CS543/ECE549 Assignment 1

Name: <Garen Hu>
NetId: <zihanhu2>

Part 1: Implementation Description

Provide a brief description of your implemented solution, focusing especially on the more "non-trivial" or interesting parts of the solution.

1. Image pre-processing:
   The images are not perfect with extra black or white borders. Compared to leaving those borders as a cue for alignment, removing them can have a better alignment result. Therefore, I slice my original image (`img = img[5: -5, 10: -10]`) to get an overall good-quality image through some experiments.

   In addition, for my base image, I also choose a smaller window of the channel 1 image(different images can have different windows). The reason for doing that is to further remove the extra borders since different images have different distributions of borders. Also, I adjust the starting point of my second the third search window to fit the different images. This process can ensure alignment at less cost.

2. Algorithm:
   I use a smaller window for my first image(Channel B)(`im1 = np.array(img[15: div, 5:-10])`) to allow my slicing window approach can slide in some range, which makes indexes in the bound.

   For my double for loop, I will iterate i and j value from 0-14. For my starting row index for the second and third images, I will choose the [height of whole image / 3 + x + i] and [2 * height of whole image / 3 + y + j]. I will modify the value of x and y to find the best match to ensure the best position is in the range.

3. Artifacts:
   Although I could delete the most of borders before the alignment, the aligned image will still have artifacts on the edges due to the rest of the small part of the borders. This artifact will show a weird color compared to the content of the image.

   Second, some images may not be perfectly aligned. It may be off by some pixel values, which results in the blur effect. This is also due to the limitation of my method since 3 individual image channels are not uniformly distributed across all input images. However, my approach can efficiently avoid this kind of artifact.

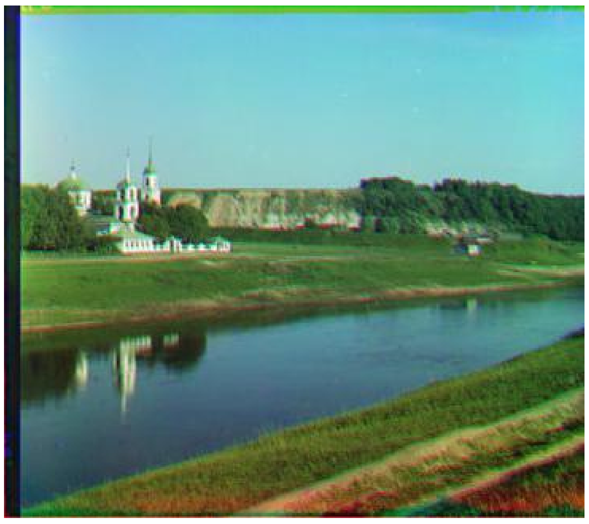
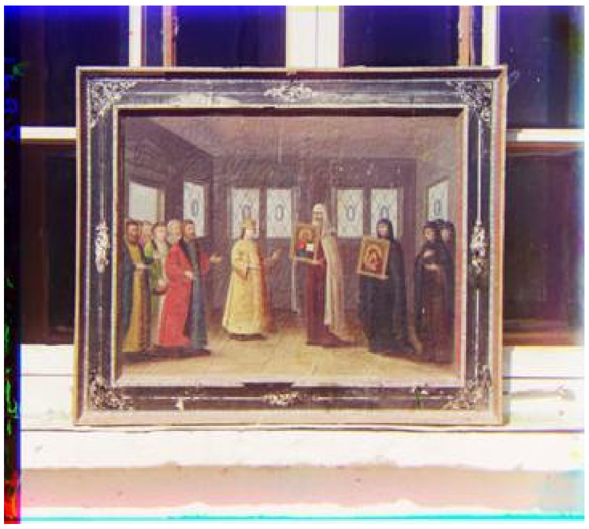
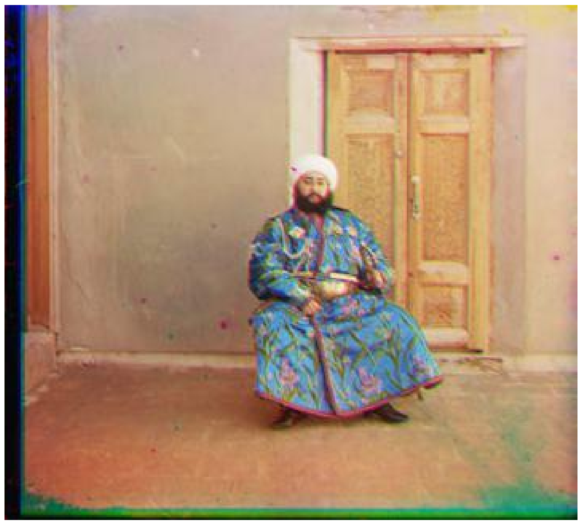
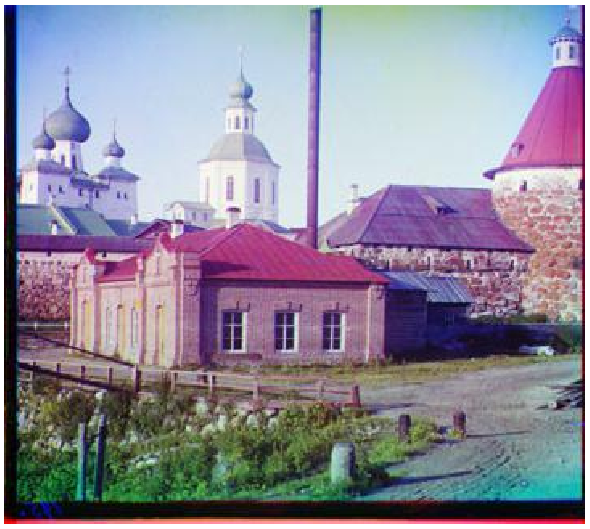
Part 2: Basic Alignment Outputs

