18-500 Final Project Report: 05/14/2021

FarmFresh Design Review Report

Ishita Kumar, Ishita Sinha, Kushagra Sharma
Electrical and Computer Engineering, Carnegie Mellon University
{ikumar, isinha, kushagrs} @andrew.cmu.edu

Abstract- **It is important to separate rotten fruits and vegetables early in the food distribution process to prevent healthy fruits and vegetables from getting spoiled. It is essential that this is done before more monetary value is added through packaging and transportation. Thus, FarmFresh, a food produce quality evaluator and sorter, can help save energy and resources as well as help prevent food wastage. Our solution uses computer vision to autonomously sort and separate healthy and rotten fruits and vegetables, thus preventing human error. It can save hours of manual labour for farmers so they can utilise their time to work on other value-adding tasks that can't be automated.**

Index Terms- Computer Vision, Fruit classification, Machine Learning, NVIDIA Jetson Nano, OpenCV

I. Introduction

The saying, one rotten apple spoils the barrel is quite literally true. If not separated, rotten fruits and vegetables can spoil fresh ones and wreak havoc in the food distribution industry. It is also important to separate rotten food produce as early as possible before more monetary value is added through packaging and transportation. Thus, to save energy and resources and prevent food wastage, it is necessary for farmers to accurately separate rotten produce before further distribution. This task must be automated to prevent human error and save farmers' time. Although food produce sorting machines exist, they are very expensive. As 91% of US farmers are small farmers, there is a vital need for a more affordable yet autonomous solution [1]. This is why we introduce FarmFresh: a low-cost food produce quality evaluator and sorter. Our solution uses computer vision to quickly and autonomously sort and separate healthy and rotten fruits and vegetables. This prevents human error and can save hours of manual labor. This saved time can instead be spent on other more value-adding tasks that cannot be automated.

FarmFresh aims to be highly accurate in sorting rotten and fresh produce. Our project will work with bananas and carrots. We plan to achieve rotten produce detection with false negatives less than 2% and false positives less than 5%. Our solution is meant to be integrated into existing conveyor belt systems, so FarmFresh will be built to accommodate the typical conveyor belt speed of 5 cm/s with produce spaced at least 0.2 cm apart. Our product will take pictures in uniformly bright lighting to mimic factory settings and achieve consistency. FarmFresh will be powered to work for 8 hours at a time so it can last a typical working day.

II. Design Requirements

To ensure that FarmFresh is useful to farmers, we considered numerous requirements from the farmer's perspective. These requirements also ensure that our product is competitive in the market space, and offers something new that the alternatives lack. A list is presented below:

- Support multiple fruits and vegetables. We want to be able to support bananas and carrots.
- Needs to be compatible with a majority of conveyor belt architectures.
- Needs to be easy to install and portable. To validate this requirement, we built our own conveyor belt system, and made sure FarmFresh is fully compatible with it.
- Needs to quickly sort fruits, and needs to be able to work on existing conveyor belt speeds of 5 cm/s. No fruit should be missed.
- Needs to have a false negative rate of less than 5%. This is to ensure that we rarely categorize rotten fruits as fresh since that can potentially be catastrophic. To get a better understanding of accuracy metrics, we found several papers online. In "A Survey on Computer Vision Technology for Food Quality Evaluation", the author was able to achieve 97% accuracy classifying cracked eggs using edge detection [2]. In a more recent paper, the authors were able to achieve 89% accuracy correctly predicting rotten vegetables such as carrots and bell peppers [3]. This is why we chose a high classification accuracy.
- Needs to have a false positive rate of less than 15%. We define a false positive as follows: a fruit is predicted as being rotten, when it is not actually rotten. This is a less severe case. We do lose value from the misclassified fruit, but at least it doesn't impact the rest of the fruits. This is why we want our product to have stringent requirements when it comes to detecting rotten fruits, at the cost of potentially increased false positives.
- Needs to be able to support a typical 9-5 day, so the battery should last at least 8 hours.
- Needs to cost less than $600 so that it is cost effective.

Fruits are usually separated from each other at intervals of around 20 cm on a typical conveyor belt, though our product can handle fruits spaced as little as 0.2 cm apart. With the conveyor belt operating at 5 cm/s, our product can detect when a fruit is in frame 100% of the time without using much computation, so we use a live video stream to detect fruits in frame, avoiding unnecessary computation (thus saving battery life), while not missing any fruits.

The main considerations when designing the mechanical aspects (conveyor belt) were the following:

- The width of the conveyor belt needs to be large enough to support placing fruits.
- The length of the conveyor belt needs to be large enough to support placing multiple fruits (at least 3) so that we can perform stress tests and simulate the process better.

These two considerations allowed us to set the width of the conveyor belt at 10 inches (the average width of a banana is 2 inches), and the length at 26 inches, so that we can place more than 3 fruits at any given time at intervals of 20 cm (7.8 inches).

III. Architecture and/or Principle of Operation

The architecture of FarmFresh consists of two major components - the conveyor belt design, which is the mechanical component, and the rest of the system, which constitutes the hardware and software parts of the product. This can be seen in our block diagram in Figure 5.

As part of this split, we must note that the conveyor belt setup itself does not depend on any part of the product, and vice versa. Our product is designed to be integrated into existing conveyor belt systems that meet our speed specifications. As a result, the product is independent of the conveyor belt, so the two components can be built separately, and they operate separately as well. The functioning of our product would not impact the conveyor belt, and vice versa.

Figure 1 provides an overview of how our product is designed to work with the conveyor belt system. As a fruit or vegetable comes along, the cameras capture an image of the fruit. Once this image has been captured, it is sent to the NVIDIA Jetson Nano. The board runs our classification algorithm on the images of the fruit and determines whether it should be classified as rotten or good. It passes a signal to the gate controller, and accordingly the gate controller rotates the gate so that the fruit or vegetable is deflected into the appropriate basket.

The entire system has the base computer as the NVIDIA Jetson Nano. To the Jetson Nano, we have connected two USB webcams, which correspond to the cameras shown in Figure 1. Once the system starts, the cameras capture a live video stream of the conveyor belt. Once the image of the fruit has been captured by both cameras, it is sent to the NVIDIA Jetson Nano. On the NVIDIA Jetson Nano board, we have our classification algorithm. The algorithm receives both of the images, resizes them, and then analyses the frames using Gaussian blurring and Canny edge detection to see if a fruit or vegetable is in the frame, entering it, leaving it, or is not in the frame. If a fruit is detected in the frame, the images are passed into the image segmentation and pixel classification algorithm.

