Faculty of Mathematics and Natural Sciences

Department of
Mathematics and
Computing Science

Determining Regions of Interest in MRI-data

W.G.J. Netjes

Advisors:

dr. J.B.T.M. Roerdink

Department of Mathematics and Computing Science

University of Groningen

R.P. Maguire

Department of Neurology

Academic Hospital Groningen

August, 1999

RuG

Determining Regions of Interest (ROI's) in MRI-data

W.G.J. Netjes
w.g.j.netjes@wing.rug.nl

August 1999

Abstract

The purpose of this project is to examine methods for automatically defining Regions of Interest (ROI's) in the brain using data from Magnetic Resonance Imaging (MRI).
Although the data set is three dimensional (3D), it can be considered as a stack of 2D transaxial slice images, which simlifies the problem of region definition.
This report is in four parts. Firstly a description of the problem with reference to the measured data from the University Hospital Groningen (AZG), to outline the data structure and quality.
Secondly a development of the ROI template, consisting of a set of ellipses, which is used as a starting estimate.
Thirdly a description of a 'snakes' based method used to register the ROI template with data from MRI, and associated Graphical User Interface (GUI).
Lastly, testing of the method is described.

Preface

I would like to thank the people who helped me with my graduation assignment. Especially, I would like to thank Dr. J.B.T.M. Roerdink, for accompanying me with a lot of time and effort. Also I would like to thank R.P. Maguire for the assistance and for providing the data that was necessary for this project and the required tests. Besides this, thanks for giving me insight in the MRI-data and the location of the different structures in the brain, and for giving me advise on how the ROI's should be defined, so that it can be used for a clinical study in a later stage, when the method developed in this project works more perfectly.

Contents

Chapter 1

Analyzing MRI-images

1.1 Description of the MRI-data

The MRI-data that is used for analysis is acquired at the Academical Hospital of Groningen (AZG). Each dataset contains one MRI-brain and is in Mayo Analyze format, which is stored in one datafile `<file>.img` and one headerfile `<file>.hdr`. The MRI-data may contain a variable number of slices of the brain. The headerfile is needed, because it gives information about the dimensions and type of the data that is used for analysis.

1.2 Reading the MRI-data in Matlab

When reading the data in Matlab, the headerfile is used for extracting the dimensions and number of slices of the MRI-data. The structure of the headerfile is implemented as a Matlab function and only returns the data that is used to read the datafile correctly.
The file `readanatmri.m` is used to read an MRI-datafile. It's syntax is:
`function[anatomical_MRI,dimx,dimy,nSlices]=readanatmri(input_file)` ,
the input_file is the name of the file where the MRI-data is stored, extensions are not necessary to open the files. The function `readanatmri` looks first in the current directory if the files to be opened exist. If so, then the MRI-datafile `<file>.img` and `<file>.hdr` will be opened for reading. With the function `hread` the necessary contents of the headerfile will be extracted. When this is done the datafile will be read and stored along the right dimensions, figure 1.1 shows how the data is exactly stored. `readanatmri` also returns the dimensions of the x-axis and y-axis, even the number of slices will be returned. If the files to be opened don't exist in the current directory, then the default directory is used to open these files. This directory can be changed in the Matlab-file `readanatmri.m` to any other directory. If the files don't exist at all, an error message is returned in Matlab's MainEditWindow.

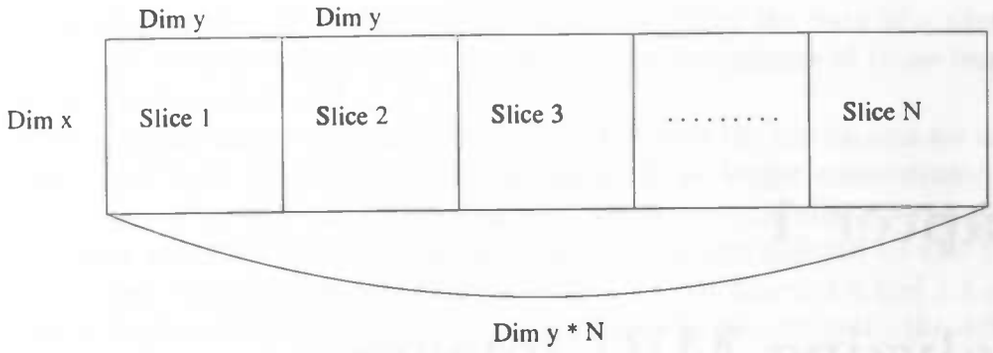

Figure 1.1: Structure of MRI-data array

1.3 Displaying the Data

Now the data finally can be read in Matlab, it is time to display the data on the screen. To do this, the data must first be rotated such that the forehead is up. This means that the eyes should be shown at the top of the image of a slice. In Matlab the builtin function `rot90` can be used. The MRI-data that is used here has been rotated 270 degrees, this is done by applying three times `rot90` or `rot90( -1)` once to an MRI-slice, which is extracted from the data, as described in figure 1.1. To display the image, the Matlab-function `imagesc` is used to stretch the image to all available grey values.

1.4 Histogram equalization

Now that it is possible to display the data, it is ready for analysis. From the MRI-data it is clear that there are a lot of structures in the brain. In some cases it's difficult to see the borders between different structures exactly. To display the data better, some data-preprocessing is applied to the MRI-scan of a dataset. First a kind of histograms are plotted, to get more information about the data, as can be seen in figure 1.2 on dataset 1229_a.

Looking to those histograms, one can conclude that they are very similar and that the intensity of the images is concentrated at two different peaks. Those peaks are located around 0 and 0.3. To avoid this, a histogram equalization is applied to the MRI-images.
In Matlab the function `histeq` is used to do this. Before using this function it is important to rescale the data between 0 and 1, otherwise the equalization would

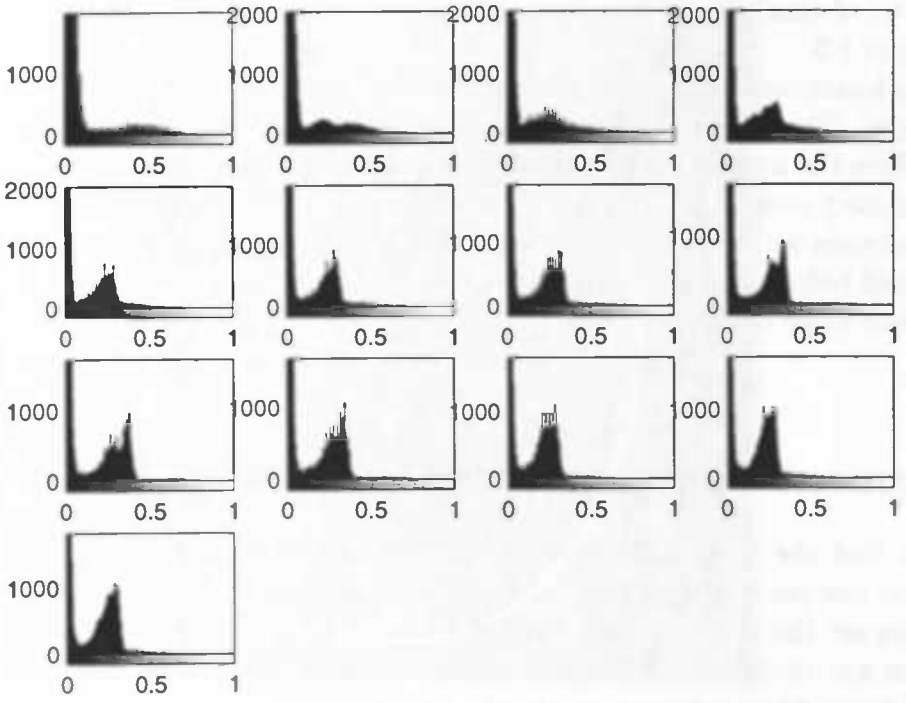

Figure 1.2: Histograms of original MRI-slices

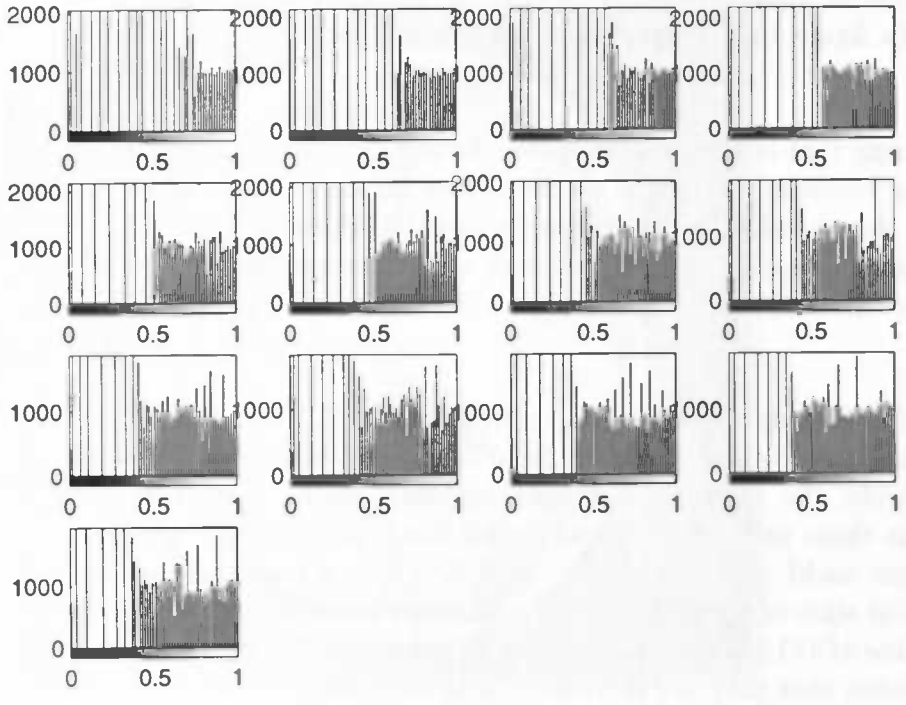

Figure 1.3: Histograms of MRI-slices after histogram equalization

not work properly. Rescaling can be done by dividing the data of a slice by the maximum value of that slice. The results of the histograms of those images can be seen in figure 1.3.
After applying histogram equalization, it is clear that the histograms are stretched more equal now. The intensity of the images is no longer concentrated around two peaks. When the images are displayed again, the contours of some structures are easier to detect now. But on some places it is still difficult to find precisely the borders between some of these structures. In the figures 1.4 and 1.5 an MRI-slice is displayed before and after histogram equalization to make the differences between the two more clear.

1.5 Segmentation using thresholding

A method to find the borders between structures is thresholding, which is a method to give one set of pixel values above a certain threshold value the value 1 and the other set the value 0. The result is a black and white image where the desired regions are visible, depending on which threshold value is used. Thresholding is used to get more information about a desired brain structure, and about how the pixel values are concentrated in such a region. In particular, the structures where one is most interested in are the 'Caudate head', 'Putamen', and 'Thalamus'. In figure 1.6 these structures are displayed in an MRI-slice.

The thresholding that is used here takes two boundary values, and an image value between those boundary values is set to 0, and otherwise the value is set to 1. For example, this thresholding is applied to the MRI-dataset '1228_a' on slice 11. In the first place this slice is equalized such that a better result is expected. In the two figures 1.7 and 1.8 it is clear that some structures can be seen.

In figure 1.7 the boundaries of the threshold are set to 700 and 800. In this figure we see the structures of the 'Caudate head' and the 'Putamen'. To make these structures visible, the values of the threshold are selected manually. This does not mean that these values are correct for all slices, because the maximum pixel value of a slice could differ. The slice used here has a maximum value of 893, but for the first slice of this dataset the maximum value is just 482 and the last slice has a value of 1118. When the maximum values of all slices are compared, it can be concluded that they are increasing with slice number (slice 1 at the top), sometimes there are exceptions. Even on different datasets the maximum values and the values of the thresholds could differ. Thus defining one pair of values for all the thresholds is not possible.

