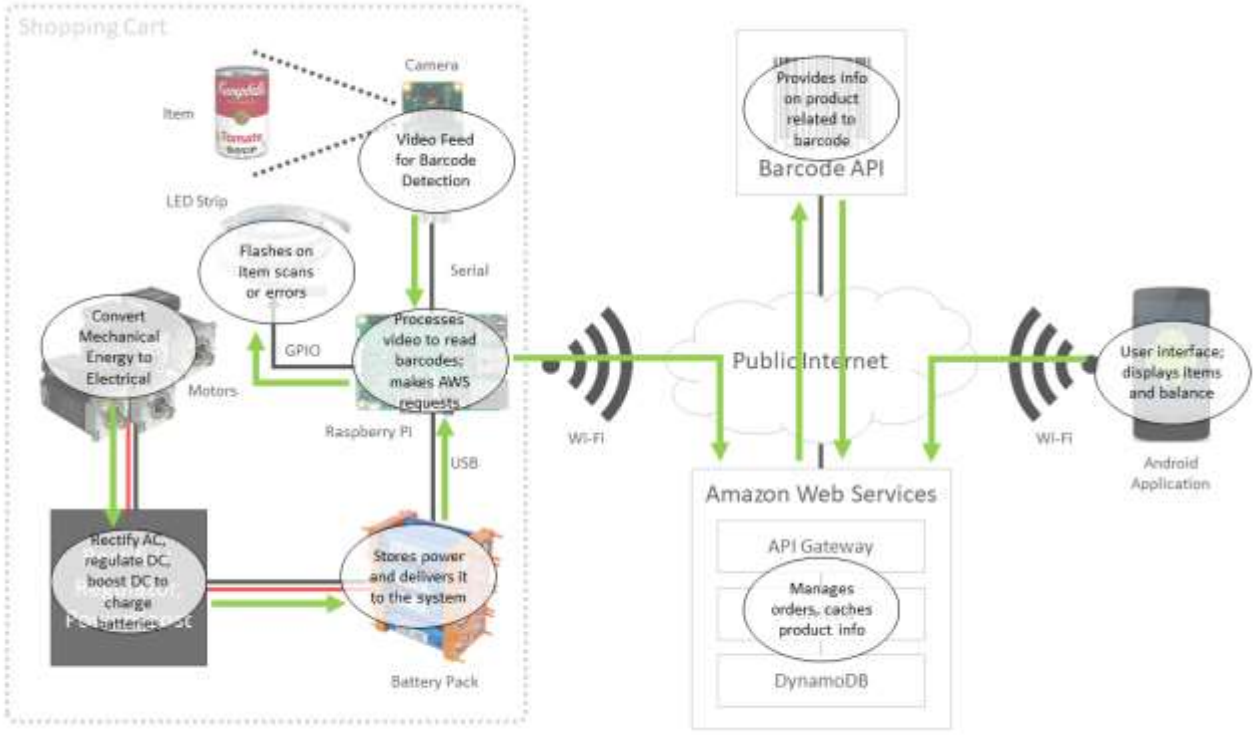

Figure VI: Component Functionalities & Relationships