Matthew Vaysfeld
I pledge my honor that I have abided by the Stevens Honor System

Code

```
function main =
main(whitetower,wtslic,skymask,skytrain,skytest1,skytest2,s
kytest3,skytest4)
    arguments
        whitetower string = "white-tower.png";
        wtslic string = "wt_slic.png";
        skymask string = "sky_mask2.jpg";
        skytrain string = "sky_train.jpg";
        skytest1 string = "sky_test1.jpg";
        skytest2 string = "sky_test2.jpg";
        skytest3 string = "sky_test3.jpg";
        skytest4 string = "sky_test4.jpg";
    end
    f1 = figure('Name',"Kmeans output");
    f2 = figure('Name',"SLIC output WhiteTower");
    f3 = figure('Name',"SLIC output wtslic");
    f4 = figure('Name',"Test1 Pixel Classification");
    f5 = figure('Name',"Test2 Pixel Classification");
    f6 = figure('Name',"Test3 Pixel Classification");
    f7 = figure('Name',"Test4 Pixel Classification");


    whitetower = imread(whitetower);
    wtslic = imread(wtslic);
    skymask = imread(skymask);
    skytrain = imread(skytrain);
    skytest1 = imread(skytest1);
    skytest2 = imread(skytest2);
    skytest3 = imread(skytest3);
    skytest4 = imread(skytest4);

    %Part 1 main
    [kmeans,averages] = kMeans(whitetower,10);
    clusteredim = showcluster(kmeans,averages,whitetower);
    figure(f1);
    imshow(clusteredim);

    %Part 2 main
```

```
    croppedImage = crop(whitetower);

    %image,maximumiterations, and local flag that tells the
program whether to
    %recompute the local shift

    SLIC = SLICalgo(croppedImage,3);
    figure(f2);
    imshow(SLIC);

    SLIC2 = SLICalgo(wtslic,3);
    figure(f3);
    imshow(SLIC2);

    %Part 3 main

    %nonsky,sky,image
    classified = pixelclass(skymask,skytrain,skytest1,1);
    figure(f4);
    imshow(classified);

    classified2 = pixelclass(skymask,skytrain,skytest2,3);
    figure(f5);
    imshow(classified2);

    classified3 = pixelclass(skymask,skytrain,skytest3,3);
    figure(f6);
    imshow(classified3);

    classified4 = pixelclass(skymask,skytrain,skytest4,3);
    figure(f7);
    imshow(classified4);

end




%Part 1
function [kmeans,averages] = kMeans(image,k)
    %randomly generate 10 centers (prob fix this)
    [row,col,~] = size(image);
    centers = zeros(k,2);
```

```
    for i = 1:k
        centers(i,1)=randi(row);
        centers(i,2)=randi(col);
    end
    centercolors = [];
    for i=1:k
        centercolors=
cat(1,centercolors,RGBat(centers(i,1),centers(i,2),image));
    end
    oldvalues = zeros(k,3);
    oldvalues(1:k,1:3) = 1000000;
    while(1 == 1)
        [clustered,averages,flag] =
make_cluster(centercolors,k,image);
        centercolors = averages;
        if(flag == 1)
           [clustered,averages,~] =
make_cluster(centercolors,k,image);
           centercolors = averages;
        end
        if((abs(max(centercolors-oldvalues,[],'all')) < 1))
            break;
        end
        oldvalues = centercolors;
    end

    kmeans = clustered;




    %compare distance of every pixel to every center
(Norm(Vector1-Vector2)
    %put pixel in cluster thats closest

end


function [clustered,averages,flag] =
make_cluster(centers,k,image)
    flag =0;
    [row,col,~] = size(image);
```

```
    clustered = zeros(row,col);
    averages = zeros(k,3);
    averageamount = zeros(k,1);
    for i = 1:row
        for j = 1:col
            clusterno = 1;
            currentdist = 100000000000;
            pointcolor = RGBat(i,j,image);
            for f = 1:k
                a = norm(centers(f,:) - pointcolor);
                if(a < currentdist)
                    clusterno = f;
