# ENGN4528/6528 Computer Vision – 2018

# Computer-Lab-2 (CLab-2)

(final version)

**Objectives:**

Goal of this lab is to help you familiar with, and practice middle-level computer vision algorithms: feature point detection, matching, image region segmentation.

**There are 3 tasks in CLab-2.**

- Task-1: Implement your own Harris Corner Detector. **(6 Marks)**
- Task-2: Implement your own K-means colour image segmentation. **(5 Marks)**
- Task-3: Play with SIFT-descriptor, and implement a SIFT-based image matching program. **(4 Marks)**

Your final marks for CLab-2 will be determined based on both CLab-2 Report.

**Expected workload:** totally 6 hours for this Lab including Report-writing time, excluding the required time for textbook reading, and for lecture-note reviewing. If you cannot complete all tasks in time, you only need to report whatever you have completed.

**====== Begin of Task Descriptions =====**

## Task-1: Implement your own Harris Corner Detector

Step-1: Read and understand the following corner detection code.

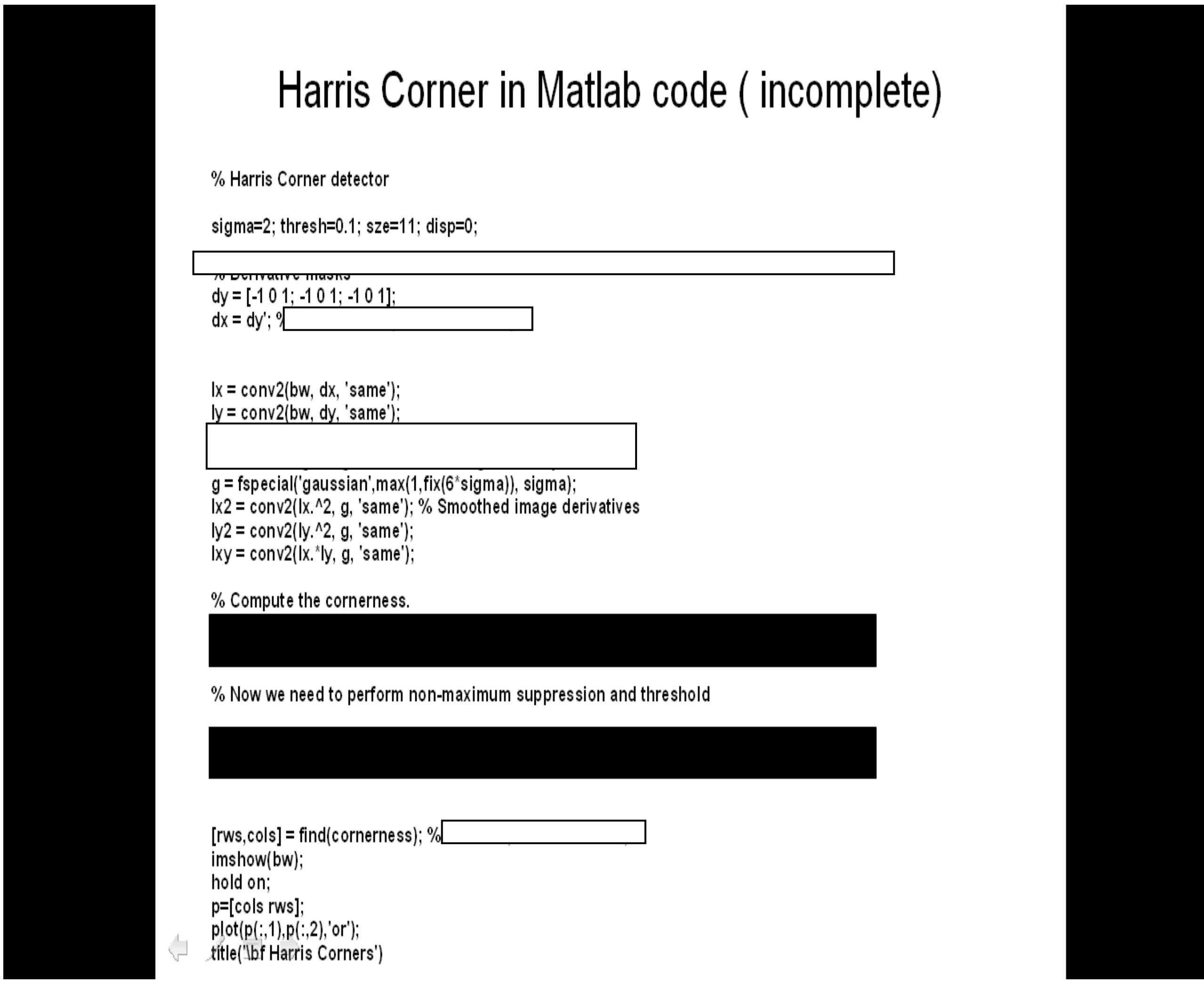
Step-2: Complete the missing part, make it a Matlab function.

