BE Computer Vision

Nathan Magnan Noé Clément Thomas Bellier

November 2020

1 Basics

Question 1

We use the cargo image, and convert it to grey scale using the *rgb2grey* MatLab method. We get the following images :

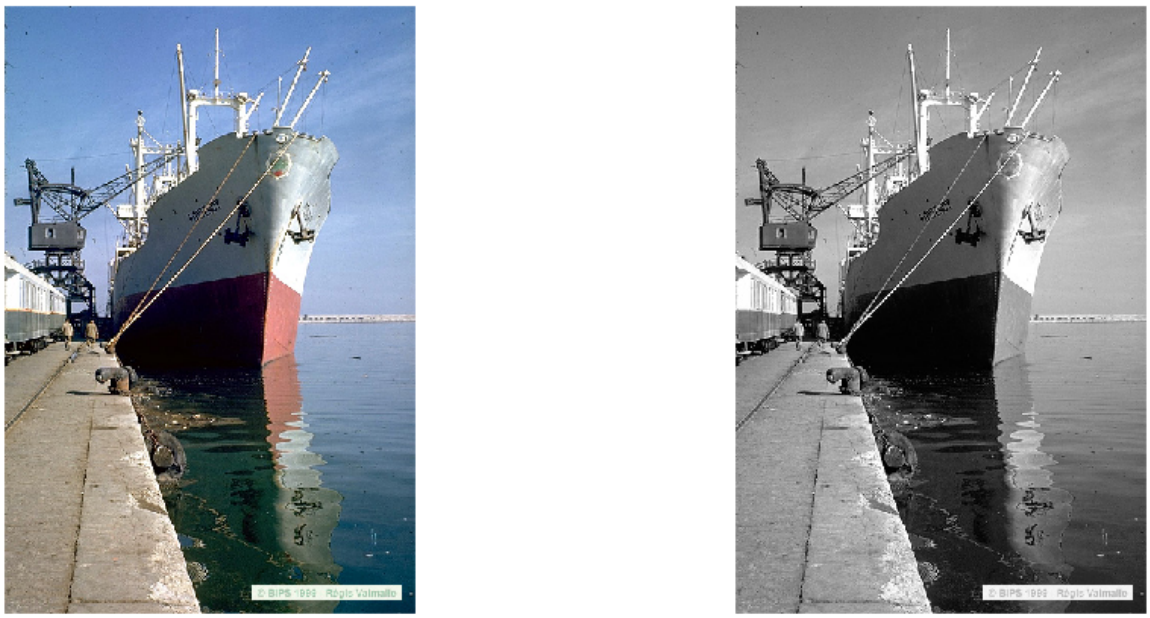

Figure 1: A color image and a greyscale image

In MatLab the color images are stored as $n \times m \times 3$ arrays, and the greyscale images as $n \times m$ arrays.
The first 2 dimensions allow for considering a given pixel. The 3^{rd} dimension allow for considering the red, green or blue information.
And the value contained in a case of the array varies between 0 and 255, it correspond to an intensity.

Question 2

We built an image where the central part has a constant value 100, and the outer part span the whole $[0; 255]$ intervals linearly, and we show it in fig. 2.

Figure 2: Grayscale illusion

Question 3

We build a matrix with stripes of random (and, *a fortioro* variable) width in fig. 3, essentially creating a bar code.

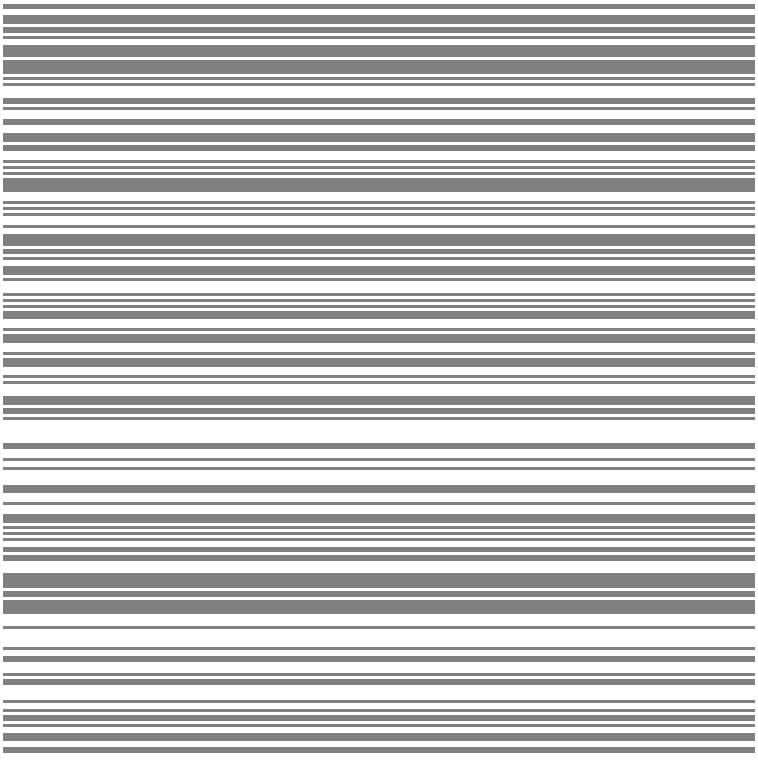

Figure 3: Image of black and with stripes of random witdh.

Question 4

We start by plotting the color image *Teinte.jpg* and its red, green and blue components in fig. 4. It appears very quickly that the red component is important in the red, orange, and pink sectors of the original image, and weak everywhere else. Similarly, the green component is strong in the green, yellow and turquoise sectors. And the blue component in the blue, turquoise and pink sectors. We just found the rule for additive coloration, for example "yellow is a mix of red and green".