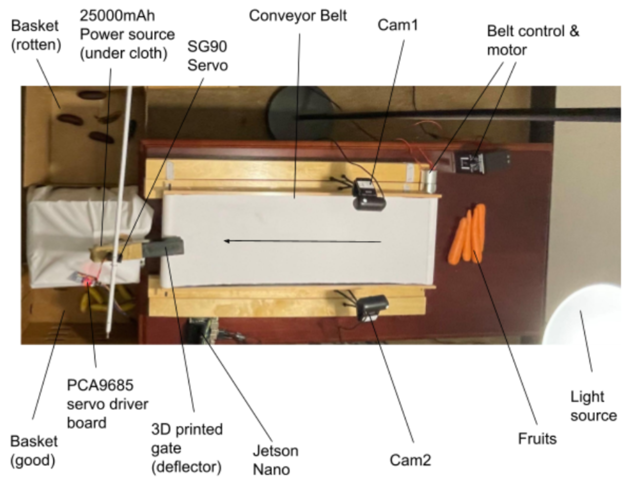


Fig 1. System picture

We have a white background for the fruits on the conveyor belt so that image segmentation can be performed with ease on the images captured. We used controlled white lighting so that all of the images can be classified more easily, since they'd all have similar brightness. The classification algorithm begins by converting the image to the HSV colour space. Post that, it performs image segmentation to isolate the fruit or vegetable from the background, obtaining the good and rotten masks for the produce. For a banana, the good mask would correspond to capturing the yellow and light brown parts of the fruit, while the rotten mask would capture the darker brown and black parts of the fruit. Once this has been achieved, we can pass the image into our classification algorithm that uses percentage thresholding in order to determine whether the fruit is rotten or not. We built 2 classifiers in order to see which one worked best. The first one involved performing a pixel by pixel analysis of the fruit in order to obtain the percentage of pixels corresponding to rotten parts of the fruit. We used a threshold to identify what percentage of the area must be rotten in order to classify the produce as rotten versus good. An alternative algorithm involved using neural networks (AlexNet) in order to predict the rottenness of the fruit based on a classifier that we developed and trained on several images. To perform all of this computation, we have used the OpenCV and Scikit-image libraries provided by Python.

Once the classification has been performed, the result is passed along to the servo driver. The servo driver is integrable with the Jetson Nano, so it receives the signal from the algorithm and facilitates communication between the Adafruit servo controller and the servo motor itself. Once the Adafruit shield receives the signal, it sends along the signal to the servo motor so that the gate can be rotated the appropriate amount in the appropriate direction in a specified amount of time. Based on the result of the classification, the gate rotates so as to put the produce in one of two baskets - the good one, or the rotten one, depending on the result of classification.

This entire product is designed to be integrated into an existing conveyor belt system. Our conveyor belt setup consists of electrical as well as mechanical components. The electrical components include the RC motor speed controller board and a 12 V DC Gear Motor for moving the conveyor belt at the appropriate speed. The mechanical components that are involved in building the conveyor belt are discussed in Section V.

This entire system is powered by a 5 V power supply. We need this entire system to last a regular 8-hour working day for the farmer, so based on the use case requirements, we have used a 25000 mAh power bank as our power source since it lasts for the requisite amount of time.

IV. Design Trade Studies

Hardware

A. *Fruit Sorting Mechanism*

For the final design of our fruit sorting system, we decided to use a servo motor to control a gate which can turn and direct the fruit into the appropriate basket. This is different from our initial design which involved pistons pushing fruit off a platform and into the right basket. We decided against the latter design because when we researched pistons, they were too slow to meet our required speeds. We looked into solenoid pistons as well, which were faster than the other types of pistons, but these were expensive. In addition, pistons were hard to configure since our team lacked the mechanical knowledge required. Furthermore, none of the pistons we researched had the necessary reach to push the fruit far enough to make it fall over the side of the conveyor belt. On the other hand, we all have experience working with servo motors and the gate system seemed to be a simpler design that met our requirements better. Thus, we chose the servo controlled gate system as our fruit sorting mechanism since it is faster and more easily configurable.

B. *Fruit Platform*

It was important for us to decide on a good system for placing our fruit as our image segmentation algorithm for fruits would be strongly affected by this choice. Our initial design involved a white rotating plate on which the fruit would be placed. We would then take pictures periodically as the fruit rotated. This would be good for image segmentation and we would only require one camera for this design. However, this design was not easily integratable with a conveyor belt system, which is our intended end goal. We also needed to test if our design would work with a conveyor belt, which was not accurately possible without moving the fruit in some way. Thus, we decided to build a conveyor belt and place fruits on it, as our end-user would, rather than using a rotating platform. This would provide a more realistic environment for our project. We cover the base on which the fruit is placed with a white material to still get the benefits for image segmentation. We still wanted to be able to access all sides of the fruit though, which led us to incorporate multiple cameras in the new design so that the fruit is captured from multiple angles. Earlier, we were thinking of using a PCB circuit in order to design the speed controller for our conveyor belt motor. However, we found a motor speed controller that was quite inexpensive and fit our needs very well, so we decided to go with that.

C. *Hardware Platform Placement*

Our design requires a platform to place the camera(s) for taking fruit pictures and the Jetson Nano for computation. We decided to use wooden planks on either side of our conveyor belt system to place our cameras since the height of these planks is easily adjustable, thus allowing us to configure the camera height properly with respect to the rest of our conveyor belt system. The conveyor belt has a motor attached to it that needs to be mounted on a platform so that it's held by something. Having the wooden planks on either side ensures that the motor is well supported. Lastly, we needed our conveyor belt to be lifted so that the material of the belt does not touch the surface it's placed on. The wooden planks are very useful here as well since they help support the conveyor belt above the surface. Earlier, we were thinking of building a shed to hold our cameras and the Jetson Nano, along with housing the light source. However, we realized that mounting the cameras in the shed could be quite complicated, and since our current cameras have tripod stands, this ensures that a shed is not required. Our design involves placing the cameras on either side of the belt to capture different angles of the fruit. We decided a 2 camera system placed on planks worked better than a single camera since it's much easier to integrate with a conveyor belt. The single camera system would have had the fruit rotating, so we would have had to stop the movement of the fruits, which is not ideal. The Jetson Nano is placed next to the conveyor belt. A uniformly bright white light source is held in position using a stand above the belt for consistency in our pictures.

D. *Computer*

We decided to use an NVIDIA Jetson Nano for our computer vision algorithms. This is because the Jetson Nano has a quad core A57 @ 1.43 GHz processor and a dedicated 128-core Maxwell GPU. This works well for our design as it has the computation power required for our algorithm to run quickly as well as to capture a live video stream. It offers a dedicated GPU for processing fruit images and computer vision. We also looked into using the Raspberry Pi but we did not think it would provide a computational power good enough for our project needs as it does not have a dedicated GPU. We chose Jetson Nano A02 over the newer B01 model (which has 2 CSI camera ports) because we wanted our design to be scalable to 4 cameras if we needed to capture more angles of the fruit to meet our accuracy metrics.

E. *Cameras*

The cameras are an important part of our design since we use them to capture a live video stream of the conveyor belt, capturing frames of the fruit as it moves on the conveyor belt. These images are then analyzed to detect whether a fruit is rotten or not. The cameras we'd select would also be connected to our computer, the Jetson Nano board. Thus, we needed a camera that could take good quality pictures and could be connected to the Jetson Nano. Initially, we had decided on using two Raspberry Pi cameras (configured via the camera multiplexer) for the needs of our project since the Jetson Nano actually has dedicated slots for these cameras and they are connected directly to the GPU. However, when we captured images using the Raspberry Pi camera modules, we noticed that a lot of the images were quite dark and did not have a picture quality good enough for our purposes. Thus, we decided on using USB cameras since they offered a much better picture quality. They, too, could be connected to the Nano directly, and the process of setting them up wasn't complicated either, making them an optimal choice. Thus, although the USB cameras were much more expensive than the Raspberry Pi cameras, they were a better fit for us since our entire algorithm was centered around the images captured.

