clear;

%Problem Number 1

tower = imread( 'white-tower', 'png' );

imshow( tower )

%k-means segmentation

k = 10

clusters = random\_centers( tower, k );

%Change the image into an array of RGBPoints

W = size(tower, 2);

H = size(tower, 1);

points(H, W) = RGBPoint( tower(H,W,1), tower(H,W,2), tower(H,W,3) );

for y = 1:H

for x = 1:W

points(y, x) = RGBPoint( tower(y,x,1), tower(y,x,2), tower(y,x,3) );

end

end

%Clean up the environment

clear tower;

%Begin the while loop

iterate = true;

while( iterate )

%Find out which cluster each point is in and add that point to the working

%average of that cluster

cluster\_amount = zeros(k, 1);

cluster\_average = zeros(k, 3);

for y = 1:H

for x = 1:W

points(y, x) = points(y, x).find\_cluster( clusters );

c = points(y, x).getcluster();

cluster\_average(c, 1) = (cluster\_average(c, 1) \* cluster\_amount(c) + points(y,x).getr()) / (cluster\_amount(c) + 1);

cluster\_average(c, 2) = (cluster\_average(c, 2) \* cluster\_amount(c) + points(y,x).getg()) / (cluster\_amount(c) + 1);

cluster\_average(c, 3) = (cluster\_average(c, 3) \* cluster\_amount(c) + points(y,x).getb()) / (cluster\_amount(c) + 1);

cluster\_amount( c ) = cluster\_amount(c) + 1;

end

end

%Create usable points out of the averages of each cluster

new\_clusters(k) = RGBPoint();

for i = 1:k

new\_clusters(i) = RGBPoint( round( cluster\_average(i, 1)), round( cluster\_average(i, 2)), round( cluster\_average(i, 3)) );

end

thresh = 150; %If RGB-Distance between two points are below this threshold, we consider them the same point.

[iterate, distance\_array] = same\_clusters( clusters, new\_clusters, thresh );

iterate = ~iterate;

disp( "Distances between SSD of old clusters averages and new ones" );

disp( distance\_array );

clusters = new\_clusters;

end %End of while loop

clear distance\_array cluster\_amount cluster\_average i iterate new\_clusters thresh

disp( "These are the resulting clusters of the k-means algorithm." )

for i = 1:k

clusters(i).show()

end

image = zeros( H, W, 3, 'uint8');

for y = 1:H

for x = 1:W

c = points(y, x).getcluster();

image(y, x, 1) = uint8(clusters(c).getr());

image(y, x, 2) = uint8(clusters(c).getg());

image(y, x, 3) = uint8(clusters(c).getb());

end

end

imshow( image );

clear k clusters points c clusters H i image k points W x y

%%This is problem 2

clear

SLIC = imread( 'wt\_slic', 'png' );

imshow( SLIC );

S = 50; %The distance between the beginning superpixel groups

W = size(SLIC, 2);

H = size(SLIC, 1);

%Change the image into the point form we used before

points(W, H) = RGBPoint( SLIC(H,W,1), SLIC(H,W,2), SLIC(H,W,3) );

for y = 1:H

for x = 1:W

points(x, y) = RGBPoint( SLIC(y,x,1), SLIC(y,x,2), SLIC(y,x,3) );

end

end

clear SLIC

%Initialize the first centers of the superpixels to be every 50 pixels

num\_super = ceil((H-S/2)/S) \* ceil((W-S/2)/S)

super\_pixels( num\_super ) = Point();

new\_super\_pixels( num\_super ) = FullPoint();

index = 1;

for y = S/2:S:H

for x = S/2:S:W

super\_pixels(index) = Point(x, y);

index = index + 1;

end

end

clear index

%Check the gradient around each superpixel, and move to the minimum in a

%3x3 area

gradient = zeros(3, 3);

for i = 1:size(super\_pixels, 2)

%gradient 2,2 is where our pixel will be

px = super\_pixels(i).x;

py = super\_pixels(i).y;

for x = -1:+1

for y = -1:+1

if px-x > 1 && px-x < W && py-y > 1 && py-y < H

gradient(x+2, y+2) = ...

points(px-x-1, py).RGB\_distance( points(px-x+1, py) ) + ...

