CS543 Computer Vision - Assignment 0

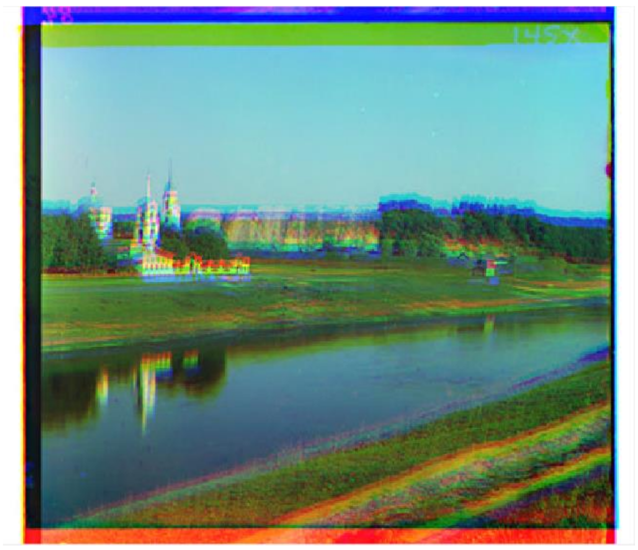
Yu Tao | yutao2@illinois.edu

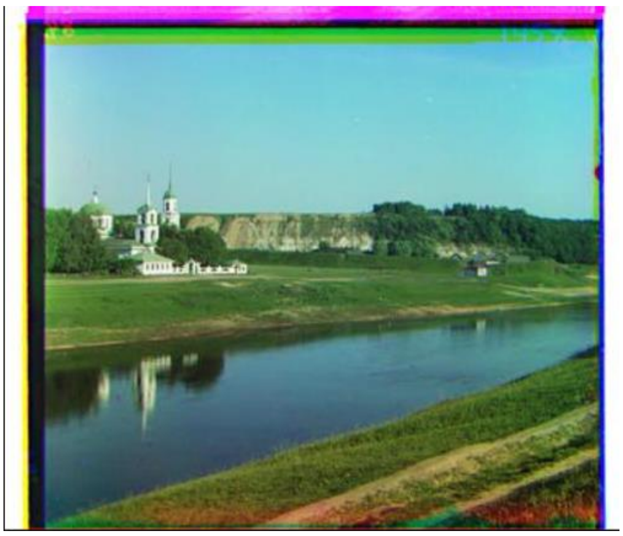
1. Basic Implementation

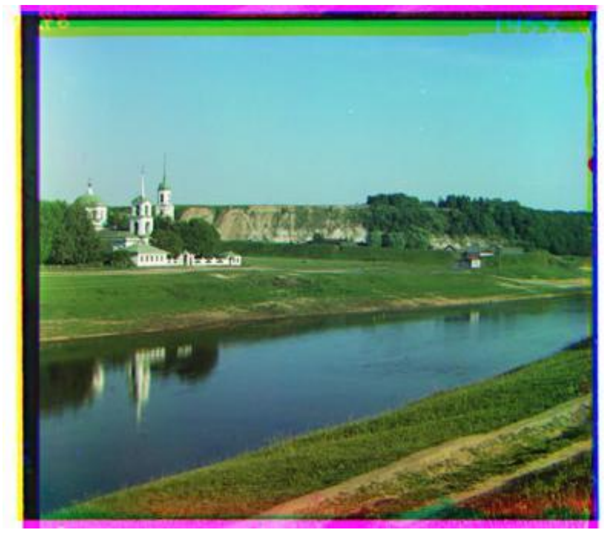
1.1 SSD

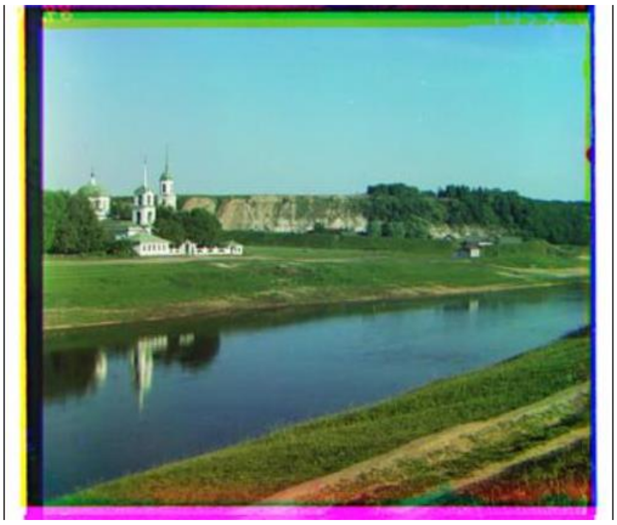
Details:
My implementation started by splitting the input image into three parts, which are B, G, R channels respectively. To align three channels, the program moves the second two channels – G and R, by [-15, 15] pixels in both x and y directions in a brute-force way, and calculate the SSD of two channels. After the above computation, pick the best one and apply the offset to the G and R channels. Lastly, use cat function to merge three channels and output the image. But there is still some blur in the images, especially in 3.jpg. Three align order give out images with kind of equal quality. Most of the time it is hard to tell the differences between those images but sometimes aligning red and blue channels to the green are better than the other two. (see the third image group)

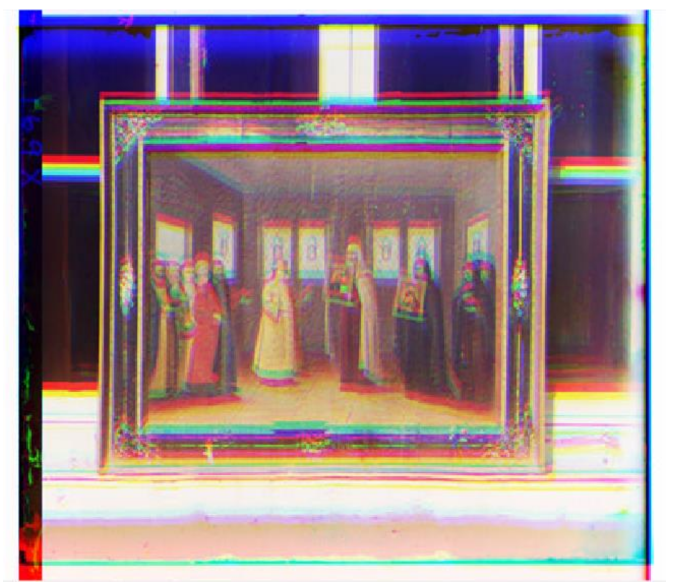
Output: Initial displacement vs aligned by SSD


Initial displacement

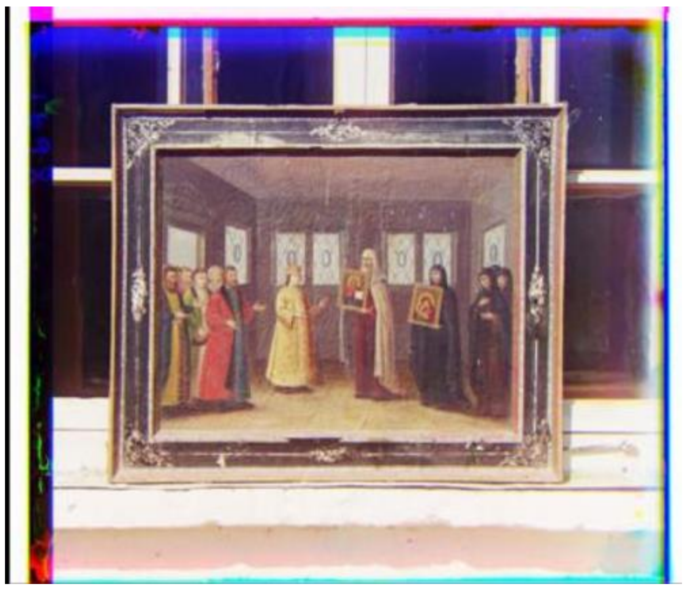

Green: [5,2]; Red: [9,1]


Red: [4,-1]; Blue: [-5,-2]

