Investigation of feature recognition in clinical CT images as a step towards adaptive radiotherapy : Progress Report

Louis Jaugey, Alix Moawad

18 décembre 2019

1 Aims and objectives of the project

Radiotherapy treatment is based on the destruction of cancerous cells by ionizing radiation. During this process, sane surrounding cells could also be affected by ionizing beam and it is therefore paramount to minimize its effect on those cells. Recent technologies have been devised to modulate the beam intensity in space, allowing a treatment highly adapted to a specific tumour. However, in order to use these techniques to their full potential, the shape and position of the tumour must be precisely known. The process of contouring the cancerous region can be long. The aim of this report is therefore to automate this process on 3-dimensional CT-scans.

2 Progress of the project

Data visualization A bank of 2D images representing slices of a 3D scan were at our disposal. The first step was to familiarize ourselves with the Python programming language and the specific libraries used for image treatment. We begun by displaying the axial, sagittal and coronal views shown on Fig. 1.

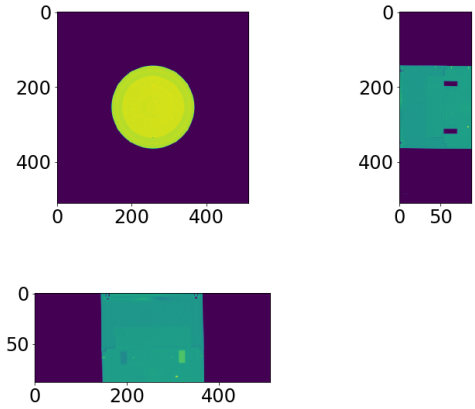


Figure 1 – Axial, sagittal and coronal views of a 3D scan of a phantom

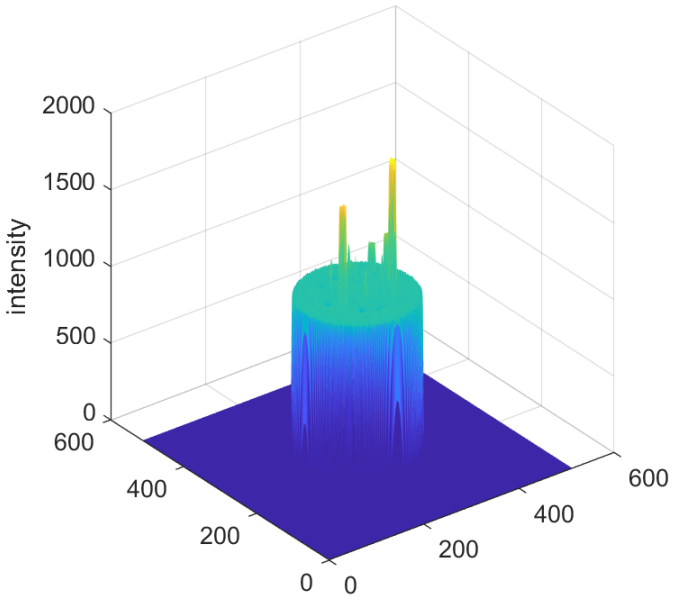
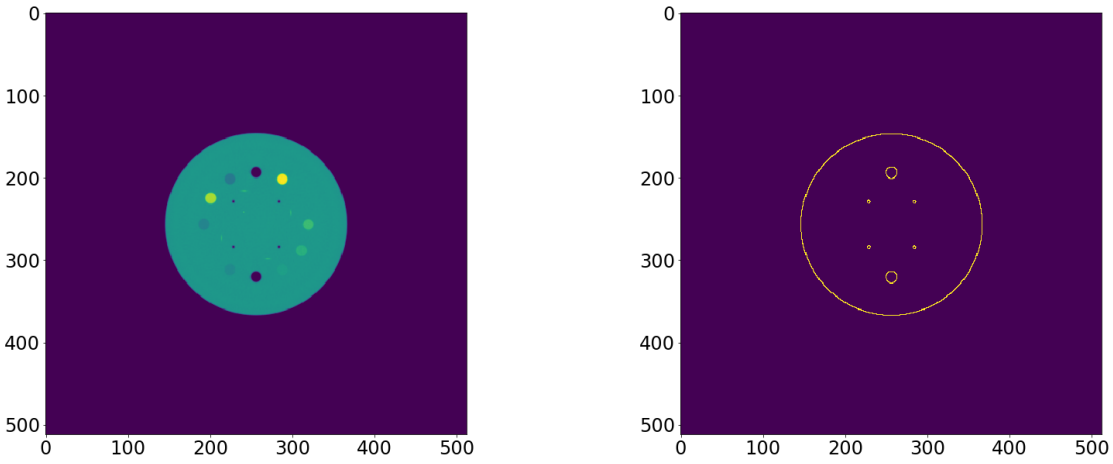


