Introduction

Merging images semantically consistently has wide-ranging applications, for instance, in image editing etc. Image alignment and stiching are classical but important problems for such applications. For instance, when we capture a panorama picture on a mobile phone, these methods come in handy. In this lab, we implement and visualize results for image alignment (Section 1) and image stitching (Section 2).

1 Image Alignment

Answers to Question 1:

1. Code for keypoint matching between a pair of images is in `keypoint_matching.py`. The given pair of images is shown in Figure 1.

2. A random subset of 10 matches for the given pair of boat images is shown in Figure 2. The tiny black dots on each image are the keypoints discovered by SIFT.

3. In order to find the transformation between a pair of images, we implement the RANSAC algorithm in `RANSAC.py`. We use the following hyperparameters which we found to work well empirically:

   - Number of random matches $K = 6$. Since we have 6 variables, in principle we can do with 3 keypoints. Empirically, we found that with 6 matches, we achieve decent results.
   - Maximum number of iterations: 500. We tried several values and found that with 500 iterations, we consistently converge at a decent solution.
   - Pixel radius to find inliers: 10. We used the value provided and found that it works well. We used this simple averaging scheme but in principle, one can smartly weight pixel intensities of the images with weights that are a function of how far the pixel is from the merging border.

   In Figure 3, we show a subset of 20 points in image 2 that are obtained via original points in image 1 transformed via the estimated parameters. In order to apply the transformation on image 1, we wrote our own custom `warp` function as well as tried using `cv2.warpAffine` out-of-the-box. We get close enough results using both except for image resizing which `cv2` performs so as to not cut any parts of the transformed image 1. This is illustrated in the Figure 4.

Answers to Question 2:

Figure 1: Given pair of images of a boat.

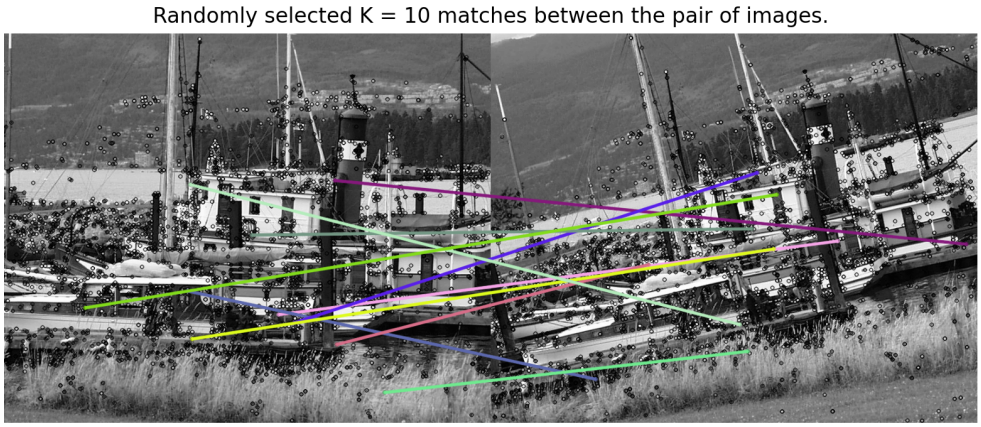

Figure 2: Random subset of matches found between the given images.

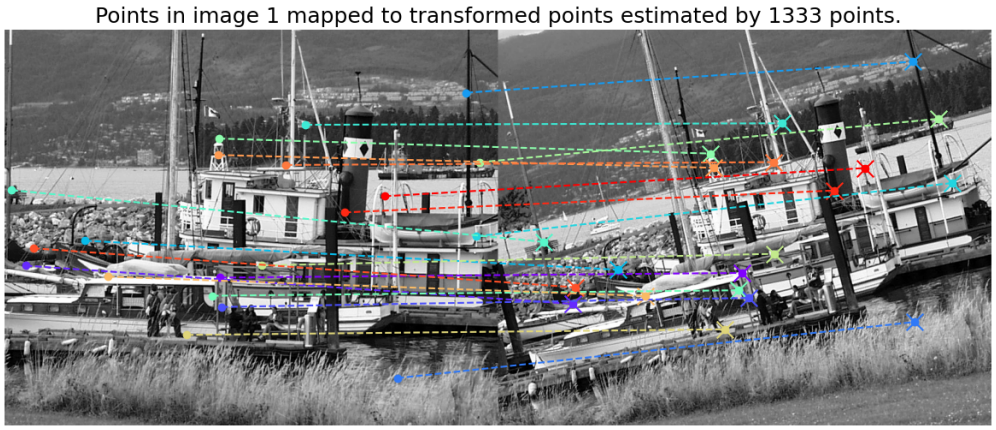

Figure 3: Subset of 20 subset of points in image 1 transformed and matched onto image 2. The crosses in image 2 show the points obtained by transforming points in image 1 via the estimated affine transform.

