Image Restoration

Cristina M. Morales Mojica, Giovanni Molina, Gustavo Aguilar, Sudipta Kar

Introduction

Image restoration is a technique that aims to improve the quality of an image. When an image is taken, it may contain degradation and noise due to the sensor, environmental conditions (light levels or temperature changes) or corruption during transmission or compression. In this project we implement different filters we can apply to obtain an image similar to the untainted image. We will discuss restoration methods applied in the spatial domain (Statistical Filters) and in the frequency domain (Periodic Filters). It is important to mention that for each domain we use a different approach when implementing the filters. In the spatial domain each pixel is independent of each other and to restore the image we apply convolution to the noisy image. On the other hand, the noise generated in the frequency domain shows a correlated noise and to apply the filter we multiply a mask by the DFT of the image.

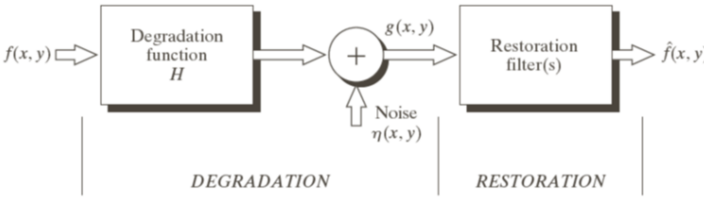


Figure 1. A model of the image degradation/restoration process.

Statistical Filters

The type of filters we implemented here are usually good at removing statistical noise, such as gaussian, salt and pepper noise which are often caused during image capture due to a faulty sensor, signal interference or environmental noise. This type of filter is applied in the spatial domain of the image, changing every pixel value individually. To do this, we define a window of size NxN which we use to convolve through the whole image (after zero-padding) and apply a filter.

We implemented three groups of statistical filters: Mean filters, Order Statistic Filters and Adaptive Filters. The first group (mean filters), consists of four simple filters that simply compute a different type of mean or average over all the values in the window during the convolution. The means are the

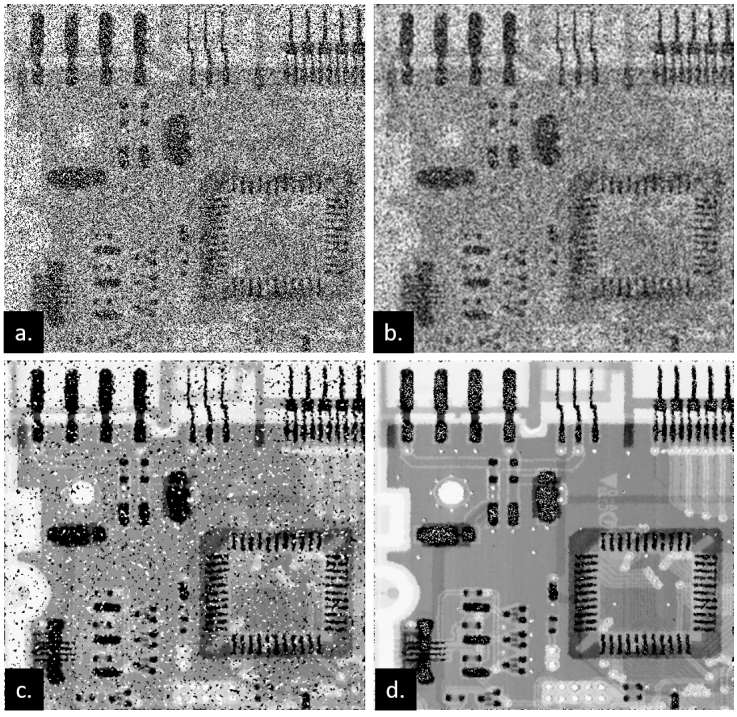


Figure 2 . Statistical filter examples. (a) Salt and Pepper noise image (b) Arithmetic Mean with window 3 (c) Median Filter with window 3 (d) Adaptive Median Filter with maximum window 7.

arithmetic mean, *geometric mean*, *harmonic mean* and the *contra-harmonic mean*. The second group of filters (order statistic), are also considerably simple. The *median, max* and *min* filters do exactly that, return the median, max and min value in the window. The *midpoint* filter returns the midpoint between the man and min value in the window and finally the *alpha-trimmed mean* filter returns the mean of a reduced (trimmed) portion of the window, which is obtained by removing the *d*/2 lowest and highest values.

