Combining Active Appearance Models and Cage Active Contours for image segmentation. Application to brain MRI images.
Abstract

We present a novel method for image segmentation based on the combination of Cage Active Contours (CAC) and the well-known Active Appearance Models (AAM) methods. The idea is to turn Active Appearance Model into a fully automatic method that does not depend on a dataset labeled with landmarks.

Thus, the first one is used for extracting the shape of the object to segment from a set of binary masks containing the contour. The particularity is that this shape is parameterized by a set of control points forming what is called a cage using mean value coordinates. The second one is used for learning a generative model that encodes variations of cages as well as texture.

The method is applied on MRI images of the brain. In particular, a dataset from MICCAI 2012 Grand Challenge on Multi-Atlas Labeling is used. We compare the results of our method against three other methods: FreeSurfer, SRC and DDLS. In addition, we placed landmarks on a subset of the data to make a comparison between AAM with landmarks versus AAM with cages in order to find out what is the cost of using cages in terms of performance.
1 Introduction

Image segmentation [53] is defined as the partitioning of an image into non-overlapping regions made up of pixels that are homogeneous with respect to some characteristic such as intensity, texture or anatomical regions in case of medical images. Instead of considering the whole data presented in an image all at once, it is better to focus on a certain region-based semantic object in image segmentation. These groups of pixels are usually connected and separated by linear or surface boundaries, 2-D and 3-D cases respectively. Thus, the goal of image segmentation is to find the regions that represent meaningful parts of objects for easier analysis purposes.

Medical image segmentation play a vital role in numerous biomedical-imaging applications such as the quantification of tissue volumes, diagnosis, localization of pathology, study of anatomical structure, treatment planning and computer-integrated surgery, abnormality detection etc. Magnetic resonance imaging (MRI), computed tomography (CT), digital mammography and other imaging modalities provide an effective means for non-invasively mapping the anatomy of a subject. However, these images suffer from noise, poor image contrast, inhomogeneity and weak boundaries that turn medical image segmentation into a very complex and challenging problem.

Among all structures dealt with by the research community, brain image segmentation is still one of the most important ones. The identification of brain structures in magnetic resonance imaging (MRI) is very important in neuroscience and has many applications such as mapping of functional activation onto brain anatomy, the study of brain development and the analysis of variability in normal brains. Further, it is useful in clinical diagnosis of neurodegenerative problems, psychiatric disorders, treatment evaluation and surgical planning [26].

Several common approaches have been appeared in the literature on medical image segmentation, each of them exploiting different feature of the image and using a distinct concept of how to define and locate the boundaries. Thresholding-based were applied in [7, 50, 25] where the segmentation is achieved by grouping all pixels with intensities greater than the threshold into one class and all other pixels into another class; region-based approaches [16, 59, 44] extract an image region that is connected based on some criteria such as intensity information and/or edges [64] in the image; clustering-based approaches based on different clustering algorithms such as Fuzzy C-Mean [8, 37], K-Means [36], Hierarchical Clustering [19] and Mean-Shift [13, 12]; level-set based approach [43, 39, 54] deforms a contour towards a previously learned shape model; artificial neural network based...
approach [51, 31] used both as a classifier and an unsupervised clustering method; and deformable models based-approach [20, 47, 61, 3, 49] which model-based techniques for delineating region boundaries by using closed parametric curves or surfaces that deform under the influence of internal and external forces and more recently, patch-based dictionary learning techniques have been also used [60].

A popular enhancement in medical image segmentation methods consists in incorporating \textit{a priori} knowledge about the structures to be segmented. It is usually represented as deformable models in the form of initial conditions, constraints on the shape parameters or into the model fitting procedure. This are the cases of Active Shape Models [17] and Active Appearance Models [15, 14], both generative models that encode shape and appearance representations respectively of the selected objects.

The present work aims to develop a method that combines \textit{a priori} knowledge given by deformable models and versatility, robustness and efficiency of level-sets approaches by combining Active Appearance Models (AAM) and Cage Active Contours (CAC) techniques respectively. The first method is used for learning a full appearance model of basal ganglia structure of the brain; the second one is used as a shape extractor for obtaining a parametrized description of the shape that allows us to construct a deformable model without having to manually indentify landmarks on the images while keeping the ability to establish a point-to-point correspondence to learn from.
2 Related work

In this section, we explain fundamental background required to fully understand our work.

2.1 Deformable models

Deformable models [46] are curves or surfaces defined within an image domain that can move under the influence of internal forces, which are defined within the curve or surface itself, and external forces, which are computed from the image data. The internal forces are designed to keep the model smooth during deformation. The external forces are defined to move the model toward an object boundary or other desired features within an image. By constraining extracted boundaries to be smooth and incorporating other prior information about the object shape, deformable models offer robustness to both image noise and boundary gaps and allow integrating boundary elements into a coherent and consistent mathematical description.

In analogy with Physics, these forces are usually described in terms of associated energies that have to be minimized in order to fit the boundary. This way the mathematical formulation of the problem is simplified and general functional iterative optimization methods like gradient descent can be used. For example, using a basic gradient descent method, the evolution of the boundary is determined by:

\[ r_{t+1} = r_t - \alpha \nabla E(r_t) \]  

(2.1)

where \( \alpha \) is the evolution rate parameter, \( r \) is a vector describing the contour and \( E(r_t) \) is the energy functional to be minimized. The energy functional can be decomposed as

\[ E(r_t) = E_{int}(r_t) + E_{ext}(r_t) + E_c(r_t) \]  

(2.2)

where \( E_{int} \), \( E_{ext} \) and \( E_c \) correspond to the internal, external and constraint energies, discussed in the next section.

A good energy functional should achieve the following features:

- Have few local minima. Otherwise, since the minimization algorithms are iterative, they are prone to get stuck easily.

- Have little dependence on starting points, i.e, the initialization of the contour. Otherwise, if a small variation would lead to very different results, the method becomes chaotic and unstable.
• Continuous and twice differentiable. Some iterative optimization methods use the second derivative to estimate the size and direction of the step.

2.2 Active contours

Active contours [35] model, also known as "snakes" because of the way the contour moves during the evolution, are one of the most popular deformable models. A snake is an energy-minimizing contour guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges localizing them accurately. In the snakes framework, the contour curve can be described as:

• Point-based snakes. The curve is an ordered collection of discrete points, called snaxels.

• Parametric snakes. The curve is defined in a continuous parametric form of the form \( r(s) = (r^x(s), r^y(s)) \), using basis functions like b-splines or Fourier exponentials defined at certain knots in the curve.

Point-based snakes can be regarded as an extreme case of parametric snakes, where the curves are defined in terms of b-splines of degree zero with 2 parameters (position x and y) for each curve point. Point-based snakes thus generally require more parameters than parametric snakes, resulting on slower optimization algorithms. Moreover, parametric snakes can incorporate the smoothness and other a priori constraints in the basis functions used, eliminating the necessity of extra energy terms.

Figure 2.1: Example of a deformation of a snake shown in [32]. The left picture corresponds with the initialization; the right picture shows the final shape of the snake after having minimized the energy.
2.2.1 Energies for Active Contours

Internal energies

As have been already mentioned, internal energies try to ensure the smoothness of the contour. The most common internal energy is the one introduced by Kass [35], which is a linear combination of the length of the curvature and its curvature, defined as

\[ E_{\text{int}} = \int_C \lambda_1(s) \| \mathbf{r}'(s) \|^2 ds + \int_C \lambda_2(s) \| \mathbf{r}''(s) \|^2 ds \] (2.3)

By adjusting the weights \( \lambda_i \) of each term, the relative importance of having shorter or smoother contours at each section of the curve can be tuned. The parameter \( \alpha_1 \) controls the "tension", while \( \lambda_2 \) controls the "rigidity".

As pointed out in [32], the second term only guarantees smoothness if the curve is parametrized by its curve length, i.e., if the parameter \( s \) is the curvilinear abscissa and the arc length between two knots is constant. To favor this condition in parametric snakes, [32] includes an additional term called "re-parametrization energy" in the internal energy. Otherwise, the knot points can accumulate in a certain section of the curve and generate pointy curves (see Figure 2.1)

External energies

External energies, also known as image energies are the key element of the snakes framework to drive the contour curve to the desired area of interest. We can divide these energies into edge-based and region-based energies. The first ones compute a property of the curve considering only the pixels on the evolving contour or close to it, while the second ones take into account pixels belonging to the inside or the outside of the curve, if it is closed.

Edge-based energies

The most popular edge energy used is the one based on the gradient of the image intensity:

\[ E_{\text{edge}} = -\int_C \| \nabla I(\mathbf{r}(s)) \|^2 ds \] (2.4)

where \( C \) denotes de contour. Minus sign of the previous equation tells to the curve that it should evolve to lay over areas with a large intensity gradient, which are expected to be region boundaries. A more informed approach can try to maximize the component of the intensity gradient perpendicular to
the curve as:
\[ E_{\text{edge}} = - \int_C \nabla I(r(s)) \cdot \hat{n}(r(s)) ds \] (2.5)
where \( \hat{n}(r(s)) \) is the unit vector perpendicular to the curve at point \( r(s) \).

One of the potential problems of using gradient-based edge energies is that the presence of noise could make the contour curve sharp and pointy. However, as pointed out in [35], this problem is often avoided by the presence of internal smoothing energies, that act as energy wells to good local minima. Also, [35] proposes the use of a spatially blurred energy functional to evolve the snake first at a coarse scale and refine later, avoiding possibly misleading local minima.

**Region-based energies** Region-based energies use statistical information about the pixels inside and outside the segmentation regions. Since many of the regions to segment involve closed contours, this kind of energies are becoming increasingly popular [10, 9, 46]. For instance, consider a white object placed on a black background. If we initialize the contour far away from the object real boundary, the gradient energy will probably get stuck in a plateau, but an energy that gets minimized when all pixels inside the contour are white and all pixels outside it are black can correctly drive the evolving contour towards the boundaries of the white object.

