MO446 – Introduction to Computer Vision
Project 1

Breno Leite
Guilherme Leite

28/08/2017

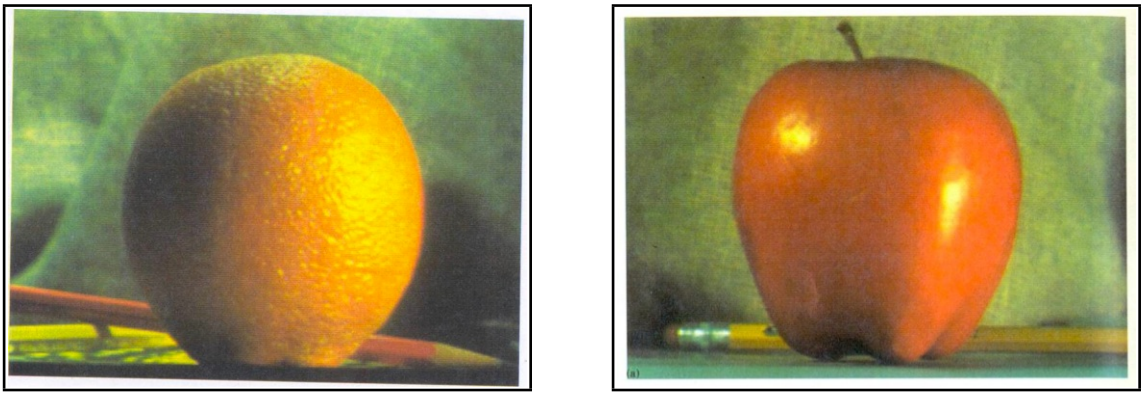
Input Images

Throughout this project some images were used as input to test the algorithms. Figure 1 was used as input for the pyramids exercises **2.1, 2.2, 2.3** and **3.1**, its dimensions are 400x300 and it is a colored image.

Figure 1: Input image for pyramids and Fourier transform exercises. (**p1-1-0**)

Important note: *The borders seen in the figures are not part of the image, they are figurative information about the starting and ending points of the image. Moreover, all the image scales in this report were changed in order to make the text more readable.*

The Figures in 2 are used for the blending exercises (**2.4** and **3.2**), all of them have dimensions 540x392 and are colored. Note that the images are slightly rotated, this will affect some results which comes from the image and not by any error on the process itself.



(a) Image 1 (**p1-2-4-0**) (b) Image 2 (**p1-2-4-1**)

Figure 2: Images 1 and 2 used for the blending operations.

Question 2 - Spatial blending

2.1) In the spectrum of image processing, convolution can be understood as the action of applying a mask to an image, in this case every pixel in the image is affected by the weighted sum of the filter applied to the pixel neighborhood. This approach requires some special attention when dealing with the edges of the image, since the application of the mask will overshoot the image borders, our solution to this problem was to extend the original image filling it's new borders with zeros, as seen in Figure **??**, by doing so the convolution simply ignore those values and don't account them into the result, a minor downside to this approach is the lessen effect of the mask around the edges, this solution was chosen for it's simplicity to implement. Additionally the convolution was tested with three smoothing masks, 3x3, 7x7 and 15x15, in comparison to the embedded solution by OpenCV our convolution was noticeably slower as seen in Table 1. As expected the masks smooth the edges around the image and suffers with a darkening close to the borders as seeing in Figure 3, with bigger masks the image suffered more distortion to the point of loosing all of its fine details, see Figure 4, the loss of such details is accounted by the size of the neighborhood that affect each pixel, thus a larger mask smooths more than a smaller one.

	Time (ms)		
Mask Size / **Convolution**	**3x3**	**7x7**	**15x15**
Implemented	5.123	4.8681	5.238
OpenCV	0.006	0.004	0.019

Table 1: Comparission of time between our implementation and OpenCV convolution.

2.2) To store the gaussian and laplacian pyramids of this project we chose a list, in which each node of it holds a level of the pyramid. Since the data structure is a list the access method is as in a array, $pyramid[i]$ returns the image in the ith pyramid level, disregarding then, the necessity to implement an access function. The strategy chosen to implement the interpolation was the bilinear interpolation. These decisions were taken regarding the time to implement and simplicity to understand. It is also worth noting that due to some confusion about the meaning of Up and Down,

Figure 3: Convolution with 3x3 mask (**p1-2-1-0**)

Figure 4: Convolution with 15x15 mask (**p1-2-1-2**)

these functions were renamed as follow: Expand referring to the action of expand the image width and height, and Contract analogously. Figure **??** shows the pyramid formed with the implemented function.

