EECS 4422/5323 Computer Vision
Assignment 3 - Model Fitting

All programming questions are to be completed in Python. This assignment is to be done individually. Do not use code from the web (other than the code instructed to use). If in doubt about external code usage, please reach out to us. We will be checking for copying using MOSS and reserve the right to give offenders (both copier and source) a zero on this assignment and pursue academic sanctions.

1 RANSAC-based Image Stitching

The goal of this question is to write a mosaic/panorama application. A panorama is a wide-angle image constructed by compositing together a number of images with overlapping fields-of-view in a photographically plausible way.

Part A:

[30 points] In this part, you will write code to construct a mosaic based on an affine transformation. The images you will work with are shown in Fig. 1 and can be downloaded here: images. An example result is shown in Fig. 2. An affine transformation is equivalent to the composed effects of translation, rotation, isotropic scaling and shear. Formally, an affine transformation of an image coordinate, $\mathbf{x}_1$, is given by the matrix equation $\mathbf{x_2} = \mathbf{T}\mathbf{x_1} + \mathbf{c}$. The unknowns are given by the elements in the 2×2 matrix $\mathbf{T}$ and the 2×1 vector $\mathbf{c}$. These image pairs have been created synthetically to ensure that an affine transformation will be suitable. *(Real mosaics are constructed with a homography image transformation. This transformation is more general than an affine transformation. Nonetheless, the same basic robust estimation architecture you are implementing in this part of the assignment applies when constructing a homography-based mosaic in Part B.)*

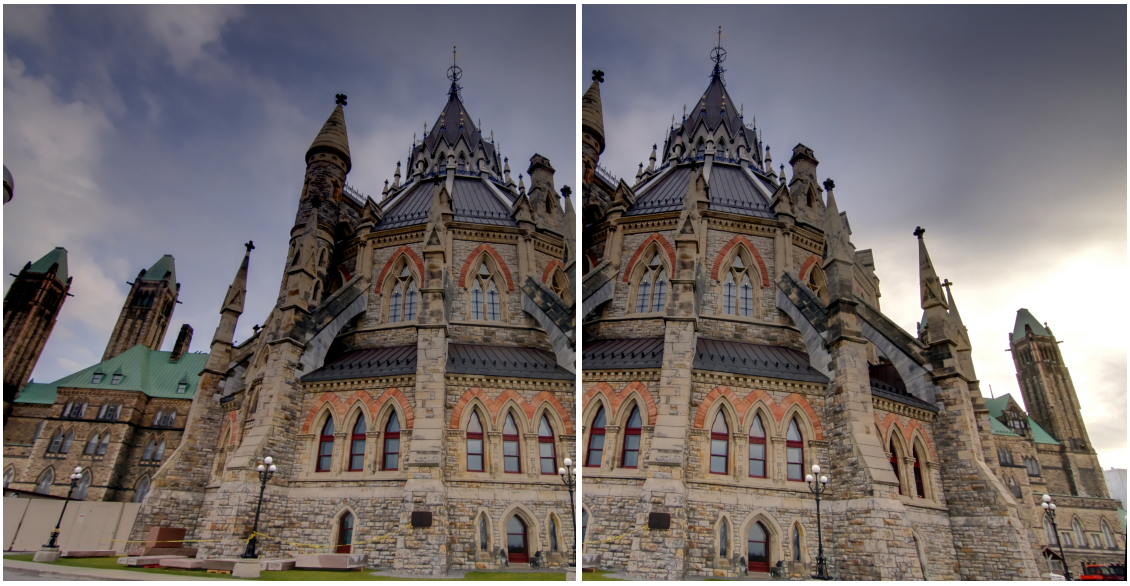

Figure 1: Images used to create the Parliament panorama using an affine transformation.