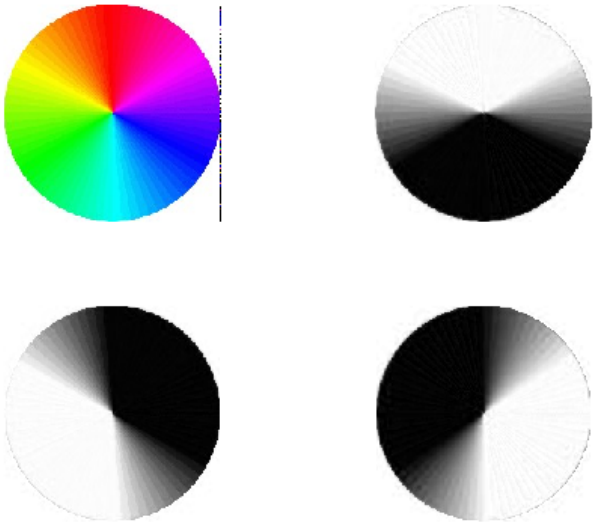

Figure 4: Division in the 3 components of the image *Teinte.jpg*. The red component is on top right, the green one on bottom left and the blue one on bottom right.

We perform the same analysis with the image *oeil.jpg* in fig. 5. We find again the same results on terms of colors, but we can add that the green component is the most easy to read. This is generally the case when we decompose an image that was made for the human eye : it is more sensitive in the green domain, so a good drawing must have a lot of contrast there.
Also, we see that the black (in the legends) is an equal mix of the 3 components (at value 0).

Finally, we perform the same analysis for the image *cargo.jpg* in fig. 6. There isn't much to add to the previous analysis, except that again the most contrast is found in the green, although it is not a man-made image.
We are owitnessing the contraposition of the preceeding argument : the human eye is most sensible in the green domain, because that's where the most information is found (on Earth). Hence the the eye evolved to be sensitive there.

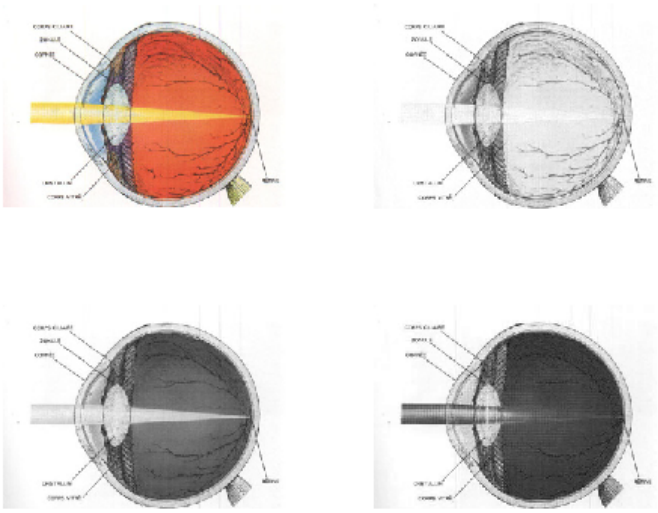

Figure 5: Division in RGB components of the image *oeil.jpg*. The red component is on top right, the green one on bottom left and the blue one on bottom right.

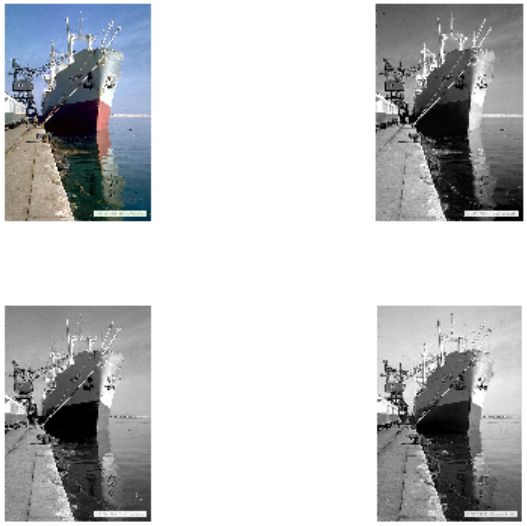

Figure 6: Division in RGB components of the image *cargo.jpg*. The red component is on top right, the green one on bottom left and the blue one on bottom right.

Question 5

Using the Wikipedia ratio and RGB data, we construct a French flag quite easily. However, my version of MatLab seemed to have some drawing issues and produced an awful blue and a red that looked white to the eye... Hence we simply reproduce the Wikipedia figure here :

The main takeaway is that to create the blue, one needs a large blue component, but also a non-zero green component. And the same thing applies to the red stripe.