F. *Servo Controller*

Earlier, we were planning on using the Raspberry Pi to control the working of our servo. However, that would mean that the Jetson Nano would have to send a signal to the Raspberry Pi, which would then be processed and sent to the servo to make the servo rotate the appropriate angle in the appropriate direction in a given amount of time. This was quite complicated since it had several parts involved. Latency was key to our entire system since we had to ensure that the servo received the signal at the appropriate time to rotate the gate. However, we were unsure if this would be achievable given the communication involved. As a result, when we found a servo driver that could control the servo and was compatible with the Jetson Nano, that mitigated this concern and made our work much easier. It was much more convenient to use the servo driver and it was also relatively inexpensive. Moreover, since we were not dealing with multiple parts, this mitigated our concern of the gate not rotating in the given span of time. In addition to this, earlier, we were planning on resetting the gate back to its initial position so that it was always in the center once the fruit had been processed. However, we realized that a lot of power was wasted in moving the gate back to its original position, especially if the next fruit or vegetable coming along the belt had the same rottenness.

Software

A. *Live Stream Capturing*

Currently, both of our cameras are programmed to be capturing a live stream of the images. These images are then resized and analysed using a frame analyzer. The frame analyzer uses Gaussian blurring and Canny edge detection. Once the edges in the frame have been detected, the frame analyzer checks if the fruit is in the frame, is entering, is leaving, or is not in the frame. We have set a high enough number of pixels as a threshold to ignore cases where there is a crease in the conveyor belt cloth or some other minor edge present. The frame analyzer works very well in our stress test with 50 bananas-- it did not miss any bananas. This entire algorithm takes only around 0.01 seconds to run. Earlier, we planned on programming the cameras to capture an image every 4 seconds. Since typical conveyor belts run at 5 cm/s and fruits are usually spaced around 20 cm apart, this was a good time bound. However, we also wanted to account for some error in placing the fruits properly. If the fruit spacing would have even a small error, this system would have been susceptible to missing the fruit or detecting it incorrectly. Moreover, if no fruit was in the frame at a given point in time, the cameras would still pass the image along to the image segmentation algorithm. Using a live stream instead of capturing images every 4 seconds worked well for us. This is because our system provides an additional benefit that even if fruits are spaced as much as 0.2 cm apart, our algorithm will be sure to capture it. This ensures that no fruit will be missed, even if there's some human error in placing the fruits. Moreover, if a fruit is not in the frame, the frame analysing algorithm would recognise that and would not send the image along to the image segmentation algorithm, thus saving the program from performing unnecessary computation.

B. *Image Segmentation*

We are using OpenCV's HSV color thresholding for image segmentation. This works a lot better than RGB thresholding as it is difficult to capture the exact RGB colors a fruit has, especially since the exact RGB color value can vary slightly from fruit to fruit and also within the fruit. As a result, if we tuned our good versus rotten masks to a specific fruit, it is very likely that the algorithm would not work on another fruit of the same kind, or that it wouldn't work even on the same fruit even sometime later. On the other hand, HSV image segmentation separates the color information (hue) into its own channel which can help us capture a range of colors associated with the fruit independent of saturation or visibility values. Through HSV image segmentation, we can be sure to capture the masks accurately for a wide range of fruits since the HSV colour space would not vary from fruit to fruit as much as the RGB colour space since parameters like the hue and saturation are usually similar for a given class of fruits, and the visibility is something that would remain constant once the light source has been fixed and all fruits are captured in a similarly lit environment.

C. *Fruit Quality Evaluation*

The fruit quality evaluation system is the backbone of our project as this is the algorithm that will decide on the classification of fruit as fresh or rotten. After segmenting the fruit from the image, we perform a pixel by pixel analysis of the good and rotten masks to detect the percentage of rotten parts based on a fixed HSV color threshold for the same. We developed a neural network based algorithm (using AlexNet) for rotten fruit classification. We wanted to compare both methods and pick the best one rather than making a tradeoff decision without results for both methods. The AlexNet classifier took much longer to classify the fruit than our pixel classification algorithm. The AlexNet classifier took up to around 8 seconds to classify the image. Moreover, it did not produce an accurate output a large number of times. However, accuracy was crucial to us since the intention of our system is to provide an accurate fruit sorting product. Our pixel classification produced a much better accuracy and ran in just 0.03 seconds on average. We tested our algorithm on a large dataset [4] consisting of 1962 good bananas and 2754 rotten ones. Our algorithm classified the good bananas accurately 97.86% of the time and it classified the rotten bananas accurately 99.2% of the time. As a result, this algorithm performed much better not only in terms of speed, but also in terms of accuracy, so it was the obvious better choice for us.

V. System Description

In this section, we present the block diagram from Figure 5 in greater detail. We will go over all the subsystems individually. Our overall design is composed of 3 main subsystems: 1) software for classifying fruits 2) hardware to take pictures, and coordinate the control mechanism for sorting 3) mechanical conveyor belt system which integrates the software and hardware parts.

A. *Software subsystem*

This subsystem is the green box named classification algorithm and the gate controller module inside the block diagram. The purpose of this subsystem is to take an image as an input and produce a single output that classifies the fruit in the image as either good or rotten, and to communicate that decision to the servo driver.

1. *HSV conversion*
   The first step in processing the images is to convert the image from RGB space to HSV space. We tried to process the images using the RGB space, but found it difficult to accurately differentiate between pixels based on color alone. The exact color threshold was hard to capture even after trying to adjust the bounds for a while. After researching, we found HSV space to fit our needs much better since it separated the hue into a separate channel independent of saturation or lightness. Hence, we converted the image to HSV space. This resulted in a significant increase in accuracy during segmentation.

2. *Image segmentation*
   Once the image was converted to HSV space, we were able to accurately segment the image and isolate the fruits based on hue (e.g yellow for banana). To accomplish this, we kept adjusting the hue, saturation and brightness bounds. Then we realized a better method would be to graph the HSV colors prominent in a fruit. With that data, we were able to create a mask that segmented the fruit from the background. One challenge we anticipated was that lighting levels fluctuate depending on the time of day but we accounted for this by hanging a fixed and uniform light source above our fruit.

3. *Image classification*
   Finally, we classify the fruit in the image as being good or rotten. We decided to analyze discolored areas inside the fruit since those symbolize decay in the fruits and vegetables we planned to tackle, bananas and carrots. Depending on the size and frequency of these discolored areas, we categorize the fruit as good or rotten. We do this by manually creating HSV masks for the rotten parts of the fruit or vegetable by researching how rotenness presents itself in the produce in question. Then, with the help

of the mask we made for the whole produce in the image segmentation phase, we check what fraction the rotten parts are of the whole produce. We ran our algorithm on a training dataset [4] for rotten and fresh bananas and observed the percentages of rotten parts the rotten bananas had and decided on a threshold for rotenness. Then we used the testing dataset [4] which contained 1962 good banana images and 2754 rotten bananas to test our algorithm and rotenness threshold. Accuracy was key to our product, so we wanted to ensure we had a contingency plan in order to pick the best possible classifier for our algorithm. Our alternative solution was to use a neural network based approach. Our second classifier used AlexNet owing to its success at classifying images. This approach had several challenges since it was not easy to code accurately, and it took a very long time to train the model. The images we had from the dataset [4] were not all very similar to our real life scenarios since they had varying backgrounds, so it was hard to train the classifier well. Even after training it, it took quite a long time to run on each image, and would not meet our speed requirements. Moreover, its accuracy wasn't great either.

