

1. We are required to learn the K-means clustering algorithm through this exercise. In this problem we create inputs for the kmeans-clustering algorithm. Each pixel in an image has certain features according to which it could be clustered with others, leading to a segment. We select the pixel's RGB values and the xy co-ordinates to be its set of distinguishing feature, and create a 5-dimensional feature vector from it. The following matlab code extracts the required features from an image matrix and generates an n X 5 feature matrix, where n is the number of pixels in a given image.

```

1  % This function takes a 3D matrix representing the RGB color map and the
2  % pixel's spatial co-ordinates. Returns a feature matrix by constructing a
3  % 5dimensional feature vector for each pixel.
4
5  function featmat = im2featurevec(img)
6  [nrow ncol cmap] = size(img);
7  for icnt = 1:cmap
8      % Separately gather the RGB components
9      colcomp = img(:, :, icnt);
10     colcomp = colcomp(:);
11     fvect(:, icnt) = colcomp;
12 end
13 rowcnt =1;
14 colcnt =1;
15 for icnt = 1:nrow*ncol
16     if(mod(icnt,nrow) ==0)
17         rowcnt = 1;
18         colcnt = colcnt + 1;
19     end
20     fvect(icnt,4) = rowcnt;
21     fvect(icnt,5) = colcnt;
22     rowcnt = rowcnt + 1;
23 end
24 featmat = fvect;
25 return

```

Listing 1: Matlab function to extract features from an image matrix and generate a feature matrix from the extracted features

2. Next we implement the k-means clustering algorithm which takes the feature matrix from the above step as input. The algorithm is also specified with the number of clusters that are to be used to put the pixels into. Initially through random selection we select the cluster centers from all the candidate pixels. After every iterations, the cluster centers are modified to reflect the more 'appropriate' centers of the pixel distribution. The pixels that are closest to the newly modified centers, are assigned to that center and this goes on until the algorithm converges. We restrict the maximum number of iterations the algorithm performs to converge as 5. The following matlab function describes the k-means clustering algorithm.

```

1  % This function implements the K Means clustering algorithm. It takes pts as
2  % input which is a 5 dimensional feature vector. It also takes Nc as the
3  % number of cluster centers to which each points are going to be assigned.
4  % Another argument Ni specifies the number of iterations, it is selected by
5  % the user.
6  function assign = getKMeansCluster(pts, Nc, Ni)
7  [nvect nfeat]= size(pts);
8  % Randomly select cluster centers, this is initial estimation, thus
9  % initassign has randomly allocated Nc indices of the feature vector array
10 initassign = uint32(nvect*rand(1,Nc));
11 % To ensure that the cluster centroids are unique
12 while(1)
13     initassign = uint32(nvect*rand(1,Nc));
14     nclus = unique(initassign);
15     if Nc == length(nclus)
16         break;
17     end
18 end
19 assign = ones(nvect,1);
20 citer = 1;
21
22 % Now we would assign the points to the cluster centroids found from the
23 % above step.

```

```

24 while citer ≤ Ni
25     for icnt =1:nvect
26         S0 = 0;
27         S = 0;
28         for jcnt = 1:Nc
29             % cntr contains a 5 dimensional row vector
30             if citer == 1
31                 cntr = pts(initassign(jcnt),:);
32             else
33                 cntr = newcntr(jcnt, :);
34             end
35
36             X = [cntr', pts(icnt,:)'];
37             % Compute distance measure
38             for kcnt =1:nfeat
39                 S = S + (X(kcnt, 1) - X(kcnt, 2))^2;
40             end
41             S = sqrt(S);
42             if (S0 == 0 | S0 >S)
43                 % This is the minimum distance so far
44                 S0 = S;
45                 % Assign points to this cluster.
46                 assign(icnt) = initassign(jcnt);
47             end
48         end
49     end
50
51     % for each clusters, find their centroids
52     clus = unique(assign);
53     Nc = length(clus);
54     % It could also happen that the centroids merging after certain number
55     % of iterations
56     for icnt =1:Nc
57         m = pts(find(assign == clus(icnt)),:);
58         for jcnt =1:nfeat
59             % Find mean along each dimension of the feature vector
60             newcntr(icnt, jcnt) = mean(m(:, jcnt));
61             % initassign would now contain the updated centroid indices
62             initassign(icnt) = icnt;
63         end
64     end
65     citer = citer + 1;
66 end
67 return

