**Color Segmentation of Images Using K-Means Clustering With Different Color Spaces by Shiv Nath Yadav( mit2018041)**

**The Idea:**

As we learned in class, the image segmentation problem is ill-defined, and usually very hard to execute, since different people can choose different segmentations for the same image.

However, as we were told, segmentation is still used extensively, and can be very effective when used on images under specific circumstances, like a defined environment, with specific features we know that are going to be present, and distinguish segments from one another.

Using these ideas, we decided to use images with known features, where most people will agree about the segmentation, and the feature we use for the segmentation is **color** : this world is the world of pictures taken from **Fruit Ninja** gameplay.

As most of you know, the different figures in the game are fruits, which usually have very different colors, and so a segmentation by colors usually gives a very correct feel of the picture, and can be a good preprocessing for stages like object recognition and even a fruit ninja solver algorithm which can play the game and maximize scores.

We used an algorithm learned in class, the K-Means Clustering algorithm, and tried to cope with its pros and cons using different methods, presented below.

**Our Goals:**

Considering the pros and cons of the main algorithm used (K-Means Clustering)

And the fact that segmentation can be perceived differently by different people, we set out minds to several goals:

-Good Segmentation : Given an image (usually from fruit ninja gameplay) in generic viewpoint, we wish to produce the best possible segmentation, by this we mean that most people would choose this segmentation if asked to make one, and it would make a good precursor for object recognition.

-Minimizing Human Interaction : As we know, the K-Means Clustering algorithm, as hinted by the name, requires a parameter, K, as part of its input to run.

Different Ks can give totally different results, and usually, there is one K which is consistent with the data points, the actual K.

The biggest disadvantage of our heavy usage of k-means clustering, is that it means we would have to think of a k each time, which really doesn’t make too much sense because we would like to algorithm to solve this on his own.

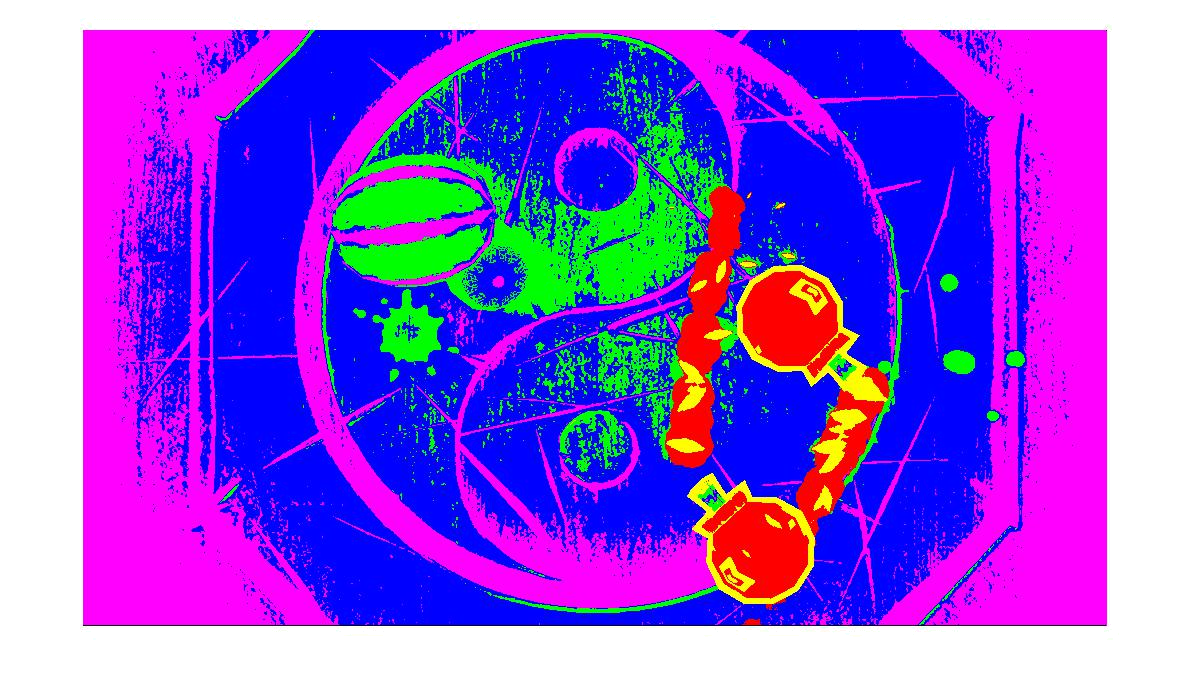
Therefore, our second big goal, is to try and make use of the data given to us, and our knowledge gained by the fact that all of the images are from the game, to try and find