### 2.3 Cage Active Contours

**Cage Active Contours (CAC)** is a segmentation framework that combines computer graphic deformation techniques and parametrized active contours [63, 6]. It introduces the idea of parameterizing a contour according to a reduced set of control points, known as *cage points*, which drive the evolution of the contour. Note that we use the term cage to refer to the polygon that allows to deform the evolving contour. This parameterization uses *mean value coordinates*, a generalization of barycentric coordinates for arbitrary planar polygons [29].

**Mean value coordinates** were initially proposed for mesh parametrization problems [23]. The author demonstrated that the interpolant generated smooth coordinates for star-shaped polygons for any point \( p \) on the plane.

The mean value coordinates of a point \( p \), \( \varphi_i(p) \), given a set of vertices \( v_i \) of a polygon of \( N \) points, \( j = 1 \ldots N \), are computed as
\[ \varphi_i(p) = \frac{w_i}{\sum_{j=1}^{N} w_j} \quad i = 1 \ldots N, \] (2.6)
and $w_i$ is computed as

$$w_i = \frac{\tan(\alpha_i/2) + \tan(\alpha_i/2)}{||v_i - p||}$$  \hspace{1cm} (2.7)

where $||v_i - p||$ is the distance between the vertex $v_i$ and the considered point $p$ and $\alpha_i$ is the signed angle of $[v_i, p, v_{i+1}]$, see Figure 2.2.

![Figure 2.2: Example of mean value coordinates computation. The i-th component of the coordinate of $p$ is computed with respect to the angles $\alpha_i$ and $\alpha_{i-1}$.](image)

Given the affine coordinates $\varphi_i(p)$ of a point $p$, the point $p$ can be recovered with

$$p = \sum_{i=1}^{N} \varphi_i(p)v_i$$  \hspace{1cm} (2.8)

If the vertices of $v_i$ of the polygon move to positions $v_i'$, the "deformed" point $p'$ can be recovered as

$$p' = \sum_{i=1}^{N} \varphi_i(p)v'_i$$  \hspace{1cm} (2.9)

where note that the point $p'$ is recovered from the affine coordinates $\varphi_i(p)$, see Fig. 2.3.

Given a set of points $\{p\}$, the affine coordinates for each point are computed in an independent way using (2.6). If a point $v_i$ is stretched in a particular direction, all the points $p$ follow the same direction with an associated weight given by $\varphi(p)$, see Fig. 2.3. The points $p$ that are near the moved vertex have higher weight, see denominator of (2.6), and thus suffer a larger "deformation" than the points which are farther where the weight is smaller and hence, they are barely affected by the deformation.
Figure 2.3: Example of the deformation of a region by means of a polygon. From right to left: the polygon vertex \( v_i \) is moved producing the consequent deformation of the region after applying the interpolation function.

The image segmentation problem is formulated as the minimization of a region-based energy \( E \) performed by means of a gradient descent process that iteratively updates the cage vertex positions \( v \), initially disposed in a way that encloses the evolving contour \( C \). These points are denominated *cage points*, what motivates the name of the framework. During the segmentation optimization, the control points are the only ones allowed to move independently, being the contour points \( \Omega_c \) displacement governed by the evolution of these control points. Note that mean value coordinates \( \varphi \) of \( \Omega_c \) with respect to cage control points \( v \) are calculated only once at the beginning of the optimization using Eq. 2.6. Every time a cage control point moves, the new position of \( \Omega_c \) can be computed (interpolated) using Eq. 2.9.

Actually, not only the contour points are controlled by the cage points, but also all of the region of pixels inside and a subset of pixels outside contour \( C \), \( \Omega_{in} \) and \( \Omega_{out} \). The subset \( \Omega_{out} \) constitute a thin band near the contour (see Figure 2.4) and is computed by applying a dilation operation as:

\[
\Omega_{out} = \text{Dilation}(\Omega_{in}, d_{out}) - \Omega_{in}
\]  

where \( d_{out} \) is the dilation width (see Figure ). This way we can avoid considering image information far from the active contour that is not important for the optimization process and can even affect it negatively.

There are many possibilities in order to define the energy \( E \) to be minimized [10, 55]. We assume that there are two segmentation regions, \( R_{in} \) and \( R_{out} \) which correspond with the image region belonging to the object of interest and the rest of the image, both having different statistical properties. Therefore, given the statistical properties of both areas, we want to maximize the likelihood of the pixels determined to be in a certain region to really belong to that region. If we use log-likelihood and image intensity
Figure 2.4: Example of a dilation in a binary mask. \( \Omega_{in} \) is the region in white color and \( \Omega_{out} \) is equals to the green plus white regions. Green region illustrates the band near the contour \( d_{out} \).

the problem can be formulated as minimizing the functional

\[
E_{\text{region}} = -\frac{1}{|\Omega_{in}|} \sum_{p \in \Omega_{in}} \log(P(I(p)|p \in R_{in})) - \frac{1}{|\Omega_{out}|} \sum_{p \in \Omega_{out}} \log(P(I(p)|p \in R_{out}))
\]

(2.11)

where \( R_{in} \) and \( R_{out} \) denote the different image regions, \( \Omega_{in} \) and \( \Omega_{out} \) denote the points that lay inside and outside the closed contour curve at that iteration, and \( I(p) \) is the gray-level intensity of a pixel \( p \). The normalizing terms \( |\Omega_{in}| \) and \( |\Omega_{out}| \) denote the number of pixels inside and outside the region enclosed by the evolving contour, respectively. Given an appropriate conditional probability functions \( P(I(p)|p \in R_{in/out}) \), this energy attains a minimum when \( \Omega_{in} = R_{in} \) and \( \Omega_{out} = R_{out} \).

One of the simplest models is the mean energy proposed by [10], which is based on the intensity distance to the mean and it is expressed as:

\[
E_{\text{mean}} = \frac{1}{|\Omega_{in}|} \sum_{p \in \Omega_{in}} (I(p) - \mu_{in})^2 + \frac{1}{|\Omega_{out}|} \sum_{p \in \Omega_{out}} (I(p) - \mu_{out})^2
\]

(2.12)

where \( \mu_{in} \) and \( \mu_{out} \) corresponds with the mean of intensity in \( \Omega_{in} \) and \( \Omega_{out} \) respectively. This energy assumes that the gray-level of pixels inside \( \Omega_{in} \) and \( \Omega_{out} \) can be modeled with a single value which in turn corresponds with
the mean. The energy will reach the minimum when all the pixel intensities in $\Omega_{in}$ are close to the mean intensity $\mu_{in}$ and all the pixel intensities in $\Omega_{out}$ are close to the mean intensity $\mu_{out}$. In case of segmenting a binary image, $\mu_{in}$ and $\mu_{out}$ always take values 1 and 0 respectively because pixels value only take either 0 or 1, turning mean energy model suitable for segmenting binary masks.

Gaussian energy is another model that can be used. If we assume that the pixel intensities of both regions follow a Gaussian distribution [12, 16], the conditional probability can be expressed as

$$P(I(p)|p \in \Omega_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(I(p) - \mu_j)^2}{2\sigma_j^2}\right)$$

(2.13)

where $\mu_j$ and $\sigma_j$ are the mean and the variance of the pixel intensities over the region $\Omega_j$ and $j \in \{in, out\}$. Using this conditional probability function, the region energy takes the form

$$E_{gauss} = \log\sigma_{in} + \frac{1}{|\Omega_{in}|} \sum_{p \in \Omega_{in}} \frac{(I(p) - \mu_{in})^2}{\sigma_{in}^2} + \log\sigma_{out} + \frac{1}{|\Omega_{out}|} \sum_{p \in \Omega_{out}} \frac{(I(p) - \mu_{out})^2}{\sigma_{out}^2}$$

(2.14)

where the constant terms have been dropped since they have no influence in the energy minimization.

### 2.4 Active Appearance Models

Active Appearance Models (AAM) [14, 15] is a generative algorithm coming from computer vision field that combines statistical model of shape variation with a model of the appearance (gray-level intensity) variations of the object of interest. Unlike Active Shape Models [17], the model can not only accommodate to the shape of new valid image examples but can also synthesize them using a gray-level model. Given a learned model of an specific object and a new example to be interpreted, the goal is to approximate it by iteratively generating new examples that are close to the one to be interpreted using the model. The generation of new examples is constrained by the model, making that only valid examples are generated, that is, examples that are consistent with the learned model.

We are given a training set of labelled images $\{I\}$ and $|\{I\}| = K$, where key landmark points are marked on each example object outlining the main
features and shape. The labelled points on a single object describe the shape of that object. More formally, the geometry of the shape of a particular object is expressed as \( k \) aligned landmarks, that in two dimensions are expressed as

\[
X_{\text{landmarks}} = (x_1, y_1, \ldots, x_k, y_k)
\]  

(2.15)

In Figure 2.5 can be seen an example of landmarks placed in a resistor. It’s important to note that each landmark \( i \in \{1, \ldots, k\} \) should represent the same physical location in order to establish a correct point-to-point correspondence between different instances of the same object. So, if landmarks \#0 and \#31 from Figure 2.5 refer to the left boundary of a resistor, all \#0 and \#31 landmarks from all the dataset must refer to that part.

Figure 2.5: Thirty-two landmarks surrounding the boundary of a resistor describing the geometry of the shape. Image extracted from [17].

