Image hybridization using bi-dimensional Fourier transforms and laplacian pyramids

Mauricio Neira
Universidad de los Andes
Cra 1 N^o 18A - 12, Bogotá - Colombia
m.neira10@uniandes.edu.co

Daniel Rodriguez
Universidad de los Andes
Cra 1 N^o 18A - 12, Bogotá - Colombia
da.rodriguez1253@uniandes.edu.co

Abstract

In this paper, we produce 2 distinct types of hybrid images using an algorithm in the frequency domain. First, we apply a bi-dimensional Fourier transform to produce an image whose interpretation depends on the viewer distance. Then, we smooth out differences between 2 similar images that have been cut in half and juxtaposed to produce an image that looks like a natural blur. We display compelling images that show the final effect of our algorithm. Finally, we discuss how to better implement this application with the aid of non-supervised filtering.

1. Introduction

An image is described by its collection of boundaries, shapes and textures. As humans, we recognize objects coarsely from their lower frequency properties and details from the high frequency components. The objects we detect in an image also depend on the relative distance to it. Two very famous exponents of this effect are the paintings Slave Market with the Disappearing Bust of Voltaire (1940) and Lincoln in Dalivision (1967) by Salvador Dalí. The first painting is known as *picture mosaic* or images which have a local and a global interpreation depending on their arrangement. The second paintings is a *hybrid image* or images with two global interpretations that vary according to the viewer distance. Today, these latter images are used as visual demonstrations in Science Museums. A database of theses examples can be found at: However, there are other applications to these principle as object detection and cryptography. As an example many algorithms detect broad objects by filtering out the high frequencies of an image to identify its bold borders. Other applications include incorporating information in the higher spectra of an image, which is unidentifiable to the naked human eye.

2. Related work

Pioneering work in hybrid images was done by A. Olivia et al. using Gaussian kernels to produce low pass filters and high pass filters over different images[2]. The results were added arithmetically pixel by pixel to produce the hybrid image. The hybrid image H can thus be expressed as[2]:

$$H = I_1 \cdot G_1 + I_2 \cdot (1 - G_2) \tag{1}$$

Where I_1 is the image with the low-pass filter and, I_2 the image with the high pass filter and G_1, G_2 their respective Gaussian kernels.

3. Images used

The *hybrid effect* works best when the two images have similar borders, orientations and colour. Thus, we selected images of politicians who look alike[1]. The original image was a banner containing the two pictures side by side. The banner was split into 2 equal parts, one containing each polititian.

For our second application we took images from two famous anime characters, which have similiar traits. We transformed our images into the color space RGB and also resized them to 256×256 to ease our algorithm. Minor differences are required to implement the algorithm to images with any size.

4. Procedure

4.1. Image hybridization

Similar to equation 1, a low pass filter and a high pass filter were applied to the different images. However, unlike the work done by A. Olivia et al., these filters were done in the frequency domain of the images through the Fourier

[1]The original image was taken from this post: https://www.semana.com/confidenciales-semanacom/articulo/duque-se-refiere-a-alvaro-uribe-como-presidente/601095

transform. Ultimately, the hybrid image H can be expressed as:

$$H(r) = F^{-1}\left[L(F[I_1], r)\right] + F^{-1}\left[I_2 - L(F[I_2], r)\right] \quad (2)$$

Where I_1, I_2 are the images that comprise the hybrid, L the low pass filter, r the threshold of the low-pass filter and F, F^{-1} the discrete, bi-dimensional Fourier transform and its corresponding inverse.

Thus, first step of the procedure was to apply the bi-dimensional Fourier transform to the images. Formally, it is defined as[3]:

$$F(u,v) = \frac{1}{NM}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)e^{-2\pi i\left(\frac{ux}{N}+\frac{vy}{M}\right)} \quad (3)$$

Its corresponding inverse is[3]:

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{M-1} F(u,v)e^{-2\pi i\left(\frac{ux}{N}+\frac{vy}{M}\right)} \quad (4)$$

Where $f(x,y)$ is the value of the image at pixel (x,y), N is the width of the image, M the height of the image and $F(u,v)$ the value of the image in its frequency domain at pixel (u,v).

