

CS 6640 Project 3

Travis Allen, u1056595

November 7, 2021

1 Part 1: FFT's

Use the built-in FFT functions in python (i.e., numpy) to compute the Fourier transform of some images and show their power spectrum. You should organize the frequencies so that zero is in the middle of the FFT result/images. You might need to use a log to see the power clearly. Show results on several images and comment on what you see in both domains. Implement a lower pass filter in the Fourier domain and do the inverse FFT to show the effects on several images.

Please see `problem_1.py` for how I implemented this.

Images and their FFTs I used the `numpy.fft.fft2()` function to compute the discrete fast fourier transform of each of my images. Then, I used `numpy.fft.fftshift()` to “retile” each image so that the center of each image corresponds to a frequency of zero. The only purpose of this step is to make the results (both intermediate and final) easier to digest. Then, I computed the log of the power spectrum of each image so the results were easier to see. These results are shown below for each image.



Figure 1: Image 0 and its fourier transform

Low Pass Filtering in the Fourier Domain Typically we think of low pass filtering in the spatial domain as convolution of the image with some function. From the properties of the fourier transform we know that convolution in the spatial domain is the same as multiplication in the fourier domain. This means that we can multiply the fourier transform of the image with the fourier transform of our filtering function, and then take the inverse fourier transform of that product to achieve the same result as convolution in the spatial domain.

I implemented this in two ways. First, I created a 2D $\text{rect}(u, v)$ function in the fourier domain ($\text{sinc}(x, y)$). This is shown below:

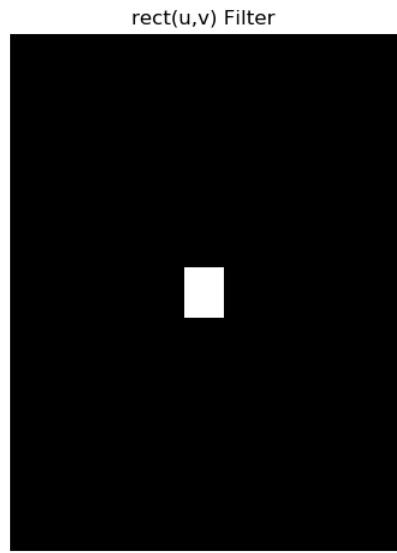


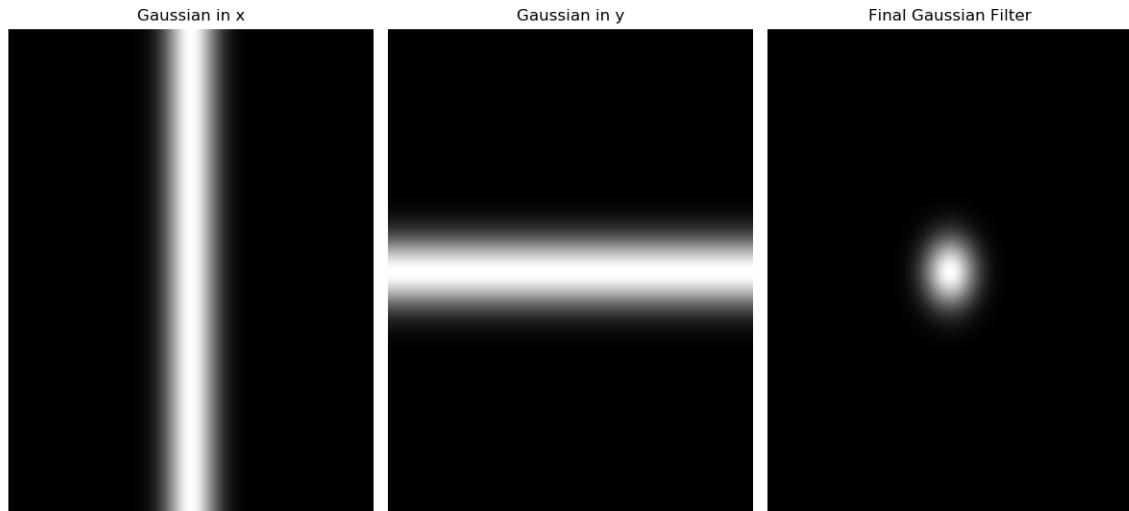
Figure 2: $\text{rect}(u, v)$ filter in the Fourier Domain

I element-wise multiplied each pixel in the filter with each pixel in the image. We know that although this is a mathematically “ideal” function, in practice it produces artifacts that we are not interested in, like ringing.



