CS543 Assignment 2

Your Name: Yiyang Liu
Your NetId: yiyang34

Part 1 Fourier-based Alignment:

You will provide the following for each of the six low-resolution and three high-resolution images:

- Final aligned output image
- Displacements for color channels
- Inverse Fourier transform output visualization for ***both*** channel alignments ***without*** preprocessing
- Inverse Fourier transform output visualization for ***both*** channel alignments ***with*** any sharpening or filter-based preprocessing you applied to color channels

You will provide the following as further discussion overall:

- Discussion of any preprocessing you used on the color channels to improve alignment and how it changed the outputs
- Measurement of Fourier-based alignment runtime for high-resolution images (you can use the python time module again). How does the runtime of the Fourier-based alignment compare to the basic and multiscale alignment you used in Assignment 1?

A: Channel Offsets

Replace <C1>, <C2>, <C3> appropriately with B, G, R depending on which you use as the base channel. Provide offsets in the **original image coordinates** (after the image has been divided into three equal parts corresponding to each channel) and be sure to account for any cropping or resizing you performed.

Low-resolution images (using channel <B> as base channel):

Image	<G> (h,w) offset	<R> (h,w) offset
00125v.jpg	(-3, 2)	(-6, 1)
00149v.jpg	(-4, 2)	(-7, 2)
00153v.jpg	(-1, 3)	(-2, 4)
00351v.jpg	(-4, 0)	(-3, 1)
00398v.jpg	(-3, 3)	(-5, 4)
01112v.jpg	(-8, 0)	(-11, 1)

High-resolution images (using channel <B> as base channel):

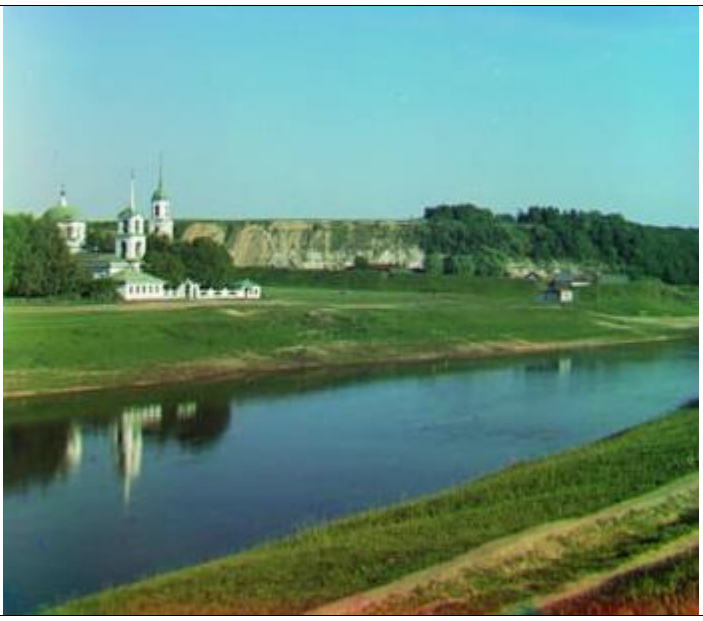
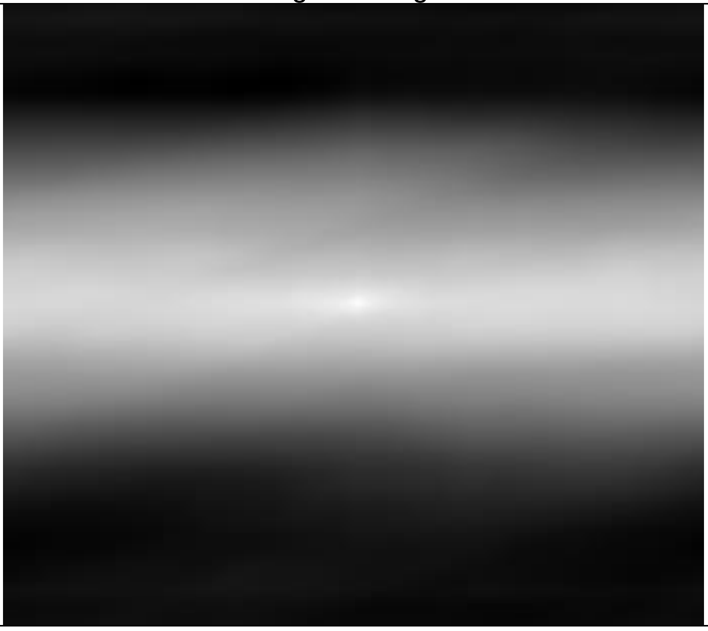
Image	<G> (h,w) offset	<R> (h,w) offset
01047u.tif	(-29, 23)	(-22, 39)
01657u.tif	(5, 23)	(-32, 33)
01861a.tif	(-26, 36)	(-40, 47)

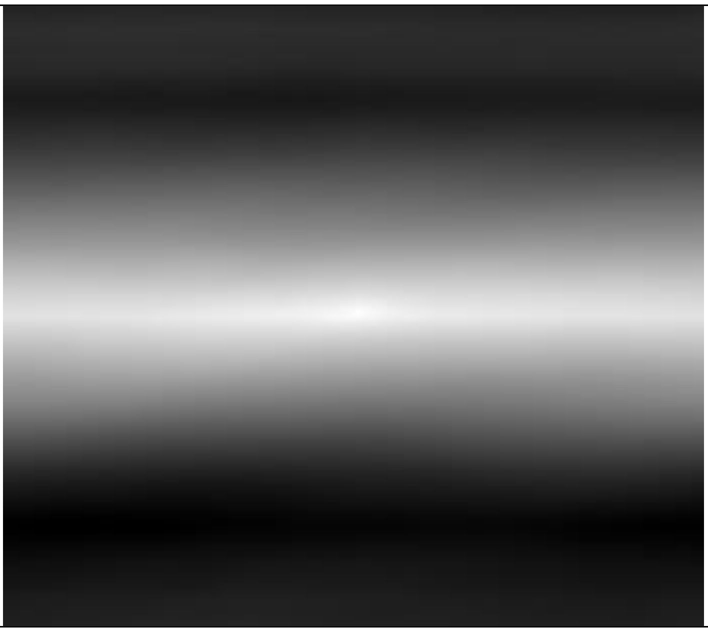
It should be noted that I performed an ***arbitrary image cropping*** to segment the original images into three distinct channels.

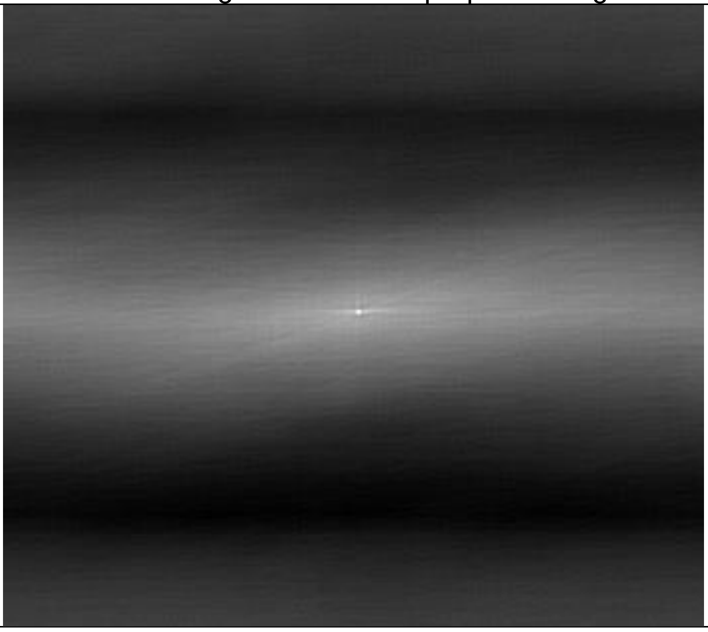
B: Output Visualizations

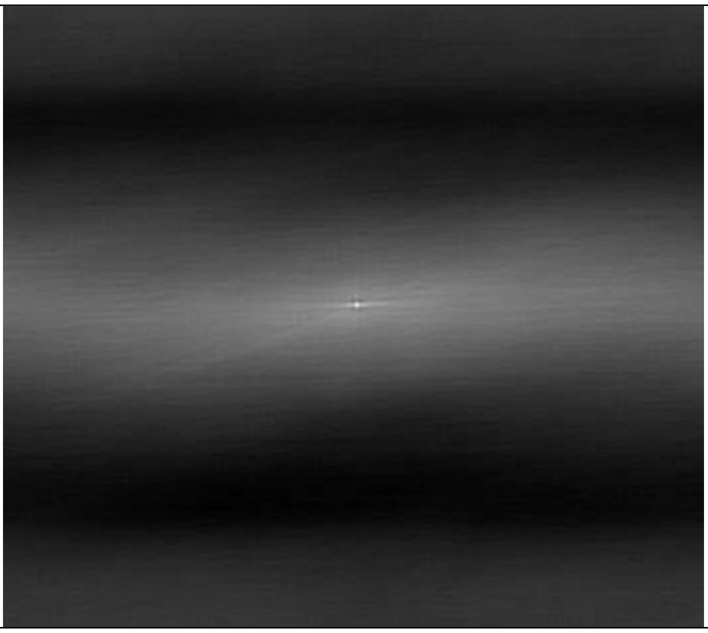
For each image, insert 5 outputs total (aligned image + 4 inverse Fourier transform visualizations) as described above. When you insert these outputs be sure to clearly label the inverse Fourier transform visualizations (e.g. “G to B alignment without preprocessing”).

00125v.jpg

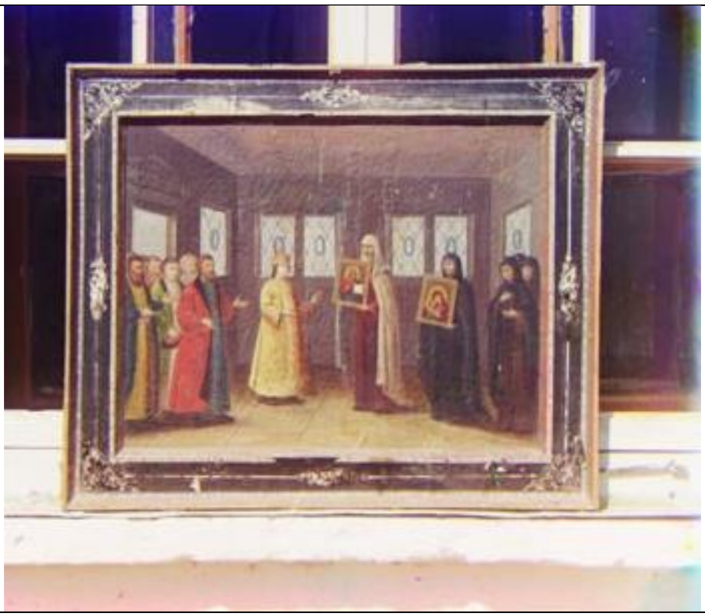
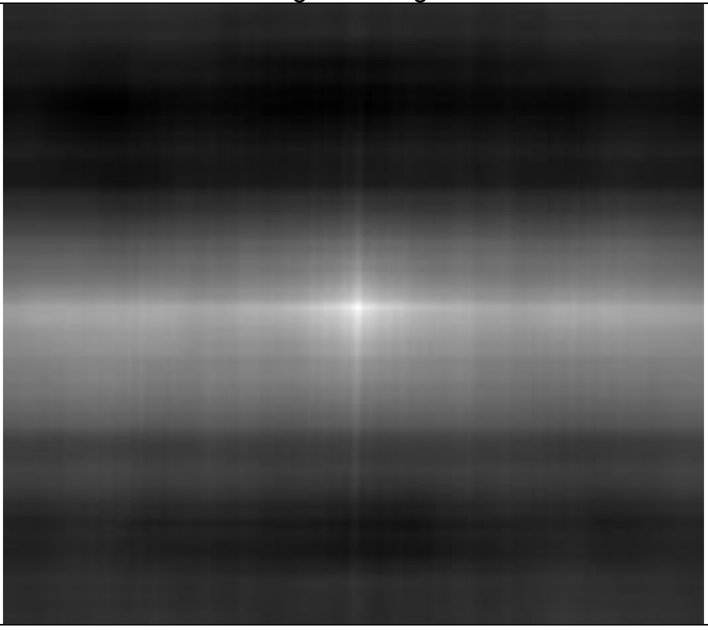

aligned image

G to B alignment without preprocessing

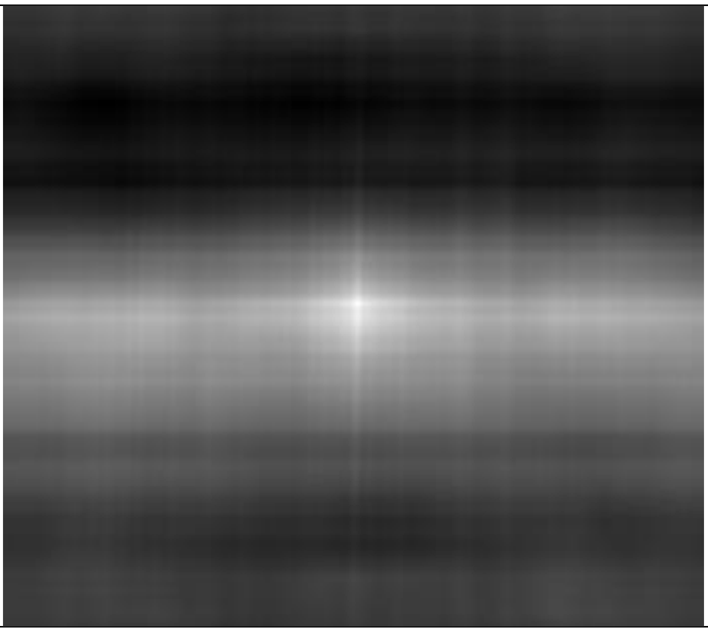

R to B alignment without preprocessing

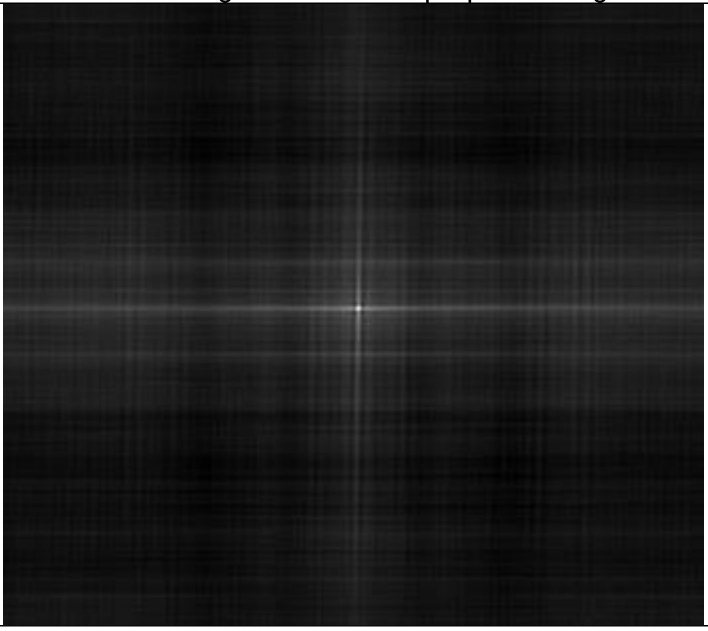

G to B alignment without preprocessing

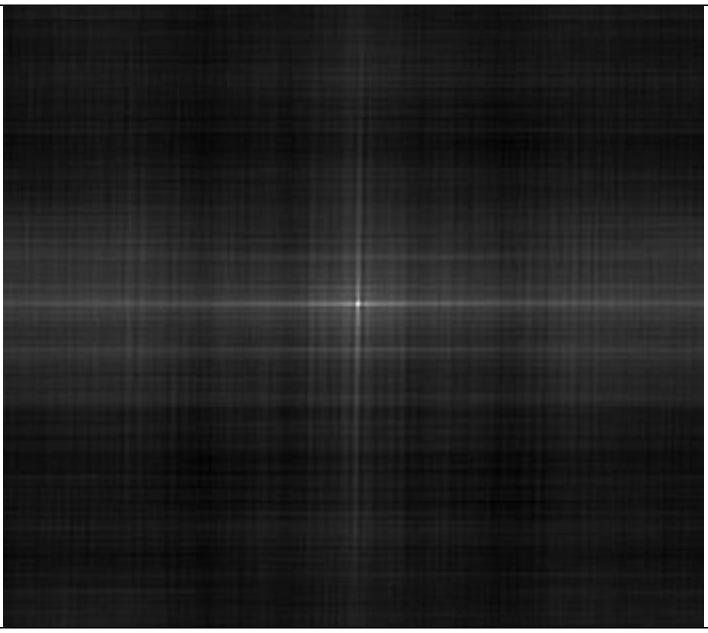

R to B alignment without preprocessing

00149v.jpg


aligned image

G to B alignment without preprocessing

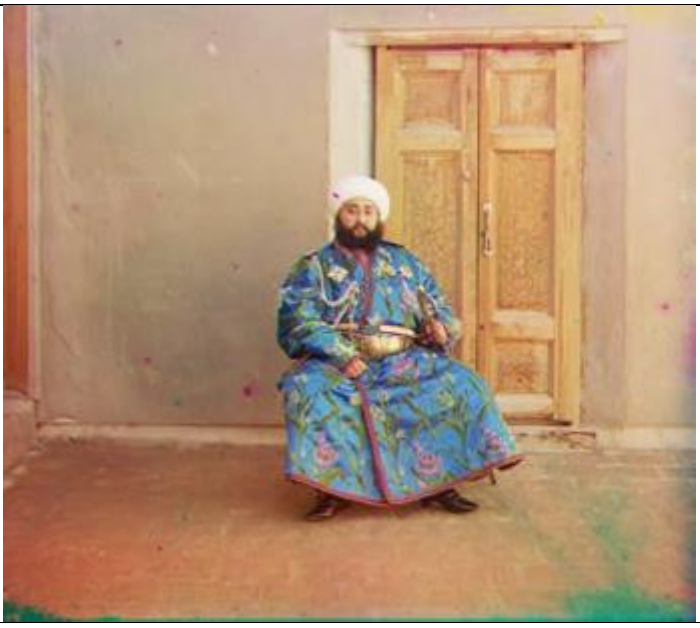
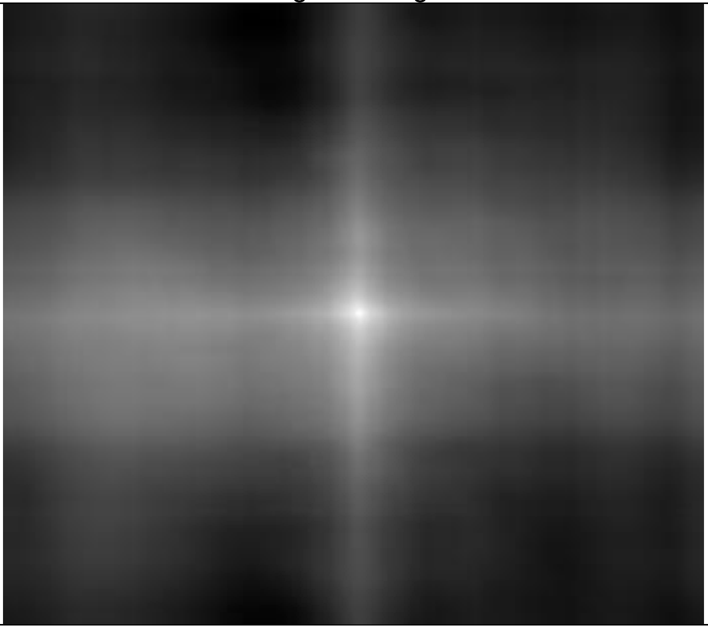