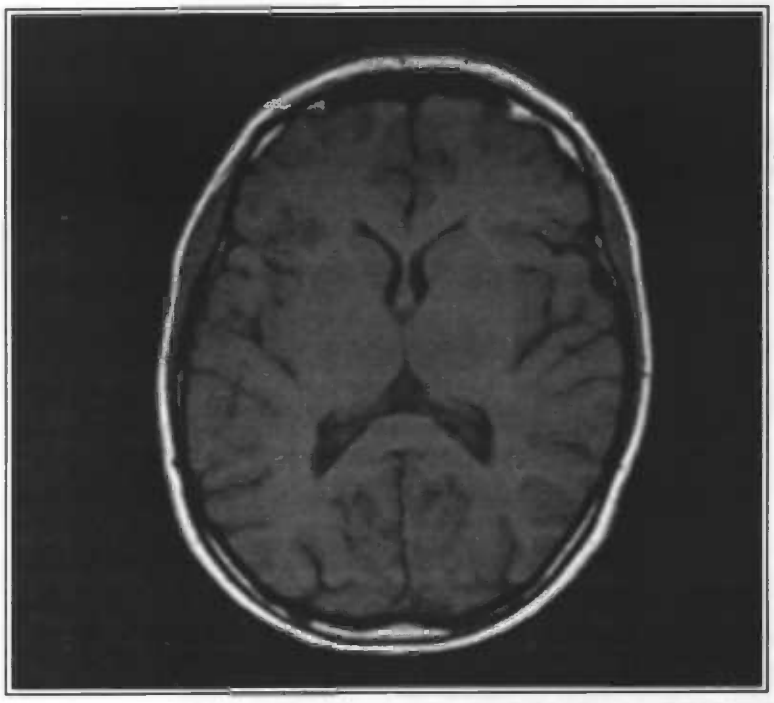

Figure 1.4: Original MRI-slice

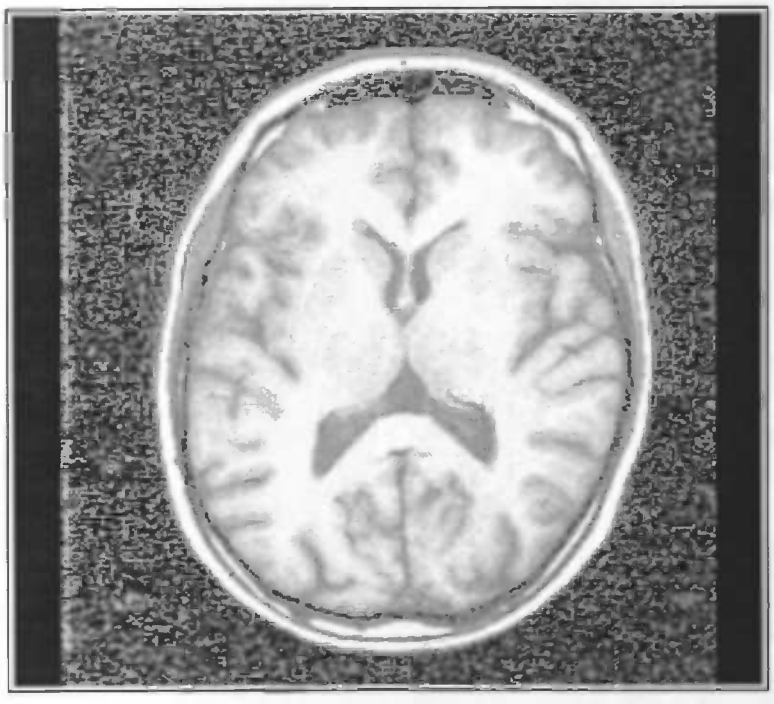

Figure 1.5: MRI-slice after histogram equalization

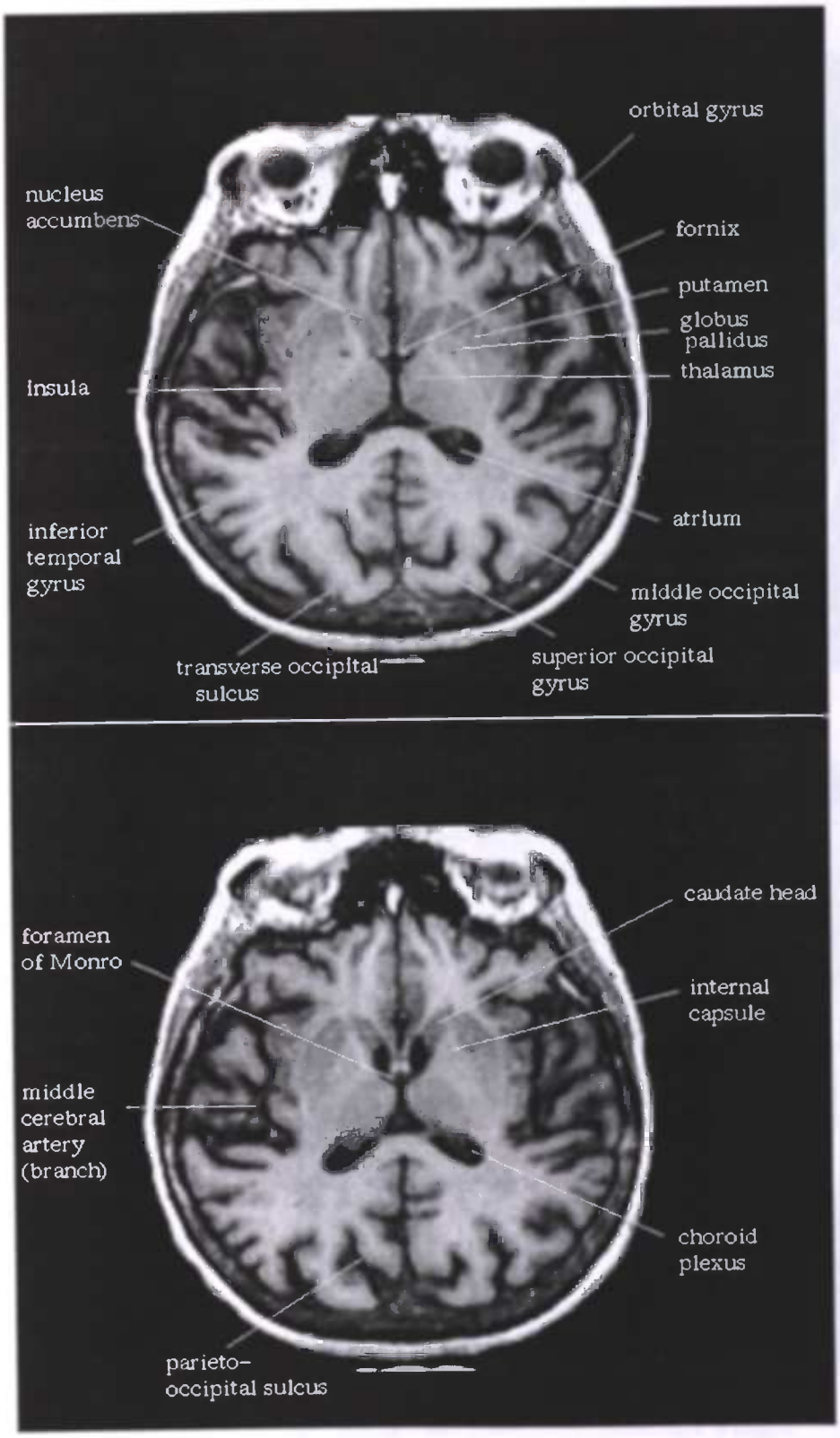

Figure 1.6: Specific brain structures

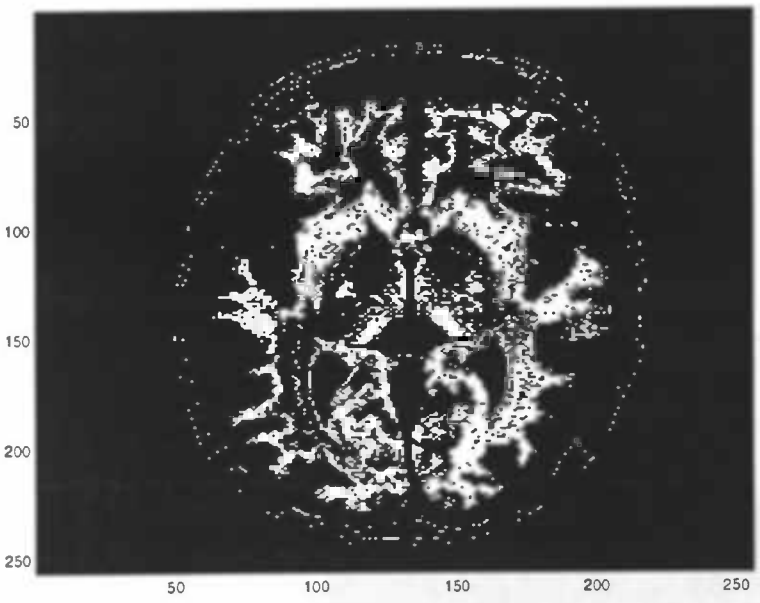

Figure 1.7: Thresholded MRI-slice with boundaries 700 and 800.

In figure 1.8 the boundaries of the threshold are set to 750 and 840 to make the Thalamus structure a little bit visible. Looking to some of those specific structures, we see that the border is not clear at all. But in some cases some structures have a greater concentration of black or white pixels than other structures. If you would like to extract some brain structures out of the images, you may need some additional information to find these structures.
In the next chapter a graphical user interface (GUI) will be described, which is implemented in Matlab, and can draw ellipses on the brain. (Other shapes can be added later.) Those ellipses are used as ROI's for a specific template to fit it on other slices in a later stage.

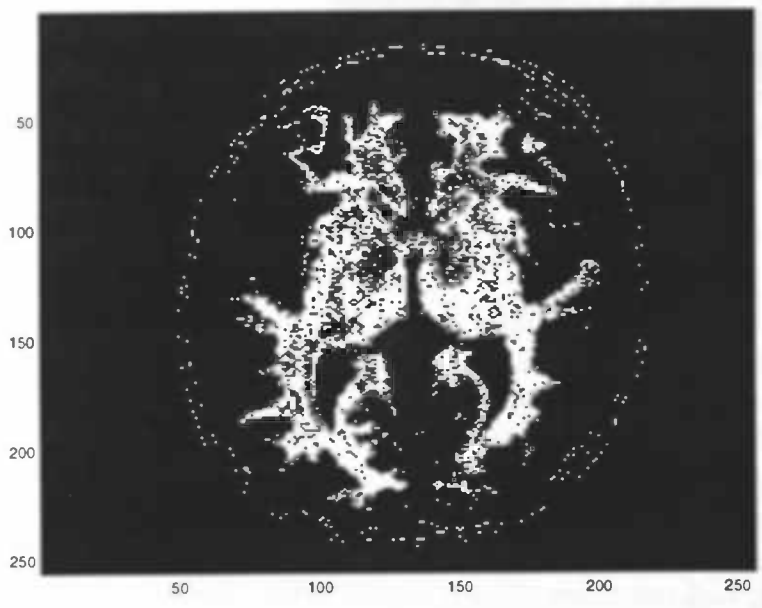

Figure 1.8: Thresholded MRI-slice with boundaries 750 and 840.

Chapter 2

Graphical User Interface

In this chapter a simple GUI for drawing ellipses in MRI-slices will be described and we discuss how it is implemented. The GUI is implemented in MATLAB 5.3, in the other versions (5.0 to 5.2) something has to be changed in the Matlab file where the GUI is described/implemented. The GUI is created to make it easy to draw a shape (ellipse) around a brain structure on which specific calculations can be performed. A problem with these shapes (ellipses) is that they do not fit a brain structure exactly. In one of the following chapters we will discuss the snake algorithm, that uses deformable contours, which can be adapted to fit brain structures more accurately.

2.1 Global description of the GUI

To develop the GUI, the Matlab function `guide` is used. This function produces one figure and the Control Panel Guide. With this Control Panel Guide the figure can be edited to a GUI. The GUI that is made here contains one window, the part of the window that displays the data, especially MRI-data. In figure 2.1 a screenshot of the GUI is displayed.
The scrollbar below the window is used to scroll through the MRI-slices of a particular brain. The number of slices that can be displayed through the scrollbar can not exceed the number of slices in the MRI-data. The text box above this scrollbar displays the number of the slice which is currently displayed. To the right of the scrollbar there is an edit box and a text box. The text box displays the name of the MRI-dataset which is currently displayed. To display an another MRI-dataset you must type the name of this MRI-dataset in the edit box. After pressing the enter-key, the function `readanatmri` as described above will read the datafile which is just entered. If the datafile does not exist some error-messages will appear in the MatlabEdit window.
The main purpose of the GUI is to draw ellipses easily in an MRI-slice. To draw an ellipse, first a midpoint is put in the MRI-slice with the mouse. Second,

the mouse should be rotated in such a way that the ellipse gets the right angle and length. If this is done, a mouse-button should be pressed again to store this length (x-radius and angle). At last the width of the ellipse should be determined by moving the mouse and then pressing the mouse-button to store this. For finetuning the ellipse, one can change the ellipse by using the buttons to the right of the window, which are defined as follows.
The top four buttons are used to move the ellipse in the MRI-slice. At each press on the button, the ellipse moves only one pixel.

- `U` This button moves the ellipse up.
- `R` Pressing this button move the ellipse to the right.
- `D` The ellipse moves down.
- `L` The ellipse moves to the left.

The next four buttons are used to resize the shape of the ellipse. If one of those buttons is pressed, the ellipse changes only one pixel in size.

- `x<` The x-radius of the ellipse will become one pixel smaller.
- `x>` The x-radius of the ellipse will become one pixel bigger.
- `y<` The y-radius of the ellipse will become one pixel smaller.
- `y>` The y-radius of the ellipse will become one pixel bigger.

The x-radius is the axis which is first created by clicking the mouse-button for the second time and determines the length of the ellipse. The y-radius is the axis which is created by clicking the mouse-button for the last time and determines the width of the ellipse. The last two buttons `'R<'` and `'R>'` can be used for rotating the ellipse counterclockwise or clockwise. Each time when this button is pressed the ellipse rotates 0.1 radians.
The four buttons `Remove`, `Clear`, `Save` and `Load` are defined as follows:

- `Remove`
  The last ellipse that is drawn on the screen will be removed from the list of ellipses and also from the screen. This is only possible if there is at least drawn one ellipse on the current MRI-slice.
- `Clear`
  All the ellipses that are displayed in the current MRI-slice will be removed.
- `Save`
  The ellipses in the current MRI-slice will be saved to a file `ellipses.dat`. A description of this file will be given in the next section.

- Load
  The ellipse(s) for the current MRI-slice will be read from the file `ellipses.dat`, if they exist.

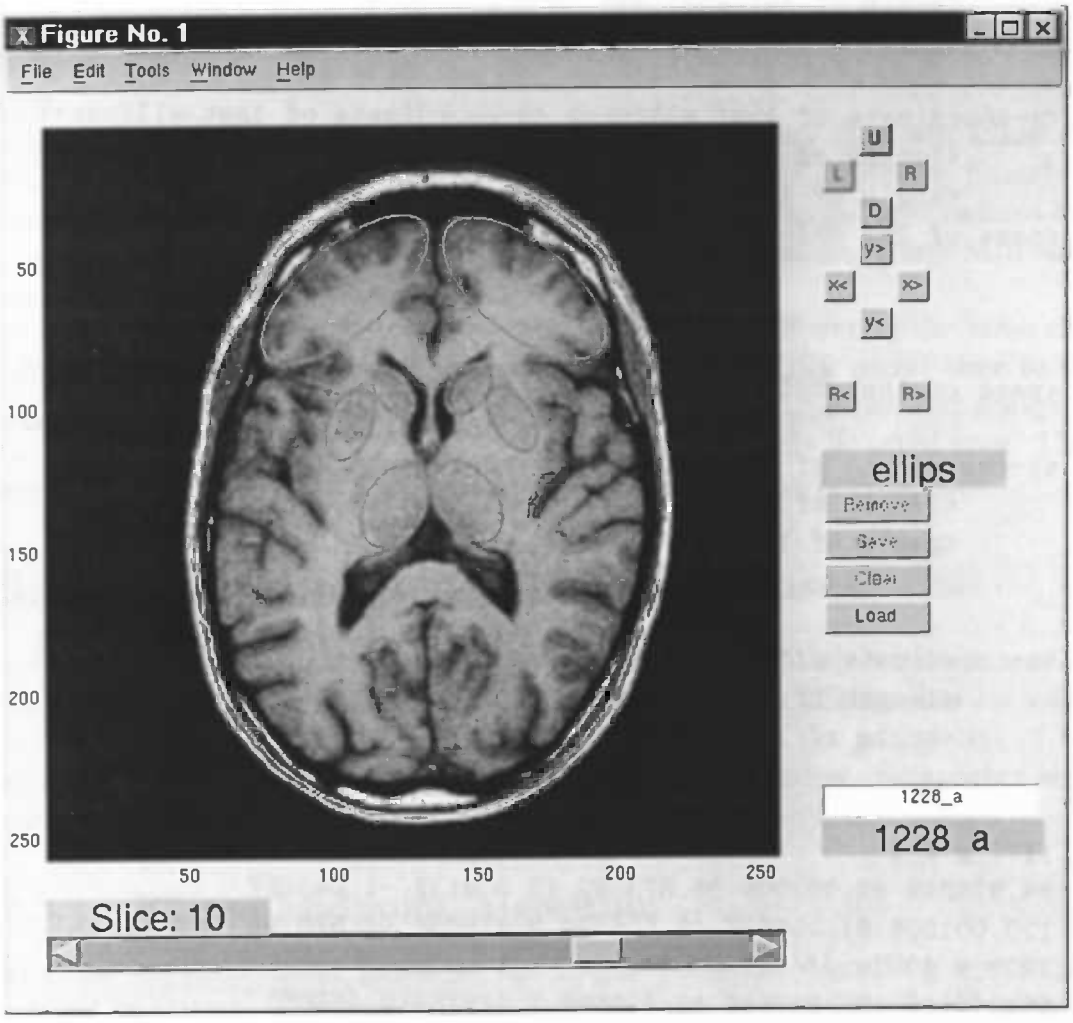

Figure 2.1: Main GUI window

2.2 Saving the ellipses to a datafile

Ellipses that are drawn in the GUI can be saved to a file. They will by default be saved to the file `ellipses.dat`. The structure of the datafile is first described symbolically and then an example is given. The sign \ means that the line continues on the next line. In the file `ellipses.dat` this cannot be used and all must placed in one line.

```
<name of 1st MRI-datafile> <number of slice> \
                                    <number of ellipses on slice>
<x-coordinate of 1st ellipse> <y-coordinate of 1st ellipse> \
      <length of 1st ellipse> <width of 1st ellipse> \
      <angle of 1st ellipse>
   .
   .
<x-coordinate of last ellipse> <y-coordinate of last ellipse>\
      <length of last ellipse> <width of last ellipse> \
      <angle of last ellipse>
<name of 2nd MRI-datafile> <number of slice> \
                                    <number of ellipses on slice>
   .
   .
<name of last MRI-datafile> <number of slice> \
                                    <number of ellipses on slice>
<x-coordinate of 1st ellipse> <y-coordinate of 1st ellipse> \
      <length of 1st ellipse> <width of 1st ellipse> \
      <angle of 1st ellipse>
   .
   .
<x-coordinate of last ellipse> <y-coordinate of last ellipse>\
      <length of last ellipse> <width of last ellipse> \
      <angle of last ellipse>


1177_a 7 2
84.974849 92.002008 36.873352 13.509132 -1.043857
120.001006 81.206827 18.506024 8.479037 -1.570796
1229_a 10 1
66.946680 138.267068 13.356868 2.119759 0.480063
1229_a 9 2
133.393360 225.142570 6.168675 9.285870 -1.570796
128.757545 65.214859 17.535496 20.819425 -0.028506
1228_a 10 6
147.300805 132.098394 15.623982 9.930701 0.991974
121.667002 133.640562 15.388160 10.215197 -1.070604
162.753521 103.311245 12.439524 6.684722 1.050784
106.608652 104.853414 13.369443 7.362768 -1.175291
121.606640 93.572289 9.987617 5.469263 0.967570
147.815895 92.516064 9.882456 6.549453 -0.895075
```

The first line in the datafile contains the basic information about the ellipses, which is as follows.

