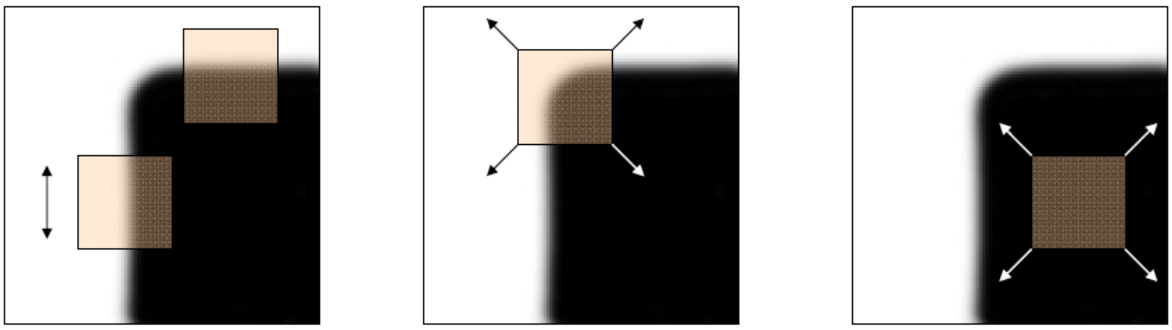
C-Lab-2 Report u5752771

Task-1: Harris Corners:

First of all, four image are read and convert to grey scale. A Gaussian kernel is used as window because in doing this more weights are given to middle pixels, while using square window gives every pixels in the window a same weight. Then by using the first order Tylor approximation, the derivative in x, y and xy direction are computed so that the cornerness can be calculated.

$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

The cornerness function can be written in the form of eigenvalue λ_1 *and* λ_2:



"edge": $\lambda_1 >> \lambda_2$, or $\lambda_2 >> \lambda_1$

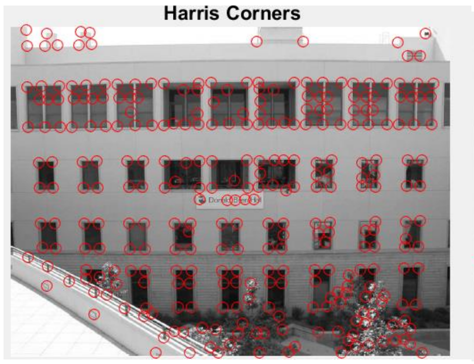
"corner": λ_1 and λ_2 are large, $\lambda_1 \sim \lambda_2$;

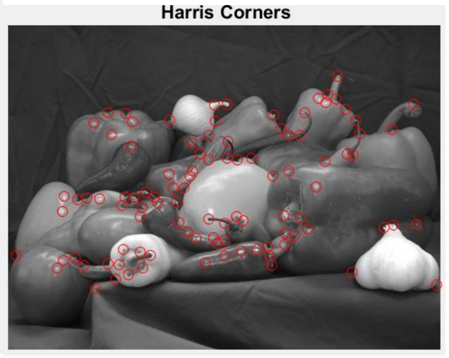
"flat" region λ_1 and λ_2 are small;

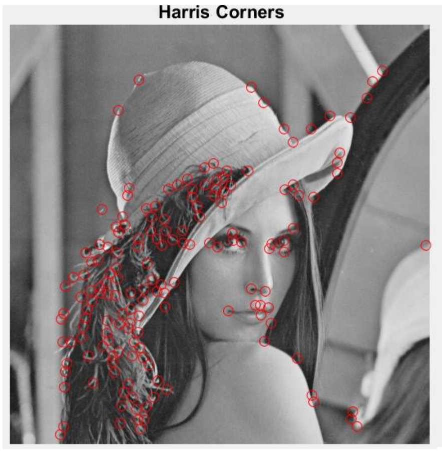
According to the cornerness function, the larger cornerness means the eigenvalue of matrix are both large and close, this gives corner in the image. Other cases indicate the pixels inside the window are either edge or flat.

