ENGN6528 Computer Vision – 2021

Computer-Lab 2 (C-Lab2)

Objectives:

This is CLab-2 for ENGN6528 Computer Vision. The goal of this lab is to help you get familiar with, and practice middle-level computer vision algorithms: feature point detection, matching, image region segmentation. Practise Eigen-Face based technique for face representation, detection, and recognition.

Note that you can use **either Matlab or Python** for your Lab assignment. Feel free to choose Python if you are more comfortable with Python. If it is the first time that you are using python and you didn't have experience with python programming before, even you are taking the python programming course at the same time, **You are strongly recommended to use Matlab to accomplish the CLAB assignment 2**. The goal of this assignment to get yourself familiar with the computer vision algorithms we have discussed in the course NOT focus on improving your programming skills, althougth this lab is an opportunity to get you quickly familiar with basic language usages of Matlab/Python for image processing and computer vision.

Special Notes:

1. Each computer lab has three weeks. Tutors/Lab instructor will provide basic supervision to two sessions. Please attend both sessions to get support.
2. Your Lab will be marked based on the overall quality of your Lab Report (PDF). The report is to be uploaded to Wattle site before the due date, which is usually on Sunday evening of Week-3 session of your lab. We are going to setup turnit in for Lab report and code separately this time. Please submit your report and code correctly to wattle.
3. Your submission should include the lab report in PDF format as well as the Lab (source) code that generating the results in the report.
4. It is normal if you cannot finish all the tasks within two 2-hour sessions — these tasks are designed so that you will have to spend **at least 9 hours** to finish all tasks including finishing your Lab report. This suggests that, before the third lab session, you must make sure that you have almost completed 80% of all the questions.

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 dishonesty that occurs. Violations of the university policy can result in severe penalties.

CLab-2 Tasks

1 Tasks Harris Corner Detector. (6 marks)

For Matlab users:

1. Read and understand the corner detection code 'harris.m'.

2. Complete the missing parts, rewrite 'harris.m' into a Matlab function, and design appropriate function signature (1 mark).

3. Please provide comments on line #13 and every line of your solution after line #20 (0.5 mark). Namely, you need to provide short comment on your code, which should make your code readable.

4. Test this function on the provided four test images (Harris-[1,2,3,4].jpg, they 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).
   Please make sure that your code could be run successfully on a local machine and generate results. If your submitted code cannot replicate your results, you may need to explain and demonstrate the results in person to tutors and the lecturer.

5. Compare your results with that from Matlab's built-in function *corner()* (0.5 mark), and discuss the factors that affect the performance of Harris corner detection (1 mark).
   It means you should not use the matlab inbuilt function* corner() *in your own implementation

6. Test this function on 'Harris-5.jpg' (Harris-5.jpg can be downloaded from wattle.). Analyse the results why we cannot get corners by discussing and visualising your corner response scores of the image. (0.5 mark)
7. Test this function on 'Harris-6.jpg' (Harris-6.jpg can be downloaded from wattle.). Plot your harris corner detector results in your report and propose a solution to obtain the salient corners which is robust to noise. (0.5mark)

Listing 1: harris.m

```
%%
% 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:

In [1]:
```
"""
CLAB2 Task-1: Harris Corner Detector
Your name (Your uniID):
"""
import numpy as np
```

In [2]:
```
def conv2(img, conv_filter):
    # flip the filter
    f_siz_1, f_size_2 = conv_filter.shape
    conv_filter = conv_filter[range(f_siz_1 - 1, -1, -1), :][:, range(f_siz_1 - 1, -1, -1)]
    pad = (conv_filter.shape[0] - 1) // 2
    result = np.zeros((img.shape))
    img = np.pad(img, ((pad, pad), (pad, pad)), 'constant', constant_values=(0, 0))
    filter_size = conv_filter.shape[0]
    for r in np.arange(img.shape[0] - filter_size + 1):
        for c in np.arange(img.shape[1] - filter_size + 1):
            curr_region = img[r:r + filter_size, c:c + filter_size]
            curr_result = curr_region * conv_filter
            conv_sum = np.sum(curr_result)  # Summing the result of multiplication.
            result[r, c] = conv_sum  # Saving the summation in the convolution layer feature map.

    return result
```

In [3]:
```
def fspecial(shape=(3, 3), sigma=0.5):
    m, n = [(ss - 1.) / 2. for ss in shape]
    y, x = np.ogrid[-m:m + 1, -n:n + 1]
    h = np.exp(-(x * x + y * y) / (2. * sigma * sigma))
    h[h < np.finfo(h.dtype).eps * h.max()] = 0
    sumh = h.sum()
    if sumh != 0:
        h /= sumh
    return h
```

In [4]:
```
# 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()

