Assignment 2

Mu-Ruei Tseng

February 15, 2024

1 Overview

In this assignment, we are required to merge the images from the Prokudin-Gorskii glass plates, which represent the Blue (B), Green (G), and Red (R) channels of the image, arranged in a top-down order. We accomplish this by aligning the R and G channels with the B channel, identifying the alignment that minimizes the Sum of Squared Differences (SSD). Task 1 involves creating a straightforward **FindShift** function capable of handling small JPEG images. In contrast, Task 2 calls for a more efficient search method using an image pyramid. This approach significantly enhances execution time by conducting the broadest search at the lowest pyramid level, i.e. the smallest image. As we ascend to higher levels, the required search range for shifts narrows. Detailed explanations of the configuration and implementation will be provided in subsequent sections.

2 Task 1

2.1 Method

To find the optimal horizontal and vertical shifts to align the Red and Green channels with the Blue channel, I first examined the input images. I noticed that all the images have borders which, if not removed, could skew the minimum SSD value and hinder the performance of finding the optimal shift. Therefore, to address this issue, I preprocessed all the JPEG images by trimming a constant border of b_0 before starting the SSD calculation.

To determine the optimal shift, I shifted the image in both the x and y directions within a certain search range, s_d. Given that we assume the displacement in both the x and y directions to be relatively small for these images, it becomes feasible to iterate through all possible shifts, i.e., from $-s_d$ to s_d, to find the minimum SSD between the R/G channel and the B channel. The shifting is achieved through circular shifting. Therefore, for this **FindShift**, we have two parameters b_0 and s_d. We set it as follows:

$$b_0 = 20, s_d = 20 \tag{1}$$

Here is the pseudo-code:

```
function findShift(im1, im2, $s_d$, $b_0 = None$):

    if $b_0$ is not $None$ do
        crop out the border $b_0$ for both im1 and im2
    end

    min_ssd ← inf

    for i = −search_range to search_range do
        for j = −search_range to search_range do
            # shifting the image
            shifted_im1 ← circ_shift(im1, (i, j))

            # calculating the SSD
            ssd = np.sum((shifted_im1 − im2) ** 2)

            if ssd < min_ssd do
                min_ssd ← ssd
                min_d_y ← i
                min_d_x ← j
            end
        end
    end

    return (min_d_y, min_d_x)
end
```

2.2 Result

Here are the alignment results with their optimal shifts using the simple **FindShift** method, see Figure 1.

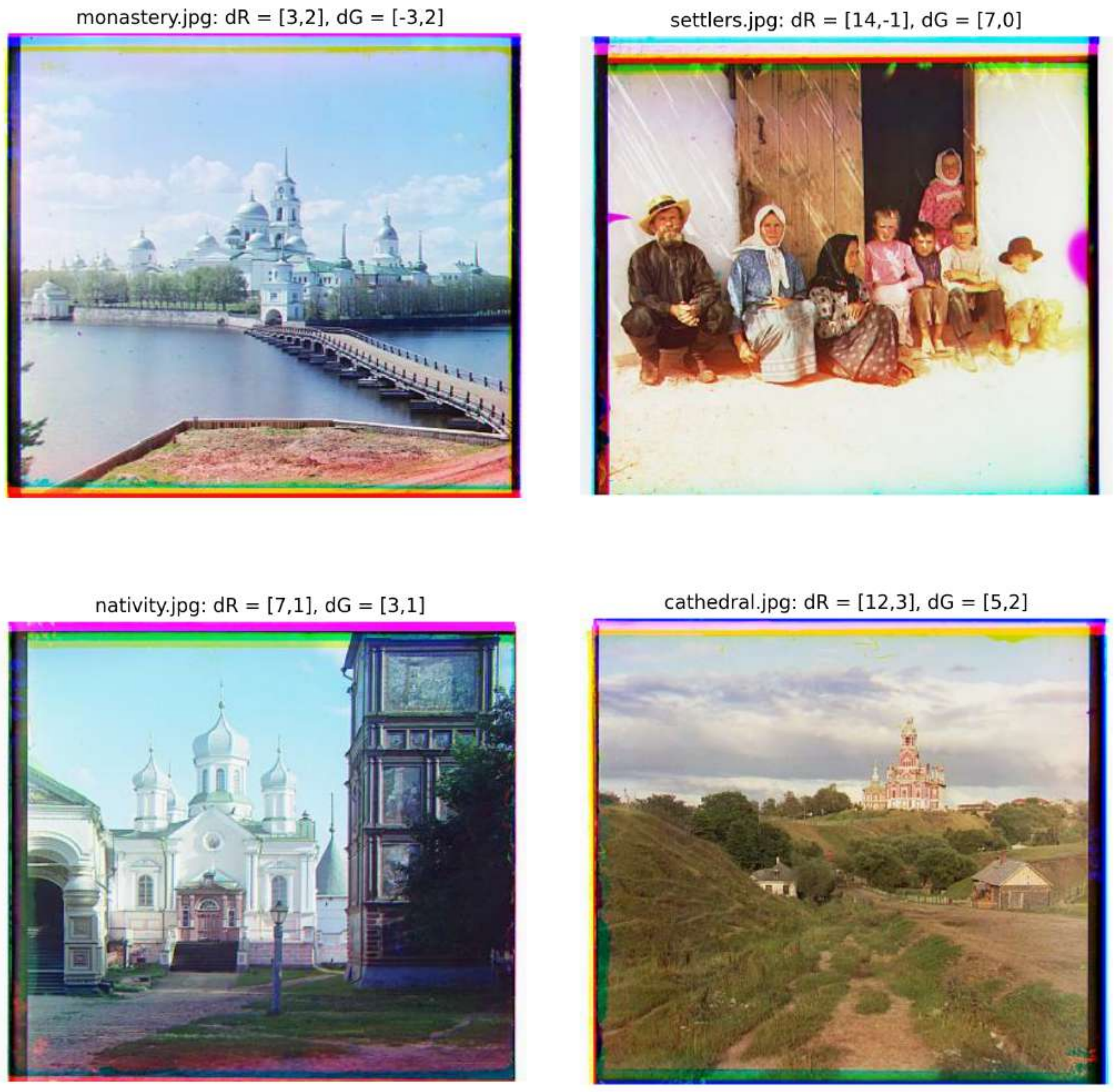
monastery.jpg: dR = [3,2], dG = [-3,2]
settlers.jpg: dR = [14,-1], dG = [7,0]
nativity.jpg: dR = [7,1], dG = [3,1]
cathedral.jpg: dR = [12,3], dG = [5,2]

