CS543 Assignment 2

Your Name: Wei Chieh Huang
Your NetId: whuang73

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.

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

Image	<G> (h,w) offset	<R> (h,w) offset
00125v.jpg	[1, 6]	[1, 10]
00149v.jpg	[2, 4]	[1, 9]

00153v.jpg	[2, 7]	[0, 2]
00351v.jpg	[0, 4]	[1, 13]
00398v.jpg	[0, 5]	[0, 12]
01112v.jpg	[0, 0]	[1, 5]

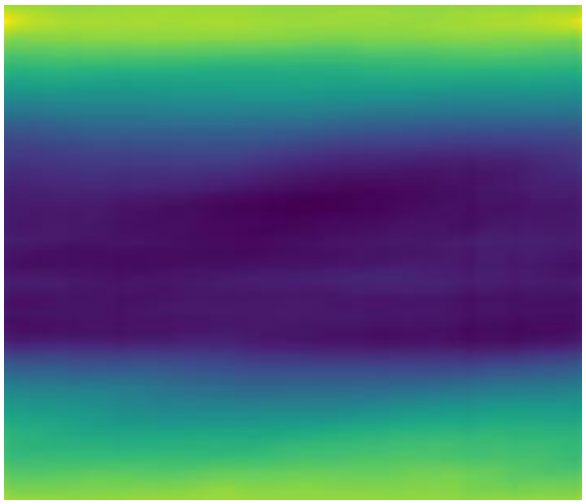
High-resolution images (using channel <B> as base channel):

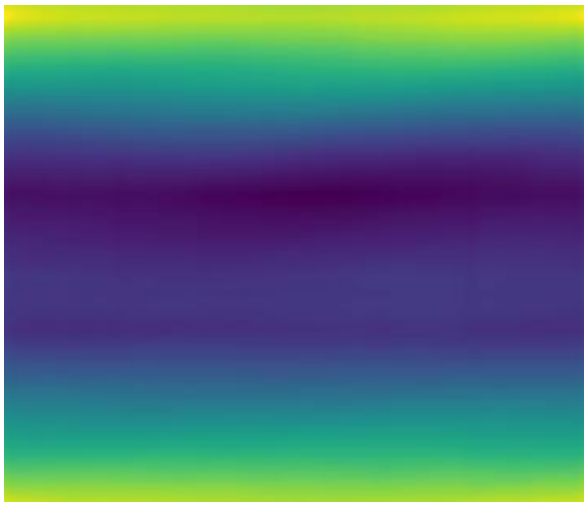
Image	<G> (h,w) offset	<R> (h,w) offset
01047u.tif	[20, 24]	[33, 72]
01657u.tif	[8, 53]	[10, 117]
01861a.tif	[39, 71]	[63, 148]

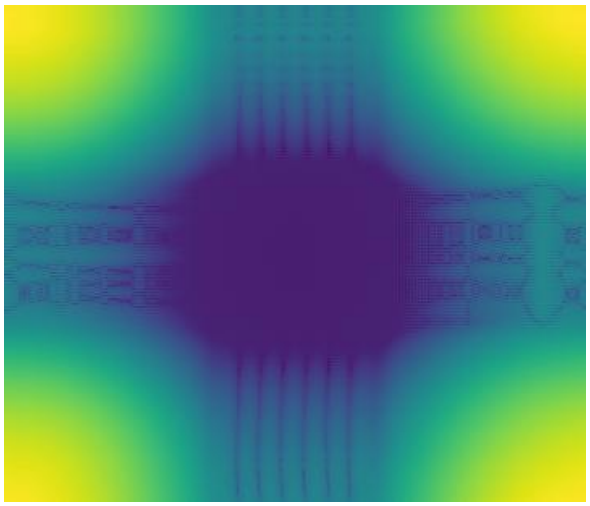
B: Output Visualizations

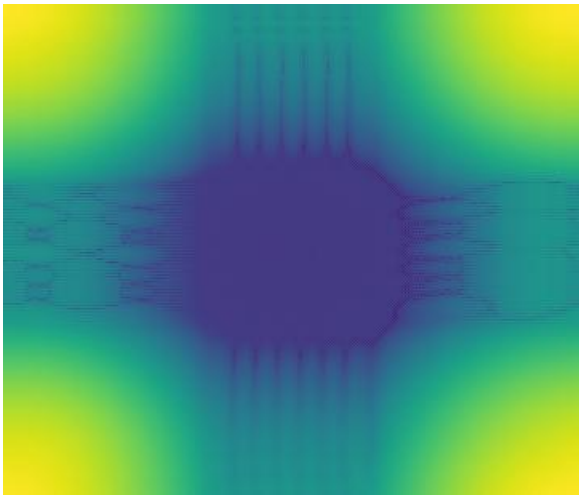
For each image, insert 5 outputs total (aligned image + 4 inverse Fourier transform visualizations) as described above. When you insert these outputs be sure to clearly label the inverse Fourier transform visualizations (e.g. “G to B alignment without preprocessing”).