- First the name of the MRI-datafile.
- Second the number of the slice of the MRI-datafile.
- Finally the number of ellipses that are drawn in the MRI-slice.

The number of ellipses gives the information on how many lines will follow the first line. Each of those lines contains the five parameters of an ellipse. Namely 'x center', 'y center', 'radius x', 'radius y' and 'angle'. The 'angle' is in radians and the other four parameters in pixel values. The next line contains a new MRI-slice, possibly a new MRI-datafile can be used.
In the first implementations no consideration was made of storing the same slice twice. If one did so, this slice was saved for the second (or more) time to the datafile `ellipses.dat`. Loading the data from this datafile resulted always in the last one that is saved. Currently, the same slice can only be saved once. Thus saving the ellipses of this slice again results in deleting the previous one.

2.3 Algorithms used to draw ellipses

To draw ellipses in Matlab an algorithm is needed. In Matlab there is a standard drawing algorithm `RECTANGLE`, which can also draw ellipses, but its major disadvantage is that the ellipses cannot be rotated and the parameters of the ellipse/rectangle must be given in Matlab's MainEditWindow. So another algorithm is implemented to do this.

2.3.1 Ellipse Midpoint Algorithm

First an implementation based on the Ellipse Midpoint Algorithm was implemented in Matlab. The details of this algorithm can be found in [2]. When using this method a new disadvantage appeared, the ellipse is drawn in a template, and this template is put down on a MRI-slice. Every transformation of the ellipse results in a complete update of the figure and this is very irritating, because sometimes one has to transform an ellipse a lot to give it the right position on a MRI-slice. Updating means that the MRI-slice with the template is drawn again.

2.3.2 Ellipse using a polyline

To avoid this redrawing another implementation of an ellipse is used. This implementation makes use of the standard Matlab function `LINE`. The command `LINE(X,Y)` draws a polyline through the points in the arrays `X` and `Y`. `X` contains the x-coordinates and `Y` contains the y-coordinates of the points. A polyline is

nothing more than a sequence of straight line segments. Drawing an ellipse is now approximated by drawing a polyline through a number of points on the ellipse. In most cases a value of 300 points is used to draw an ellipse by a polyline, using an implementation from D.G. Long, Brigham Young University:
`ftp://ftp.mathworks.com/pub/contrib/v5/graphics/ellipse.m`
and is based on the original Matlab-file `circles.m`.
In the original implementation of D.G. Long, the function `ellipse.m` draws directly an ellipse as a polyline in the current figure. This function returns an object handle of the polyline when it is assigned to a variable. Every time when this function is called a new polyline is drawn with another object handle. In our implementation we have adapted it so that only the arrays `X` and `Y` by which the ellipse is defined are returned. This is done because it is now possible to control the data of the ellipse in another environment. The function `LINE` with the parameters `X` and `Y` is called in Matlab only for the first time. When this ellipse will be transformed, only the `X` and `Y` data should be assigned to this object handle without calling the function `LINE`. In Matlab this can be done in this way.

```
set(object handle of the ellipse, 'XData', X, 'YData', Y);
```

Using the function `LINE` has also another advantage compared to the midpoint algorithm. As mentioned earlier, drawing or redrawing the ellipse using the midpoint algorithm causes flickering, because the whole MRI-slice is redrawn. However in the implementation used now the ellipse is an object and no redrawing of the MRI-slice is necessary when transforming or drawing the ellipse. Only the changed points are updated, so no flicker in the figure is seen anymore.
A `LINE` has also the advantage that it uses its own dimensions/pixelsize, which is much finer than the image resolution. Besides, this kind of ellipses leaves the MRI-slice nearly undisturbed. In figure 2.1 some of these ellipses are visible.
The syntax of the ellipse that is used now is as follows.
`[LineX,LineY]=ellipse(ra,rb,ang,x0,y0,C,Nb)`, where

- `ra` is the radius of the semimajor axis of the ellipse.
- `rb` is the radius of the semiminor axis of the ellipse.
- `ang` is the angle by which the ellipse is rotated.
- `(x0,y0)` is the point where the ellipse is centered.
- `C` is the color of the ellipse, but isn't used.
- `Nb` specifies the number of points used to define the ellipse.

Because the shapes of some brain structures are much more complicated than ellipses, additional information or algorithms may be needed to determine the form of those structures.

Chapter 3

Snake algorithm

As described above, an additional algorithm is needed to extract the Regions Of Interest from the MRI-slices. A method which can extract these ROI's is the snake algorithm [3]. A snake is a deformable model for finding the contour of an object. Imagine a snake as an elastic band, which is placed around the object and then contracts until it fits exactly around the object. Another possibility is to place the snake inside the object and let it expand. The snake is represented by a certain number of elements, called *snaxels*.

3.1 Method

The idea behind a snake is that there are several energies making it move. In an original article by Kass, Witkin and Terzopoulos [1] the total energy is computed as follows

$$E_{snake} = \int_0^1 [E_{int}(v(s)) + E_{image}(v(s)) + E_{con}(v(s))]ds, \tag{3.1}$$

where E_{int} represents the internal energy of the contour due to bending or discontinuities, E_{image} represents the image forces and E_{con} the external constraints. The function $v(s)$ denotes the snaxel at arc length s from the 'beginning' of the snake.
Williams and Shah have come up with a greedy algorithm for minimizing the energy E_{snake} as described in [4]. The algorithm is not guaranteed to give a global minimum, but according to the authors the experimental results were comparable to other methods. Furthermore the greedy version is much faster than the original as it determines the minimum in a neighbourhood with a size of m, the complexities are $O(nm)$ versus $O(nm^3)$, where n is the number of snaxels.
In this algorithm the total energy

$$E = \int (\alpha(s)E_{cont} + \beta(s)E_{curv} + \gamma(s)E_{image})ds \tag{3.2}$$

should be minimised.
The first two terms correspond to E_{int} in equation 3.1 and E_{con} has been omitted. The parameters α, β and γ are used to balance the relative influence of the three terms. The function of the term E_{cont} is to keep the snaxels evenly distributed along the snake. The algorithm uses the difference between the average distance between points, $\vec{d}$, and the distance between the two points under consideration: $\vec{d} - |v_i - v_{i-1}|$. Here v_i defines snaxel i at position (x_i, y_i). In other words E_{cont} is the deviation of the distance between two points to the average distance. Keeping this energy minimal is equal to keeping the deviation minimal which means that the elements of the snake are evenly spaced.
The second term in equation 3.2 is curvature. Since E_{cont} causes the snaxels to be evenly spaced, it is possible to approximate the curvature in a point by the second derivative. This means that E_{curv} is defined by $|v_{i-1} - 2v_i + v_{i+1}|^2$. Naturally it is only an estimation of the curvature, but it is close enough and this definition is computationally much more efficient than more precise methods.
The last term, E_{image}, is the image force. This is represented by the gradient magnitude, normalised as defined by equation 3.3. Simply dividing the value by 255 will not emphasize differences between gradient magnitude enough. For example, the difference between a gradient magnitude of 240 and one of 255 is significant, but this is not clear from their normalised values. Therefore, given the magnitude at a point (mag) and the minimum (min) and maximum (max) in its neighbourhood, the normalised value is calculated as follows

$$(min - mag)/(max - min). \tag{3.3}$$

This definition does, however, give problems in areas where the gradient magnitude varies little. In such cases the difference between the minimum and maximum values is very small, leading to a relatively large difference between in normalised values even though there is not an edge in sight. Williams and Shah propose to artificially set a minimum of 5 for the denominator. This is done by changing min to max -5 if the difference between the minimum and maximum is less than 5. Their example considers a neighbourhood with all points having values 47, 48 or 49. With the standard normalisation procedure (dividing by 255) these points would have normalised values 0, -0.5 and -1. Using equation 3, these values would be -0.6, -0.8 and -1, which gives a more accurate description of the uniformity of the area.
All three energies are normalised, since they all add to the total energy and therefore need to be within the same range.
The algorithm consists of a loop over the elements in the snake, for each element determining whether to move it or not. This process is repeated until less than a certain number of elements is moved.
To determine a new position for the snaxel s, the three energies are computed for every point in the neighbourhood of the snaxel. Using these energies, a total

energy is calculated for each neighbour n as follows

$$E_n = \alpha_s E_{cont,n} + \beta_s E_{curv,n} + \gamma_s E_{image,n}. \tag{3.4}$$

The position corresponding to the minimum of these energies is the new position. If this is a different point from the current position of the snaxel, the snaxel is moved and the variable, which denotes the number of snaxels moved, is increased. Next, two for-loops are included to determine whether to relax the curvature for the snaxel. This is necessary to allow sharp corners. The method for computing this curvature is different from the one used before. That calculation is performed many times and thus needs to be computationally efficient. Since the snaxels are relatively evenly spaced, this formula is a reasonable estimation of the curvature. The method used in this case is related to the angle between the vectors and is defined by

$$c_i = |\frac{\vec{u}_{i+1}}{|\vec{u}_{i+1}|} - \frac{\vec{u}_i}{|\vec{u}_i|}|^2, \tag{3.5}$$

where $\vec{u}_i = (x_i - x_{i-1}, y_i - y_{i-1})$. The value c_i is related to the angle θ between the incoming and outgoing edges of snaxel v_i in the following manner: $(2\sin(\theta/2))^2$,with $0 \leq \theta \leq \pi$. Therefore it is easier to define a threshold for this angle.
The curvature of the snaxel needs to be larger than that of the neighbouring snaxels. This way only the curvature of the true corner is relaxed. Furthermore the curvature must exceed a certain threshold. Also the gradient magnitude must be large to ensure that the corner is near an edge.

3.2 Implementation of the snake algorithm

The implementation of the snake algorithm was originally done in SCILIMAGE by Pluim and Westenberg. See their report for more details [3]. Because the analysis of the MRI-scans is performed in Matlab 5.3, the implementation of the snake algorithm is adjusted to a so Matlab Applications interface (API) MEXC-file. An application interface program can be written in 'C' and can access Matlab internal matrix structures. This MEXC-file has additional functions which are necessary to interact with Matlab. This MEXC-file contains its own specific datastructures as in SCILIMAGE. To call a MEXC-file in Matlab it must contain the function `mexFunction` and has the next syntax: `void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])`, where

- `int nlhs` is the number of output arguments.
- `mxArray *plhs[]` contains the output variables.
- `int nrhs` is the number of input arguments.

- `const mxArray *prhs[]` contains the input variables.

The datastructures in Matlab and C are represented differently from each other, so it is necessary to transform some of these datastructures to the other. The first element of the fourth argument `prhs[0]` of the snake algorithm (`mexFunction`) is given an input image containing an MRI-slice. This image is in Matlab represented as a 2D-array with the dimensions 256*256 pixels, where the first pixel is at coordinate (1,1) (By the way: In Matlab the coordinates are represented by row first and then column).
In the MEXC-file used here, this 2D-array is defined as a pointer. Defining the 2D-array simply as an array structure is not possible, because the dimensions of the image must then be known at compile-time. In most cases this is 256 by 256, but sometimes there could be exceptions. To avoid this a pointer-type is used, because this has the advantage that the size of it can be determined at run-time using `malloc`. The `malloc()` function obtains a block of memory, which is not separated by something else in that specific memory. Now there is allocated a block of memory, it is possible to store the data from Matlab that is necessary for the snake algorithm in it. The data (MRI-slice) in Matlab is stored as a 2D-array and must be stored as a 1D-array in C. The data will be stored row by row in this 1D array, see figure 3.1 for the details. With the MEXC-function `mxGetPr` it

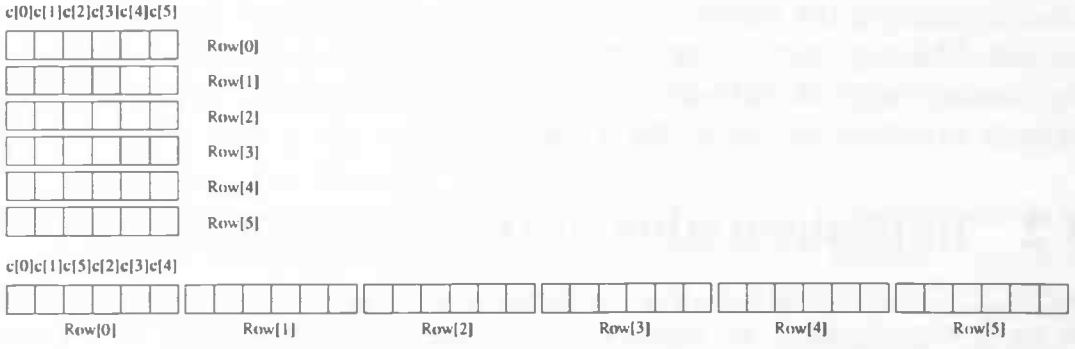

Figure 3.1: Top: 2D-array representation. Bottom: 1D-array representation.

is possible to access the pointer to the first element of the array in Matlab. This array is stored as a 1D-array in Matlab's memory, column by column. With the next C-code it can be transformed to a 1D-array in C, row by row.

```
m=mxGetM(prhs[0]);  /* get the dimension of the row */
n=mxGetN(prhs[0]);  /* get the dimension of the column */
in=malloc(sizeof(double)*m*n); /* allocation of array */

for (row=0; row < m; row++)     /* m is the dimension of a row */
  for (col=0; col < n; col++)
                              /* n is the dimension of a column */
```

```
    {
      /* prhs[0] is the first input argument */
      *(in+n*row+col)=(mxGetPr(prhs[0]))[row+col*n];
    }
```

Accessing data in a 1D-array has the problem that it cannot be accessed like a 2D-array through just giving the coordinate points of the row and column. To access the data it needs the same construction as transforming the data, using the dimensions of the row and column. Thus accesing the data at point (x, y) goes as $*(\mathbf{array} + \mathbf{length(row)} * x + y)$, where **array** is the pointer to the first element of the data.
After the snake algorithm is applied to the input image, an output image is returned as result. Of course this is also a 1D-array, row by row, which must be transformed to a 1D-array, column by column in Matlab's memory, so that it can be accessed as a 2D-array in Matlab itself. To do this a 2D-array is created in Matlab using the MEXC-function `MxCreateDoubleMatrix`, which has three parameters defining the dimension of the row and column and the type of the data to be `REAL` or `COMPLEX`. The dimensions are the same as the input image, and the image contains only `REAL` values. Besides defining the array, it must also be allocated in Matlab's memory using the MEXC-function `mxCalloc`. At last the data from the returned output image must be transformed to the array in Matlab. It is placed in Matlab's memory in a 1D-array and column by column. The 2D-array in Matlab is finally assigned to the first output variable `plhs[0]` in the `MexFunction`. The next piece of C-code makes the above clear.

```
plhs[0]=mxCreateDoubleMatrix(m,n,mxREAL);
start_of_real_data=
            (double *)mxCalloc(m*n, mxGetElementSize(plhs[0]));
next_data=start_of_real_data;

for (row=0; row < m; row++)
  for (col=0; col < n; col++)
    {
      *next_data++=*(out+n*col+row);
    }
mxSetPr(plhs[0], start_of_real_data);
```

