CS543/ECE549 Assignment0

Chaohua Shang (chaohua2)

Implementation:

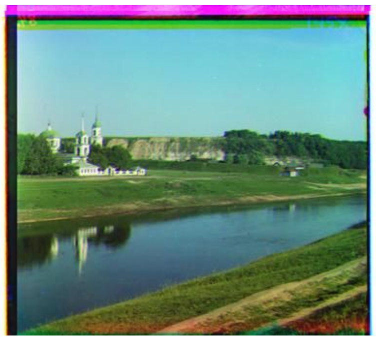
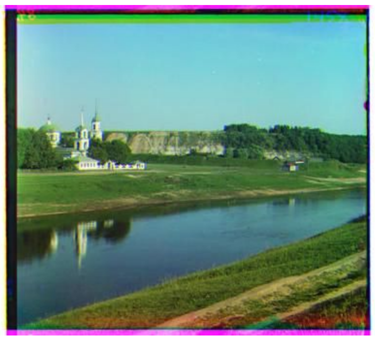
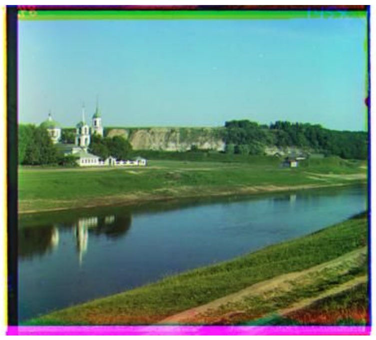
As the instruction described, first, separate the image into three channels and then apply some metrics to evaluate the result to find the best displacement vectors that were used to align the channels. And stack three channels to display the colorized image.

SSD

This implementation exhaustively search over a window of possible displacements [-15,15] pixels to find the best displacement vectors. For scoring the result, use the sum of squared differences between two images *sum(sum((image1-image2).^2))*. The best alignment will give the smallest sum.

But the images are pretty blur after I stacked three channels together. Since the borders of the photograph have little information and sometimes can disturb the result, so I **cropped the borders** of three channels(crop 10 pixels on each side) and implement the same method to align the image. The image is much clearer. Also, I tried **different orders of aligning the channels** to compare the result. Below is the result I got. I **highlighted the best displacement vectors** for each image from my point of view. Actually, different orders of aligning the channels may give the same displacement result, for example, picture two and picture four.

We can see that for most of image, aligning all the other two channels to Blue would give an reasonable output. But for *00153v.jpg* the third row, aligning channel B and R to G gives the best result. We can see that three channels of this image are much more different than each other compared with other images.

Align to channel B	Align to channel G	Align to channel R
		

Offset: G[6,0] R[10,0] **Runtime: 1.1094**	Offset: B[-6,0] R[4,-1] Runtime: 1.1250	Offset: B[-10,0] G[-4,1] Runtime: 1.1250
Offset: G[4,2] R[9,2] **Runtime: 1.2656**	**Offset: B[-4,-2] R[5,0]** **Runtime: 1.0313**	**Offset: B[-9,2] G[-5,0]** **Runtime: 1.1563**
Offset: G[7,1] R[11,9] Runtime: 1.2188	**Offset: B[-7,-1] R[7,1]** **Runtime: 1.1719**	Offset: B[-11,9] G[-7,-1] Runtime: 1.0781
Offset: G[4,0] R[13,0] **Runtime: 1.0156**	**Offset: B[-4,0] R[9,0]** **Runtime: 1.0469**	Offset: B[-13,0] G[-9,0] Runtime: 1.0938

