CS180 Project 1: Yash Jain

Images of the Russian Empire: Colorizing the Prokudin-Gorskii photo collection

Introduction

This project focuses on looking at image slabs given in R, G, B scales that when with no manipulations placed on top of each other lead to blurry images. The goal of the project is to not only place the images on top of each other but to also generate 'aligned' and non-blurry images. We focus on a few approaches -- initially only optimizing for image quality and soon after speed. Initially, we started by retrieving the images from the slab in B, G, R scales. Using np.stack to stack the B, G, R images worked to produce a colored image but failed to produce a CLEAR image. Thus, the algorithm looks to identify an (x, y) vector such that you can shift the G and R images onto the blue image and retrieve a clear and non blurry image.

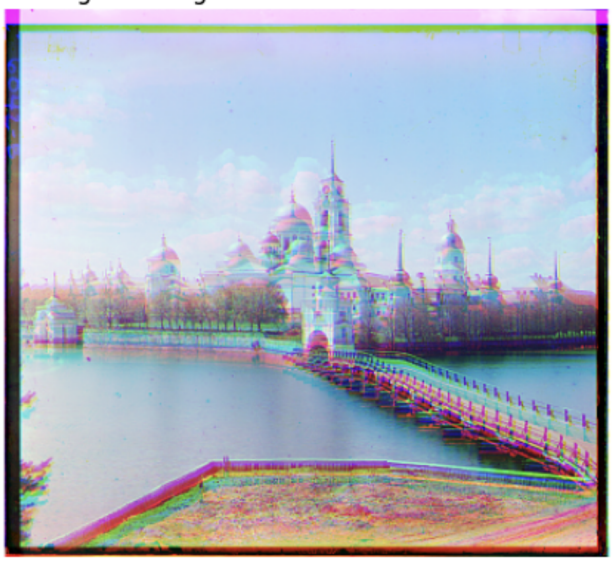
Algorithm & Description Choices:

After converting the slab image into 3 channels, I initially started by experimenting with different error metrics. Now you might be wondering what the error metric is — we use the blue channel as the source image and attempt to align the red and green channel source images to the blue image. Alignment is defined as perturbations in the X and Y direction to shift all pixels by some X and Y value. We are attempting to minimize error in other cases maximize values depending on the metric used.

The image quality didn’t improve much — a sample below:

```
Green Displacement Vector: (-6, 0)
Red Displacement Vector: (9, 1)
```

Aligned Image with EUCLIDEAN DISTANCE



Next, I wanted to experiment with other metrics that could possibly work and came across NCC which is the Normalized Cross Correlation Coefficient

$$NCC = \frac{\boldsymbol{A}}{|A|} \cdot \frac{\boldsymbol{B}}{|B|} = \sum_i \frac{A_i}{|A|} \cdot \frac{B_i}{|B|}$$

This metric performed well and attached are images and the respective vector displacements:

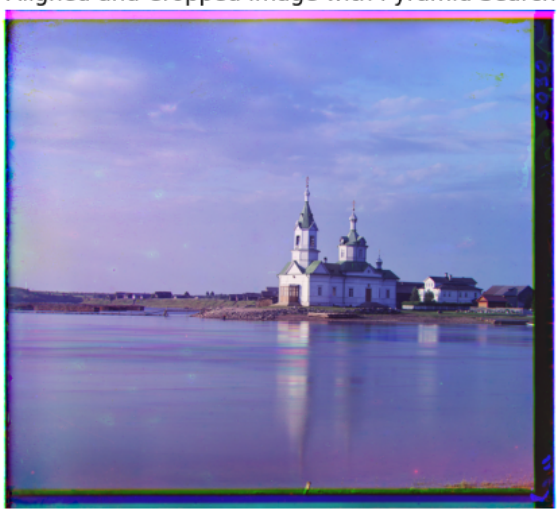
The following images contain BOTH .TIF and .JPG images for single scale and pyramid implementation. The runtimes for each of these fall under 6 seconds — though runtimes for JPG is < 1 second.

Aligned and Cropped Image with Pyramid Search

Vector Green: (3, 2)

Vector Red: (11, 3)

Aligned and Cropped Image with Pyramid Search



Vector Green: (25, 4)

Vector Red: (58, -4)

Aligned and Cropped Image with Pyramid Search

Vector Green: (42, 5)

Vector Red: (87, 32)

Aligned and Cropped Image with Pyramid Search

Vector Green: (-3, 2)

Vector Red: (3, 2)

Aligned and Cropped Image with Pyramid Search

Vector Green: (78, 29)

Vector Red: (176, 37)

Aligned and Cropped Image with Pyramid Search

Vector Green: (51, 9)

Vector Red: (111, 12)

