Project 1

Background

Sergei Mikhailovich Prokudin-Gorskii was a pioneering photographer who captured images across the Russian Empire using red, green, and blue filters. These glass plate images were later digitized. The project aims to align these channels accurately using single-scale and multi-scale techniques, minimizing visual artifacts. Methods like normalized cross-correlation and edge detection are employed to recreate vibrant, historically significant images.

Method

Single-scale alignmen

General

Use **NCC** instead of L2 to have better performance.

The code is designed to align the RGB channels of an image to produce a well-aligned color image. It begins by using `imread` to load the image, converting it into a floating-point format. After obtaining the image's dimensions, the code splits it into blue, green, and red channels. To enhance the alignment process, `canny_edge_detection` is applied to each channel to extract edges using the Canny algorithm.

Next, the green and red channels are aligned to the blue channel. This is achieved using the `normalized_cross_correlation` function, which measures the similarity between two images, and the `align_channel` function, which finds the best shift for aligning one channel to a reference channel. The alignment is performed by comparing the central regions of the channels and maximizing the normalized cross-correlation score.

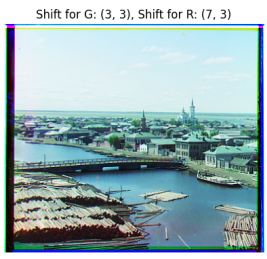
Once the optimal shifts are determined, they are applied to the respective channels. The channels are then cropped to their smallest common size before being combined into the final color image. Then using the function to remove the useless border. Finally, the resulting image is displayed using `matplotlib`, with the title 'Final Aligned Image' and the axis turned off for better visualization.

Details

1. **Import Libraries**:
    - Use `cv2` for image processing.
    - Use `numpy` for array operations.
    - Use `skimage.io` to read images.
    - Use `matplotlib.pyplot` to display images.
2. **Border Removal**: The `remove_border` function detects and removes the borders from the image by applying adaptive thresholding and contour detection, ensuring that only the central content remains for accurate processing.

3. **Edge Detection**: The `canny_edge_detection` function uses the Canny algorithm to extract edge features from the image for better alignment.
4. **Normalized Cross-Correlation (NCC)**: The `normalized_cross_correlation` function calculates the similarity between the central regions of two images to assess alignment quality.
5. **Channel Alignment**: The `align_channel` function searches for the best alignment position within a specified maximum offset range. Only the central region of the image is used for alignment to reduce edge alignment errors.
6. **Apply Offset**: The `apply_shift` function translates the channel according to the calculated offset.
7. **Image Reading and Processing**: Read the image and convert it to float format. Split the image into red, green, and blue channels.
8. **Alignment Process**: Align the green and red channels with the blue channel respectively. Print the alignment offsets.
9. **Compose and Display Image**: Combine the aligned channels into a color image. Then remove the border. Use `matplotlib` to display the result.

Result

Multi-scale alignment

General

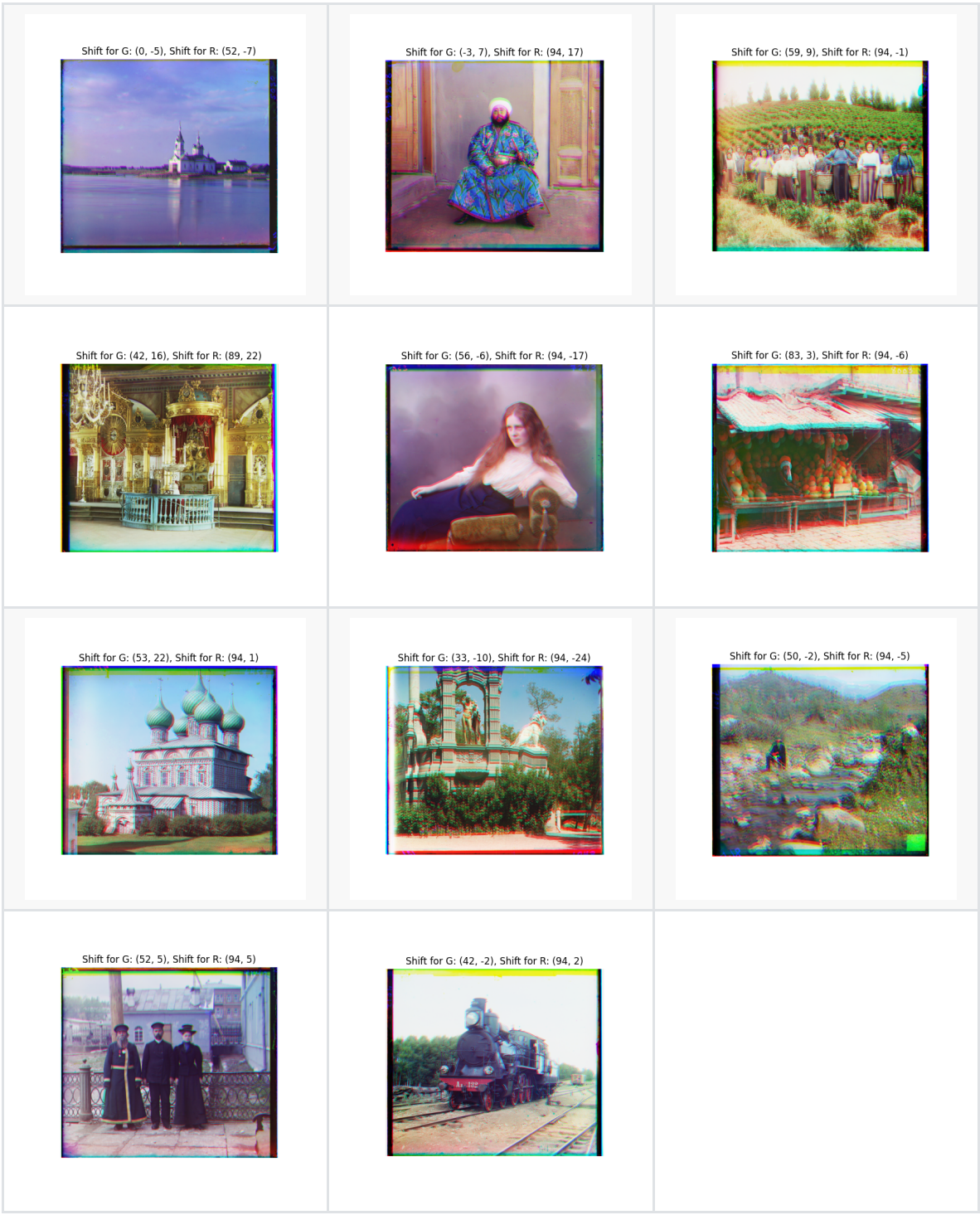
Use **pyramid** to handle `.tif` pictures.