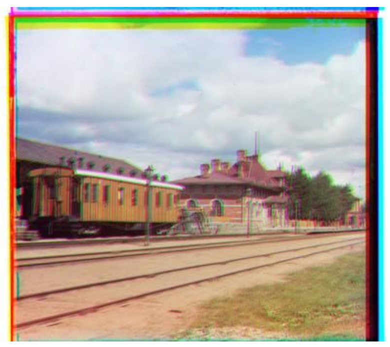
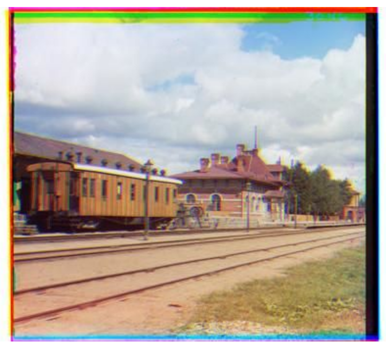
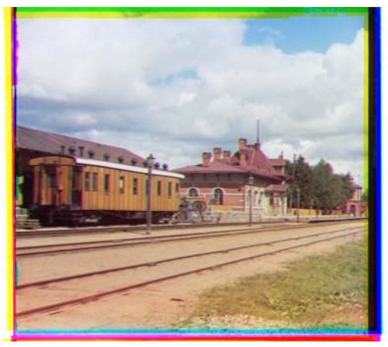
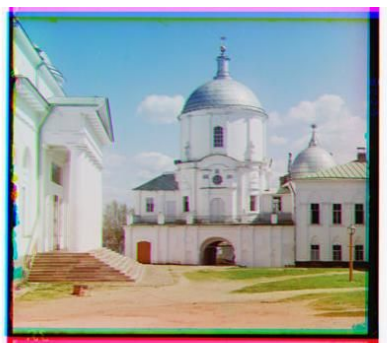
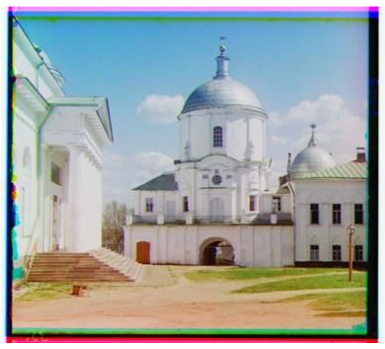
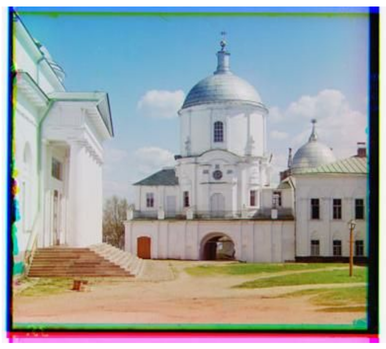
		
Offset: G[6,0] R[11,4] Runtime: 1.1406	Offset: B[-6,0] R[6,0] Runtime: 1.2188	**Offset: B[-11,4] G[-6,0]** **Runtime: 1.2344**
		
Offset: G[0,1] R[5,1] **Runtime: 1.1563**	Offset: B[0,-1] R[5,1] Runtime: 1.2188	Offset: B[-5,-1] G[-5,-1] Runtime: 1.0781

Table1 SSD alignment

NCC

I also implemented NCC which is simply the dot product between the two images normalized to have zero mean and unit norm to get the best alignment. To implement this,
Same as SSD, some of the images I got when I operated on the full image are blur or misaligned. So I **cropped the borders of image** (crop 5% on each side) and do NCC on these cropped image and then get (x,y) displacement vectors. And I found some images are still not well aligned (see table 2) . So I increased the crop area (crop about 25% on each side) for those images, the results are much better, but still some image are not aligned. So I tried to **crop only one image** (crop about 25% on each side) and keep the original size for reference image[1], the results are much better. This is because the border area contains some noises which can become the distribution for real image information when calculate the mean of the image in NCC. And when using `normxcorr2` in matlab, according to website [1], it is important that one is smaller than other. The final results are shown in Table3, runtime that is less than 0.5 is implemented with the second adjustment. Same as SSD part, I highlighted the best displacement vectors for each image.

We can see that NCC is much faster than SSD. This is because in SSD, we exhaustively search over the window, and he time complexity for SSD is O(n^2). In our case, n is 31 (-15 to 15). So, SSD can not deal with big or high resolution images.

