Computer Vision Laboratories

Project 5

EE/CPE 428 - 03

Computer Vision

Winter 2023

Group 12

Students: Fadi Alzammar, Ryan Geisen, Nathan Jaggers

Project Due: 03/09/23

Instructor: Dr. Zhang

Part A - Strawberry Color Analysis

Assignment

In this assignment, you will analyze strawberry pixels and non-strawberry pixels in different color spaces and identify the best color component(s) to perform strawberry detection.

1. To build the strawberry-color model for strawberry detection, use all 20 images from the folder PartA, and manually draw a polygonal region of interest (ROI) around the strawberry to include as many strawberry pixels as possible in the ROI while leave non-strawberry pixels outside the ROI. Create a set of 20 mask images in which 1's indicate strawberry pixels and 0's indicate non-strawberry pixels. Label and save those mask images. Since the 20 images are of different sizes, you might want to first resize images with high resolutions to keep all images at around the same size of the smallest image.
2. Examine color distributions for strawberry and non-strawberry pixels in RGB color space. For this, construct a histogram of all strawberry pixels from the images in folder PartA with the help of the mask images and compare it with the histogram of all non-strawberry pixels from the same set of images. You need to construct a total of 3 sets of histograms. Examine color distributions between strawberry and non-strawberry pixels. What do you observe?
3. Experiment with two other color spaces- normalized rgb and HSV (Here since hue is defined on a ring, it might be advantageous to represent strawberry hue in a continuous range, how can you do it?). Again, examine color distributions for the strawberry and non-strawberry pixels in those color spaces.
4. What is/are the best feature(s) for characterizing strawberry-color? Why?

Procedure

To prepare for the color analysis of the strawberries, we first need to adjust our dataset by resizing the 20 images to all be the same size. This can be accomplished by using the imresize function and by passing the values 183 and 259. These values were chosen because they were found to be the smallest dimensions amongst all the photos. An unintended consequence of resizing the photos is that some of them were slightly distorted from their original form as seen in figures 1 and 2, however they were still usable as a part of the data set. The masks to differentiate strawberry pixels from non-strawberry pixels were created using ROIpoly. Note how we are defining strawberry pixels to be the red area of the fruit and not the green leaves that sit atop it.

Using the properly sized images and the newly created masks, we can now do a color analysis on the images. For each image we can observe the RGB values for the pixels that make up the strawberries and the background. When accessing an image we can isolate a color by selecting a channel of the image, then we can take that matrix and use the mask to separate the pixel based on if it is used for a strawberry or not.

```
color = (im{i}(:,:,channel));
…
c_s = double(color(masks{1} == 1));
c_b = double(color(masks{1} == 0));
```

We can then take these sorted values and concatenate them into two vectors that hold strawberry pixels and background pixels, both with the same RGB channels. Using the color data gathered from all 20 images, we can create histograms for the RGB used in strawberries vs the background.

Continuing the color analysis, normalized RGB and HSV follow a similar process. For normalized RGB all the steps are the same but we also normalize the color value of interest against all the color values in the pixel before producing the histogram. The following is an example of that.

```
% example
% Normal_Red =  R/(R+G+B)
n_rgb_s{1} = rgb_s{1}./(rgb_s{1} + rgb_s{2} + rgb_s{3});
```

To do an analysis for HSV we first must convert the RGB images to HSV format using rgb2hsv. Then the process is the exact same as for RGB, separate by channels and find which values go to a strawberry vector or background vector, then create a histogram. The only caveat with this is when creating the histogram for hue, red is split between the really low bins and really high bins. Hue is a continuous range so introducing an offset allows for the viewing of a continuous red range.

Results

Figure 1 : Original Strawberry image at 3456 x 5184

Figure 2 : Resize of image at 183 x 259

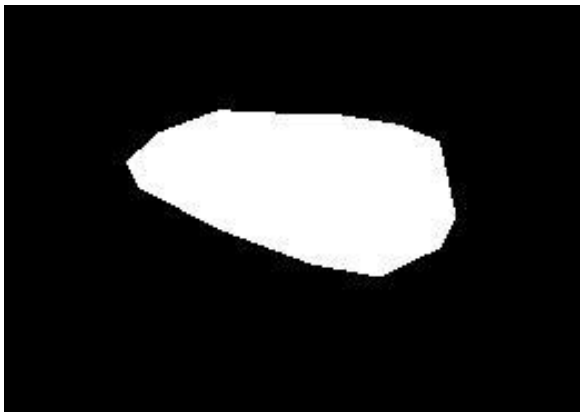

Figure 3 : Strawberry Mask

When separated into RGB histograms, shown in Figure 4, we see that the red components of most background pixels fall in the intensity range of about 25 to 150. This differs from red components of strawberry pixels, which are closer to 250, almost saturating that color intensity for most pixels. The green components of strawberry pixels tend to lie in the lower range of intensities but are still fairly spread out, while those of background pixels again favor the midrange, this time from 25 to 200. Both the backgrounds and strawberries show a lack of intense blue components.