00125v.jpg

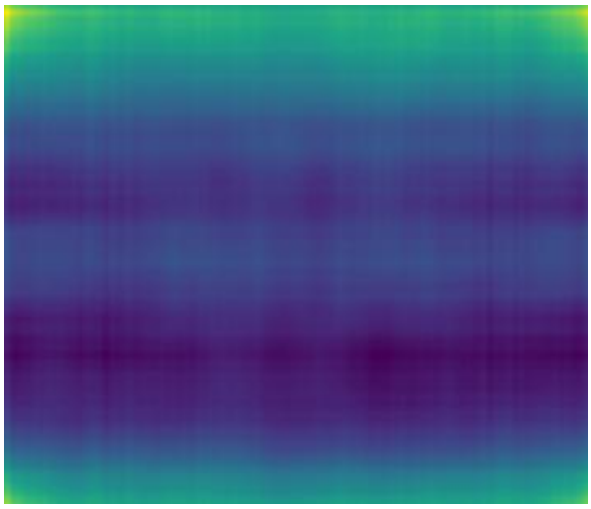


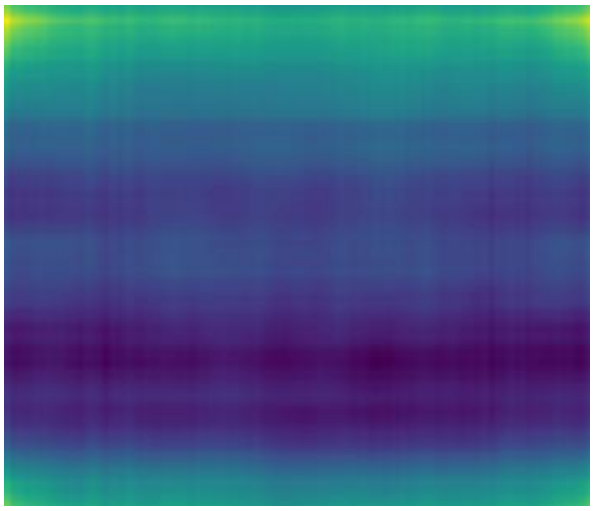


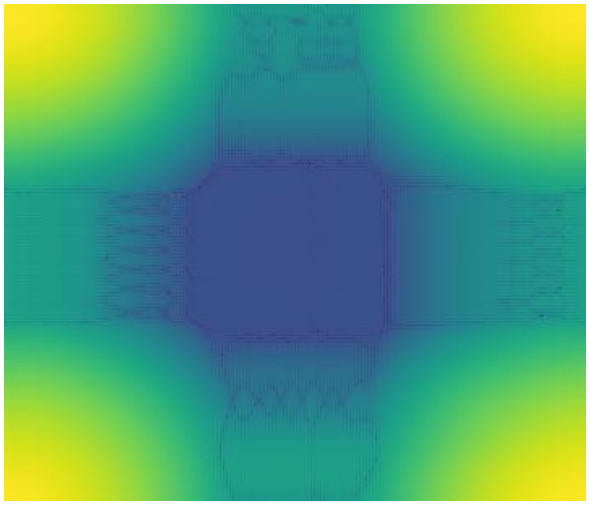


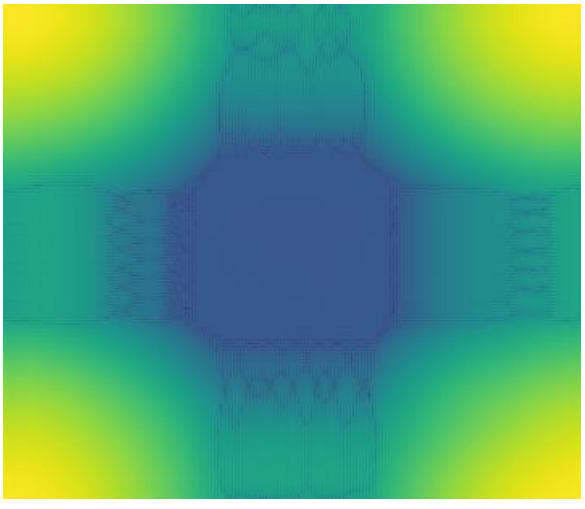


00149v.jpg

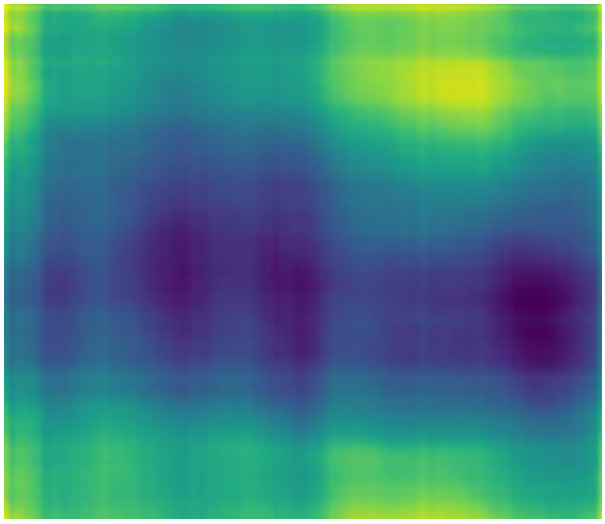


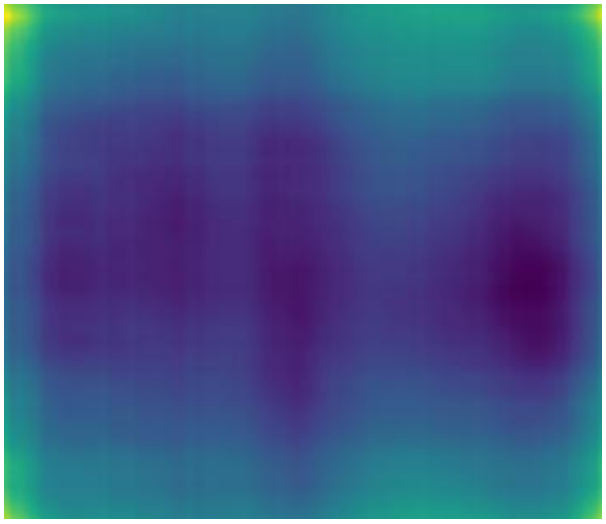


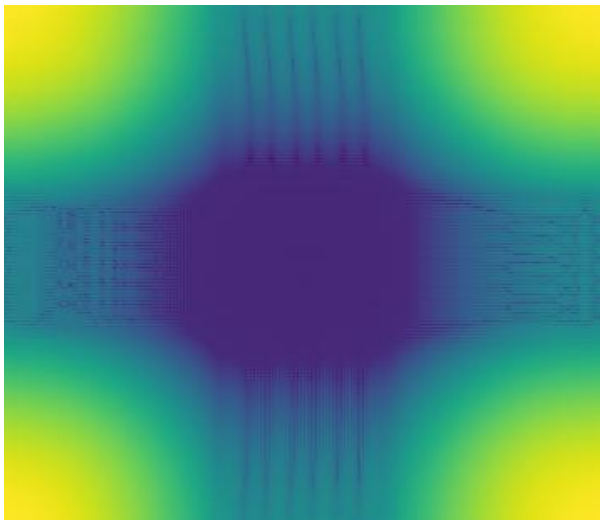


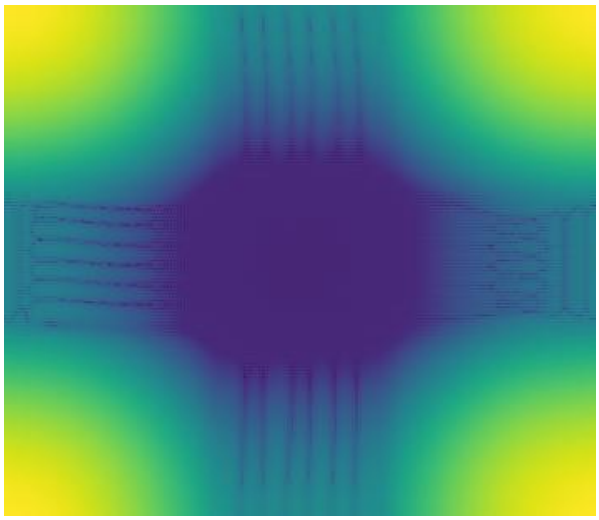


00153v.jpg

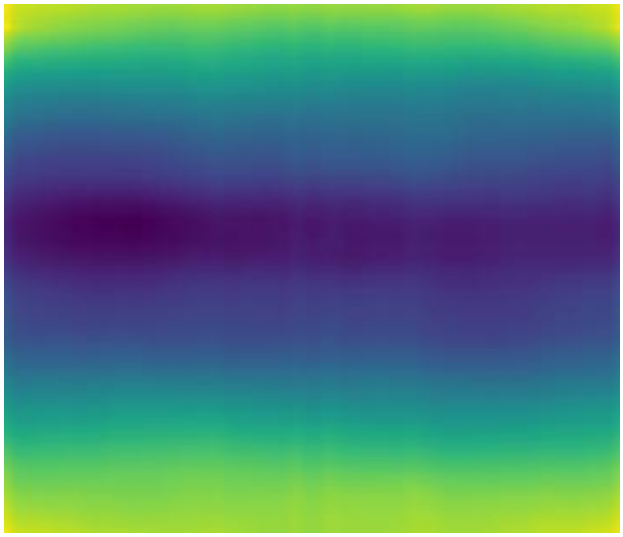


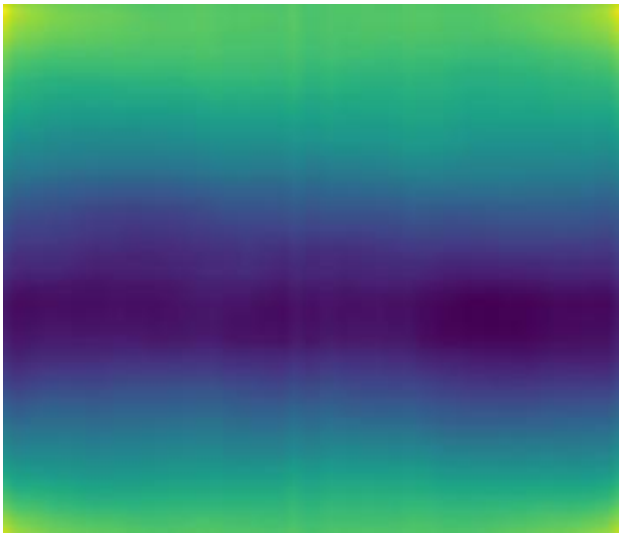


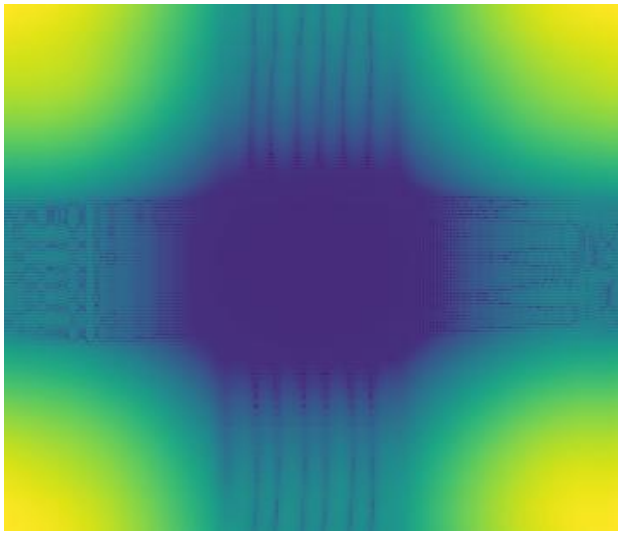


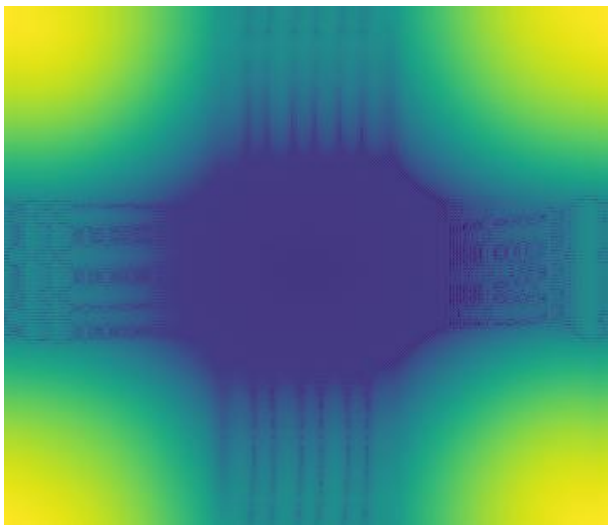


00351v.jpg

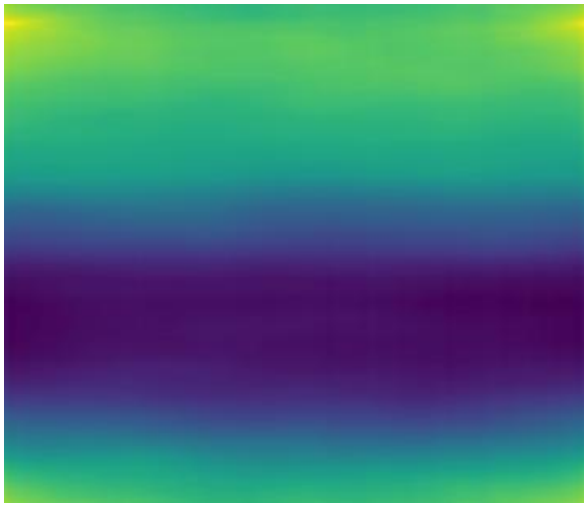


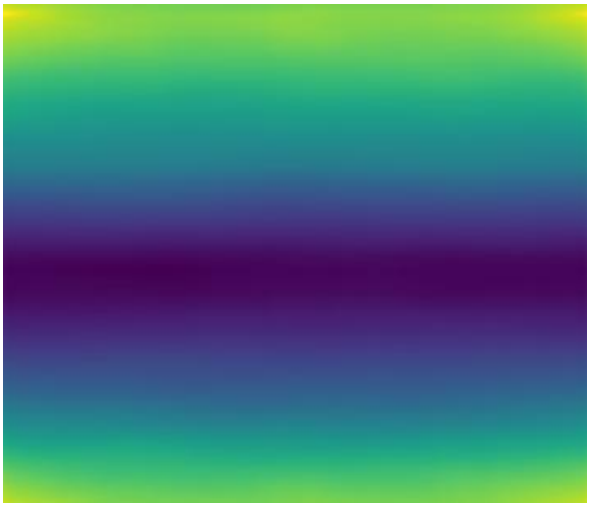