2.3) The laplacian pyramid demanded more attention to it's details, such as the fact that the last level of the pyramid is a copy of the same level at the gaussian pyramid. This particular level is used to perform the reconstruction of the original image, first the image is extended and interpolated, to match the previous level size, then it is summed with the previous level of the pyramid, the repetition of these steps result in the original image that was compressed in the pyramid, Figure **??** shows the laplacian pyramid.

2.4) In this exercise the blending was achieved by the following steps:

Starting at the top of the laplacian pyramid Apply the mask and its negative to each image blend the images and reconstruct the previous pyramid level repeat

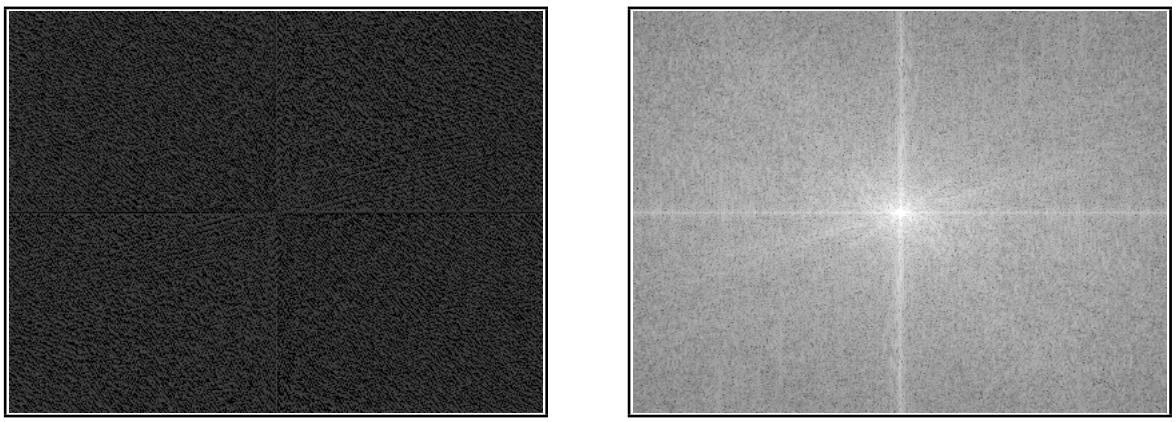
Question 3 - Frequency Blending

In this question, we will present some experiments developed using *numpy* and *OpenCV* function in order to transform the image from the spatial domain to the frequency domain. The process will be divided into two different experiments.

On the first one (**3.1**), we will use the transformation and make some modifications to the phase and magnitude in order to see the impact on the image recontructed. On the second experiment (**3.2**), we will be blending the two images used into the question **4.2** in order to check how it is compared to the spatial blending.

3.1) Exploring Fourier Space

In order to explore the fourier space, we implemented two functions. One to transform an image into magnitude and phase, and the other to reconstruct the image from magnitude and phase. The Figure 5 shows the magnitude and phase obtained transforming the Figure 1 to the frequency domain.



(a) Phase Image (**p1-3-1-0**) (b) Magnitude Image (**p1-3-1-1**)

Figure 5: Phase and Magnitude images created on the fourier transform using image **p0-1-0**.

In order to create these images, the results of the fourier transform were shifted to the center and then reduced by the functions, $20 * log_e(magnitude)$ and $40 * log_e(phase)$. These values were used to mantain the values between 0 and 255, which could be showed as an image. These values are translated back when doing the reverse of the fourier transform.

The Figure 5a shows the phase of the image, it is hard to obtain some information from this image. However, it is respective to the directions of the image. Moreover, the Figure **??** shows the values to the magnitude, the intensity shows the values of the magnitude. In both Figures, light pixels indicates high values.

We can have a little more information with the magnitude image, it is noticeably that the image is bright which indicates that the image on Figure **??** has more high frequency than low frequencies.