Firstly, in order to build statistical model of shape variation, landmark points are aligned, usually applying Procrustes Analysis [21]. Later, a PDM (Point Distribution Model) [18] is computed by applying eigen-analysis such as Principal Component Analysis [33] over \( X_{\text{landmarks}} \), assuming that the distribution of landmarks is gaussian. Then, any new shape \( x'_{\text{landmark}} \) is generated as a linear combination of eigenvectors as

\[
x'_{\text{landmark}} = \bar{x}_{\text{landmark}} + P_s b_s
\]  

(2.16)

where \( \bar{x}_{\text{landmark}} \) is defined as the mean shape across all training images, \( P_s \) is a set of orthogonal modes of variation and \( b_s \) is a set of shape parameters that can be seen as scaling values for each principal component. In fact, we are building a statistical model of the variability of \( X_{\text{landmark}} \) with respect to the mean. By modifying the variable \( b_s \), an infinite number of shapes can be defined. In figure 2.6 can be seen the plot corresponding to the distribution of landmarks of the resistor shown in 2.5.

To build a statistical model of the grey-level appearance, each example image is warped so that its control points match the mean shape (using a
Figure 2.6: Point Distribution Model corresponding to landmarks of the resistor from 2.5. Note the cloud of points created. Each of them is the scatter diagram of the distribution of the $i$ landmark position.

triangulation algorithm). Then, a sampling of the grey-level information $g_{im}$ is performed from the shape-normalised image over the region covered by the mean shape. The gray-level appearance of any shape-normalized training image can be then described by a vector $g$. Applying PCA to the $g$ vector of all images we obtain the linear model:

$$g = \bar{g} + P_g b_g$$

(2.17)

where $\bar{g}$ is the mean normalised grey-level vector, $P_g$ is a set of orthogonal modes of variation and $b_g$ is a set of grey-level parameters.

The shape and appearance of any example can thus be summarised by the vectors $b_s$ and $b_g$. Since there may be correlations between the shape and grey-level variations, a further PCA is applied as follows. For each example, a concatenated vector is generated as

$$b = \left( W_s b_s \middle| b_g \right) = \left( W_s P_s^T (x - \bar{x}) \middle| P_g^T (g - \bar{g}) \right)$$

(2.18)

where $W_s$ is a diagonal matrix of weights for each shape parameters, allowing for the difference in units between the shape and grey models (see below). PCA is applied on these vectors giving the further model

$$b = Qc$$

(2.19)

where $Q$ is the matrix of eigenvectors and $c$ is a vector of appearance parameters controlling both the shape and grey-levels of the model:

$$x = \bar{x} + P_s W_s^{-1} Q_s c$$

(2.20)

$$g = \bar{g} + P_g Q_g c$$

(2.21)

For a given $c$, an example image can be synthesized by generating the shape-free grey-level image from the vector $g$ using Eq. (2.21) and warping it using the control points described by $x$ using Eq. 2.20.
2.5 Objectives

The present method attempts to combine Cage Active Contours (CAC) and Active Appearance Models (AAM) into a single method that benefits from the strong points of both algorithms. From one hand, CAC is a good technique for fitting a parametrized curve to a region of interest of an image where this parametrization is done using a set of control points. On the other hand, AAM is a good approach for learning a model of any desired object, that is, its possible shape and texture information, for later on using it to search similar instances of the same object on an unseen images in order to solve object detection, recognition and segmentation tasks.

However, AAM strongly relies on a set of manually placed points that describe the shape of a particular object, also known as ”landmarks”. Therefore, by representing the shape as a parametrization of a set of control points, we avoid the placement of landmarks and extend the range of applicability of AAM method. For instance, in the case of medical domain, such landmarks must be placed by an expert that is not always available.

In addition, we are giving prior knowledge of the object to segment to the ”snake” achieving two things: we are giving a better starting initialization since the initial curve will have a shape close to ones of the dataset and we are limiting the acceptable shape and texture that the snake can take to the ones seen in the dataset. That will avoid snake turning into an unfeasible appearance, understood as a unrealistic shape and texture.
3 Cage Active Appearance Models

This section will be devoted to explain the details of the proposed method. Some familiarity with Active Appearance Models and Cage Active Contours is required in order to correctly understand the concepts, see Section 2 for details.

Recall the main problem that we are addressing: given a dataset of images \( \{I\} \) and \( |\{I\}| = K \), the goal is to build a system for automatic labeling (segmentation) by means of building a model of cage and appearance variations of a certain object of interest using the images in the given dataset and fitting it to new image instances containing the same object. However, the difference now is that instead of having a set of landmarks for each image, we are given a set of binary masks \( \{M\} \) and \( |\{M\}| = K \), where \( M_i \in \{M\} \) corresponds with the binary mask of the object of interest appearing in \( I_i \in \{I\} \). The implication of this fact will be discussed in 3.1 more in detail.

In the following sections will be explained the details of learning the shape and texture models.

3.1 Modeling shape

We have previously seen in section 2.4 how the classic approach for modeling the shape uses a set of landmarks in order to represent the shape of an object. Landmarking is a good technique for representing shapes between different object instances of the same class since they define common points between them. However, landmarking has several drawbacks:

- It is very time consuming since landmarks have to be defined manually image by image. In cases where datasets have large amounts of images, landmarking becomes unfeasible unless time is not an issue.

- In some domains such as medical images it is very difficult to correctly place landmarks without the help of an expert because some prior knowledge of the shape is needed.

Above reasons may turn manually landmarking into a barrier in some domains when trying to apply image segmentation techniques that rely on it. Our proposed framework addresses this problem in the following manner: instead of modeling the shape as a set of landmarks points, we will model the set of control points from the cage that govern a curve; in other words, we build a Point Distribution Model of cages. If this curve is fitted to an specific contour of an object by moving the points of the cage, then we have
the shape represented as the control points defining the curve. Therefore, we use the cages as a shape descriptors so that its control points can be seen as a kind of "landmarks" because they indirectly describe the geometry of the shape. Note that this curve fitting procedure is automatically carried out by applying Cage Active Contours algorithm (see Section 2.3 on the set of binary masks \( \{ M \} \)).

The following subsections are devoted to explain more in detail the steps performed in order to extract the cages for each binary mask in \( \{ M \} \) and how the model of cage variations is learned from this set of cages.

### 3.1.1 Extracting cages

We are given a set of binary images \( \{ M \} \) containing the mask of the object to segment. We assume that images are already aligned so that they lie in the same coordinate system by means of image registration techniques [42, 65](see 4.1). Each binary mask implicitly defines the shape \( s \) that corresponds with the Cartesian points (x,y) falling in the boundary between the black and white region. Recall that our goal is to fit a curve \( C \) to match, as close as possible, the shape \( s \) of each binary mask in the dataset. Thus, we perform the following steps:

1. **Initialize \( C \)**
   
   The first step is to give an initialization for the evolving curve \( C \). A majority voting over \( \{ M \} \) is performed and a threshold is applied. Thus, we

![Figure 3.1: Block diagram of the extraction of cages from the binary mask dataset.](image)

1. **Initialize \( C \)**
   
   The first step is to give an initialization for the evolving curve \( C \). A majority voting over \( \{ M \} \) is performed and a threshold is applied. Thus, we
sum all binary masks and divide by the total number of masks in order to get a probability of a pixel to be part of a mask. Later, we binarize the image according to a threshold and we get $M_0$. From it, we extract the points belonging to the contour as the initialization for $C$ defined as $C_0$. Mathematically we have

$$C_0 = \text{binarize} \left( \frac{1}{K} \sum_i M_i \right) \tag{3.1}$$

where $K$ is the number of binary masks in $\{M\}$ and

$$\text{binarize}(X) = \begin{cases} X_{ij} = 0 & \text{if } X_{ij} < \text{threshold} \\ X_{ij} = 1 & \text{if } X_{ij} \geq \text{threshold} \end{cases}$$

where $X_{ij}$ is the pixel located at $(i,j)$ position. The value of the threshold is experimentally tuned in a cross-validation protocol as described in section 5.4.

2. Enclose $C_0$ with a cage

An initial cage $v^0$ must be defined, that is, setting the initial position of each vertex (control point) of the cage that will enclose the initial curve initial curve $C_0$. A finite number of vertex $N$ must be defined. Despite there is no restriction about the shape that a cage has to take, the common choice is to use an ellipse-shaped or rectangular-shaped one. In particular, for this work, we used ellipse-shaped cages and therefore, in order to get $v^0$ we compute the ellipse that encloses $C_0$ where the ellipse is restricted to have $N$ number of vertex using the classic parametric equation of an ellipse.

3. Compute affine coordinates

We have just defined initial curve $C_0$ and its respective initial cage $v^0$. We now compute mean value coordinates of each point $p \in C_0$ with respect to the initial cage $v^0$. Let us define $F_{C_0}$ as the matrix containing mean value coordinates of $C_0$. This way, we are parameterizing the points of curve $C_0$ with a set of control points from $v^0$. The computation is done according to section 2.3.

4. Automatic segmentation using CAC

With curve $C_0$, initial cage $v^0$ and affine coordinates $F_{C_0}$ that parametrize $C_0$ according to $v^0$, we apply CAC method using the mean energy $E_{\text{mean}}$ (Eq.
2.12) in order to segment each of the binary masks from the set \( \{ M \} \) with the goal to learn the possible configurations that \( v \) can take. Note that the same \( C_0 \) and \( v^0 \) is used for all \( \{ M \} \), meaning that initial curve and cage are dependent on \( \{ M \} \) and thus part of the model.

Recall that the idea behind CAC is to move control points of \( v^0 \) whose in turn govern the movement of \( C_0 \) so that the region of pixels being inside and outside of the curve, \( \Omega_{in} \) and \( \Omega_{out} \) respectively, are as much homogeneous as possible in terms of pixel intensity value. This movement is driven by minimizing the selected energy \( E \) by means of gradient descend optimization with respect to control points of \( v^0 \). A visual example can be found in Figure 3.2.