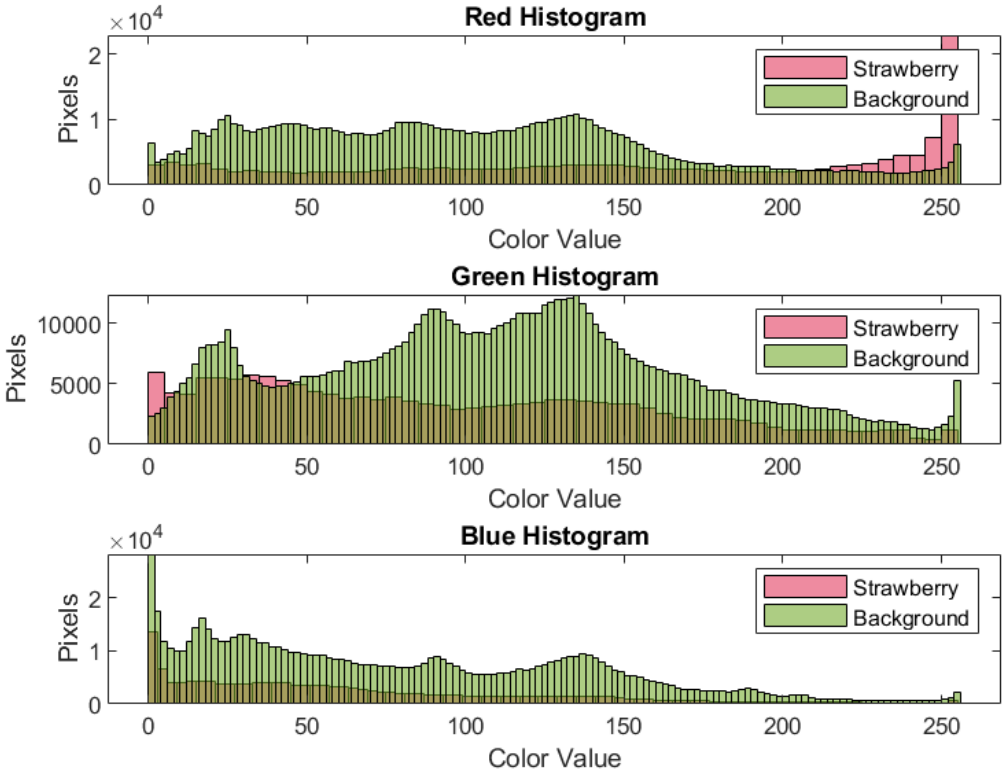

Figure 4 : Comparing RGB Strawberry and Background Histograms

The normalized RGB color space, represented in Figure 5, shows a slightly less significant difference between background and strawberry pixels than the pure RGB color space did. There is still a higher occurrence of large normalized red components among strawberry pixels, but the difference is much smaller than before due to the diverse, and potentially large, intensities of blue and green present shown previously. It still does show a similar pattern as before, with lower intensity green and blue components for strawberry pixels, so the same method could be used, identifying strawberry pixels by high red intensities.

