CS543/ECE549 Assignment 1

Name: Hongbo Zheng
NetId: hongboz2

Part 1 : Implementation Description

Single-scale alignment with SSD, NCC, SSD of edges, NCC of edges

The single-scale alignment with SSD, NCC, SSD of edges, NCC of edges can be summarized as the following 5 steps. The algorithm is tested with 4 different loss functions SSD, NCC, SSD of edges, NCC of edges. The edge variant loss function is performed by `cv2.Canny()` which captures the edges from the original images [1].

1. Load Image
   Read the image using `PIL.Image`

2. Border Removal
   a. Border Identification based on Pixel Value Threshold
      i. During border removal, the algorithm inspects each row of the image, and it keeps track of the pixel positions that exceed a predefined white pixel value threshold or fall below a set black pixel value threshold, both of which are configured in `config.py`
   b. Record Border Candidates
      i. While scanning pixels within the border search range configured in `config.py`, the algorithm identifies the maximum and minimum y-indices that meet the established threshold conditions. These maximum and minimum y-indices are stored separately in candidate lists for the left and right borders.
   c. Border Determination
      i. To obtain the definitive borders, the algorithm selects the most frequently proposed value from each candidate list. The y-index with the highest frequency in the left border candidate list becomes the left border. Similarly, the y-index with the highest frequency in the right border candidate list becomes the right border.
   d. Top and Bottom Border Removal
      i. The above algorithm is designed to remove the undesired left and right borders from an image. The top and bottom border removal algorithm follows a similar approach but focuses on column search.

3. Vertical Image Separation
    a. Initial Separation Estimation
        i. The algorithm begins by making an initial estimation for the vertical separations of the top two images. It divides the image height by three (`image_height // 3`) to create a rough estimate of where the separation might occur.
    b. Search Border Within Threshold
        i. Within a predefined border search threshold configured in `config.py`, the algorithm conducts a search both upwards and downwards from the estimated separation point.
    c. Candidate Dictionary Creation
        i. For each row within the defined search range, the algorithm calculates the sum of pixel values over a narrow pixel range centered around the row's midpoint. These sum values are then stored in a candidate dictionary, with x-index of the row as the key and the corresponding sum of pixel values as the value.
    d. Optimal Separation Row
        i. To find the optimal separation between the top two images, the algorithm identifies the x-index with the lowest sum of pixel values (pixel values that are really close to black) within the candidate dictionary. This x-index indicates the most suitable row to separate the top two images effectively.
    e. Subsequent Separation
        i. For the bottom two images, the algorithm uses the same approach with a different initial estimation which is `(image_height // 3) * 2`
    f. Image Resizing
        i. The image resizing algorithm first identifies the minimum width and height among the three cropped images. These dimensions serve as the reference for resizing. Rather than merely cropping from the top or bottom of the image, the resizing algorithm resizes the images by removing any extra rows of columns on both sides.

4. Best Displacement Search Algorithm
    a. Exploration of Best Channels
        i. The algorithm iterates through the color channels (blue, green, and red) one by one treating each as a candidate base channel.
    b. Displacement Score Map
        i. For each base channel iteration, it calculates the displacement required for aligning the two comparison channels with the base channel. A tuple of base channel and two displacements are inserted into the displacement score map as a key, and the score from the loss function is recorded as its corresponding value.
    c. Selection of Best Displacement
        i. All displacement tuples, along with their corresponding scores, are stored in a dictionary. The algorithm selects the displacement tuple with the

highest score (in the case of Normalized Cross-Correlation NCC), or the lowest score (in the case of Sum of Squared Differences SSD).

5. Image Stacking and Composition
    a. Displacement Alignment
        i. The algorithm begins by identifying the most suitable base channel among the three color channels. The chosen base channel serves as a reference for alignment. Using the selected base channel and its associated displacement information, the algorithm aligns the remaining two channels which ensures the all three channels share a common frame of reference.
    b. Cropping and Stacking Channels
        i. Once the overlapping region is determined, the algorithm performs cropping operation on each channel to retain the maximum overlapping region by the three color channels. Finally, the blue, green, and red channels are stacked together to construct the final image.

Multiscale alignment with SSD, NCC

The multiscale alignment with SSD and NCC is similar to the algorithm described above. The only difference is that the image pyramid algorithm is used to search for the best displacement in Step 4: Best Displacement Search Algorithm.