```

Listing 2: Matlab script depicting k-means clustering algorithm

3. Followed by the k-means clustering, we analyse our results by interpreting the clustering assignment. The code written below in Matlab does the analysis. The function described here takes the assignment as an argument alongwith the image resolution. A switch *isboundary* is provided to return the boundary image. The boundary is detected by computing horizontal and vertical gradient of a derivative filter. We use Sobelfilter in this case. After the gradient is obtained, we compute the magnitude of the gradient and threshold it to obtain the segment boundaries.

```

1 % Code to display the image with the cluster number for all the pixels in
2 % the image.
3 function img = processCluster (assign, irow, icol, isboundary)
4     fact = 10;
5     % Defining coefficients for Sobel filter
6     a = [1 2 1];
7     b = [1 0 -1];
8
9     % Horizontal and vertical gradient filter.The gradient function is the
10    % sobel operator
11    fx = 1/4 * a' * b;
12    fy = 1/4 * a * b';
13
14    img = reshape (assign, irow, icol);
15    if isboundary == 1
16        % detect boundaries of each cluster

```

```

17     rx = conv2(img, fx, 'same');
18     ry = conv2(img, fy, 'same');
19     % Obtain magnitude
20     mag = sqrt(rx.^2 + ry.^2);
21     img = uint8(floor(mag));
22
23     % Alternative approach to the gradient function, use the gradient
24     % function
25     %[rx ry] = gradient(img);
26     %mag = sqrt(rx.^2 + ry.^2);
27     %img = uint8(floor(mag));
28
29     img = uint8(255*ones(irow, icol)) - img;
30 else
31     % separate each pixel values by a factor of 10
32     img = uint8(255*ones(irow, icol) - fact.*img);
33 end
34 return

```

Listing 3: Matlab function to process the cluster assignment and return a sophisticated interpretation that could be displayed

- We show the results of the assignment as follows. This is the output of the execution of the k-means algorithm as is on the feature matrix. We have chosen number of clusters = 5.

```

1 % code for testing Program 1 and program 2.
2 clear all; close all; clc;
3 basedir = '/home/subh/courses/cap5415/ps3/';
4 imgdir = 'ps3images/';
5 Nc = 5;
6 Ni = 5;
7
8 files = dir([basedir imgdir '*.jpg']);
9 flidx = find(~[files.isdir]);
10 nImg = length(flidx);
11
12 for icnt = 1:nImg
13     flname = files(flidx(icnt)).name;
14     img = im2double(imread([basedir imgdir flname]));
15     [imrow imcol cmap] = size(img);
16
17     % Results before normalizing the feature vectors
18     % Obtain feature matrix for the given image
19     tic
20     featmat = im2featurevec(img);
21     toc
22     tic
23     assign = getKMeansCluster(featmat, Nc, Ni);
24     toc
25     % Display cluster assignments after final iterations
26     imgclus = processCluster(assign, imrow, imcol, 0);
27     f = flname(1:length(flname)-4);
28     imwrite(imgclus, [basedir imgdir 'clus' f '.png'], 'png');
29     % Only display boundaries
30     imgclusb = processCluster(assign, imrow, imcol, 1);
31     imwrite(imgclus, [basedir imgdir 'clusB' f '.png'], 'png');
32     % Displaying the results
33     figure(2*icnt-1);
34     subplot(1,3,1);imshow(img, []);drawnow;
35     subplot(1,3,2);imshow(imgclus, []);drawnow;
36     subplot(1,3,3);imshow(imgclusb, []);drawnow;
37
38     % Results after normalizing the feature vectors
39     tic
40     featmat = im2featurevecnorm(img);
41     toc
42     tic
43     assign = getKMeansCluster(featmat, Nc, Ni);
44     toc
45     imgclus = processCluster(assign, imrow, imcol, 0);
46