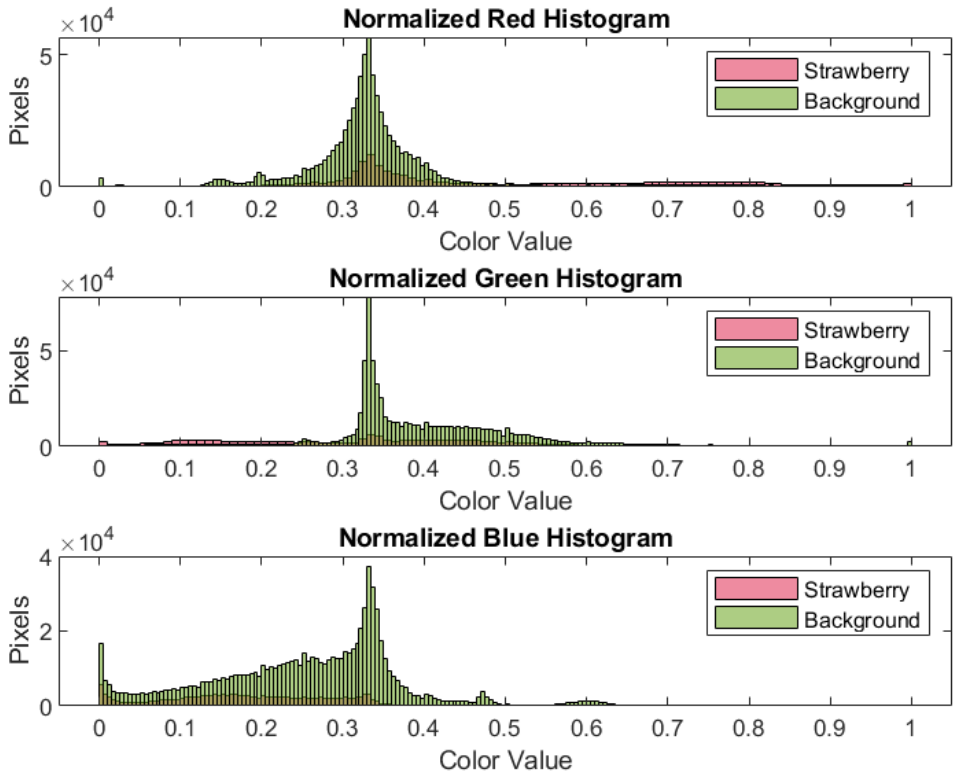

Figure 5 : Comparing Normalized RGB Strawberry and Background Histograms

The HSV color space, represented with Figure 6, shows the object in a very different perspective from what we've seen so far. With the strawberries mostly being a vibrant, bright red, it makes sense that the saturation and value components both have consistently high intensities, though the hue interestingly appears strongly at very low and very high values. The background shows a diverse range of saturation and value intensities, with the saturation slightly favoring low intensities and value slightly favoring the middle range, though both pick back up sharply at the high end. For both strawberry and background pixels, the hue peaks strongly at intensities of 0.2 to 0.3, though with strawberry pixels this gets overshadowed by the abundance of values at each extreme.

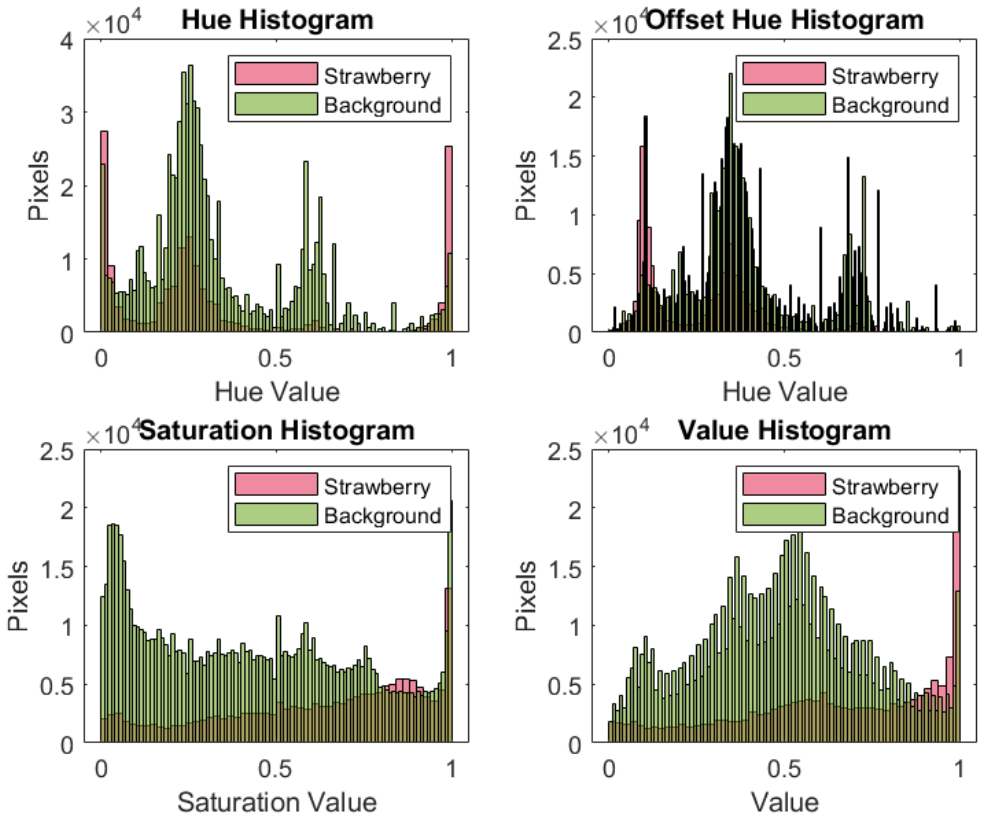

Figure 6 : Comparing HSV Strawberry and Background Histograms

It seems the hue and value histograms of the HSV color space could do a good job of identifying strawberries, with the very high concentration of extreme hue intensities and high value intensities. Red in the RGB or normalized RGB color spaces also seems like a good candidate, as one would naturally expect for strawberries. Of these, we would favor the pure red component, as the others have a fair amount of overlap between background and strawberry intensities even at the regions pointed out. This would result in more background pixels to be falsely identified as strawberries.

Part B - Strawberry Finding and Counting

Assignment

In this assignment, you will perform image segmentation using both the K-Means and EM algorithms. You will also try to find and count the strawberries in the image based on the segmentation results.

1. Run K-Means and EM clustering algorithms using RGB values of a color image. Perform image segmentation on images in folder Part B. Display and discuss the segmentation results with different initializations and different values of K. Show the best result.
2. Based on the segmentation results in the last step, develop a strawberry finding and counting algorithm. Place a white bounding box around the detected strawberry.
3. Select the best feature (or features) from Part A and develop a strawberry counting algorithm. Compare the result with that in step 2. Note: You can use the MATLAB built-in functions for K-Means and EM implementation.

