UMBC
CMSC 491/691 Computer Vision

Homework 3
100 points (120 counting extra credit)

Due: April 21, 2024 23:59 Eastern

Answer all problems according to the instructions provided.
Written problems may be typeset or handwritten.
Submit one `.zip` file named `LastName_FirstName_hw3.zip` containing a single `pdf` named `hw3.pdf` with the answers to both the written and coding sections. The pdf **must** contain relevant code snippets, outputs and results, and other requested information for each task. Put all other files (code, data, outputs, etc.) in a single directory named "`code`"

1 2D Transformations (20)

1. Derive a single homogenous 3x3 matrix for rotation of angle θ around a point (a, b). You must show steps to justify the derivation (not merely giving the answer).
2. A square has vertices p1 = (1, 1), p2 = (2, 1), p3 = (2, 2) and p4 = (1, 2). Calculate the new vertices of the square after a rotation about p2 through an angle of 45 degrees.
3. Consider an image transform that maps point (x, y) to (x', y') via the following equation:

$$x' = ax + by + t_x + \alpha x^2 + \beta y^2; \qquad y' = cx + dy + t_y + \gamma x^2 + \theta y^2$$

   Given two images with point-correspondences (x, y) to (x', y'), derive a technique to solve all parameters in closed-form. What is the minimal number of points needed to solve this?
   Note: you don't have to actually solve this problem, just setup the way to solve it.
4. Construct an image of 300×300 pixels, with a white irregular quadrilateral (i.e. four unique corners) approximately size 50×50 pixels is centered in the picture on a black background. Perform the following operations: (1) Translate the image by (30, 100) and (2) Rotate the image by 45 degrees using OpenCV commands about the original center. Display the rotated quadrilateral.

2 Writing your Own Image Stitching Algorithm (80)

The goal of the assignment is to automatically stitch images acquired by a panning camera into a mosaic as illustrated in Figures 1 and 2 below:

2.1 Algorithm Outline

1. Choose one image as the reference frame.
2. Estimate homography between each of the remaining images and the ref- erence image. To estimate homography between two images, we use the following procedure:

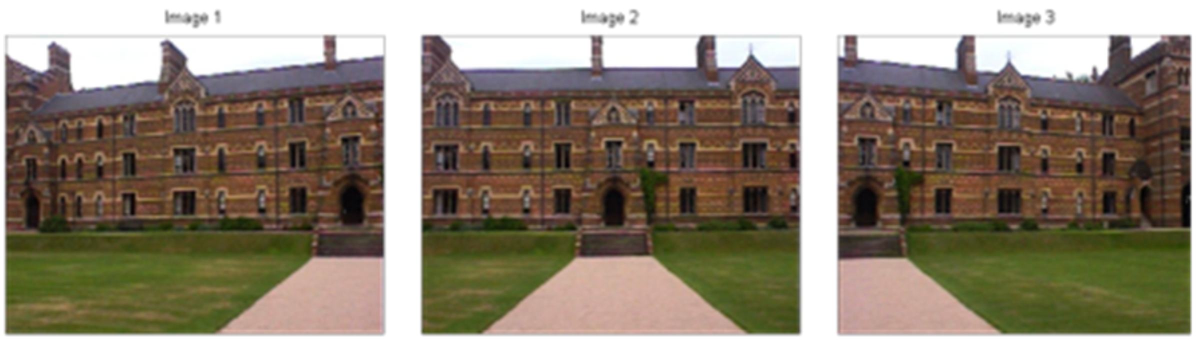

Figure 1: Three images acquired by a panning camera.

  a Detect local features in each image (using SIFT).
  b Extract feature descriptor for each feature point.
  c Match feature descriptors between two images.
  d Robustly estimate homography using RANSAC.

3. Warp each image into the reference frame and composite warped images into a single mosaic.

2.2 Tips and detailed description of the algorithm

The algorithm will be described on the example of stitching images shown in Figure 1. For each part, show the code snippet in the pdf report.

1. Choosing the reference image. Choosing the middle image of the sequence as the reference frame is the preferred option, as it will result in mosaics with less distortion. However, you can choose any image of the sequence as the reference frame. If you choose image 1 or 3 as the reference frame, directly estimating homography between images 1 and 3, will be difficult as they have very small overlap. To deal with this issue, you might have to "chain" homographies, e.g. $H_{13} = H_{12}H_{23}$.

2. Estimate homography:

   (a) Match feature descriptors between two images. That is, you will need to find pairs of features that look similar and are thus likely to be in correspondence. We will call this set of matched features "tentative" correspondences. See https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html on how to use in-built openCV functions to detect and describe SIFT features. To match features, you can use functions described here: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html.

