HW3 – submission 16.1.18 23:55

Guide lines

1. You should submit all function and script files written in MATLAB/Python. Your code should be well documented and clear. The code should run from **any** computer and include all path definitions (You should take care of this in the code).
2. Please divide the code by questions.
3. Final report – should include explanations on the implementation and the execution, answers to the questions, results, conclusions and visual results. Do elaborate on all parts of the algorithms/solution. **Please submit a PDF file and not a DOC file.**
4. Please post question regarding this HW on the facebook group (English only): https://www.facebook.com/groups/294298727746314/
5. The grades are highly depended upon the analysis depth of the report.
6. HW could be submitted in pairs.
7. Please include your ID in the report.
8. Eventually submit one compressed file including the code + images PDF.

Good luck!

Question 1 – Eigen-Faces and PCA for face recognition (30 points):

In this exercise, you will implement the Eigen-faces algorithm for face recognition. Use the provided stub code (`eigenface.m`) and implement the algorithm according to following steps:

1. Prepare the training set:
   a. The training set is loaded using the function `readYaleFaces.m` (line 3 on the given code). Read the documentation for more details.
   b. Compute the mean face of the trainset and subtract it from all the training images.
   c. Show an image representing the mean face.

2. Compute the eigen-faces:
   a. Implement a function that finds the r largest eigen-vectors of the trainset covariance matrix:
   $$\Sigma = E\left[\left(X-\mu\right)\left(X-\mu\right)^T\right]$$
   To avoid memory issues, use SVD as describes in the tutorial, or read the relevant section in [1]. Implement your algorithm accordingly and explain in detail.
   b. Show the **first** five eigen-faces as images (corresponding to the **largest** eigenvalues).
3. Reconstruction:
   a. For each image in the train set $\{x_j\}$, subtract the average face (calculated in 1.b) and calculate its low dimension representation vector $y_j = W^T\left(x_j - \bar{x}\right)$ according to the $r = 25$ largest eigen-faces (W is a matrix build from the top r eigen-faces).
   b. Reconstruct each of the images in the train set $\{x_j\}$ according to its representation y_j. Compute the following representation error:
   RMSE error – The average error per pixel in absolute value and in percent (out of 256 gray levels). Please scale the reconstructed image to have values in the range of [0,255].
   c. Describe and explain your results. What is the average representation error? Could we have foreseen it (hint: eigenvalues)?
4. Recognition:
   a. For each image in the **Test set** $\{x_i\}$, find the representation vector y_i according to the $r = 25$ top eigen-faces (don't forget to subtract the train set average face). What is the mean representation error? Was there any change in comparison to the train set? Explain. Demonstrate a good reconstruction and a bad one.
   b. Classify each image from the test set (use only the images of people who appear also on the train set – use the face_id parameter). This can be done by using nearest neighbor classifier between each y_i to the train set represent vectors $\{y_j\}$. For nearest neighbor classification, use the matlab function `fitcknn.m`. What are the success ratios (for face_id)?
   c. Explain why each of the unsuccessful classifications failed
   d. Propose an algorithm that will improve the results obtained here. (tip: use the eigenfaces as an input to the new algorithm)

References:

[1] Turk, Matthew, and Alex Pentland. "Eigenfaces for recognition." *Journal of cognitive neuroscience* 3.1

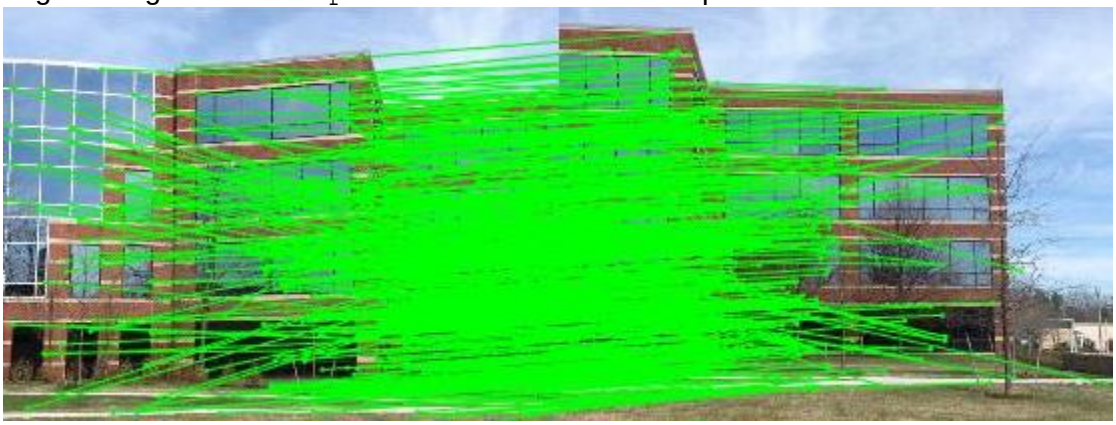
[2] A Tutorial on Principal Component Analysis - Jonathon Shlens