Procedure

To perform the K-Means segmentation and the EM segmentation the desired images must first be read into the workspace. The images must then be resized so they are all 183 x 275 pixels. The image is then reshaped so that it is a 2 dimensional array so that it is 50,325x3 pixels. This reshaped array is then passed into the kmeans function along with the value k. K was chosen based on the number of clusters the cluster() function (used in the EM algorithm) would allow to converge. This was optimal because the more clusters we were able to use, the better defined the strawberries were, so the easier it was to separate the strawberries from the background. The fewer clusters used, the more the strawberries blended into the background and

the harder it was to extract them from the image. After running the image through the kmeans function, the image is then reshaped back to its original 183x275 shape.

For the EM algorithm the reshaped image must first be fitted to a gaussian mixture model with k clusters. This model was then passed into the cluster() function which separates the points into k clusters based on their probability of fitting in the cluster. Finally the image was reshaped back to its original 183x275 shape again. The results for the segmentation on all 6 images can be seen in figures 7-12.

To place boxes on the images, the label that was for the strawberries was chosen to make the image binary. The binary image was then passed into bwlabel to determine the continuous blobs that may or may not represent the strawberries in question. From those blobs any that were below a certain threshold were zeroed out so that only the strawberries were left. This threshold was chosen through trial and error which until only the strawberries were left in the binary image. The center of this image was found using regionprops and was labeled with a box and it's relative number to count the number of strawberries. The results of this program were very accurate for all images except for the final image due to the size and number of strawberries as well as the number of clusters limited by the cluster function for this image.

Results

When generating segmented images, quality of results, runtime, and ease of separation of strawberries from the environment was heavily reliant on K. There were several times when K had to be tailored to the image or algorithm because matlab would time out before finishing computations on a segment. Also looking at the figures below we can see how a higher or lower K value would affect how much information we could extract from the image.

