Fruit Finder

Liam Waterbury, Ethan Brown

CSSE 463 Image Recognition

8 December 2023

ABSTRACT

We seeked to find a solution to automating the process of detecting and classifying fruit in an image. This is a step to further explore the influential domain of image detection. Through conversion of the images to the HSV color space, color value ranges were extracted for each fruit allowing for the satisfying pixels to be identified. This resulted in noisy results that were cleaned with morphological operations and adaptive region size filtering. After the cleaning, our system boasted a 100% accuracy with the training set. The system did not miss any fruit and did not identify any extra fruit. However, when evaluated on a test image with different conditions, assumptions during the creation of the system were exposed.

1. INTRODUCTION

The integration of computers into automation has resulted in incredible technological leaps. However, how can the innately human experience of seeing be automated with a machine? In this paper, we seek to teach a machine to be able to identify and classify different fruits in a photo.

The automation of such a detection task is incredibly important and influential. Currently, a human would need to be paid to perform the classification, resulting in an inevitable cost. If a machine could be taught to replace this human's responsibility, the cost would only consist of the amortized development expenses and power requirements of the machine. In addition, the use of a computer would speed up the classification itself. While the task of identifying fruit may not have the largest application, automating human vision has endless possibilities. Security guards can be replaced with cameras, postal services can automatically sort mail, and vehicles can make determinations on their surroundings. The problem of fruit finding, in particular, is useful because it can be approached in a few unique ways and can be performed to a relatively high success rate with simplistic techniques.

The process of teaching a computer to 'see' is very challenging. We must first understand and break down what humans do when seeing. We know what an apple looks like versus a banana; however, how do we know this? Is it the shape, the size, or the color? Even within color, we know bananas are yellow but we can still identify a green unripe banana.

Our approach primarily consisted of isolating color features from the images. We took the converted HSV images as inputs to the system and outputted a mask of predicted fruit pixels and an overall prediction of the number of fruit in the image. Each unique fruit operates in a certain range within the color space; therefore, we used these ranges to identify each type of fruit independently. However, there are overlaps in these ranges, major outliers (banana bruising or highlights on an apple), and plenty of background noise. Therefore, in addition to color ranges, our approach utilized unique structure elements for the morphological operations and dynamic region measuring to remove smaller noises.

The system was trained on three primary photos (Figure 1) In addition, the system was construction on the assumptions that only oranges, apples, and bananas would be present, a particular type of dark apple would be used, none of the same fruit would be overlapped, and that the background would not fall in the color ranges of the fruits. These assumptions can be removed and addressed with future work.

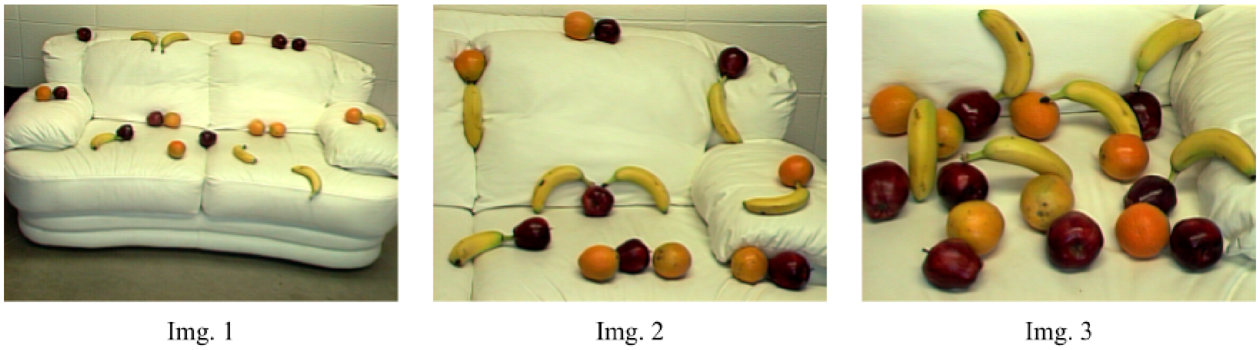


Figure 1: Primary photos used for the development of the Fruit Detection model.

2. PROCESS

This project was very exploratory in nature and several of our ideas did not result in successful results. The following subsections will outline the successful methodologies and techniques employed in our final system.

