ENGN6528 Computer Vision – S2, 2020
Computer Lab-2 (CLab2)

Objectives:

This is CLab-2 for ENGN6528. The goal of this lab is to help you become familiar with, and practice mid-level computer vision algorithms such as feature point detection, matching, image region segmentation, eigen-faces technique for face representation, detection and recognition.

Note, in this lab, you are free to chooseMatlab or Python and lab task descriptions are provided with both languages. If you have not used Matlab or Python before, this lab is an opportunity to get you quickly familiar with basic language usages and relevant libraries for image processing and computer vision. Please note that Python is now increasingly used in computer vision, and we encourage you to practise it in this course.

Special Notes:

1. Each Computer Lab task lasts for three weeks and has two lab sessions: session-A and session-B in the first two weeks. Tutors/Lab instructors will provide basic supervision to both sessions. The third week has no lab, which is for you to complete and submit the lab report.
2. Your lab will be marked based on the overall quality of your lab report. The report is to be uploaded to Wattle site before the due time, which is usually on the Sunday evening of the third week after the announcement of computer lab tasks. (e.g. Sunday of Week4 for Clab1).
3. Your submission includes the lab report in PDF format as well as the lab code that generates the experimental result.
4. It is normal if you cannot finish all the tasks within the two 2-hour sessions -- these tasks are designed so that you will have to spend about 9 hours to finish all the tasks including finishing your lab report. This suggests that, before attending the second lab session (e.g. the lab in Week3 for Clab1), you must make sure that you have almost completed 80%.

Academic Integrity

You are expected to comply with the University Policy on Academic Integrity and Plagiarism. You are allowed to talk with / work with other students on lab and project assignments. You can share ideas but not code, you should submit your own work. Your course instructors reserve the right to determine an appropriate penalty based on the violation of academic integrity that occurs. Violations of the university policy can result in severe penalties.

C-Lab-2 Tasks

Task-1: Harris Corner Detector (5 marks)

1. Read and understand the corner detection code 'harris.m' (harris.py for python).
2. Complete the missing parts, rewrite 'harris.m (py)' into a Matlab (python) function, and design appropriate function signature (1 mark).
3. Comment on line #13 in harris.m (correspondingly line #53 in harris.py) and every line of your solution after line #20 (line #60 in harris.py) (0.5 mark).
4. Test this function on the provided four test images (can be downloaded from Wattle). Display your results by marking the detected corners on the input images (using circles or crosses, etc) (0.5 mark for each image, 2 marks in total).
5. Compare your results with that from Matlab's built-in function *corner()* (correspondingly python function *cv2.cornerHarris()*) (0.5 mark), and discuss the factors that affect the performance of Harris corner detection (1 mark).

In your Lab Report, you need to list your complete source code with detailed comments and show corner detection results and their comparisons for each of the test images.

For Matlab users:

```
%%
% CLAB2 Task-1: Harris Corner Detector
% Your name (Your uniID)
%
sigma = 2; thresh = 0.01; % Parameters, add more if needed
% Derivative masks
dx = [-1 0 1;-1 0 1; -1 0 1];
dy = dx'; % dx is the transpose matrix of dy
% compute x and y derivatives of image
Ix = conv2(bw,dx,'same');
Iy = conv2(bw,dy,'same');

g = fspecial('gaussian',max(1,fix(3*sigma)*2+1),sigma);
Ix2 = conv2(Ix.^2,g,'same'); % x and x
Iy2 = conv2(Iy.^2,g,'same'); % y and y
Ixy = conv2(Ix.*Iy,g,'same'); % x and y

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task: Compute the Harris Cornerness                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task: Perform non-maximum suppression and                                %
%       thresholding, return the N corner points                           %
%       as an Nx2 matrix of x and y coordinates                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

For python users:

```
# Parameters, add more if needed
sigma = 2
thresh = 0.01

# Derivative masks
dx = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
dy = dx.transpose()
import matplotlib.pyplot as plt

bw = plt.imread('0.png')
bw = np.array(bw * 255, dtype=int)
# computer x and y derivatives of image
Ix = conv2(bw, dx)
Iy = conv2(bw, dy)

g = fspecial((max(1, np.floor(3 * sigma) * 2 + 1), max(1, np.floor(3 * sigma) * 2 + 1)), sigma)
Iy2 = conv2(np.power(Iy, 2), g)
Ix2 = conv2(np.power(Ix, 2), g)
Ixy = conv2(Ix * Iy, g)

##########################################################################
# Task: Compute the Harris Cornerness
##########################################################################