Figure 1: Task 1: Results of Alignment with Smaller Images. *Top Left:* monastery.jpg, *Top Right:* settlers.jpg, *Bottom Left:* nativity.jpg, *Bottom Right:* cathedral.jpg. $dR = [d_y, d_x]$ indicates the vertical (y-axis) and horizontal (x-axis) shifts required to align the R channel with the B channel.

Figure 1 shows the alignments for all the images.

- monastery.jpg:
  - R channel: shift downwards by 3, rightwards by 2
  - G channel: shift upwards by 3, rightwards by 2
- settlers.jpg:

  - R channel: shift downwards by 14, leftwards by 1
  - G channel: shift downwards by 7

- nativity.jpg:
  - R channel: shift downwards by 7, rightwards by 1
  - G channel: shift downwards by 3, rightwards by 1

- cathedral.jpg:
  - R channel: shift downwards by 12, rightwards by 3
  - G channel: shift downwards by 5, rightwards by 2

3 Task 2

For Task 2, we need to work with larger images, namely, the .tif files. In these cases, the amount of shift required for optimal alignment might be larger. For instance, for an image of size 128x128, assume the maximum shift does not exceed one-fourth of the original image size; the search shift distance can be limited to approximately -20 to 20. However, for images of size 1024x1024, the search range may need to increase by 8 times in both the x and y directions. This adjustment leads to a total of 64 times more operations required to find the optimum shift.

To address the issue, we can create an image pyramid with multiple **levels** that keeps resizing the original into smaller images. We then perform the largest search (-s_d to s_d) on the smallest image and fine-tune within a smaller range (-s_f to s_f) for the upper levels. We denote the image I_0 as the original image and s_0^* as the optimal shift where s_0^* is smaller than some large value s_D. If we downscale the image resolution by 2 for each level, i.e. reduce halving both width and height, we can obtain the following at any level i:

$$\text{Area}(I_i) = \left(\frac{1}{2^i}\right)^2 \text{Area}(I_0) \tag{2}$$

$$s_i^* = \frac{s_0^*}{2^i} <= \frac{s_D}{2^i} \tag{3}$$

Assume we have n levels, i.e. $level = 0, 1, ..., n-1$. In the smallest image ($level = n-1$), we have $s_{n-1}^* <= \frac{s_D}{2^{n-1}} = s_d$, where $s_d << s_D$. This helps us find the shift at a smaller image, assume the optimal shift we found is denoted as s_{n-1}. Therefore we have:

$$|s_{n-1}^* - s_{n-1}| < \epsilon \tag{4}$$

, where ϵ is a small number. Next, for $level = n-2$, we can multiply the equation above by 2 and get:

$$|s_{n-2}^* - 2s_{n-1}| < 2\epsilon = s_f \tag{5}$$

This means that we can first shift the image by $2s_{n-1}$ and refine the shift around $-s_f$ to s_f only to obtain s_{n-2}. We keep performing the same method until it reaches $level = 0$. This method largely reduces the amount of search needed.

Therefore, the overall pseudo-code for the **FindShift** algorithm with larger images is:

```
function findShiftPyramid(im1, im2, $n$, $b_0$, $s_d$, $s_f$):
    crop out the border $b_0$ for both im1 and im2

    build the image pyramid with n levels

    s ← 0
    for i=n−1 to 1 do
        if i==n−1 do
            $\_s$ ←findShift(im1, im2, $s_d$) # same function as task 1
        else
            $s$ ←$2s$
            shifted_im1 ← shift im1 by $s$
            $\_s$ ←findShift(shifted_im1, im2, $s_f$)
        end
        $s$ ←$s + \_s$ # finetune the shift
    end
    return $s$
end
```

For this **FindShift** function for larger images, we have four parameters to set: n, b_0, s_d, s_f. We set it as follows:

- n: the number of levels. Here we set it as 4.
- b_0: the border to crop out initially. Here we set it as 200.
- s_d: the search range at the smallest image (from $-s_d$ to s_d). Here we set it as 20.
- s_f: the search range for other images in the image pyramid. Here we set it as 2.

3.1 Results

Here are the alignment results with their optimal shifts using the FindShift method with an image pyramid, see Figure 2, 3, and 4. The same notation is used for representing the shifts in both R and G channels towards the B channel in this section.