the K automatically, and so, create a segmentation without any human giving “hints” to the algorithm.

**Our Work:**

We first wanted to find an interesting way to segment the image by the colors present in it we read about an idea that uses the k means clustering algorithm, which involves mapping the image pixels to the RGB color space, and just using the k means clustering on it (i.e if pixel 4,5 holds the values red:50,green:30,blue:20, we map it to the point (50,30,20) on a 3d space representing the RGB values).

We made some function to adjust the input to the format the k means procedure expects to get, and ran a few tests, the results weren’t too impressive in our eyes (and can be seen in the “Results” section of this paper, but here is a taste of how it looks like (after processing and coloring each segment with a different color):



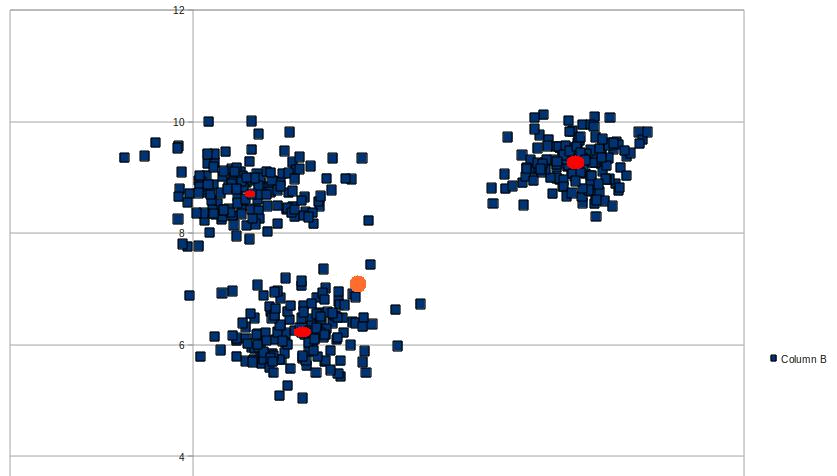
We decided to bite the bullet, and keep using this method, while focusing on ways to minimize the K parameter, and try to estimate it automatically (the motivation to keep the K parameter as small as possible, was to keep different intensity levels of the same color on the same segment, for example trying to keep light green and dark green in the same segment).

We learned about different ways to choose the right K for the algorithm, like in several algorithms people made : G-Means clustering, which uses statistics and hypotheses about the data to estimate the K, and X-means algorithm which didn’t seem to give a result that would satisfy.

Because of that we tried to think of a way, using to our advantage the fact that images from the game world aren’t too diverse.

Our basic hypothesis for this part of the work, was that if given a data set of n clusters, when we run K means clustering with parameter k=m , m>n,

The centers of the clusters created by the algorithm would be closer to each other than if we ran the algorithm with k=n (the actual number of clusters), or at least, the minimum distance between the cluster centers would decrease.



And so we made a sketch of an algorithm to minimize the K parameter,

We start with a high value of K, and iteratively reduce it, until we meet a certain condition, which usually has to do with the minimal distance between the centers of the segmentation produced by the k-means algorithm.

Now, the problem is to choose a condition which will make the algorithm stop and the “right” k.

We had an idea, on how to choose this condition, based on our knowledge of the fruit ninja images, like the colors existing in the world.