The third and last group of filters we implemented are the adaptive filters. This filters are generally more computationally complex than the previous filters as they take into account information about the statistics of the noise in the image. To obtain these statistics, we select a section of the image from which we compute the mean and variance of the noise. Then we can use this information to compute the *adaptive local filter* which helps avoid blurring the edges of the images because it takes into account the local variance of the noise in the image. The other filter we implemented is the *adaptive median filter* which is similar to the median filter except that it dynamically increases the size of the window (until a maximum specified size) depending on the noise of the image. In Figure 2 we show an image with salt and pepper additive noise and the resulting images after applying different filters.

Periodic Filters

Periodic filters are methods applied in the frequency domain. The noise is easy to identify as they appear like 'white dots' in the image's DFT (Figure 3). In this section we will describe the implementation of four periodic filters: Bandreject, Notch, Inverse Filter and the Wiener filter.

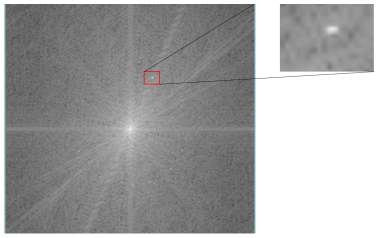


Figure 3 . Example of noise in the DFT

Bandreject and Notch

For the implementation of this two filters we assume that the image has no degradation and only has noise. The **bandreject** filter is a mask of a ring that is place on top of the 'white dots' that are generating the noise (Figure 5c). Because we could not find images with this description we generated our own noise were random pixels in a selected radius are changed to 255 to simulate this periodic noise (Figure 4). The **notch** filters, on the other hand, are black dots placed in the location where the white spikes are located. To identify the location of the noise, we binarized the

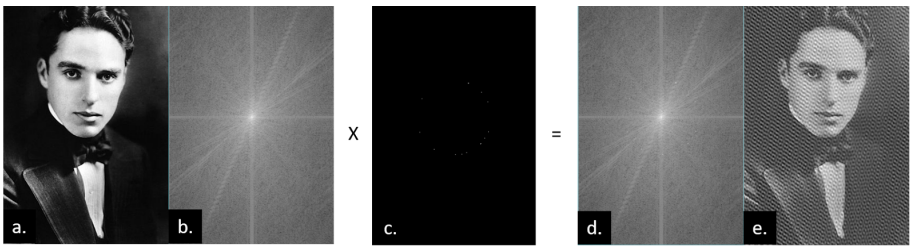


Figure 4 . Periodic noise generation pipeline. (a) Original image (b) DFT of the image (c) Mask to add noise, all pixels are 1 and those that are going to generate the noise are 255. (d) DFT with noise (e) Noisy image.

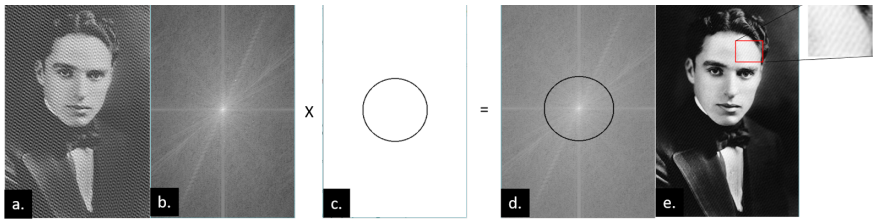


Figure 5 . Bandreject filter implementation. (a) Noisy image (b) DFT of the noisy image (c) Bandreject filter (d) DFT with the filter (e) Restored image with some artifacts.

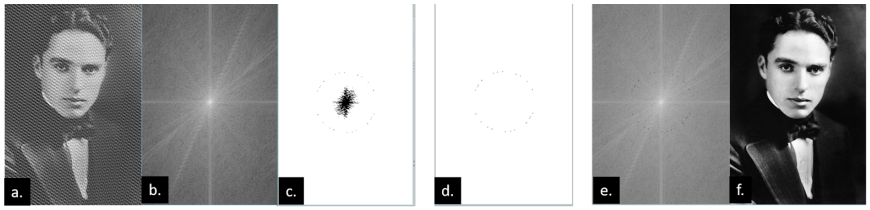


Figure 6 . Notch filter implementation. (a) Noisy image (b) DFT of the noisy image (c) Binarized image, threshold being the 4th standard deviation from the mean of the DFT histogram (d) Binarized image, with a removal box of 45, also the mask (e) Restored image with no artifacts.

