CS83/183, Fall 2015, Problem Set # 2

October 20, 2015

Due: November 3, 11:59pm

This problem set requires you to provide written answers to a few technical questions. In addition, you will need to implement some learning algorithms in MATLAB and apply them to the enclosed data sets. The questions requiring a written answer are denoted with **W**, while the MATLAB programming questions are marked with **M**.

You must provide your written answers in paper form and leave them in the course mailbox located near the lobby of the Sudikoff building (the mailbox is tagged with the label CS83/183). Since each problem will be graded by a different person, we ask that you provide a separate submission for each individual problem. For example, for problem 3 of this homework assignment you should turn in a single page or a set of pages stapled together containing your answers to questions 3(a) and 3(e). **Please use the cover pages included at the end of this document. For each problem, fill out the corresponding cover page by writing your name, date and time of submission and use it as front page for your problem submission. A penalty of 5 points will be applied if you fail to follow these instructions.**

Instead, your MATLAB code must be submitted via Canvas. We have provided MATLAB function files with an empty body for all the functions that you need to implement. Each file specifies in detail the input and the output arguments of the function. A few of the functions include some commands to help you get started with the implementation. We have also provided scripts that check the sizes of the inputs and outputs of your functions. Do not modify these scripts. In order to allow us to test your code, it is imperative that you do not rename the files or modify the syntax of the functions. Make sure to include all the files necessary to run your code. **If we are unable to run one of your functions, you will receive 0 points for that question.** Please place all your files (data, .fig files automatically generated by the provided scripts, and function files) in a single folder (**without subfolders**). Zip up the entire folder, and upload the compressed file to Canvas.

Please do not use functions from Matlab toolboxes (e.g., the Image Processing Toolbox or the Optimization Toolbox) for your homework code unless instructed to do so. If in doubt, please ask. While student discussion is permitted, each student must write his/her homework solution and code *independently.*

1. **[W, 10 points]** Prove that normalized correlation is invariant to affine intensity changes. In other words show that, if two images I, I' are such that $I'(x, y) = aI(x, y) + b$ where a, b are arbitrary fixed constant values for all pixels (x, y), then the normalized correlation of I and I' with an arbitrary filter f produces response maps R and R' that are identical, i.e., $R = R'$.

2. **[W, 10 points]** *This problem is for graduate credit only: if you are enrolled in COSC83 (instead of COSC183) you do not need to solve this problem.*
Show that the correlation of an image I with a filter f, i.e., $f \otimes I$, can be expressed as a single matrix-vector product. Please explain in detail what the matrix and the vector represent in terms of I and f (e.g., discuss what are the entries of each row of the matrix). Credit is given only for a complete and rigorous explanation.

3. **[35 points]** The enclosed file `smme.m` computes the maximum and minimum eigenvalues of the second moment matrix for an input image. The orientation of the eigenvector associated to the maximum eigenvalue is also returned.

   (a) [**W**, 5 points] How can you use these eigenvalues to do edge detection, rather than corner detection? The set of detected edges should not include corners. For this part, you don't have to do any programming yet. Just explain how you would do it.

   (b) [**M**, 8 points] Write a MATLAB function `FindEdges` that finds edges based on this idea. Please use the following syntax for your function:
   `[Es, En] = FindEdges(I, thresh, n)`
   where `I` is the input grayscale image represented as a $M \times N$ matrix, `thresh` is a 2×1 vector containing the thresholds required by your function, `n` is the parameter passed as second argument to `smme` (which should obviously be called by your function), `Es` is a $M \times N$ matrix of doubles containing the edge strength at every pixel, and `En` is a $M \times N$ matrix encoding at each pixel the edge *normal* direction (i.e., the direction orthogonal to the edge) as a value in $[0, \pi]$. The edge strength should be nonzero only at edge pixels (i.e., it should be zero at corner points or non-edges). Your function should detect edges merely based on an analysis of the eigenvalues of the second moment matrix and should not perform any postprocessing, such as non-maximum suppression or hysteresis thresholding. After you have coded your function run the script `q3b.m` to visualize the edge detection results obtained for the input image `venice.pgm`. The script uses parameter values `n=5` and `thresh=[130 30]`. You are encouraged to leave these parameters unchanged as they have been chosen to produce good edge detection results. You should be able to find a way to use them effectively in your function...