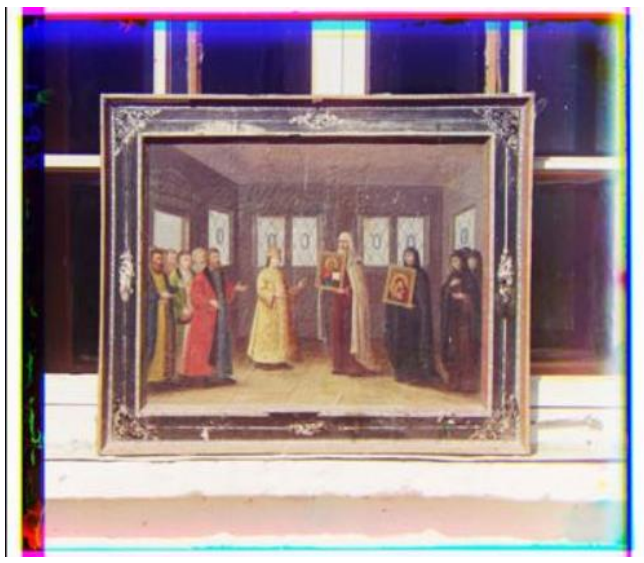

Green: [-4,1]; Blue: [-9,-1]



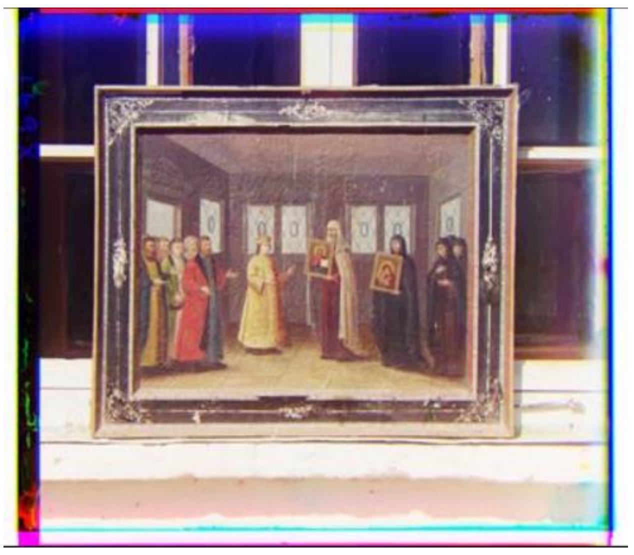
Initial displacement



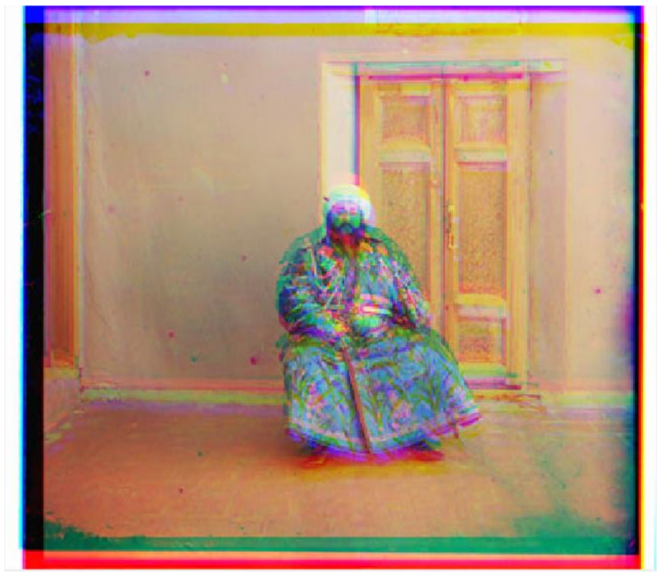
Green: [4,2]; Red: [9,2]

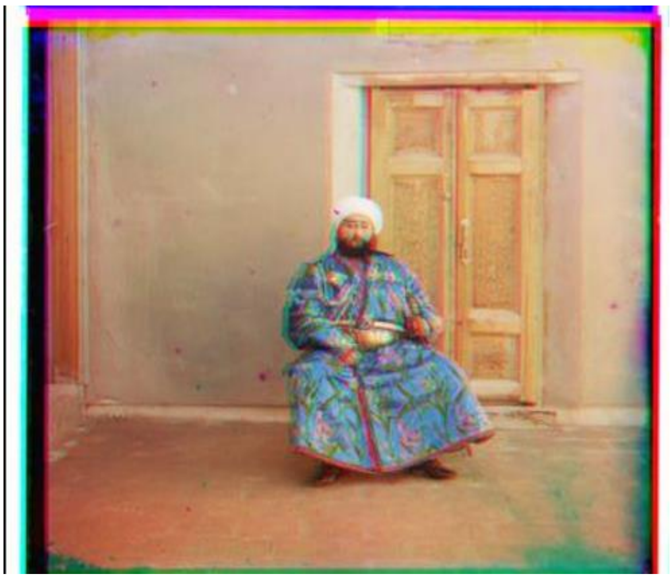


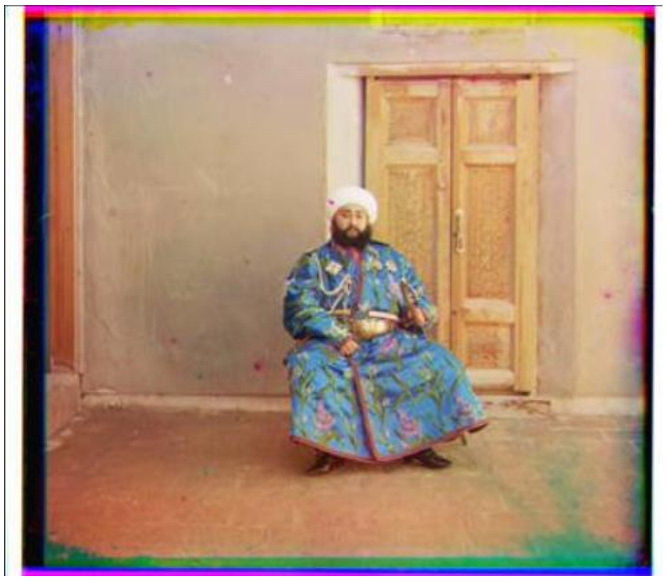
Red: [5,0]; Blue: [-4,-2]

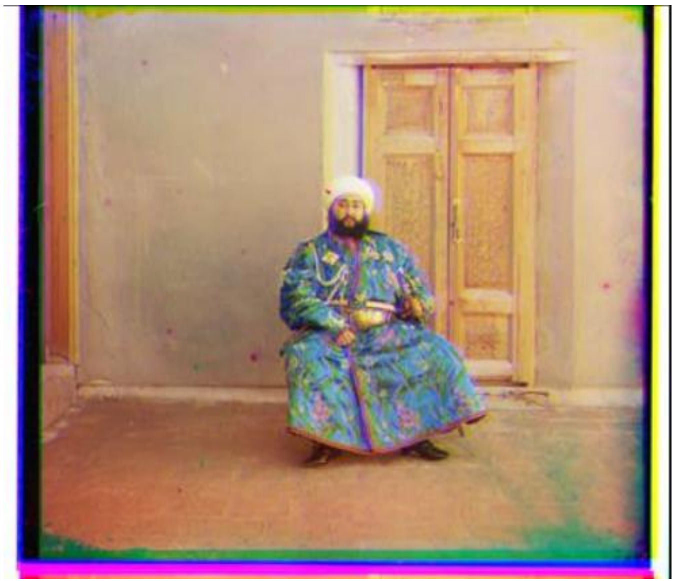


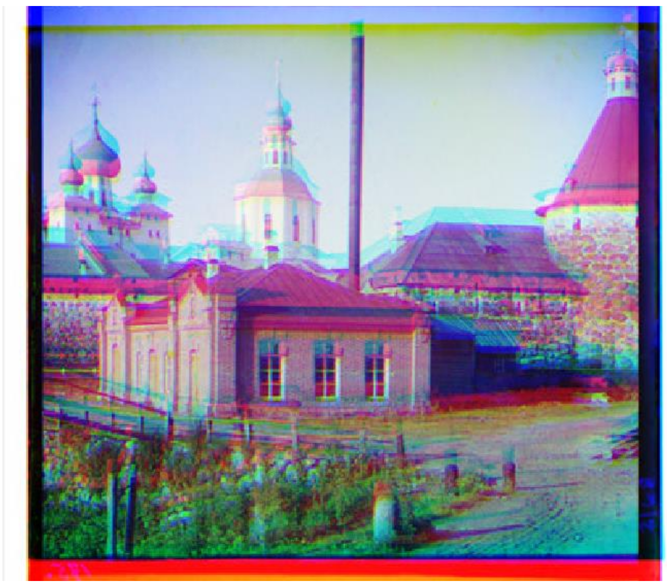
Green: [-5,0]; Blue: [-9,-2]