In the beginning the snake algorithm was especially designed for `SCILIMAGE`, but for using it in Matlab there were some aspects which could not be used. In `SCILIMAGE` the main function `compute_snakes` had only thirteen arguments. The syntax of this function was as follows.

```
int compute_snakes(IMAGE *in, IMAGE *out, double t1, double t2,
                   double gamma, int snake_type, int init_type,
```

```
double maxdev, int neighbourhood,
int nr_snakes, int nr_init, int nr_its,
double des_length)
```

The parameters `in` and `out` are simply the input and output images. To control the edge relaxation there are two kinds of thresholds, namely `t1` for the gradient magnitude and `t2` for the angle. As mentioned earlier, the variables α, β and γ are used in computing the total energy, only the γ is included to control, because it is the main interesting one according to [3]. The other two are also used, but cannot be changed by the user.
The next parameter, `snake_type`, represents the direction of the inflation force. The value 1 will give a contracting snake, and the value 0 an expanding one.
The parameter `init_type` denotes the initialisation type for the snakes. When a value is chosen it should be the same for all snakes. The value 1 means that the user can define some points of the initial contour. A value 0 means that the user can define one point as the center of a circle with a certain radius. This will then be the initial contour of the snake. Further the threshold `maxdev` is used, this is a measure for the deviation of the gradient. The three parameters `neighbourhood`, `nr_snakes` and `nr_init` are used to control the neighbourhood diameter, the number of snakes and the number of points to initialise the snakes. The diameter of the neighbourhood is usually a small value, because larger values increase the chance of the snake moving to another brain structure in the MRI-slice. Sometimes the snake algorithm does not end, so the variable `nr_its` is used to define the maximum number of iterations that the algorithm may perform. Finally there is the parameter `des_length`, which is the desired length of the edge between two snaxels.
The function `compute_snakes` is also used in Matlab, but it is called inside the `mexFunction`. During the tests to examine the results it was decided to add some more parameters to the function `compute_snakes`. In `SCILIMAGE` it was possible to click on the image to define the points of a contour or a mid-point of an initializing circle, but to do this in the `MEXC-file` is not possible. To avoid this, of the coordinates of the points are defined in Matlab and stored in an array and this array is given as a parameter to the `mexFunction`. And in this function the array is given as a parameter to the function `compute_snakes`.
Another set of parameters that was added to the function `compute_snakes` are the height and width of the input image. In `SCILIMAGE` there is a particular datastructure that contains the image, and this datastructure also has the possibility to determine the dimensions of the image. Because this datastructure is removed from the `MEXC-file`, and a pointer-type is used now to define an image, it is no longer possible to extract the dimensions from this data-type. So the dimensions are now given as parameters to `compute_snakes`. The parameter `out` only returns an image with the contours of the final snakes and the pixel values within these snakes are set to 255, which means that the area within a snake is

filled. But besides the parameter out another parameter is added, which contains the coordinates of the snaxels of these snakes. This is necessary, because the coordinates are used in Matlab to draw the snakes as polylines, which has a greater advantage (see Chapter 2) than adding the template out on the MRI-slice. This makes it easier for the user to see if a snake is located around a specific brain structure.
During the tests it became also clear that another variable should also be given as parameter of `compute_snakes` to control it. This variable `alpha` determines how much the energy E_{cont} weighs in the calculation of the total energy (eq(3.2)), which should be minimised. The new syntax of `compute_snakes` is as follows

```
int compute_snakes(double *in, double *out, double t1, double t2,
                   double alpha, double gamma, int snake_type,
                   int init_type, double maxdev,int neighbourhood,
                   int nr_snakes, int nr_init, int nr_its,
                   double des_length, int height, int width,
                   Snake *s, int *points
                   ),
```

where (only the added variables are mentioned)

- `double alpha` is the variable to control the weight of the energy E_{cont}.
- `int height` The height of the input image.
- `int width` The width of the input image.
- `Snake *s` contains the coordinates of the final snakes.
- `int *points` This parameter has the coordinates of the contours of the initial snakes.

3.3 GUI

To make it easier for the user a Graphical User Interface is implemented. The interface for the snake algorithm is added to the GUI that was described earlier to draw ellipses on an MRI-slice. This GUI from which a screenshot is displayed in figure 3.2 can be started in Matlab by entering the command `snakedrawing`. From the figure it is clear that there are made some changes in comparison with the implementation of the GUI for drawing ellipses only. The window is made a little wider so that there can be placed some more buttons and textedit boxes. Right in the middle of the figure we see the button `Load Snake`, this button is used to load a template which contains some contours. When the snake algorithm is called, these contours are used for the initialisation of the snakes. Which template is loaded depends on the value that is entered in the textedit

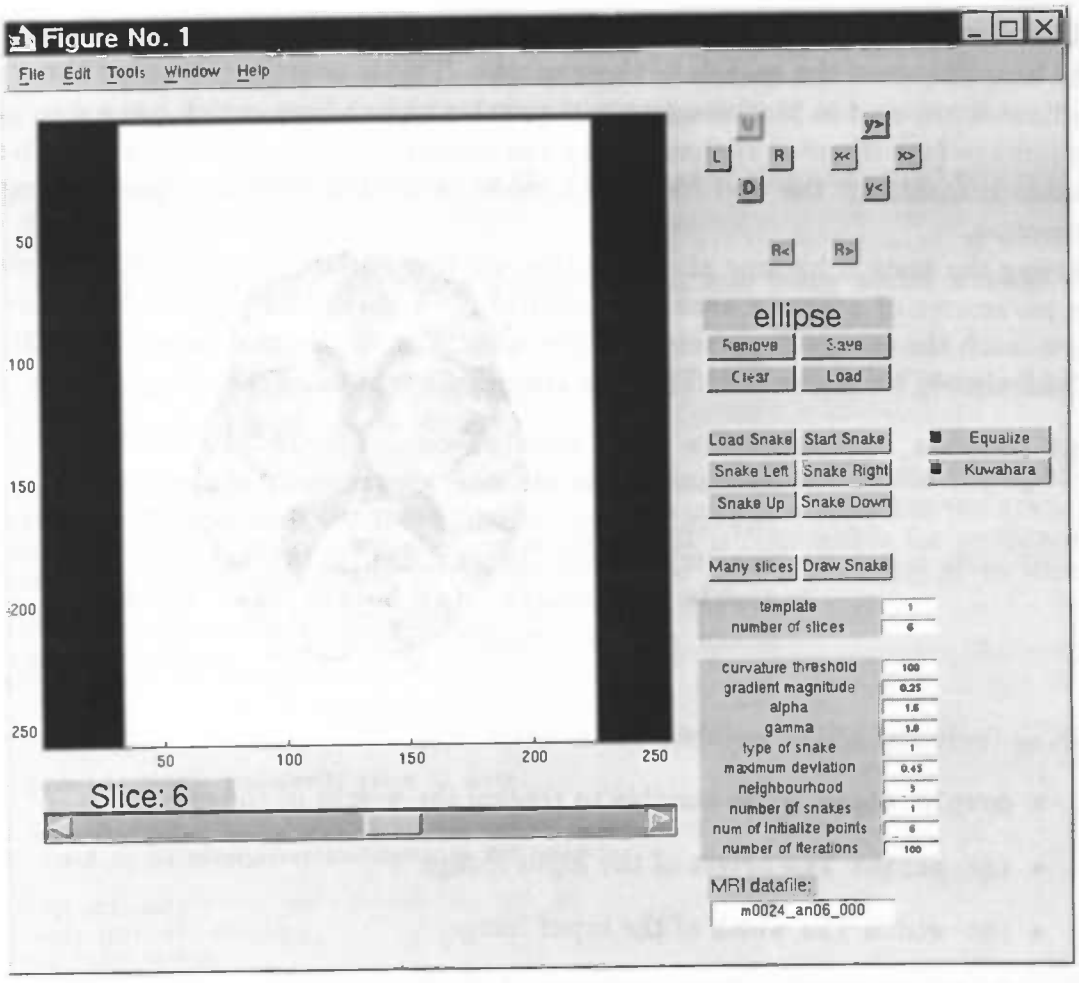

Figure 3.2: Graphical User Interface for the snake algorithm.

box behind the text `template`. A value that is out of range will give an error in Matlab's MainEditWindow, and no template is loaded. The template that is needed will be loaded from the file `snaketemplates.dat` and its syntax is as follows.

```
<number of template> <number of snakes>
<number of points of first snake>
<x-coordinate of first point> <y-coordinate of first point>
<x-coordinate of second point> <y-coordinate of second point>
   .
   .
   .
<x-coordinate of last point of first snake> <y-coordinate of last
                                              point of first snake>
<number of points of second snake>
```

```
<number of points of last snake>
<x-coordinate of first point> <y-coordinate of first point>
<x-coordinate of second point> <y-coordinate of second point>

<x-coordinate of last point of last snake> <y-coordinate of last
                                                point of last snake>
```

Thus first the number of the template is given, this is the same number that should be entered in the textedit box behind the text `template`. The next number is the number of contours that are drawn in that template and which are used for the initialisation of the snakes. On the next line the number of points whereby a contour is described will be mentioned. On each line that follows this line the coordinate of a point of the contour is given. After the last line with a point of this contour, the next line contains again the number of points describing the contour, but this could also be the next number of a template or nothing at all. This depends on how many contours there are in a template, and how many templates there are.
With the button `Start Snake` the snake algorithm will be applied to the current MRI-slice with the current template. As earlier mentioned, the snake algorithm needs some parameters to call it. The values for these parameters can be entered in the textedit box which are displayed after the text box `template`. A brief description of these textedit boxes is as follows

- `curvature threshold`: threshold for the curvature.
- `gradient magnitude`: threshold for the gradient magnitude.
- `alpha`: defines the influence of the contour on the total energy. This makes the snaxels evenly distributed along the snake.
- `gamma`: defines the influence of the gradient on the total energy.
- `type of snake`: Only two values are valid, 0 for an expanding and 1 for a shrinking one.
- `maximum deviation`: determines the deviation of the gradients.
- `neighbourhood`: the diameter of the window in which a snaxel is moved.

The next two parameters are only used when the user wants to draw its own template instead of loading one from the datafile `snaketemplates.dat` as described

earlier. The values of these parameters must be entered before clicking on the button `Draw Snake`, otherwise the default or last values are used. The default values are displayed when the GUI is started in Matlab.

- `number of snakes` determines the number of snakes that will be drawn in the MRI-slice.
- `num of initialize points` determines out how many points a snake exists.
- `number of iterations` determines the maximum number of iterations the snake algorithm may perform.

Not all the parameters that are necessary for the snake algorithm are displayed in the GUI. In most cases the value is just a constant or is determined automatically. The subfunction in the Matlab-file `snakes.m` that is responsible for performing the snake algorithm is `SStart`. The details of this piece of code are given here.

```
%----------------------------------------
% Subfunction SStart
%----------------------------------------

case 'SStart'

  global objHandle snakehandle slice t2 inits;

% Determination of the indexes, where the MRI-slices are
% stored in the image_array containing an MRI-datafile.

  i=[1:nSlices];
  start_inx(i)=(i-1)*dimy+1;
  end_inx(i)=i*dimy;

% Using colormap jet for seeing better details in the MRI-images

  colormap(jet);

% Extracting the current MRI-slice from the image_array

  newplots= rot90(image_array(:,(start_inx(slice):end_inx(slice))),1);

% Depending on the checkboxes in the GUI the next two operation
% are used for image-preprocessing for better results.

  if (get(findobj(gcbf,'Tag','SEqualize'),'Value')==1)
     newplots= (histeq(newplots./max(max(newplots)))).*max(max(newplots));
  end;
  if (get(findobj(gcbf,'Tag','SKuwahara'),'Value')==1)
     newplots=round((kh(newplots,5)));
  end;

% The different values for the parameters of the snake algorithm
% are read from the GUI.

  curvature=str2num((get(findobj(gcbf,'Tag','SCurv'),'String')));
  gradient=str2num((get(findobj(gcbf,'Tag','SGradMag'),'String')));
  alpha=str2num((get(findobj(gcbf,'Tag','SAlpha'),'String')));
  gamma=str2num((get(findobj(gcbf,'Tag','SGamma'),'String')));
  snaketype=str2num((get(findobj(gcbf,'Tag','SType'),'String')));
  maxdev=str2num((get(findobj(gcbf,'Tag','SMaxDev'),'String')));
```

```
  nbh=str2num((get(findobj(gcbf,'Tag','SNeighbourhood'),'String')));
  numofsnakes=str2num((get(findobj(gcbf,'Tag','SNuberOfSnakes'),'String')));
  numofinits=str2num((get(findobj(gcbf,'Tag','SNumberOfInits'),'String')));
  iterations=str2num((get(findobj(gcbf,'Tag','SIterations'),'String')));


% The next loop performs the call to the MEXC-file snake.c
% which calculates the snakes by the current template.
% This is done for all snakes in the template, one snake at
% a time.

  for i=0:length(inits)-1
    [y,t1]=snake(newplots,curvature,gradient,alpha,gamma,
                 snaketype,1,maxdev,nbh,1,inits(i+1),
                 iterations,4.0,t2(1+2*i:2+2*i,1:inits(i+1)));

% resulting snakes are stored in variable s1 and number of
% snaxels of a snake is stored in inits2

     s1(1+2*i,1:length(t1))=t1(1,:);
     s1(2+2*i,1:length(t1))=t1(2,:);
     inits2(i+1)=length(t1);
     clear y t1;
  end;

% replace the initial contours by the resulting snakes
% of the snake algorithm

  for i=0:length(snakehandle)-1
    set([snakehandle(i+1)], 'XData', [s1(1+2*i,1:inits2(i+1))],
                            'YData', [s1(2+2*i,1:inits2(i+1))]);
  end;

% clear temporary variables
  clear newplots s1 inits2 i;
```

The four buttons Snake Left, Snake Right, Snake Up and Snake Down right below the buttons Load Snake and Start Snake are used to move the current template with the contours to the left, right, upwards or downwards position. Each time when one of these button is pressed the template moves just on pixel size. With the button Draw Snake it is possible for the user to draw a number of snakes in the MRI-slice and to use this as a template. When the button is pressed the values in the textedit boxes behind number of snakes and num of initialize points are used to determine how many snakes the user intends to draw and of how many points a snake exists. By clicking on the MRI-slice the user defines the points of a contour. The contours will be visible when all points are entered, thus clicking (number of snakes)*(num of initialize points) times in the MRI-slice.
The button Draw Snake takes immediately care of postponing the drawing of ellipses. Normally, when there is clicked on the MRI-slice the GUI starts drawing ellipses, but clicking on the button Draw Snake deactivates this, such that the user can draw the contours. When the last point of the last contour is set it is again possible to draw ellipses.
Left of the button Draw Snake we see the button Many slices, this button is almost the same as the button Start Snake, but this performs the snake algo-

rithm on more MRI-slices, starting with the slice which is currently displayed. The number of slices to which the snake algorithm is applied depends on the number that is entered in the textedit box behind the text `number of slices` in the GUI. The templates for these slices are loaded from the file `snaketemplates.dat` in increasing order, starting by the number that is entered in the textedit box behind the text `template`. Thus each slice gets another template for the snake algorithm. If the same template is needed for different slices it should be defined more than once in the file `snaketemplates.dat`. After the snake algorithm is performed the result will be a window where the output images with the filled regions are displayed.
Finally there are two checkboxes, which are not explained yet. These checkboxes are used to give the current MRI-slice some preprocessing. This takes places when the `Start Snake` or `Many slices` button is pressed and the snake algorithm is performed. The input image for this algorithm will first undergo this preprocessing if one of the checkboxes is activated. If the first checkbox `equalization` is activated then there will be applied a histogram equalization to the current MRI-slice, else there happens nothing at all. Activating the second checkbox `kuwahara` will apply an edge-preserving filter to the MRI-slice. A detailed description of this filter can be found in the next section.
If the user activates both checkboxes the MRI-slice will be first equalized and hereafter the equalized image will be applied by the Kuwahara filter. During some experiments it became clear that the other possibility, where the Kuwahara filter is performed before the histogram equalization, gives a result that is not well for the snake algorithm. The homogenous areas that are formed by the Kuwahara filter are then changed in non homogenous areas. In the figures 3.3 and 3.4 below it is possible to see the differences between the two methods. In figure 3.4 the brain structures in concerning have almost the same grey values as the structures around it. This is especially the case for the Caudate Head. Equalizing before the Kuwahara filter is done to keep the results of the Kuwahara filter intact.