Figure 7 & 8 : Image Segmentation using K-means and EM algorithms with K = 4

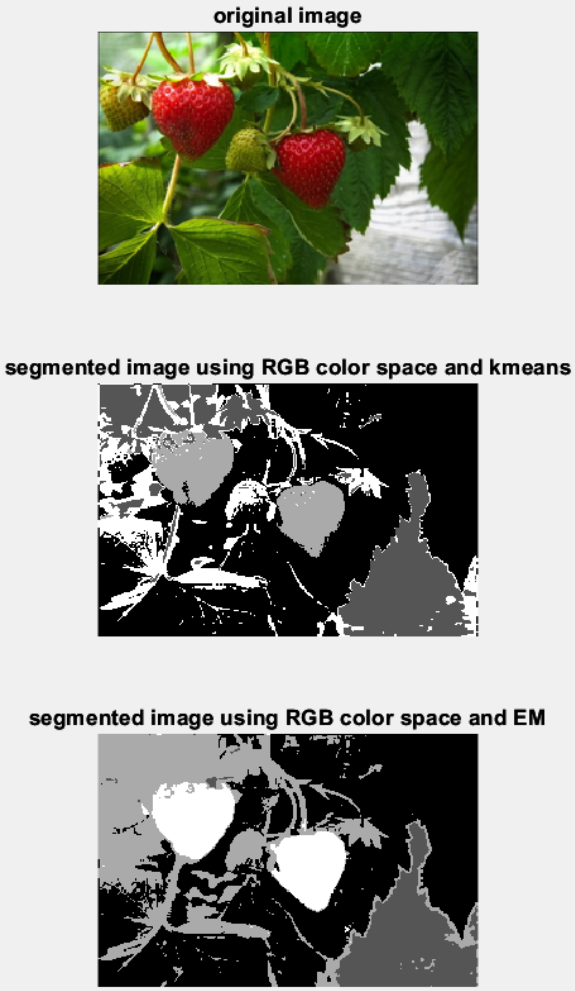

Figure 9 : Image Segmentation using K-means and EM algorithms with K = 4

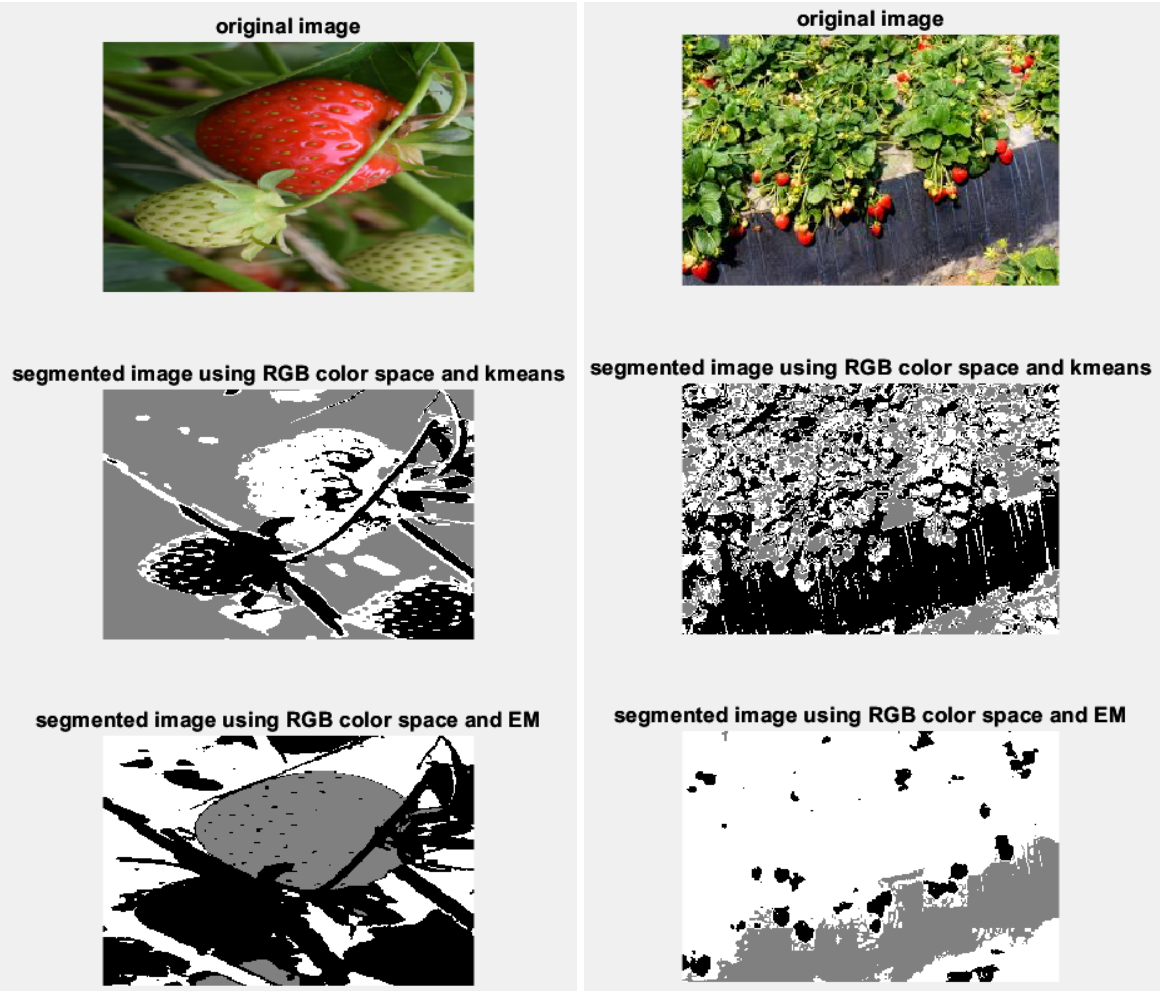

Figure 10 & 11 : Image Segmentation using K-means and EM algorithms with K = 3

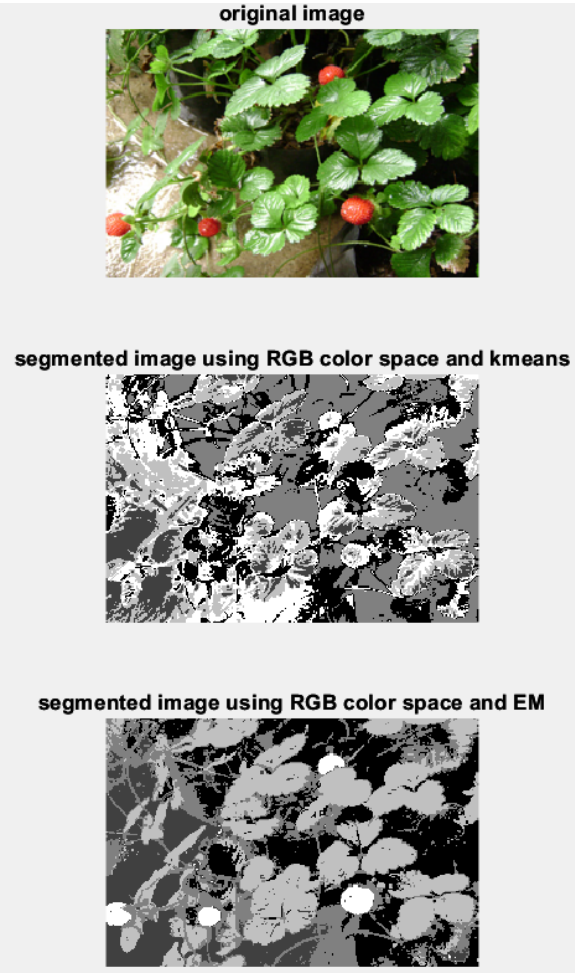

Figure 12 : Image Segmentation using K-means and EM algorithms with K = 5

For all of the segmentation results, the EM algorithm provided the best results. The strawberries were much more clearly picked out from the background and the segmentation was much more refined and had less blocky artifacts. These segmentations are seen above in figures 7-12

Figure 13 & 14 : Bounding Box of Segmented Strawberries

Figure 15 & 16 : Bounding Box of Segmented Strawberries

Figure 17 & 18 : Bounding Box of Segmented Strawberries

Figure 19 & 20 : Bounding Box and Counting of Segmented Strawberries

Figure 21 & 22 : Bounding Box and Counting of Segmented Strawberries

Figure 23 & 24 : Bounding Box and Counting of Segmented Strawberries

The strawberries were clearly found and boxed. This worked very well for the first 5 images but for the final image some of the strawberries were lost in the background and others were considered part of the same cluster as other strawberries. For the second image the script also counted unripe strawberries which was not ideal but was necessary to include the smaller strawberries hiding in the bush. The RGB color space was determined perfectly acceptable for these images since it easily picked out all of the strawberries.