Image pyramid Algorithm:

The image pyramid recursive function is implemented as follow. Recursively call the function until the number of pyramid levels is decremented to 0. Then, the algorithm is going to search for the displacement of the downsized images. After that, the displacement result is returned, and `numpy.roll()` is performed on the compare channel to align with the base channel image. Since the displacement is downsized by 2, the displacement values are multiplied by 2 and return the previous level of image pyramid. The previous three steps are repeated until the function finishes.

What implementation choices did you make, and how did they affect the quality of the result and the speed of computation?

The Border Removal and Image Vertical Separation steps help increase the quality of the final results. The final results are free with unwanted white or black borders and have relatively larger overall image area because it crops out each channel image with its original size. For the single-scale alignment algorithm, Image Stacking and Composition algorithm is performed rather than simply called `np.roll()` for the other two channels. In this way, there will not exist some weird colors at the border of the result images.

Additionally, the Border Removal and Image Vertical Separation steps for low-resolution images are actually light weighted which barely slow down the speed of the computation.

What are some artifacts and/or limitations of your implementation, and what are possible reasons for them?

The Border Removal and Image Vertical Separation steps actually slows down the speed of the computation of the high-resolution images because there are too many pixels to search for. However, these two steps actually help produce very good results in the end.

Furthermore, the Border Removal algorithm does not work too well for the high-resolution woman image because the black pixel color located at the top and right sides of the original image is not pure. The color tends to be more grayish, therefore, the Border Removal algorithm does not detect the borders.

Part 2: Basic Alignment Outputs

A: Channel Offsets

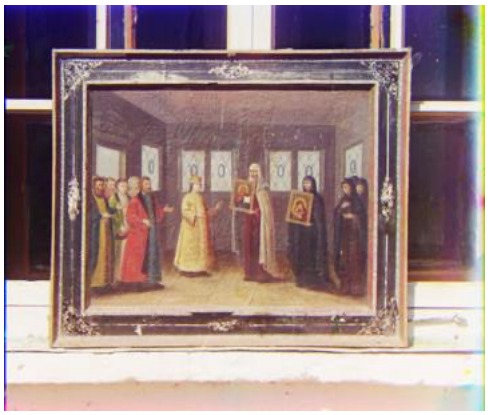
Single-scale Alignment with SSD		
Image (base channel)	(h,w) offset (channel)	(h,w) offset (channel)
00125v.jpg (G)	(-10, -10) (B)	(-6, -1) (R)
00149v.jpg (R)	(6, -2) (B)	(9, 0) (G)
00153v.jpg (R)	(2, -4) (B)	(-2, -2) (G)
00351v.jpg (B)	(-6, 0) (G)	(-10, -1) (R)
00398v.jpg (B)	(-2, -1) (G)	(-10, 10) (R)
01112v.jpg (B)	(-2, -10) (G)	(-10, -10) (R)

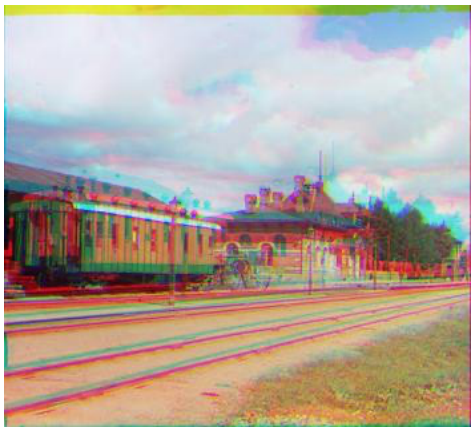
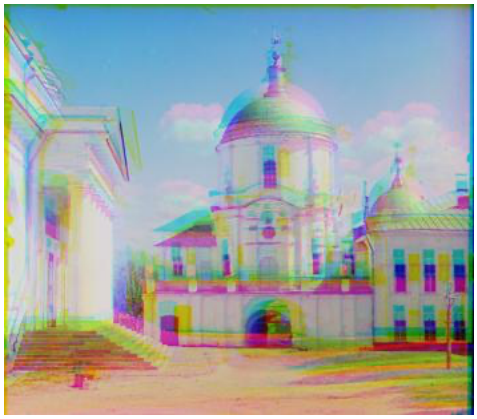
Single-scale Alignment with NCC		
Image (base channel)	(h,w) offset (channel)	(h,w) offset (channel)
00125v.jpg (G)	(1, -2) (B)	(-7, 1) (R)
00149v.jpg (G)	(-3, -2) (B)	(-9, 0) (R)
00153v.jpg (G)	(4, -3) (B)	(1, 2) (R)
00351v.jpg (G)	(6, 0) (B)	(-7, 1) (R)
00398v.jpg (G)	(0, -2) (B)	(-1, 1) (R)
01112v.jpg (G)	(10, 0) (B)	(3, 1) (R)