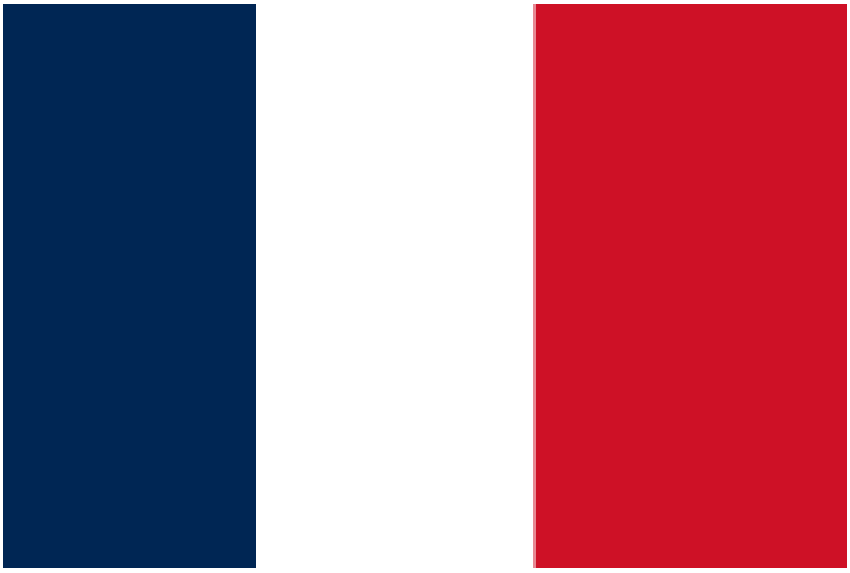

Figure 7: French flag, with the dimensions used by the navy. Courtesy of Wikipedia.

Question 6

We work with the image *Teinte.jpg*, and we map its 3 HSV components in fig. 8 :

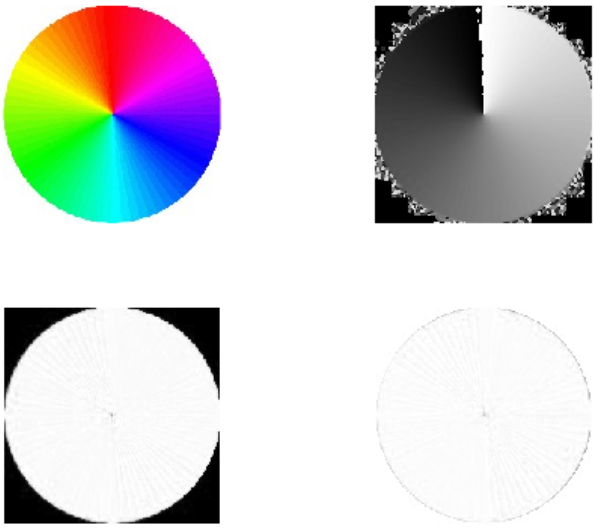

Figure 8: Division in HSV components of the image *cargo.jpg*. The hue component is on top right, the saturation one on bottom left and the value one on bottom right.

We first observe that the new matrix is still a $n \times m \times 3$ matrix, but the 3^{rd} dimension does not correspond to choosing a RGB component anymore, but instead a HSV component. In particular, we cannot use the the MatLab function *imshow* to plot this matrix as an image. Also, the values in the array are not between 0 and 255 anymore, but between 0 and 1. We also observe that the image only presents saturated colors (bottom plot). Also the values component is minimal except on the black boundary. This was expected, as the value parameter is often similar to a "darkness" factor.
The hue image is harder to analyze. Hue correspond to "color", so it is no surprise that the hue parameter evolves linearly when we go round the circle of colors. We see that the extremal hue values are both in the middle of the red, by convention.

By imposing a linear saturation scale on the vertical axis, a linear hue scale on the horizontal axis, and a minimal value, we reproduce fig. 9 :

Figure 9: HSV color space

Question 7

The true value of the parameters are $\alpha = 0.2126$, $\beta = 0.7152$ and $\gamma = 0.0722$. But by playing with the parameters, we find that parameters $\alpha = 0.26$, $\beta = 0.6$ and $\gamma = 0.14$ reproduce a greyscale image that is closer to the one that *rgb2gray* yields, as shown in fig. 10. But that might simply be because our greyscale representations hides the greens in a saturated white, hence we do not see the contrasts in that domain.

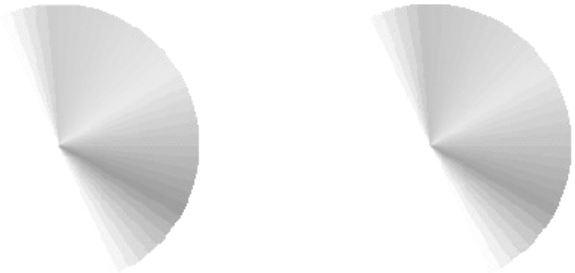

Figure 10: Comparison between MatLab's greyscale transformation, and ours.

Question 8

For a greyscale image, the histogram gives the proportion of pixels in given bin of darkness, it is especially usefull to anaylse the contrasts in the image. For instance, if there are only dark pixels, it is interesting to renormalize the values in the pixels : the contrast will be greatly improved and we will see more details on the final picture.

We plot the greyscale image, and its histogram, of the image *SpainBeach.jpg* in fig. 11.
First we observe a high, thin peak in the distribution in the dark domain. This is typical of a lack of contrast, which is

usually an issue. But here, it is due to the sea, on which we do not expect to see any detail. Hence we can accept this lack on contract.
Then we observe a broad peak in the middle domain. It correspond to the city, on which we would like as much contrast as possible to see details. Hence it would be interesting to renormalize the greyscale to use the entire middle range.
Finally, we observe a small peak in the white domain. Is is due to the waves, and since it is saturated, information has been lost so we cannot hope to raise its contrast.