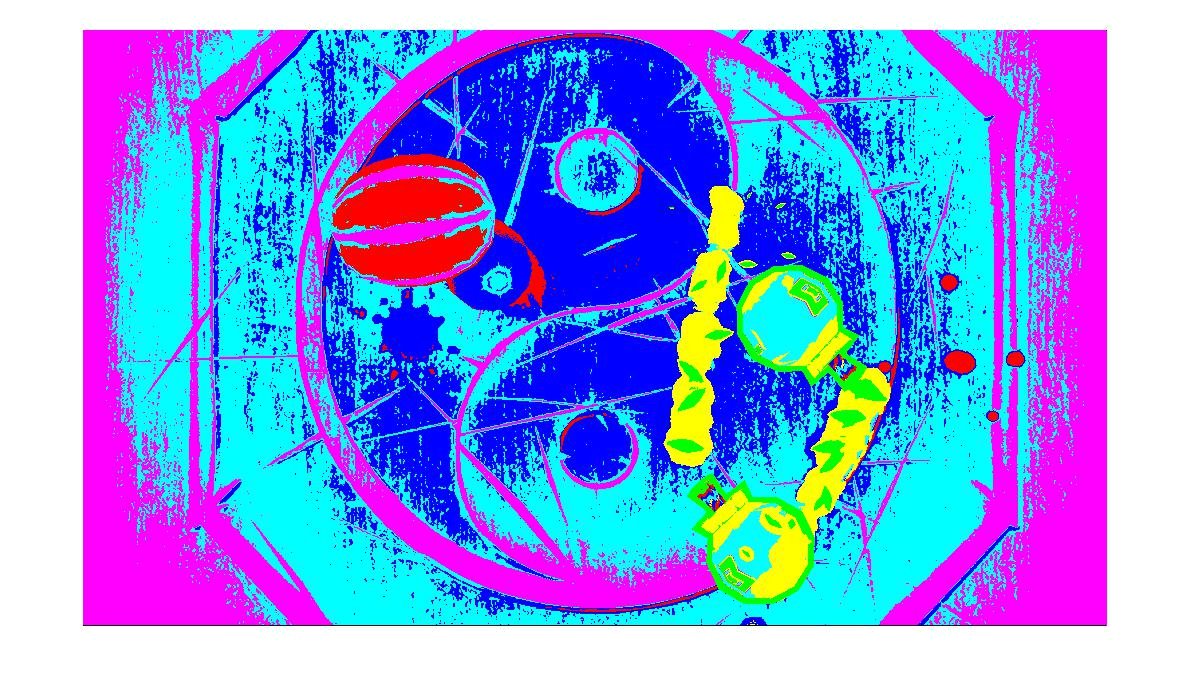
We used matlab to sample the RGB values, of the colors we wanted to separate, we wanted to get the algorithm to a point where it wouldn’t unify those colors under the same segment. We chose the colors of the different fruits available, and of course the background color.

If we compute the minimal distance between the colors (in the RGB 3d space),

We can now use a condition, that will force the k to be small enough that the different segments produced will be at least as far from one another, as the different colors of the game world, and so we hope to make sure that we wont get a segment with colors “close to one another”. (we relied heavily on our basic hypothesis, combined with hope that different intensities of the same color, would be closer to one another than different colors at all, and since decreasing K increases the minimal distance, our algorithm ends.)

After the results, we now tried a fully automatic run of the algorithm, by sampling the different colors from **all** the different fruits in the game, and using the minimal distance between the fruits as a starting point for an terminating condition.

Here’s an example of the results:



The automatic run, estimated a k=6, as opposed to what we thought was the optimal segmentation which was k=5.

Considering that this is a np-hard problem, the results were fine, but we still thought they weren’t satisfying, and still remember that the optimal results as shown above(when the “right” k is given) weren’t too good either.

All of this lead us to try a different approach, we identified that our main issue with this result, is that areas that we perceive to be one segment, were assigned to different segments in the result, for example, the background got separated to two segments. Some may argue that if we segment the image based only on colors, but we wanted it to be a good pre-processing for a later, object recognition algorithm.

We had several ideas, for example, we used the theory we learned in class to think of a different kind of condition for our algorithm (keeping the basic sketch of the algorithm as it is) : we know from the class, that different “shades” of the same color (different intensities) are basically multiplication by a constant of a basic vector in the RGB space, therefore, if in the data set we have several clusters, each one representing a different intensity level of our color, they will all be around one slant in the RGB space, and therefore can be merged together.