2.1 Data Source

Our data source consisted of 3 core photos of bananas, oranges, and apples arranged on a white piece of furniture. The images were taken from varying distances and with varying closeness between fruit. Finally, a fourth photo of the fruit packed into a bowl is used to stress the performance of our system on a more complex image. The fruit photos are from the University of Rochester [1].

2.2 Preprocessing

Before operating on any of the images, we first converted them from RGB color space to HSV color space using the built-in MATLAB functions. We opted to work with HSV because it more closely aligns with human color perception and is less sensitive to lighting changes.

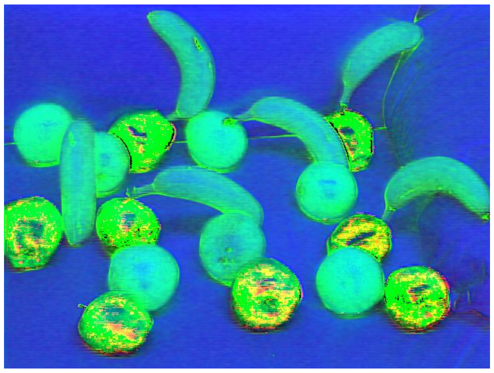


Figure 2: Converted HSV image represented in RGB space.

2.3 Fruit pixel models

To classify regions of the image as particular fruit, we used HSV value thresholds. We used MATLAB's imtool to view the pixel values for each fruit on the HSV image. We used trial and error, visually confirming the results until we found HSV ranges that created accurate masks.

Table 1: HSV Thresholds for each fruit.

Fruit	Hue	Saturation	Value
apple	[0.00, 0.06] [0.96, 1.00]	[0.58, 1.00] [0.00, 1.00]	[0.01, 0.58] [0.00, 1.00]
orange	[0.04, 0.10] [0.08, 0.12]	[0.90, 1.00] [0.65, 1.00]	[0.55, 1.00] [0.55, 1.00]
banana	[0.12, 0.19]	[0.62, 0.92]	[0.47, 1.00]

In addition to the direct thresholds, we utilized conditional thresholds for orange and apple identification. Table 1 shows the HSV thresholds used for each of the fruits. There are two sets of thresholds for the apples and oranges, which captures pixels that fall into *either* of the two thresholds.

The two thresholds are needed for the apples since red on the hue scale is represented by values close to both 0 and 1. This is because the hue values are represented on a circle, with red values close to 0 radians.

Similarly, the oranges need two thresholds to account for some odd coloring on some of the oranges.

As seen in Figure 3, there are many additional pixel clusters that will be cleaned up further on in the process.

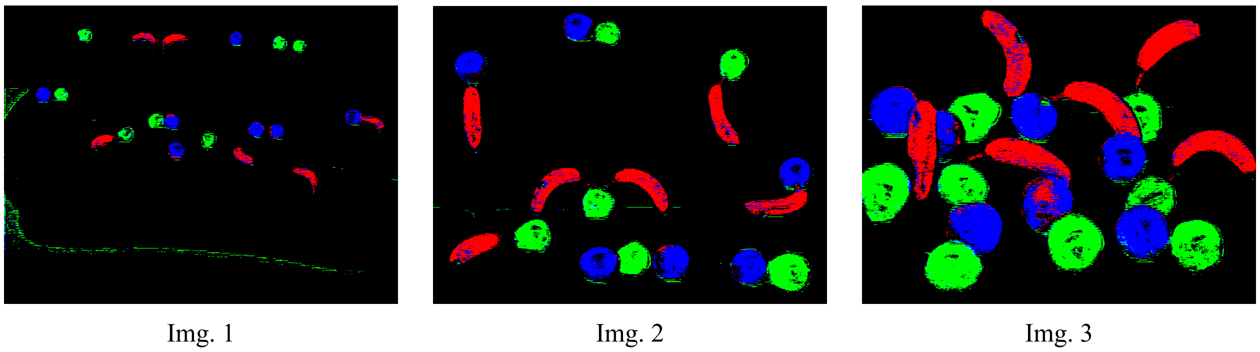


Figure 3: Results from the color thresholds. The bananas, apples and oranges are represented by the red, green, and blue pixels, respectively.

2.4 Morphology

In order to increase the accuracy of our fruit finder, we decided to use morphological operations on each of the fruit masks. For each fruit, we used a different number of morphological operations, and varying structure elements, as each of the fruits have their own challenges.

