Name: Remaldeep Singh
UID: u1143744
Email: u1143744@utah.edu

The main.m file in the project directory is used to call the phaseCorrelation() method. There are commented lines in the main.m file for different inputs. Uncomment them for different inputs.

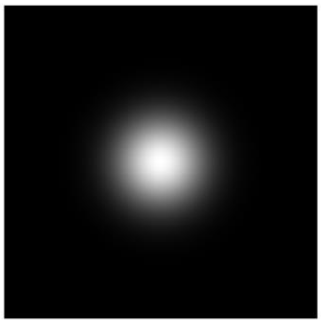
Phase Correlation

This part is in the phaseCorrelation.m file. The method takes in a folder as an input. The folder contains all the files that need to be stitched into a mosaic. It saves the mosaic image in the Outputs folder at the end as “mosain.png”.

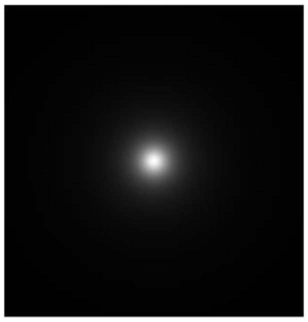
phaseCorrelation() method starts by reading all the files in the folder and padding the images to make them of the same size. **padImages()** method is used for this. The padded images are stored in the **set images**. I am padding the image by **replicating the corner pixels** which performed much better than padding the image with zeros.

Next createLowPassFilter() method is called to create a low pass filter. I am using **Guassian** distribution to create the low pass filter with sigma 50. For each image a Fourier transform is calculated and stored in the fourierIm set. Before calculating the Fourier transform, each image is converted to grayscale, histogram equalized and converted to double.

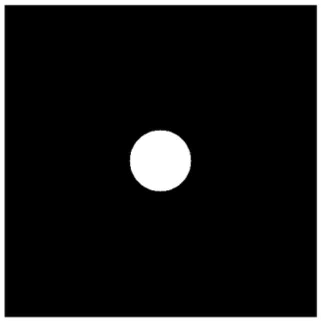
Gaussian with sigma=50.



Butterworth with sigma=50, n=1



Ideal low pass with sigma=50.



The next step is to create an image **canvas** that is of the size ”2*(no of images) – 1”. Since in the worst-case images can stack only on one side hence 2 times the no of images. The first image in the images set is stitched at the center of the canvas and a list “**imagesOnCanvas**” is created to keep track of all the images present on the canvas and their top left positions. Now we iterate over each Fourier image and their possible combination with other Fourier images. This is inside the double for loop.

Inside the double for loop phase correlation is calculated with 2 Fourier images “i and j” and we take into account only the real part of the correlation. For the real part “**realG**”, **findPeak()** method is called. This method first finds the peak in the image and then thresholds the realG image with a value 20 less than the peak. I just applied a general 80:20 rule here. With this thersholded image I call the **floodFill()** method (this is taken from first project and modified), to get the sum of X indices and Y indices in **avgV** and the count. This method then returns the rounded average X and Y and the maximum value of the peak. The thresholding on the peak value is 0.002, any value less than this denotes that the two images are not overlapping.

Since we know the peak value index **maxIndex,** we have four quadrants now where the image can fit. For each quadrant the part of the two images is extracted and a method **calculateDIC()** is called which gives back a value denoting how correlated these images are. The calculateDIC() uses the following formulae:

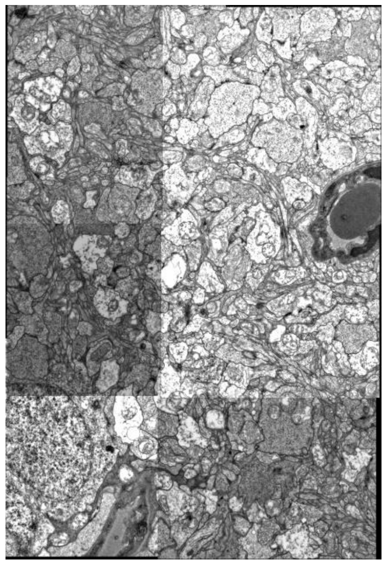
$$r_{ij} = \frac{\sum_m \sum_n [f(m+i, n+j) - \bar{f}][g(m,n) - \bar{g}]}{\sqrt{\sum_m \sum_n [f(m,n) - \bar{f}]^2 \sum_m \sum_n [g(m,n) - \bar{g}]^2}}.$$