```



Figure 1: The above figures depict how the kmeans clustering algorithm performs on the original image (on the left), Kmeans cluster in the middle and boundary extracted from the k-means cluster (to the right)

```
47 | f = filename(1:length(filename)-4);  
48 | imwrite(imgclus, [basedir 'norm' f '.png'], 'png');  
49 |
```

```

50  imgclusb = processCluster(assign, imrow, imcol, 1);
51  imwrite(imgclus, [basedir 'normB' f '.png'], 'png');
52  figure(2*icnt);
53  subplot(1,3,1);imshow(img, []);drawnow;
54  subplot(1,3,2);imshow(imgclus, []);drawnow;
55  subplot(1,3,3);imshow(imgclusb, []);drawnow;
56
57  % Results with a different set of feature vectors
58  tic
59  featmat = im2filtfeat(img, 5);
60  toc
61  tic
62  assign = getKMeansCluster(featmat, Nc, Ni);
63  toc
64  imgclus = processCluster(assign, imrow, imcol, 0);
65  f = filename(1:length(filename)-4);
66  imwrite(imgclus, [basedir 'filt' f '.png'], 'png');
67  imgclusb = processCluster(assign, imrow, imcol, 1);
68  imwrite(imgclus, [basedir 'filtB' f '.png'], 'png');
69  end

```

Listing 4: Matlab program that collectively tests each functions described in this assignment

5. As we observe that the k-means algorithm does not quite show us the segments that we are looking for. The segmented images from (4) are too far fetched for visual interpretation. This is because the column vectors in the feature matrix vary through a wide range and hence they need to be normalized in order to produce a perceivable output. In order to normalize we make sure that each column vectors representing the feature values have zero mean and unit variance. This is done by subtracting the column mean from each column vector and dividing each column by its standard deviation. The following function does the normalization of the feature vectors.

```

1  function featmat= im2featurevecNorm(img)
2  [nrow ncol cmap] = size(img);
3  for icnt = 1:cmap
4      % Separately gather the RGB components
5      colcomp = img(:, :, icnt);
6      colcomp = colcomp(:);
7      fvect(:, icnt) = colcomp;
8  end
9  rowcnt =1;
10 colcnt =1;
11 for icnt = 1:nrow*ncol
12     if(mod(icnt,nrow) ==0)
13         rowcnt = 1;
14         colcnt = colcnt + 1;
15     end
16     fvect(icnt,4) = rowcnt;
17     fvect(icnt,5) = colcnt;
18     rowcnt = rowcnt + 1;
19 end
20 for icnt =1:5
21     % zero mean
22     fvect(:, icnt) = fvect(:, icnt) - mean(fvect(:, icnt));
23     % Unit variance
24     fvect(:, icnt) = fvect(:, icnt)./std(fvect(:, icnt));
25 end
26 featmat = fvect;
27 return

```

Listing 5: Matlab function to normalize feature vectors

After the features are normalized, we again run the kmeans clustering algorithm and enlist the result as follows. Clearly these results are more discernable and shows how normalization of data is important.

6. Finally, we discuss another methodology of choosing feature vectors as specified in the problem set. The feature vectors are generated by convolving the image pixels with a 3×3 Gaussian filter of different σ values starting from 0.25.