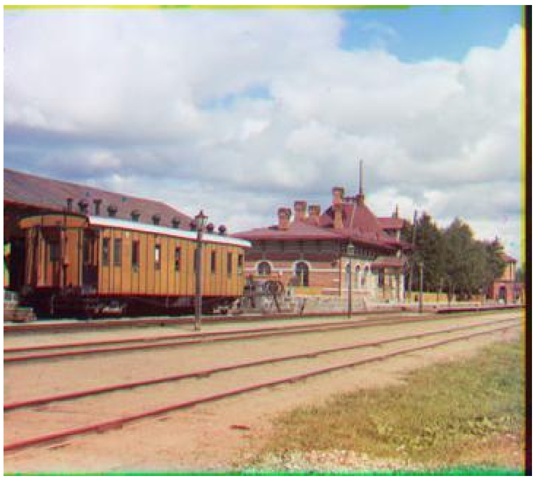
A: Channel Offsets

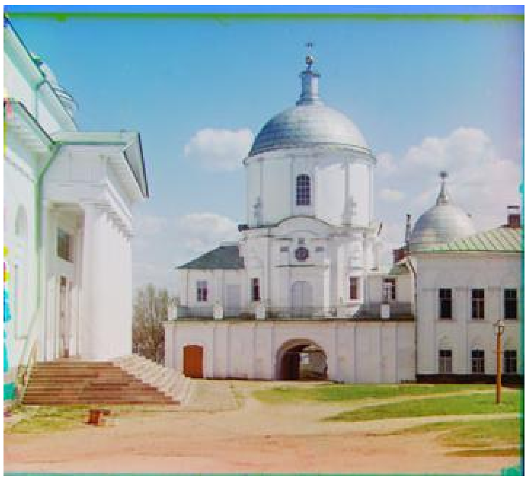
Using channel B as a base channel:**(offset is the index of top left corner pixel)**

Image	G (h,w) offset	R(h,w) offset
00125v.jpg	(356, 14)	(691, 14)
00149v.jpg	(357,13)	(693,13)
00153v.jpg	(355,13)	(688,10)
00351v.jpg	(357,15)	(689,14)
00398v.jpg	(356,13)	(690,15)
01112v.jpg	(346,15)	(682,14)

B: Output Images





Part 3: Multiscale Alignment Outputs

I used a similar approach as basic alignment, in which I create a smaller window of my base image(Channel B) to remove the boundaries. My window for 01047u.tif and 01657u.tif is `im1 = img[105 : height - 207, 100 : -201]` which img is my original image, but my offset is relative to the original big image. To optimize the tradeoff between the running time and precision of alignment. I choose 3 levels of the pyramid. In this case, my images still have solid information to find the alignment offset without losing too much speed.

For the coarsest scale image(smallest image), I choose a big range(15) of searches to find the best matching point. For the lower level, I create a 5 by 5 window to reduce the cost.