- emir.tif:
  - R channel: shift downwards by 88, rightwards by 44
  - G channel: shift downwards by 49, rightwards by 24
- three_generations.tif:
  - R channel: shift downwards by 111, rightwards by 10
  - G channel: shift downwards by 53, rightwards by 14
- train.tif:
  - R channel: shift downwards by 87, rightwards by 31
  - G channel: shift downwards by 42, rightwards by 5
- icon.tif:

  - R channel: shift downwards by 89, rightwards by 23
  - G channel: shift downwards by 40, rightwards by 17

- village.tif:
  - R channel: shift downwards by 137, rightwards by 22
  - G channel: shift downwards by 65, rightwards by 12
- self_portrait.tif:
  - R channel: shift downwards by 174, rightwards by 35
  - G channel: shift downwards by 78, rightwards by 28
- harvesters.tif:
  - R channel: shift downwards by 124, rightwards by 13
  - G channel: shift downwards by 59, rightwards by 16
- lady.tif:
  - R channel: shift downwards by 113, rightwards by 11
  - G channel: shift downwards by 47, rightwards by 8
- turkmen.tif:
  - R channel: shift downwards by 115, rightwards by 26
  - G channel: shift downwards by 55, rightwards by 18

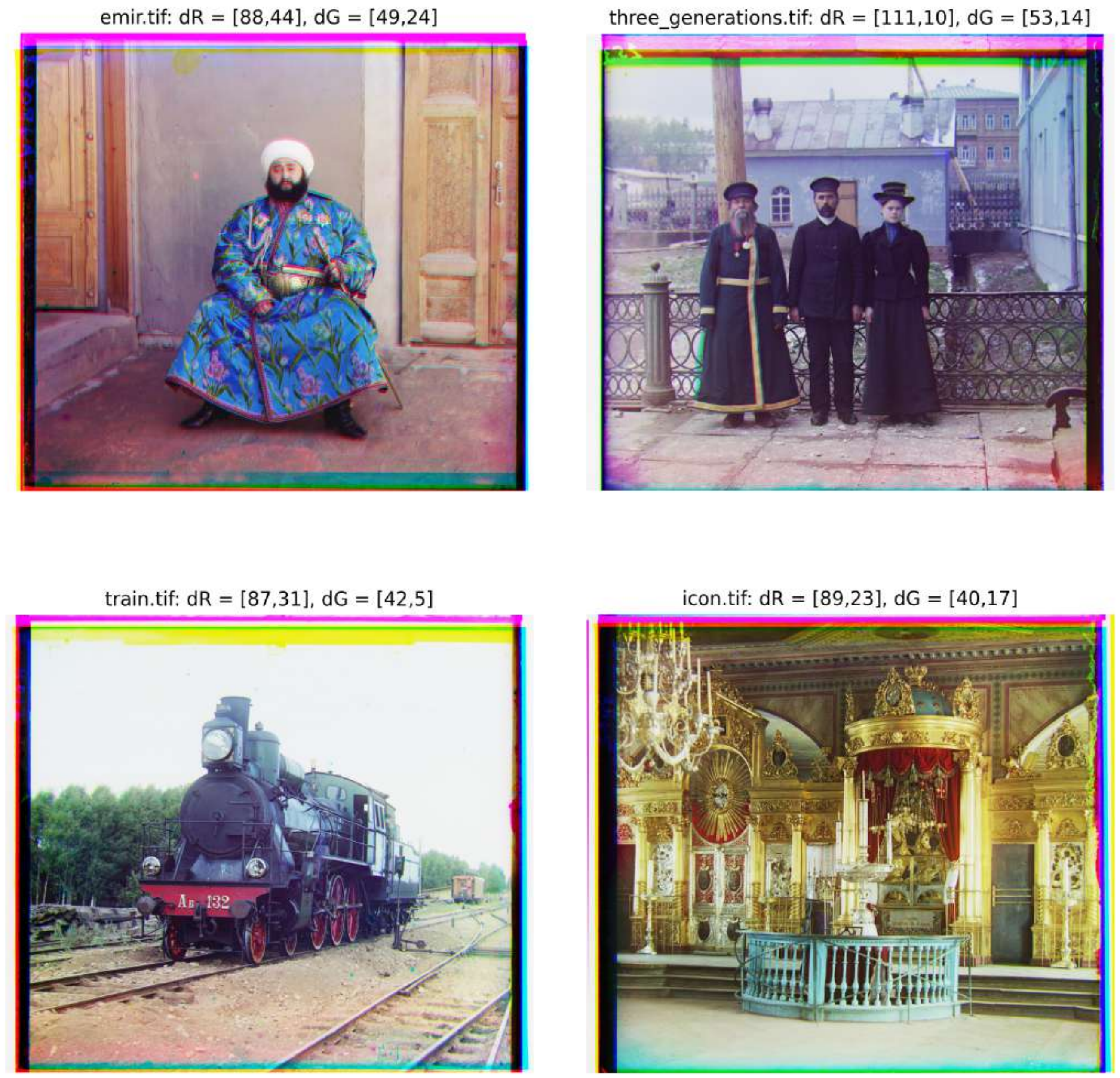
emir.tif: dR = [88,44], dG = [49,24]
three_generations.tif: dR = [111,10], dG = [53,14]
train.tif: dR = [87,31], dG = [42,5]
icon.tif: dR = [89,23], dG = [40,17]

Figure 2: Task 2: Results of Alignment with Larger Images. *Top Left:* emir.tif, *Top Right:* three_generations.tif, *Bottom Left:* train.tif, *Bottom Right:* icon.tif.

