CS543 Assignment 2

Your Name: Hongbo Zheng
Your NetId: hongboz2

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 <C1> as base channel):

Image (base channel)	<C2> (h,w) offset (channel)	<C3> (h,w) offset (channel)
00125v.jpg (B)	(-1, 2) (G)	(-7, 1) (R)
00149v.jpg (B)	(3, 2) (G)	(-6, 1) (R)
00153v.jpg (G)	(4, -3) (B)	(1, 2) (R)
00351v.jpg (B)	(-6, 0) (G)	(-13, 1) (R)
00398v.jpg (B)	(0, 2) (G)	(0, 4) (R)
01112v.jpg (G)	(10, 0) (B)	(3, 1) (R)

High-resolution images (using channel <C1> as base channel):

Image (base channel)	<C2> (h,w) offset (channel)	<C3> (h,w) offset (channel)

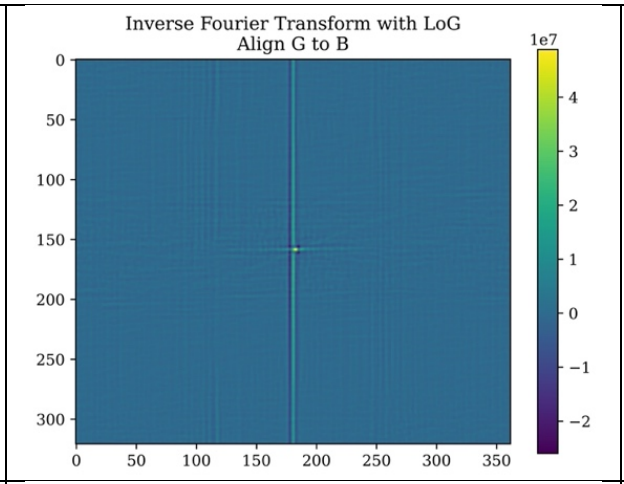
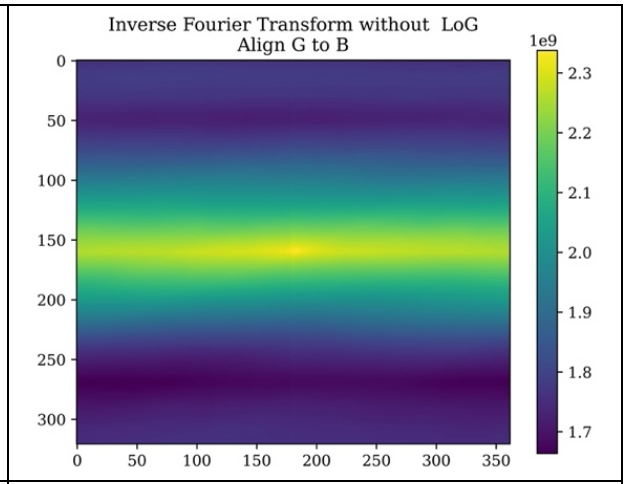
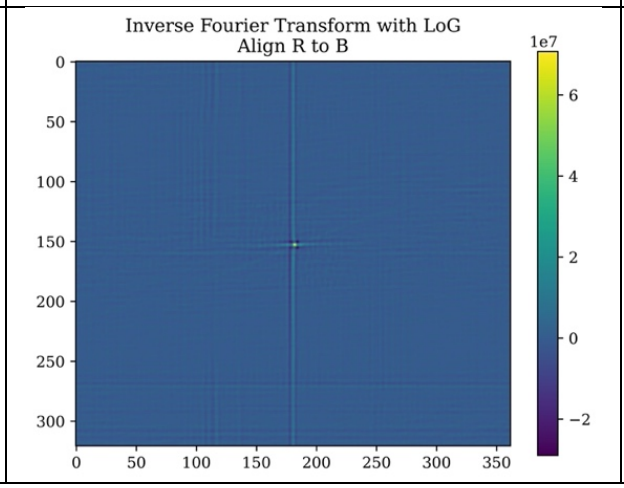
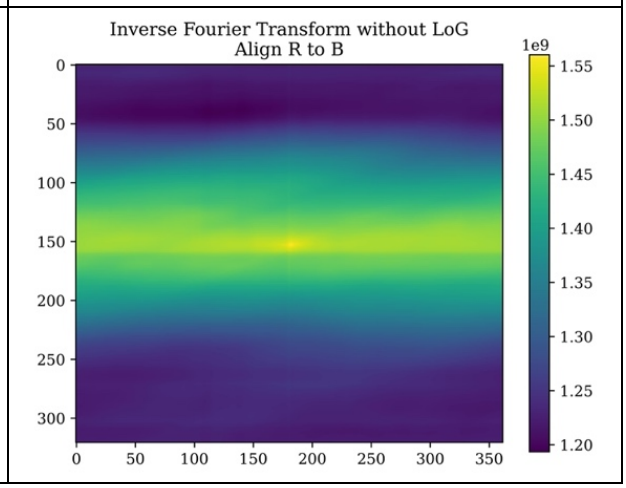
01047u.tif (B)	(-104, 9) (G)	(-80, 13) (R)
01657u.tif (B)	(-37, 19) (G)	(-33, 33) (R)
01861a.tif (B)	(-29, 39) (G)	(1, 62) (R)

B: Output Visualizations

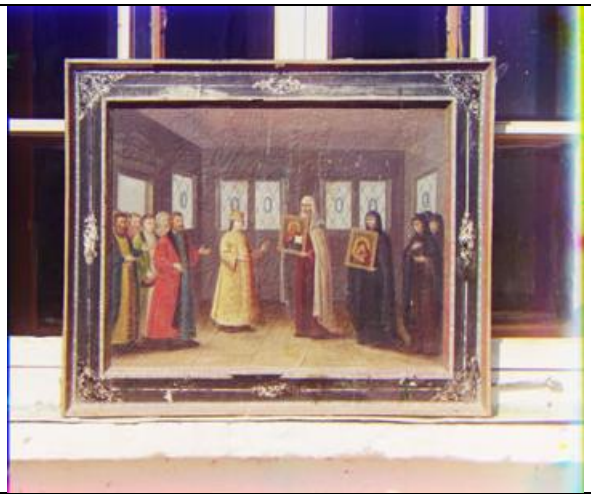
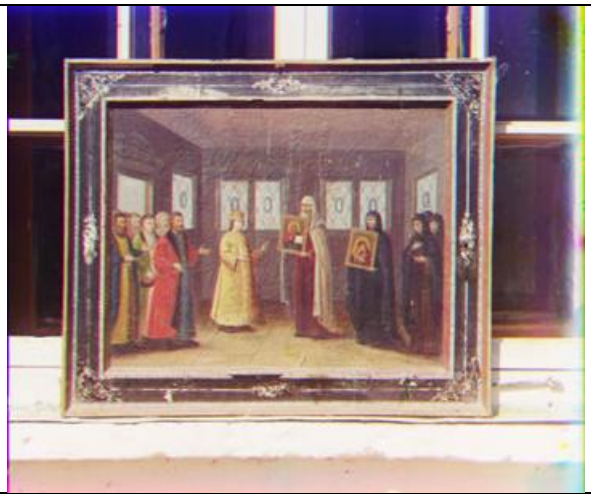
For each image, insert 5 outputs total (aligned image + 4 inverse Fourier transform visualizations) as described above. When you insert these outputs be sure to clearly label the inverse Fourier transform visualizations (e.g. “G to B alignment without preprocessing”).

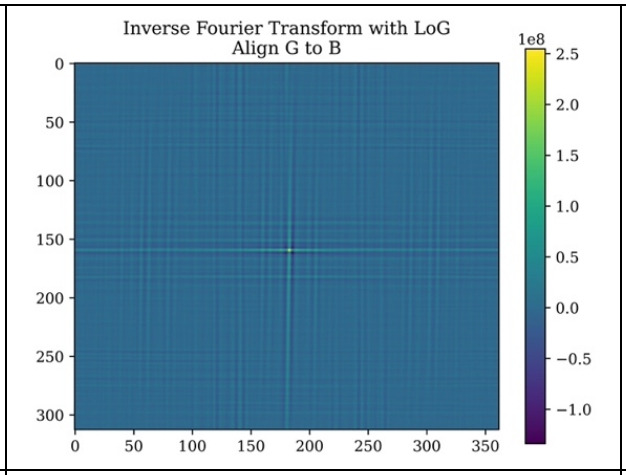
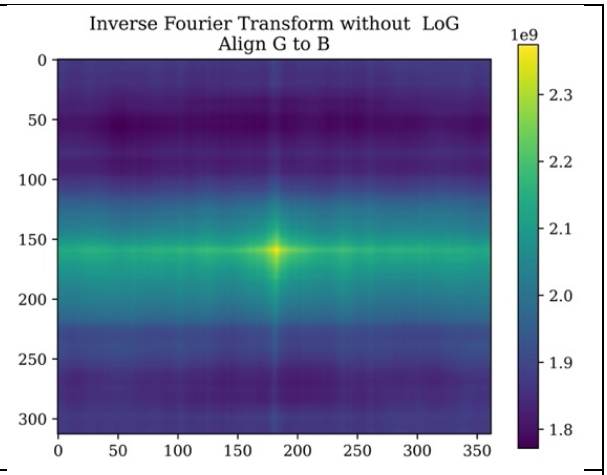
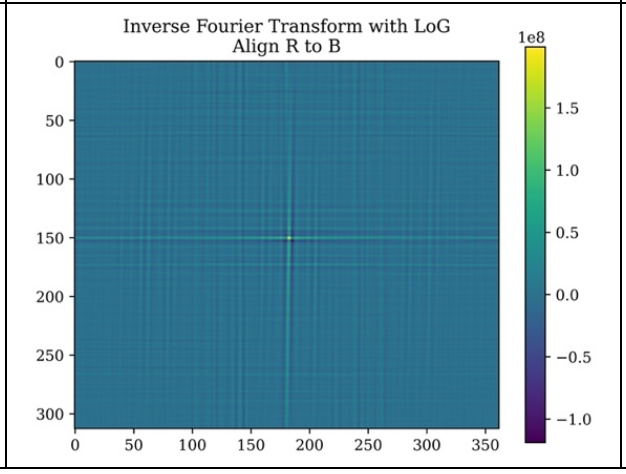
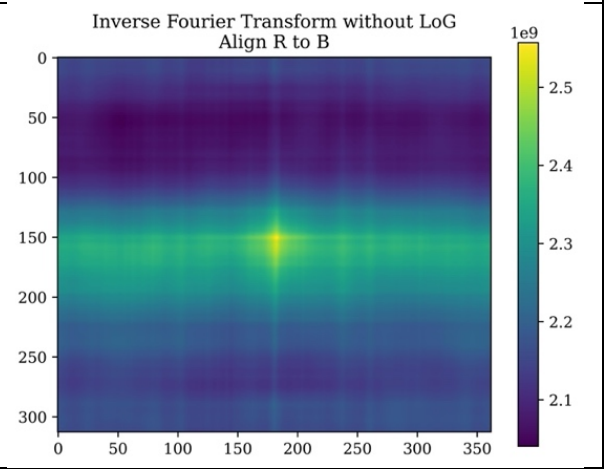
00125v.jpg

	LoG Preprocessing	No LoG Preprocessing
Aligned Images (B)		

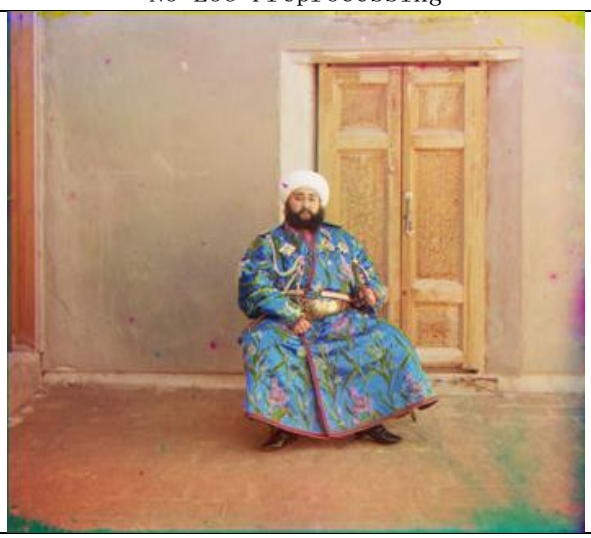
Inverse FT Align G to B		
Inverse FT Align R to B		

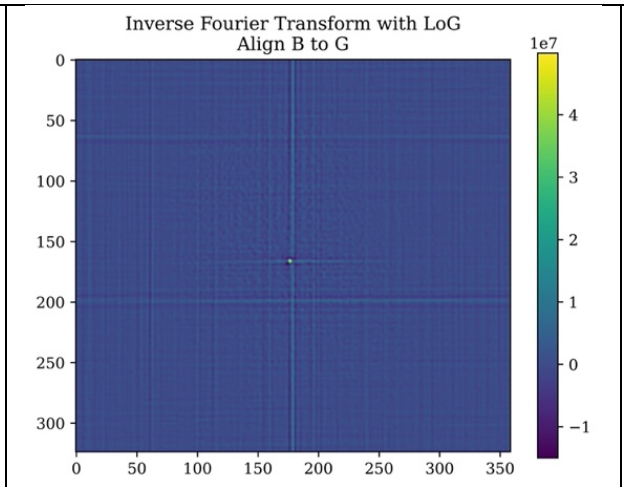
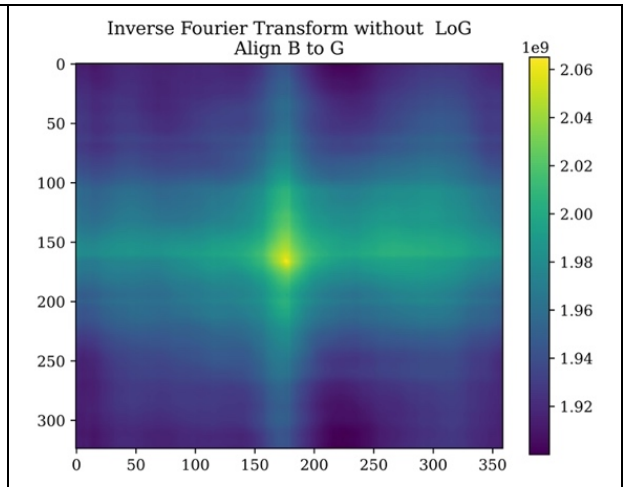
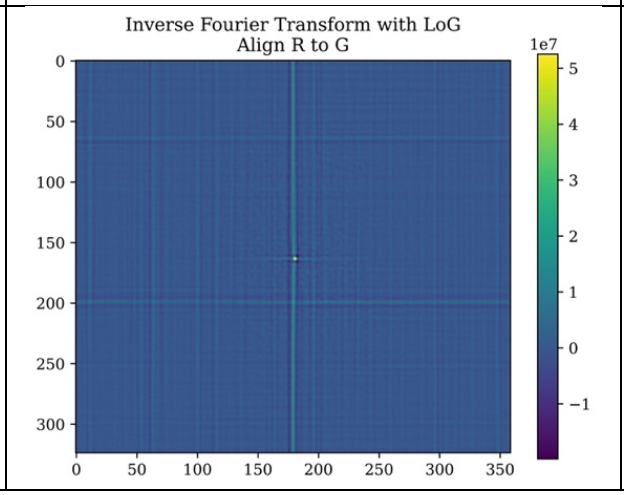
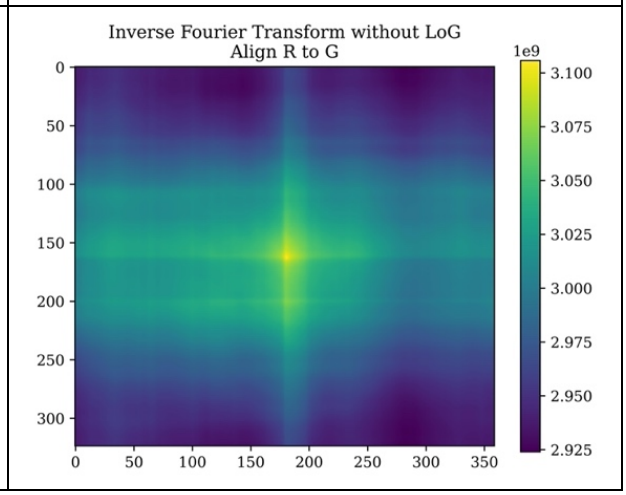
00149v.jpg

	LoG Preprocessing	No LoG Preprocessing
Aligned Images (B)		

Inverse FT Align G to B		
Inverse FT Align R to B		

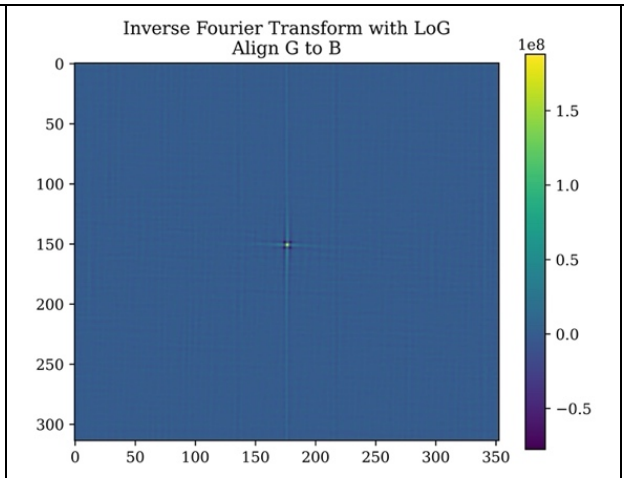
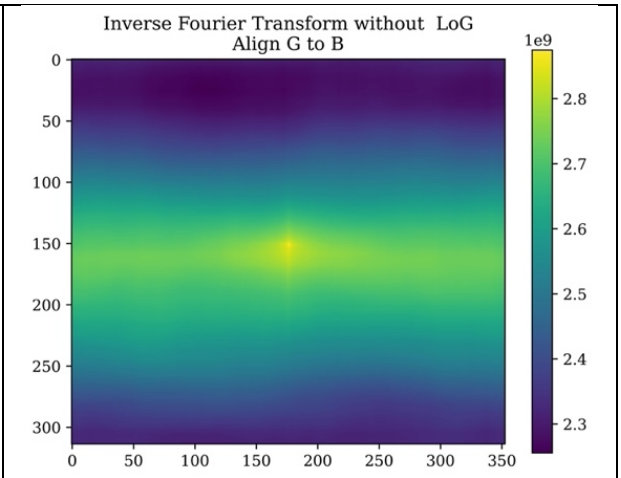
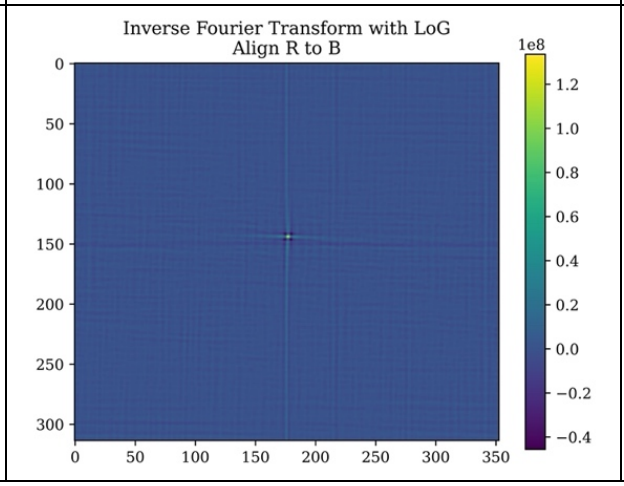
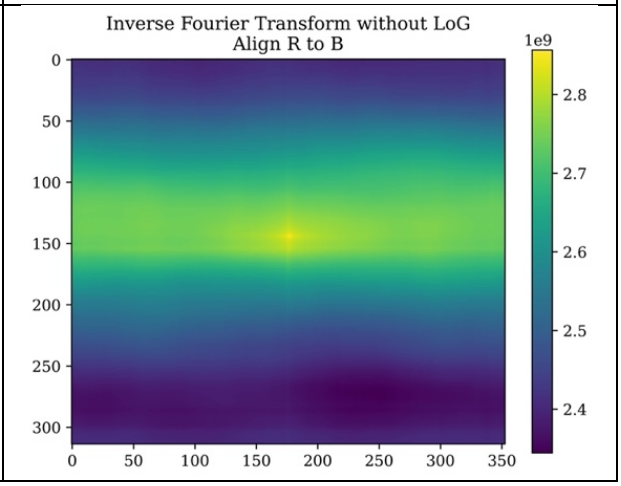
00153v.jpg

