An evaluation of image segmentation techniques for MRI scans

by

Claire Wagenaar

to obtain the degree of

Bachelor of Science

in

Applied Mathematics

Student number: 4713036
Thesis committee: Dr.ir. M.B. van Gijzen (supervisor)
Dr. Y. van Gennip

DELFT UNIVERSITY OF TECHNOLOGY
June, 2020

Een evaluatie van beeldsegmentatietechnieken voor MRI scans

door

Claire Wagenaar

ter verkrijging van de graad van

Bachelor of Science

in

Technische Wiskunde

Studentnummer: 4713036
Beoordelingscommissie: Dr.ir. M.B. van Gijzen (begeleider)
Dr. Y. van Gennip

TECHNISCHE UNIVERSITEIT DELFT
Juni, 2020

Preface

Before you lies the bachelor thesis *"An evaluation of image segmentation techniques for MRI scans"*. This thesis has been written in order to obtain the degree of Bachelor of Science in Applied Mathematics. The research has been done under the supervision of Martin van Gijzen, within the Numerical Mathematics department of the faculty EEMCS at Delft University of Technology.

When going through the list of possible bachelor projects, I was immediately interested in this project about image segmentation. What appealed to me, was the social relevance of this project. This bachelor project is a small link of a bigger project, in which an affordable MRI scanner is developed for low-income countries, that will be used to detect hydrocephalus. The main focus of this bachelor project is to implement and test several image segmentation methods. These methods will eventually be used to segment MRI brain scans, if they turn out to be helpful. Hence, I felt like this would be an opportunity to put my knowledge in something tangible, relevant and real.

In the beginning of March, I started reading about image segmentation, which was a subject I initially had never heard of. The book that helped me gain general knowledge was *Digital Image Processing* written by R. Gonzalez and R. Woods [8], in which several methods were clearly explained. Next, I gradually started implementing the easier methods and from the beginning of mid April on, I worked full-time on implementing, extending and testing the methods.

I want to thank Martin van Gijzen for his knowledge and support. Whenever I did not know how to proceed, he gave me advice and new ideas that kept me on the right track. Next, I would like to thank Yves van Gennip for taking a seat in my thesis committee. I also wish to thank Leiden University Medical Center for providing me their MRI scans. My final word of thanks goes out to my fellow students, parents and boyfriend for reviewing parts of my report, their wise words and keeping me motivated.

I hope you enjoy reading this thesis.

Claire Denise Wagenaar
Delft, June 2020

Abstract

In this thesis, several image segmentation techniques will be tested that eventually will be applied to MRI brain scans in order to detect hydrocephalus. The methods include Sobel edge detection, Canny edge detection, active contour model (also known as snakes), k-means clustering and region growing. Furthermore two extensions are discussed. We propose a method to complete disconnected edges and, as an extension to the k-means clustering algorithm, we suggest a manner to reassign pixels to different clusters.
The focus of the first part of the research lies on explaining and illustrating these methods and extensions. Two test images are used, namely the Shepp-Logan Phantom and an MRI scan of a recreated Shepp-Logan Phantom. In the second part, we evaluate each method by applying the methods to two independent images, that is an MRI scan of an apple and a CT scan of a brain with hydrocephalus. We discuss whether methods are useful and behave as expected. Additionally, we investigate ways to combine methods.

Contents

1 Introduction **1**

2 Methodology **3**

3 Edge based methods **5**
3.1 Basic formulation of edges 5
3.2 Sobel edge detection 6
3.2.1 Method description 7
3.2.2 Result 7
3.3 Canny edge detection 7
3.3.1 Method description 8
3.3.2 Result 10
3.4 Active contour model 10
3.4.1 Mathematical formulation 10
3.4.2 Result 15
3.5 Completing edges 16
3.5.1 Finding loose ends 16
3.5.2 Connecting edges 18

4 K-means clustering **23**
4.1 Method description 23
4.2 Result 24
4.3 Influence of the choice of initial cluster centers 24
4.4 Reassigning pixels to different clusters 26
4.4.1 Method description 26
4.4.2 Result 27

5 Region growing **28**
5.1 Method description 28
5.2 Result 29

6 Numerical experiments **30**
6.1 K-means clustering and region growing 30
6.1.1 The influence of different cluster centers 31
6.1.2 Reassigning pixels 33
6.2 Sobel and Canny edge detection 35
6.3 Active contour model 37
6.3.1 Sensitivity of the parameters 38

7 Conclusion and discussion **41**
7.1 Conclusion 41
7.2 Recommendations 42
7.3 Running time 43

A Manual for the Python code **45**
A.1 K-means clustering 45
A.2 Reassigning pixels 46
A.3 Region growing 46
A.4 Sobel edge detection 47
A.5 Canny edge detection 47
A.6 Completing edges 47
A.6.1 Finding loose ends 47
A.6.2 Connecting edges 48
A.7 Active contour model 48

Bibliography **49**

1 Introduction

When we, humans, look at an image like Figure 1.1a, we immediately know what we see. Namely, we recognize a slice of a watermelon, with six pits, its peel, the pulp and the background with the watermelon's shadow. However, in a world where technology is rapidly developing, the question arises: how can computers see the same in images as what we see?

(a) Photograph of a watermelon, source [15]. (b) Segmented image of the watermelon

Figure 1.1: Example of image segmentation.

This is done by image segmentation. Image segmentation is the process of dividing an image in separate areas, where pixels in one area share the same properties, such as color or texture. An image is a collection of pixels, with different pixel values. By image segmentation, each of these pixels get a label. Pixels that share the same properties, get the same label and are grouped together into one area. This process gives the image a more meaningful and simple representation and makes it easier to apply other image processing techniques, such as image recognition. Segmentation helps to retrieve relevant parts of an image and to detect boundaries and shapes of objects. For instance, Figure 1.1b shows the result of segmenting Figure 1.1a. We can easily distinguish between the background, the pits, the shadow etcetera.

