CS764: Assignment 4

Report

Siddharth Saha 170100025 **Parth Shettiwar** 170070021 **Parikshit Bansal** 170050040

1 Piecewise Affine Transformations

The amplitude A and the cycles completed were found out manually to be 50 and 1.5cycles $= 3\pi$ respectively. Then the input image was divided into 20 segments along the x-axis. and the angle covered too was divided into 20 parts. Each of the parts now has a corresponding range of phase which perturbs the image in that segment.

Looking at the point (0,0) in original image mapping to (0,50) in distorted image, the distortion formula can be thought of as

$$y_{distorted} = y_{original} + 50(1 + sin(\frac{3x_{original}\pi}{512}))$$

Now we use affine transform to solve this. Each segment is treated as an individual image, whose desired cooridnates are $(0,0),(0,512/20),(437,0),(437,512/20)$ and the distorted coordinates are $(ld,0),(ld,512/20),(437+rd,0),(437+rd,512/20)$, where $ld = 50(1+sin(\frac{3x_{left}\pi}{512}))$ and $rd = 50(1+sin(\frac{3x_{right}\pi}{512}))$, i.e. phase at left and right ends. We have 4 points here but affine transform needs just 3 so we can randomly pick any 3. The final images from each of the segments are then concatenated to give the final result.

The y dimension in final image size is reduced by $2A$ i.e 100 to remove all the padding black spaces.

Q : How to select grid points?

The original image is divided into 20 rectangle and the corners of the rectangles are used as grid points as expalined above.

Note : The image given is not a perfect transform. This can be seen in the mimatch between the location of sin wave peaks at the bottom and the top of the image. The same is reflected in our results. But even under these imperfect conditions our reconstructed image seems to be doing quite well.

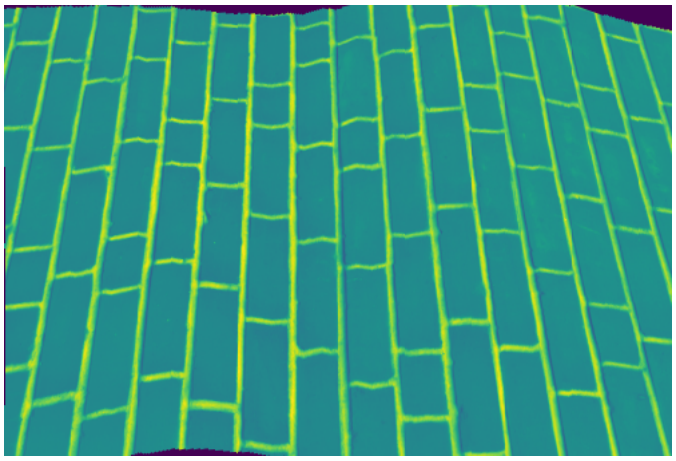


Figure 1: Recovered Brick

References used : [1]

2 Manual Mosaicing

In this Part, the aim was to simulate the camera Panaroma mode for 2 images, using manually choosen point coorrespondences. So, given 2 images, we should produce a combined single image with proper transformation of either image, keeping other image as reference.

2.1 Process

The methodology adopted was similar to the previous lab.

- First using `cv.imshow()`, `cv.setMouseCallback()` `cv2.moveWindow()` and `cv2.resizeWindow()`, we show images and finding the corner points in both images. Take atleast 5-6 points.
  Usage: when both images pop up, click a point on left image and then click its correspondence on right image. Do this for as may points as you can find. Press Enter, once done.
- We then find the perspective transform matrix keeping image 1 as reference. using `cv2.findHomography()`
- Now we offset the matrix by pre-multiplying it with an offset matrix

$$\texttt{offset} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{\texttt{Height}}{2} \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

  This is done to vertical centre align the images
- Now we just transform the image 2 using the matrix obtained from previous step using `cv2.warpPerspective`. Final image size is kept as $(2 \times Width, 2 \times Height)$.
- Finally we place our reference image at vertical centre of Output image and left aligned.
- Now we just remove the black spaces and output the smallest rectangular fit for our image. This is simply done by finding the max and min x and y non zeroo coordinates

2.2 Results

We worked on the 2 set of images given already in the folder. Furthermore we added 3 more self-clicked images from my One plus 6 phone camera of the nearby surroundings. The next page shows the results of all 5 pairs of images.
It was observed that sometimes we need more points for transformation as at less points, it incorrectly transforms with respect to reference image. Also points should be such that they cover all areas of image. On an average atleast 5-6 points were used for computing the matrix. Secondly it was observed that sometimes, the size of output image is difficult to obtain. Hence as mentioned in previous section, a fixed size of $(2 \times Width, 2 \times Height)$ was used. However as seen in the 3rd result below, still output image can have some parts lying outside these dimensions.
On an average we got a fairly good outputs in each case, with sometimes the case that seam at junction of 2 images wasnt even observed like in the 3 rd example in results below.