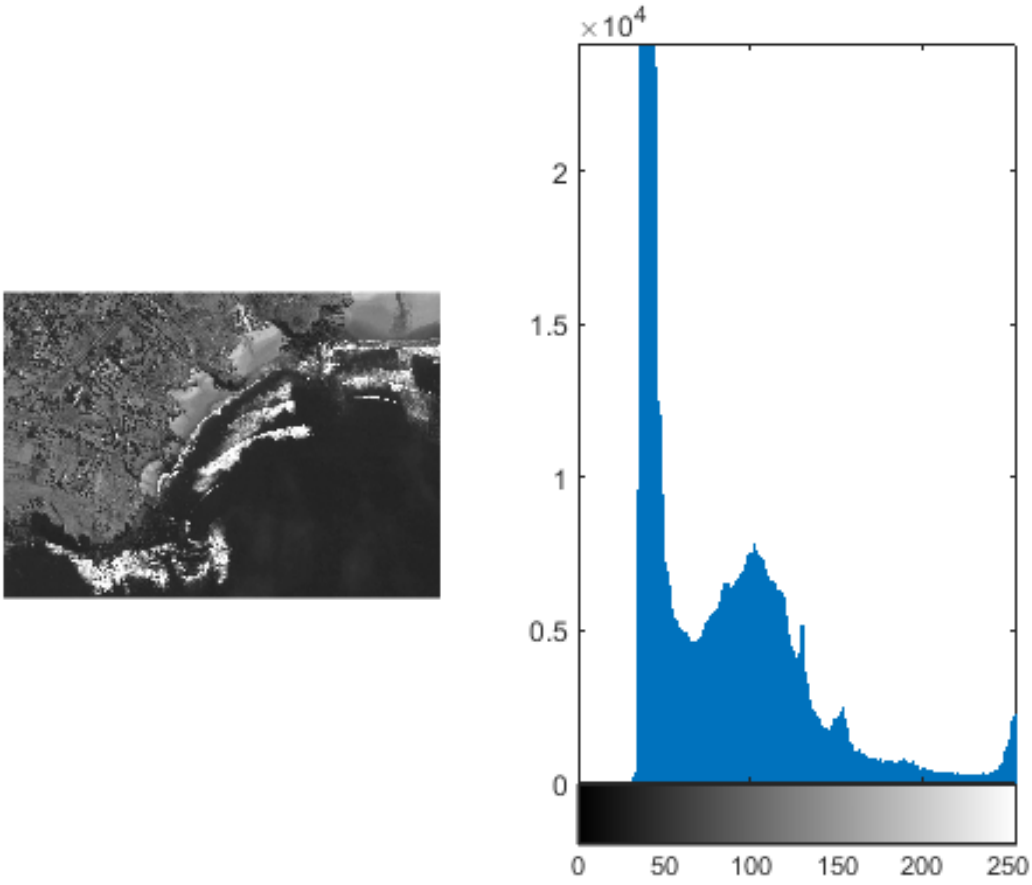

Figure 11: Histogram of the greyscaled version of image *SpainBeach.jpg*.

Question 9

The 2 mystery images have nearly no contrast, hence it is nearly impossible to detect what they represent. But simply re-equilibrating their histograms (in each band for the RGB image) to use the entire range from 0 to 255 (with the function *histeq*), we can clearly see in fig. 12 that they represent the city-center of Toulouse, and the Earth seen from space.

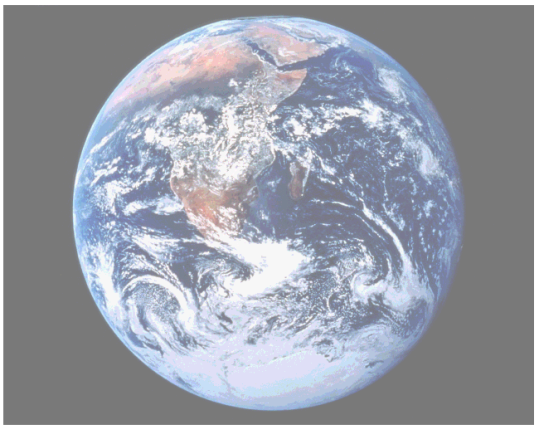

Figure 12: Constrast-corrected mystery images.

Question 10

We load the image *SpainBeach.jpg* and find that the beach is a very low-contrast region in all 2 RGB domains, with R values higher that 150, and blue values lower than 100. Hence we create a new image with the pixels respecting these criterions in black and the other pixels in white. We present the result in fig. 13, from which it is clear that the whole beach selected, and that only the beach and a few houses Are selected. The house have brick roofs, they are indistingushable from the beach without geometric considerations. Hence we can be happy with these result.

Figure 13: Position of the beach extracted from the image *SpainBeach.jpg* with a very basic algorithm.

We should also add that we tried the same algorithm on the greyscale version of the image, but the results were not nearly as good, because of the degeneracy between colors.

Question 11

We apply blur flitering and edge filtering on the images *Stripes.jpg* and *Toulouse.bmp*, which correspond to the following kernels :

$$\underline{\underline{H}}_{blur} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{1}$$

$$\underline{\underline{H}}_{edges} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

We can easily apply these filters thanks to the functions *fspecial* and *imfilter*, and we get the results presented in fig. 14.

We first observe in these images that the blur filter smoothes both images, which corresponds to a loss of information, by cutting the high spatial frequencies in the images (this is very obvious on the stripes images). On the stripes images, it corresponds to a loss of information that strongly degrades the image.
On the real image, we could have expected that the blurring would erase the noise, and that the blurred image would be just as good as the original one. But this is not what we observed, the blurred image contains less information indeed than the crisp one.

As for the edge-filter, there isn't much to say. It did outline edges on the stripes image, but those were already obvious. And on the real image, it did outline the buildings, which was expected. But it also outlined a lot of noise. Hence to get a better result, we applied the blurring filter, **then** the edge filter to the image. And indeed, the buildings appear much more clearly (see fig. 15).