This research is part of a bigger research [3]. Namely, Leiden University Medical Center and TU Delft aim to develop an affordable MRI scanner for low-income countries in Africa [20]. MRI scanners are usually expensive, difficult to operate and hence unavailable to many doctors. The MRI scanner will be used to detect hydrocephalus. This is a disease that affects many newborns and children in Africa, where too much cerebrospinal fluid is found in the brains, which results in impaired brain function [14]. Image segmentation will be used to segment head scans, that

are made with the developed MRI scanner. Since each material in the head (for instance bone, water, brain) shows a different intensity in the MRI scan, it would not be hard to distinguish between the different parts of the head. Segmentation makes it for instance possible to detect the amount and location of water in the brain.
The goal of this research is to implement various image segmentation techniques and test those techniques on real images.

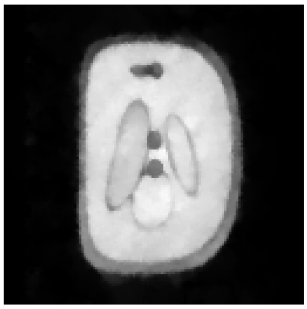
Finally, we give the reader a short overview of this thesis. First, the method of the research will be explained in Chapter 1. In the following three chapters we will explain the methods, that is the edge based methods in Chapter 3 (including Sobel edge detection, Canny edge detection and active contour model), k-means clustering in Chapter 4 and region growing in Chapter 5. For each of these methods, a description will be given followed by some examples. For the edge based methods and the k-means clustering we will also give an extension. Next, in Chapter 6 we execute several experiments, to evaluate the results and the use of the methods. The conclusion and discussion can be found in Chapter 7. Finally, the appendix contains an instruction manual of the Python code.

2 Methodology

In this chapter, we will discuss the approach of this research.

In the first part of the research, several methods will be studied, implemented in Python, and tested. These methods include: Sobel edge detection, Canny edge detection, active contour model, k-means clustering and region growing. In this research, only gray-scale images will be considered, which means that the pixel intensities are between 0 (black) and 1 (white).
The methods will be tested on a couple of test images, to determine if the implementation works. To begin, we have the Shepp-Logan Phantom [16], shown in Figure 2.1a. This is a standard test image and is often used as a model for the human head. The different regions of the image are very clear by eye and each part has an even intensity. Therefore it is easy to predict what an algorithm should return, and it is thus easy to check if the implementation works.
A different Shepp-Logan Phantom is the one shown in Figure 2.1b. This is an MRI scan of several liquids, that each return different intensities. The MRI scanner that was used is the self-made scanner, mentioned in Chapter 1. The measurement of this Shepp-Logan Phantom was taken upside down. The segments are still clear by eye, however the intensities are not even. It is interesting to test on MRI scans, since in the end, the methods are used to segment MRI scans as well.

(a) Shepp-Logan Phantom simulated in Python.



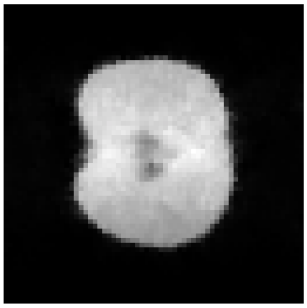
(b) Shepp-Logan Phantom measured by an MRI scanner.

Figure 2.1: Images that are used to implement and test the methods.

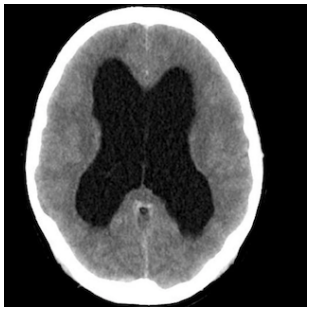
After the testing is done, we use an independent set of images to evaluate the methods. In the first place, we consider Figure 2.2a. This is a slice from a three-dimensional MRI scan, that is taken by the same MRI scanner that was used to

obtain Figure 2.1b.

In addition, we use Figure 2.2b, which is a brain scan that indicates hydrocephalus [17]. This image is interesting to segment since the ultimate goal is to apply image segmentation on such brain scans containing these shapes. Roughly said, the white region is bone, the dark gray region is the brain and the black X-shaped region in the middle of the image are dilated ventricles containing cerebrospinal fluid. Usually, these are much smaller. The ventricles are the most relevant part of the scan, since its size indicates hydrocephalus. Note that the transition from the brain to the ventricles is not smooth. Image segmentation can be used to obtain the precise shape, and the volume of these ventricles.
The evaluation is quantitative, since the images will be analyzed visually.



(a) MRI scan of an apple.



(b) CT scan of a brain with hydrocephalus, source: [17].

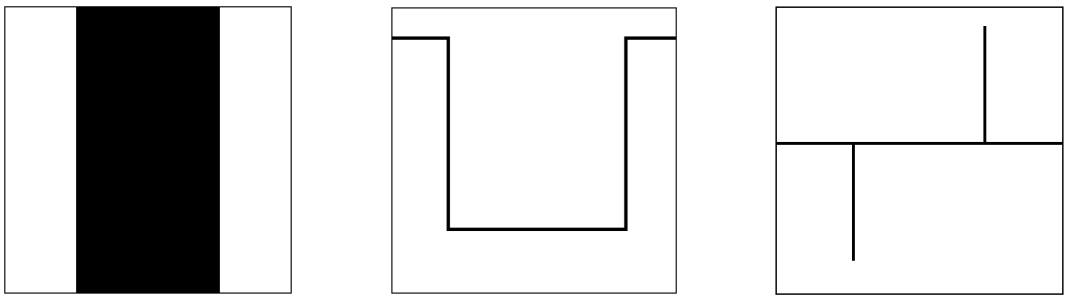
Figure 2.2: Images that are used to evaluate the segmentation methods.

