CS6640: Image Processing
Final Project
Active Contours Models

Arthur Coste: *coste.arthur@gmail.com*

December 2012

Contents

1 Introduction

As studied a lot in the previous projects, one of the most important and challenging task in Image Processing is to extract objects from images. To study their shapes, characterize them and process them, one of the most important challenge is to extract their contour and boundaries. We already covered widely edge detection using derivative filters to extract edges, we also used Hough transform to characterize most geometrical objects. In this last project we are going to introduce an other method to segment objects in an image. This project is relying on several techniques we already studied and implement in previous projects such as Gaussian Filtering, Thresholding to create binary images, edge detection and other techniques.

While doing this project, we reviewed a lot of various documents, papers and implementation of this technique. Due to prior knowledge of this technique I really wanted to implement and work on Euler Lagrange algorithm, which easy to implement and an accurate method to perform segmentation using active contours. Another method was introduced in class an rely on a greedy implementation, we will present it and present our implementation.

While doing this project several issues and questions appeared and I had to rely on some external resources to improve some points. Some functions could be improve, using Euler Lagrange is a good method but it does not take in account some features such as contour dilatation which was originally presented by Kass *et al.*

In this report we will cover the theoretical presentation of active contour models and some interesting points related to its implementation. We will then provide results that we will present and analyse.

The implementation is also realized using MATLAB, and here are the related functions for this project.

Note: The following MATLAB functions are associated to this work. Each of them is linked an set up with a test image.

- *select_points.m* : $[x, y] = select\_points(InputImage)$
- *greedy_snakes.m* : Stand alone function
- *snakes.m* : Stand alone function using Euler Laplace method

Note: I used images that were provided and made all the other drawing or images so there is no Copyright infringement in this work.

The focus in the code and in this project has been given to Euler Lagrange Method which appears to be mathematically more interesting and is really nice to implement. We tried to implement the greedy method but it's not a success.

2 Theoretical presentation

An active contour model, also called a snake is technique for detecting and segmenting objects using deformable curves which will match the objects. It relies on a deformable model controlled by an energy minimization function. This energy depends on the image and also on the parameter we want to give to our deformable contour.

2.1 Basics

In this project, we are going to use and handle bi dimensional contours. Here is the basic mathematical set up of these curves. To simplify the model we will use the curve parametric model. Let's call γ our contour, where γ can be either open or closed curve. We will discuss later this aspect.

$$\gamma : \begin{cases} [a,b] \subset \mathbb{R} \rightarrow \mathcal{D} \subset \mathbb{R}^2 \\ s \mapsto \gamma(s) = (\gamma_x(s), \gamma_y(s)) \end{cases} \tag{1}$$

The previous set up for our deformable curve allows us to have the two types of curves i.e open or closed if $\gamma(a) = \gamma(b)$.

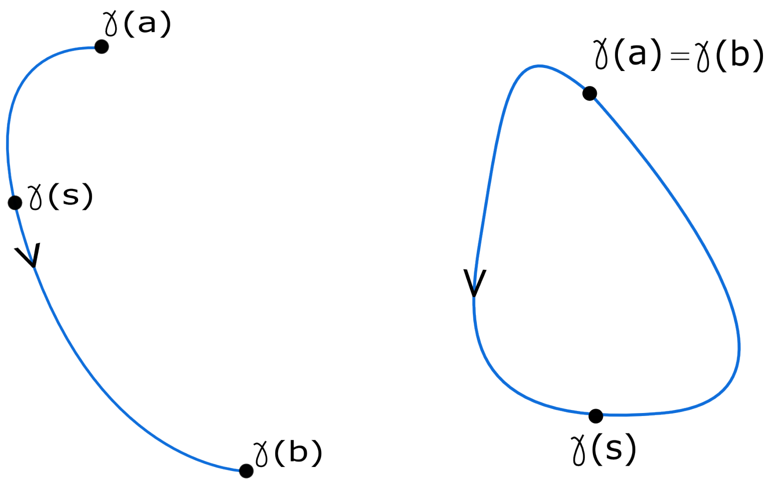

Figure 1: Opened and closed contours

2.2 Model

By creating an active contour, we want to create a curve whose behaviour is going to be constrained by two aspects.
The first and most important is due to the objective which is to perform a segmentation based on object and shape detection. So we need our contour to converge to the edges of the object we are interested in.
The second constraint lies on the model of the contour we want to have. Indeed, we want to obtain a deformable model which means that we would like to have a continuous curve where the curvature

will match the shape curvature. So this implies to have a theoretical model we can implement to describe the contour's own behaviour.

These two previous properties are going to provide us with two components for the function that describes the contour deformation's behaviour on the image. This function is described as an energy function that we are going to present. Due to the description we made before our contour function is going to be called $\gamma(s)$

2.2.1 External Energy

The external energy is the component of our behaviour function that describe how the deformable curve will match with objects of the image. Indeed the external energy is the function that will constrain the contours displacement. To be attracted by a shape, we need to use a function that have the following properties: have at least one local minimum and be monotonic on areas around this point.
To have such a function in an image we are going to use its gradient. Indeed, around edges the gradient is presenting this two characteristics of presenting a local extrema and having a monotonic behaviour. We will enforce this behaviour with a preprocessing consisting in a Gaussian Smoothing of the image to improve this aspect. So we can express the external energy function as :

$$E_{ext} = P(\gamma(s)) \tag{2}$$

Where P stands for a potential attraction field onto the edge of an object. So the underlying idea here is to use an attraction field and a potential energy on that field that we will minimize to reach an edge of an object. So we have to consider this energy not only locally but for the whole contour $\mathcal{C}$ and to plug in the previous equation the properties we mentioned above.

If the contour is closed we have:

$$E_{ext} = \oint_{\mathcal{C}} ||\nabla I||^2(\gamma(s))\ ds \tag{3}$$

Or if we just consider a non closed contour going from A to B:

$$E_{ext} = \int_A^B ||\nabla I||^2(\gamma(s))\ ds \tag{4}$$

Where I is representing the input image and ∇ is the spatial gradient function defined by:

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) \tag{5}$$

Due to the fact that we want to minimize this energy, we are going to take the opposed value and introduce our Gaussian smoothing to enforce convergence to a local minimum:

$$E_{ext} = -\int_A^B ||\nabla(G_n * I)||^2(\gamma(s))\ ds \tag{6}$$

Where G_n is a Gaussian weighted kernel of dimension n.

Using a weighting parameter on the external energy will allow us to increase the "visibility" of the gradient field by the snakes. So we can write it :

$$E_{ext} = -\delta \int_A^B ||\nabla(G_n * I)||^2(\gamma(s))\ ds \tag{7}$$

Where δ is a real weighting value which for obvious reason would be positive.

2.2.2 Internal Energy

The internal energy is the component of the behaviour function that describe the physical properties of our contour like smoothness or continuity and curvature. It is composed of two terms, the first one is describing the contour behaviour regarding elasticity or smoothness so it's a function of the first derivative of our contour. The second term is describing the curvature model of the curve and is function of the second derivative of our contour. If we put that into a mathematical form, we have:

$$E_{int} = f(\gamma'(s)) + g(\gamma''(s)) \tag{8}$$

The functions f and g are just going to be the Euclidean norm of the function. Then we need to specify that we want the energy for the whole curve $\mathcal{C}$ so not only for one spatial location s so we are going to sum this energy along the curve. Which will lead to two different cases.

If our contour is closed:

$$E_{int} = \oint_{\mathcal{C}} ||\gamma'(s)||^2 + ||\gamma''(s)||^2\ ds \tag{9}$$

Or if we just consider a non closed contour going from A to B:

$$E_{int} = \int_A^B ||\gamma'(s)||^2 + ||\gamma''(s)||^2\ ds \tag{10}$$

The previous definition is defining a model of smooth curve or function to describe the contour's behaviour, which goes through the use of it's derivative and in our case in their minimization. But, depending on the application and the choices of the user, the influence of these two aspects are not always equally important. So they need to be adjustable according to the situation. So the final definition of internal energy will be given by:

$$E_{int} = \int_A^B \alpha||\gamma'(s)||^2 + \beta||\gamma''(s)||^2\ ds \tag{11}$$

Where $(\alpha, \beta) \in \mathbb{R}^2$ in a general definition but we will see that there are certain restrictions to their possible values. We also chose the general case of contour, because indeed the closed contour is just a particular case where $A = B$. Here is an illustration of how minimizing internal energy will convert a used initialized shape into a lower internal energy contour:

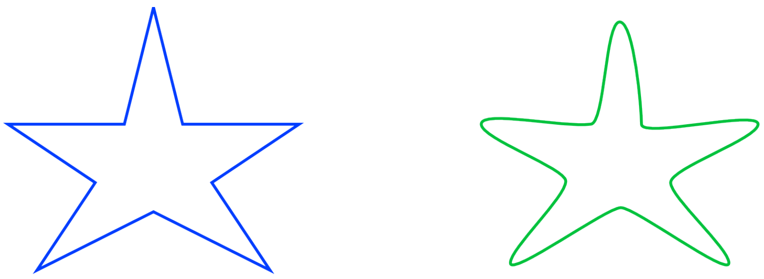

Figure 2: Effects of minimizing internal energy

Indeed, as we can see on the previous image, a shape with straight and sharp angles requires a lot more internal energy than a more continuous shape.

