CS543/ECE549 Assignment 1

Name: Andrew Ko
NetId: hyunjun5

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.

- What implementation choices did you make, and how did they affect the quality of the result and the speed of computation?
- What are some artifacts and/or limitations of your implementation, and what are possible reasons for them?

For image to be processed, my approach was to divide the vertical images into 3 each of which would correspond to B, G, R channels of the original colored image. Interestingly, some of the images are better aligned when cropped while some of them are better with the borders. Many trials and errors of how many pixels I should cut horizontally and vertically allowed me to find the best aligned images within [-15,15] displacement window. Run time was faster for those that had no borders. The core implementation of the code is in the displace(). In this function is where I take in the base image and the one of the other two channel images to find the best alignment of the two. It has a for loop that iterates through every possible combination of [-15,15] displacement and scores each using one of the two metrics: sum of squared difference or normalized cross correlation. When displacing the channel image, instead of getting rid of the range of pixels that are less than displacement values, I used numpy's roll function to move them to the other side of the dimensions so there won't be a dimension mismatch. For ssd, the minimum was the best displacement and for ncc, the maximum would be the best displacement, but I used the maximum of the sum of correlation to include all pixels into account. Generally, the ssd took less computing time to calculate. For multiscale images, instead of using recursion within my displace() to better estimate the displacement as I grow the size of images, I used a for loop from a quarter size of the original size. The loop runs 3 times from a quarter size to original size, each time finding the best displacement for that size level and doubling it and the image in the next iteration to find the best displacement for that pyramid level until it finds the best estimate for the original image size and align the images. If I had used recursion and provided coarser image than a quarter size, the displacement estimate would have been more accurate but at the cost of speed. Also, for multiscale solution, as it traverses the image pyramid, the estimate of displacement is also doubled, which means that it could possibly have missed the best displacement if it was in between the original estimate and the scaled estimate for the next level in the pyramid.

Some of the artifacts that I see in my aligned images are bands of the color corresponding to a channel near the borders of the image. I believe this is likely due to the imperfect crop of the borders if the image is cropped at all. Another cause may be due to the limited search space for the best displacement of pixels in the other two channels. For speed of computation, the search space was limited to [-15,15] which should be in general wide enough to find a good displacement but may not have been the best since there are some artifacts.

Part 2: Basic Alignment Outputs

For each of the 6 images, include channel offsets and output images. Replace <C1>, <C2>, <C3> appropriately with B, G, R depending on which you use as the base channel.