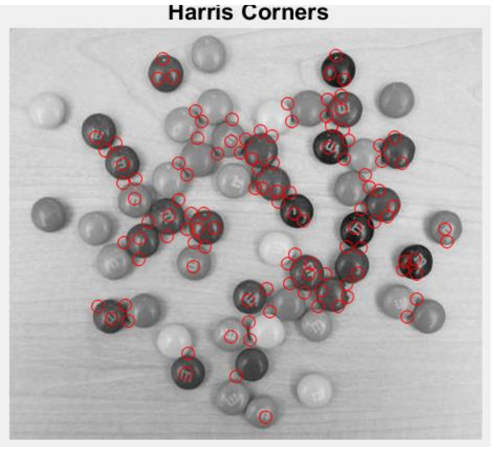
After cornerness is found, non-maximum suppression is performed by first, finding pixels with maximum intensity in the maximum filter kernel, and replace those pixels with value-1 pixels. Since we have the value-1 pixels the locations of those value-1 pixels, which is the corner location, can be easily found by tresholding. Also, corner found in the border should be eliminated, due to different padding of image border before convolution with the filter, it is would probably introduce a lot of pixel discontinuities. This discontinuities can be treated as "corners" due to the inner properties of the cornerness computation method. The picture below is tested under threshold of 0.001.

C-Lab-2 Report u5752771

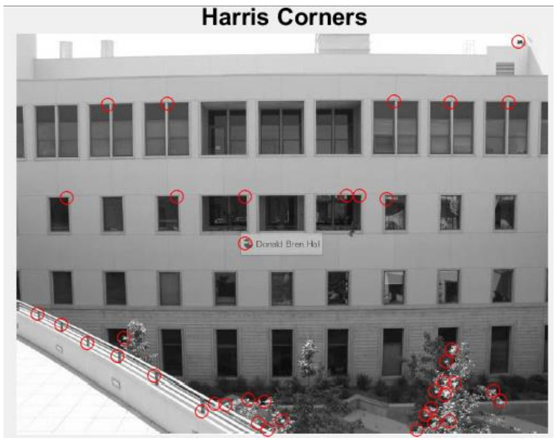






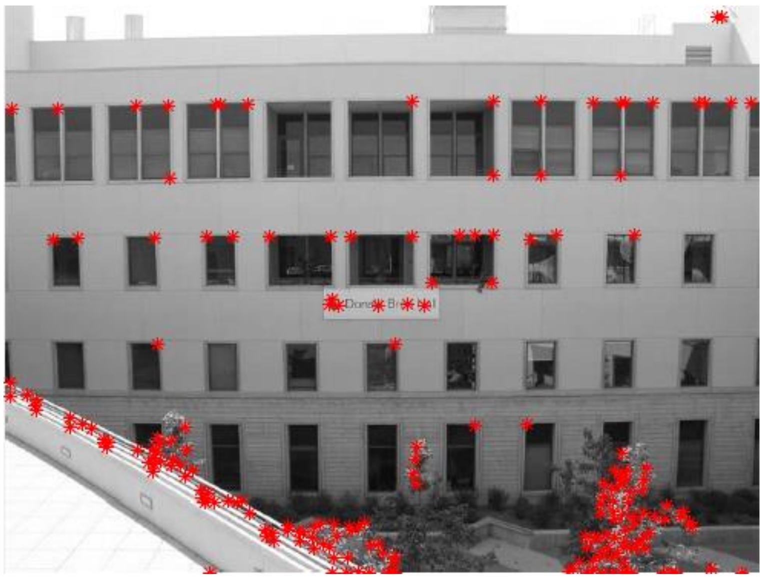


For a larger threshold of 0.05, less corners are plotted because most of the corners are discarded during tresholding. The diagram on the right hand side shows the result.



The diagram below shows the result of Matlab built-in corner function with default parameter. The function perform Harris corner algorithm to find the cornerness. There are few difference in the resulting image. First is the amount of corner being found. This may due to the choice of threshold and the K value. Different threshold and the K value result in different result after cornerness calculation and non-maximum suppression. Second, in the diagram below some corner in the border are marked. This is not a desirable result. Due to different padding method applied to the image, non-necessary distortion of the image are introduced. This effect is eliminated in the self-programmed code.

