Assignment 1:

Homography & Panorama

Due Date: 03/12/20

The problem:

Given a pair of images (src.jpg, dst.jpg) and a file of matching points (matches.mat), we want to make a projective transformation of the source image in order to merge it with the destination image, and build a panorama image.
For motivation for the final result, you may view examples in the link:
http://www.cs.bath.ac.uk/brown/autostitch/autostitch.html

During the exercise, we will start by creating a system for calculating homography (2D projective transformation) from a list of matching points. Next, we'll be adding in the ability to cope with outliers. Finally, we use this system to build a panorama in a (semi) automatic approach.

The exercise will be implemented in python 3.6 or higher. It is recommended using the PyCharm Community (download) or a different IDE of your choice.

Python packages in this assignment: pillow, numpy, scipy, random, matplotlib, time and opencv. Any additional libraries must be approved by the teaching assistant.

A few preparatory steps:

- Load the images src.jpg, dst.jpg (appears at the beginning of test_script.py).
- Load the file matches_perfect.mat , notice that this is a dictionary containing two keys: match_p_src, match_p_dst.
- Display the matching points on both images and check if they are indeed a perfect match.
- Similarly, load the file matches.mat, this file contains in addition to correct match points, some mismatching pairs (outliers).
- Display the points on both images and notice the mismatched points.
- *Tip – you may use the function matplotlib.pyplot.scatter to draw points on an image.

Note - All the functions that you will be required to write in the exercise must be combined into one file named ex1_functions.py.

Part A: Homography computation

1. Build a system of equations of the form $A\underline{x} = \underline{b}$, as learned in class, for **projective** transformation. Attach the development to your exercise solution. How do we get the conversion matrix from the equation system?

2. Write a function that estimates the transformation coefficients from source (src) to destination (dst), from the equation system in section 1.

   Use the following API:

def compute_homography_naive(mp_src, mp_dst):

Input:

mp_src – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the src image.

mp_dst – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the dst image.

Output:

H – Projective transformation matrix from src to dst.

3. Load the matches_perfect.mat file and calculate the transformation coefficients using the compute_homography_naive function. Present the result.

4. Implement the transformation function from the source to the destination using the **Forward Mapping** transform, and display the source image after a projective transformation, according to the coefficients obtained in section 3.

5. What are the problems with Forward Mapping and how are they reflected in the image you received?

6. Now load the matches.mat file, and repeat sections 3 and 4. Did you get a different result? Explain. The image may be too large to display, specify. (You can shrink it and then present it)

Part B: Dealing with outliers

7. Implement a function that calculates the quality of the projective transformation model.

   Use the following API:

def test_homography(H, mp_src, mp_dst, max_err):

Input:

mp_src – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the src image.

mp_dst – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the dst image.

max_err – A scalar that represents the maximum distance (in pixels) between the mapped src point to its corresponding dst point, in order to be considered as valid inlier.

Output:

fit_percent – The probability (between 0 and 1) validly mapped src points (inliers).

dist_mse – Mean square error of the distances between validly mapped src points, to their corresponding dst points (only for inliers).

8. Implement a function that calculates the source-to-target coefficients that deal with outliers by using **RANSAC** (use the functions you built in previous sections).

   Use the following API:

def compute_homography(mp_src, mp_dst, inliers_percent, max_err):

Input:

mp_src – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the src image.

mp_dst – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the dst image.

inliers_percent – The expected probability (between 0 and 1) of correct match points from the entire list of match points.

max_err – A scalar that represents the maximum distance (in pixels) between the mapped src point to its corresponding dst point, in order to be considered as valid inlier.

Output:

H – Projective transformation matrix from src to dst.

9. Suppose there are 30 match points and it is known that 80% of them are correct. What is the number of randomizations needed in this case to guarantee 90% confidence? Of 99%? How many iterations must be done to cover all options?

10. Load the matches.mat file and calculate the transformation coefficients using the compute_homography function. Present the obtained coefficients, as well as the source image after projective transform using forward mapping. Compare the results you got to the results in sections 4 and 6.

Part C: Panorama creation

11. Implement the transformation function from the source to the destination using the **Backward Mapping** transform, which uses Bi-linear interpolation, and display the source image after a projective transformation, according to the coefficients obtained in section 10. Compare to the image obtained in section 10.

12. Implement a function that produces a panorama image from two images, and two lists of matching points, that deal with outliers using RANSAC (use the functions from previous sections).

    Use the following API:

def panorama(img_src, img_dst, mp_src, mp_dst, inliers_percent, max_err):

Input:

img_src – Source image expected to undergo projective transformation.

img_dst – Destination image to which the source image is being mapped to.

mp_src – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the src image.

mp_dst – A variable containing 2 rows and N columns, where the i column represents coordinates of match point i in the dst image.

inliers_percent – The expected probability (between 0 and 1) of correct match points from the entire list of match points.

max_err – A scalar that represents the maximum distance (in pixels) between the mapped src point to its corresponding dst point, in order to be considered as valid inlier.

Output:

img_pan – Panorama image built from two input images.

<u>Guidelines</u>:

- Use the Forward Mapping transform on the source image corners to create a bounding rectangle for the output image.
- Use Backward Mapping to perform the transformation.
- For overlapping areas, select pixel values of the destination image.

13. Run the panorama function for the src.jpg and dst.jpg images, using the points from the matches.mat file. Set 80% inliers and a maximum error of 25 pixels. Present the output panorama.

14. Use a pair of your own images to build a panorama (using the panorama function). Images should be called src_test.jpg and dst_test.jpg. Note that it is preferable to use images of the same size.
    Run the create_matching_points.py file to produce the matches_test.mat points file with 25 matching points. Make sure there are at least 10% incorrect matching points in the list (outliers). Present the input images, along with the marked matching points, and present the output panorama.

Submission instructions:

- A document containing reference to all sections of the exercise must be submitted, showing all the results and answering all questions (no code is required in the document).
- All API-defined python functions of the assignment must be submitted, as well as any associated functions that you wrote (all functions should be in ex1_functions.py). The functions will be automatically run and tested.
- Attach your pair of images (src_test.jpg and dst_test.jpg) as well as your matching points file (matches_test.mat).
- Check that you are able to run the attached **test_script.py** without making any changes to it. This will only be possible if all the functions you need to write in the exercise appear in ex1_functions.py.
- **The solution with all relevant files must be submitted in the submission box in the model within a zip file named:**

  assignment1_ID1_ <id_1> _ ID2_ <id_2>

  If the zip file exceeds 50MB, you may email it to: berkovitz1@mail.tau.ac.il
  The subject of the email should be stated as follows:
  Assignment 1 ID1: <your_id_number> ID2: <your_id_number>
- Late submission will result in grade reduction.

Good Luck!