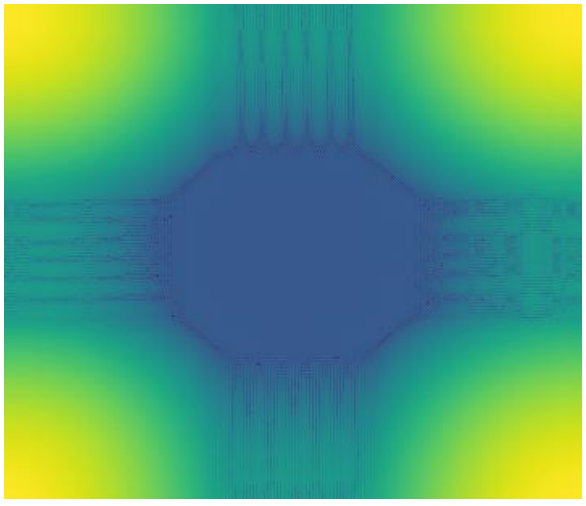


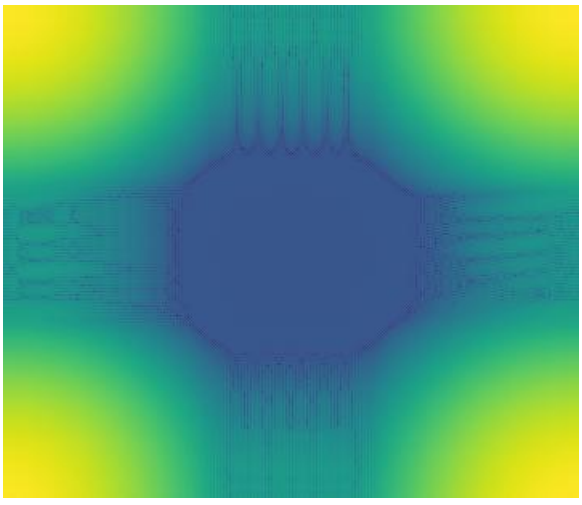


00398v.jpg

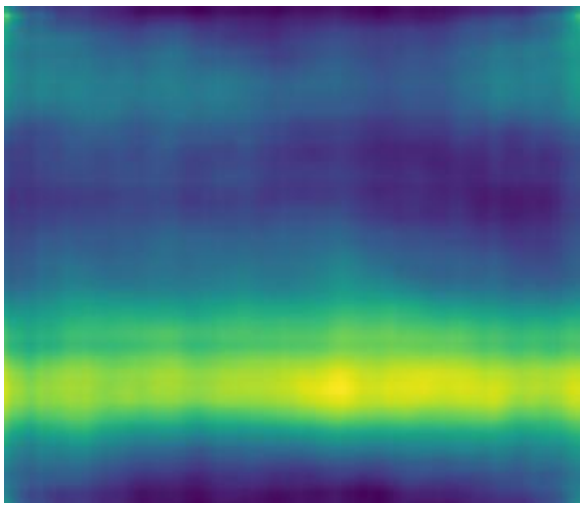


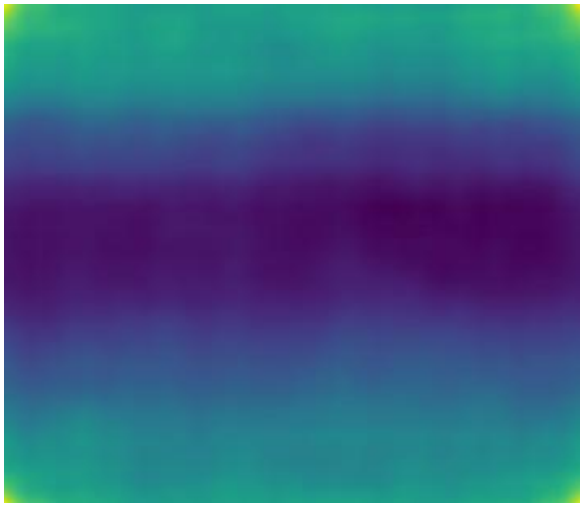


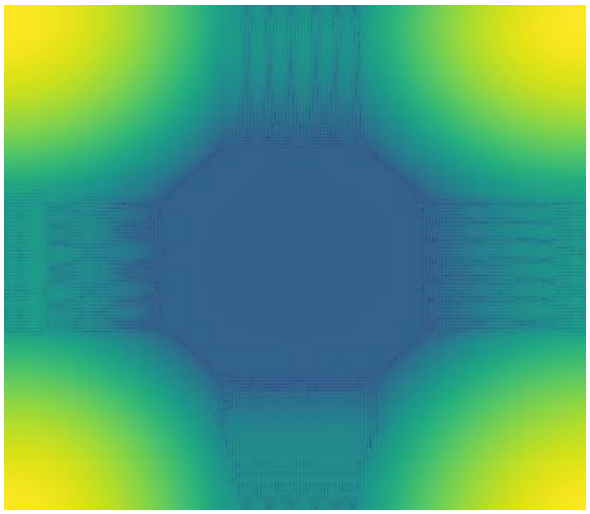




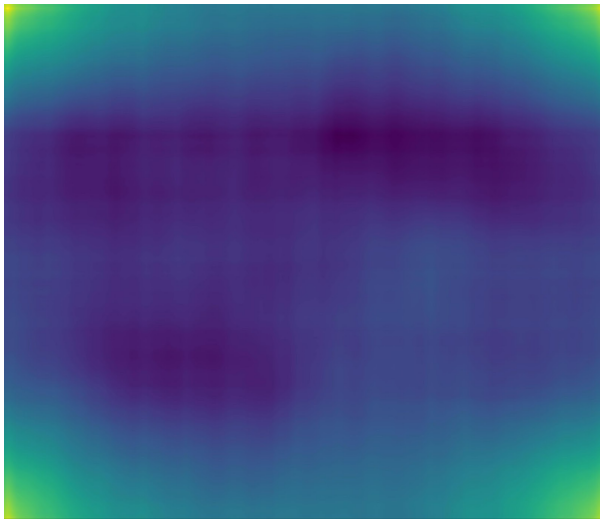
01112v.jpg

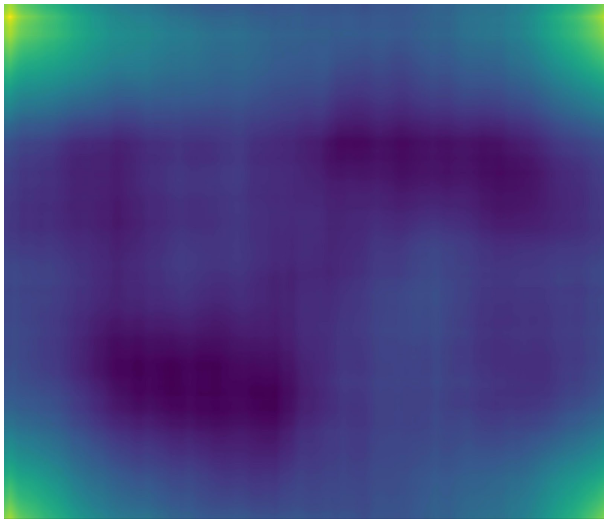


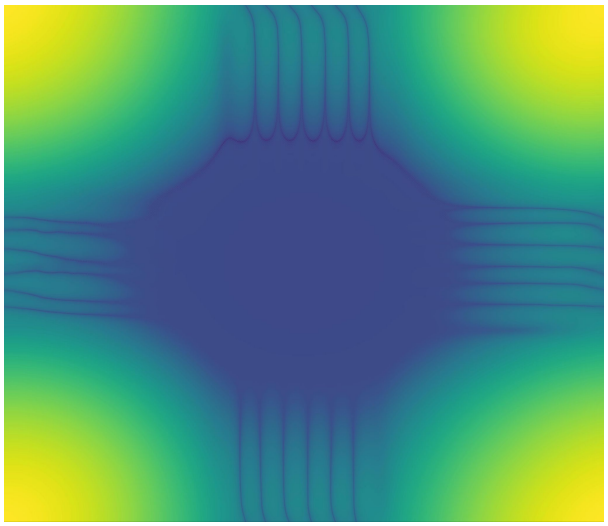


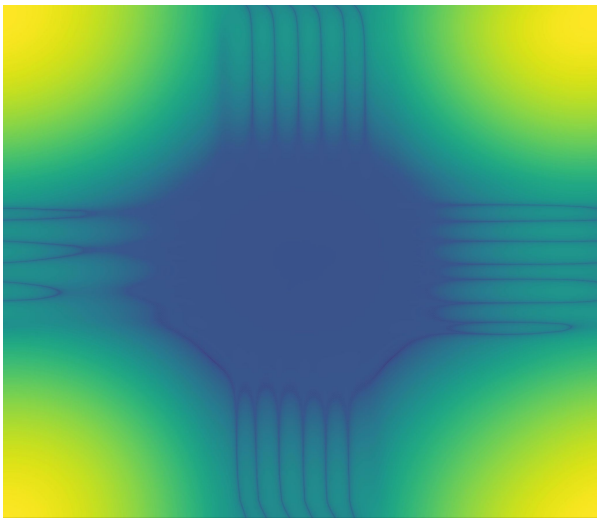


01047u.tif

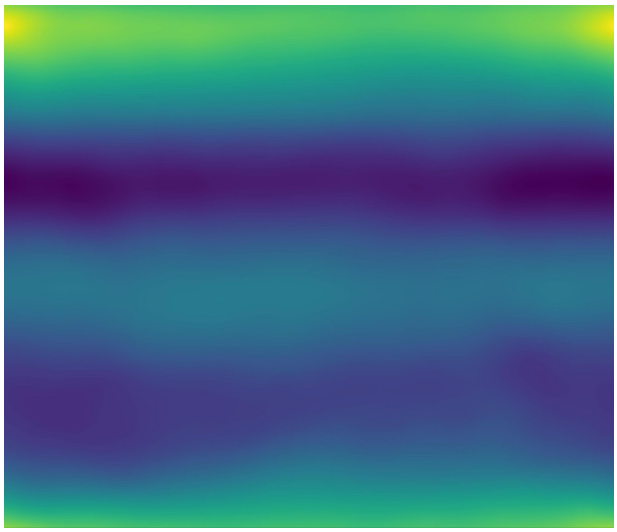


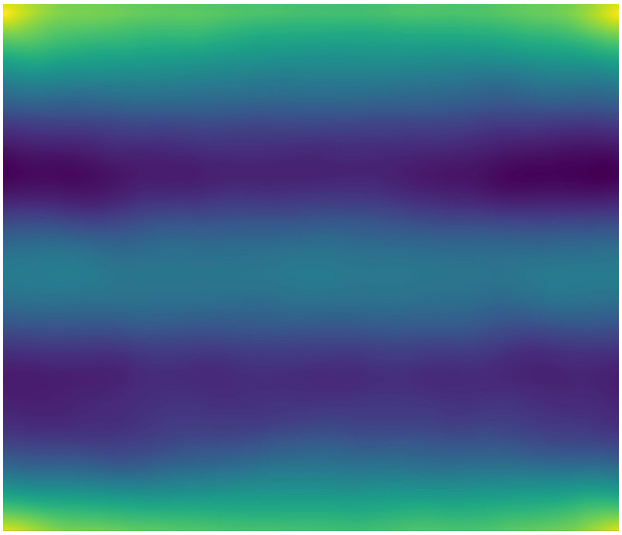


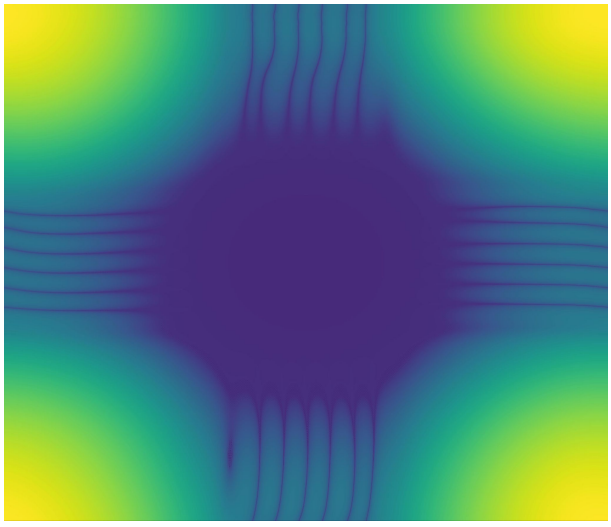


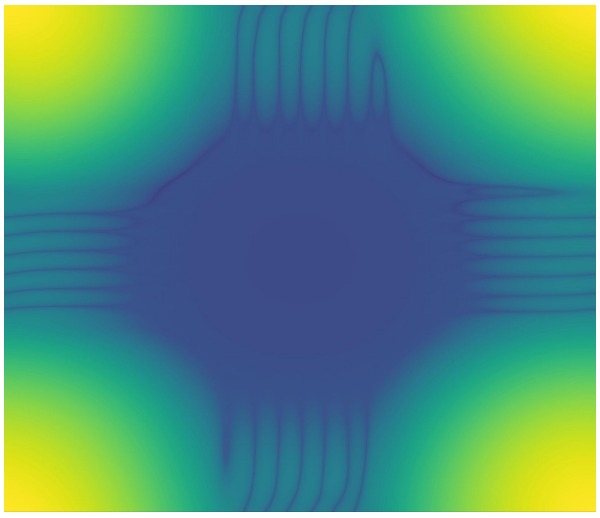


01657u.tif

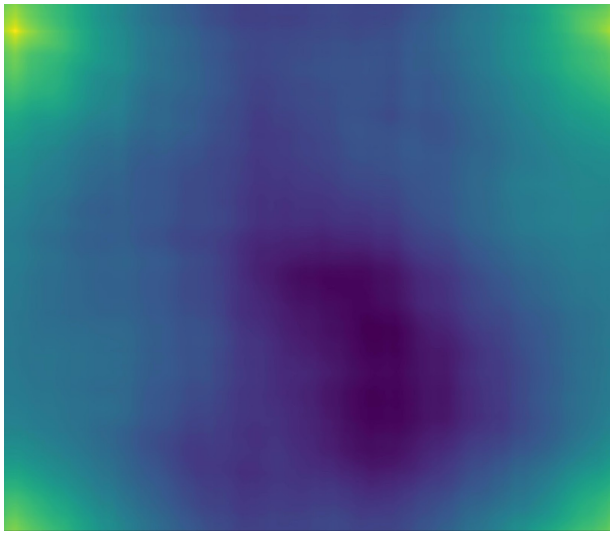