C-Lab-2 Report u5752771



Code of task 1:

```
clc,clear;
figure; my_harris('Right.jpg' );
figure; my_harris('peppers.png' );
figure; my_harris('Lenna.png' );
figure; my_harris('mandm.png' );
img = imread('Right.jpg');      %read image
img = im2double(img);           %covert image to double
img = rgb2gray(img);            %covert image to gray
c = corner(img);                %get the corner by bulitin function
imshow(img);                    %show image
hold on
plot(c(:,1), c(:,2), 'r*');     %hold figure and map the corner
```

```
function [data_clusters, cluster_stats] = my_harris(imageFile )
img = imread(imageFile);                            %load image
img = rgb2gray(img);                                %convert it to gray scale
bw = im2double(img);                                %convert it to class double
sigma = 2;                                          %sigma of gaussian kernel
thresh =0.001;                                      %threshold value filter small cornerness
sze = 11;                                           %size of maximum filter
disp = 0;                                           %display parameter
dy = [-1 0 1; -1 0 1; -1 0 1];                      %gradient filter in y direction
dx = dy';                                           %gradient filter in x direction
lx = conv2(bw,dx,'same');                           %gradient image in x direction
ly = conv2(bw,dy,'same');                           %gradient image in y direction
g = fspecial('gaussian',max(1,fix(6^sigma)),sigma); %define gaussian kernel
lx2 = conv2(lx.^2, g, 'same');                      %smoothed image derivatives in x direction
ly2 = conv2(ly.^2, g, 'same');                      %smoothed image derivatives in y direction
lxy = conv2(lx.*ly,g, 'same');                      %smoothed image derivatives in xy direction
                %compute the cornerness
K = 0.05;                                           %empirical constant 0.01-0.1
R = (lx2 .* ly2) - (lxy.^2) - K * ( lx2 + ly2 ).^ 2;     %cornerness
                %non-maximum suppression
mx = ordfilt2(R, sze^2, ones(sze));                 %maximum filter find max in cornerness function
radius = sqrt(sze-1);                               %define radius of the window
bordermask = zeros(size(R));                        %create bordermask same size as the image
bordermask(radius+1:end-radius, radius+1:end-radius) = 1;%set outer ring to 1, eliminate corner due to
padding method
cornerness = (R==mx) & (R>thresh) & bordermask;     %find maxima, threshold, and apply bordermask
[rows,cols] = find(cornerness==1);                  %return all the corner coordinates
imshow(bw);                                         %show original image
hold on;                                            %hold the image in same figure
```

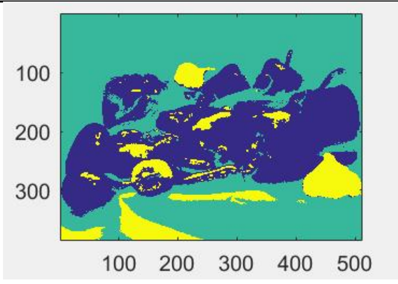
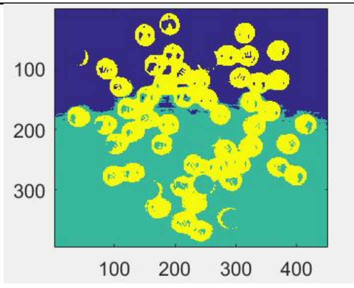
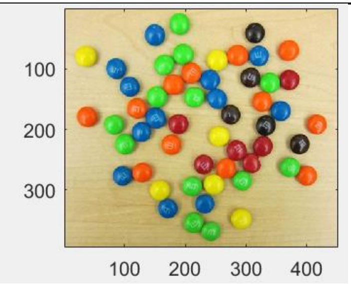
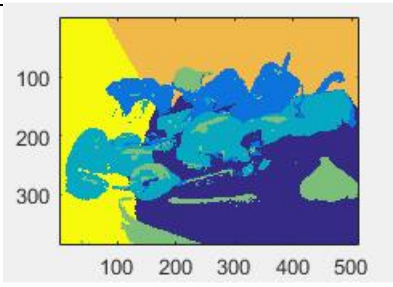
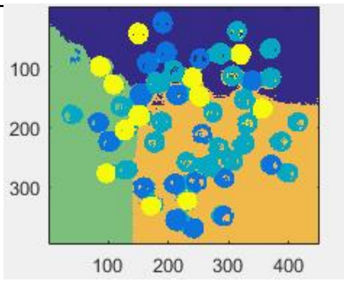
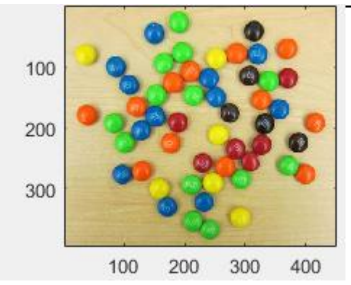
```
p = [cols rows];                                          %collect rows and cols of corners
plot(p(:,1),p(:,2),'or');                                 %plot on the original image
title('\bf Harris Corners');                              %create title for the image
```