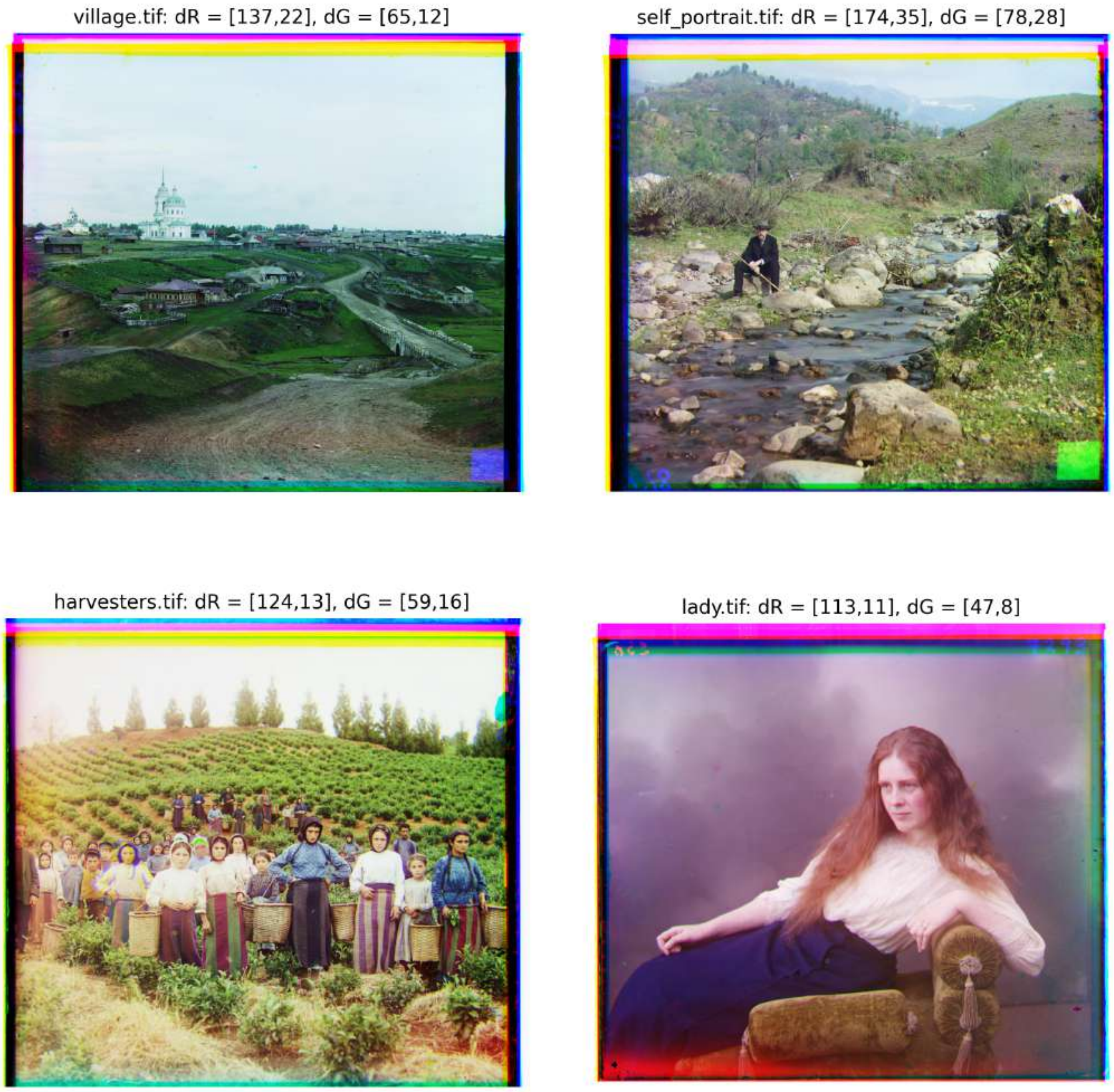
village.tif: dR = [137,22], dG = [65,12]
self_portrait.tif: dR = [174,35], dG = [78,28]
harvesters.tif: dR = [124,13], dG = [59,16]
lady.tif: dR = [113,11], dG = [47,8]

Figure 3: Task 2: Results of Alignment with Larger Images. *Top Left:* village.tif, *Top Right:* self_portrait.tif, *Bottom Left:* harvesters.tif, *Bottom Right:* lady.tif.

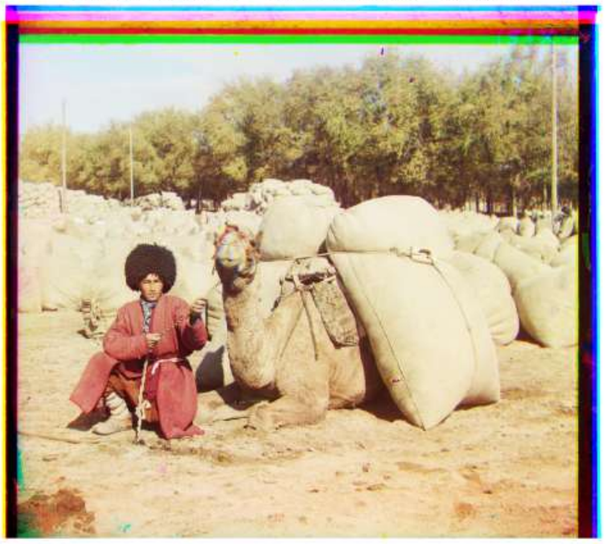

Figure 4: Task 2: Results of Alignment with Larger Images. turkmen.tif.