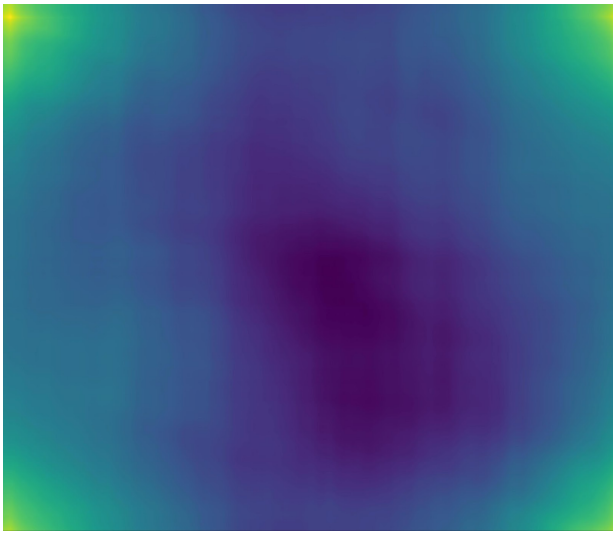


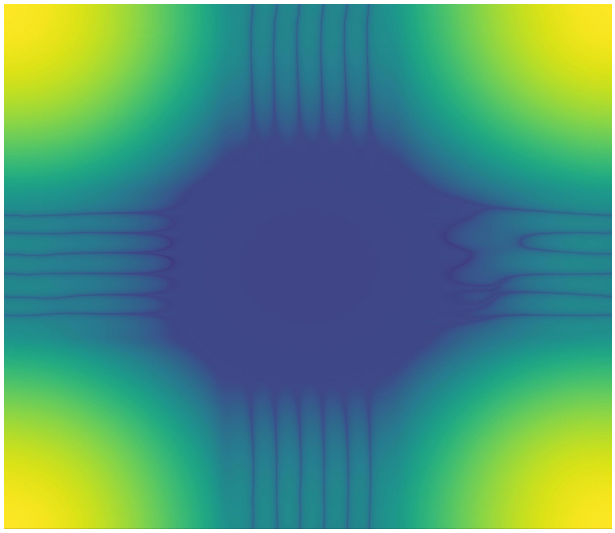


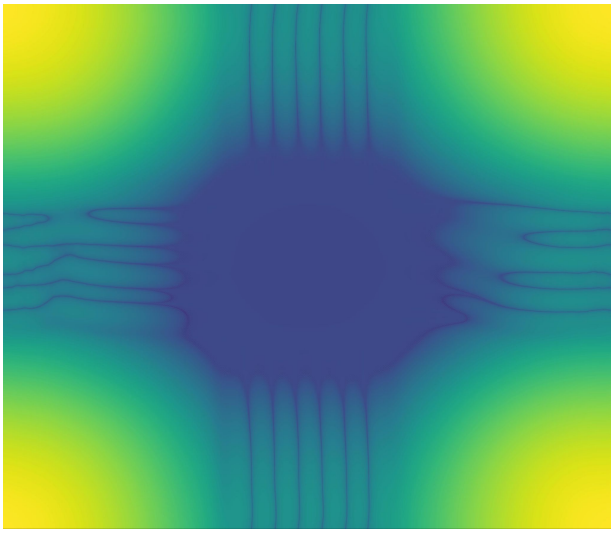


01861a.tif









C: Discussion and Runtime Comparison

- Discussion of any preprocessing you used on the color channels to improve alignment and how it changed the outputs
    - I used the scipy.ndimage.gaussian_filter as my preprocessing function. For image before preprocessing I got most of the image aligned properly except for 153v.jpg. After preprocessing, the displacement of my alignment has changed a little but just a few pixel difference. All the image looks pretty much the same as the image before