Panorama Stitching

This homework is intended provide students with the opportunities to create panoramas using the matching routine.

The zip file containing sample images and code is **hw5.zip**. This assignment consists of several Python script files.

main_pano.py is the main script for the panorama part of the assignment.

To make the assignment more tangible, we've implemented some boilerplate code in hw_utils.py, which you SHOULD NOT modify. Your function implementations for the entire assignment should be stored in solution.py. And get and use the functions associated with *FindBestMatches* that you used in the last HW04. You are free to use main_proj.py and main_pano.py to get started. If you are using Juypter notebook, please modify the provided main files, which should be straightforward.

> Hand in all parts of this assignment in PLATO (both the code and report PDF file as specified). To get full marks, your functions (i.e., *.py files) must not only work correctly, but also must be clearly documented with sufficient comments for others to easily use and understand the code. You will lose marks for insufficient or unclear comments. In this assignment, you also need to hand in scripts showing tests of your functions on all the cases specified as well as the images and other answers requested. The scripts and results (as screenshots or otherwise) should be pasted into a single PDF file and clearly labeled. Note that lack of submission of either the code or the PDF will also result in loss of points.

Panorama

In this question, you are writing a program to create **panoramas.** To perform a sanity check on your Python environment, you could run*: python test_pano.py* If the program runs smoothly and produce this **panorama image**, you are good to go.

(test_pano.py 실행화면)

Overview: Given a list of images from the same scene, we can create a panorama by the following process. To simplify things, we pick one image in the image list as the reference image, and all other images are the source images. The ultimate goal is to find a homography matrix for each source image to project it to the reference image frame. A homography matrix is determined by at least 4 matches. You can compute a homography matrix using the function *cv2.findHomography* in OpenCV. The matches can be obtained through your *FindBestMatches* implementation.

Since there will be false matches from *FindBestMatches*, using those matches would yield an inaccurate homography matrix. Instead, we use RANSAC to obtain a subset of the matches that have very similar homography matrices (a.k.a the consensus set). A match `(p_src, p_ref)` is treated as an inlier if the projection (call it `p_proj`) of the point `p_src` is at most `tol` away from `p_ref` in the reference frame in Euclidean distance.

Mathematically, dist(p_proj - p_ref) <= tol and p_proj := ProjToRefFrame(p_src).

At each RANSAC iteration, you randomly select 4 matches to compute the homography matrix. Use the homography matrix to project all keypoints of the source image to the reference frame. Count the number of inliers under this projection, and keep track of the

largest consensus set. At the end of the loop, you compute the final homography matrix using the largest consensus set. Then, with the boilerplate code, each source image is warpped to the reference image frame using the final homography matrix. In case of overlapping region between images, a naive weighted average is used so it is normal to see some artifacts on the boundaries of images.

Note: you can actually use Laplacian pyramid blending from Assignment 2 to obtain better results here, but this is not required and beyond the scope of the assignment.

HINT: We strongly recommend you study the file *test_pano.py* to understand how each component of the boilerplate code interacts with one and another. The file also shows how to make use of some of the OpenCV functions.

The assignment

1. (40 points) **Keypoint Projections**: To check if a match is an inlier, we must know how to project 2d points from source image frame to the reference frame using a homography matrix. Your task is to implement the function *KeypointProjection(xy_points, h)*. This function takes in an array of 2d points `xy_points` (in xy format) and project the point using the homography matrix `h`. The code documentation and assertions should give you a sense of input and output formats you need to follow.

   Hint: You first convert the 2d points to homogeneous coordinate, perform the projection by a matrix multiplication, and convert back the projected points in homogeneous coordinate to the regular coordinate by dividing through the extra dimension. If the extra dimension is zero, you should replace it with `1e-10` to avoid dividing by zero. You can do *python main_proj.py* to visualize your results of projection. The script will project (randomly sampled) keypoints from a source image using the homography matrix. We've provided in `test.pkl` into the reference frame. The brightyellow dots are the points in the source figure. Their projections are plotted in the reference image with the same color. The red dots are the corresponding reference points (remember, we have matches). If your implementation is reasonable, you should obtain a figure similar to this **one**, where the red and yellow dots overlap a lot. Note that the source image is rotated, but the homography projection captures the rotation.

   Include resulting "Hanging1-Hanging2" images showing your best projection in a PDF.