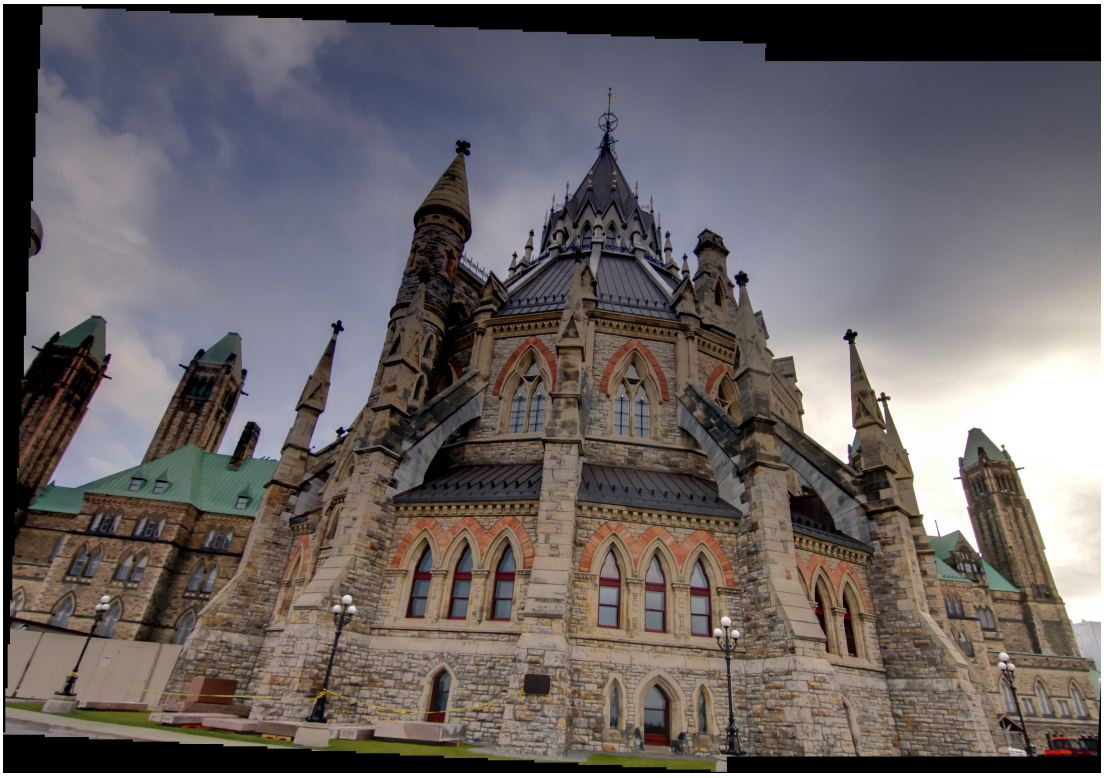

Figure 2: An example (affine) panorama result using the Parliament images.

1. **Preprocessing** Load both images and convert to grayscale.

2. **Detect keypoints and extract descriptors** Compute image features in both images. The feature detector and descriptor you will be using is SIFT. Use the OpenCV library to compute SIFT features. The function for computing SIFT features is `cv2.SIFT_create().detectAndCompute()`; see this tutorial for details about using SIFT in OpenCV. To visualize the detected SIFT keypoints and their respective orientations in the image, use the function `cv2.drawKeypoints`.

3. **Match features** Compute distances between every SIFT descriptor in one image and every descriptor in the other image. For fast computation of (squared) Euclidean distances, you can use either `scipy.spatial.distance.cdist(X,Y,'sqeuclidean')` or `kornia.feature.match_snn(tensor1, tensor2)`.

4. **Prune features** Select the closest matches based on the matrix of pairwise descriptor distances obtained above. You can select all pairs whose descriptor distances are below a specified threshold, or select the top few hundred descriptor pairs with the smallest pairwise distances. Display the locations of these inlier matches in both images by using `plot_inlier_matches` (provided in the starter code).

5. **Robust transformation estimation** Implement RANSAC to estimate an affine transformation mapping one image to the other. Use the minimum number of pairwise matches to estimate the affine transformation; state this minimum number as a comment in your code. Since you are using the minimum number of pairwise points, the transformation should be estimated using the inverse transformation rather than least-squares. Inliers are defined as the number of transformed points from image 1 that lie within a user-defined radius of ρ pixels of their pair in image 2. You will need to experiment with the matching threshold, ρ, and the required number of RANSAC iterations.

6. **Compute optimal transformation** Using **all the inliers** of the best transformation found using RANSAC (i.e., the one with the most inliers), compute the final transformation with least-squares.

7. **Create panorama** Using the final affine transformation recovered using RANSAC, generate the mosaic and **display the colour mosaic result to the screen**; your result should be similar to the result in Fig. 2. Warp one image onto the other using the estimated transformation. To do this, you can use either the functions `skimage.transform.ProjectiveTransform` and `skimage.transform.warp` together or use `kornia.geometry.warp_perspective`. Create a new image large enough to hold the mosaic and composite the two images into it. You can create the mosaic by taking the pixel with the maximum value from each image. This tends to produce less artifacts than taking the average of warped images. To create a colour mosaic, apply the same compositing step to each of the colour channels separately.

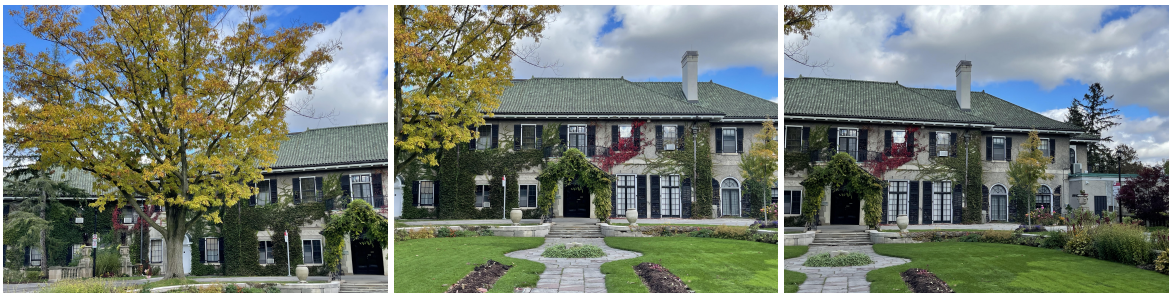

Figure 3: Glendon Hall input images used to create the panorama with a homography transformation.