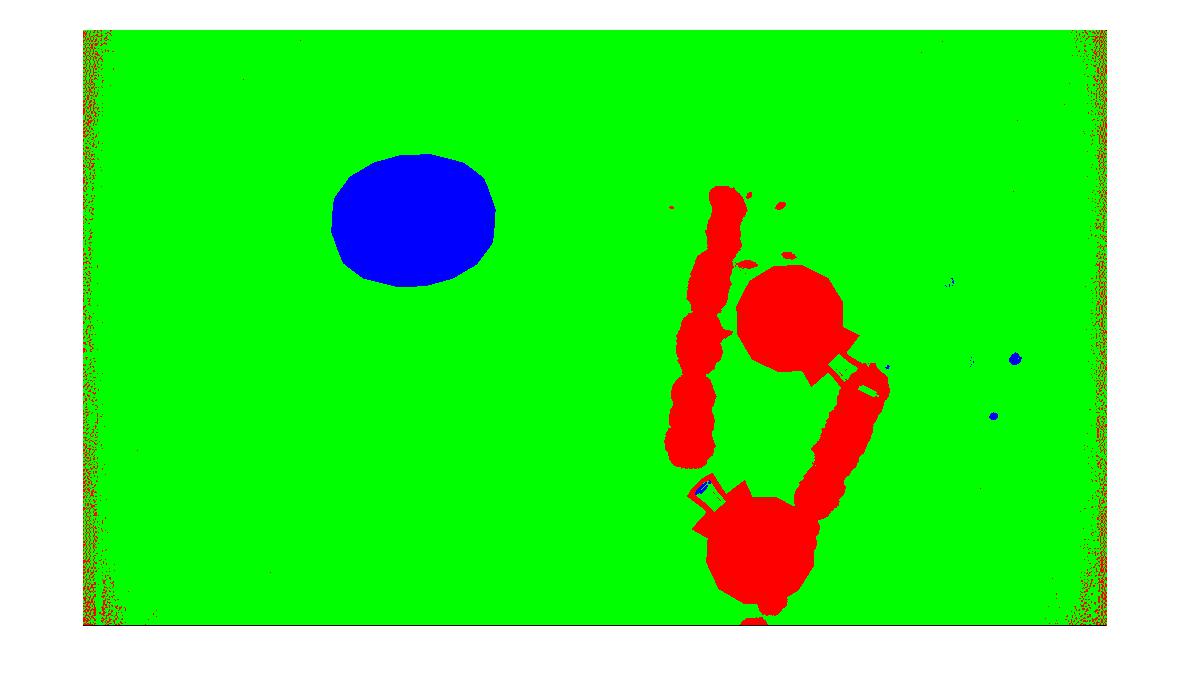
But as we started implementing this method, we suddenly realized there is a much simpler solution.

As we learned, different representations of the same object can be useful for different applications, each representation has its pros and cons.

Our main issue was to get rid of different intensity level of the same color hue, And that’s when we realized there was another representation to color images, the HSV color space.

The HSV color space has different parameters for the HUE, the SATURATION and the VALUE (intensity) of the pixel, and so we get easy access to the actual hue of the pixel, without any concerns about different intensity levels.

We gave it a try, and the results (as shown in the “Results” section) are much better than before :



Now we used our former ideas on the new color space and these are examples of the

results:

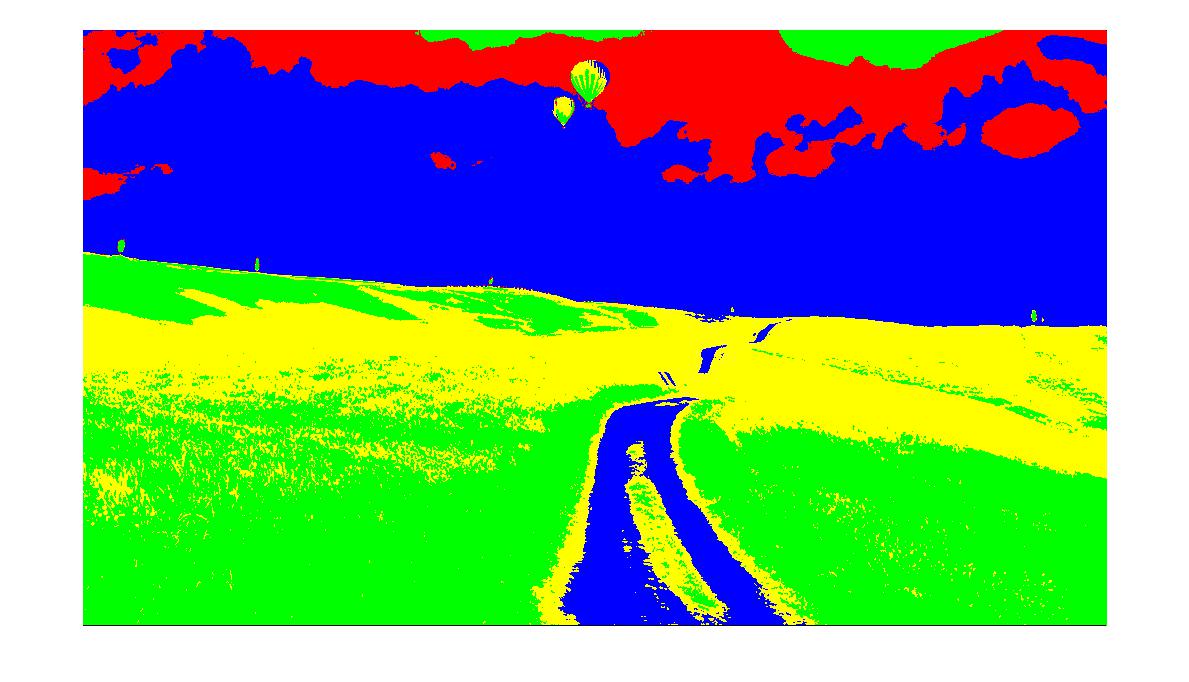
Again, the fully automatic algorithm isn’t perfect, the usage of the minimum distance condition isn’t enough to accurately estimate the K parameter but we do ideas on how to improve it.

In the restricted world of Fruit Ninja images, HSV based segmentation did much better than the RGB version. The fact that we can ignore the pixels intensity level opened up a way to clearly segment the picture based on hue only, which, in this scenario specifically is very good.

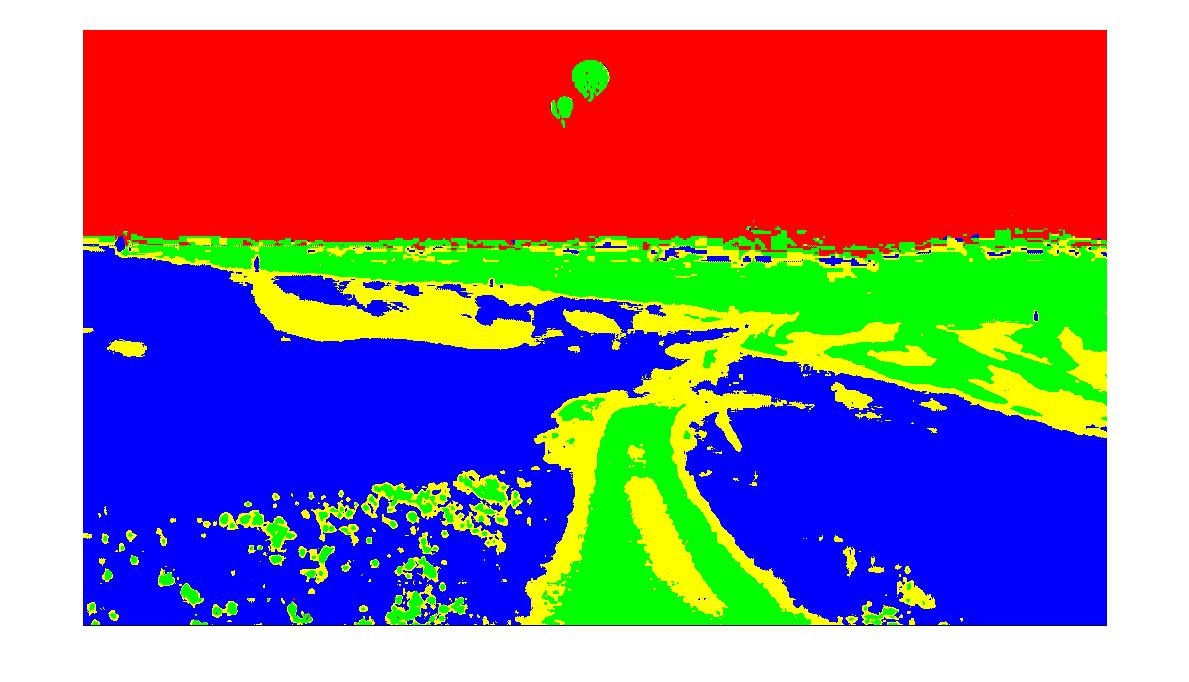
However we wanted to emphasize the fact that no representation is better than the other, so we provide an example of a real world image, where obviously our trick of using the HSV color space to eliminate different intensities of the same hue isn’t performing too well, as opposed to RGB