Shot on OnePlus


Shot on OnePlus


Shot on OnePlus

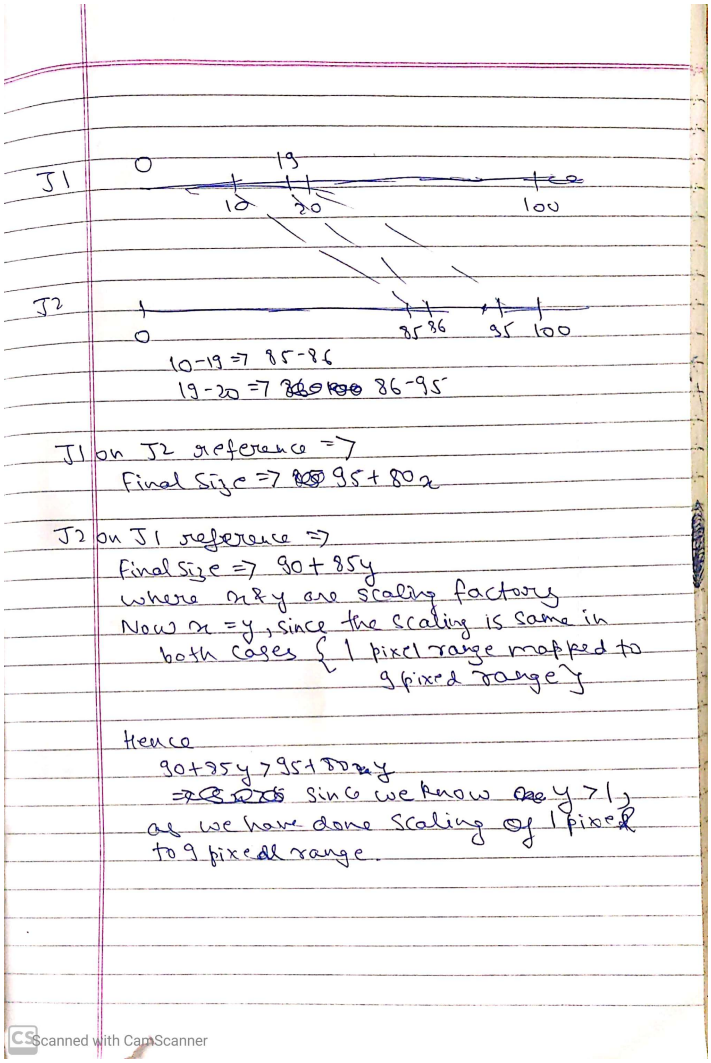

Shot on OnePlus
Shot on OnePlus
Shot on OnePlus

Figure 2: 5 pairs of images tested for manual panaroma task. Of these, second and fourth example were already given in the folder and rest were self clicked of nearby surroundings

Questions

Q.2.1: How many point-correspondences did you choose? Which OpenCV library function did you use to find the transformation in (2)? and why?
Ans As mentioned before, we used atleast 5-6 point correspondences, sometimes we used more too (Like in example 3, we used 11-12 points pairs). Since its manual task, more points are preferred and give better results. We used `cv2.findHomography()` function for the transformation in (2).
Q.2.2 Explain in the reflection essay how many did datasets you make, and why you chose the one(s) you did.
Ans Additional to the given 2 pairs of images, we captured 3 more pairs of images of nearby surroundings. The important criteria was having good feature points or in other words corner points in both images which can be mapped. Hence objects like buildings or windows or grass lawns which have strong edges and corner points were taken. If such points cant be found, then its difficult to map the points in 2 images. Hpowever this can be solved by taking more and more point pairs between 2 images (even if they are not exactly mapped).
Q.2.3: i. Consider a case in which we keep I2 as the reference image. List the ways or properties in which output after mosaicing will change and, the ways in which it will remain the same.
ii. Consider two 1-D images J1 and J2 of size 100 pixels each. Both share some common points (correspondences). J1 has correspondences in range of [10,20] and J2 has correspondences in the range of [85,95]. Will the size of the output image (after mosaicing J1 and J1) change if we keep J1 as reference image vs if we keep J2 as the reference image. Explain.
Ans i) The main difference will lie in which image will be kept over another. In our algorithm, the reference image is always kept above the other image. Hence if we swap the reference image, that image will be shown for the common portion between 2 images. Secondly now the second image will be orientated parallely to the output image and first image will be tilted and transformed to match the point correspondences. Also, the final size of output image can also be different. This can happen