Limitation: In order to get clearer image, I have to crop the border of image. The border size that we choose to crop will influence the output image. For example, when I crop 5% border of image, output shown in table 2 is misaligned. But when I crop only one image by 25% and keep the reference image unchanged, the output image is well aligned.

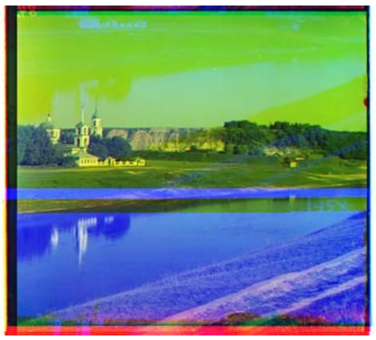
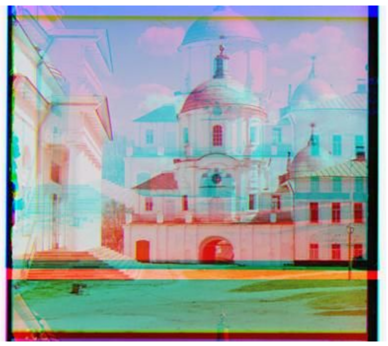
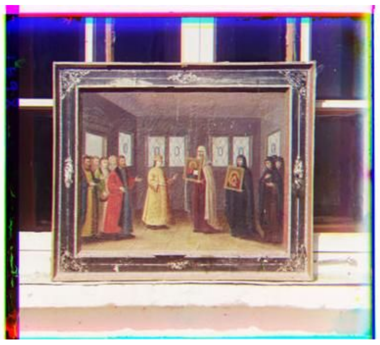
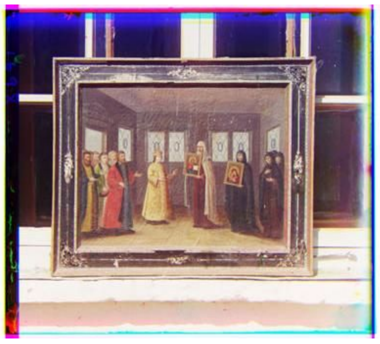
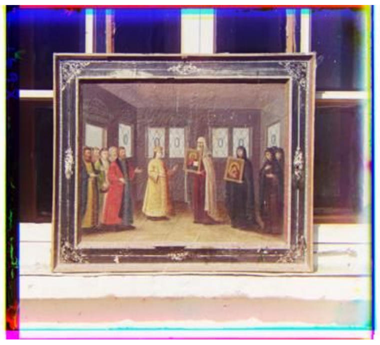
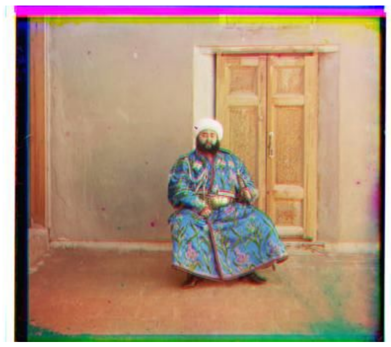
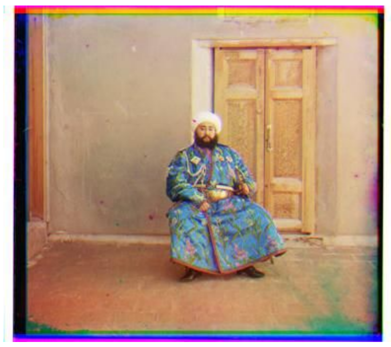
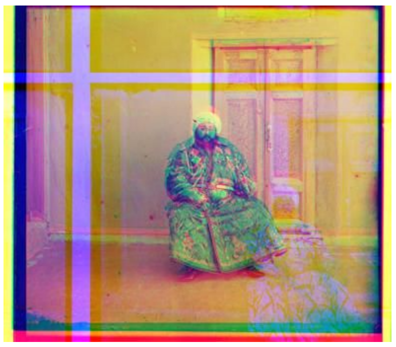
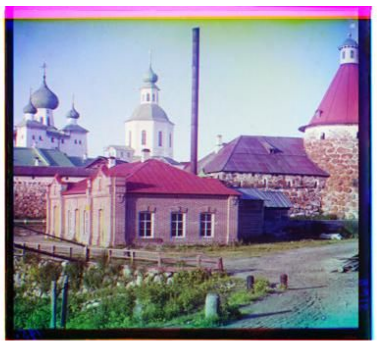
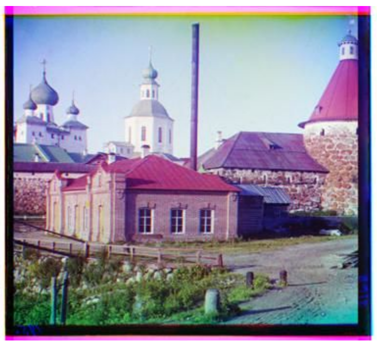
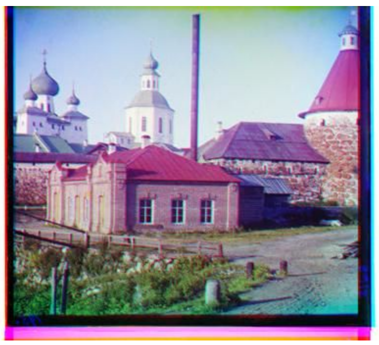
	
