Project 4 Report

# Project Objective

Project 4 will apply 2 clustering techniques, K-means and EM, to segmenting image. Some more requirements with this project are:

* Use random initialization for K-means
* Use K-means as an initialization for EM
* Compare 2 algorithms on their performance and the sensitivity to the initialization
* Plot the objective function vs. the iteration number for both K-means and EM

In this project, Matlab is used as the programming language. Two moisac images are given as the input to segment. And two ground-truth maps are given as the foundation for evaluate the performance.

# Theoretical Background

## Clustering

Clustering is a field which the objective is to segment an image into different regions based on their features. For example,



To cluster or segment an image, the well-known methods usually use the image’s feature space as the input, where each point reflects the feature vector at a pixel. In this project, we introduce 2 common algorithms: K-means (hard clustering) and EM (soft clustering).

## K-means

### K-means formulation

 is a given set of unlabeled data samples in a d-dimensional space. A partition of D, denoted as, is a way to divide D into subsets such that. The problem of clustering is formulated as



In which, *f(.)* is the objective function (criterion, cost function,…)

K-means approach assumes that the number of classes, k, is given. The objective function in K-means is so-called intra-class divergence, which is:



Two basic steps in K-means are:

* Assume the cluster centers are known, and allocate each data point to the closest cluster center.
* Assume the allocation is known, and compute a new set of cluster centers. Each center is the mean of the points allocated to that cluster.

These steps are repeated many times until the intra-class divergence doesn’t change significantly or the centers don’t change.

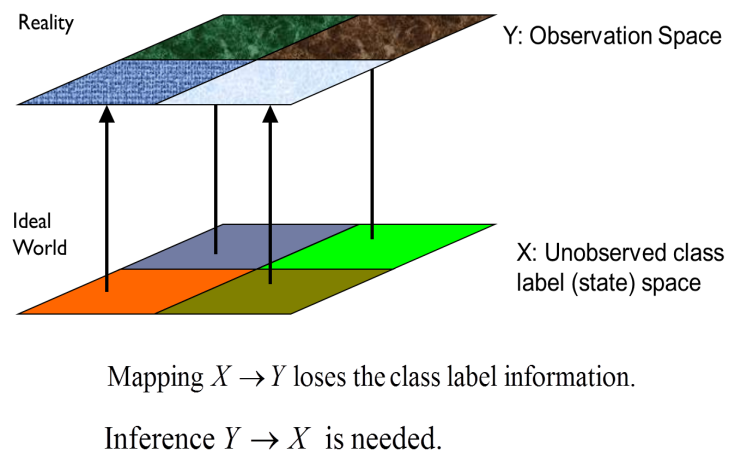
### K-means limitations

* K-mean depends strongly on the initialization. Therefore, it can be trapped into local minimum/maximum.
* K-means does NOT consider the spread of different clusters, structure of each cluster and proportion of different clusters.

## EM

### EM formulation

Before introducing EM, we restate the missing data problem briefly.



In this problem, X is the ideal world (ground-truth data) while Y is the reality world (observed data). The problem is that the class label information is not available in Y and that is the mission of EM algorithm.

Y: Observation Space

X: Unobserved class label (state) space

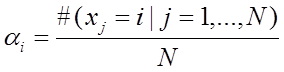


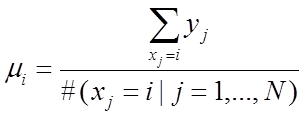
EM solves the missing data problem by using two-steps iteration followed an initialization, as following:

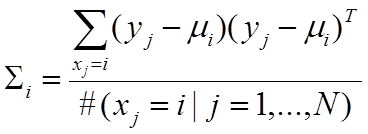
* Initialization: initialize the parameters:



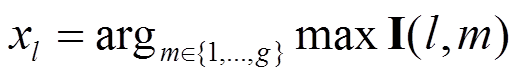
Generally, K-means output is used as the initialization for EM. Assuming that {x1, x2,…, xN} is the result of K-means, EM initialization can be computed as:

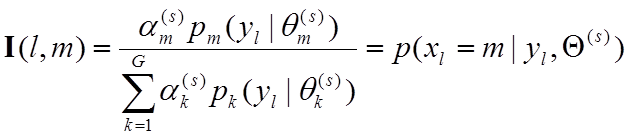




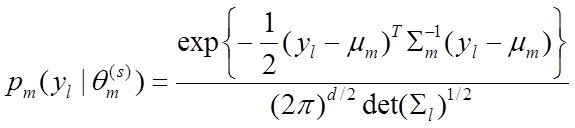


* Step 1 (E-Step): Estimate the missing data in terms of the posterior probability of each data sample:

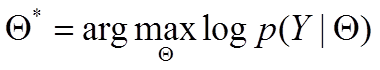


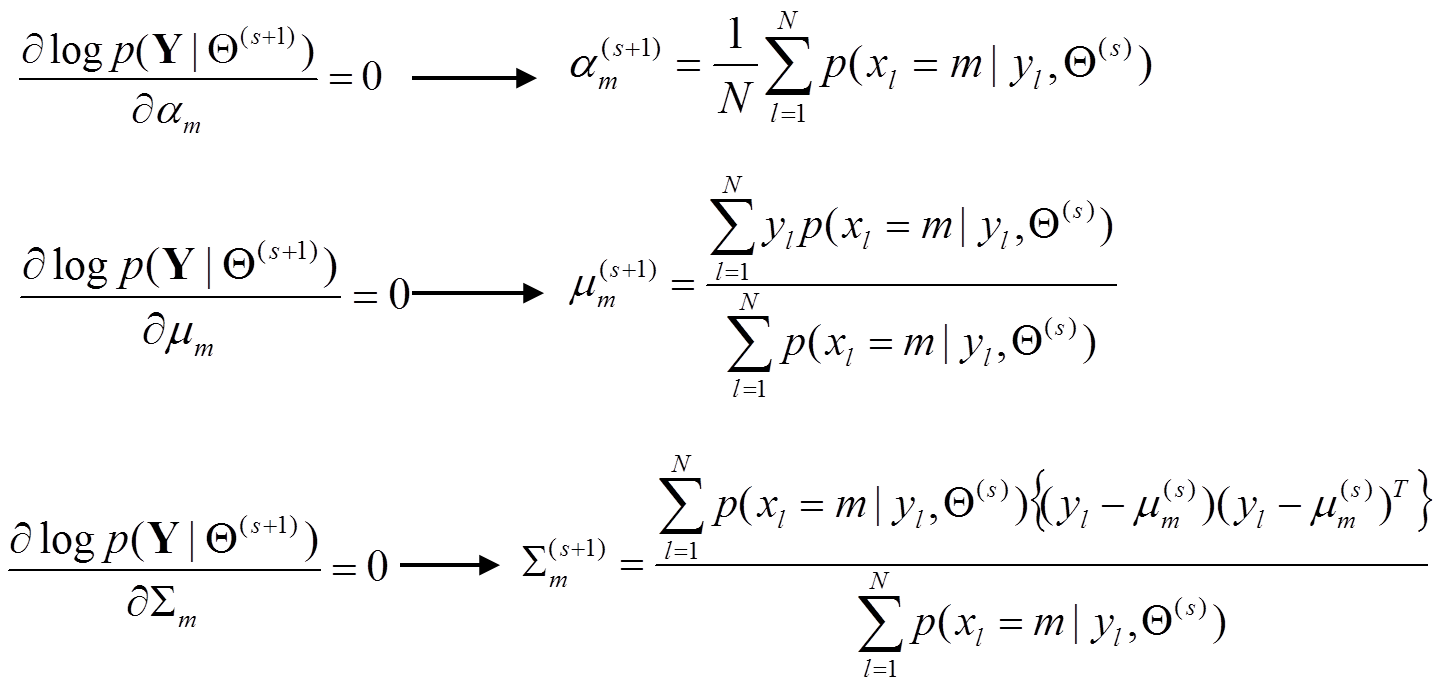


In which,



* Step 2 (M-Step): From the estimated missing data, to obtain the maximum likelihood estimate of the parameters:





# Algorithm and Implementation

## K-means



## EM



# Experimental Results

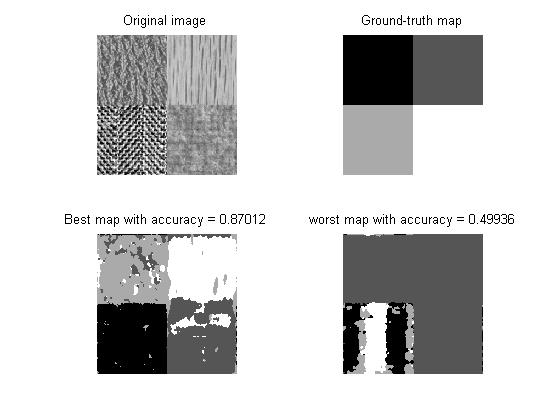
## K-means

In the experiment, we use following parameters:

* **K-means:**
  + **NUM\_INIT = 10; %Number of time doing K-Mean**
  + **MAX\_IT = 20;%Max iteration number per time**
  + **EPS = 0.001;**
  + **init\_type = 'rand';%init\_type: 'rand'; 'furthest', 'plusplus'**
* **Feature Space (Gabor):**
  + **nscale = 4;%Number of scale**
  + **norient = 6;%Number of orientation**
  + **minWaveLength = 3;% Wavelength of smallest scale filter.**
  + **mult = 2;%Scaling factor between successive filters.**
  + **sigmaOnf =0.65;%Ratio of the standard deviation of the Gaussian describing the log Gabor filter's transfer function**
  + **% in the frequency domain to the filter center**
  + **% frequency.**
  + **dThetaOnSigma = 1.5;%Ratio of angular interval between filter orientations**
  + **% and the standard deviation of the angular Gaussian**
  + **% function used to construct filters in the**
  + **% freq. plane.**

Focusing on the core procedure of the K-means algorithm, as well as observing how strongly K-means depends on the initialization, above parameters was fixed through the program.

With this configuration, the program gave following output with input ‘moisacA’ :



The detail of one iteration is shown below:

*K-Means: [02/10]*

*K-Mean Steps...*

*Iteration:[02]---Obj\_Fn:[21818.9427]---Accuracy:[0.75]---Duration:[0.552(s)]*

*Iteration:[03]---Obj\_Fn:[21516.3071]---Accuracy:[0.79]---Duration:[0.560(s)]*

*Iteration:[04]---Obj\_Fn:[21360.2402]---Accuracy:[0.82]---Duration:[0.556(s)]*

*Iteration:[05]---Obj\_Fn:[21275.2313]---Accuracy:[0.84]---Duration:[0.553(s)]*

*Iteration:[06]---Obj\_Fn:[21243.0670]---Accuracy:[0.85]---Duration:[0.561(s)]*

*Iteration:[07]---Obj\_Fn:[21230.8962]---Accuracy:[0.86]---Duration:[0.553(s)]*

*Iteration:[08]---Obj\_Fn:[21226.4663]---Accuracy:[0.86]---Duration:[0.562(s)]*

*Iteration:[09]---Obj\_Fn:[21224.7418]---Accuracy:[0.87]---Duration:[0.562(s)]*

*Iteration:[10]---Obj\_Fn:[21224.0879]---Accuracy:[0.87]---Duration:[0.556(s)]*

*Iteration:[11]---Obj\_Fn:[21223.8678]---Accuracy:[0.87]---Duration:[0.552(s)]*

*Iteration:[12]---Obj\_Fn:[21223.7611]---Accuracy:[0.87]---Duration:[0.561(s)]*

*Iteration:[13]---Obj\_Fn:[21223.7132]---Accuracy:[0.87]---Duration:[0.554(s)]*

*Iteration:[14]---Obj\_Fn:[21223.6959]---Accuracy:[0.87]---Duration:[0.554(s)]*

*Iteration:[15]---Obj\_Fn:[21223.6888]---Accuracy:[0.87]---Duration:[0.570(s)]*

*Iteration:[16]---Obj\_Fn:[21223.6844]---Accuracy:[0.87]---Duration:[0.563(s)]*

*Iteration:[17]---Obj\_Fn:[21223.6830]---Accuracy:[0.87]---Duration:[0.554(s)]*

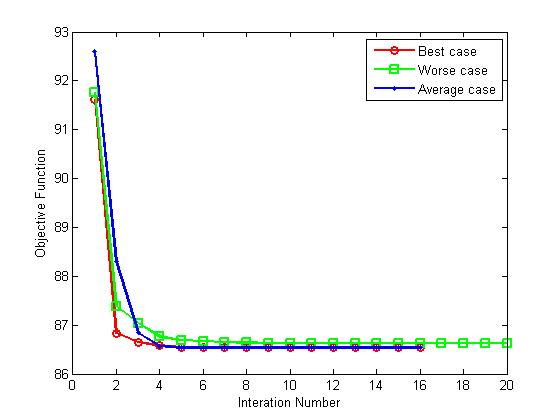
*Iteration:[18]---Obj\_Fn:[21223.6827]---Accuracy:[0.87]---Duration:[0.560(s)]*

*Iteration:[19]---Obj\_Fn:[21223.6826]---Accuracy:[0.87]---Duration:[0.556(s)]*

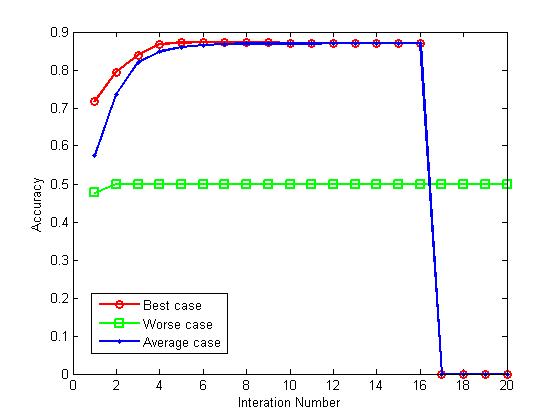
*Iteration:[20]---Obj\_Fn:[21223.6826]---Accuracy:[0.87]---Duration:[0.554(s)]*

*Done K-means with one initialization in 11.144 (s).*

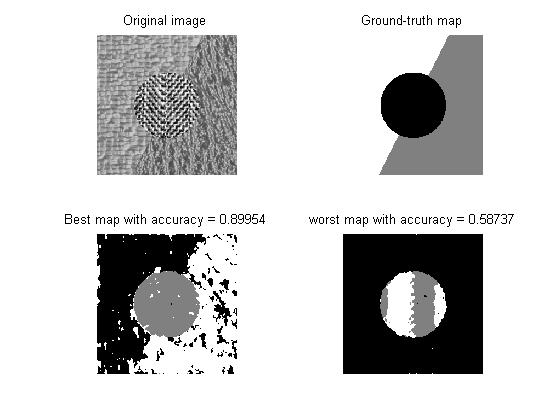
Plot the objective function (20\*log10) vs. iteration number:



And plot the accuracy vs. iteration number:



With moisacB input image, we have following results:



The detail of one iteration is shown below:

*K-Means: [03/10]*

*K-Mean Steps...*

*Iteration:[02]---Obj\_Fn:[26616.4824]---Accuracy:[0.68]---Duration:[0.468(s)]*

*Iteration:[03]---Obj\_Fn:[25141.1003]---Accuracy:[0.65]---Duration:[0.462(s)]*

*Iteration:[04]---Obj\_Fn:[24224.2678]---Accuracy:[0.68]---Duration:[0.470(s)]*

*Iteration:[05]---Obj\_Fn:[22477.5792]---Accuracy:[0.74]---Duration:[0.471(s)]*

*Iteration:[06]---Obj\_Fn:[21749.8440]---Accuracy:[0.79]---Duration:[0.469(s)]*

*Iteration:[07]---Obj\_Fn:[21386.0075]---Accuracy:[0.84]---Duration:[0.470(s)]*

*Iteration:[08]---Obj\_Fn:[21142.7505]---Accuracy:[0.87]---Duration:[0.482(s)]*

*Iteration:[09]---Obj\_Fn:[21060.3864]---Accuracy:[0.89]---Duration:[0.477(s)]*

*Iteration:[10]---Obj\_Fn:[21041.2210]---Accuracy:[0.89]---Duration:[0.490(s)]*

*Iteration:[11]---Obj\_Fn:[21036.8909]---Accuracy:[0.90]---Duration:[0.475(s)]*

*Iteration:[12]---Obj\_Fn:[21035.7873]---Accuracy:[0.90]---Duration:[0.475(s)]*

*Iteration:[13]---Obj\_Fn:[21035.5546]---Accuracy:[0.90]---Duration:[0.478(s)]*

*Iteration:[14]---Obj\_Fn:[21035.4786]---Accuracy:[0.90]---Duration:[0.475(s)]*

*Iteration:[15]---Obj\_Fn:[21035.4610]---Accuracy:[0.90]---Duration:[0.473(s)]*

*Iteration:[16]---Obj\_Fn:[21035.4581]---Accuracy:[0.90]---Duration:[0.482(s)]*

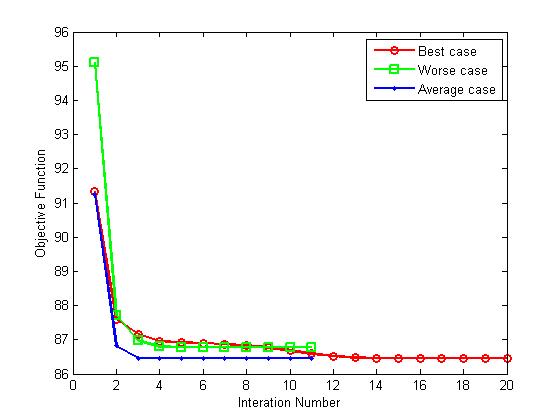
*Iteration:[17]---Obj\_Fn:[21035.4576]---Accuracy:[0.90]---Duration:[0.473(s)]*

*Iteration:[18]---Obj\_Fn:[21035.4570]---Accuracy:[0.90]---Duration:[0.472(s)]*

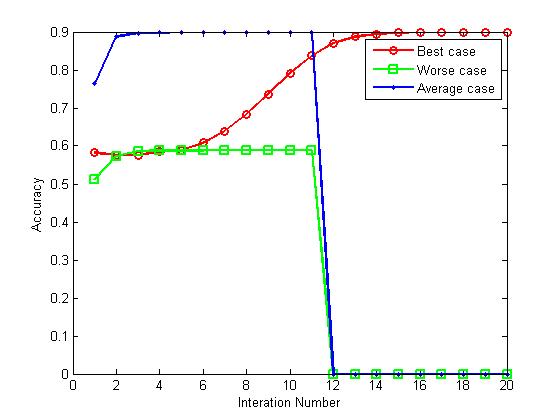
*Iteration:[19]---Obj\_Fn:[21035.4570]---Accuracy:[0.90]---Duration:[0.466(s)]*

*Done K-means with one initialization in 8.982 (s).*

And plot the objective function vs. iteration number of moisacB



Plot the accuracy vs. iteration number of moisac B:

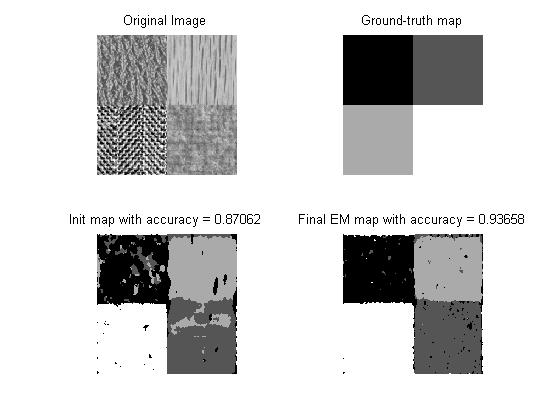


From above results, we can see that:

* K-means depends strongly on the initialization. If we do K-means with a bad initialization, the result is very bad (the green line). Moreover, with bad initialization, the algorithms made a very little improvement of objective function and accuracy in each iteration.
* The objective function (intra-divergence) converged quite quickly with an arbitrary initialization. After 4-6 iterations, this cost didn’t change much.
* The best accuracy this implementation can reach is 0.9 (90% correct).

## EM

In EM, we use the best result of K-means as the initialization. We fixed the maximum number of iteration EM\_MAX\_IT is 10 and threshold to stop iteration is EM\_EPS = 0.000001. The result of EM for image A is shown below:



The detail of EM process is shown below:

*Begin EM process...*

*Initializing...EM params...E-Steps:[4/4]*

*Done Initializing EM.*

*Iteration [01]-icLogLF:[1918484.8061]-cLogLF:[1865327.1310]-Accuracy[0.926]-Duration[8.539(s)]*

*Begin iteration....*

*Iteration [02]...E-Steps:[4/4]-icLogLF:[1963105.8795]-cLogLF:[1952543.5786]-Accuracy[0.942]-Duration[16.030(s)]*

*Iteration [03]...E-Steps:[4/4]-icLogLF:[1973037.7350]-cLogLF:[1970499.9705]-Accuracy[0.943]-Duration[15.675(s)]*

*Iteration [04]...E-Steps:[4/4]-icLogLF:[1974432.0638]-cLogLF:[1973403.5307]-Accuracy[0.940]-Duration[15.679(s)]*

*Iteration [05]...E-Steps:[4/4]-icLogLF:[1974706.5582]-cLogLF:[1973934.6201]-Accuracy[0.939]-Duration[15.525(s)]*

*Iteration [06]...E-Steps:[4/4]-icLogLF:[1974789.8722]-cLogLF:[1974082.3188]-Accuracy[0.938]-Duration[15.571(s)]*

*Iteration [07]...E-Steps:[4/4]-icLogLF:[1974822.1705]-cLogLF:[1974111.9485]-Accuracy[0.937]-Duration[15.561(s)]*

