Doran Walsten, Ron Boger

November 4, 2014

Colorizing the Prokudin-Gorskii photo collection of images of the Russian Empire

Figure 1: Reconstructed colored image of a Russian house, after aligning and adding image enhancement effects

The goal of this project was to colorize Sergei Mikhailovich Prokudin-Gorskii's black and white images of the Russian empire. Each of his images were provided as a set of 3 images, where the top image was taken with a Blue filter, the middle with a

Green filter, and the bottom with a Red filter. The solution to reconstructing the original image is not as trivial as simply grouping together the R, G, and B channels, as Gorskii moved the camera slightly as he was taking the pictures.

Our general approach to aligning the three images was to choose the green plate as the template/common image and find the transformation from the blue and red images to that image. Green was chosen as template image because the human eye has the highest sensitivity to green wavelengths; thus it was imperative that if the algorithm did not perfectly align all images, green could still extremely well aligned with at least one color to create the best possible color image. This transformation between the images was found using a 2-dimensional RANSAC algorithm. To align the images, we applied ideas from the *sift_ mosaic* method in VLFeat to superimpose them with greater precision.

Before even going into the algorithms to find an affine transformation, we needed to make a few modifications to Gorskii's images. All of the images came in a filmstrip format with Blue(B) as the top image, Green(G) as the middle image, and Red(R) as the bottom image. To extract good images from this filmstrip, we divided split up the filmstrip into thirds, and then concatenated the edges of the R, G, and B images. This was important to do, as many of the images have extra noise from the scanning and the paper they are on, which alters the accuracy of the SIFT keypoints.

After making the necessary adjustments to the images, the algorithm proceeds by fitting an affine transformation from the Blue to the Green image, followed the transformation from the Red to the Green image using RANSAC. Our RANSAC follows these main procedures:

1) Generate a set of two point correspondences between image 1 and image 2. This is done by using *vl_ sift* to find the feature points and descriptors of both images and then using the *vl_ ubcmatch* to find the closest matches between feature points and

thus create a candidate set of point correspondences. We chose to use SIFT because of its robustness in finding keypoints in an image with invariance to illumination, scale, and rotation.

a) To add extra accuracy to the SIFT descriptors, SIFT keypoints that within 3% of the edges of the image are discarded. This was an issue we realized, as despite our efforts to crop the image, some images have noise/borders due to being scanned at a slight angle with respect to the scanner. We hypothesized that this decreased the accuracy of our affine transformation, and indeed after making the change, we did see slight improvements in accuracy.

b) We had a lot of fun adjusting the peak threshold of *vl_sift* and *vl_ubcmatch* to see how it impacted feature selection and matching. We found that sometimes, there would be too many feature points/poor matches for the algorithm to effectively run through them. At other times, we would make the threshold too high (in either case) and the opposite would occur. In the end, we found that the default peak threshold is good enough to match images at the low-resolution level. However, the threshold does need to increase as the size of the image increases.

2) Choose 2 point correspondences at random from the candidate set

3) We found A and T using the previously derived formulas:

$$A = YX^T(XX^T)^{-1}, \quad T = \bar{y} - A\bar{x}$$

Where $X \in R^{2x2}$ is composed of the 2 selected points from the first image, $Y \in R^{2x2}$ is composed of the 2 selected points from the second image, and $\bar{y}$ and $\bar{x}$ refer to the average difference between the coordinate positions of a point correspondence made between image 1 and image 2.

a) We tried computing the transformation and translation in a different way after implementing the full algorithm. We used 3 correspondences and one equation

that computes A and T simultaneously (using homogeneous coordinates). Surprisingly, this yielded a significantly less accurate result despite the fact that the least squares approximation for 2 point correspondences by its mathematical construction performs worse on noisy images.

4) After analytically determining the affine matrix and translation, the error is computed by:

$$\|y_j - (Ax_j - T)\|_2$$

Where the number of inliers is all pairs that satisfy:

$\|y_j - (Ax_j - T)\|_2 < \epsilon$, where ϵ is a threshold for the error

   a) We felt that despite the loss in memory efficiency, it would help to store the number of inliers, transformation matrix, and translation vector in a cell for each iteration. This makes our algorithm more general yet specific to our situation. Instead of arbitrarily picking a threshold for the number of inliers within the method itself, we assume in most cases that with enough iterations we will get many appropriate transformation matrices and translation vectors. Arbitrarily picking a threshold for the number of inliers for these transformation leads to results with decreased repeatability, as some images generate significantly more feature points due to their size or number of distinctive points. We let our master function, *rgb_ merge*, decide if this is enough inliers or not.

5) We repeated this process for a given number of iterations, storing each computed A, T, and the number of inliers associated with those transformations.

6) After the completion of all the iterations, we determined which transformation produced the most inliers and returned that transformation.