since transformation matrix is different for the 2 cases and hence different effects of transformation on different images will be seen even though point corresppondes are same. For example, we observed in yard image, if we swap the reference image then the transformed image is highly elongated. Further if sizes of image are different, then there is possible scaling also happening, because of which the final output sizes will be quite different depending on what is the size of reference image.
For Similarity, we can say that a homography matrix can be found between the output images in the 2 cases. Since finally, both will be some scaled/rotated/affine or perpective transform of each other.
Ans i) The sizes of output image in2 cases can be different. This can happen for example in the case when, say pixel 10, 11 are mapped to pixels 85 and 95 respectively. Now when J1 is kept as reference image, then J2 will have shrinkage and become smalled to match the point correspondences 10 and 11. Similarly, when J2 is kept as reference image, J1 will be subjected to scaling and become elonagted to match its point correspondences to 85 and 95.
Ans ii) The sizes of output image in 2 cases can be different. This can happen for example in the case when, say pixel 10, 19 range is mapped to 85-86 range say. Followed by 19-20 range being mapped to 86-95 range. Following image, we did the calculation, to compare the lengths of 2 images. This primarlity happens due to the non-symmetry of this mapping and shrinkage and elongation in different images.



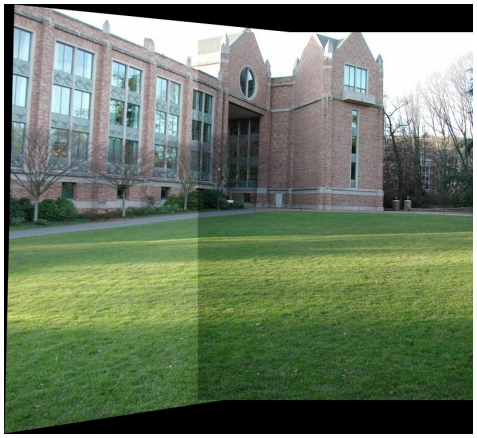
3 Auto Mosaicing

The goal here was to automate the above process.

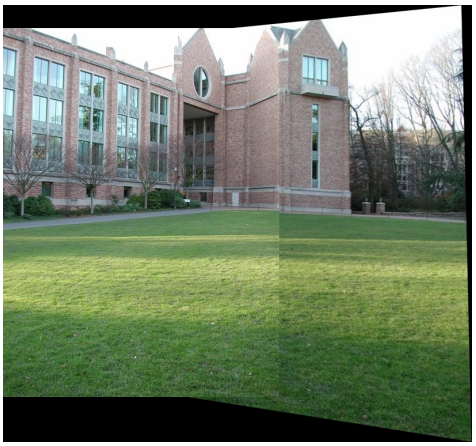
3.1 Results

(a) I1

(b) I2



(c) I1 as reference



(d) I2 as reference

Figure 3: Campus Auto Mosaicing

(a) I1 (b) I2

(c) I1 as reference (d) I2 as reference

Figure 4: Yard Auto Mosaicing

(a) I1 (b) I2

(c) I1 as reference (d) I2 as reference

Figure 5: Ledge Auto Mosaicing

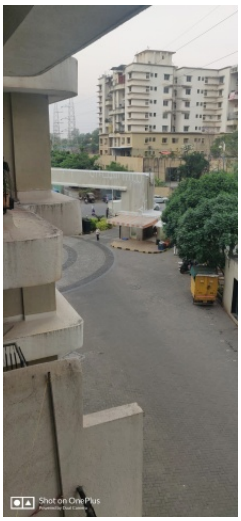
(a) I1

(b) I2

(c) I1 as reference

(d) I2 as reference

Figure 6: Society Auto Mosaicing



(a) I1

(b) I2

(c) I1 as reference

(d) I2 as reference

Figure 7: Gate Auto Mosaicing

3.2 Questions

Q.2.2.1: Did you filter? If yes, then how did you filter? How many point correspondences did you find before and after filtering?