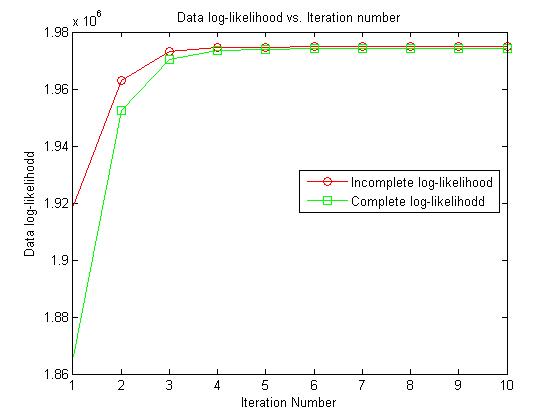
*Iteration [08]...E-Steps:[4/4]-icLogLF:[1974837.4452]-cLogLF:[1974128.7487]-Accuracy[0.937]-Duration[15.451(s)]*

*Iteration [09]...E-Steps:[4/4]-icLogLF:[1974845.4993]-cLogLF:[1974138.9719]-Accuracy[0.937]-Duration[15.334(s)]*

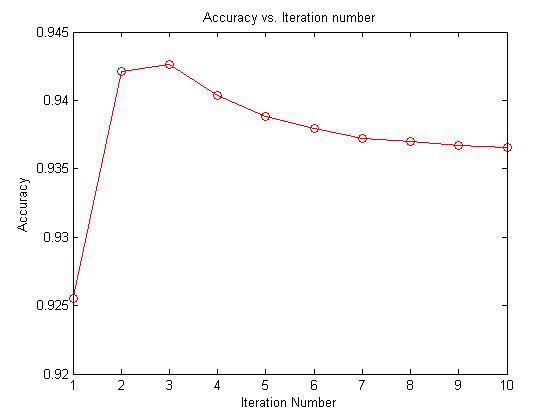
*Iteration [10]...E-Steps:[4/4]-icLogLF:[1974849.9737]-cLogLF:[1974141.1735]-Accuracy[0.937]-Duration[15.420(s)]*

*Done EM in 148.788(s)*

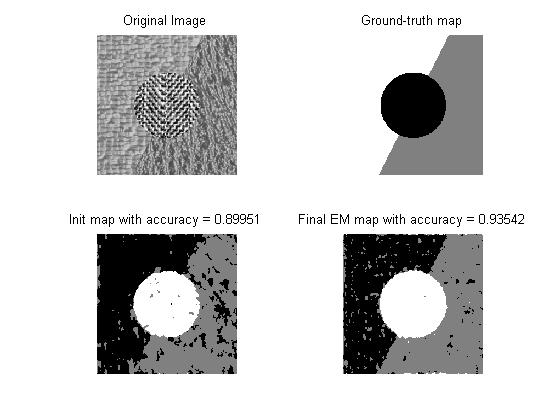
Plot the objective function (in-complete data log-likelikhood, complete data log-likelihodd) vs. iteration number:



Plot the accuracy vs. iteration number:



Experimental result with image B:



The detail of EM process for this image:

*Begin EM process...*

*Initializing...EM params...E-Steps:[3/3]*

*Done Initializing EM.*

*Iteration [01]-icLogLF:[1666351.8231]-cLogLF:[1643223.4472]-Accuracy[0.940]-Duration[6.503(s)]*

*Begin iteration....*

*Iteration [02]...E-Steps:[3/3]-icLogLF:[1680343.7426]-cLogLF:[1675969.6613]-Accuracy[0.945]-Duration[11.993(s)]*

*Iteration [03]...E-Steps:[3/3]-icLogLF:[1682921.5338]-cLogLF:[1681037.2825]-Accuracy[0.943]-Duration[12.109(s)]*

*Iteration [04]...E-Steps:[3/3]-icLogLF:[1683598.0679]-cLogLF:[1682256.6451]-Accuracy[0.941]-Duration[11.886(s)]*

*Iteration [05]...E-Steps:[3/3]-icLogLF:[1683825.0087]-cLogLF:[1682616.1556]-Accuracy[0.939]-Duration[11.890(s)]*

*Iteration [06]...E-Steps:[3/3]-icLogLF:[1683910.5730]-cLogLF:[1682753.8008]-Accuracy[0.938]-Duration[12.104(s)]*

*Iteration [07]...E-Steps:[3/3]-icLogLF:[1683944.9421]-cLogLF:[1682802.6676]-Accuracy[0.937]-Duration[11.620(s)]*

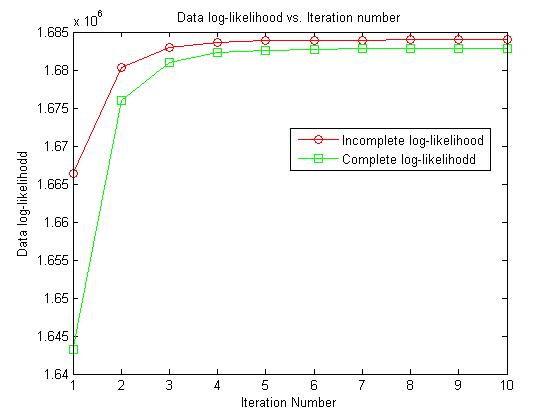
*Iteration [08]...E-Steps:[3/3]-icLogLF:[1683959.2534]-cLogLF:[1682819.6645]-Accuracy[0.936]-Duration[11.707(s)]*

*Iteration [09]...E-Steps:[3/3]-icLogLF:[1683965.3413]-cLogLF:[1682825.2348]-Accuracy[0.936]-Duration[11.636(s)]*

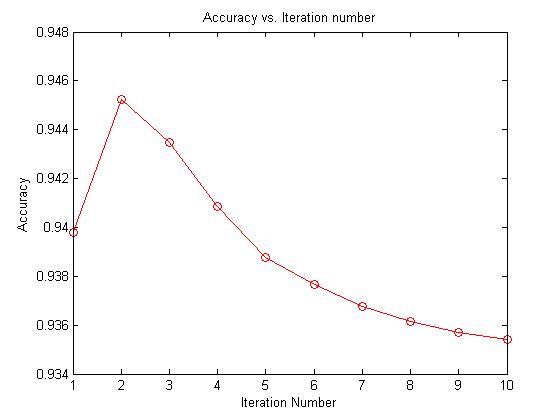
*Iteration [10]...E-Steps:[3/3]-icLogLF:[1683967.9636]-cLogLF:[1682826.7943]-Accuracy[0.935]-Duration[11.656(s)]*

*Done EM in 113.105(s)*

Plot the objective function vs. iteration number:



Plot the accuracy vs. iteration number:



Summary about EM:

* The highest accuracy is 0.94
* Incomplete and complete data log-likelihood converge when the iteration increases

## Comparison

|  |  |  |
| --- | --- | --- |
|  | **K-means** | **EM** |
| Best Accuracy | 0.9 | 0.94 |
| Time/Iteration | 0.5s / iteration | 12s /iteration |
| Complexity | Simpler | More complex |
| Initialization | Random. Very sensitive to initialization. | K-means. Less sensitive than K-means |
| Decision | Hard (0 or 100% belong to one class) | Soft(0<=p<=100 % belong to one class) |

In a word, EM is better than K-means in the accuracy but slower than K-means in running-time. Clearly, EM is better than K-means in terms of accuracy. When K-means stops, (that means it can’t improve the result significantly any more), EM can even improve the best result of K-means.

