Project 4 Report

1 Intro to Homographies

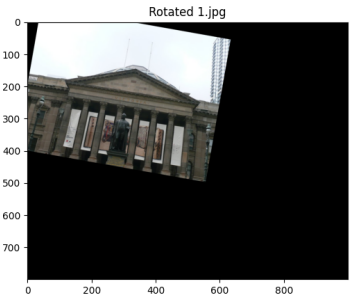

Figure 1: Rotated 10 degrees clockwise

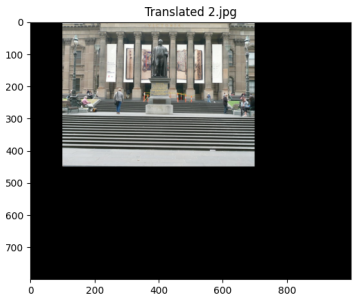

Figure 2: Translated 100 pixels right

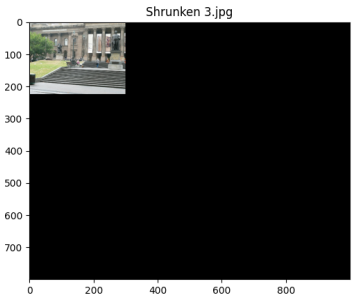

Figure 3: Scaled down by half

First we rotated 1.jpg by 10 degrees clockwise. The rotation matrix is a combination of translating, rotating, and translating back to the original position. The image is translated so that its center is at the origin and the rotation can be applied correctly. Next we translate 2.jpg 100 pixels to the right. This translation is performed using a simple 3×3 translation matrix that shifts the image horizontally. Lastly, 3.jpg is scaled down by half. This is also just performed using a 3×3 scaling matrix that shrinks the image by 0.5 in both x and y dimensions.

Each transformation is applied to the corresponding image using the `cv2.warpPerspective` function, and the output images are displayed on a 1000×800 px canvas.

2 Panoramic Stitching

Computing SIFT Features

We applied the SIFT algorithm to detect key points and extract descriptors from each image, using functions including `SIFT_create()` , `sift.detectAndCompute`, and `cv2.drawKeypoints`.

Figure 4: Image 1.jpg with key points

Figure 5: Image 2.jpg with key points

Figure 6: Image 3.jpg with key points

The figures above demonstrate the key points obtained from the SIFT algorithm overlaid on top of the images. As usual, the key points represent the distinctive and scale-invariant features in the image, while the descriptors provide a unique representation of the local image region surrounding each key point. These invariant features establish correspondences between different images, even if the images are transformed relative to each other. This enables us to align them and create a fused image further down the line.

Matching Features

We calculate the top 100 matches between the SIFT feature descriptors of 1.jpg and 2.jpg as well as 2.jpg and 3.jpg based on their L_2 distances. First, we obtain the distance matrix between the descriptors of the two input images by using `distance_matrix`. Based on this matrix, we obtain a list of index pairs representing the best 100 matches.

The `cull_invalid` function is used to remove any invalid matches from the list of best matches. This ensures that the indices are within the range of the key points for each image. The valid matches are converted to `cv2.DMatch` objects, which represent the matching information between two key points. This allows the feature matches to be passed into the `cv2.drawMatches()` function, which visualizes the matching key points between the image pairs.

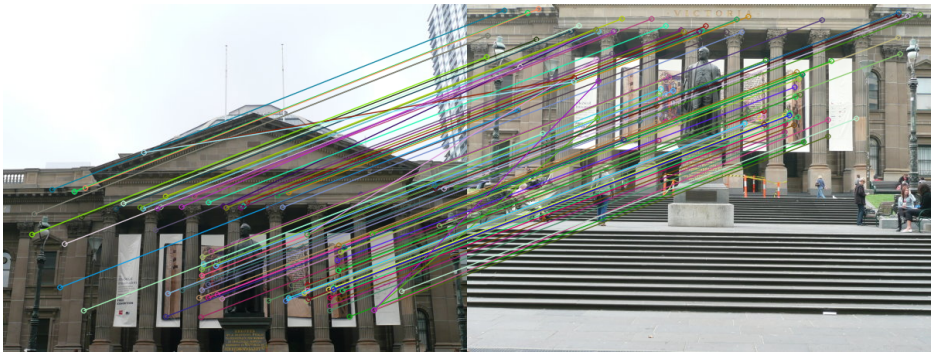

Figure 7: Visualization of the top 100 SIFT feature matches between 1.jpg and 2.jpg

Outlined in the image above are the top 100 feature matches between images 1.jpg and 2.jpg. These matches are the ones potentially used to establish a correspondence between the images. We observe that a majority of the matches follow a consistent direction, indicating mostly accurate correspondence. However, some matches appear less optimal and deviate from the prevailing pattern, which may introduce some noise or inaccuracies in the image alignment process. Despite these outliers, the majority of the matches are reliable and contribute to the overall alignment of the images.

Figure 8: Visualization of the top 100 SIFT feature matches between 2.jpg and 3.jpg

By inspection, it appears that there are fewer bad correspondences in the feature matches between images 2.jpg and 3.jpg, as we do not observe many significant outliers. This suggests that the matching process has been more successful in identifying reliable feature matches between them, resulting in more accurate correspondences.