2.3 Contour global energy

So thanks to what we presented before we can present the contour energy function :

$$E_{snake} = E_{int} + E_{ext} = \int_A^B \alpha||\gamma'(s)||^2 + \beta||\gamma''(s)||^2 \; ds - \delta \int_A^B ||\nabla(G_n * I)||^2(\gamma(s)) \; ds \tag{12}$$

As we said before, this active contour method relies on an energy minimization technique. So now that we have defined the energy function we need to set up a method to minimize it.

2.4 Optimization

Our goal in this section is to find the optimal parameters that will minimize the previous energy function we defined. This parameters are position vectors that will define the position of the snake which minimize this energy. Indeed we are looking for $\gamma(s) = (\gamma_x, \gamma_y)(s)$ such as E_{snake} is minimal.

The optimization formulation of our problem is then:

$$s_{optimal} = \underset{\gamma \in \mathcal{F}}{argmin} \; \mathbf{E}(\gamma(s)) \tag{13}$$

Which means that we want to find the value of s which is the argument that makes the curve γ, of the set of possible curve, of minimal value regarding the energy function $\mathbf{E}$.

In finite dimension one of the most common method is to use the gradient descent method. Indeed this method comes from function analysis because we know that if the gradient of a function is equal to zero it means we reached a local extrema of the function.
The gradient descent is a first order optimization technique based on the first order Taylor series decomposition. Other methods using higher order decomposition could also be used such as Newton's method. In this project, the gradient descent technique will be used when we will present the Euler Lagrange snake implementation which relies on it.

The general set up of such an optimization problem involves small variations modelled using a time evolution variable and the following equation:

$$X^{k+1} = X^k + t^k d^k \tag{14}$$

Where t represent the time step used and d represent the direction in which this time step modification is performed. In the gradient descent method, as we explained before, we rely on the first order Taylor decomposition :

$$f(X^k + t^k d^k) = f(X^k) + t\nabla X^{kT} d \tag{15}$$

In this optimization problem we are trying to solve, we want to minimize the energy function in its time evolution. This leads to the following imposed condition on our algorithm:

$$f(X^{k+1}) < f(X^k) \tag{16}$$

So finally using the previous condition on the previous Taylor decomposition, we end up with the optimal direction evolution formulation :

$$\nabla X^{kT} d < 0 \Longrightarrow d = -\nabla X^k \tag{17}$$

The following picture illustrates the idea of this algorithm in one dimension space, but could easily be extended to multidimensional spaces.

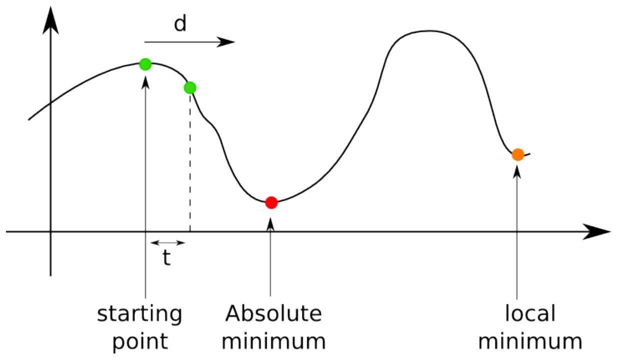

Figure 3: Optimisation parameters

As shown in the picture above, one of the drawback of this numerical method is that the minimum we are going to find is not necessary the absolute minimum of the function and it really depends on the initialization. And in certain situations we might want not to find the absolute minimum but just only the closest local minimum.

2.5 Initialization

As we presented in the previous section when we discussed optimization algorithm, our algorithm will converge to a local minimum which is not necessary the absolute minimum of the function or the one that suits the shape we want to segment. Moreover, an initial condition is necessary to initialize the optimization. To increase the success of convergence to the right minimum, we are going to rely on the user to initiate the contour around the shape he is interested in.

This aspect will be supported by a function we already developed in a previous project which is a point selection algorithm. It will be probably also be modified to work with closed contours.

3 Active Contour Model

3.1 Gaussian smoothing

An important part of the set up to perform a segmentation using active contour is to pre process the image using a Gaussian filtering. In a previous project, we have already explored the implementation using convolution. In this project we are going to present an other use of a smoothed image and we are going to explain why this step is important.

If our input image has a good quality, its edges are going to be extremely well defined, which means that their are going to be abrupt. This is going to be bad for our optimization algorithm and the snake energy evolution because the external energy relies on the use of the image gradient. In fact, we need to attract progressively the snake to the edge of the object, so we need to "attract" it. To do that we need to "spread" the edges around to create a more smoothed gradient around the edge. This aspect was presented in project 2 where we showed that filtering was "spreading the edge around". Here is the result we obtained at that time:

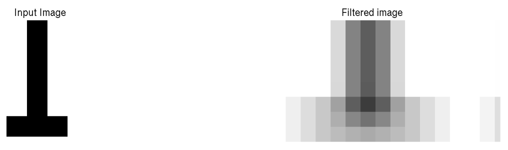

Figure 4: Filtering effects on edges

And this is really a property we want to have to give to our gradient : a smooth behaviour:

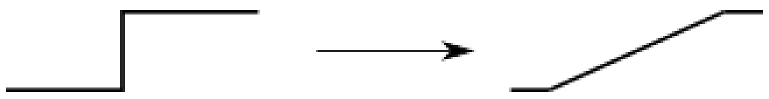

Figure 5: Edge transformation used to compute the gradient

So we decided to look at the gradient field behaviour in such a situation. To do so we considered a simple nicely defined edge of an object and then computed the gradient field around that edge. Our test image was just composed of a square shape where we firstly did not apply filtering and then we applied it.

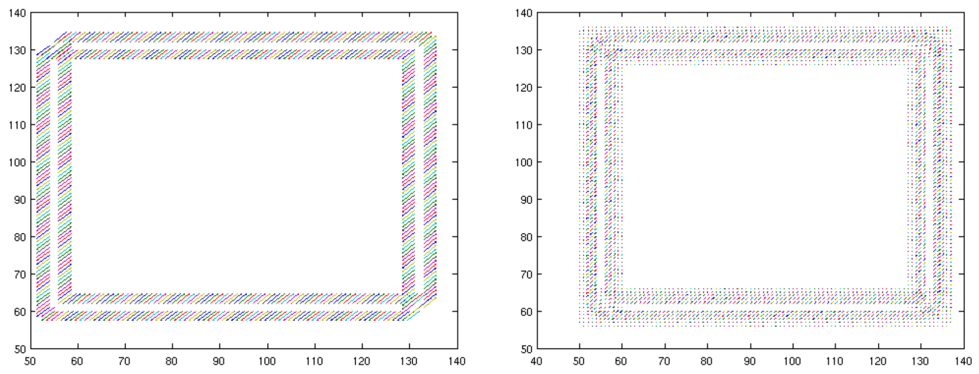

Figure 6: Comparison between the original and smoothed gradient field around an object edge

As we can see on the previous picture, the results looks pretty different. In fact, on the first image, due to the sharp structure of the edge the gradient is defined only for one single pixel on each side of the edge. On the second image, due to smoothing, the edge has been spread so our gradient is defined for a larger number of pixels.

An other important thing regarding our snake presentation is that the edge is clearly, in both situations, characterized by a set of pixels where the gradient is equal to zero, which represent an extrema of the intensity distribution function along a single row or line of pixels of the image.

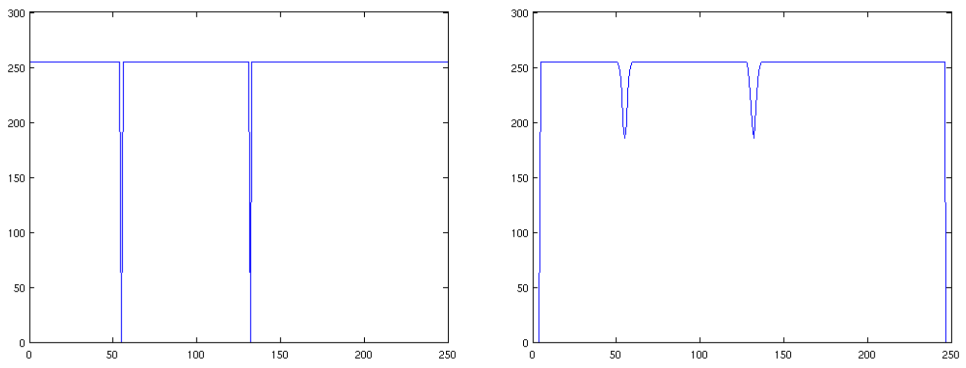

Figure 7: Comparison between the original and smoothed intensity function for a slice of the image

On the previous image, we compare the intensity distribution along one row of the image and we can clearly see that in both case, at edge location we reached a minimum where the gradient is equal to zero. Smoothing has an influence on the size of the spread edge and on the value of the local minimum. Indeed, the bigger the spread the lower the height of the gap associated to the edge.

An other important part of our external energy function is to take the opposite of the gradient to invert its direction. Here are illustrations that shows why.
So here is a comparison between the original and the smoothed gradient field around an edge of an object.