      - As the first step, use Euclidean distance to compute pairwise distances between the SIFT descriptors. Then, you need to perform "thresholding".
      - For thresholding, use the ratio between the first and the second nearest neighbors. This is described in Section 5 and Fig 6b in "Multi-Image Matching using Multi-Scale Oriented Patches" by Brown et al.[1] [1]. Consult Figure 6b in the paper for picking the threshold. Note: You can ignore the fast indexing described in section 6 of the paper. You can visualize the tentative correspondences between two images by displaying the feature displacements.
      - For example, to visualize tentative correspondences between image 1 and 2:
        (i) show image 1,
        (ii) show detected features in image 1 (display only region centers as points, do not worry about the regions' scale)

[1] http://matthewalunbrown.com/papers/cvpr05.pdf

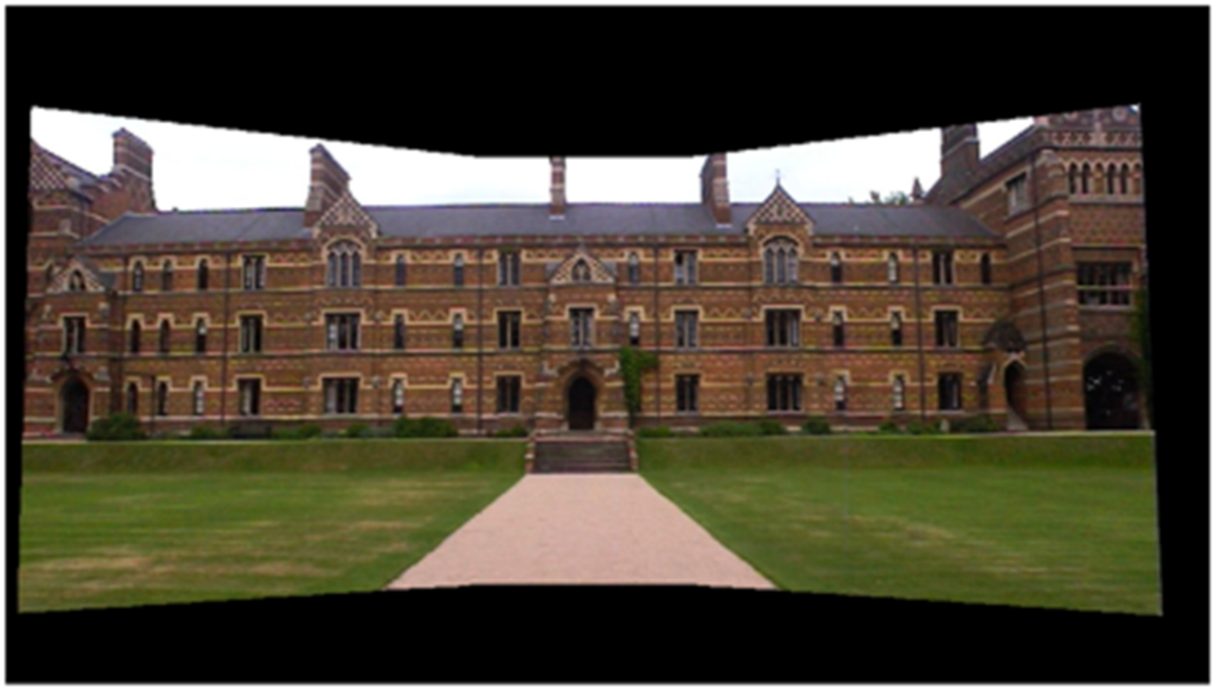

Figure 2: Images stitched to a mosaic.

(iii) show displacements between detected features in image 1 and matched features in image 2 by line segments. This is illustrated in Figure 3

(b) Robustly estimate homography using RANSAC. Use a sample of 4-points to compute each homography hypothesis.

(i) Write a function of the form and show your code snippet:

$$H = computeH(im1\_pts, im2\_pts)$$

where, again, $im1\_pts$ and $im2\_pts$ are 2×n matrices holding the (x,y) locations of n(= 4) point correspondences from the two images and H is the recovered 3×3 homography matrix. In order to compute the entries in the matrix H, you will need to set up a linear system of n equations, i.e. a matrix equation of the form $Ah = 0$ where h is a vector holding the 9 unknown entries of H. The solution to the homogeneous least squares system $Ah = 0$ is obtained from the SVD of A by the right singular vector corresponding to the smallest singular value. For more details on homography estimation from point correspondences see this note written by David Kriegman [2].

(ii) For RANSAC, a very simple implementation performing a fixed number of sampling iterations is sufficient. You should output a single transformation that gets the most inliers in the course of all the iterations. Write a function of the form and show your code snippet:

$$H, \text{max\_inliers} = ransac(corr, thresh, iter)$$

For the various RANSAC parameters (number of iterations, inlier threshold), play around with a few "reasonable" values and pick the ones that work best. You should display the set of inliers as illustrated in figure 3. Finally, after you find the set of inliers using RANSAC, don't forget to re-estimate the homography from all inliers.