Wherein i and j is 0 for me since we are only interested in the image correlation without shifting. From the four quadrants we get four values, we take the max value and that quadrant for stitching to the canvas. Before stitching we check if the image is present on the canvas already or not. If it is present, then we skip the image (this is done by not having a condition for iIndex ~= 0and jIndex ~= 0). iIndex is 0 when the image is not present on the canvas and ~0 when the image is present on the canvas. **isPresentOnCanvas()** method is used to return the proper index. Based on the quadrant where stitching will happen that particular if condition is fulfilled and inside we check which of i or j are present on the canvas (with index ~=0 and jIndex ~= 0). Respectively the new top left of the to be stitched image is calculated (**newY, newX**) from the top left image already on the canvas (**imX, imY**). And then that image is stitched on the canvas. Consequently, the image index is stored in the imagesOnCanvas list with the position of the top left corner.

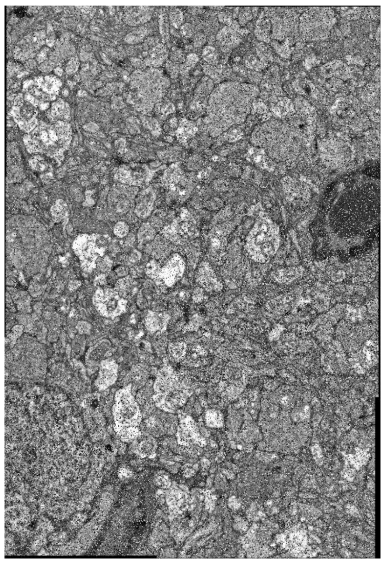
At the end of the **phaseCorrelation()** method, the canvas is resized with the **reduceCanvasSize()** method. This method calculates the minimum and maximum of x and y of the images on the canvas from the list of **imagesOnCanvas** and strips that part out from the canvas. This new stripped image is returned from the method. The output is written in the output folder.

Some example outputs:

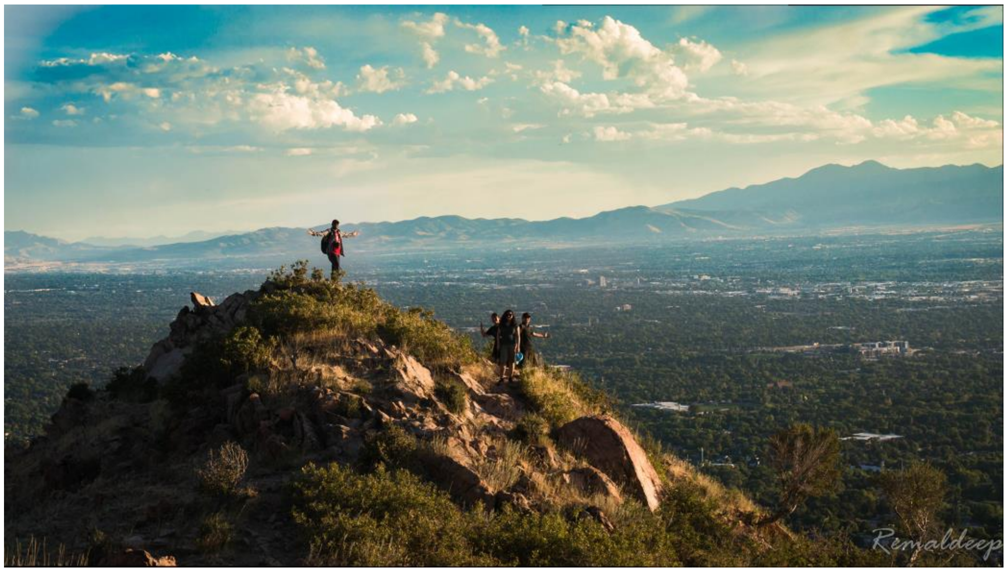
Cell image stich.