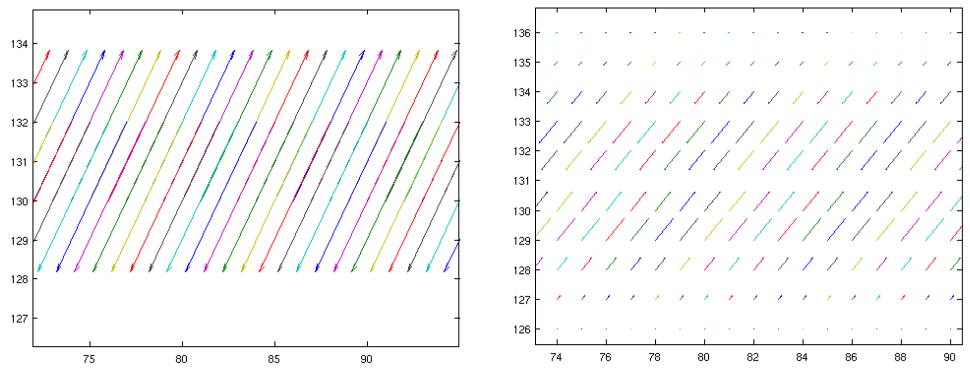

Figure 8: Comparison between the original and smoothed gradient field around an object edge

We can clearly see on the previous image the important properties we mentioned before. Firstly the gradient field is directed to the edge instead of going away, so it will attract the snakes. Secondly this gradient field is spread more around the edge to have a wider influence. Finally this spread is dependant on the kernel size and a trade off with the initialization distance.
When computing the external energy, we take the opposed value of the gradient norm. Indeed, we can see on the image that the gradient norm is increasing while getting closer to the edge, so because we want to minimize energy we need to make it become lower as we get closer.

An other reason of applying a Gaussian filtering is regarding the noise issue. In fact, it's pretty common to have noisy images, and noise is modifying a lot the external energy of the image. So by smoothing the image we intend to reduce in a certain way the influence of noise. We will describe this aspect with illustrated results.

3.1.1 Kernel settings

The kernel size and type is really important as we mentioned before because it influences the way the edge is spread around. It also as a consequence of our simple convolution implementation reduce the size of the resulting image. This aspect can of course be noticed while looking at the intensity curve of the smoothed image where we can clearly see null areas on the sides of the intensity curve. There is a choice to make in the design of the filtering kernel that we will try to illustrate with results.

3.2 Contour Construction

As we said before, we are going to rely on the user to define an initial shape around the object that will serve as an initialization set up. Here are some results of user initialized shapes to perform segmentation using active contours.

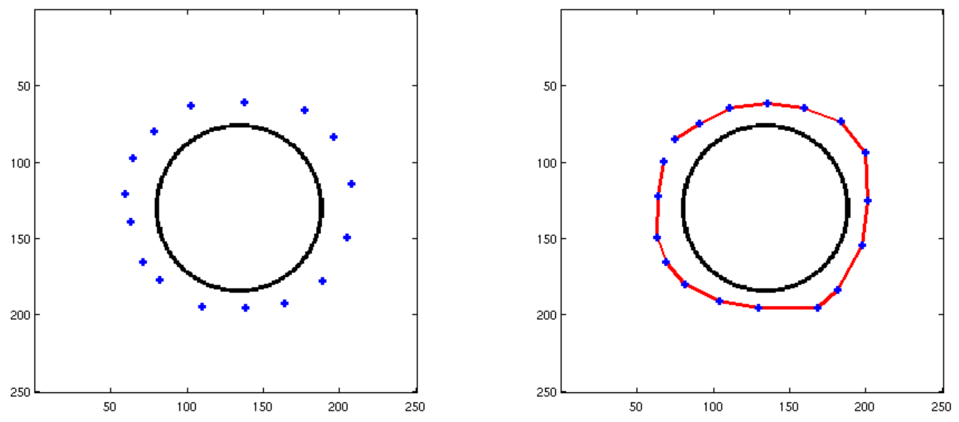

Figure 9: Contour initialization

3.3 Greedy method

The greedy method is an implementation technique used to simplify the implementation of the minimization of energy without having to perform an optimization algorithm technique such as the gradient descent. It works under the assumption that finding for each point of the contour the closest local energy minimizing neighbour will converge to the overall global minimum of the contour. In general algorithm implementation, a greedy algorithm is rarely a good and optimal solution. So we are going to try to implement it and show how it works but we will also present a more accurate solution using global minimization.

For each point which belong to the contour we made, we we going to compute the energy for a small neighbourhood of surrounding pixels. If it exists a point where the computed energy is lower than the current energy we are going to move our contour point to this new location. This will modify the curvature with its contour neighbours but this will be taken in account when computing the energy of the following point. We perform these operations for the whole contour as long as it exists points with lower energy in the neighbourhood of each contour point.

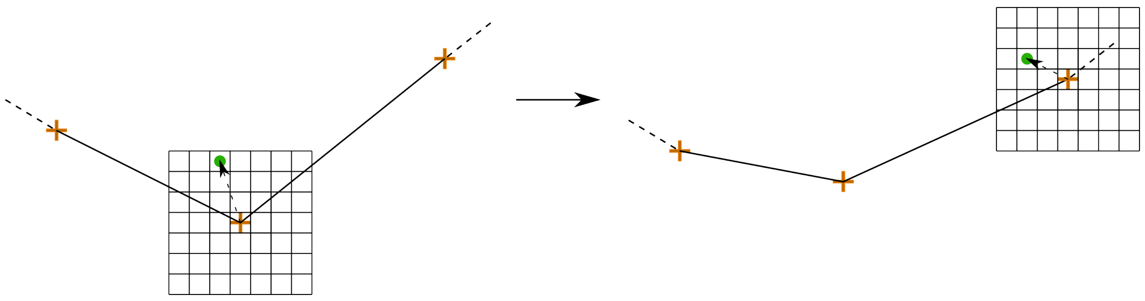

Figure 10: Greedy algorithm illustration

The previous figure shows how the greedy algorithm will go from pixel to pixel to move it to a new location of lower energy. To do so we will rely on the computation of the energy we presented before. Here is a presentation of the external energy driven minimization. It means that at that point we don't use the internal energy of the curve. So our contour is a non flexible contour. The first step as presented before is to smooth the image to spread the edge gradient field to attract the snake. The concern is that here we would need a very large filter to spread it as much as possible. So here, the type of convolution we use, and the way we define our Gaussian filter is really important. Because even if our image is pretty small the bigger the kernel the longer the smoothing. So generating the Gaussian kernel based on convolution of box functions (Heavyside functions) is a way to generate two 1 dimensional kernel to use the separability property of this implementation of the Gaussian filter to have a quicker smoothing computation. Here is a result of smoothing on our test image :

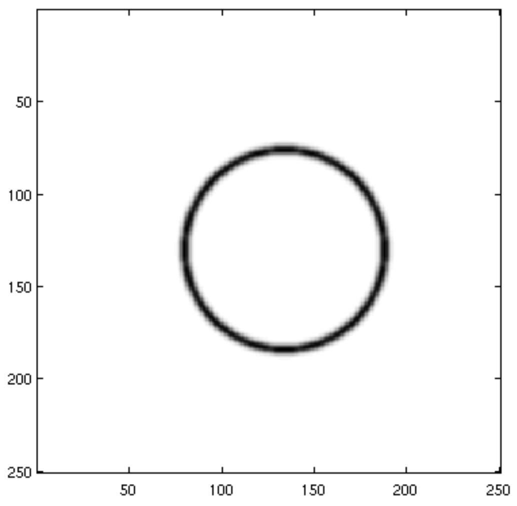

Figure 11: Smoothed cercle test image with generated Gaussian kernel of dimension 9

Then we use the computation of the gradient map we made to get the gradient magnitude at each point and then make the contour move to a new location with lower external energy which means closer to the boundary of the image.

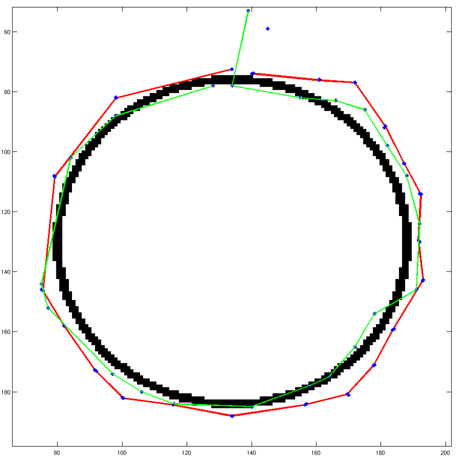

Figure 12: The red contour is the initialization contour and the green one the modified contour

This result is just a preliminary result after a couple of iterations, but it shows that some points really get attracted by the contour. An other issue at stake is the size of the kernel we use to compute external energy around our current contour point. Indeed the bigger could appear the better to move quickly to the border, but the risk is to be attracted by positions that are maybe not the one we want to go to.

3.3.1 Stopping criterion

In the original method presented by Kass *et al.* it seems to have no indications about a stopping criterion on the snakes evolution. When using the greedy method we have two stopping criterion. The first one is if the number of points that move at each iterations fall under a certain threshold value. The other one being that the number of iterations reaches another threshold value. Let's call ϵ the threshold on the number of points and η the threshold value on the number of iterations.