Figure 3: Original image and fourier-domain $\text{rect}(u, v)$ filtered image. Note that this is the same as convolution with a $\text{sinc}(x, y)$ function in the spatial domain. This filter appears to darken the image as well

Next, I created a 2D Gaussian distribution function in the fourier domain. We know that $\mathcal{F}\{\exp(-\pi u^2)\} = \exp(-\pi x^2)$, i.e. the fourier transform of a gaussian is a gaussian, so it was not necessary that I create this filter in the fourier domain. To make the filter, I multiplied a gaussian in x with a gaussian in y :



The results of this filter are shown below:



Figure 4: Original image and fourier-domian, gaussian low pass filtered image. Notice the slight blurring in the filtered image.

2 Part 2: Phase Correlation

Please see `problem_2.py` for my implementation of this part. Please note that to run the code you will need `numba`.

First, I implemented raw phase correlation with no low pass filtering. This works according to the following equation:

$$p(x, y) = \mathbb{F}^{-1} \left[\frac{F^*(u, v)G(u, v)}{|F^*(u, v)G(u, v)|} \right]$$

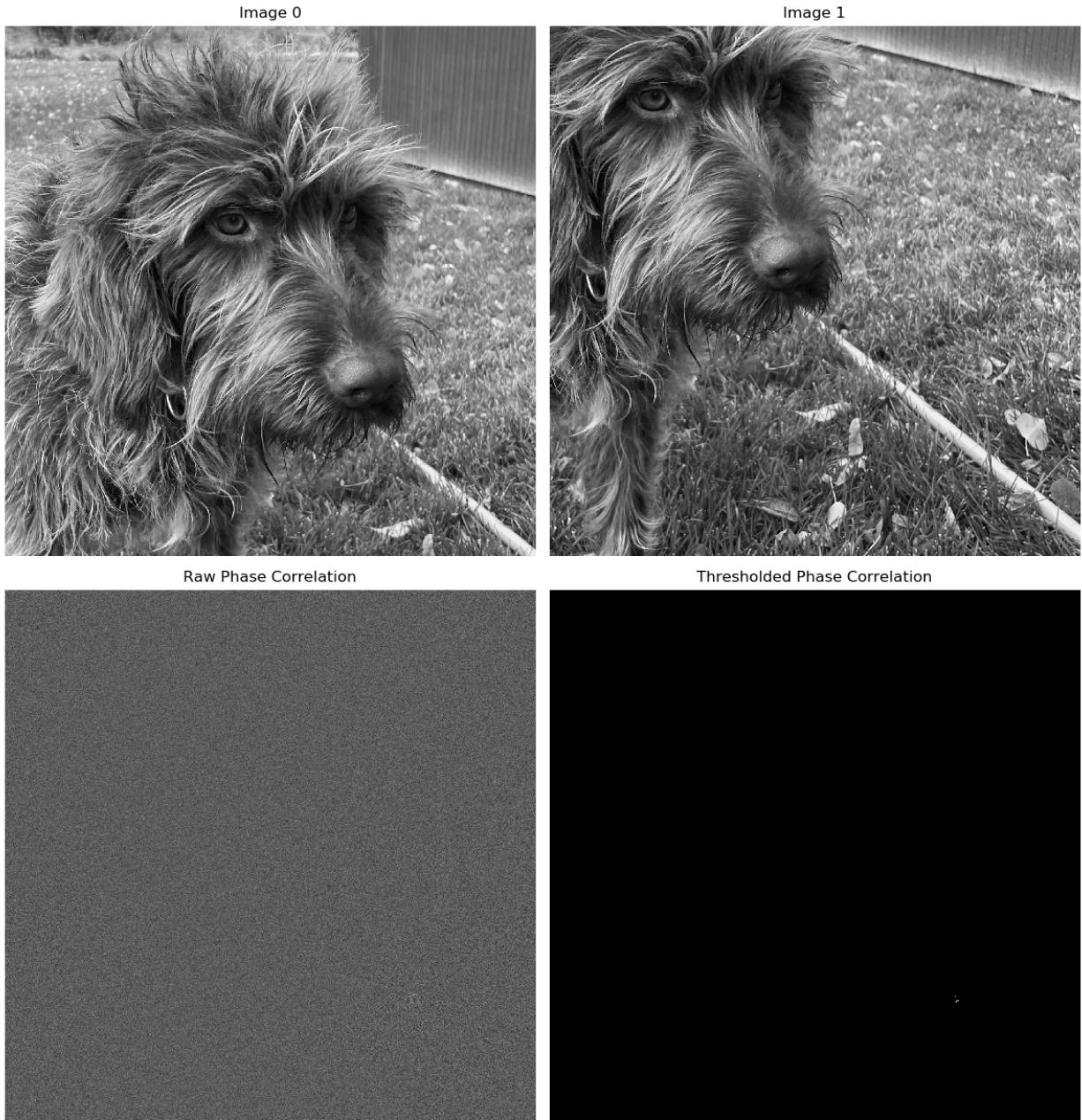


Figure 5: Phase correlation of images 0 and 1. The raw correlation data is shown on the left, and a thresholded version is shown on the right. The thresholded version is provided because it is much easier to interpret. See the lower right corner for the pixel of highest intensity.

Next, I implemented a low pass filter on the above phase correlation before I took the inverse fourier transform. I used the same gaussian low pass filter from part 1 for this. Curiously, I think these results are worse.