In particular, we employed the region-based mean energy \( E_{\text{mean}} \) expressed in Eq. 2.12. The reason is because binary masks can only take two possible values for the intensity, black or white, so \( E_{\text{mean}} \) is enough complex to lead \( C \) towards a good enough fit. Thus, a \( C \) good enough fit would end up having \( \Omega_{in} \) equals to an homogeneous white pixel region and \( \Omega_{out} \) an homogeneous black pixel region. The outcome we receive is the cage \( v^i \) that minimizes \( E_{\text{mean}} \) on binary mask \( M_i \). In other words, \( v^i \) is the best cage possible that describes the real contour/shape of \( M_i \).

At the end of this step, we will obtain a set of cages \( \{ v \} \) where \( |\{ v \}| = K \) and \( v^i \) corresponds with the cage of \( M_i \).

### 3.1.2 Building the cage model

In the previous section, we have seen how to compute the set of cages \( \{ v \} \) that best fit the shape described by the set of binary masks \( \{ M \} \). Given the cage \( v^i \) that describes the shape of the binary mask \( M_i \), we flat its control points as concatenating them into a vector so that

\[
f_i = (v^i_1, v^i_2, \ldots, v^i_N)^T \quad i = 1 \ldots K,
\]

where \( v^i_j \) corresponds with \((x, y)\) Cartesian coordinates from the \( j \)th control point of \( i \)th cage. We further construct shape matrix \( X_{\text{cage}} \) as the concatenation of \( f_i \):

\[
X_{\text{cage}} = (f_1, f_2, \ldots, f_K)^T
\]

In order to build the linear model of shape variations of cages, we apply Principal Component Analysis on \( X_{\text{cage}} \). From the analysis, we extract matrix \( P_v \) consisting in a set of orthonormal base vectors (also known as principal component), their associated eigenvalues \( \lambda_i \) and the mean of the control points of cages \( \bar{v} \). Note that \( \bar{v} \) corresponds with the mean shape of
Figure 3.2: Example of CAC fitting the contour of a binary mask corresponding with a slice of caudate. The evolving curve is marked in red while the cage is draw in red. Observe how the green curve moves towards contour at each iteration.

the cages and in turn, produces the mean shape $\bar{s}$ of the modeled object that can be used to generate the mean mask $\bar{M}$. We keep $\tau$ principal components vectors, which are enough to explain $e_{\text{var}}$ amount of variance. Usually this amount of variance is set to be $0.95 \leq e_{\text{var}} \leq 1$. Any cage $\mathbf{v}'$ can then be approximated (generated) by the model using the following linear combination expression:

$$\mathbf{v}' = \bar{\mathbf{v}} + P_{\mathbf{v}} \mathbf{b}_{\mathbf{v}}$$

(3.4)

where $\bar{\mathbf{v}}$ is the mean cage, $P_{\mathbf{v}}$ is a set of orthogonal modes of variation and $\mathbf{b}_{\mathbf{v}}$ is a vector of cage parameters. Therefore, by giving values to cage parameters $\mathbf{b}_{\mathbf{v}}$, the model can generate infinite number of valid examples of cages $\mathbf{v}'$. In order to recover the original shape $s_i$, that is, the contour of the binary mask $M_i$, we perform:

$$s_i = F_{C_0} \mathbf{v}^i$$

(3.5)

where $s_i$ is a set of $(x,y)$ points defining the boundary of the binary mask $M_i$, $F_{C_0}$ is a matrix where its rows contain affine coordinates $\varphi$ of each point $p_i \in C_0$ with respect to $\mathbf{v}^0$ and $\mathbf{v}^i$ is a cage formed by a set of $N$ control points. The previous equation can be interpreted as deforming the curve $C_0$
according to the control points of the cage $v^i$. Note that it is equivalent to the Eq. 2.8 and 2.9 but expressed in a matrix notation.

The underlying idea behind Eq. 3.5 is that any deformation of the curve $C_0$ can be achieved just by determining the control point values in $v^i$; in other words, setting whatever coordinates positions in any of the control point $v^j$ will lead $C_0$ to take different shapes.

### 3.2 Modeling texture

So far we have seen in section 3.1 how the shape information has been modeled. As we pointed out in section 2.4, a full appearance model is made up of both shape and texture information. In this context, texture information is understood as the pattern of intensities or colors across an image patch, i.e the pixel intensity values of an image patch. The goal is then to construct a generative model able to capture pixel intensity variations on a set of images.

#### 3.2.1 The shape-free patch

Image warping [24] consists in finding a transformation which maps all positions in one image plane to positions in a second plane. One application is found in the image alignment problem where the goal is to align two images that they contain the same visual information but have different geometry. The alignment is then performed by finding a suitable geometric transformation (translation, rotation, scale...) that best match a set of point-to-point correspondences between the pair of images.

In relation with the previous explanation, image warping is strongly related with our work because we will need to compare to vector of textures that come from different shapes and thus they will have different size. In order to be able to compare them, both textures must be projected, i.e warped, to a common frame of reference called the “shape-free patch” which in turn corresponds with the contour described by the control points of the mean cage $\bar{v}$. In fact, this can be understood as a normalization of the shape using the mean shape as the reference.

More in detail, in order to build a model of texture variations, the first step would be to sample from image $I_i \in \{I\}$ the pixel intensity values located under the region covered by its binary mask $M_i$ to form the texture vector $g_i$. However, since $M_i$ can have different shape $s_i$ with respect to each other and consequently the length of $g_i$ would be different too, a shape-normalization must be previously performed in order to do a cor-
rect sampling. This normalization implies a warping operation over $I_i$ to transform the image into the the common frame of reference or "shape-free patch".

In Cootes et al. [14] was reported the use of a triangulation algorithm for warping an image, being Delaunay [38] one of the most well-known methods. Instead, we will use already known *mean value coordinates* (recall section 2.3) for warping an image in a very straightforward way and thus avoid to use another method.

In particular, in order to perform a shape-normalization by means of warping we firstly compute the mean value coordinates of all the pixel positions of the $I_i$ with respect to the mean cage $\bar{v}$, since it is the cage that parametrize the target shape. Let us define these coordinates as $F_{I_i}$. Then, we do

$$P' = F_{I_i}v^i$$

(3.6)

where $P'$ is a matrix containing new pixel coordinates $(x, y)$ as a result of the warp, $F_{I_i}$ is a matrix containing the mean value coordinates of every pixel position in $I_i$ with respect to mean cage $\bar{v}$ and $v^i$ is cage parameterizing the original shape of $I_i$. $P'$ must be seen as the backward mapping of the pixel’s positions; in other words, $P'(i, j)$ will contain the new position of the pixel $(i, j)$. Then we sample the original image with the new pixel positions as:

$$I^w_i = I_i(P')$$

(3.7)

where $I^w_i$ corresponds with the warping of $I_i$. Finally, sampling $I^w_i$ under the white region of the mask generated from the contour expressed by the mean cage $\bar{v}$ will give the texture vector $g_i$ shape-normalized. Note here of the advantage that any warp can be achieved just by switching $v^i$ for any cage $v$ in equation 3.6. Therefore, we can express any warp given an image $I$ and a target cage $v^t$ and a source cage $v^s$ as

$$W(I; v^t, v^s) = I(F_{I}(v^s))$$

(3.8)

where here $F_{I}$ denotes the mean value coordinates of pixel’s position of I with respect to $v^t$.

### 3.2.2 Building the texture model

Recall the formulation of the problem: we are given a set of images $I_1 \ldots I_K$ and a set of cages $v^1 \ldots v^K$ as a result of CAC minimization over a set of
binary masks $M_1 \ldots M_K$ where $M_i$ is the binary mask belonging to $I_i$ (see 3.1.1). First we compute the mean cage $\bar{v}$ as:

$$
\bar{v} = \frac{1}{K} \sum_{i=1}^{K} v^i
$$

(3.9)

Texture vectors $g_i$ are extracted by warping $I_i$ to obtain $I_i^w$ using Eq. (3.6) and sampling pixel intensity on values on $I_i^w$ over the region covered by the mean shape $\bar{s}$ using Eq. 3.7, see section 3.2.1 for more details. We further construct texture matrix $G$ as the concatenation of each texture vector $g_i$:

$$
G = (g_1, g_2, \ldots, g_M)^T
$$

(3.10)

In order to build a linear model of texture variations, we apply Principal Component Analysis on $G$. From the analysis we extract matrix $P_g$ consisting in a set of orthonormal base vectors (also known as principal component), their associated eigenvalues $\lambda_i$ and the mean of the pixel intensity values $\bar{g}$. We keep $\tau$ principal components vectors, which are enough to explain $\epsilon_{\text{var}}$ amount of variance. Usually this amount of variance is set to be $0.95 \leq \epsilon_{\text{var}} \leq 1$. Any texture vector $g$ can then be approximated by the model using:

$$
g = \bar{g} + P_g b_g
$$

(3.11)

where $\bar{g}$ is the mean texture, $P_g$ is a set of orthogonal modes of variation and $b_g$ is a vector of texture parameters.

### 3.3 Combining models of shape and texture

To finally build the full Active Appearance model, we concatenate shape and appearance parameters $b_v$ and $b_g$ as follows

$$
b = \begin{pmatrix}
W_v b_v \\
- b_g
\end{pmatrix} = \begin{pmatrix}
W_v P_v^T (v - \bar{v}) \\
- P_g^T (g - \bar{g})
\end{pmatrix}
$$

(3.12)

where $W_v$ is a diagonal matrix of weights for each shape parameters, allowing for the difference in units between the shape and gray models (see 3.3.1). Applying a further PCA we have

$$
b = Qa
$$

(3.13)

where $Q$ is the matrix of eigenvectors and $a$ is the vector of appearance parameters controlling both the cage and texture of the model. What we
have now is a full appearance model able to generate valid instances of the

cage and the texture by giving values to appearance parameters \( a \). Mathematically, this is expressed as

\[
\mathbf{v}' = \bar{\mathbf{v}} + P_v W_v^{-1} Q_v \mathbf{a} \\
\mathbf{g}' = \bar{\mathbf{g}} + P_g Q_g \mathbf{a}
\]  

(3.14)

(3.15)

\( \mathbf{v} \) corresponds with the cage associated to appearance parameters \( a \); alternatively, \( \mathbf{g} \) is the shape-normalized texture also associated to appearance

parameters \( a \).