                    currentdist = a;
                end
            end
            clustered(i,j) = clusterno;
            averages(clusterno,1) = averages(clusterno,1) +
pointcolor(1);
            averages(clusterno,2) = averages(clusterno,2) +
pointcolor(2);
            averages(clusterno,3) = averages(clusterno,3) +
pointcolor(3);
            averageamount(clusterno) =
averageamount(clusterno) + 1;
        end
    end
    for t = 1:k
        if(averageamount(t) == 0)
            %takes random value from biggest cluster
            flag = 1;
            s=1;
            for m = 1:k
                if (averageamount(m) > averageamount(s))
                    s=m;
                end
            end
            while(1==1)
                g=randi(row);
                h=randi(col);
                if(clustered(g,h) == s)
                    point = RGBat(g,h,image);
                    averages(t,1) = point(1);
                    averages(t,2) = point(2);
                    averages(t,3) = point(3);
```

```
                    break;
                end
            end

         else
            averages(t,1) = averages(t,1) /
averageamount(t);
            averages(t,2) = averages(t,2) /
averageamount(t);
            averages(t,3) = averages(t,3) /
averageamount(t);
        end
    end
end

function RGB = RGBat(x,y,image)
    RGB = double(reshape(image(x,y,:),[1,3]));

end

function showcluster =
showcluster(clustered,averages,image)
    [row,col] = size(clustered);
    showcluster = image;
    for i = 1:row
        for j=1:col
            showcluster(i,j,1) =
averages(clustered(i,j),1);
            showcluster(i,j,2) =
averages(clustered(i,j),2);
            showcluster(i,j,3) =
averages(clustered(i,j),3);
        end
    end
end


%Part 3


function pixelclass = pixelclass(nonsky,sky,image,sigma)

pixelclass = image;
[row,col,~] = size(image);
```

```
[nsrow,nscol,~] = size(nonsky);

%Change to sigma of 3 when testing other images
noskygauss = imgaussfilt(nonsky,sigma);
skygauss = imgaussfilt(sky,1);
skypoints = [];
nonskypoints = [];

firstcolor = nonsky(1,1,:);

%Seperate the masked image into sky and nonsky points
for i=1:nsrow
    for j = 1:nscol
        if(noskygauss(i,j,:) == firstcolor)
            skypoints = cat(1,skypoints,skygauss(i,j,:));
        else
            nonskypoints =
cat(1,nonskypoints,skygauss(i,j,:));
        end
    end
end

[~,nonskymeans] = kMeans(nonskypoints,10);
[~,skymeans] = kMeans(skypoints,10);

means = cat(1,nonskymeans,skymeans);


%Figure out which word each pixel is closest too
for i = 1:row
    for j = 1:col
        clusterno = 1;
        currentdist = 100000000000;
        pointcolor = RGBat(i,j,image);
        for f = 1:20
            a = norm(means(f,:) - pointcolor);
            if(a < currentdist)
                clusterno = f;
                currentdist = a;
            end
        end
        if(clusterno > 10)
            pixelclass(i, j, :) = reshape([0, 255, 0], 3,
[]);
```

```
%               pixelclass(i,j,1) = 0;
%               pixelclass(i,j,2) = 255;
%               pixelclass(i,j,3) = 0;
        end
    end
end



end


%Part 2


function croppedImage = crop(image)

%I crop the image to account for 50 by 50 blocks
croppedImage = image(11:710,16:1265,:);

end

function centers = divfifty(image)
    [row,col,~] = size(image);
    nr = row/50;
    nc = col/50;

    %Find Centroid of each matrix
    centers = zeros(0,5);
    for i=25:50:row
        for j = 25:50:col
            centers = cat(1,centers,[double(i) double(j)
double(image(i,j,1)) double(image(i,j,2))