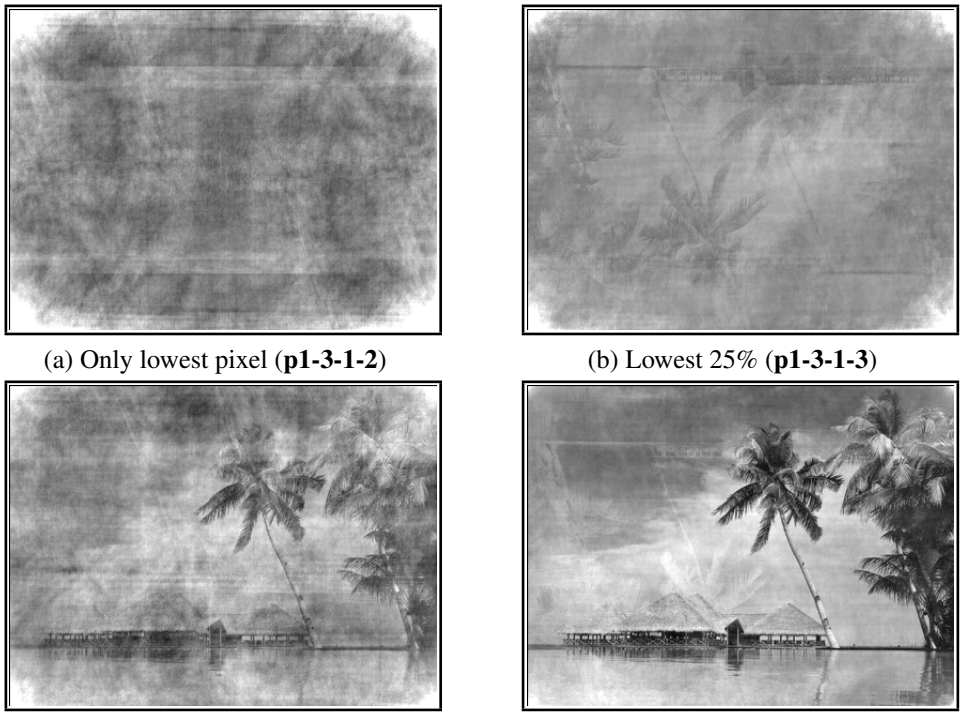
In the next experiments, we will be changing the values of the phase and magnitude images before doing the reverse of Fourier transform. This process should reconstruct the original image, we will be changing the images by percentages. So, in every experiment this process was made using 1 pixel, 25%, 50%, 75% and 100% of the pixels from the phase or magnitude image.

This pixels might be extract in two different ways, greatest or lowest (e.g. 25% lowest pixels on phase image). In the image reconstruct using 100% of the pixels all the images were perfectly reconstruct, a effect which was expected (Figure 6). All the following experiments were made using the Figure 1 in grayscale, the grayscale was used for more simplicity and further on one experiment will show the results in a colored image.

Figure 6: Reconstructed image after inverse Fourier transform using 100% of pixels in magnitude and phase images. (**p1-3-1-6, p1-3-1-11, p1-3-1-16, and p1-3-1-21**)

In order to show the differences when changing the phase or magnitude image, we will divide the experiments into two parts:

- **Phase** - First we will be changing the phase image and performing the inverse of Fourier. In this experiment we were expecting to the position of the objects on the image to move. The Figure 7 show the results when using only the lowest values of the phase image.



(a) Only lowest pixel (**p1-3-1-2**)

(b) Lowest 25% (**p1-3-1-3**)