The implementation of the fourier transform and its inverse can be done using *scipy*'s functions *fft* and *ifft*,[1] as shown below:

```
def fourier(im):
    res = []
    for i in range(3):
        res.append(
            fftpack.fft2(im[:,:,i])
            )
        res[i] = np.fft.fftshift(
                                res[i]
                                )
    res = np.dstack((
                    res[0],
                    res[1],
                    res[2]))
    return res
```

It is important to note that fast Fourier transform will produce an image whose quadrants are shuffled. The image will not have the peak wavelengths unless these cuadrants are reordered[2]. This is done with *numpy*'s function *fft.fftshift*. As was done in the code.

[2]As explained in this stackExchange post: https://dsp.stackexchange.com/questions/37320/shifting-in-fft

Lastly, the image comes in RGB format. The fourier transform needs to be applied to every single channel sepparately. That corresponds to the indices $0, 1, 2$ in the code above.

Similarly, an analoguous implementation for the inverse Fourier transform can be done:

```
def ifourier(im):
    res = []
    for i in range(3):
        res.append(
            np.fft.fftshift(im[:,:,i]))
            res[i] = fftpack.ifft2(
                res[i]).real
    res = np.dstack((
                    res[0],
                    res[1],
                    res[2]))
    res[res<0] = 0
    res[res>255] = 255
    res = res.astype(int)
    return res
```

Apart from the discussed in the implementation of the forward Fourier transform, due to rounding issues, the numbers assigned could be outside of the set $\mathbb{Z}\bigcap[0,255]$. Appropriate clipping and casting is done to ensure proper format.

After obtaining the images in their frequency domain, the low-pass filter was implemented. Since frequencies in the vicinity of $(0,0)$ in the transformed image are by definition the low frequencies, a cutoff threshold (radius) r was defined where any point of the image farther from the origin to this threshold was eliminated. this can be expressed as:

$$L(I(x,y), r) = \begin{cases} I(x,y) & \text{if } \sqrt{x^2+y^2} <= r \\ 0 & \text{if } \sqrt{x^2+y^2} > r \end{cases} \quad (5)$$

This means the low-pass filter has an adjustable parameter r that will determine the amount of low frequencies that will be taken from I_1 and subtracted from I_2. This threshold radius is the key parameter of our algorithm since it determines the proportion of the Fourier transform stored. This parameter will determine the resulting hybrid (and quite dramatically so).

This filter was implemented using the following 2 functions:

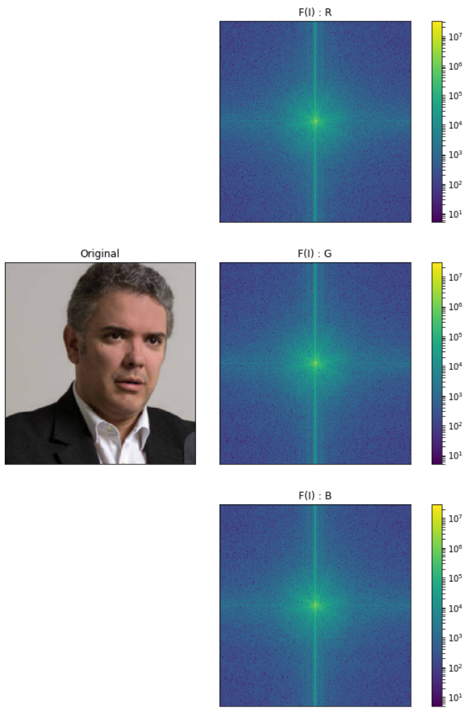


Figure 1. The bi-directional fast fourier transformation for each one of the RGB channels of the image whose low frequencies will be used.

```
def inCircle(x,y,r):
    if((x**2+y**2)**0.5<r):
        return True
    return False

def lowFilter(im,r):
    lenx,leny,n = im.shape
    for i in range(lenx):
        for j in range(leny):
            for k in range(n):
                if(not inCircle(
                        i-lenx//2,
                        j-leny//2,r)):
                    im[i,j,k]=0+0j
    return im

def highFilter(im,r):
    lowim = im.copy()
    lowim = lowFilter(lowim,r)
    return im-lowim
```

The code snippet above iterates through each of the 3 channels of the image and kills the pixels that lie outside the established threshold. The high-pass filter simply calculates the low-pass filter and subtracts it from the original image.