## Color Image

In above experiments, the image we worked with is grayscale image. In this section, we will try K-means with some color images. To emphasize the color of each pixels, we added RGB features to existing Gabor feature, as following:

**%% Compute normalized feature vectors**

**fv = fv\_space(img);**

**color\_fv = zeros(rows\*cols, 3);**

**for i = 1:rows**

**for j = 1:cols**

**color\_fv((i-1)\*cols + j, 1) = rgb\_img(i,j,1);**

**color\_fv((i-1)\*cols + j, 2) = rgb\_img(i,j,2);**

**color\_fv((i-1)\*cols + j, 3) = rgb\_img(i,j,3);**

**end**

**end**

**min\_color\_fv = min(color\_fv);**

**max\_color\_fv = max(color\_fv);**

**for i = 1:rows\*cols**

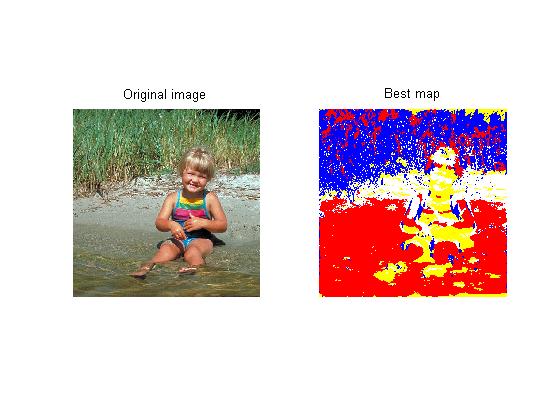
**color\_fv(i,:) = (color\_fv(i,:) - min\_color\_fv)./(max\_color\_fv - min\_color\_fv);**

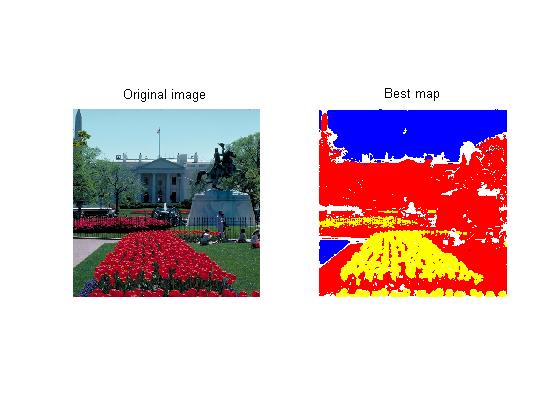
**end**

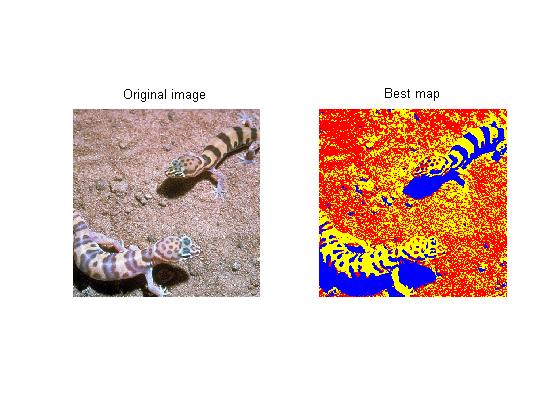
**fv = [color\_fv fv];**

Here are the results:









Via above examples, we can see that by adding color features, clearly that the pixel has the same colors are more likely grouped together. And Gabor feature, as we’ve known, is good at texture grouping.

# Discussion and Conclusion

In this project, I have introduced the basic concepts of clustering and two common algorithms: K-means and EM. Shortly, EM provides the better accuracy (0.94), which 5% higher than K-means (0.9). However, EM is much slower than K-means in term of time per iteration (12 seconds vs. 0.5 seconds).

In the scope of project, to focus on just the core algorithms, focus on how the algorithms are sensitive to initialization (not on benchmark accuracy), I have fixed some parameters that can be not the optimal ones for the accuracy (Gabor parameters, for example). However, it is good enough.

I also realized that accuracy is not always proportional with objective function, both in K-means and EM. Sometimes, the better objective function doesn’t give us the better accuracy. But in this context, K-means accuracy is more proportional with objective function than EM. Besides, the initialization strategy should the key-point to improve K-means as well as EM. In my code, I have implemented not only random initialization for K-means, but also furthest-based and K-mean++ strategies. However, it seems not to show an improvement on the final results.

CODE

**clc;**

**clear all;**

**close all;**

**%% PARAMETERS**

**ROOT = '../imgs/';**

**IMG\_NAME = 'mosaicA.bmp';**

**MAP\_NAME = 'mapA.bmp';**

**NUM\_INIT = 10; %Number of time doing K-Mean**

**MAX\_IT = 20;%Max iteration number per time**

**EPS = 0.0001;**

**K = 4;%Number of class**

**init\_type = 'rand';%init\_type: 'rand'; 'furthest', 'plusplus'**

**%% load image**

**fprintf('Loading image...\n');**

**img = imread([ROOT,IMG\_NAME]);**

**truthImg = imread([ROOT,MAP\_NAME]);**

**[rows cols] = size(img);**

**fprintf('Done loading image.\n');**

**%% Compute normalized feature vectors**

**fv = fv\_space(img);**

**%% K-MEAN STEPS**

**final\_cluster\_vt = cell(NUM\_INIT, 1);**

**final\_psa\_vt = zeros(NUM\_INIT, 1);**

**map\_series\_vt = cell(NUM\_INIT, 1);**

**psa\_series\_vt = cell(NUM\_INIT, 1);**

**objfn\_series\_vt = cell(NUM\_INIT, 1);**

**for t = 1:NUM\_INIT**

**fprintf('K-Means: [%02u/%02u]\n', t, NUM\_INIT);**

**[final\_cluster\_vt{t} final\_psa\_vt(t) map\_series\_vt{t} objfn\_series\_vt{t} psa\_series\_vt{t}] = k\_means(fv, K, MAX\_IT, EPS, truthImg, init\_type);**

**end**

**%% Show best results**

**[pcc\_sort idx\_sort] = sort(final\_psa\_vt, 'descend');**

**cluster\_best = final\_cluster\_vt{idx\_sort(1)};**

**map\_best = reshape(cluster\_best, cols, rows);**

**map\_best = map\_best';**

**cluster\_worse = final\_cluster\_vt{idx\_sort(end)};**

**map\_worse = reshape(cluster\_worse, cols, rows);**

**map\_worse = map\_worse';**

**fprintf('The best accuracy: %1.2f\n', pcc\_sort(1));**

**fprintf('The worse accuracy: %1.2f\n', pcc\_sort(end));**

**figure; subplot(2,2,1);imshow(img,[]);title('Original image');**

**subplot(2,2,2); imshow(truthImg, []);title('Ground-truth map');**

**subplot(2,2,3); imshow(map\_best, []);title(['Best map with accuracy = ', num2str(pcc\_sort(1))]);**

**subplot(2,2,4); imshow(map\_worse, []);title(['worst map with accuracy = ', num2str(pcc\_sort(end))]);**

**% Plot objective function vs. iteration number of best case/worse**

**% case/medidum case**

**figure;**

**plot(20\*log10(objfn\_series\_vt{idx\_sort(1)}), '-ro','LineWidth',2);xlabel('Interation Number');ylabel('Objective Function');hold on;**

**plot(20\*log10(objfn\_series\_vt{idx\_sort(end)}),'-gs','LineWidth',2);xlabel('Interation Number');ylabel('Objective Function');hold on;**

**plot(20\*log10(objfn\_series\_vt{idx\_sort(4)}),'-b.','LineWidth',2);xlabel('Interation Number');ylabel('Objective Function');legend('Best case','Worse case','Average case');**

**%% Plot accuracy vs. iteration number of best case/worse case/m. case**

**figure;**

**plot(psa\_series\_vt{idx\_sort(1)}, '-ro','LineWidth',2);xlabel('Interation Number');ylabel('Accuracy');hold on;**

**plot(psa\_series\_vt{idx\_sort(end)},'-gs','LineWidth',2);xlabel('Interation Number');ylabel('Accuracy');hold on;**

**plot(psa\_series\_vt{idx\_sort(4)},'-b.','LineWidth',2);xlabel('Interation Number');ylabel('Accuracy');legend('Best case','Worse case','Average case');**