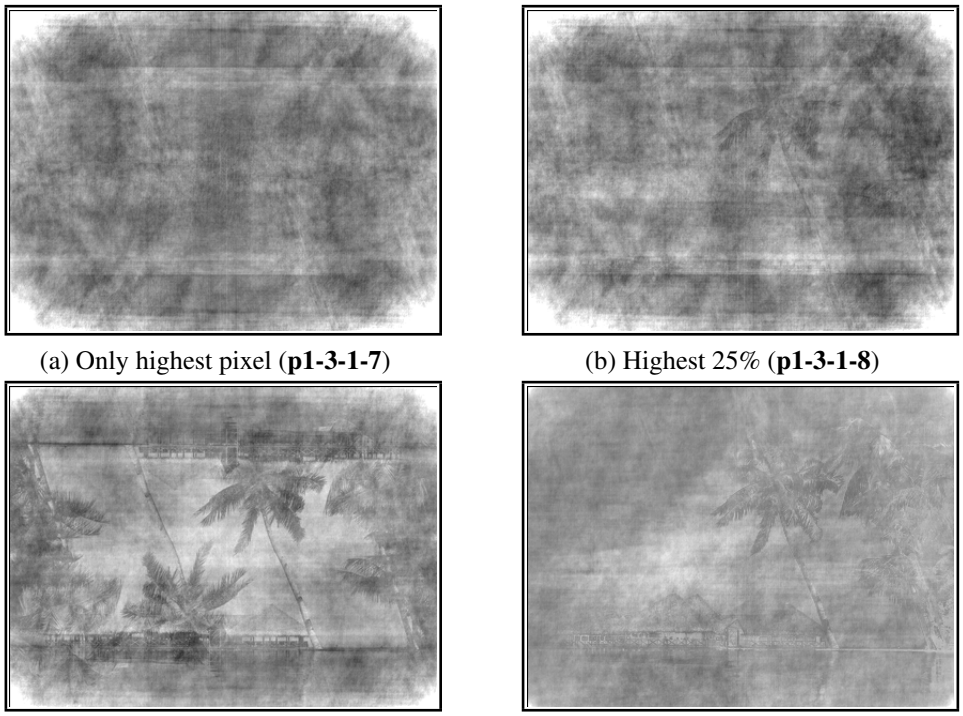
(c) Lowest 50% (**p1-3-1-4**)

(d) Lowest 75% (**p1-3-1-5**)

Figure 7: Results of inverse Fourier transform using the lowest values on the phase image.

As we can see, the image shows us information about the orientation of the image. In this way, selecting different values changes the orientation of the image. Another interesting fact is the upside down elements, which occurs because of the range in which the transform is working, from 0 to 380 degrees.

In other hand, the Figure 8 shows the results when using the highest values to the phase image.



(a) Only highest pixel (**p1-3-1-7**) (b) Highest 25% (**p1-3-1-8**)

(c) Highest 50% (**p1-3-1-9**) (d) Highest 75% (**p1-3-1-10**)

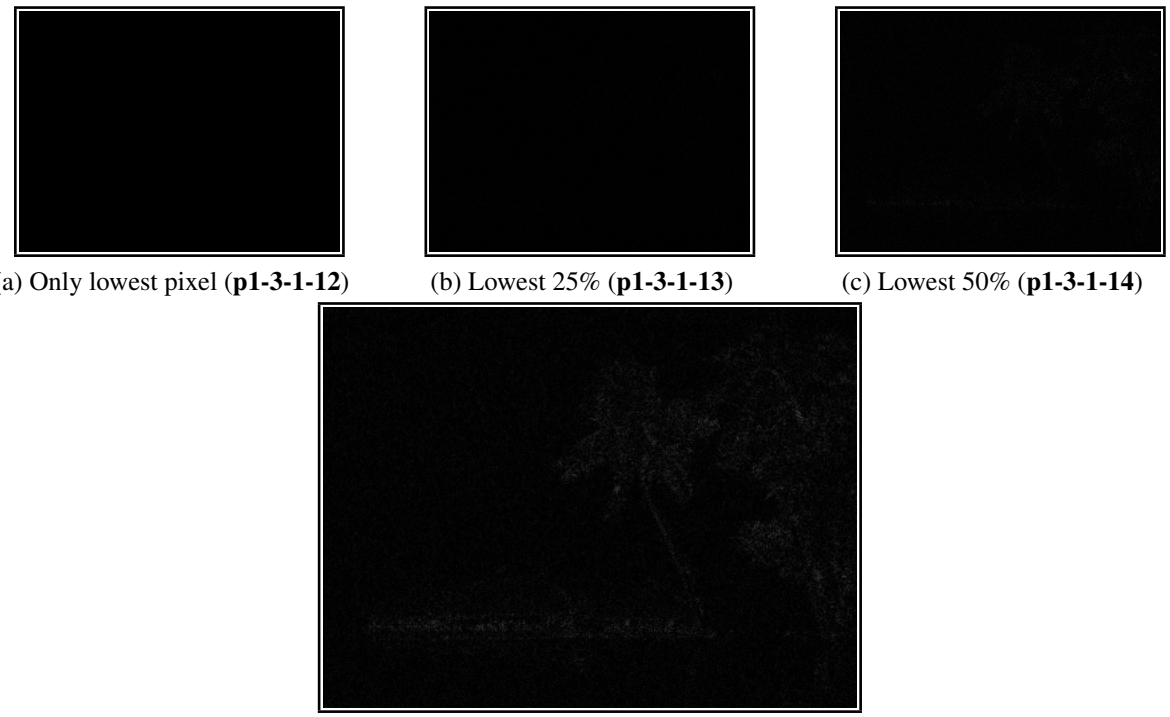
Figure 8: Results of inverse Fourier transform using the greatest values on the phase image.