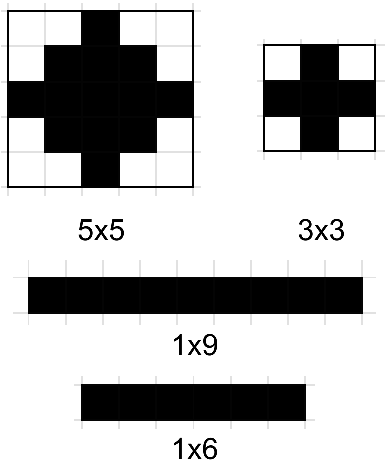


Figure 4: Labeled structure elements used for morphological operations. Black squares represent a 1 in the structure element, while white squares represent a 0. For brevity, each of these 4 structure elements will be referenced using the dimensions of the structure element in tables 2-4.

The banana masks had the simplest morphology out of the three different kinds of fruit. We began by using the 3x3 structure element to erode the small pixel clumps that were unintentionally caught by our HSV threshold. Then, two dilations are done on the banana mask in order to fill in holes.

Table 2: Banana mask morphological operations and structure elements.

	Op 1	**Op 2**	**Op 3**	**Op 4**
S.E.	3x3	5x5	5x5	N/A
Morphology	Erode	Dilate	Dilate	N/A

For the morphology on the orange, we began with the same approach as the bananas. We started by eroding, using a smaller structure element, then dilated with a larger element. Again, this is because there were a lot of small clusters of pixels that needed to be removed, but the oranges themselves frequently had a large hole in the middle, causing our finder to think it is two separate oranges. For the third operation, we used close to get rid of any leftover small pixel islands. Additionally, we continued to run into issues with the oranges being split in two during erosion operations. In order to fill the small gap, we opted to use a special structure element. This structure element is long and horizontal, such that it is able to recognize small divides in oranges and fill the gap.

Table 3: Orange mask morphological operations and structure elements.

	Op 1	**Op 2**	**Op 3**	**Op 4**
S.E.	3x3	5x5	3x3	1x9
Morphology	Erode	Dilate	Close	Dilate

Finally, we needed to perform morphological operations on the apple mask. Because of the glare and variation in pixel values, the initial apple mask left a lot of small holes, resulting in the fruit finder overcounting the apples. To fix this, we followed a similar procedure as the banana and apple, however we opted to use a smaller structure element for the first dilate operation. We did not need as large of a structure element for the dilate since the groupings of differently colored pixels was not very large. For the second dilate on the apple mask, we used the same technique as the orange mask, allowing us to fix large, vertical gaps in the mask.

Table 4: Apple mask morphological operations and structure elements.

	Op 1	**Op 2**	**Op 3**	**Op 4**
S.E.	3x3	3x3	1x6	N/A
Morphology	Erode	Dilate	Dilate	N/A

Figure 5 shows the results of these morphological operations on the masks. The morphology significantly improved the initial mask; however, there still looks to be some inaccuracies.