4 Extra Credits

4.1 Automatic cropping

To perform automatic cropping after the images are aligned, I used two steps:

- Alignment Correction
- White/Black Border Removal

4.1.1 Alignment Correction

When aligning the R channel with the B channel, a circular shift is employed. This involves shifting by d_x in the x-coordinate and d_y in the y-coordinate. Taking d_x as an example, a positive d_x signifies that the image is shifted to the right to align with the B channel. As a result, any pixels that are shifted out of the image frame on one side (for instance, the right side, if $d_x > 0$) re-enter the image frame from the opposite side (in this case, the left side). Consequently, to maintain the integrity of the image and focus on the correctly aligned content, it is necessary to crop out this circularly shifted area.

This same principle is applied when aligning the G channel with the B channel. The valid region is then determined by identifying the intersection between the shifted channels' valid areas.

4.1.2 White/Black Border Removal

After correcting for alignment by removing the invalid shifted part, the function proceeds to eliminate the white and black borders surrounding the image content. Take removing white borders as an example. This is accomplished by identifying pixels that are significantly brighter than a specified percentile threshold; in this case, the percentile is set at 95, which is presumed to indicate the presence of a white border. I inverted the mask so that the area indicates the area that is not white, i.e. the non-white-border area. The function then calculates the median positions of the start and end points for each row and column within the non-border area to determine the cropping boundaries. This process effectively isolates the main content of the image from its surrounding white space. The same procedure is applied to remove the black border, with the percentile set at 5.

4.1.3 Results

Figure 5 shows some sample results of the automatic cropping. We can see that the colored region and also the white and black borders are successfully removed from the original aligned image.

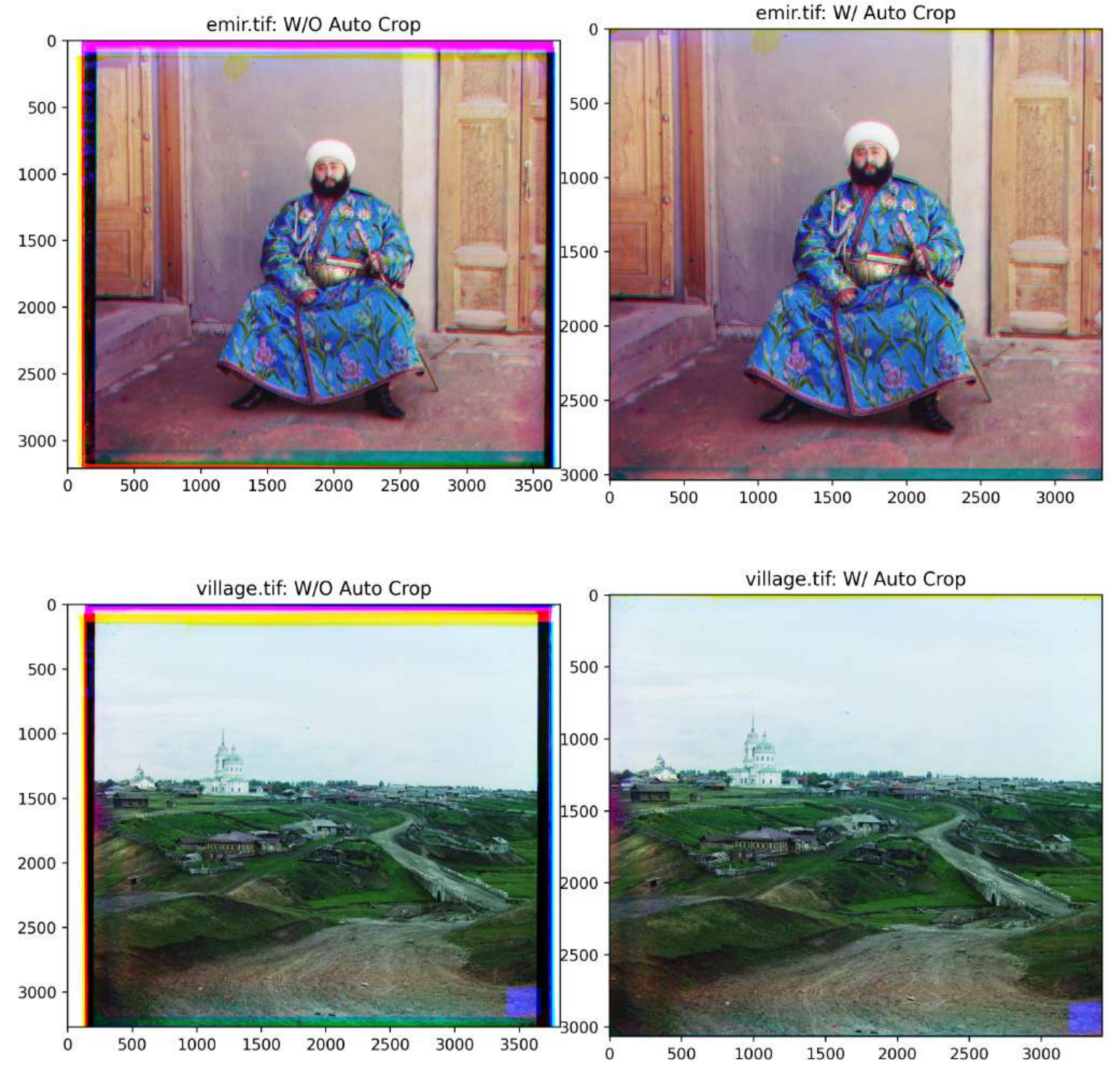

Figure 5: Automatic cropping: Images on the left display the original composition without cropping, and those on the right demonstrate the result after automatic cropping.