4. *Gate controller*
   Once the classification algorithm makes a decision about the quality of the fruit, a signal is sent to the gate controller system. The purpose of this system is to communicate the classification decision with the Adafruit servo driver, so that the driver can move the servo motor and thus the diverter gate appropriately.

B. *Hardware subsystem*

This subsystem is responsible for coordinating the different moving parts of the design, and controlling the flow of information. It consists of the NVIDIA Jetson Nano 2 GB, camera subsystem, the Adafruit servo controller, the 5 V DC servo motor, and the 25000 mAh Krisdonia power bank.

1. *NVIDIA Jetson Nano 2 GB*
   We used the NVIDIA Jetson Nano 2 GB as the central hub for all hardware related tasks. It was used to coordinate all the other pieces. For a discussion of why we picked this particular model, refer to the earlier tradeoffs section.

2. *Camera subsystem*
   Initially, we wanted the camera subsystem to consist of 2 Raspberry Pi camera modules V2, and the arducam multi camera adapter module V2.2. The multi camera module had support for upto 4 cameras, which is why we went with this method since we wanted to mitigate the risk of not having enough angles and pictures of the fruit. That is, we wanted our design to be scalable to 4 cameras in case we needed more angles of the fruit to meet the accuracy metrics. However, we found that two cameras were more than sufficient, and we were able to meet the accuracy requirements comfortably. Thus, we abandoned the initial design in favor of two tripod mounted USB cameras, which can be seen in the final setup picture. We chose USB cameras over the CSI cameras because the former had better resolution; since we were replacing 4 cameras with 2, we needed the two cameras to be of good quality.

3. *Adafruit 16-channel 12-bit PWM/Servo Driver*
   The main purpose of this system is to facilitate communication between the Jetson Nano and the servo motor that will control the gate sorting mechanism once the classification algorithm section produces an output. We need this module since it's not possible to control the servo motor directly from the Nano. A schematic for installation is shown below. This is what ended up building.

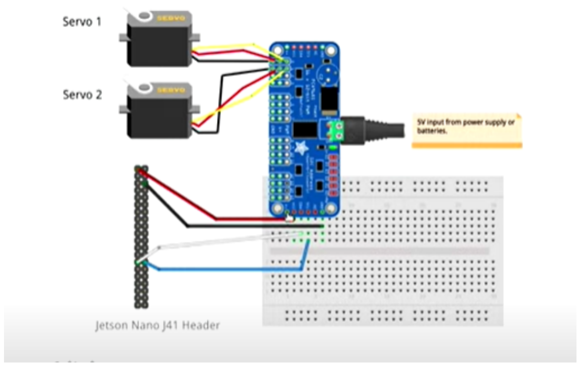


Fig 2. Adafruit Servo Controller & Jetson Nano Interface [5]

4. *Sorting mechanism*
   The sorting mechanism consists of the servo motor and the 3D printed gate that is attached directly to the shaft of the motor. The idea is that as the fruits come down the conveyor belt, the gate will block their movement and direct them to the appropriate basket. More details can be found in Figure 1.

5. *Power*
   We used the 25000 mAh krisdonia power bank to power the entire system. We chose this so that we can power the Jetson Nano for upwards of 8 hours, which is one of our requirements. We also used this power bank to power the Adafruit servo motor shield.

C. *Mechanical subsystem*

The mechanical subsystem consists of the conveyor belt. The system is divided into 2 parts, the mechanical side, which is the actual belt, and the electronics side which is the system that drives the belt. The 12 V DC gear motor is the point of contact between the 2 subsystems.

1. *Mechanical*
   The conveyor belt system consists of 2 pieces of wood that act as side panels, and a laminated piece of wood that goes inbetween. The side panels are exactly one inch longer than the laminated wood, which lets us place the rollers (PVC pipes of 1 inch diameter) in between the side panels of wood, tangent to the central piece of laminated wood. An image of this partial setup is presented below.

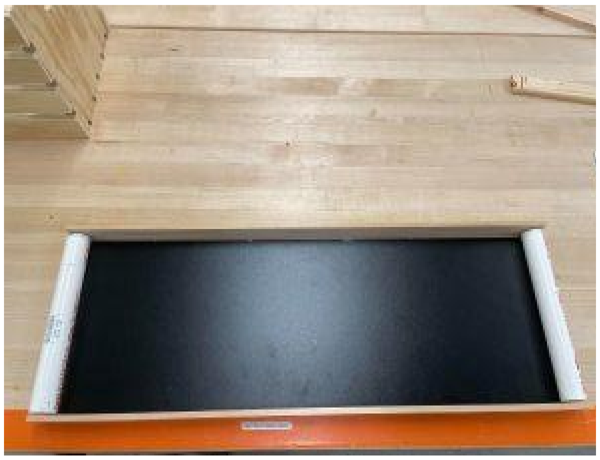


Fig 3. Partially Completed Conveyor Belt System

The motor is connected to the 3D printed shaft coupler and attached to one of the rollers. The belt goes on top of the rollers. This completes the mechanical portion of the conveyor belt. A finished image is presented in Figure 4.

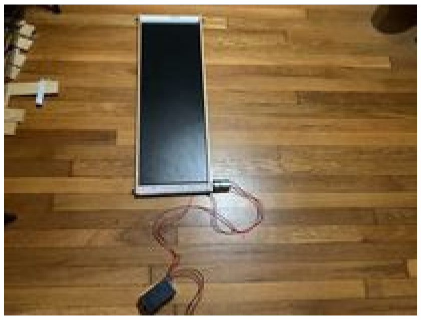


Fig 4. Top View of Completed Conveyor Belt System

Note that the above image includes the motor controller. A more detailed version of the setup can be found in Figure 1. A discussion of the dimensions of the conveyer belt and why we chose them can be found in Section 2. Other necessary miscellaneous parts (and their dimensions) required to complete the belt can be found in the budget and parts list later in the report.

2. *Electronic*
   The electronics subsection consists of just the RC motor speed controller board (not depicted in the images above, but available in the budget section). This board lets us control the speed of the motor and achieve the required 5 cm/s. Note that this board is powered separately (not using the power bank), since we want to maintain independence between the mechanical system and the hardware/software systems. Since conveyor belts aren't connected to batteries in farms, any appropriate power source is sufficient. We wanted to try this method before our plan B method, which was to assemble the PCB seen in the image above, since this method is easier and cheaper. If the RC motor controller board had not worked, we would have made the PCB using the 555 speed controller chip. We had the appropriate schematic of the PCB (and the appropriate gerber files), and the components that needed to be soldered on.

VI. Testing and Validation

In this section, we describe the different tests we performed in order to verify that we were meeting the requirements we had set out to meet in our specification. The requirements here are listed in order of importance.