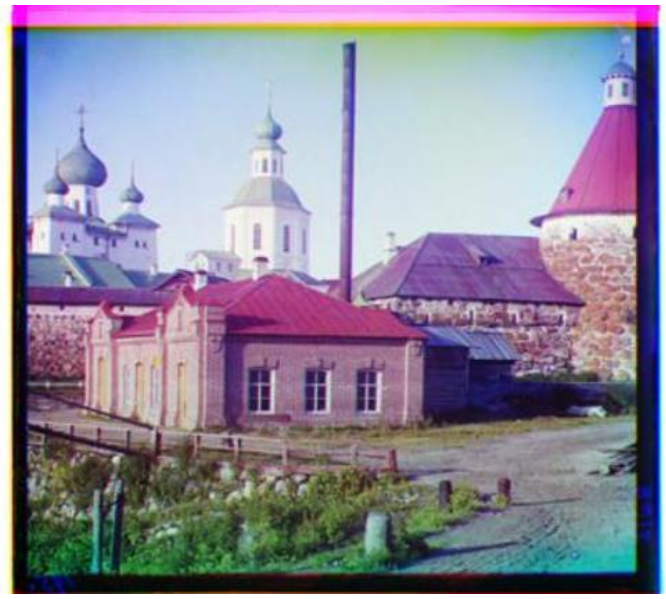

Initial displacement

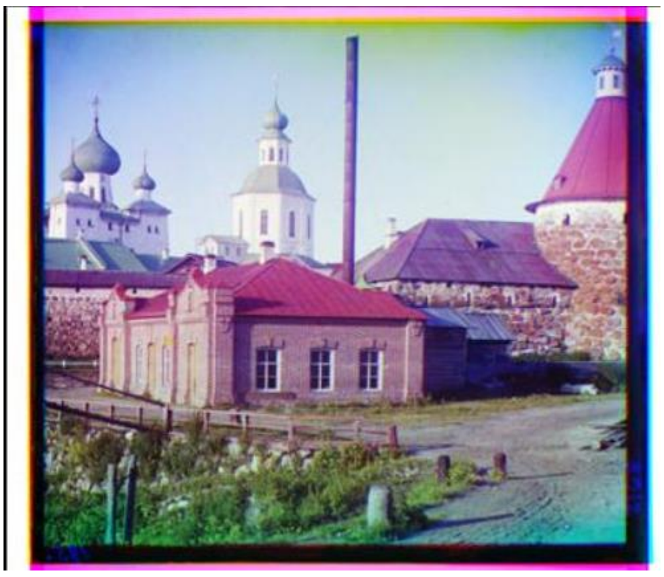

Green: [7,2]; Red: [14,0]

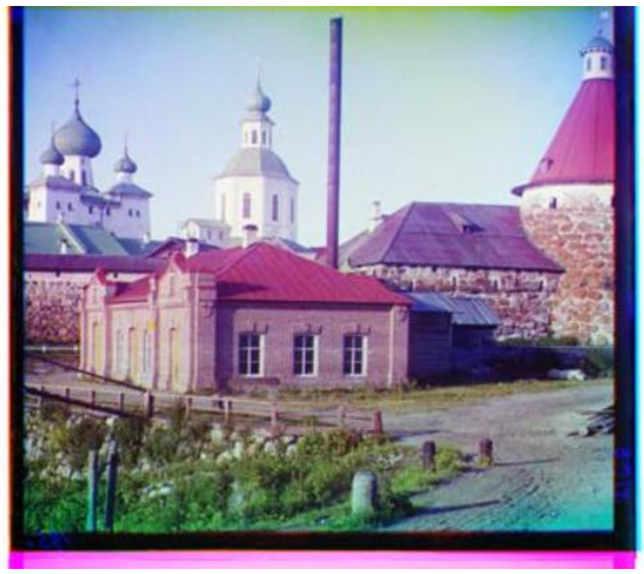

Red: [7,2]; Blue: [-7,-2]

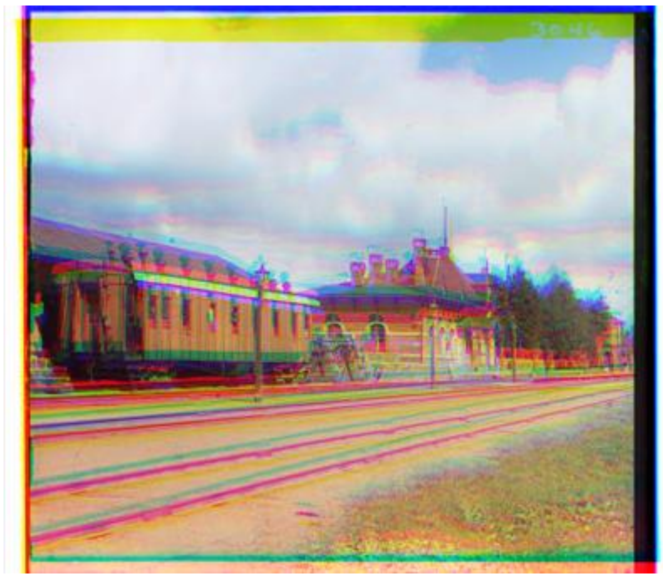

Green: [-7,-2]; Blue: [-14,0]


Initial displacement

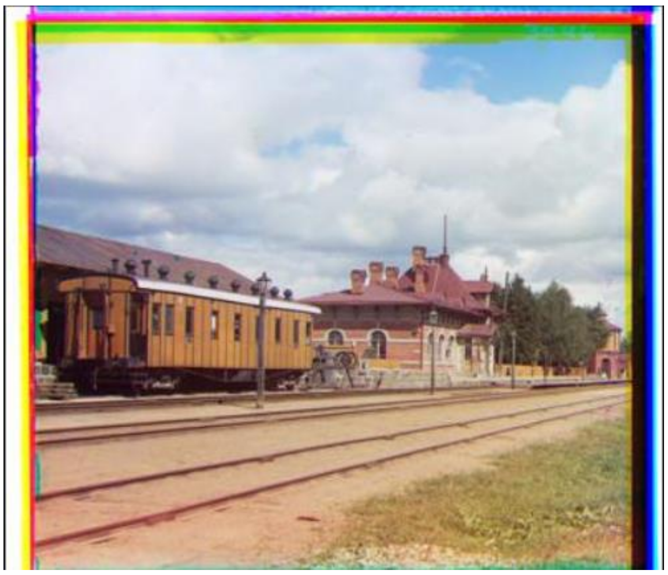

Green: [4,1]; Red: [13,1]


Red: [9,1]; Blue: [-4,-1]

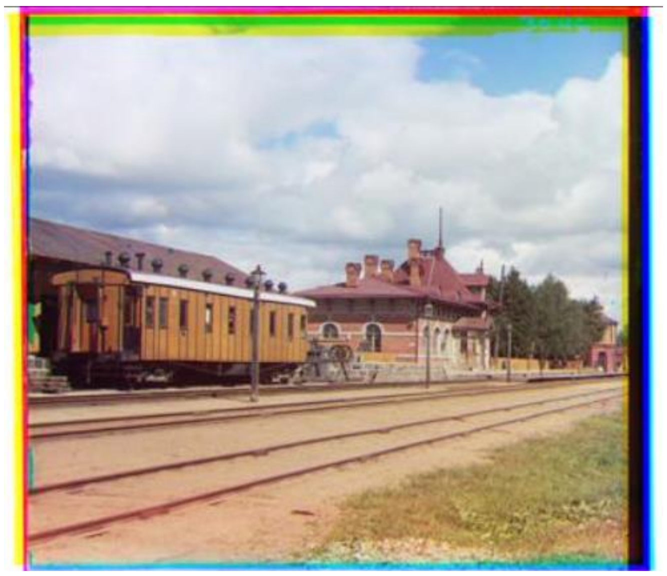

Green: [-9,-1]; Blue: [-13,-1]



