**Nitin chakravarthy Gummidela**

**UIN : 427006656**

**Computational Photography – Homework -1**

Sergei Mikhailovich Prokudin-Gorskii image alignment and automatic colorization:

Approach:

The inputs were three channels placed one below the other. So the first step was to divide the input into three pieces of same height. The first piece was used as the blue channel, the second as red and the final piece as the green channel. The ends of the three channels were cropped for better results as they had a black border and wasn’t a part of the image. The cropped images were then aligned and colorful images were produced by placing one on top of another. The metric used to compare the accuracy of the alignment was Normalized cross correlation. Red and green channels were rolled with their corresponding x and y displacements and aligned with the blue channel.

Two approaches were used to produce the result. The first approach involved a brute-force search algorithm. **‘getShift()’** method in the code uses this approach. The problem with this approach is that it needs a large radius for look up and it takes a long time (approx. 57 mins for .tif images).

So to make the approach better the images were recursively downsampled by a factor of 2 and then displacements were calculated at each level and these displacements were propagated back every level. This approach reduced the time taken for each image drastically from approximately an hour to 3-4 mins per image. The end condtion in this case was taken as 400px height where a basic search was performed. **‘getShiftPyramid()’** method in the code uses this approach.

But, recursion in general consumes a lot of RAM since it must store unnecessary levels information. This makes the process slower and also consumes more memory. An alternative to this implementation is a bottom- up version of the recursive strategy. In this approach, we calculate the number of levels we need to downsample and we start at the lowest level and we loop back till we reach the size of the image. The memory used and the time taken reduces even further in this approach. **‘getShiftPyramidIterative()’** method implements this approach.

**Problems faced:**

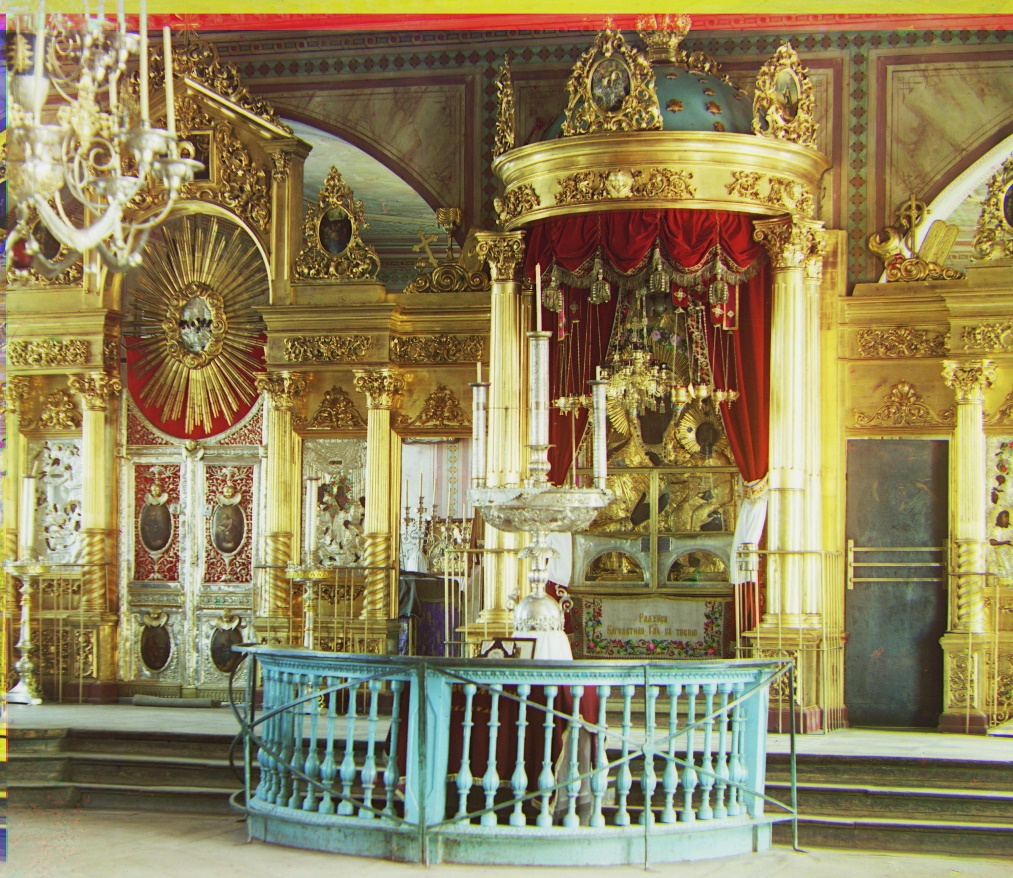
1. The initial problem I faced was while importing the images. Initially I used skImage to import images (both .jpg and .tif). But skimage imports jpgs and tiffs in a different scale. Therefore to rectify this problem. I used OpenCV to import the image and used image\_as\_flot() method to convert the values into a 0-1 scale.
2. The second problem I’ve faced was to find a function similar to to MATLAB’s circShift. The np.roll() method can be used to simulate a similar functionality.
3. Another problem was the effect of edges on the normalized cross correlation score. Since the edges were not a part of the image and they were cropped out to reduce this effect.
4. All the images except self-portrait needed a search window of 25 pixels. The remaining images were able produce good results even in the case of 15 pixel search window.

The images are as follows:



























The following are the X, Y shifts and the time taken for each image in the best approach.

**cathedral.jpg**

blue-green disp : x: 5 y: 2

blue-red disp : x: 12 y: 3

Time(sec): 11.088362693786621

**emir.tif**

blue-green disp : x: 48 y: 24

blue-red disp : x: 102 y: 55

Time(sec): 144.5384964942932

**harvesters.tif**

blue-green disp : x: 56 y: 16

blue-red disp : x: 112 y: 14

Time(sec): 250.76738667488098

**icon.tif**

blue-green disp : x: 40 y: 17

blue-red disp : x: 88 y: 23

Time(sec): 179.0032992362976

**lady.tif**

blue-green disp : x: 48 y: 9

blue-red disp : x: 112 y: 11

Time(sec): 260.7537806034088

**monastery.jpg**

blue-green disp : x: -3 y: 2

blue-red disp : x: 3 y: 2

Time(sec): 11.240784645080566

**nativity.jpg**

blue-green disp : x: 3 y: 1

blue-red disp : x: 7 y: 0

Time(sec): 11.402505874633789

**self\_portrait.tif**

blue-green disp : x: 78 y: 29

blue-red disp : x: 176 y: 37

Time(sec): 234.59457087516785

**settlers.jpg**

blue-green disp : x: 7 y: 0

blue-red disp : x: 14 y: -1

Time(sec): 11.089340209960938

**three\_generations.tif**

blue-green disp : x: 52 y: 13

blue-red disp : x: 112 y: 11

Time(sec): 256.393372297287

**train.tif**

blue-green disp : x: 40 y: 5

blue-red disp : x: 87 y: 32

Time(sec): 178.38299417495728

**turkmen.tif**

blue-green disp : x: 56 y: 21

blue-red disp : x: 112 y: 27

Time(sec): 223.81914901733398

**village.tif**

blue-green disp : x: 64 y: 12

blue-red disp : x: 112 y: 21

Time(sec): 273.53444600105286

We can see that for the small .jpg images it takes approximately 11 seconds to give a good result, while for the larger tiff images it takes approximately 250 seconds which is 4mins.