double(image(i,j,3))]);
        end
    end
end

function gradientRGB = gradientRGB(image)
    [row,col,~] = size(image);
    gradientRGB = zeros(row,col);
    gradientR = magnitude(image(:,:,1),1);
    gradientG = magnitude(image(:,:,2),1);
    gradientB = magnitude(image(:,:,3),1);
```

```
    for i = 1:row
        for j = 1:col
            gradientRGB(i,j) = sqrt(gradientR(i,j)^2 +
gradientG(i,j)^2 + gradientB(i,j)^2);
        end
    end

end

function  newcentroids =
localCenters(oldcenters,gradientRGB,image)
    newcentroids = zeros(0,5);
    [row,~] = size(oldcenters);
    for i = 1:row

        minval = 1000000;
        cr = oldcenters(i,1);
        cc = oldcenters(i,2);
        nr = 0;
        nc = 0;
        for f = cr-1 : cr+1
            for t = cc-1 : cc+1
                if(gradientRGB(f,t) < minval)
                    minval = gradientRGB(f,t);
                    nr = f;
                    nc = t;
                end
            end
        end
        newcentroids = cat(1,newcentroids,[double(nr)
double(nc) double(image(nr,nc,1)) double(image(nr,nc,2))
double(image(nr,nc,3))]);

    end

end

function [clustered,averages] =
centroidUpdate(centroids,image,scalingfactor)

    [row,col,~] = size(image);
    [cs,~] = size(centroids);
    clustered = zeros(row,col);
    averages = zeros(cs,5);
```

```
    averageamount = zeros(cs,1);
    for i = 1:row
        for j = 1:col
                clusterno = 1;
                currentdist = 1000000000;
                pointvector = [i/scalingfactor
j/scalingfactor RGBat(i,j,image)];
                for f = 1:cs
                    if( sqrt((centroids(f,1) - i)^2 +
(centroids(f,2) - j)^2) < 100)
                        newcenter =
[centroids(f,1)/scalingfactor centroids(f,2)/scalingfactor
centroids(f,3:5)];
                        a = norm(newcenter - pointvector);
                        if(a < currentdist)
                            clusterno = f;
                            currentdist = a;
                        end
                        clustered(i,j) = clusterno;
                        averages(clusterno,3) =
averages(clusterno,3) + pointvector(3);
                        averages(clusterno,4) =
averages(clusterno,4) + pointvector(4);
                        averages(clusterno,5) =
averages(clusterno,5) + pointvector(5);
                        averageamount(clusterno) =
averageamount(clusterno) + 1;
                    end
                end
        end
    end
        for t = 1:cs
                averages(t,1) = centroids(t,1);
                averages(t,2) = centroids(t,2);
                averages(t,3) = averages(t,3) /
averageamount(t);
                averages(t,4) = averages(t,4) /
averageamount(t);
                averages(t,5) = averages(t,5) /
averageamount(t);
        end
end

function SLIC = SLICalgo(croppedImage,maxiter)
```

```
    i = 0;


    centers = divfifty(croppedImage);
    [row,~] = size(centers);
    gradienttest = gradientRGB(croppedImage);

    newcenters =
localCenters(centers,gradienttest,croppedImage);
    oldvalues = ones(row,5) * 1000000;
    while(i < maxiter)
        if(i ~=0)
            newcenters =
localCenters(newcenters,gradienttest,croppedImage);
        end
        [clustered,averages] =
centroidUpdate(newcenters,croppedImage,.8);
        newcenters = averages;
        i = i+1;
        if((abs(max(newcenters-oldvalues,[],'all')) < 3))
            break;
        end

    end

    SLIC = showSLIC(clustered,averages,croppedImage);

end


function showSLIC = showSLIC(clustered,averages,image)
    [row,col] = size(clustered);