Regarding the set up of the η value, it mainly depends on the user choice, or the programmer choice. Indeed this number of iteration will have to be set in relation to the size of the kernel we use to compute external energy and the distance between the initialization contour and the targeted shape. Indeed, if we initialize the contour too far from the shape with a small kernel and a too low iteration threshold the algorithm will fail in reaching the shape. On the other hand if we have a too large number of iterations and a too close initialization we will spend time looking a contour that oscillate in the edge origin without significant improvements. This is maybe where the ϵ criterion can become useful.

In fact regarding the set up of the ϵ threshold on the number of modified points per iterations on the contour it's pretty hard. In fact, we have to keep in mind that we are dealing with an active contour on a discrete space. So the nice minimum properties we have in a well defined continuous space might no longer exist and we can find several points minimizing energy. This will lead to multiple points which slightly move at each time because the displacement of only one point could modify curvature and continuity of the whole structure and make all of the other points slightly move. So this last criterion seems to be really tough to set up correctly. We introduced it to explain how we can constrained the snake's evolution but we are not going to use it.

3.4 Results

The first test to see if our implementation was working was to analyse the evolution of the contour. In this implementation, we had to see each point being displaced over time in the way we selected them because we are iterating over points to make them move. This test was successful, so we could test the accuracy of our implementation regarding the segmentation objective to have a curve that will match the edges of an object.

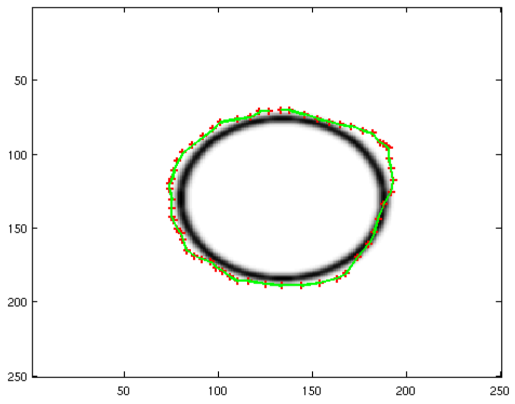

Figure 13: Result of the greedy algorithm on a circle

It does not appear really smooth and really good, because the set up of the α and β coefficient does play an important role in the shape of the contour and in some situation such as the previous one it's better to enforce the smooth and continuous behaviour using important values for those coefficients. Another things at stake in this implementation is the size of the smoothing kernel in relation to the size of the minimization window. In fact we have to use a minimization window *i.e* the neighbourhood around which we look for lower energy values for a given point on the snake. This minimization window has to be of the same size of the kernel we used to produce the best results. Indeed we think there might be a relationship between the way we smooth the edge and the way we are looking for an lower energy point. An other thing to mention is the constraint with this implementation to have an initial contour which is not too far from the object we try to segment, because if it's too far the convergence is not ensured because the attracting gradient field will not be visible enough. Taking in account all the previous aspects we mentioned allowed us to produce some not too bad results using a bit more complex shape.

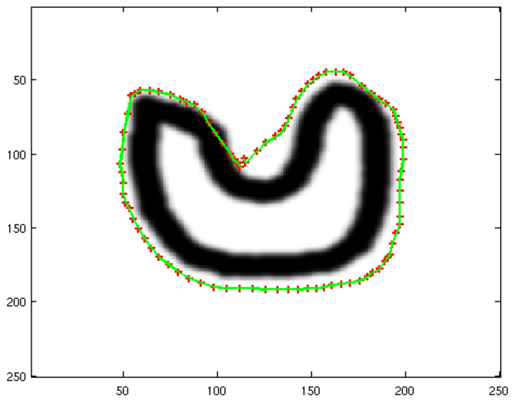

Figure 14: Greedy algorithm producing not too bad result using few iterations

The number of points and their spacing along the contour is also a very important point in this method where we iterate on those points position. Indeed, in this implementation we do not interpolate between points, we just link them with lines, so if there are too few points the results is "blocky" so we need to define a lot, but defining them with constant spacing is really hard.

As we showed before, this successful convergence of this algorithm and probably of the implementation we made, and which could be poor, remain uncertain. Indeed, it depends on the initialization, on the most accurate set up of the weighting parameters, on the numbers and the way points are defined and finally on the number of iterations. This might appear bad, but there is also some good points. The internal energy minimization works pretty well, because we ended with contours smooth and continuous even id the initialization was far from it. And secondly the convergence works in most cases, but due to the fact that the modification of one point is propagated through internal energy and the fact that we did not enforce the gradient descent method, our contour can sometimes diverge instead of converging to the shape in order to minimize the internal energy.

I was not completely satisfied with the results I obtained, my implementation might not be the best, so I wanted to work on a different method, where the mathematical properties rely on optimization techniques and whose implementation is using matrices and numerical solutions of differential equations. This method rely on calculus variations and is Euler Lagrange method and is presented and analysed in the second part of this report.

4 Euler Lagrange Minimization

4.1 Theoretical presentation

As we mentioned before, we have to deal with an optimization problem, an energy minimization, to find the most accurate position of our snake. An other way to solve that issue is to use Euler Lagrange method. Indeed we want to solve the following problem:

$$\gamma_{optimal} = \underset{\gamma \in \mathcal{F}}{argmin}\ \mathbf{E}(\gamma(s)) \tag{18}$$

Euler Lagrange method is using calculus of variations to solve that issue and state that to minimize energy using higher order derivatives. Here is an overview of this method.
Indeed Euler Lagrange theory state that to minimize

$$\int E(s, \gamma, \gamma', \gamma'')ds \tag{19}$$

We need to solve :

$$\frac{\partial E}{\partial \gamma} - \frac{\partial}{\partial s}\frac{\partial E}{\partial \gamma'} + \frac{\partial^2}{\partial s^2}\frac{\partial E}{\partial \gamma''} = 0 \tag{20}$$

To solve the previous equation, we convert it into a time differential system where our contour function $\gamma(s)$ becomes a time dependent function $\gamma(s,t)$ so the equation to solve becomes :

$$\frac{\partial \gamma}{\partial t}(s,t) = -\frac{\partial E}{\partial \gamma}(s,t) - \frac{\partial}{\partial s}\frac{\partial E}{\partial \gamma'}(s,t) + \frac{\partial^2}{\partial s^2}\frac{\partial E}{\partial \gamma''}(s,t) \tag{21}$$

So if we plug in our definition of the energy function $\mathbf{E}$, we obtain the following equation to solve:

$$\frac{\partial \gamma}{\partial t}(s,t) = \alpha\frac{\partial^2 \gamma}{\partial s^2}(s,t) - \beta\frac{\partial^4 \gamma}{\partial s^4}(s,t) + \delta\nabla(||\nabla(G_n * I)||^2)(\gamma(s,t)) \tag{22}$$

Due to the dimension of our γ function which is defined on $\mathbb{R}^2$ we can decompose the previous equation into a system of two scalar equations that can be numerically solved:

$$\begin{cases} \frac{\partial \gamma_x}{\partial t}(s,t) = \alpha\frac{\partial^2 \gamma_x}{\partial s^2}(s,t) - \beta\frac{\partial^4 \gamma_x}{\partial s^4}(s,t) + \delta\nabla_x(||\nabla(G_n * I)||^2)(\gamma(s,t)) \\ \frac{\partial \gamma_y}{\partial t}(s,t) = \alpha\frac{\partial^2 \gamma_y}{\partial s^2}(s,t) - \beta\frac{\partial^4 \gamma_y}{\partial s^4}(s,t) + \delta\nabla_y(||\nabla(G_n * I)||^2)(\gamma(s,t)) \end{cases} \tag{23}$$

The following picture illustrates what we presented so far regarding this method:

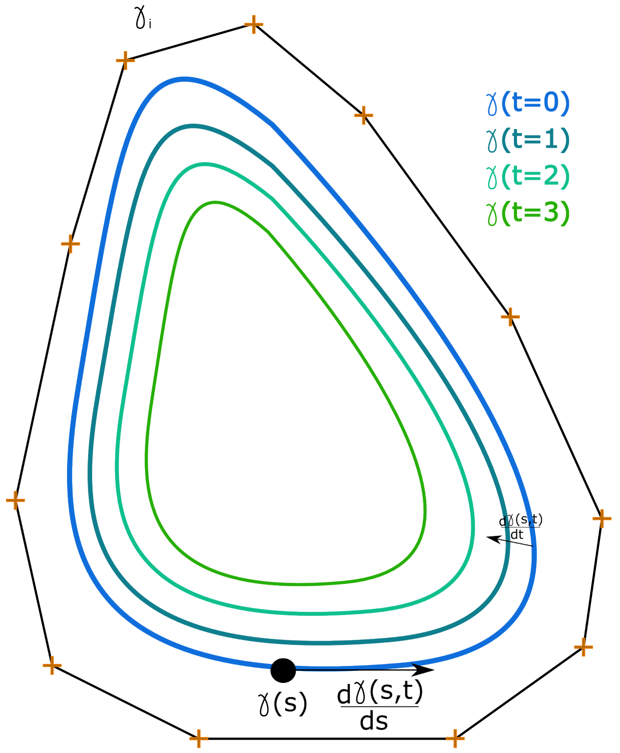

Figure 15: Euler Lagrange optimization method

4.2 Discrete space implementation

In a image, and in general problems where we want to solve differential equations, we evolve in a discrete space. Indeed, the previous equations we presented might not have a closed form solution, so we need to use an approximation with a convergence criterion of the optimization algorithm. For an implementation issue, we are going to consider a closed contour.