In order to reconstruct the full image \( I_{model} \) associated to \( a \), we just have
to de-normalize \( \mathbf{g}' \) to fit the shape parametrized by the cage \( \mathbf{v}' \). This way we will have texture \( \mathbf{g}' \) projected against its original shape \( s \) and the image will be fully reconstructed. Therefore, given appearance parameters \( a \), we plug \( a \) into equations (3.15) and (3.14) so we get from the model the texture vector \( \mathbf{g}' \) and its respective cage \( \mathbf{v}' \) that in turn defines its original shape. From that, we apply a warping operation over obtained \( \mathbf{g}' \) by plugging the obtained \( \mathbf{v}' \) as target cage and \( \bar{\mathbf{v}} \) as source cage in equation (3.8. The result is the texture sample \( \mathbf{g}' \) projected to its respective shape.

### 3.3.1 Comparing Pixel-distances and Intensity

Since the cage parameters, \( \mathbf{b}_v \), has units of pixel distance and the texture parameters \( \mathbf{b}_g \) has units of pixel intensity they will obviously not commensurate without the weighting of \( W_v \).

A simple method to estimate \( W_v \) devised in [18] is to weight uniformly with the ratio, \( r \), of the total variance in shape and texture as seen in the training set. Remembering that the variance of parameter \( b_i \) where \( b \in \{ \mathbf{b}_v, \mathbf{b}_g \} \) equals \( \lambda_i \) we have:

\[
W_s = r I = \begin{bmatrix} r & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r \end{bmatrix} \\
r = \frac{\lambda_g}{\lambda_v} \quad , \quad \lambda_g = \sum \lambda_{g_i} \quad , \quad \lambda_v = \sum \lambda_{v_i}
\]  

(3.16)

(3.17)

### 3.4 Segmentation of new images

We now have a model that encodes the possible variations of cage shapes and texture of the considered object. Given an unseen (test) image \( I_{test} \),
the goal is to adjust appearance parameters $a$ so that the synthetic instance generated by the AAM model matches the instance of the new image as closely as possible. Thus, segmentation of a new image turns into a search problem where appearance parameters have to be optimized in order to find the best synthesized image by the model that matches the best with respect to the test image.

Let us first clarify how an instance is generated from appearance parameters $a$. As we have previously seen, $a$ parameters control the generation of new instances as each parameter $a_i \in a$ is a weight that can be interpreted as how much eigenvector $i$ contributes to the reconstruction of the instance. Given $a$, we apply Eq. 3.15 to obtain normalized texture vector $g_m$, which is the texture that the AAM generates, and Eq. 3.14 to get the cage $v$ associated to that particular texture. Recall that the cage is parameterizing the contour of the shape so applying Eq. 2.9 we recover the points belonging to the contour and use them to generate a binary mask $M$. The last step is to undo the shape normalization by projecting $g_m$ to its original shape parameterized by $v$; in other words, fill $M$ with texture vector $g_m$. But as we pointed out in Section 3.2.1, a transformation (warp) must be find in order to map every pixel located in the white region of $M$ with the corresponding pixel of $g_m$ in order to perform a correct fill of $M$.

As we have said, we have to optimize $a$ so that the instance generated by the AAM model matches the instance appearing in the new image as closely as possible. In the previous paragraph, We have explained how the texture shape normalized $g_m$ is generated by the model but still we haven’t mentioned anything how to obtain ”the instance appearing in the new image”. Let us define it as $g_t$. Once we computed $g_m$ and $v$ from $a$, we compute mean value coordinates of each pixel in $I_{test}$ with respect to control points from the mean cage $\bar{v}$ and we apply Eq. 2.9 using $v$; this way, we are applying a backward transformation that maps each pixel of $I_{test}$ with the warp of itself according to the shape described by $v$. Applying Eq. 3.7 we get the test image warped so the region located under the binary mask of the mean shape contains texture $g_t$.

Realize here the relationship between evolving curve, the cage and the AAM parameters. When we determine new values for $a$, we are generating a new cage that in turn is moving the evolving curve $C$.

Mathematically, the residual to be minimized is

$$r(a) = g_t - g_m$$  \hspace{1cm} (3.18)

where $g_t$ is the shape-normalized texture vector of the test image and $g_m$ is the texture vector corresponding to the current appearance parameters $a$. 

25
In particular, the statement of the optimization problem is defined as

$$\arg_{a} \min \sum_i r_i(a)^2$$ \hspace{1cm} (3.19)

where \( r_i \) is the residual at i-th pixel between \( g_s \) and \( g_m \) computed in Eq. (3.18). Hence, the previous equation can be written as

$$\arg_{a} \min \sum_i (g_t(i) - g_m(i))^2$$ \hspace{1cm} (3.20)

In matrix notation, the previous equation can be expressed as

$$\arg_{a} \min r(a)^T r(a)$$ \hspace{1cm} (3.21)

The energy \( E \) with respect to parameters \( a \) is then expressed as

$$E(a) = r(a)^T r(a) = \sum_i (g_t(i) - g_m(i))^2$$ \hspace{1cm} (3.22)

The proposed optimization problem appears at first to be a difficult high-dimensional optimization problem since the appearance models can have many parameters. In addition, fitting an AAM to an image is a non-linear optimization problem \([45]\). Many methods and algorithms have been reported in order to find the best way to solve the problem. In particular, we followed the approach presented in the original AAM article \([15]\) that is based on a Gauss-Newton algorithm for solving non-linear least squares problem. As we said, the main idea is to refine iteratively parameters \( a \) so that at each iteration the energy is continuously minimized, driving \( a \) to find a minimum for the energy \( E \) defined in Eq. (3.22). More formally, \( a \) values are obtained by successive approximation as

$$a^{k+1} = a^k + \Delta a$$ \hspace{1cm} (3.23)

so that \( \Delta a \) must fulfill \( E(a^{k+1}) < E(a^k) \). \( \Delta a \) is defined as

$$\Delta a = -Rr(a)$$ \hspace{1cm} (3.24)

where \( R \) is computed as

$$R = (J^T J)^{-1} J^T$$ \hspace{1cm} (3.25)

where \( J \) corresponds with the Jacobian matrix defined as

$$J_{i,j} = \frac{dr_i}{da_j} = \frac{\partial r_i}{\partial a_j}$$ \hspace{1cm} (3.26)
$J$ is estimated by numeric differentiation, that is, the $j$th column is computed by systematically displacing each parameter $a_i$ from the known optimal value with a magnitude equals $k\sigma_j$ on training images and computing a weighted average of the residuals. In a standard optimization scheme, it would be necessary to recalculate $\frac{\delta r}{\delta a}$ but according to [15], each attempt to match the model to a new image is actually a similar optimization problem and can be considered approximately fixed and hence, estimate it once from a training set.

The algorithm used for fitting an AAM to an unseen image can be seen in Alg. 1. It is basically a classic gradient descend optimization algorithm which performs a finite number of iterations so that in each one the parameters are changed according to the gradient in order to decrease the error produced by the prediction.

In particular, we are given the fully trained AAM model, the matrix of estimation of gradients $R$, the initial mask $M_0$, the mean value coordinates of both the image position and the evolving curve $C$ and the image to test $I_{test}$. The algorithm starts by initializing appearance parameters vector $a$ to zero, producing the mean appearance and therefore mean cage $\bar{v}$ and mean texture $\bar{g}$ from Eq. 3.14 and 3.15. Note that at this point, $\bar{g}$ corresponds with the current texture estimation $g_m$. We sample from $I_{test}$ pixels that belong to the white region of the mean mask $\bar{M}$ to get the texture $g_t$. We compute the residual $r$ between the texture of the test image and the one predicted by the model, $g_t$ and $g_m$. The, algorithm starts iterating until the termination criteria is not satisfied. This criteria is defined as either the difference between the last energy and the current is less or equal than a tolerance and that the algorithms haven’t performed more than a finite number of iterations. In each iteration, we enter in another level of optimization loop where this is focused on choosing the best learning rate by decreasing it exponentially if the last energy is not minimized with respect to the energy obtained in the previous iteration. Once the learning rate value is computed, the algorithm proceeds to update parameters $a$ by subtracting the gradients estimated in $R$ with respect to the value of the residual weighted by the learning rate. The result is that we get new appearance parameters $a'$ that in turn provide a new estimations of the cage and the texture. Once we have new appearance parameters, we compute again the error and the energy and we compare it with the last one obtained, if it is lower then the learning rate is good enough and proceed to the next iteration; otherwise, iterate until find an appropriate learning rate. If the energy don’t decrease, then declare convergence and proceed to compute the final binary mask from the shape parametrized by the cage.
Algorithm 1 Segmentation of a new image

Input:
- Active Appearance Model of the object to segment
- Matrix $R$ containing the approximations of the gradients.
- Mean mask $\bar{M}$
- Test image $I_{test}$

Output:
- Segmentation mask indicating which pixels belong to the object in the given $I_{test}$.