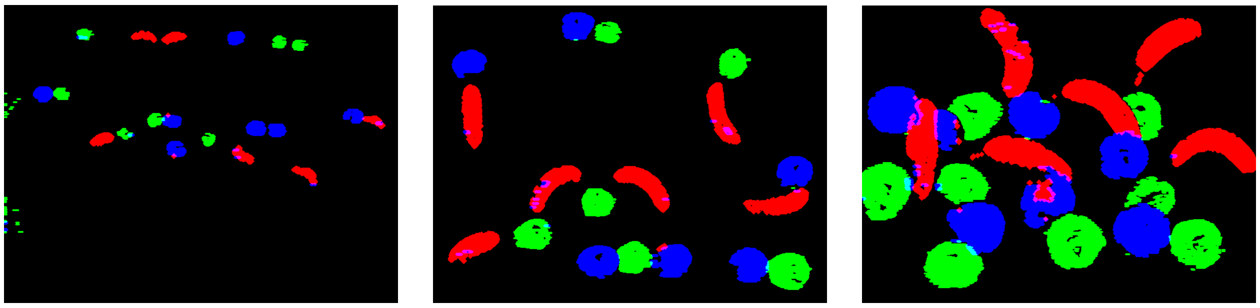

Img. 1 Img. 2 Img. 3

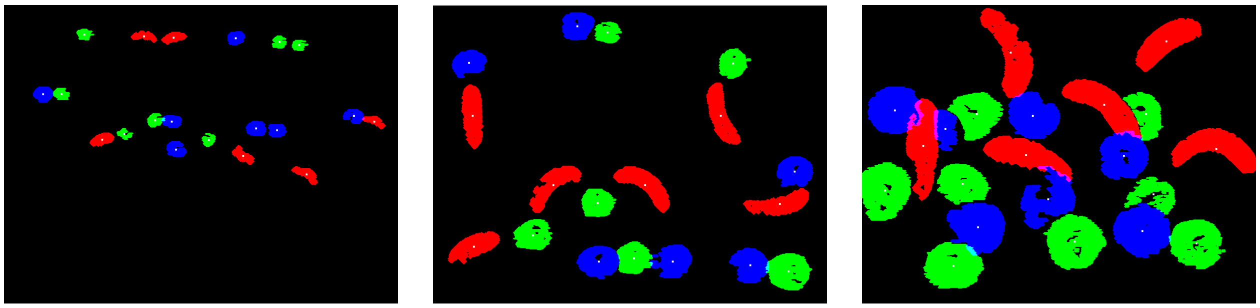
Figure 5: Results after the morphological operations. The bananas, apples and oranges are represented by the red, green, and blue pixels, respectively.

2.5 Post-processing

In addition to morphological operations, we performed post-processing to remove smaller, noisy clusters from the image. We utilized an adaptive approach to make the system more flexible across the varying distances within training images and any future test images.

We used the post-morphological regions to calculate an average region size. Next, we filtered out any regions that were less than a two thirds of the size of the average using the bwareaopen MATLAB function. This proved very effective in removing smaller regions and noise along the edge of the photos and resulted in a more accurate final result, as seen in Figure 6.

3. RESULTS



Img. 1 Img. 2 Img. 3

Figure 6: Final results from the complete fruit finder process. The bananas, apples and oranges are represented by the red, green, and blue pixels, respectively. Additionally, the white dot represents the centroid for a given piece of fruit.

At the beginning of the process the fruit finder grossly overcounted the number of each fruit in all of the images. However, when the final system was run against the training images, we found that the final mask was able to correctly predict the number of fruit for the original three images.

Table 5: Fruit count results, broken up by process stage, image and kind of fruit.

Image	Apple	Banana	Orange
Initial Mask (2.3)			
1	273	86	336
2	165	103	499
3	280	281	912
Tray	504	220	670
Morphology Mask (2.4)			
1	21	8	15
2	8	7	19
3	14	16	21
Tray	42	5	11
Final Mask (2.5)			
1	7	6	7
2	6	6	6
3	8	6	7
Tray	7	1	1
Actual Count			
1	7	6	7
2	6	6	6
3	8	6	7

Additionally, we included an extra image found in figure 7, which will be discussed further in this paper.

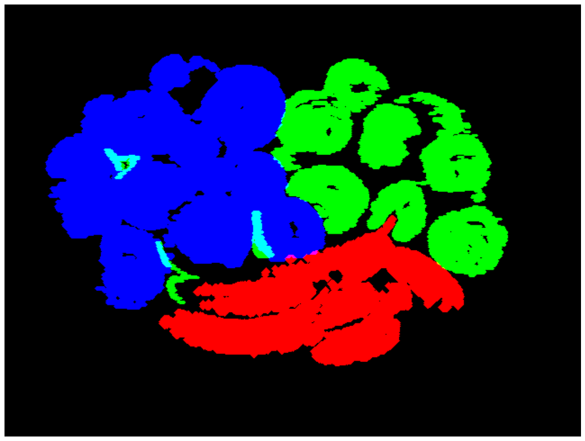


Figure 7: Additional image showing the mask results with fruit placed in a bowl, overlapping.

4. DISCUSSION

Overall, the fruit finder performed very well on the original images provided. After setting the HSV threshold, applying morphological operations, and removing smaller noise regions, we were able to accurately predict the number of each fruit in each of the original three images.

Looking at the resulting images, we can visually see that the banana has consistent and relatively clean shapes. This is likely due to the bananas having a large HSV threshold range, and not as many bright highlights or inconsistent coloring. However, the brown bruises on the bananas were not taken explicitly into account, so the fruit finder would currently have trouble with too ripe or browning bananas.

Even after the morphology and post processing, the resulting apple and orange masks show some holes and inconsistencies. When trying to fill these holes too much, we found that the smaller pixel islands grew too much to easily remove.