4.2 Automatic contrasting

In this section, we present the results of applying automatic contrast versus not applying it. This action is conducted after image alignment. To adjust the contrast automatically, I identify the minimum and maximum pixel values within the image. However, due to the presence of borders that are entirely black or white, I opt for a more refined approach. Rather than using the direct minimum and maximum values, I select the 5th and 95th percentiles as my minimum (v_m) and maximum (v_M) values, respectively. This method ensures a more balanced contrast adjustment by avoiding the extremes of the pixel value range. Therefore, the adjusted intensity of the image is determined as follows:

$$I_{ac} = \frac{I_0 - v_m}{v_M - v_m} \tag{6}$$

Since I use the 5th and 95th percentiles as my minimum and maximum values, I then need to clip the image to the range of 0 to 1.

4.2.1 Results

Figure 6 shows the comparison between images with and without automatic contrast enhancement across several examples. It shows that with automatic contrast, the images appear clearer and more vivid, showcasing sharper details and a broader range of brightness and color. This enhancement brings out the subtle differences in textures and colors.

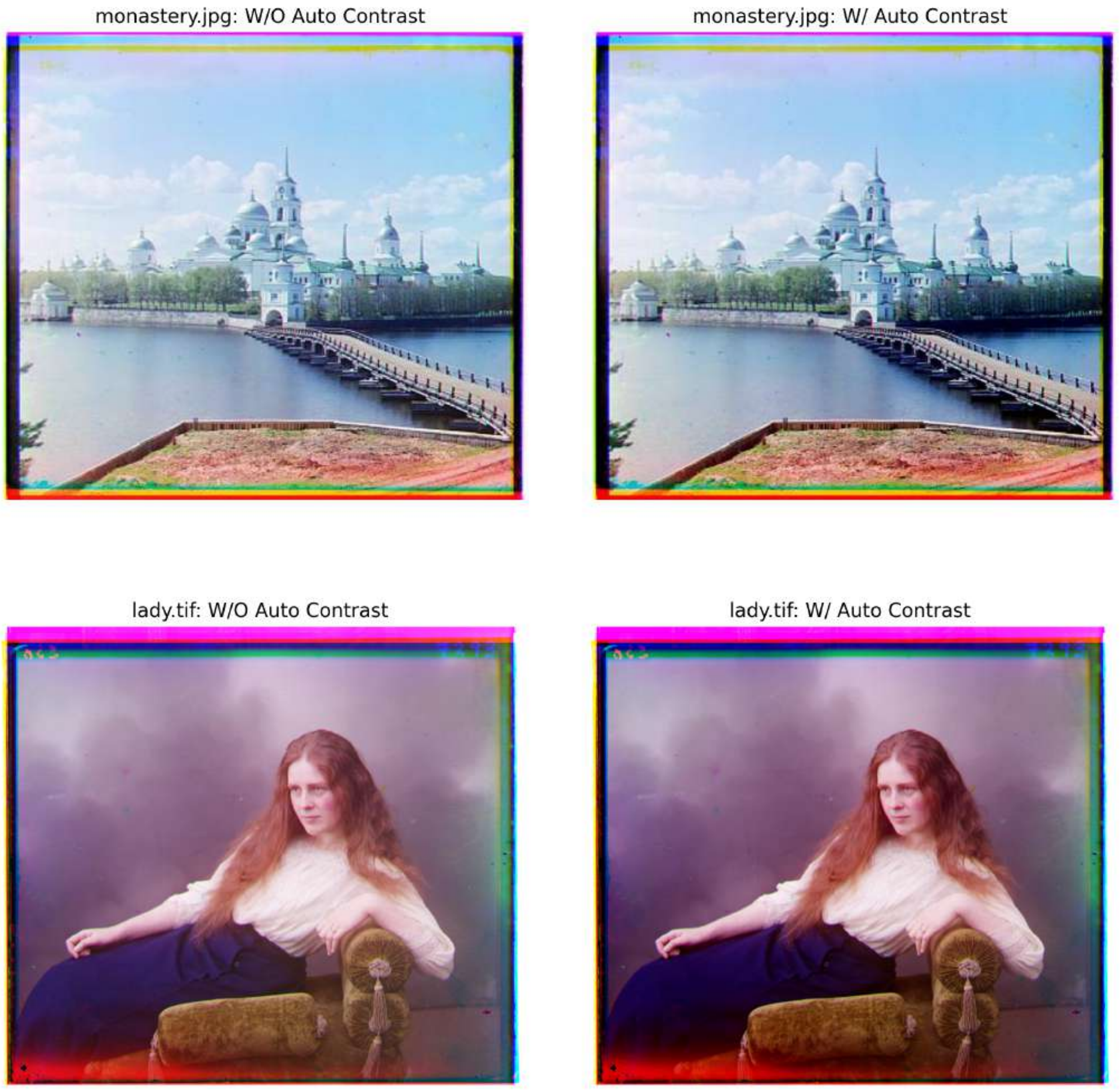
monastery.jpg: W/O Auto Contrast
monastery.jpg: W/ Auto Contrast
lady.tif: W/O Auto Contrast
lady.tif: W/ Auto Contrast

Figure 6: Automatic contrasting: Images on the left display the original composition without contrasting, and those on the right demonstrate the result after automatic contrasting.