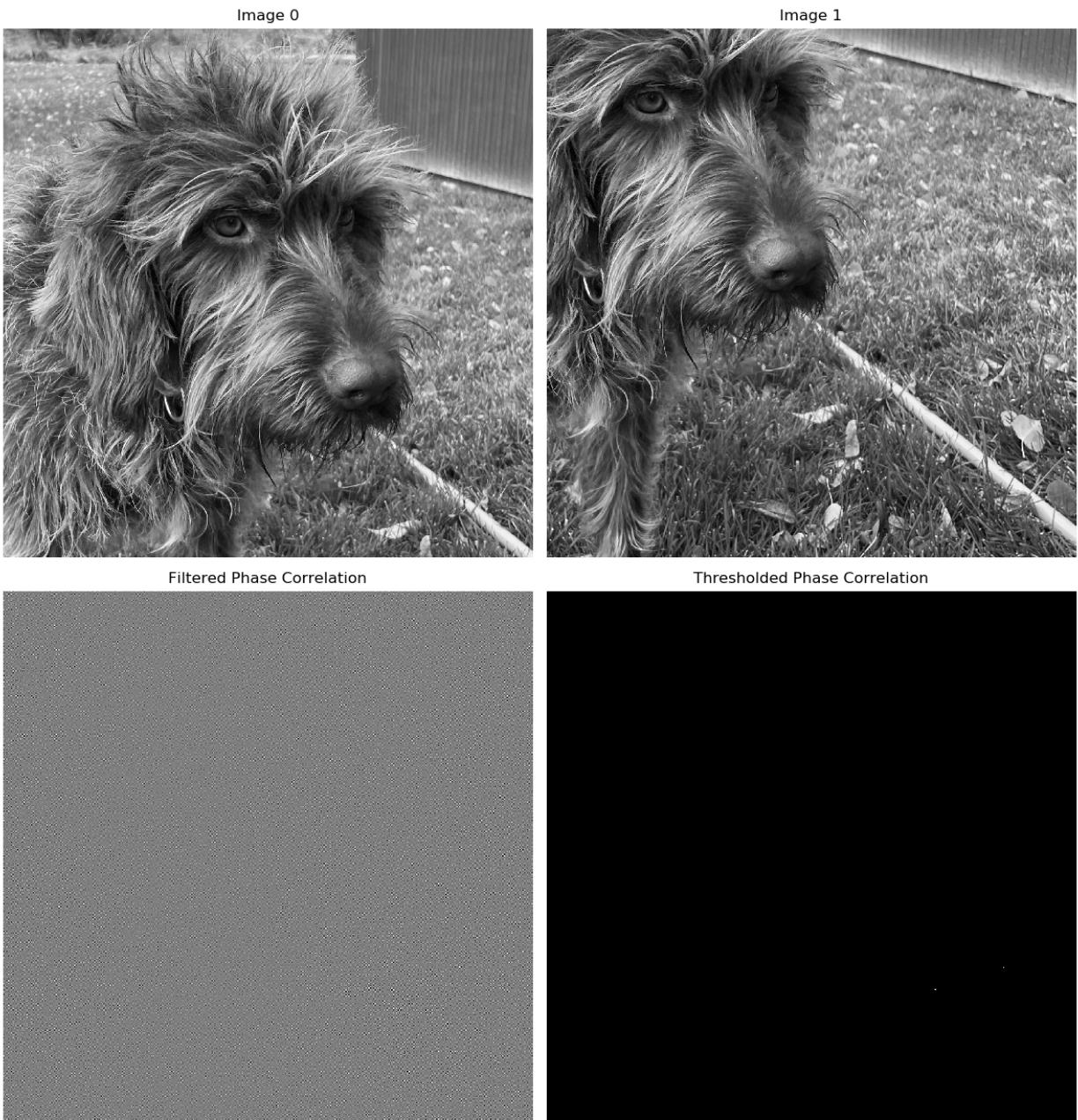


Figure 6: Phase correlation of images 0 and 1. The low pass filtered correlation data is shown on the left, and a thresholded version is shown on the right. The thresholded version is provided because it is much easier to interpret. It is slightly harder to see, but there is a white pixel at the location of the highest intensity in the lower right corner.

3 Peak Finding

3.1 Algorithm Development - Naïve Implementation

