Computer Vision

Assignment 1 Report
By Chris Jimenez

I - Colorizing the Prokudin-Gorskii photo collection

Introduction
The astonishing images from Prokudin-Gorskii's photo collection are from his time during his travels across the Russian empire in the early 1900's. Using a technology that essentially differentiated between the R, G and B channels, he took photos that, at the time, could only be seen in full effect using special projectors. Technology has exponentially increased, and the photo collection lives on digitally in the Library of Congress.

Goal
With that said, the goal for this part of the assignment was to produce a color RGB image using the digitized Prokudin-Gorskii glass plate images with as few artifacts.

To do this, the three color channel images were extracted and placed on top of each other, making sure that they're aligned which should result in a single RGB color image. This, and also the remainder of this assignment, was done using MATLAB. This program was tested using six input images with their respective R, G, and B channels. They could be seen in Figure 1 below.

Figure 1: The 6 input images used to test the MATLAB program that will colorize/align them. The images show the B, G, R color channels from top to bottom respectively.

Method
The first thing that was done was dividing each input image in to three parts, separating each color channel evenly. Then the G channel was aligned with the B channel and the R channel was aligned with the B channel, essentially using the B channel as the "anchor" color channel. They were aligned by function called align() which takes, as inputs, two color channel images as parameters and returns the alignment of them, which is itself an image.

The alignment was done using the Sum of Square differences formula(SSD). Iterating over a window of [-15, 15] pixels, the channel being aligned was shifted and using that shifted image, the SSD between it and the base image, in this case the B channel, was calculated and compared to the best value and recorded. The best value was the smallest SSD recorded when iterating between all [-15, 15] pixels in the x and y direction. The x and y value that correspond to the best value was recorded as well. They can be seen below in Figures 2-7.

Once the color channels were aligned together, the result color image was displayed and saved. The result color image for each image in Figure 1 can be seen below in Figures 2-7 with their corresponding G and R channel displacement. The result images were a bit off, which probably has something to do with the SSD calculation. Using the alternative approach, normalized cross-correlation, could've produced better results.

Figure 2: The first image tested.
G:(7,-1), R(10, -2)

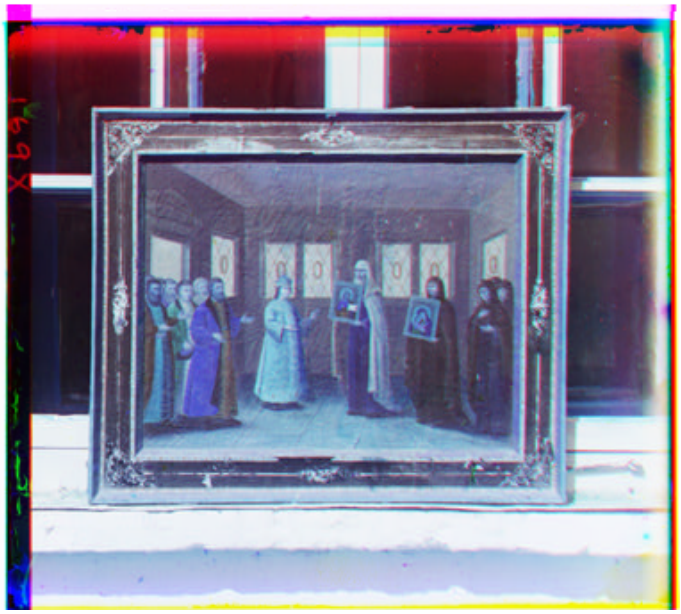

Figure 3: The second image tested.
G:(4, 1), R(9, 1)

Figure 4: The third image tested.
G:(4,0), R(13, -1)

Figure 5: The fourth image tested.
G:(15, 2), R(11, 3)

Figure 6: The fifth image tested.
G:(7, -1), R(8, -2)

Figure 7: The sixth image tested.
G:(0,1), R(8, 3)

II - Harris Corner Detector

Introduction

For this part of the assignment, the Harris corner detector was implemented in MATLAB. The function takes the following variables as inputs:

(1) `im` – A gray-scale image(2D array)
(2) `threshold` - "cornerness" threshold
(3) `sigma` - Standard dev. of the Gaussian used to smooth the 2nd moment matrix
(4) `radius` - Radius of non-maximal suppression

The function was applied to an image which outputs a 2xN matrix of corner locations on the image. The function also displayed the image with the corner locations superimposed. In this case, the image that was used for testing was the Einstein image, which can be seen in Figure 8. There are two key that were used for the function. The first computes the second moments at each point in the image, which will be smoothed by a Gaussian function w(u,v):

$$A = \sum_u \sum_u w(u,v) \left(\begin{bmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{bmatrix} \right)$$

I_x^2 = square of x derivative
I_y^2 = square of y derivative
I_{xy} =(y derivative)*(x derivative)

And the second equation computers the “cornerness” measure using A as the image:

$$M = det(A) - v \cdot trace^2(A)$$ -where *v* is 0.04

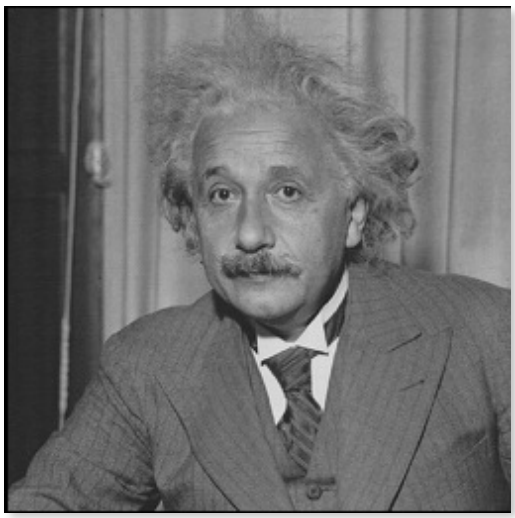

Figure 8: The Einstein image used to test the corner detection function.

Goal

The goal was to apply the function to the Einstein image with a *threshold* of 5e-4, *sigma* of 3, and *radius* of 3. The function created was called `cornerdetect.m` where it takes the input parameters mentioned earlier. Figures 9-12 below show the Einstein image superimposed with the following transformations, respectively:

(1) none
(2) rotated by 45 degrees using `imrotate`
(3) multiplying the intensities by 1.5
(4) resizing to half the resolution using `imresize`

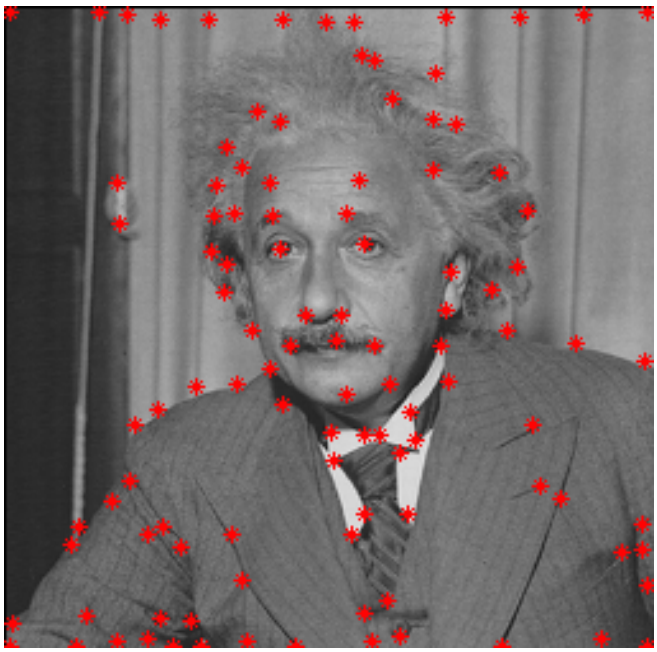

Figure 9: The Einstein image superimposed with the corner points.