# computer x and y derivatives of image
Ix = conv2(bw, dx)
Iy = conv2(bw, dy)
```

In [5]:
```
g = fspecial((max(1, np.floor(3 * sigma) * 2 + 1), max(1, np.floor(3 * sigma) * 2 + 1)), sigma)
```

In [6]:
```
Iy2 = conv2(np.power(Iy, 2), g)
Ix2 = conv2(np.power(Ix, 2), g)
Ixy = conv2(Ix * Iy, g)
```

In [7]:
```
######################################################################
# 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
######################################################################
```

1. Read and understand the above corner detection code (page 3). Note that we have separated code in different parts/blocks (which could be referred easily by its block number in later questions.)
2. Complete the missing parts, rewrite them to 'harris.py' as a python script, and design appropriate function signature (1 mark).
3. Comment on the codes in block #5 (corresponding to line #13 in "harris.m", namely the Matlab version) and every line of your solution in block #7 (0.5 mark).
   Namely, you need to provide short comments on your code, which should make your code readable.
4. Test this function on the provided four test images (Harris-[1,2,3,4].jpg, they 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).
   Please make sure that your code could be run successfully on a local machine and generate results. If your submitted code cannot replicate your results, you may need to explain and demonstrate the results **in person to tutors and the lecturer.**

5. Compare your results with that from python's built-in function *cv2.cornerHarris()* (0.5 mark), and discuss the factors that affect the performance of Harris corner detection (1 mark).
6. Test this function on 'Harris-5.jpg' (Harris-5.jpg can be downloaded from wattle.). Analyse the results why we cannot get corners by discussing and visualising your corner response scores of the image. (0.5 mark)
7. Test this function on 'Harris-6.jpg' (Harris-6.jpg can be downloaded from wattle.). Plot your harris corner detector results in your report and propose a solution to obtain the salient corners which is robust to noise. (0.5mark)

In your Lab Report, you need to list your complete source codes with detailed comments and show corner detection results and their comparisons for each of the test images. Namely, you can take a snapshot of your code with comments and insert it in the report. You also need to submit your source code.

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 following **two images** as shown in Fig.1(you can download them from wattle). Please first convert one image (in PNG-type (Portable Network Graphics)) of 16bit to 8bit image before the following process.

Fig. 1

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 your data (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 that encodes: (1) L^* - lightness of the color; (2) a^* - the color position between red and green; (3) b^* - the position between yellow and blue; (4) x, y - pixel coordinates.
   Please compare segmentation results
   (1) using different numbers of clusters (1 mark).
   [It means you need to visualise results using different number of clusters.]

   (2) and with and without pixel coordinates (1 mark).

   [It means you need to visualise results for cases in which you only use [L, a, b*] as features or you use [L, a*,b*, x, y] as features for clustering.]*

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, summarize the key steps in the report (0.5 mark), and then implement K-means++ in your standard algorithm as a new initialization strategy (0.5 mark). Compare the image segmentation performance *(e.g., in terms of convergence speed and segmentation results [by plotting the segmentation results in the report])* of this new strategy, with that of standard K-means, using different numbers of clusters and the same 5-D point representation, **namely [L, a*, b*, x, y,] (L, a, b , pixel coordinates)** as that from previous question (0.5 mark).

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. Please unzip the face images and get an idea what they look like. Then please 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 least your face recognition system should be able to complete the following tasks:

   (1) 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.

   (2) Perform PCA on the data matrix (1 mark), and display the mean face (1 mark). Given the size and the large number of input images, directly performing eigen value decomposition of the covariance matrix would be slow. Please read lecture notes and

find a faster way to compute eigen values and vectors, explain the reason (1 mark) and implement it in your own code (1 mark).

(3) Determine the top k principal components and visualize the top-k eigen- faces in your report (1 mark). You can choose k=10 or k=15.

(4) For each of the 10 test images in the Yale-Face dataset, please read in an image as the reference one, and determine its projection onto the basis spanned by the top k eigenfaces. Use this projection as feature to perform a nearest-neighbour search over all 135 faces, and find out the top three face images that are most similar to the reference one.

Show these top 3 faces next to the test image in your report (1.5 marks). Please report and analyze the recognition accuracy of your method (1 mark).

(5) Read in one of your own frontal face images. Then run your face recognition system on this new image. Display the top 3 faces in the training folder that are most similar to your own face (1 mark).

(6) Repeat the previous experiment by pre-adding the other 9 additional images of your face into the training set (a total of 144 training images).

[It mean you add yourself as an additional person to the training set with 9 images of yourself. It leads to 16 different people in the dataset and each of them having 9 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 value-decomposition, please always remember to subtract the mean face and, when reconstructing images based on the first k principal components, add the mean face back 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. For example, you **should not** use either Matlab built-in function, such as *pca(), or cv2.PCACompute() in opencv, python.*