4.3 Automatic white balance

In this part, we apply automatic white balance to the aligned image and observe its effect on the original aligned images. To prevent borders from affecting the white balance result, we use the automatic cropping function discussed previously before adjusting the white balance. There are two steps involved in performing automatic white balance:

1. Calculate the illuminant
2. Manipulating the colors to counteract the illuminant

4.3.1 Calculate the illuminant

The assumption underlying this step is that the color of the light source, or the illuminant, uniformly affects all colors within the scene. To compute the illuminant, we calculate the average color of each RGB channel across the image, which we define as the illuminant vector $i = \{i_r, i_g, i_b\}$:

$$i_r = \text{mean}(I_r), \quad i_g = \text{mean}(I_g), \quad i_b = \text{mean}(I_b) \tag{7}$$

4.3.2 Manipulating the colors to counteract the illuminant

To neutralize the color cast, each color channel is adjusted towards a predefined neutral gray value (v_g), which is set to 0.5 in a normalized color range (0 to 1) or 128 in an 8-bit unsigned integer (uint8) representation. The adjustment for each channel is performed as follows, ensuring the illuminant's influence is mitigated:

$$I'_r = I_r \times \frac{v_g}{i_r}, \quad I'_g = I_g \times \frac{v_g}{i_g}, \quad I'_b = I_b \times \frac{v_g}{i_b} \tag{8}$$

The resulting image I' is composed of the adjusted channels:

$$I' = [I'_r, I'_g, I'_b] \tag{9}$$

4.3.3 Results

Figure 7 presents a comparison between images with and without automatic white balance across several examples. It demonstrates that with automatic white balance applied, the images appear closer to their true colors. Specifically, the white areas are nearly identical to true white, whereas in the original images, these areas exhibit a warm and yellowish tint, even when they are supposed to be white.

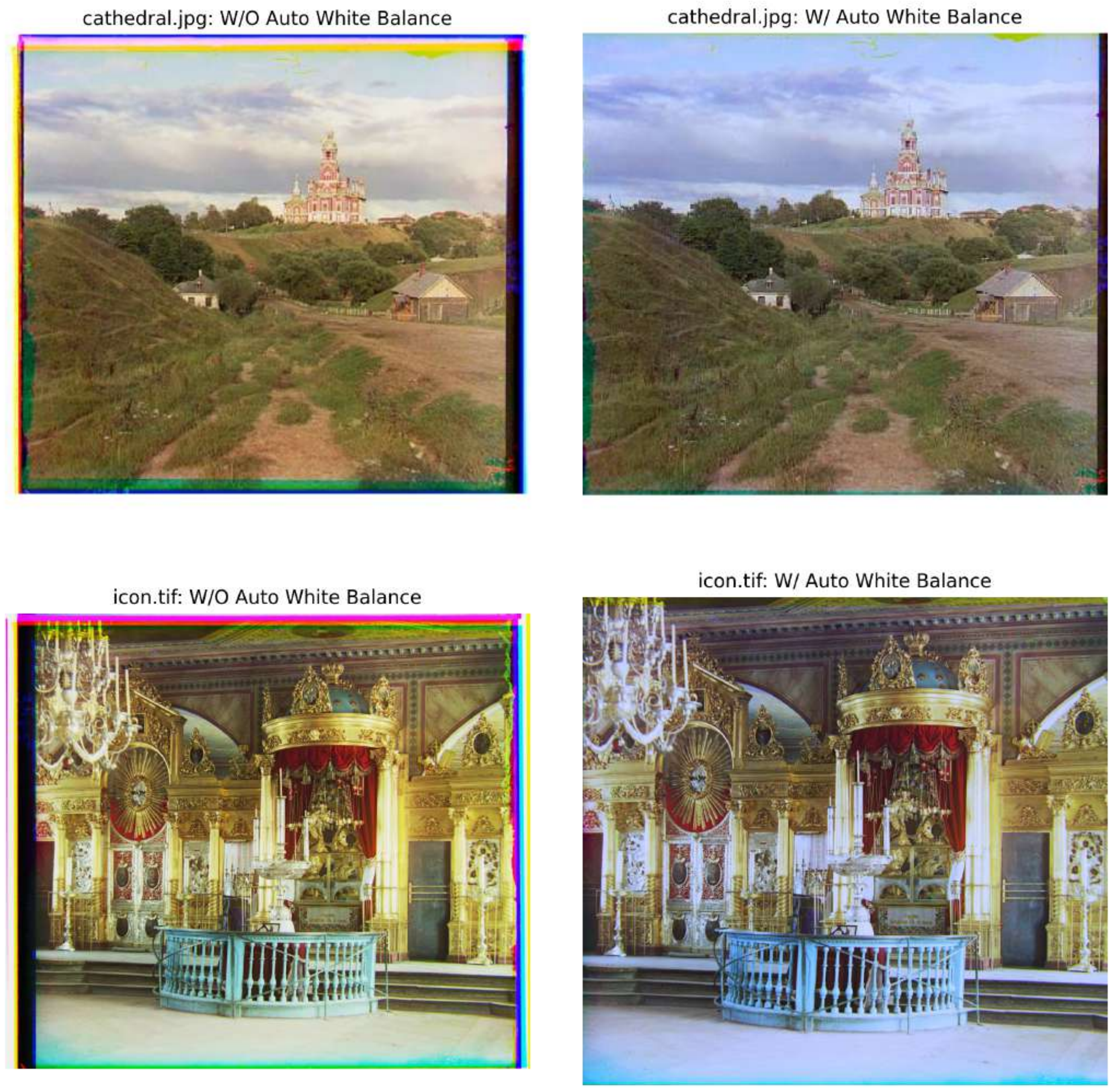
cathedral.jpg: W/O Auto White Balance
cathedral.jpg: W/ Auto White Balance
icon.tif: W/O Auto White Balance
icon.tif: W/ Auto White Balance