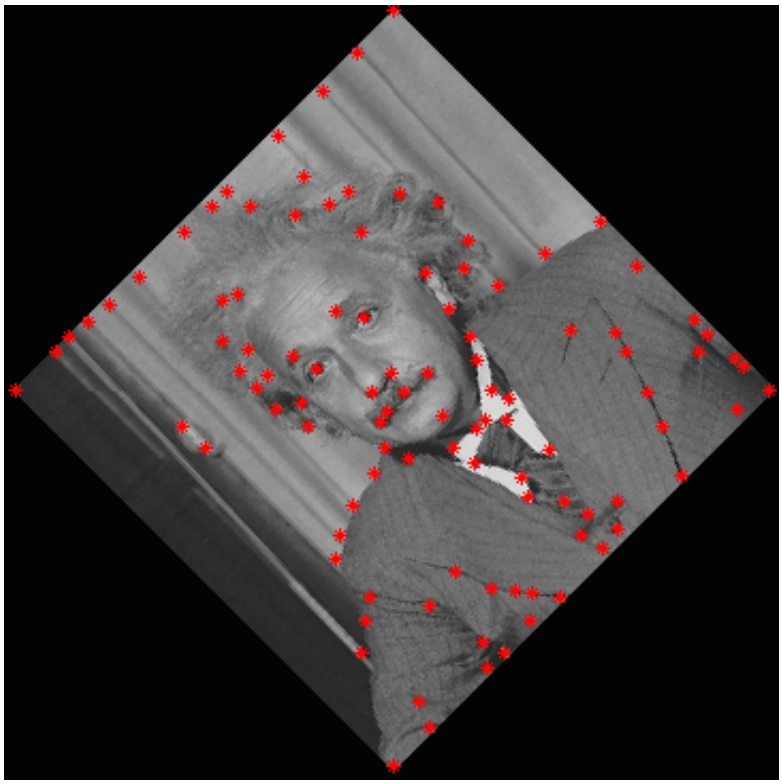

Figure 10: The Einstein image rotated by 45 degrees and superimposed with the corner points.

Figure 11 The Einstein image resized to half and superimposed with the corner points.

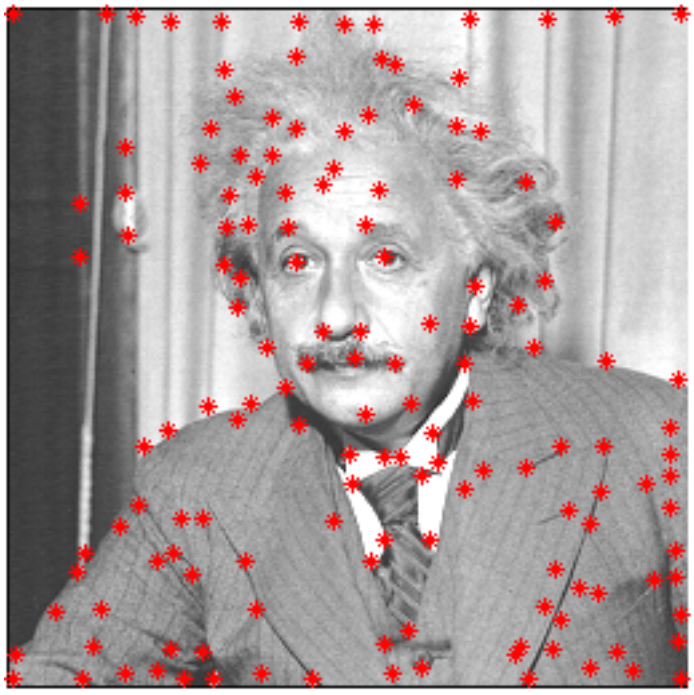

Figure 12: The Einstein image intensified by 1.5 and superimposed with the corner points.

III - Scale-invariance

For this part of the assignment, the scale-invariant measures used in blob detectors were looked at. In this case the Einstein image, shown in Figure 8, was used. The fixed location that will be examined is the following, which corresponds to the center of the tie:

```
row 186
col 148
```

For the first part of this assignment the Laplacian operator with standard deviation or sigma = 3, was generated and visualized using the `mesh()` function in MATLAB. The Laplacian Operator can be seen as :

$$\nabla^2 I = (\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2})$$

and its scaled-normalized version is

$$\nabla^2 I_n = \sigma^2(\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2})$$

The visualization can be seen below in Figure 13.