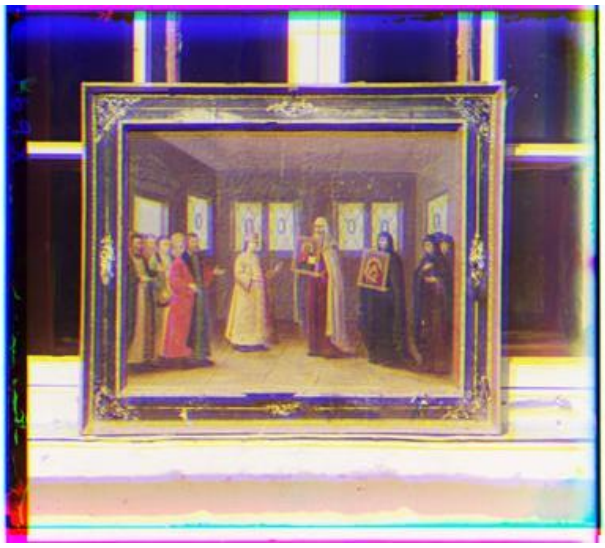
A: Channel Offsets

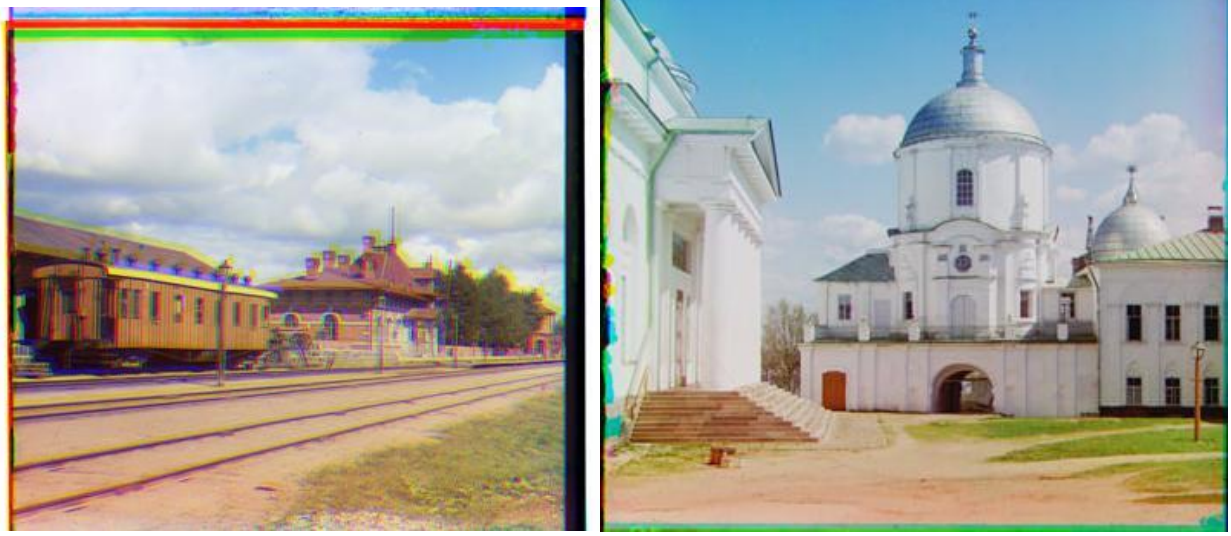
Using channel <C1> as base channel:

Image	<C> (h,w) offset	<C3> (h,w) offset
00125v.jpg	<G> (-1,-1)	<R> (1,-1)
00149v.jpg	<B> (-5, 3)	<G> (-5,-1)
00153v.jpg	<B> (-10 -2)	<G> (-10. -3)
00351v.jpg	<G> (4, 0)	<R> (13,1)
00398v.jpg	<G> (9, -1)	<R> (15, -2)
01112v.jpg	<B> (0, 0)	<R> (5, 1)

B: Output Images

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





Part 3: Multiscale Alignment Outputs

For each of the 3 high resolution images, include channel offsets and output images. Replace <C1>, <C2>, <C3> appropriately with B, G, R depending on which you use as the base channel. You will also need to provide an estimate of running time improvement using this solution.