    showSLIC = image;
    for i = 2:row-1
        for j=2:col-1
            showSLIC(i,j,1) = averages(clustered(i,j),3);
            showSLIC(i,j,2) = averages(clustered(i,j),4);
            showSLIC(i,j,3) = averages(clustered(i,j),5);

            for f = i-1 : i+1
                 for t = j-1 : j+1
                     if(clustered(f,t) ~= clustered(i,j))
                        showSLIC(i,j,1) = 0;
```

```
                            showSLIC(i,j,2) = 0;
                            showSLIC(i,j,3) = 0;
                            break;
                        end
                    end
                end

            end
        end


end








%Stuff from Previous Assignments
%Pads Matrix
function padded_array = pad(image,sigma)
    %original rows, original colums
    [or,oc] = size(image);
    new_sigma = 5*sigma;
    m = zeros(or+(2*new_sigma), oc+(2*new_sigma),
"double");

m(new_sigma+1:or+new_sigma,new_sigma+1:oc+new_sigma)=image;

    %Top Left Corner
    for r = 1:new_sigma+1
        for c = 1:new_sigma+1
            m(r,c) = m(new_sigma+1,new_sigma+1);
       end
    end
    %Left side
    for r = new_sigma+1:or+new_sigma-1
        for c = 1:new_sigma+1
            m(r,c) = m(r,new_sigma+1);
        end
    end
     %Bottom Left Corner
     for r = or+new_sigma:or+(2*new_sigma)
```

```
        for c = 1:new_sigma+1
            m(r,c) = m(or+new_sigma,new_sigma+1);
        end
    end
     %Top
    for r = 1:new_sigma
         for c = new_sigma+1:oc+new_sigma
             m(r,c) = m(new_sigma+1,c);
         end
     end
     %Top Right Corner
     for r = 1:new_sigma+1
        for c = oc+new_sigma+1:oc+(2*new_sigma)
             m(r,c) = m(new_sigma+1,oc+new_sigma);
        end
     end
     %Right Side
     for r = new_sigma+1:or+new_sigma-1
         for c = oc+new_sigma+1:oc+(2*new_sigma)
             m(r,c) = m(r,oc+new_sigma);
         end
     end
     %Bottom Right Corner
     for r = or+new_sigma:or+(2*new_sigma)
         for c = oc+new_sigma+1:oc+(2*new_sigma)
             m(r,c) = m(or+new_sigma,oc+new_sigma);
        end
     end
     %Bottom
     for r = or+new_sigma:or+(2*new_sigma)
         for c = new_sigma+1:oc+new_sigma
             m(r,c) = m(or+new_sigma,c);
        end
     end
    padded_array=m;

end

%Applies any filter to an image
%Takes in a filter, image, and the extended image
function apply_filter = app(filter,image,ext_image)
    [row,col] = size(image);
    [re,ce] = size(ext_image);
    [~,len] = size(filter);
```

```
    newx = re - row;
    newy = ce - col;
    image2 = zeros(row, col);
    for r = 1:row
        for c = 1:col
            newr = r +((newx)/2);
            newc = c +((newy)/2);
            piece = ext_image((newr)-((len-
1)/2):(newr)+((len-1)/2),(newc)-((len-1)/2):(newc)+((len-
1)/2));
            %mult_matrix = mult_matrices(filter,piece);
            mult_matrix = filter .* piece;
            image2(r,c) = sum(mult_matrix(:));
        end
    end
    apply_filter = image2;
end



%Gets the Gaussian Filter
function get_gaussian = gauss(sigma)
   size = 10*sigma;
   filter = zeros(size-1,size-1);
   for r = 1:size-1
        for c = 1:size-1
             x = abs(r - size/2);
             y = abs(c - size/2);
             filter(r,c) = (1/(2*pi*sigma))*exp((-(x^2 +
y^2))/(2*sigma^2));
        end
   end
   sumof = sum(filter(:));
   get_gaussian = filter;
end

%Applies the Gaussian filter
function appgauss = appgauss(image,sigma)
    %Do the Gauss
     gaussian = gauss(sigma);
     padded_matrix = pad(image,sigma);
     appgauss = app(gaussian,image,padded_matrix);
end
```