Part B:

[10 points] In this part, you will write code for constructing a panorama based on a homography transformation. The three images of York University's Glendon Hall (see Fig. 3) to be composed are available here: images. An example result is shown in Fig. 4. You should reuse the code from Part A but swap out the parts that refer to the affine transformation with the homography. It is recommended to register the left and right views with the middle one.

The minimum number of point correspondences to estimate a homography is four. Using a homography yields a set of homogeneous linear equations, $\mathbf{Ax} = \mathbf{0}$. The solution to both the system of homogeneous equations consisting of four point correspondences and homogeneous least squares,

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{Ax}\| \tag{1}$$

subject to the constraint

$$\|\mathbf{x}\| = 1, \tag{2}$$

is obtained from the singular value decomposition (SVD) of $\mathbf{A}$ by the singular vector corresponding to the smallest singular value. Use either `U,S,V = numpy.linalg.svd(A)` or `U,S,V = torch.linalg.svd(A)`, where `V[len(V)-1]` gives the smallest singular value.

1. Using your RANSAC-based homography code, generate the mosaic using the given image pair and **display the colour mosaic result to the screen**.

2. Run your code on an image triplet of your own choosing and **display the colour mosaic result to the screen**. Make sure the images you choose have significant overlap; otherwise, you will not be able to establish correspondences. Further, for a homography to be valid, the images can either be obtained from rotating in the same place OR from multiple vantage points if the scene is planar or approximately planar.

Submission Details

Submit all Python files and images required for the various parts of the assignment to run, preferably in a single Jupyter notebook. Your submission should clearly indicate the single file for the grader to run. The script should be partitioned similar to the steps outlined in the assignment. It is also expected that your code is commented. **If your code does not run we cannot and will not mark it.**

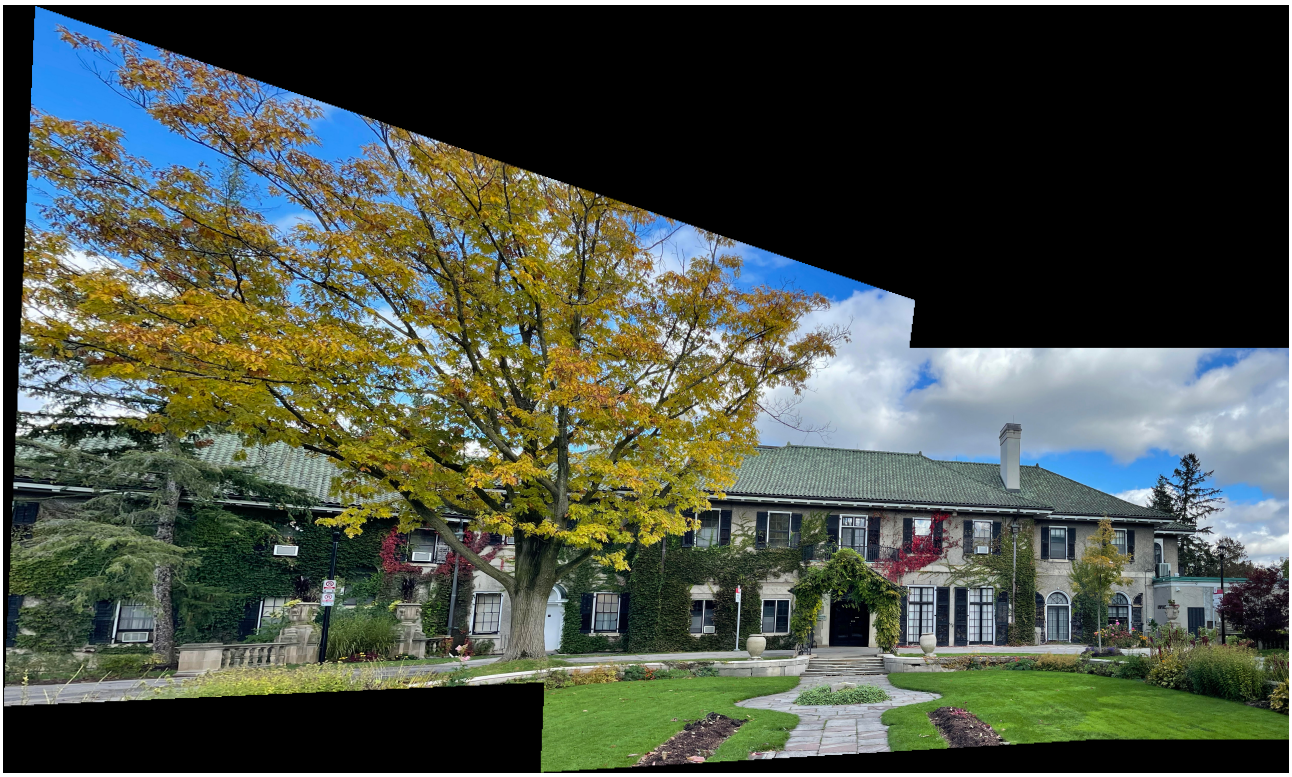

Figure 4: An example (homography) panorama result using the Glendon Hall images.