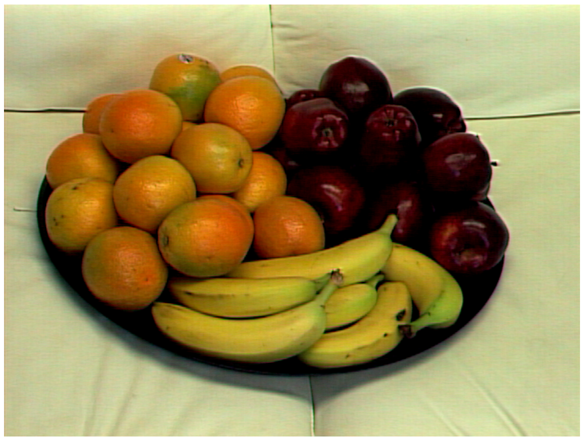


Figure 7: Fruit tray image used to test the system.

The system did not perform as accurately on the fruit tray image (Figure 7, 8) as on the 3 training images. This can be attributed to the different layout that the fruits are in, compared to the training images. The fruits are packed together and arranged by type. This exposed the assumption of our system that fruit would be separated from others of the same type. As such, the system counted fruit by summing up the unique regions. However, in the fruit tray, the bananas and oranges formed one large region. With this being said, the system impressed with identifying the fruit in a different background. The image was still taken on the white couch; however, it was also on a black tray. This did not exist in any of the training images, but the system still successfully identified the fruit pixels without errors derived from the different background.

As in any computer vision task, many variations in images should be expected. The first variation is the distance from fruit to camera. Our system handles this variation well because the distance does not affect the color ranges that each fruit exists in. In addition, the adaptive region filtering provides additional integrity to varying fruit sizes. However, if the fruit are too small it would create issues with keeping many false positives, since the average would be lower and closer to smaller clusters of noise.

Anothing variation is in the layout of the fruit. The system was trained on images of sparsely layed out fruit and runs into issues when the layout does not match this. In order to handle this variation, the model would need ways to be able to differentiate between the different individual fruits in a region. This could be done with edge detection or with shape analysis. This is discussed further in future work.

Finally, variations in lighting and setting are to be expected in use case scenarios such as grocery stores. The system does not completely handle this variation. It can handle some background changes as long as the background is not within the color range of the fruit. However, a bright yellow background would break our model because of the overlap with the banana color thresholds. This could be solved by attempting to detect background color and shape in order to intentionally exclude it from the mask. In addition, major lighting changes are not handled. This could be improved by adding more logic to the color ranges. Given the nature of HSV, with lighting changes, only one of the color channels would change drastically. If this could be isolated, the color thresholds could adaptively update based on the detected lighting level of an image.

5. CONCLUSIONS AND FUTURE WORK

Color proved to be a powerful tool for initially detecting and classifying potential fruit; however, extensive cleaning was necessary for accurate results.

There is still a lot of room to improve the morphology operations done on the masks. Because the initial masks were over-counting the fruits, it is clear that the variation in the pixel values is large enough to always require some form of morphology. However, due to the short timeframe on this project, we only optimized the morphology and structure elements in reference to the training images. Given more time, it could prove beneficial to do more research and perform more tests to allow for the fruit finder to function better on additional images. It would also be interesting to look into dynamically creating the structure elements so it is not the same for each image.

In addition, adaptive region trimming was a clean solution to small regions or noise on the edges of the photo or overlapped with other fruit regions.

The fruit tray image (Figure 7) stressed the model and exposed assumptions related to how individual fruits were being counted.

There are several avenues that could be taken to continue the project. Firstly, given another week or so, we would like to resolve the problem of undercounting fruit when the same type of fruit is overlapping. This problem was especially prevalent in the fruit tray image (Figure 7). We would like to utilize edge detection techniques in order to find borders within these large continuous regions of fruit. This would allow for these large regions to be broken up and result in a more accurate final count. While the edge detection aids in the count prediction, it does not impact the mask generated. With more time, we would like to implement an algorithm that fills holes in the masks. It was common that reflective spots or bruising on the fruit resulted in gaps in the mask, and an algorithm could go through and resolve these holes. Certain considerations and edge cases such as a ring of the same type of fruit would need to be considered. Finally, with a lot more time to put towards this project, we would like to find a way to add the shape of the fruit as a feature to the system. However, better yet, we would like to experiment with convolutional neural networks to allow for the network to determine what features are important for the detection and classification. This would require a much larger and more diverse data set and an immense amount of time labeling said data set. However, the resulting model would be much more accurate and prepared for complex and different photos.