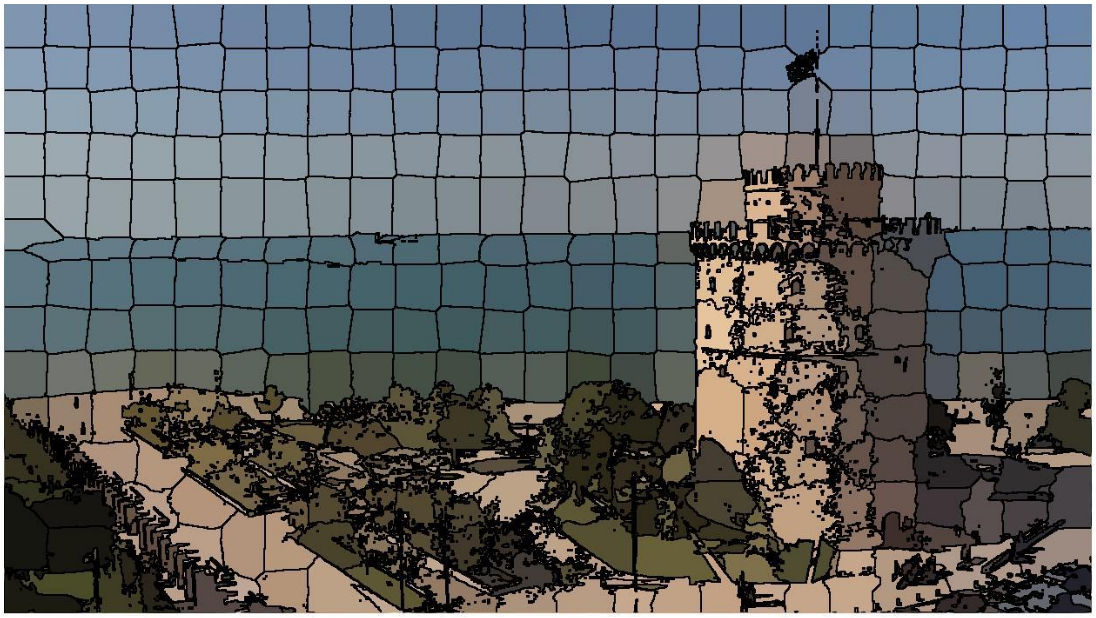
```
%Applies the Vertical Sobel Filter
function sob1 = sobel1(gaussimage)
     single_pad = pad(gaussimage,1);
     sobelfilt1 = [-1 0 1; -2 0 2; -1 0 1];
     sob1 = app(sobelfilt1,gaussimage,single_pad);
     sob1 = double(sob1);

end

%Applies the Horizontal Sobel Filter
function sob2 = sobel2(gaussimage)
     single_pad = pad(gaussimage,1);
     sobelfilt2 = [1 2 1; 0 0 0; -1 -2 -1];
     sob2 = app(sobelfilt2,gaussimage,single_pad);
     sob2 = double(sob2);
end

 function magnitude = magnitude(image,sigma)
     %Apply gauss on image, apply both sobel on gauss
image, calc gradient
     %magnitude for each using formula, under certain
threshhold get rid of
     %pixels
     [row,col] = size(image);

     %Do the Gauss
     gaussimage = appgauss(image,sigma);

     %App both Sobs
     sob1 = sobel1(gaussimage);
     sob2 = sobel2(gaussimage);
     image2 = gaussimage;

     %Gradient Matrix
     for r = 1:row
         for c= 1:col
             image2(r,c) = sqrt(((sob1(r,c))^2) +
((sob2(r,c))^2));
             %Under threshold remove
%              if(image2(r,c)<threshold)
%                 image2(r,c) = 0;
%              end
         end
```

```
    end
    magnitude = image2;
end
```

