Lab 3

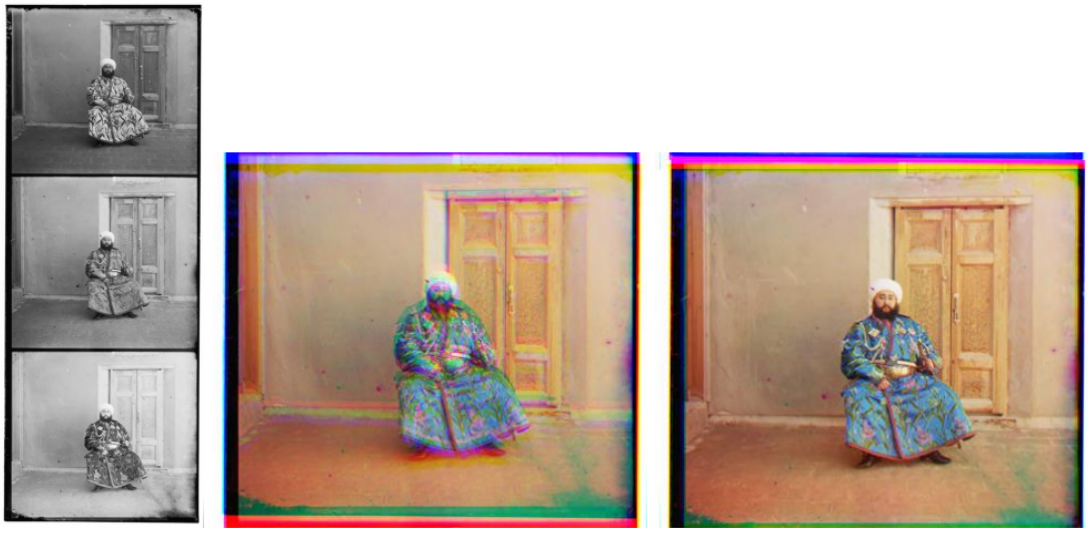
Robotics Algorithms (CSE468/568)

Submitted by:

ATHARV MAKARND SARAF - 50418885

Problem Statement:

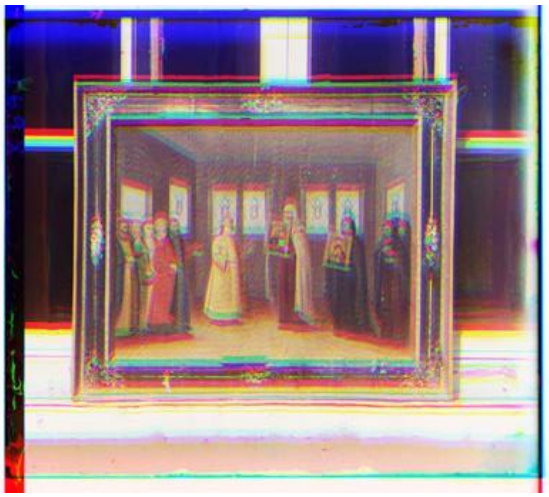
Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a photographer who, between the years 1909-1915, traveled the Russian empire and took thousands of photos of everything he saw. He used an early color technology that involved recording three exposures of every scene onto a glass plate using a red, green, and blue filter. Back then, there was no way to print such photos, and they had to be displayed using a special projector. Prokudin-Gorskii left Russia in 1918, following the Russian revolution. His glass plate negatives survived and were purchased by the Library of Congress in 1948. Today, a digitized version of the Prokudin-Gorskii collection is available on- line at http://www.loc.gov/exhibits/empire/gorskii.html. The goal of this assignment is to learn to work with images in Matlab/Octave by taking the digitized Prokudin-Gorskii glass plate images and automatically producing a color image with as few visual artifacts as possible. In order to do this, you will need to extract the three color channel images, place them on top of each other, and align them so that they form a single RGB color image. The assignment file (lab3.tar.gz) contains six images (img*.jpg) that you should run your algorithm on. Your program should take a glass plate image as input and produce a single color image as output. The program should divide the image into three equal parts and align the second and the third parts (G and R) to the first (B). For each image, you will also need to print the (x,y) displacement vector that was used to align the parts. Detailed description is below. Example images are shown in Figure below.



Sample image given to you as three separate images, one each for each color channel (left); Unaligned color image (center); Aligned color image (right)

Simple Color Images:

We are given 6 grayscale, BGR, column image respectively from top to bottom. We split the images based on the size of the overall image provided. Since there are 3 parts vertically, we split the image matrix in 3 parts vertically to assign BGE channels respectively. We can find that the image is 1024 which is not divisible by 3 hence a way out one can pad it with zero's. Her we could have done the padding of 1 increasing the row's to 1026, making it divisible by 3. We the concatenate the 3 matrices to form color. We obtain images saved as image*-color.jpg in the folder with this report.


Aligning Images:

Looking at the above results we can see, the images have the right colour intensity but are not properly aligned resulting in a blur effect.. Therefore, we use aligning algorithms namely Sum of squared differences (SSD) and normalized cross correlation (NCC).

1] sum of Squared Differences:

In SSD, we used the split matrices of BGR and cropped 2 parts out of the R and G channels, we will be using B as the reference. These cropped parts are then convolved over a certain possible shift. We then calculate the sum of squared difference for the particular overlapped window. This Sum gives us the idea of how correlated the windows are. The displacement with the best score is selected and the individual channel is shifted in x and y direction using these displacements. This process is done for 2 channels first, the template which is considered blue in our case and red or green. This shift is the actual misalignment which we remove by comparing the windows. The sum of squared difference is calculated by:-

SSD $$s = \sum_{(u,v)\in \mathbf{I}} (\mathbf{I}_1[u,v] - \mathbf{I}_2[u,v])^2$$

Finally, the shifted images are concatenated to give an aligned color image. This method is used in 'im_align1.m' included in the submission folder along with aligned images saved as 'image*-ssd.jpg' where '*' is the image number corresponding to the images provided.