points(px, py-y-1).RGB\_distance( points(px, py-y+1));

end

end

end

%Gradient has been calculated for each point around the superpixel in a

%3x3 window

minx = 1;

miny = 1;

min\_distance = gradient(1, 1);

for x = 1:3

for y = 1:3

if gradient(x, y) < min\_distance

min\_distance = gradient(x, y);

minx = x;

miny = y;

end

end

end

%minx and miny point to the least value in gradient

new\_super\_pixels(i) = points( px+minx-2, py+miny-2).to\_full\_point(px+minx-2, py+miny-2).set\_cluster(i);

end

clear gradient min\_distance minx miny

super\_pixels = new\_super\_pixels;

%All the superpixels are now set

%Run k-means in a 100 by 100 neighboorhood around each superpixel

iterate = true;

while( iterate )

distance = Inf( W, H ); %accessed by a y,x pair. Stores distance to nearest superpixel.

%Initialized to inf because initialliy, it's closest to no superpizel.

for i = 1:num\_super

%We do this for each superpixel

px = super\_pixels(i).x;

py = super\_pixels(i).y;

%We search in a 2S by 2S square for associated pixels

for x = px-S:px+S

for y = py-S:py+S

%Make sure the pixel we're testing is within image

%boundaries

if x > 0 && x <= W && y > 0 && y <= H

dis = super\_pixels(i).full\_distance( points(x, y), x, y );

if dis < distance(x, y)

%Set the cluster and distance to the new center

distance(x, y) = dis;

points(x,y) = points(x,y).set\_cluster(i);

end

end

end

end

end

sup\_amount = zeros(num\_super, 1); %Holds the amount of pixels contained within each superpixel.

sup\_average = zeros(num\_super, 5); %Holds the average of R, G, B, x, y for each superpixel

for y = 1:H

for x = 1:W

%Find the averages of each superpixel so we can move them

c = points(x, y).getcluster();

sup\_average(c, 1) = (sup\_average(c, 1) \* sup\_amount(c) + points(x,y).getr()) / (sup\_amount(c) + 1);

sup\_average(c, 2) = (sup\_average(c, 2) \* sup\_amount(c) + points(x,y).getg()) / (sup\_amount(c) + 1);

sup\_average(c, 3) = (sup\_average(c, 3) \* sup\_amount(c) + points(x,y).getb()) / (sup\_amount(c) + 1);

sup\_average(c, 4) = (sup\_average(c, 4) \* sup\_amount(c) + x) / (sup\_amount(c) + 1);

sup\_average(c, 5) = (sup\_average(c, 5) \* sup\_amount(c) + y) / (sup\_amount(c) + 1);

sup\_amount( c ) = sup\_amount(c) + 1;

end

end

%Create usable data out of the averages of each superpixel

new\_super\_pixels( num\_super ) = FullPoint();

distance\_arr = zeros( num\_super, 1);

for i = 1:num\_super

new\_super\_pixels(i) = FullPoint(...

round( sup\_average(i, 1)),...

round( sup\_average(i, 2)),...

round( sup\_average(i, 3)),...

round( sup\_average(i, 4)),...

round( sup\_average(i, 5)) );

distance\_arr( i ) = new\_super\_pixels(i).distance( super\_pixels(i) );

end

%Determine whether or not to iterate

thresh = 150;

if max( distance\_arr ) < thresh

iterate = false;

end

%Set superpixels for the next iteration

super\_pixels = new\_super\_pixels;

end %End of k-means iteration

clear distance distance\_arr min\_distance new\_super\_pixels sup\_amount sup\_average

%super\_pixels contains the final versions of super\_pixels

%Each pixel in the points array has its cluster set accordingly

%Building the resulting image again

image = zeros( H, W, 3, 'uint8');

for y = 1:H

for x = 1:W

c = points(x, y).getcluster();

image(y, x, 1) = uint8(super\_pixels(c).getr());

image(y, x, 2) = uint8(super\_pixels(c).getg());

image(y, x, 3) = uint8(super\_pixels(c).getb());

end

end

%TODO add the black lines between each cluster boundary

for y = 3:H

for x = 3:W

if points(x,y).getcluster() ~= points(x-1,y).getcluster() ||...

points(x,y).getcluster() ~= points(x,y-1).getcluster()

%If a cluster has changed, make this pixel black

for i = y-1:y

for j = x-1:x

%Set a three by three square to be black

image(i,j, 1) = uint8(0);

image(i,j, 2) = uint8(0);

image(i,j, 3) = uint8(0);

end

end

end

end

end

imshow( image );

disp( "SLIC Algorithm has finished" );