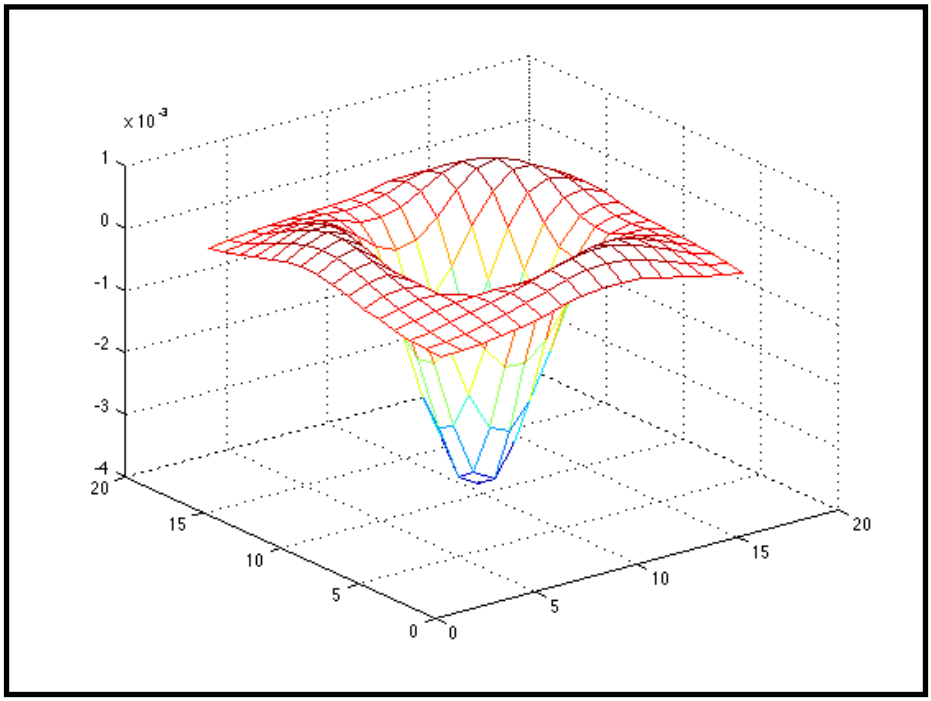

Figure 13: Mesh visualization of a Laplacian filter.

The Einstein image was then loaded in and another copy with half of its dimensions was made. The Laplacian filter was applied to both the full image and the half image and the response at the fixed location mentioned above was observed.

The laplacian filter, and its normalized version was applied to both the images through a loop of sigma values from 3 to 15 in increments of 0.4. The results can be seen in the plots in Figure 14. The green curve corresponds to the half size image, while the blue curve corresponds to the full-size image. As noted, the normalized operator yields two curves that have very similar maxima, whereas the non-normalized operator does not.

