CSCI 4830/5722 Computer Vision, Spring 2019
Instructor: Fleming
Homework 3, Due Saturday, March 2nd, by 11:00 pm

Problem Set

Q1. Show that the following is True or False:
If the zero vector is part of a set of vectors that set is dependent.

Q2. Show that the following is True or False:
If a set of vectors is dependent so is any larger set which contains it.

Q3. What is separability? How does separability affect computational complexity?

Q4. Show that convolving an image with a discrete, separable 2D filter kernel is equivalent to convolving with two 1D filter kernels. Estimate the number of operations saved for an NxN image and a $(2k + 1) \times (2k + 1)$ kernel.

Q4. What happens when we convolve a Gaussian with another Gaussian?

Q5. What is Dimensionality Reduction and how is it important with respect to Image Processing? Elaborate on one advantage and one disadvantage.

Q6. Show that for a 2×2 matrix A, with eigenvalues $\lambda 1$, $\lambda 2$
(a) $\det(A) = \lambda 1 * \lambda 2$

(b) $\mathrm{trace}(A) = \lambda 1 + \lambda 2$

Q7. MATLAB Programming: write a script that compares the performance of a 3x3 mean filter, a 3x3 median filter and Gaussian filters with different values of σ. Get familiar with the following MATLAB functions: `imnoise`, `medfilt2`, `conv2`, `filter2`, `fspecial`, `imfilter`, and `edge`.

A. Your script should load image *peppers.png*, convert it from RGB to grayscale (use function `rgb2gray`), and initially add *salt & pepper* noise to the image. Apply five filters to the image:
   - a 3x3 mean filter
   - a 3x3 median filter

- a Gaussian filter with σ = 1/3 pixel
- a Gaussian filter with σ = 1 pixel
- a Gaussian filter with σ = 1.5 pixels

Plot each one of the images and comment on what works best.

B. Repeat the same steps, but instead of *salt & pepper* noise, add Gaussian white noise with mean 0 and σ = 1 in the [0, 255] range (or σ = 1/256 in the [0, 1] range). Apply five filters to the image:
   - a 3x3 mean filter
   - a 3x3 median filter
   - a Gaussian filter with σ = 1/3 pixel
   - a Gaussian filter with σ = 1 pixel
   - a Gaussian filter with σ = 1.5 pixels

   Plot each one of the images and comment on what works best.

Q6. MATLAB Programming: write your own implementation of the Harris Corner Detector algorithm.

A. Write a function `harris` that implements the Harris Corner Detector:

```
[corner_coords, descriptors] = harris(I, w, threshold,
suppression)
```

   Follow the steps outlined in lecture (Lecture 15, recording and slides on Moodle). Your function will accept four input arguments:
   - `I` - an RGB image
   - `w` - the width of the Gaussian filter window, used initially for de-noising. Please note that the algorithm you will implement uses a w(u,v) Gaussian window function, where `W(x,y)` is a `w x w` size neighborhood around pixel at coordinates `(x,y)`.
   - `threshold` - the threshold necessary for separating the values of the cornerness function R
   - `suppression` - a boolean which determines if non-maximum supression is applied

   The function should return two variables:
   - `corner_coords` - a matrix with 2 columns, representing the `(x,y)` coordinates of the feature points selected.

- `descriptors` - a matrix with 9 columns, representing the pixel intensities (in grayscale) of a 3 x 3 patch around each feature point.

Plot the results of the corner detection, with and without non-maximum suppression.. You can use any image of your choosing, for example one of the images used for the image stitching assignment.

B. Now test how well your corner detector behaves under rotation, translation, and scale of the image. You can do this by a simple exercise in matching. Take one test image (you can use any image of your choosing) and prepare a rotated, translated, and scaled version of that image.

   Plot the results of the corner detection in each image and comment on how well the features in the original image have matching features in the rotated, translated and scaled versions.

C. Repeat the test at point B. This time, compare the results of the corner detector in the original image with four new versions of it:
   - Prepare a brighter version of the original image by *adding* a constant *positive* offset to all pixel values
   - Prepare a darker version of the original image by *adding* a constant *negative* offset to all pixel values
   - Prepare a brighter version of the original image by *multiplying* a constant *positive* offset to all pixel values
   - Prepare a darker version of the original image by *multiplying* a constant *negative* offset to all pixel values

   Plot the results of the corner detection in each image and comment on how well the features in the original image have matching features in the four new versions.