##########################################################################
# Task: Perform non-maximum suppression and
#       thresholding, return the N corner points
#       as an Nx2 matrix of x and y coordinates
##########################################################################
```

Task-2: K-Means Clustering and Color Image Segmentation (5 marks)

In this task, you are asked to implement your own K-means clustering algorithm for colour image segmentation, and test it on the provided two images. Please note that the PNG-type (Portable Network Graphics) images are in the 48-bit format, and you need to first convert them to 24-bit images.

1. Implement your own K-means function *my_kmeans()*.The input is the data points to be processed and the number of clusters, and the output is several clusters of points (in matlab you can use a cell array for clusters). Make sure each step in K-means is correct and clear, and comment on key code fragments (1.5 marks).
2. Apply your K-means function to color image segmentation. Each pixel should be represented as a 5-D vector {L*,a*,b*,x,y} where L* - lightness of the color; a* - the color position between red and green; b* - the position between yellow and blue; x, y - pixel coordinates. Please compare segmentation results 1) using different numbers of clusters (1 mark), and 2) with and without pixel coordinates (1 mark). Explain the reasoning behind using L*a*b* color representation instead of RGB for k-means clustering (0.5 mark).
   Hint: First convert the image from RGB to CIELAB color space to get L*,a*,b* values and consider the top-left pixel as the origin.
3. The standard K-means algorithm is sensitive to initialization (e.g. initial cluster centres/seeds). One possible solution is to use K-means++, in which the initial seeds are forced to be far away

from each other (to avoid local minimum). Please read the material http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf, and then implement K-means++ (Section 2.2 in the pdf) in your standard algorithm as a new initialization strategy and comment appropriately (0.5 mark). Compare the image segmentation performance (e.g., convergence speed and segmentation results) of this new strategy, with that of standard K-means, using different numbers of clusters and the same 5-D point {L*,a*,b*,x,y} representation from previous question (0.5 mark).

Task-3: Face Recognition using Eigenface (10 marks)

In this task, you are given the Yale face image dataset Yale−FaceA.zip. The dataset contains a *training_set* of totally 135 face images captured from 15 individuals (9 images from each individual). You are also given 10 test images in the *test_set* directory.

1. Unzip the face images and get an idea what they look like. Then take 10 different frontal face images of yourself, convert them to grayscale, and align and resize them to match the images in Yale-Face. Explain why alignment is necessary for Eigen-face (0.5 mark).

2. Train an Eigen-face recognition system. Specifically, at a minimum your face recognition system should do the following:

    a. Read all the 135 training images from Yale-Face, represent each image as a single data point in a high dimensional space and collect all the data points into a big data matrix. Display the mean face (0.5 mark).

    b. Perform PCA on the data matrix (1 mark). Given the size of the input image, direct eigen decomposition of covariance matrix would be slow. Read lecture notes and find a faster way to compute eigenvalues and vectors, explain the reason (1 mark) and implement it in your code (1 mark).

    c. Determine (by writing code) the minimum number of principal components required to capture 95% of the total variation of the data. Here the total variation is defined as the sum of all the eigenvalues. (1 mark)

    d. Determine the top-k principal components and visualize the top-k eigen-faces in your report (1 mark). You can choose k=12 for this and the rest of the steps.

    e. For each of the 10 test images in Yale-Face, read in the image, determine its projection onto the basis spanned by the top-k eigenfaces. Use this projection for a nearest-neighbour search over all the 135 faces, and find out which three face images are the most similar. Show these top-3 faces next to the test image in your report (1.5 marks). Report and analyze the recognition accuracy of your method (1 mark).

f. Read in one of your own frontal face images. Then run your face recognizer on this new image. Display the top-3 faces in the training folder that are most similar to your own face (0.5 mark).

g. Repeat the previous experiment by pre-adding the other 9 of your face images into the training set (a total of 144 training images). Note that you should make sure that your test face image is different from those included in the training set. Display the top-3 faces that are the closest to your face (1 mark).

Hints:

1. A simple way to do alignment is to manually crop (and rotate if necessary) the face region, resize the face image to a standard shape, and make sure the facial landmarks are aligned – e.g. centre of eyes, noses, mouths are roughly at the same positions in an image.
2. In doing eigen-decomposition, always remember to subtract the mean face and, when reconstructing images based on the first k principal components, add the mean face back in at the end.
3. You can use Matlab's/Python's functions for matrix decomposition and inverse matrix (e.g., *eigs(), svd(), inv()* for matlab and *numpy.linalg.eig(), numpy.linalg.svd(), numpy.linalg.inv()* for python) to implement PCA. Other than these, you should not use Matlab's/Python's built-in PCA or eigenface function if there is one.

========== END of C-Lab-2 ==========