Figure 2 – Surface plot of the intensity of slice 60

Threshold method The goal of the first edge detector algorithm was to be able to determine the shape of the phantom's outer boundary. The method used for this detection is the simplest developed in this project. As discussed below, it does have a lot of limitations.

Fig.2 shows the base image with the intensity represented on the z axis. It gives a surface in a 3D space. Using this plot, the height (or the intensity) of the plateau in green (see Fig.2) can be determined and its value defines a threshold. The algorithm then looks for the pixels whose intensity go from an intensity below this threshold to an intensity above it. The results of this method are shown in Fig.3.

This algorithm can detect the outer boundary of the phantom, which was the primary goal of this first algorithm. Some holes inside the phantom are also detected. Indeed, these holes are filled with air. Their intensity is then comparable with the one outside the phantom : we go from an intensity below the threshold to an intensity above it (or the opposite) by going through these holes.



(a) Initial image of slice 60

(b) Threshold method applied to slice 60

Figure 3 – Results of the threshold method

As one can see with the results presented in Fig. 3, this method cannot detect the different regions in the phantom if their intensity is above the threshold. Most holes are therefore not displayed on Fig. 3 and this algorithm is not applicable for further use.

Derivatives peaks in 1D This method is based on the variation (or the derivative) of the intensity. Unlike the previous algorithm, this should allow the detection of all the shapes inside the phantom. This method was first applied to a single row of the matrix representing the image. The intensity of the 200th row is shown on Fig.4.

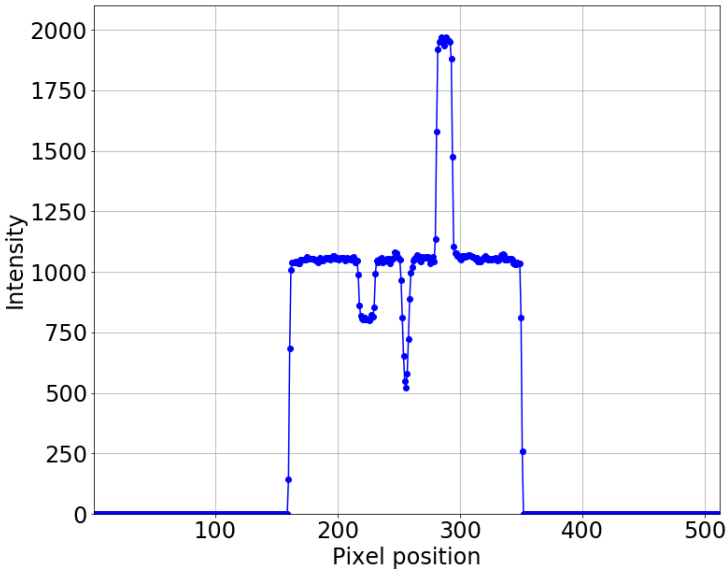


Figure 4 – Intensity of 200th row (slice 60)

The first step in the development of this algorithm was to determine the derivative of the intensity. Since a single row was studied, only the horizontal derivative was computed using the centred finite differences :

$$\left.\frac{\partial I}{\partial x}\right|_i \approx \frac{I_{i+1} - I_{i-1}}{2h} \propto (I_{i+1} - I_{i-1}) \quad (1)$$

where I_i is the intensity of the ith pixel in a chosen row and h is the distance between two pixels. Since h is constant, we only consider the intensity difference in (1). Using this property, the directional derivative of I is computed on a single row of the image. The peaks

of the derivative give the points where the rate of change of the intensity is locally maximum. This is the case when we move from one region of the phantom to another.