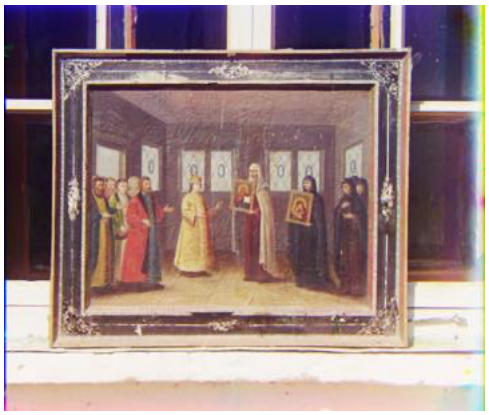
Single-scale Alignment with SSD Edges		
Image (base channel)	(h,w) offset (channel)	(h,w) offset (channel)
00125v.jpg (B)	(-2, 2) (B)	(-7, 1) (R)
00149v.jpg (R)	(6, -2) (B)	(9, 0) (G)
00153v.jpg (R)	(3, -4) (B)	(-1, -2) (G)
00351v.jpg (R)	(10, -1) (B)	(7, -1) (G)
00398v.jpg (G)	(0, -3) (B)	(-1, 1) (R)
01112v.jpg (G)	(10, 0) (B)	(3, 1) (R)

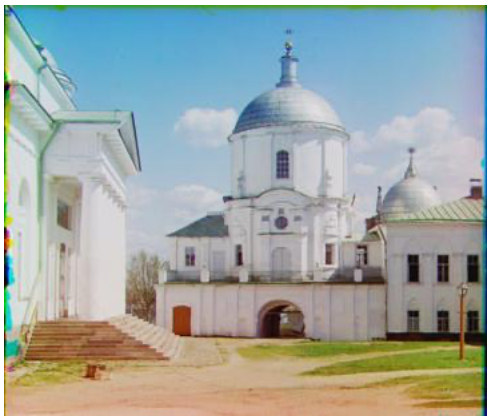
Single-scale Alignment with NCC Edges		
Image (base channel)	(h,w) offset (channel)	(h,w) offset (channel)
00125v.jpg (R)	(7, -1) (B)	(6, 1) (G)
00149v.jpg (G)	(-3, -2) (B)	(-9, 0) (R)
00153v.jpg (G)	(4, -3) (B)	(1, 2) (R)
00351v.jpg (G)	(6, -1) (B)	(-7, 1) (R)
00398v.jpg (G)	(0, -3) (B)	(-1, 1) (R)
01112v.jpg (G)	(10, 0) (B)	(3, 1) (R)

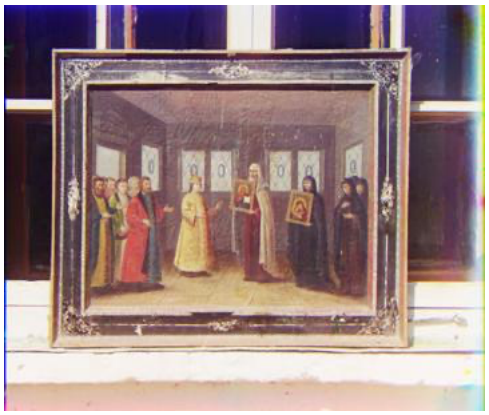
B: Output Images

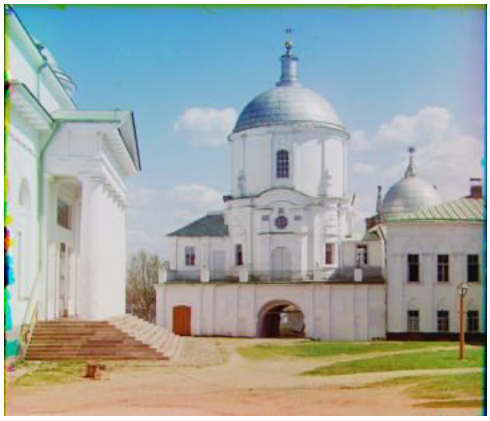
Single-scale Alignment with SSD	
Image (base channel)	jpeg file
00125v.jpg (G)	
00149v.jpg (R)	
00153v.jpg (R)	