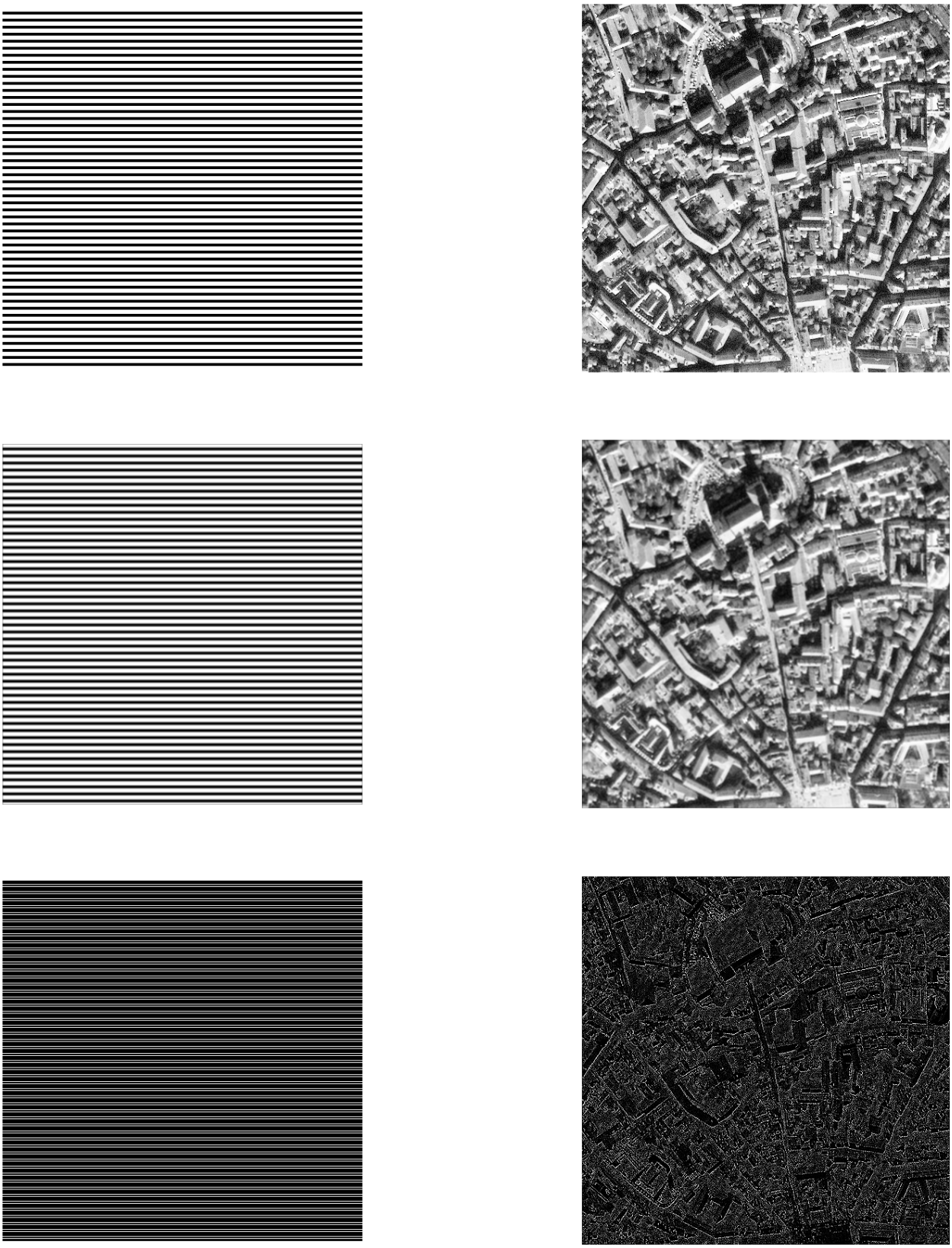

Figure 14: Blur and Edge fliters applied on the images *Stripes.jpg* and *Toulouse.bmp*. The top pictures represent the unflitered images, the middle pictures the blurred images, and the bottom pictures the Edge-filtered images.

Figure 15: Buildings extracted from the image *Toulouse.bmp*, simply by using filters.

Question 12

For this question, we followed the following filtering strategy :

- First, we reduces the background noise thanks to a non-linear MatLab filter dedicated to random independent noise, via the function *wiener2*. We can have it work on many pixels to estimate and eliminate the noise as much as possible. We had it work on the 400 pixels closest to the pixel of interest.

- Then we use an averaging filter. The goal is not to get rid of noises, but to spread the luminosity peaks. For a large star, the central pixels are all surronded by bright pixels, so the output will be a bright pixel. But for small stars, even the innermost pixels are close to the edge, hence the output will be a dark pixel.
  The otherway to see this is that we diffuse / spread the luminosity peaks, hence the peaks with the least energy disppear.
  The important step is to choose properly the averaging distance. We remark that the largest stars have a diameter of about 15 pixels, so that will be our averaging distance.

- Finally the last step is a non-linear filter that we constructed ourselves. It simply selects the pixels with a luminosity above a given level, and reject the others. Hence the star that were spreaded out by the 2^{nd} filter will be rejected.
  This last step is efficient algorithmically, but we still do not find it elegant for one simple reason : we had to tune the acceptance level to get exactly 5 stars. It would not work on another image...
  Still, it is the best method we found using solely spatial filters and no geometric method.

This pipeline yields the results in fig. 16, from which it is clear that the 5 main stars were selected.