4.2.1 Spatial discrete implementation

In the previous continuous equations we have second and fourth order derivatives of the curve to compute. To compute them in a discrete space we can rely on a technique we already used for discrete filtering where we implemented discrete derivative filters using Pascal's coefficients. In a previous assignment we have already shown that these coefficients can be obtained using box convolutions.
So our second order derivative will be defined by:

$$\frac{\partial^2 \gamma_x}{\partial s^2}(s,t) = \gamma_x(s_{k+1},t) - 2\gamma_x(s_k,t) + \gamma_x(s_{k-1},t) \tag{24}$$

Which can then be computed as a matrix

$$\mathbf{D_2} = \begin{bmatrix} -2 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2 & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \tag{25}$$

And out forth order derivative is defined by:

$$\frac{\partial^4 \gamma_x}{\partial s^4}(s,t) = \gamma_x(s_{k+2},t) - 4\gamma_x(s_{k+1},t) + 6\gamma_x(s_k,t) - 4\gamma_x(s_{k-1},t) + \gamma_x(s_{k-2},t) \tag{26}$$

Which has the following associated matrix:

$$\mathbf{D_4} = \begin{bmatrix} 6 & -4 & -1 & 0 & \cdots & 0 & 1 & -4 \\ -4 & 6 & -4 & 1 & \cdots & 0 & 0 & 1 \\ 1 & -4 & 6 & -4 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & \cdots & -4 & 6 & -4 \\ -4 & 1 & 0 & 0 & \cdots & 1 & -4 & 6 \end{bmatrix} \tag{27}$$

We now have a discrete version of two of the three members of the right part of our previous equation. The last one end up being the spatial derivative of the Potential energy made from the convolution of the gradient of the image with the Gaussian kernel.

4.2.2 Temporal discrete implementation

As we did before we are now going to compute the discrete temporal derivative of $\gamma(s,t)$. The temporal variable is going to be described as $t_k = k\Delta t$. So we have ;

$$\gamma_x(s,t_k) = \gamma_x(s,k) \tag{28}$$

So if we apply a time differentiation on the γ function we obtain:

$$\frac{\partial \gamma_x}{\partial t}(s,t) = \frac{\gamma_x(s,k) - \gamma_x(s,k-1)}{\Delta t} \tag{29}$$

4.2.3 Conclusion

If we conclude the previous presentation, we introduced matrix notations, an easy way using differentiation to compute a temporal derivative. So if we combine all of the previous, we end up with an easier and clearer discrete system to implement and solve.

$$(Id + \Delta t(-\alpha D_2 + \beta D_4))\gamma(s,k) = \gamma(s,k-1) + \Delta t P(\gamma_x(s,k-1), \gamma_y(s,k-1)) \tag{30}$$

Which end up being :

$$A\gamma(s,k) = b(s,k-1) \Rightarrow \gamma(s,k) = A^{-1}b(s,k-1) \tag{31}$$

So it means that we can numerically, with a discrete implementation solve the equations presented before.

This leads to a comment about the implementation we are going to make. Contrary to the greedy method where we scan a neighbourhood around a point to move the snake to, here, we are computing the new coordinates based on equations. So we will need to perform an interpolation to get the location of pixels that suit to the result we are going to obtain using our equations.

4.3 Results

The process is the same as in the greedy implementation, we apply our Gaussian Filtering on the input image and then ask the user to implement an initial shape. Then we solve the previous system to compute the evolution of the contour. The use of the discrete derivation matrices $\mathcal{D}_2$ and $\mathcal{D}_4$ will allow us to compute very easily and efficiently the continuity and the smoothness of the curve. We then use the gradient information to compute the external energy. Once we have everything, we can set up the matrix $\mathcal{A}$ which describe the whole energy system we described above. And finally we iterate until the energy reaches a local minimum and after a certain number of iterations.

Here is a result of the implementation we described applied on a set of images:

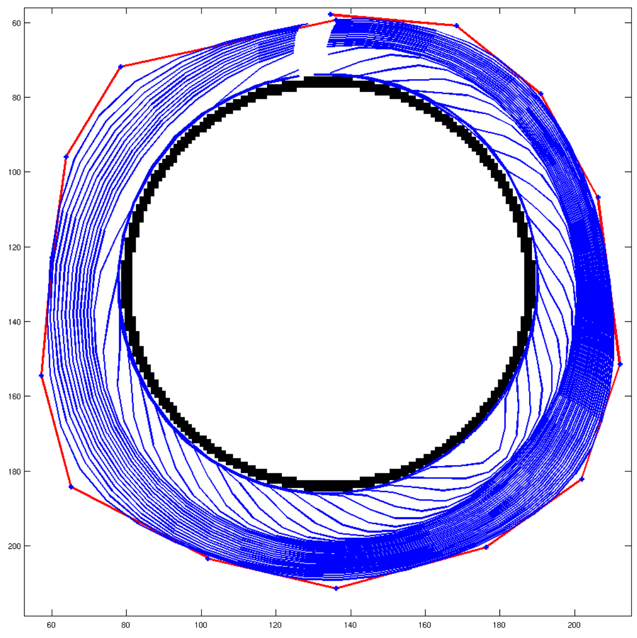

Figure 16: Iterations of the snake before convergence

As we can see on the previous picture, we initialized our snake ”relatively” far from the targeted test shape. During the first iterations the snakes moves but very slowly. This is due to our extension factor on the gradient field. Indeed, the gradient magnitude field is, after smoothing, really low in intensity. So to make it visible we multiply it by a very high value that will make it visible by the snake. But visible does not imply that the variation will be important. But once we get closer and closer to the targeted shape the magnitude gets bigger and bigger so it attracts the snakes more strongly which increases the distance between the intermediate contours to finally have enough energy to jump to the boundary of the shape. Here is displayed the last iteration of the snake and as we can see it really matched the edge pretty well.

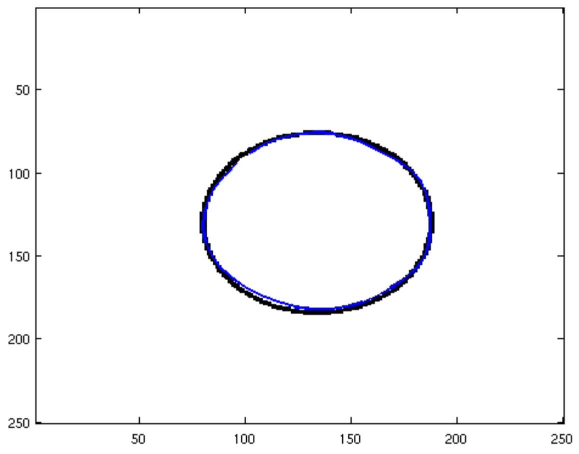

Figure 17: Our snake converging on the boundary of the circle test image

The circle is an easy shape to test the convergence of the contour because it's "shape energy" is low. In fact there is no strong corners so no points where the snake will need to have a higher internal energy due to the corner.

So an other test structure will be to use a square to illustrate the previous point related to corners and to show that we need to adjust the values of our parameters to have a better fit to the shape we try to segment.

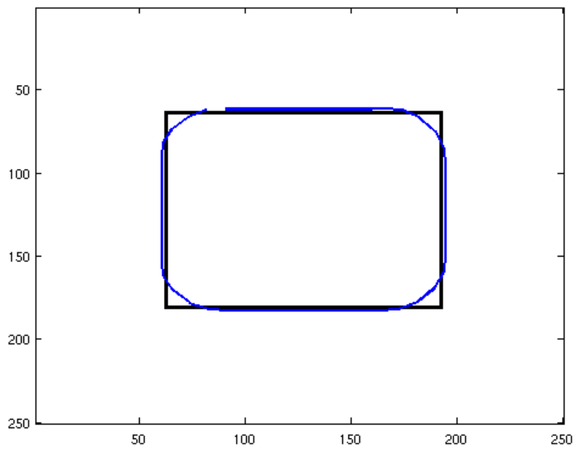

Figure 18: Convergence of snake to square shape

Indeed, as we can see on the previous picture, if we use the same parameters as we used with the circle shape, we can clearly see that the most important point is to have a smooth and continuous contour which does not succeed with angular structures such as the square. So let's reduce the influence of those two parameters.

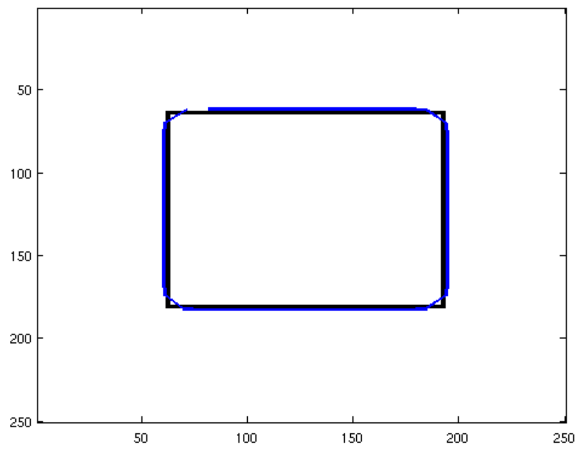

Figure 19: Convergence of snake to square shape

This modification allowed our contour to match more the corners of the shape but not completely, so a more significant modification, by setting the values of α and β below one should have a stronger effect.