Estimating the Homographies

We compute the homography matrices using the `cv2.findHomography()` function. To do so, we first separated the matched key points obtained previously so that one list contains the key points in one image and another list contains the corresponding key points in the other image. Both lists are converted to `float32` data type to ensure compatibility with the functions we are using. We use these lists as the inputs for the `cv2.findHomography()` function, along with the RANSAC method and a reprojection error threshold of 2 pixels. The resulting homographies, then, represent the necessary transformations that align image 1.jpg with image 2.jpg and image 3.jpg with image 2.jpg. We also get the masks that indicate which key points are considered inliners. Since these matches survived the RANSAC filtering

process, they serve to better align the images and ensure a more seamless stitching.

We now visualize these inliers. To do so, we iterate through the matched key points as well as the masks and include a key point if its corresponding mask value is 1. The inliers are then visualized using `cv2.drawMatches()`.

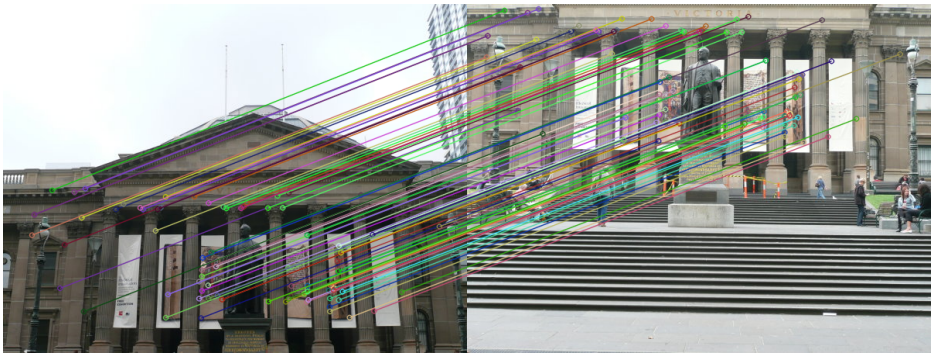

Figure 9: Visualization of RANSAC inliners between 1.jpg and 2.jpg

As expected, the inliers obtained after applying the RANSAC algorithm demonstrate considerable improvements in terms of coherency and reliability compared to the original feature matches. The inliers appear less noisy and display more uniformity in their distribution across the image pairs, indicating a higher degree of correspondence accuracy.

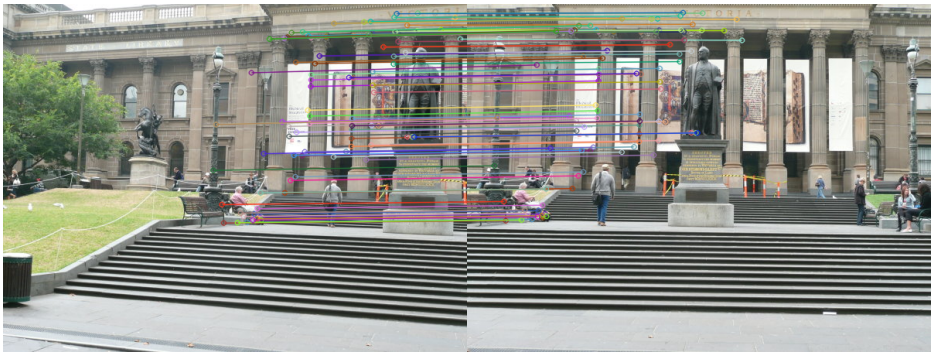

Figure 10: Visualization of RANSAC inliners between 2.jpg and 3.jpg

The RANSAC algorithm also removed some matches between 2.jpg and 3.jpg, albeit somewhat unnecessarily. The initial feature matches between these images were already quite consistent and well-aligned. In some cases, the removed matches were actually valid, such as the correspondences at the base of the statue. This suggests that the current RANSAC configuration, specifically the reprojection error threshold, may require further tuning to better distinguish between inliers and outliers so that valuable matches are preserved.

Warping and Translating Images

Having completed all the required steps, we can now stitch the three images together. To ensure that the fused image fits within the canvas, we combine the previously obtained homographies with a translation matrix that shifts the image 350 pixels to the right and 300 pixels down. These combined homographies are used to align images 1.jpg and 3.jpg with 2.jpg. For 2.jpg, we only need to apply the translation since it serves as the reference image.

The `cv2.warpPerspective()` function applies the computed homographies to the images, transforming and translating them appropriately. These warped images are fused together into a single panoramic image using the `np.maximum()` function, which effectively blends the three images together by selecting the maximum pixel value at each location. This results in a panoramic view of the scene, created by aligning and fusing images 1.jpg, 2.jpg, and 3.jpg.

Figure 11: Panoramic view created by aligning and stitching images 1.jpg, 2.jpg, and 3.jpg

We clearly see both the strengths and limitations in our approach. The alignment of static components, such as buildings and stairs, is well-executed, providing a seamless transition between the images. However, when objects in the images move between the time each photograph was taken, we see inconsistencies in the final stitched panorama. This can be observed in the form of semi-transparent objects, such as people walking through the scene. Additionally, there are minor discrepancies in brightness or color between the three images. To further improve the quality of the final image, more advanced blending techniques could be employed.