**%% Write to video the best case**

**best\_frames = map\_series\_vt{idx\_sort(1)};**

**for i = 1:MAX\_IT**

**if best\_frames{i}**

**frame(:,:,1) = best\_frames{i};**

**frame(:,:,2) = best\_frames{i};**

**frame(:,:,3) = best\_frames{i};**

**frame = floor(255\*frame/4);**

**img\_seg\_movie(i) = im2frame(uint8(frame));**

**else**

**break;**

**end**

**end**

**movie2avi(img\_seg\_movie,'kmean.avi','fps',1, 'compression', 'none');**

**function fv = fv\_space(img)**

**% FV\_SPACE: compute the feature vectors**

**% TuanND**

**% 03/17**

**fprintf('Computing feature vectors...\n');**

**nscale = 4;%Number of scale**

**norient = 6;%Number of orientation**

**minWaveLength = 3;% Wavelength of smallest scale filter.**

**mult = 2;%Scaling factor between successive filters.**

**sigmaOnf =0.65;%Ratio of the standard deviation of the Gaussian describing the log Gabor filter's transfer function**

**% in the frequency domain to the filter center**

**% frequency.**

**dThetaOnSigma = 1.5;%Ratio of angular interval between filter orientations**

**% and the standard deviation of the angular Gaussian**

**% function used to construct filters in the**

**% freq. plane.**

**% Gabor filtered images**

**[rows cols] = size(img);**

**N = rows \* cols;**

**num\_layer = nscale \* norient;**

**layer = gaborconvolve(img, nscale, norient, minWaveLength, mult, sigmaOnf, dThetaOnSigma);**

**%Extract feature vector for each pixel**

**layer = reshape(layer, 1, []);**

**temp = zeros(rows, cols, num\_layer);**

**for k = 1:num\_layer**

**temp(:,:,k) = abs(layer{k});**

**end**

**fv = zeros(N, num\_layer);**

**for i = 1:rows**

**for j = 1:cols**

**fv((i-1)\*cols + j, :) = squeeze(temp(i,j,:));**

**end**

**end**

**min\_fv = min(fv);**

**max\_fv = max(fv);**

**for n = 1:N**

**fv(n,:) = (fv(n,:) - min\_fv)./(max\_fv - min\_fv);**

**end**

**fprintf('Done computing feature vector.\n');**

**end**

**function [cluster psa map\_vt objfn\_vt psa\_vt] = k\_means(fv, k, max\_it, eps, truthImg, init\_type)**

**% KMEAN**

**% TuanND**

**% 03/21**

**km\_startTime = tic;**

**fprintf('K-Mean Steps...\n');**

**[rows cols] = size(truthImg);**

**map\_vt = cell(max\_it, 1);**

**objfn\_vt = zeros(max\_it, 1);**

**psa\_vt = zeros(max\_it, 1);**

**%Step 1:Set Nc =1 (iteration number).**

**it = 1;**

**%Step 2: Choose randomly a set of K means**

**c\_fv = init\_kmeans(k, fv, init\_type);**

**%Step 3:For each vector x, compute for each k=1,2,…,K, and assign x to the**

**%cluster with the nearest distance**

**[cluster objfn\_vt(1)] = k\_classify(fv, c\_fv);**

**map = reshape(cluster, cols, rows);**

**map = map';**

**psa\_vt(1) = accuracy(truthImg, map);**

**map\_vt{1} = map;**

**%Step 4: Loop update center and classify**

**while (it < max\_it)**

**it\_startTime = tic;**

**it = it + 1;**

**%Update center (means)**

**new\_c\_fv = update\_cfv(fv, cluster, c\_fv);**

**%Classify based on new means**

**[new\_cluster objfn\_vt(it)] = k\_classify(fv, new\_c\_fv);**

**%Stop if no significant change on objective function or centers**

**delta\_obj\_fn = abs(objfn\_vt(it) - objfn\_vt(it-1));**

**if (delta\_obj\_fn < eps) %encounter stop condition**

**map = reshape(new\_cluster, cols, rows);**

**map = map';**

**psa\_vt(it) = accuracy(truthImg, map);**

**map\_vt{it} = map;**

**it\_elapsedTime = toc(it\_startTime);**

**fprintf('\tIteration:[%02u]---Obj\_Fn:[%5.4f]---Accuracy:[%02.2f]---Duration:[%3.3f(s)]\n', it, objfn\_vt(it), psa\_vt(it), it\_elapsedTime );**

**break;**

**else**

**c\_fv = new\_c\_fv;**

**map = reshape(new\_cluster, cols, rows);**

**map = map';**

**psa\_vt(it) = accuracy(truthImg, map);**

**map\_vt{it} = map;**

**cluster = new\_cluster;**

**it\_elapsedTime = toc(it\_startTime);**

**fprintf('\tIteration:[%02u]---Obj\_Fn:[%5.4f]---Accuracy:[%02.2f]---Duration:[%3.3f(s)]\n', it, objfn\_vt(it), psa\_vt(it), it\_elapsedTime );**