3 Edge based methods

In this chapter, we explain some edge based methods. As the name says, these methods aim to find edges of an image. Edges can be found by using the gradient of an image. In the first part of this chapter, we will explain this principle. Furthermore, two methods to find edges will be discussed, namely Sobel edge detection and Canny edge detection. The latter consists of multiple stages which will result in an image containing only relevant and thin edges. Additionally, the active contour model is discussed. Finally, we will propose a method to detect unfinished edges and complete those.

3.1 Basic formulation of edges

An edge can be defined as a sudden change in the gray-level intensity in an image, since at an edge the intensity changes from one to another. For an ideal edge (Figure 3.1a), the transition is quick and the intensity profile, with the gray-level on the vertical axis, will be a vertical line (Figure 3.1b). At the position where the two different intensities meet, so where the edge is, a jump in the gray-level is found. However, in reality, the gray-level may be smoothly changing, instead of suddenly. This results in a sloping profile, instead of a vertical line.



(a) Ideal edge (b) Corresponding intensity profile. (c) Corresponding derivative.

Figure 3.1: An ideal edge with its gray-level intensity profile and derivative. Since the change in intensities is sharp, the profile is a vertical line.

We have seen that edges are changes in the intensity in an image. Such changes can be found by taking the first derivative. The first derivative measures the change

of intensity of a function, where in this case the image is the function consisting of pixel values. Places where differences are found to be large, indicate the presence of an edge. Figure 3.1c shows the corresponding derivative of an ideal edge. The two peaks belong to the two intensity changes.

Let $f(x, y)$ denote the image at point (x, y), then the gradient is defined as follows:

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} G_x \\ G_y \end{pmatrix} \tag{3.1}$$

Two properties can be derived from this expression. First, the magnitude of the gradient, which represents the strength of the edge. The stronger the edge, the stronger the change in intensity, thus the higher the magnitude. The magnitude is given by:

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.2}$$

Secondly, the direction of the gradient is given by:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{3.3}$$

The direction of the gradient is perpendicular to the direction of the edge. The gradient direction points to the light side, while the edge direction has the light side on its right, see also Figure 3.2 that shows both directions. In this case, the angle of the gradient is 45°and the direction of the edge is 135°.

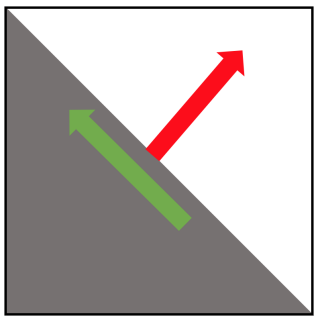


Figure 3.2: The direction of the edge in green and the direction of the gradient in red.

3.2 Sobel edge detection

We now know that edges can be detected by using the gradient of the image, so next we will explain how the gradient can be computed.

3.2.1 Method description

The gradient at a point in an image can be found by using a 3 × 3 mask, which represents the 8-connected neighborhood of a pixel (Figure 3.3a). The values z_i are the gray values of the pixels in the image.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

(a) Mask that defines a 3 × 3 neighborhood

-1	0	1
-2	0	2
-1	0	1

(b) Sobel mask to compute G_x

1	2	1
0	0	0
-1	-2	-1

(c) Sobel mask to compute G_y

Figure 3.3: Masks that are used to compute the gradient

The Sobel masks [18] may be used to compute the gradient in the x- and y-direction (Figures 3.3b and 3.3c). The partial derivatives G_x and G_y of Equation 3.1 at point z_5 are calculated by element-wise multiplying the Sobel masks with the 3 × 3 region and taking the sum subsequently:

$$G_x = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{3.4}$$

$$G_y = (z_1 + 2z_2 + z_3) - (z_7 + 2z_8 + z_9) \tag{3.5}$$

When the gradient is calculated over a uniform area, the difference and therefore the magnitude is 0. Since the values of G_x and G_y add up to a maximum of 4 and a minimum of -4 (both indicating strong edges), the magnitude should be normalized to make sure the gray values lie between 0 and 1.
The multiplication of the masks should be done for the whole image, by shifting the masks in Figure 3.3 over every pixel. This procedure, applying the Sobel masks over the whole image, is called Sobel edge detection. There are many variations of the Sobel masks as shown in Figure 3.3b and 3.3c that are used similarly. However, in this research, we will only use the Sobel masks.

3.2.2 Result

Figure 3.4 shows the result when Sobel edge detection is applied to the measured Shepp-Logan Phantom. Figure 3.4a and Figure 3.4b show the gradient in the x- and y-direction. Note that in Figure 3.4c the stronger edges are brighter than the weaker edges.

3.3 Canny edge detection

Next, we discuss the Canny edge detection algorithm [4], which is a method to find the most relevant edges in an image. It consists of five steps, that will be discussed in this section, after which a result is presented.

(a) Gradient in the x-direction

(b) Gradient in the y-direction

(c) Magnitude of the gradient

Figure 3.4: Sobel edge detection applied to the measured Shepp-Logan Phantom

3.3.1 Method description

First, we will explain each step of the method.

Blurring

The first step is to blur the image by applying a Gaussian filter. This makes the images smoother and reduces the noise.

Calculate the gradient

Next, we find the image gradient, which can be done by using the Sobel filters (Figure 3.3) as discussed in Section 3.2.1. We willl also need both the gradient magnitude and the gradient direction. These can be calculated by Equation 3.2 and Equation 3.3.