Reflection

Fadi

This project covered the analysis of the RGB, normalized RGB, and HSV color spaces, and implementation of the K-means and EM algorithms. The RGB color spaces were, of course, already intuitive for me, but I didn't have a great intuition for how hue and saturation affected a color until now. Likewise, I already had a pretty good understanding of K-means segmentation from the lecture, but it took some time to understand EM. I found it interesting to see how the results of these could change so drastically run-to-run, which makes me consider the effects that may have in real-world systems utilizing them, for example false negatives or positives of a quality assessment process.

Ryan

I learned that the EM segmentation functions that are inherent to matlab are rather fickle and that the output of the clusters() function is not always the same. I found this to be incredibly frustrating as it made the results of each image change with each run of the program. I know this is just the nature of the EM algorithm but it was extremely annoying to deal with. I enjoyed seeing how the segmentation results turned out though and thought it was cool that the EM algorithm was so much better than the k-means. It only makes me even more in awe at how well the mean-shift algorithm works.

Nathan Jaggers

This project has been a very fun and interesting exploration of color analysis and segmentation with K-means and EM. During Part A it was interesting to see how my initial thoughts on which features would best describe the strawberries would change over time. Trying to interpret the

normalized RGB was both intuitive and an obstacle. I think this portion was very effective in reinforcing the information we learned about during the lecture. Through both these exercises I have seen how lighting can play a big role in the results of an image. In Part B it was very fun to see K-means and EM algorithms at work, it also became painfully obvious how computationally intensive they were especially when you want to segment the image more. Just like we discussed in class there seems to be a sweet spot for the K value, one that gets you enough information without being overfit.