preprocessing. However, somehow img 153v.jpg was still slightly off. I'm thinking it might be places in the cropping method of mine that causes the issue since it has the same issue during assignment1 for 153v.jpg. However, all other images are working perfectly, so it could also be the image itself that has problem with it. Overall it is hard for me to tell how much it improve after preprocessing since I can't really tell a big difference before and after 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?
    - When using Fourier-based alignment, the average runtime for jpg images were about 0.1 second each. For high resolution(tif) images it takes about 6 seconds for each image. Comparing to the multiscale alignment method which normally takes around 5 seconds for each jpg images, and it never finished running or taking too long to run for tif image. Where I estimate the run time were about 9 minutes for tif image. With the image pyramid method used in Assignment1, the runtime were around 15 seconds for each tif image. In any scenario tested so far, Fourier-based alignment has a better performance on runtime.

Part 2 Scale-Space Blob Detection:

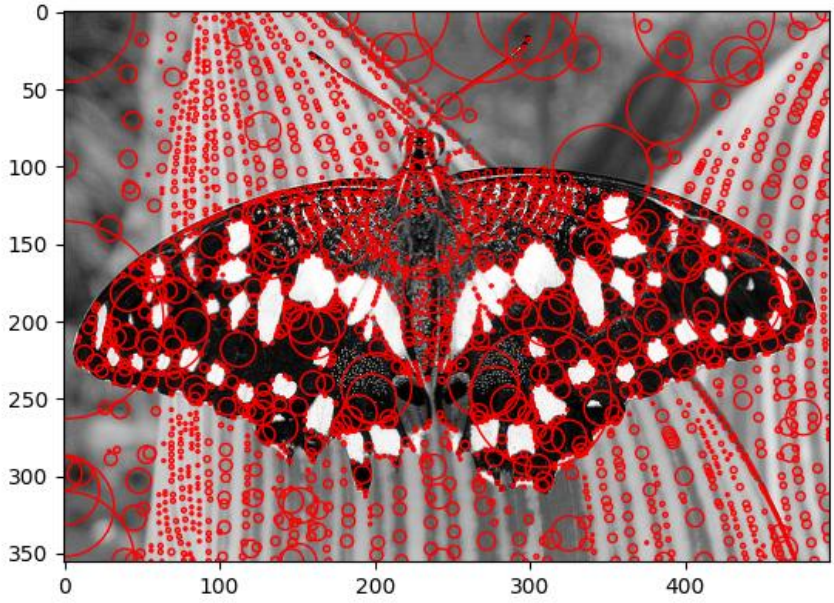
You will provide the following for ***8 different examples*** (4 provided, 4 of your own):