R to B alignment without preprocessing

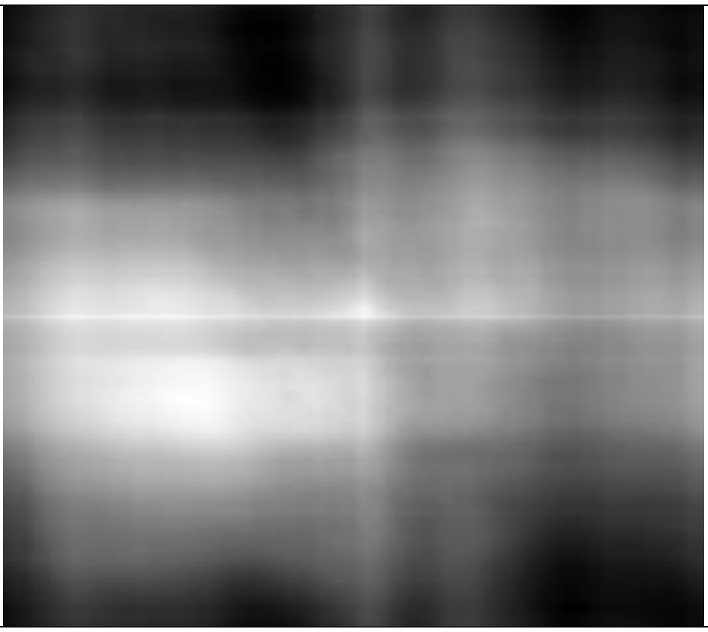

G to B alignment without preprocessing

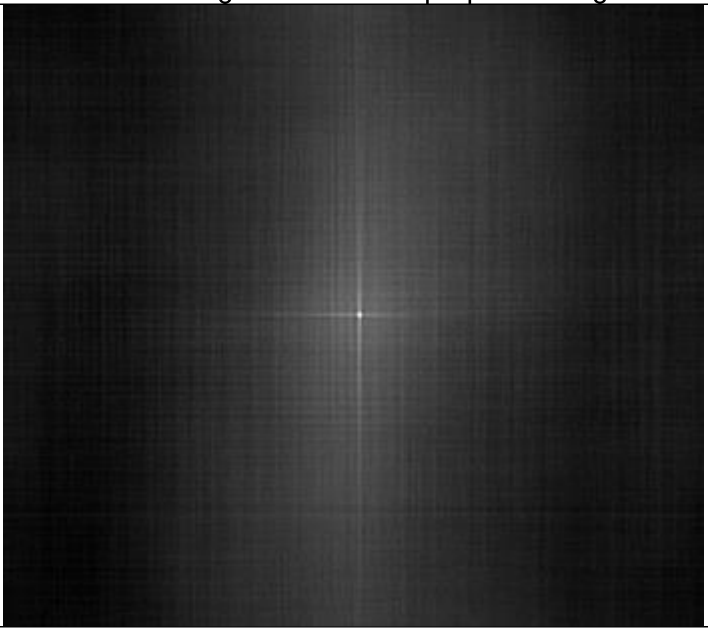


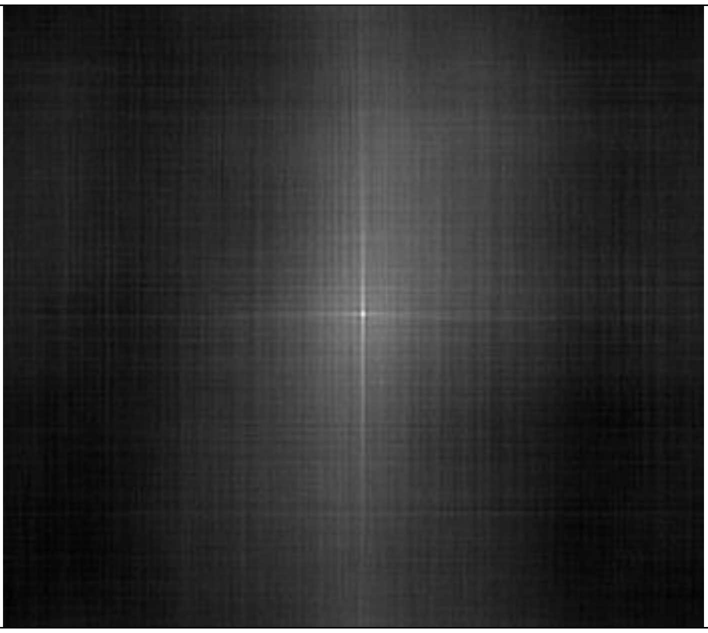
R to B alignment without preprocessing

00153v.jpg

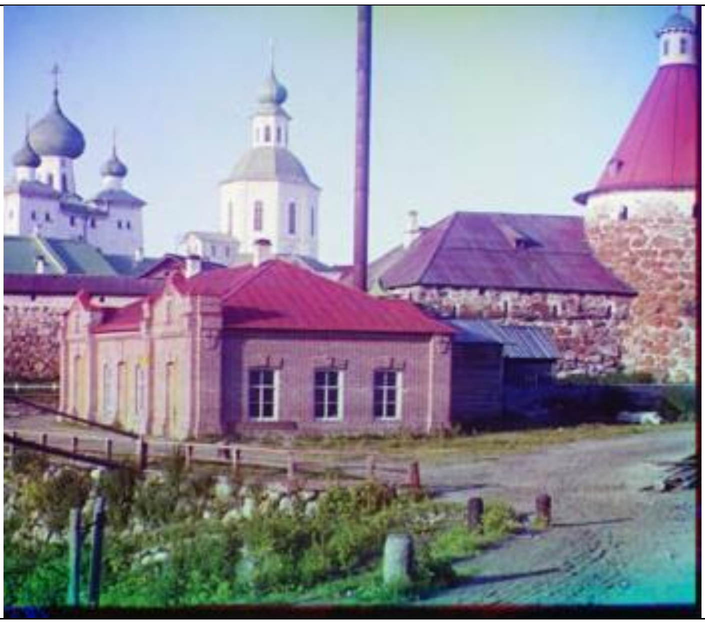

aligned image

G to B alignment without preprocessing

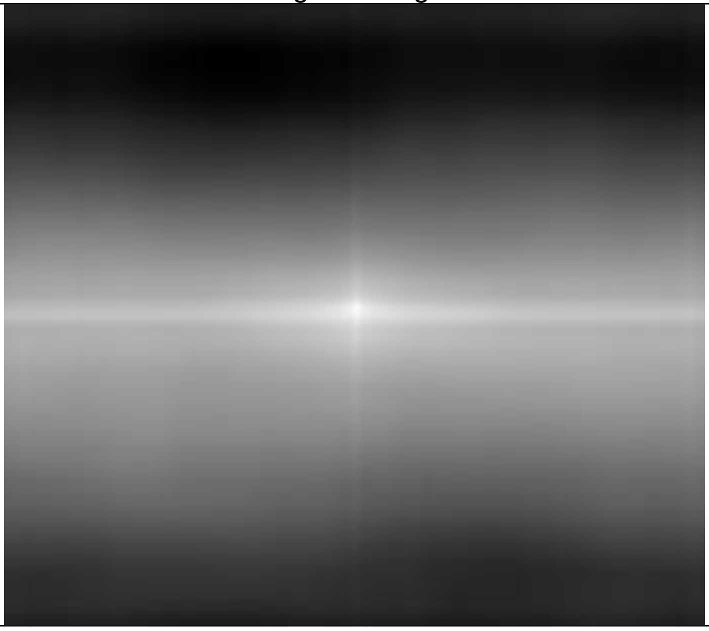

R to B alignment without preprocessing

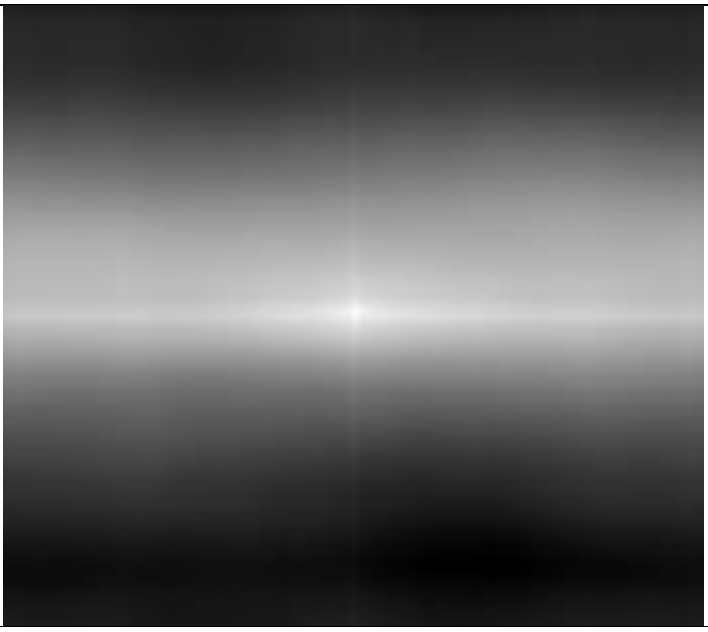
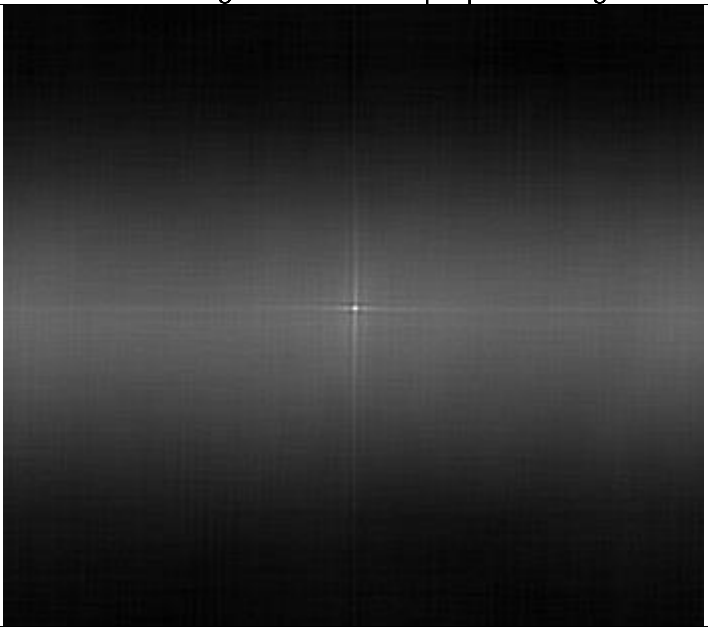

G to B alignment without preprocessing

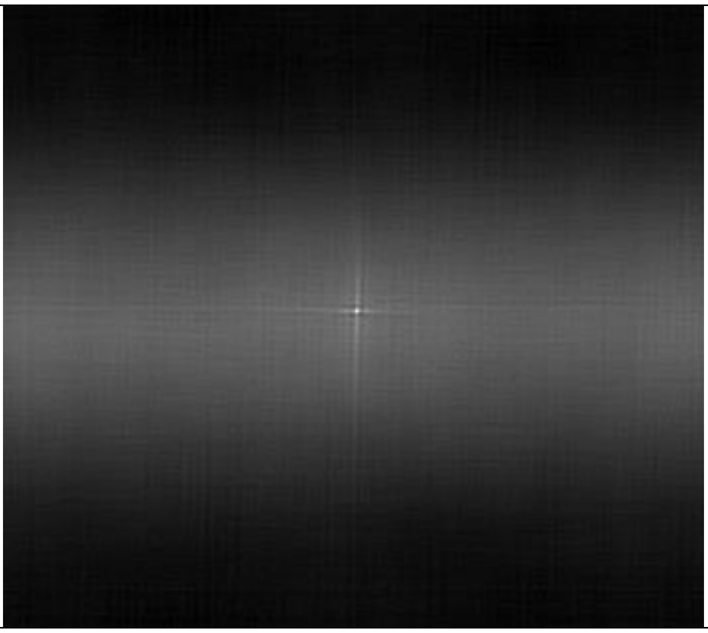

R to B alignment without preprocessing

00351v.jpg


aligned image

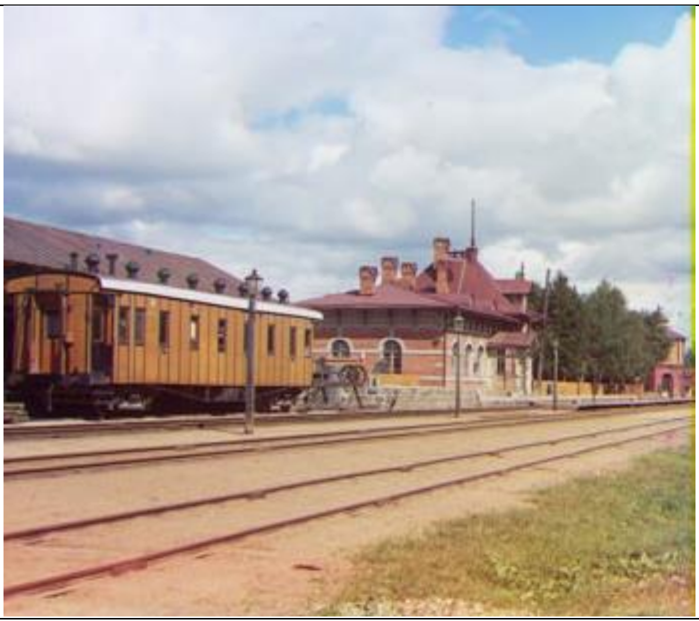
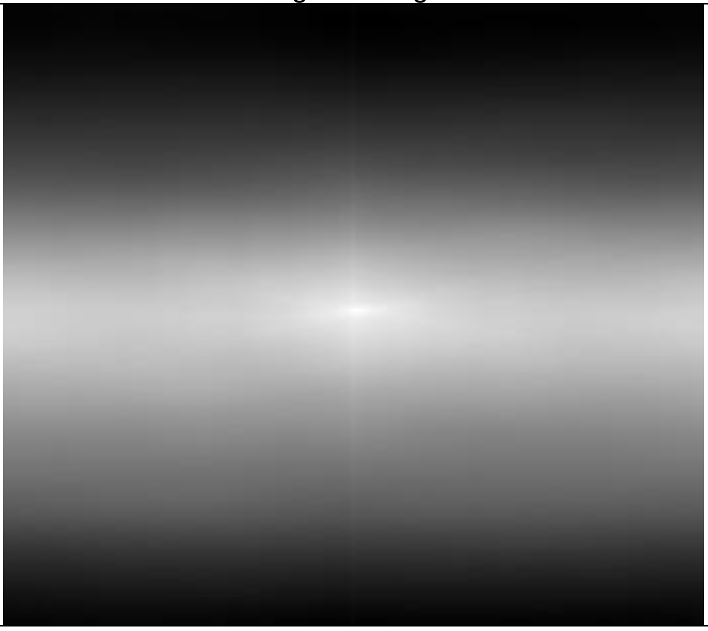