A. *Accuracy*

For our purposes, we define false negatives as rotten bananas that were classified as good, and false positives as good bananas that were classified as rotte. We had set out to achieve an accuracy such that the false negatives were less than 5% and the false positives were less than 15%. We prioritised achieving a lower percentage of false negatives than false positives because it's not so harmful if some part of the good produce gets classified as rotten. This is because good produce classified as rotten results in a loss in some of the profits, so while it should be mitigated, it is not as much of an issue as rotten produce being classified as fresh. This is because if rotten produce is misclassified as good, it could cause all of the good produce to rot as well, resulting in a much greater loss. We tested our algorithm on a large dataset [4] that consisted of 1962 good banana images and 2754 rotten banana ones. Our algorithm produced only 0.8% false negatives and only 2.14% false positives, which was much better than the accuracy targets we had set out to meet. When we tested our algorithm classification on fruits moving on the conveyor belt whose images were captured by the cameras, our algorithm had a classification accuracy of 100%.

B. *Speed*

Typical conveyor belts run at 5 cm/s, so our goal was to ensure that our algorithm runs at a speed that can match these conveyor belt speeds. Since fruits are typically placed approximately 20 cm apart on a typical conveyor belt, we wanted our algorithm to take at most 4 seconds to classify the produce appropriately. Our algorithm took an average of merely 0.04 seconds to classify each fruit. It took 0.01 seconds to send the signal to the servo driver, and an additional 0.2 seconds for the gate to rotate to the appropriate angle. As a result, our entire algorithm completed processing for a given fruit in just around 0.25 seconds, which was much better than the goal we had set out to accomplish. Moreover, our algorithm can perform well even with varying conveyor belt speeds. We tested our algorithm by varying the speed of the motor controller and noticed that even with varying conveyor belt speeds, our algorithm processed the fruit in time for the gate to divert it in the appropriate direction.

C. *Fruit Spacing*

We wanted our product to be able to handle fruits spaced at a gap of 20 cm, with an error of 5 cm, to simulate the spacing of fruits on a typical conveyor belt system. In order to test this, we mimicked the spacing requirements by placing bananas on the conveyor belt every 2 seconds. We placed the fruit every 2 seconds instead of placing it every 4 seconds like a typical conveyor belt system because we want to ensure that our system could perform well even at goals that were much harder to achieve than real life settings. We stress tested this with 50 fruits, placing a fruit every 2 seconds, and the algorithm performed accurately on all of them.

D. *Produce Detection*

It was important for us to ensure that our algorithm accurately detected all of the fruits so that no fruit was missed. This is crucial since if the algorithm accidentally missed even a single fruit, it could result in a rotten fruit being misclassified. As a result, all of the good produce could get ruined owing to the misclassification of a single fruit. In order to ensure we met this requirement, we stress tested our product with 50 fruits. The algorithm accurately detected all of the fruits and the gate deflected each one of them in the appropriate direction based on the output that the classification algorithm produced.

E. *Ease of Installation*

Our goal was for our system to be easily integrable with existing conveyor belt systems. We wanted our fruit sorting solution to be inexpensive in order to meet the use case requirements for small scale farmers. As a result, we did not want farmers to need to purchase conveyor belt systems specifically for using our system since they're quite expensive. As a result, we designed a system that is independent of the conveyor belt. In order to ensure that this was the case, we tested our system with a conveyor belt we built. The algorithm was on the Jetson Nano and the diverter received a signal independent of the conveyor belt system being used. Moreover, since our algorithm completed processing a given fruit in just 0.25 seconds, we know that our algorithm would be quick enough to be integrable with any existing conveyor belt system, irrespective of its speed since typical conveyor belts don't run at speeds high enough to render our algorithm slow.

F. *Battery Life*

We wanted to ensure that our product can last a typical working day for the farmer. For a typical working day, which is usually from 9 am to 5 pm, we would require that the entire product be powered for approximately 8 hours. We tested this by using our

25000 mAh Krisdonia power bank to power the system for 8 hours. The power bank managed to successfully keep the system running for 8 hours, thus ensuring we met our use case requirements well.

VII. Project Management

A. Schedule

The first five weeks of the semester were spent in finalising the project idea, developing a well thought out design for it, and finalising our parts list. After that, Ishita Kumar started looking into image segmentation methods. In the meantime, Kushagra worked on understanding how the NVIDIA Jetson Nano works and Ishita Sinha worked on developing an edge detector to detect fruits in frame and developed the rottenness classifier. After the image segmentation had been completed, Ishita Sinha integrated the image segmentation with the pixel classification algorithms and tested the pixel classification algorithm, tuning the parameters as necessary. Post that, the team worked on setting up the conveyor belt and testing how well the algorithm performed on real fruits for the interim demo.

After that, Ishita Kumar worked on developing a frame analyser while Ishita Sinha worked on the algorithm for fruit detection. After this, the team met to set up the product. Next, Ishita Sinha integrated the software onto the Jetson Nano, and then, the team tested the product. Once this was done, we conceptualised the idea for our diverter and went through several prototypes before finalising one that worked. Then, Kushagra automated the servo rotation using the servo driver, followed by which Ishita Sinha performed final code integration. After this, we performed several tests to ensure we met our requirements. Then, we worked on our stretch goals which included adding carrots to the model and checking whether the AlexNet classifier worked better or the custom pixel classifier did. The schedule is included in Figure 6.

B. Team Member Responsibilities

While each team member had a role in every task to ensure that everyone had a holistic understanding of the working of the entire product, we split responsibilities on the basis of who'll be leading the task. The primary and secondary responsibilities of each team member are shown in the chart below:

Team Member	**Primary Responsibility**	**Secondary Responsibility**
Ishita Kumar	Image segmentation	Frame analyser
Ishita Sinha	Rottenness classifiers	Code integration
Kushagra Sharma	Conveyor belt setup	Servo driver setup

C. Budget

Our budget and parts list has been included in Figure 7. The main hardware tools we used include an NVIDIA Jetson Nano, 2 USB web cameras, a servo motor and the servo driver, and the parts needed to construct the gate. The mechanical component we built is the conveyor belt, so we have also included costs corresponding to parts needed to build it. We had purchased a Raspberry Pi camera module and the Arducam

D. Risk Management

As part of the project, one of our biggest risks with respect to meeting the use case requirements was ensuring we met the accuracy targets we had set out to accomplish. Our accuracy targets were to achieve a false negative rate less than 5% and a false positive rate less than 15%. We needed to be more contrite about the false negatives versus the false positives. In order to meet our accuracy targets, we designed multiple classification algorithms so that we weren't dependent on a single software approach, and kept the option open for installing more cameras if we needed more angles of the fruit.

Next, another major risk was if we could not meet the conveyor belt speed or if we missed fruits. Our product is designed to be integrated into existing conveyor belt systems, assuming they work at a speed of 5 cm/s. We were assuming 4 seconds to process a given fruit, so the distance between 2 fruits would be around 20 cm. However, we wanted to ensure we didn't miss a fruit if the speed is too high, or classify the same fruit twice if the speed is too low, or if there are spacing issues. Initially, our algorithm was taking around 5 seconds to run from start to finish, so we were quite concerned about it meeting speed requirements. However, we made several code optimisations and were finally able to meet our requirements.

Another great risk we were facing was with respect to building a conveyor belt. None of us have ever built anything mechanical, so it was possible we wouldn't be able to build it, or it would not meet our system requirements. Since the actual product needs to be integrated into existing conveyor belt systems, a conveyor belt isn't really part of the product itself, so even if we couldn't build a conveyor belt well, it shouldn't affect the performance of our product. To mitigate this risk, we planned on using a treadmill as a backup conveyor belt, and also included building the conveyor belt in an earlier stage in our schedule, so we had time to build it and know if we needed to use a treadmill or not. Thankfully, it worked out well.