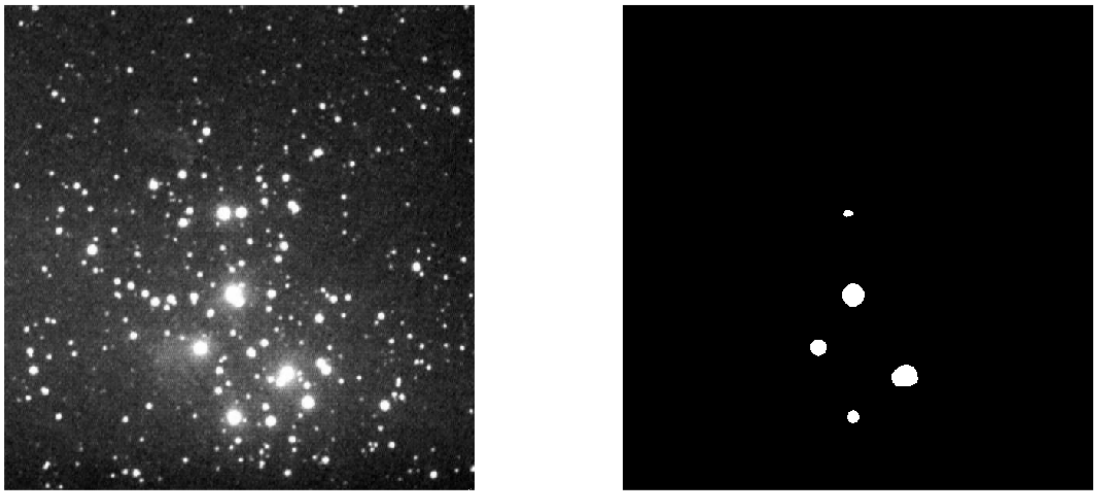

Figure 16: Original image of the sky on the left, and the 5 stars selected by our filtering pipeline on the right.

2 Fourier Transform

Question 13

As we can see on figure 17, because our image is constant while moving horizontally, the FT is only on the central vertical stripe ($k_x = 0$), with the dominant frequencies corresponding to our pattern.

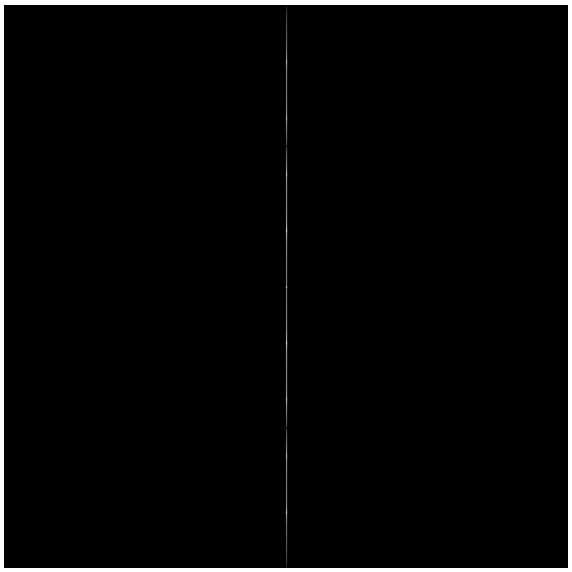

Figure 17: Fourier transform of the stripes.

Question 14

As we can see on figure 19 and figure 18, because our image is still constant while moving horizontally, the FT is still only on the central vertical stripe. However, the blur cuts the high frequencies, that's why we can see that the more aggressive is the blur, the more the FT moves toward the center.

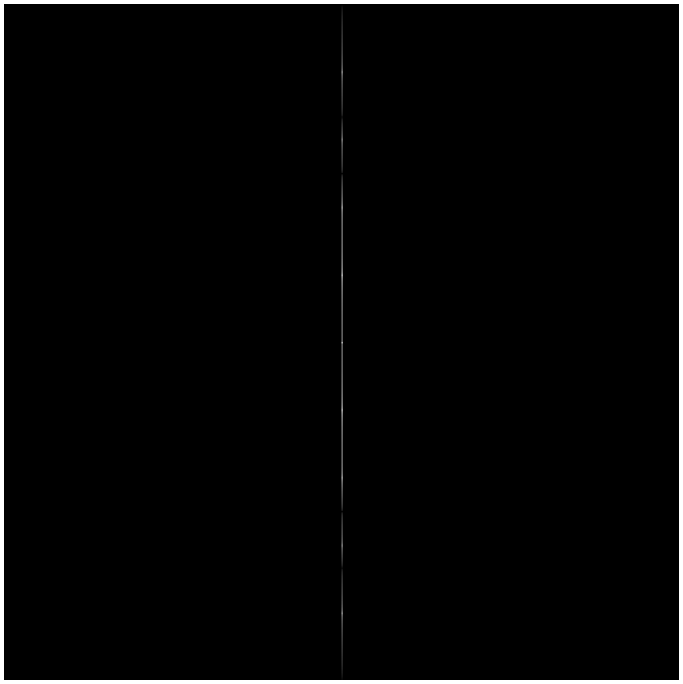

Figure 18: Fourier transform of the stripes with mean blur.