We found the point correspondences using the Brute Force Matcher provided by OpenCV, `cv2.BFMatcher`. For each descriptor in the first set, this matcher naively finds the closest descriptor in the second set by trying each one [3]. ORB provides binary descriptors, where the descriptors are bitstrings and not numbers [4]. Thus we had to use the Hamming Norm. We enabled the `crossCheck` parameter to filter consistent pairs of correspondences.

We sorted based on the Hamming distances (metric similar to edit distance for bitstrings) and chose the top 20 matches. Even the top matches are susceptible to mis-identifying similar-looking but unrelated points to be correspondent points. These matches are treated as outliers. In order to deal with these, we used the RANSAC (Random Sample Consensus) method while constructing the homography matrix with the reprojection error threshold set as 4 [2].

Before filtering, we obtained an average of 150-200 matches. We filtered the top 20 and further subjected them to outlier elimination.

Q.2.2.2: How did you choose the values for parameters in the process of finding 'good' point-correspondences?

For filtering the sorted list of matches, we first chose all the points. This gave us frequent bad results. Our intuition is that the RANSAC outlier elimination fails to reach a consensus in the presence of several bad matches.

We chose to filter top 10 points but ran into occasional mis-behavior. Our intuition is that if the top matches are concentrated in a small patch in both the images, it fails to adapt to the perspective change in the remaining part of the overlap.

Filtering the top 20 points worked the best for us.

For choosing the RANSAC reprojection error threshold, we used the heuristic [2] that it should lie in the range 1 and 10. We explored a couple of files and `ransacReprojThreshold`=4 worked well for us.

Q.2.2.3: Did the dataset you used earlier work for the manual case work as well? Explain.

The automatic algorithm worked for the Figure 6 (Society) despite the "OnePlus" watermark at the lower left (possibly due to the tree background underplaying its resemblance).

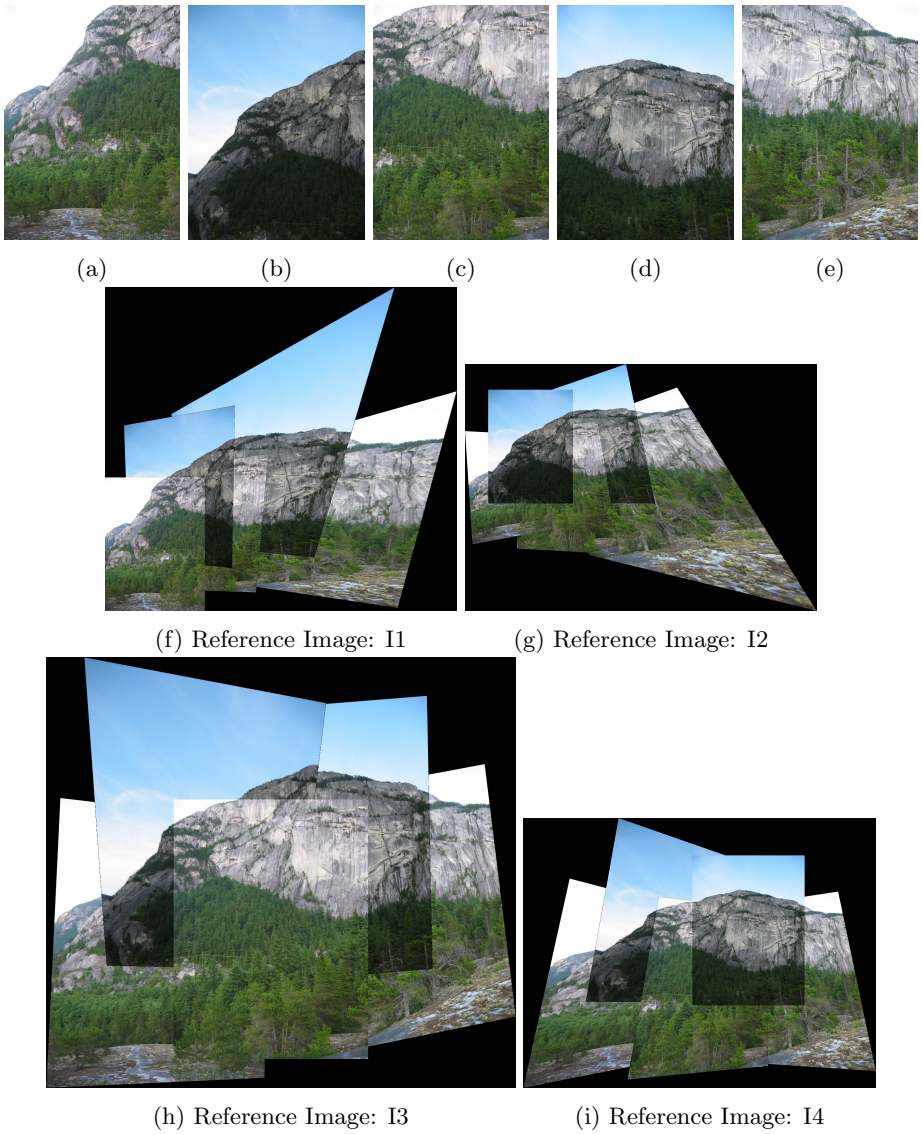
It failed for the Figure 7 (Gate). Here all the top correspondences seem to have been between elements of the "OnePlus" logo. Thus the reference image exactly overlays the transform image in both cases (c) and (d).