While performing the SSD there are few observation that can seen:

- varying the ranges by which we shift/rote our window for match we vary the computation time. Higher the ranges more the computation, resulting in greater computation time. I chose the values to be [-15,15]
- The template size that we chose also plays a critical role, we played with certain window sizes and selected the ones which gave the same accuracy but least computation size.
- the noise at the borders of the image affects the matching process. we henced removed certain columns and rows to avoid misalignment resulting due to them.
- We did observe that varying the window size affected any one of the images. I was abe to adjust 5 of of 6 images.

Sample images:-

the following shift values where observed :

1. x1final = 9,x2final=5 ,y1final =1 ,y2final =2
2. x1final = 9,x2final=4 ,y1final =2 ,y2final =2
3. x1final = -4,x2final=7 ,y1final =-11 ,y2final =3
4. x1final = 13,x2final=4 ,y1final =1 ,y2final =0
5. x1final = 11,x2final=5 ,y1final =4 ,y2final =2
6. x1final = 5,x2final=0 ,y1final =1 ,y2final =0

2] Normalized cross correlation

In NCC, we split the image provided in 3 parts and crop out 2 parts out of the individual channels for faster processing. Both the size and the orientation of the correlation windows are determined according to the characteristic scale and the dominant direction of the interest points. For this, we use a template out of the blue channel and a part out of other channels and apply the following

formula to obtain translation displacement values.

ZNCC $$s = \frac{\sum_{(u,v)\in \mathbf{I}} \mathbf{I}_1[u,v] \cdot \mathbf{I}_2[u,v]}{\sqrt{\sum_{(u,v)\in \mathbf{I}} \mathbf{I}_1^2[u,v] \cdot \sum_{(u,v)\in \mathbf{I}} \mathbf{I}_2^2[u,v]}}$$

The displaced individual channels are then concatenated with the template image to obtain aligned images. It is noted that varying the template size and image size affects the resultant concatenated image. We cannot use different sizes since we are using 'dot product' of the matrices. Thus, keeping this in mind, we obtain a set of image sizes that produce acceptable images which are saved as 'image*-ncc.jpg' where '*' is the image number corresponding to the images provided.

Sample Images:

- NCC normalizes using the square of the complete image into the squared sum of the template making it ivarient to illumination variation.

the following shift values where observed :

7. x1final = 9,x2final=4 ,y1final =1 ,y2final =2
8. x1final = 9,x2final=3 ,y1final =2 ,y2final =2
9. x1final = 12,x2final=6 ,y1final =-4 ,y2final =3
10. x1final = 0,x2final=2 ,y1final =1 ,y2final =0
11. x1final = 11,x2final=4 ,y1final =4 ,y2final =3
12. x1final = 5,x2final=0 ,y1final =1 ,y2final =0

Feature-based Alignment:

For feature-based alignment, we use harris corner detector to obtain corners out of the image. Corners are an abrupt increase in intensity of pixels as compared to parts with no corners. Thus we use the following method to determine corners.

i. Compute x and y derivatives of the image using Sobel filters, use them to produce 3 images (I_x^2, I_y^2, and I_x*I_y).
ii. Compute the Harris matrix H for each pixel.
iii. Compute corner response function R = Det(H)/Tr(H), and threshold R. Try threshold values on the UI.
iv. We save R values and change matrix component to 1 for values where R is greater than the threshold.
v. Then we create a vector of greatest 200 values of R and use indices of those values to get the top 200 features detected by the algorithm.

As we see below, Figure1 shows all the detected features above a threshold and Figure2 shows the top selected features of image 6. Similarly, this is done for all images.

Note that, changing values of 'kpp','threshold','crop' and region changes the size of white spots (1's in the matrix of C). Greater value of threshold gives less features. Bigger size of 'crop' gives less detailed features.

Figure 1 - All detected features

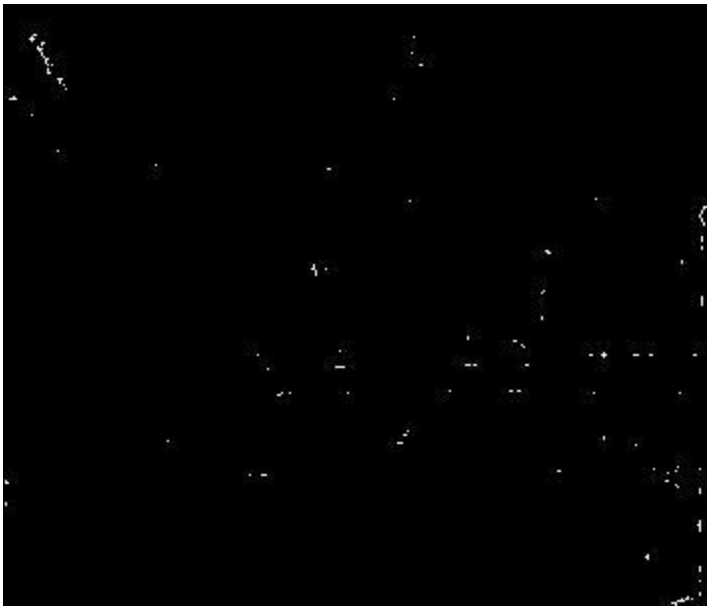


Figure 2 - Top selected features

After this, once we obtain features of red, blue and green channels of the images, we use RANSAC algorithm on the images to get an aligned stored as 'im_align3.m'.