We want to compute the phase correlation of two overlapping images and find the relative location of one with respect to the other so that when we overlay them they line up. Without loss of generality, we will call these images image 1 and image 2.

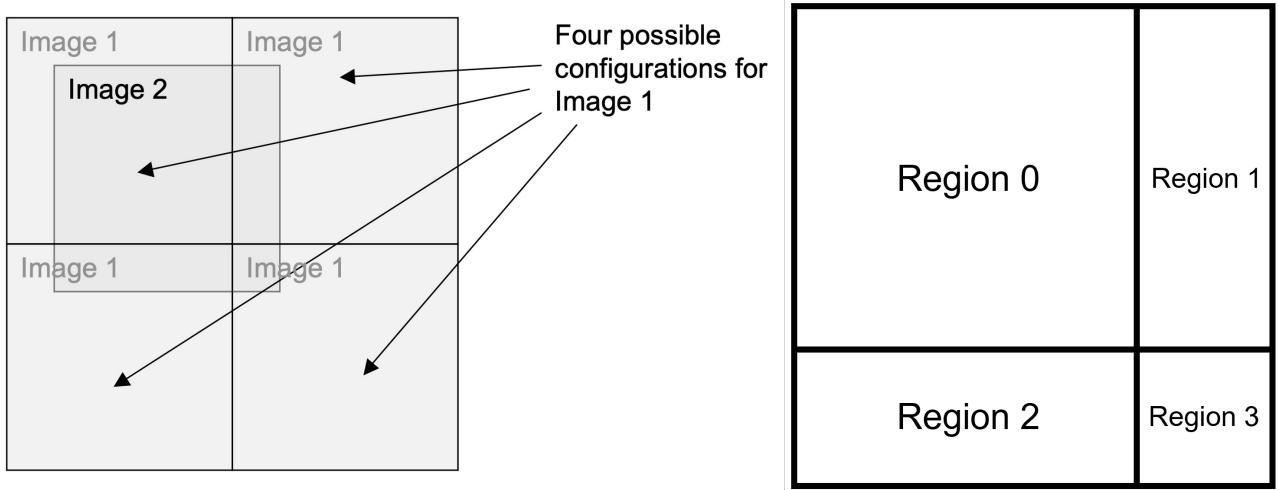


Figure 7: Possible locations of image 1 with respect to image 2, and the regions in image 2 defined by these relative locations.