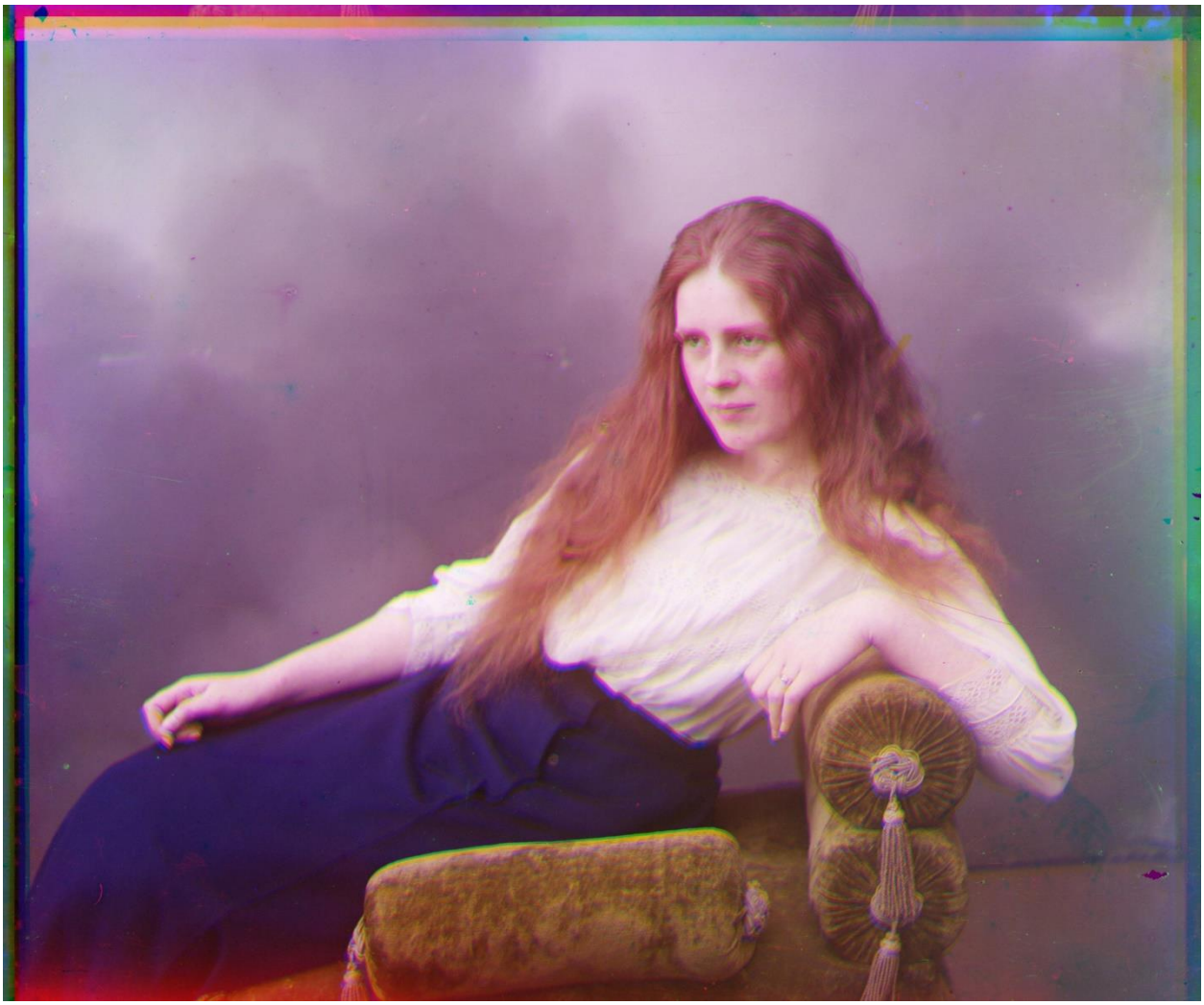
A: Channel Offsets

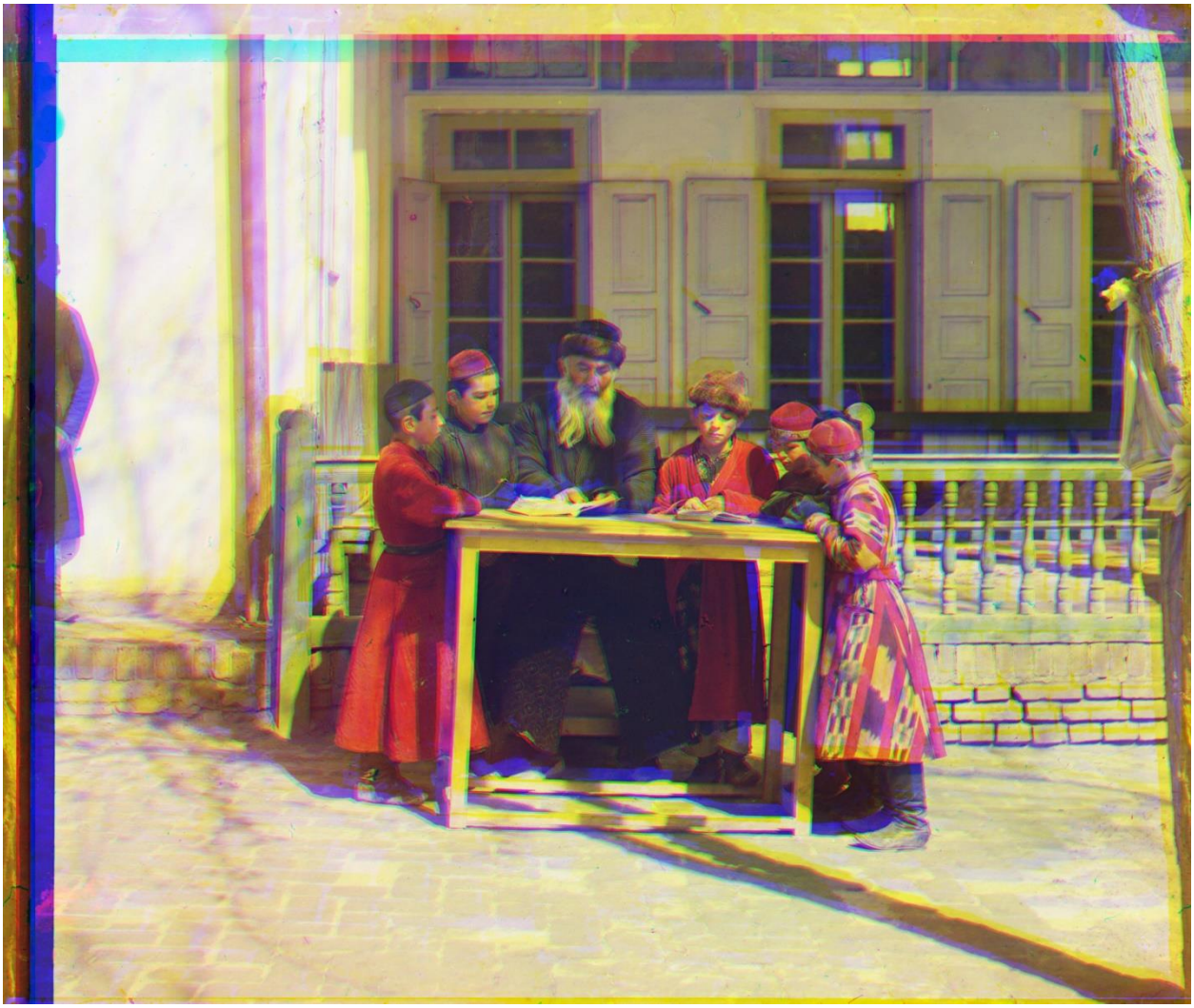
Using channel <C1> as base channel:

Image	<C2> (h,w) offset	<C3> (h,w) offset
01047u.tif 17.2	<B> (26,18)	<R> (21,6)
01657u.tif 15.47	<G> (35, 35)	<R> (35, 35)
01861a.tif 14.98	<G> (44,49)	<R> (36,49)

B: Output Images

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





C: Multiscale Running Time improvement

Report improvement for the multiscale solution in terms of running time (feel free to use an estimate if the single-scale solution takes too long to run). For timing, you can use the python `time` module, as described in the assignment instructions.

These times were measured when the algorithm used [-15,15] as the search space in the possible window of displacement for the best alignment. Some images yielded better results when I reduced the search space or increased the search space. For consistency and to compare results fairly the time reported is for search range of [-15,15] displacement.

For 01047u.tif, it took about 17.195s to run.
For 01657u.tif, it took about 15.473s to run.
For 01861a.tif, it took about 14.982s to run.

Without using the Image pyramid,

For 01047u.tif, it took about 60.196s to run.
For 01657u.tif, it took about 58.978s to run.
For 01861a.tif, it took about 60.393s to run.

Part 4 : Bonus Improvements

Post any extra credit details with outputs here.