Task-2: k-means clustering

First of all, image are read and converted to LAB colour space. Combined with the x and y coordinate of each pixels, by reshaping the image into vector forms, the features of image are created. The features includes L, A, B colour channel and x and y coordinate of the each pixels.

Next, the number of cluster we want are determined, and random label are created for each pixels. Each pixels now have non-relevant label to its feature (LABXY). Then pixels belong to each specific cluster are grouped. The means are calculated for each grouped pixels and for each feature inside that group. After that, Euclidean distances between ever pixels in the image and specific cluster centre are calculated, same operation for all the other clusters. Then those distances are compared. If the distance for a specific pixel are the closest to one cluster centre than to all the other cluster centre, a new, closest cluster label is tagged to that pixel, same operation for every pixels. This generate a new set of cluster labels for every pixels. Then by iterating the process above, the pixels can be clustered.

Clustering result:

3 clusters	
	 
6 clusters	
	 

Also, k-means method are likely to fell into local minima, this gives non desirable results. Due to random label generation at the beginning, each time the program execute is high likely that the result will be different. Some ways of clustering are legitimate in terms of algorithm, but they don't reflect any useful results. Moreover, the algorithm are slow because each iteration is calculation-intensive, this can be worse if the number of cluster are increased. One way to speed up the program is instead of calculating the Euclidean distance, we can get rid of the square root in the distance function. This does not affect the performance of the algorithm. Use KD-tree for the search, search less features, can also speed up the algorithm.

