Homework – 6 (CS 577)

Name: Sridhar Mocherla

Note:

a. This assignment was done individually.
b. Some parts of my implementation involving K-Means Clustering are quite slow and results will take a while to appear. Also Part D involving texton map is quite slow.
c. I have also attempted the optional questions Part D and Part E.
d. Code snippets part of the implementation are included in the Appendix at the end.

PART – A:

1. We use the VLFeat MATLAB API to obtain the keypoints and descriptors. We find the Euclidean distance between each pair of keypoints. In selecting the subset of best matches, we obtain the ideal set of matches for a value of **P=20** (20% of total slide keypoints). The length of the lines indicate the scale of the keypoint and red arrow indicates the direction.
   With this value, we obtain the below set of images ordered as per the order specified in the question.

Figure 1: Every 3rd KP for slide and frame 1 for N determined for P=20 (Euclidean)

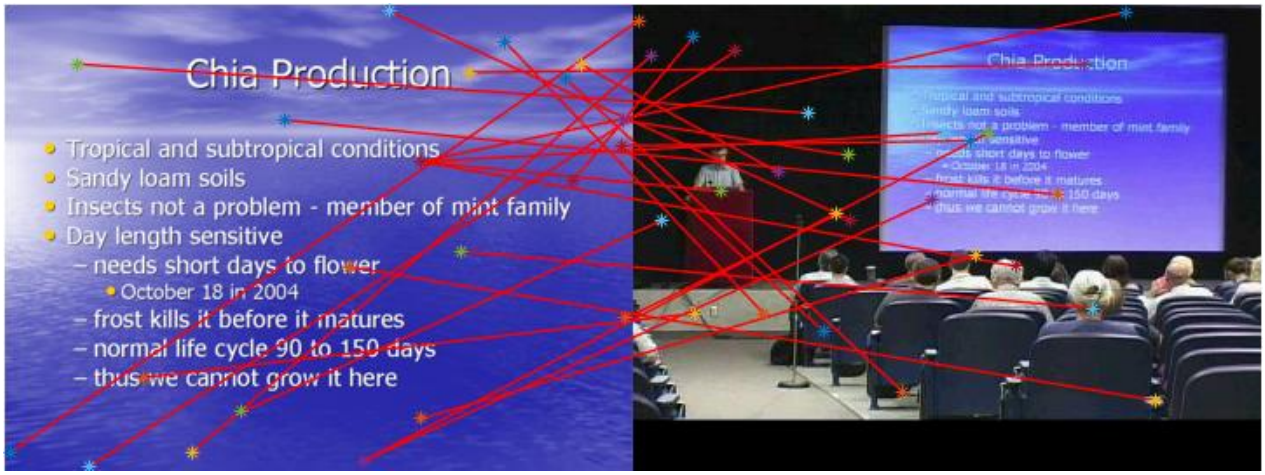


Figure 2: Matching keypoints for slide and frame 1 for P=20 (Euclidean) and for every 3rd pair

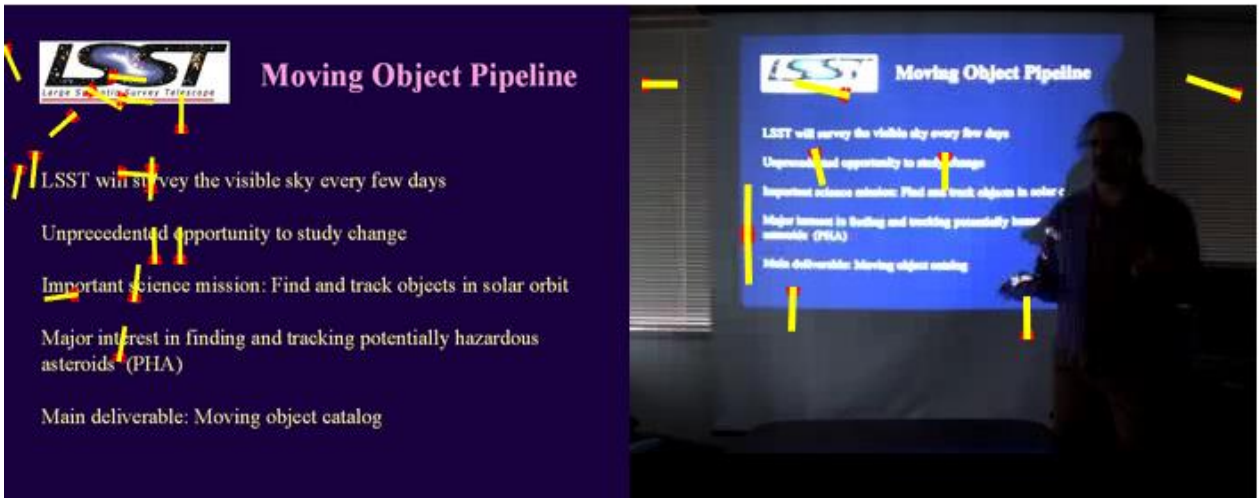


Figure 3: Every 3rd KP for slide and frame 2 for N determined for P=20 (Euclidean)

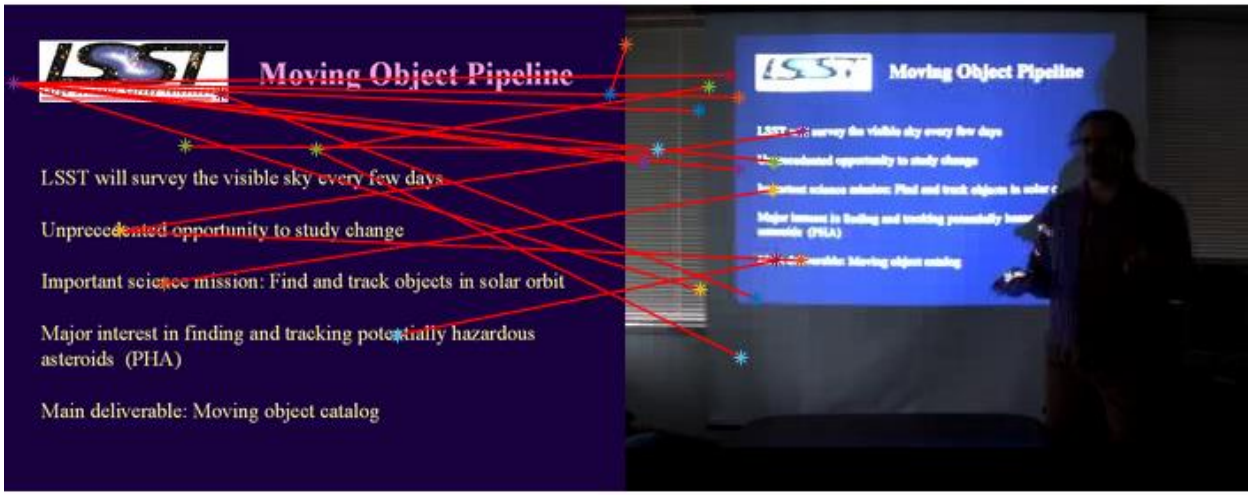


Figure 4: Matching keypoints for slide and frame 1 for P=20 (Euclidean) and for every 3rd pair