	LoG Preprocessing	No LoG Preprocessing
Aligned Images (G)		

Inverse FT Align B to G		
Inverse FT Align R to G		

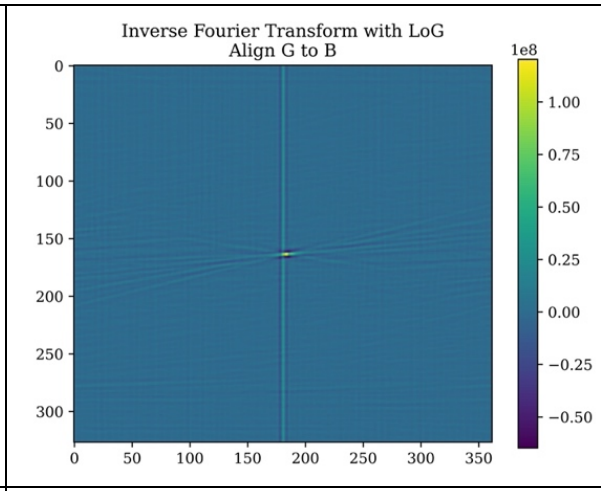
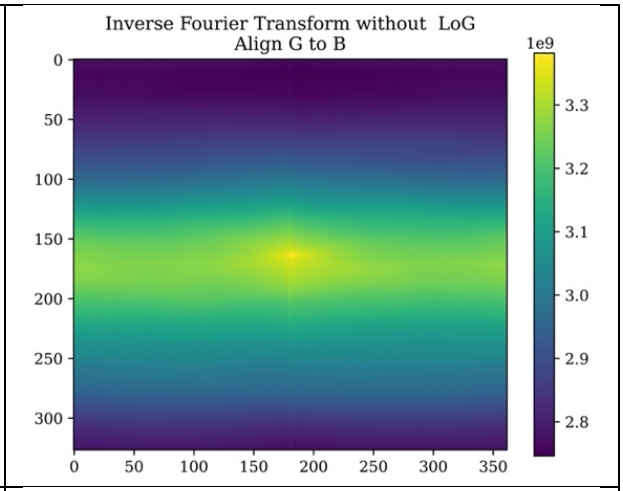
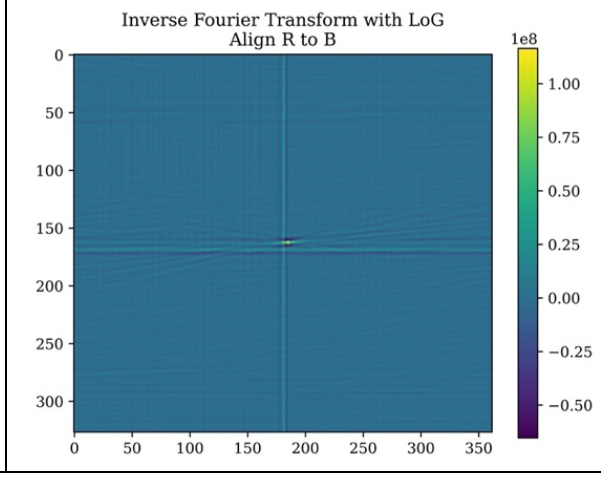
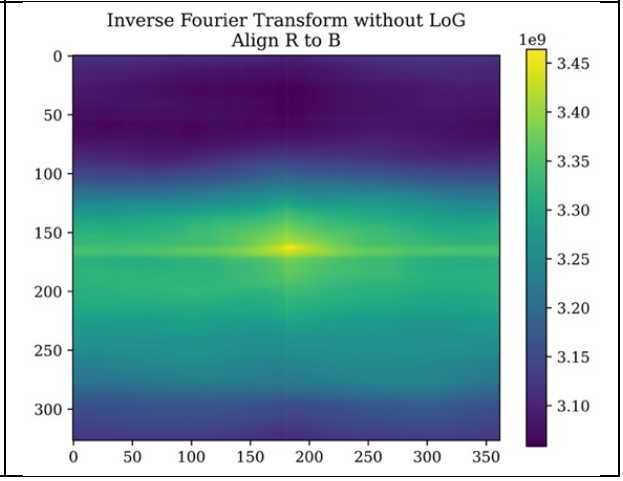
00351v.jpg

	LoG Preprocessing	No LoG Preprocessing
Aligned Images (B)		

Inverse FT Align G to B		
Inverse FT Align R to B		

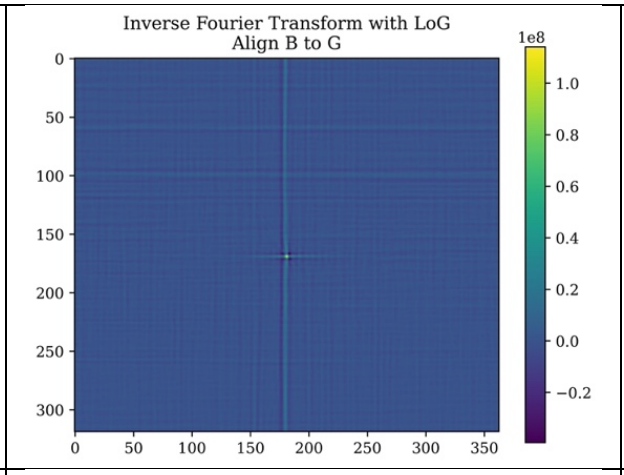
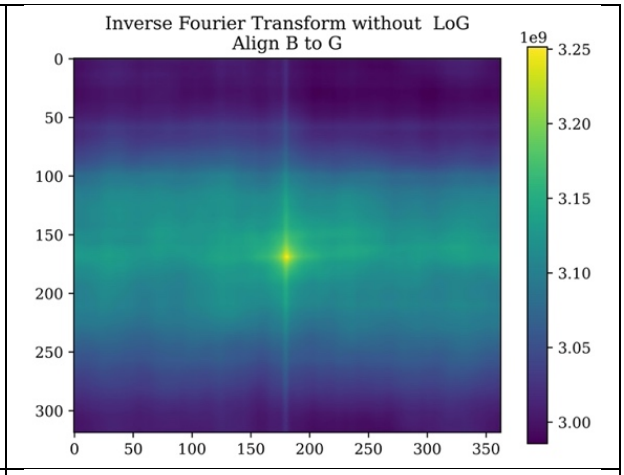
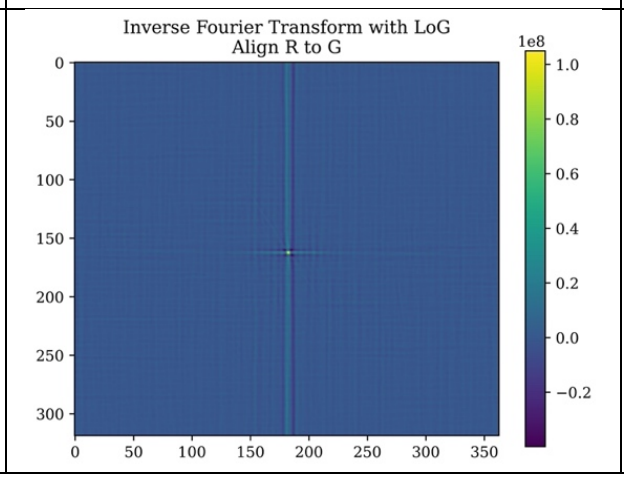
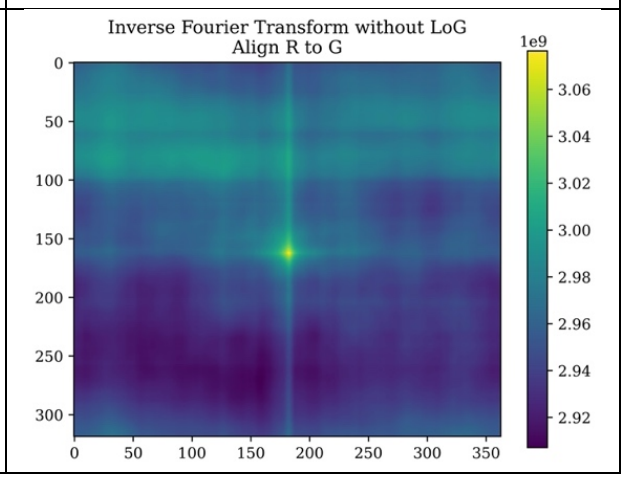
00398v.jpg

	LoG Preprocessing	No LoG Preprocessing
Aligned Images (B)		

Inverse FT Align G to B		
Inverse FT Align R to B		

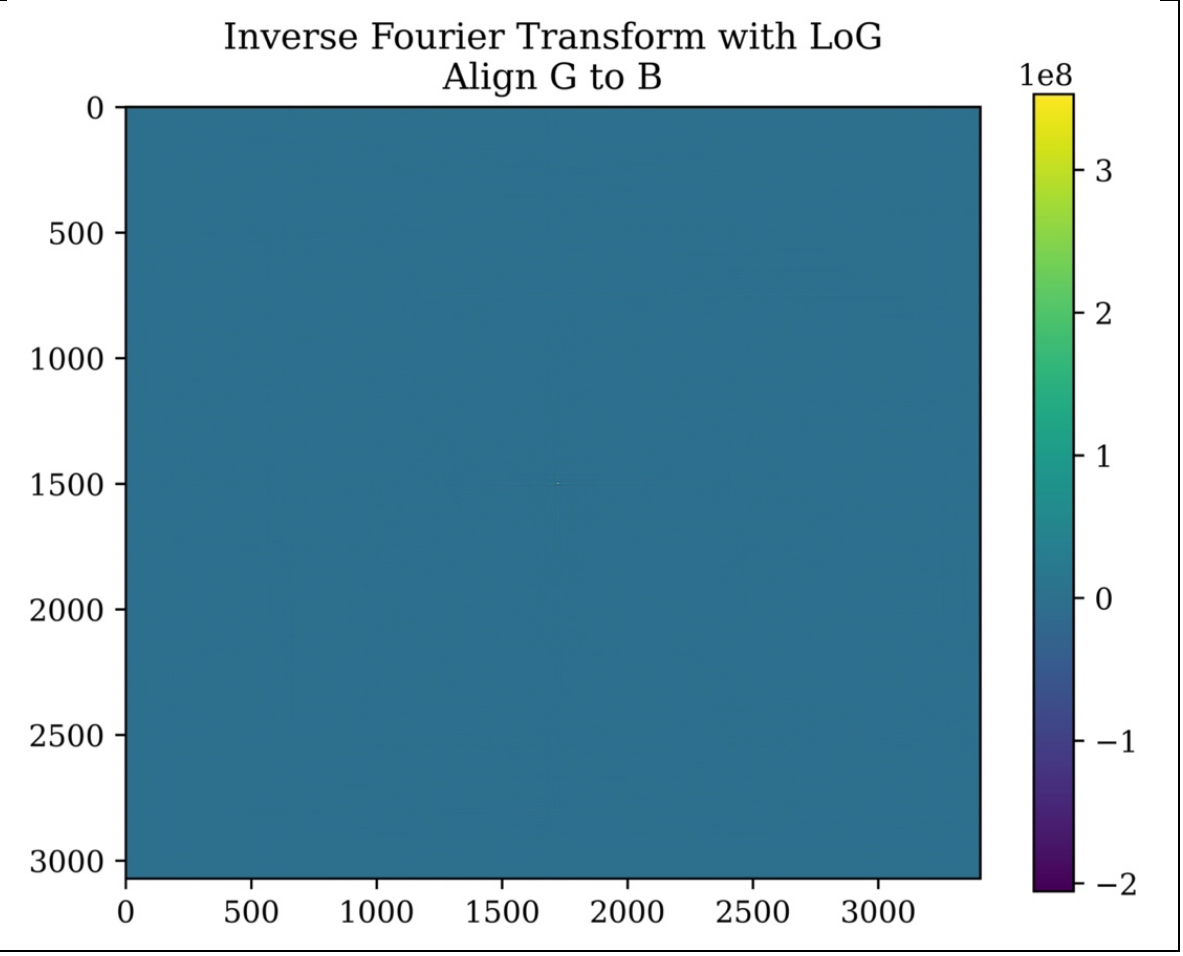
01112v.jpg

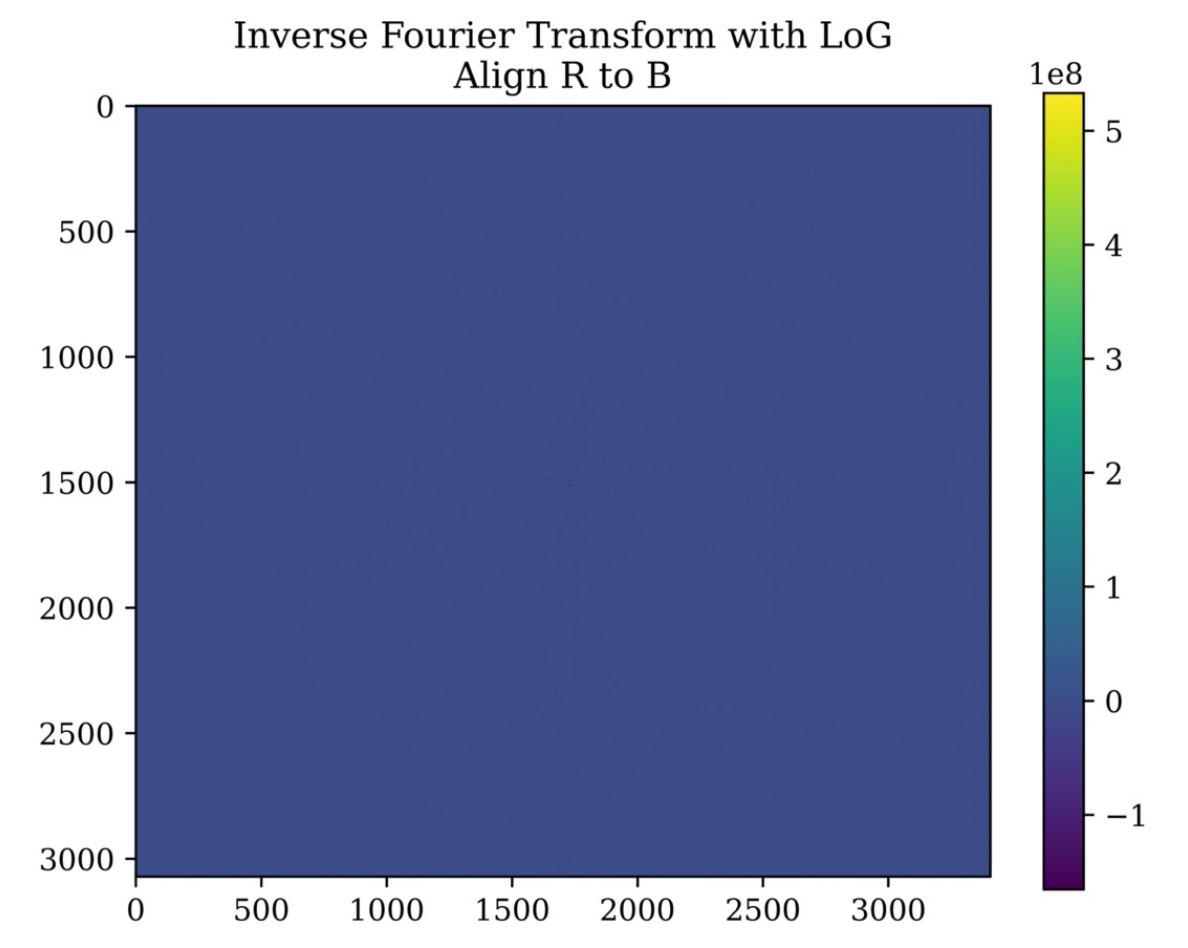
	LoG Preprocessing	No LoG Preprocessing
Aligned Images (G)		