3.4 Kuwahara filter

In the beginning of this report it was described that the equalization is used to give more detailed structures in the brain. Trying to get better results/images some image preprocessing filters/algorithms are applied to the data. Edges play an important role in the perception of images as well as in the analysis of images. As such it is important to be able to smooth images without disturbing the sharpness of edges and, if possible, the position of edges. A filter that accomplishes this goal is termed an edge-preserving filter and one particular example is the Kuwahara filter. This Kuwahara filter tries to make a more homogeneous image of the original one, but keeps the edges sharp. In the next steps the structure of this

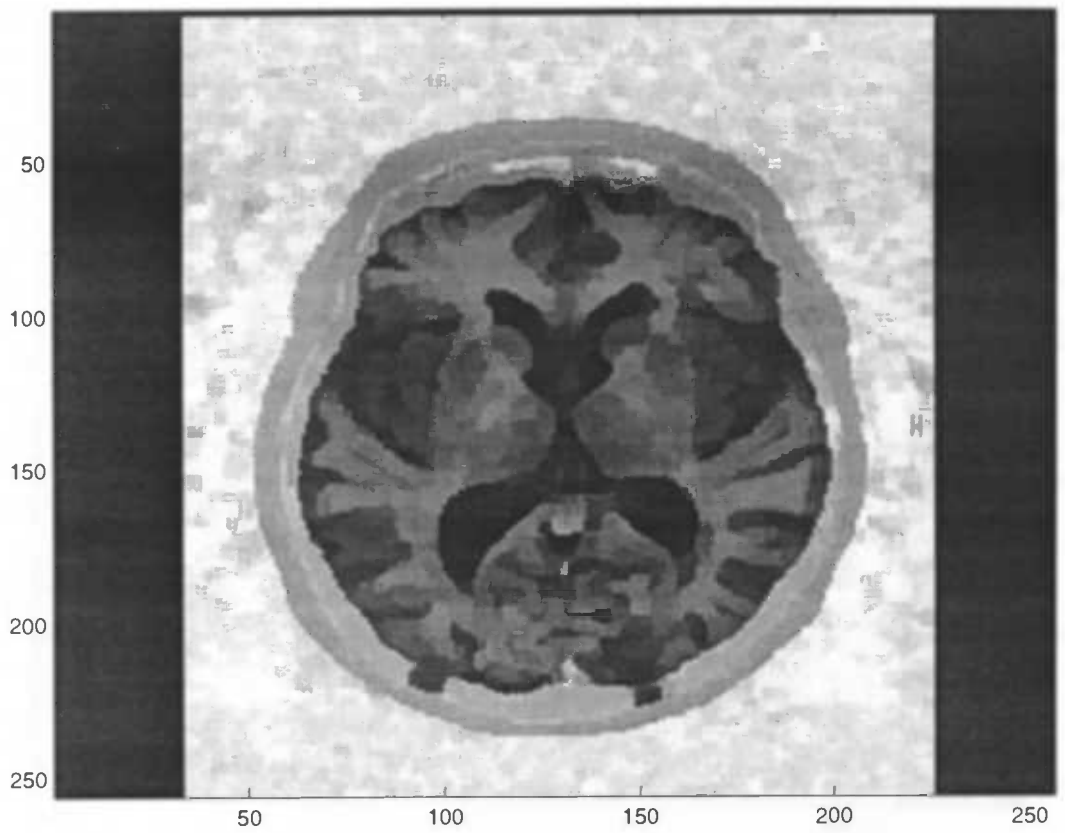

Figure 3.3: Equalizing before Kuwahara filter.

filter and how it precisely works will be described.

3.4.1 Description of the Kuwahara filter

The Kuwahara filter can be implemented for a variety of different window shapes, although the algorithm will be described for a square filter window of size $j = k = 4n + 1$, where n is an integer. This window is then partitioned in four equal square regions of size $((j + 1)/2) * ((k + 1)/2)$ as shown here below.

$$\begin{pmatrix} A & AB & B \\ AC & ABCD & BD \\ C & CD & D \end{pmatrix}$$

Looking to the four regions A, B, C and D above one sees that they overlaying each other. The four regions overlay only at one column and one row in the center of the filter window. In the center the four regions overlay altogether by one value. For all those four regions the variance and the mean is calculated. The

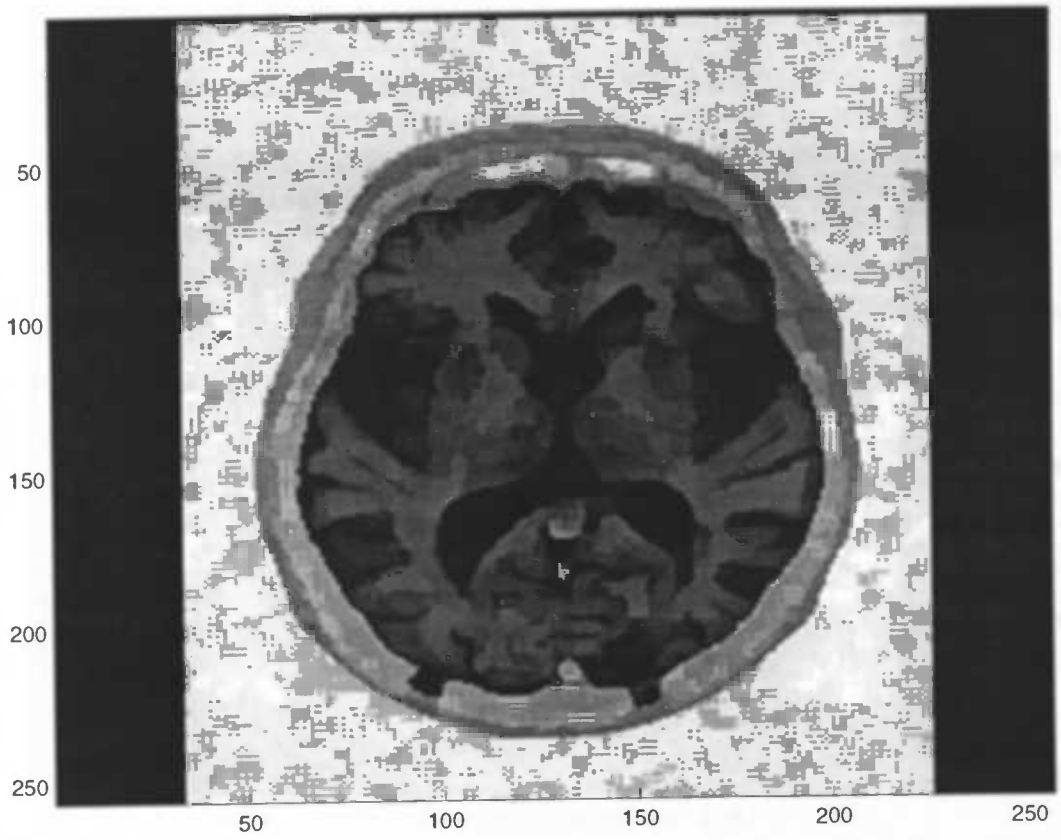

Figure 3.4: Kuwahara filter before equalizing.

variance formula gives us a measure of spread, the larger the number the greater the spread. The formula for the variance is as follows

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\vec{x_i} - \vec{x})^2$$

At first glance the variance formula seems a bit overwhelming and mysterious but the symbols are as follows: s^2 is the variance, n is the size of the data, i is the $\vec{x_i}$'th data element, and $\vec{x}$ is the mean. The mean is calculated as

$$\vec{x} = \frac{\sum_{i=1}^{n}\vec{x_i}}{n}$$

After calculating the variance and the mean in all four regions the output value of the center pixel in the window of the Kuwahara filter is the mean value of that region that has the smallest variance.

3.4.2 Implementation of the Kuwahara filter

The implementation of the Kuwahara filter was first done in Matlab using a `.m`-file. But when the `.m`-file was used it took a long time before the result was ready.

So we came to the decision to implement it also in a MEXC-file. This MEXC-file must also contain the function `MexFunction` to interact with Matlab, as described for the snake algorithm in section 3.2. The `kuwahara` function is called with two input and one output parameters. The first input parameter contains the image to which the Kuwahara filter is applied. The second input parameter determines the size of the Kuwahara filter that is performed. The output parameter contains the image after the Kuwahara filter is applied.
In the body of the `MexFunction` the input and output arrays should be transformed to the right sizes just as done by the snake algorithm.
Here we notice that the Kuwahara filter is not defined for the borders. This results in an output image where the borders are filled with zeros. The syntax of the function `kuwahara` in the MEXC-file is

```
void kuwahara(double *src, double *dst, int m, int n,
              int filter_size)
```

where

- `double *src` is the input image.
- `double *dst` is the output image.
- `int m` is the height of the input image.
- `int n` is the width of the input image.
- `int filter_size` is the size of the Kuwahara filter.

In the next piece of code we see how the filter is applied to the image.

```
void kuwahara(double *src, double *dst, int m, int n,
              int filter_size)
{
  int k,l,row,col;
  double sum[4], mean[4], var[4], vartmp;
  double *array_ptr, *tmp_array_ptr;
  int block_size= filter_size/2;

  array_ptr=malloc(sizeof(double)*sqr(block_size+1));

  /* For each point in the image without the white borders
      the following is calculated */

  for (row=block_size;row<=m-block_size;row++) {
    for (col=block_size;col<=n-block_size;col++) {
```

```
/* area A of Kuwahara filter:
   mean and variance */
tmp_array_ptr=array_ptr;
sum[0]=0;
for (k=block_size;k>=0;k--)
  for (l=block_size;l>=0;l--)
    {
      *tmp_array_ptr=*(src+(row-k)*m+col-l);
      sum[0]+=*tmp_array_ptr++;
    }
mean[0]=sum[0]/9;

tmp_array_ptr=array_ptr;
var[0]=0;
for (k=0;k<(int)sqr(block_size+1);k++)
  {
     var[0]+=sqr(*tmp_array_ptr++-mean[0]);
  }
var[0]/=8;

/* area B */
/* likewise, only indexes change to 1 */

/* area C */
/* likewise, only indexes change to 2 */

/* area D */
/* likewise, only indexes change to 3 */

/* calculation value of output image
   searching of minimum variance and
   assigning that mean to the output image */

if (var[0]<var[1]) {
  vartmp=var[0];
  *(dst+row*m+col)=mean[0];
}
else
{
  vartmp=var[1];
  *(dst+row*m+col)=mean[1];
}
if (var[2]<vartmp) {
```

```
            vartmp=var[2];
            *(dst+row*m+col)=mean[2];
          }
          if (var[3]<vartmp) {
            *(dst+row*m+col)=mean[3];
          }
        }
      }
      free(array_ptr);
}
```

From this code it is clear that there are no considerations made about the borders in the final output image, because the filter in not applied to the whole input image.

Chapter 4

Tests and results

In this chapter we will discuss the results and the tests that are performed on the MRI-data that is acquired at the AZG. At first the snake algorithm was only implemented in the package SCILIMAGE. To use this package with the MRI-data, this MRI-data must first be prepared so that it can be read in SCILIMAGE. In Matlab it is possible to save an image to different filetypes, and the one that is used here to save some MRI-slices is the TIFF-format, because this type is also available in SCILIMAGE.
In SCILIMAGE it is not possible to read the whole MRI-data at once. To read more slices, the MRI-data must be separated over different files, each containing one slice.
When we just started to apply the snake algorithm on the MRI-data only a few MRI-data files were available. After reading an MRI-datafile in Matlab using the function `readanatmri` the function `Imwrite` is used to save a slice as a file. This function is called in the following way.
`imwrite(image,'filename.tif','Tiff','Compression','None')` where

- `image` is the name of the image that needs to be saved.
- `'filename.tif'` is the name of the file in which the image is saved.
- `'Tiff` is the format of the image, a tiff image in this case.
- `'Compression'` and `'None'` No compression is applied on the image that is saved. These parameters are only specific for a tiff image.

Only the slices with the structures of concern in the brain are analyzed and considered. In the beginning the contour of a structure, used as input for the snake algorithm, is set in an image by clicking some points in that image.
As mentioned earlier the snake algorithm needs also a lot of parameters to control the growth and form of the snake. Using the snake algorithm for the first time, the default values for the different parameters are used. The default values are defined in the next table.

Parameter	Value
curvature threshold	100
gradient magnitude	0.25
alpha	1.0
gamma	1.2
type of snake	1
maximum deviation	0.35
neighbourhood	3
number of snakes	1
number of initialize points	6
number of iterations	100

Table 4.1: Default values for the snake algorithm

To get the best results from the snake algorithm it is necessary to change the parameters often and to try different combinations of these values. Only the parameters that give the snake another form are worth changing. The MRI-data to which the snake algorithm was first applied is MRI-dataset 1229_a which was also used in the beginning of this report for thresholding. The structures that are first analyzed are the 'Caudate head' and the 'Thalamus', because these structures are the easiest to find and the borders are the most clear. The 'Putamen' which is also analyzed in a later stage is not always visible or it is not clear at all, compared to the other structures. The slices 11 and 12 are interesting in particular, because these slices have the above mentioned structures. In the next two figures (4.1 and 4.2) slice 11 is displayed in the original state with a contour around the Left Caudate Head, also the slice is displayed after the snake algorithm is applied to this slice with the indicated contour. By the way, the left side of the brain is displayed in the right side of the image. Thus the Left Caudate Head is located on the right side of the image.

When trying to change the parameters it became clear that a value of 0.45 for the maximum deviation often gives a much better result than the default value of 0.35. Also greater values have been used, but this gives not always a satisfactory result. The snake grows for larger values of maximum deviation to other structures, and does not fit the brain structure that we liked to fit.

The curvature threshold was also changed many times but this gave in all experiments that were performed almost no changes.

To see the differences between the images the snake algorithm is extended by one parameter. This parameter indicates if the coordinate points of the contour should be read from a file or, in the old manner, by just clicking some points in the image. Reading the contour from a file makes it possible to use always the same contour on the different images, so that we can compare the effects of the different parameters for the snake algorithm in the same MRI-slice.