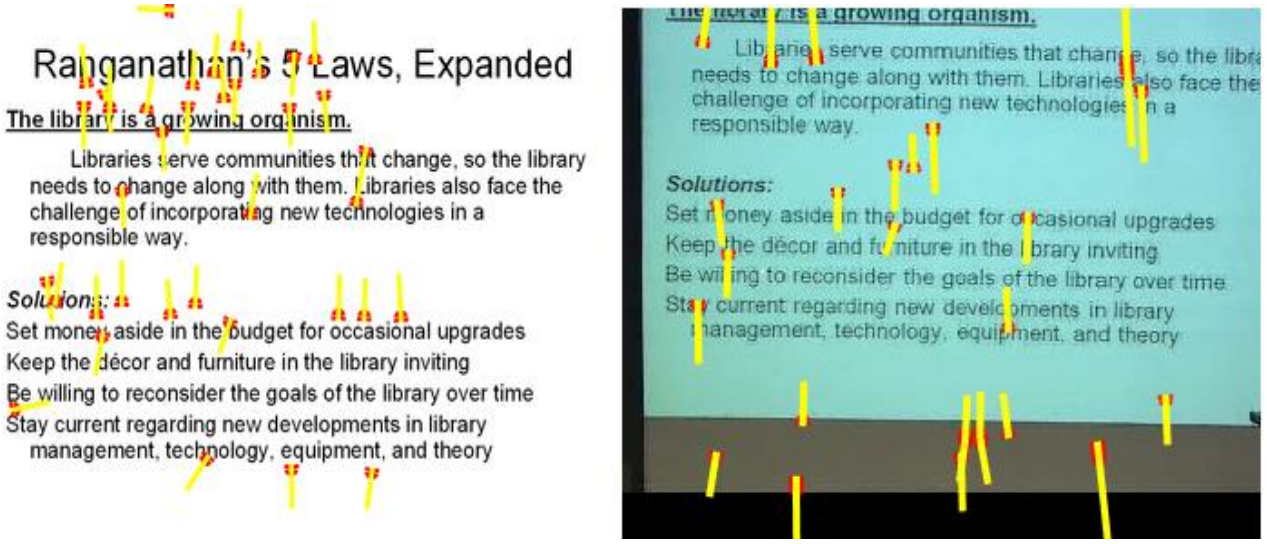


Figure 5: Every 3rd KP for slide and frame 3 for N determined for P=20 (Euclidean)

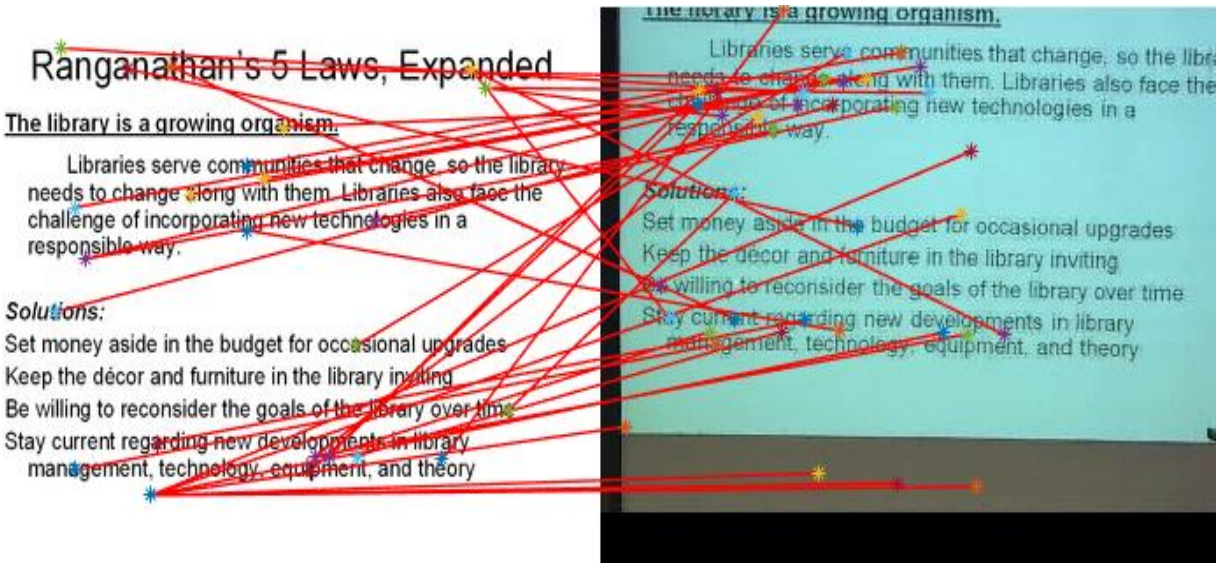


Figure 6: Matching keypoints for slide and frame 1 for P=20 (Euclidean) and for every 3rd pair

2. Using angular distance:

   We calculate the angle between each pair of keypoints using the dot product of the pair of keypoints divided by the product of their norms, which gives cosine of the angle between them.

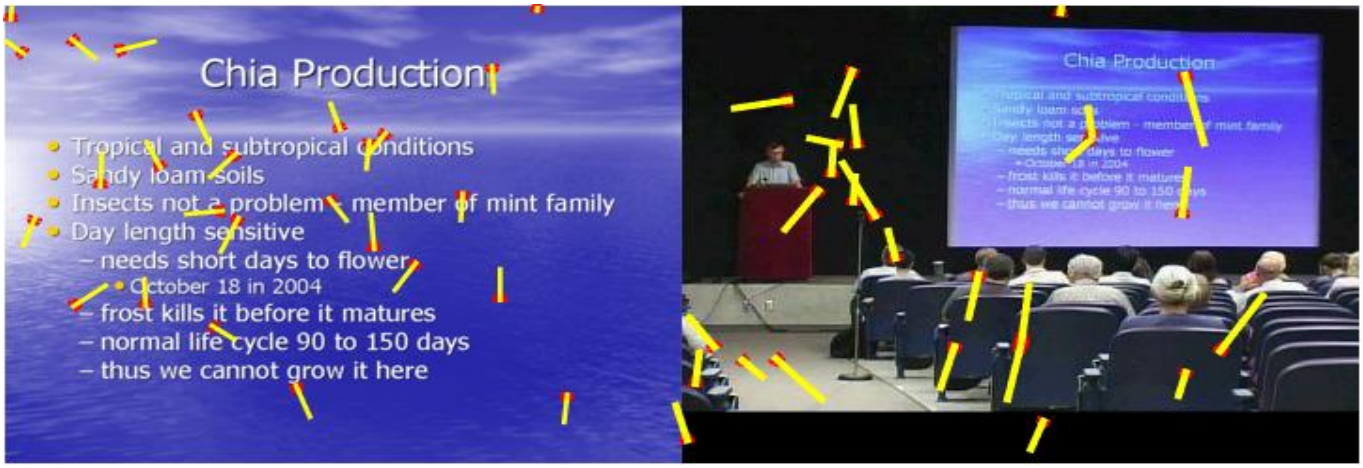


Figure 7: Every 3rd KP for slide and frame 3 for N determined for P=20 (Angular)

Figure 8: Matching keypoints using P=20 for slide and frame 2 (Angular)