Noisy cell image stich.

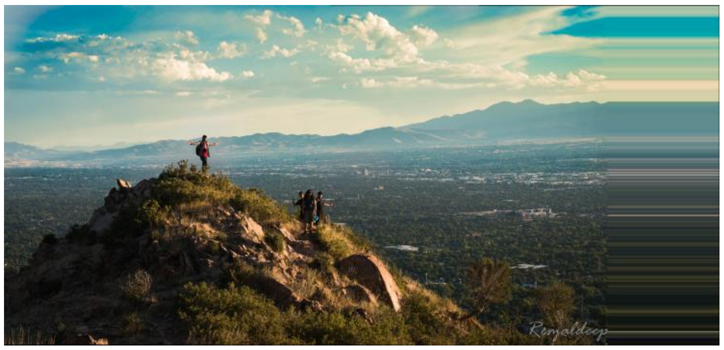


Stitched image of unequal parts



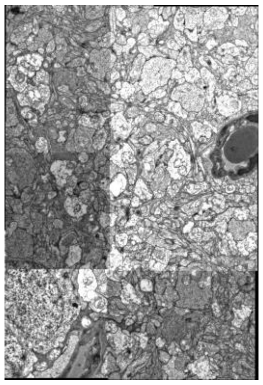
Experiments:

1) Image stitching with padded inputs:

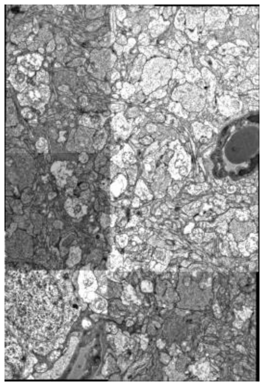


2) The outputs for different low pass filters, at sigma 50, have almost the same output but slight variations in the peaks, hence slight variations in the image stitching.

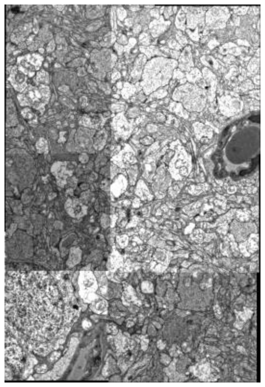
Ideal Low pass.



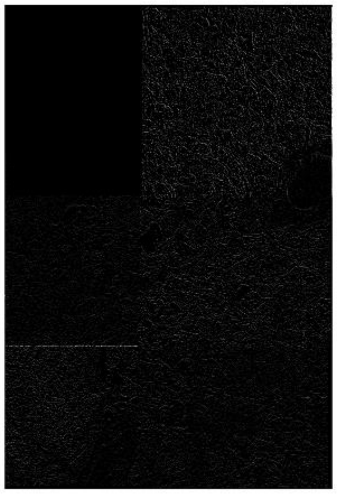
Butterworth.



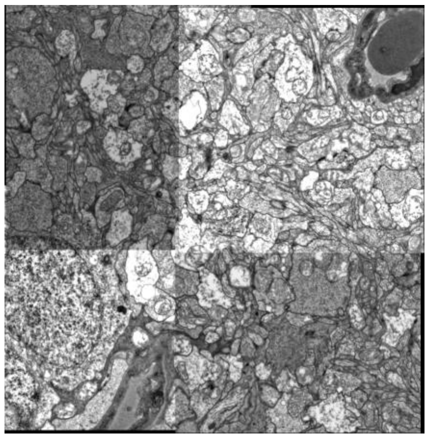
Guassian.



The difference between Guassian and Butterworth outputs. There is very slight variation.



3) For Low pass filter kernels I found that the sigma value of 50 works best here. For value less than that it sometimes skips some of the tile stitching.



4) By experimenting with power ‘n’ for the butterworth kernel I couldn’t find visible difference but the mosaic stitching became more and more closer to ILPF with higher power, maybe because butterworth of higher power replicates the ripple effect of ILPF.

References:

https://en.wikipedia.org/wiki/Digital_image_correlation

https://www.mathworks.com/matlabcentral/answers/40705-how-to-delete-a-column-from-the-matrix