**end**

**end**

**psa = psa\_vt(it);**

**km\_elapsedTime = toc(km\_startTime);**

**fprintf('Done K-means with one initialization in %3.3f (s).\n', km\_elapsedTime);**

**end**

**function c\_fv = init\_kmeans(K, fv, type)**

**% INIT\_KMEANS: initialize a set of k means**

**% TuanND**

**% 03/17**

**[num\_pixel dim] = size(fv);**

**rng('shuffle');**

**c\_fv = zeros(K, dim);**

**switch type**

**case 'rand'**

**init\_pos = randi(num\_pixel, K, 1);**

**for i = 1:K**

**c\_fv(i,:) = fv(init\_pos(i),:);**

**end**

**case 'furthest'**

**init\_pos = zeros(K,1);**

**init\_pos(1) = randi(num\_pixel, 1);**

**c\_fv(1,:) = fv(init\_pos(1),:);**

**for i = 2:K**

**d2c = dist2center(fv, c\_fv(1:(i-1),:));**

**[dummy idx] = max(max(d2c,[],2));**

**c\_fv(i,:) = fv(idx,:);**

**end**

**case 'plusplus'**

**init\_pos = zeros(K,1);**

**init\_pos(1) = randi(num\_pixel, 1);**

**c\_fv(1,:) = fv(init\_pos(1),:);**

**for i = 2:K**

**d2c = dist2center(fv, c\_fv(1:(i-1),:));**

**d2c = d2c.^2;**

**min\_d2c = min(d2c,[],2);**

**min\_d2c = min\_d2c./sum(min\_d2c);**

**idx = randsample(1:num\_pixel, 1, true, min\_d2c);**

**c\_fv(i,:) = fv(idx,:);**

**end**

**end**

**end**

**function d2c = dist2center(fv, c\_fv)**

**%dist2center: distance between each pixel to K means**

**%TuanND**

**%03/17**

**K = size(c\_fv, 1);**

**num\_pixel= size(fv, 1);**

**d2c = zeros(num\_pixel, K);**

**for i = 1:num\_pixel**

**for k = 1:K**

**d2c(i,k) = norm(fv(i,:) - c\_fv(k,:));**

**end**

**end**

**end**

**function obj\_fn = obj\_func(d2c, cluster)**

**%OBJ\_FUNC: compute objective function value**

**%TuanND**

**%03/17**

**obj\_fn = 0;**

**num\_pixel = length(cluster);**

**for i = 1:num\_pixel**

**k = cluster(i);**

**obj\_fn = obj\_fn + d2c(i,k)^2;**

**end**

**end**

**function [cluster obj\_fn] = k\_classify(fv, c\_fv)**

**%K\_CLASSIFY classify each pixel to the nearest center**

**% TuanND**

**% 03/17**

**num\_pixel = size(fv, 1);**

**cluster = zeros(num\_pixel, 1);**

**d2c = dist2center(fv, c\_fv);%D(i,j,k) is the distance from (i,j) to center k**

**for i = 1:num\_pixel**

**[dij idx] = min(d2c(i,:));**

**cluster(i) = idx;**

**end**

**%Compute objective function**

**obj\_fn = obj\_func(d2c, cluster);**

**end**

**function new\_c\_fv = update\_cfv(fv, cluster, c\_fv)**

**% UPDATE\_CFV : update new means**

**% TuanND**

**% 03/21**

**[K dim] = size(c\_fv);**

**new\_c\_fv = zeros(K, dim);**

**for k = 1:K**

**fv\_ind = fv(cluster == k, :);**

**new\_c\_fv(k,:) = mean(fv\_ind);**

**end**

**end**

**clc;**

**clear all;**

**close all;**

**%% PARAMETERS**

**ROOT = '../imgs/';**

**IMG\_NAME = 'mosaicB.bmp';**

**MAP\_NAME = 'mapB.bmp';**

**NUM\_INIT = 10; %Number of time doing K-Mean**

**MAX\_IT = 20;%Max iteration number per time**

**K\_EPS = 0.00001;**

**K = 3;%Number of class**

**init\_type = 'rand';%init\_type: 'rand'; 'furthest', 'plusplus'**

**EM\_MAX\_IT = 10;**

**EM\_EPS = 0.000001;**

**%% load image**

**fprintf('Loading image...\n');**

**img = imread([ROOT,IMG\_NAME]);**

**truthImg = imread([ROOT,MAP\_NAME]);**

**[rows cols] = size(img);**

**fprintf('Done loading image.\n');**

**%% Compute normalized feature vectors**

**fv = fv\_space(img);**

**%% K-MEAN STEPS**

**final\_cluster\_vt = cell(NUM\_INIT, 1);**

**final\_psa\_vt = zeros(NUM\_INIT, 1);**

**map\_series\_vt = cell(NUM\_INIT, 1);**

**psa\_series\_vt = cell(NUM\_INIT, 1);**

**objfn\_series\_vt = cell(NUM\_INIT, 1);**

**for t = 1:NUM\_INIT**

**fprintf('K-Means: [%02u/%02u]\n', t, NUM\_INIT);**

**[final\_cluster\_vt{t} final\_psa\_vt(t) map\_series\_vt{t} objfn\_series\_vt{t} psa\_series\_vt{t}] = k\_means(fv, K, MAX\_IT, K\_EPS, truthImg, init\_type);**

**end**

**fprintf('Done K-Means as Init. for EM\n');**

**%% Init for EM**

**fprintf('Begin EM process...\n');**

**fprintf('Initializing...');**

**emStartTime = tic;**

**% init variables**

**alpha = cell(EM\_MAX\_IT,1);**

**nuy = cell(EM\_MAX\_IT, 1);**

**sigma = cell(EM\_MAX\_IT, 1);**

**I = cell(EM\_MAX\_IT, 1);**

**em\_map\_vt = cell(EM\_MAX\_IT, 1);**

**icLogLF = zeros(EM\_MAX\_IT, 1);**

**cLogLF = zeros(EM\_MAX\_IT, 1);**

**psa\_vt = zeros(EM\_MAX\_IT, 1);**

**it = 1;**

**[pcc\_best idx\_best] = max(final\_psa\_vt);**

**k\_cluster\_best = final\_cluster\_vt{idx\_best};**

**k\_map\_best = reshape(k\_cluster\_best, cols, rows);**

**k\_map\_best = k\_map\_best';**

**% xik = map\_label(truthImg, k\_map\_best, K);**

**[alpha{1} nuy{1} sigma{1}] = init\_em(fv, k\_cluster\_best, K);**

**fprintf('EM params...');**

**% E-STEP**

**[I{1} em\_map\_vt{1} icLogLF(1) cLogLF(1) psa\_vt(1)] = em\_e\_step(fv, K, alpha{1}, nuy{1}, sigma{1}, truthImg, k\_map\_best);**

**itElapsedTime = toc(emStartTime);**

**fprintf('\nDone Initializing EM.\n');**

**fprintf('Iteration [%02u]-icLogLF:[%4.4f]-cLogLF:[%4.4f]-Accuracy[%2.3f]-Duration[%3.3f(s)]\n', it, icLogLF(it), cLogLF(it), psa\_vt(it), itElapsedTime);**

**fprintf('Begin iteration....\n');**

**while (it < EM\_MAX\_IT)**

**itStartTime = tic;**

**it = it + 1;**

**fprintf('Iteration [%02u]...', it);**

**%M-STEP**

**[alpha{it} nuy{it} sigma{it}] = em\_m\_step(fv, I{it-1}, nuy{it-1});**

**%E-STEP**

**[I{it} em\_map\_vt{it} icLogLF(it) cLogLF(it) psa\_vt(it)] = em\_e\_step(fv, K, alpha{it}, nuy{it}, sigma{it}, truthImg, em\_map\_vt{it-1});**

**delta\_icLogLF = abs(icLogLF(it) - icLogLF(it-1));**

**itElapsedTime = toc(itStartTime);**

**fprintf('-icLogLF:[%4.4f]-cLogLF:[%4.4f]-Accuracy[%2.3f]-Duration[%3.3f(s)]\n', icLogLF(it), cLogLF(it), psa\_vt(it), itElapsedTime);**

**if delta\_icLogLF < K\_EPS**

**break;**

**end**

**end**

**emElapsedTime = toc(emStartTime);**

**fprintf('Done EM in %3.3f(s)\n', emElapsedTime);**

**%% Show the final result**

**close all;**

**fprintf('Showing the result...\n');**

**fprintf('The accuray of the final map: %4.4f', psa\_vt(it));**

**figure;**

**subplot(2,2,1); imshow(img, []);title('Original Image');**

**subplot(2,2,2); imshow(truthImg, []); title('Ground-truth map');**

**subplot(2,2,3); imshow(k\_map\_best, []); title(['Init map with accuracy = ', num2str(pcc\_best)]);**

**subplot(2,2,4); imshow(em\_map\_vt{it}, []); title(['Final EM map with accuracy = ', num2str(psa\_vt(it))]);**

**%plot objective function vs. iteration**

**figure;**

**plot(1:it, icLogLF, '-ro', 1:it, cLogLF, '-gs');xlabel('Iteration Number'); ylabel('Data log-likelihodd');legend('Incomplete log-likelihood', 'Complete log-likelihodd');**