Suppose image 1 is an array of mn elements, where m is the number of rows and n is the number of columns. We can only compute the phase correlation of two images if they are of the same size, so it follows that image 2 is an mxn array as well. If we compute the phase correlation of images 1 and 2, we have an image where the value of each element is the phase correlation at that location. We can find the indices of the maximal element in the array and call this location (λ_x, λ_y) . Note that in this case, the axis corresponding to the second dimension is defined as positive when we traverse *down* the image, contrary to our intuition from Cartesian axes. We are using python, so as far as the computer is concerned we can say that $(\lambda_x, \lambda_y) \equiv \text{image2}[\lambda_y, \lambda_x]$.

Region 0: Consider the placement of image 1 on top of image 2 that defines region 0. In this case, we wish to place the origin of image 1 (the upper left corner) a directed distance of $((-(n - \lambda_x), -(m - \lambda_y)))$ from the origin of image 2. This means that region 0 in image 1 is defined as the rectangle whose vertices are located at $(n - \lambda_x, m - \lambda_y)$, $(n, m - \lambda_y)$, $(n - \lambda_x, m)$, and (n, m) . We want to compute the phase correlation of this region with the same region in image 2, whose vertices are located at $(0, 0)$, $(\lambda_x, 0)$, $(0, \lambda_y)$, and (λ_x, λ_y) . Once we have computed the phase correlation of these two regions, we can record the maximum intensity of the phase correlation.

Region 1: Consider the placement of image 1 on top of image 2 that defines region 1. In this case, we wish to place the origin of image 1 a directed distance of $((\lambda_x, -(m - \lambda_y)))$ away from the origin of image 2. Thus, the vertices of region 1 in image 1 are given by $(0, m - \lambda_y)$, $(n - \lambda_x, m - \lambda_y)$, $(0, \lambda_y)$, and $(n - \lambda_x, m)$. The vertices of region 1 in image 2 are $(n - \lambda_x, 0)$, $(n, 0)$, $(n - \lambda_x, \lambda_y)$, and $(n - \lambda_x, m)$. We then compute the phase correlation of these two regions and record the maximum intensity of the phase correlation.

Region 2: Consider the placement of image 1 on top of image 2 that defines region 2. In this case, we wish to place the origin of image 1 a directed distance of $((-(n - \lambda_x), \lambda_y))$ away from the origin of image 2. Thus, the vertices of region 2 in image 1 are given by $(n - \lambda_x, 0)$, $(n, 0)$, $(n - \lambda_x, m - \lambda_y)$, and $(n, m - \lambda_y)$. The vertices of region 2 in image 2 are $(0, \lambda_y)$, (λ_x, λ_y) , $(0, m)$, and (λ_x, m) . Once again, we compute the phase correlation of these two regions and record the maximum intensity.

Region 3: Finally, consider the placement of image 1 on top of image 2 that defines region 3. In this case, we wish to place the origin of image 1 a directed distance of $((\lambda_x, \lambda_y))$ away from the origin of image 2. Thus, the vertices of region 3 in image 1 are given by $(0, 0)$, $(n - \lambda_x, 0)$, $(0, m - \lambda_y)$, and $(n - \lambda_x, m - \lambda_y)$. The vertices of