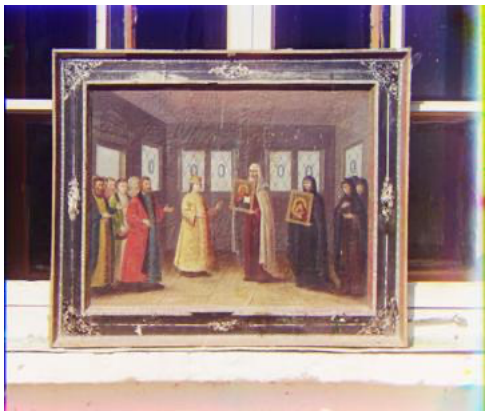
00351v.jpg (B)	
00398v.jpg (B)	
01112v.jpg (B)	

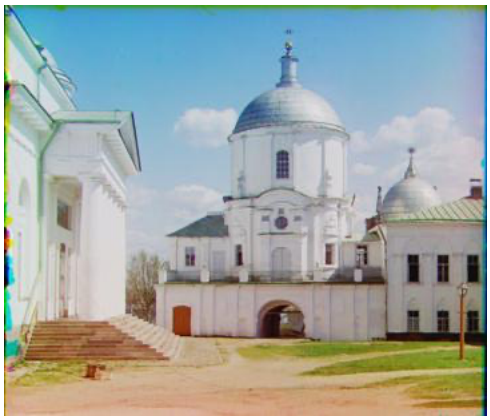
Single-scale Alignment with NCC	
Image (base channel)	jpeg file
00125v.jpg (G)	
00149v.jpg (G)	
00153v.jpg (G)	

00351v.jpg (G)	
00398v.jpg (G)	
01112v.jpg (G)	

Single-scale Alignment with SSD Edges	
Image (base channel)	jpeg file
00125v.jpg (B)	
00149v.jpg (R)	
00153v.jpg (R)	

00351v.jpg (R)	
00398v.jpg (G)	
01112v.jpg (G)	

Single-scale Alignment with NCC Edges	
Image (base channel)	jpeg file
00125v.jpg (R)	
00149v.jpg (G)	
00153v.jpg (G)	

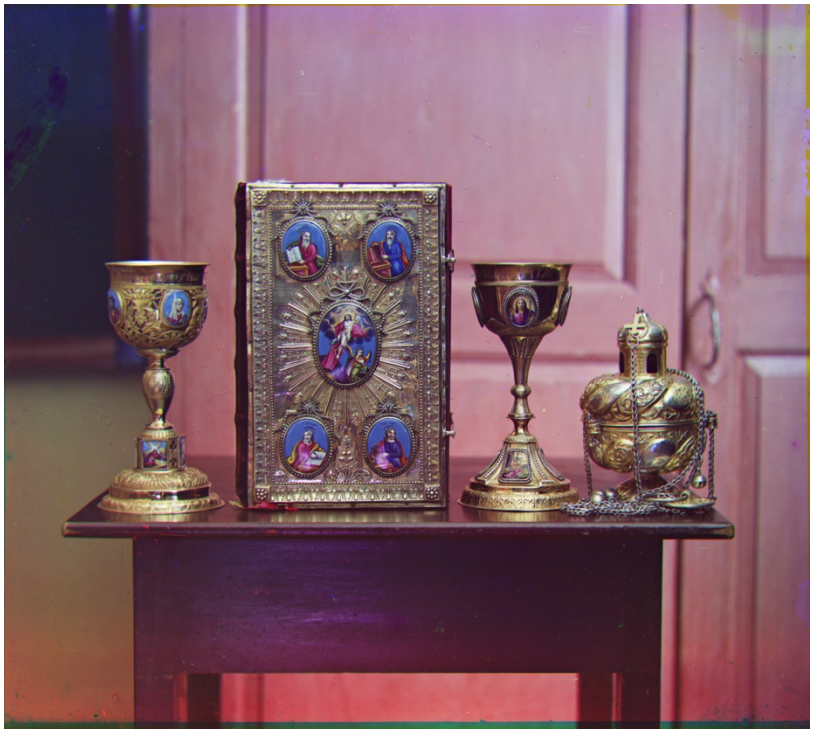
00351v.jpg (G)	
00398v.jpg (G)	
01112v.jpg (G)	