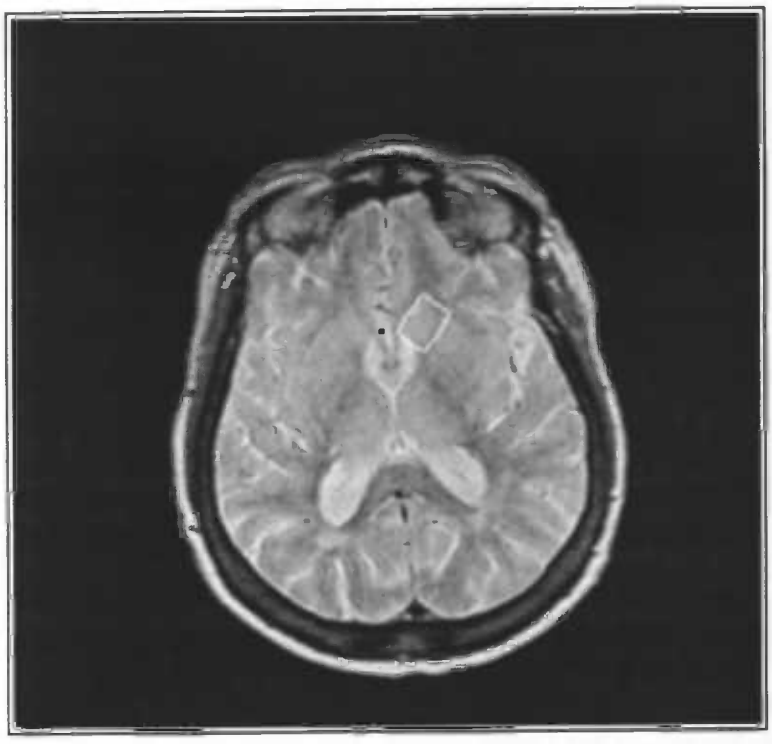

Figure 4.1: Original with contour around L. Caudate Head

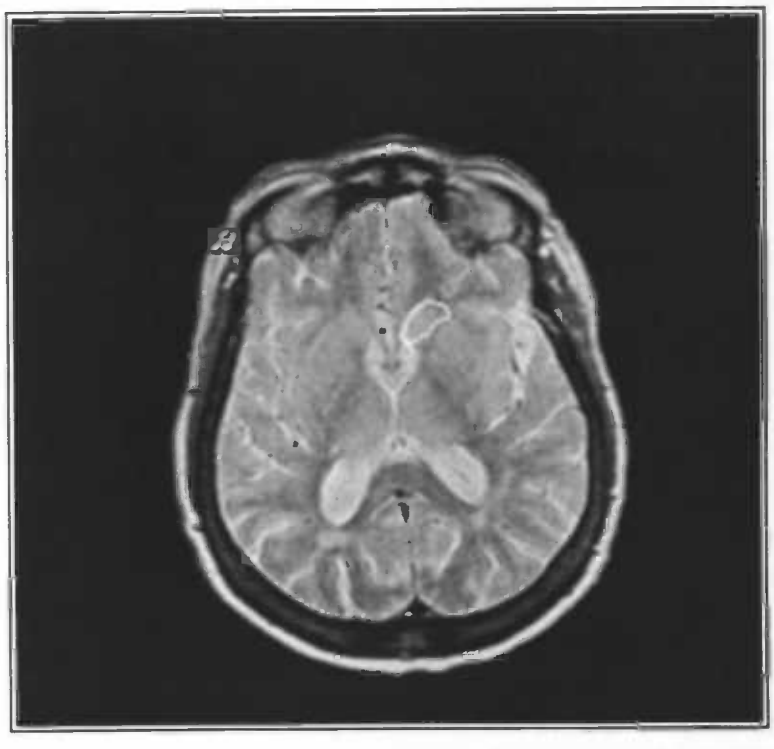

Figure 4.2: Original with contour after snake algorithm around L. Caudate Head

Besides changing the parameters for the snake algorithm the MRI-data is also exposed to some preprocessing. As mentioned earlier in this report the borders between some structures are not always quite visible. Equalizing these images results always in an image where the structures are much better visible, although the 'Putamen' is one of the structures that may hardly be visible. This is because the grey values are almost the same as the grey values of the surrounding structures.
In the images it can be seen that in most cases the snakes are not quite the same. Some of those snakes have some peaks or are connected to other brain structures, which is not the purpose. The Kuwahara filter is used here to make the image more homogeneous, but it pays attention to the borders between the different structures. As mentioned in the section 'Kuwahara filter', the size of the filter can differ. First we tried a size of 5 and later a size of 9. Looking to the images after applying the Kuwahara filter it became clear that a size of 5 gives the best result. The major disadvantage of a size of 9 is that the different brain structures are more connected, which results in one structure where the boundaries are sometimes barely visible, especially the structures which are closely together. Images 4.3 and 4.4 show the difference between the results for both sizes for the Kuwahara filter.

4.1 Tests

Finally the snake algorithm implemented in Matlab is used to test a large set of anatomical MRI-data. These data exist of three kind of types. The next three figures (4.5, 4.6 and 4.7) show these types. The third one is usually the one that appeared most of the time and gives the best results for performing the snake algorithm. When we only had of a few datasets to our disposal, as mentioned earlier, they were all of this kind of type. The first one, which is in some cases nearly white, is the one that appeared often in the anatomical MRI-data. For dealing with this kind of image type some additional tests were performed to see if the parameters for the snake algorithm also work for this MRI-data. In a later stage of the tests we found out that this kind of MRI-slices were originally taken in a different size. Transforming this data to Mayo Analyze format, those MRI-slices were padded with zeros to the dimension of 256*256. When this MRI-slice is displayed in Matlab using the function `imagesc` this resulted in such an image. This is caused by the huge number of zeros for the grey values, so that the remaining grey values are transformed to less values of a colormap. The colormap gray is used most of the time.
For the snake algorithm implemented in Matlab it is now also possible to control the parameter `alpha` which controls the contour of the snake.
Before applying the snake algorithm to the MRI-data, a template was created with the rough contours of the brain structures that should be analyzed. After

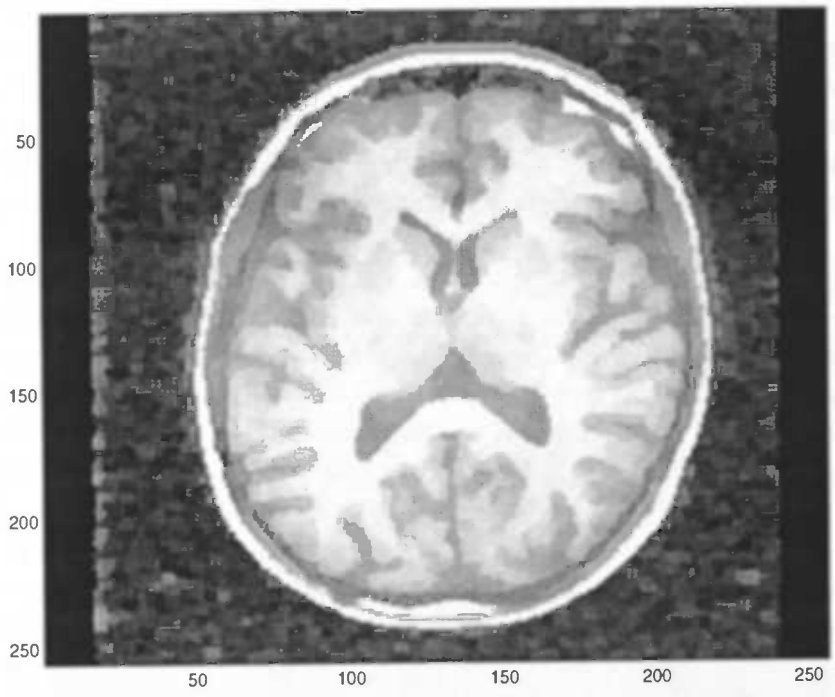

Figure 4.3: Result slice 10 of MRI-data 1229_a applied with Kuwahara filter with size 5

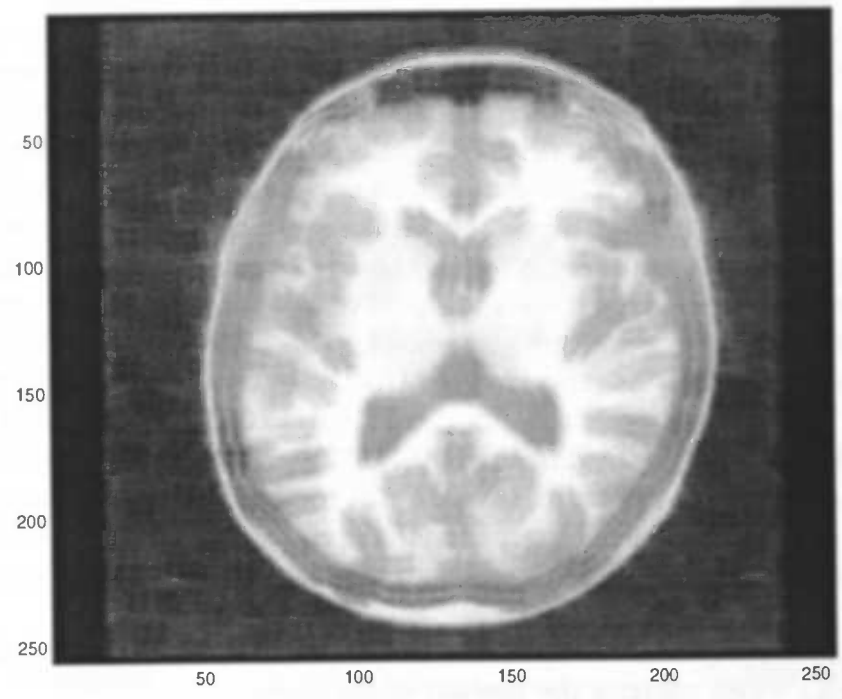

Figure 4.4: Result slice 10 of MRI-data 1229_a applied with Kuwahara filter with size 9

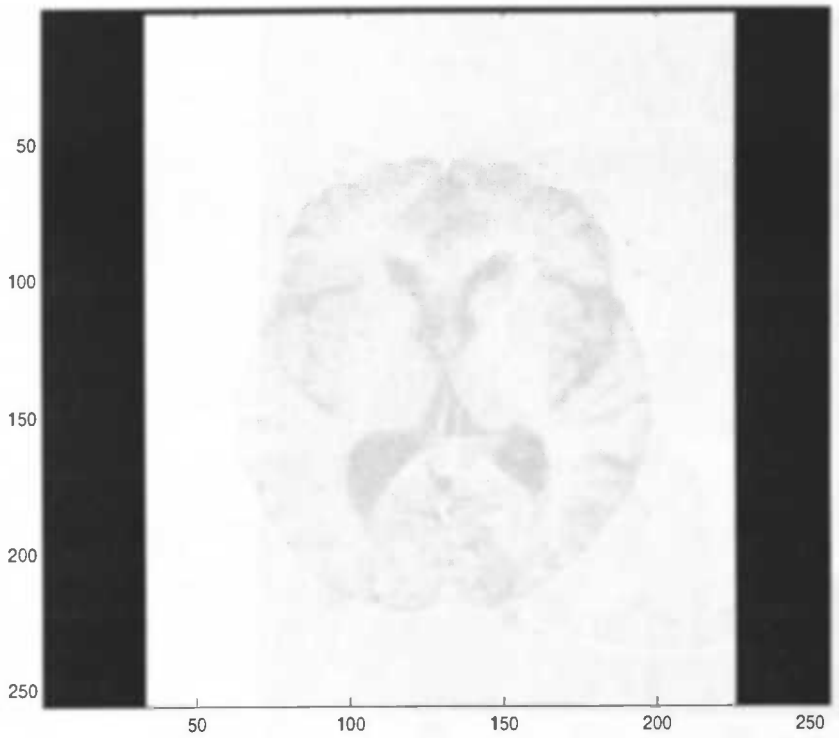

Figure 4.5: Three types of MRI-images (1)

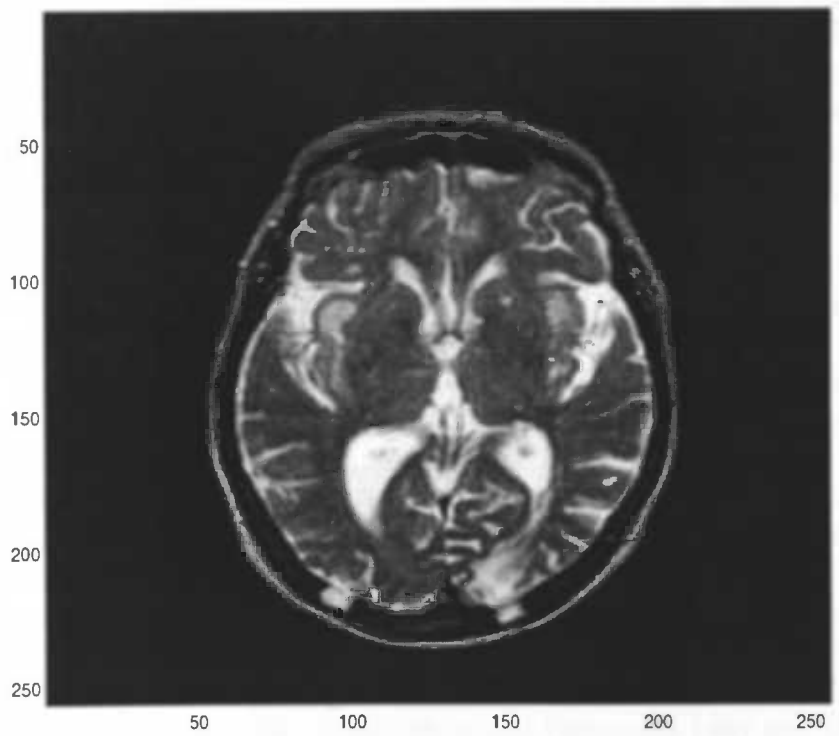

Figure 4.6: Three types of MRI-images (2)

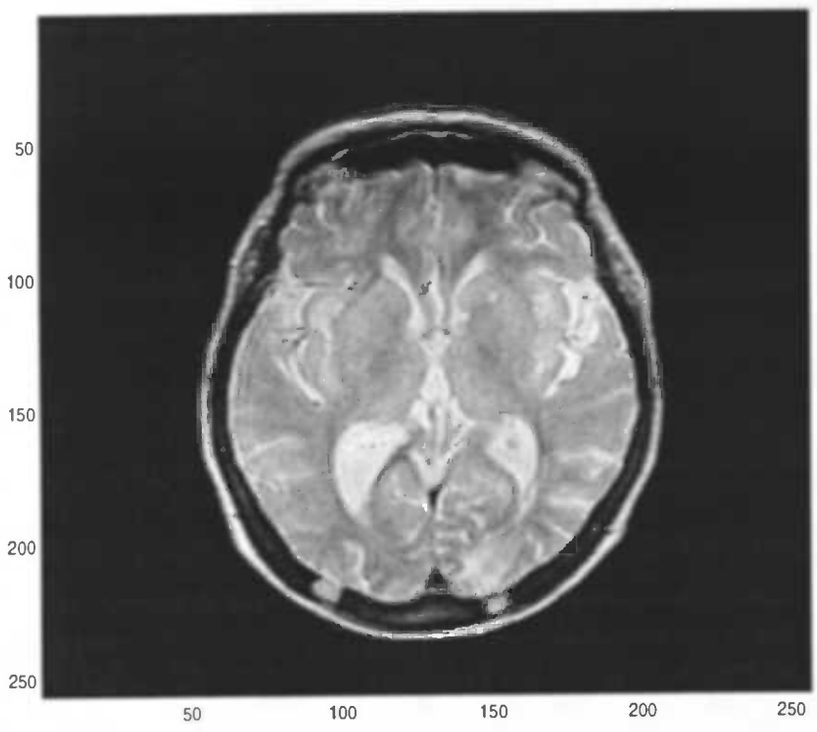

Figure 4.7: Three types of MRI-images (3)

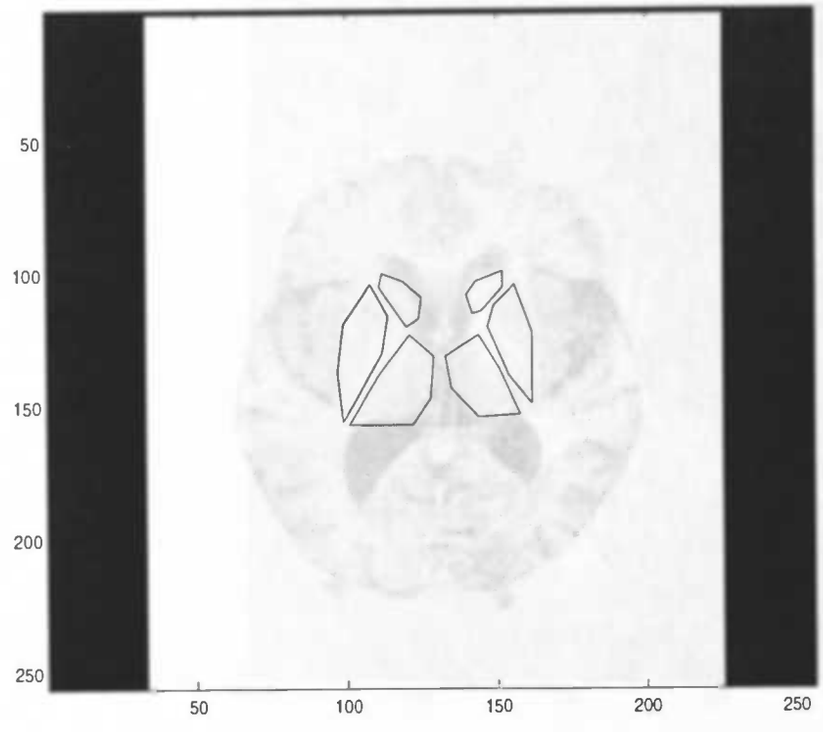

Figure 4.8: Example of MRI-slice 6 from m0024_an06_000 with template

reading a datafile in Matlab and selecting a slice, the template is loaded and drawn on the MRI-slice in the GUI. In figure 4.8 an MRI-slice of the MRI-data with this contour is displayed before it is used in the snake algorithm. The MRI-datafile m0024_an06_000 is used here, and in particular slice 6. Figure 4.9 displays the same MRI-slice but now the snake algorithm is performed with the default values. The MRI-slice is also prepared with a histogram equalization and the Kuwahara filter.