- original image
- output of your circle detector on the image
- running time for the "efficient" implementation on this image
- running time for the "inefficient" implementation on this image

You will provide the following as further discussion overall:

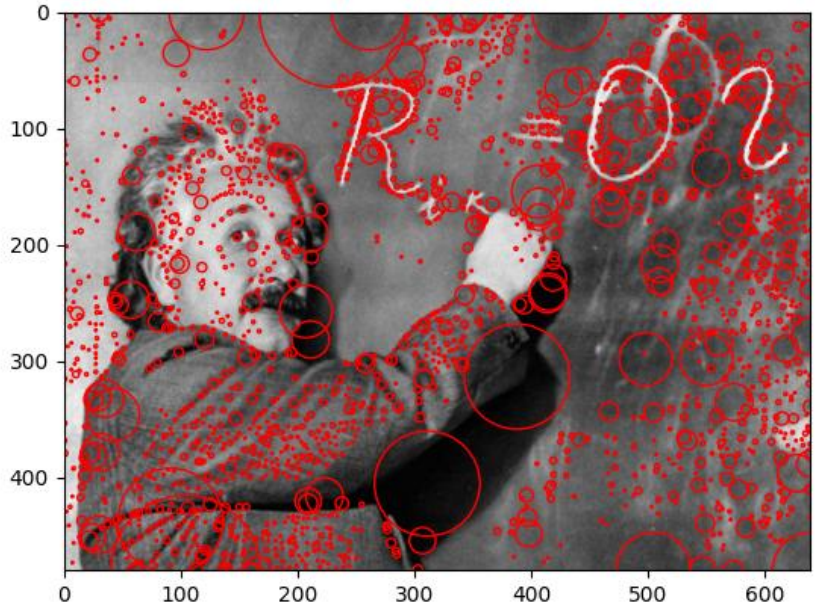
- Explanation of any "interesting" implementation choices that you made.
- Discussion of optimal parameter values or ones you have tried

Example 1: Butterfly (threshold: 0.02)



~5.5 Second for increase scale
~5.3 Second for downsample

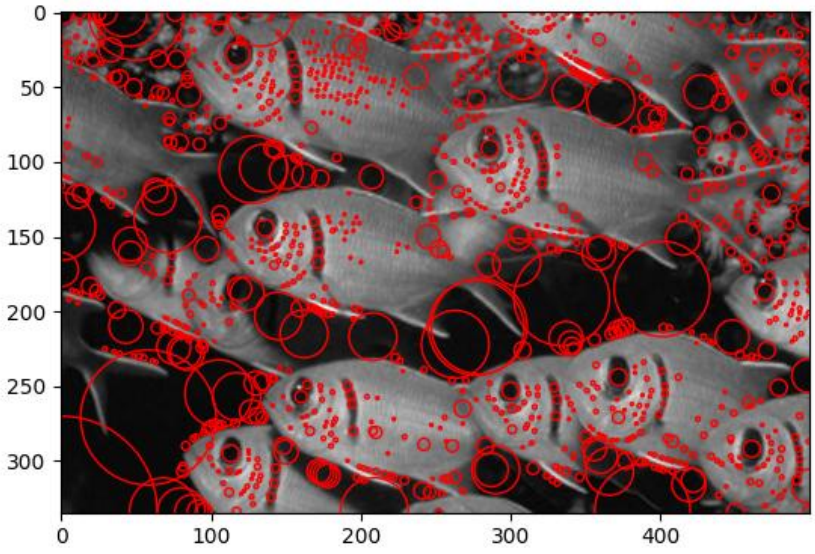
Example 2: Einstein (threshold: 0.02)



~7 Second for increase scale

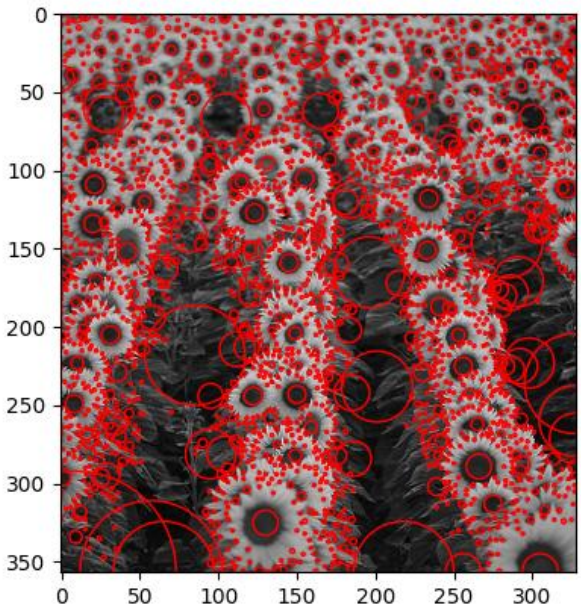
~6.6 Second for downsample

Example 3: Fishes (threshold: 0.02)



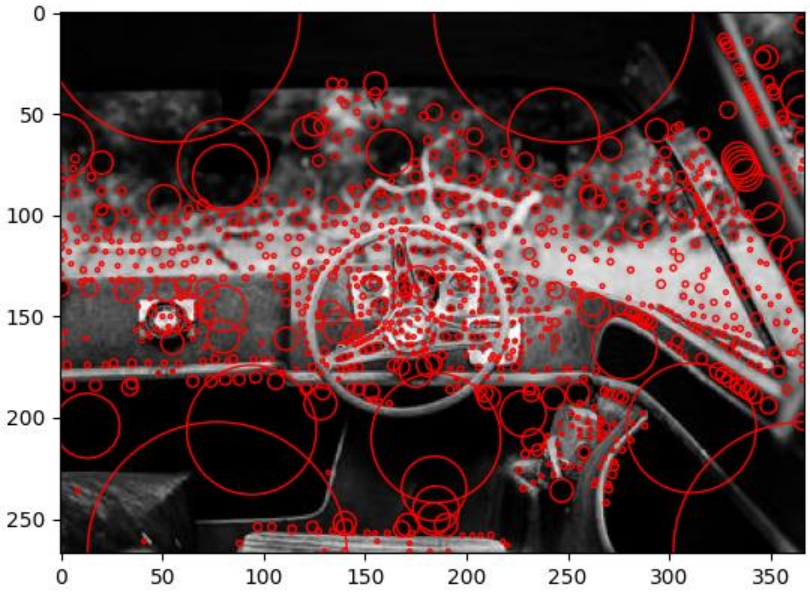
~4 Second for increase scale
~3.6 Second for downsample