[2]https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_estimation.pdf

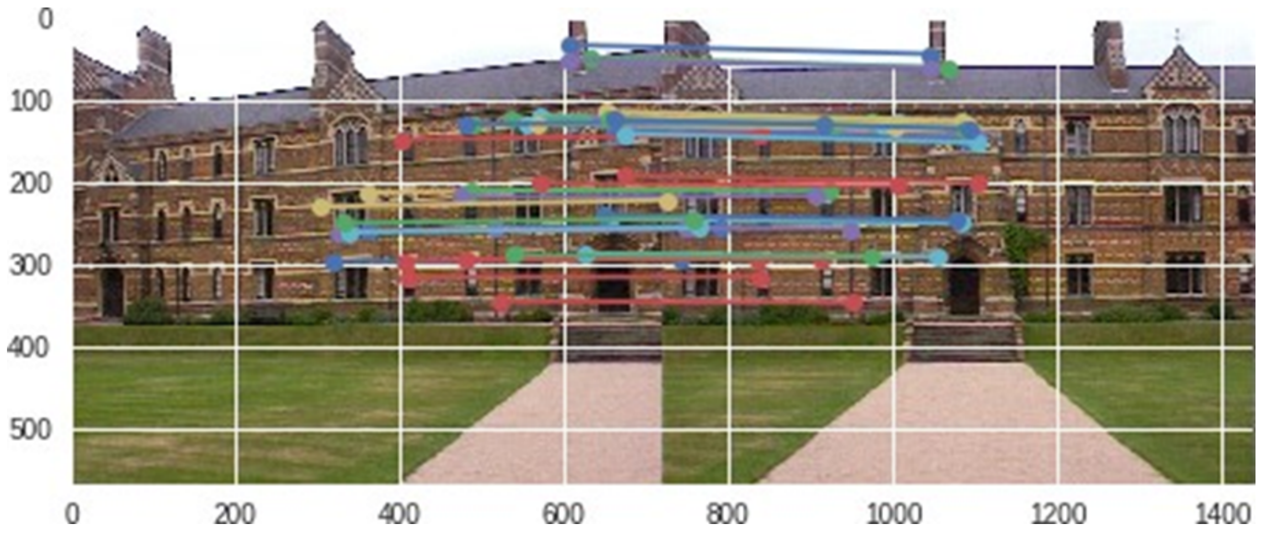

Figure 3: Enter Caption

3. Warping and compositing: Warp each image into the reference frame using the estimated homographies and composite warped images into a single mosaic. In Python, you can make use of the OpenCV function `cv2.warpPerspective`.

2.3 Test the Algorithm

Implement the stitching algorithm for the three images given to you: `keble_a.jpg; keble_b.jpg; keble_c.jpg`, which are the left, center, and right views respectively.

2.4 Submission Instructions

In the pdf report, include the following for the `keble` example:

1. Show the raw matches in each image
2. Show the visualization of the set of tentative matches and the set of inliers for one image pair (similar to figure 3).
3. Show the output of stitching image-1 and image-2.
4. Show the final mosaic (output of stitching all images together, similar to figure 2).

In the code directory, submit the following:

- a python script `homography.py` which takes three command line arguments (paths of two images) and saves the homography matrix as a .npy file at the location given by the third argument . The script should be executable as:

```
python homography.py <image-1-path> <image-2-path> <H-path>
```

- two python scripts `stitching_2.py` and `stitching_3.py` which use the outputs of `homography.py` to stitch the images together. Expected usage:

```
python stitching_2.py <image-1-path> <image-2-path> <H12-path>
python stitching_3.py <image-1-path> <image-2-path> <image-3-path> <H12-path>
<H23-path>
```

2.5 Extra Credit: Final Step (10)

Your first image stitching script probably resulted in black regions surrounding the panorama itself. Those wouldn't really be aesthetically great to display if you were shipping a finished product. Add another module that finds a rectangle inside your mosaic, s.t. it is the largest rectangle in the image without any black parts. Show a code snippet that allows you to take the mosaic and return a artifact free mosaic. Display the final result.

2.6 Extra Credit: Stitching using OpenCV inbuilt Tools (5)

Use the OpenCV built-in library `cv2.Stitcher` that can stitch images together. Show your code snippet. What are the technical differences (eg. between the feature detector, descriptor, matcher, and homography estimator) that you wrote from scratch vs the ones written by OpenCV developers? Does the OpenCV function do something additional? You might have to read the documentation to find out. See: `https://docs.opencv.org/3.4/d1/d46/group__stitching.html`.

2.7 Extra Credit: Stitch Xue and Ye's images (5)

In class, we discussed the incredible story of Xue and Ye from Qingdao, China. We have provided you the images that they took (`xue.png; ye.png`). Use the algorithm that you developed above and stitch the two images together! The best outcomes will be sent to the organizers of the International Conference on 3D Vision! [3]

References

[1] Matthew Brown, Richard Szeliski, and Simon Winder. Multi-image matching using multi-scale oriented patches. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 510–517. IEEE, 2005. 2

[3] `https://x.com/3DVconf/status/1718471443128733706?s=20`