Inverse FT Align B to G		
Inverse FT Align R to G		

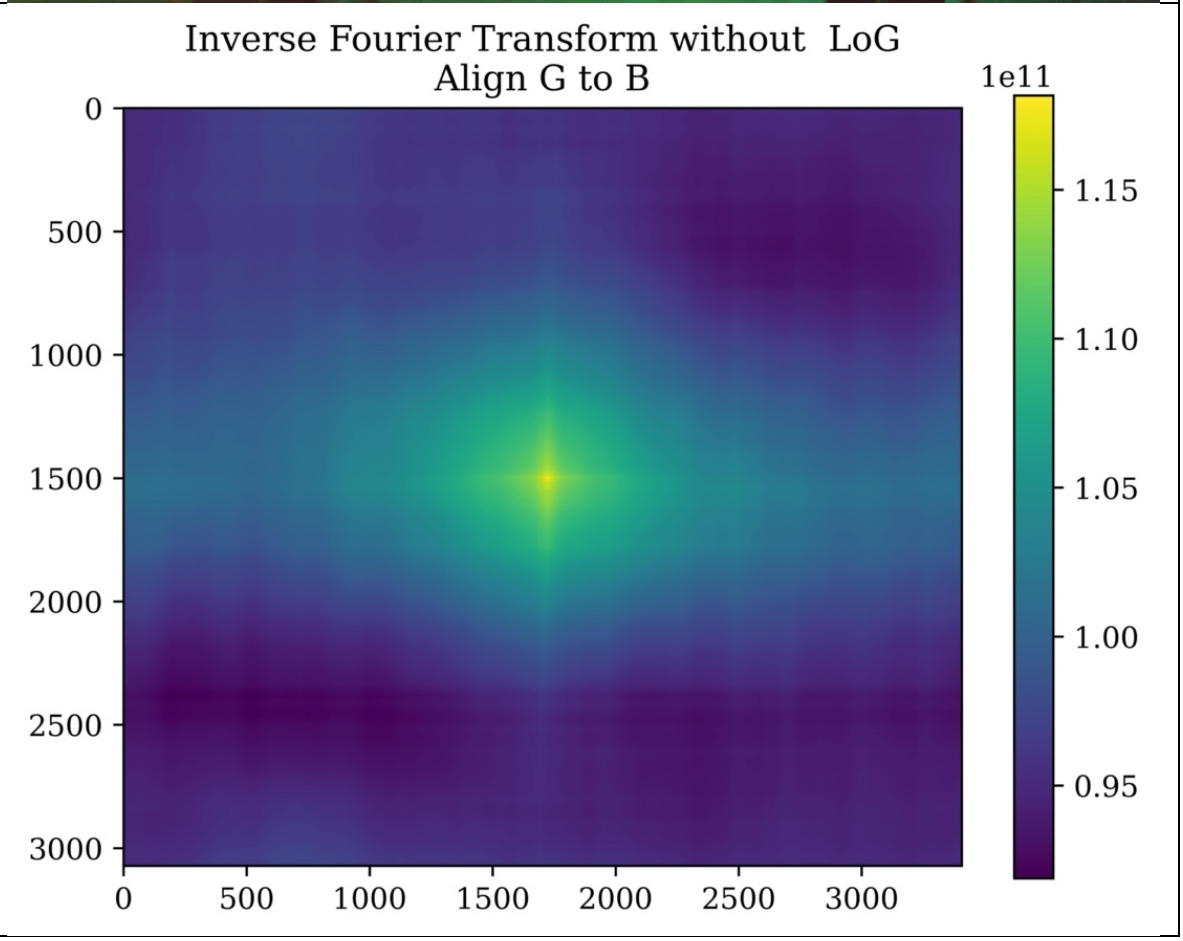
01047u.tif

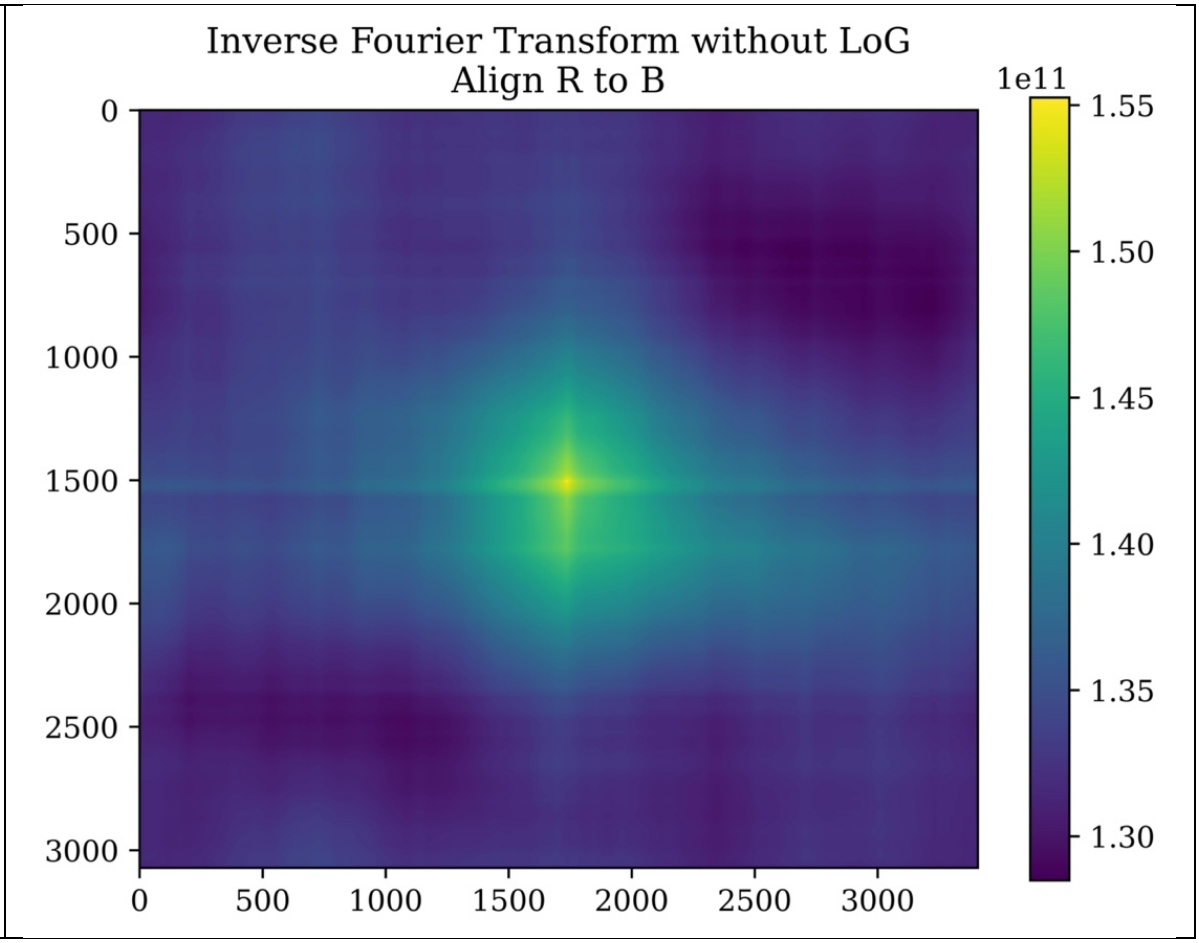
	LoG Preprocessing

Aligned Image (B)	
Inverse FT Align G to B	

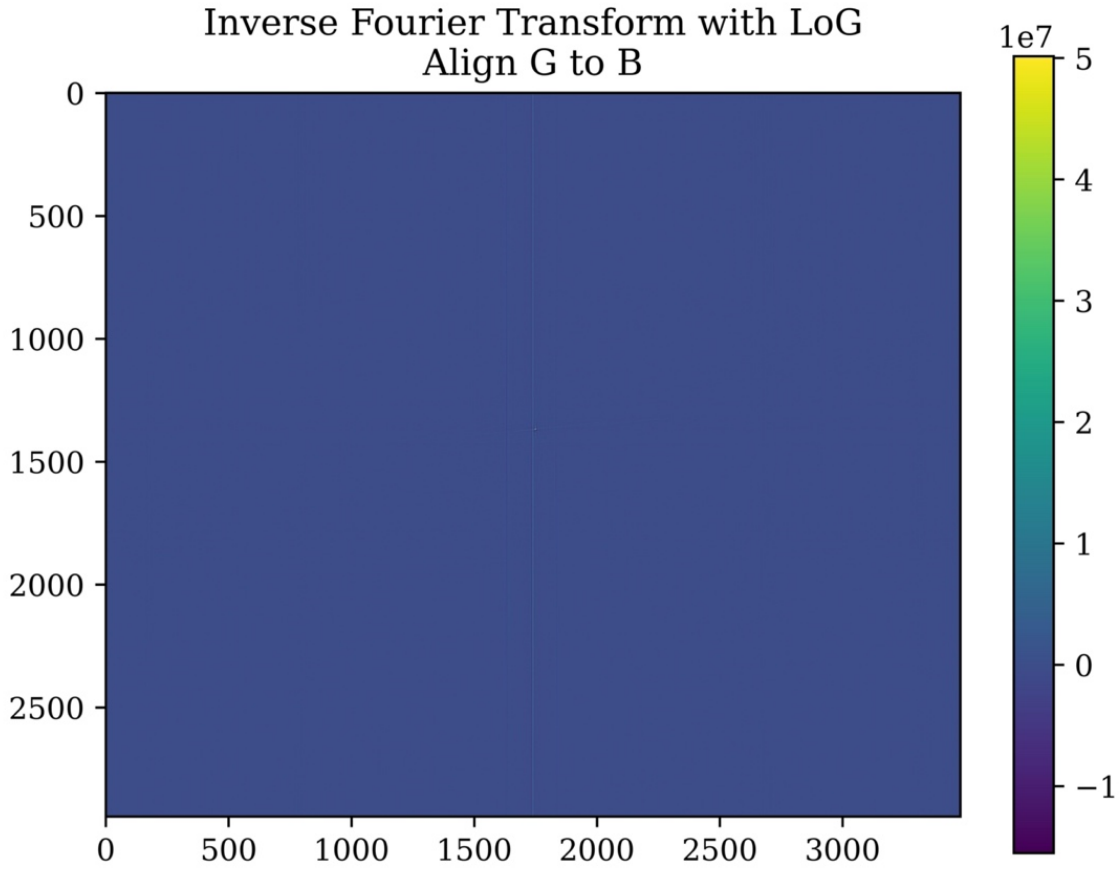
Inverse FT Align R to B	

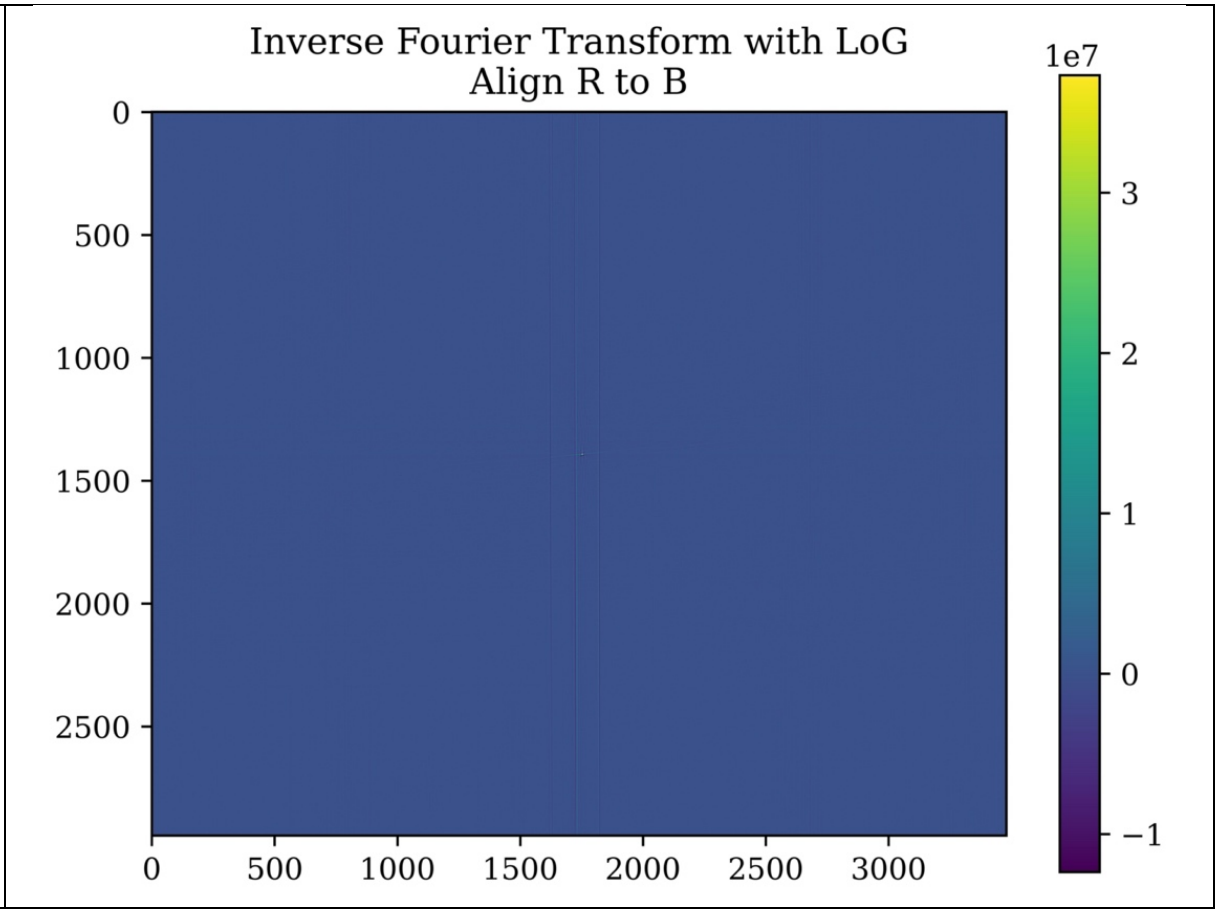
	No LoG Preprocessing

Aligned Image (B)	
Inverse FT Align G to B	

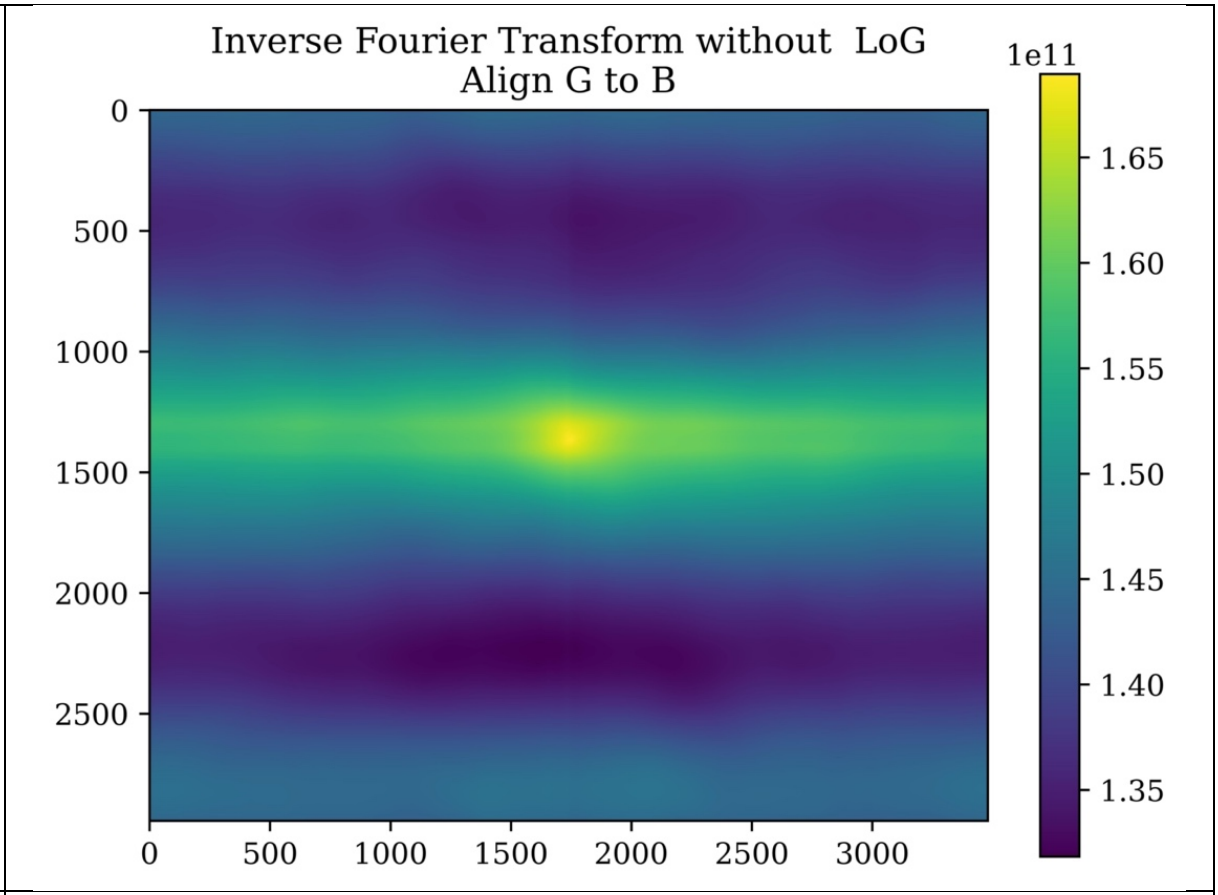
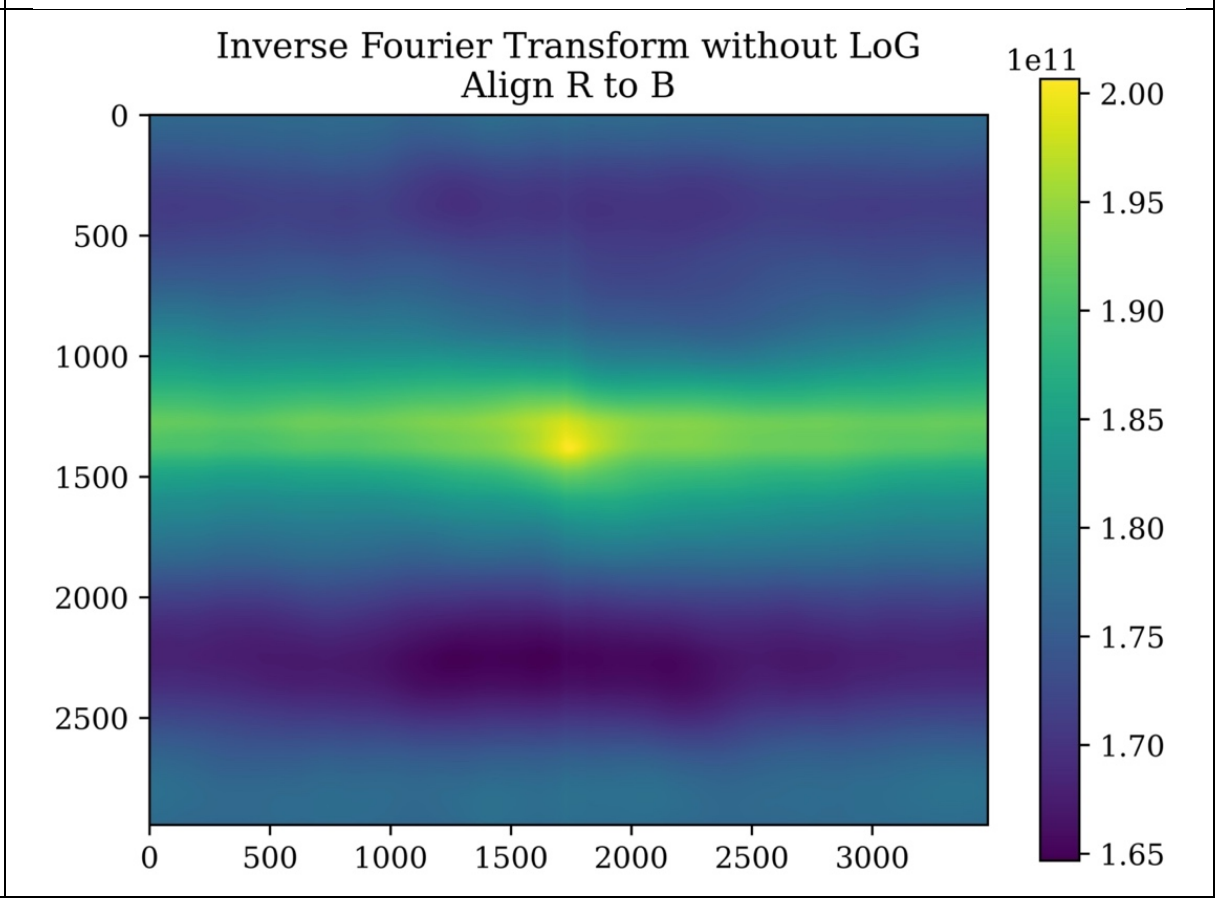
Inverse FT Align R to B	