Example 4: Sunflowers (threshold: 0.02)



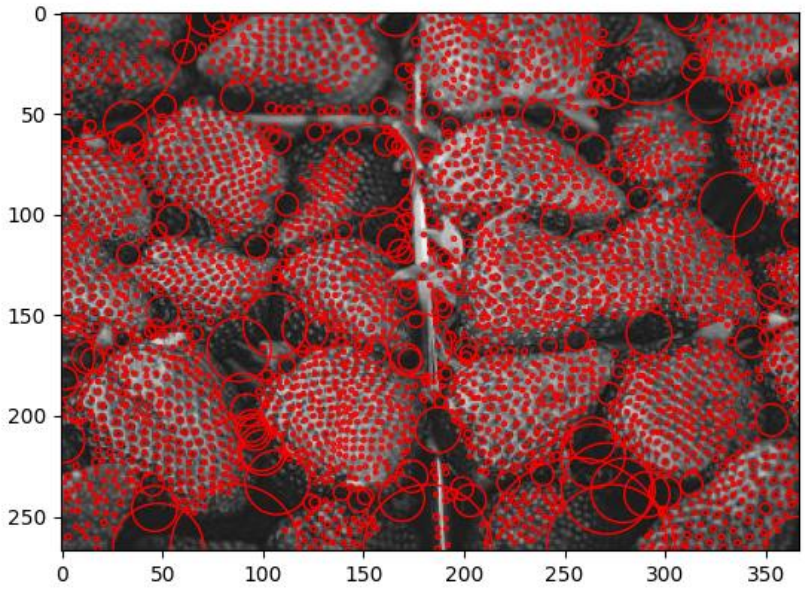
~5 Second for increase scale
~4.7 Second for downsample

Example 5: Car (threshold: 0.015)



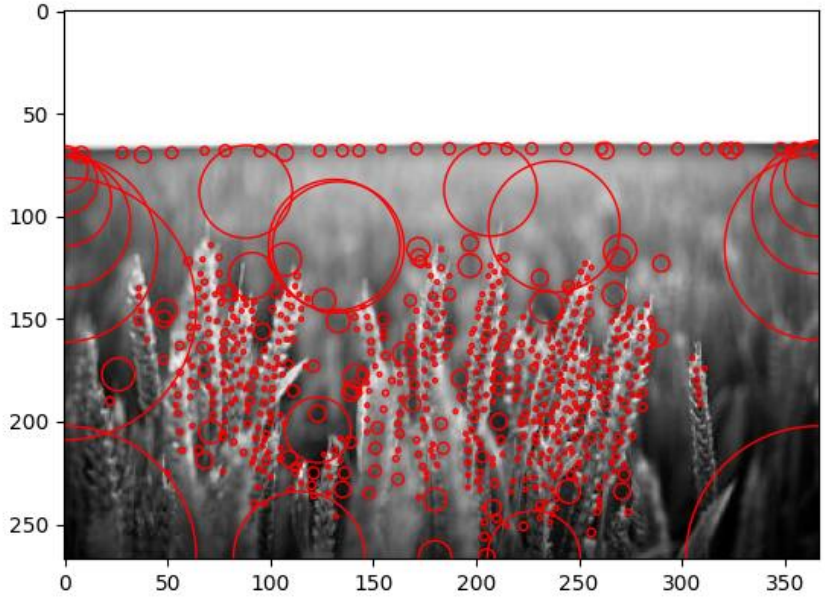
~2.7 seconds for increase scale
~2.2 seconds for downsample

Example 6: Strawberry (threshold: 0.02)



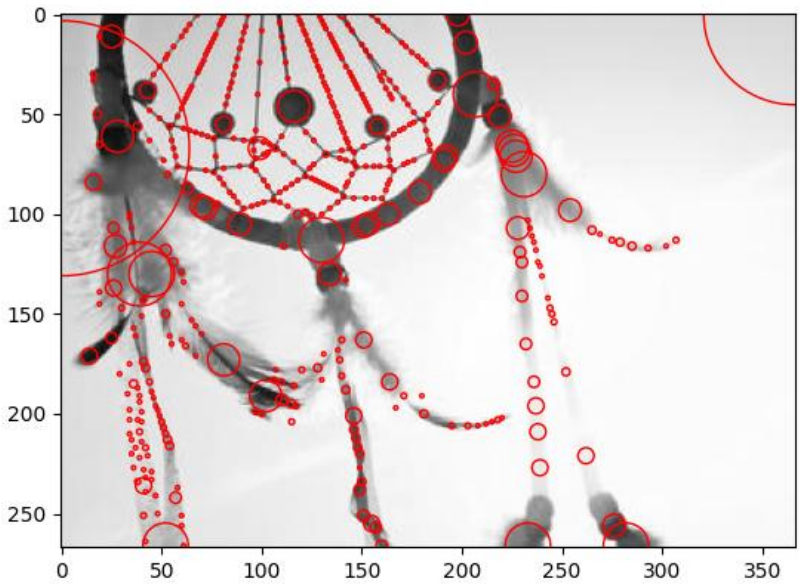
~6.5 Seconds for increase scale
~6.1 Seconds for downsample

Example 7: plant (threshold: 0.02)



~2.2 Seconds for increase scale
~2 Seconds for downsample

Example 8: Net (threshold: 0.015)



~2 Second for increase scale
~1.8 Second for increase scale

Discussion:

- Explanation of any "interesting" implementation choices that you made.
    - There isn't any very interesting implementation for this part of the assignment. I uses rank_filter as my NMS approach. Since there wasn't much documentation or example used in a similar way of this assignment on both rank_filter and generic_filter function. I go with the one that I feel more comfortable and have a better understanding on. Otherwise, most of my implementation was just follow along with the homework description. I did not use numpy.clip in my implementation since I think if we are using threshold and np.where. It wasn't too necessary to use that function in my opinion.
- Discussion of optimal parameter values or ones you have tried
    - The optimal parameter for me is 0.015~0.02 on the threshold. If the threshold goes under 0.015, it normally detect small blob that are not obvious to see or places that should not have a blob. If the threshold goes over 0.02, it would only detect very few blobs. Therefore, based on my result, 0.015~0.02 is the optimal threshold for me.

Bonus:

Blob-Detection Extra Credit

- Discussion and results of any extensions or bonus features you have implemented for Blob-Detection