Procedure:
1. Set initial appearance parameter vector $a = 0$. This gives $g_m$ and cage $v$ that corresponds with the mean appearance.
2. Set $E_{old} = 0$
3. Set $I^w = W(I_t; \bar{s}, v)$ using Eq. 3.8 and sample pixels of region defined by $\bar{M}$ to get $g_t$
4. Compute energy residual $r(a)$ using Eq. 3.18 and $E_{new}$ using Eq. 3.22
5. while $\text{abs}(E_{old} - E_{new}) > TOL$ and $iter < MAXITER$
6. $E_{old} = E_{new}$, $iter \leftarrow iter + 1$
7. for $i = 0 \ldots depth$
8. $\alpha = 0.5^i$
9. Update parameters $a' = a - \alpha Rr(a)$
10. Generate new $v$ and $g_m$ using Eq. 3.14 and 3.15 with $a'$
11. Repeat steps 3 and 4 to compute $r'(a')$ and $E_{new}$
12. If $E_{new} < E_{old}$, break
13. From optimized $a$ use Eq. 3.14 to get the cage $v$
14. Apply Eq. 3.5 to obtain shape $s$
15. return $M_{seg}$ as the binary mask generated from $s$
4 Application to brain MRI images

In this section it is given an overview of the full system and the details of how it was applied to segment caudate nuclei, putamen, pallidum and accumbens structures in magnetic resonance images.

The major part of the application has been implemented using MATLAB and uses the library SPM8, which is a toolbox for MATLAB that uses the concepts of Statistical Parametric Mapping [52] to implement a series of routines to register, segment, normalize and analyze brain imaging data. It was designed to analyze fMRI data, but can be used out-of-the-box to work with MRI data. However, there is a module written in C corresponding with the Cage Active Contours implementation and several Bash scripts for establishing the interoperability between MATLAB and C modules.

Figure 4.1: Overview of the segmentation system. Illustration for one slice.

The approach is illustrated in Figure 4.1 and is explained as follows. Let us assume that we have the MR grayscale brain image volumes of a set of subjects and their corresponding manually segmented left and right brain structures as a binary mask volumes.

1. Register the volumes containing gray-scale MRI images corresponding
to each subject as well as the corresponding volumes regarding the manual segmentations of the desired structures in order to bring all of them to a normalized space.

2. Construct an Active Appearance Model for the appearance variations of the desired brain structures on each slice, using the process described in Section 3 over the manual segmentations of the structures in the normalized space. There will be a different AAM for each slice and structure.

3. Perform the segmentation of each structure in the normalized space, using the procedure described in Section 3.4 over the slices of the MR brain image volume of the unsegmented subject.

4. Apply the inverse registration transformation to the segmented volume found in the previous step to obtain the final segmentation in the original space.

These steps are explained in detail in the next sections.

4.1 Registration and normalization

The first step consists on bringing the image volumes of all the subjects to the same coordinate space (MNI space [11]) so we can statistically analyze the deviations in contour shape from the template later. From a Machine Learning perspective, what we are doing is ensuring that all instances share the same features (which are coordinates in this case) so we can learn patterns from them. To do so, we warp the image volumes of both the segmented subjects and the unsegmented one to match the atlas. This process is known as image registration.

SPM 8 includes a function to perform image registration using a combined model [17]. This function is based in a probabilistic framework that combines tissue classification, bias correction and image registration into a single generative model based on Gaussian mixtures to match the white matter, gray matter and cerebro-spinal fluid of the subjects volumes being processed to defined probabilistic templates. Since we have no specific needs, we use the default parameters of SPM for the integrated tissue classification and normalization process.

Apart from the normalized image volumes, the function produces by default, among other things, the parametric specification of the spatial normalization transformations for every subject (direct and inverse). These transformations can be used later to transform from the original space to
the normalized one (direct) and from the normalized space to the original one (inverse). We use these parametrized transformations to bring all the manual segmentations to the normalized space of the atlas.

This pre-processing registration step can be seen as equivalent to the rotation and scaling step to minimize the sum of squared distances between landmark points. Also, as noted in [1], a prior registration step can help to initialize the segmenting contour closer to the structure of interest. Without a proper initialization our method would probably fail to converge to the correct solutions in most cases, since it relies solely on local cues close to the segmentation region.

It is important to note that, while the original manual segmentations are given as a binary masks (either a voxel does belong to a given structure or it does not), the corresponding masks in the normalized space are no longer binary but gray-level, being more interpretable as probability maps. This is so because the registration transformation is not in general a one-to-one mapping between voxels, but one voxel in the original space can end up laying partially over several voxels and therefore the intensity values must be interpolated (here we use the linear interpolation methods from SPM and MATLAB).

4.2 Training a model for each slice

One model is trained for each slice and structure using both the normalized manual segmentations and gray-level MRI images.

In particular, binary masks of manual segmentations are plugged into the CAC method in order to obtain the corresponding cages, one per binary mask. Once cages are extracted, a statistical model of cage variations is built as explained in Section 3.1. On the other hand, MRI images are sampled, shape-normalized and analyzed in order to build a statistical model of texture variations (see Section 3.2 for more details). Finally, both models are combined to form the appearance model, which encodes both shape and texture information into a single set of parameters (see Section 3.3). An example of a learnt model of the caudate left for the slice 118 can be seen in Figure 4.2 where two first appearance parameters are displaced in order to illustrate the variability learnt.

The output of this step is a set of $L$ models where $L = |\{S\}|$, and $\{S\}$ is a set of slices. Note that each model is trained exclusively with binary masks and gray-level images corresponding with the corresponding slice.

It is important to note that there are three main parameters controlling the behavior of the method that have to be tuned in order to get the best
performance of the algorithm. They are the following ones:

- **Number of cage points**: corresponds with the cardinality of the set of points $v$ to be placed surrounding the evolution contour $C$. In general, the more points, the more $C$ can be adapted to the desired shape; however, more computations must be done. Hence, there is a trade-off between number of cage points and computational time cost.

- **Dilate size**: corresponds with the size in pixels of the ribbon used for computing the energy (see Section 2.3). The large the value, the bigger is the surface of pixels to take into account when computing the energy of Eq. 2.12.

- **Distance of the cage w.r.t the contour**: corresponds with the distance measured in pixels between the contour and $C$ and the cage. Since a ellipse-shaped cage has been used, this parameter corresponds with the dilation of the radius performed on the ellipse. The less distance, the more impact have control points over $C$; in other words, this parameter controls the intensity of the movement of $C$.

- **Threshold** for computing the initial mask: corresponds with the threshold for deciding the pixels that will conform the initial mask and in turn, the initial evolving curve.

Another parameter, not considered in the fine tuning, is $e_{\text{var}}$. It controls the amount of variance that the model has to kept (see Sections 3.1.2 and 3.2.2) and in turn, the size of parameters of the models as shape, texture or appearance. Is is commonly set to 0.95 or 0.98, being 0.98 our case.

### 4.3 Testing a new image

This step corresponds when a new volume coming from an unseen subject arrives to our system. The first step is to perform a registration and normalization to the same space that our model was trained in order to be able to treat it properly. The registration and normalization is performed using the library SPM8.

Once the volume is registered, we input each slice into the $L$ AAM models, one per slice, in order to get the segmentation of all the volume. Recall that this is a 2D segmentation system where each AAM has been trained with only example images from a certain slice, thus it segment in a slice-wise from. The system runs $L$ times Algorithm 1 in order to segment all the slices using the $L$ AAM models. From that, we obtain the output
of $L$ binary masks in the normalized space. This set of binary masks are arranged in a way that they construct a 3D volume.

The final step is then to revert the normalization by applying the inverse registration to the 3D Volume in order to obtain it in the original space. Each slice of the volume is then extracted obtaining the segmentation mask in the original space. The last step is then to label those pixels under the white region of the binary mask as pixels belonging to that structure.
5 Experiments

This section is devoted to explain the details regarding the experiments of the method that have been performed for this work in order to evaluate the proposed system in the segmentation task of some subcortical structures of the brain’s basal ganglia as caudate, accumbens, pallidum and putamen.

Also, we verify the results obtained in these structures comparing them with the ones obtained by state-of-the-art methods such as FreeSurfer and SRC and DDLS. Moreover, we compared the performance between our proposed method and the standard version of AAM by landmarking a set of slices of caudate and pallidum in order to see if there exists a difference between using cages for representing the shapes and using manually placed landmarks.

5.1 Dataset

We used a dataset consisting of 35 control subjects and their corresponding manual segmentations (ground-truth segmentations), made public by the MICCAI 2012 Grand Challenge on Multi-Atlas Labeling\(^1\). The dataset is made up of 13 males and 22 females, all subjects being right handed. Their ages ranged from 19 to 90 years old with an average of 32. Images were taken using a de-faced T1-weighted structural MRI technique, with a scan resolution of 256x256 pixels containing 60 slices. Thus, each subject corresponds with a 3-D volume of 256x256x60 images.

From the manual segmentations, we generated the isolated volumes for the according structures, caudate left and right, accumbens left and right, pallidum left and right and putamen left and right respectively, by generating a binary mask per slice using the part of the image labeled with the desired structure and discarding the rest.

The atlas tissue probability maps (corresponding to gray matter, white matter and cerebro-spinal fluid) used for image registration are the standard ones shipped with SPM, which are a slightly modified version of the ICBM Tissue Probabilistic Atlas\(^2\).

In order to make the comparison between our proposed method and the standard version of AAM, we manually placed landmarks in three slices of caudate and pallidum structures. In particular, we landmarked slices 117 to 119 of left and right caudate and slices 109 to 111 of the left and right

---

\(^1\)MICCAI 2012 Grand Challenge on Multi-Atlas Labeling https://masi.vuse.vanderbilt.edu/workshop2012/index.php/Main_Page

\(^2\)Available at http://www.loni.usc.edu/atlastes/Atlas_Detail.php?atlas_id=7
pallidum for all 35 subjects. The slices corresponds with the central ones in each respective structure. Note that in total we landmarked 420 images.

5.2 Methods

In order to give a better intuition about the performance of the proposed method, we made a comparison with current state-of-the-art methods DDLS, SRC and FreeSurfer. The last will be used as our baseline for measuring the performance of our method.