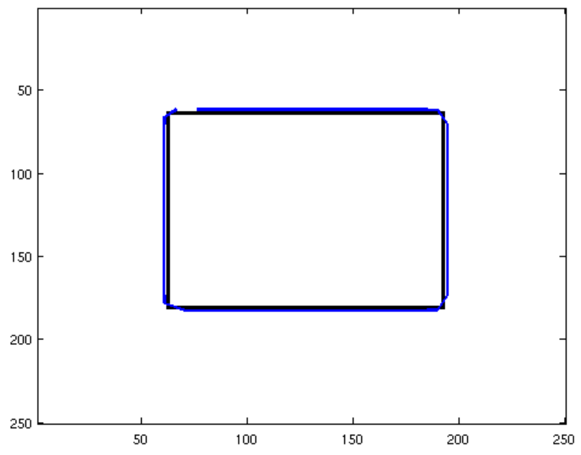

Figure 20: Convergence of snake to square shape

And this is a better result, but not perfect, because this would require some improvements to be able to set to 0 the parameters when reaching a corner and keep them to regular values on other part of the contour to ensure continuity and smoothness elsewhere. Indeed, this aspect could be linked to mathematical issues of convergence. In my opinion, the most suitable example would be the approximation of discontinuities in step functions using Fourier Transform. That's exactly the same idea, where we try to map a smooth and continuous shape onto a discontinuity. For our global snake presentation and implementation, solving this issue would require a corner detection of the image, and a way to define a transfer function for the α and β coefficient according to the properties of the shape.

Here is another example of this aspect which illustrate the previous point with a more complex structure.

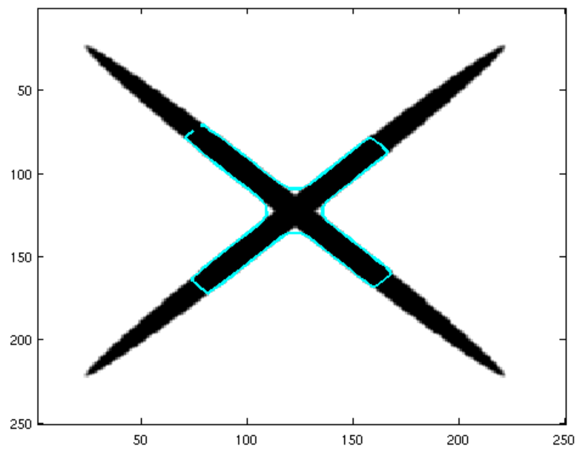

Figure 21: Convergence of snake to the shape with $\alpha = 2.0$ and $\beta = 2.0$

As we can see on the previous result, having high values of continuity and smoothness does not work on this type of structure. But as soon as we lower them we obtain a better fitting of the contour on shape.

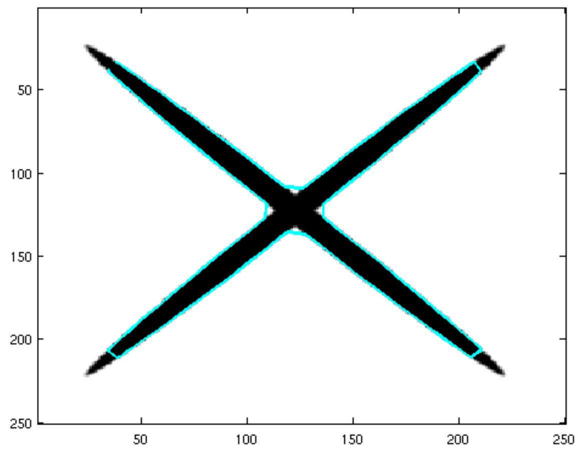

Figure 22: Convergence of snake to the shape with $\alpha = 0.1$ and $\beta = 0.6$

We cannot set them to 0 because we need to be able to optimize the curve energy to make the snake evolve over time. Let's conclude these examples with applications on segmentation of real objects.

In this test we made, we wanted to segment the corpus callosum from an MRI image. The point of this example is to emphasize the necessity of a preprocessing.

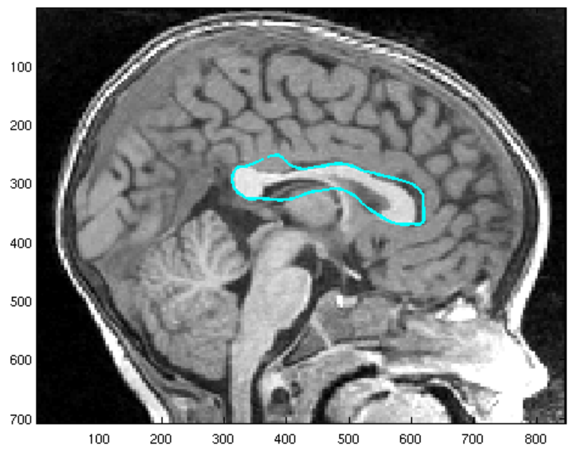

Figure 23: Corpus callosum segmentation using our active contour

As we can see, the contour match more or less, but the convergence is really really slow, due to a non uniform gradient map. Indeed, we don't have a minimum clearly defined, even if there is a slight black area around the corpus callosum we have no proof that the contour will be attracted. So two things need to be done to make it work faster and more accurately. The first thing is to threshold the image to reduce the surrounding elements. In our case this is pretty easy because the corpus callosum is white and the surrounding white matter appears light gray. Then once we have done that to work with the minimization strategy presented above we need to invert the image intensity which just consist in applying the following equation:

$$\text{NewIntensity} = 255 - \text{OldIntensity} \tag{32}$$

Because corpus callosum is a main component of the brain, after thresholding that's an important feature that remains very well defined.

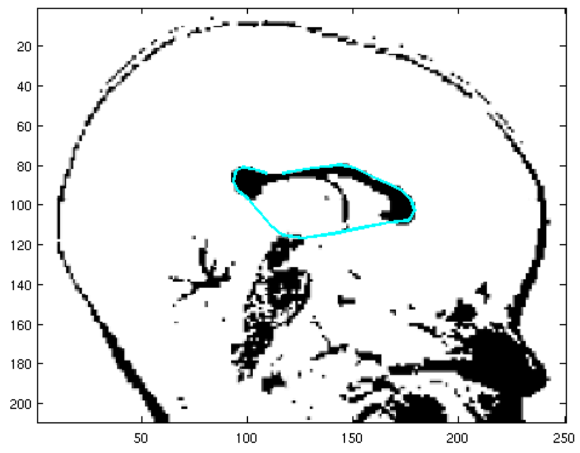

Figure 24: Inverted Binary Corpus callosum image segmentation using our active contour

This result, which is not very good, provide us the opportunity to discuss the user initialization step. Of course the convergence to what we want to segment depend on the surrounding environment. If we are too far it could take very long to reach the desired object and in our case with a limited number of iterations it could also never reach it. Or we can have what happens in the previous image and our contour is initialized to close from other objects and stay stuck to them instead of moving. So for all those reasons, a good preprocessing aiming at isolating our area of interest from other structures, reducing noise and inverting image is important.

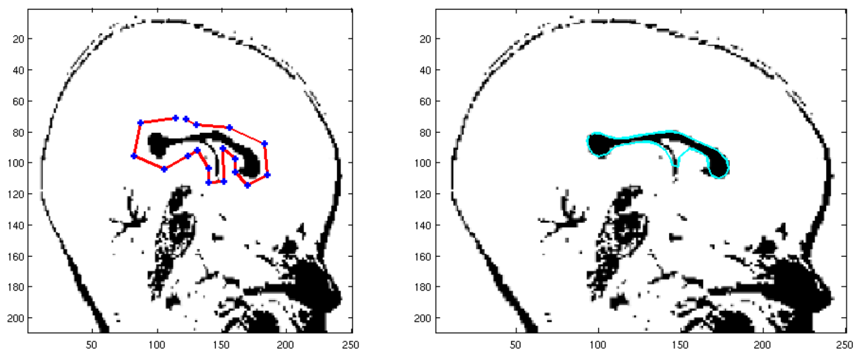

Figure 25: Good user initialization and resulting segmentation

There are other aspects we can present which represent some drawbacks of the method we implemented. In fact, we only implemented a contracting snake, which means that it will only try to match with structures by contracting it's shape. And that could become an issue when the user initialization is close to the original shape and that the shape present protuberances where it would be nice to have a dilating behaviour to match them completely.