Non-maximum suppression

Third of all, we apply non-maximum suppression, in order to make the edges thinner. We use the gradient magnitude as well as the direction of the gradient, which we calculated in the previous step. For each pixel, we compare its magnitude to the pixels previous and next. When the gray value is greater than both the previous and the next pixel, it will be preserved, otherwise it will be suppressed. The pixels that are preserved get the same value in the output image as in the image of the magnitude, while the suppressed pixels will be marked black (a value of 0).

To obtain the previous and next pixels, we use the direction of the gradient. Recall that the direction of the gradient is perpendicular to the direction of the edge itself. The angle is rounded to one of the four possible orientations: horizontal, vertical, diagonal or antidiagonal. The orientation and the pixels previous and next are given as follows, for angles between -180°and 180°(see also Figure 3.5):

- When the angle is in the interval [0, 22.5°), [-22.5°, 0°], [-180°, -157.5°) or [157.5°, 180°], the gradient direction is horizontal, hence the pixel is compared to the pixels on the left and right.

- When the angle is in the interval [22.5°, 67.5°) or [-157.5°, -112.5°), the gradient direction is diagonal, so the pixel is compared to the pixels in the upper right and lower left corner.
- When the angle is in the interval [67.5°, 112.5°) or [-112.5°, -67.5°), the gradient direction is vertical, so the pixel is compared to the pixels above and under.
- When the angle is between [112.5°, 157.5°) or [-67.5°, -22.5°), the gradient direction is anti diagonal, so the pixel is compared to the pixels in the upper left and lower right corner.

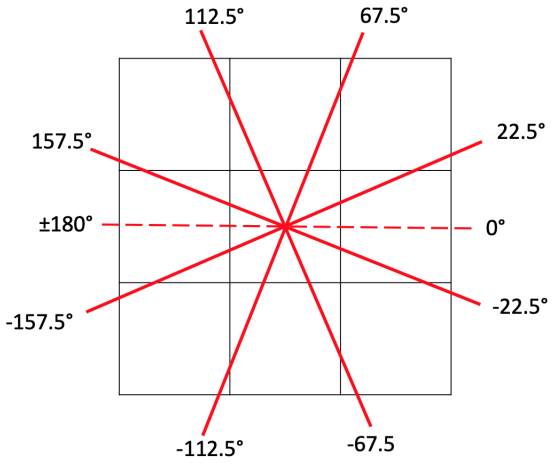


Figure 3.5: Angles are rounded to one of the four possible orientations.

The output of the non-maximum suppression is an image with edges of mostly one pixel wide (except for instance in the corners), since only the strong pixels of an edge are preserved.
Note that the direction and its reverse are handled similarly, so it does not matter if the direction is for instance from left to right or from right to left.

Double thresholding

The fourth step is to apply double thresholding. Say there are two thresholds T_1 and T_2, then there are three possibilities for the pixel values in the output image. Let $f(x, y)$ denote the pixel intensity at coordinate (x, y) of the image after the non-maximum suppression, and $g(x, y)$ the intensity of the image after thresholding. A pixel can then either be marked as weak, medium or strong, with the corresponding intensity values of 0, 0.5 or 1. Which one of the three values the pixel gets, depends on the intensity of the pixel value in $f(x, y)$. The output image is defined as follows:

$$g(x,y) = \begin{cases} 0, & \text{if } f(x,y) \leq T_1, \\ 0.5, & \text{if } T_1 < f(x,y) < T_2 \\ 1, & \text{if } f(x) \geq T_2 \end{cases}$$

The weak pixels are suppressed, while the medium and strong pixels are preserved with a different intensity.

Hysteresis

The final step is hysteresis. In the previous step, weak pixels are already removed, since they got the value 0 and the strong pixels are certainly preserved with a value of 1. Therefore, we should now decide on the medium pixels with a gray value of 0.5, whether they are relevant or not. For all these medium pixels, we check if they are connected to a strong pixel in the image after double thresholding. If so: they are also marked strong and hence preserved, else they are marked weak, and hence suppressed. The image after hysteresis contains only pixels with a value of 0 or 1.

We now explained the five steps that make up the Canny edge detection. The input of this algorithm is the image, the amount of Gaussian smoothing that is used during blurring (referred to as σ) and the low and high threshold that determine the weak, medium and strong edges (T_1, T_2). The output is an image containing the relevant thin edges of the original image, with only gray values of 0 (black, the intensity of the background) or 1 (white, the intensity of the edge). We refer to such an image as a binary image.

The Python implementation used in this research is partially based on the implementation described in [11].

3.3.2 Result

When we apply the above procedure to the measured Shepp-Logan Phantom, we get the following result. We used $\sigma = 1$, $T_1 = 0.1$ and $T_2 = 0.2$. Figure 3.6c contains pixel values that are the same as in Figure 3.6b. In Figure 3.6d the values are either 0, 0.5 or 1 and Figure 3.6e contains only pixels with a value of 0 or 1.

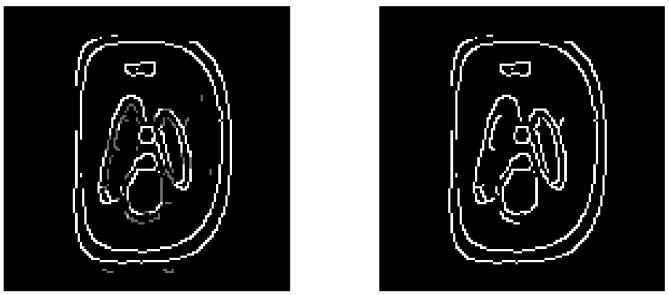
3.4 Active contour model

In this section, we discuss the active contour model. This is a way to find the contour of an object in an image. First, we describe the method mathematically and finally, we show some results.