4 Generalized Mosaicing

In this section, we generalize to 'n' images.

4.1 Results

4.1.1 Mountain



(a) (b) (c) (d) (e)

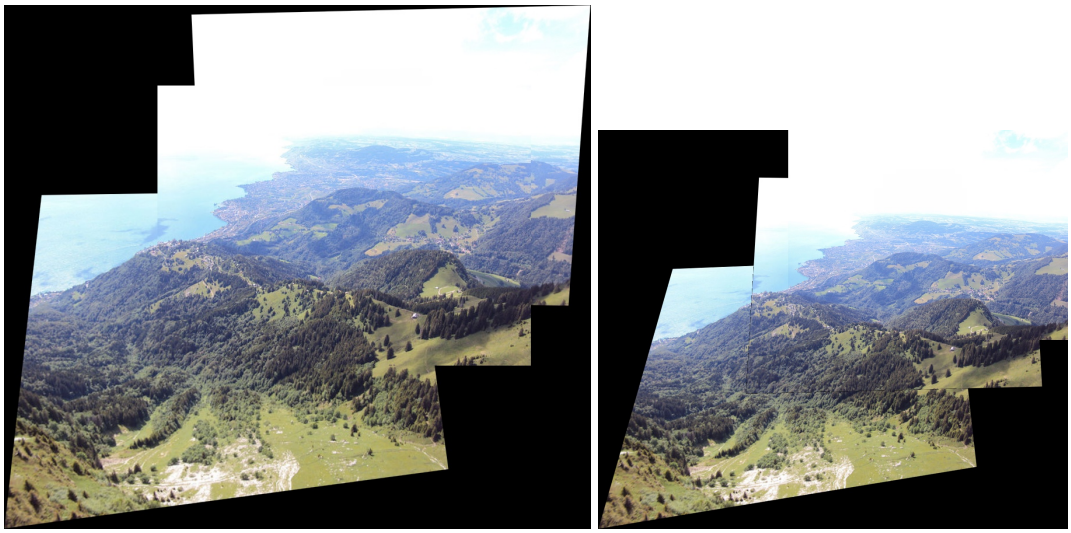
(f) Reference Image: I1 (g) Reference Image: I2

(h) Reference Image: I3 (i) Reference Image: I4

Figure 8: Mountain Generalized Mosaicing. Reference image I3 gives best result. Program crashes on taking reference image I5.

4.1.2 Ledge

(a) (b) (c)



(d) Reference Image: I1 (e) Reference Image: I2

Figure 9: Ledge Generalized Mosaicing. Reference image I1 gives best result. Program crashes on taking reference image I3.

4.1.3 Yard

(a) (b) (c) (d) (e)

(f) Reference Image: I2 (g) Reference Image: I3 (h) Reference Image: I4

Figure 10: Yard Generalized Mosaicing. Reference image I3 gives best result. Program crashes on taking reference image I1.

4.1.4 Room

(a) (b) (c)

(d) Reference Image: I1 (e) Reference Image: I2 (f) Reference Image: I3

Figure 11: Room Generalized Mosaicing. Reference image I2 gives best result.

4.1.5 Yosemite

(a) (b) (c) (d)

(e) Reference Image: I1 (f) Reference Image: I2 (g) Reference Image: I3

Figure 12: Yosemite Generalized Mosaicing. Reference image I2 gives best result. Program crashes on taking reference image I4.