In figure 4.10 the same slice with the same template is displayed again but now some of the default values have been changed. In the next table only the parameters that are changed are displayed. The parameters that are not mentioned have default values.

Parameter	Value
alpha	1.6
gamma	1.8
maximum deviation	0.45

One sees that the value of the maximum deviation is changed to 0.45, this value is the new value that is used. The values of gamma and alpha are also increased and give the snake a more regular contour. Increasing the values of these two parameters further results often in a snake that has some sharp corners at some places. In figure 4.11 this is shown on slice 6 of dataset m0024_an06_000. In the figure one sees that some snakes have sharp corners. In particular at the Right Putamen (top of snake) and the Right Thalamus (left of snake). The value 2.5 is used for the parameter alpha and the value 2.2 for the parameter gamma.
Deriving conclusions from just one experiment is not possible, so we applied this template also to another dataset from the m0024 series. Slice 8 is taken from the MRI-dataset m0024_an08_000 and the snake algorithm is applied. In the next two figures (4.12 and 4.13) the results of this slice are displayed, one with and one without the equalization and Kuwahara filter.

From the figures one sees that the best results are this time achieved without preprocessing the MRI-slice. It is clear that the contours are more smooth. For example, both the Thalamus and the Caudate Head in the figure without the equalization and Kuwahara filter are much more smooth. In the figure where the equalization and Kuwahara filter are applied, the snakes around the brain structures have much more sharp points and are not so regular. The snake algorithm has also been applied on the MRI-slice with only the Kuwahara filter activated. The snakes around both Caudate Head and Thalamus on this slice gives also good roundings (no figure). Looking to the figures again we see that the Putamen in the figure without the equalization and Kuwahara filter are better than the other one, but it is still not as we would like. There are still too much interruptions in the snake, that is, the snakes are not smooth enough. Also the snakes around

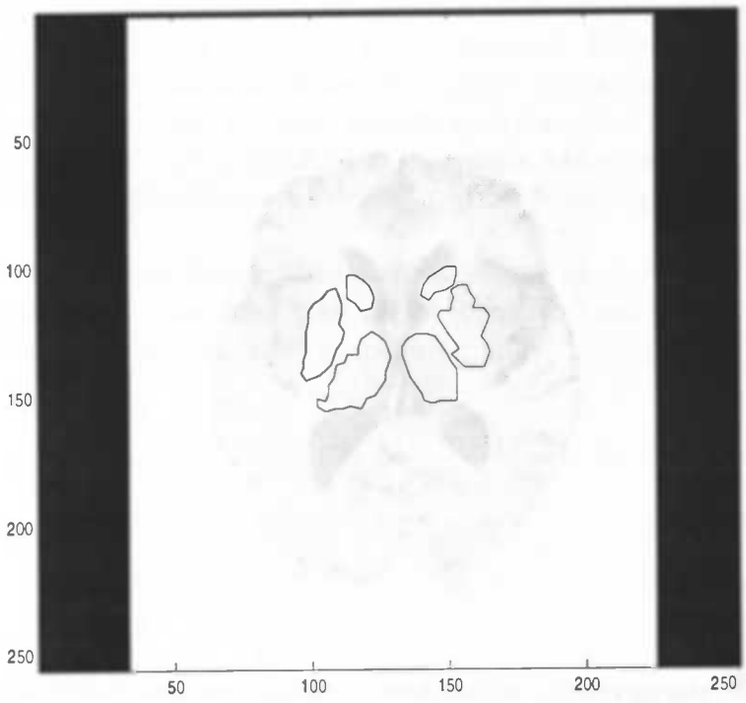

Figure 4.9: Result of snake template after snake algorithm with default values.

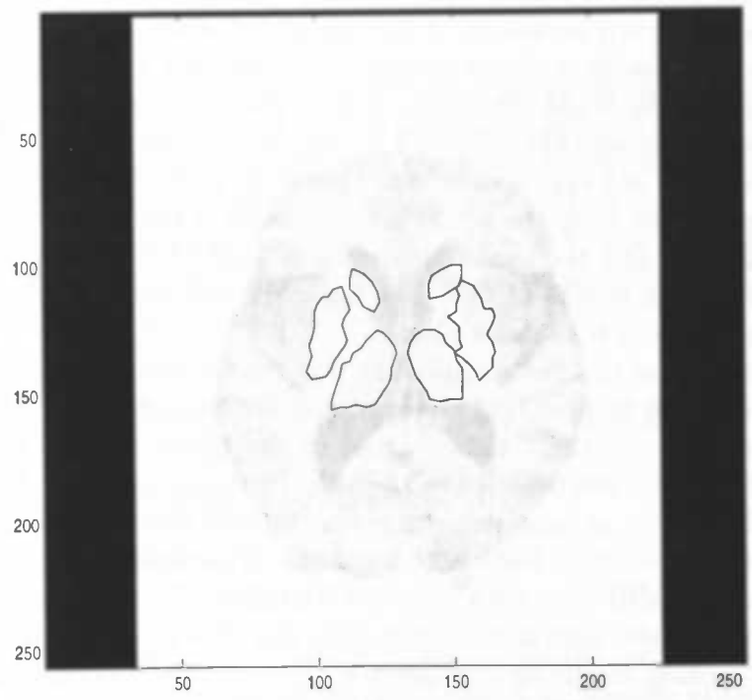

Figure 4.10: Result of snaketemplate after snake algorithm with non-default values.

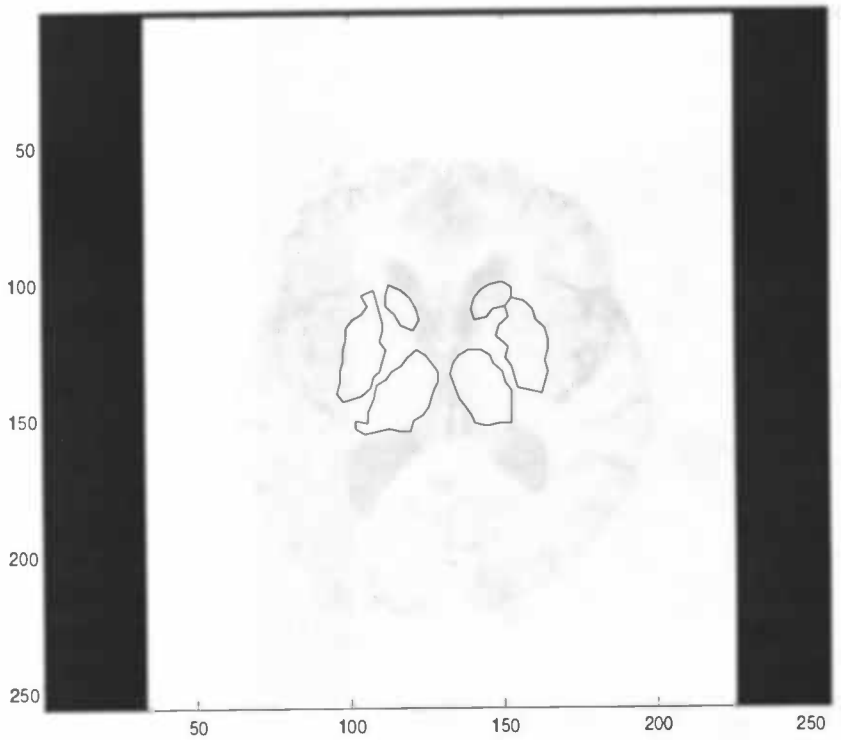

Figure 4.11: Result of snake template after snake algorithm with large values.

the Putamens in figure 4.12 are much more better than in figure 4.13, because both these snakes lie more around the real structure of the Putamen, which is smaller than the snakes. The Right Putamen in figure 4.12 is more down than that it should be.
In the figure without the equalization and Kuwahara filter the snake is connected to the other structures of the brain. This is visible because the snakes around the Left Caudate Head and Left Thalamus are attached to the snake around the Left Putamen.
In figures 4.14 and 4.15 two more slices are viewed with the template after the snake algorithm has been applied. In the first one the equalization and Kuwahara filter is applied and in the second not. Comparing both those slices with each other one can conclude that this time the figure where the equalization and Kuwahara is applied is the best one. Looking at the left Caudate Head it is visible that the snake around this structure is "exactly" around it, when using the equalization and Kuwahara filter. In the other one this snake is too big and is connected to the Left Putamen. Also the Right Thalamus and the Right Putamen are overlaying each other.
To make it more clear again two figures (4.16 and 4.17) are displayed, but now the snake algorithm is applied on slice 18 of MRI-dataset m0024_an03_000.
In figure 4.16 it is clear that the snakes are more smooth than in figure 4.17. In

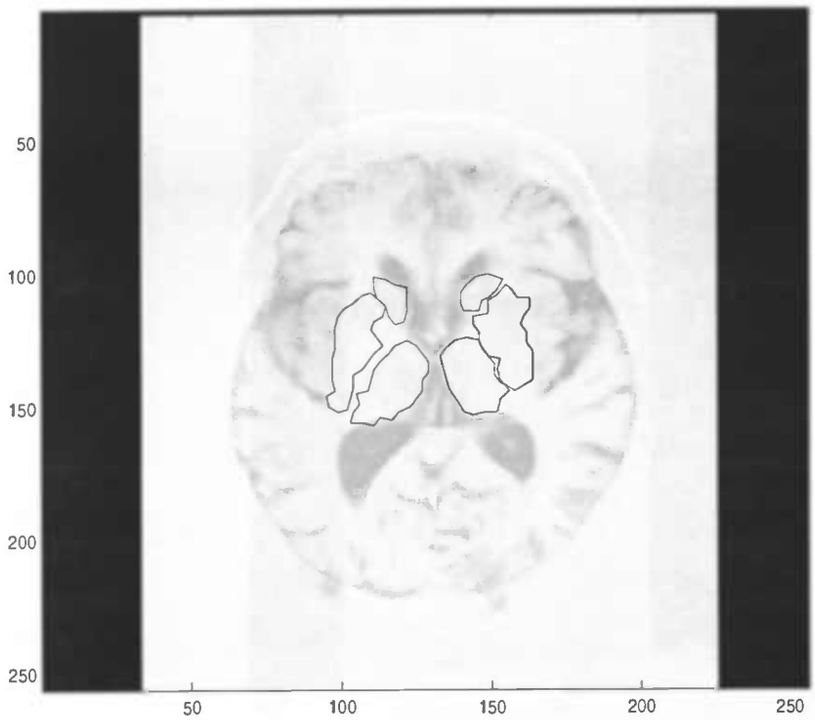

Figure 4.12: Result of slice 8 of dataset m0024_an08_000 with equalization and Kuwahara filter.

figure 4.17 the snakes do still have a lot of sharp corners.
Finally two figures (4.18 and 4.19) are displayed of an MRI-dataset from another series. The first one displays the MRI-slice with the template and the next one is the MRI-slice after performing the snake algorithm. When looking to figure 4.18 one can see that the template is not located quite well. The left side of the template matches quite well to the brain structures, but the right side of the template is too much to the right and top.

4.2 Testing on multiple datasets

MRI-dataset m0023_ant02_000

Slices are padded with zeros to 256*256.

- Slice 17
  Takes ± 3 seconds to optimise loaded pattern. Snakes are irregular, but difficult to assess the goodness of fit because of the problems with the zero padding and dynamic range.

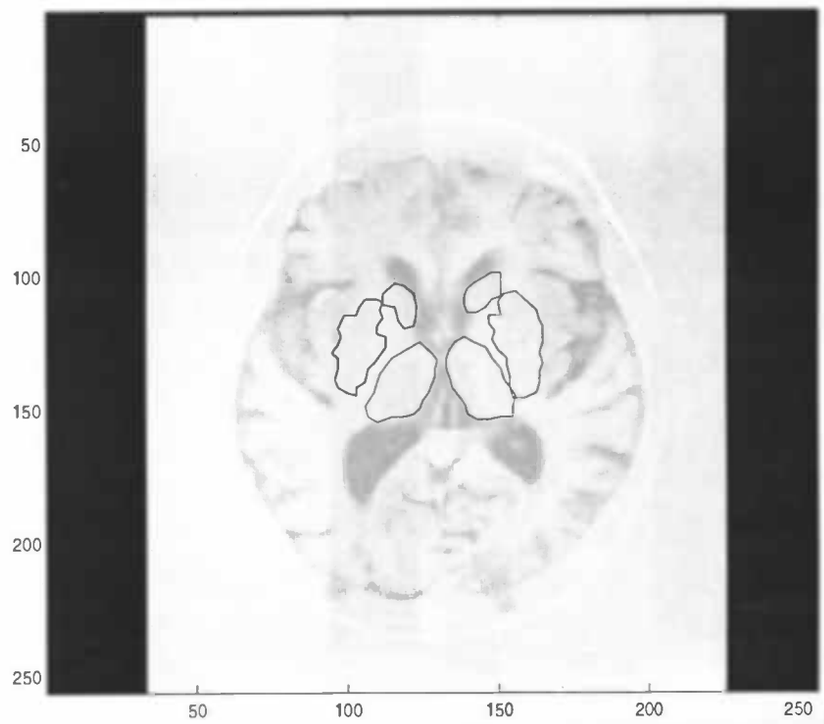

Figure 4.13: Result of slice 8 of dataset m0024_an08_000 without equalization and Kuwahara filter.

- Slice 18
  The left caudate contour is clearly on the ventricle after fitting. Right caudate ROI is not visible. Repeating the experiment gives the same result.

- Slice 19
  Again the regions are fitted even in the ventricles, if the start values are given in the ventricles.

- Slice 20
  In this slice mainly ventricles in the centre, but fitting the contours will fit in a constrained way within the ventricles.

MRI-dataset m0024_an02_000

Slices are of size 256*256, and not padded with zeros.

- Slice 17
  Pattern fits reasonably well, but there is very low contrast in the image and it is difficult to verify the fit.