3.4.1 Mathematical formulation

The active contour model [12], also referred to as a "snake", is a curve that is moved over an image, aiming to minimize its energy. It is attracted to features such as edges, lines or corners, which makes it useful to obtain contours.
The curve is parametrized as: $\mathbf{s}(p) = (s^x(p), s^y(p))^T$, for $p \in [0, 1]$. The vectors $\mathbf{s}^x$ and $\mathbf{s}^y$ contain the x- and y-coordinates of the snake. Often a closed curve is used, then it holds that: $\mathbf{s}(0) = \mathbf{s}(1)$. Starting with a initial curve given by the user, the line will move under the influence of internal and external forces into a shape with minimal energy. To achieve this, an energy functional is introduced that should be minimized. The energy function consists of an internal energy and an external energy, as defined in [12], Section 2:

(a) Gaussian blurring (σ=1) (b) Gradient magnitude (c) Non-maximum suppression



(d) Double thresholding (e) Hysteresis

Figure 3.6: Result of the Canny edge detection method after each step.

$$E_{snake} = \int_0^1 E_{int}(\mathbf{s}(p)) + E_{ext}(\mathbf{s}(p))dp \tag{3.6}$$

The internal energy controls the contour of the snake and consists of a continuity and a curvature term:

$$\begin{aligned} E_{int} &= E_{cont} + E_{curve} \\ &= \frac{1}{2}(\alpha(p)|\mathbf{s}'(p)|^2 + \beta(p)|\mathbf{s}''(p)|^2) \end{aligned} \tag{3.7}$$

In this equation, α controls the amount of stretch and β controls the amount of curvature. Thus, a lower α makes the snake more elastic and a lower β makes the snake less smooth and thus able to form corners.

On the other hand, we have the external energy, which is defined on the image. It consists of an image energy and some constraint energy given by the user:

$$E_{ext} = E_{img} + E_{const}$$

The E_{img} causes the snake to move towards features in the image, such as lines, edges or corners. It is given as follows ([12], Section 3):

$$E_{img} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term}$$

The line energy is defined as the intensity of the image $I(x, y)$, which will attract the snake to dark or light lines in the image, depending on the sign of w_{line}. The edge energy is based on the gradient of the image: $-|\nabla I|^2$. This draws the snake to edges. Last, E_{term} refers to the termination energy, that is used to detect corners,

for which the direction of the gradient is used. The mathematical description of this energy is quite extensive and since we will not use it in this research, we will not discuss them here.
The variables w indicate the weight for each type of energy. In this research, the active contour model is used to detect edges, hence w_{line} and w_{term} are set to zero and w_{edge} to one. Often, a gaussian filter is applied to the original image to make the edges smoother, when computing E_{edge}.

Now that we know how to interpret the energy of the snake (Equation 3.6), we continue to describe the method numerically. We will follow the implementation as described in [5], [10] and [19].

We want to find $\mathbf{s}(p)$ such that the following function is minimized:

$$E_{snake} = \int_0^1 \frac{1}{2}(\alpha(p)|\mathbf{s}'(p)|^2 + \beta(p)|\mathbf{s}''(p)|^2) + E_{ext}(\mathbf{s}(p))dp \tag{3.8}$$

The curve for which this holds, is the final snake. We solve this problem by the Euler-Lagrange equation, which can be used to solve an optimization problem by rewriting it into a differential equation [6]. We assume $\alpha(p)$ and $\beta(p)$ are constants that do not depend on p. For function 3.8, we introduce the integrand

$$L(p, \mathbf{s}, \mathbf{s}', \mathbf{s}'') \coloneqq \frac{1}{2}(\alpha|\mathbf{s}'(p)|^2 + \beta|\mathbf{s}''(p)|^2) + E_{ext}(\mathbf{s}(p)) \tag{3.9}$$

The corresponding Euler-Lagrange equations for this problem are ([5], p. 18):

$$\frac{\partial L}{\partial \mathbf{s}} - \frac{d}{dp}(\frac{\partial L}{\partial \mathbf{s}'}) + \frac{d^2}{dp^2}(\frac{\partial L}{\partial \mathbf{s}''}) = 0 \tag{3.10}$$

$$\nabla E_{ext} - \alpha\frac{d^2\mathbf{s}}{dp^2} + \beta\frac{d^4\mathbf{s}}{dp^4} = 0 \tag{3.11}$$

$$\alpha\frac{d^2\mathbf{s}}{dp^2} - \beta\frac{d^4\mathbf{s}}{dp^4} - \nabla E_{ext} = 0 \tag{3.12}$$

The gradient of the external energy is a force, so from now on we write $\mathbf{F}_{ext}$ instead of $-E_{ext}$. Equation 3.12 can be hard to solve, so we convert the snake into a function of time: $\mathbf{s} = \mathbf{s}(p, t)$. We assume that when a minimum has been found, the derivative of $\mathbf{s}$ with respect to time is zero ([19], p. 20). In this case we get Equation 3.12 back. Thus, the snake converges to a stable state, where the curve does not change in time, so we can look for a stationary solution of the equation:

$$\frac{\partial \mathbf{s}}{\partial t} = \alpha\frac{\partial^2\mathbf{s}}{\partial p^2} - \beta\frac{\partial^4\mathbf{s}}{\partial p^4} + \mathbf{F}_{ext} \tag{3.13}$$

Since the curve $\mathbf{s}$ consists of an x- and y-component, we will from now on separate the two components and try to find an expression for both s^x and s^y. The terms f_x and f_y denote the two components of the external force $\mathbf{F}_{ext}$. Equation 3.13 is

separated as follows:

$$\frac{\partial \mathbf{s}^x}{\partial t} = \alpha \frac{\partial^2 \mathbf{s}^x}{\partial p^2} - \beta \frac{\partial^4 \mathbf{s}^x}{\partial p^4} + \mathbf{f}_x \tag{3.14}$$

$$\frac{\partial \mathbf{s}^y}{\partial t} = \alpha \frac{\partial^2 \mathbf{s}^y}{\partial p^2} - \beta \frac{\partial^4 \mathbf{s}^y}{\partial p^4} + \mathbf{f}_y \tag{3.15}$$

To solve these equations numerically, we make a discrete approximation as proposed in [10], p. 19. Instead of $\mathbf{s}(p,t)$ we will write $\mathbf{s}_{i,t}$, where $i = 0, 1, \ldots, N-1$ and N is the number of points of the curve. Since we have a closed contour, we have: $\mathbf{s}_{-1,t} = \mathbf{s}_{N-1,t}$, $\mathbf{s}_{0,t} = \mathbf{s}_{N,t}$, $\mathbf{s}_{1,t} = \mathbf{s}_{N+1,t}$ and so on ([19], p. 19).