Lastly, we were running quite on the clock with having our gate still pending in the last week since we did not have a good prototype earlier. As a result, we were concerned about completing the project in time. However, I'm happy we were able to. This was due to the slack time we had accounted for in our plan. Since we had several moving parts, we ran the risk of not being able to integrate all of the different parts of the project in a smooth fashion, but it all worked out in the end.

VIII. Ethical Issues

FarmFresh aims to serve as an affordable and value-adding tool for farmers. The product is especially targeted towards small-scale farmers who may not have been able to afford the expensive alternative fruit sorting machines such as by TOMRA. Although we have tested on bananas and carrots, the algorithm can also be adapted to other fruits and vegetables. However, there are some ethical points to think about. Since our product looks at only discolored rotten areas on produce right now, it may miss other types of deformation on fruits and vegetables that make produce unsuitable for markets. It would be a waste of money to transport such unsuitable produce to markets to be rejected later on, so it is important we clearly specify the use case for FarmFresh to farmers which is detecting if produce is rotten primarily through discoloration. In other instances, certain produce rots without showing much discoloration and rather shows more deformation. However, FarmFresh is biased towards the former type of rotting and so farmers who deal with the latter type of produce will not be able to avail the benefits of autonomous produce quality sorting. While FarmFresh makes this clear in its design specification, it is important to think about who is being left out of advantages through new technology. This gap can be filled by others in the food produce sector with new and cheaper innovation or even our group in the future by extending our algorithm to work on deformation using edge detection and other computer vision techniques.

It is also important for FarmFresh to not only be accurate-- for which we have tested-- but also robust in real world conditions, such as the conveyor belt systems in farms. Its breakdown would be very disadvantageous and distressing to farmers, especially our target customer base of small-scale farmers who may not have the resources at hand to recover from such failure. So, it is important for us to test our product in real farms before we deploy our solution.

IX. Related Works

In this section, we detail some other projects and products that are similar to FarmFresh. One of the biggest companies currently operating in the market for sensor based sorting solutions is the Japanese company TOMRA. TOMRA has several high end products that automatically sort fruits and vegetables. One such product is Blizzard, which is a multi-purpose optical food sorting machine.

The importance of automated food sorting technology is increasing as an increasing number of farms adopt automated picking and harvesting mechanisms, as these picking machines cannot distinguish good fruits from bad fruits. Recently, Abundant Robotics created an automated apple picker system [6], and one way to increase efficiency of the system might be to install an automated sorting mechanism within the picking stage. This would save time, energy and money, and is an area of further research.

There has also been research done in algorithms to detect rotten parts in a wide variety of different fruits and vegetables. One particularly interesting paper by Nossier et. al utilises KNN and SVM models [7]. The paper was able to distinguish between 4 different fruits, detect rotten fruits consistently, while maintaining high accuracy rates. The paper achieved this by applying numerous K-Nearest Neighbors algorithms (e.g fine, medium, coarse, cosine).

Finally, there has been extensive research done in the field of CNNs and their use in image classification. In “Pure-CNN: A framework for fruit images classification”, Kausar et. al did a particularly thorough treatment of the subject [8]. The CNN model they developed could be an area of further research.

X. Summary

Our system was able to meet the requirements comfortably; we had a higher than expected accuracy rate in classifying both good and rotten fruits. We were also able to meet the speed of conveyor belt requirements since we used the motor speed controller. This gave us flexibility to finetune the speed and overcome any limitations due to friction between belt and wood or belt and rollers. We had carefully picked the power source, so that metric was satisfied. Finally, we tested our algorithms and hardware on an actual conveyor belt setup to demonstrate ease of installation. Hence, we were able to meet all our goals for the MVP. With the added time, we were able to incorporate additional fruits as well.

One of the major limitations of the system was the jitter in conveyor belt speed as a result of uneven friction across surfaces. Given enough time, we would definitely address this, and make sure the speed of the belt doesn't fluctuate too much. The reason for this potential jitter was the stitched point in the fabric, which often got stuck momentarily. Given more time, we would make the joint seamless. Another very simple improvement we could make to mitigate this would be to buy a more powerful motor (one with more torque).

Apart from starting early, the biggest advice to future teams is to plan extensively with your teammates. It really helps when all team members are on the same page, and there is fluid communication. Planning early also allows prototyping, and enables finding areas to improve the design, and come up with easier designs.

References

1. https://www.nass.usda.gov/Publications/AgCensus/2007/Online_Highlights/Fact_Sheets/Farm_Numbers/small_farm.pdf
2. https://www.researchgate.net/publication/312494475_A_Survey_on_Computer_Vision_Technology_for_Food_Quality_Evaluation
3. https://www-sciencedirect-com.proxy.library.cmu.edu/science/article/pii/S0924224403002711
4. https://www.kaggle.com/sriramr/fruits-fresh-and-rotten-for-classification
5. https://www.arducam.com/docs/camera-for-Jetson-Nano/multiple-cameras-on-the-Jetson-Nano/arducam-multi-camera-adapter-on-the-Nano/
6. Simonite, Tom. “Apple-Picking Robot Prepares to Compete for Farm Jobs.” *MIT Technology Review*, MIT Technology Review, 3 May 2017, www.technologyreview.com/2017/05/03/152012/apple-picking-robot-prepares-to-compete-for-farm-jobs/.
7. no, Ann & Ahmed, Seif. (2019). Automatic Classification for Fruits' Types and Identification of Rotten Ones Using k-NN and SVM. International Journal of Online and Biomedical Engineering (iJOE). 15. 47. 10.3991/ijoe.v15i03.9832.
8. A. Kausar, M. Sharif, J. Park and D. R. Shin, "Pure-CNN: A Framework for Fruit Images Classification," *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2018, pp. 404-408, doi: 10.1109/CSCI46756.2018.00082.