   (c) [**M**, 9 points] Now implement the function
   `Enms = nonmaxsuppression(Es, En)`,
   which performs non-maximum suppression (without hysteresis thresholding). The function receives as input the edge strength `Es` and the edge normal `En` as computed by the function `FindEdges`.

   Use the following pseudocode as a reference to implement a simple version of non-maximum suppression. Let E_s be the edge strength image and E_n the edge normal image. Define $d_1, \ldots, d_4$ to be the $0°, 45°, 90°$, and $135°$ discretized edge normal directions. Let E_{nms} denote the output of non-maximum suppression. For each pixel (i, j) such that $E_s(i, j) > 0$, perform the following steps:

   i. find the direction d_k that best approximates the edge normal $E_n(i, j)$;
   ii. if $E_s(i, j)$ is smaller than at least one of its two neighbors along d_k, set $E_{nms}(i, j) = 0$; otherwise assign $E_{nms}(i, j) = E_s(i, j)$.

After you are done implementing your function, run the script `q3c.m` to see the results. Your detected edges should now appear as boundary segments having a thickness of 1-pixel.

(d) [**M**, 8 points] Now implement the function `FindCorners` based on a simple analysis of the maximum eigenvalue of the second moment matrix (again, your function should invoke `smme`). Note that `thresh_scalar` is a scalar since, unlike in the case of edge detection, you only need a single threshold to perform corner detection. Your function should return a matrix that is non-zero only at corner points where it should give a measure of the cornerness of the point (the corner strength). After you are done with your function run the script `q3d.m` to see the results. Again, you should not have the need to change the value of `thresh_scalar` (set to 120) in order to obtain good corner detection.

(e) [**W**, 5 points] Now try running the script `q3e.m` which will run your edge and corner detection functions on the original Venice photo `I` but also on a smoothed version of this image obtained with the command `gaussn(I, 15)` where `gaussn` applies a Gaussian filter having size specified by the second argument. Compare and comment qualitatively the two results in terms of the edges and corners that were produced. You should also explain why the two sets of edges (and corners) differ so significantly. When you comment the results ignore the border effects.

4. **[M, 45 points]** In this exercise, you will implement an algorithm to create a mosaic from the pair of input images `('church1.jpg', 'church2.jpg')`.

(a) [**M**, 6 points] Write the function `CollectGTpoints` to manually collect corresponding ground truth points from two images using the MATLAB `ginput` function. Given the two images as input, your function should visualize them side by side as a single photo. It should then interactively request the user to click (with the left mouse button) on a point of the left image, and then on the corresponding point in the second picture. It should visualize each correspondence after it has been clicked by the user. It should continue this manual collection of corresponding points until the user clicks the right mouse button. The function should return the N corresponding points in two $N \times 2$ matrices `P1` and `P2` where N is the number of correspondences and each row contains the x and y coordinates of a point, in this order. Thus `P2(i,:)` should be the right-image point corresponding to `P1(i,:)` of the left image. Now run the script `q4a.m` which will invoke your function and save your correspondences in the file `q4a.mat`. Pick visually distinctive points (i.e., good feature points) as doing so will simplify your task in part (d). Your set of correspondences should be large enough to enable *robust* estimation of the homography matrix. Thus it should be larger than the minimum number of correspondences needed to estimate the parameters. If you are unhappy about your selection of points, simply delete the file `q4a.mat` and rerun your script `q4a.m`. Please make sure to include your final correspondence file `q4a.mat` in the zip file of your homework submission as it will be used to test your functions.

(b) [**M**, 10 points] Write the function `H = ComputeHomography(P1, P2)` that takes a set of corresponding image points as input and outputs the associated 3x3 homography matrix. As discussed in class, in order to calculate the homography you need to solve a homogeneous least squares system $A\mathbf{h} = \mathbf{0}$, where $\mathbf{h}$ is a vectorized version of the homography

matrix. The solution can be obtained by calculating the singular value decomposition of A. The vector $\mathbf{h}$ will be given by the singular vector corresponding to the smallest singular value. You can compute it in MATLAB as follows: `[U,S,V]=svd(A); h = V(:,end);`. If you implemented things correctly you should now be able to run the script `q4b.m` which uses your correspondences selected in part (a) to compute the homography matrix and create a mosaic of the two images by warping one image onto the other.