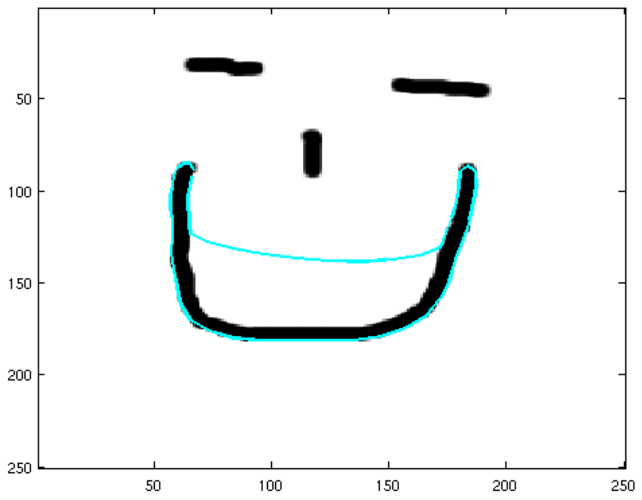

Figure 26: Example of a "length" or contraction issue

Indeed in this case we can clearly see that the length of the snake and its elasticity is too low so it cannot match perfectly with the shape. To make it match we should be able to extend the contour on the inside part of the shape where it does not match. This is one of the major issues of the current used snake algorithm which does not have a strong and wide enough gradient field to attract the curve and make it deform. This issue is known as a concavity issue and could be solved using more advanced methods or by increasing a lot the attraction weighting factor to make the gradient of the image very large and create an initial contour close to the shape.

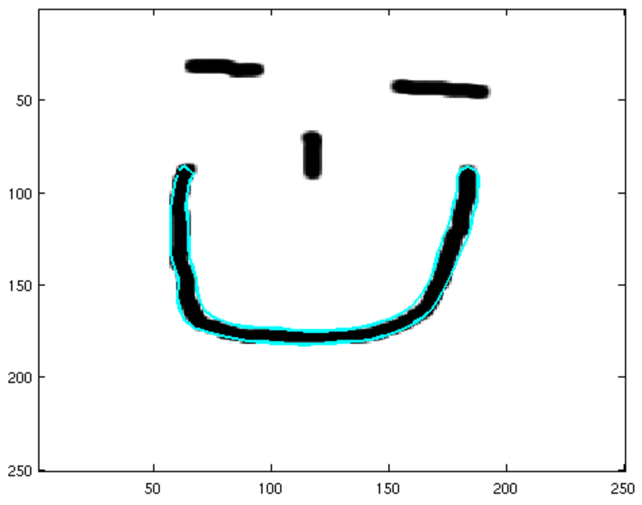

Figure 27: Solution to concavity issue with a very important value of δ : 900

And a direct consequence of this is that whatever happens our method will stick to connected shapes. In fact, the shape cannot split in several smaller contours according to encountered objects. Some techniques allow to do that but it's impossible to do that based on our implementation. So if several objects are included in our initialized contour, we are going to produce a single contour, that will have minimal energy according to what it will encounter. Here is an illustration.

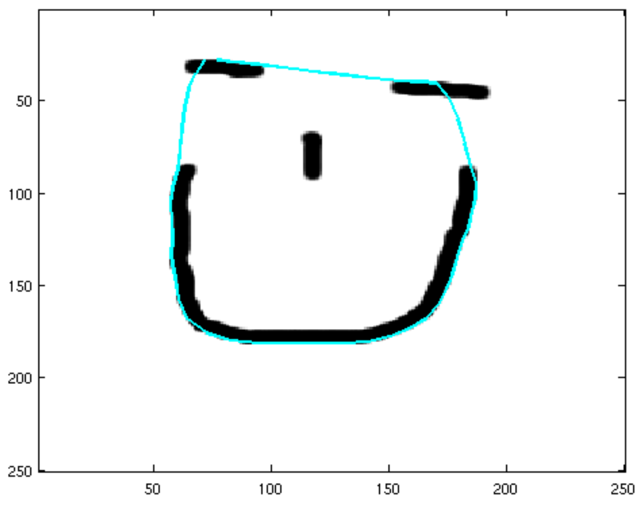

Figure 28: Example of a connection issue

In the previous result, the dominating structure is the big U shape at the bottom which is going to influence a lot the rest of the contour and will even force the contour to go inside some objects to optimize energy.

4.4 Variable weighting factors

As we defined in our contour energy function:

$$E_{snake} = E_{int} + E_{ext} = \int_A^B \alpha||\gamma'(s)||^2 + \beta||\gamma''(s)||^2 \; ds - \delta \int_A^B ||\nabla(G_n * I)||^2(\gamma(s)) \; ds \tag{33}$$

We have weighting factors : $(\alpha, \beta, \delta) \in \mathbb{R}^3$ in what we presented before, we assumed that these weighting factor would be constant values. But we could also have chosen to define them as function of the spatial position along the contour and have : $(\alpha(s), \beta(s), \delta(s))$

$$E_{snake} = \int_A^B \alpha(s)||\gamma'(s)||^2 + \beta(s)||\gamma''(s)||^2 \; ds - \delta(s) \int_A^B ||\nabla(G_n * I)||^2(\gamma(s)) \; ds \tag{34}$$

The advantage of this method has been presented before when we introduced the corner issue. Indeed, having variable weighting factors could provide a better fitting to our active contour but it will require to compute some derived images such as corner maps, and a matching between snake points and actual image features to adjust parameters.

An other technique of variable weighting factors could be find in the literature. Indeed, there is a technique of random generation of α, β and δ at each iteration. Indeed, we start with a user initialization and an initial set of values for those parameters. Then at each point we generate a set of random parameters, and try to optimize energy based on this new set of values. Several parameters set can be generated at each step to pick up the best set which will provide the most accurate fit with the shape we try to segment.

4.5 Generalization to 3 dimensions

Thanks to the model we presented and used through this project, we define our active contour with a function $\gamma(s,t)$ where the variable s is a space variable so the previous equations still stands for a 3 dimensions implementation of this technique with now $s = (x, y, z)$.

Most of the point we presented before are important steps and techniques that are used in professional software to perform active contour based segmentation. One of them is ITKSnap which use three dimensional snakes and with a first step being a preprocessing of the image aiming at the feature of interest. ITKSnap being an advanced an professional software it has more complex and accurate methods relying on ITK to implement active contours.

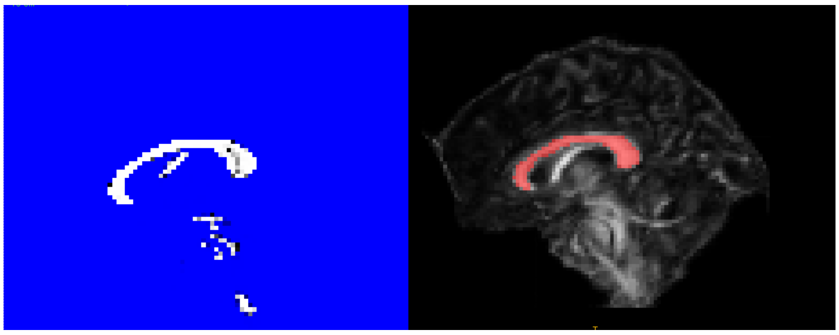

Figure 29: Threshold and active contour using ITKSnap

To obtain this segmentation we used an other possible implementation of the active contour which here could be called active surface called balloon snake. This idea here is to define a sphere inside the object and let the sphere grow to map the shape of the object as if we were inflating a balloon. We obtain then a 3D model of the corpus callosum that we can use for various purposes.

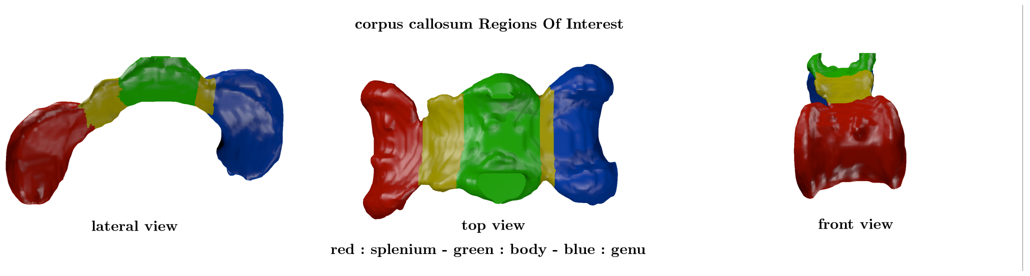
corpus callosum Regions Of Interest
lateral view
top view
front view
red : splenium - green : body - blue : genu

Figure 30: Example of 3D segmentation, labelled for tractography

4.6 Curve subdivision

As we presented before, to have nice and smooth results we need a lot of points to define our snake. This is pretty fastidious if the user has to select several dozen points to have a smoother contour has I did to present some results. So the idea here would be to have a function that will subdivide in equal length interval the line connecting two dots. By doing so the user will only have to define is approximate shape around the object with a dozen point and the function will provide us with several subdivisions on each segment to have smooth lines.

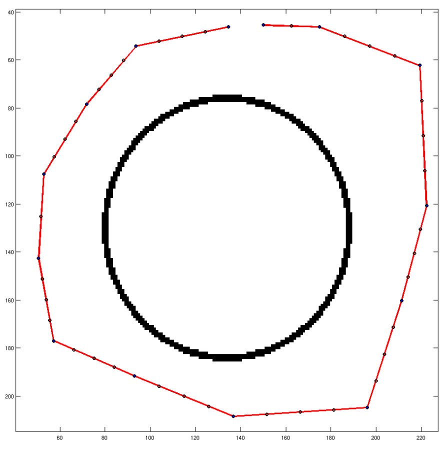

Figure 31: Example of subdivision of our initial contour

Here is an example where each segment has been equally divided with more points to give a smoother behaviour when the snakes evolves over time. This is a source of improvement regarding the user experience where it's more convenient to only define a raw shape with half a dozen points rather than defining manually 30 of them to have a "smooth" contour. There exists some code to do that which is available but here the goal was just to present a source of improvement, regarding our project it's not too bad to define a lot of points.