1. FreeSurfer is a set of tools for analysis and visualization of structural and functional brain imaging data developed at the Martinos Center for Biomedical Imaging by the Laboratory for Computational Neuroimaging. FreeSurfer segmentation method is based on a probabilistic atlas approach [22].

2. SRC [30] is a theoretical framework for signal classification of sparse representation (factorization) of signals over an over-complete basis (dictionary). It is a Sparse dictionary learning approach where combines the discrimination power of the discriminative methods with the reconstruction property and the sparsity of the sparse representation that enables one to deal with signal corruptions: noise, missing data and outliers. This approach is therefore capable of robust classification with a sparse representation of signals.

3. DDLS [5] is a discriminative dictionary learning approach very similar to SRC. They differ in the way they build the dictionary where SRC uses the whole patch library and DDLS learns a compact-task specific dictionary and a classifier for each target.

5.3 Evaluation metrics

For giving a measure of the quality of the segmentation, we used the well known Dice coefficient (also known as Sørensen index). This metric is used for comparing the similarity between two sets; in our case, we use it for computing the similarity between two binary masks: the manual segmentation (ground-truth) and the provided by our method. This comparison will give us the measure of how far is our method for being perfect and thus what is the quality of the automatic segmentations compared to the manual ones. Mathematically, Dice coefficient is defined as

\[
Dice(A, B) = \frac{2|A \cap B|}{|A| + |B|}
\]  

(5.1)
where A and B are the two sets. More simply, this formula represents the size of the union of 2 sets divided by the average size of the two sets.

A value of 0 indicates no overlap; a value of 1 indicates perfect agreement. Higher numbers indicate better agreement, and in the case of segmentation indicate that the results match manual segmentation better than results that produce lower Dice coefficients.

5.4 Validation protocol

We make use of very well known cross-validation technique in order to validate the results obtained. In particular, cross-validation is a technique used for assessing how the results of a statistical analysis will generalize to an independent data set; in other words, is a prediction of the performance of the method over unseen data sets, i.e the out-of-sample error. Cross-validation it is also used for model selection, understood as finding the best configuration of hyper-parameters for the method that give the best performance on a particular dataset.

In particular, we are interested on computing the out-of-sample Dice coefficient from the given data. The best way to estimate it is using a nested cross-validation, that is two cross validation running: the outer one is responsible of partitioning the data in k-folds, one fold for testing and the rest for learning plus validation; the inner one is the responsible of re-partitioning the set made up of learning plus validation in k-folds where one is used for validating the model and the rest for learning. Therefore, the inner cross-validation performs model selection to find the best configuration of hyper-parameters, also known as grid search, and thus the best model meanwhile the outer computes the out-of-sample error for each fold.

Our system implements a 5*5-fold cross-validation protocol using a grid search strategy for finding the optimal hyper-parameters in each fold, the values of them are shown in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number control points</td>
<td>8</td>
</tr>
<tr>
<td>Dilation size (in pixels)</td>
<td>3, 5, 8, 10</td>
</tr>
<tr>
<td>Distance of the cage (in pixels)</td>
<td>3, 5, 8, 10</td>
</tr>
<tr>
<td>Thresold for computing $M_0$</td>
<td>0.2, 0.3</td>
</tr>
</tbody>
</table>

Table 1: Grid of hyper-parameters
5.5 Results

In this section we will present the results obtained. Results from table 2 took about two days of computation on a Intel Core i3 machine.

5.5.1 Quantitative

Table 2 shows the results of applying our method in order to segment four brain subcortical structures from the Basal Ganglia such as Caudate, Accumbens, Pallidum and Putamen on MICCAI 2012 Grand Challenge on Multi-Atlas Labeling dataset. Rows correspond with subjects and columns correspond with structures. Realize that each column shows the arithmetic mean of its respective right and left structure part results. The mean and standard deviation of each structure are reported at the bottom of the table. The values equate to dice coefficients computed between the ground truth and the resulting mask of our method using a nested 5*5-fold cross-validation protocol in order to correctly compute the out-of-sample dice coefficient.

Looking at figures located in Table 2, it can be seen that except for Accumbens structure, the rest achieved more than 80% of Dice coefficient, being Putamen the best one with 83.8% while the worst was Accumbens achieving a 72.81%. One reason that could help to understand this behaviour is that Accumbens is the smallest structure among all analyzed in this work turning the learning of the AAM model into a difficult task. Moreover, standard deviation was the worst in Accumbens with a 9.8% while the best, again, was achieved by Putamen with a 4.9%. These figures turn Putamen into the best segmented structure by the proposed method in terms of accuracy.

Table 3 shows the comparison between the proposed method and three more methods extracted from [5]. In order to make a fair comparison, the same dataset was used for each method. Each column corresponds with a structure, being Basal Ganglia the average of all of them while rows correspond with the method used. Again, the values corresponds with the mean of Dice coefficients of left and right parts of each structure. Realize that the results of our method are extracted from Table 2.

The first thing that Table 3 shows is that DDLS method is the best both in average and taking each structure individually, reaching a Dice coefficient of 84.1%. In contrast, FreeSurfer was the worst method with a 72.5% in average as well as in each structure individually; in the middle we have our method with a 80.01% and SRC with a 83.3%. As occurred in the particular results of our method in Table 2, Accumbens struture was the
worst structure in all the methods, and especially in FreeSurfer where it only achieved a 55.2%, representing a gap of 17.6% with respect to our method and 20.6% with respect to DDLS. In contrast, Putamen achieved the best results in all the methods where the best method was DDLS with a 90.1% and FreeSurfer was the worst with a 78.6%.

Focusing on the performance achieved by our method, we can observe that our proposed method performed better in each structure as well as in average than our baseline method FreeSurfer since the difference in Dice of both is about 7.61% for the average of all structures. However, both DDLS and SRC still achieve better results than our method with a difference of 3.9% and 3.19% respectively.

In Table 4 can be seen the results of the comparison between Cage AAM and regular AAM using landmarks. Again, the results were obtained applying a 5*5 cross-validation in case of Cage AAM and a 5-fold cross validation in standard AAM since no hyper-parameter exist for it. Each row belong to a slice of a determined structure, and each row correspond with the dice coefficient result where these are obtained averaging for all subjects and for left and right parts of the structure. The last column belongs to the difference between Cage AAM and AAM columns where the sign of the measure tells which one is better.

Regarding the results, we can observe that the performance of AAM with landmarks is slightly better than AAM using cages in all slices, being the exception slice 109 of the pallidum where Cage AAM performance was 1.03% superior. The worst result for Cage AAM was slice 111 of pallidum where it performed 2.64% worse with respect to AAM. In average, we obtained that Cage AAM peformed worse than AAM a total amount of 0.96%.

In fact, this negative difference of 0.96% can be interpreted as the estimation of the price in terms of performance that one have to pay for avoiding landmarking if one wants to apply deformable models technique for a segmentation task, in other words is the loss estimation in terms of Dice coefficient of using cages instead of manually placed landmarks.

### 5.5.2 Qualitative

An example of qualitative results of each structure can be seen in Figure 5.1. The green contour corresponds with the manual segmentation, considered here as the ground-truth, and red contour corresponds with the segmentation of the method.

Appreciate how the red contour doesn’t perfectly match the green contour in any structure meaning that the method doesn’t perfectly segment.
Figure 5.1: Examples of segmentation of subject 16. From left to right, top to bottom: Pallidum, Putamen, Accumbens and Caudate. Green contour is the manual segmentation; red contour corresponds with the segmentation of the method.

that structure. In case of Putamen left structure (top-right image), the contour corresponding with the result of the segmentation is left far from the real shape of the manual segmentation as it is bigger and too roundly compared with the original shape. This can be an example of the case when the gradient descend algorithm employed for testing a new image (see Alg. 1) is stucked in a local minima so that the the contour obtained is far from the optimum.

In Figure 5.2 can be seen qualitative results of the comparison between our proposed method and regular AAM with the landmarked images. As can be seen, segmentation results of our method seem to over-segment in the sense that the red contour is bigger than the ground-truth; in contrast,
Figure 5.2: Examples of segmentation of subject 16 of the proposed method (left column) and regular AAM using landmarks (right column). Top row corresponds with pallidum (slice 111) and bottom row with caudate (slice 118). Green contour is the manual segmentation; red contour corresponds with the segmentation of the method.