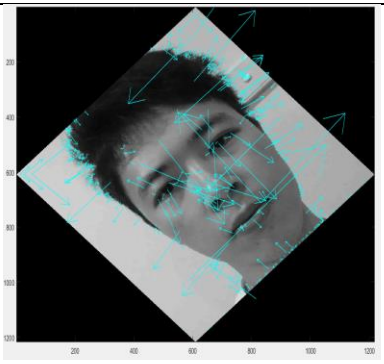
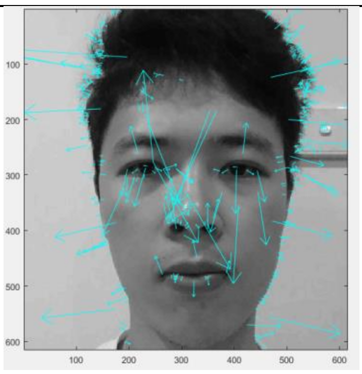
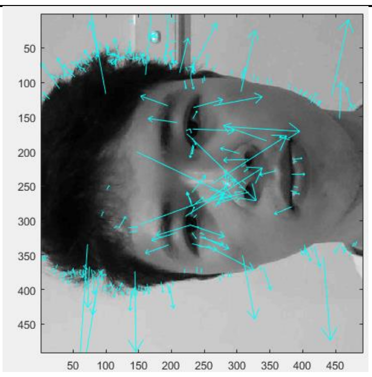
Code of task 2:

```
clc,clear;
nc = 3;                                                   %define number of clusters in final result
[data_clusters, cluster_stats] = my_kmeans(nc,'peppers.png' );  %run my kmeans function
[data_clusters, cluster_stats] = my_kmeans(nc,'mandm.png' );    %run my kmeans function
```

```
function [data_clusters, cluster_stats] = my_kmeans( nc, imageFile )
img = imread(imageFile);                                 %read image
img = im2uint8(img);                                       %convert to class uint8
cform = makecform('srgb2lab');                             %specify color transformation structure
lab = applycform(img, cform);                              %converts the color values into lab
[rows, cols, ncolors] = size(img);                         %get size of rows, cols, and colors
npixels = rows * cols;                                     %get total pixel number
[x,y] = meshgrid(1:cols,1:rows);                           %two mashgrid x,y,size[384 512]
features = img;                                            %lab intensity as features 1,2,3
factor = 1 ;                                               %geometric constraints
features(:,:,4) = factor* x;                               %image x coordinates as features 4
features(:,:,5) = factor* y;                               %image y coordinates as features 5
features = reshape( features, [npixels 5] );               %reshape matrix aline the 5 features
features = im2double(features);                            %convert feature to double
for i=1:size(features,2)                                   %[npixels, 5]
features(:,i) = features(:,i) - mean(features(:,i));       %zero mean
features(:,i) = features(:,i) / norm(features(:,i));       % normalize
end
data = features;
ndata = size(data,1);                                      %number pixels
ndims = size(data,2);                                      %number of features dimension (5)
random_labels = floor(rand(ndata,1) * nc) + 1;             %random initialization,random label for each
pixel
data_clusters = random_labels;                             %collect current label
cluster_stats = zeros(nc,ndims+1);                         %[3 6]
distances = zeros(ndata,nc);                               %[196608 5] change to [196608 3]
temp_dist = zeros(ndata,ndims,nc);                         %matrix to store distance
while(1)
    pause(0.03);                                           %pauses execution for n seconds before continuing
    last_clusters = cluster_stats;                         % Make a copy of cluster statistics for
    % comparison purposes.  If the difference is very small, the while loop will exit.
    for c=1:nc                                             %for each cluster
        [ind] = find(data_clusters == c);                  %find all data points assigned to this cluster
        num_assigned = size(ind,1);                        %how many numbers of pix are assigned for one
label
        if( num_assigned < 1 )                             %some heuristic codes for exception handling.
            disp('No points were assigned to this cluster, some special processing is given below');
            max_distances = max(distances);                %calculate the maximum distances from each
cluster
            [maxx,cluster_num] = max(max_distances);
            [maxx,data_point] = max(distances(:,cluster_num));
            data_clusters(data_point) = cluster_num;
            ind = data_point;
            num_assigned = 1;
        end                                                %end of exception handling.
        cluster_stats(c,1) = num_assigned;                 %save number of points per cluster,plus the mean
vectors
        if( num_assigned > 1 )
            summ = sum(data(ind,:));                       %sum all pix on the cn
            cluster_stats(c,2:ndims+1) = summ / num_assigned;%average value of all the pix at that cn
        else
            cluster_stats(c,2:ndims+1) = data(ind,:);
        end
    end
    diff = sum(abs(cluster_stats(:) - last_clusters(:)));   % Exit criteria
    if( diff < 0.1 )                                         %change the diff to decide when to exit
        break;
    end
    for c=1:nc
        temp(c,:) = dist(cluster_stats(c,2:ndims+1),data'); %matrix to store the distances
    end
    distances = transpose(temp);                             %get the distances matrix
    for i = 1:ndata
            [~,data_clusters(i,1)]=min(temp(1:nc,i));        %get the closest value between clusters
    end
    displayclusters( img, data_clusters )                    %displace clusters
end
```