**RGB:**



**HSV:**



To conclude, we achieved various results of the color segmentation problem, the major step we did was the move to the HSV color space to get a clearer segmentation, of course there are many things we can improve in the future, mostly the K recognition feature which still requires work to be fully functional

**Results:**

Here we will give several samples of the results we achieved, with some annotations

RGB:

As explained before, for these scenarios (Fruit Ninja images), the RGB isn`t performing exceptionally well, but here are some of our results for future comparison

**Original**

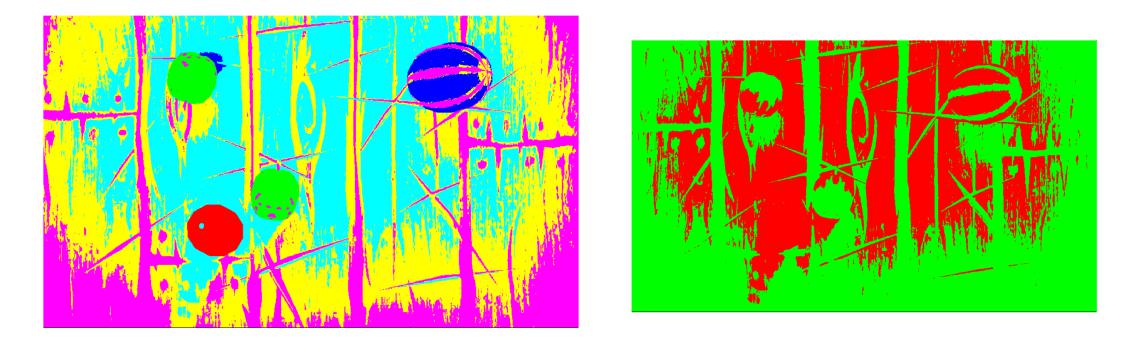
**Optimal**

**Auto**

K=5



K=2



**Conclusions:**

On segmentation:

In some cases, when dealing with a restricted world of images, segmentation can be very useful, and not too complicated to perform.

As seen in our case, a world where color segmentation usually is the “actual” segmentation of the image, we can get very good results, which could be easily combined in a larger system, for further processes like object recognition, and even go as far as real time fruit ninja solver.

On representations:

We were taught in class about the importance of having different representations of the same object, and how each one opens up a world of tools to use on it, lets you get a part of the information easily, while hiding some of the redundant information (for a specific use).

While working on this project, we finally had a taste of that, as switching to a different representation was exactly what we needed as we got stuck, and it gave us the option to improve the quality of our project greatly.

However, as we demonstrated before, this doesn’t mean that the HSV representation is better generally, because for real images, the RGB representation gave us better results, and this made us accept the sentence we heard in class entirely.