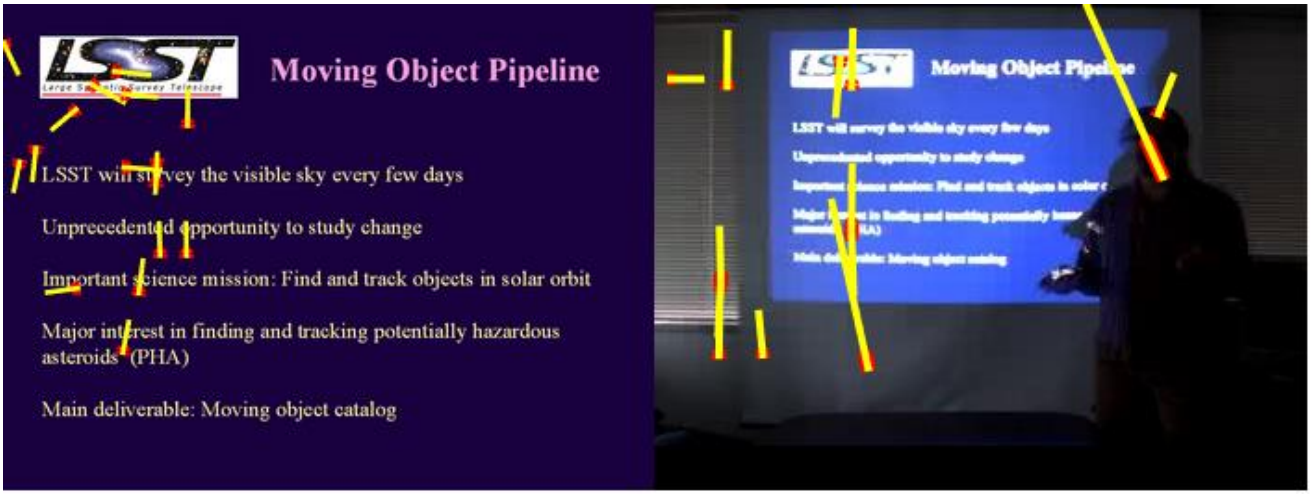


Figure 9: Every 3rd KP for slide and frame 2 for N determined for P=20 (Angular)

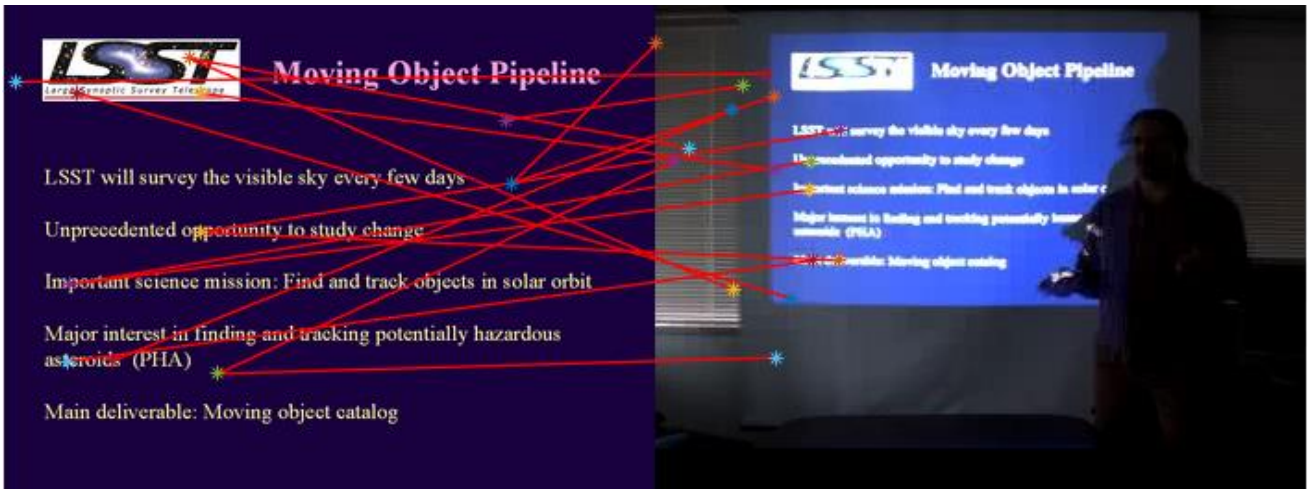


Figure 10: Matching keypoints for slide and frame 2 with P=20% (Angular)

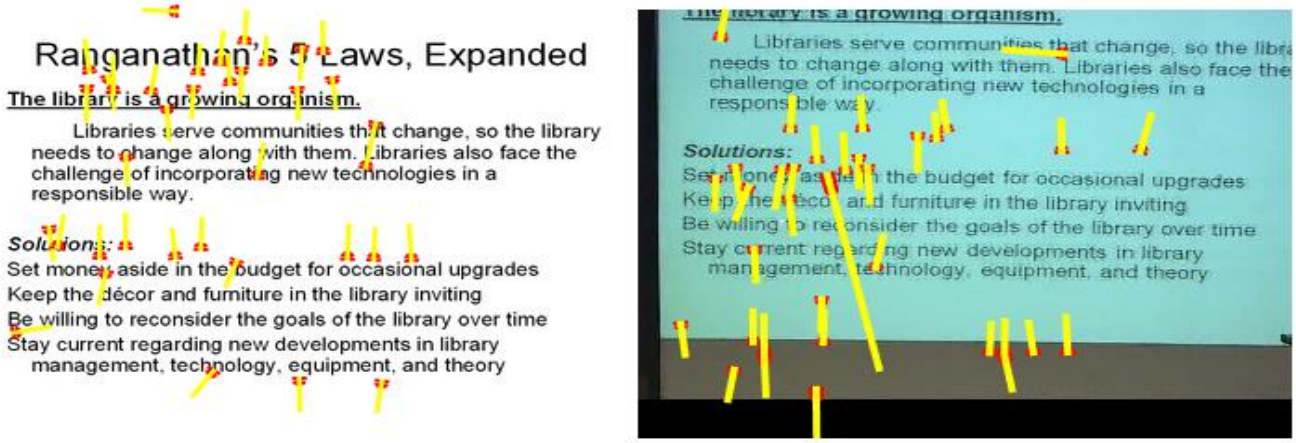


Figure 11: Every 3rd KP for slide and frame 3 for N determined for P=20 (Angular)

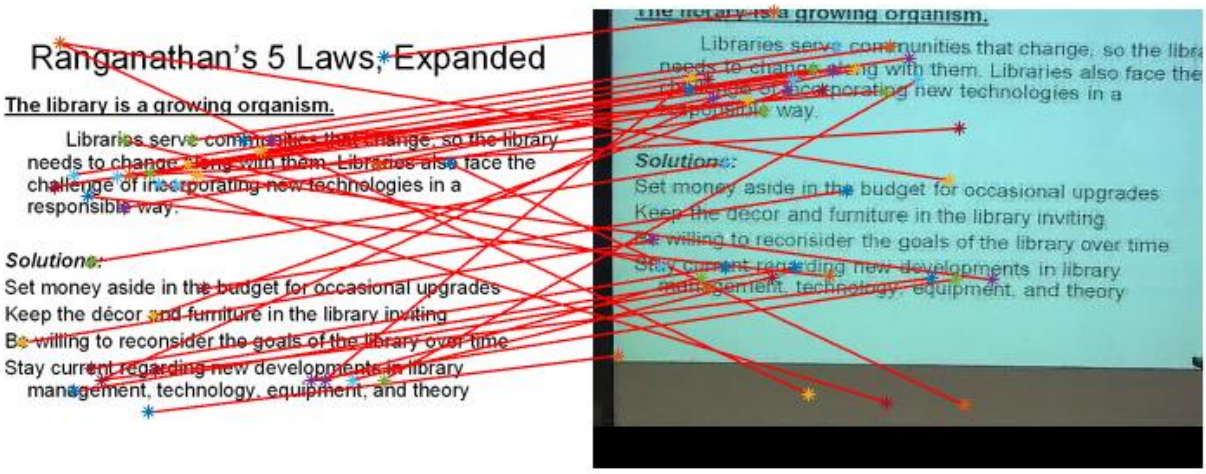


Figure 12: Matching keypoints for slide and frame 3 with P=20 (Angular)

3. Using ratio of distances of Nearest and Second Nearest Neighbor:

   Based on David Lowe's paper we determine the N matches by determining the ratio between nearest neighbor for a point to the second nearest neighbor for that point using Euclidean distance and then comparing with a pre-defined threshold. In this case, we use 0.7 as the threshold. The results based on this computation are given in the figures 13,14,15,16,17 and 18.