Offset: B[189,0] G[-4,0] Runtime: 0.6250	Offset: G[0,0] R[-63,3] Runtime: 0.6719

Table2. Some images are misaligned when crop 5% of border

Align to channel B	Align to channel G	Align to channel R
Offset: G[6,0] R[10,1] Runtime: 0.6563	**Offset: B[-6,1] R[4,0]** **Runtime: 0.6563**	Offset: B[-11,-2] G[-5,0] Runtime: 0.3594

		
Offset: G[4,2] R[9,2] Runtime: 0.6875	**Offset: B[-4,-2] R[5,0] Runtime: 0.6563**	Offset: B[-9,2] G[-5,0] Runtime: 0.6250
		
Offset: G[7,2] R[12,4] Runtime: 0.7031	**Offset: B[-8,-1] R[8,2] Runtime: 0.5781**	Offset: B[68,76] G[0,0] Runtime: 0.7188
		
Offset: G[4,0] R[13,0] Runtime: 0.8750	**Offset: B[-4,0] R[9,0] Runtime: 0.7969**	**Offset: B[-13,0] G[-9,0] Runtime: 0.4219**

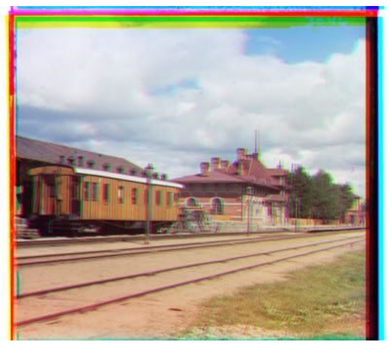
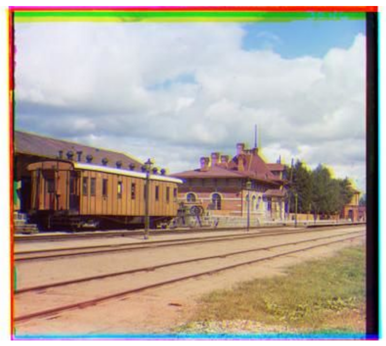
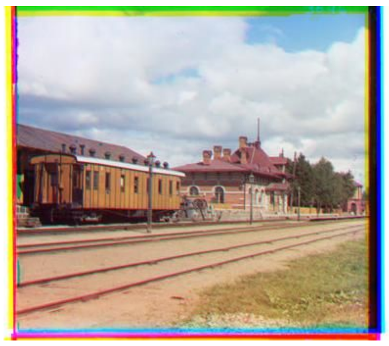
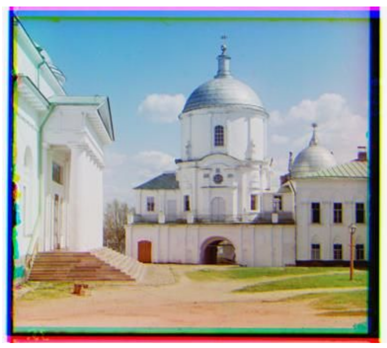
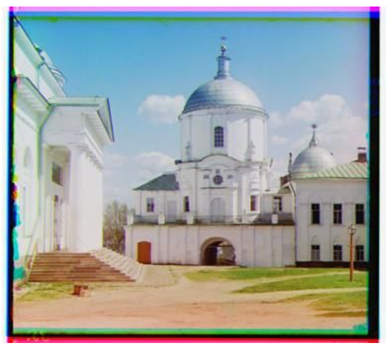
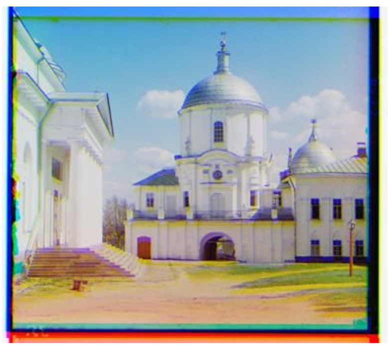
		