01657u.tif

	LoG Preprocessing
Aligned Image (B)	
Inverse FT Align G to B	

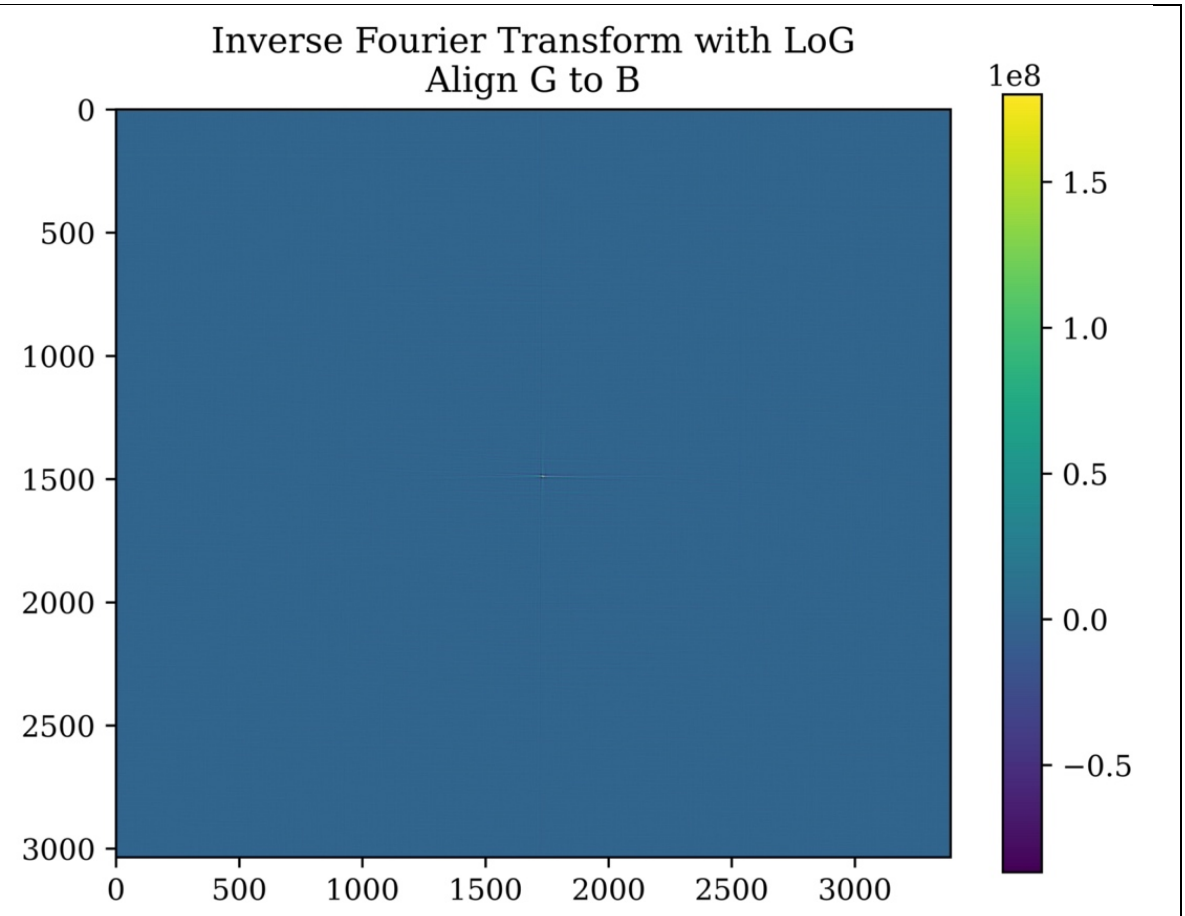
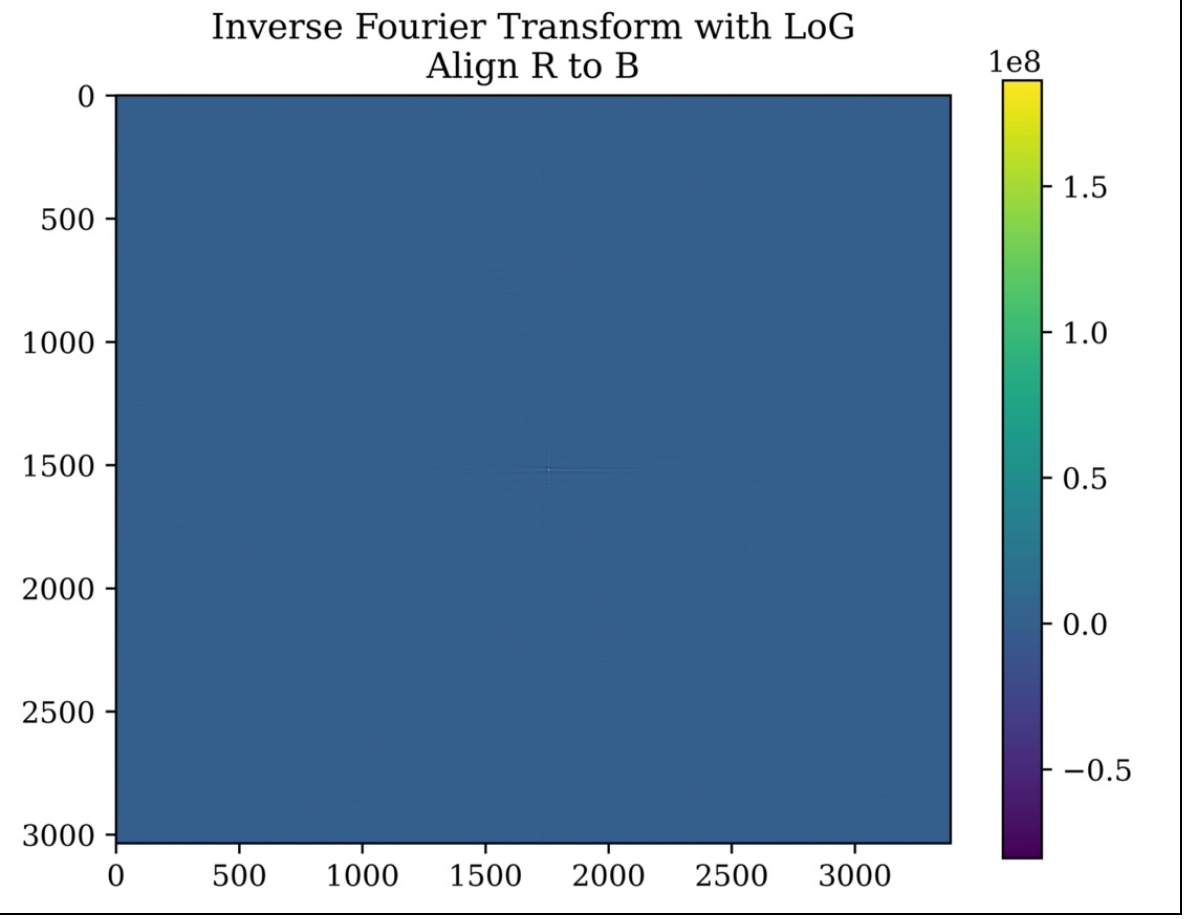
Inverse FT Align R to B	

	No LoG Preprocessing
Aligned Image (B)	

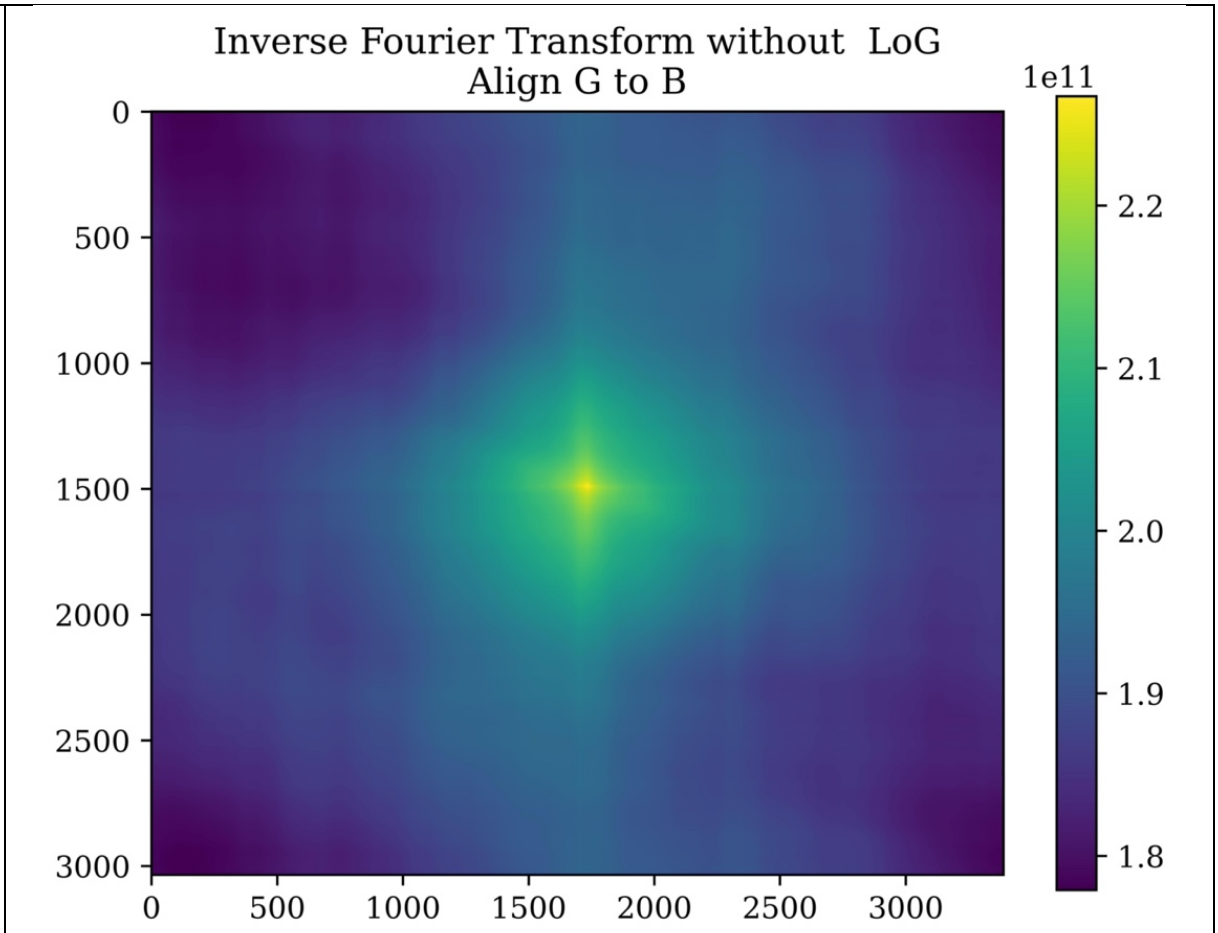
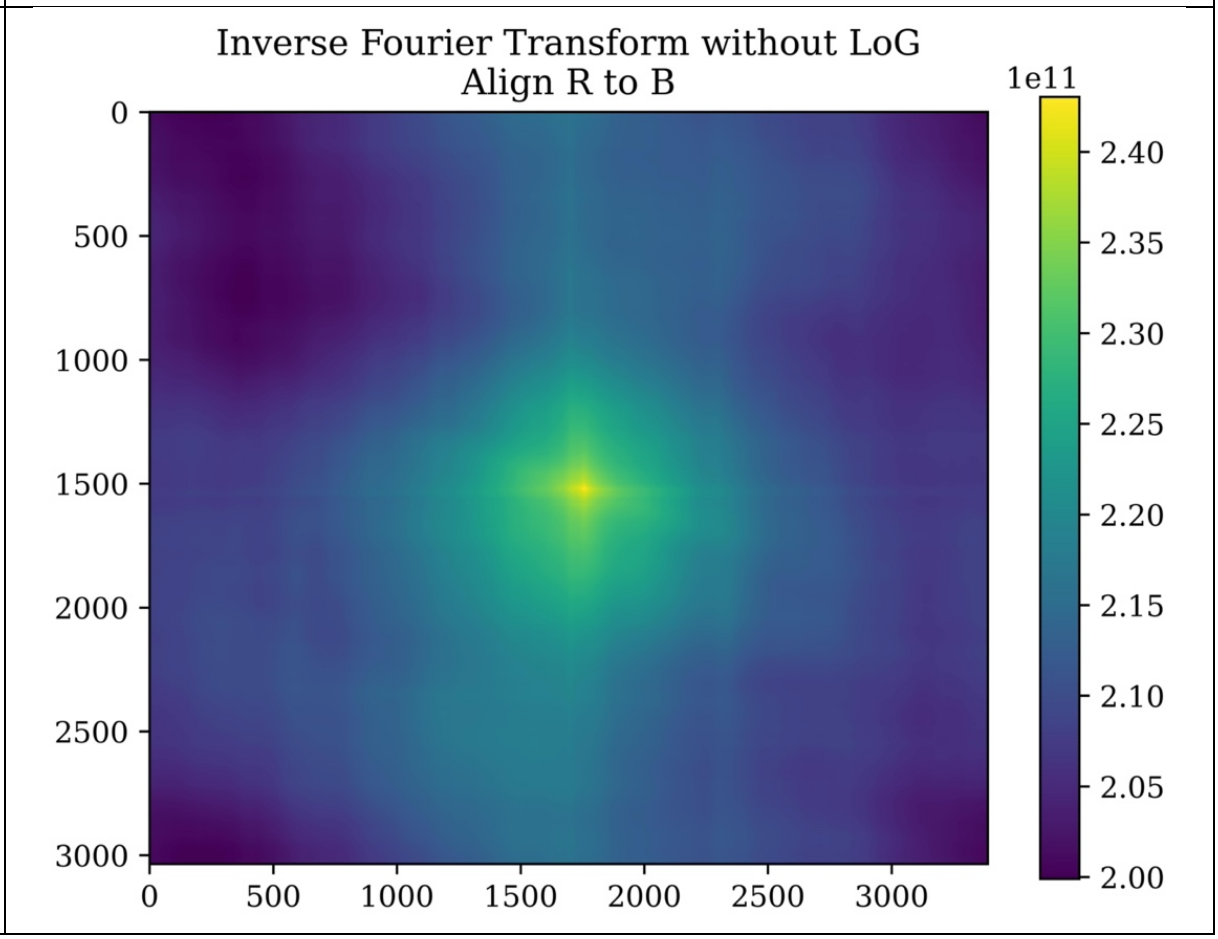
Inverse FT Align G to B	
Inverse FT Align R to B	

01861a.tif

	LoG Preprocessing
Aligned Image (B)	

Inverse FT Align G to B	
Inverse FT Align R to B	

	No LoG Preprocessing
Aligned Image (B)	

Inverse FT Align G to B	
Inverse FT Align R to B	

C: Discussion and Runtime Comparison

- Discussion of any preprocessing you used on the color channels to improve alignment and how it changed the outputs

  Initially, I implemented and tried Laplacian of Gaussian filter by myself, and the values of the 3x3 kernel is the following: [[0, -1, 0], [-1, 4, -1], [0, -1, 0]]. However, the final preprocessing I used on the color channels to improve alignment is the Laplacian of Gaussian filter. In my implementation, I first applied cv2.GaussianBlur, and then applied cv2.Laplacian to highlight all the edges in the color channels. However, the final results with preprocessing seem to be almost identical to the results without preprocessing. However, the inverse fourier transform outputs of the color channels with preprocessing is different from that of the color channels without preprocessing. The maximum response is easy to see and locate in the inverse fourier transform outputs of the color channels with preprocessing because it is mostly a single point with large magnitude of response. However, the maximum response of the inverse fourier transform outputs of the color channels without preprocessing has some noise in it which do not highlight the peak value as clear as the inverse fourier transform outputs of the color channels with preprocessing.

- 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?

  The fourier-based alignment runtime for the low-resolution images and the high-resolution images are roughly 1.35 seconds and 15.35 seconds. The single scale alignment takes about 1.83 seconds on the low-resolution images and 253.48 seconds on the high-resolution images. The multiscale alignment on the high-resolution takes about 75.63 seconds which is ~70% faster than the single scale alignment on the high-resolution images. The fourier-based alignment runtime for high-resolution images is approximately 75% faster than the multiscale alignment and is roughly 94% faster than the single scale alignment.