REFERENCES

[1] Images courtesy of Randal Nelson, Department of Computer Science, University of Rochester, accessed March 2006.

APPENDIX

A. Code

```
% Read in file

mf1 = imread('mixed_fruit1.tiff');

mf2 = imread('mixed_fruit2.tiff');

mf3 = imread('mixed_fruit3.tiff');

tray = imread('fruit_tray.tiff');

imwrite(findFruit(mf1, 1), 'FruitFiner/Initial/fig1.png');

imwrite(findFruit(mf2, 1), 'FruitFiner/Initial/fig2.png');

imwrite(findFruit(mf3, 1), 'FruitFiner/Initial/fig3.png');

imwrite(findFruit(tray, 1), 'FruitFiner/Initial/tray.png');

imwrite(findFruit(mf1, 2), 'FruitFiner/Morphed/fig1.png');

imwrite(findFruit(mf2, 2), 'FruitFiner/Morphed/fig2.png');

imwrite(findFruit(mf3, 2), 'FruitFiner/Morphed/fig3.png');

imwrite(findFruit(tray, 2), 'FruitFiner/Morphed/tray.png');

imwrite(findFruit(mf1, 3), 'FruitFiner/Final/fig1.png');

imwrite(findFruit(mf2, 3), 'FruitFiner/Final/fig2.png');

imwrite(findFruit(mf3, 3), 'FruitFiner/Final/fig3.png');

imwrite(findFruit(tray, 3), 'FruitFiner/Final/tray.png');

function retValue = countFruit(banana_cc, apple_cc, orange_cc)

   [~, numBananaRegions] = bwlabel(banana_cc);

   [~, numOrangeRegions] = bwlabel(orange_cc);

   [~, numAppleRegions] = bwlabel(apple_cc);


   totalFruit = numBananaRegions + numOrangeRegions + numAppleRegions;

   fprintf('Num Banana Regions => %d\n', numBananaRegions);

   fprintf('Num Orange Regions => %d\n', numOrangeRegions);

   fprintf('Num Apple Regions  => %d\n', numAppleRegions);

   fprintf('Total Regions      => %d\n\n', totalFruit);

end
```

```
% if stage == 1 => return img before morphology
% if stage == 2 => return img after morphology, before area drop
% else          => return final image with centroids
function retValue = findFruit(img, stage)
   hsv_img = rgb2hsv(img);
   % Establish Fruit HSV Ranges
   banana_h = [.12, .19];
   banana_s = [.62, .92];
   banana_v = [.47, 1.00];

   orange_h = [.04, .1];
   orange_s = [.62, .92];
   orange_v = [.55,1.00];

   apple_h = [.00, .06];
   apple_s = [.58, 1.00];
   apple_v = [.01, .58];

   % Define HSV
   H = hsv_img(:,:,1);
   S = hsv_img(:,:,2);
   V = hsv_img(:,:,3);

   % Extract Masks
   banana_mask = zeros(size(hsv_img, 1), size(hsv_img, 2));
   banana_find = find(H >= banana_h(1) & H <= banana_h(2) & ...
                      S >= banana_s(1) & S <= banana_s(2) & ...
                      V >= banana_v(1) & V <= banana_v(2));
   banana_mask(banana_find) = 1;
```

```
orange_mask = zeros(size(hsv_img, 1), size(hsv_img, 2));
orange_find = find((H >= orange_h(1) & H <= orange_h(2) & ...
                     S >= 0.9) | ...
                    (H >= 0.08 & H <= 0.12 & ...
                     S >= 0.65) & ...
                    V >= 0.55);
orange_mask(orange_find) = 1;

apple_mask = zeros(size(hsv_img, 1), size(hsv_img, 2));
apple_find = find((H >= apple_h(1) & H <= apple_h(2) & ...
                   S >= apple_s(1) & S <= apple_s(2) & ...
                   V >= apple_v(1) & V <= apple_v(2)) | ...
                  (H >= 0.95));
apple_mask(apple_find) = 1;

% Save figure before morphological operations
if stage == 1
    countFruit(banana_mask, apple_mask, orange_mask);
    img(:, :, :) = 0;
    img(:, :, 1) = uint8(banana_mask) * 255;
    img(:, :, 2) = uint8(apple_mask) * 255;
    img(:, :, 3) = uint8(orange_mask) * 255;
    retValue = img;
    return
end

