Image Processing - Exercise 4

Liel Amar, liel.amar, 211771993

Introduction

In this exercise, we were to create a blending algorithm that receives low-resolution and high-resolution images and stitches them together to create a well-detailed image. This is done using several concepts we've seen in class, such as *Feature Points Extraction* and *Feature Points Matching* using the SIFT algorithm, *RANSAC* for computing the high-consensus transformation model, *Image Warping*, and *Image Blending*.

Algorithm

The algorithm used in this exercise is as follows.

1. We import and load the two images - one high-resolution image and one low-resolution image. The input for this stage is two paths to the files to load, and the output is two images, described as 3D matrices.
2. Using the SIFT algorithm, we find feature points and descriptors for each image. The SIFT algorithm takes an image as an input and then computes the DOG (Difference of Gaussians) matrices. It selects the local maxima as key points, by comparing the current pixel with the 8 pixels surrounding it in the same level, 9 pixels in the level above, and 9 pixels in the level below. After the key points were obtained, for each key point, SIFT looks at a window around that pixel and computes the gradient of each nearby pixel, to create a histogram of 36 directions, taking into account the magnitude of each gradient as well. This way, SIFT can be rotation invariant. It then re-computes a 16x16 gradient window around each pixel and divides it into 16, 4x4 windows. Each window value is determined by the gradient of its 16 pixels, and the value is a vector of length 8. SIFT takes these 16 windows of length 8 and creates a descriptor, which is a vector of length 126.
3. With the feature points and descriptors calculated in the previous step, we loop over all feature points in the first image, and for each feature point in the second image, we compute the squared difference between their descriptors. We then return the two points with the least squared difference, which are the two best matches. We then filter outliers by computing the $\frac{1NN}{2NN}$ value (1NN being the squared difference of the first found point and 2NN being the squared difference

of the second found point), and we only return points that are below a certain threshold.

4. Afterward, we use the found matches from the previous step to compute the homography matrix through RANSAC. We input RANSAC all of our matches, and by continuously iterating, subsampling 4 points, computing a homography matrix, and looking for a large consensus, RANSAC returns the model (homography matrix) that best fits the data.
5. Then, we warp the second image with the inverse of the found homography matrix to make sure the first and second images align properly. This is done by applying the homogeneous transformation matrix on each (X, Y) pair (pixel) with backward warping.
6. Lastly, we create a mask out of the warped, second image, and blend the first and second images using the mask.

Implementation Details

To implement the algorithm described above, I've used two libraries: *Numpy* and *OpenCV*. I used these two specifically, as they are very rich with image-processing functionality.

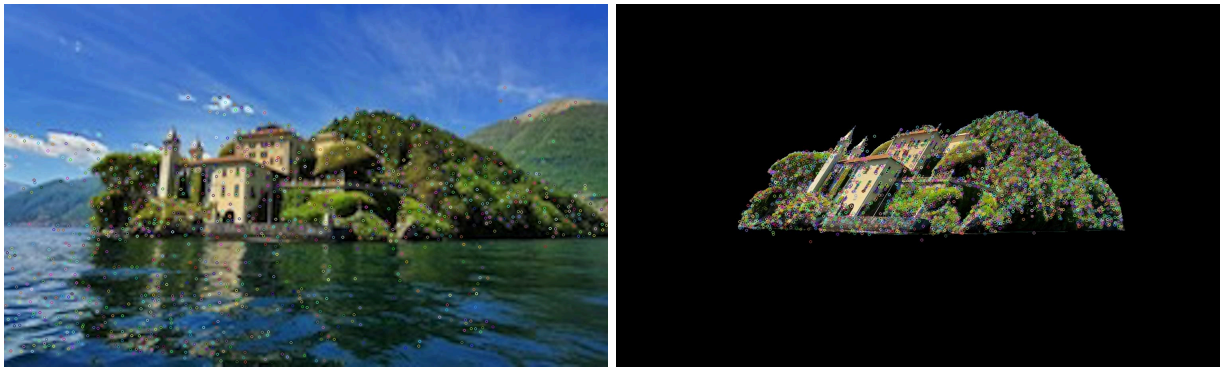
As for the implementation itself, I started by using *OpenCV's imread*, which reads the two images into *Numpy* matrices describing the images. Then, I create an *OpenCV SIFT* object through *SIFT_create*, and use *detectAndCompute* for each image, to compute a list of feature points and a descriptor for each feature point. The next step, which is computing the good matches between the first and second images' feature points, is being done through *OpenCV's BFMatcher*, which is essentially a brute-force matcher that can receive certain parameters to behave differently. I use the BFMatcher with the L2 norm and no cross-checking, which means for each feature point in the first image, the algorithm would go over all feature points in the second image and compute the distance between them using the L2 norm. In addition, since we're not using cross-checking, we don't require the points to be the best for each other. This allows us to then input the *knnMatch* method with our descriptors and $K = 2$, which in turn returns, for each feature point in the first image, the two points in the second image with the descriptors that have the lowest L2 distance. When we have the two best matches for each feature point in image one, we compute $\frac{1NN}{2NN}$ and take a threshold of 0.75 so this way we can discard outliers (points which we couldn't necessarily find a deterministic match for). Once we only have "matches" left, we use *OpenCV's findHomography* to find the homography matrix. We pass the good points from the first image and their corresponding points in the second image, and we specify the function