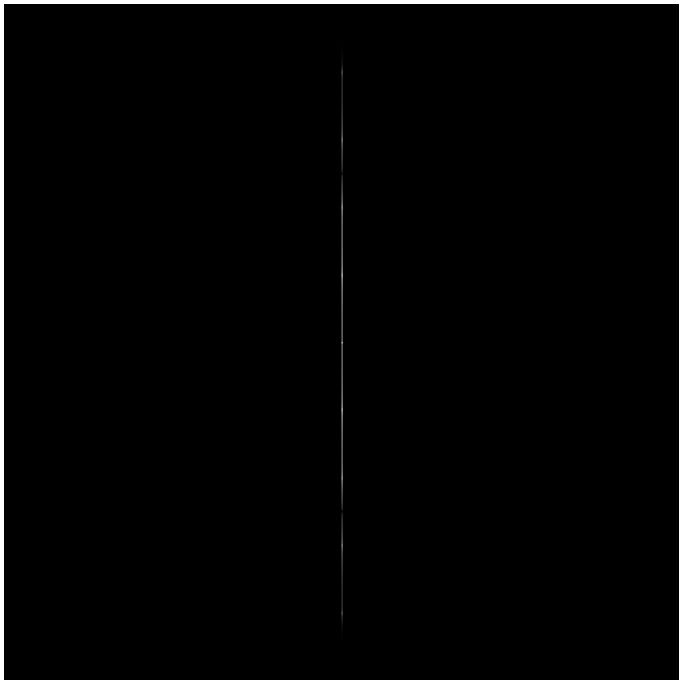

Figure 19: Fourier transform of the stripes with Gaussian blur.

Question 15

To extract the field from the image, we need to multiply the FT with the FT of a pattern similar to the specific field we want ; for that we build simple rotated stripes, shown in figure 20. (It is zoomed, the original is only 40 by 40 pixels). We then use a Gaussian blur to avoid cuts in the field detection.

This product gives us the image in figure 21, where we can clearly see the field already. We then create a binary mask with a simple threshold to get the final image, in figure 22.

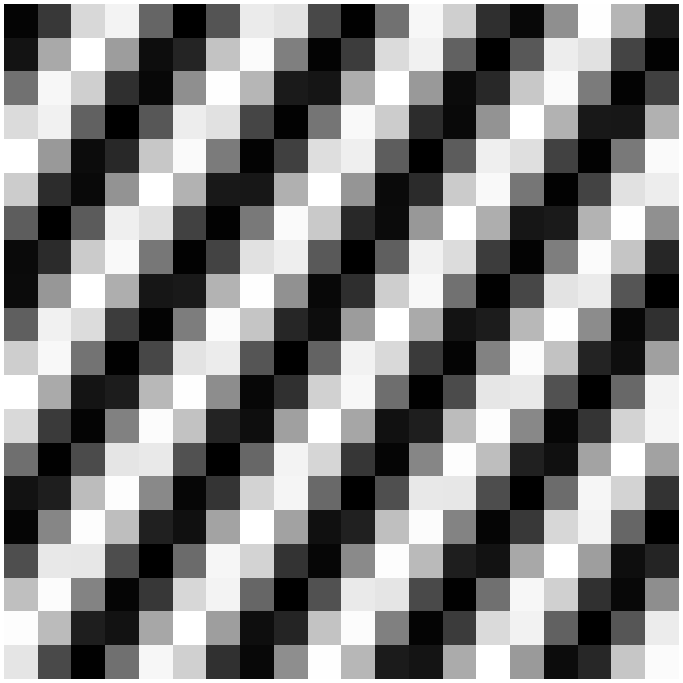

Figure 20: Pattern for the field extraction.

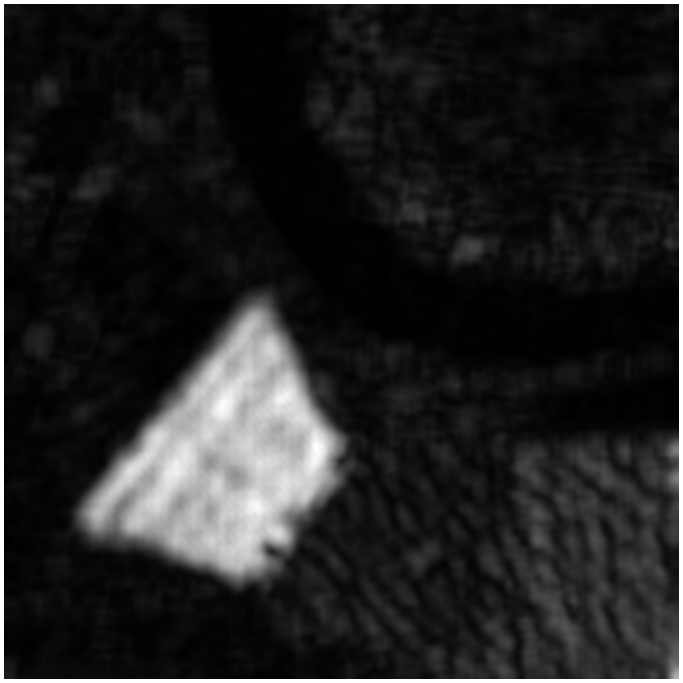

Figure 21: Linear filtering of the fields using our pattern.

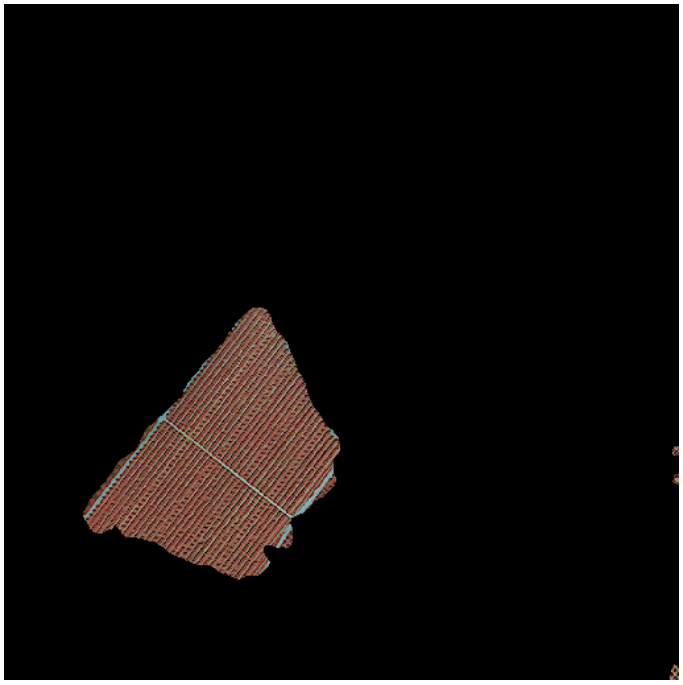

Figure 22: Field extracted from the image.