To get a better-aligned image, I need to adjust my window for my base image and the starting index of my second and third channels to improve my running time. I modify my window to `im1 = img[150 : height - 57, 100 : -101]` to remove correct borders for 01861a.tif.

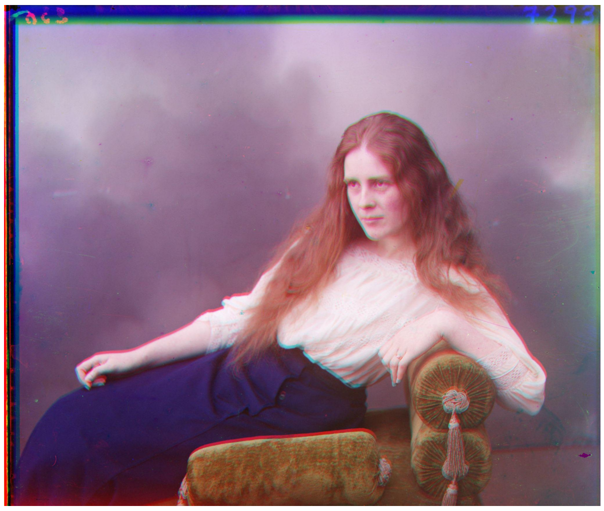
A: Channel Offsets

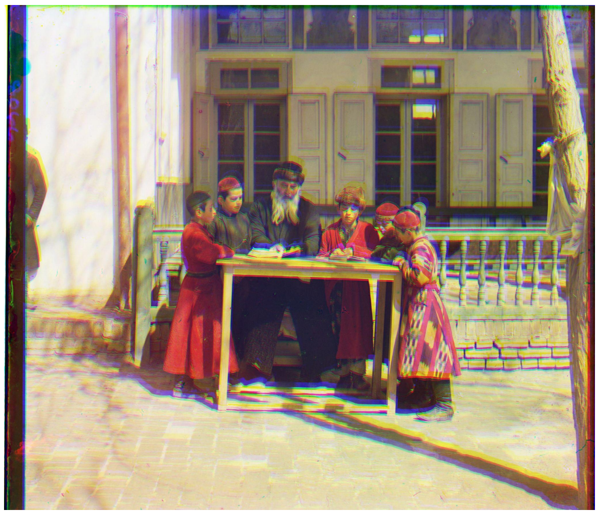
Using channel B as the base channel:**(offset is the index of the top left corner pixel)**

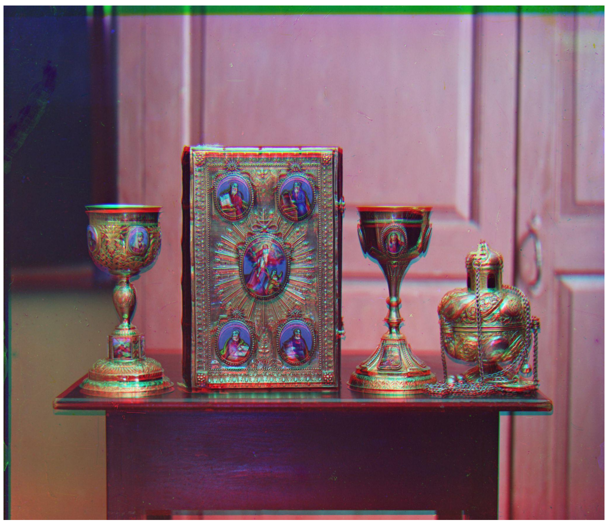
Image	G (h,w) offset	R (h,w) offset
01047u.tif	(3295, 80)	(6440, 64)
01657u.tif	(3262, 92)	(6430, 82)
01861a.tif	(3277, 76)	(6418, 54)

B: Output Images

Insert the aligned colorized outputs for each image below (in compressed jpeg format):







C: Multiscale Running Time improvement

The running time of my approach is about 10 seconds including the cost of image processing. This approach is faster than the naive method I used in part 1. I also use recursion to reduce the amount of code.

Part 4: Bonus Improvements

Description:

First, I made some experiments on the pixel values of the artifacts. They are all very low pixel values since the border color is dar. Therefore, when the sum of the pixel value is less than a certain value, I will remove this row or column. Second, if the difference between two neighboring pixels is large which is agreed to 3 channels, I will remove that column or row as well. Finally, I start looping from the center of the row and the center of the column. It is best to do it for four edges. To speed up, I can do two edges and crop the same amount on the opposite side. The results are shown below, which are pretty good.