On choosing the right K for the K-Means algorithm:

We discovered that even though the K-Means algorithm does a great job, the fact that you have to choose a k makes using it a bit problematic.

When we tried to make the process automatic, even though we were in a restricted world of images, the job wasn’t easy, and obviously our results are not perfect, but not too far away.

On images of unknown source, the problem becomes much harder (We found the numbers we used to terminate the reduction of K based on samplings from fruit ninja images and manipulations on the results, general pictures are not supposed to work this way.)

After that’s done we can “solve” the game using that picture, find the best stroke you can do to maximize score.

And the last stage is to make an online Fruit Ninja solver using a video from the game.

-Finding the right K : The method we used to find the right k is far from perfect, to improve it we can try different methods, like the one we explained in the RGB section on slant based cluster merging, or trying to reach the correct K from “below”, i.e begin with low K value and reach a condition from above.

**My Code:**

function [I\_seg,cent] = colorseg(I,bg,mindi) I\_crop=I(105:700,:,:); I\_hsv=rgb2hsv(I\_crop);

tnai=0;

k=8;

I\_meansr=hsv2kmeans(I\_hsv);

while tnai<mindi

k=k-1;

[I\_means,centers]=kmeans(I\_meansr,k); tnai=mindistance(centers);

tnai

end

k=k+1;

[I\_means,centers]=kmeans(I\_meansr,k);

I\_set=kmeans2rgb(I\_means,bg);

figure()

imshow(I\_set)

k

I\_seg=I\_means;

cent=centers;

end

function [I\_seg,cent] = colorseg3(I,bg,mindi)

I\_crop=I(105:700,:,:);

%I\_hsv=rgb2hsv(I\_crop);

tnai=0;

k=8;

I\_meansr=rgb2kmeans(I\_crop);

while tnai<mindi

k=k-1;

[I\_means,centers]=kmeans(I\_meansr,k); tnai=mindistance(centers); tnai

end

k=k+1;

[I\_means,centers]=kmeans(I\_meansr,k); I\_set=kmeans2rgb(I\_means,bg);

figure()

imshow(I\_set)

k

I\_seg=I\_means;

cent=centers;

end

function [I\_p] = kmeans2rgb(I\_means,bg)

if bg==1

f=1;

elseif bg==0

f=4;

end

rows=595\*1024;

I\_p=zeros(595,1024,3);

counter=zeros(10,1);

for k=1:rows

counter(I\_means(k,1),1)=counter(I\_means(k,1),1)+1;

end

for k=1:rows

if I\_means(k,1)==1 && counter(1,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[100,0,0];

elseif I\_means(k,1)==2 && counter(2,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[0,100,0];

elseif I\_means(k,1)==3 && counter(3,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[0,0,100];

elseif I\_means(k,1)==4 && counter(4,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[100,100,0];

elseif I\_means(k,1)==5 && counter(5,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[100,0,102];

elseif I\_means(k,1)==6 && counter(6,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[0,100,101];

elseif I\_means(k,1)==7 && counter(7,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[50,100,96];

elseif I\_means(k,1)==8 && counter(8,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[100,50,97];

elseif I\_means(k,1)==9 && counter(9,1)<rows/4\*f

I\_p(floor(k/1024)+1,mod(k-1,1024)+1,:)=[100,150,98];

elseif I\_means(k,1)==10 && counter(10,1)<rows/4\*f I\_p(floor(k/1024)+1,mod(k-

1,1024)+1,:)=[100,0,150];

end

end

end

function [Out] = rgb2kmeans(I)

rows=1024\*595;

Out=zeros(rows,3);

for i=1:595

for j=1:1024

Out(((i-1)\*1024)+j,1)=I(i,j,1);

Out(((i-1)\*1024)+j,2)=I(i,j,2);

Out(((i-1)\*1024)+j,3)=I(i,j,3);

end

end

end

function [Out] = hsv2kmeans(I)

rows=1024\*595;

Out=zeros(rows,1);

for i=1:595

for j=1:1024

Out(((i-1)\*1024)+j,1)=I(i,j,1);

end

end

end