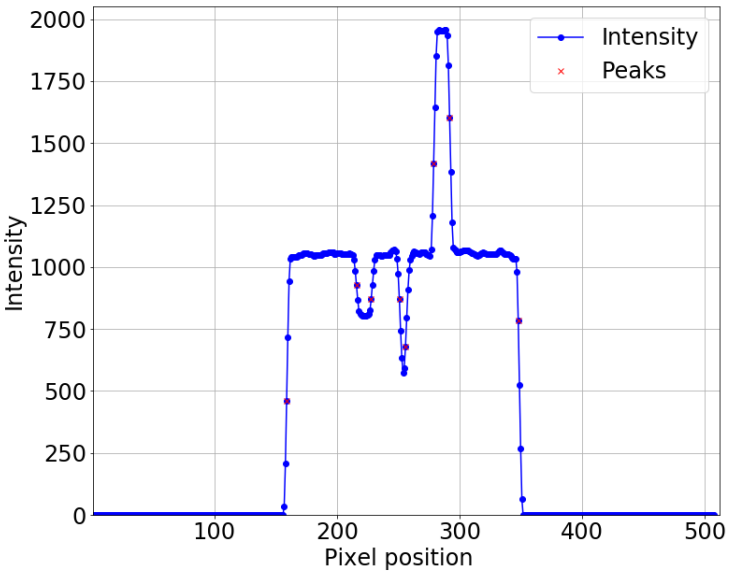


Figure 5 – Intensity of 200th row with peaks showed in red crosses (slice 60)

So the use of the *findpeaks* function (from *scipy.signal* library) on the derivative should give the boundaries in the phantom. Before using *findpeaks*, the image needed to be smoothed since the original one was noisy. To do so, a running mean was applied on a horizontal subset of size 5. The result obtained with this method is presented in Fig.5. Comparing this figure with Fig.4, one can see the smoother aspect of the second image.

The obtained peaks on Fig.5 give the borders of the different regions. However, there seems to be a bias with the *findpeaks* function. Indeed on Fig.5 the peaks are selected sooner (one pixel sooner) when the derivatives is negative than when it is positive. Though a single pixel of difference is not such an issue. Moreover, *findpeaks* requires a threshold that defines the minimal height of the peaks and this choice is arbitrarily. For this reason a different method, based on statistical values, was explored and presented in the following section.

Statistical method on the derivatives In the same way as the previous method, we focused on directional derivatives of intensity. Here, we use a subset of size n on which the mean μ_1 and variance σ_1 of the intensity is computed. The mean intensity μ_2 of the next subset is compared with μ_1 and σ_1 in the following way :

$$\mu_2 > \mu_1 + C\sigma_1 \tag{2}$$

where C is a given coefficient.

If this condition is fulfilled then the pixel in the middle of the second subset considered as a contour. The condition (2) can only be true for a positive derivative (from a lower to a higher intensity), the condition must therefore be tested in all four directions (i.e. left to right, top to bottom and so on). This method was applied on the entire image and the different steps are presented in Fig.7.