(c) [**M**, 5 points] So far we have been relying on manually selected correspondences. Now we want to use computer vision to automatically establish correspondences between your two sets of points `P1` and `P2`. Thus, from now on we will stop assuming that the correct correspondences are already given by the order of points. In order to establish correspondences, we need feature descriptors associated to the points. Write a function `[F1, F2] = ExtractPatches(P1, P2, I1, I2, w)` that extracts patches of size $w \times w$ centered at points `P1` and `P2` from images `I1` and `I2`, respectively. For this purpose, convert `I1` and `I2` to grayscale images (using `rgb2gray`) inside your function `ExtractPatches` and form descriptors simply by "flattening" the intensity values in each patch to one-dimensional vectors. Thus, the output arguments `F1`, `F2` should be matrices of size $N \times w^2$, where N is the number of points in `P1` and `P2`. Because P1 and P2 have real-valued entries, round these coordinates to obtain pixel (i.e., integer) coordinates for the centers of the patches.

(d) [**M**, 7 points] Write a function
`[D, i, j] = EstablishCorrespondences(F1, F2, thresh)`
that finds matches between the two sets of feature descriptors `F1` and `F2`. In order to do so your function must compute the Euclidean distance between every descriptor in `F1` and every descriptor in `F2`. Store these distances in the $N \times N$ output matrix `D` where `D(m,n)` denotes the Euclidean distance between `F1(m,;)` and `F2(n,;)`. Your function should select automatically the correspondences by thresholding these pairwise distances using the scalar argument `thresh`. The correspondence indices should be stored in the output arguments `i`, `j`, which are $n \times 1$ vectors, with the interpretation that the pair `(i(k), j(k))` defines the k-th correspondence indicating that `F1(i(k),:)` matches `F2(j(k),:)`. Now run the script `q4d.m`. If you have coded `ExtractPatches` and `EstablishCorrespondences` properly it should show you the automatically selected correspondences. Chances are that some of the correspondences will be incorrect due to feature ambiguity (we are simply using the flattened grayscale patches, which are a bad feature representation). Don't worry about it, you'll have a chance to fix this problem via RANSAC.

(e) [**M**, 17 points] In this part you will implement RANSAC to robustly estimate the homography matrix from noisy correspondences. Your method will be tested on the automatically established correspondences from part (d). However, if all these matches are correct (which is unlikely), then our script `q4e.m` will automatically add 5 incorrect correspondences to mimic typical correspondence error. Our script will show the result of computing the homography and the mosaic based on the incorrect set of correspondences.

Now implement the function to estimate the correct correspondences and homography via RANSAC from noisy correspondences. It should have the following syntax:
`[HRANSAC, icorrect, jcorrect] = ComputeHomographyRANSAC(P1, P2, inoisy, jnoisy, s, niter, ithresh)`.
This function should obviously exploit your function `ComputeHomography` from part (b)

in order to compute the homography matrix for each candidate subset of correspondences. Note that each candidate homography should be tested over the complete sets of N points `P1` and `P2` in order to count the number of inliers (see additional notes in the header of the function stub that we have provided). After you have coded your function, run the script `q4e_ransac.m`. It will invoke your RANSAC function on the noisy correspondences and use your estimated homography matrix to generate the mosaic. The parameters `s` (the sample size), `niter` (the number of RANSAC iterations, and `ithresh` (the distance threshold to declare inliers) have been carefully chosen for you in our script so that, with very high probability, your function should return the correct homography matrix and find the correct set of correspondences. Don't change these parameter values.

HW2 | CS83/183, Fall 2015

Problem 1

First name _____ **Last name** _____

Date and time of submission _____

HW2 | CS83/183, Fall 2015

Problem 2

First name _____ **Last name** _____

Date and time of submission _____

HW2 | CS83/183, Fall 2015

Problem 3

First name _____ **Last name** _____

Date and time of submission _____