DFT of the image using as threshold the mean of its histogram with the option for the user to add standard deviations to obtain better results (Figure 6c). Then the code proceeds with a blob detection algorithm. Because the center of the DFT has high frequencies, the algorithm identifies the center as noise. To address this problem, we added in the GUI the option to select a box to exclude notches in the center to be applied to the DFT with noise (Figure 4d).

Inverse Filter and Wiener Filter

The Inverse filter and the Wiener filter take into consideration the degradation function. There are different methods for identifying the degradation function of an image, but for this we would either need the equipment used and perform experiments or have detailed information about how the image was damaged. For simplicity, we generated blur noise using a turbulence function and implemented the filters keeping in mind this information (Figure 7). In a real case scenario, it will be harder to identify a good degradation function to restore the image and the results may not be as good as we are presenting. The inverse filter equation does not consider the noise of the image, giving undesired results (Figure 8). For this reason we also implement the Wiener filter that does take into consideration both the degradation function and the noise of the image (Figure 9) .

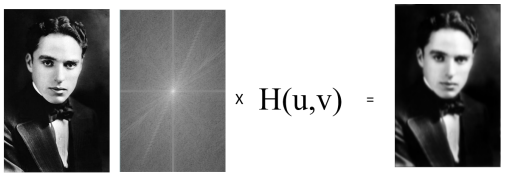


Figure 7 . Blur generation pipeline. We multiply the image DFT with the degradation function. In this case, the function simulates turbulence, and the constant k is the level of turbulence

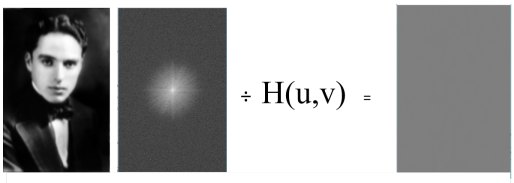


Figure 8 . Inverse filter implementation. We divide the image DFT by the same degradation function.

F(u,v) =

$H(u,v)$: degradation function

$H^*(u,v)$: complex conjugate of $H(u,v)$

$|H(u,v)|^2 = H^*(u,v)H(u,v)$

$S_\eta(u,v) = |N(u,v)|^2$ = power spectrum of the noise

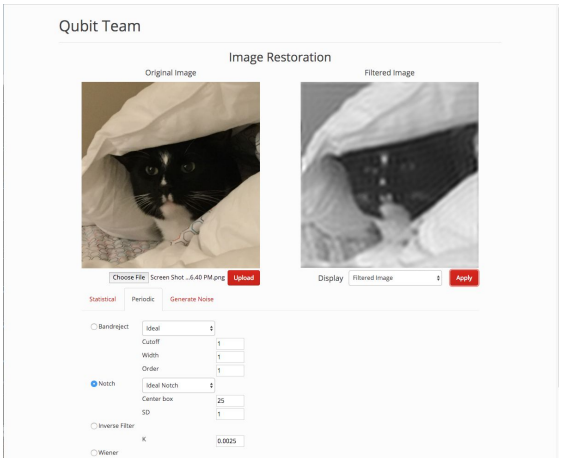
$S_f(u,v) = |F(u,v)|^2$ = power spectrum of the undegraded image

$$F(u,v) = \left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$

Figure 9. Wiener filter implementation. We applied the above equation, estimating the value for S_n/S_f to 0.001. This values correspond to the original image and the noise. Since we don't have the real noise the user can estimate this value in the GUI.

Software

We have implemented the mentioned techniques in a web application. The application has been built using the WebFramework Flask. Flask integrates a communication between the client and the server using Ajax (Asynchronous JavaScript And XML). Ajax handles the request and responses, which we use to send the selected parameters of the UI to the Python backend where the algorithms are applied. Besides Flask, we also used other libraries for different purposes. For instance, we did the GUI using HTML5 and CSS3 with a Bootstrap template. In addition, we also used jQuery to handle specific functionality in the user experience.



The user can upload an image from the local directory. Then the user can choose the parameters from the tab content being displayed. Once the values are chosen, the user can press the **Apply** button. That will send a request to the server, which will reply with the filtered image. Alternatively, the user can display more data from the **Display** dropdown below the filtered image (options such as histogram, DFT, etc.). Take in account that the server takes time to process the algorithms in some cases, for which we display a loading image while you wait.

Conclusion

In this report we presented our work in image restoration. We implemented statistical filters and periodic filters which we used to remove noise and image degradation from images. Our implementations worked considerably well, showing that the resulting images can be restored close to the original quality. As part of our project, we designed and developed a Web GUI that makes it easy and intuitive for the user to select different filters, tune parameters and perform image restoration for a degraded image.