Part 2 Scale-Space Blob Detection:

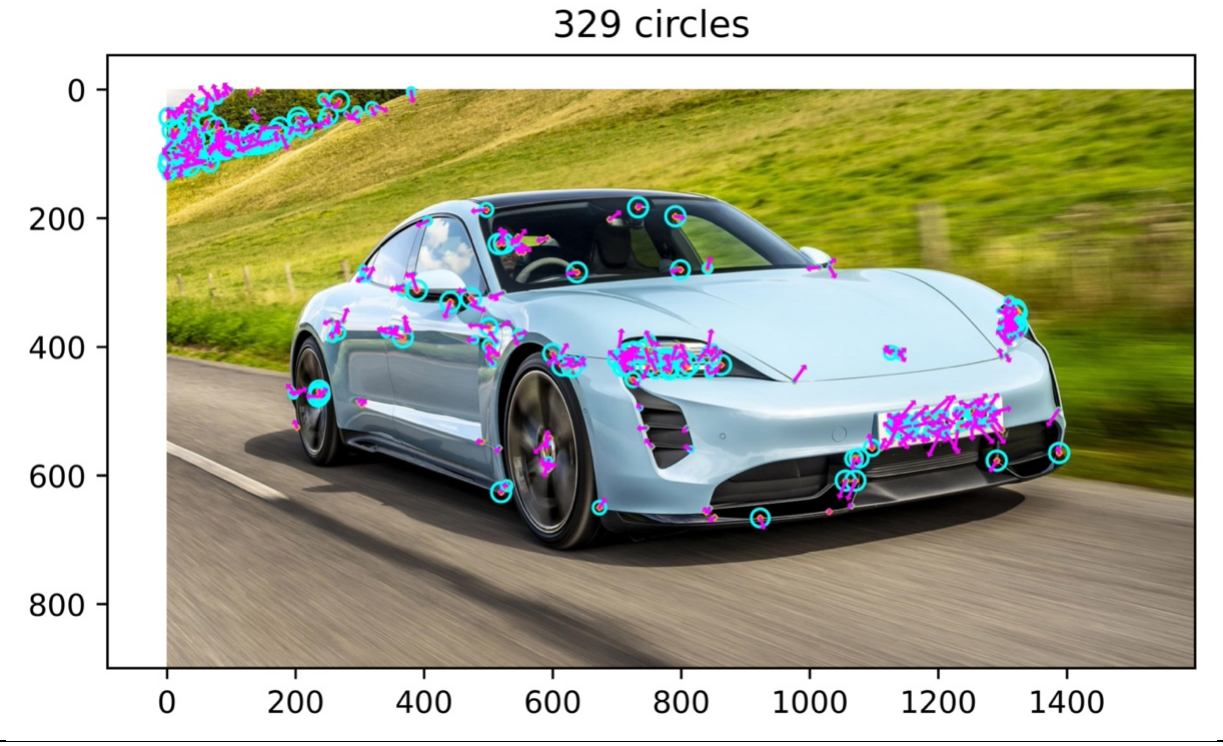
You will provide the results for ***4 different examples chosen by your own***:

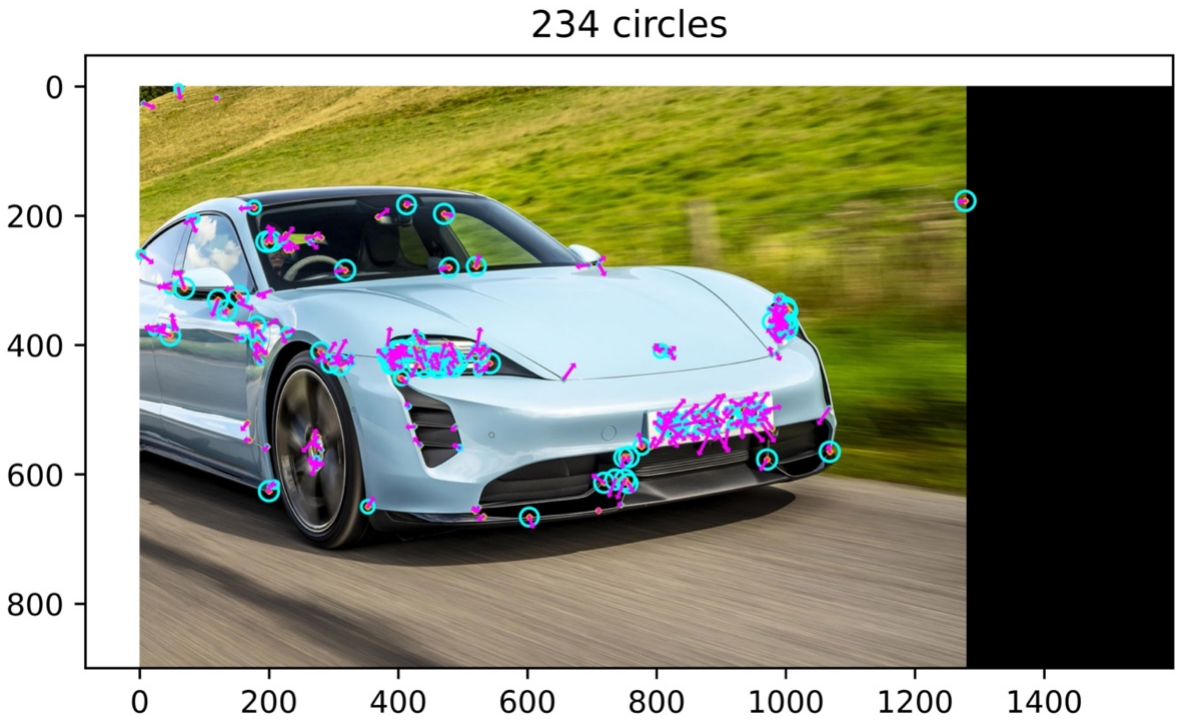
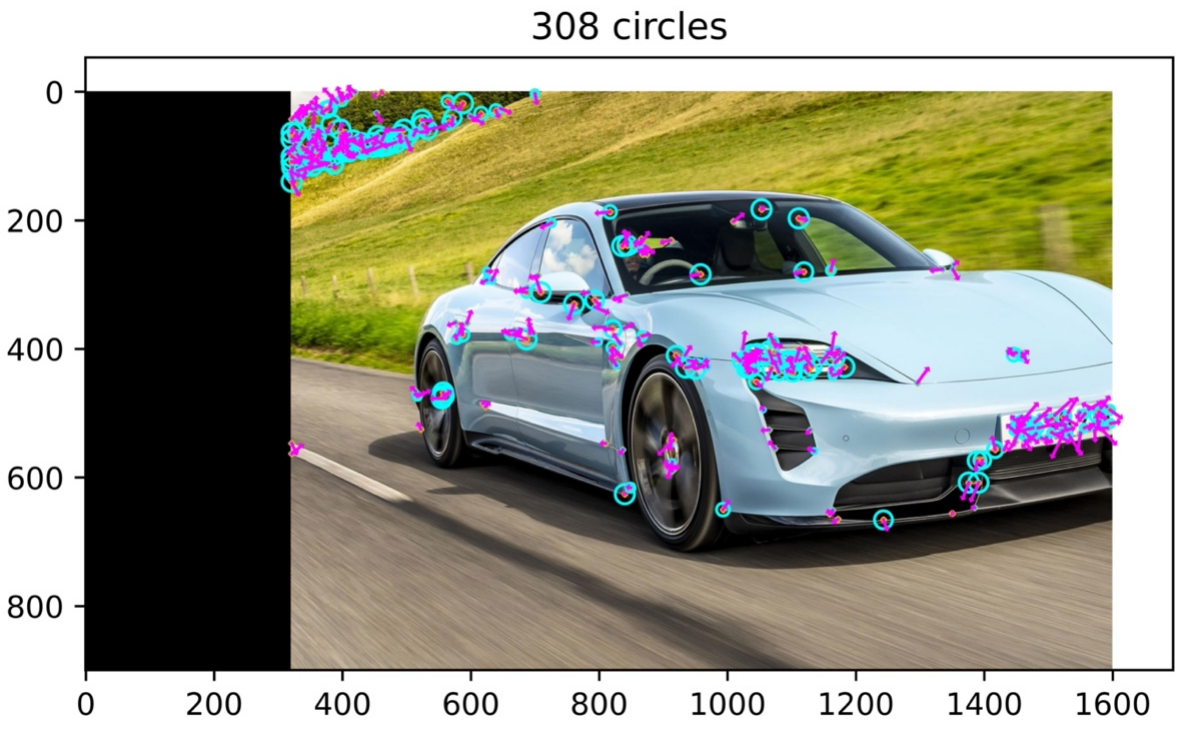
- Original image
- Each of the five modified images (shift, rotate, scale)

You will provide the following as further discussion overall:

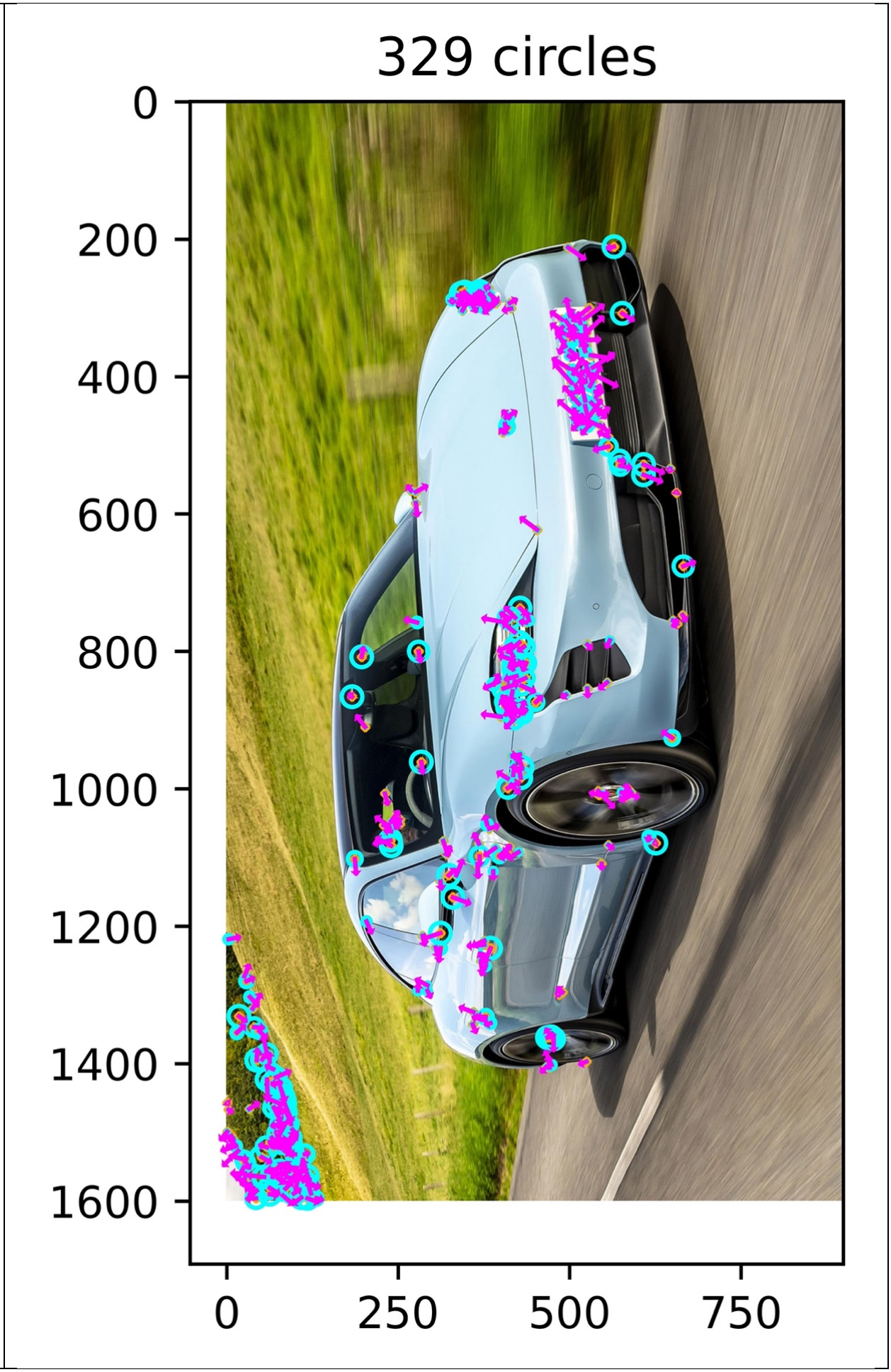
- Explanation of any "interesting" implementation choices that you made.

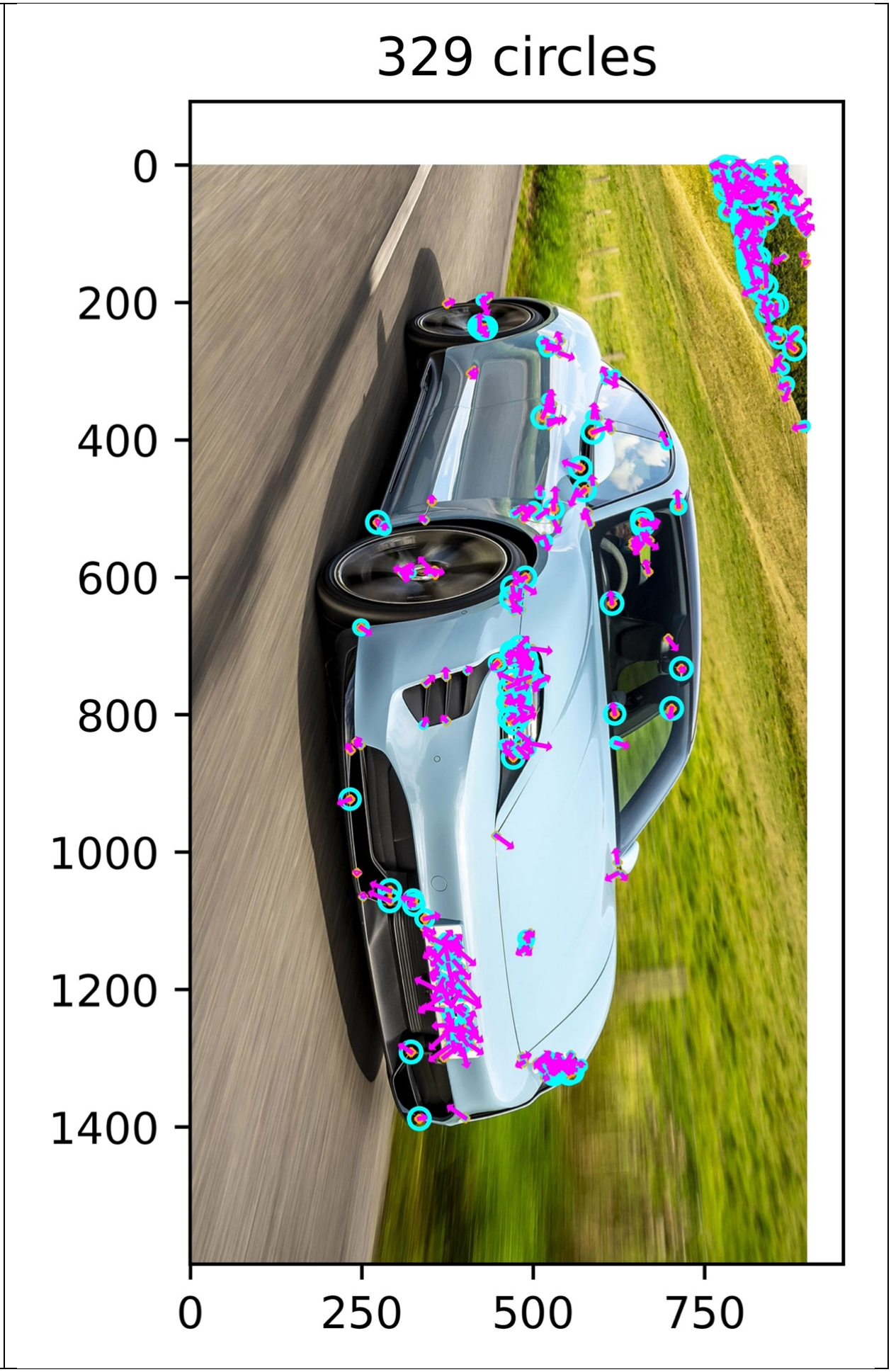
Example 1:

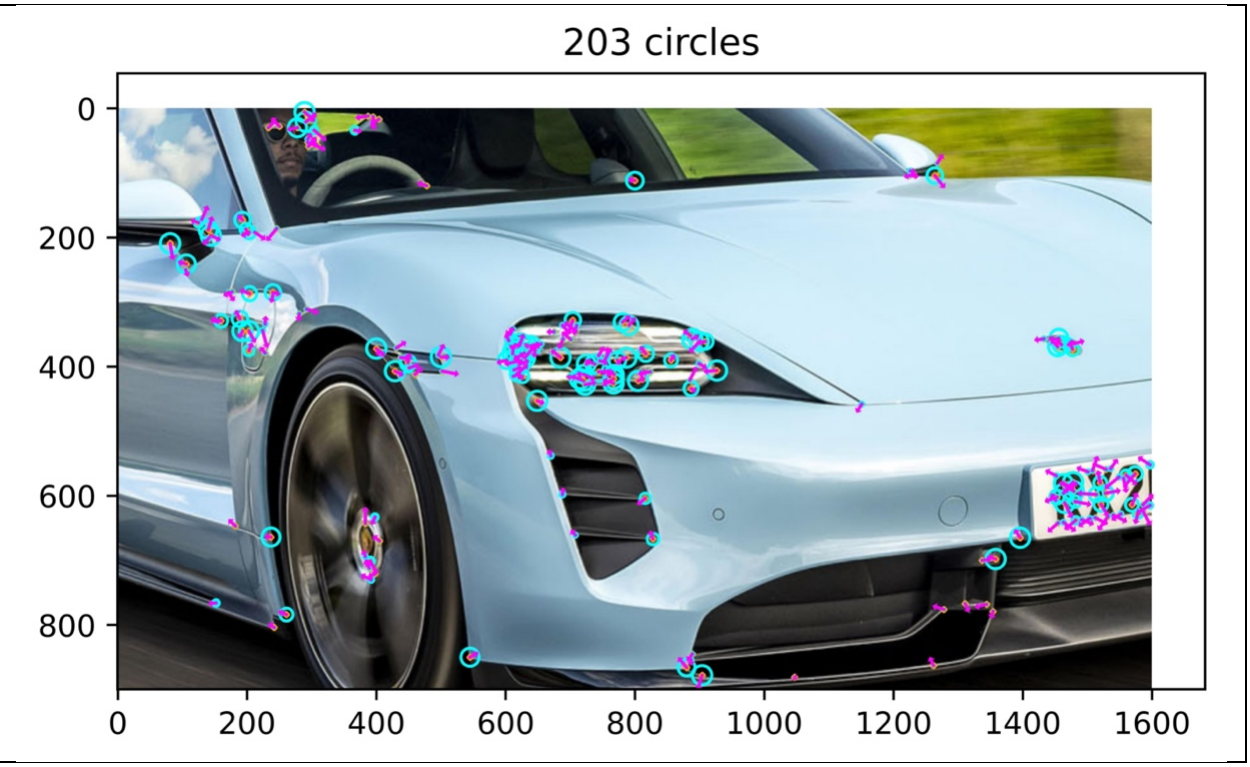
Image	
Base Image	

Shifted 20% Left	
Shifted 20% Right	

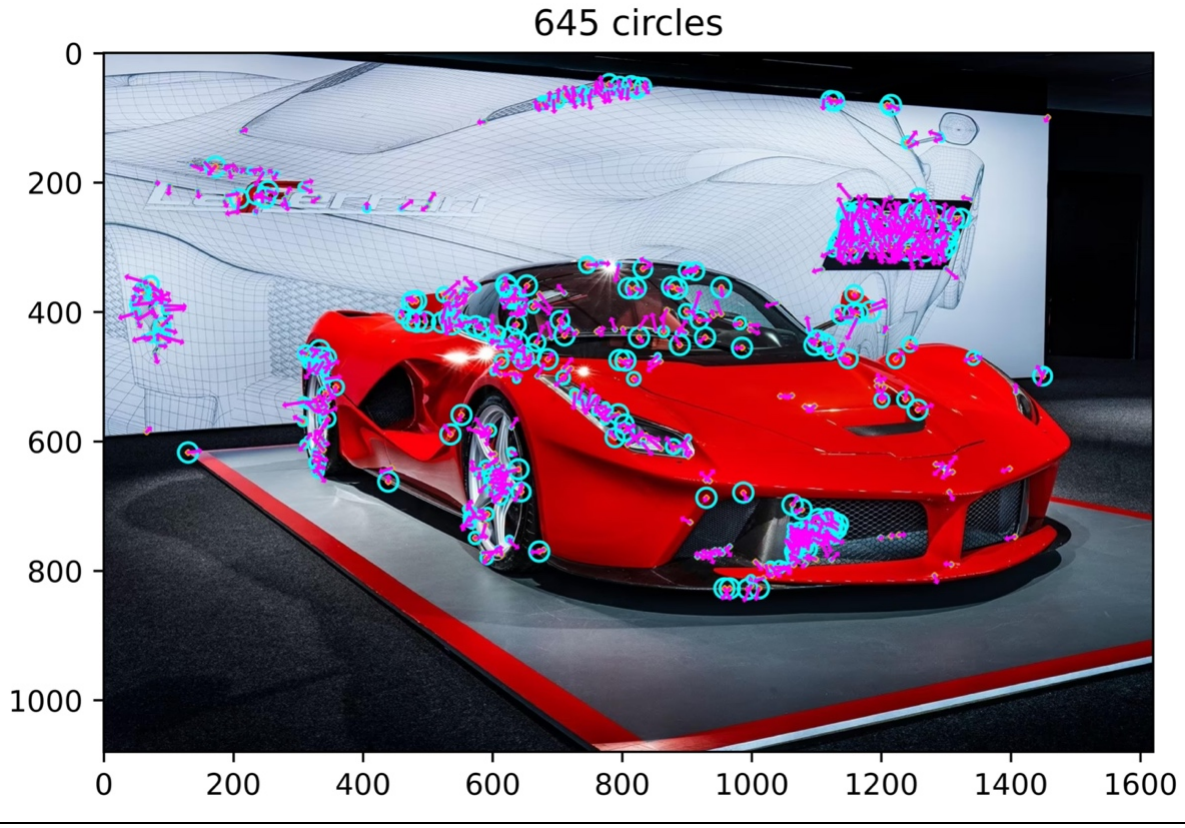
Rotated
90° CCW

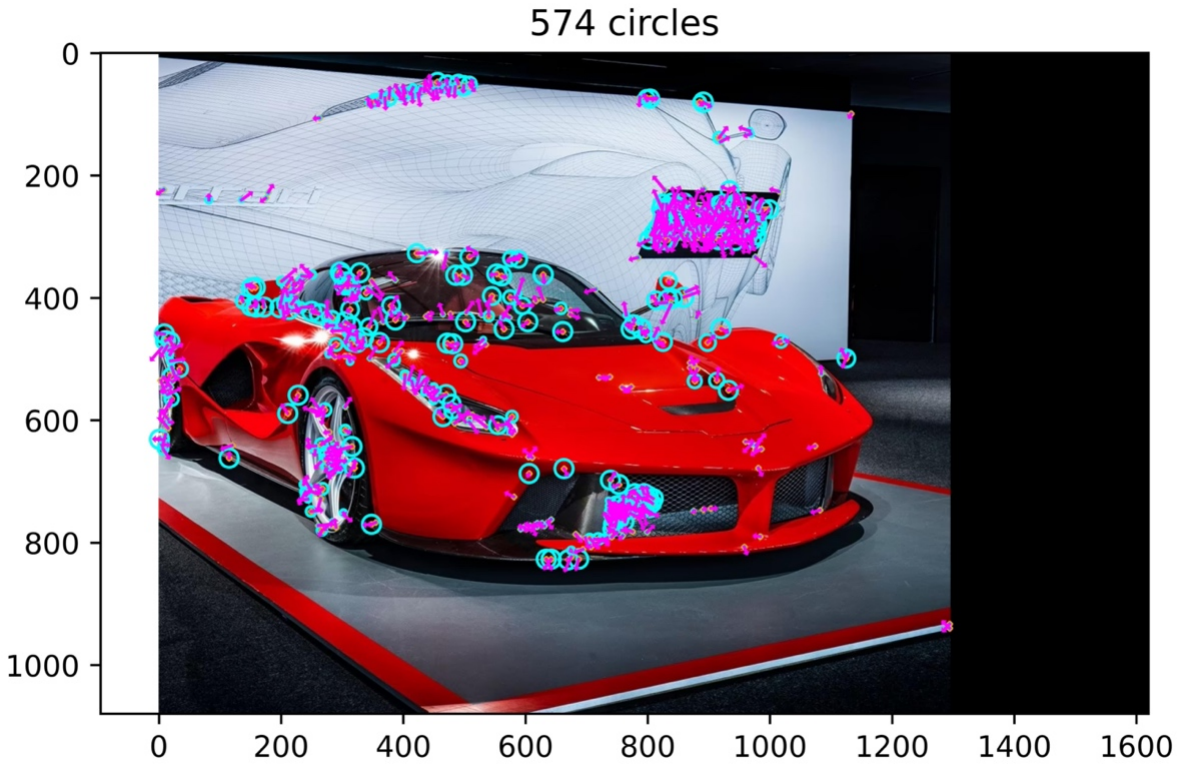
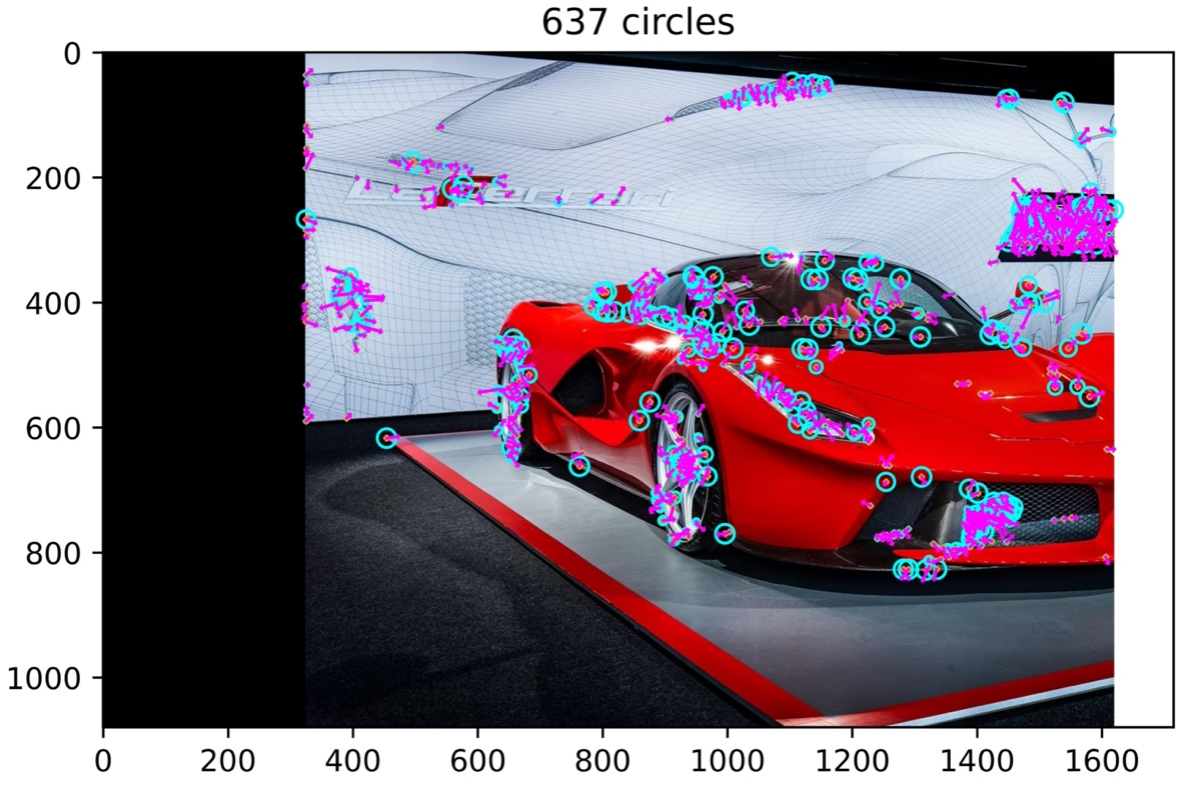


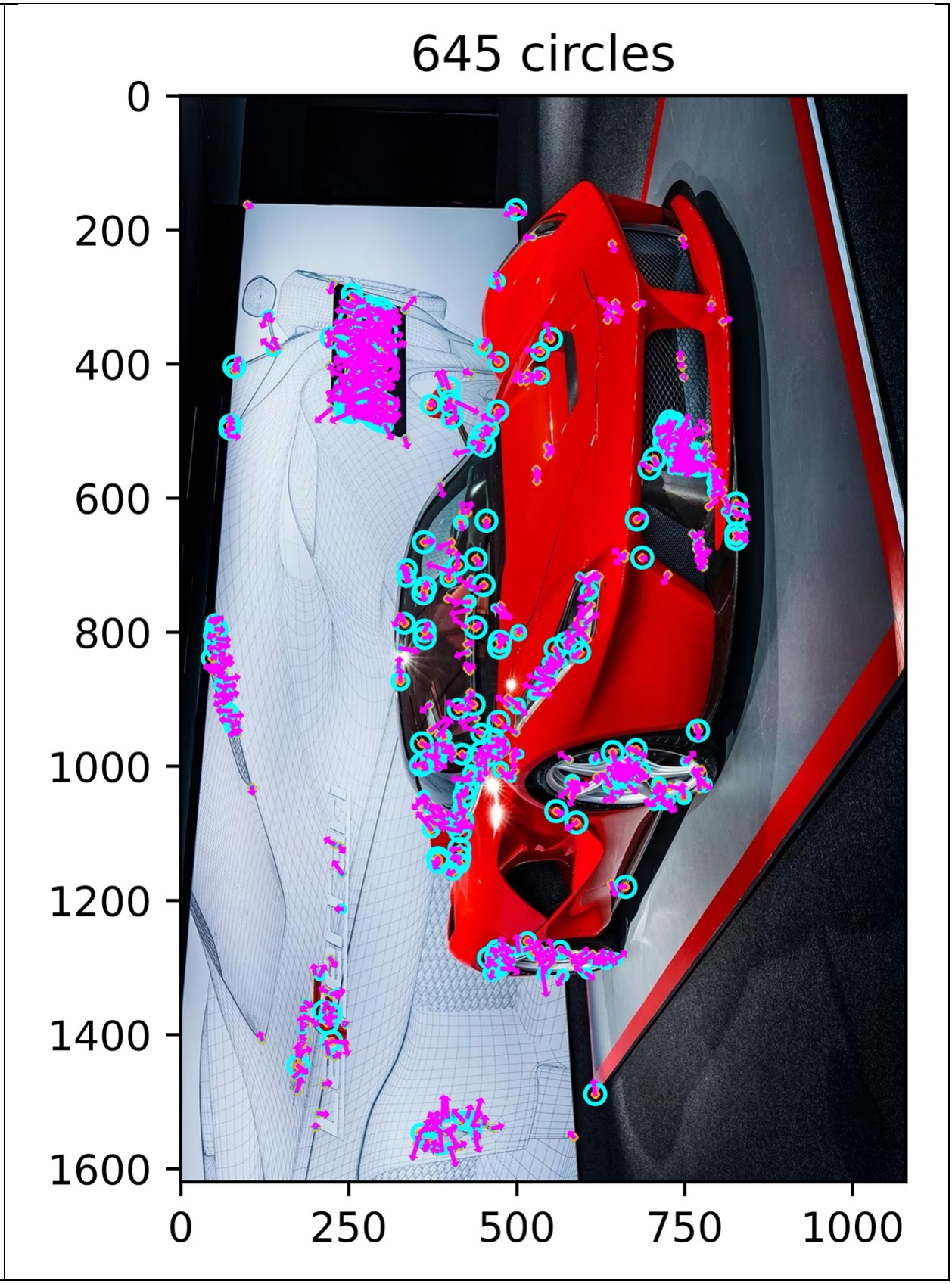
Rotated 90° CW	

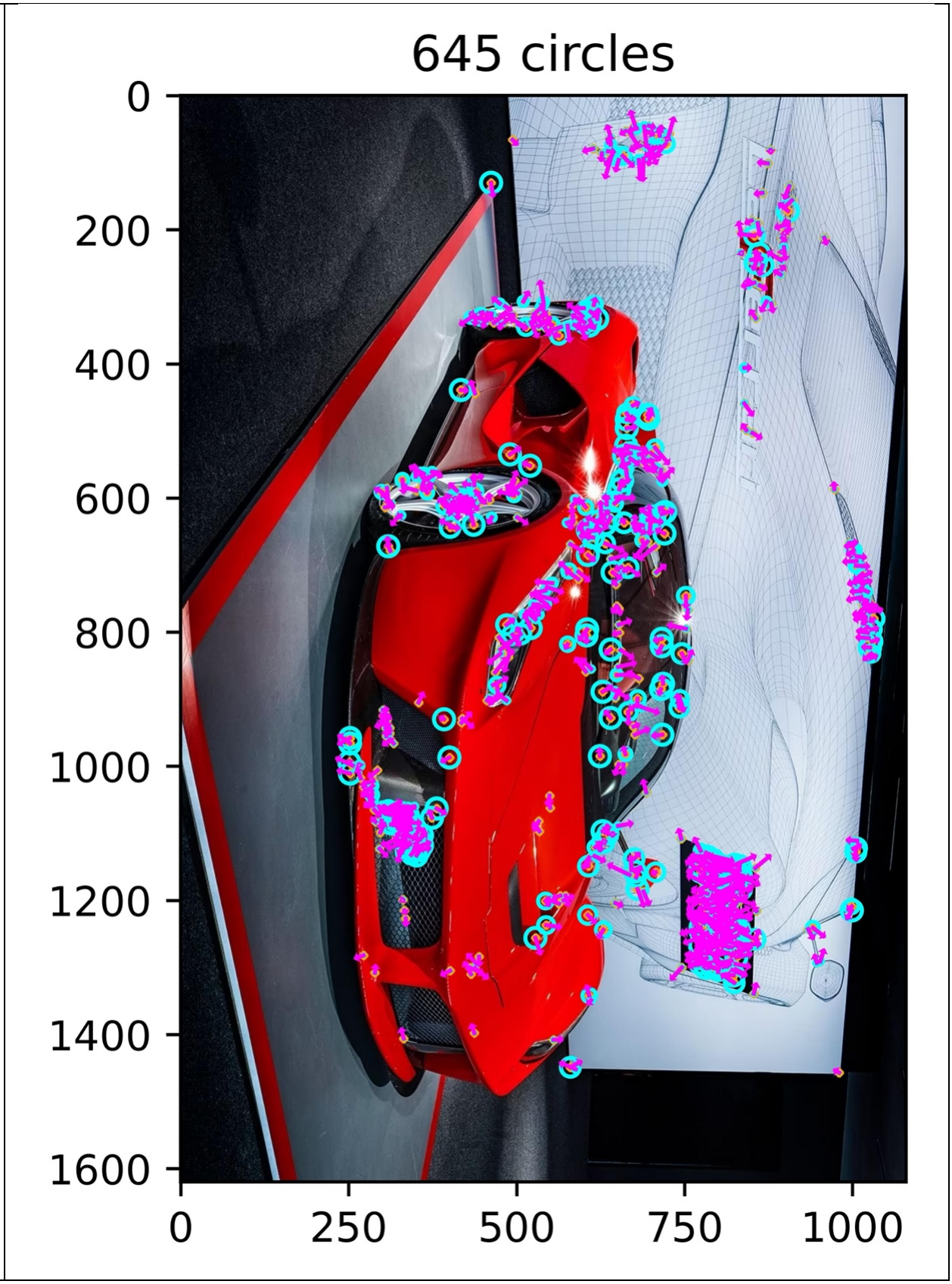
Enlarged by 2	

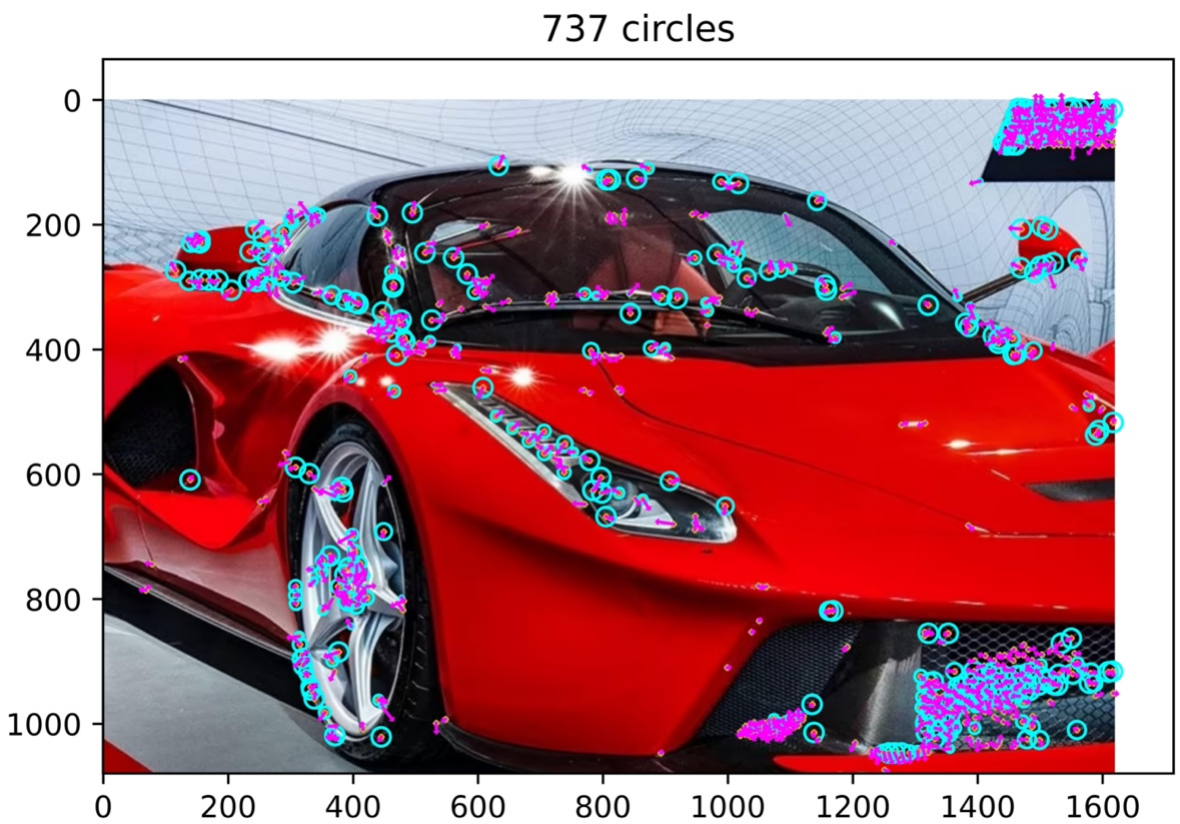
Example 2:

Image	
Base Image	

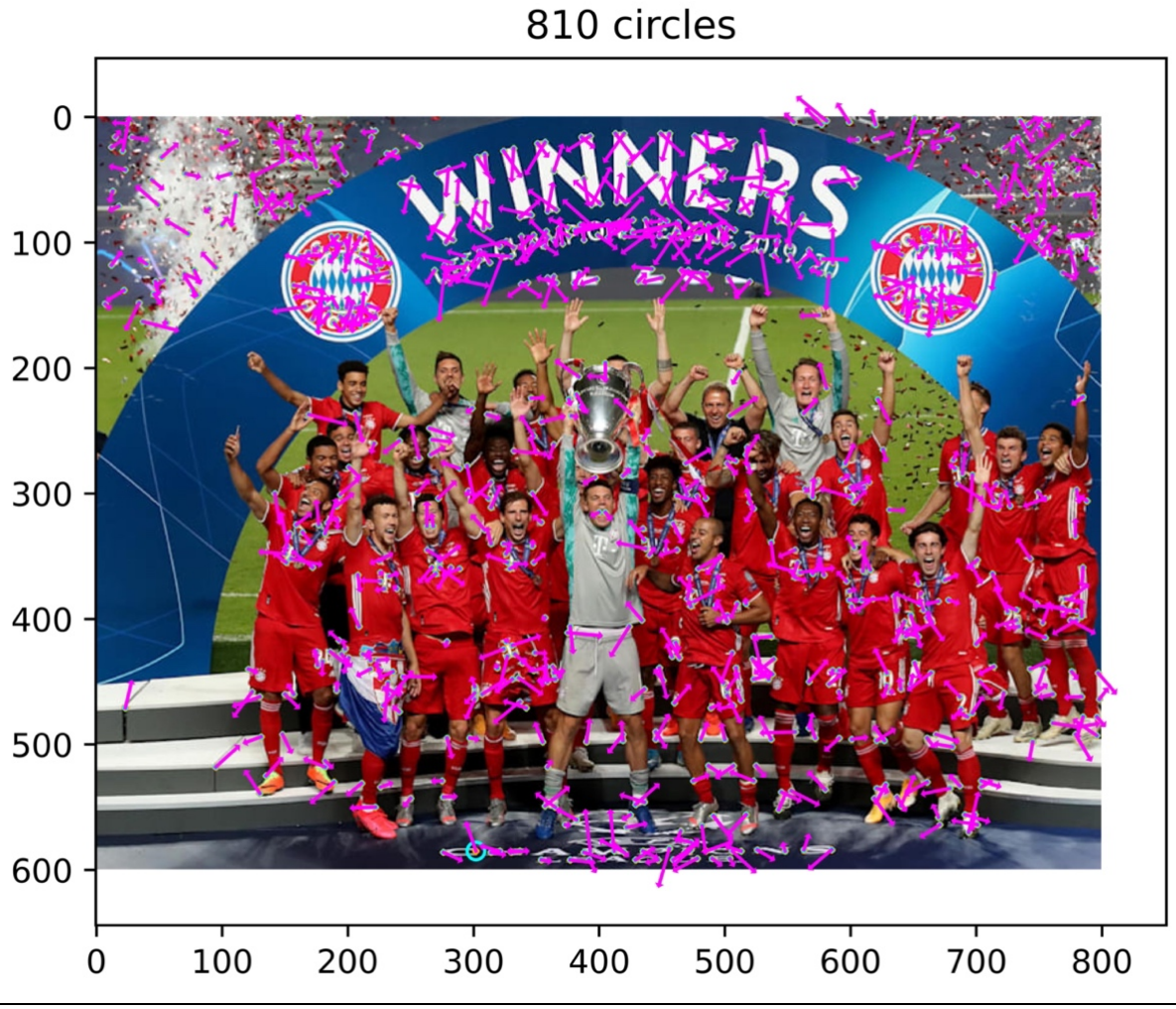
Shifted 20% Left	
Shifted 20% Right	

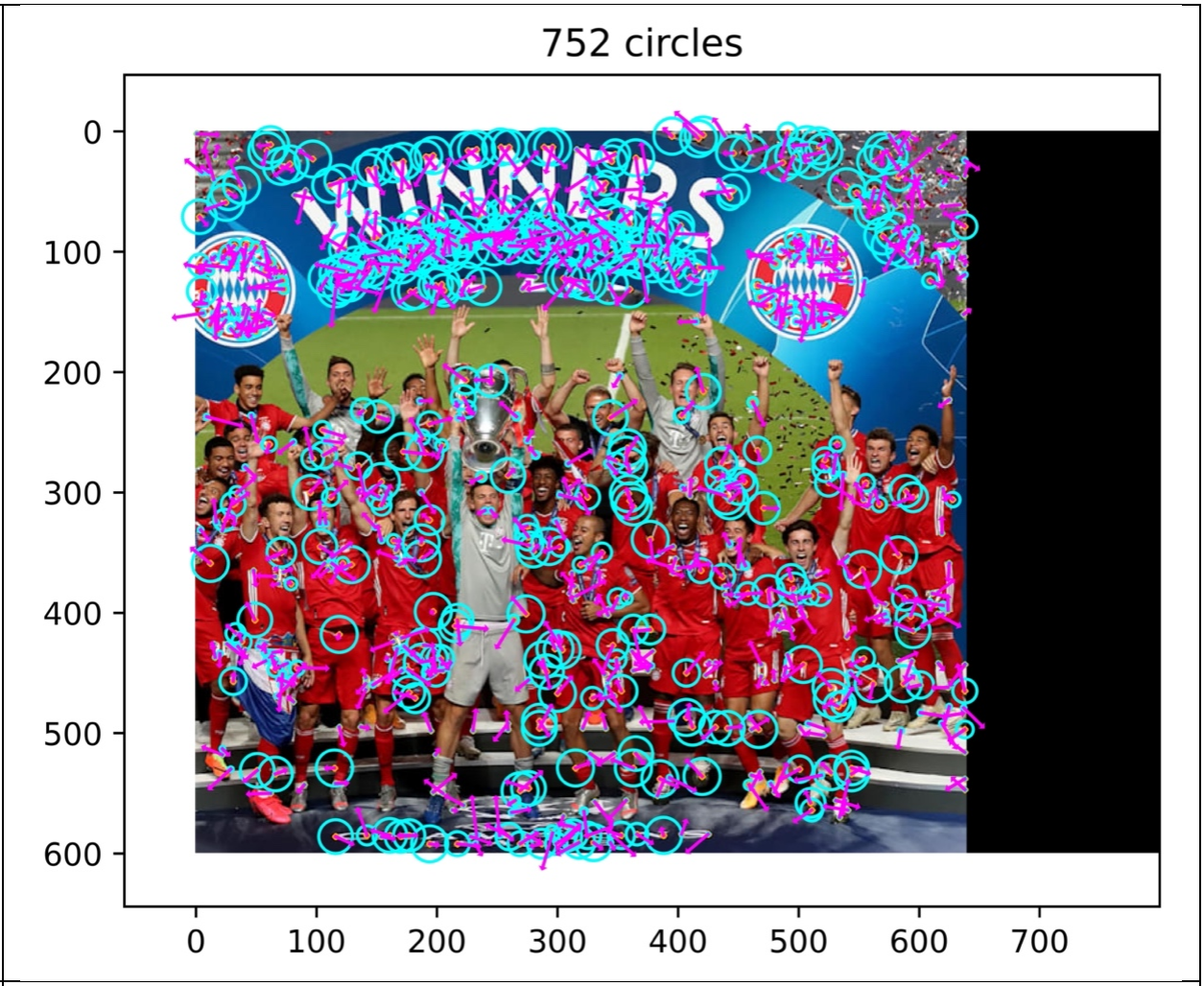
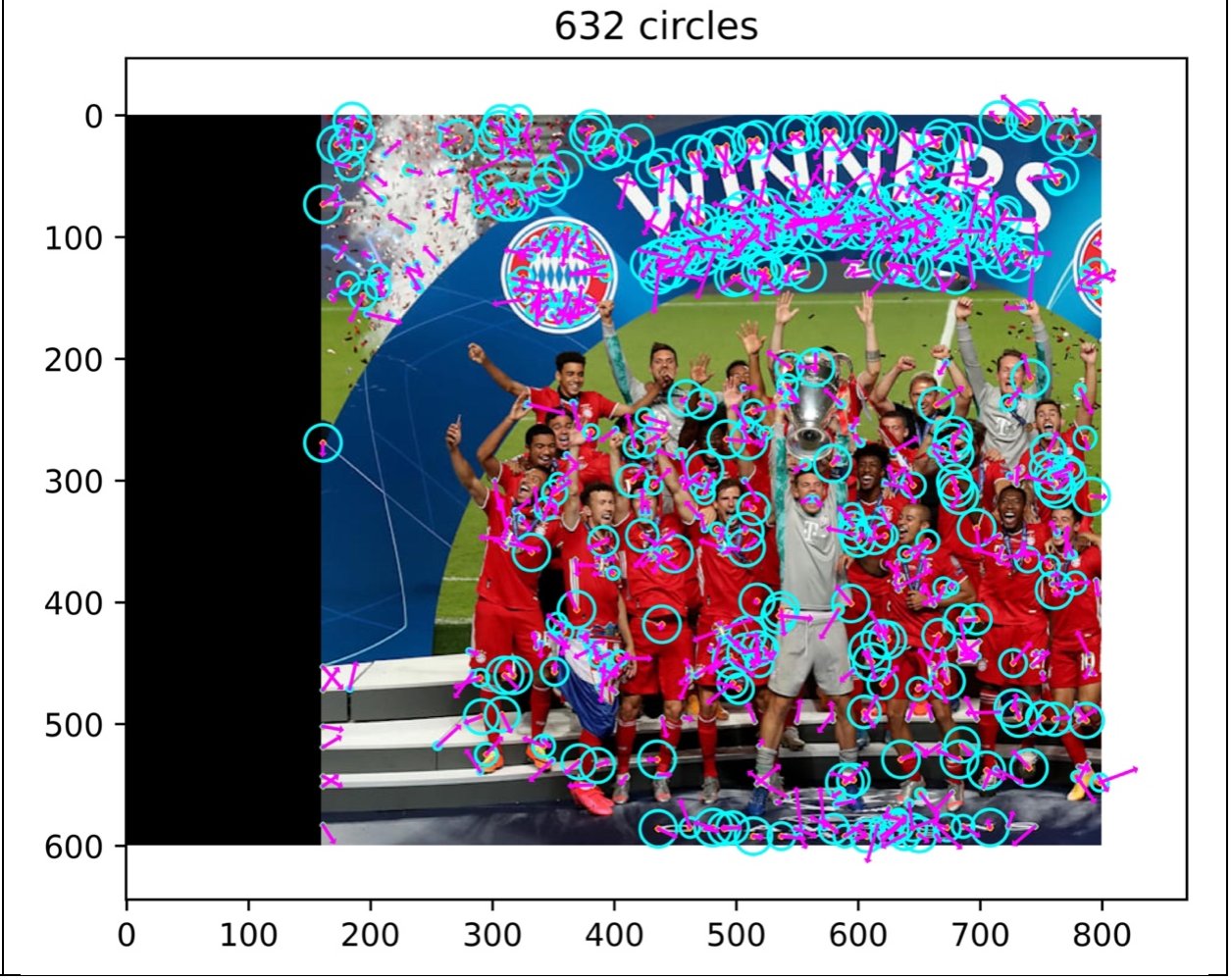
Rotated 90° CCW	

