Image Processing - Exercise 4

Yotam Gardosh, yotam.gardosh, 208541334

Introduction

The goal of the exercise was to blend a low-resolution image with a hight-resolution segment of the same image. The main techniques used in the program were feature extraction and matching, RANSAC for outlier rejection, and image warping.

Algorithm

The solution was structured into the following septs:

Detecting and Computing Features: I used the SIFT algorithm for detecting and describing key points in each image. The algorithm's input is and image and the output is a set of key points and their descriptors.

Finding and matching good features between images: I've used the RANSAC algorithm to find the best set of features while rejecting outliers and the homography that aligns matching features. The inputs are matched features; the output is a homography matrix.

Image Warping and Blending: Using the homography matrix, the high-resolution image is warped to the perspective of the low-resolution image, and the two are blended based on a mask generated by thresholding. The input are the 2 images, the mask and the homography and the output is the blended image.

Implementation Details

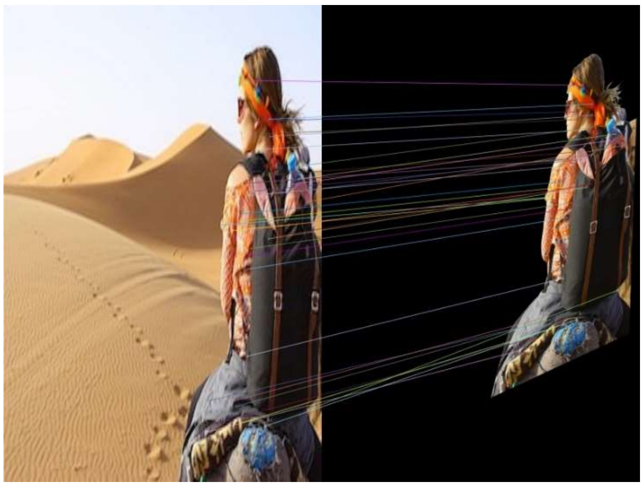
Detecting and Computing Features: was done with the “detect_and_compute_features” function that uses OpenCV's SIFT detector and descriptor with the cv2.SIFT_create() and detectAndCompute methods.

Finding and matching good features between images: was done with the “match_features” and “find_homography” functions, these use OpenCV's Brute-Force Matcher (cv2.BFMatcher) to match the SIFT descriptors between the two images, applying a threshold to select the best matches based on the distance. Then I used cv2.findHomography with the RANSAC algorithm to compute the homography matrix based on the matched feature points.

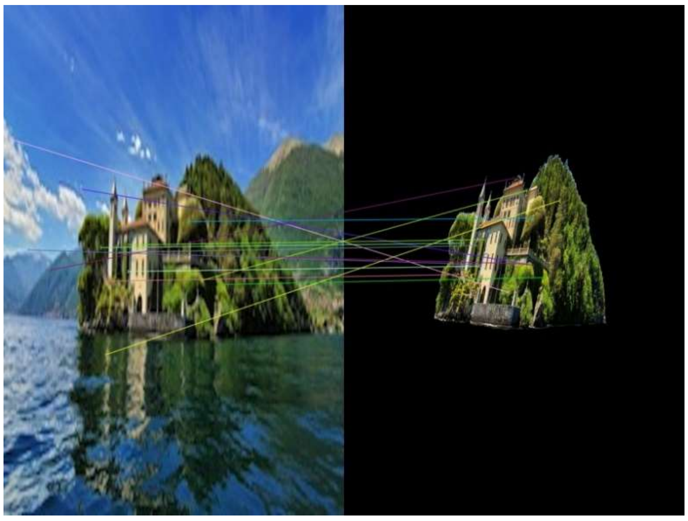
Image Warping and Blending: was done with the “warp_and_blend_images” and “mask_blend_images” functions as well as the “create_mask” function for the mask creation.
Warping is done with cv2.warpPerspective, using the homography matrix. Followed by creating the mask for either images without an alpha channel or by using OpenCV's cv2.cvtColor and cv2.threshold that are used to convert images to grayscale and apply a binary threshold or for images with an alpha channel were I could directly thresholding the alpha values to create a mask. Then to blend I used NumPy's np.where to combine images based on the binary mask.

Visual Results

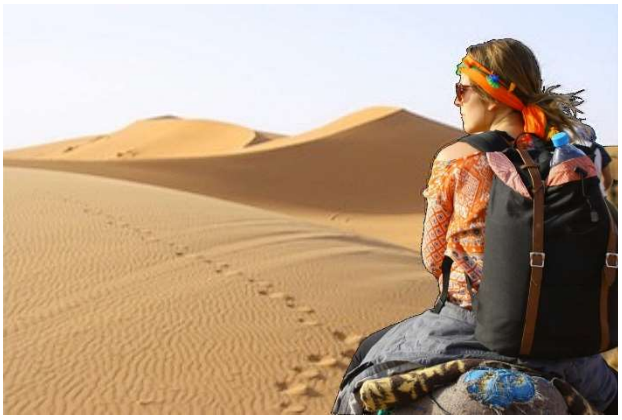
For the desert image the best hyper parameters found after testing were nearest-neighbours threshold in the RANSAC algorithm to be 75% and for the mask creation the intensity threshold was 9 this allowed me to match the following features:

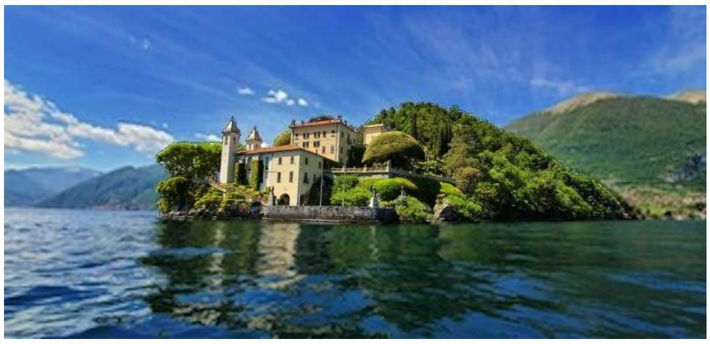


For the lake image the best hyper parameters found after testing were nearest-neighbours threshold in the RANSAC algorithm to be 60% and for the mask creation the intensity threshold was 9 this allowed me to match the following features:



The final blended images:





Conclusion

This project was an amazing learning tool to apply the techniques of feature finding and description, match selection and homography calculation, as well as blending. The process of trial and error used to find the best features goes to show how each image is unique and requires dedicated time and a lot of thought behind hyper parameter selection to get the right features for calculating the correct homography. In this project I also learned about alpha channels and how, for images that contain them, they can be used to create an excellent mask for an image.