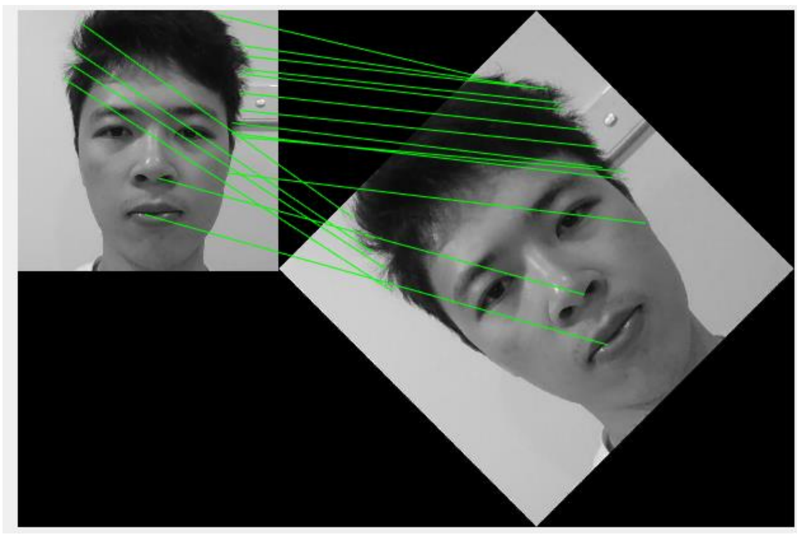
Task-3 SIFT:

First of all three image are created. By using sift code the sift detector found the keypoint location. Each returned matrix of locs from sift.m consists of 4 information: row, column, scale and orientation. The showkeys function actually takes all the information of locs to plot the image. In each image, the blue arrow is the keypoint. The length of arrow represent the scale, the direction represent the orientation.

For line drawing task, first of all ratio are determined as 0.6 and matrix to hold the successful pair are created. By using the location, scale and orientation generated by the sift detector, sift descriptor are computed. The descriptor are the histogram of oriented gradients within the local window of size at the intrinsic scale. The Euclidean distances between one descriptor in image1 and all the descriptors in image2 are computed. To find the best descriptor pair, the Euclidean distances are sort. If the best match distance is less than the second best match distance times ratio, the descriptors in image1 and image2 are said to be paired. This indicates that the best match should be distinct enough compared to the second match, so that the best match is valid best match. This processes iterate through all the descriptors in image1. Thus all the paired descriptors in two image can be found.

Since the location are known. With the locations of successfully paired descriptors reviewed by sift detector, lines can be easily drew between two successfully paired descriptors by using matlab built-in command. The diagram below shows the result.



Task-3 SIFT:

```
clc,clear;
[img, descriptor, locs] = sift('face90.jpg');          %find sift keypoints img1
showkeys(img, locs);
[img1, descriptor1, locs1] = sift('face0.jpg');          %find sift keypoints img1
showkeys(img1, locs1);
[img2, descriptor2, locs2] = sift('face45.jpg');         %find sift keypoints img2
showkeys(img2, locs2);
ratio = 0.6;                                             %define matching ratio
Pair = zeros(1,size(descriptor1,1));                     %create match pairs
for i = 1 : size(descriptor1,1)
   distance = dist(descriptor1(i,:),descriptor2');       %calculate Euclidean distance between
descriptors
   [value,index] = sort(distance);                       %sort distance
   if (value(1) < ratio * value(2))                      %closest match compare to sencond closest
match
      Pair(i) = index(1);                                %if valid collect the closest match, non-max
suppresion
   else                                                  %discard value if not the max value
      Pair(i) = 0;                                       %otherwise set 0
   end
end
imgAppend = imread('face_append.jpg');                   %load appended face iamge
figure;imshow(imgAppend);hold on;                        %show the imgAppend as background
for i = 1:10:size(descriptor1,1)                         %change the amount of line by spacing
  if (Pair(i) > 0)
    line([locs1(i,2) locs2(Pair(i),2)+size(img1,2)], ...  %cerate multiply lines array
         [locs1(i,1) locs2(Pair(i),1)], 'Color', 'g');   %x,y coordinate
  end
end
hold off;
```