Step-3: Comment on every line of the code.

Step-4: Test this function on the provided five test images (on Wattle).

Step-5: Display your results by overlaying the detected corners onto the image.

Step-6: Compare your results with that by using Matlab's inbuilt corner () function.



## Harris Corner in Matlab code ( incomplete)

```
% Harris Corner detector

sigma=2; thresh=0.1; sze=11; disp=0;

dy = [-1 0 1; -1 0 1; -1 0 1];
dx = dy'; %

Ix = conv2(bw, dx, 'same');
Iy = conv2(bw, dy, 'same');

g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);
Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed image derivatives
Iy2 = conv2(Iy.^2, g, 'same');
Ixy = conv2(Ix.*Iy, g, 'same');

% Compute the cornerness.
[redacted]

% Now we need to perform non-maximum suppression and threshold

[redacted]

[rws,cols] = find(cornerness); %
imshow(bw);
hold on;
p=[cols rws];
plot(p(:,1),p(:,2),'or');
title('\bf Harris Corners')
```

In your Lab Report, you need to list your completed source code, with detailed comments, and the corner detection result comparison for each of the test images.

## Task-2: Implement your own k-means clustering function, and use it for colour image segmentation

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:

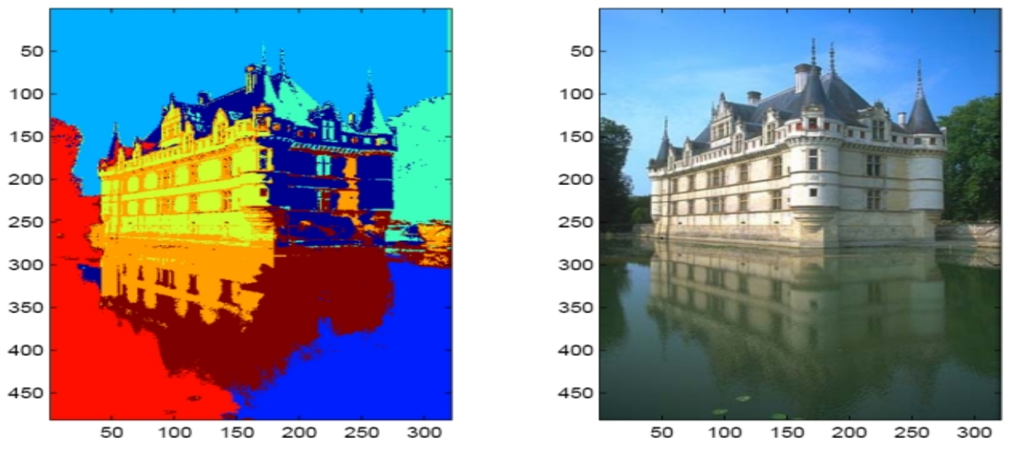
(1) Peppers.png (Please note: this png image is in 48-bit format (16x3=48), you need

to first convert it to a 24-bit colour image first. )

(2) mandm.png

In doing this task, you must implement your own my_Kmeans function; you MUST NOT use Matlab's kmeans() function.

The following example illustrates a typical k-means segmentation result, where the feature vectors used were [ L, a*, b*, x, y,] ( L, a, b , pixel coordinates). Your segmentation results should look similar.



Some supporting Matlab functions are provided below, but some key lines are missing, and you need to complete them.

Your task is to read, understand and complete the codes, make it work on your test images. Note: your solution must be based on the provided supporting functions.

**Your program can be organised in the following steps:**

**Step-1: read in an input colour image.** ( If it is not a 24-bit RGB image, then you need to convert it to a 24-bit RGB image.)

```
img = imread('mandm.png');

imshow(img), title('Input colour image');
```

**Step-2: convert the image to the La*b* colour space:**

```
cform   =  makecform('srgb2lab');
lab = applycform(img, cform);
%% L_Image = lab(:, :, 1);  % this is the L channel image.
%% A_Image = lab(:, :, 2);  % this is the a channel image.
%% B_Image = lab(:, :, 3);  % this is the b channel image.
```

**Step-3: Form 5-dimensional feature vector:**

Use the following provided supporting function im2feature(). (Read and understand the function; if you find bugs or typos in the provided code (, which I am not sure), please fix them.