Part 3: Multiscale Alignment Outputs

A: Channel Offsets

Multiscale Alignment with NCC		
Image (base channel)	(h,w) offset (channel)	(h,w) offset (channel)
01047u.tif (G)	(37, -21) (B)	(5, 13) (R)
01657u.tif (G)	(109, -4) (B)	(13, 2) (R)
01861a.tif (G)	(28, -38) (B)	(31, 22) ®

B: Output Images

Multiscale Alignment with NCC	
Image (base channel)	jpeg file
01047u.tif (G)	

01657u.tif (G)	
01861a.tif (G)	

C: Multiscale Running Time improvement

Single-Scale Alignment	
Average Run-time of Low-Res Images	0.2303555965s

Low-Resolution Image Shape (W x H)	
Average Shape of All Low-Res Images	1024 x 396.1666667

High-Resolution Image Shape (W x H)	
Average Shape of All High-Res Images	9659.6667 x 3747.6667

Multiscale Alignment	
Estimate Average Run-time of High-Res Images	72.73564928s

Multiscale Alignment	
Actual Average Run-time of High-Res Images	20.55625429s

$$\frac{20.55625429\text{s} - 72.73564928\text{s}}{72.73564928\text{s}} \times 100\% \approx -71.7384\%$$

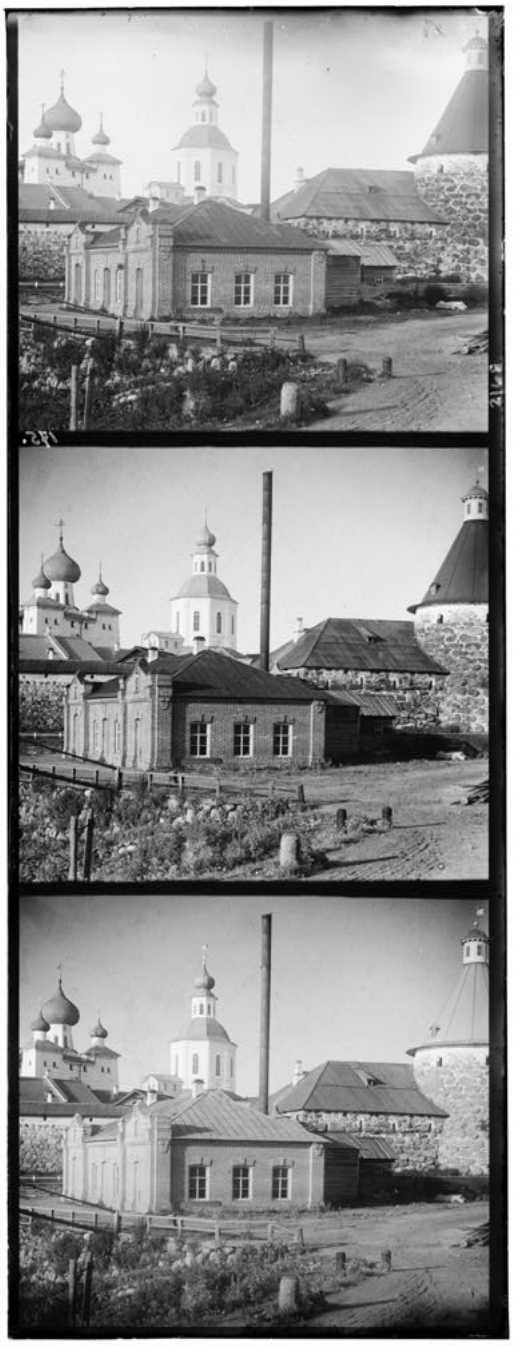
Decrease -71.7384% amount of the single-scale run-time.