Subsequently, the derivatives with respect to p can be approximated by finite differences. The second and fourth derivatives on the curve at position i, with step size $h = \frac{1}{N}$ defined on the contour, are given by the central differences:

$$\xi_i^{(2)} = \frac{\xi_{i-1} - 2\xi_i + \xi_{i+1}}{h^2} \tag{3.16}$$

$$\xi_i^{(4)} = \frac{\xi_{i-2} - 4\xi_{i-1} + 6\xi_i - 4\xi_{i+1} + x_{i+2}}{h^4} \tag{3.17}$$

Substituting the approximations of the derivatives in equations 3.14 and 3.15 gives the following equations:

$$\begin{aligned} \frac{\partial s_{t,i}^x}{\partial t} = & \frac{\alpha}{h^2}(s_{t,i-1}^x - 2s_{t,i}^x + s_{t,i+1}^x) - \frac{\beta}{h^4}(s_{t,i-2}^x - 4s_{t,i-1}^x + 6s_{t,i}^x - 4s_{t,i+1}^x + s_{t,i+2}^x) \\ & + f_x(s_{t,i}^x, s_{t,i}^y) \qquad (3.18) \\ = & -\frac{\beta}{h^4} s_{t,i-2}^x + (\frac{\alpha}{h^2} + \frac{4\beta}{h^4}) s_{i-1,t}^x + (-\frac{2\alpha}{h^2} - \frac{6\beta}{h^4}) s_{i,t}^x + (\frac{\alpha}{h^2} + \frac{4\beta}{h^4}) s_{i+1,t}^x \\ & - \frac{\beta}{h^4} x_{t,i+2} + f_x(s_{t,i}^x, s_{t,i}^y) \qquad (3.19) \end{aligned}$$

Similarly for the y-component:

$$\begin{aligned} \frac{\partial s_{t,i}^y}{\partial t} = & \frac{\alpha}{h^2}(s_{t,i-1}^y - 2s_{t,i}^y + s_{t,i+1}^y) - \frac{\beta}{h^4}(s_{t,i-2}^y - 4s_{t,i-1}^y + 6s_{t,i}^y - 4s_{t,i+1}^y + s_{t,i+2}^y) \\ & + f_y(s_{t,i}^x, s_{t,i}^y) \qquad (3.20) \\ = & -\frac{\beta}{h^4} s_{t,i-2}^y + (\frac{\alpha}{h^2} + \frac{4\beta}{h^4}) s_{i-1,t}^y + (-\frac{2\alpha}{h^2} - \frac{6\beta}{h^4}) s_{i,t}^y + (\frac{\alpha}{h^2} + \frac{4\beta}{h^4}) s_{i+1,t}^y \\ & - \frac{\beta}{h^4} s_{t,i+2}^y + f_y(s_{t,i}^x, s_{t,i}^y) \qquad (3.21) \end{aligned}$$

These equations, excluding the external forces, can be written in matrix notation, with the following matrix $\mathbf{A}$ of size N $\times$ N:

$$\begin{pmatrix}
\frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & -\frac{\beta}{h^4} & 0 & \cdots & 0 & -\frac{\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} \\
\frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & -\frac{\beta}{h^4} & 0 & \cdots & 0 & -\frac{\beta}{h^4} \\
-\frac{\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \ddots & & & 0 \\
0 & -\frac{\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \ddots & & & \vdots \\
\vdots & 0 & \ddots & \ddots & \ddots & & & 0 \\
0 & \vdots & & & & \ddots & & -\frac{\beta}{h^4} \\
-\frac{\beta}{h^4} & 0 & & & & & \ddots & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} \\
\frac{\alpha}{h^2}+\frac{4\beta}{h^4} & -\frac{\beta}{h^4} & 0 & \cdots & 0 & -\frac{\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4} & \frac{\alpha}{h^2}+\frac{4\beta}{h^4}
\end{pmatrix}$$

Equations 3.19 and 3.21 may now be compactly written as:

$$\frac{\partial \mathbf{s}_t^x}{\partial t} = \mathbf{A}\mathbf{s}_t^x + f_x(\mathbf{s}_t^x, \mathbf{s}_t^y) \tag{3.22}$$

$$\frac{\partial \mathbf{s}_t^y}{\partial t} = \mathbf{A}\mathbf{s}_t^y + f_y(\mathbf{s}_t^x, \mathbf{s}_t^y) \tag{3.23}$$

Where $\mathbf{s}_t^x$ and $\mathbf{s}_t^y$ are the position vectors containing the N coordinates of the snake at time t.

