Colorizing the Prokudin-Gorskii photo collection

Image Processing and Analysis - TNM087

Linköping University

Michael Novén
micno751

January 10, 2015

Contents

List of Figures

1 Introduction

The aim of this report is to extract information from negatives taken by Prokudin-Gorskii in early 20th-century Russia and to remake them into high quality color images. Automatic cropping will be implemented to compensate for unwanted borders that can appear after processing. The level of whiteness will also be adjusted in the final image to balance the difference between the separate RGB channels. The program should work in realistic runtime regardless of the quality of the negatives, this means that both *.jpg* and *.tif* images should run in realistic time.

2 Method

2.1 Materials

The images was taken from the website *http* : *//www.loc.gov/pictures/collection/prok* and the software *Matlab* was used for all the processing.

2.2 Read in the negatives

The original images contained all the RGB channels in that order from bottom to top, which can be seen in (a) of figure (1). The first step was to separate them into three different images once they were read into *Matlab*. It was assumed that every separate channel was $\frac{1}{3}$ of the total height of the image. The only important parts is the center parts of the images. This means that the borders has to be cut off so 10% of the outer border was removed to simplify the processing which can be seen in (b) of figure (1).

(a) Original image

(b) Cropped green channel

Figure 1: Original image divided and cropped

Just adding the different channels together will not suffice because each color channel of the image is not taken at the exact same position as the others.

2.3 Registration

In order to align the separate color channels, a couple of different known methods could be used. The aim was to get a displacement vector for each color channel by performing a search over a window with possible displacements. The window was chosen to be of size $[-15, 15]$.

2.3.1 Methods for calculating the displacement vector

Two common methods for template matching are SSD (Sum of squared differences) and NCC (Normalized cross correlation). SSD is a simple algorithm which take the sum of the square of the difference between each pixel in equation (1) to measure the distance. The minimum value then defines the displacement vector.

$$\sum_{ij \in W} (I_1(i,j) - I_2(x+i, y+j))^2 \tag{1}$$

NCC instead computes the dot product between two normalized vectors. This means that if the angle between two vectors is zero, the dot product will be one. Thus the maximum value in the range $[0, 1]$ was searched for by using the formula in equation (2).

$$\frac{\sum_{ij \in W} I_1(i,j) I_2(x+i, y+j)}{\sqrt{\sum_{ij \in W} I_1^2(i,j) \sum_{ij \in W} I_1^2(x+i, y+j)}} \tag{2}$$

The displacement that had the best matching score with both methods was then chosen and applied to the current color channel with the command $circshift()$ which circulary shifts the elements in an image. One of the resulting images is shown in figure (3), which shows that there is no significant difference between the methods. SSD however was considered faster than NCC, so it was used as the standard method through the rest of the experiment.

(a) Normalized cross correlation

(b) Sum of squared distance

Figure 2: Comparison between the two methods to get the displacement vector

2.3.2 Image pyramids

Calculating the displacement vector between large matrices became a very time consuming process for the high resolution $.tif$ images. A different approach was needed and an image pyramid was implemented. The images were scaled down and good matches was tried to be found in the lowest scale. The resulting offset could then be used to be an initial guess in a finer scale. The command $imresize()$ was used to make every image half the size of the one before and recursive calls was made on the function itself to get the coarsest scale and proceed from there. The idea of scaling down the image is illustrated in figure (3).

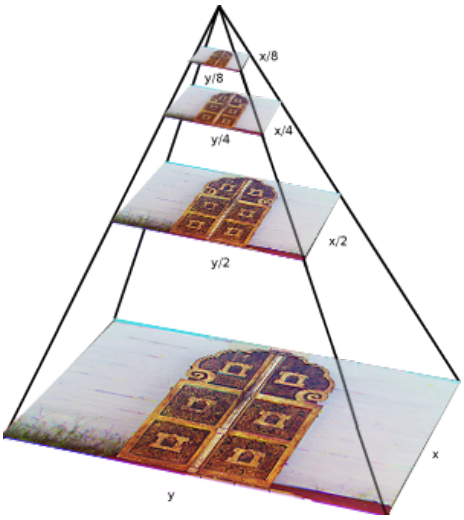
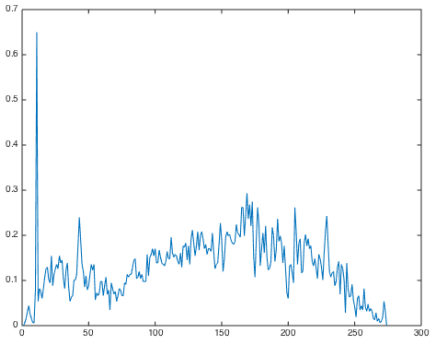