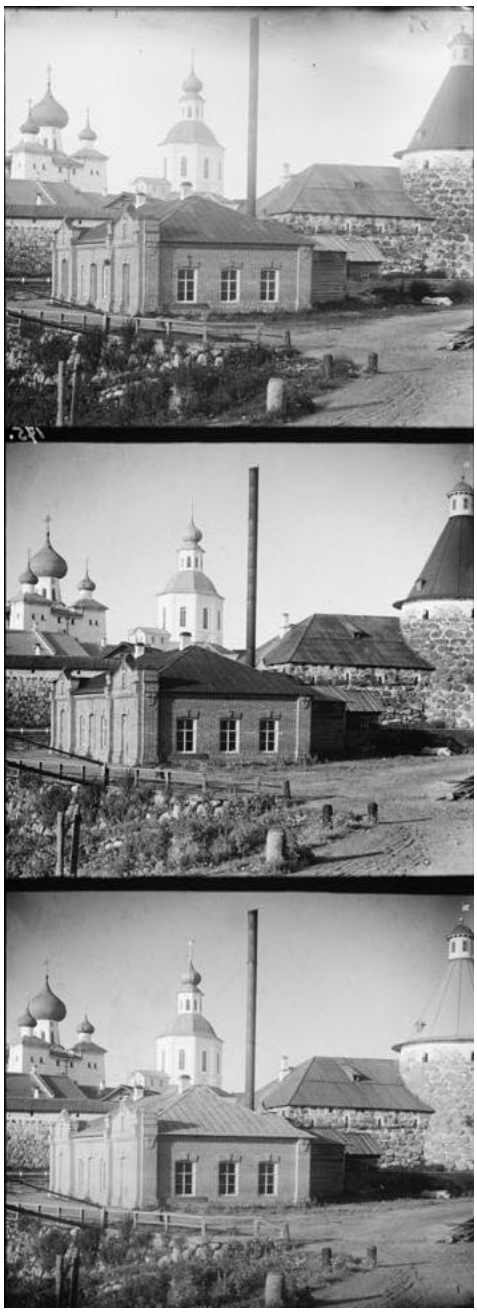
Part 4 : Bonus Improvements

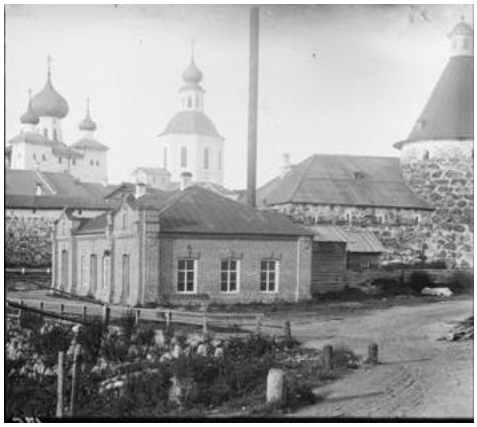
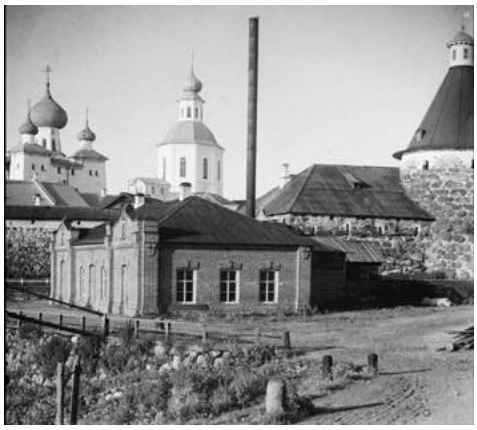
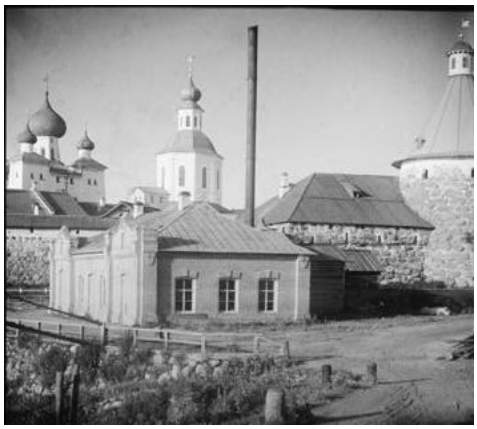
The Border Removal, Vertical Image Separation, and Image Stacking and Composition steps are implemented to improve the quality of the output. The detailed implementation is described in Part 1: Implementation Description.

The SSD of edges and NCC of edges results are shown above.

Bonus Implementation Example Outputs (jpeg format)

00351v.jpg Original Image	

00351v.jpg Border Removal Image (Left & Right)	

00351v.jpg (G) Vertical Image Separation & Border Removal Image (Top & Bottom) Stack in order of B, G, R	  

00351v.jpg (G) Result Image	

Reference

[1] https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html