Offset: G[5,0] R[11,4] Runtime: 0.7969	**Offset: B[-6,1] R[6,1] Runtime: 0.6250**	Offset: B[-12,-5] G[-7,-2] Runtime: 0.3438
		
Offset: G[-1,-1] R[4,0] Runtime: 0.1875	**Offset: B[0,-1] R[5,1] Runtime: 0.7344**	Offset: B[0,0] G[-5,-1] Runtime: 0.0938s

Table3. SSD alignment

Bonus points
Multiscale alignment
Since the images given are high resolution and contain more pixels, apparently ssd or ncc will take a long time. Below is the one results that I get applying ncc directly on the full size high resolution images.It took about 230s to get the result, and other image took about the same time to output the final image. (I cropped about 15% border on each side using method 2).So I implemented the image pyramid as described in instruction.

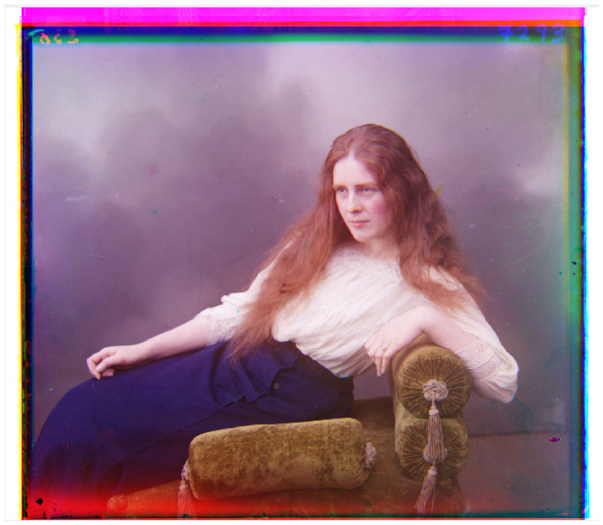
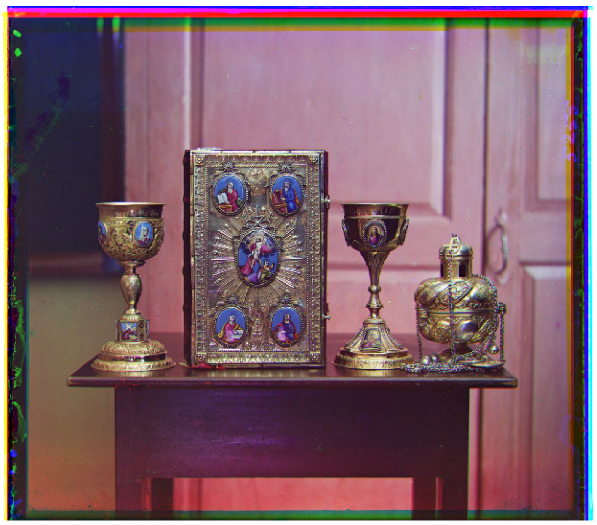
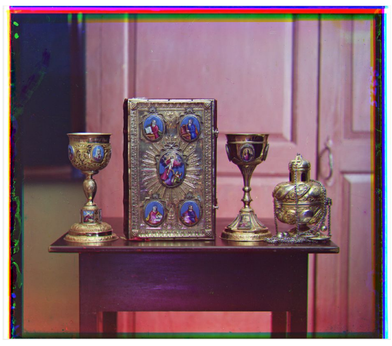
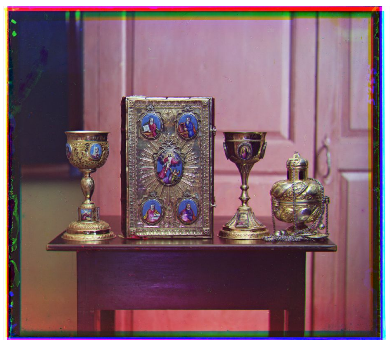
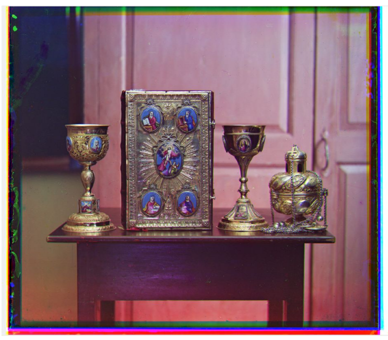
	