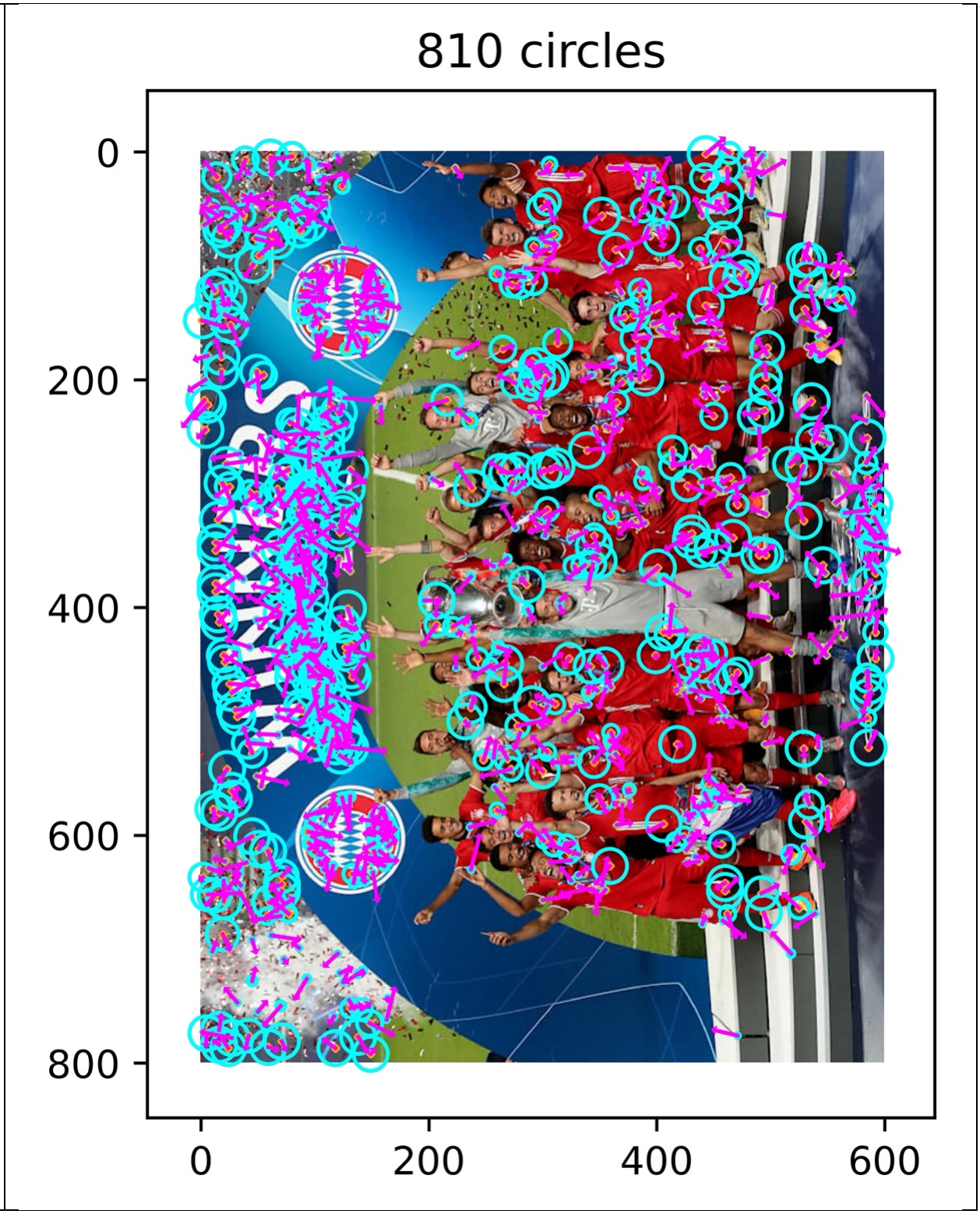
Rotated 90° CW	

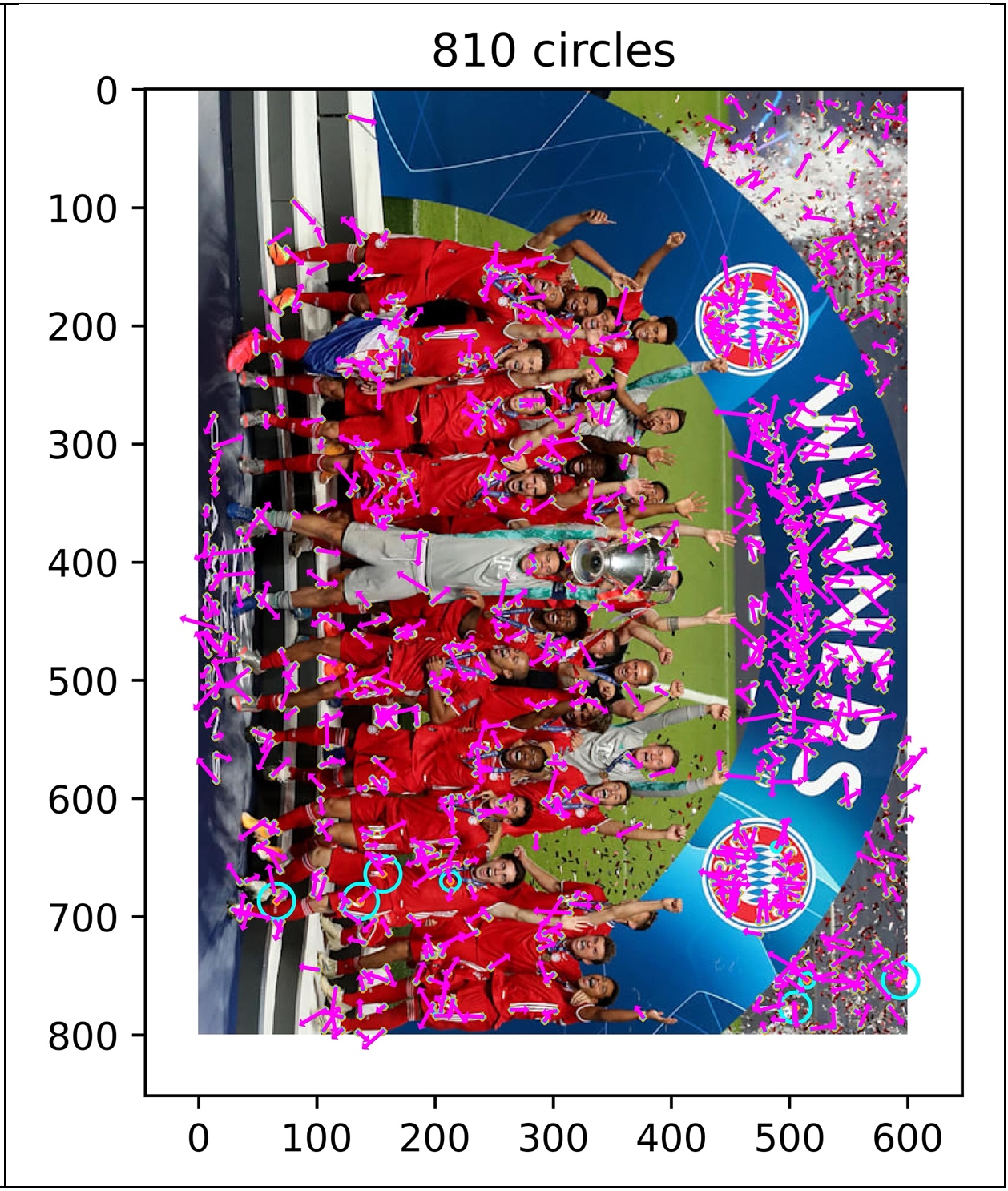
Enlarged by 2	

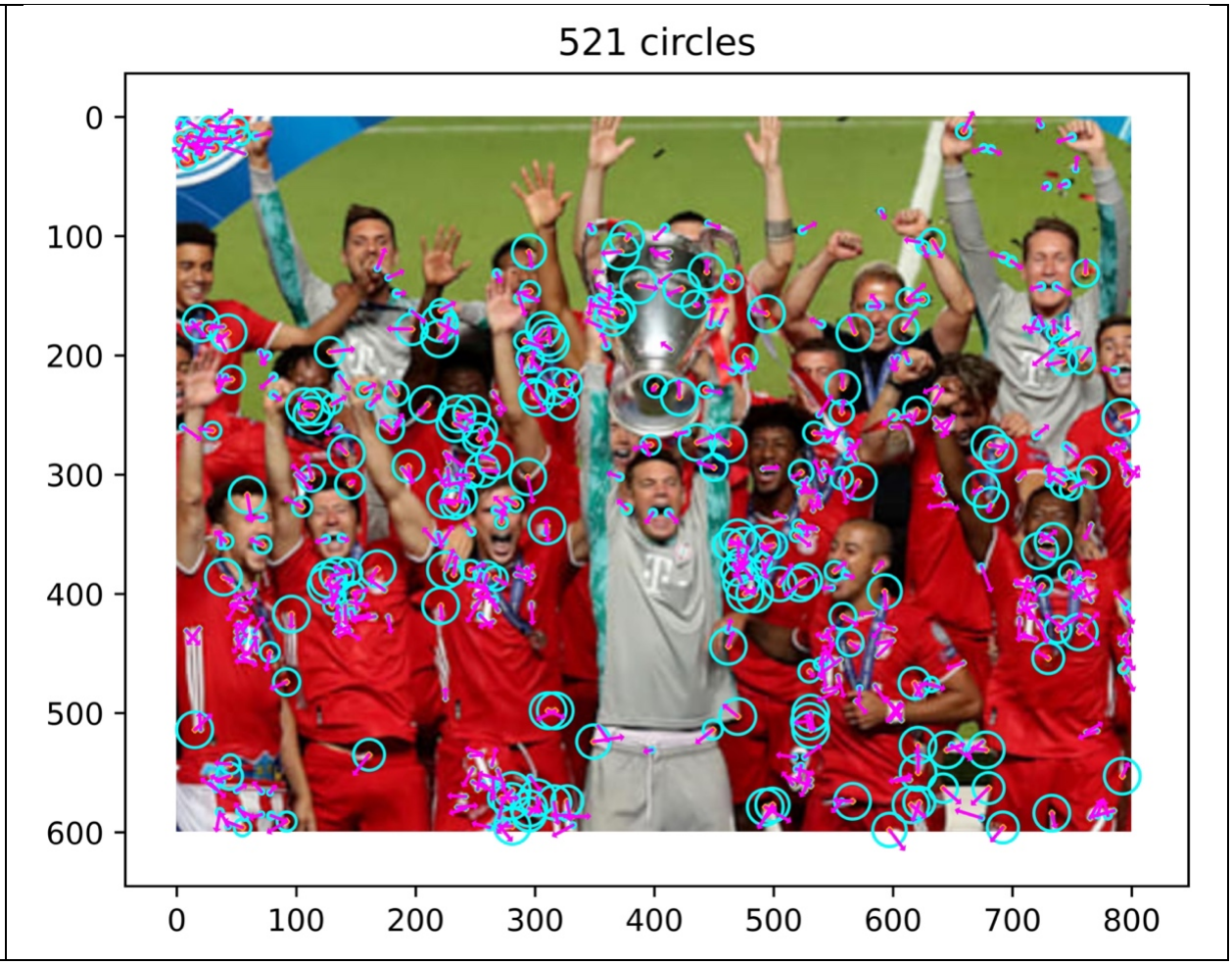
Example 3:

Image	
Base Image	

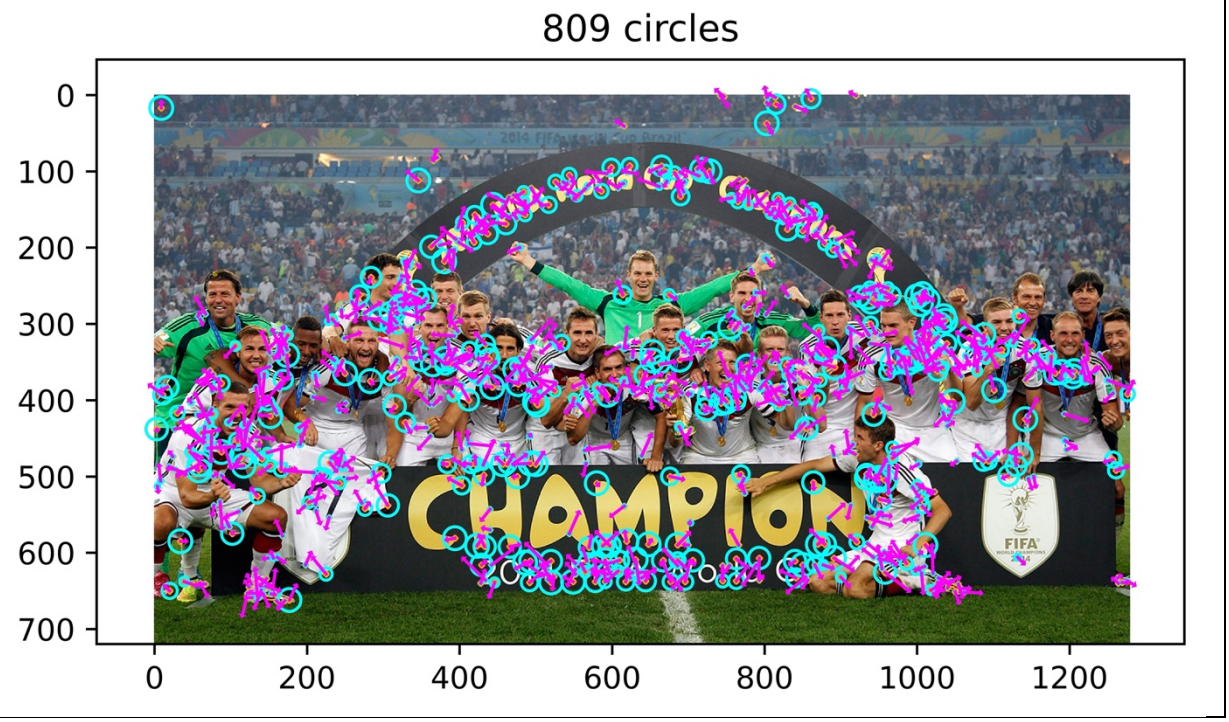
Shifted 20% Left	
Shifted 20% Right	

Rotated 90° CCW	

Rotated 90° CW	

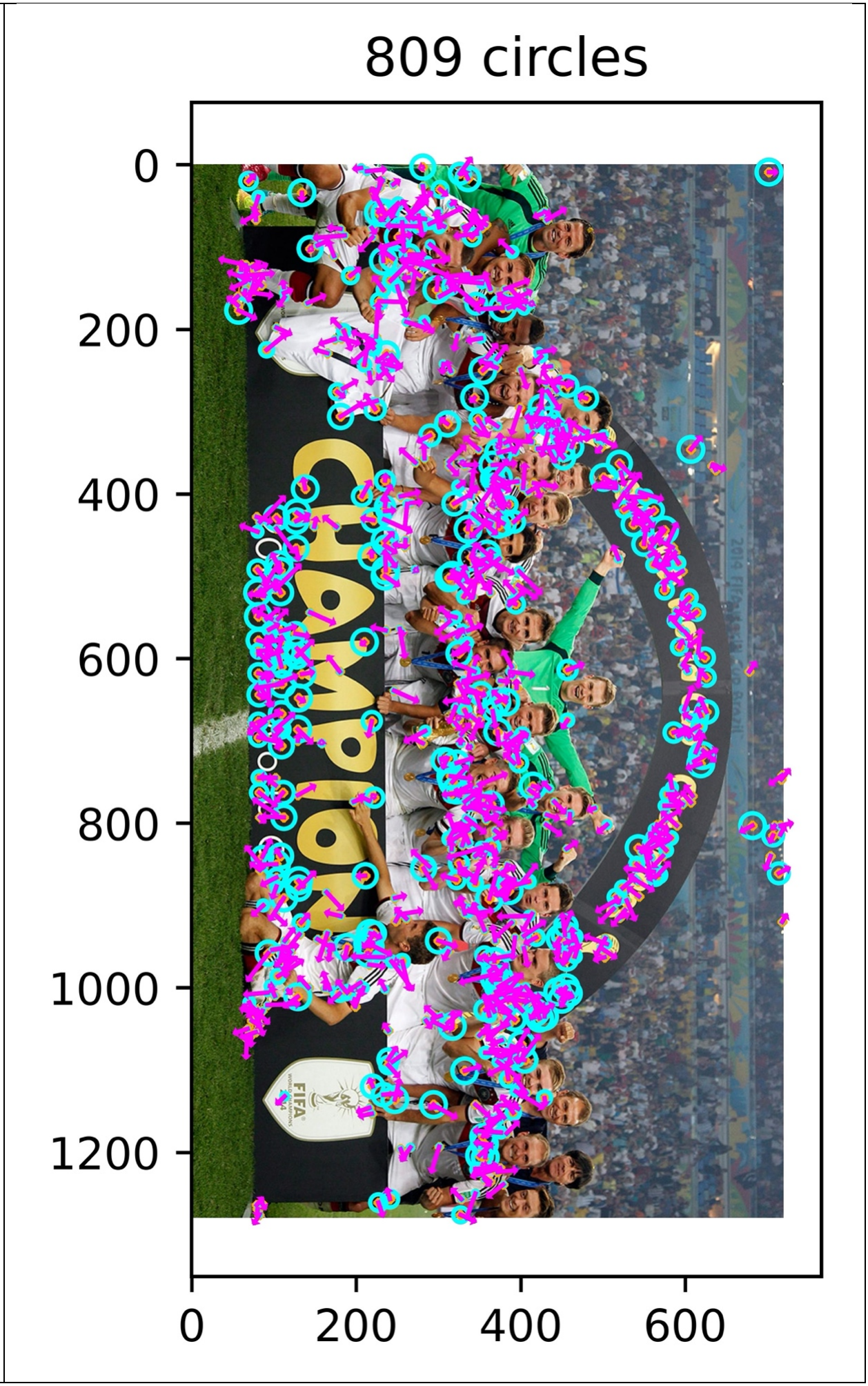
Enlarged by 2	

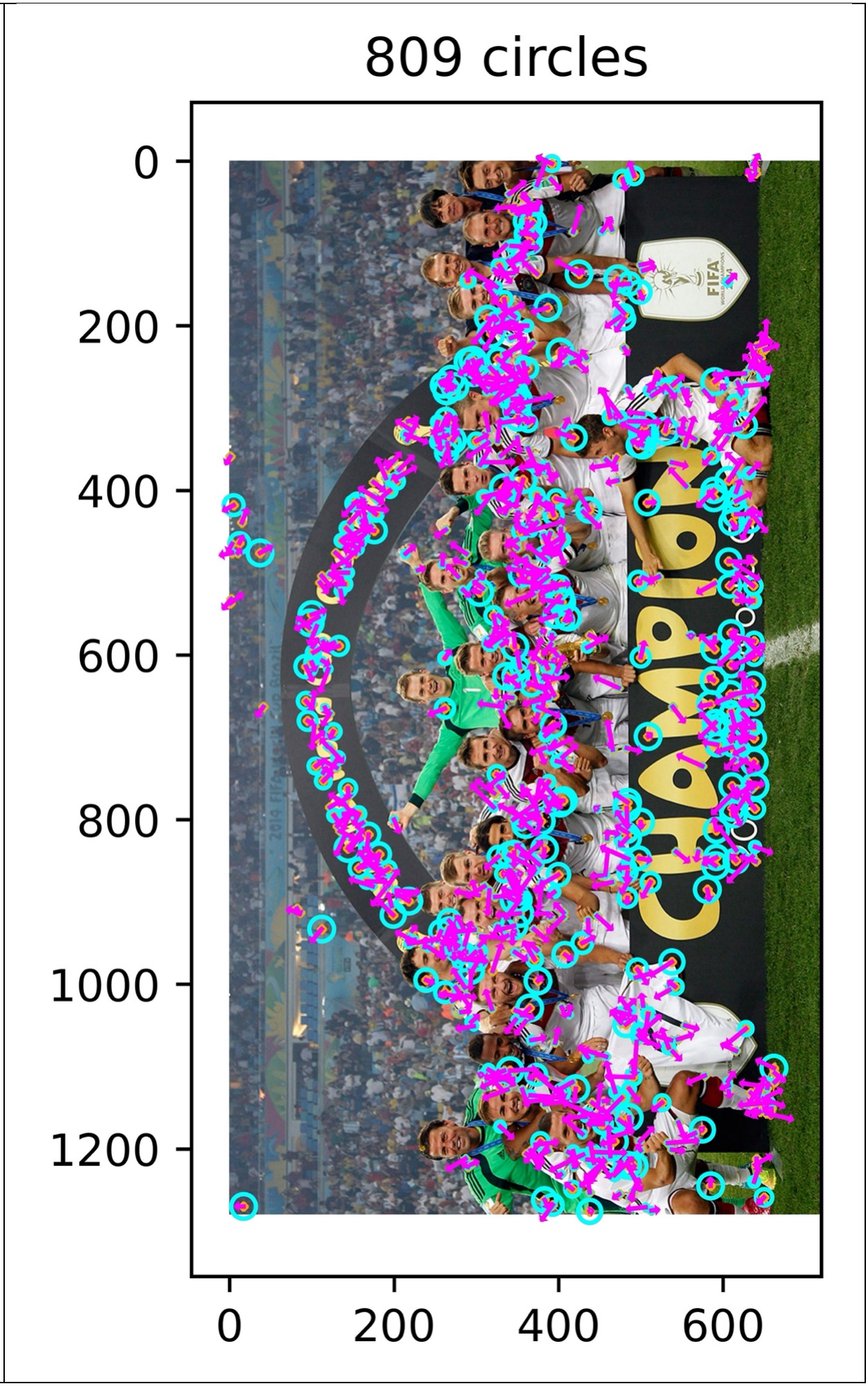
Example 4:

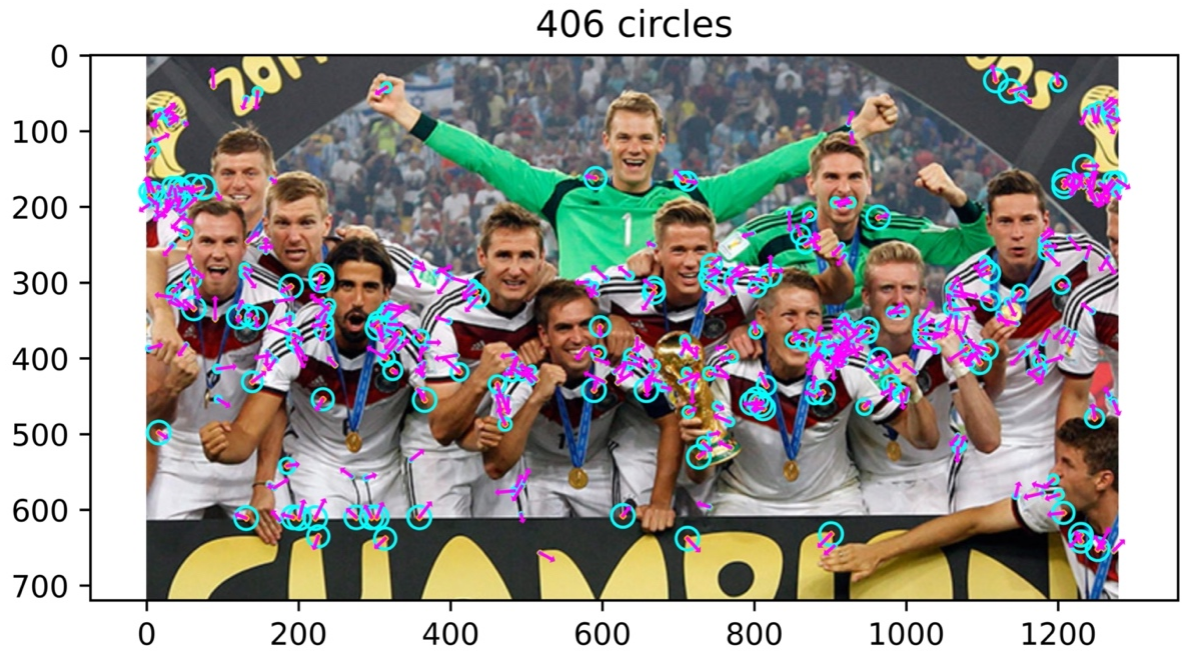
Image	
Base Image	

Shifted 20% Left	
Shifted 20% Right	

Rotated 90° CCW



Rotated 90° CW	

Enlarged by 2	

Discussion:

- Explanation of any "interesting" implementation choices that you made
  1. Read the image with cv2.imread
  2. Convert the image into gray scale and divided by 255.0 so that all pixel values are between 0 to 1
  3. Use cv2.goodFeaturesToTrack to find all corners of the gray-scale image
  4. Tune the hyperparameters of the function cv2.goodFeaturesToTrack so that there won't be too many corners displaying on the images which is overcrowded
     a. max_corners = 0, return all corners
     b. quality_level = 0.05, suppressed some of the corners
     c. min_dist = 5, the minimum distances between each two corners is 5 pixels, avoid crowded corners
     d. blk_size = 5, average block for computing a derivative covariation matrix over each pixel neighborhood
     e. k = 0.06 free parameter of the Harris detector
  5. Apply Laplacian of Gaussian filter with sigma=1 repeatedly combined with feature pyramid algorithm with 15 levels to detect the maximum LoG response to use radius of the blob.
  6. Apply Sobel filter twice in both x-direction and y-direction to find the orientation of the blobs

  According to the results above, when images are rotated, the locations of the corners and the size of the blobs do not change, but the orientations rotated together with the images. When the images are shifted left or right, the number of corners may decrease, but the orientations do not change. When the images are enlarged by a factor of 2, the number of corners and their locations may change because Harris corner detector is not invariant to scale.