The code aligns RGB channels using a pyramid method and removes borders to create a well-aligned color image. It starts by loading the image, converting it to a floating-point format, and splitting it into blue, green, and red channels. Each channel is preprocessed with Gaussian blur to reduce noise. The green and red channels are aligned to the blue channel using a multi-level pyramid approach. This involves measuring similarity with `normalized_cross_correlation` and finding optimal shifts with `align_channel`. The shifts are applied using `apply_shift`, and channels are cropped to the smallest common size. The `remove_border` function eliminates borders using adaptive thresholding. Finally, the aligned channels are combined into the final RGB image, displayed with `matplotlib`.

Details

1. **Import Libraries**:
    - Use `cv2` for advanced image processing operations, like contour detection and thresholding.
    - Use `numpy` for handling matrix and array manipulations.
    - Use `skimage.io` to read and load images in different formats.
    - Use `matplotlib.pyplot` to display and visualize the final image.
2. **Border Removal**: The `remove_border` function detects and removes the borders from the image by applying adaptive thresholding and contour detection, ensuring that only the central content remains for accurate processing.
3. **Channel Preprocessing**: The `preprocess_channel` function applies Gaussian blur to each channel of the image. This step smooths the image and reduces noise, improving the accuracy of the subsequent alignment process.
4. **Normalized Cross-Correlation (NCC)**: The `normalized_cross_correlation` function computes the similarity between two image regions, helping to determine the best alignment by maximizing the correlation value between channels.
5. **Channel Alignment**: The `align_channel` function searches for the optimal shift for aligning an image channel with the reference channel. It performs this by rolling (shifting) the channel over a defined pixel range and calculating the NCC for each possible shift.
6. **Multiscale Pyramid Alignment**: The `pyramid_align` function enhances the alignment by using an image pyramid. It aligns channels at multiple scales, first aligning smaller, downscaled images for coarse adjustment, and then refining the alignment at full resolution, improving precision and reducing computational load.
7. **Apply Offset**: The `apply_shift` function takes the computed shifts (offsets) and applies them to the corresponding image channel, effectively translating the image to achieve the best alignment.
8. **Image Reading and Preprocessing**: The image is read and converted to floating-point format, and the RGB channels are separated. Each channel is preprocessed (Gaussian blur) before being aligned.
9. **Alignment Process**: Both the green and red channels are aligned to the blue channel using the multiscale pyramid alignment method. The calculated offsets for both the green and red channels are printed to verify the shifts applied.
10. **Compose and Display Image**: After aligning and cropping the channels to the smallest common size, they are stacked back together into a composite RGB image. The final image is displayed using `matplotlib`, and the borders are removed to clean up the output.

Result



Bells and Whistles

Remove border

The process begins by converting the input image from floating-point to 8-bit format, scaling pixel values from `[0, 1]` to `[0, 255]`. The blue channel is extracted, and adaptive thresholding is applied to create a binary image, segmenting pixels into black and white using a Gaussian-weighted method. The binary image is then inverted, making the background black and the foreground white. A 5x5 dilation kernel enhances the foreground, highlighting the border. Contours are detected, with the largest assumed to be the border. The `boundingRect` function calculates the minimum bounding rectangle, and the image is cropped to remove the border, returning the cropped result.

effect comparison :

Before	After
Shift for G: (5, 2), Shift for R: (8, 3) 	Shift for G: (5, 2), Shift for R: (8, 3) 
Shift for G: (56, -6), Shift for R: (94, -17) 	Shift for G: (56, -6), Shift for R: (94, -17) 
Shift for G: (-3, 7), Shift for R: (94, 17) 	Shift for G: (-3, 7), Shift for R: (94, 17) 

Use cv2

Using functions of `cv2` library, it works pretty well and fast.

The image is split into blue, green, and red channels, each normalized to `[0, 1]`. An `align` function aligns these channels to a reference by normalizing them to `[0, 255]`, applying Gaussian blur, and using Canny edge detection. Phase correlation calculates the optimal shift, and `cv2.warpAffine` applies it. After alignment, the channels are combined into an RGB image. The `remove_border` function scales the image to `[0, 255]`, extracts the blue channel, and uses adaptive thresholding to create a binary image. It inverts and dilates the binary image, finds the largest contour, computes its bounding box, and crops the image to remove the border.

Before	After