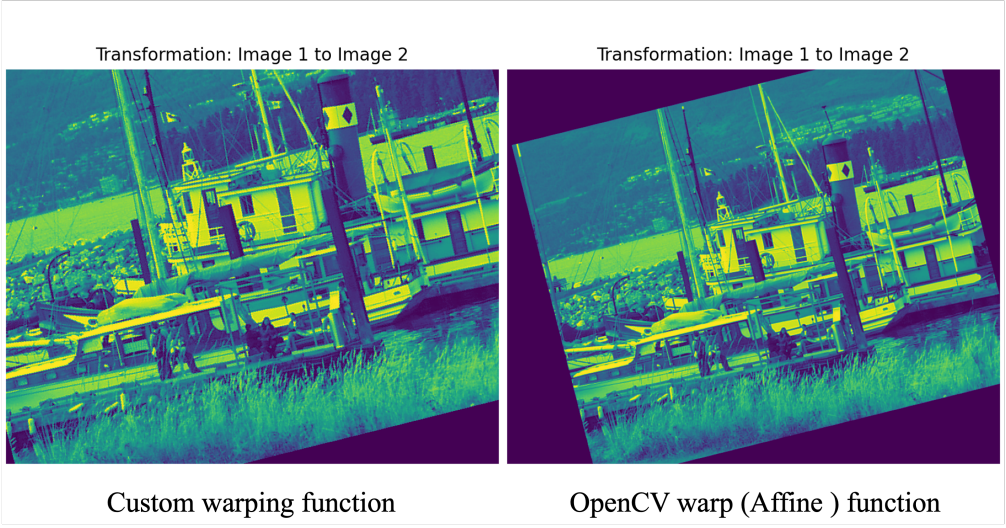

Figure 4: Comparison of our custom warping function vs the one provided within `cv2`.

Figure 5: Image warping results with varying number of maximum iterations in RANSAC algorithm. The number of iterations used in each image from left to right was 10, 50, 100, 200, and 500 respectively

1. In order to reliably solve for the transformation parameters, we need atleast 3 matches since we have 6 variables and each match provides two equations. In practice, more than 3 matches would often help to get less noisy solutions. We have used 6 in our implementation.

2. Empirically, on an average, we found > 200 iterations needed to consistently find a good set of matches to find the transformation parameters. We tried maximum iterations

   $$k \in \{10, 50, 100, 200, 500\}$$

   and the results (warped images) are shown in Figure 5. However, note that due to the inherent random nature of the algorithm, even using > 200 iterations may not provide the best fit since one can always be unfortunate to have picked significant number of outliers. On the contrary, one can also get fortunate and find a good set of parameters in very few iterations.

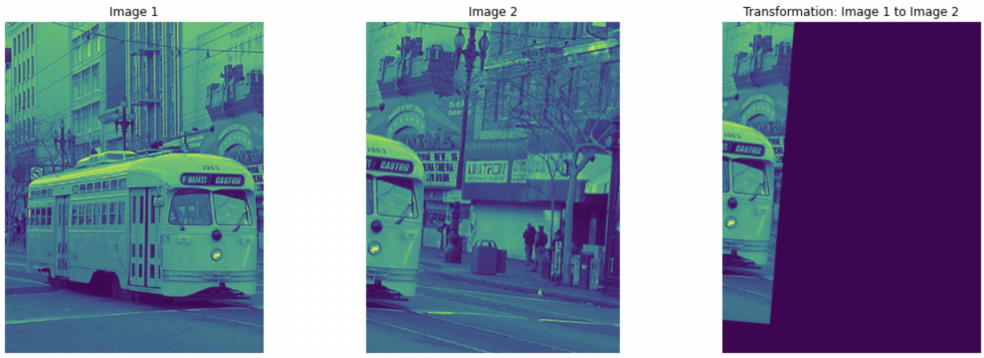

Figure 6: RANSAC algorithm applied to the pair of images to find the best affine transformation.

2 Image Stitching

Procedure

1. We compute the transformation between the given images from the previous part and the result is shown in Figure 6. Note that we use grayscale images for this step since color does not matter in finding the geometric transformation. We tried tuning parameters like the number of iterations and matches and found `num_matches=4, max_iter=2000` to work well for our case.

2. We apply inverse of the estimated transformation to transform image 2 in the coordinate space of image 1. Then, we find the intersectional area to find the size of the stitched image. This is illustrated in Figure 7.

3. Finally, once we get a bounding box which would eventually become our stitched image, we combine images 1 and 2 as follows:

   - For pixels (x, y) that lie in the image 1 but not in image 2, we pick the intensity from image 1.
   - For pixels (x, y) that lie in the image 2 but not in image 1, we pick the intensity from image 2.
   - For pixels (x, y) that lie in the image 1 as well as image 2, we take an average of their intensity value.

Answers to question 1:

1. The code for stitching images is provided in `stitch.py`.

2. The result for given pair of images has been provided in Figure 8.

Figure 7: Intermediate stage in stitching images: On the left is the image 1 and on the right is image 2, both in their own co-ordinate spaces. A larger canvas is considered only to show all the images and transformations together. Then, red points denote the corners of image 2 transformed with the estimated parameters. Finally, using these, we can compute a bounding box (green) which shall contain our stitched image.

Figure 8: Image stitching applied the the given sample pair of images.

Conclusion

In summary, we looked at techniques for image alignment and image stitching based on the assumption of affine transform from one image to another. We implemented both these methods and analyzed results for the sample image cases.