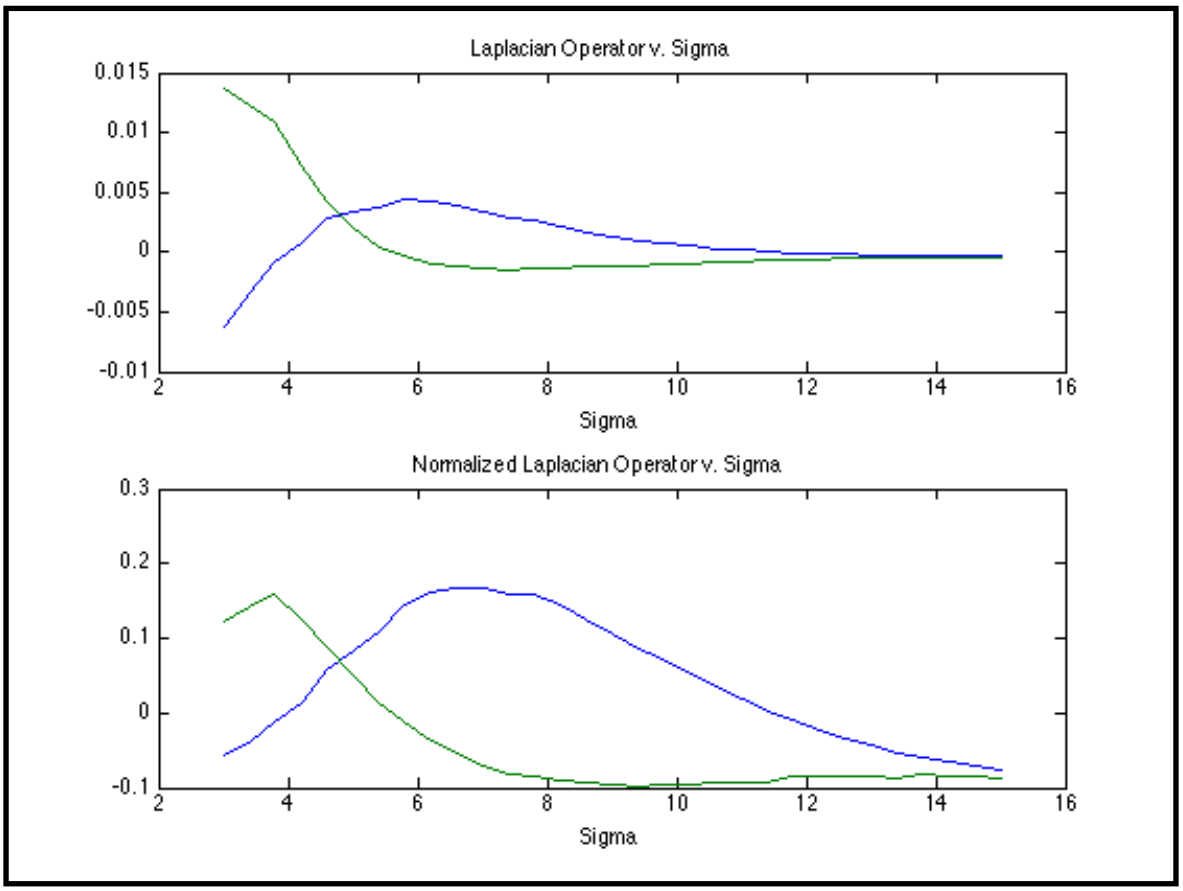

Figure 14: The Laplacian operators and its normalized version as a function of sigma(from 3-15 increments of 0.4)

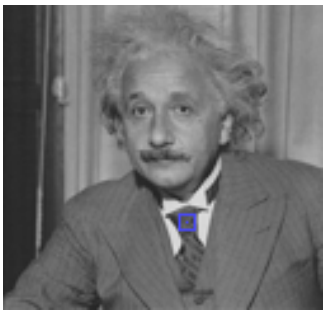

Figure 15: Half-sized image of Einstein with ellipse superimposed at the fixed location.

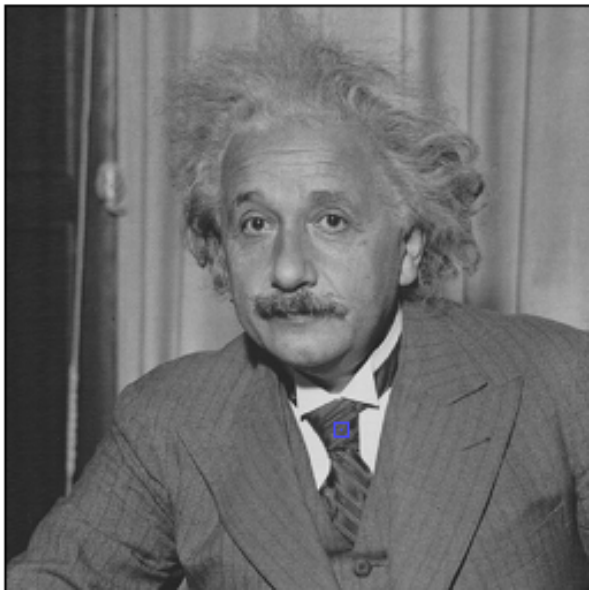

Figure 16: Full-sized image of Einstein with ellipse superimposed at the fixed location.

V - Image Alignment

This part of the assignment a function created in MATLAB takes two images as input and computes the *affine transformation* between them. This would then allow us to align one of the images to the other. The following are the steps, as described by professor Fergus in lecture 7, of how to perform the alignment:

(1) Find local image regions in each image.
(2) Characterize the local appearance of the regions.
(3) Get set of putative matches between region descriptors in each image.
(4) Perform RANSAC to discover the best transformation between images.