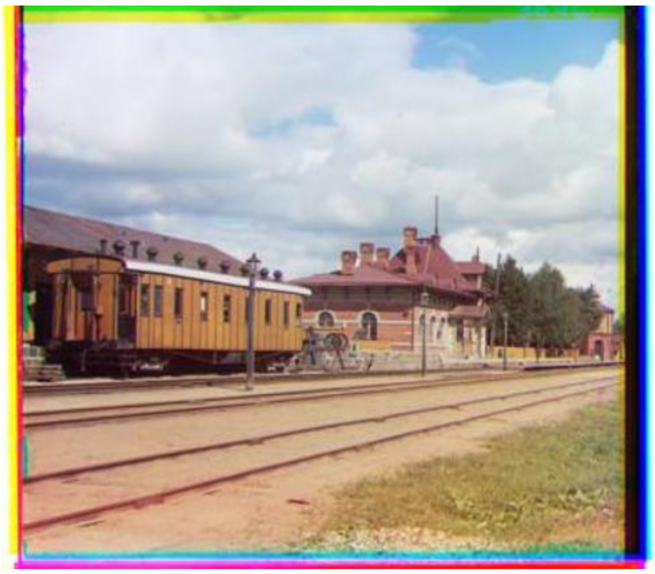
Initial displacement



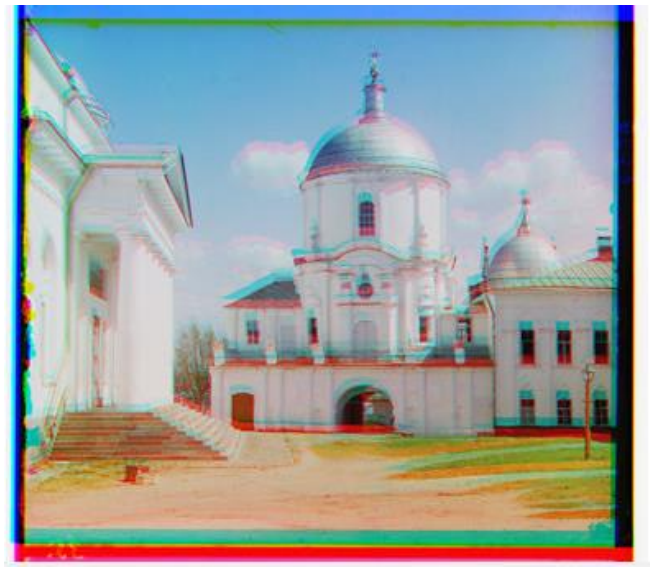
Green: [5,3]; Red: [11,4]

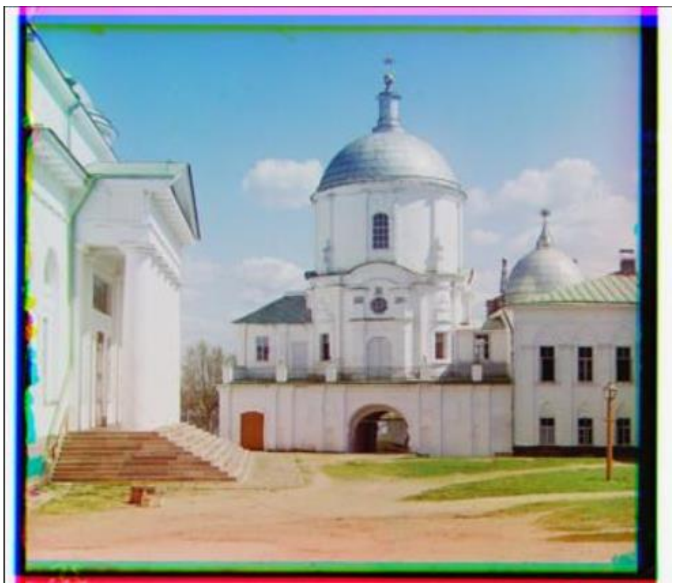


Red: [6,1]; Blue: [-5,-3]

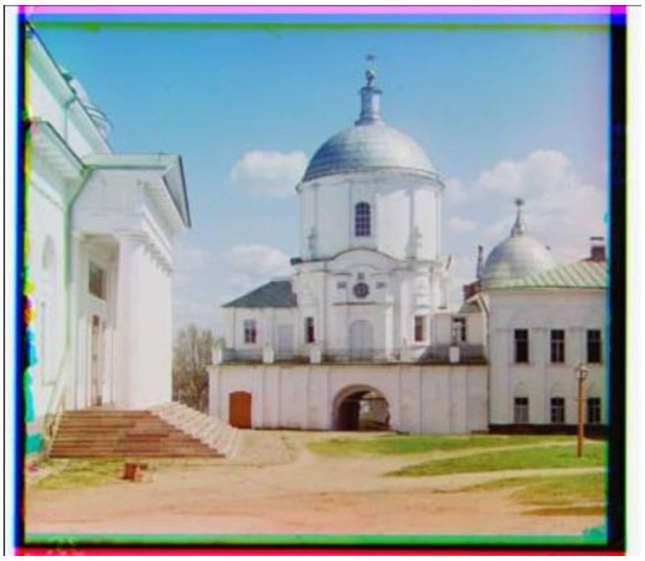


Green: [-6,-1]; Blue: [-11,-4]

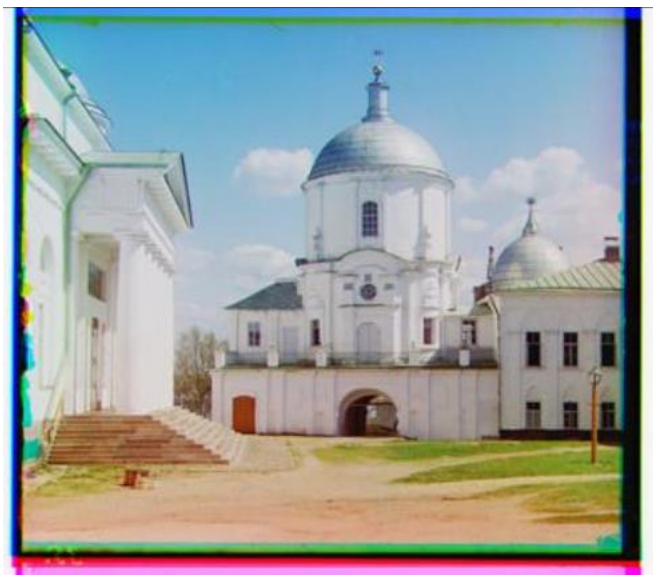




Initial displacement



Green: [0,0]; Red: [5,1]



Red: [5,1]; Blue: [5,1]

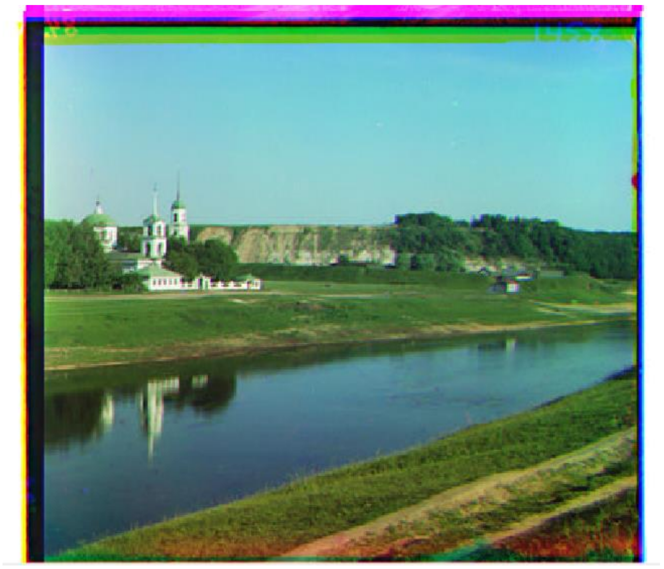
Green: [0,0]; Blue: [-5,-1]

1.2 NCC

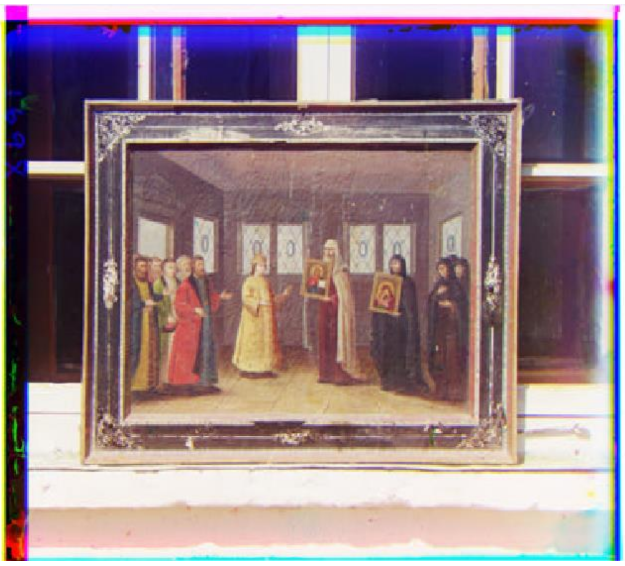
Details:
Roughly the same as SSD but use normalized cross-correlation to find the best offset.