G to B alignment without preprocessing

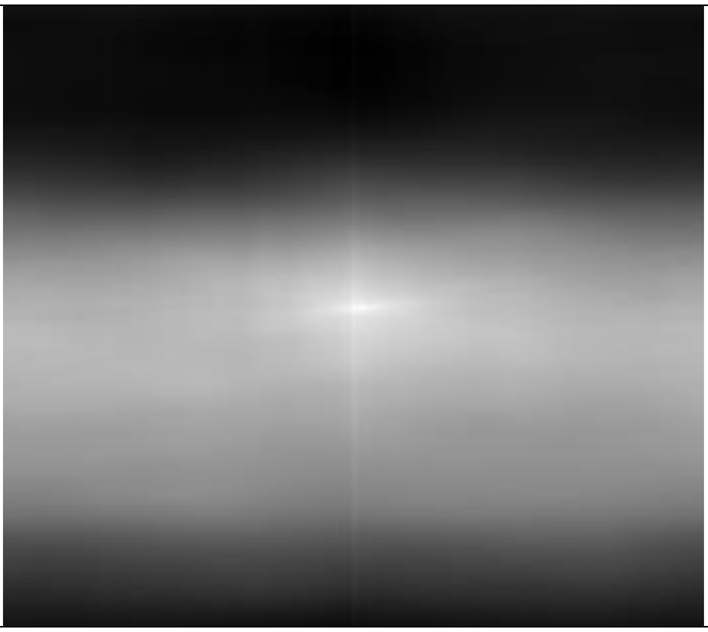

R to B alignment without preprocessing

G to B alignment without preprocessing

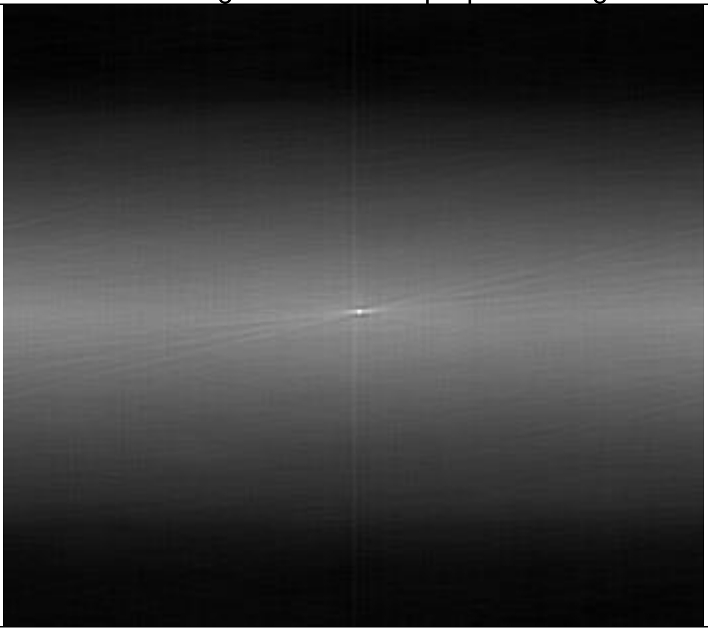


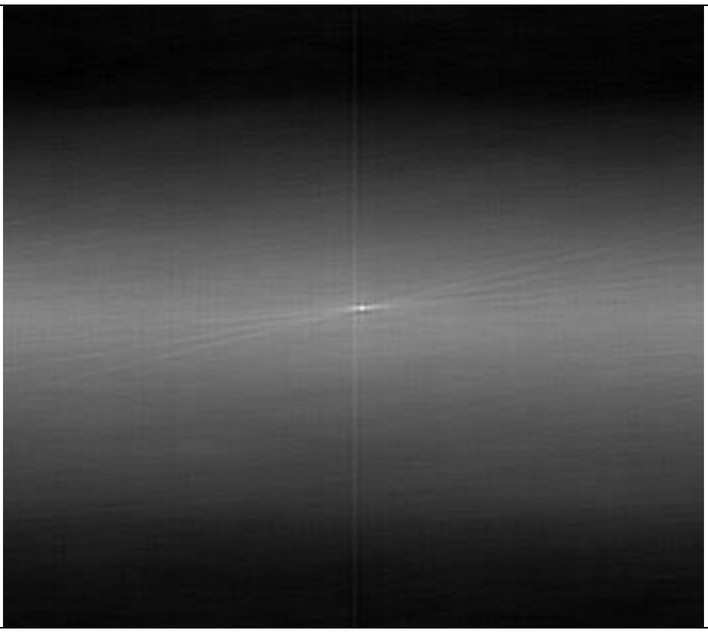
R to B alignment without preprocessing

00398v.jpg

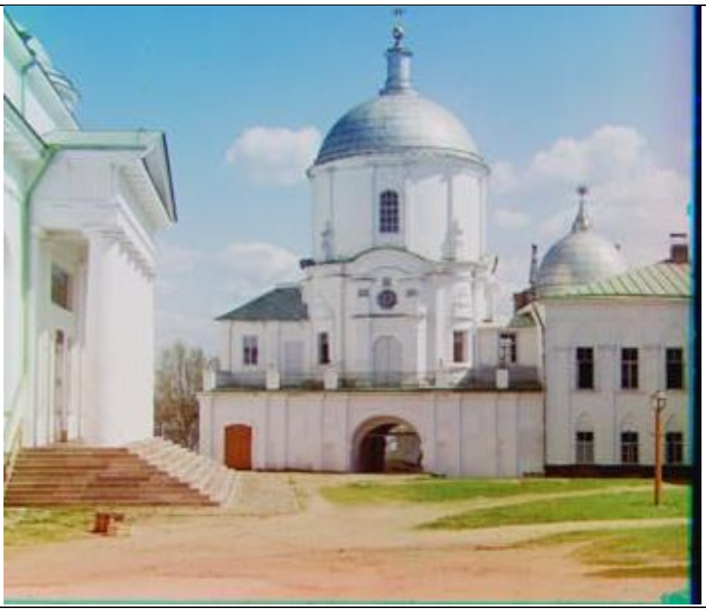
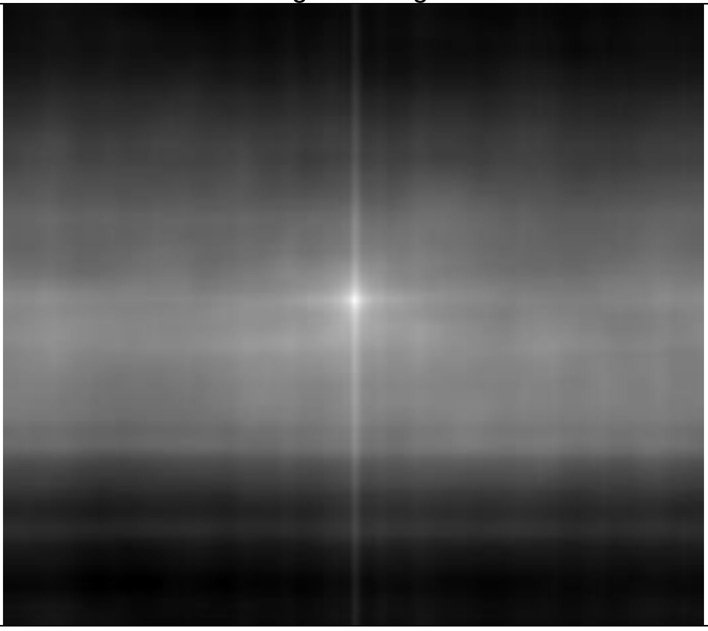

aligned image

G to B alignment without preprocessing

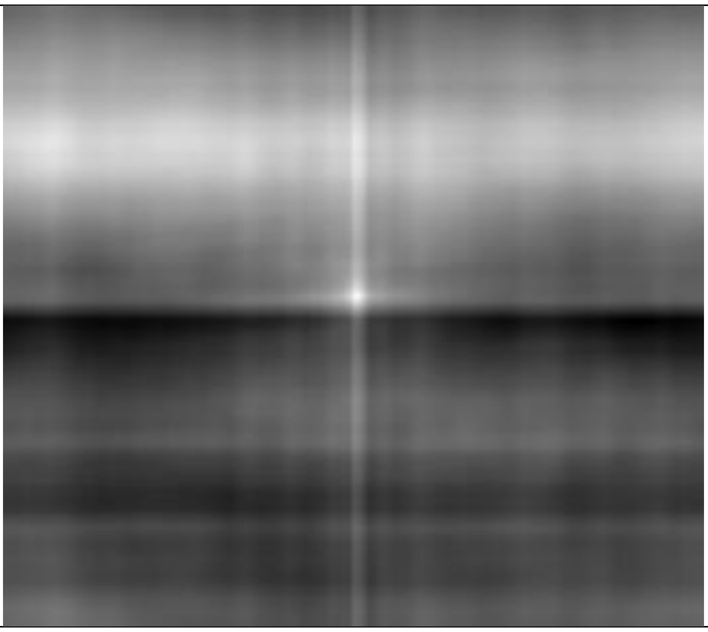
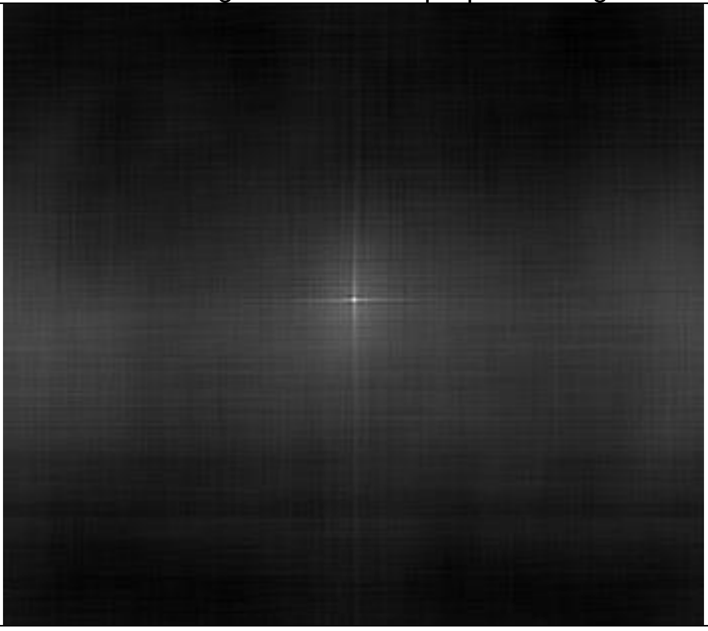