Teamwork

Fadi

Did the segmentation portion of Part B

Ryan

Did strawberry finding portion of part B

Nathan

Did Part A of Project

Appendix

Part A

```
%% Project 5
%
% EE/CPE 428 - Computer Vision
% Winter 2023
%
% Group 12: Nathan Jaggers, Fadi Alzammar, Ryan Geisen
%
%% Part A
close all;
clear;
clc;
%%
% read in images from folder
dataDir = "Part A Data/";
filePattern = fullfile(dataDir, "s*");
theFiles = dir(filePattern);
im = cell(20,1);
for k = 1:length(theFiles)
   imname = theFiles(k).name;
   im{k} = imread(dataDir + imname);
end
% go through images and resize them to min length and width
% this ends up squishing some of the images,
for i = 1:size(im,1)
   im{i} = imresize(im{i}, [183 259]);
   % use the following to display images one by one
   %imshow(im{i});
   %input("Next");
end
%%
% use ROIpoly to crop out areas of interest in image
% save masks in folder
% if mask already exsits, skip it
dataDir = "Part A Masks/";
filePattern = fullfile(dataDir, "m*");
theMasks = dir(filePattern);
masks = cell(20,1);
maskNames(1:20) = "";
for i = 1:length(theMasks)
   maskNames(i) = theMasks(i).name;
end
for i = 1:size(im,1)
   mask = sprintf("m%s.jpg",int2str(i));
   if (find(maskNames == mask))
       masks{i} = imread(dataDir + mask);
       masks{i} = (masks{i})>200;
       imshow(masks{i});
```

```
        input("Next");
    else
        masks{i} = roipoly(im{i});
        imshow(masks{i});
        maskName = sprintf("Part A Masks/%s",mask);
        imwrite(masks{i},maskName);
        input("Next");
    end
end
%%
% go through images with masks and create histogram of strawberry pixels vs
% non strawberry pixels for R, G and B
% store results of separating pixels by color and strawberry-ness
rgb_s = cell(3,1); rgb_b = cell(3,1);
hsv_s = cell(3,1); hsv_b = cell(3,1);
% loop through channels to isolate rgb or hsv values
for channel = 1:3
    % loop through each image and mask. pull out strawberry and background
    % pixels
    for i = 1:length(im)
        color = (im{i}(:,:,channel));
        alt = rgb2hsv(im{i});
        vector = alt(:,:,channel);
        % store strawberry and background pixels in different vectors
        c_s = double(color(masks{1} == 1));
        c_b = double(color(masks{1} == 0));
        v_s = double(vector(masks{1} == 1));
        v_b = double(vector(masks{1} == 0));
        %concatonate information from single image to total dataset
        rgb_s{channel} = [rgb_s{channel}; c_s];
        rgb_b{channel} = [rgb_b{channel}; c_b];

        hsv_s{channel} = [hsv_s{channel}; v_s];
        hsv_b{channel} = [hsv_b{channel}; v_b];
    end
end
%%
% create histograms for RGB
figure('Name','Red');
histogram(rgb_s{1});
hold on;
histogram(rgb_b{1});
title('Red Histogram');
legend('Strawberry','Background');
hold off
figure('Name','Green');
histogram(rgb_s{2});
hold on;
histogram(rgb_b{2});
title('Green Histogram');
legend('Strawberry','Background');
hold off
figure('Name','Blue');
histogram(rgb_s{3});
```

```
hold on;
histogram(rgb_b{3});
title('Blue Histogram');
legend('Strawberry','Background');
hold off
%%
% now create histograms for normalized rgb
n_rgb_s = cell(3,1);
n_rgb_b = cell(3,1);
n_rgb_s{1} = rgb_s{1}./(rgb_s{1} + rgb_s{2} + rgb_s{3});
n_rgb_s{2} = rgb_s{2}./(rgb_s{1} + rgb_s{2} + rgb_s{3});
n_rgb_s{3} = rgb_s{3}./(rgb_s{1} + rgb_s{2} + rgb_s{3});
n_rgb_b{1} = rgb_b{1}./(rgb_b{1} + rgb_b{2} + rgb_b{3});
n_rgb_b{2} = rgb_b{2}./(rgb_b{1} + rgb_b{2} + rgb_b{3});
n_rgb_b{3} = rgb_b{3}./(rgb_b{1} + rgb_b{2} + rgb_b{3});
%%
% create histograms for normalized RGB
figure('Name','Norm Red');
histogram(n_rgb_s{1});
hold on;
histogram(n_rgb_b{1});
title('Normalized Red Histogram');
legend('Strawberry','Background');
hold off
figure('Name','Norm Green');
histogram(n_rgb_s{2});
hold on;
histogram(n_rgb_b{2});
title('Normalized Green Histogram');
legend('Strawberry','Background');
hold off
figure('Name','Norm Blue');
histogram(n_rgb_s{3});
hold on;
histogram(n_rgb_b{3});
title('Normalized Blue Histogram');
legend('Strawberry','Background');
hold off
%%
% create offset Hue vector to see red values together on histogram
offset = 0.1;
offset_hue_s = size(hsv_s{1});
for counter = 1:length(hsv_s{1})
   if (hsv_s{1}(counter) + offset) <= 1
       offset_hue_s(counter) = hsv_s{1}(counter) + offset;
   else
       offset_hue_s(counter) = hsv_s{1}(counter) - (1-offset);
   end
end
offset_hue_b = size(hsv_b{1});
for counter = 1:length(hsv_b{1})
   if (hsv_b{1}(counter) + offset) <= 1
       offset_hue_b(counter) = hsv_b{1}(counter) + offset;
   else
```

```
        offset_hue_b(counter) = hsv_b{1}(counter) - (1-offset);
    end
end
%%
% create histograms for HSV
figure('Name','Hue');
histogram(hsv_s{1});
hold on;
histogram(hsv_b{1});
title('Hue Histogram');
legend('Strawberry','Background');
hold off
figure('Name','Offset Hue');
histogram(offset_hue_s);
hold on;
histogram(offset_hue_b);
title('Offset Hue Histogram');
legend('Strawberry','Background');
hold off
figure('Name','Saturation');
histogram(hsv_s{2});
hold on;
histogram(hsv_b{2});
title('Saturation Histogram');
legend('Strawberry','Background');
hold off
figure('Name','Value');
histogram(hsv_s{3});
hold on;
histogram(hsv_b{3});
title('Value Histogram');
legend('Strawberry','Background');
hold off
```