Output: aligned by NCC

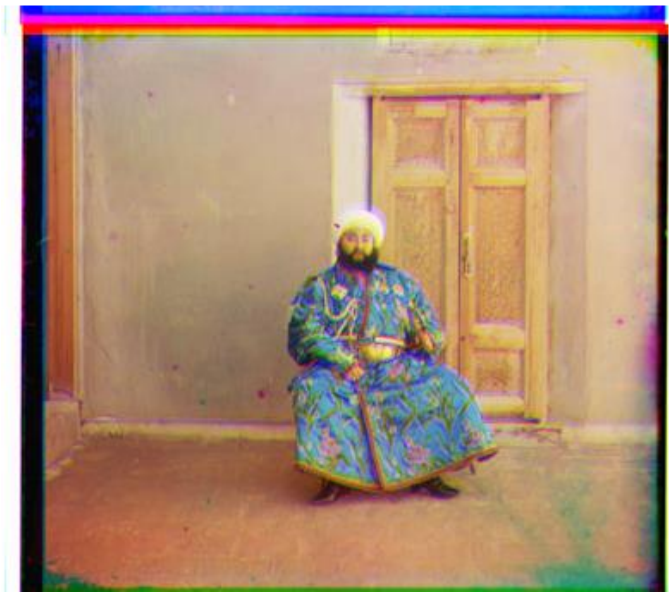
Green: [5,1]; Red: [9,1]



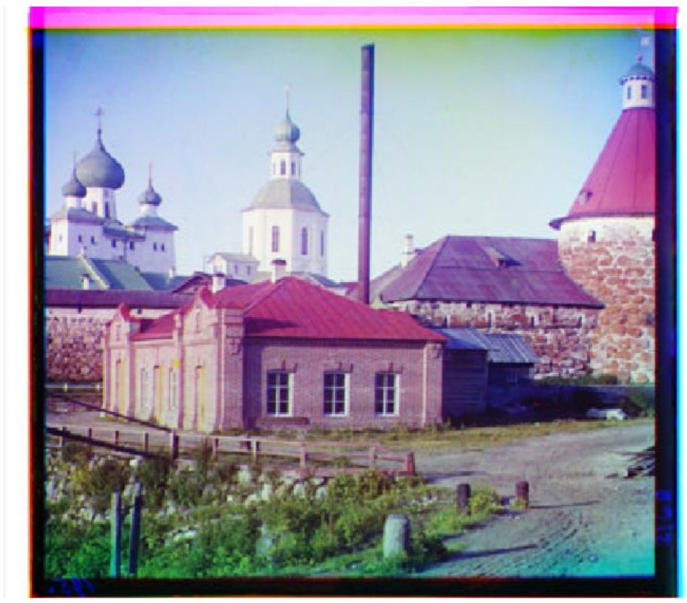
Green: [4,2]; Red: [9,2]



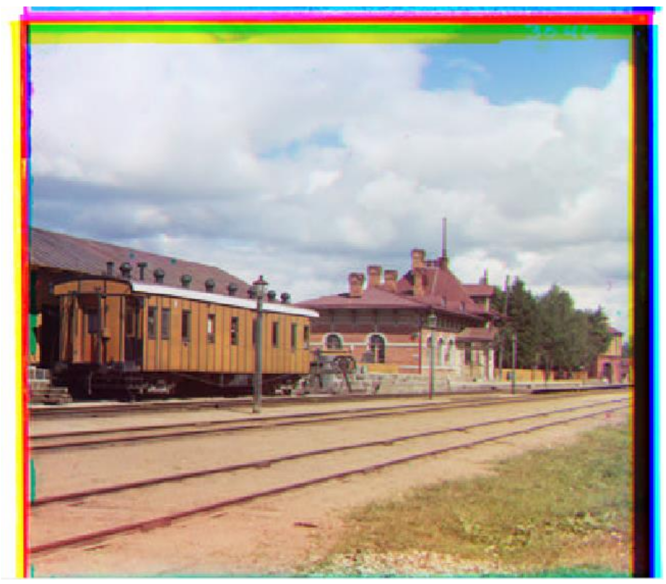
Green: [11,1]; Red: [17,3]



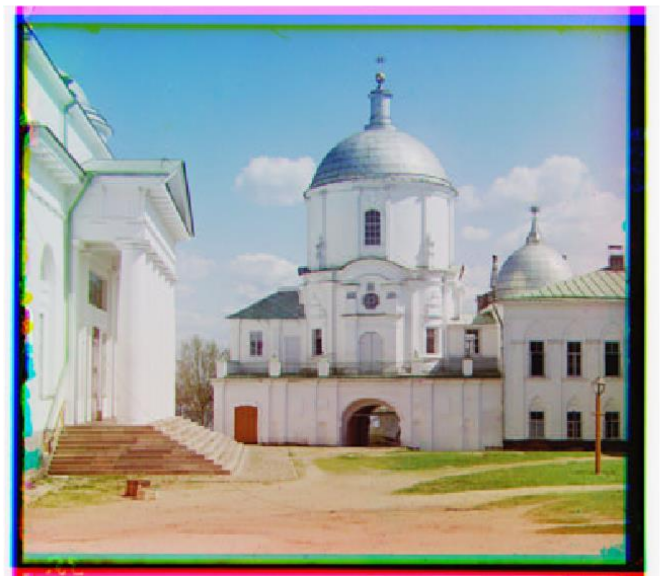
Green: [4,0]; Red: [13,1]



Green: [5,2]; Red: [11,4]



Green: [0,0]; Red: [5,1]



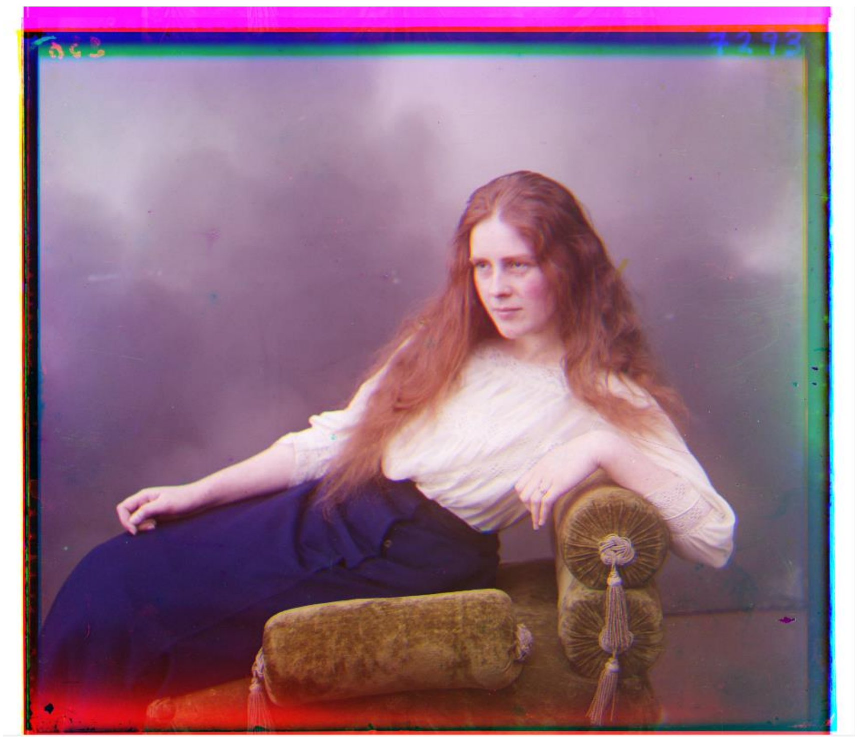
2. Improvements (Each section in 2 is for extra credit, 4 in all)