R to B alignment without preprocessing

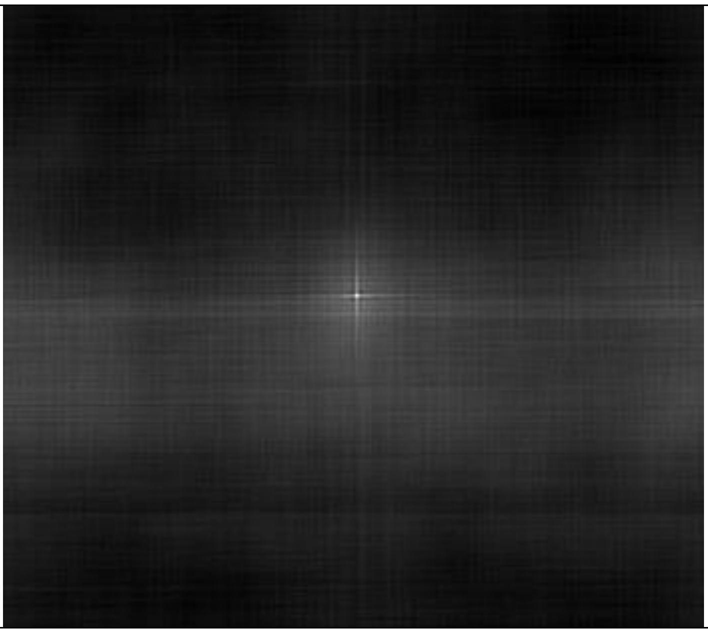

G to B alignment without preprocessing


R to B alignment without preprocessing

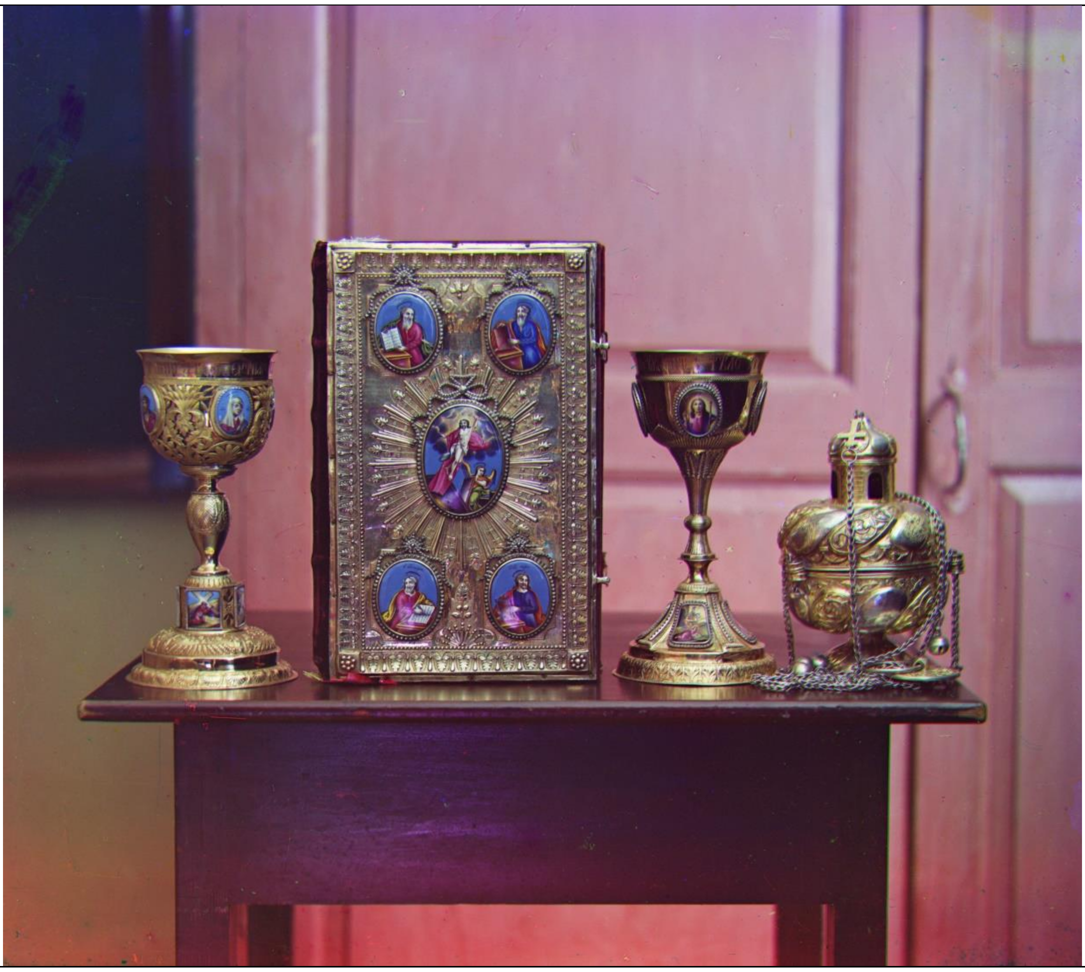
01112v.jpg

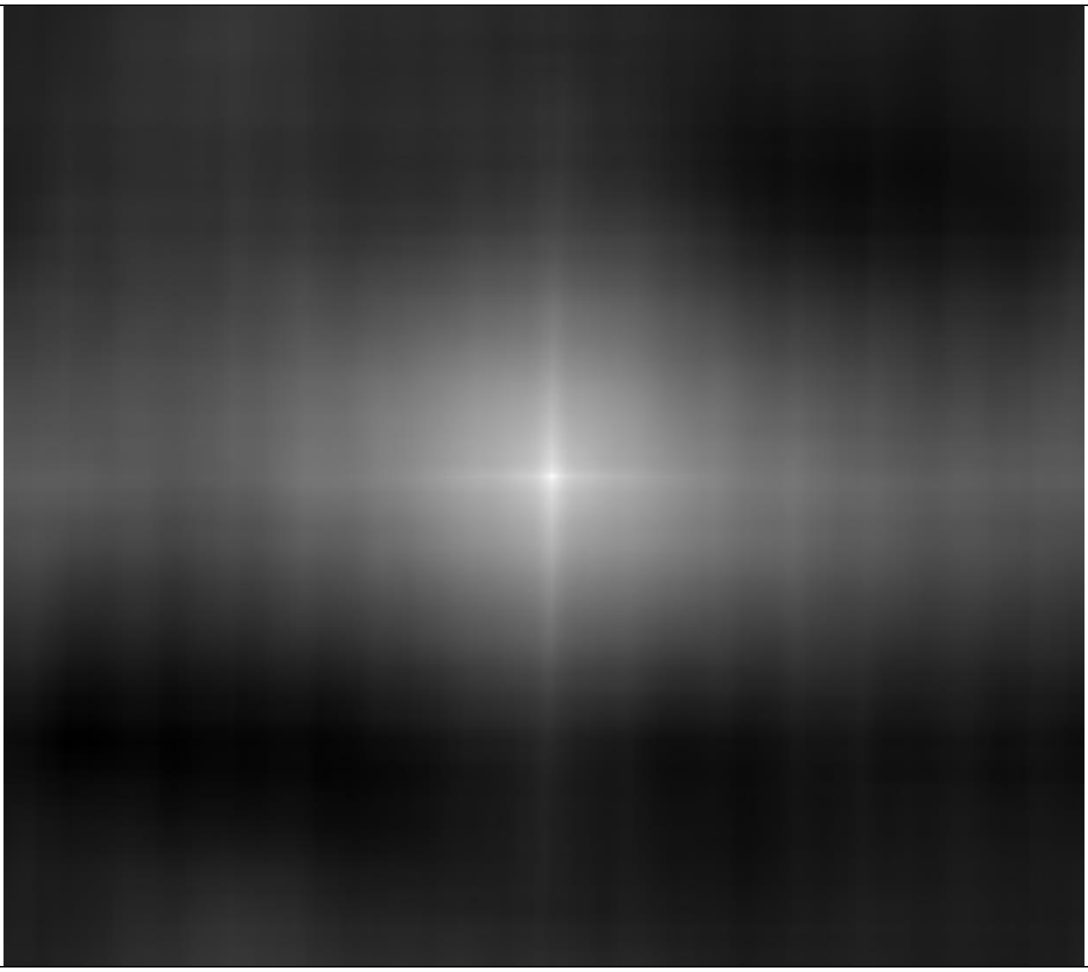

aligned image

G to B alignment without preprocessing

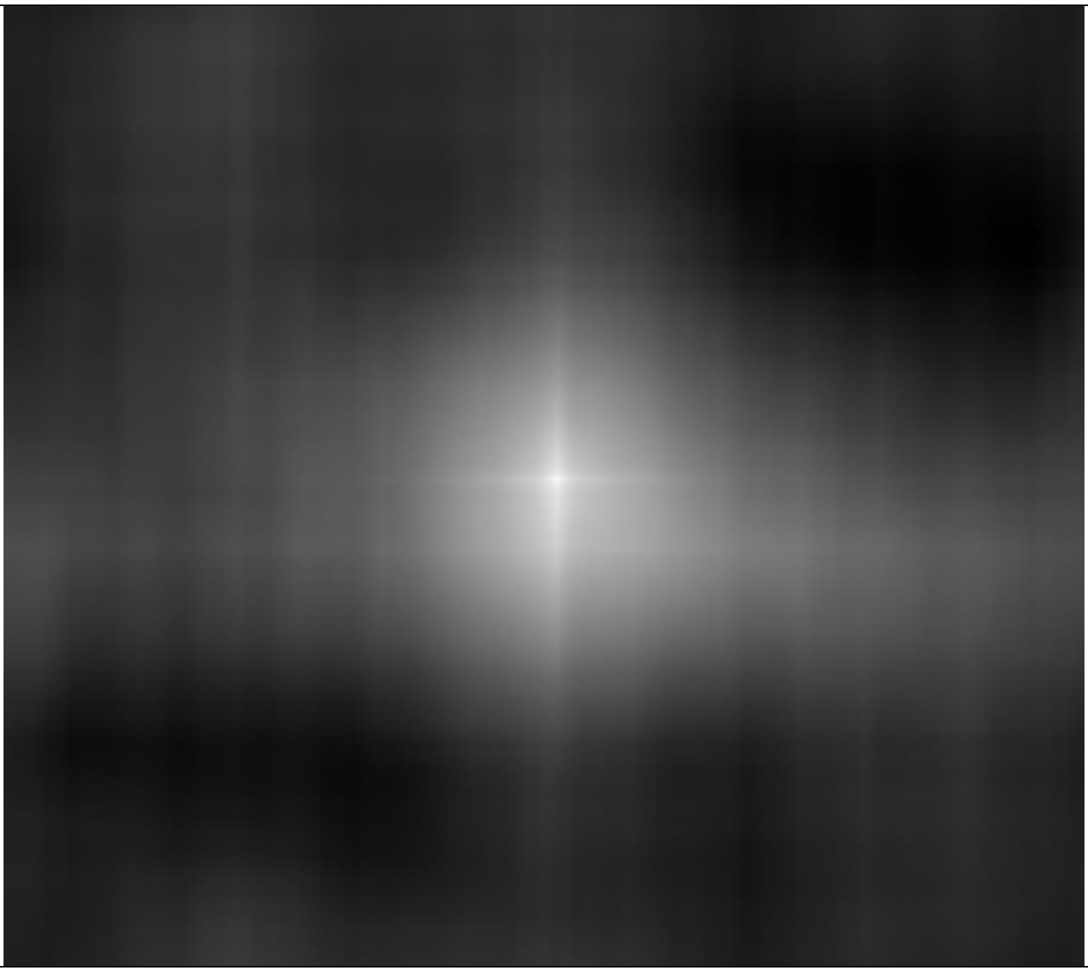

R to B alignment without preprocessing

G to B alignment without preprocessing

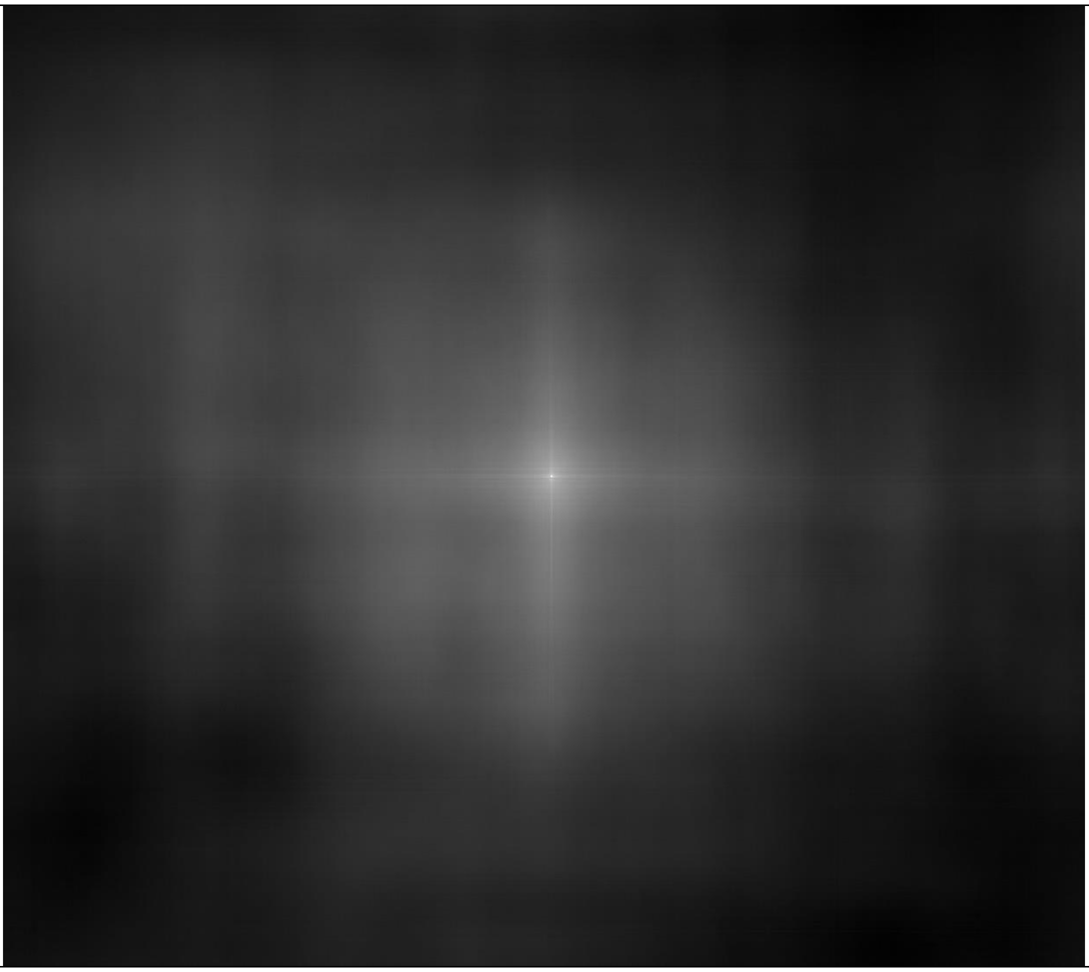

R to B alignment without preprocessing

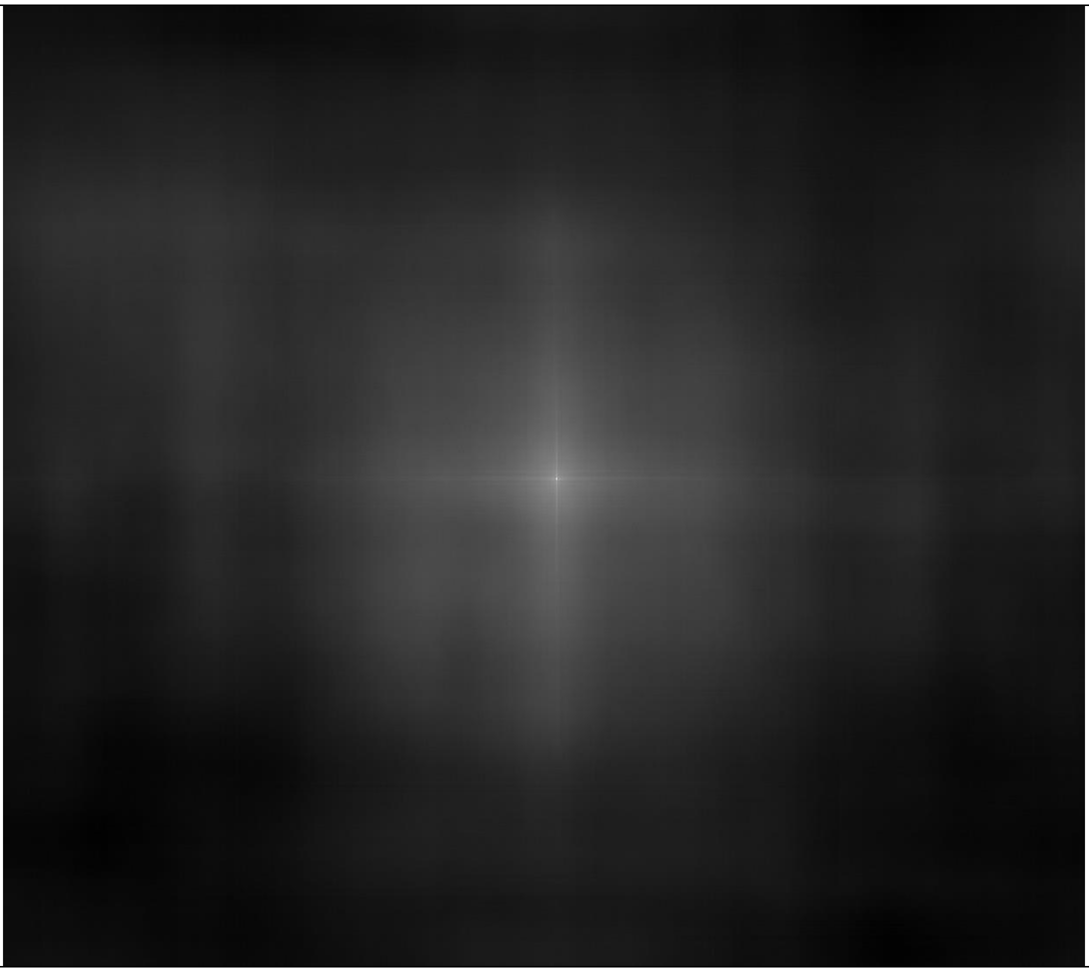
01047u.tif


aligned image


G to B alignment without preprocessing

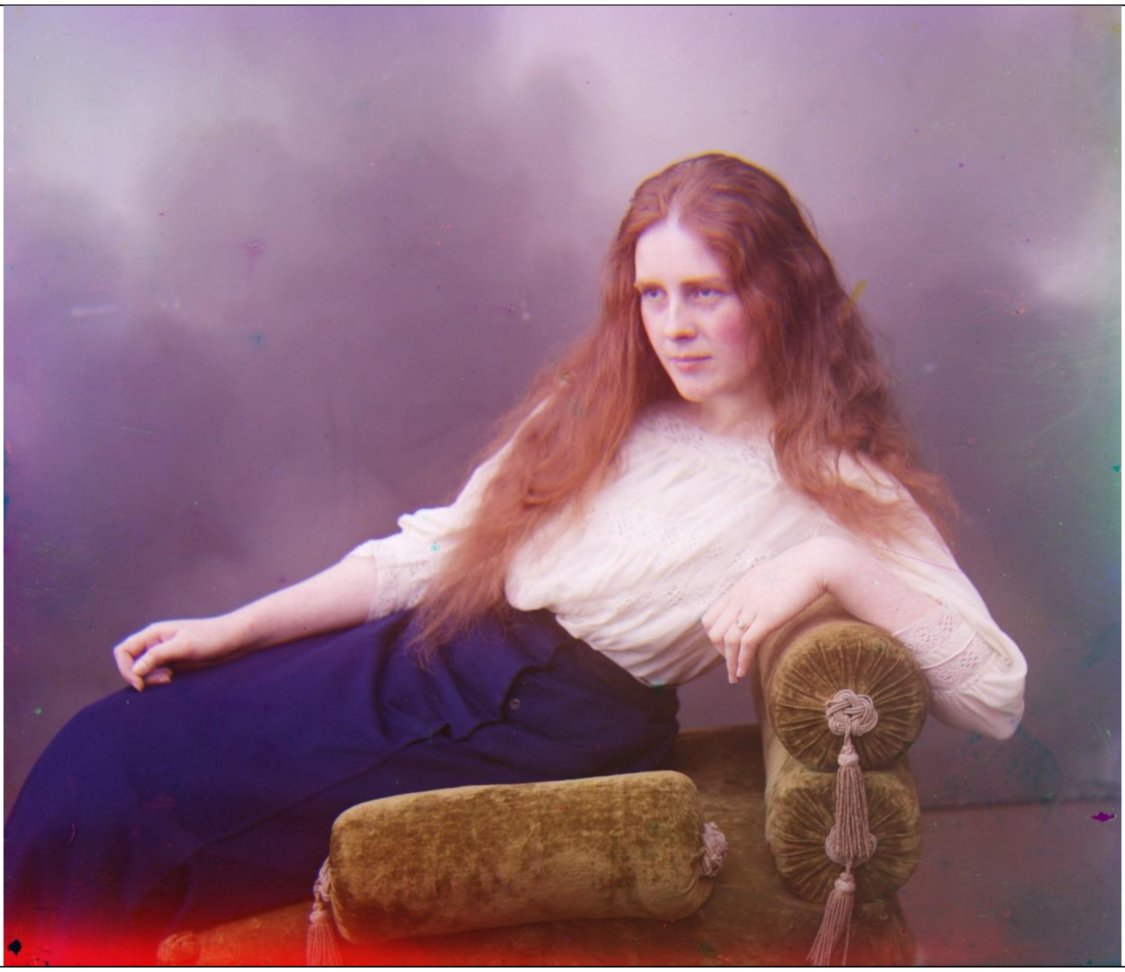

R to B alignment without preprocessing

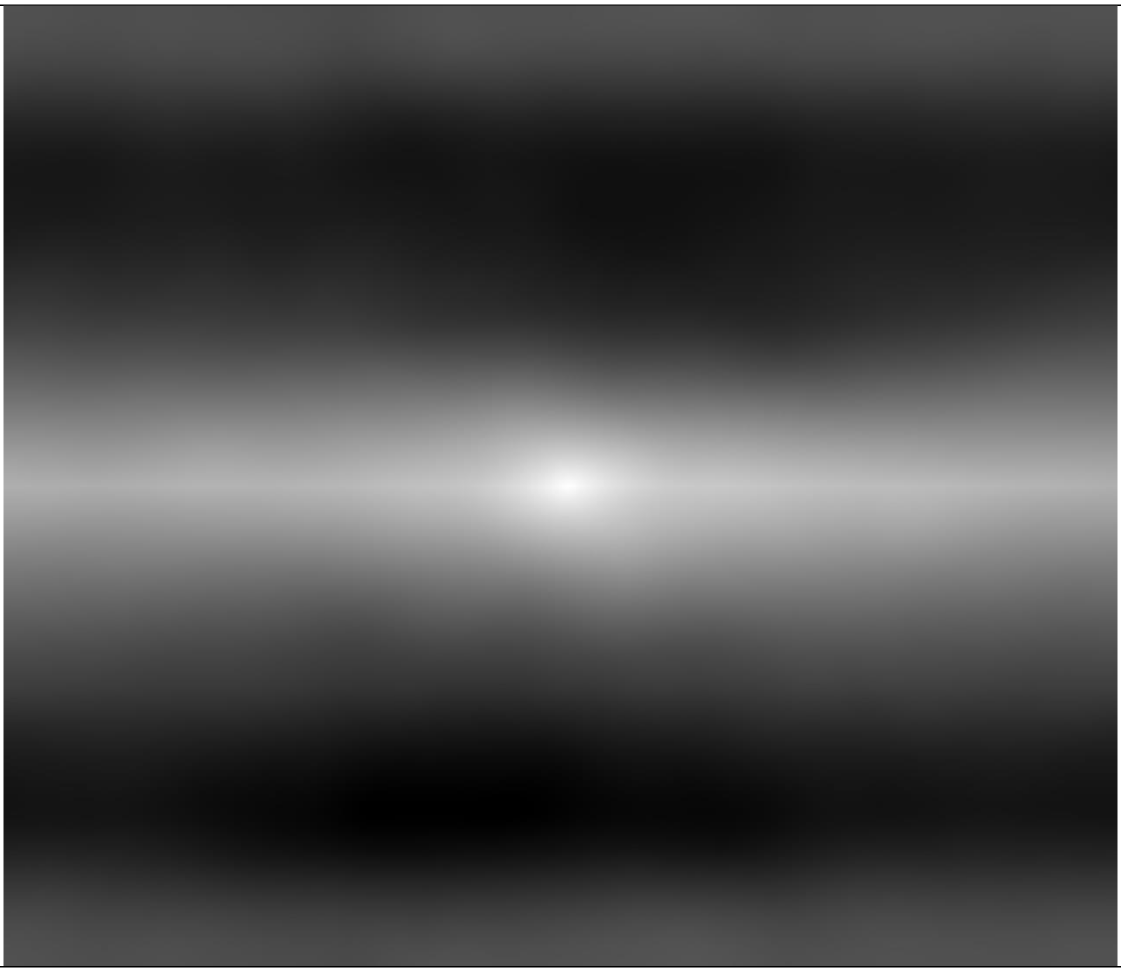

G to B alignment without preprocessing

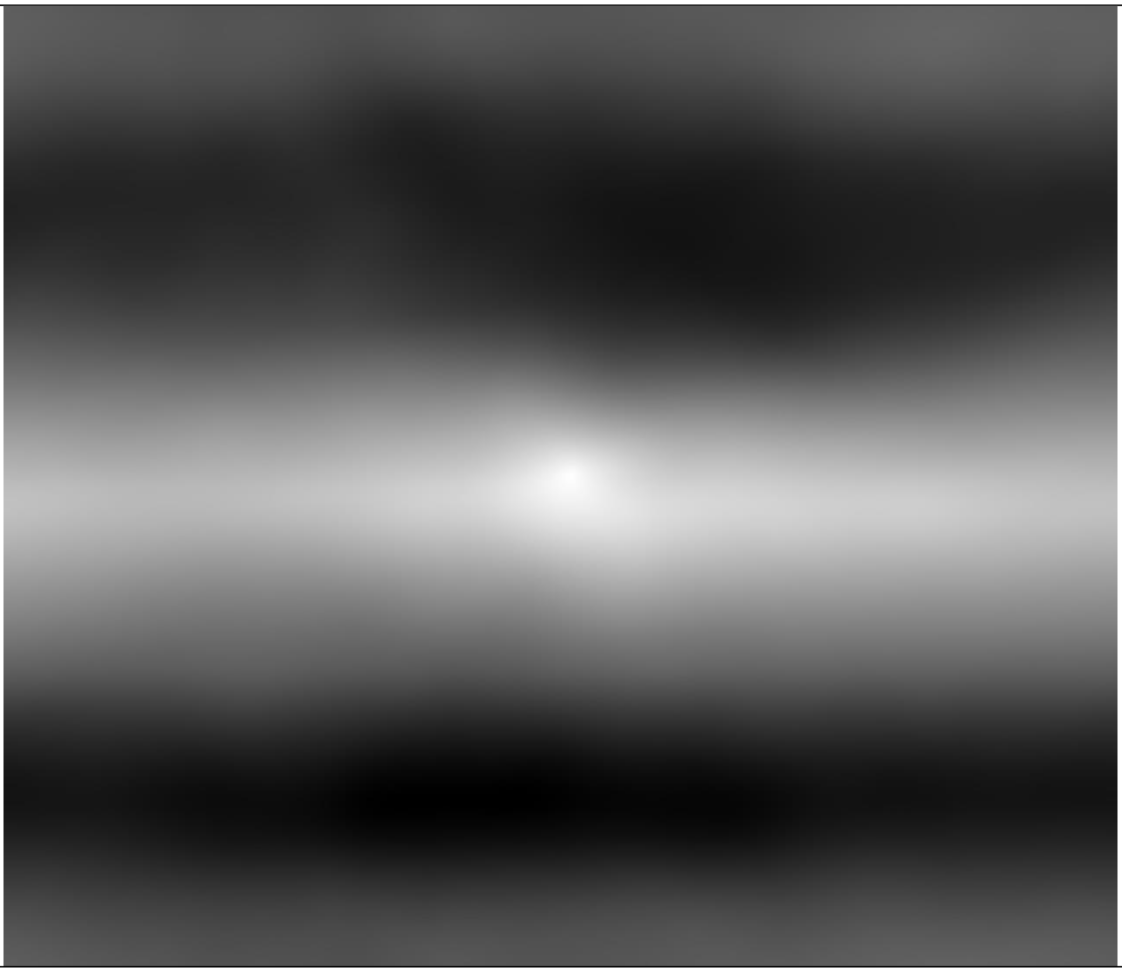


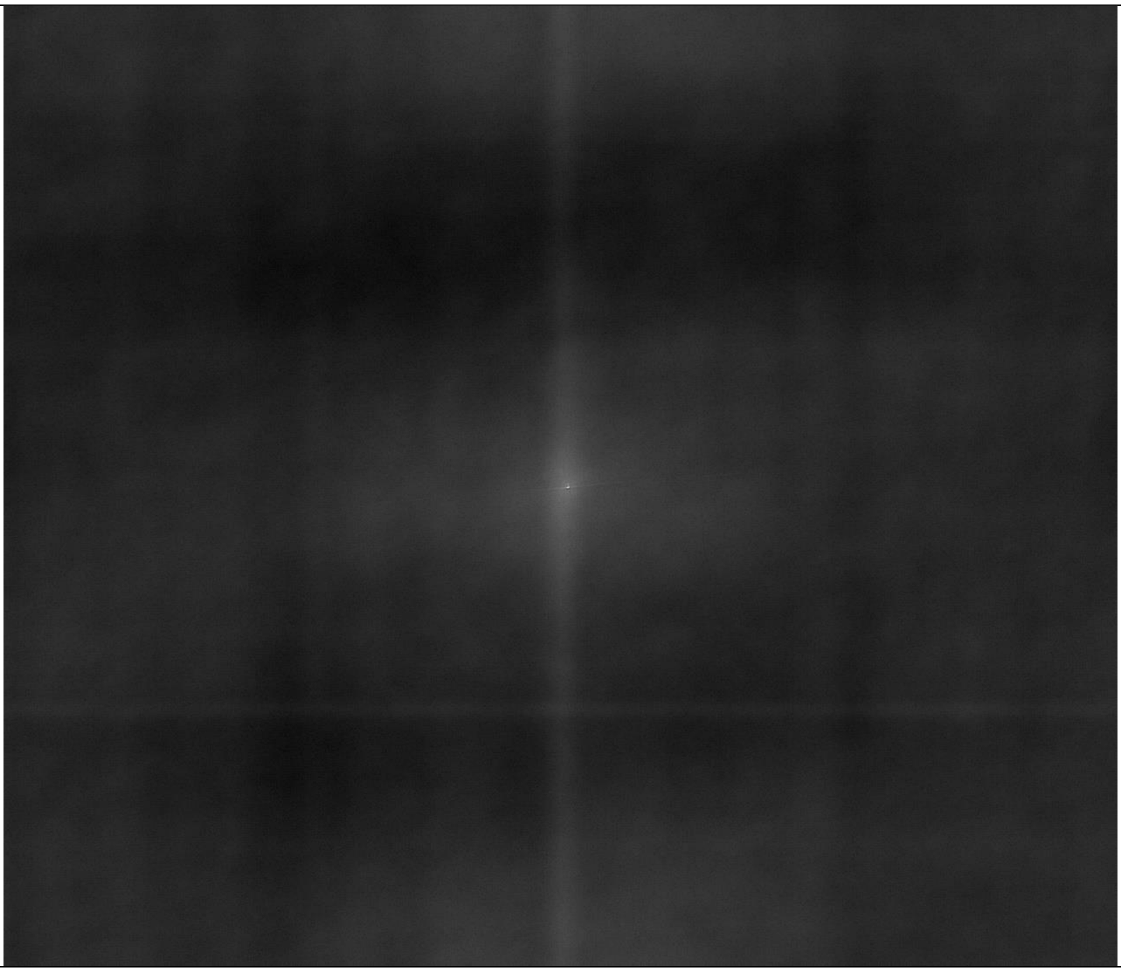
R to B alignment without preprocessing

01657u.tif

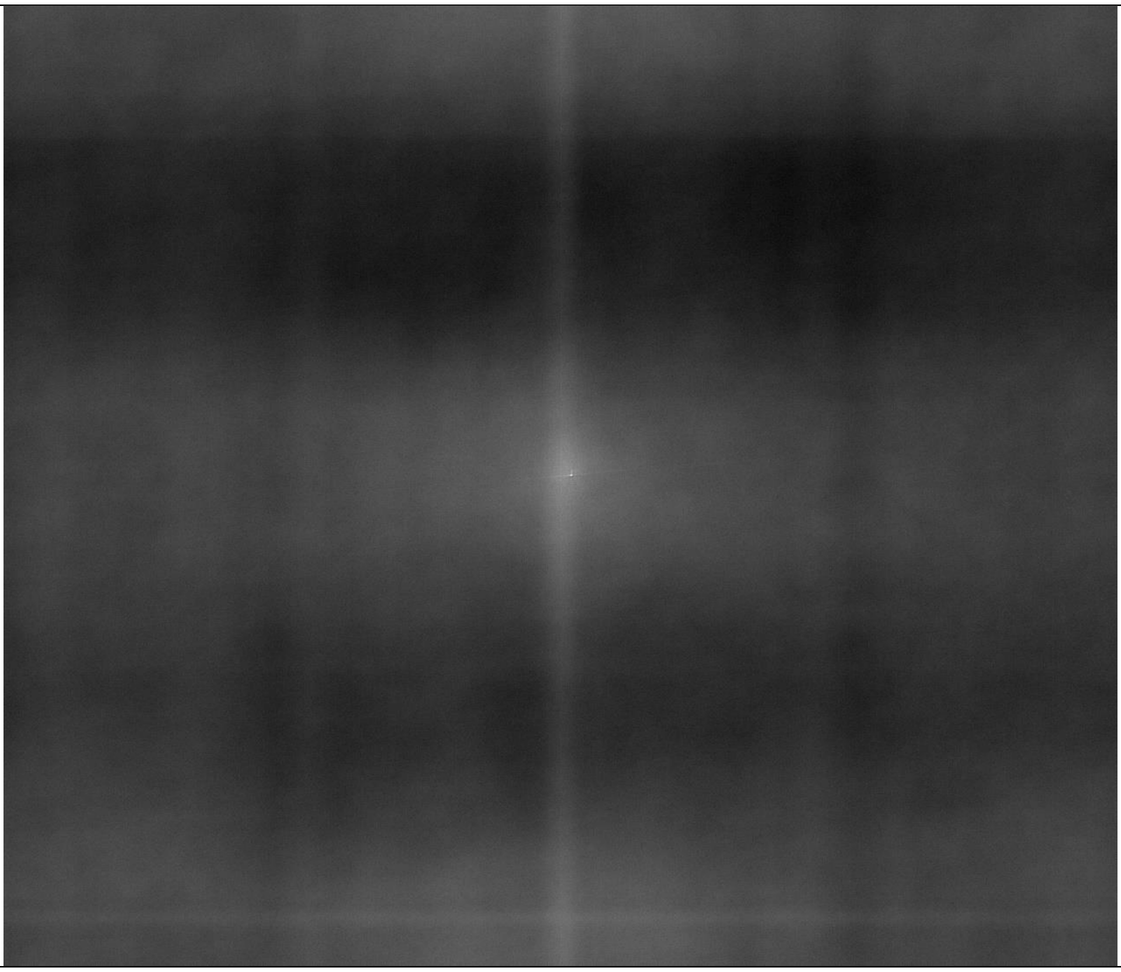

aligned image


G to B alignment without preprocessing


R to B alignment without preprocessing

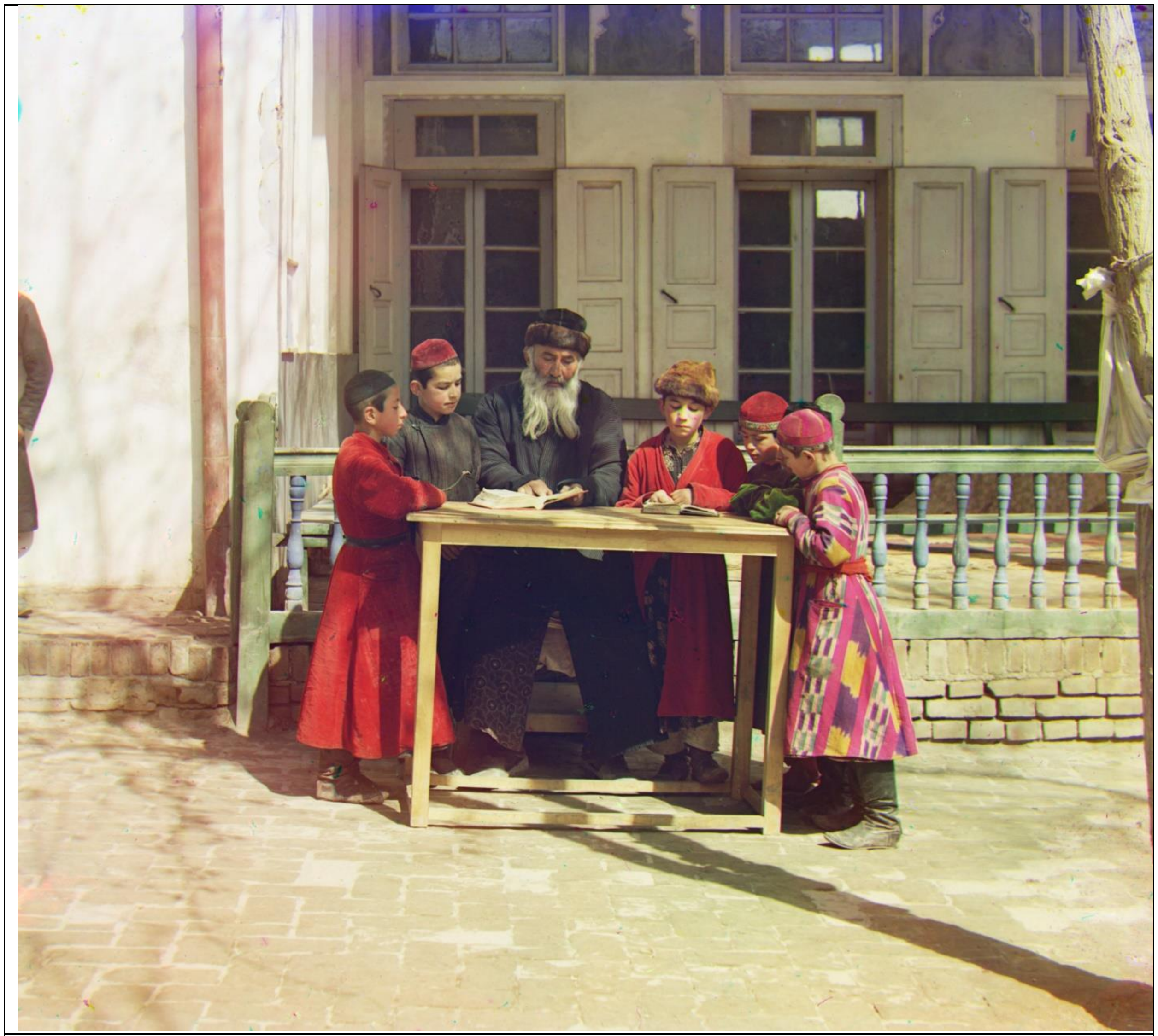


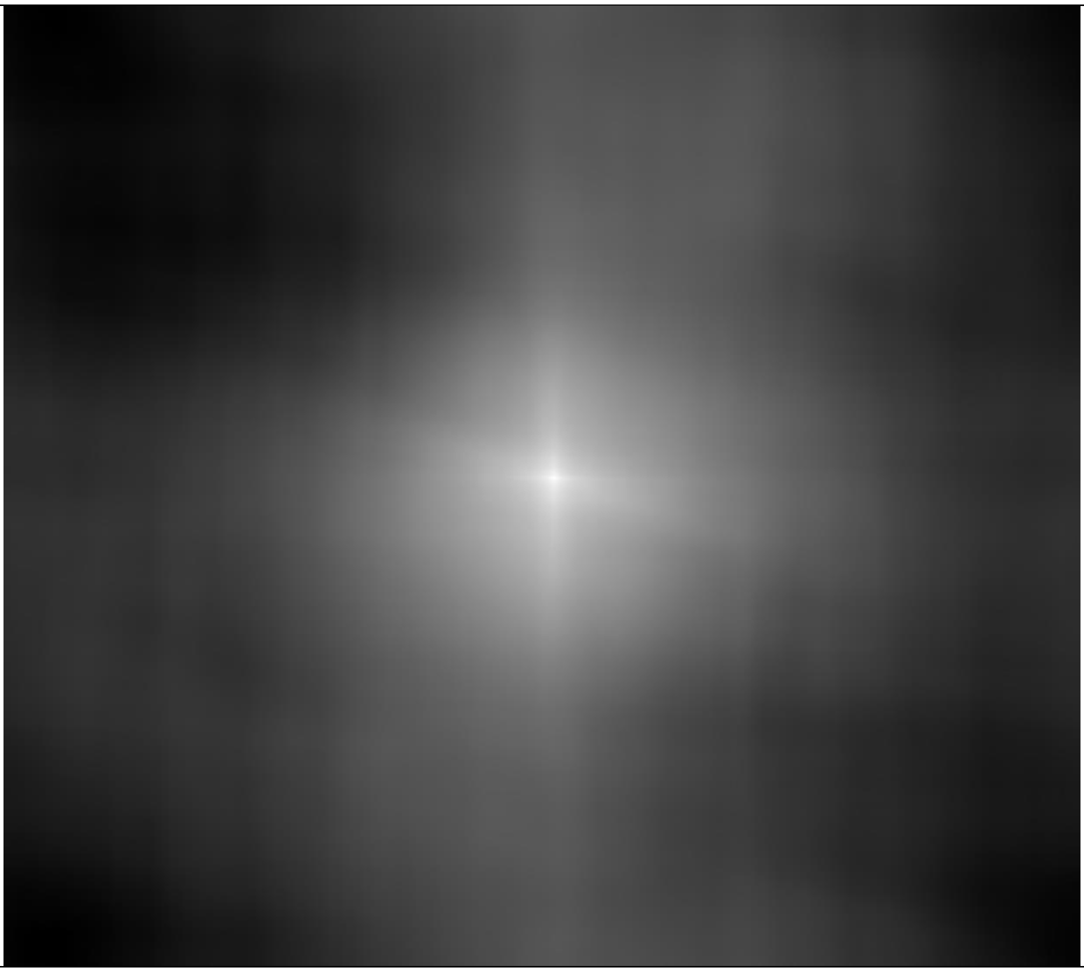
G to B alignment without preprocessing



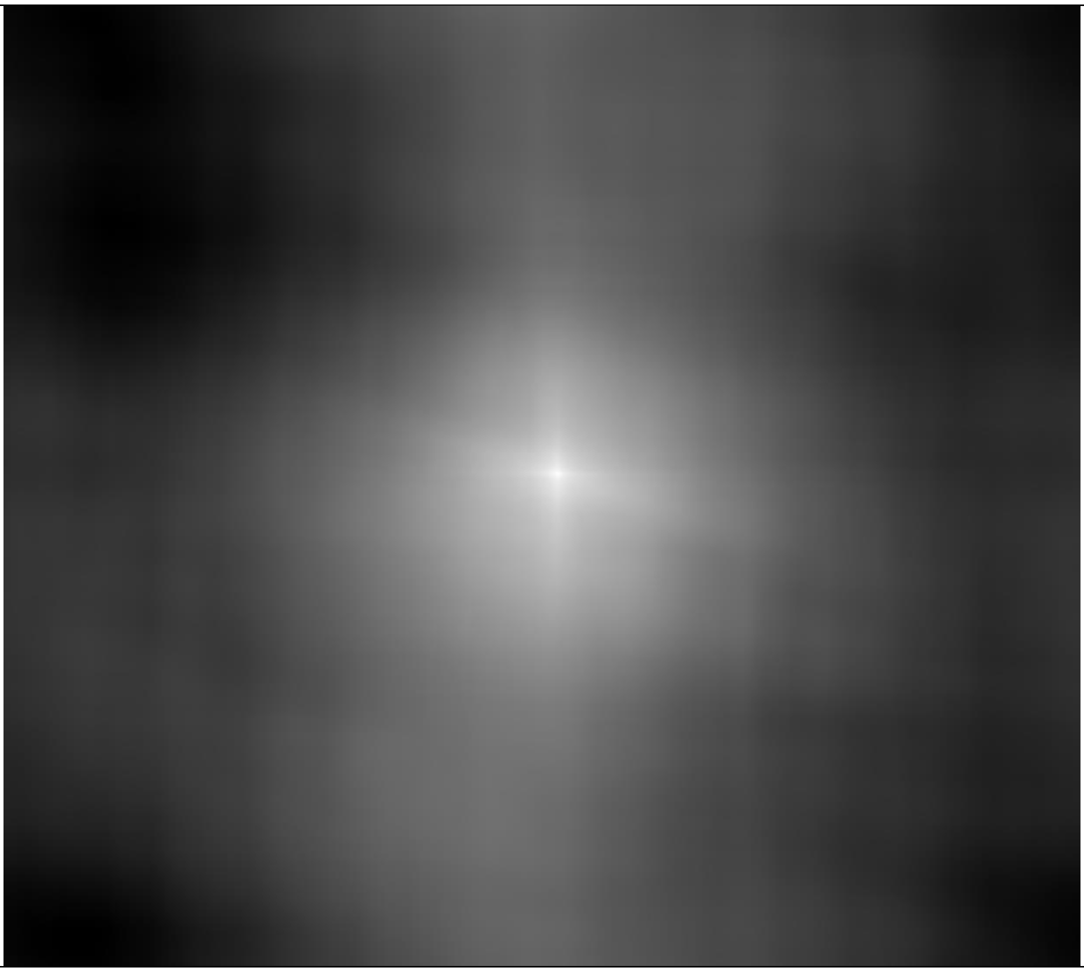
R to B alignment without preprocessing

01861a.tif

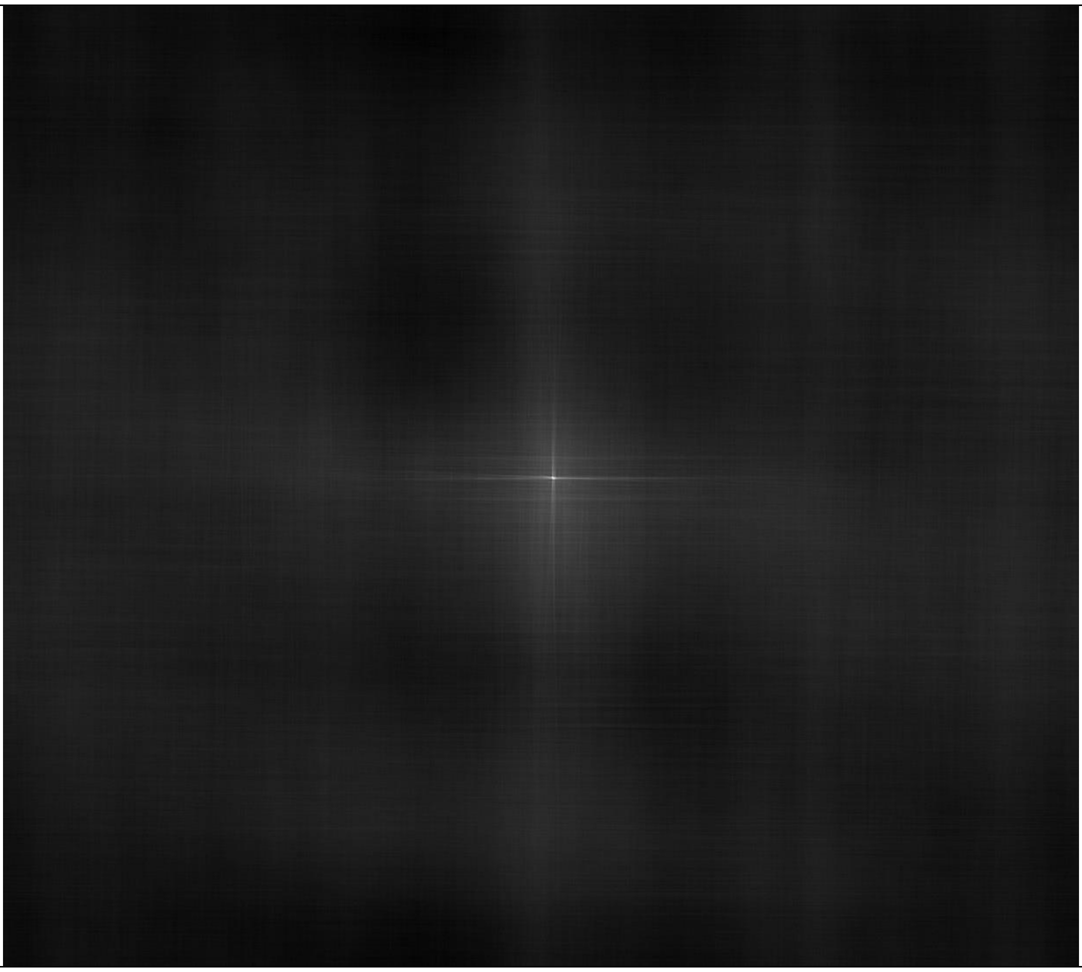

aligned image