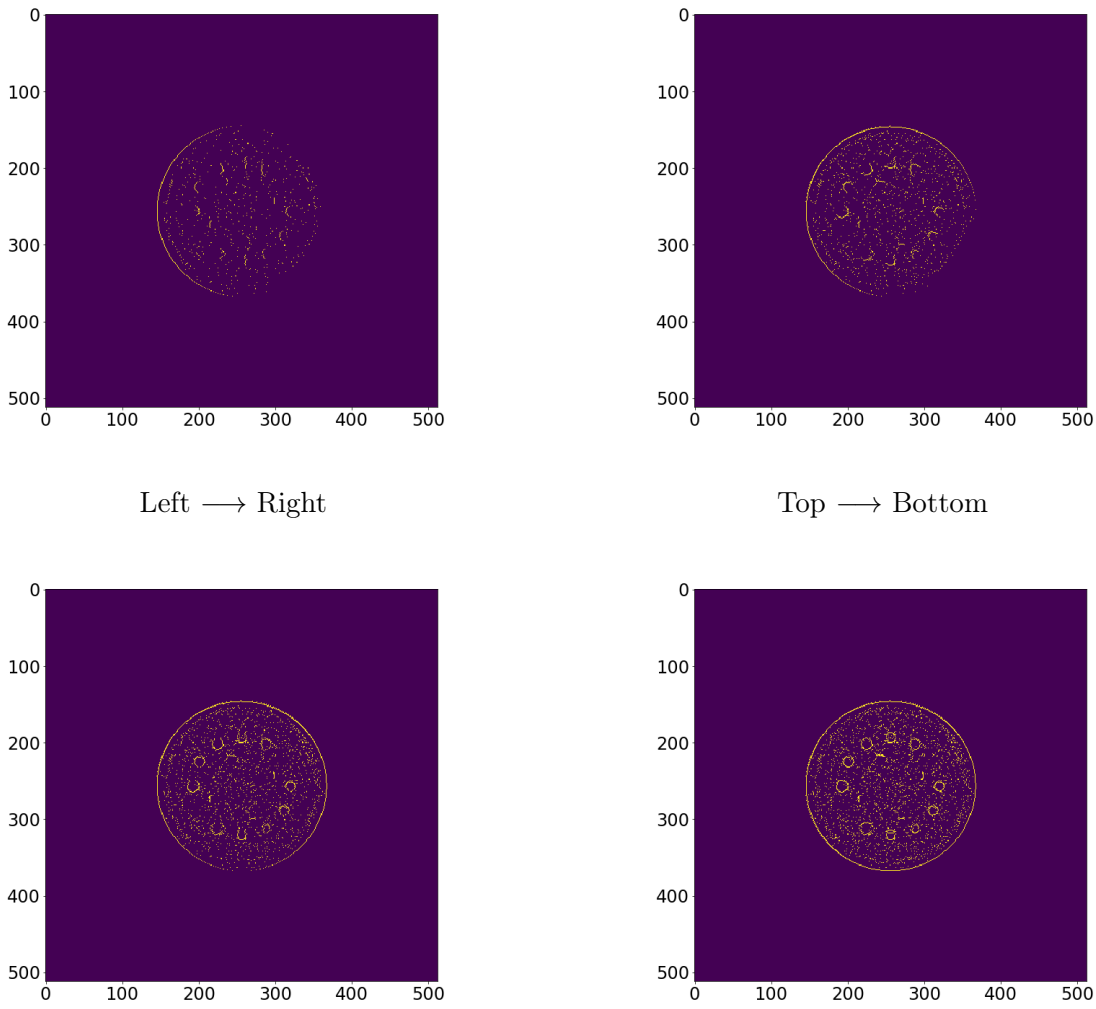


Figure 7 – The different steps of the statistical method.

This technique has a clear issue with noise. A simple algorithm that removes the isolated pixels was developed and the final result is shown on Fig.8. This algorithm looks for pixels which are considered as a boundary and remove them if they are the only one in a 3×3 matrix centred on this pixel.

This method is able to detect the different regions inside the phantom. However, the boundaries are not smooth and above all, some of them are not closed. Furthermore, although most of the noise was filtered, there still remain some artefacts.

Since the results obtained with this method were not satisfying, we decided to further develop the previous algorithm.