Aligned and Cropped Image with Pyramid Search

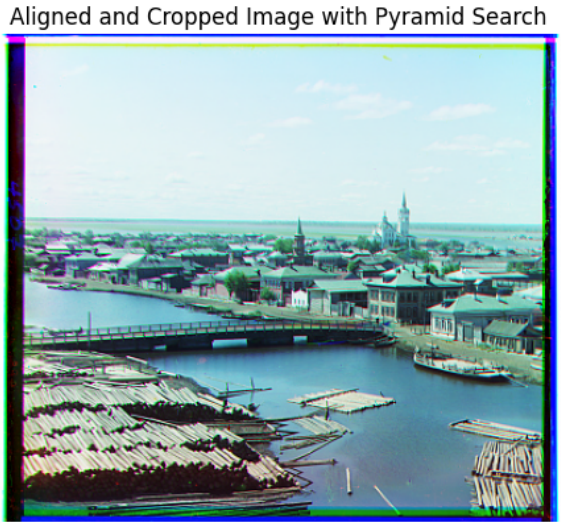
Vector Green: (51, 9)

Vector Red: (111, 12)

Aligned and Cropped Image with Pyramid Search

Vector Green: (53, 14)

Vector Red: (112, 11)



Vector Green: (59, 16)

Vector Red: (124, 13)

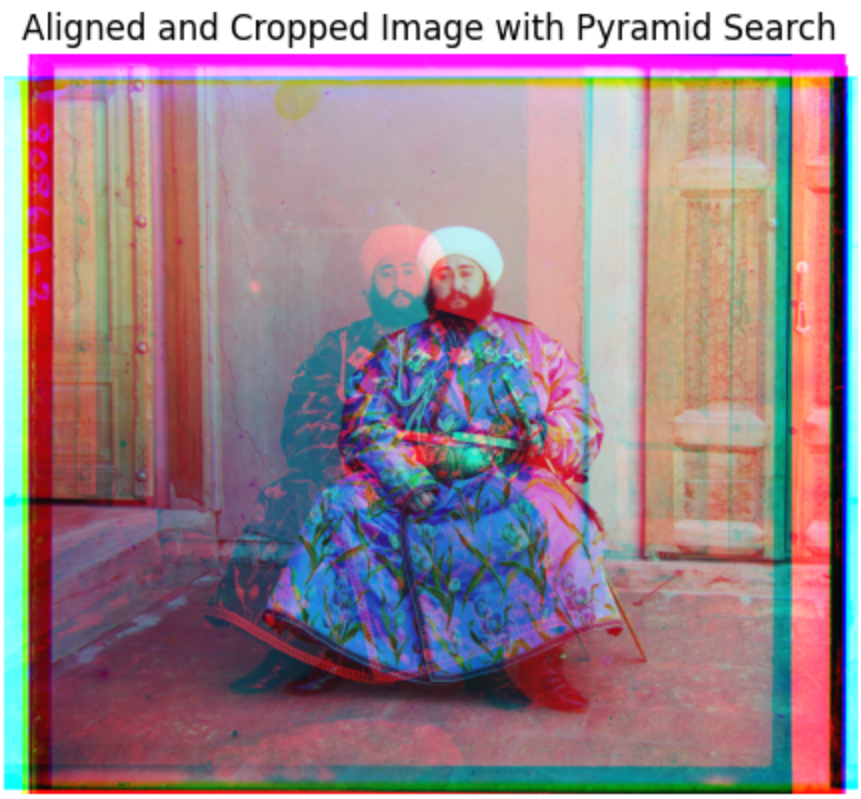
Vector Green: (3, 3)

Vector Red: (6, 3)

These images were generated based on alignment via NCC (Normalized Cross Correlation Coefficient). A problem I ran into while generating these images was for .TIF images was the processing time was quite high. Thus to counter this problem, I implemented Pyramid Search. This downsizes the image until one of the images reaches 32 pixels. Then I set up a 2×2 search grid that goes through the image and runs the same NCC based error metric to align the images. And this optimization led to the speed to go from upwards of 5 minutes to a few seconds (3-7 seconds).

Failure Case for NCC & Solution Found

However, a failure case was the emir image on which another metric may perform better.




```
Displacement for G: [49, 24]
Displacement for R: [98, -206]
```

Now to account for this — I began looking into other metrics and came across SSIM which an image similarity score widely used in computer vision.

SSIM (Structural similarity index measure)

The following defines the formula for SSIM:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

We can use the SSIM as the value we optimize and maximize for when shifting and displacing the red and green channel images to identify optimal placement. After implementing this — the changes seen in image quality for the emir were phenomenal.