Part B

main

```
%Read in all the images
sberries{1} = imresize(imread("t1.JPG"), [183,275]);
sberries{2} = imresize(imread("t2.JPG"), [183,275]);
sberries{3} = imresize(imread("t3.JPG"), [183,275]);
sberries{4} = imresize(imread("t4.JPG"), [183,275]);
sberries{5} = imresize(imread("t5.JPG"), [183,275]);
sberries{6} = imresize(imread("t6.JPG"), [183,275]);
%Segment the images
[m,n] = size(sberries{1}(:,:,1));
segmented_k={};
segmented_em={};
for i=1:6
    %Choose the appropriate number of clusters for each image
```

```
   if i == 1
       k=4;
   elseif i ==2
       k=4;
   elseif i==3
       k=3;
   elseif i ==4
       k=4;
   elseif i ==5
       k=5;
   else
       k=3;
   end
   %reshape the image to be 2 dimensional
   x = reshape(sberries{i},m*n,3);
   %find the kmeans clusters
   [x_clustered_k, center] = kmeans(double(x), k,'Distance', 'sqeuclidean');
   %reshape segmented kmeans image to original shape
   segmented_k{i}=reshape(x_clustered_k,m,n);
   %fit image to gaussian mixed distribution
   x_dist=fitgmdist(double(x),k);
   %Cluster the image to k clusters
   x_clustered_em=cluster(x_dist, double(x));
   %Reshape the segmented EM image to original shape.
   segmented_em{i}=reshape(x_clustered_em, m, n);
end
%%
%Display segmented images with original image
for i = 1:6
   figure()
   subplot(311), imshow(sberries{i},[]), title('original image')
   subplot(312), imshow(segmented_k{i}, []), title('segmented image using RGB color
space and kmeans')
   subplot(313), imshow(segmented_em{i},[]), title('segmented image using RGB color
space and EM')
end
%%
strawbs = {};
foundstrawbs = {};
for im = 1:6
   if im == 1
       %label binary image
       strawbs{im} = bwlabel(segmented_em{im}==3);
       %find blobs not large enough to be strawberries
       foundstrawbs{im} = StrawFind(strawbs{im}, 500);
       %Get rid of blobs not large enough to be strawberries
       strawbs{im} = NonStrawClear(strawbs{im}, foundstrawbs{im});
       %Box and number strawberries
       BoxStrawbs(sberries{im},strawbs{im},80,70)
   elseif im == 2
       %label binary image
       strawbs{im} = bwlabel(segmented_em{im}==4);
       %find blobs not large enough to be strawberries
       foundstrawbs{im} = StrawFind(strawbs{im}, 150);
```

```
        %Get rid of blobs not large enough to be strawberries
        strawbs{im} = NonStrawClear(strawbs{im}, foundstrawbs{im});
        %Box and number strawberries
        BoxStrawbs(sberries{im},strawbs{im},50,50)
    elseif im == 3
        %label binary image
        strawbs{im} = bwlabel(segmented_em{im}==2);
        %find blobs not large enough to be strawberries
        foundstrawbs{im} = StrawFind(strawbs{im}, 1000);
        %Get rid of blobs not large enough to be strawberries
        strawbs{im} = NonStrawClear(strawbs{im}, foundstrawbs{im});
        %Box and number strawberries
        BoxStrawbs(sberries{im},strawbs{im},200,100)
    elseif im == 4
        %label binary image
        strawbs{im} = bwlabel(segmented_em{im}==4);
        %find blobs not large enough to be strawberries
        foundstrawbs{im} = StrawFind(strawbs{im}, 1500);
        %Get rid of blobs not large enough to be strawberries
        strawbs{im} = NonStrawClear(strawbs{im}, foundstrawbs{im});
        %Box and number strawberries
        BoxStrawbs(sberries{im},strawbs{im},50,75)
    elseif im == 5
        %label binary image
        strawbs{im} = bwlabel(segmented_em{im}==5);
        %find blobs not large enough to be strawberries
        foundstrawbs{im} = StrawFind(strawbs{im}, 150);
        %Get rid of blobs not large enough to be strawberries
        strawbs{im} = NonStrawClear(strawbs{im}, foundstrawbs{im});
        %Box and number strawberries
        BoxStrawbs(sberries{im},strawbs{im},25,50)
    else
        %label binary image
        strawbs{im} = bwlabel(segmented_em{im}==1);
        %find blobs not large enough to be strawberries
        foundstrawbs{im} = StrawFind(strawbs{im}, 100);
        %Get rid of blobs not large enough to be strawberries
        strawbs{im} = NonStrawClear(strawbs{im}, foundstrawbs{im});
        %Box and number strawberries
        BoxStrawbs(sberries{im},strawbs{im},25,50)
    end
end
```

StrawFind

```
function outarray = StrawFind(segim,thresh)
```

```
outarray = [];
%For all blobs
for i = 1: max(segim(:))
   %If there are less pixels than the threshold
   if sum(segim(:)==i) > thresh
       %add the index to an array of valid values
       outarray = cat(2,outarray,i);
   end
end
end
```

NonstrawClear

```
function outim = NonStrawClear(segim,labelarray)
[m,n] = size(segim);
%for the entire image
   for i = 1:m
       for j =1:n
           %if the pixel value is not in the strawberry array
           if ~ismember(segim(i,j), labelarray)
               %clear value
               segim(i,j) = 0;
           end
       end
   end
   %relabel image so that labels are minimum numbers
   outim = bwlabel(segim);
end
```

BoxStrawbs

```
function BoxStrawbs(im,bim, width, height)
%Find the center of the blobs
centroids = regionprops(logical(bim),'Centroid');
%Turn centroids into an array
centroids = cat(1, centroids.Centroid);
%display image
figure()
imshow(im)
%Place a Box around the strawberries and number them
for i = 1:length(centroids(:,1))
   %get left edge of box
   xleft = centroids(i,1)-width/2;
   %get bottom edge of box
   ybot = centroids(i,2)-height/2;
   %Put rectangle around strawberries
   rectangle('Position',[xleft, ybot, width, height], 'EdgeColor', 'w')
   %number the strawberries
   text(centroids(i,1),centroids(i,2),string(i),'FontWeight','bold',
'HorizontalAlignment','center')
end
```