One obstacle that we encountered during our algorithm development was the tradeoff between efficiency and accuracy. For the high-resolution images, RANSAC ran rather slowly, and generating features and finding matches with the VLFeat tools took much too long. As a result, we decided to try our hand at downsampling/making an image pyramid. We downsampled until we got to an image close to the size of the low-resolution images (~400 x 400 pixels). We tried two methods of getting the global A and T. In the first, we just found the transformation at the "top" of the pyramid(smallest) and applied this transformation to the original image. In the second image, we tried computing A and T for a given level, apply it to the level below, then run RANSAC again on that level to get a new A and T. We applied these steps at each level in our pyramid until we reached the original image level. In theory, this method would create a better affine transformation, as we would be continually improving the determined transformation as we went up the pyramid to reach the original image. Regrettably, in practice this did not perform consistently across all images, and added a considerable amount of running time to the algorithm. Conceptually, the lack of consistency makes sense, as continually applying affine transformations operates similar to a power series/Euler approximation/Picard integration. Although often we do indeed improve the approximation at each step and "converge" at the desired image, sometimes even one faulty transform can cause the image to "diverge" from the target image. Thus, we concluded that using the determined transformation from the most downsampled image produced an accurate transformation for the regular image, where:

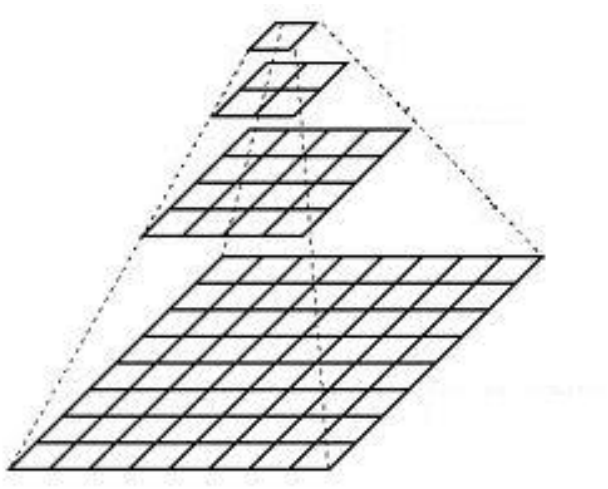

Figure 2: Pyramid of downsmapled images

$$A_{regular\ image} = A_{downsample\ image}\ and\ T_{regular\ image} = (T_{downsampled}) * 2^n,$$

where n is the number of times the image was downsampled.

In addition to doing well, this method only takes 5-6 seconds to perform the alignment for a high-resolution image. (See *tic* and *toc* in the code).

We referred to the *sift_ mosaic* method to figure out how to align the images based on A and T. The steps of the algorithm are outlined below:

1) Transform the corners of both the Blue and Red images based on the transforms computed from RANSAC to find the new size of these images.

2) Determine the smallest and largest value in both the x and y directions. This allows us to generate a meshgrid of all these points.

3) The meshgrid is used in conjunction with *vl_ imwbackward* to find points in the original image that map to the exact points in the meshgrid that we generated. We then generate an RGB image by setting each row of a 3D matrix to the appropriate color.

This worked very well. We originally didn't implement this because we thought that simply overlaying the images after applying the respective transformations would be enough. Surprsitingly, this worked well for some images, but the results were substantially better after aligning using the logic from *sift_ mosaic*. Our original method was nice and quick, but did not align the images very well in most cases.

We also tried a few image enhancement techniques in MATLAB to see if we could improve the image even further (and get a little extra credit). Commented out in the code is a de-blurring for the image. Although it seems like this would help with some of the images, it causes some aliasing in the others, so we chose to leave it out. Two more figures after the original image are made. The first is an image using the *imcontrast* function of MATLAB. This makes no noticeable difference to some images, but to others it does improve the quality. The 2nd figure produced is the image after being passed through the function *adapthiteq*. This enhances the contrast and specific

areas of the image based on their historgram values, and produces a noticeable effect that looks similar to taking a picture with HDR.

Although Gorskii perhaps did not do a great job holding his camera in place as he replaced the filters, he was truly a man ahead of his time. In conclusion, this project was really fascinating and cool to implement. There were a lot of ups and downs depending on how well the images aligned, and that made the work fun. Once we figured out how the VLFeat functions worked, it was a matter of designing the RANSAC and aligning algorithms to finish the job. However, our functions are not perfect and still make mistakes. For example, our code has the most trouble when dealing with images with spires on buildings. This is due to the fact that some filters do not show a significant difference between the building and the sky due to the filter changes between images. Consequently, no features nor matches are detected in these regions of the image. This resulted in good transformations for other parts of the image, yet produced some aliasing near the spires. This is an area that we would like to improve upon - accounting for differences in contrast for specific filters.

Results (Low resolution, followed by high resolution):

Figure 4

Figure 3: The man in the assignment instructions


Ав 132

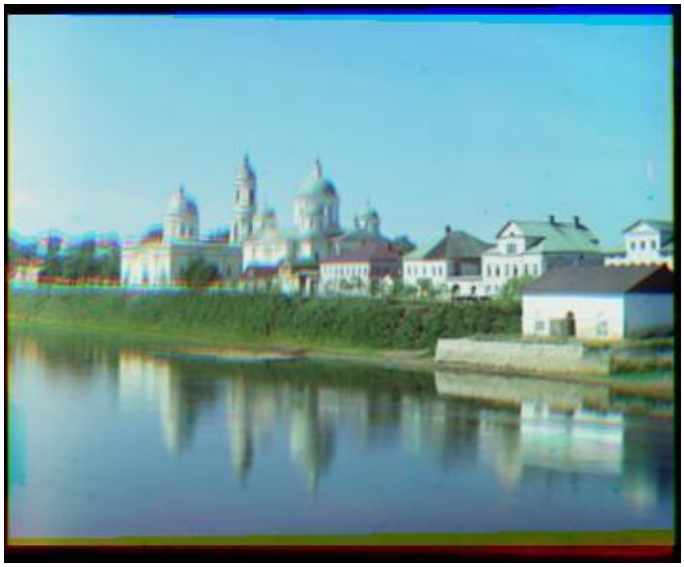

Figure 5: High resolution image #1. Notice the detail on the people on the side.

Figure 6: An example of a high resolution with spires, as mentioned in the report

Figure 7: Our favorite high resolution image, this time without contrast enhancements