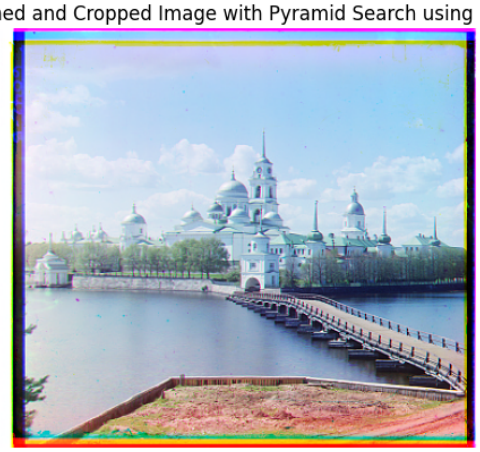
```
Green Vector Displacement: (-50, -23)
Red Vector Displacement: (-106, -41)
```

Using the SSIM as a metric to optimize for allowed for increased image quality. What this opens the door to is the idea that using these different error & similarity metrics are a great way of trying to find the optimal displacement vector. Each metric lead to different results — some better, some slightly worse.

Attached are images with SSIM as the metric being optimized on.

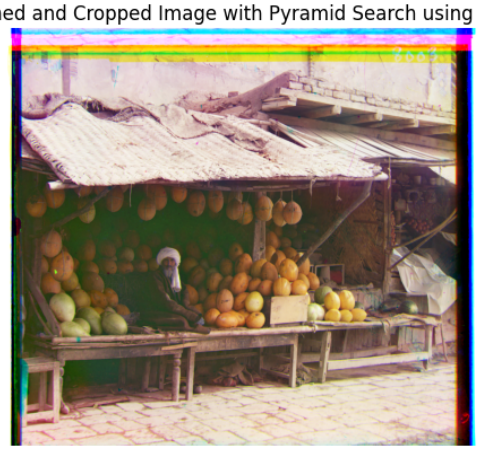
Aligned and Cropped Image with Pyramid Search using SSIM

Aligned and Cropped Image with Pyramid Search using SSIM

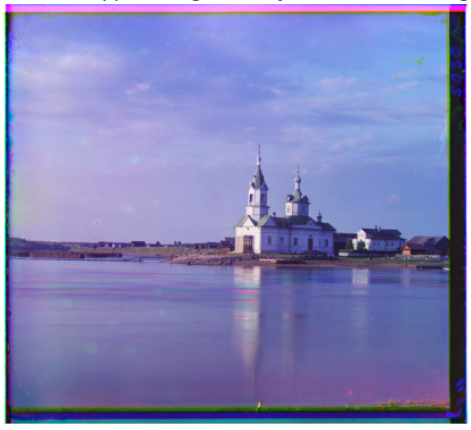


Aligned and Cropped Image with Pyramid Search using SSIM

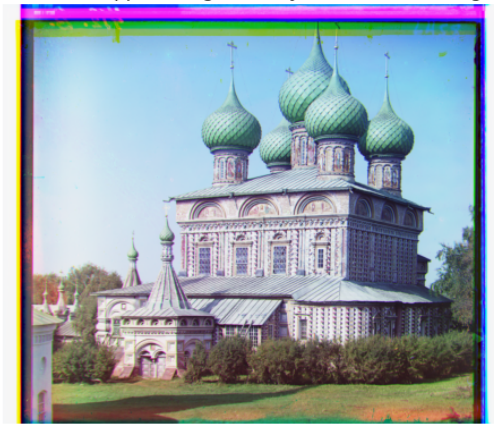
Aligned and Cropped Image with Pyramid Search using SSIM



Aligned and Cropped Image with Pyramid Search using SSIM

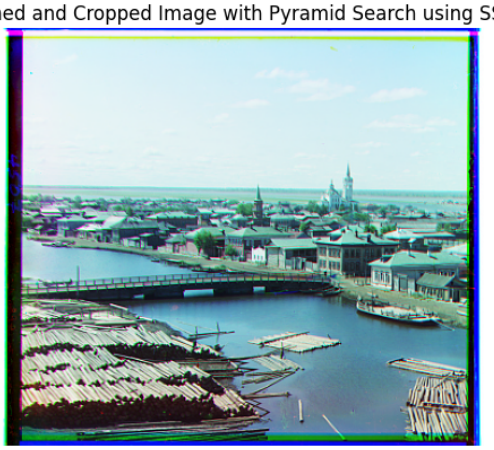


Aligned and Cropped Image with Pyramid Search using SSIM



Aligned and Cropped Image with Pyramid Search using SSIM

Aligned and Cropped Image with Pyramid Search using SSIM



Final Remarks and Conclusion:

In this project, we converted slabs of .JPG and .TIF images that were initially grayscale into RGB images and also successfully aligned the images correctly in terms of the 3 different color channels. More importantly, I implemented and experimented with different error metrics such as Euclidean Distance, SSIM, Normalized Cross Correlation Coefficient and no error metric at all.

In order to ensure fast processing, I also implemented the downsizing so that we can quickly process over the images and return an alignment shift vector.