An interesting feature is that the images contain complement information, as example the Figure 8b has less information than Figure 7d, which completes the information missing on the first image. The same effect might be seen in the other images.

- **Magnitude** - In this part, we will be doing similar experiments, but this time, changing the magnitude image. In this part, we expect to see different results on the intensity of the image, which is described by the magnitude on the frequency domain. The Figure 5b shows the different images obtained on the inverse Fourier transform.

  By doing this operation, we are using only the low frequency pixels from the image. The first row on image is almost all black pixels, the reason for that is the high frequency on the image used. However, we can see some of the contours being marked in Figure 9d. This effect produces a image similar to the convolved image on Figure **??**, which is a High-Pass-Filter. Moreover, The Figure 10 show the same process using the high frequency on the magnitude.

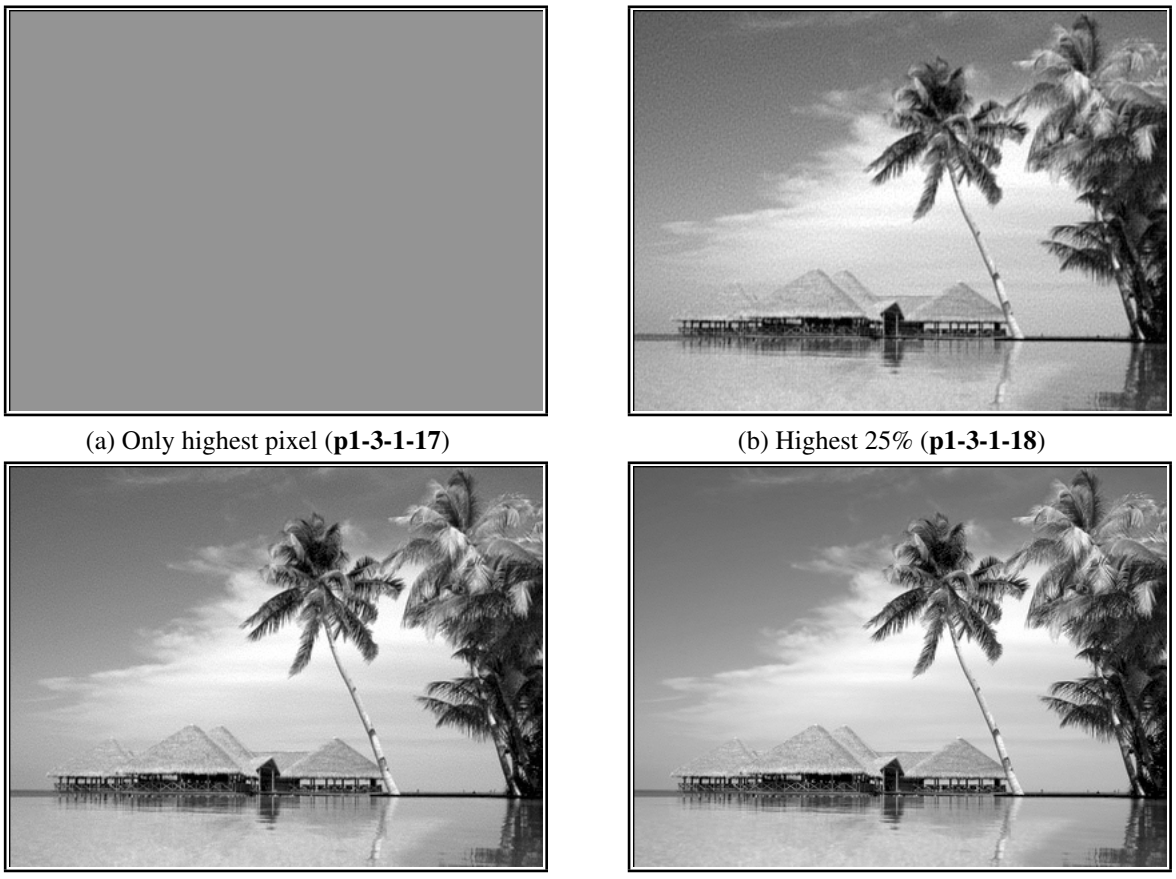
  As we would expect, the high frequencies on the image were maintained. So, even the using only 25% of the pixels on magnitude image the reconstructed image is really good. We can note that the quality of the reconstructed image is proportional with the percentage used on the magnitude image. The first image (Figure 11a) does not have enough information to reconstruct the image, but the color information is showing the predominant color intensity (with just the highest pixel).