Figure 3: Illustration of the image pyramid

2.4 Automatic cropping

Automatic cropping was implemented to get rid of the borders around the resulting image. Canny edge detection was used for this purpose and each color channel of the images were transformed into logical matrices with zeros and ones at all the positions which is shown in (a) of figure (4). The mean values of the horizontal positions were then plotted and is shown in (b) of figure (4). The image was scanned from top to bottom respectively bottom to top to detect some mean values that was over a certain threshold value. If that was the case, is was considered to be a border and it was simply cut off. For example, it is possible to see from the plot in figure that there is an unwanted border near the top of the image. To reduce the amount of errors, only 10% on each side of the edged image was investigated. The same method was used for the vertical lines. An image with border before and after processing can be seen in figure (4).

(a) Edge detection of the red channel



(b) Mean intensity values for each horizontal line

(c) With border

(d) Without border

Figure 4: Automatic cropping

2.5 Automatic white balance

It may be possible that some color channels affect the final image more than others. In that case the white balance through the image had to be adjusted. The algorithm to do this is very simple, the color channels were once again extracted from the cropped image. The ideal values for red, green and blue was set to equal as $\frac{1}{2}$. The mean values for every color channel was then calculated to measure the difference between the ideal values and the current value. The difference was then simply added to the old color channels combining an image that looks much more realistic which can be seen in figure (6).

(a) An image with a high red channel

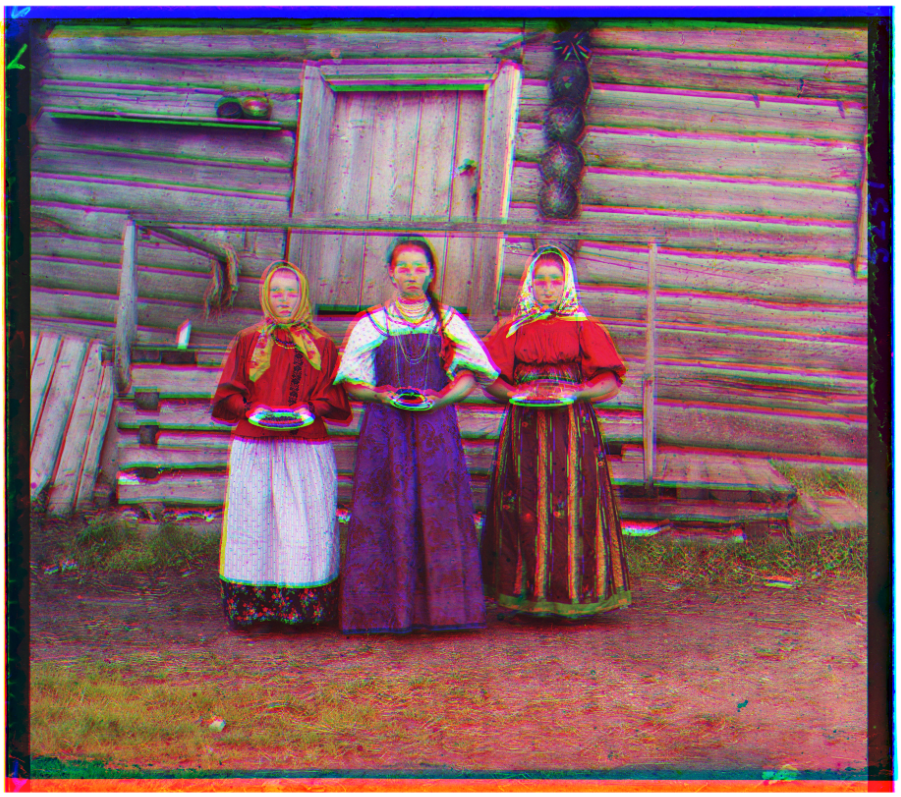
(b) Adjusted color channels

Figure 5: Automatic white balance

3 Results

To the left, the colored channels is just concatenated together on top of each other. To the right is the final processed image. The calculation time of the high resolution images is displayed as a caption.

3.1 Low resolution images



Elapsed time: 13.66 s

Elapsed time: 14.02 s

4 References

1. $http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/\_weg22/can\_tut.html$
2. $https://siddhantahuja.wordpress.com/tag/sum-of-squared-differences/$ 3.
$http://persci.mit.edu/pub\_pdfs/RCA84.pdf$