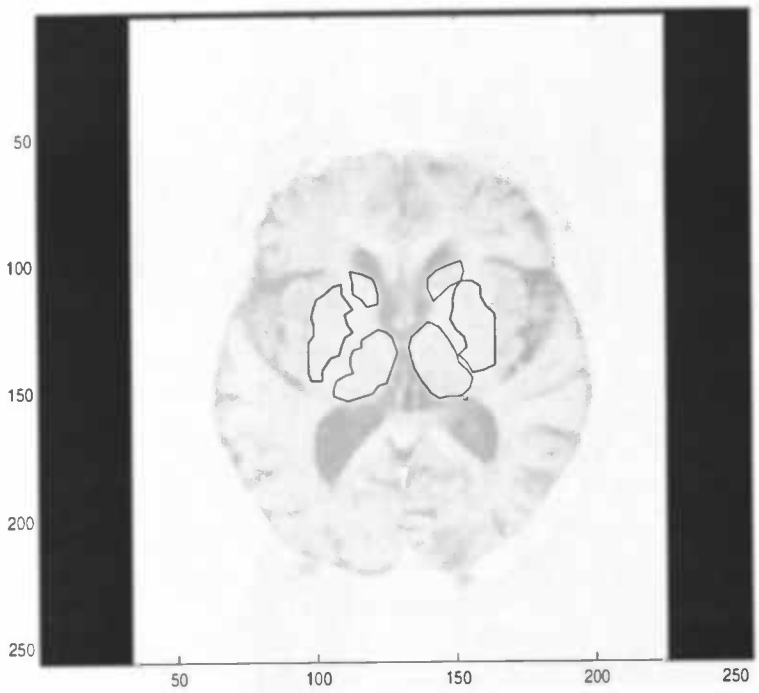

Figure 4.14: Result of slice 6 of dataset m0024_an07_000 with equalization and Kuwahara filter.

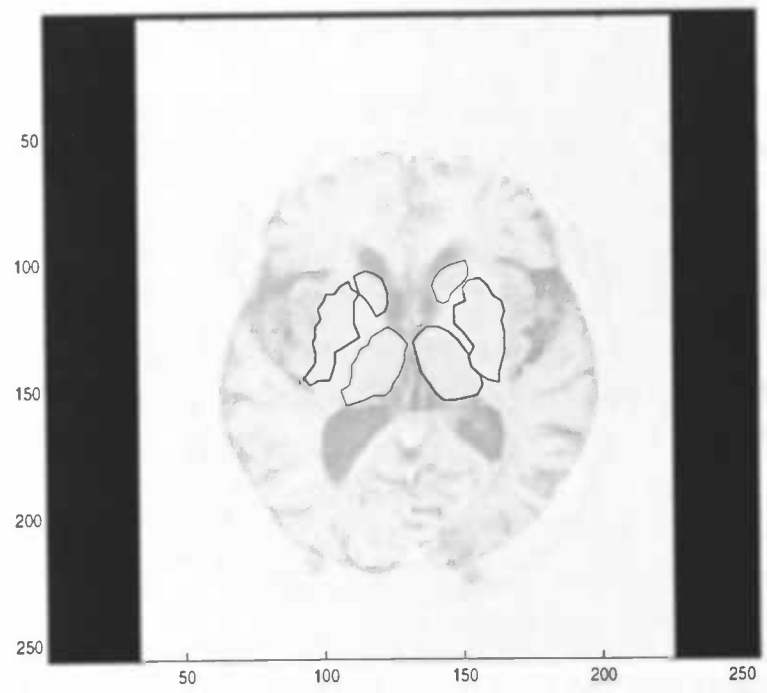

Figure 4.15: Result of slice 6 of dataset m0024_an07_000 without equalization and Kuwahara filter.

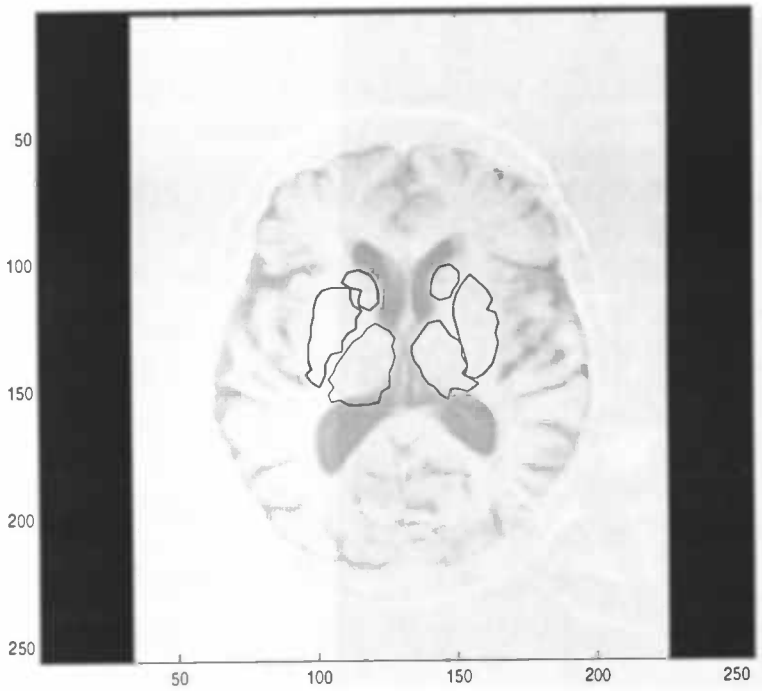

Figure 4.16: Result of slice 18 of dataset m0024_an07_000 with equalization and Kuwahara filter.

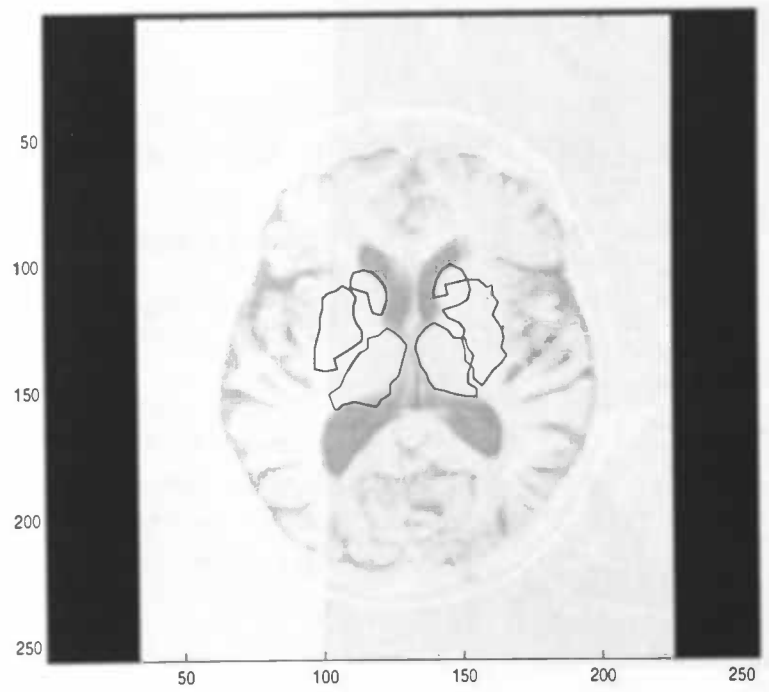

Figure 4.17: Result of slice 18 of dataset m0024_an03_000 without equalization and Kuwahara filter.

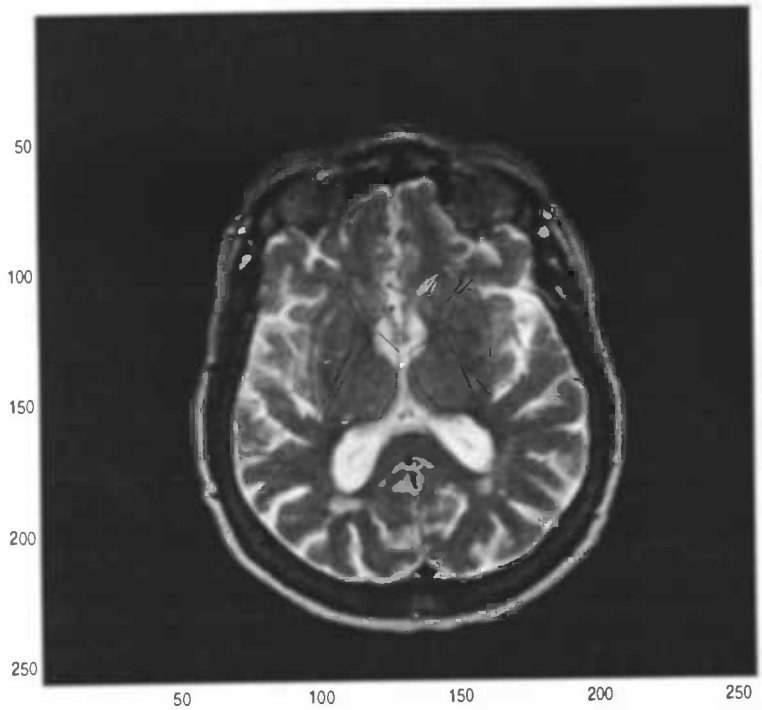

Figure 4.18: Slice 19 of dataset m0026_an02_000 with template.

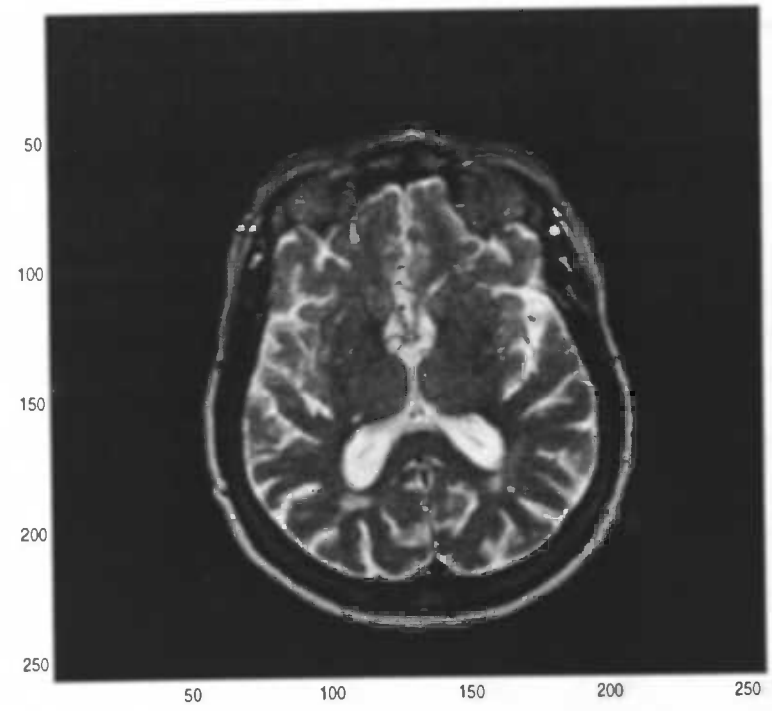

Figure 4.19: Result of slice 19 of dataset m0026_an02_000 with equalization and Kuwahara filter.

- Slice 18
  Pattern is fitted well, but there is a large difference in the shape of regions Right/Left for both the caudate and the putamen. Thalamus ROI extends into the ventricle.

- Slice 19
  Left caudate has a very small contour. Difference in shapes of all structures Right/Left.

- Slice 10
  ROI's are fitted close to the starting positions. Thalamus is in the centre of the ventricles.

MRI-dataset m0024_an03_000

Slices have again been padded with zeros.

- Slice 18
  Left caudate ROI is reasonable, but as with other ROI's, there is a lot of variation in the shape of the contours.

- Slice 19
  Left caudate is missing, other regions are fitted around their start values. Thalamus extends into ventricles.

- Slice 20
  Regions centred around start values, irregular shapes. This slice does not appear to contain the basal ganglia.

MRI-dataset m0024_an06_000

Slices are padded with zeros, but show the basal ganglia more clearly than the other MRI-datasets.

- Slice 6
  The contours fit reasonably well on the Right/Left Thalamus, Right Caudate and Right/Left Putamen. Right Putamen contour is ragged. This image was used to define the ROI template, so that the goodness of fit may be dependent on the starting values.

- Slice 7
  The contours are fitted badly, but start shapes, don't represent shapes which are apparent in the image.

MRI-datset m0024_an07_000

Contrasts in these slices are better.

- Slice 6
  The contours fit well, especially around the borders of the ventricles, suggesting that the algorithm performs well at higher gradients / contrasts.

MRI-dataset m0024_an08_000

Same results as for m0024 above, the contours are more ragged.

MRI-dataset m0025_an02_000

High contrast image, however sliced in a strange direction. Difficult to find a slice which corresponds to the template. Image has been acquired in an interleaved mode, so that there are two sets of images.

MRI-dataset m0025_an03_000

Also not useful, as an02 for this study. Contrast is very low because of the 'padding' effect.

MRI-dataset m0026_an02_000

Tried with the ROI pattern, again, with poor starting estimates, the result is poor. Shifting the ROI pattern improves the result but contours are ragged.

MRI-dataset m0026_an03_000

Although contours are visible, the pattern does not fit well, even when it is moved to provide better starting estimates. Note that there is a problem with padding with zeros again here.

- Slice 9
  Poor fitting of ROI's - padding effect.

MRI-dataset m0027_ant03_000

- Slice 9
  Poor fitting of regions, although they are visible - padding effect.

MRI-dataset m0054_ant01_000

Example slice from another dataset in /data/psych*

- Slice 8
  Template ROI's are off centre. Move them using move function which is very slow (one pixel per click) with updates. Centre Right Caudate on the apparent Right Caudate then gives nice results. Also drawing the snake onto the image and then using algorithm without Kuwahara filter give good results.

MRI-dataset m0051_ant01_000

- Slice 6
  Fitting template ROI's does not give a good result, but moving the head of Left Caudate in the template to the position in the image, yields a reasonable fit.

4.3 Summary of the tests

The GUI snakes application read all MRI images properly.
Snakes do not fit well to the apparent anatomical structures, however if the starting contours are moved to reasonable starting positions, the snakes will converge to a sensible solution. This solution will be affected by other local gradients in the image, i.e. ventricles, if they are included within the starting contours.
The MRI images show the basal ganglia with varying contrast. One major problem is the padding of images smaller than 256*256 with zeros, this causes the dynamic range to be disturbed, so that the full contrast is not available for the algorithm. This is a problem with image conversion of the original images, and is not an effect of the snakes algorithm.
Testing in 3D has not been applied, because the distance between the slices from the MRI-data is large. Thus a single template is applied on a single slice. The regions that we would like to be extracted, appeared in most cases on one or two slices, which is not enough for extracting a brain structure in 3D.

Chapter 5

Conclusions and recommendations

Defining a ROI of a particular structure in the brain is not easy. The results of defining and fitting default ROI-patterns depend heavily on the MRI-data that is acquired for analysis. Comparing the different MRI-datasets with each other, one can conclude that there are sometimes a lot of differences. The concept of using a fixed pattern of ROI's as starting values for the snake algorithm is not robust. The pattern must be adjusted before using the algorithm to apply fine contour discrimination.
The parameters for the snake algorithm are changed a lot, but obtaining default values which can be used for all MRI-data is not possible. This is also caused by the quality of the data.
Work has to be done before this method can be used to analyse clinical studies. The test section of the project has shown that snakes might be used to define contours, but that flexibility of definition of the starting values is important. The quality of the data also needs to be improved - by changing the acquisiton sequence, this should be given the highest priority.
The source code of the snake contour fitting and filter algorithms needs to be well documented and systematically organised to allow further work on this subject within MATLAB.
An option to improve the snake algorithm is to extend it with predefined contours. This means that the contour of the snake can only grow to a specified type like an ellipse.
The 3D aspect of the application needs to be improved, at the moment it is still working on individual slices. Current data sets include the whole brain, but the basal ganglia appear on at best one slice. In order to get more morphological information, the axial slice thickness needs to be reduced.

Bibliography

[1] M. Kass A. Witkin and D. Terzopoulos. Snakes: Active contour models. In *International journal on computer vision*, 1(4), 1987.

[2] Donald Hearn and M. Pauline Baker. *Computer Graphics (C version)*. Prentice-Hall International, Inc., 2nd edition, International Edition, 1997.

[3] J. Pluim and M. Westenberg. Segmentation of vertebrae (laboratorium 9), April 1999.

[4] D.J. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. In *CVGIP: Image Understanding*, 55(1), January 1992.