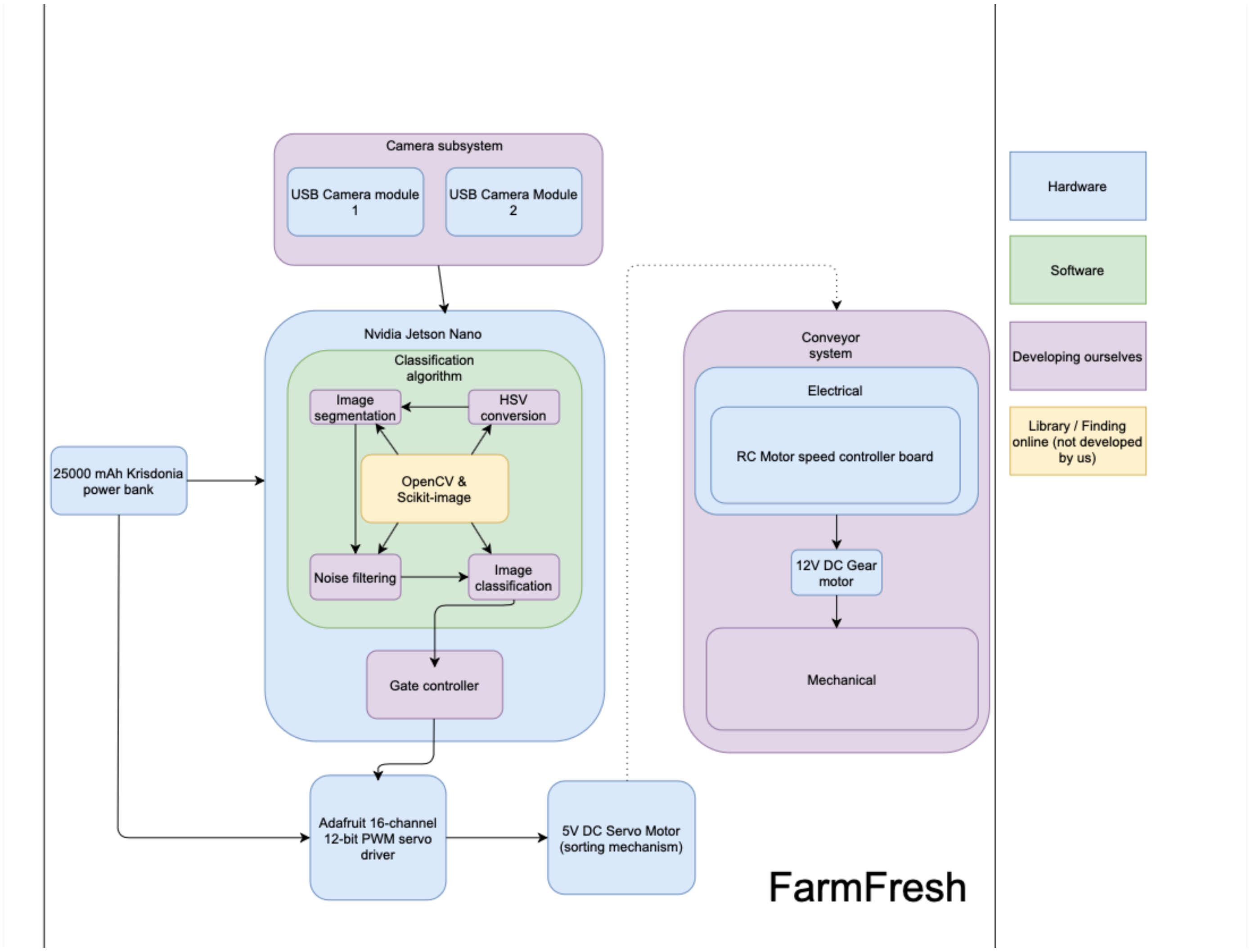
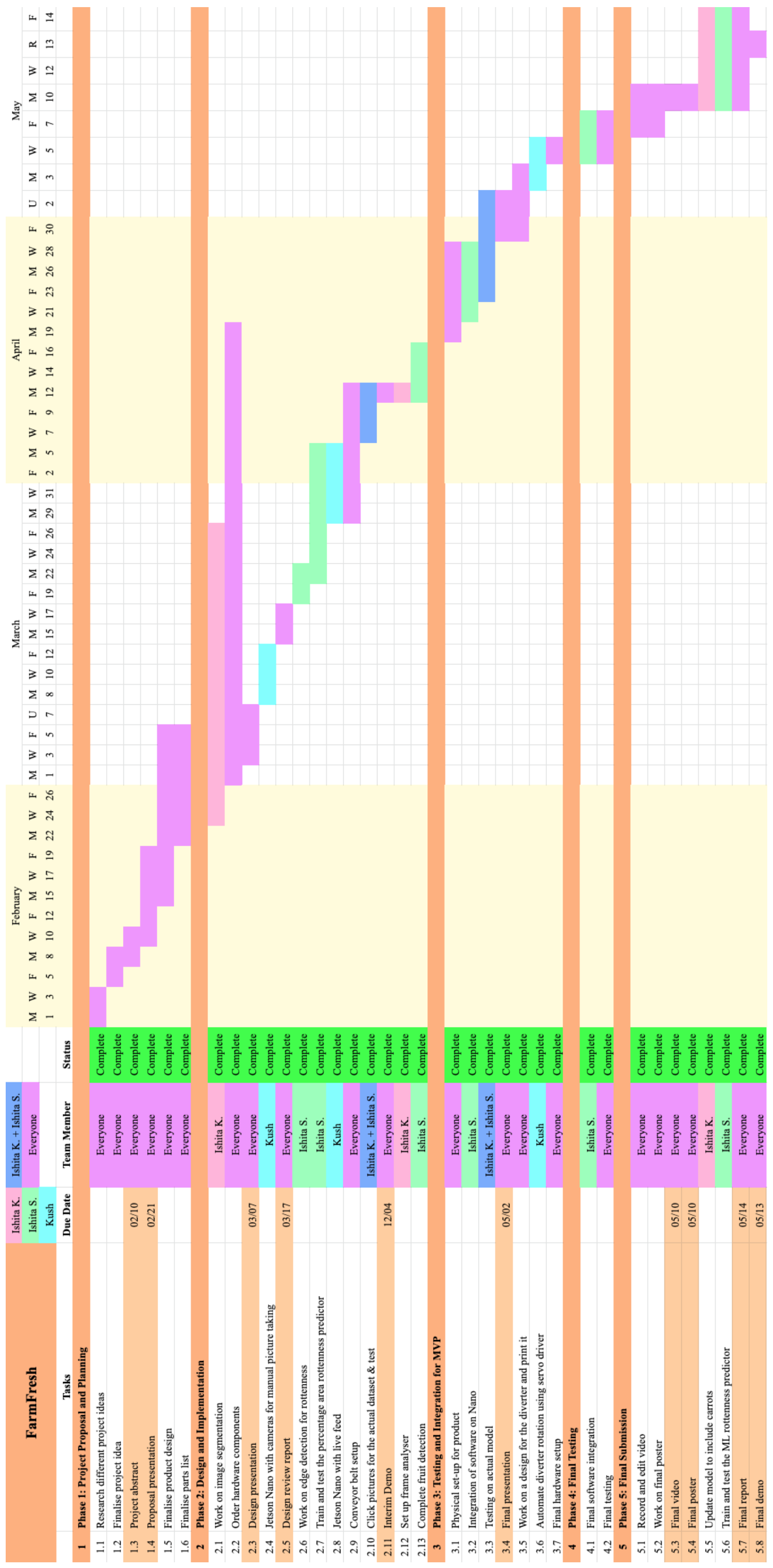