Figure 13: Every 3rd KP for slide and frame 1 using NN/SNN ratio (>0.7)

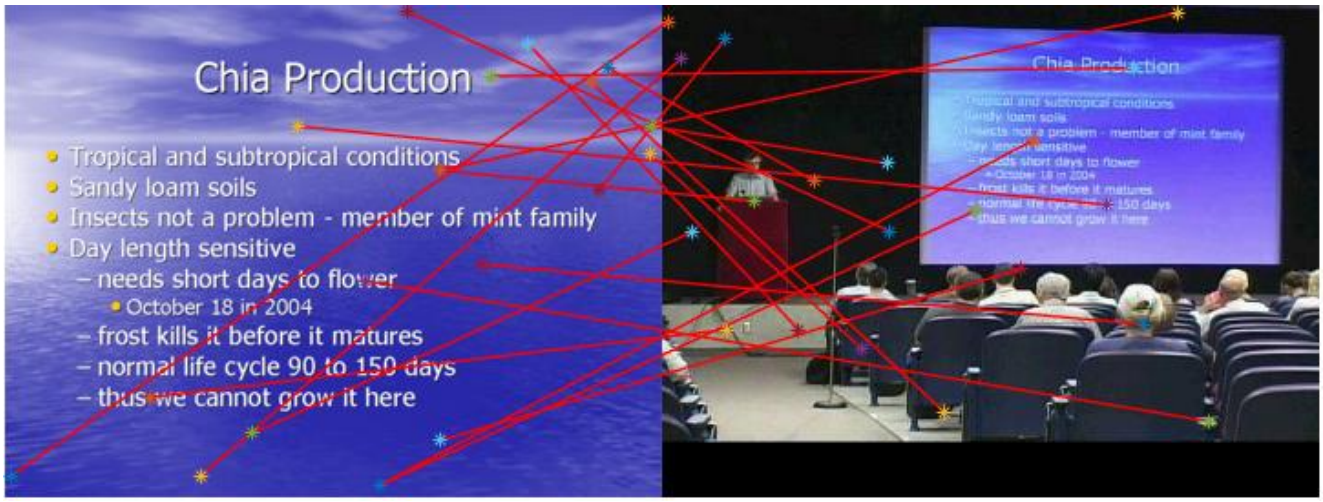


Figure 14: Matching KPs for slide and frame 1 using P=20 and NN/SNN ratio (>0.7)

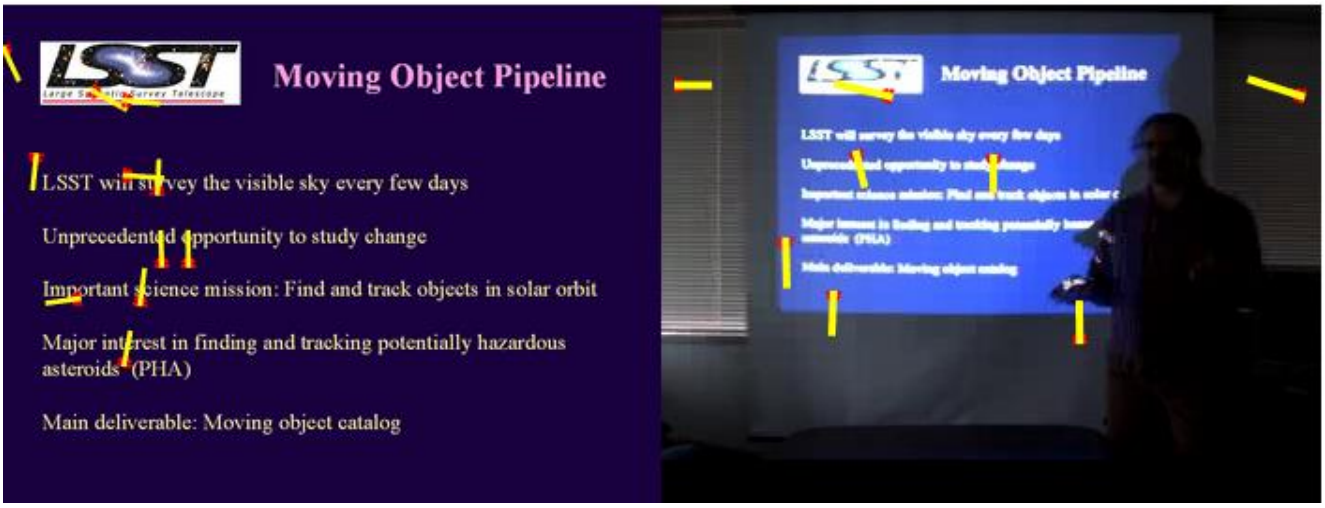


Figure 15: Every 3rd KP for P=20 and NN/SNN (>0.7) for slide and frame 2

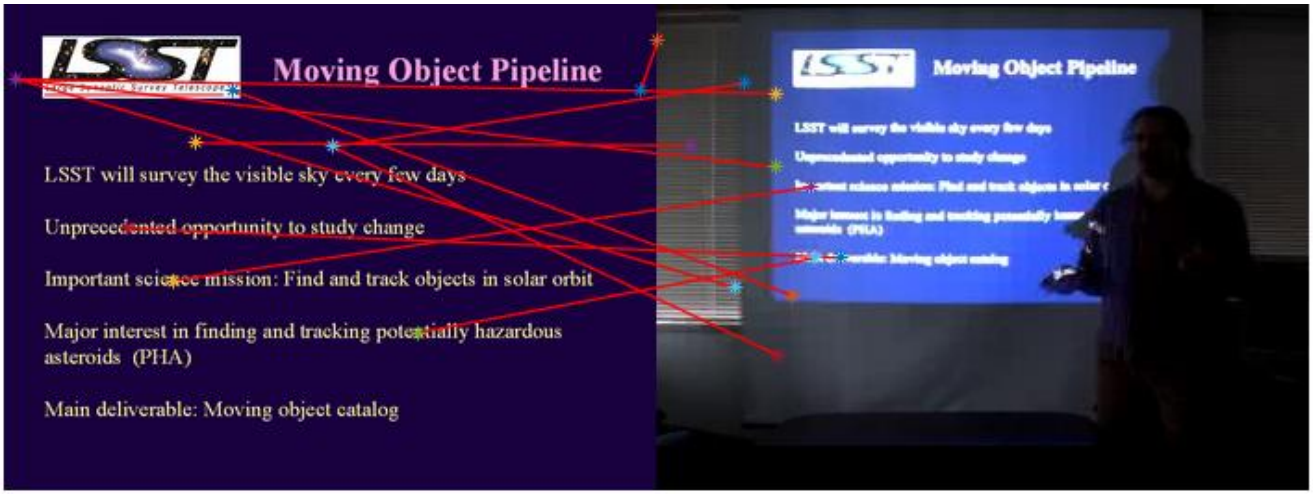
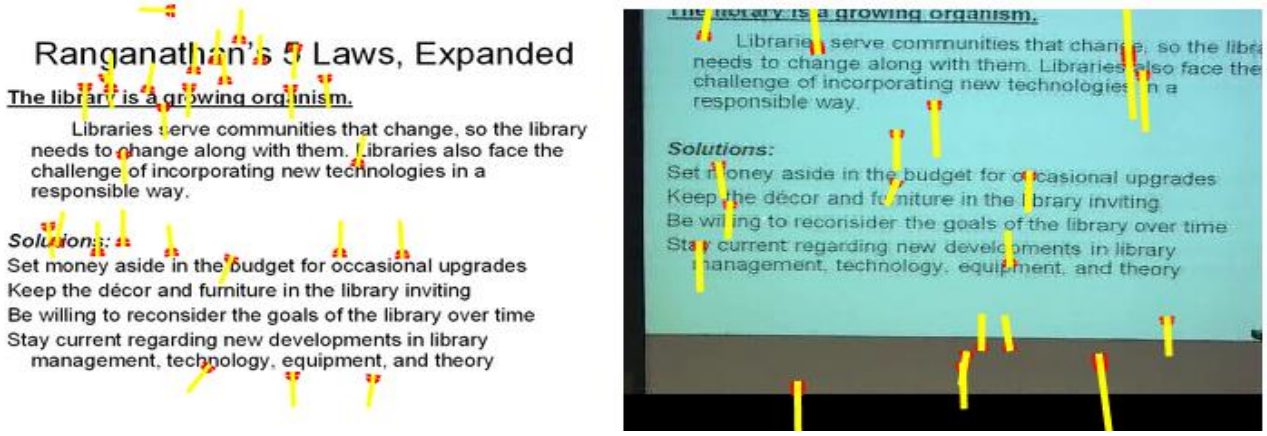