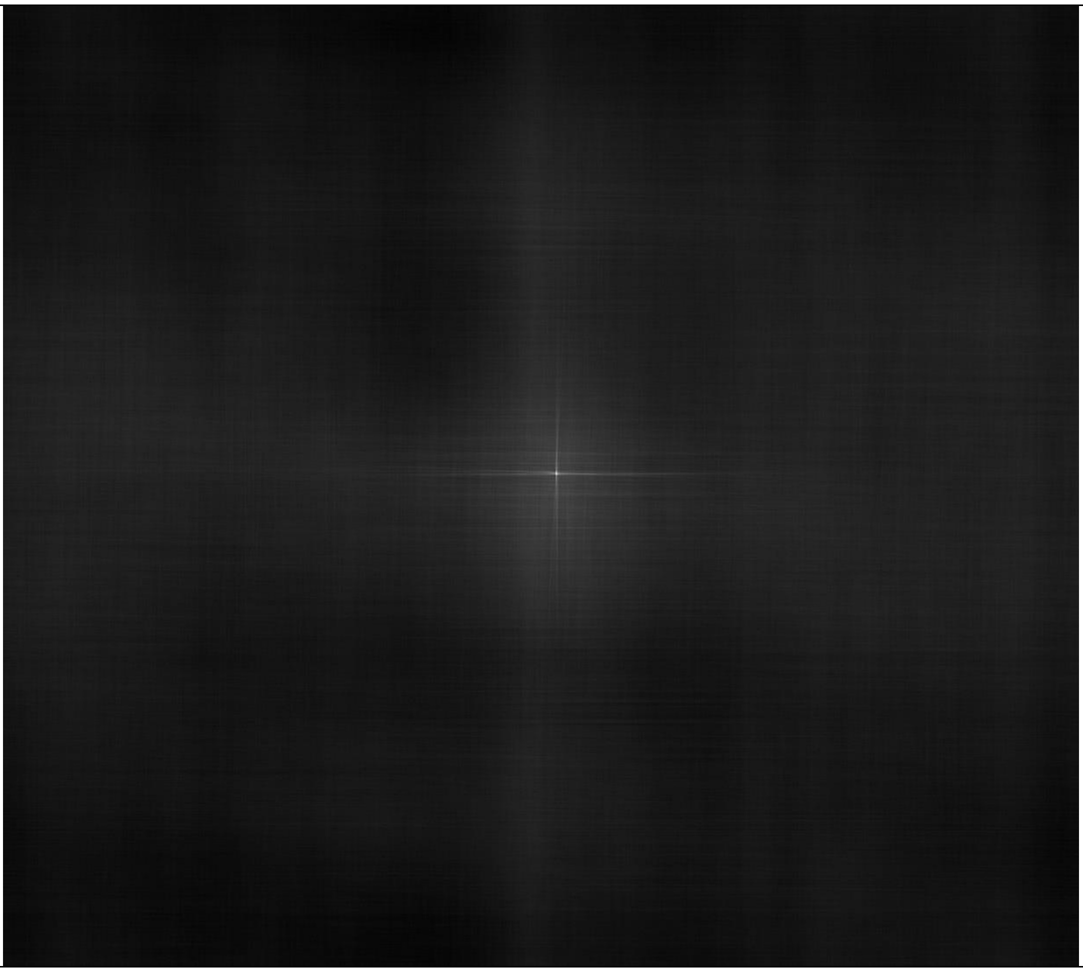


G to B alignment without preprocessing



R to B alignment without preprocessing

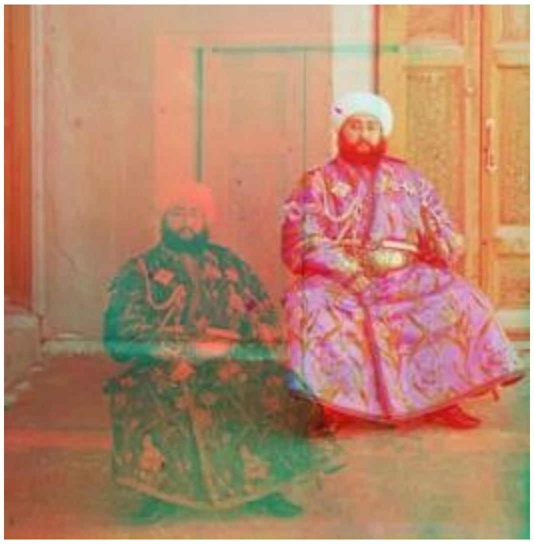

G to B alignment without preprocessing



R to B alignment without preprocessing

C: Discussion and Runtime Comparison

The peak of the Inverse Fourier transform output becomes significantly more pronounced following preprocessing steps such as sharpening. Consequently, the alignment results improve noticeably when preprocessing is applied.



An example of failure

In certain instances, the maximum value acquired without preprocessing can be misleading, as illustrated in the above figure. Preprocessing, on the other hand, has the capability to effectively rectify this issue.

Runtime comparison:

File name	Run time without preprocessing	Run time with preprocessing
merged_00125v.jpg	0.02905	0.03305
merged_00149v.jpg	0.02861	0.04118
merged_00153v.jpg	0.02902	0.04140
merged_00351v.jpg	0.03562	0.03133
merged_00398v.jpg	0.03355	0.04133
merged_01112v.jpg	0.03129	0.04005
merged_01047u.tif	6.551	5.639
merged_01657u.tif	8.104	5.720
merged_01861a.tif	8.365	5.579

When contrasting these findings with those obtained in Assignment One, it is evident that the computational time has been notably diminished by approximately 50%. This observation underscores the superior efficiency of the Fast Fourier Transform (FFT) method over the Normalized Cross-Correlation (NCC) technique in the context of image alignment.