**title('Data log-likelihood vs. Iteration number');**

**%plot accuracy vs. iteration**

**figure;**

**plot(1:it, psa\_vt, '-ro');xlabel('Iteration Number'); ylabel('Accuracy');**

**title('Accuracy vs. Iteration number');**

**%% create video**

**for i = 1:it**

**frame(:,:,1) = em\_map\_vt{i};**

**frame(:,:,2) = em\_map\_vt{i};**

**frame(:,:,3) = em\_map\_vt{i};**

**frame = floor(255\*frame/4);**

**em\_mv(i) = im2frame(uint8(frame));**

**end**

**movie2avi(em\_mv,'emB.avi','fps',1, 'compression', 'none');**

**function [alpha nuy sigma] = init\_em(fv, cluster, K)**

**% INIT\_EM: init for EM**

**% TuanND**

**% 03/23**

**[num\_pixel dim] = size(fv);**

**alpha = zeros(K,1);**

**nuy = zeros(K, dim);**

**sigma = cell(K, 1);**

**for k = 1:K**

**k\_index = find(cluster==k);**

**size\_group\_k = length(k\_index);**

**alpha(k) = size\_group\_k/num\_pixel;**

**k\_fv = fv(k\_index,:);**

**nuy(k,:) = mean(k\_fv);**

**temp\_sigma = 0;**

**for j = 1:size\_group\_k**

**diff = k\_fv(j,:) - nuy(k,:);**

**temp\_sigma = temp\_sigma + diff'\*diff;**

**end**

**sigma{k} = temp\_sigma/size\_group\_k;**

**end**

**end**

**function [new\_alpha new\_nuy new\_sigma] = em\_m\_step(fv, I, nuy)**

**% EM\_M\_STEP: m-step in EM**

**% TuanND**

**% 03/26**

**[num\_pixel num\_class] = size(I);**

**dim = size(fv, 2);**

**new\_alpha = zeros(num\_class, 1);**

**new\_nuy = zeros(num\_class,dim);**

**new\_sigma = cell(num\_class, 1);**

**for m = 1:num\_class**

**sumIk = sum(I(:,m));**

**new\_alpha(m) = sumIk/num\_pixel;**

**ts = zeros(dim,dim);**

**tn =zeros(1, dim);**

**for l = 1:num\_pixel**

**tn = tn + fv(l,:) \* I(l,m);**

**ts = ts + I(l,m) \* (fv(l,:)-nuy(m,:))' \* (fv(l,:)-nuy(m,:));**

**end**

**new\_nuy(m,:) = tn/sumIk;**

**new\_sigma{m} = ts/sumIk;**

**end**

**end**

**function [I em\_map icLogLF cLogLF psa] = em\_e\_step(fv, K, alpha, nuy, sigma, truthImg, cur\_map)**

**% EM\_E\_STEP: do e\_step in EM**

**% TuanND**

**% 03/23**

**fprintf('E-Steps:[4/4]');**

**[num\_pixel dim] = size(fv);**

**[rows cols] = size(truthImg);**

**xik = zeros(num\_pixel, K);**

**for i = 1:rows**

**for j = 1:cols**

**k = cur\_map(i,j);**

**xik((i-1)\*cols + j, k) = 1;**

**end**

**end**

**I = zeros(num\_pixel, K);**

**pLF = zeros(num\_pixel, K);**

**for m = 1:K**

**fprintf('\b\b\b\b\b');**

**fprintf('[%1u/%1u]',m, K);**

**sk = sigma{m};**

**det\_sigma = det(sk);**

**denominator = ((2\*pi)^(dim/2)) \* sqrt(det\_sigma);**

**for l = 1:num\_pixel**

**dif = fv(l,:) - nuy(m,:);**

**pLF(l,m) = exp(-(1/2) \* dif / sk \* dif')/denominator;**

**end**

**end**

**icLogLF = 0;**

**cLogLF = 0;**

**for l = 1:num\_pixel**

**norm\_denominator = sum(pLF(l,:).\*alpha');**

**icLogLF = icLogLF + log(norm\_denominator);**

**for m = 1:K**

**I(l,m) = alpha(m) \* pLF(l,m)/norm\_denominator;**

**if(alpha(m) \* pLF(l,m) ~= 0)**

**cLogLF = cLogLF + xik(l,m) \* log(alpha(m) \* pLF(l,m));**

**end**

**end**

**end**

**[max\_prob em\_map] = max(I,[],2);**

**em\_map = reshape(em\_map, cols, rows)';**

**psa = accuracy(truthImg, em\_map, K);**

**end**

**clc;**

**clear all;**

**close all;**

**%% PARAMETERS**

**ROOT = '../imgs/';**

**IMG\_NAME = 'geckos.png';**

**MAP\_NAME = 'geckos.png';**

**NUM\_INIT = 1; %Number of time doing K-Mean**

**MAX\_IT = 10;%Max iteration number per time**

**EPS = 0.1;**

**K = 3;%Number of class**

**init\_type = 'rand';%init\_type: 'rand'; 'furthest', 'plusplus'**

**%% load image**

**fprintf('Loading image...\n');**

**rgb\_img = imread([ROOT,IMG\_NAME]);**

**img = rgb2gray(rgb\_img);**

**rgb\_truthImg = imread([ROOT,MAP\_NAME]);**

**truthImg = rgb2gray(rgb\_truthImg);**

**[rows cols] = size(img);**

**fprintf('Done loading image.\n');**

**%% Compute normalized feature vectors**

**fv = fv\_space(img);**

**color\_fv = zeros(rows\*cols, 3);**

**for i = 1:rows**

**for j = 1:cols**

**color\_fv((i-1)\*cols + j, 1) = rgb\_img(i,j,1);**

**color\_fv((i-1)\*cols + j, 2) = rgb\_img(i,j,2);**

**color\_fv((i-1)\*cols + j, 3) = rgb\_img(i,j,3);**

**end**

**end**

**min\_color\_fv = min(color\_fv);**

**max\_color\_fv = max(color\_fv);**

**for i = 1:rows\*cols**

**color\_fv(i,:) = (color\_fv(i,:) - min\_color\_fv)./(max\_color\_fv - min\_color\_fv);**

**end**

**fv = [color\_fv fv];**

**%% K-MEAN STEPS**

**final\_cluster\_vt = cell(NUM\_INIT, 1);**

**final\_psa\_vt = zeros(NUM\_INIT, 1);**

**map\_series\_vt = cell(NUM\_INIT, 1);**

**psa\_series\_vt = cell(NUM\_INIT, 1);**

**objfn\_series\_vt = cell(NUM\_INIT, 1);**

**for t = 1:NUM\_INIT**

**fprintf('K-Means: [%02u/%02u]\n', t, NUM\_INIT);**

**[final\_cluster\_vt{t} final\_psa\_vt(t) map\_series\_vt{t} objfn\_series\_vt{t} psa\_series\_vt{t}] = k\_means(fv, K, MAX\_IT, EPS, truthImg, init\_type);**

**end**

**%% Show best results**

**close all;**

**[pcc\_sort idx\_sort] = sort(final\_psa\_vt, 'descend');**

**cluster\_best = final\_cluster\_vt{idx\_sort(1)};**

**map\_best = reshape(cluster\_best, cols, rows);**

**map\_best = map\_best';**

**c{1} = [1 1 0];**

**c{2} = [1 0 0];**

**c{3} = [0 0 1];**

**c{4} = [1 1 1];**

**c{5} = [0 0 0];**

**rgb\_map\_best = zeros(rows, cols, 3);**

**for i = 1:K**

**[ri ci] = find(map\_best == i);**

**for j = 1:length(ri)**

**rgb\_map\_best(ri(j), ci(j), :) = c{i};**

**end**

**end**

**fprintf('The best accuracy: %1.2f\n', pcc\_sort(1));**

**figure; subplot(1,2,1);imshow(rgb\_img,[]);title('Original image');**

**subplot(1,2,2); imshow(rgb\_map\_best, []);title('Best map');**