Figure 16: Matching KPs for P=20 and NN/SNN(>0.7) for slide and frame 2



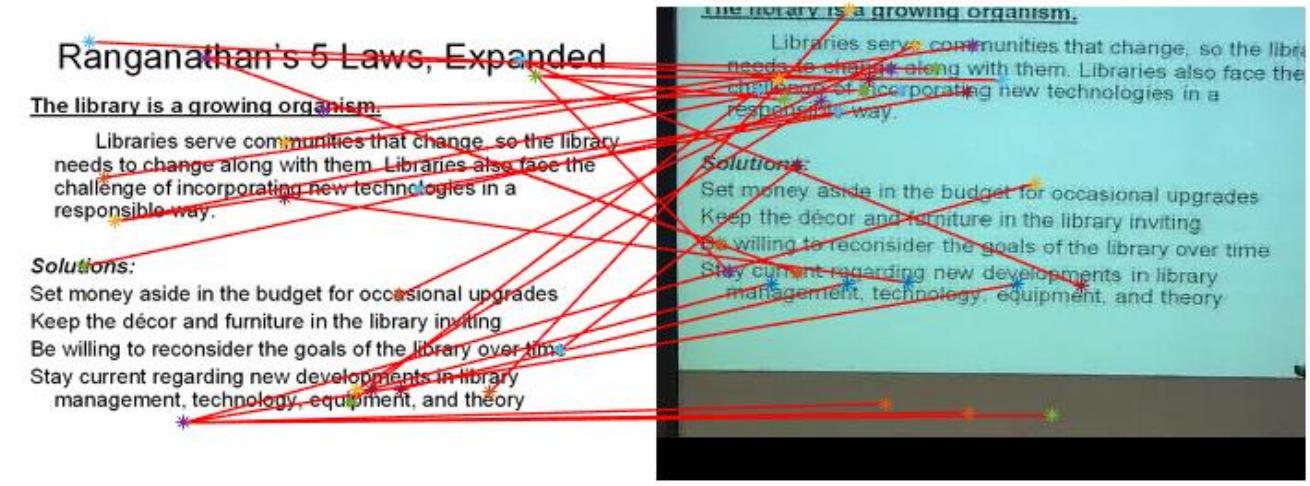


Figure 17: Matching KPs for slide and frame 3 using NN/SNN ratio (>0.7) for P=20

4. Confusion matrices - matching all slides to all frames

We use the NN/SNN distance ratio with 0.7 threshold established in previous example to compute the confusion matrix required.

We match each slide with each frame and try to find the number of matches. The confusion matrix is thus given as in Figure 18. We see that slide1 matches with frame1 as maximum matches for slide1 is with frame1.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

Figure 18: Confusion matrix

Since we are trying to find exact 1-1 match, the number of matches obtained seem less.

PART – B:

5. K-Means Clustering:

First we determine the results with K=5. We show the segmented images in comparison with the original image on the left. We print the value of the objective function at each iteration and continue clustering until convergence.

Figure 19: sunset.tiff on the left with the segmented image on the right with K=5

We show results for K=5 and K=10 for each input image in figures 19-24.

Figure 20: tiger-1.tiff on the left with the segmented image on the right for K=5

Figure 21: tiger-1.tiff on the left with the segmented image on the right for K=5

K=10

Figure 22: sunset.tiff on the left with the segmented image on the right for K=10

Figure 23: tiger-1.tiff on the left and the segmented image on the right for K=10

Figure 24: tiger-2.tiff on the left and the segmented image on the right for K=10

6. Adding pixel coordinates (X,Y) to feature vector (R,G,B)

Now, we add the spatial aspect of a pixel to the feature vector including a scaling factor λ. We observe the results for a few images by varying λ=1 to λ=15. The major difference we can see is that the clusters are a lot more coherent as locality clusters pixels of same color and at closer distance. By varying λ, we see that the region bounded by a cluster increases in the figures 26 and 27.

K=5 and λ=1

Figure 25: sunset.tiff with segmented image on right with K=5 and λ=1

Figure 26: tiger-1.tiff with segmented image on right with K=5 and λ=1

Figure 27: tiger-2.tiff with segmented image on right with K=5 and λ=1

K=5 and λ =5

Figure 28: tiger-1.tiff with segmented image on right with K=5 and λ=1

Figure 29: tiger-1.tiff with segmented image on right with K=5 and λ=5

Figure 30: tiger-1.tiff with segmented image on right with K=5 and λ=5

K=5 and λ = 10

Figure 26: sunset.tiff with segmented image on right with K=5 and λ=10

Figure 26: tiger-1.tiff with segmented image on right with K=5 and λ=10

K=5 and λ=15

Figure 27: sunset.tiff with segmented image on right with K=5 and λ=15

Figure 28: tiger-1.tiff with segmented image on right with K=5 and λ=15

PART – C:

Texture Segmentation using filters and K-Means:

In this problem, for each image (Grayscale) we apply a set of Gaussian filters with sigmas 0.62, 1, 1.6 with a window size 3. We choose these filters based on Forsyth's suggestion in the textbook. This acts as a "spot" filter. For each image, we compute mean squared sum of the responses and treat it as a feature vector. Then we perform K-Means clustering with K=5 and each cluster has a random color assigned to it. The resultant segmented image shows the texture. Figures 29-31 visualize the texture for the given images.

Figure 29: Segmented image indicating texture for black and white image of sunset.tiff

Figure 30: Segmented image indicating texture for black and white image of tiger-1.tiff

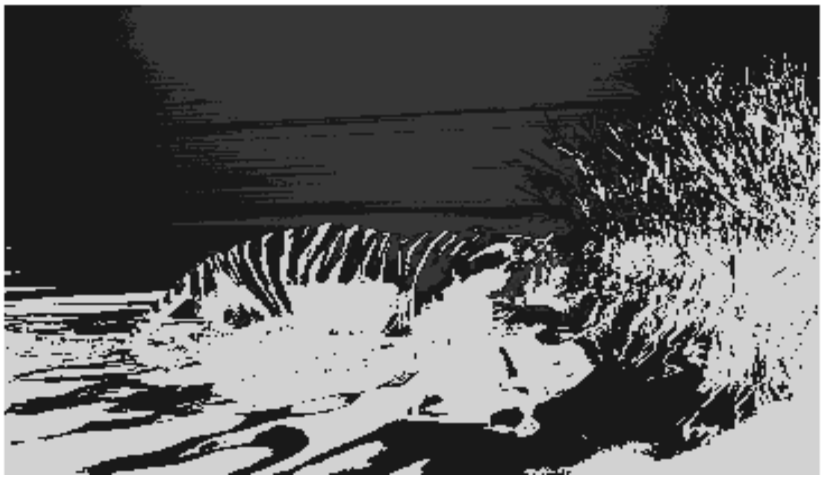


Figure 31: Segmented image indicating texture for black and white image of tiger-2.tiff

Feature Vector with RGB

Apart from the filter responses, we include the RGB values at particular point along with the filter response at that pixel in feature vector and perform K-Means clustering on this. We visualize texture by assigning random colors to each cluster. Figures 32-34 show the textures using this feature vector.