Fig 5. Block Diagram

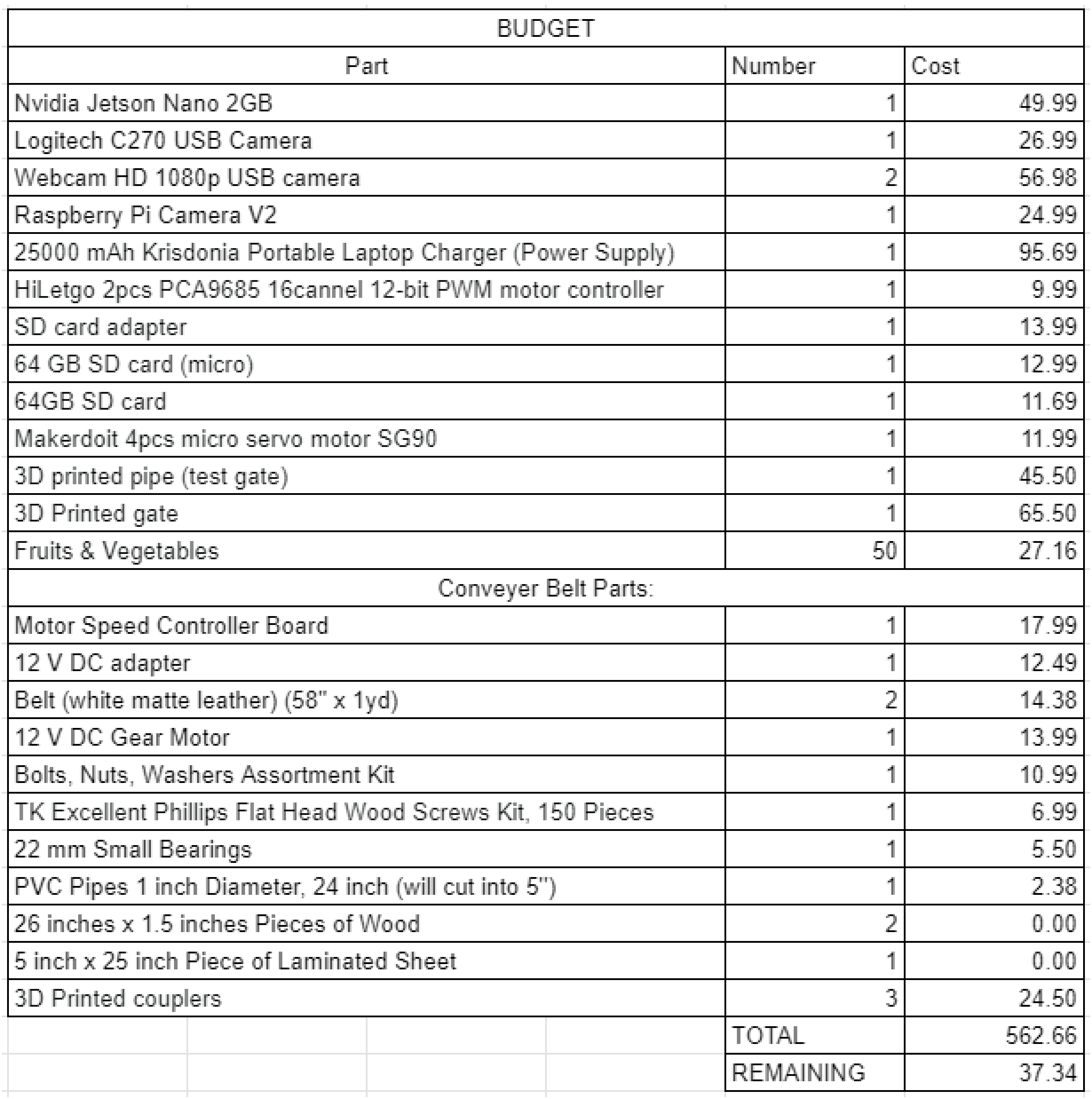


FarmFresh

Ishita K.	Ishita S.	Kush	Ishita K. + Ishita S.	Everyone

	Tasks	Due Date	Team Member	Status
1	**Phase 1: Project Proposal and Planning**			
1.1	Research different project ideas		Everyone	Complete
1.2	Finalise project idea		Everyone	Complete
1.3	Project abstract	02/10	Everyone	Complete
1.4	Proposal presentation	02/21	Everyone	Complete
1.5	Finalise product design		Everyone	Complete
1.6	Finalise parts list		Everyone	Complete
2	**Phase 2: Design and Implementation**			
2.1	Work on image segmentation		Ishita K.	Complete
2.2	Order hardware components		Everyone	Complete
2.3	Design presentation	03/07	Everyone	Complete
2.4	Jetson Nano with cameras for manual picture taking		Kush	Complete
2.5	Design review report	03/17	Everyone	Complete
2.6	Work on edge detection for rottenness		Ishita S.	Complete
2.7	Train and test the percentage area rottenness predictor		Ishita S.	Complete
2.8	Jetson Nano with live feed		Kush	Complete
2.9	Conveyor belt setup		Everyone	Complete
2.10	Click pictures for the actual dataset & test		Ishita K. + Ishita S.	Complete
2.11	Interim Demo	12/04	Everyone	Complete
2.12	Set up frame analyser		Ishita K.	Complete
2.13	Complete fruit detection		Ishita S.	Complete
3	**Phase 3: Testing and Integration for MVP**			
3.1	Physical set-up for product		Everyone	Complete
3.2	Integration of software on Nano		Ishita S.	Complete
3.3	Testing on actual model		Ishita K. + Ishita S.	Complete
3.4	Final presentation	05/02	Everyone	Complete
3.5	Work on a design for the diverter and print it		Everyone	Complete
3.6	Automate diverter rotation using servo driver		Kush	Complete
3.7	Final hardware setup		Everyone	Complete
4	**Phase 4: Final Testing**			
4.1	Final software integration		Ishita S.	Complete
4.2	Final testing		Everyone	Complete
5	**Phase 5: Final Submission**			
5.1	Record and edit video		Everyone	Complete
5.2	Work on final poster		Everyone	Complete
5.3	Final video	05/10	Everyone	Complete
5.4	Final poster	05/10	Everyone	Complete
5.5	Update model to include carrots		Ishita K.	Complete
5.6	Train and test the ML rottenness predictor		Ishita S.	Complete
5.7	Final report	05/14	Everyone	Complete
5.8	Final demo	05/13	Everyone	Complete

Fig 6. Schedule Gantt Chart



BUDGET		
Part	Number	Cost
Nvidia Jetson Nano 2GB	1	49.99
Logitech C270 USB Camera	1	26.99
Webcam HD 1080p USB camera	2	56.98
Raspberry Pi Camera V2	1	24.99
25000 mAh Krisdonia Portable Laptop Charger (Power Supply)	1	95.69
HiLetgo 2pcs PCA9685 16cannel 12-bit PWM motor controller	1	9.99
SD card adapter	1	13.99
64 GB SD card (micro)	1	12.99
64GB SD card	1	11.69
Makerdoit 4pcs micro servo motor SG90	1	11.99
3D printed pipe (test gate)	1	45.50
3D Printed gate	1	65.50
Fruits & Vegetables	50	27.16
Conveyer Belt Parts:		
Motor Speed Controller Board	1	17.99
12 V DC adapter	1	12.49
Belt (white matte leather) (58" x 1yd)	2	14.38
12 V DC Gear Motor	1	13.99
Bolts, Nuts, Washers Assortment Kit	1	10.99
TK Excellent Phillips Flat Head Wood Screws Kit, 150 Pieces	1	6.99
22 mm Small Bearings	1	5.50
PVC Pipes 1 inch Diameter, 24 inch (will cut into 5")	1	2.38
26 inches x 1.5 inches Pieces of Wood	2	0.00
5 inch x 25 inch Piece of Laminated Sheet	1	0.00
3D Printed couplers	3	24.50
	TOTAL	562.66
	REMAINING	37.34

Fig 7. Budget and Parts List