% Perform Morphological Operations
% Perform openings to remove smaller pixel areas
% Mess around a lil for report
SE = strel([0 1 0; 1 1 1; 0 1 0]);
```

```
SE_large = strel([0 0 1 0 0; ...
                  0 1 1 1 0; ...
                  1 1 1 1 1; ...
                  0 1 1 1 0; ...
                  0 0 1 0 0]);
orange_fill_SE = strel(ones(1, 9));
apple_fill_SE = strel(ones(1, 6));
% Banana
banana_mask = imerode(banana_mask,SE);
banana_mask = imdilate(banana_mask,SE_large);
banana_mask = imdilate(banana_mask,SE_large);

% Orange
orange_mask = imerode(orange_mask, SE);
orange_mask = imdilate(orange_mask, SE_large);
orange_mask = imclose(orange_mask, SE);
orange_mask = imdilate(orange_mask, orange_fill_SE); % Fill in weird green gaps

% Apple
apple_mask = imerode(apple_mask, SE);
apple_mask = imdilate(apple_mask, SE);
apple_mask = imdilate(apple_mask, apple_fill_SE); % Fill in weird gaps
% Return morphology mask before chunk drop
if stage == 2
    countFruit(banana_mask, apple_mask, orange_mask);
    img(:, :, :) = 0;
    img(:, :, 1) = uint8(banana_mask) * 255;
    img(:, :, 2) = uint8(apple_mask) * 255;
    img(:, :, 3) = uint8(orange_mask) * 255;
    retValue = img;
    return
end

% Grouping Operations and Connected Component Analysis
banana_cc = bwlabel(banana_mask, 8);
banana_regions = regionprops(banana_cc, 'Area');
bananaAverageArea = mean([banana_regions.Area]);

orange_cc = bwlabel(orange_mask, 8);
orange_regions = regionprops(orange_cc, 'Area');
orangeAverageArea = mean([orange_regions.Area]);

apple_cc = bwlabel(apple_mask, 8);
apple_regions = regionprops(apple_cc, 'Area');
appleAverageArea = mean([apple_regions.Area]);

% Trim based on CCA
banana_cc_clean = bwareaopen(banana_cc, ceil(bananaAverageArea));
orange_cc_clean = bwareaopen(orange_cc, ceil(orangeAverageArea));
apple_cc_clean = bwareaopen(apple_cc, ceil(appleAverageArea / 1.5));

countFruit(banana_cc_clean, apple_cc_clean, orange_cc_clean);
% Apply just a color to the mask
combo_color_img = img;
combo_color_img(:, :, :) = 0;

combo_color_img(:, :, 1) = uint8(banana_cc_clean) * 255;
combo_color_img(:, :, 2) = uint8(apple_cc_clean) * 255;
combo_color_img(:, :, 3) = uint8(orange_cc_clean) * 255;
% Add centroids to image
```

```
   bananaRegionMeasurements = regionprops(banana_cc_clean, 'Centroid');

   bananaCentroids = cat(1, bananaRegionMeasurements.Centroid);

   appleRegionMeasurements = regionprops(apple_cc_clean, 'Centroid');

   appleCentroids = cat(1, appleRegionMeasurements.Centroid);

   orangeRegionMeasurements = regionprops(orange_cc_clean, 'Centroid');

   orangeCentroids = cat(1, orangeRegionMeasurements.Centroid);


   combo_color_img = setCentroidPixelsWhite(combo_color_img, [bananaCentroids;
appleCentroids; orangeCentroids]);

   retValue = combo_color_img;

end

function retValue = setCentroidPixelsWhite(image, centroids)

   for k = 1:size(centroids, 1)

       x = round(centroids(k, 1));

       y = round(centroids(k, 2));

       pixelCoords = [y, x; y, x+1; y, x-1; y+1, x-1; y+1, x; y+1, x+1; y-1, x-1;
y-1, x; y-1, x+1];

       for idx = 1:size(pixelCoords, 1)

           coordY = pixelCoords(idx, 1);

           coordX = pixelCoords(idx, 2);

           image(coordY, coordX, 1) = 255;

           image(coordY, coordX, 2) = 255;

           image(coordY, coordX, 3) = 255;

       end

   end

   retValue = image;

end
```