Figure 32: Segmented image indicating texture of sunset.tiff with inclusion of RGB

Figure 33: Segmented image indicating texture of tiger-1.tiff with inclusion of RGB

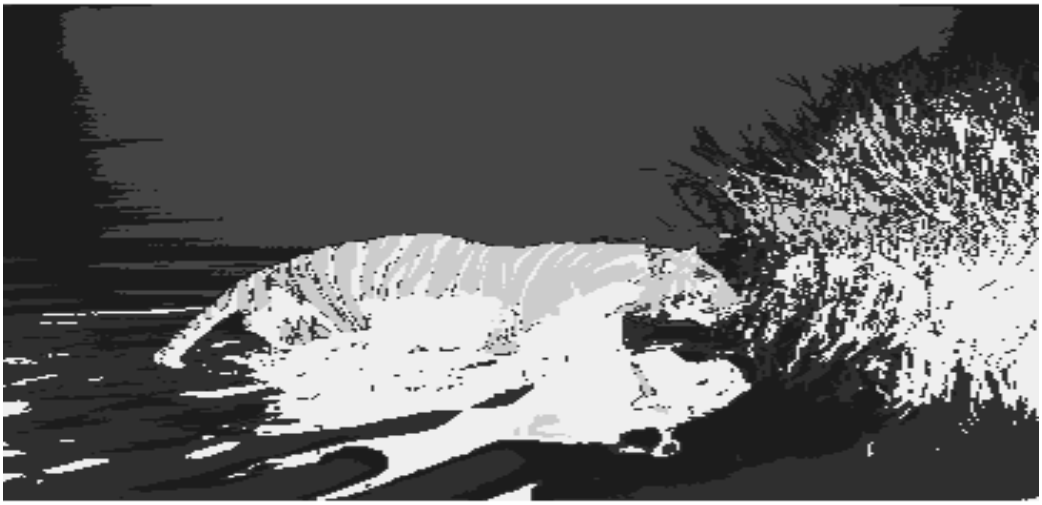


Figure 34: Segmented image indicating texture of tiger-2.tiff with inclusion of RGB

Location and color

Like the previous question, along with the filters response at a pixel and RGB, we include the location (I,j) in the feature vector and perform K-Means clustering with K=5. We see the results in Figures 35-37.

Figure 35: Segmented image indicating texture of sunset.tiff with inclusion of RGB and location

Figure 36: Segmented image indicating texture of tiger-1.tiff with inclusion of RGB and location

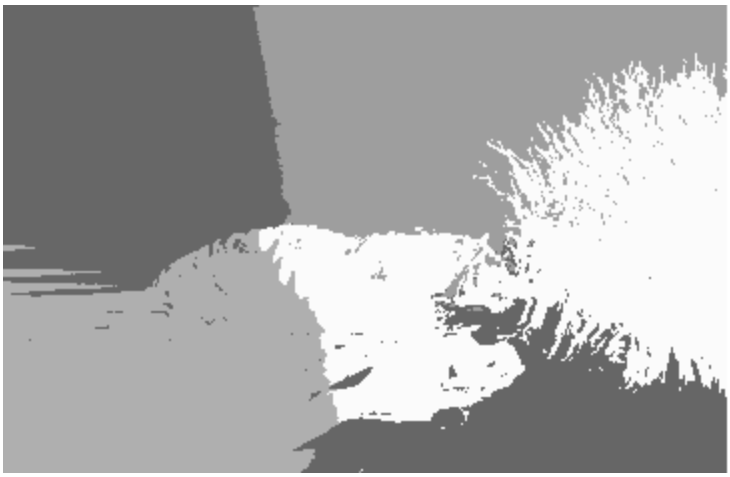


Figure 37: Segmented image indicating texture of tiger-2.tiff with inclusion of RGB and location

PART- D

Texton map:

In this problem, we take a sample set of images of the same surface (but made of different materials) as a training set. These have been obtained from the Columbia-Utrecht (curet) database.

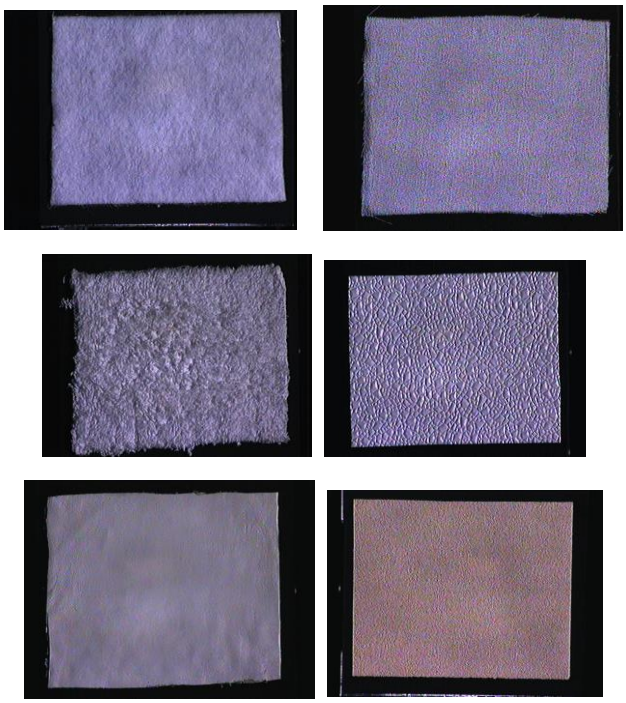


Figure 38 Sample images used for creating texton dictionary

Since training takes considerable time, we take only 3 of the images (first three) and compute the feature vectors. Firstly, we convert the image to grayscale and apply 3 Gaussian filters ("Spot" filter) with sigmas 0.62, 1, 1.6 with W=3 and store the responses as feature vector. Then we apply K-Means clustering on this feature vector with K=5 to get means for this image. We do this similarly for the other 2 images and obtain a total of 15 textons (centroids).

Now we take the first image in Figure 38 and compute the texture map. For each pixel, we take the feature vector and compute the nearest texton using Euclidean distance. The texton map at this pixel will hold the value of the texton. As a result, we get the texton map for the image using just the filter response as feature vector as in Figure 39. We also compute the histogram for each texton.