(1) and (2) was performed by using David Lowe's SIFT feature detector and descriptor representation code. The two images that were aligned can be seen side by side in Figure 17.

Figure 17: The two images that will be aligned using the created function.

(3) was performed by calculating the closest neighbor amongst the image 2 descriptors with each image 1 descriptor. That is, for each image 1 descriptor, compute the closest neighbor amongst the set of image 2 descriptors. To test the functioning of the RANSAC portion of the program, an upper limit threshold of 0.9 was used, which would get rid of some very wrong neighbor matches but still allow a few erroneous results in the set. The image descriptors for the scene and the book can be seen in Figure 18 and Figure 19 respectively.

Figure 18: The scene image with its image descriptors.

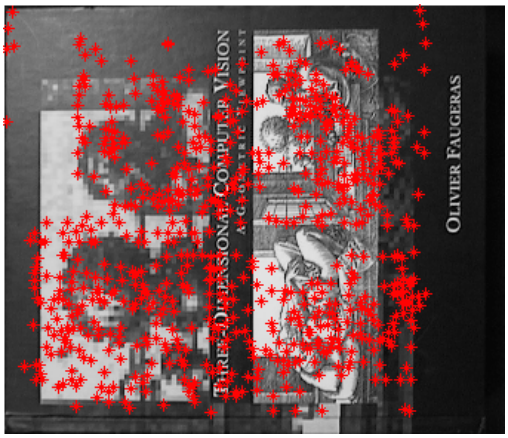

Figure 19: The book image with its image descriptors.

The function that performed the RANSAC operation is `ransac.m`. This function used the RANSAC algorithm to determine the best transformation parameters to ultimately apply to, in this case, the book in Figure 17, so that its orientation could match that seen in the "scene" image where the same book is also located. The RANSAC loop algorithm is the following:2

(1) Randomly select a seed of matches
(2) Computer transformation form seed group
(3) Find inliers to this transformation
(4) If the number of inliers is sufficiently large, re-computer least-squares estimate of transformation on all of the inliers

Then ultimately keep the transformation with the largest number of inliers.

The best transformation found by the program was applied to the image of the book. The transformed image should have had the same orientation and size seen in the "scene" picture seen above. However, the program had a different output. The angle at which the book is tilted can be seen in the output image but the corners of the book didn't correspond. That is, the output image was just a stretched out parallelogram version of the book. The output image can be seen in Figure 20. The values for the transformation matrix, H, can be seen below.

$$H = \begin{bmatrix} 0.6251 & 0.4852 & 0.6056 \\ 0.0024 & 0.5916 & 0.0027 \\ 0 & 0 & 1.000 \end{bmatrix}$$

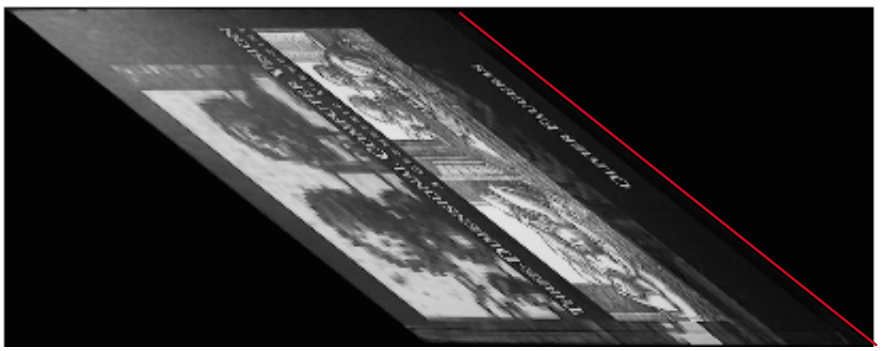

Figure 20: The output image with the book orientation to that seen in the scene image.

The output image didn't completely orientate to the way the book was orientated in the scene image. The top left and the bottom left points didn't move respectively to the rotation. However, the angle at which the output image is rotated seems to be roughly the same as the book rotated angle in the scene image. The angle is exaggerated by the red line in Figure 20.