Offset: G[53,7] R[115,9] Runtime: 232.0625	Offset: G[23,18] R[70,32] Runtime: 204.3281

Table4. Apply Ncc directly on full size image

First, separate the image into three channels and scale down the image by factor 2. If the image height is larger than 500, then repeat scaling down. Then, run the alignment function on small image using NCC(crop 10% border on each side) and return (x,y) displacement vectors. Multiply 2 ^ (times of scaling down) to this displacement vector to get the (x,y) offset for original images. Then circshift two channels and stack them to the third channel to get the colorized image. Below is the output when I resize the image in 3 levels(scale down 3 times). We can see that the runtime is reduced significantly. Also, I found that when I add one more level to scale down, the output is obviously misaligned. This is because the small image lose too much informations when we scale down, so the displacement vectors that obtained on small image can not be applied to big image.

Align to channel B	Align to channel G	Align to channel R
		

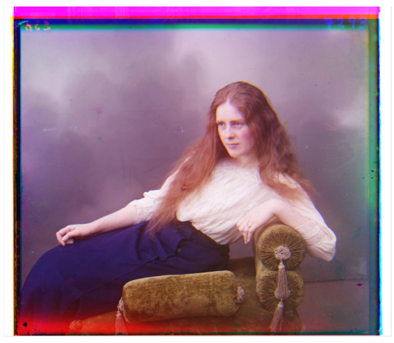
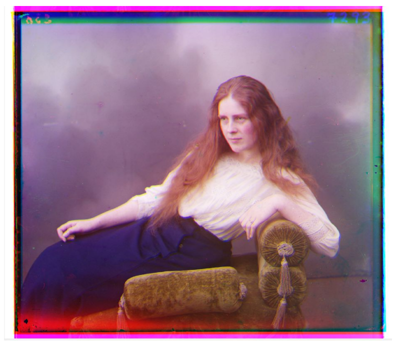
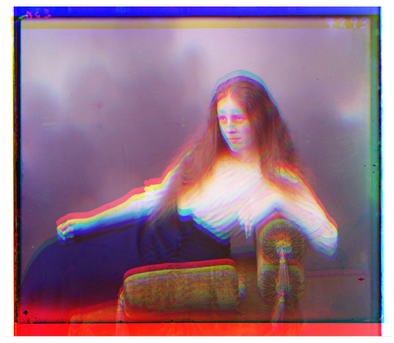
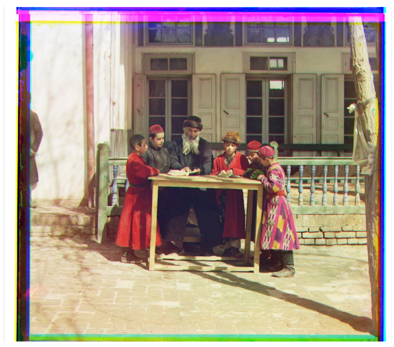
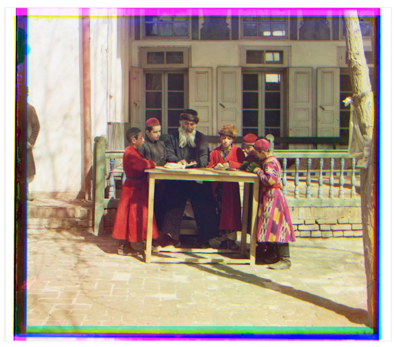