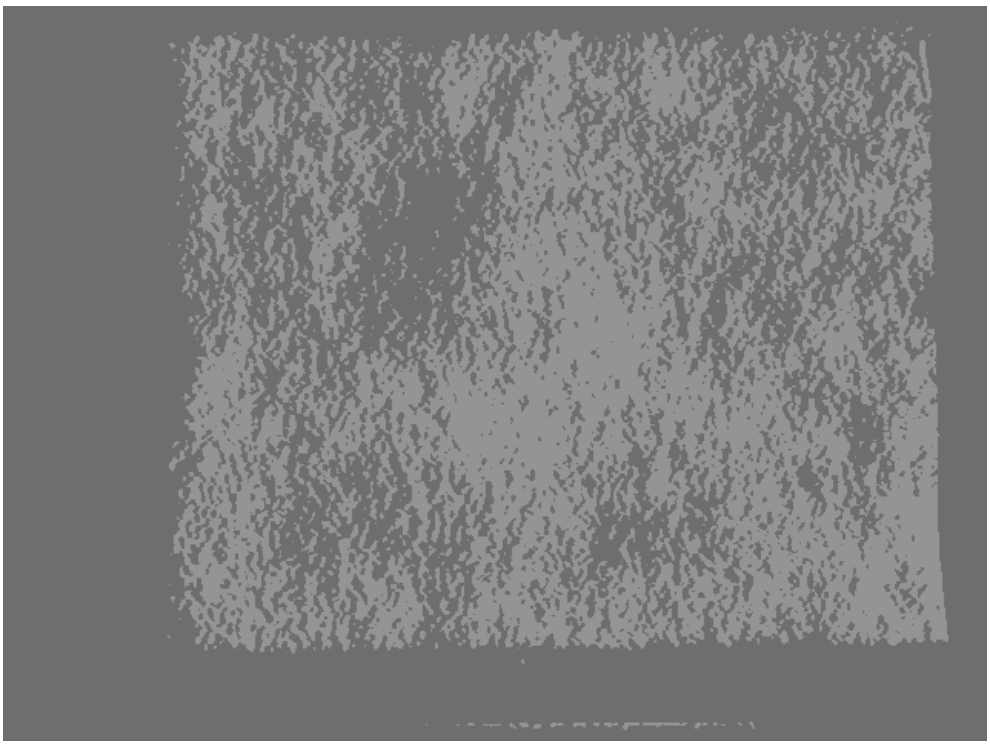


Figure 39 Texture map of first image in figure 38 using only filter responses

Now for each pixel, we append the RGB values at that pixel to the feature vector and then apply the same process as above. The result is in Fig 40 with texture boundaries much clearer.

Figure 40 Texture map of 1st image in Figure 38 using FV = Filter response + RGB

We also append the location of a pixel to the feature vector and then compute the textons and as result the texton map as show in Figure 41.

Figure 41: Texture map of 1st image in Figure 38 using FV = Filter response + RGB + location

We run the same code on tiger-1.tiff with only filter response to get the result in Figure 42 and filter response + RGB in figure 43 and filter response + RGB + location in Figure 44. .

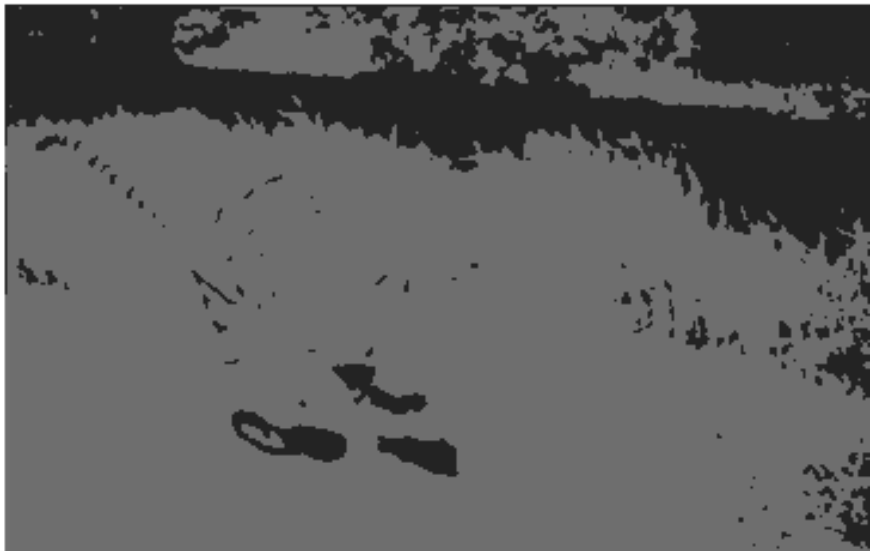


Figure 42: Texture map for tiger-1.tiff using Feature vector without RGB, location

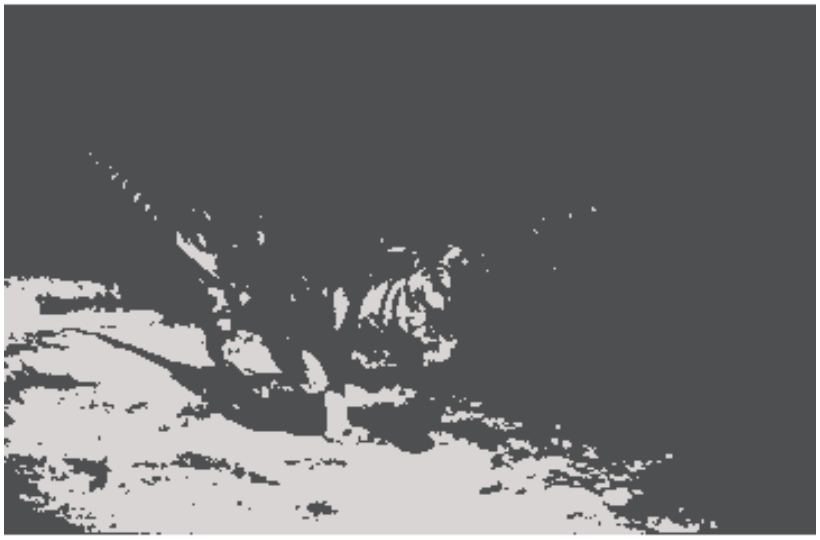


Figure 43: Texture map for tiger-1.tiff using Feature vector with RGB

Figure 44: Texture map for tiger-1.tiff for FV + RGB + location.

PART – E

HW5 Part B – 2 (OCR)

For the given image, we choose the below 5 templates for matching.

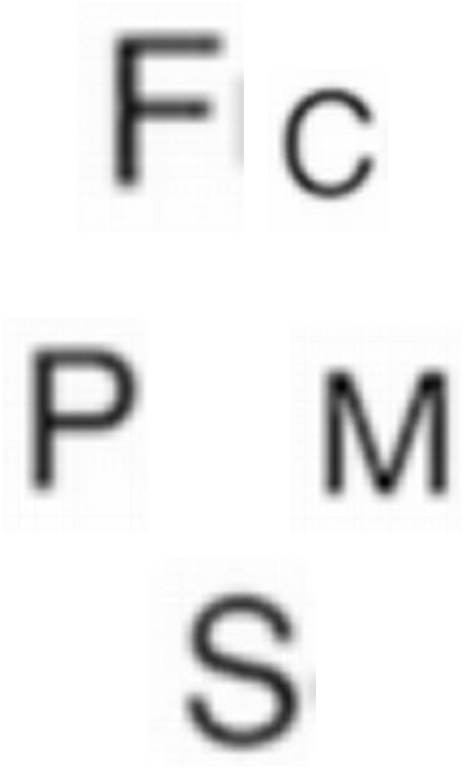


Figure 45: Templates used in OCR process for matching.

We use correlation (corr2) between each of the templates and the portions of sub image (portion of the image which is the size of the template). We determine the areas with maximum correlation and highlight the region with rectangle.

We detect all the templates in the image accurately as shown in Figure 14. We ensure that each of the sub image templates sum up to zero if their mean is equal to each of the pixels.

Figure 46: Red highlighted boxes indicating template matches

The confusion matrix C is perfectly diagonal as all the letters are accurately classified. Therefore, the matrix would be as in Figure 15.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 15: Confusion matrix – rows would indicate letters P, S, M, F and C, all of which are accurately classified.

NOTE: The corr2 function is very slow and hence the time taken for the letters to be identified and the rectangular boxes to be overlaid over them can be in the order of minutes.