3 Deblurring

Question 16

On the spectral domain, convolution becomes a multiplication. Hence the kernel $\mathcal{H}$ amplifies some frequencies of the original image and reduces other frequencies. And the Gaussian Noise $\mathcal{B}$ adds some new frequencies, and modifies the amplitude of already existing frequencies, thus perturbing the image.

Question 17

We have an starting expression for $\mathcal{H}$:

$$\begin{aligned} \mathcal{H}(u) = & \quad \frac{1}{2T+1} \frac{\sin\left(2\pi \frac{u}{N}\left(T+\frac{1}{2}\right)\right)}{\sin\left(\pi \frac{u}{N}\right)} \\ = & \quad \frac{sinc\left(\pi u \frac{2T+1}{N}\right)}{sinc\left(\pi u \frac{1}{N}\right)} \end{aligned} \tag{2}$$

Now we can consider $2T+1$ is much larger than 1. In these conditions, the top term evolves quickly, while the bottom term remains saturated at 1 for a long time. Hence for $u < \frac{N}{2T+1}$, only the top term remains.
And for larger values of u, the top term is nearly equal to 0, hence the total filter is also close to zero, and *a fortiori* close to the top term.
Finally, we have :

$$\forall u, \mathcal{H}(u) \approx \quad sinc\left(\pi u \frac{2T+1}{N}\right) \tag{3}$$

Question 18

We compare the spectrums of the original image and of the blurred image :

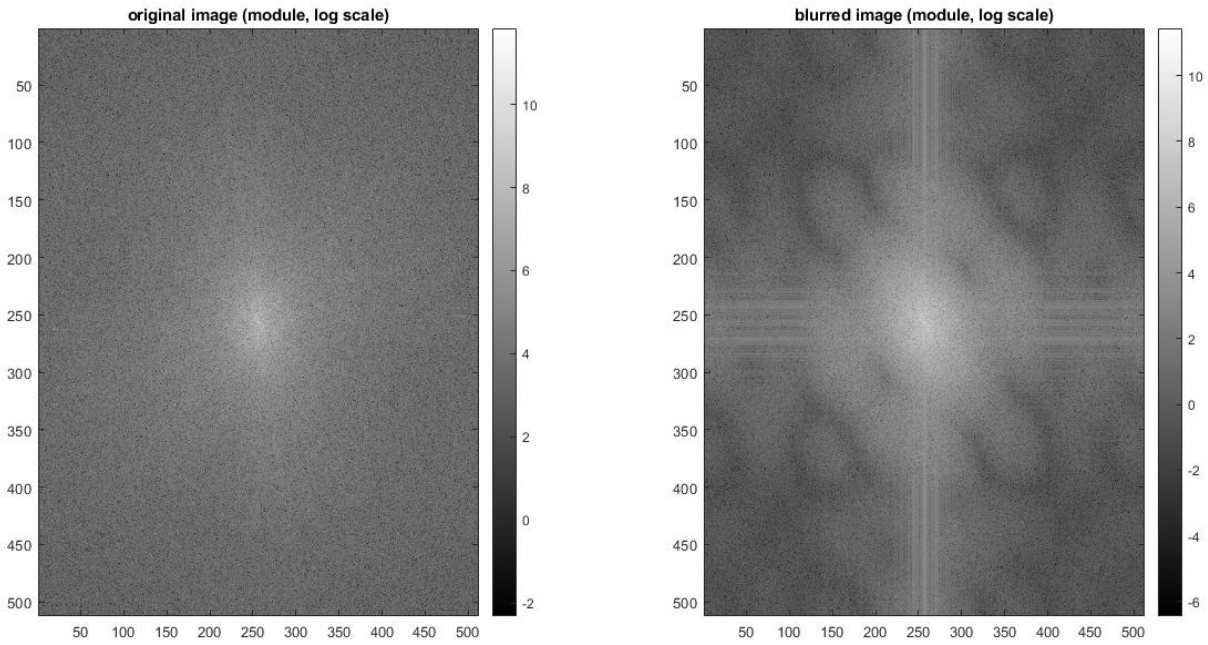

Figure 23: Sepctrum of the original picture on the left, Spectrum of the blurred image on the right, in log scale

We can see that there are horizontal bands (and vertical ones). These are the peaks of the sinus cardinal function we found in question 17. What is important is not their number (there an infinite number of peaks, but most are hidden behind the noise), but their period. We count 4 periods in 60 pixels, thus $\frac{60\times(2T+1)}{512} = 8$ or equivalently $T \approx 35$. This value is much higher than the one we expcted ($T \approx 3$). We do not know why...

Question 19

To complete the algorithm, we can simply add the two following lines :

```
B=GaussianNoise;
DeblurredImage=Image+G*B;
```

Question 20

We apply our inverse filtering method on image *marcheurs.jpg* and we find the following result:

Figure 24: Deblurred image *marcheurs.jpg* using Weiner method.

The sensation of speed with unclear and blur areas has disappeared, but we have now a sensation of dotted lines, like *pointillism*, the technique of painting used, for example, by Georges Seurat.

Question 21