(a) Only lowest pixel (**p1-3-1-12**) (b) Lowest 25% (**p1-3-1-13**) (c) Lowest 50% (**p1-3-1-14**)

(d) Lowest 75% (**p1-3-1-15**)

Figure 9: Results of inverse Fourier transform using the lowest values on the magnitude image.

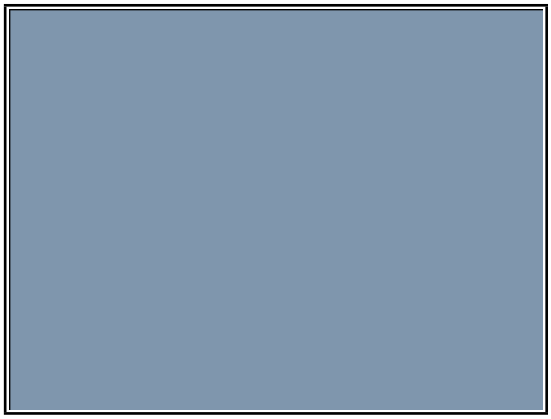


(a) Only highest pixel (**p1-3-1-17**) (b) Highest 25% (**p1-3-1-18**)

(c) Highest 50% (**p1-3-1-19**) (d) Highest 75% (**p1-3-1-20**)

Figure 10: Results of inverse Fourier transform using the highest values on the magnitude image.

In order to show that our implementation could be used with colored images, we used same experiment as before. The Figure 11 shows the result of using the highest values on the magnitude image on a colored image.



(a) Only highest pixel (**p1-3-1-22**)

(b) Highest 25% (**p1-3-1-23**)

(c) Highest 50% (**p1-3-1-24**)

(d) Highest 75% (**p1-3-1-25**)

(e) Highest 100% (**p1-3-1-26**)

Figure 11: Results of inverse Fourier transform using the highest values on the magnitude image using colored images.

In order to use a colored image, the Fourier transformation was processed three times, one for each color channel. Note that the Figure 11e is the reconstruction of the Figure 1, which indicates that the process of the Fourier transform was successfull.

3.2) Blending

In order to do the blending on the frequency domain, we first transformed both images from spatial to frequency domain, using Fourier transform. After this, we have two vectors for each image representing the magnitude and phase values. Our first approach was to sum the vectors together, this approach obviously did not work (Figure 12). The phase vector, responsible to the imaginary part (on polar coordinates), could not be summed in this way, the reason is we would lose information.

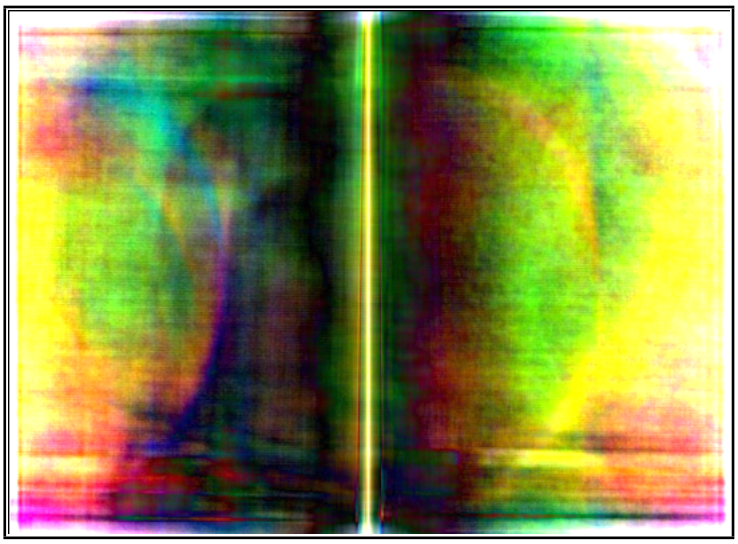


Figure 12: Blending adding phase and magnitude vectors.