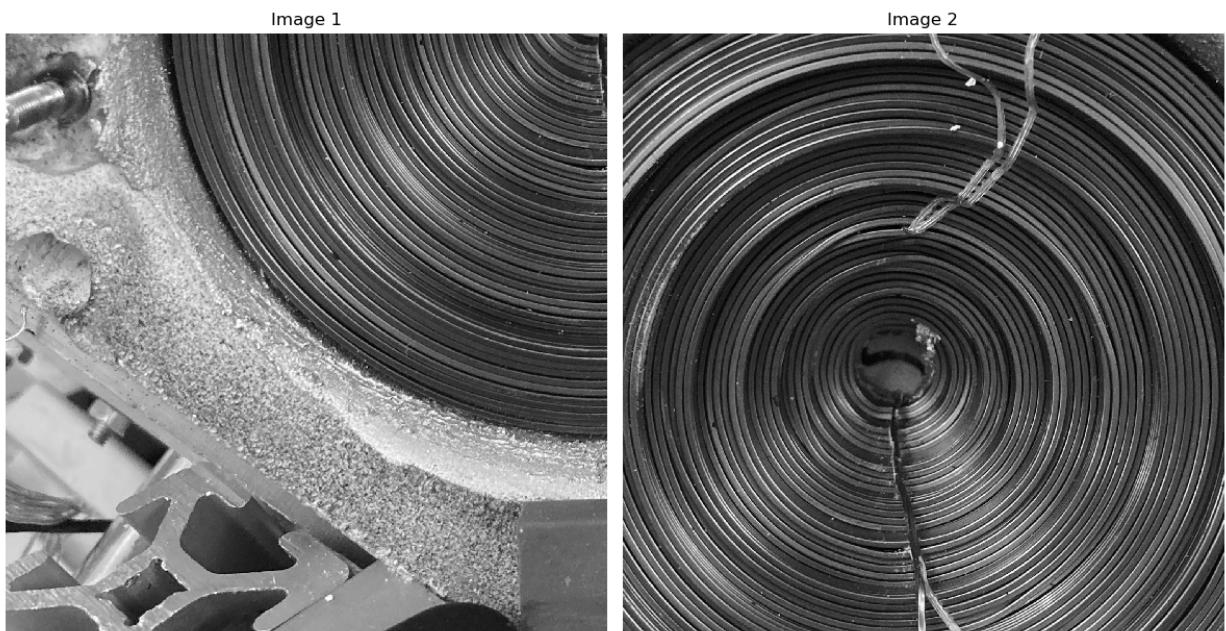
region 3 in image 2 are (λ_x, λ_y) , (n, λ_y) , (λ_x, m) , and (n, m) . For the last time we compute the phase correlation of these two regions and record the maximum intensity.

Now we have four maximum intensities, one coming from each region. We expect three of them to be meaningless (though not necessarily of the same value or order of magnitude) and one to be the maximal among the four. The region which contributes this intensity dictates the appropriate placement of image 1 on top of image 2.

Now we must define a canvas on which to place the images so we can visually inspect the results of this approach. We can easily determine the size of the canvas based on our new knowledge of the correct location of image 1 with respect to image 2. If we end up with region 0, the canvas must have $(2n - \lambda_x)$ columns and $(2m - \lambda_y)$ rows. If we end up with region 1, the canvas must have $(n + \lambda_x)$ columns and $(2m - \lambda_y)$ rows. If we end up with region 2, the canvas must have $(2n - \lambda_x)$ columns and $(m + \lambda_y)$ rows. Finally, if we end up with region 3, the canvas must have $(n + \lambda_x)$ columns and $(m + \lambda_y)$ rows.

3.2 Results - Naïve Implementation

Using the above algorithm I was able to take the following pairs of images:



And stitch them together into the following mosaics:

Two-image Mosaic



Two-image Mosaic



I have called this the “naïve implementation” because I am simply finding the maximum phase correlation and locating purely based on that metric. This has no built in low-pass filtering method (like finding centroids of connected components, or some other clever method) and could be susceptible to unforeseen difficulties relating to high frequency noise. However, this is unlikely because we can think of high frequency noise as being related to the number of sharp edges in an image. But, both of these images are composed of many high frequency signals so if this method really were so delicate, these particular images should expose this weakness. However, looking at the results we can see that these reconstructions are nearly perfect, indicating that this is a valid way of constructing mosaics that appears to be immune to high frequency noise.