Part 2 Scale-Space Blob Detection:

You will provide the results for ***4 different examples chosen by your own***:

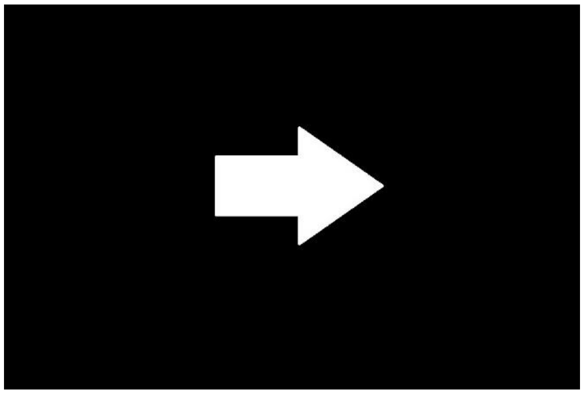
- Original image
- Each of the five modified images (shift, rotate, scale)

You will provide the following as further discussion overall:

- Explanation of any "interesting" implementation choices that you made.

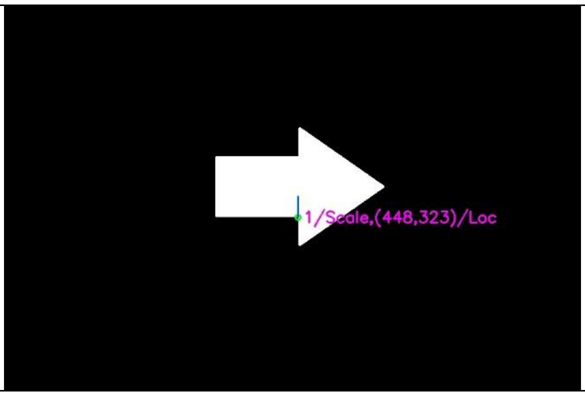
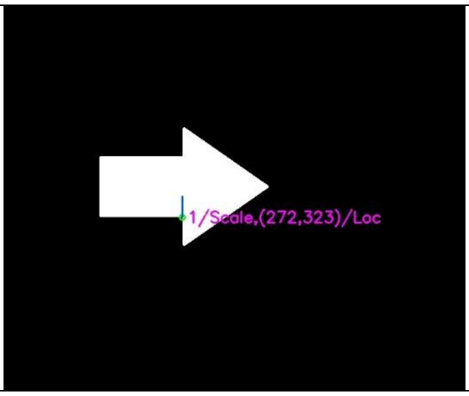
Example 1:

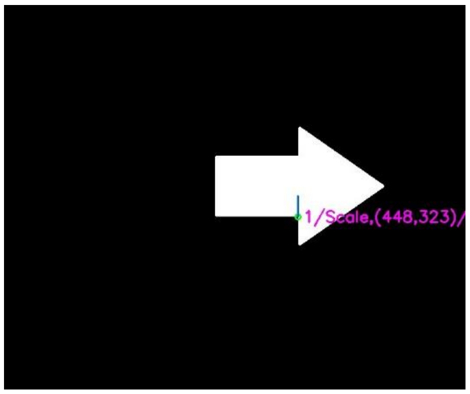
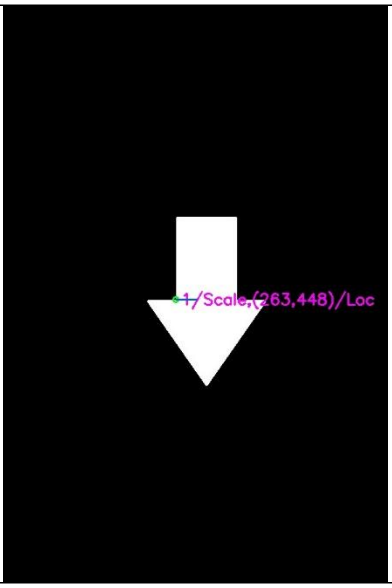
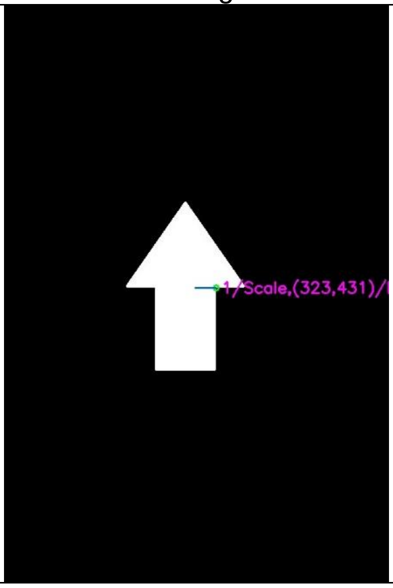
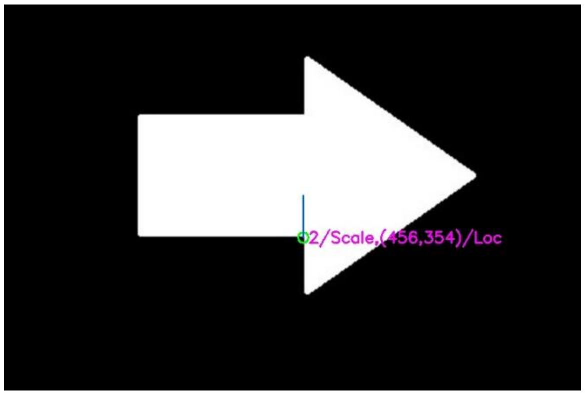
Original image:



Source: https://www.meccanismocomplesso.org/en/opencv-python-harris-corner-detection-a-method-to-detect-corners-in-an-image/, adapted by author for CV assignment, fall 2023

Algorithm output:

	
Without transformation	Shift left

	
Shift right	Rotate clockwise
	
Rotate counterclockwise	Enlarge and crop

Running time:

"efficient" implementation	3.675s

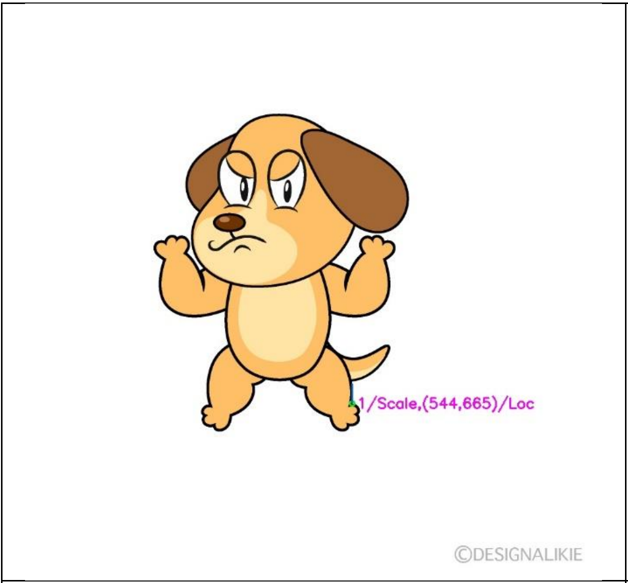
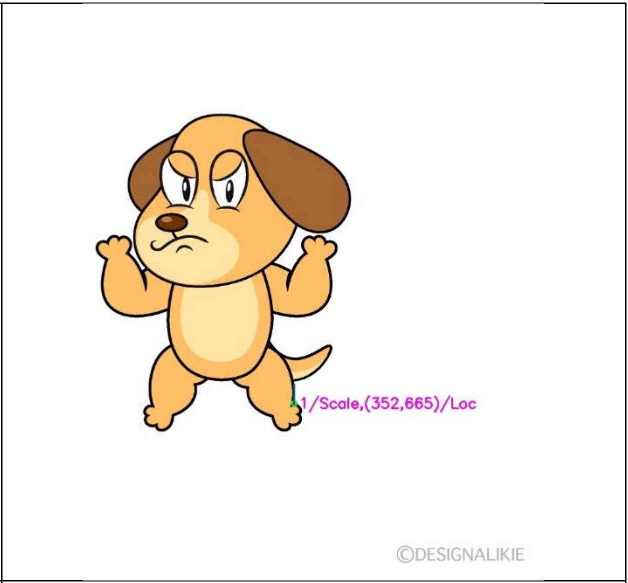
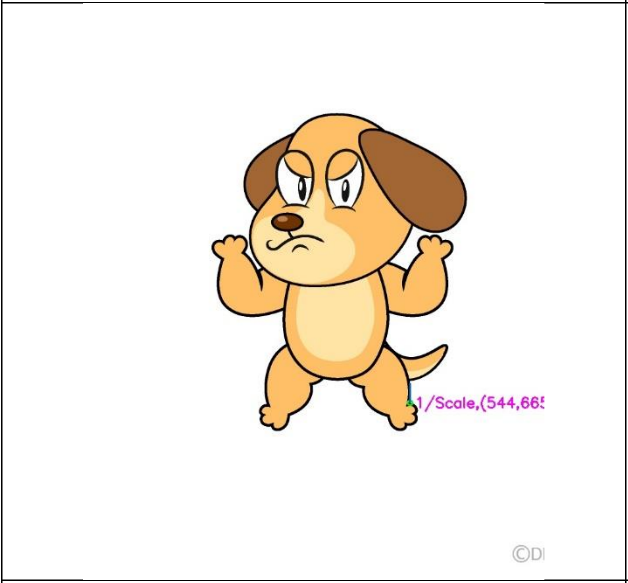
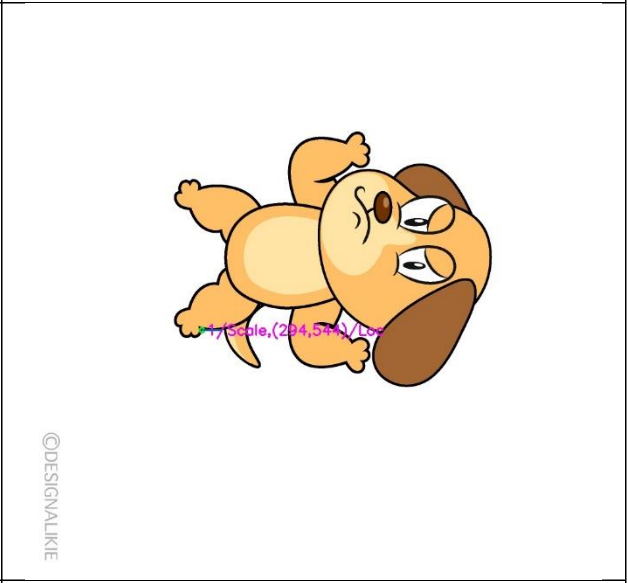
Example 2:

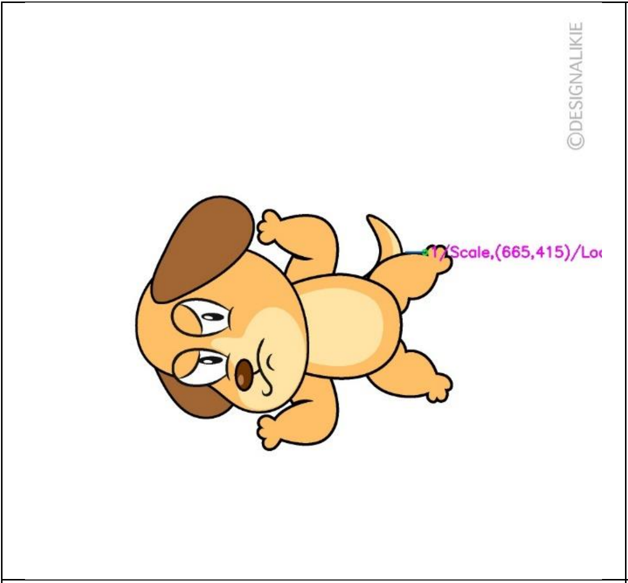
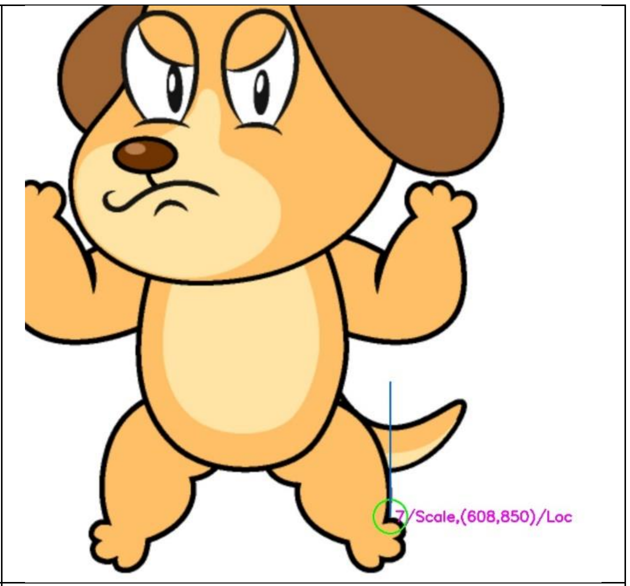
Original image:

©DESIGNALIKIE

Source: https://charatoon.com/?id=396, adapted by author for CV assignment, fall 2023

Algorithm output:

	
Without transformation	Shift left
	
Shift right	Rotate clockwise

	
Rotate counterclockwise	Enlarge and crop

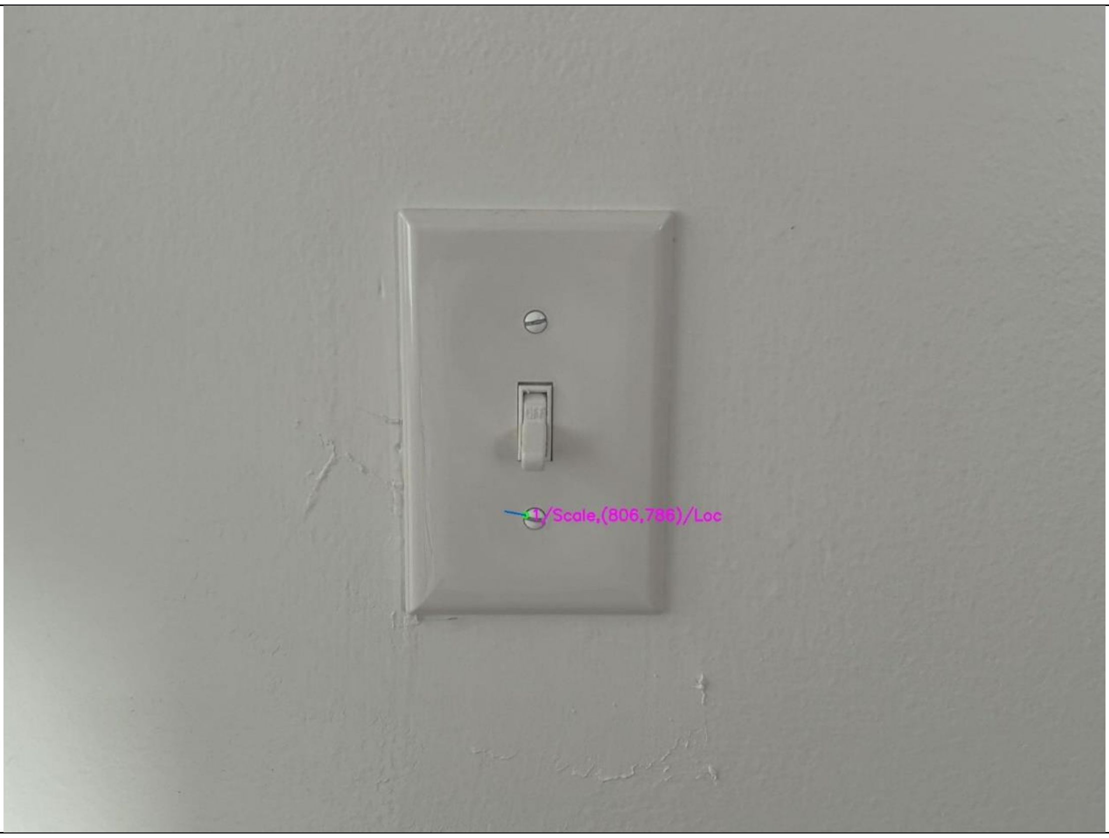
Running time:

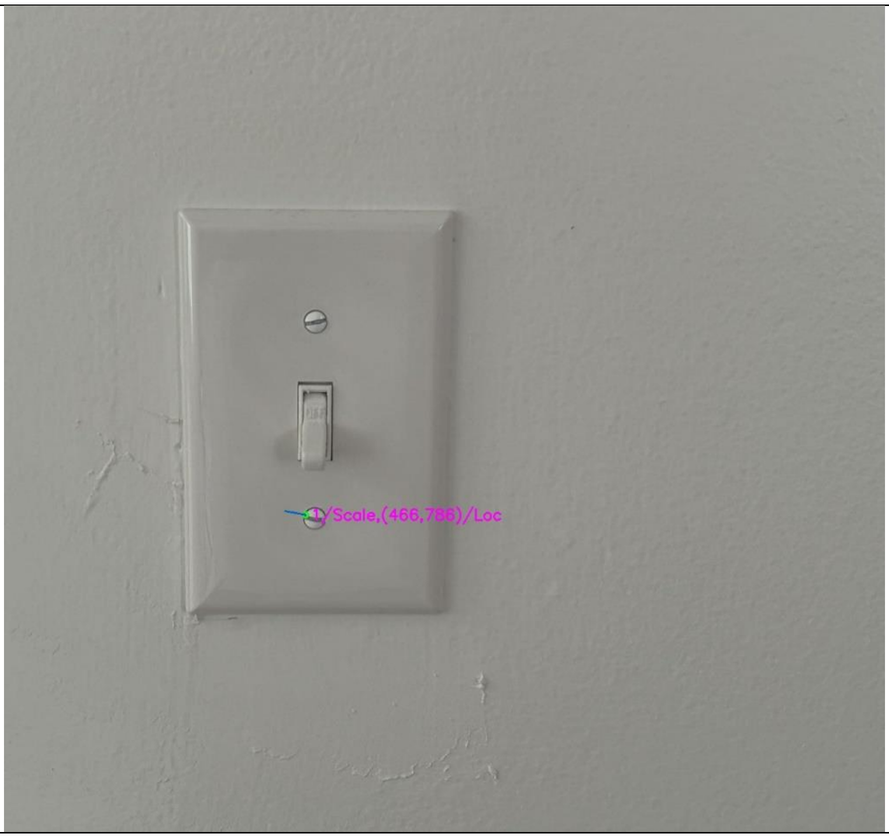
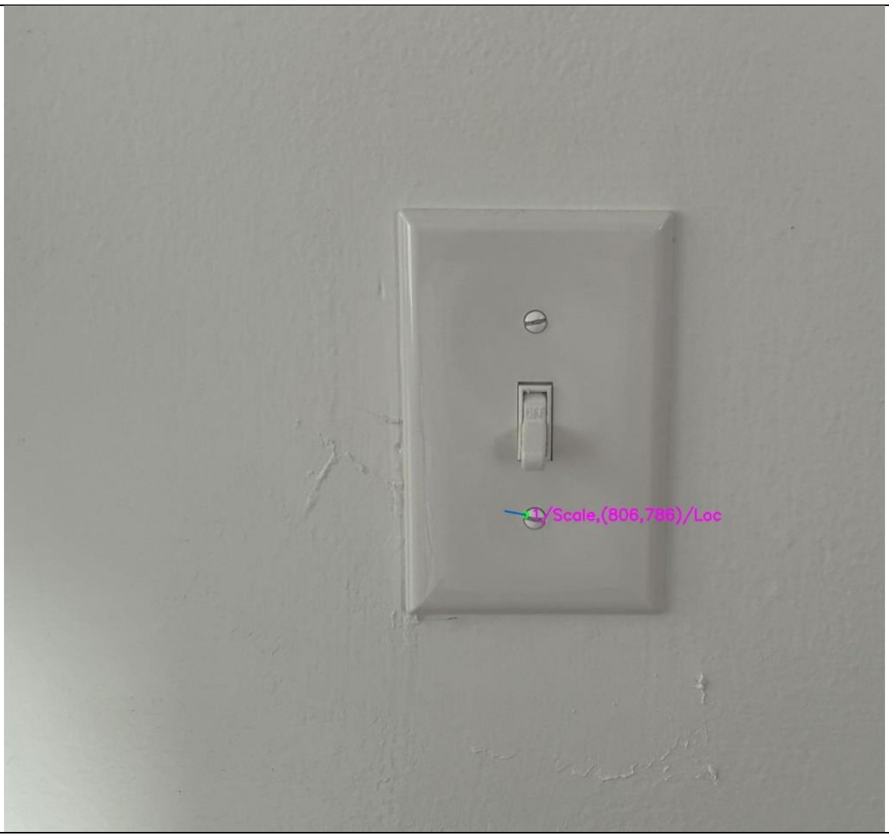
"efficient" implementation	6.609s

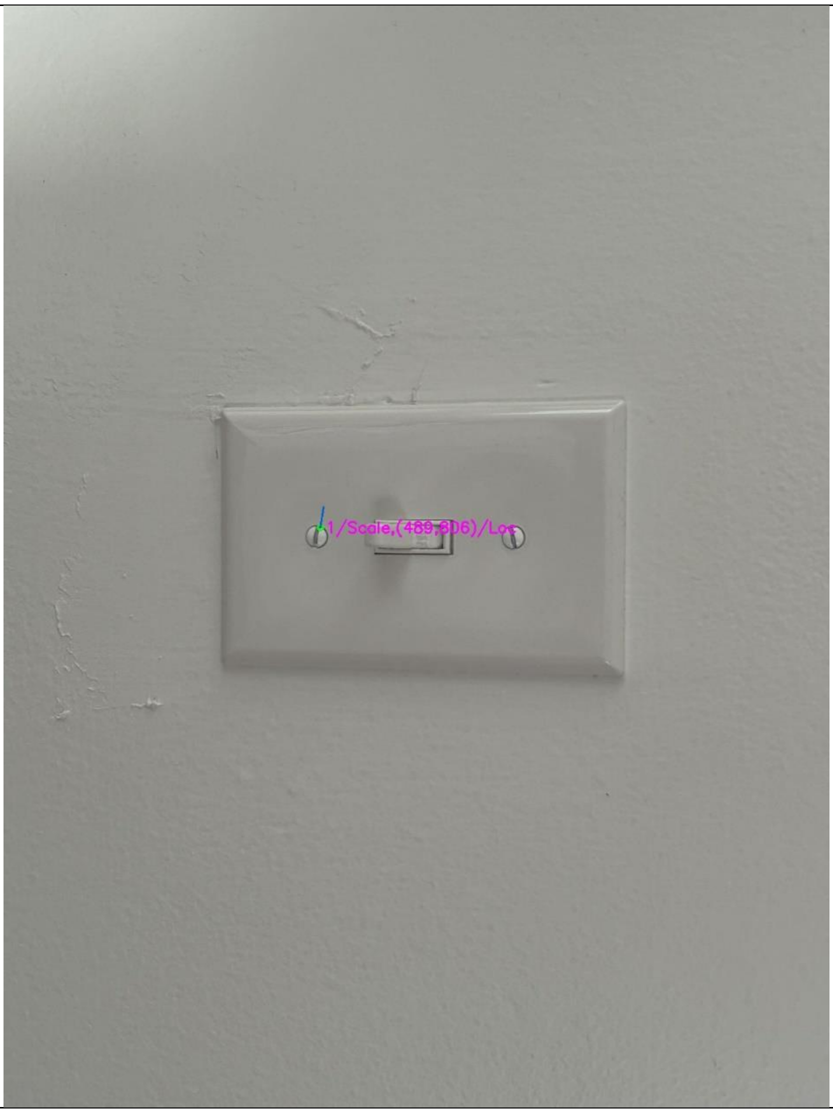
Example 3:

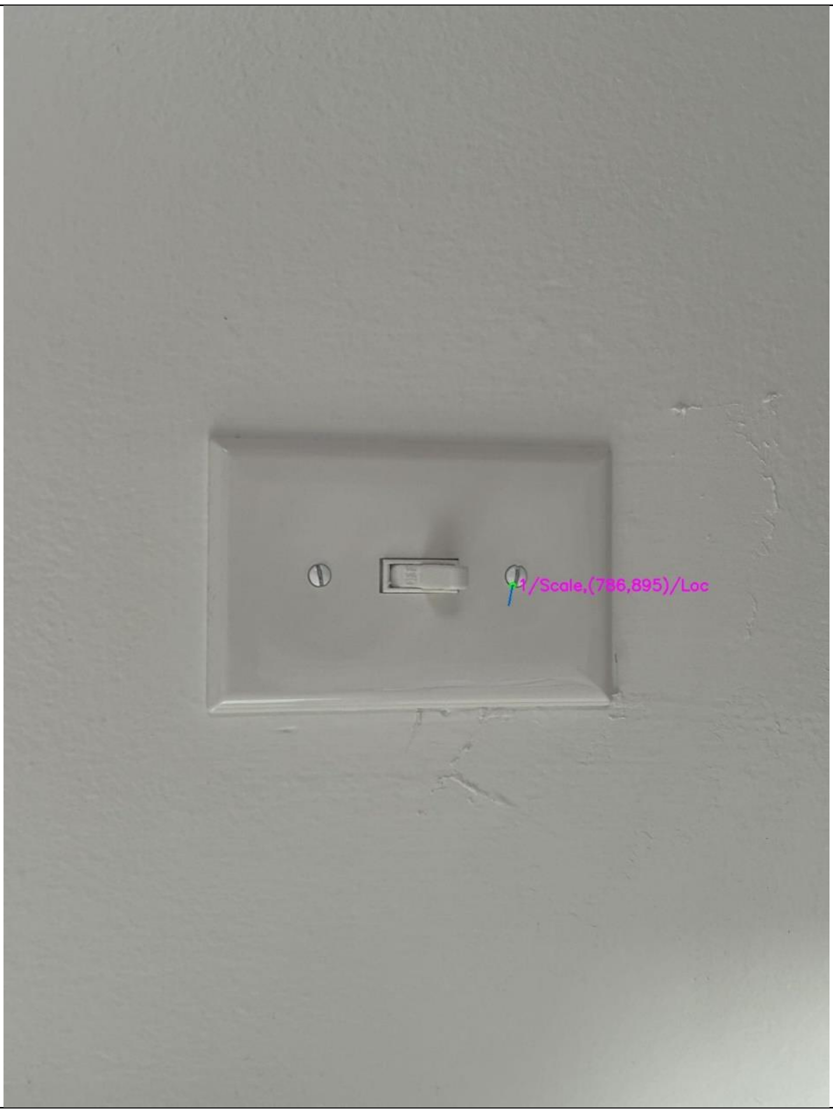
Original image:

Algorithm output:

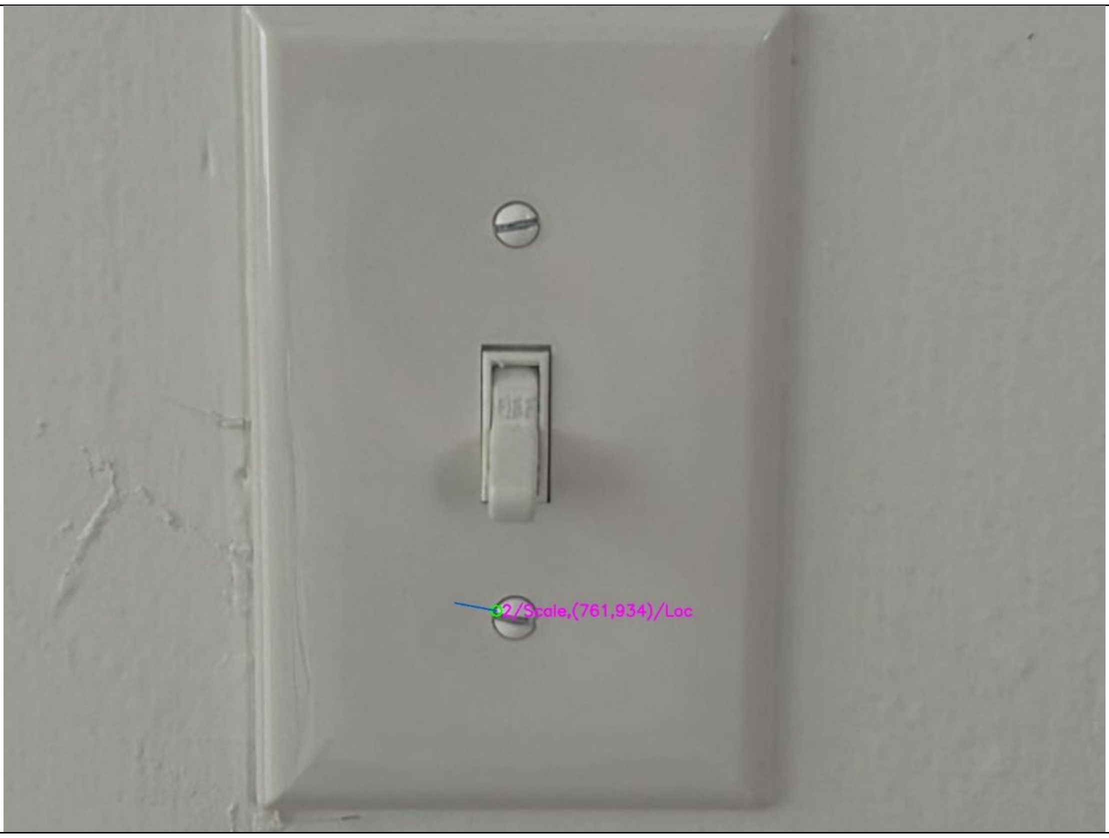

Without transformation


Shift left

Shift right


Rotate clockwise



Rotate counterclockwise


Enlarge and crop

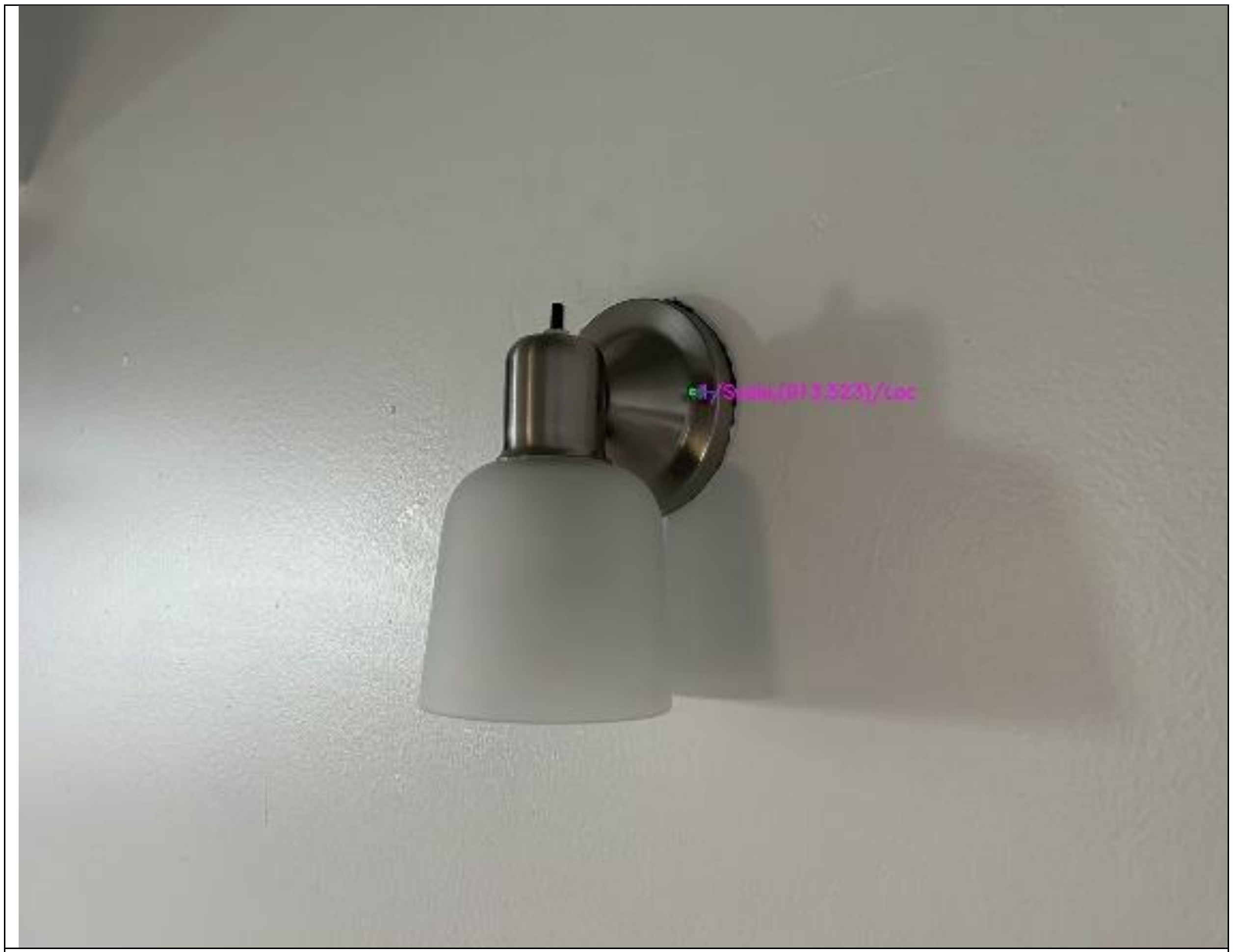
Running time:

"efficient" implementation	15.572s

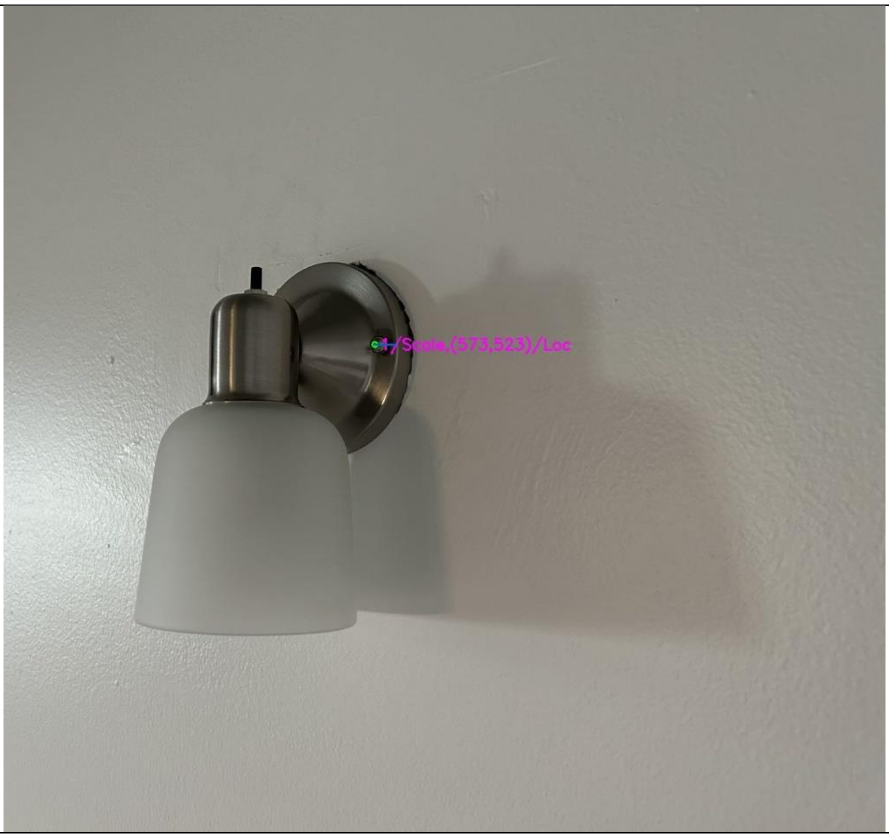
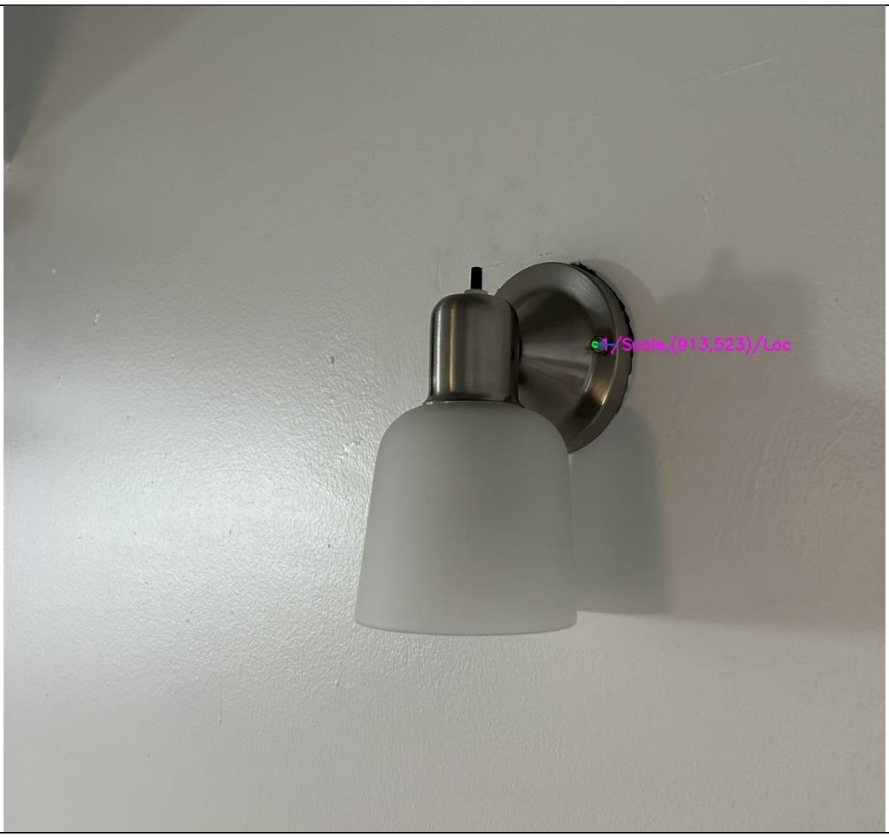
Example 4:

Original image:

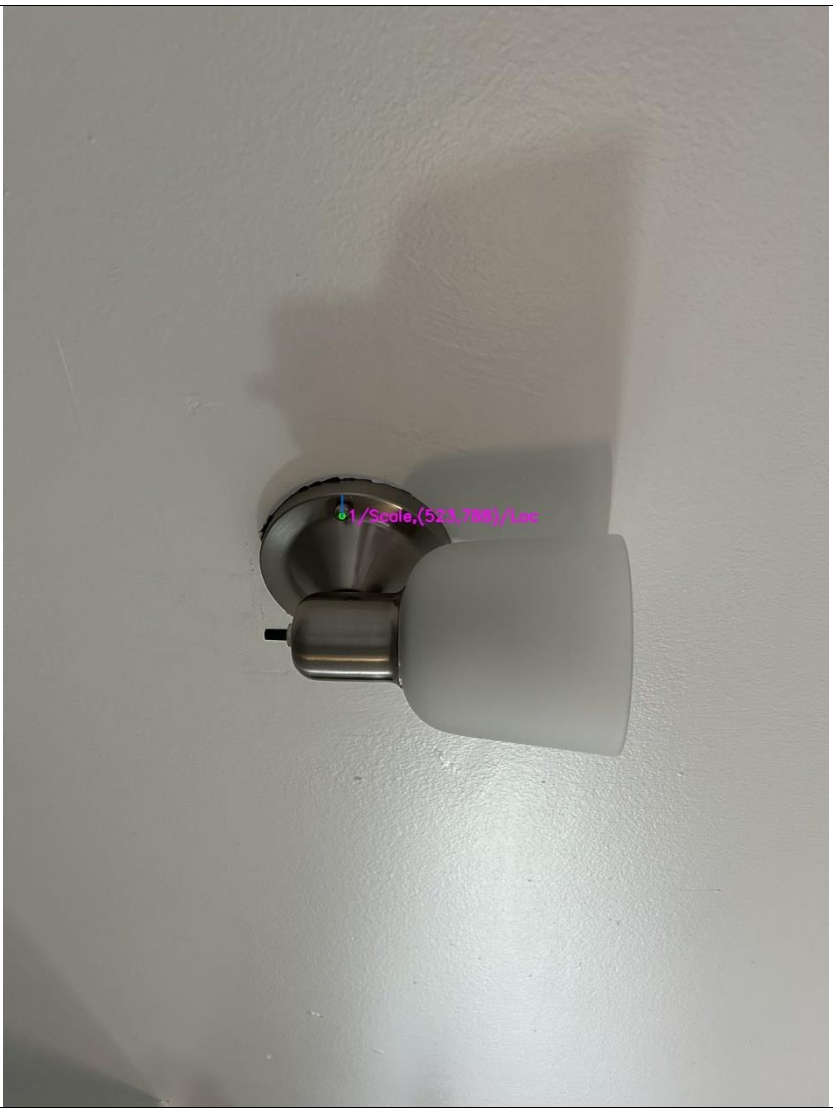
Algorithm output:

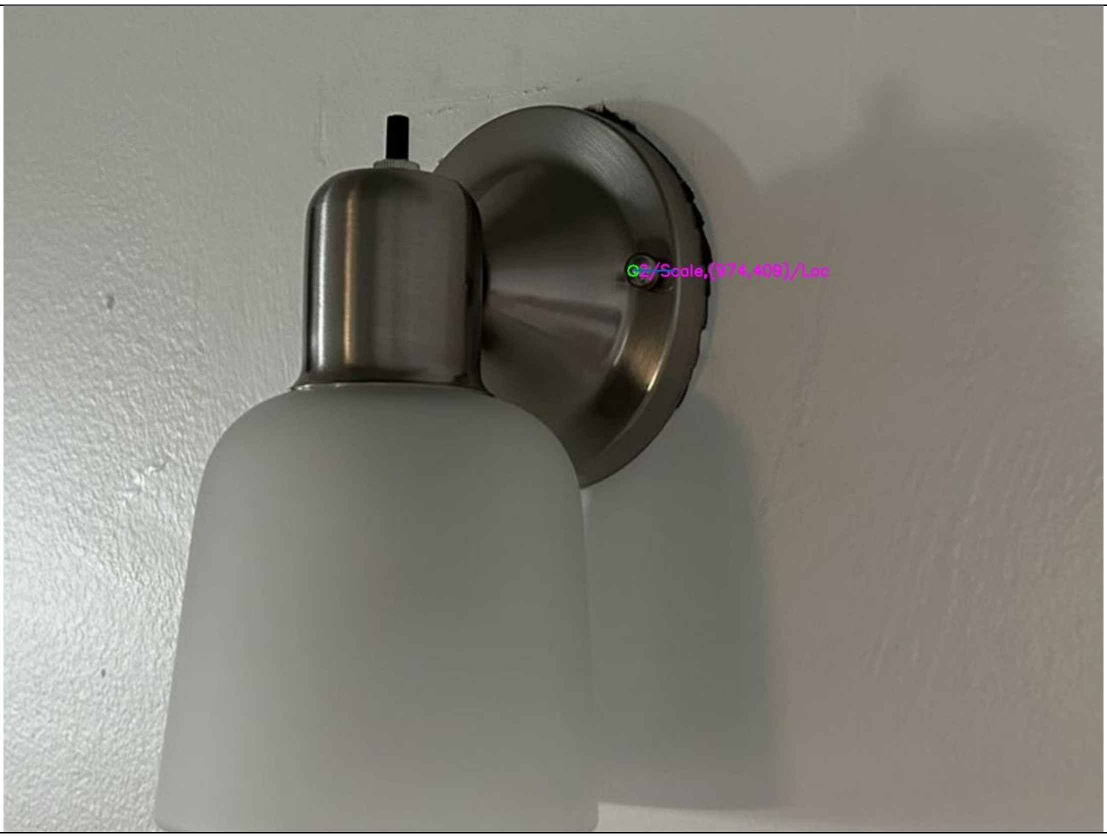


Without transformation


Shift left

Shift right

Rotate clockwise


Rotate counterclockwise


Enlarge and crop

Running time:

"efficient" implementation	15.321s

Discussion:

To enhance visualization, I adjust the corner detection threshold in the Harris detector to a value of one. This setting ensures that only the corner with the highest corner value undergoes further processing for blob detection. As a result, the output typically displays only a few corners, often just one.

The Harris detector, a function provided by the OpenCV library, is employed to identify corners within the image. For each corner detected, I determine the optimal scale value by evaluating log responses across a range from 1 to 7. It's important to note that I scale the log response by the square of the scale as normalization. The window size for blob detection is subsequently set to six times the best scale value. After applying an arctangent function, each pixel within the window represents an orientation value. To identify the most prevalent orientation, I utilize a histogram function.

An issue arises regarding the orientation value's output range, which is $(-\pi, \pi]$, while the input range expected by the histogram function should ideally be $[-\pi, \pi)$. To address this, I resolved the issue by mapping output values equal to π to negative π.