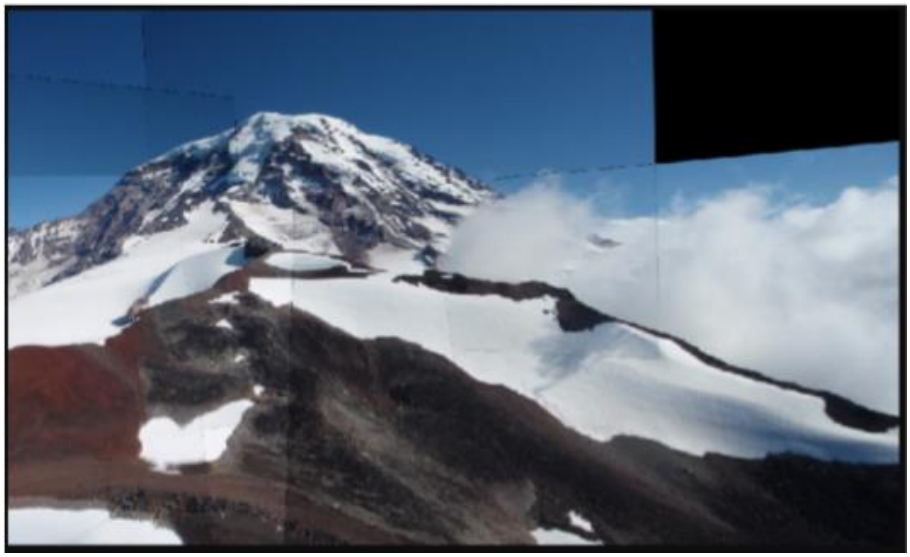
2. (60 points) **RANSAC Homography**: Now that you can project 2d points from the source images to the reference image using a homography matrix. The next step is to use RANSAC to find the biggest consensus set of matches to compute the final homography matrix. In the function *RANSACHomography(xy_src, xy_ref, num_iter, tol)*, the parameters `xy_src` and `xy_ref` store the xy coordinates of matches between a source image and a reference image. The matches are from *FindBestMatches*.

   Specifically, you run RANSAC for `num_iter` times. At each iteration, you randomly pick 4 matches of points to compute a homography matrix. Then you project all keypoints in the source image to the reference image using the computed homography matrix. You compute the Euclidean distance between each projected point to its correspondance in the reference frame. If, for a match, the projected point gives a distance no more than `tol`, the match is considered an inlier. The consensus set of this iteration is the set of inliers. Across iterations, you will keep track of the largest consensus set, and at the end of the loop, you compute and return the final homography matrix using the largest consensus set.

   Like other questions, we've prepared a script for you to visualize the results. You can run *python main_pano.py*

   You can change various parameters such as `num_iter`, `tol`, `ratio_thres` in the *main_pano.py*. Experiment with different combination of `num_iter` and `tol`. How does each of the two parameters contributes to the final panorama result? Include the discussion and the created panoramas in your report.

   Hint: if you implemented *RANSACHomography* correctly, and selected the right parameters, you can get a panorama that looks like **this.**



Don't worry about the artifacts on the boundaries. They are from a naive blending implementation.

You can produce fountain40.png, garden034.png and irving_out365.png. The order of digits on each photo indicates the order in your panorama program. For example, irving_out_365.png denotes that your reference image is irving_out3.png, and you project irving_out6.png and irving_out5.png into the reference frame.

Include resulting "fountain4-fountain0", "garden0-garden3-garden4" and "irving_out3-irving_out6-irving_out5" images showing your best panorama in a PDF.

Deliverables

You will hand in your assignment in PLATO. You should hand in one zip file including two files, a file containing your code (i.e., *.py file) and a PDF report. In code, All your answer must be in solution.py. In addition, hand in a PDF document showing your solutions (python function), scripts (i.e., records of your interactions with the Python shell) showing the specified tests of your functions as well as the images and other answers requested. Also, this file must have sufficient comments for others to easily use and understand the code. The PDF file must be organized and easily readable / accessible.

There must be 4 captures in your PDF. 1 for 2-1("Hanging1-Hanging2"), and 3 for 2-2("fountain4-fountain0", "garden0-garden3-garden4" and "irving_out3-irving_out6-irving_out5").

Assignments are to be handed in before 11:59pm on their due date.