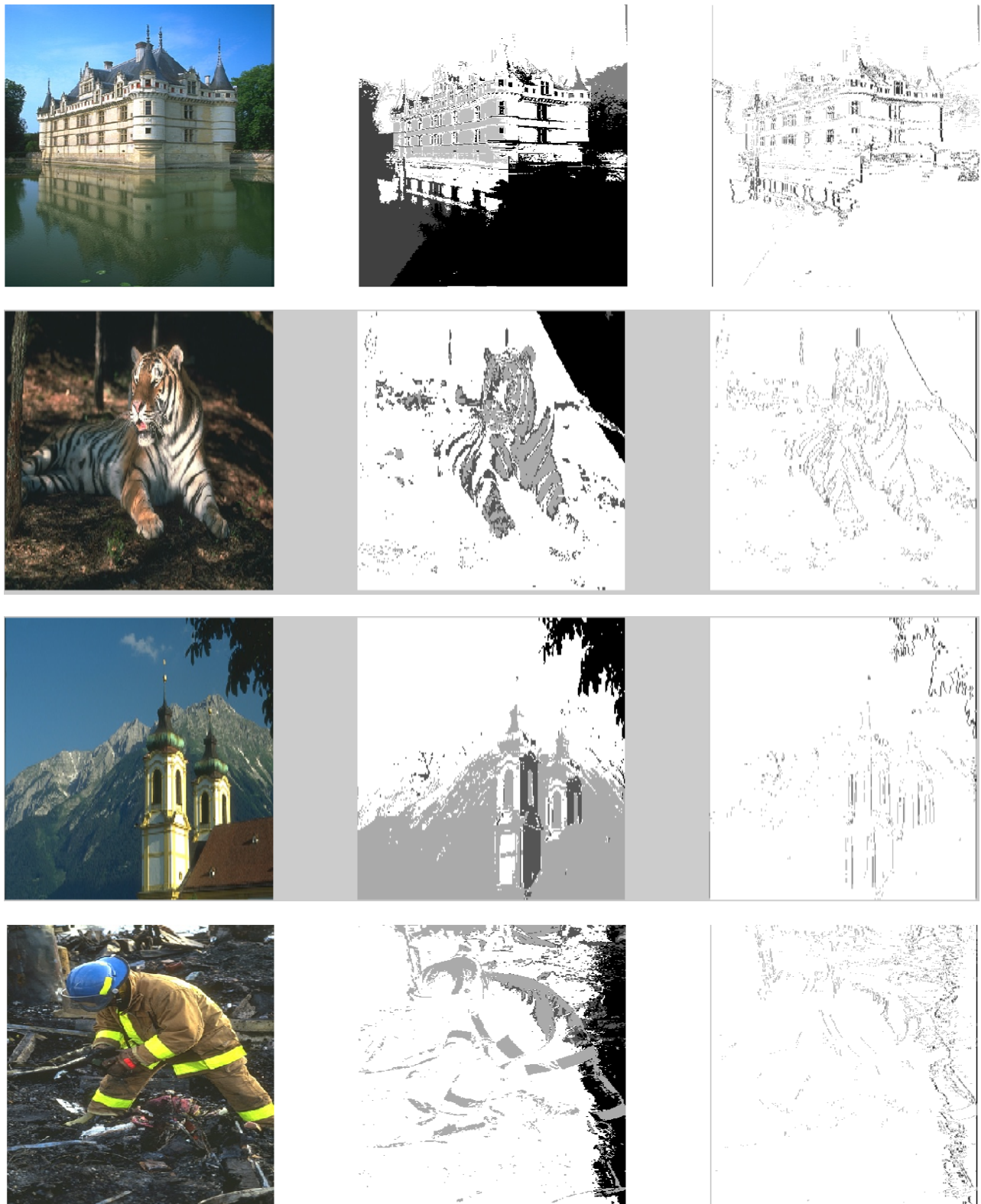


Figure 2: Results of Kmeans clustering with normalized feature vectors: Original image (on the left), Kmeans cluster assignment on image pixels (in the middle) and boundary extracted from the k-means cluster (to the right)

```

1 | % Experimenting with different features. Each pixels RGB values are
2 | % condensed and then the pixel's response to a 3x3 gaussian filters with
3 | % different sigma values are determined to create the feature matrix
    
```

```

4 function fvect= im2filtfeat(img, nfeat)
5 [nrow ncol cmap] = size(img);
6 sum = zeros(nrow*ncol,1);
7 for icnt = 1:cmap
8     % Separately gather the RGB components
9     colcomp = img(:,:,icnt);
10    sum = sum + colcomp(:);
11 end
12 % Calculate responses of each pixel to Gaussian Filters of different
13 % scales
14 for jcnt=1:nfeat
15     f = fspecial('gaussian',[3 3], 0.25*jcnt );
16     fvect(:, jcnt) = conv2(sum, f, 'same');
17 end
18 return
    
```

Listing 6: A different way of creating feature vectors

Here are the results for one such image.



Figure 3: Results of Kmeans clustering with normalized feature vectors, features obtained by using Gaussian filters of different scales : Original image (on the left), Kmeans cluster assignment on image pixels (in the middle) and boundary extracted from the k-means cluster (to the right)

Table 1: This table gives a comparative understanding of the execution time(in seconds) of the various algorithms used in this assignment on 4 input images. FVC stands for Feature Vector Computation and KMC means K-Means Clustering. The average runtime is recorded on a 2Ghz Dual core PC with 2GB of Physical Memory.

Image File Name	FVC(RGB + xy)	KMC	FVC(Normalized RGB + xy)	KMC	FVC(Gaussian filter)	KMC
102061.png	0.080314	151.515172	0.157869	155.305041	0.190910	111.594003
108082.png	0.074756	145.549344	0.135500	128.163298	0.123551	76.301317
126007.png	0.071796	149.203385	0.136364	131.914882	0.133056	89.853058
285079.png	0.071442	148.770938	0.135888	149.255373	0.123480	91.189011