********************

```
 function [features] = im2feature(img)

[rows, cols, ncolors] = size(img);

npixels = rows * cols;

% Each feature vector consists of [ L,a,b,x,y]

[x,y] = meshgrid(1:cols,1:rows);

features = img;

factor = 1.0 ;     % In your experiment, you may also change the value of 'factor' to e.g. 10, or 0.1, to see whether there is any change to your final results.

features(:,:,4) =  factor* x;

features(:,:,5) =  factor* y;

features = reshape( features, [npixels 5] );

% Normalize the features

for i=1:size(features,2)

    % Zero mean

    features(:,i) = features(:,i) - mean(features(:,i));

   % Normalize

    features(:,i) = features(:,i) / norm(features(:,i));

end
```

********************

**Step-4: implement your own k-means algorithm by completing the following supporting function ( a matlab file "my_kmeans.m" is provided on Wattle):**

function [data_clusters, cluster_stats] = my_kmeans( data, nc )

% This function performs k-means clustering on data , given (nc) = the number of clusters.

**Step-5: display the segmentation result, by calling the following supporting function:**

Function displayclusters( img, clusters )

**Step-6: End.**

## Task-3: Play with SIFT code, and implement an image match algorithm

Step-1: Read in one of your frontal face images, and generate three rescaled and rotated versions.

For example, you may rotate your image by 5 degrees, 15 degrees, and 45 degrees, and rescale them by a scale-factor of 1.2, 1.4, and 0.8, respectively.

Save all the three new images to your local directory.

Step-2: Read, understand and test the provided SIFT.m code on your three images.

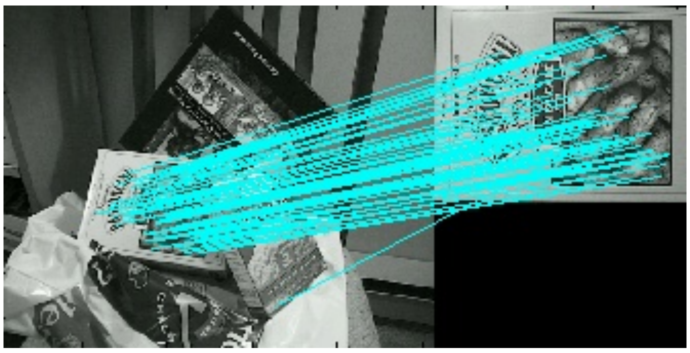
Use showkey.m to display the detected SIFT descriptors on each image.

(Note: you need to copy all the provided files to your local directory. )

Step-3: Display two images side-by-side on a 1x2 image panel. Compute their SIFT feature points, find feature matches between the two images, and draw lines connecting the matched SIFT feature points.

Hint: In order to find matches , you need to exhaustively test for every SIFT feature in the first image, and try to find its best match in the second image. This is done by comparing the Euclidean distance between the two 128-dimensional SIFT feature descriptors. Two SIFT descriptors are found to be a "matching pair" if and only if their distance is less than a ratio times the distance to its second closest match (, i.e., for the sake of non-maximum suppression). This ratio is usually set between 0.6--0.8.

In the step of drawing lines, in order to avoid cluttered visualisation, you may only show the lines for the first 10-15 SIFT matches. An example SIFT matching result is shown below.



**====== End of all 3 Tasks in CLab-2 =======**

**Lab-Report Requirement for C-Lab-2:**

Please use the same Lap-Report template as you used for CLab-1 report.

Upload **a single PDF file** (with file name: CLab-2-Report-Uxxxxxx.pdf) containing the following contents:

(a) Sample results from each of the Tasks, with necessary comments, descriptions.

(b) Answer any technique questions in the Task instructions.

(c) Attach at the end of the PDF file your Matlab source code for each tasks. The source codes must be carefully commented. (In some cases, if your code is very long, then you may only provide the main part (core part) of your matlab code. You may leave out other supporting code.).

**Note: Please do not upload any ZIP file, or any Matlab source file (i.e. ".m") to Wattle site. A single PDF file (containing everything) is all that you need to upload to Wattle.**

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