Next, we choose a time integration method to approximate the time derivatives on the left-hand side. Because of simplicity and stability, we apply the Euler Backward method with time step Δt to Equations 3.22 and 3.23. This yields:

$$\frac{\mathbf{s}_t^x - \mathbf{s}_{t-1}^x}{\Delta t} = \mathbf{A}\mathbf{s}_t^x + f_x(\mathbf{s}_t^x, \mathbf{s}_t^y) \tag{3.24}$$

$$\frac{\mathbf{s}_t^y - \mathbf{s}_{t-1}^y}{\Delta t} = \mathbf{A}\mathbf{s}_t^y + f_y(\mathbf{s}_t^x, \mathbf{s}_t^y) \tag{3.25}$$

Here we neglected the truncation error of order $\mathcal{O}(\Delta t)$. Since the external forces are non-linear in $\mathbf{s}_t$, the equations might be hard to solve. Hence, we evaluate these functions at $\mathbf{s}_{t-1}^x$ and $\mathbf{s}_{t-1}^y$ instead of $\mathbf{s}_t^x$ and $\mathbf{s}_t^y$. Shifting gives an error of order $\mathcal{O}(\Delta t)$. This error is of the same order as the truncation error, hence evaluating the external force at a previous time step does not increase the order of the error. This gives the following equations:

$$\frac{\mathbf{s}_t^x - \mathbf{s}_{t-1}^x}{\Delta t} = \mathbf{A}\mathbf{s}_t^x + f_x(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y) \tag{3.26}$$

$$\frac{\mathbf{s}_t^y - \mathbf{s}_{t-1}^y}{\Delta t} = \mathbf{A}\mathbf{s}_t^y + f_y(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y) \tag{3.27}$$

To make Equation 3.26 explicit, we rewrite it:

$$\frac{\mathbf{s}_t^x - \mathbf{s}_{t-1}^x}{\Delta t} = \mathbf{A}\mathbf{s}_t^x + f_x(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y) \tag{3.28}$$

$$\mathbf{s}_t^x = \mathbf{s}_{t-1}^x + \Delta t \mathbf{A}\mathbf{s}_t^x + \Delta t f_x(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y) \tag{3.29}$$

$$\mathbf{s}_t^x - \Delta t \mathbf{A}\mathbf{s}_t^x = \mathbf{s}_{t-1}^x + \Delta t f_x(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y) \tag{3.30}$$

$$\mathbf{s}_t^x(\mathbf{I} - \Delta t \mathbf{A}) = \mathbf{s}_{t-1}^x + \Delta t f_x(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y) \tag{3.31}$$

$$\mathbf{s}_t^x = (\mathbf{I} - \Delta t \mathbf{A})^{-1}(\mathbf{s}_{t-1}^x + \Delta t f_x(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y)) \tag{3.32}$$

Now we can calculate the position vector $\mathbf{s}^x$ at time t, since all values on the right-hand side are known. By rewriting Equation 3.27 in a similar way, we also obtain the equation for the position vector $\mathbf{s}^y$ at time t:

$$\mathbf{s}_t^y = (\mathbf{I} - \Delta t \mathbf{A})^{-1}(\mathbf{s}_{t-1}^y + \Delta t f_y(\mathbf{s}_{t-1}^x, \mathbf{s}_{t-1}^y)) \tag{3.33}$$

So for every t, we obtain a new curve, which causes the snake to "move". These equations have to be solved iteratively to make the snake evolve towards a shape that minimizes the energy function of Equation 3.8. The output of this method is the snake $\mathbf{s}(p)$ for which a stationary solution of Equation 3.13 is found.

In this research, we used the Python implementation as given in [9]. This implementation considers a pre-set number of iterations. Additionally, the initial curve often an ellipse, with $p \in [0, 2\pi]$ instead of $p \in [0, 1]$.

While the output of the algorithm is straightforward, the input of this algorithm is quite extensive. The following parameters should be fine tuned to obtain a good result:

- the initial curve $\mathbf{s}(p)$
- α, which controls the continuity of the curve
- β, which controls the curvature of the curve
- the time step Δt
- the amount of smoothing σ
- the number of iterations n

In the implementation, the step size h is taken equal to 1, since this variable is intertwined with the variables α and β.

3.4.2 Result

Figure 3.7 shows the result when active contour model is applied on the measured Shepp-Logan Phantom.

The red curve is the initial curve, parametrized by:

Figure 3.7: Active contour model is applied on the measured Shepp-Logan Phantom.

$$\begin{cases} x(p) = 66 + 27\cos(p) \\ y(p) = 60 + 40\sin(p) \end{cases}$$

for $p \in [0, 2\pi]$. The blue curve is the converged snake that follows the contour of the Shepp-Logan Phantom.

The following values were used to obtain this result: $\alpha = 0$, $\beta = 0.05$, $\Delta t = 20$, $\sigma = 5$ and we used 30 iterations.

3.5 Completing edges

We propose a new algorithm which can be used to detect and complete unfinished edges, in order to create a closed boundary. Before edges can be completed, their ends have to be found. In section 3.5.1 we explain how these so-called loose-ends are detected. Subsequently, in section 3.5.2 is explained how edges are connected. In this research, only binary images are used, containing solely black and white pixels.

3.5.1 Finding loose ends

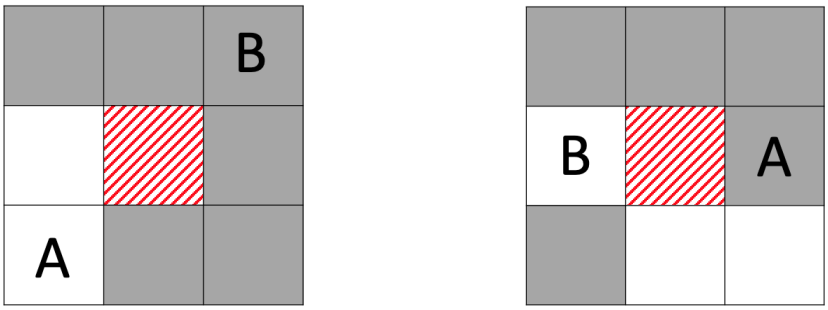
In order to complete an edge, the location of the loose end of the edge is needed. This is the last point of an edge of which the rest of the edge is grown. A pair of criteria are drawn up to define a loose end in a binary image:

1. A pixel p with only one neighbor in its 8-connected neighborhood is marked as a loose end.
2. A pixel p with exactly two neighbors in its 8-connected neighborhood are marked as a loose end if:
   (a) Either the pixel before or the pixel after pixel p is suppressed while the other one is preserved, and
   (b) The pixel before or after pixel p that is suppressed, is not connected to a pixel besides pixel p in the 8-connected neighborhood of pixel p.