Figure 7: Automatic white balance: Images on the left display the original composition without white balance, and those on the right demonstrate the result after white balance.

4.4 Better features

In this section, we aim to utilize a superior feature for aligning images, specifically focusing on edges. To prepare each color channel for edge detection, I initially converted the image from float to uint8 and applied Gaussian Blur with a kernel size of 5x5 to minimize noise, followed by the implementation of the Canny edge detector to identify the edges. These edge images from each channel are then employed to align the images, utilizing the optimal shift method described in Task 2.

The Canny edge detector requires the specification of two thresholds: the lower threshold (t_l) and the upper threshold (t_u). These thresholds are pivotal in determining which gradients are strong enough to be considered for edge formation. The lower threshold (t_l) identifies the minimum gradient intensity necessary for a pixel to be considered a potential edge. Conversely, the upper threshold (t_u) specifies the minimum intensity at which a pixel should be considered part of an edge. Essentially, gradients with intensities above t_u are "sure-edge", while those below t_l are discarded, and gradients between t_l and t_u are classified based on their connectivity to certain edges. Here's how I defined the thresholds:

$$t_l = \max\{0, (1-\sigma) \times v\}, t_u = \min\{255, (1+\sigma) \times v\} \tag{10}$$

, where $\sigma = 0.33$ and v is the median value of the image.

4.4.1 Results

Figure 8 demonstrates the comparison between using the regular RGB feature for alignment and using edges on the picture 'emir.tif'. We can observe that aligning the channels using edges enhances the alignment result.

- emir.tif
  - Regular Feature: dR = [88,44], dG = [49,24]
  - Better Feature: dR = [107,40], dG = [49,24]

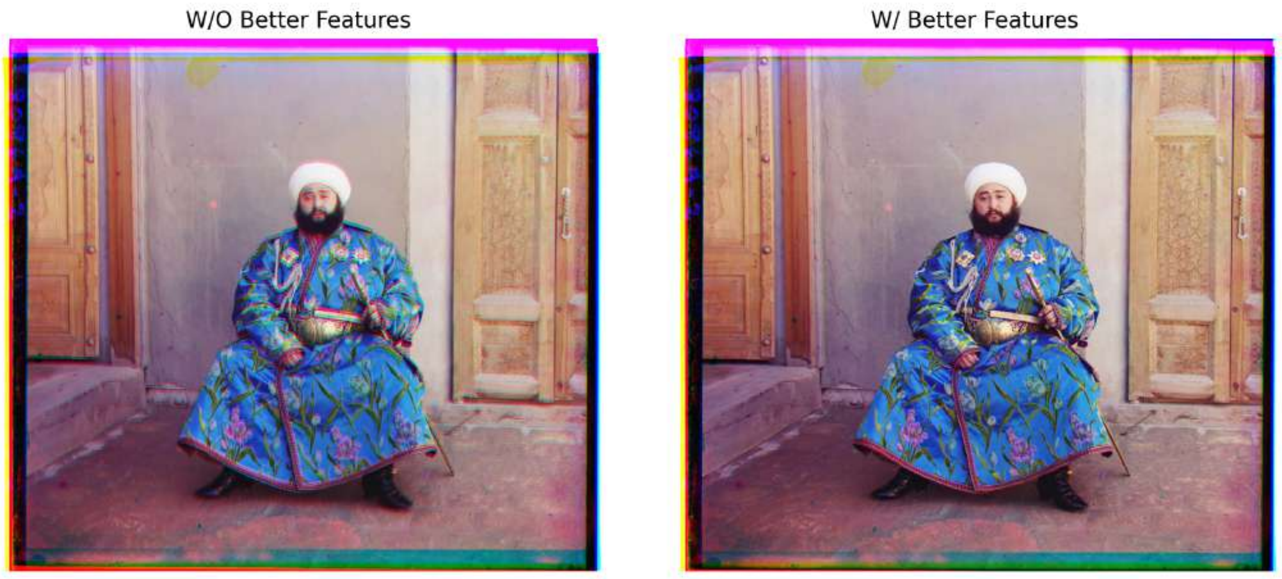

Figure 8: Better feature: Show the result on emir.tif. The image on the left employs default alignment, whereas the image on the right utilizes edge images from each channel to calculate the Sum of Squared Differences (SSD).