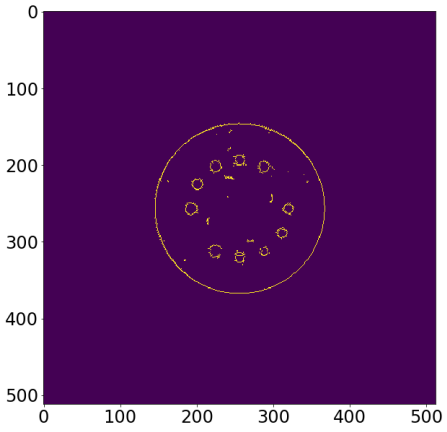
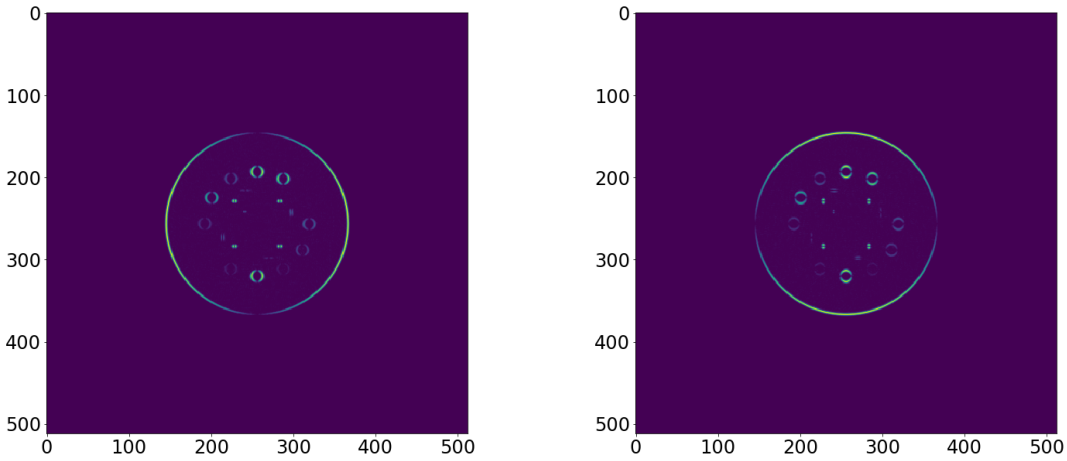


Figure 8 – Statistical method results

Derivatives peaks in 2D The primary difference with the first method presented before is that a Gaussian filter was applied to the image. This filter does a convolution product between the intensity and a 5×5 matrix whose components are given by a 2D Gaussian function with variance σ. The function *gaussian filter* is implemented in the *scipy.ndimage* library and its arguments are the raw image and σ. We then compute the directional derivatives of the intensity in both x and y directions as you can see on Fig.9.



(a) Horizontal derivatives

(b) Vertical derivatives

Figure 9 – Directional derivatives of the intensity

In order to define the boundaries, we used a method of "edge tracking by hysteresis". The method requires 3 categories of pixel : high, medium and low intensity pixels. Pixels from the first category are automatically considered as a boundary, those from the second category are a boundary only if they are connected to pixels from the first category. Finally, pixels from the last category are ignored. To assign each pixel to its category, we apply *find peaks* twice with two different thresholds. If a peak intensity is above both thresholds, it is a high intensity pixel, if it is in between, it is in the second category and it is ignored if it is below both thresholds. Since this method is used on 2D images, *find peaks* must be applied, on both rows and columns of the derivatives matrices.