AAM with landmarks seem to keep the green contour more adjusted with respect to the ground-truth but the segmented zones are smaller.
Table 2: Dice coefficient results for each structure and subject using a 5*5-fold cross validation. Each structure is the result of the average between its left and right part.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Caudate</th>
<th>Accumbens</th>
<th>Pallidum</th>
<th>Putamen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8556</td>
<td>0.6888</td>
<td>0.8074</td>
<td>0.8686</td>
</tr>
<tr>
<td>2</td>
<td>0.8562</td>
<td>0.7037</td>
<td>0.7846</td>
<td>0.8307</td>
</tr>
<tr>
<td>3</td>
<td>0.8907</td>
<td>0.7359</td>
<td>0.8369</td>
<td>0.8786</td>
</tr>
<tr>
<td>4</td>
<td>0.7694</td>
<td>0.832</td>
<td>0.7827</td>
<td>0.8459</td>
</tr>
<tr>
<td>5</td>
<td>0.8893</td>
<td>0.7784</td>
<td>0.8608</td>
<td>0.9103</td>
</tr>
<tr>
<td>6</td>
<td>0.7664</td>
<td>0.7807</td>
<td>0.8490</td>
<td>0.8648</td>
</tr>
<tr>
<td>7</td>
<td>0.8434</td>
<td>0.7912</td>
<td>0.8600</td>
<td>0.8839</td>
</tr>
<tr>
<td>8</td>
<td>0.9085</td>
<td>0.6941</td>
<td>0.6992</td>
<td>0.7575</td>
</tr>
<tr>
<td>9</td>
<td>0.9187</td>
<td>0.8409</td>
<td>0.8632</td>
<td>0.8935</td>
</tr>
<tr>
<td>10</td>
<td>0.8782</td>
<td>0.8173</td>
<td>0.7824</td>
<td>0.8419</td>
</tr>
<tr>
<td>11</td>
<td>0.8822</td>
<td>0.7825</td>
<td>0.8663</td>
<td>0.8443</td>
</tr>
<tr>
<td>12</td>
<td>0.8446</td>
<td>0.8146</td>
<td>0.8713</td>
<td>0.8724</td>
</tr>
<tr>
<td>13</td>
<td>0.8759</td>
<td>0.8062</td>
<td>0.7935</td>
<td>0.8527</td>
</tr>
<tr>
<td>14</td>
<td>0.8779</td>
<td>0.7345</td>
<td>0.8096</td>
<td>0.7954</td>
</tr>
<tr>
<td>15</td>
<td>0.8511</td>
<td>0.8091</td>
<td>0.8537</td>
<td>0.8684</td>
</tr>
<tr>
<td>16</td>
<td>0.9108</td>
<td>0.8052</td>
<td>0.7875</td>
<td>0.7927</td>
</tr>
<tr>
<td>17</td>
<td>0.8117</td>
<td>0.5886</td>
<td>0.8208</td>
<td>0.8626</td>
</tr>
<tr>
<td>18</td>
<td>0.7940</td>
<td>0.6575</td>
<td>0.7220</td>
<td>0.7759</td>
</tr>
<tr>
<td>19</td>
<td>0.8483</td>
<td>0.6879</td>
<td>0.7358</td>
<td>0.7800</td>
</tr>
<tr>
<td>20</td>
<td>0.7468</td>
<td>0.7263</td>
<td>0.8157</td>
<td>0.8032</td>
</tr>
<tr>
<td>21</td>
<td>0.8726</td>
<td>0.7886</td>
<td>0.8328</td>
<td>0.9106</td>
</tr>
<tr>
<td>22</td>
<td>0.8522</td>
<td>0.8007</td>
<td>0.8201</td>
<td>0.8476</td>
</tr>
<tr>
<td>23</td>
<td>0.8825</td>
<td>0.8192</td>
<td>0.8141</td>
<td>0.8508</td>
</tr>
<tr>
<td>24</td>
<td>0.8535</td>
<td>0.6431</td>
<td>0.8385</td>
<td>0.8295</td>
</tr>
<tr>
<td>25</td>
<td>0.8754</td>
<td>0.7166</td>
<td>0.7912</td>
<td>0.8087</td>
</tr>
<tr>
<td>26</td>
<td>0.8564</td>
<td>0.806</td>
<td>0.8427</td>
<td>0.8505</td>
</tr>
<tr>
<td>27</td>
<td>0.8618</td>
<td>0.7837</td>
<td>0.8596</td>
<td>0.9031</td>
</tr>
<tr>
<td>28</td>
<td>0.8441</td>
<td>0.5426</td>
<td>0.7886</td>
<td>0.8461</td>
</tr>
<tr>
<td>29</td>
<td>0.8397</td>
<td>0.8303</td>
<td>0.8801</td>
<td>0.8662</td>
</tr>
<tr>
<td>30</td>
<td>0.7660</td>
<td>0.7168</td>
<td>0.8108</td>
<td>0.8390</td>
</tr>
<tr>
<td>31</td>
<td>0.8118</td>
<td>0.452</td>
<td>0.8756</td>
<td>0.8625</td>
</tr>
<tr>
<td>32</td>
<td>0.6849</td>
<td>0.6178</td>
<td>0.8574</td>
<td>0.8562</td>
</tr>
<tr>
<td>33</td>
<td>0.7045</td>
<td>0.6075</td>
<td>0.7351</td>
<td>0.7695</td>
</tr>
<tr>
<td>34</td>
<td>0.6127</td>
<td>0.7681</td>
<td>0.7346</td>
<td>0.7990</td>
</tr>
<tr>
<td>35</td>
<td>0.7707</td>
<td>0.5169</td>
<td>0.5512</td>
<td>0.6673</td>
</tr>
<tr>
<td>Mean</td>
<td>0.8317</td>
<td>0.7281</td>
<td>0.8067</td>
<td>0.8380</td>
</tr>
<tr>
<td>Std</td>
<td>0.0681</td>
<td>0.0981</td>
<td>0.0648</td>
<td>0.0496</td>
</tr>
</tbody>
</table>
Table 3: Table containing the comparison of our method against several methods for some structures of the Basal Ganglia using Dice coefficient metric.

<table>
<thead>
<tr>
<th>Structure and slice</th>
<th>Cage AAM</th>
<th>AAM</th>
<th>Difference in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caudate slice 117</td>
<td>0.7998</td>
<td>0.8031</td>
<td>-0.33</td>
</tr>
<tr>
<td>Caudate slice 118</td>
<td>0.8135</td>
<td>0.8264</td>
<td>-1.29</td>
</tr>
<tr>
<td>Caudate slice 119</td>
<td>0.8213</td>
<td>0.8355</td>
<td>-1.42</td>
</tr>
<tr>
<td>Pallidum slice 109</td>
<td>0.8239</td>
<td>0.8136</td>
<td>1.03</td>
</tr>
<tr>
<td>Pallidum slice 110</td>
<td>0.8126</td>
<td>0.8239</td>
<td>-1.13</td>
</tr>
<tr>
<td>Pallidum slice 111</td>
<td>0.7822</td>
<td>0.8086</td>
<td>-2.64</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td>0.8088</td>
<td>0.8185</td>
<td>-0.96</td>
</tr>
</tbody>
</table>

Table 4: Comparison about AAM with cages versus AAM with landmarks. Dice coefficient metric is used. Last column indicates the Dice difference between Cage AAM and AAM. A negative value means AAM with landmarks is better and a positive value means that Cage AAM is better.
6 Conclusions

In this work we have presented a new method for automatic segmentation based on the well-known deformable object technique Active Appearance Models, where its distinctive feature resides in the omission of the need for landmarks in order to represent the shape of an object. This feature is achieved via adjusting a curve, parametrized by a set of control points, to match the contour of a binary masks that in turn defines the real shape of the object using Cage Active Contours algorithm.

The presented method was applied to segment several structures from the brain’s Basal Ganglia part on a dataset containing 35 subjects. Since it is a 2D method, one model per slice and structure is trained with the set of training images and then tested with the set for testing using a $k \times k$-fold cross validation protocol, being 5 the number of folds used. From the validation, we obtained the out-of-sample Dice coefficient for each subject and structure. Moreover, we compared the obtained results with other three methods in order to contextualize the performance of the proposed method.

Results obtained shown that the proposed method outperformed our baseline method as Dice coefficient was higher in each structure. However, it exists a short gap in terms of performance between non-baseline methods and the proposed method, indicating that these methods are superior in the segmentation task as Dice coefficient was lower than DDLS and SRC methods. Nonetheless, we anticipate that this results can be improved by means of using a more advanced technique for fitting an AAM model to new images as it exists a dedicated research line that tries to improve the performance of AAM in the testing step.

Moreover, results shown that there exist a difference between using AAM with cages and using it with landmarks. However, this difference in performance is less than 1% with respect to landmarks, little enough taking into account the high time cost of marking a dataset where it easily may imply thousands of images even for a single structure for several subjects and some prior knowledge of the structure is required for correctly placing landmark. Therefore, it clearly appears a trade-off: using cages and thus fully automate the segmentation task using AAM or turning it into a semi-automatic procedure by labeling the data.

Setting aside the results obtained, we can claim that our work introduced an improvement of the AAM model when this is applied for segmenting images in the medical domain because two events take place on it: public databases usually come with manual segmentations and these are delivered as binary mask format. Since our method avoids landmarking, a very time-
consuming and tedious task, our method becomes a good option for the aforesaid domain.

To conclude, this work has pretended to make an step in the direction of turning Active Appearance Model into a fully automatic method by proposing a new approach taking advantage of Cage Active Contours. We achieved the objective of materializing the idea by implementing ant testing the method against a real database of brain MRI images and from that we obtained promising results. However, this approach has a large margin of improvement that we list in the next section.
7 Future work

In this section we are going to summarize the possible work that can be done in the future that might be useful for improving the proposed method and to continue exploring the possibilities of it.

As pointed out in the conclusions, by means of using a better fitting algorithm of an AAM model given a test image may improve the results. There exist plenty of different methods and approaches for that problem: inverse compositional approach methods such as Simultaneous Inverse Composition and Robust Inverse Composition [2, 45], regression approach methods such as Iterative Error Bound Minimisation [56], discriminative approach such as Haar-like Feature based [57], Adaboost based [40], evolutionary algorithms approach as genetic based [58] or evolution strategy based [27, 41]

As we explained in the document, the presented method is a 2D segmentation approach, meaning that a given structure is segmented slice per slice. An interesting improvement would be to work with Active Appearance Models specially designed for 3D volumes as it is considered in [4, 28, 48]. In relation with that, an extension of mean value coordinates for working in $\mathbb{R}^3$ space [34] and the Cage Active Contours framework in 3D explored in [62]. Therefore, the idea is to move the proposed method to work in a 3D space in order to give the ability of segmenting a full volume at once with the goal of learning a better representation given a certain structure as a volume.

It is known that Active Appearance Models and Deformable Models in general have been applied with great success on medical images such as MRI belonging to multiple anatomical human parts. In relation to that, another possible extension would be to try to apply the proposed method on other domains apart from medical images such as faces or whatever kind of object in order to explore how this method will perform in objects that can potentially have more variability among instances of the same class.
References


