Name: Xiaolin Li
ID: 455398598

Reflection on COMPSCI 773 A6

In the second half of COMPCI 773, the 3 phases used as an assignment are combined into Assignment 6-Image Stitching. First, we extracted the Harris corners of the two images (which should look similar), and then computed the putative matches by applying Normalized cross-correlation (NCC), then use RANSAC loop and Direct linear transformation (DLT) to get the best homography H, and finally apply warping function (code provided) to compute the image warping. The final image of the plot is the final result of image stitching. In this regard, I am going to reflect and summarize each phrase separately.

PHASE 1 – FEATURE DETECTION

The primary algorithm used in this phase is Harris Corner Detector. In this part, I first came across the NumPy package, which is equivalent to a list, but it runs faster than the list on most functions, which opened my eyes.

The algorithm runs for the left image and right image separately.

1. Input the image I and convert it into grayscale
2. Compute Sobel derivative filter to get Ix and Iy
3. Compute the square of derivatives Ix^2, Iy^2 and IxIy as the components of second moment matrix M with window function w(x, y) = 1
4. Get the Gaussian filters gx, gy and gxy using the square of derivatives and specific kernel size and sigma, where the kernel size must be a positive odd number and the sigma can be calculated by sigma = 0.25*(kernel_size-1)
5. Compute Cornerness score by formula C = gx * gy - gxy**2 - a*((gx + gy)**2) where a is the Harris constant setting to be 0.04
6. Apply threshold on Cornerness score which is to make sure all the corner is non-negative
7. Extract Harris corners as (x, y) tuples with the strength of the Cornerness score in a data structure if the Cornerness score on (x, y) is larger than all its eight immediate neighbours (non-maximal suppression)
8. Compute the first 1000 strongest corner and store the coordinates as tuples inside a list

The plotting image contains 1000 Harris corners in each image, and none of them on the edge of the mountain as the corners had been applied threshold.

The main point of confusion in this phase is the swap of my thoughts on width and height. For an image, its origin is the upper left corner, and when I write, I always think that the origin is at the lower-left corner, which leads to the coordinates (x, y) of all corners of the final image being written as (y, x). Eventually, I corrected this question by changing it back to (x, y), and the final plotting obtained was similar to the correct answer.

Secondly, I think this phase can be more optimized. Although the running time is about 100 seconds, optimizing specific parts of the filter (for example, changing the for loop to the NumPy pad function when padding or cancelling the padding directly) should reduce the complexity and improve the overall running speed.

PHASE 2 – EXTRACTION OF DESCRIPTORS AND MATCHING

The algorithm used in this phase is the Feature matching algorithm. This phase uses the 1000 strongest Harris corners (descriptors) of each image finally obtained in phase 1.

1. Precompute the function NCC with 2 arrays as parameters
2. Precompute the function of the ratio of best and second matching
3. First pad in the original image (px_array_left and px_array_right) so that n x n grids can be extracted from the original image as the window size is n (15 is a reasonable window size for this phase, but it is negotiable)
4. Perform brute force matching of descriptors from the left image with descriptors from the right image
   - 4.1. For all left image descriptors, loop over all right image descriptors
     - 4.1.1. Create an empty list called NCC_list which will contain variables (corner in the left image, matching corner in the right image, NCC matching score)
     - 4.1.2. Compute NCC matching score through function NCC in step 1 with a n x n block in the left padding image and a same size block in the right padding image
     - 4.1.3. If the length of NCC_list is smaller than 2, then append the variables, remove the duplication and sort it from largest to smallest according to the NCC matching score
     - 4.1.4. If the length of NCC_list is equal to 2, then we can set the larger NCC score to be bestMatch and the smaller one to be secondBestMatch. Compare the new NCC with bestMatch and secondBestMatch. If it is larger than the best Match, keep this matching as bestMatch, and change the secondBestMatch to be the original bestMatch; if it is larger than secondBestMatch but smaller than bestMatch, change the secondBestMatch to be this matching.
   - 4.2. Finally, we will get a final_bestMatch and final_secondBestMatch, then calculate their ratio using the function in step 2. If the ratio is smaller than 0.9, report the match with the final_bestMatch as a putative match

The plotting image contains around 330 matchings from the left image to the right image in the combined image.

For the NCC function, I tried two different methods. Among them, using numpy.std to calculate the denominator part of the NCC will take a long time to run the entire phase 2, and sometimes it will not run at all. In the end, I directly follow the formula to get the corresponding parts one by one and then combine them. For this problem, it is possible that my code originally overlaps too many loops to get stuck when running.

For the optimized code, I may try to delete the padding part and directly use the original image array to obtain n x n blocks to calculate the value of NCC.

PHASE 3 – WARPING

The algorithms used in this phase is the RANSAC loop and DLT. The array used in this phase adopts the matching pairs in the matching list in phase 2.

1. Set inlierMappingThreshold to be around 1-2 pixels, I set it to be 1 pixel in my coding
2. Set numberOfRandomDraws to be bigger than 1000, and I set it to be 1001
3. Precompute a DLT function with a matching array contains ((x, y), (x', y')) pairs as

parameter and output a 3x3 homography H

4. There will run steps 5-7 numberOfRandomDraws times
5. Randomly select a four matches seed group, which are distinct, and check through function pointsAreCollinear to see if any triple of points is collinear, if True, then randomly select another seed group
6. Compute homography H from the seed group by applying DLT function
7. Find inliers to homography H if the distance between the matching right image point and the transformed left image point after applying H is smaller than the inlierMappingThreshold
8. Return the list of homography H, the highest number of inliers and the corresponding set of inliers
9. Outside the RANSAC loop, recompute the DLT function on the returned corresponding set of inliers and get a H' which is the best homography H we found
10. Put H' with the two images inside the provided warpingPerspectiveForward function to get an array of the warping image

The plotting image plots the warping array stated in step 10. The code I wrote can only show a warping grayscale image that is not in colour; though I had tried to put the initial images as the input images, it came out with a blue version.

Given that there are many conversion parts in my RANSACLoop function, I may change it appropriately to optimize the code.

In addition, although the RANSAC loop only has guarantees if there are less than 50% outliers, if I add a line "if numOfInliers/len(matching_array) > 0.5" outside if-else statements at the end of the function (see Figure 1), my final return item does not exist, so I can only ignore this condition. Still, the final plotting image looks similar to the template.

```
if len(InlinerList) == 0:
    InlinerList.append([H, NumberOfInliners, setOfInlinerMatching])
else:
    if NumberOfInliners > InlinerList[0][1]:
        InlinerList[0] = [H, NumberOfInliners, setOfInlinerMatching]
```

Figure 1

In conclusion, the above are my thoughts and processes for the three phases. I found that some conversions and padding arrays in my code may be deleted, and for some particular parts, I need to study more in-depth to optimize them.