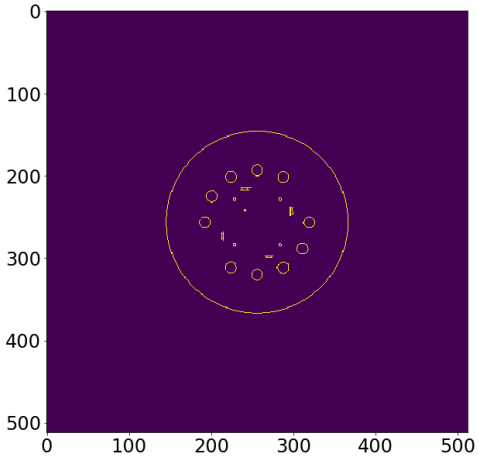
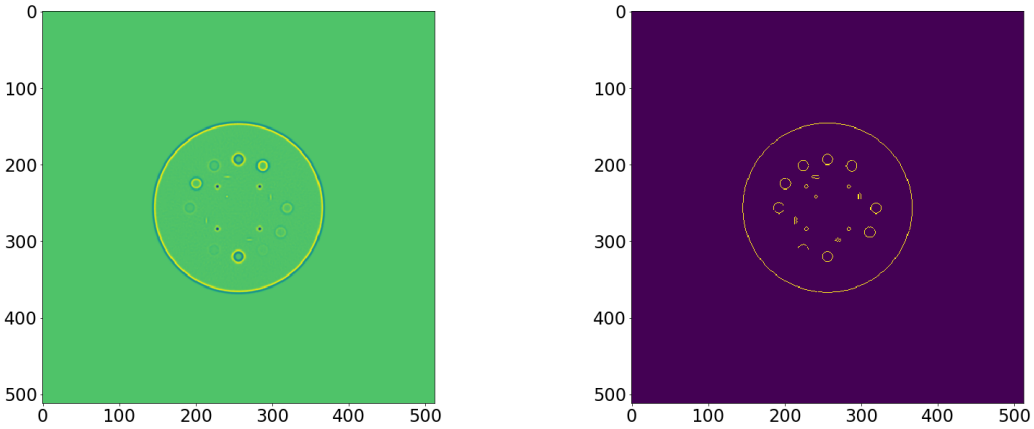


Figure 10 – Final results with the directional derivatives method

Laplacian The two previous codes used directional derivatives of the intensity matrix. In order to find the contours, we had to find the maximums of the derivatives. We could also work on the Laplacian operator instead of the derivatives. Indeed, the maximums of the derivatives are equivalent to the zero crossings of the Laplacian operator. We chose to use central finite difference approximations to compute the 2nd order derivative of the intensity matrix. The zero

crossings are detected in both x- and y-directions. We will give an example on one row : for one pixel studied, if the pixel value on its left is negative and the pixel value on its right positive (or the opposite), then the pixel in between is considered as a zero crossing. This pixel will then be part of a border. However, this method also takes into account background noise and it could give false zero crossings and so false borders. To avoid this, only a certain percentage of the brightest contours are considered as borders. This quantity is arbitrary and can be optimize for each slices of the phantoms CT-scan. The algorithm of isolated pixels is once again used to remove noise. As you can see on Fig.11b, the contours are not well defined for every holes. Some of the holes inside the phantom are not closed and other are not displayed. This can be explained with Fig.11a. Computing the Laplacian with 2nd-order finite differences seems to make disappear contours with the lowest contrasts so that they could not be differentiated from the background. The Laplacian method could induce the loss of information unlike the Derivatives method.



(a) Horizontal derivatives

(b) Vertical derivatives

Figure 11 – Laplacian method results

Canny Filter In this section, we used the Canny edge detector already implemented in the *skimage.feature* library. This algorithm is a standard method in 2D image treatment. Canny edge detector is able to find contours in a 2D image and works as follows : a gaussian filter is first applied to the image to reduce the image's noise. The intensity gradient is then computed at every point of the image : the contours are defined at local maxima of the gradient. The final step is the edge tracking by hysteresis as explained in the Derivative peaks in 2D method. The matrix obtained with the Canny filter is displayed in Fig.13. At this stage of the algorithm, two categories of pixels are defined : the ones considered as forming borders and the others. Only the category containing pixels forming borders will be of interest here. For now, we only know if these pixels are part of a border but we do not know which one.