Output Images

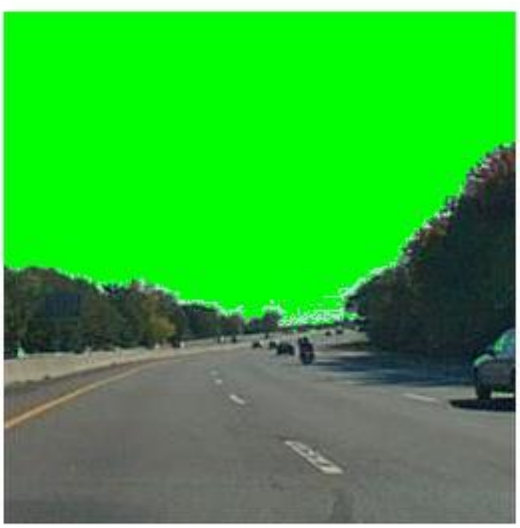
Kmeans output (k=10)

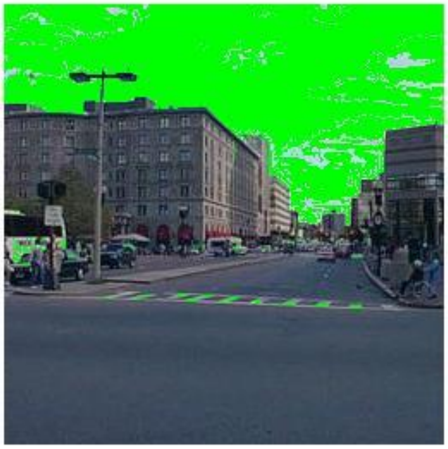
SLIC output on whitetower (maxiter =3)



SLIC output on wt_slc (maxiter =3)

Pixel Classification outputs





Implementation

This assignment was divided into 3 parts, finding the kmeans where k=10, finding SLIC, and doing pixel classification. I will talk about each part individually, but part 1 was heavily used in part 3.

Part 1 Kmeans

Our assignment for part 1 was to find the kmeans on the white tower image with a k = 10. In order to do this I implemented the kmeans algorithm which does the following. It generated 10 random points and the color values associated with them to use as initial seeds for the k initial clusters. I then group every pixel in the image and generate a matrix with the clusters they belong to, as well as the average color of those clusters. An important thing to note is that I never keep track of all of the points per cluster as this is unnecessary. I then use those averages as the new clusters and keep running the cluster generator function until the average values of the cluster values stop changing. I then return the clustered matrix and the new average colors for those clusters, finishing the kmeans algorithm. To display this I make a showclusters function that loops through the clustered function and assigns the color associated with the cluster that point belongs to.

Part 2 SLIC

SLIC was an algorithm that was divided into six parts, which I accordingly split my code into. Firstly, I initialized a centroid in the center of every 50 by 50 block in the image. An important thing to note is that I crop the image if it does not divide evenly into 50. You can do this in other ways but I decided this would be the most simple and the affect on the output would be negligible. After, I did the local shift of every centroid based on the gradient that I calculated for RGB. After this I checked which pixels belonged to each centroid, again keeping track of this in a matrix, and then used the averages I received to recompute the centroids. I did include the optimization by only checking centroids that are 100 pixels away from the pixel. I update the centroids until I either get the max iterations (in this case 3) or if the centroid values stop changing (convergence). I then return the clustered matrix and in order to display it I run through that matrix doing what I did in part 1 but also checking if the surrounding pixels of each pixel is from a different cluster, and making those pixels black.

Part 3 Pixel Classification

Pixel Classification starts with creating the mask from the training image (sky_train). I did this using gimp and used a red color. I then classified each pixel into two data sets, sky-pixels and non-sky pixels by checking whether each pixel was red or not. After getting those two data sets I compute kmeans on each data set with k=10 (used function from part 1) and then get 10 sky clusters and 10 non-sky clusters. After this I run a double for loop on the image I am getting rid of the sky in (test images) and check each pixel and which cluster its closest to. If this cluster is a sky cluster than I change it to green, if not then I leave it as is. An important note is that I used the gaussian filter on the images as I felt this yielded better resulrs.