After this approach, we have tried to implement the blending using a different values for the images into the magnitude as phase. As example: we would get 50% of highest values of both images and them sum them up. We were expecting that the sum of the zeros, inserted on the 50% lowest values, would work using a certain mask. This approach was not a disaster, but it still a bad blending over all (Figure 14).

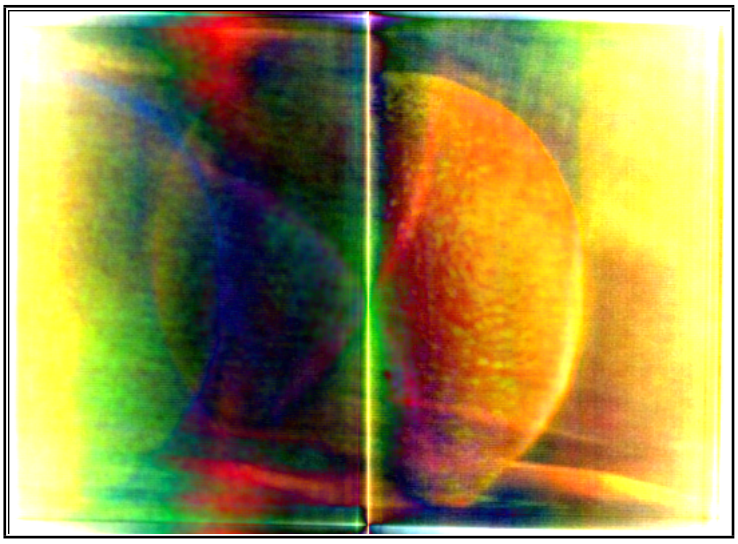


Figure 13: Blending adding phase and magnitude vectors using only 50% highest values.

Also these tries, we started thinking about our results. Our conclusion was that no merging between the phase and magnitude would work as a blending, moreover, we concluded that our approach was not working as expected. Looking at the images we could see that a great number of information was being lost, even just adding them or trying to use a mixture of pixels from magnitude and phase.

After some thinking, we figure it that the summation should work. However, we were dealing with complex numbers and we were adding the phase as it was a normal number. In order to fix it, we first construct a *numpy* complex function using $magnitude * np.exp(1j * phase)$, in which $magnitude$ and *phase* are vectors. After creating one function for each image, we were able to sum them up, which the *numpy* would do using the correct complex summation. The Figure 2 shows the result of this process.

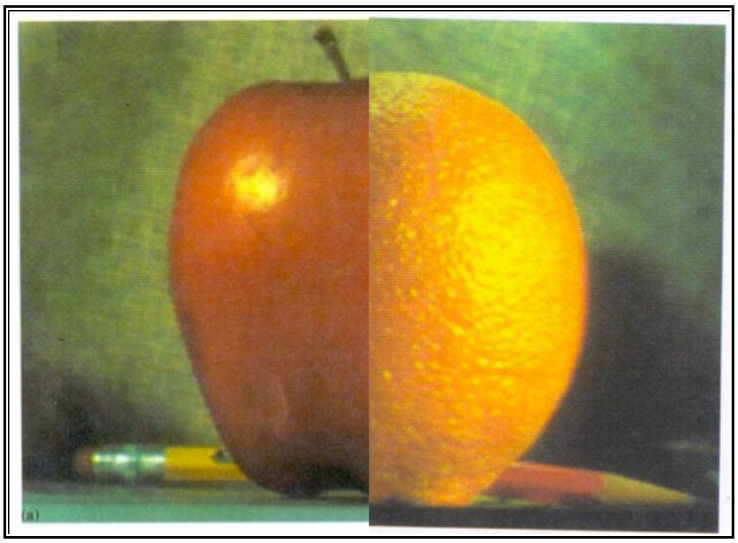


Figure 14: Blended image using the *numpy* summation of complex numbers.

The generated image is a blend of two images, but if it is compared to the blending generated using pyramids (Figure **??**) the contrast of the images are too high. After that some filtering approaches were tried to solve the contrast, but nothing gave a better result.

We then tried to use the pyramids with the frequency domain, but we had problem with the up sampling and down sampling. Because we could not change pixels arbitrary, mainly because one pixel on frequency domain represents more information from the image, as seen in exercise ***3.1*** *Exploring Fourier Space*. We did not have enough time to explore more this blending, but our intuition is that it might be possible if some "tricks" are applied to the frequency domain in order to blend the images.