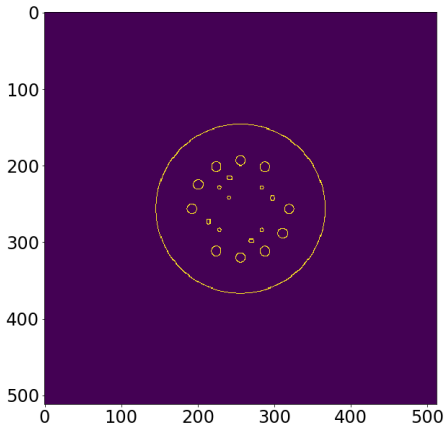
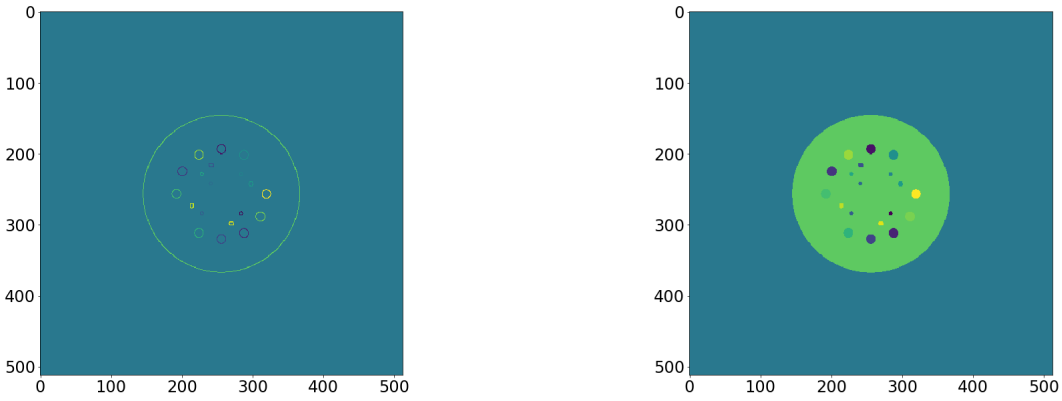


Figure 12 – Contours found by the Canny filter

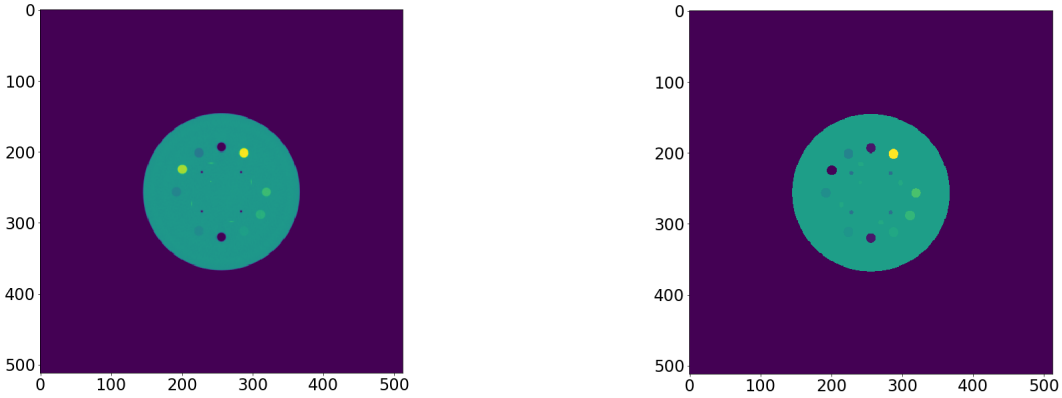
The next step is then to differentiate borders within this category of pixels. To do so, we iterate over the matrix given by the Canny Filter. When we find a pixel belonging to the category of interest, this pixel is considered as part of a border numbered 1. We then search recursively around this pixel for the other pixels belonging to the same border. The same process is repeated as many times as the number of different borders found in the image. To see better the colors used to describe the different borders, we filled each borders with the color of their contour. The result is shown on Fig.13. To be more coherent, we then decided not to use random colors but colors defined as the mean intensity inside each border. As you can see on Fig. 14, we retrieved the initial image but with each border detected.



(a) Separation borders result

(b) Filled borders result

Figure 13 – Separation borders results on slice 60 with randomly colored and filled borders



(a) Initial image of slice 60

(b) Canny filter result with holes filled with mean intensities

Figure 14 – Comparaison between the initial slice 60 and final result by applying Canny filter as described

However, parameters must be given in order to use this implemented function. The low and high thresholds are defined quite arbitrarily : the aim was to display as much pixels as possible without considering noise as borders. So a balance had to be found in order to get as close as possible to our goal.