5 Implementation

5.1 Contour construction

As we mentioned in the theoretical presentation, our algorithm relies on an initial contour determined by the user. This contour serves as initialization of our energy minimization problem. To do so we are using a point selection that we already implemented in the third project when we where defining landmarks to perform registration.

```
but = 1;
while but == 1
   clf
   imagesc(I);
   colormap(gray)
  hold on
  plot(x, y, 'r-','linewidth',3);
   plot(x, y, 'b+','linewidth',3);
   axis square

   [s, t, but] = ginput(1);

x = [x;s];
y = [y;t];

end
```

5.2 Gaussian Smoothing

We explained that to enforce the convergence of our active contour to the boundaries of the shape we are trying to segment, we need to smooth the image. To do so we will rely on the implementation of convolution we already made in project 2 when studying filtering. The size of the kernel and its type (homogeneous, Gaussian...) was discussed before. In this implementation, we used box convolution to approximate a Gaussian filter as we already proved in project 2 that it was not a too bad approximation.

```
init = [1,1];
gauss = [1,1];
for i=1:9
     gauss = conv(gauss,init)/sum(2*gauss);
end
weight=gauss'*gauss;
for i = ceil(size(weight,1)/2) :1: size(I,1)-size(weight,1)+ceil(size(weight,1)/2)
     for j = ceil(size(weight,2)/2) :1: size(I,2)-size(weight,2)+ceil(size(weight,2)/2)
         convol=0;
         %compute convolution for the neighbourhood associated to the kernel
```

```
        for a = 1:size(weight,1)
            for b=1:size(weight,2)

            convol = convol + (weight(a,b)*I2(i-a+ceil(size(weight,1)/2),
             j-b+ceil(size(weight,2)/2)));

            end
        end
        I(i,j)=convol;
    end
end
```

5.3 Greedy Snake Implementation

Here is the code for our implementation of the greedy snake method. It might not be the best and most accurate one, it seems to be working but some improvement are required to maybe make it more accurate.

```
 alpha=4.0;beta=3.0;
   while(ite < 12)
    for k=1:length(x)

        % make closed contour
        xnext = circshift(x, 1);
        ynext = circshift(y, 1);
        xprev = circshift(x, -1);
        yprev = circshift(y, -1);

        E_ext_loc=(E_ext(y:y+2*floor(size(weight,1)/2),x:x+2*floor(size(weight,1)/2)));

        % defines neigbors for contour gradient computation
        Nx = ones(1, size(weight,1))' * [-floor(size(weight,1)/2):1:floor(size(weight,1)/2)]
         + x(k);
        Ny = [-floor(size(weight,2)/2):1:floor(size(weight,1)/2)]' * ones(1, size(weight,1))
         + y(k);

        % compute gradient
        Gx = Nx - xprev(k);
        Gy = Ny - yprev(k);
        normGrad = sqrt(Gx.^2+Gy.^2);

        % define second order neighbors for curvature computation
        Nxx = ones(1, size(weight,1))' * [-floor(size(weight,1)/2):1:floor(size(weight,1)/2)]
         + x(k);
```

```
     Nyy = [-floor(size(weight,1)/2):1:floor(size(weight,1)/2)]' * ones(1, size(weight,1))
      + y(k);

     % compute second derivative
     Gxx = xprev(k) - 2 * Nxx + xnext(k);
     Gyy = yprev(k) - 2 * Nyy + ynext(k);
     normSecDer = sqrt(Gxx.^2+Gyy.^2);

     En = alpha.*normGrad + beta.*normSecDer +3.0*(E_ext_loc);
     minE= min(min(En));
     [r, c] = find(En == min(min(En)));
     k;
     E_ext_loc(r,c)
     %if E_ext_loc(r,c) <= E_pt(k)
         x(k) = x(k) + (c(1) - (size(weight,1)+1)/2);
         y(k) = y(k) + (r(1) - (size(weight,1)+1)/2);
         E_pt(k)=E_ext_loc(r,c);
    %end
         clf
         imagesc(I)
         hold on
          %plot(x,y,'r+','LineWidth',2)
          plot(x,y,'g-','LineWidth',2)
          pause(0.1)

        % sum(E)


 end
 ite=ite+1;
end

      plot(x,y,'r+','LineWidth',2)
     plot(x,y,'g-','LineWidth',2)
```

5.4 Euler Lagrange Snake Implementation

As presented before, Euler Lagrange is a method to solve the energy minimization problem at stake in this project.

```
%Second Order discrete derivation Matrix D_2
V=-2*ones(X,1);
W=1*ones(X-1,1);

D_2=diag(V)+diag(W,1)+diag(W,-1);
D_2(1,X)=1;
D_2(X,1)=1;

% Fourth Order discrete derivation Matrix D_4
V=6*ones(X,1);
W=-4*ones(X-1,1);
U=ones(X-2,1);

D_4=diag(V)+diag(W,1)+diag(W,-1)+diag(U,2)+diag(U,-2);
D_4(1,X)=-4;
D_4(X,1)=-4;
D_4(1,X-1)=1;
D_4(X-1,1)=1;
D_4(2,X)=1;
D_4(X,2)=1;

%Compute External Energy based on the gradient of the image gradient's norm
[TempX,TempY]=gradient(I);
nn=(TempX.^2+TempY.^2).^(1/2); %Compute norm
[Px,Py]=gradient(nn);

% Build matrix A
dt=0.05; %Time step for temporal integration
alpha=0.4;
beta=1.0;
delta=500;
A=eye(X)+dt*(-alpha*D_2+beta*D_4);
A_inv=inv(A);

new_Vx=Vx;
new_Vy=Vy;

i=0;
% stopping criterion based on iterations
while(i~=650)
```

```
    Inter_Px=interp2(Px,new_Vx,new_Vy); %contour interpolation
    Inter_Py=interp2(Py,new_Vx,new_Vy);
    old_Vx=new_Vx;
    old_Vy=new_Vy;
    new_Vx=A_inv*(old_Vx+dt*delta*Inter_Px);
    new_Vy=A_inv*(old_Vy+dt*delta*Inter_Py);
    i=i+1;
    NewI=[new_Vx,new_Vy];
    if(mod(i,10)==0) % speed up display
        clf
        imagesc(I2);
        hold on
        plot(new_Vx,new_Vy,'c','LineWidth',2);
        pause(0.3);
    end
end
```

6 Conclusion

In this last project, we saw that we could use different implementations to perform a segmentation using active contours. These methods rely on various Image Processing techniques we already studied and present in previous projects.

This segmentation technique is really interesting and quite intuitive but it also has some drawbacks. Indeed, has we mentioned briefly in the presentation it really depends on the user initialization and on the stopping criterion. Because we could initialize too far and do not converge correctly or if a part of the contour goes inside the object our model will not be able to deform the curve to make it dilate. Another aspect is object proximity which could influence the accuracy of the detection. Indeed if two objects are too close, it's highly possible that the snake could, according to the user initialization, be attracted by the boundaries of the other if they are stronger than the object's we want to segment. This leads to discuss the background of the image, if there is noise or a non homogeneous background these are factors which will influence the snake convergence. This is why the pre processing step was really important. Binarization allows us to get rid of certain surrounding objects that are closed to our object if their intensity levels are compatible for thresholding. Once we have a binary image, if the objects are too close we can still perform erosion of surrounding objects to reduce their size.

We showed that our implementation of the greedy method was really poor, because we focused on the Euler Lagrange method which is more accurate and allow to perform an overall energy minimization while the greedy method perform local minimization and hopes to converge to global minimum which is rarely the case, and probably maybe because our implementation is not good and correct enough.

Some more advanced method exists to improve the issues regarding the poor convergence issues and initialization issues and particularly regarding concavity. One of them is using another external force in the computation of the external energy. This force is called the Gradient Vector Flow and is a diffusion of the gradient of the image. This allows to create a larger area of attraction for the snake and solve the concave issue. This has been developed by Chenyang Xu, and Jerry L. Prince and it allows to have almost random initialization because the attraction field is stronger while in our case if the initialization is too far from the object our algorithm is failing.

This project was a good occasion to review a fundamental technique of image processing to perform segmentation. We review the basic methods to perform segmentation but we also analyse the weakness of the methods we used and looked into the literature to see how other people made these methods precise and accurate. Applications of this active contour techniques are very important in image processing and in many video analysis for shape segmentation over time to study their evolution over time in a video sequence.

References

[1] Wikipedia contributors. Wikipedia, The Free Encyclopaedia, 2012. Available at: http://en.wikipedia.org, Accessed October-November, 2012.

[2] M. Kass,A.Witkin, D.Terzopoulos, Snakes: Active Contour Models, International Journal of Computer Vision, 1988.

[3] Chenyang Xu, and Jerry L. Prince, Gradient Vector Flow: A New External Force for Snakes, IEEE Proc. Conf. on Comp. Vis. Patt. Recog. (CVPR'97)

[4] R. C. Gonzalez, R. E. Woods, Digital Image Processing, Third Edition, Pearson Prentice Hall, 2008.

[5] D. Rohmer, Contours deformables: Snakes, Application a la segmentation, Cours MSO1-Im CPE Lyon, 2010.

List of Figures