2.1 Pyramid

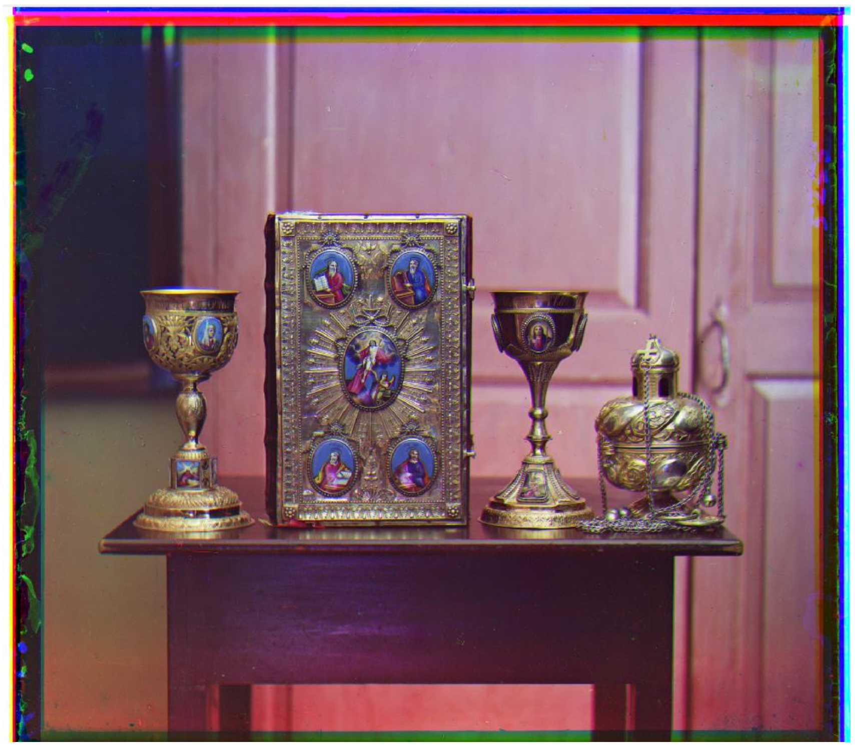
Details:
Since the brute-force search method can be time-consuming when the image is extremely large, we need a faster search procedure – image pyramid. My program calls the *impyramid* function in MATLAB for 4 times and thus reduces 15/16 of the height and width of the image. Either SSD or NCC works well in the small images and the whole procedure takes about **several seconds** for each large image, which is an impressive improvement over the brute-force method (which takes **dozens of seconds**). I also tried some other different rounds for reducing the resolution of the images but they either cost too much time or have unclear output images. Therefore, I think 4 times of reducing is a good choice for our cases.

Output: (filenames are like '11_pyramid.tif')

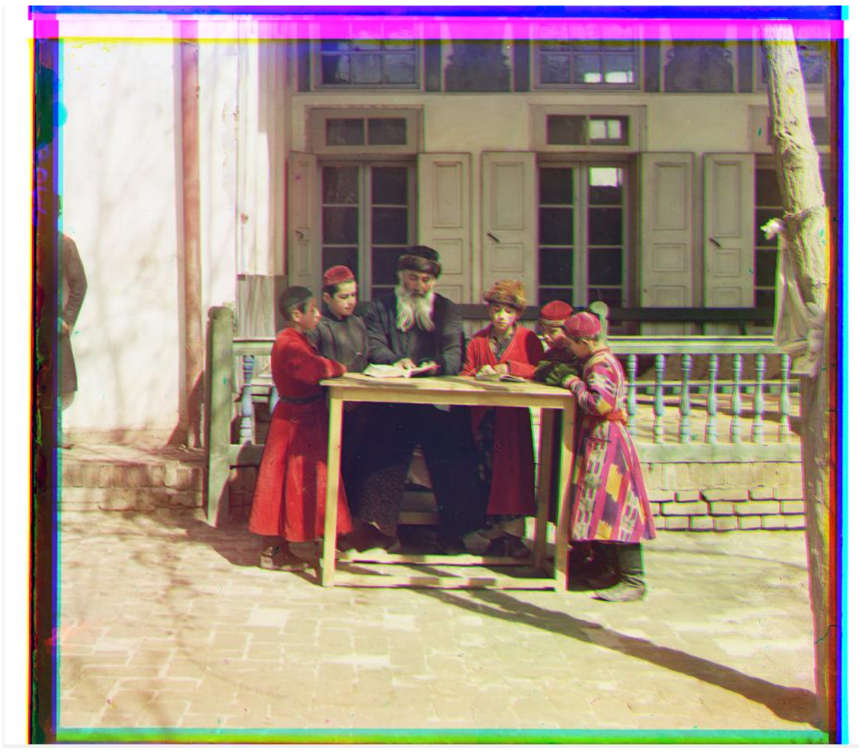
Green: [32,16]; Red: [80,32]



Green: [48.0]; Red: [112,0]



Green: [64,32]; Red: [144,64]

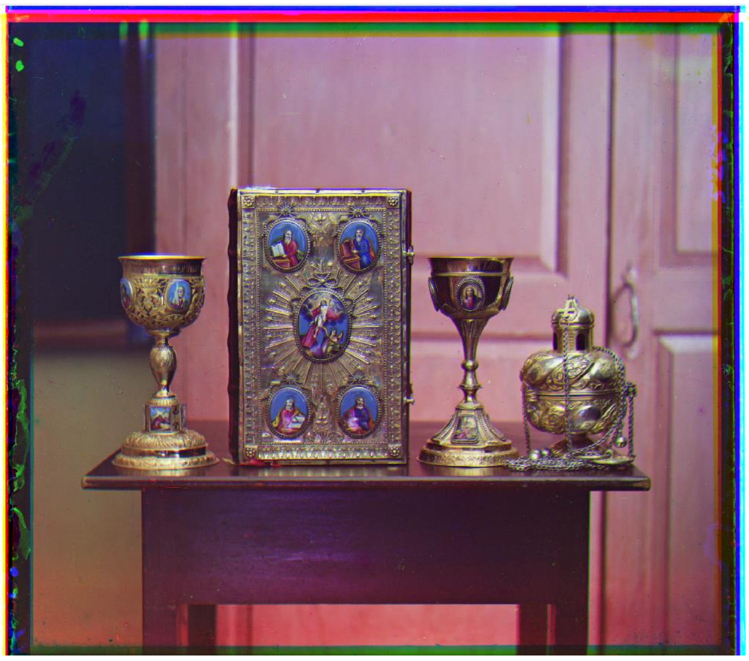
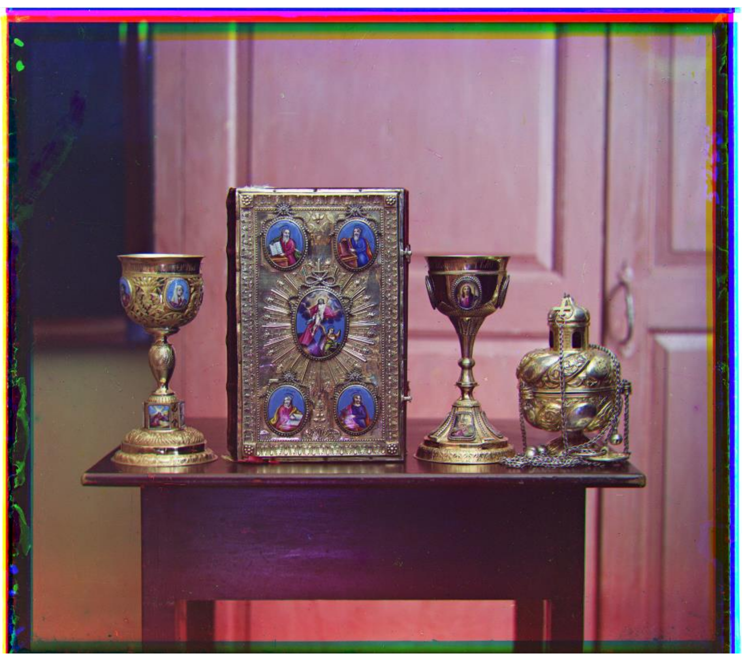


2.2 Crop channels before align – speed up too!
Since the R, G, B channels were not taking at the same time, either people or camera could move slightly. Therefore, the images cannot overlap perfectly on the borders. However, in the method of SSD and NCC, we take every pixel into the computation to **find the best offset**, which affects the correctness of the offset. To solve this issue, I crop two channels before apply SSD or NCC. More specifically, I set a lower bound and an upper to crop the image to a slightly smaller size, which lays in the center of the origin one. Based on the cropped image, we can then use any search procedure we want. After we get the offset value, we expand the offset so that they can be applied to the original channels. For example, if the image is cropped by 10% of the width both on left and right side. The x offset will be added 10% of the width. The output images demonstrate they this solution greatly reduce the blurring of the images.