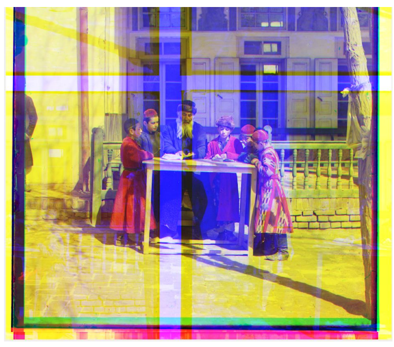
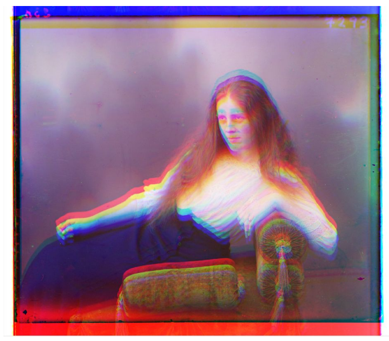
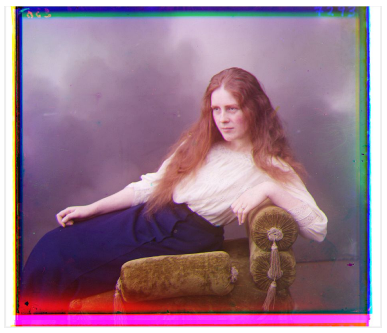
Offset: G[16,24] R[32,72] **Runtime: 2.1563**	**Offset: B[-16,-24] R[16,48]** **Runtime: 2.0625**	**Offset: B[-32,-72] G[-16,-48]** **Runtime: 2.1250**
		
Offset: G[0,56] R[0,120] **Runtime: 2.1563**	Offset: B[0,-40] R[0,64] Runtime: 2.1563	Offset: B[0,0] G[0,0] Runtime: 1.7344
		
Offset: G[40,72] R[64,144] Runtime: 2.1094	Offset: B[-32,-64] R[24,72] Runtime: 2.1563	Offset: B[1592,672] G[-24,72] Runtime: 2.2813

Table5. Image pyramid alignment

Other improvements.

edge detection alignment

Instead of dealing with the original image based on color intensity, I extracted the edge of image to obtain the change of intensity. I used edge(I,'Canny') filter in matlab to get the edge image. And then implement NCC alignment on edge image to get the displacement vectors. This is pretty good. Below is the result comparison based on same setting.

	
original Offset: B[0,0] G[0,0] Runtime: 1.7344	Edge detection Offset: B[0,-64] G[-16,-112] Runtime: 2.4375

[1]https://www.mathworks.com/help/images/examples/registering-an-image-using-normalized-cross-correlation.html
[2]https://inst.eecs.berkeley.edu/~cs194-26/fa14/upload/files/proj1/cs194-dq/desai_nishant_proj1/