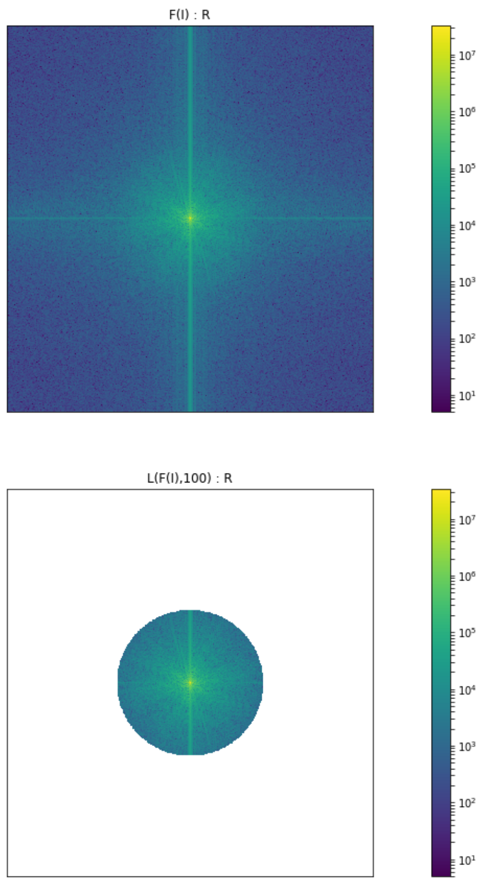


Figure 2. The R channel of the Fourier transform of Duque (up) and its low-pass filtered correspondence using an r threshold of 100.

The third step in the process is to return their images to their original space using the inverse Fourier transform (see equation 4). Much like what was done in the first step with the forward Fourier transform (equation 3).

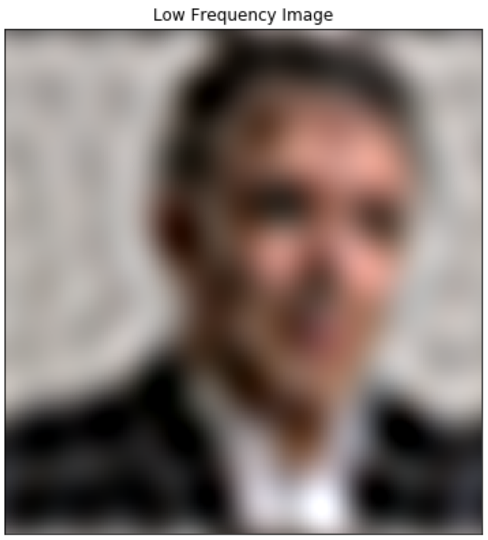
Finally, the two resulting images are summed arithmetically and the output is the hybrid.

A commented notebook with our image processing and results can be at:

```
https://github.com/mneira10/IBIO4490/
blob/master/04-Hybrid/Solucion.ipynb
```

4.2. Image

Given two images we attempt to merge their high frequencies into left and right halves. A first approach would simply extract high frequencies from each image, as in the previous section, and later combine them. However this creates a sharp border which feels unnatural (see image X). To



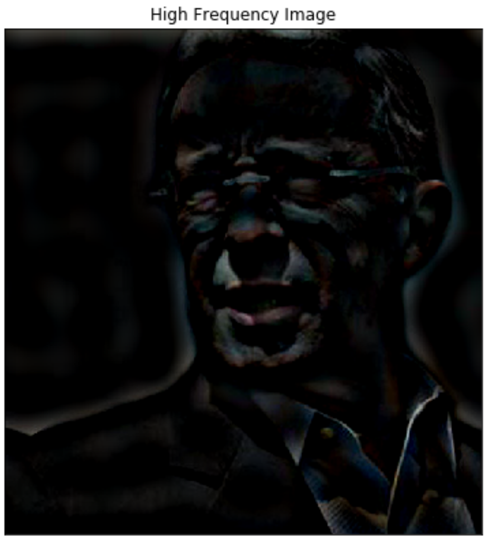


Figure 3. Images of Duque (top) and Uribe (bottom) obtained from the inverse Fourier transform of their low filter and high filter images in the frequency space.

bypass this problem we first extract the low frequencies of the merged image and later add high frequencies of each original image in the corresponding half. This procedure has the advantage of blurring the sharp border.