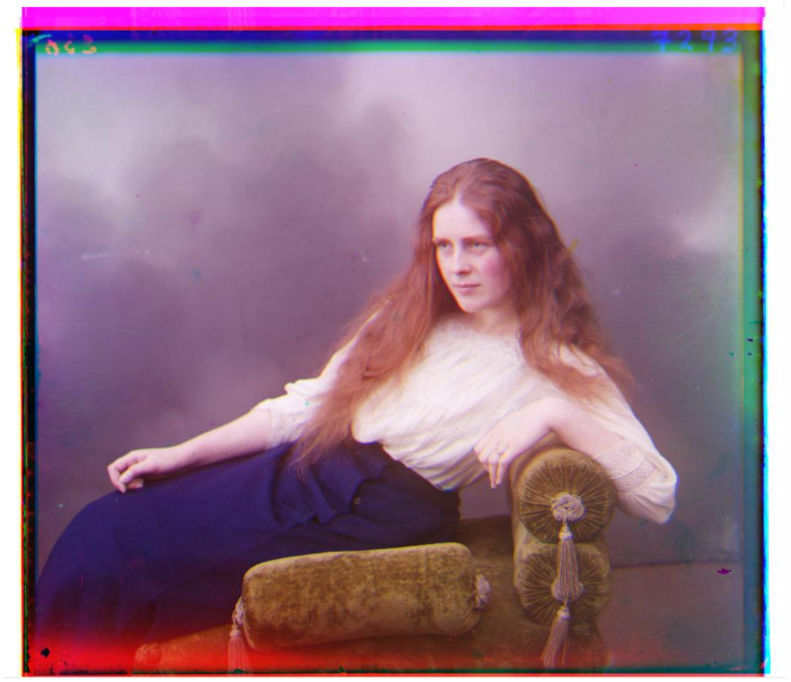
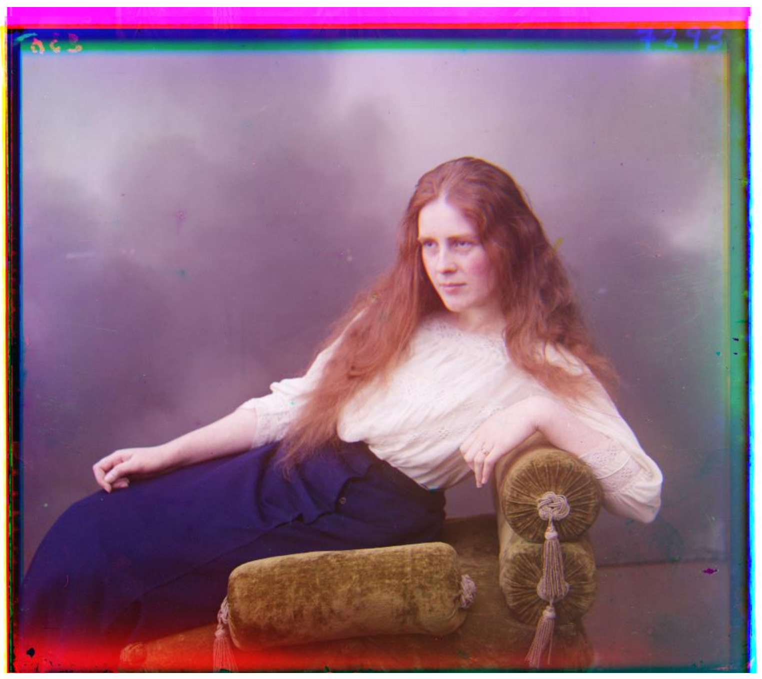
Another advantage of this solution is that it also reduces the number of pixels we need to calculate, which speeds up our program too! It only takes **less than 10s** to process a large image. (of course, without pyramid)

Output: (The pictures on the right are the ones with cropping channels)
PS: you may not tell the slight difference between two images here in PDF, but you can look up the nearly PERFECT pictures in the zip package and find out the impressive improvement. (filenames are like ‘11_align_ncc_crop_channel.tif’)

Green: [24,19]; Red: [71,33]

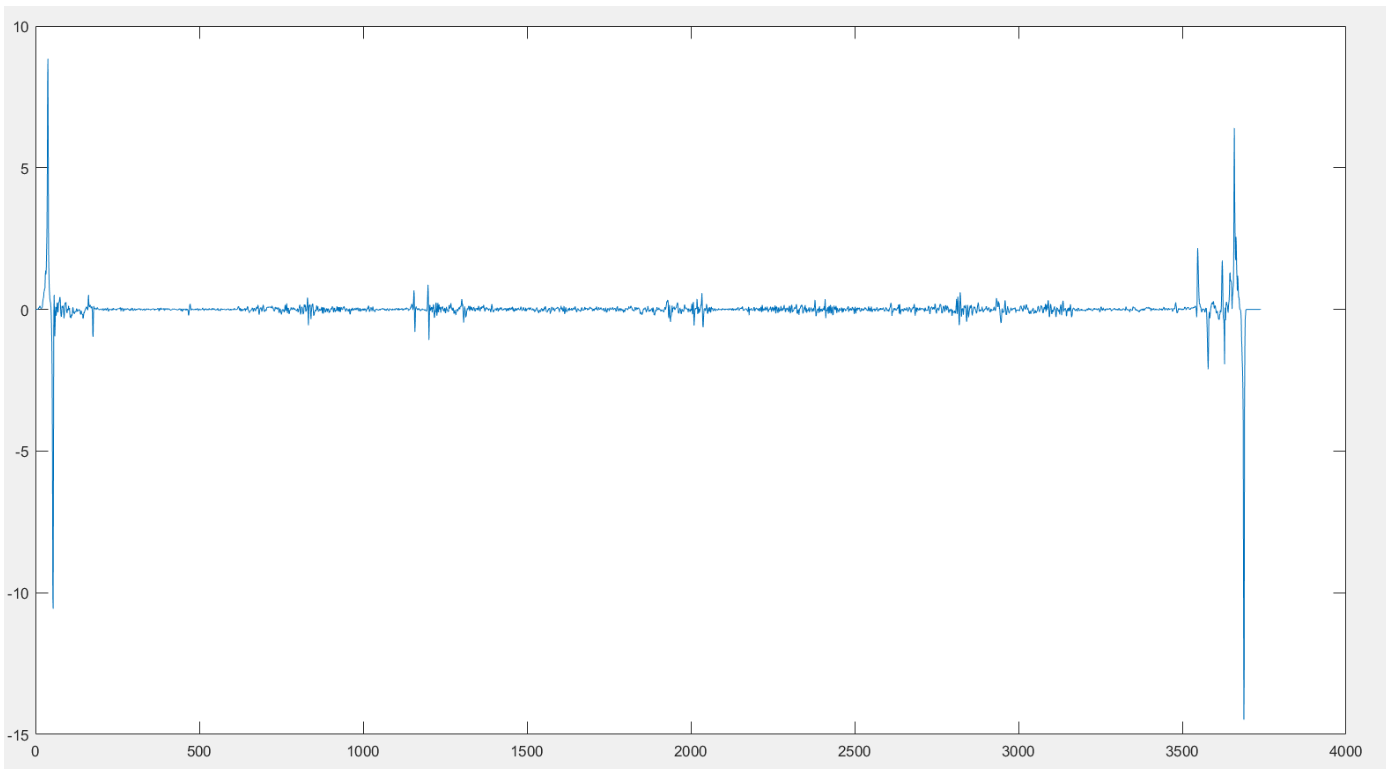
Green: [54,4]; Red: [117,6]

Green: [70,37]; Red: [146,62]