to use the RANSAC algorithm with a reprojection threshold of 5, meaning we allow an error of 5 pixels at max. This algorithm works by iteratively selecting 4 points, computing a homography matrix out of these points, and counting how many points agree with this matrix. RANSAC returns the homography with the largest consensus. Once we get a homography matrix, we compute its inverse, re-import the two images as float matrices, and apply the inverse homography to the second image with *OpenCV's warpPerspective*, by passing it the second image, the inverse homography matrix, and the target image size (which is the same size as the first image). This way the two images are now aligned. Lastly, we create a mask of the same shape as the warped, second image through *Numpy*, we blur the mask with a 7x7 Gaussian mask, as it gave the best results, and we combine the two images by taking the first image where the mask's value is not 1, and the second where it is 1.

A challenge I faced during this exercise was blending the two images seamlessly. It seemed like using Exercise 3's pyramid blending didn't give perfect results, so I decided to try another approach, which was creating a new mask, blurring it, and using it to blend the images. In addition, I had to try various hyper-parameters to find the ones that gave me the best results. These are the KNN threshold, the RANSAC reprojection threshold, and the Gaussian kernel size. I was also thinking about trying min-cut or dynamic programming, however, it didn't seem fit as min-cut would have most likely performed poorly as the cut needs to be between low and high-resolution images, and dynamic programming won't be able to get the shape correctly.

Visual Results

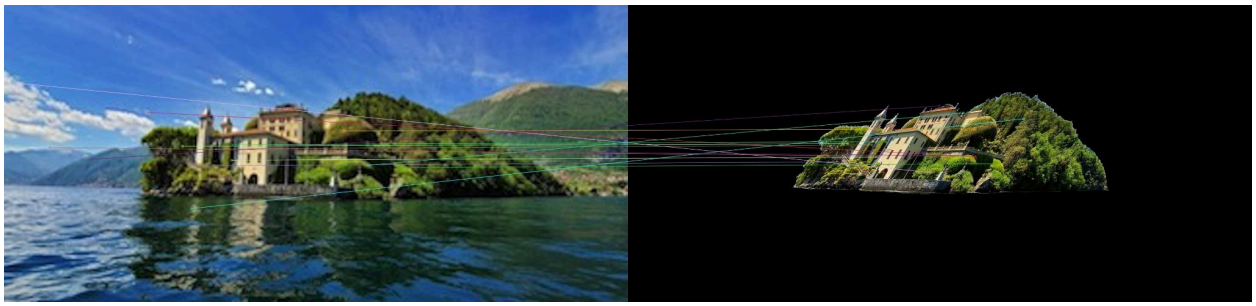
Let's break down the algorithm with visualizations.



Above, we can see the low-resolution and the high-resolution images, after we use SIFT to compute their feature points and descriptors. Each pixel that's surrounded by a circle represents a key point returned from the SIFT algorithm. If we try very hard, we can see that some points appear in both images, and later on, these points would be the ones

we are interested in, as they are inliers which would help us compute the homography matrix.


In the next two pictures, we can see that there are way fewer feature points shown. This is because we filter out outliers by matching points using a Brute-Force KNN matcher, and we only which their best two matches comply with $\frac{1NN}{2NN} < KNN_{THRESHOLD}$. This ensures that the points we are left with, are points that we are fairly certain about their match (since we want the distance between the best and second-best match to be quite large).



Above, we can see the actual matches between the remaining points. We are left with much fewer points, but the matches are pretty good. We still have some outliers (which we'll deal with later), but overall, most points have a good and accurate match.


Almost there - we can see that after computing the inverse homography matrix to the second image, it becomes almost perfectly aligned with the same object in the first image. This then helps us blend the two images, with the mask generated in the left image.


And lastly, we blend the two images and the result is a well-detailed and high-resolution subject (the building and forest over the lake, in this case), with the low-resolution and blurred background (the lake, mountains, and sky, in this case). Although we got a good result, it is not flawless. We can still see the blending effect around the edges and in some cases, it doesn't look natural. In the second image, we can even see some of the background being higher resolution due to an imperfect cut in the second image.

Conclusion

To sum everything up, we have seen that now, we're able to use Feature Detection, Descriptors, RANSAC, and backward warping to our advantage, adjust two images to be aligned similarly, and then blend them to increase the resolution of some parts of the image. Not only that, but we can further improve our algorithm to create panoramic images through the usage of SIFT's Feature Detection and much more. This exercise shows us how each step is crucial, eliminating outliers as much as we can, so that the final matches we get have a higher percentage of inliers, making it possible to find the correct homography with RANSAC.