Appendix

Code Snippets

PART – A

Find distances and draw points

```
function[] =
draw_kp_matches(i,metric,slide_feature,frame_feature,N,slide_kp,frame_kp)
  slide_kp_rows = size(slide_kp,2);
    frame_kp_rows = size(frame_kp,2);
 dist = zeros(frame_kp_rows,slide_kp_rows);
 if strcmp(metric,'euclid')==1
    for k=1:frame_kp_rows
       for l=1:slide_kp_rows
           dist(k,l) = sqrt(sum((slide_kp(:,l)-frame_kp(:,k).^2)));
       end
    end
 elseif strcmp(metric,'angle')==1

      for k=1:frame_kp_rows
       for l=1:slide_kp_rows
           dist(k,l) = computeAngle(slide_kp(:,l),frame_kp(:,k));
       end
     end
 end

    min_frame_dist = zeros(frame_kp_rows,1);
    min_frame_slide_index = zeros(frame_kp_rows,1);
    for j=1:frame_kp_rows
       [minVal,index] = min(dist(j,:));
       min_frame_dist(j) = minVal;
       min_frame_slide_index(j)=index;
    end

    frame_image_color = imread(strcat('frame',int2str(i),'.jpg'));



    x=[];y=[];
```

```
    frame_points=[];slide_points=[];




    slide_image_color = imread(strcat('slide',int2str(i),'.tiff'));
    slide_img = [];
    if size(slide_image_color,3)==4
     slide_img = slide_image_color(:,:,1:3);
    else
        slide_img = slide_image_color;
    end









 figure;
      imshowpair(slide_img, frame_image_color, 'montage')
     hold on;



      for j=1:3:N
       feature = frame_feature(:,j);
       x = feature(1);
    y = feature(2);
       u = feature(3) * cos(feature(4)); % convert polar (theta,r) to
cartesian
       v = feature(3) * sin(feature(4));
       h=quiver(x,y,u,v);
        h.Color = 'red';
        h.LineWidth=6;
          x(1) = feature(1);
       y(1) = feature(2);
       x(2) = x(1) + 10*feature(3) * cos(feature(4));
        y(2) = y(1) + 10*feature(3) * sin(feature(4));

       plot(x, y,'Color','y','LineWidth',2);


      % set(gca, 'XLim', [1 10], 'YLim', [1 10]);

      end
    for j=1:3:N

       min_index = min_frame_slide_index(j);
       feature = slide_feature(:,min_index);

        x = feature(1)+size(frame_image_color,2);
    y = feature(2);
```

```
       u = feature(3) * cos(feature(4)); % convert polar (theta,r) to
cartesian
       v = feature(3) * sin(feature(4));
       h=quiver(x,y,u,v);
        h.Color = 'red';
        h.LineWidth=6;

       x(1) = feature(1);
       y(1) = feature(2);
       x(2) = x(1) + 10*feature(3) * cos(feature(4));
        y(2) = y(1) + 10*feature(3) * sin(feature(4));

       plot(x+size(frame_image_color,2), y,'Color','y','LineWidth',3);


    end
    hold off;

    figure;
    imshowpair(slide_img, frame_image_color, 'montage')
    hold on;
    for k=1:3:N
          min_index = min_frame_slide_index(k);
          feature_slide = slide_feature(:,min_index);
          feature_frame = frame_feature(:,k);
          p1 = [feature_slide(1);feature_slide(2)];
          p2 = [feature_frame(1)+size(frame_image_color,2);feature_frame(2)];
          h = line([p1(1) ; p2(1)], [p1(2) ; p2(2)]) ;

          set(h,'linewidth', 1, 'color', 'r') ;
    end

    for k=1:3:N
          min_index = min_frame_slide_index(k);
          feature_slide = slide_feature(:,min_index);
          feature_frame = frame_feature(:,k);
          p1 = [feature_slide(1);feature_slide(2)];
          p2 = [feature_frame(1)+size(frame_image_color,2);feature_frame(2)];
          hold on;
          plot(p1(1),p1(2),'*');
          hold on;
          plot(p2(1),p2(2),'*');
    end

   hold off;
```

PART B

K-Means clustering

```
function[] = kMeansCluster(imageName,k)
    img = imread(imageName);
```

```
    temp_img = imread(imageName);
    r_means = zeros(k,1);
    g_means = zeros(k,1);
    b_means = zeros(k,1);


    redPlane = img(:, :, 1);
    greenPlane = img(:, :, 2);
    bluePlane = img(:, :, 3);
    cluster_means=[];
    for i=1:k
        temp1 = randi([0,size(img,1)]);
        temp2 = randi([0,size(img,2)]);
       r_means(i) =  redPlane(temp1,temp2);
       g_means(i) = greenPlane(temp1,temp2);
       b_means(i) = bluePlane(temp1,temp2);
       cluster_means = [cluster_means;r_means(i),g_means(i),b_means(i)];
    end

   prev_means = zeros(size(cluster_means));
    clusters = cell(k,1);
    errors = zeros(k,1);

    rms_error = -1;
    while 1
        clusters = cell(k,1);
           for l=1:size(img,1)
               for m=1:size(img,2)
                   r_val = redPlane(l,m);
                   g_val = greenPlane(l,m);
                   b_val = bluePlane(l,m);
                   point = [r_val;g_val;b_val];
                   clusterId = findNearestCluster(point,cluster_means,k);
                   clusters{clusterId} =
[clusters{clusterId};point(1),point(2),point(3)];
               end
           end

           for i=1:k
              if size(clusters{i},1) == 0
                  point = get_random_point_from_clusters(clusters,k);
                  clusters{i} = [clusters{i};point(1),point(2),point(3)];
              end
           end

           for i=1:k
               cluster = clusters{i};
               temp = cluster';
               r_m = mean(temp(1,:));
               g_m = mean(temp(2,:));
               b_m = mean(temp(3,:));
               prev_means(i,:,:) = cluster_means(i,:,:);
               cluster_means(i,:,:) = [r_m,g_m,b_m];
           end
```

```
            for i=1:k
               cluster = double(clusters{i});
               temp = cluster;
               cluster_mean = cluster_means(i,:,:);
               error = 0.0;
               for j=1:size(temp,1);

                     p = temp(j,:);
                     error = error + sqrt(sum((p-cluster_mean).^2));
               end
               errors(i)=error;
            end
            prev_error = rms_error;
             rms_error = sum(errors);
             if prev_error~=-1 && prev_error == rms_error
                break;
             end
    end

    for i=1:size(img,1)
        for j=1:size(img,2)
           pixel = [redPlane(i,j);greenPlane(i,j);bluePlane(i,j)];

           mean_color = get_mean_from_clusters(pixel,cluster_means,k);
           img(i,j,1) = mean_color(1);
           img(i,j,2) = mean_color(2);
           img(i,j,3) = mean_color(3);
        end
    end


    figure;
   imshowpair(temp_img,img,'montage');

end
```

PART – E
OCR

The code snippet for template matching is:

```
for i=1:num
    template_image = rgb2gray(images{i});
    [h,w] = size(template_image);

    corr_matrix = zeros(H,W);

    for j=1:H-h
        for k=1:W-w
                sub_image = double(image(j:(j+h-1),k:(k+w-1)));
                if mean(sub_image(:)) == sub_image(1,1)
                    for s=1:h
```

```
                        for t=1:w
                            if mod(t,2)==0
                                sub_image(s,t) = (-1)*sub_image(s,t);
                            end
                        end
                    end
                end
                result = corr2(template_image,sub_image);
                if isnan(result)
                   result=0;
                end
                corr_matrix(j,k)=result;
            end
        end
       % max_corr = max(abs(corr_matrix(:)));
        [maxCorrValue, maxIndex] = max(abs(corr_matrix(:)));
    [yPeak, xPeak] = ind2sub(size(corr_matrix),maxIndex(1));
        rectangle('Position',[xPeak yPeak h w],'edgecolor','r');
    end
end
```