To do this we first crop images into halves by setting appropriate zeros. Then we sum up their halves to get a merged image with a very sharp border. Later we build Gaussian pyramids for each image separately. To do so we apply a Gaussian filter on the image and downsize it by half. We iterate this procedure at least 4 times. The result is shown in the next figure.

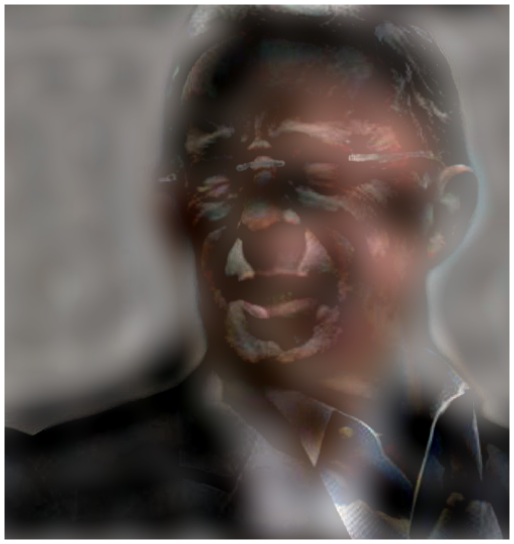


Figure 4. Resulting hybrid images from summing the two images in figure 3.

In order to do a proper merging we have to build also a Laplacian Pyramid. Laplacian pyramids are obtained as the difference between consecutive levels of the Gaussian pyramid (bearing into account the proper scaling). In the following code snippet we build Gaussian pyramids with the aid of the function pyrDown and pyrUp from the library Open CV. This function applies a Gaussian filter and resizes the image in half. Hence, our main contribution is properly handling the levels of the Gaussian pyramid to extract the Laplacian Pyramid.

After constructing the Laplacian Pyramids for the two images we blend them in the following heuristic. We start with the lowest frequency and lowest size image corresponding to the lowest level in the Laplacian pyramid. Then we increase its size two-fold and add the *details* stored in the Laplacian level of the corresponding size. We iterate this procedure until we are at the last level where the image reconstructed has the same size as the original image. The following code snippet displays our implementation of the previous method.

Another approach would be to extract the low frequency component of the merged image to obtain the coarse details of the image. We can do this using the function lowFilter implemented in the last section. Again, the threshold radius r will determine the quality of the output. After we add high frequency components of the original images on corresponding halves to recapture the high definition. Note that this two approaches are not equivalent. The latter, however, is faster and easier to implement.

5. Results

We have implemented an algorithm in the frequency domain to produce hybrid images and merged hybrid images. The key parameter of our simulations is the threshold radius r that determines the frequency cutoff. Working in the *time domain* (real images) allows for an easier calibration of the parameters. On the contrary, the Fourier transform is not very intuitive and one is led to several experiments to determine proper parameters. As future work we suggest on working in non-supervised processing that would identify a useful cutoff for the images. Such method could be based on the energy in the frequency domain of the signal or in simpler heuristics like histograms on the axis.

References

[1] Scipy.fftpack.fft — SciPy v1.2.1 Reference Guide. https://docs.scipy.org/doc/scipy/reference/generated/scipy.fftpack.fft.html.

[2] A. Oliva, A. Torralba, and P. G. Schyns. Hybrid images. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 527–532. ACM, 2006.

[3] M. S. Rzeszotarski, F. L. Royer, and G. C. Gilmore. Introduction to two-dimensional Fourier analysis. *Behavior Research Methods & Instrumentation*, 15(2):308–318, 1983.