4.1.6 Newspaper

(a) (b) (c) (d) (e)

(f) Reference Image: I2 (g) Reference Image: I3

Figure 13: Newspaper Generalized Mosaicing. Reference image I3 gives best result. Program crashes on taking reference image I1, I4, I5.

4.2 Questions

Q.2.3.1: Explain the method you followed. Do you get the result if you choose any image to be the reference image? If no, what issues are you facing? Give your opinion on reason behind these issues.

We iteratively run the `stitchPairOfImages()` routine defined in the `helper.py` script. We position the reference image at the center of a large blank canvas and stitch an adjacent transform image in the sequence. This mosaic image becomes the reference image for the next iteration of `stitchPairOfImages()` and we update the transform image to the next adjacent image in the sequence. (More specifics illustrated in comments in `helper.py`). This ensures that the reference image is undistorted and un-overwritten, thus staying intact.

We use a masking technique to get rid of black grooves while overlaying reference image on transformed image [6].

No, we do not get appealing results occasionally on choosing the extreme indexes as reference images. This owes to error propagation in the perspective transform. If one transform image has moderate skew error, the next adjacent transform image gets heavily skewed. The program crashes as either the demands for the blank canvas dimensions exceeds the system's capacity or the perspective transform matrix itself runs into a singularity.

Q.2.3.2: Here, we assumed that you know the sequence in which you have to stitch the images. How would you modify your approach if you don't know the sequence? Describe the approach briefly.

A brute force technique can be used to know the best sequence for stitching the images. We can run for $\binom{n}{2}$ (where n is the number of images to be mosaiced) combinations of pair of images and check whether `cv2.BFMatcher` gives good mappings of feature points between 2 images and set `crossCheck` param to be true. This will ensure that the two features in both images match each other. Further we will check the best matching Hamming distance value and set a threshold to consider this pair of images having any point correspondence. If the distance is very high, then we will safely assume that pair of images dont have any point correspondence. Once we know all the possible pairs of images which have any overlap or point correspondences based on above process, we will just take the best (n-1) pairs from this set (based on Hamming distances of best matchers points) to cover all images and then run our algo to make our final mosaiced image.

Q.2.3.3: How does the functionality you are providing (or could provide) differ from the Stitcher class.

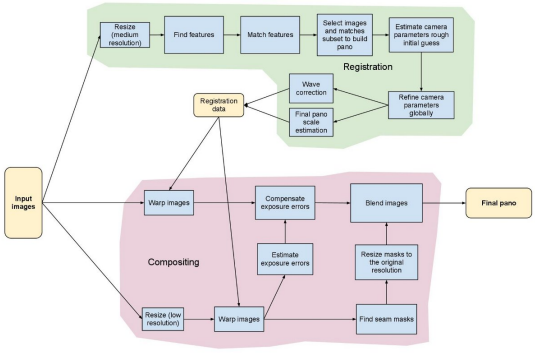


Figure 14: Stitcher class pipeline [7]

The differences between our functionalities from the Stitcher class are as follows:

1. The class does a preliminary resize to medium resolution to find and match features. We have directly worked with the input image dimensions.

2. Their approach involves multi-step estimation of camera parameters going from a rough initial guess and refining it in steps.

3. They rescale the input images to low resolution to obtain seam masks which they use to blend the stitched images. We have suggested a rudimentary Gaussian blur for the same.

Q.2.3.4: The mosaicing results show a "seam". What techniques can be used to remove the seam?

The seam between 2 images mainly come due to the different lighting conditions between 2 images. For removing this, we can use the Histogram matching technqiue to match the histograms of 2 images and hence equalise the lighting conditions. This can be followed by applying a gaussian filter on the "seam" line to blur it as we know, gaussian filter acts as a Blurring filter.

References

[1] https://docs.opencv.org/master/

[2] https://docs.opencv.org/master/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780

[3] https://docs.opencv.org/4.5.1/d3/da1/classcv_1_1BFMatcher.html#ac6418c6f87e0e12a88979ea57980c020

[4] https://answers.opencv.org/question/147525/which-norm-is-the-best-to-match-descriptors/

[5] https://stackoverflow.com/questions/13538748/crop-black-edges-with-opencv

[6] https://stackoverflow.com/questions/2169605/use-numpy-to-mask-an-image-with-a-pattern

[7] https://docs.opencv.org/3.4/d1/d46/group__stitching.html