Question 3 – Image Stitching (70 points):

In this exercise you will implement an image stitching algorithm, based on SIFT descriptors. You are provided with an implementation of SIFT (Created by Andrea Vedaldi UCLA Vision Lab - Department of Computer Science, University of California).
In this exercise, the following Matlab function are **prohibited**:

cp2tform, imtransform, tformarray, tformfwd, tforminv, maketform, imwarp or any other matlab function which calculates a transformation or warping.

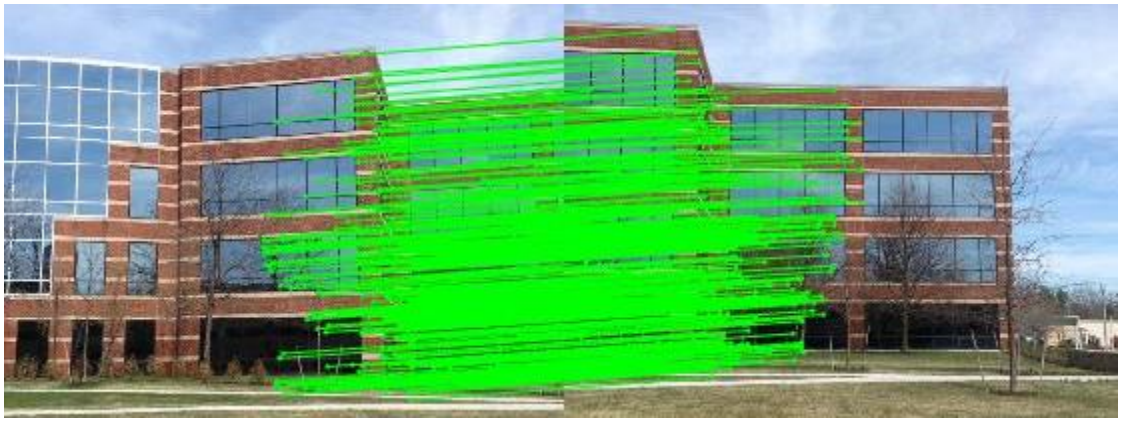
1. Choose a single pair (two consecutive images) for the rest of this work from the folder "building".
2. For each image (in the chosen pair), convert to greyscale and extract its SIFT features and descriptors, using the function `sift.m`.
3. Find matching key-points: use the function `siftmatch` which gets two sets of SIFT descriptors (corresponding to two images) and return matching key points. Show all matching key-points on the colored images using the function `plotmatches`. For example:



4. Implement a function that gets a set of matching points between two images and calculates the **projective** transformation between them. The transformation should be H_{3x3} homogenous matrix such each key-point in the first image, p, will map to its corresponding key-points in the second image, $\tilde{p}$, according to $p = H\tilde{p}$. How many points are needed?
5. Find all the inliers: Implement a function that finds the transformation according to all the inliers matches, using RANSAC algorithm. At each of the algorithm iteration:
   a. Randomly choose **4** pairs of matching key-points $\{p, \tilde{p}\}$ and calculate the projective transformation according to them, such that for each i $p_i = H\tilde{p}_i$.
   b. Map **all** the coordinates from the first image to the second one. calculates the mapping error.
   c. Save the number of inliers in the current iteration: the number of points that have a mapping error of less than 5.

   Repeat for 1000 iterations. Return the transformation H corresponding to the maximal number of inliers (remember to re-compute it using all the inliers).

   Show all matching inliers key-points after RANSAC. For example:



useful functions: imshow, hold on/of, plot, mldivide

6. **Image warping:** Implement a function that gets an input image and a projective transformation and returns the projected image. Please note that after the projection there will be coordinates which won't be integers (e.g sub-pixels), therefor you will need to interpolate between neighboring pixels. For color images, project each color channel separately.
7. **Note1:** You will need to pad the input image appropriately so that the transformed image will not get cropped at the edges. The padding doesn't have to be exact, just add enough so that the entire warped image is visible.
   Note2: imwarp() is not allowed. You need to write your own implementation of warping.

Useful functions: interp2

8. **Stitching:** Implement a function that gets two images after alignment and returns a union of the two. The union should be a simple overlay of one image on the other. Leave empty pixels painted black.