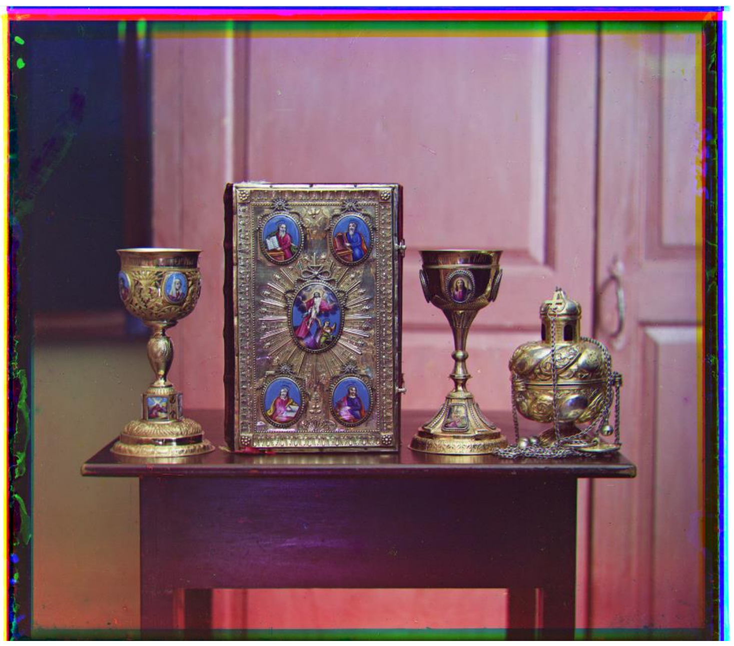
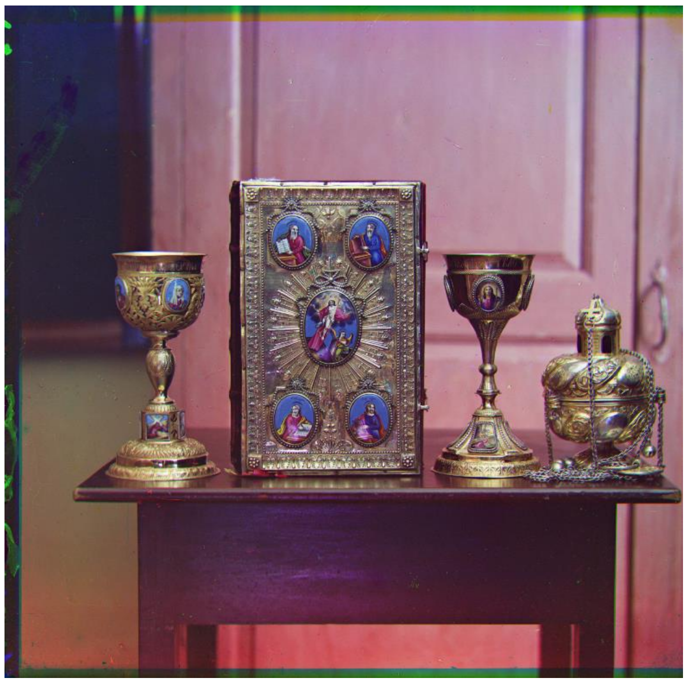
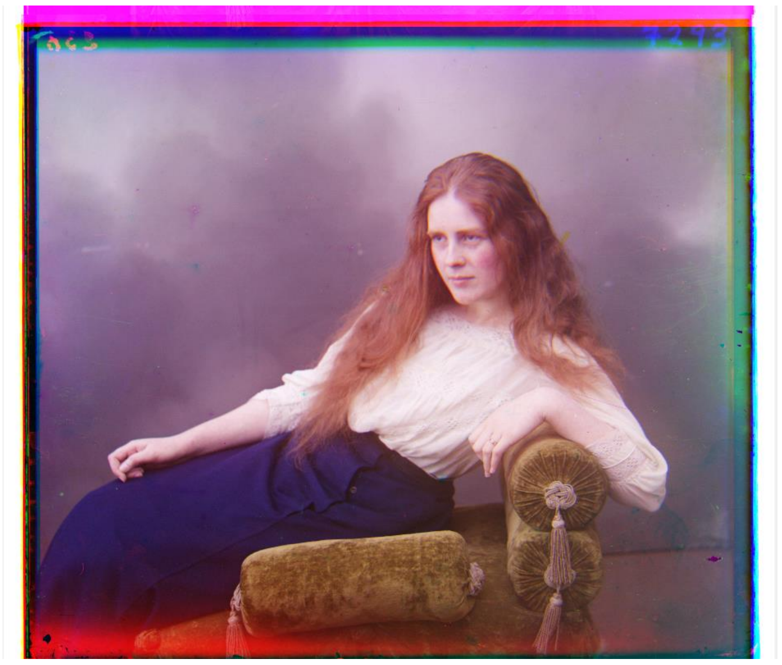
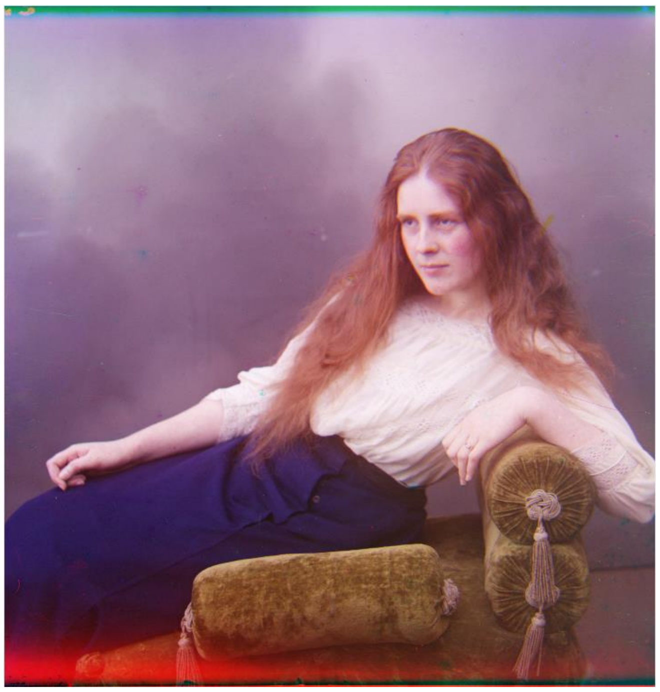
2.3 Crop image

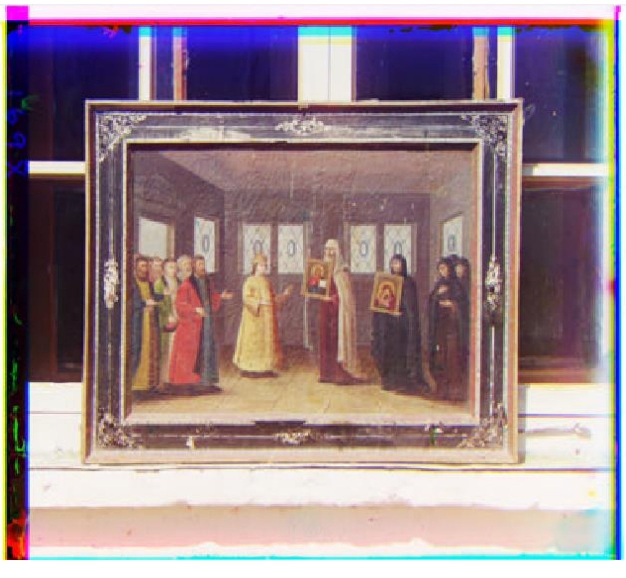
Detail:
Due to unperfect overlapping on the borders of the images, there are always some annoying white, black or other color borders. I implemented a cropping function to crop these borders to get rid of the bad stuff. The point is that the information in the good parts of the images generally agrees across three color channels, whereas the borders it does not, I took two channels to find out what should be new border of the images. What I do the in the *get_border* function is that, every time I took one row from each of the two channels, can compute their distance with the *pdist* function in MATLAB. After the iteration through every row of the channels, I got a distance vector. After calculate the difference between two adjacent distances, we got a diff vector. For example, here is a diff vector from 11.tif.

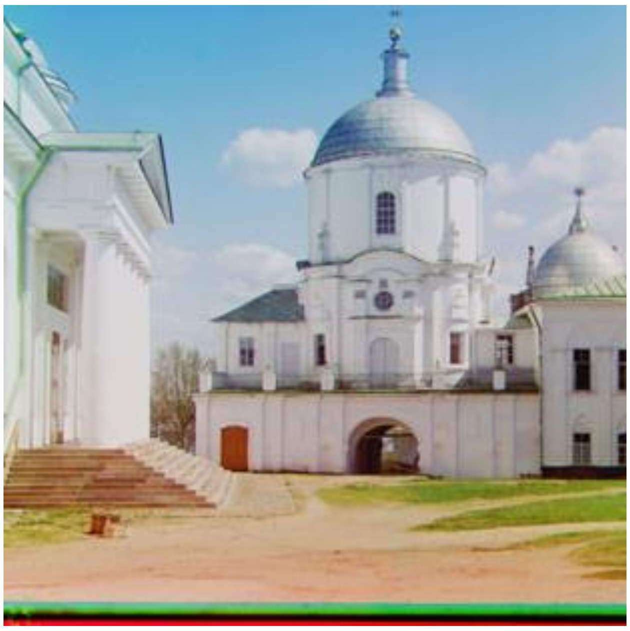


You can see that the diff values at the left and right side are much larger than the ones in the middle, so we can set the threshold to 3, for example, and find the middle-most indices, whose values are greater than the threshold. And cut the image at those indices.

Output: (filenames are like ‘11_crop.tif’)





2.4 Adjusting Contrast

Details:

Since the borders of the images are usually black and white, the images may not be well-contrast-scaled. Therefore, I took the pixels with intensity in the range from 10 to 90 percent and expanded their intensities to 0 to 100 percent to enhance their contrast. The output images are **more colorful and attractive**.

Output: Here are some of the results. (You can see the rest in the zip package)