To obtain the pixels before and after, the direction of the edge is used (recall that this direction is perpendicular to the direction of the gradient, that can be calculated by Equation 3.3), which is in turn obtained by Sobel edge detection. This should be applied to the original image. The angle is rounded to one of the eight directions: there are four orientations possible (horizontal, vertical, diagonal and anti-diagonal) and each orientation can be reversed. This procedure is similar to the non-maximum suppression, since the same intervals are used as shown in Figure 3.5. The pixels before and after are found in the line of the orientation. For instance, the edge direction of the red pixel in Figure 3.8a is -150°, which is rounded to a reversed diagonal orientation. So, the pixels before and after are located at the upper right and lower left of the red pixel respectively.

Example

To illustrate why the last condition needs to be extended with subcondition b, we consider two examples in Figure 3.8. The gray pixels are background and the white pixels are edges. The pixel that is highlighted in red, referred to as pixel p, is checked for a loose end and hence also part of the edge. The letters B and A indicate the pixel before and after. It can be seen that in both examples, either B or A is white, while the other one is suppressed, so criterion 2a is met in both cases.



(a) Pixel that meets criterion 2b

(b) Pixel that does not meet criterion 2b

Figure 3.8: Two examples of a pixel (highlighted in red) with two neighbors (marked with white) in its 8-connected neighborhood. The pixel before and after are marked with B and A.

Now, let's take a look at the difference between Figure 3.8a and Figure 3.8b. In Figure 3.8a, the pixel before is suppressed and not connected to any other pixel than the highlighted pixel p. Hence criterion 2b is satisfied and pixel p is marked as a loose end.
On the other hand, in Figure 3.8b, the pixel after is suppressed and connected to two other pixels: the pixel below A and the pixel below pixel p. This connection makes that the edge does not stop at pixel p but continues. Note that when only the pixel below A *or* the pixel below p is marked, condition 2b is still not met and the highlighted red pixel is still not a loose end.

Result

Figure 3.9 shows the loose ends in red. This is an image of the edge of the measured Shepp-Logan Phantom, which was found by executing Canny edge detection, without applying Gaussian blurring. To reduce the amount of loose ends, small groups of pixels, or single pixels (that are not surrounded by any neighbors) may be removed from the image. This is also done in Figure 3.9, for groups smaller than four pixels. The loose ends indicated in red are indeed the points one expects to be loose ends.

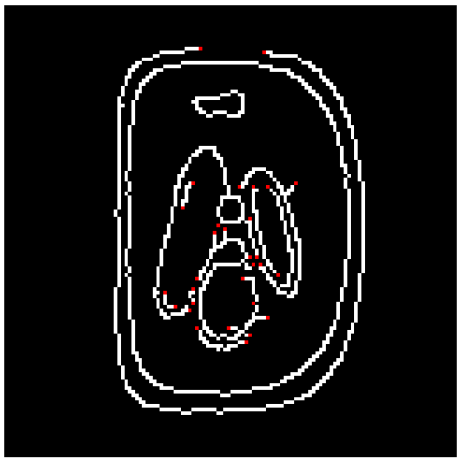


Figure 3.9: The loose ends of the edges of the measured Shepp-Logan Phantom, marked in red.

3.5.2 Connecting edges

When the loose ends are found, the edges can be connected. The same binary image is used, that was used for finding loose ends. Connecting is again done by using the direction of the loose end of the edge, in a similar way as in Section 3.5.1. Namely, the orientation of the loose end in combination with the gray values of the pixels before and after, indicate what the position will be of the next pixel. Points that are marked as an edge, will get the new value of 1 (white), so the output image with the new edges will still be binary.

For instance, recall Figure 3.8a, where the loose end is highlighted red. Suppose its direction is still -150°, which is rounded to a reversed diagonal orientation, as indicated with the blue arrow in Figure 3.10. Hence, the next pixel will either be at the upper right or the lower left. Note that the pixel after is white because it is already part of the edge. Therefore, the pixel before should be marked as an edge and it gets the value 1, as shown in Figure 3.10. Now, this pixel becomes a new loose end and the procedure is repeated, until the edge reaches a pixel that is already part of the image.
Notice that the direction of the edge does not necessarily point to the next point of the edge, for example in Figure 3.10 where the edge is grown in the opposite direction. The position of the next edge point depends solely on the intensities of the pixels previous and next.

When using this algorithm, the odds are that edges will be drawn that do not improve the image or that do not exist at all. To reduce this effect, we make some

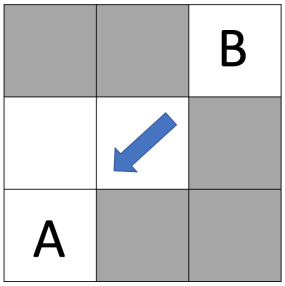


Figure 3.10: An edge is grown from the loose end of Figure 3.8a, since the pixel before, indicated with B, is now also part of the edge. The blue arrow indicates the direction of the loose end.

assumptions in addition to the above procedure.

First, when a connection consists of more than n pixels, the connection is not taken into account, so the edge is suppressed. In this research, n will be taken equal to 20.

Secondly, it may occur that two pixels that are marked as loose ends, lie close to each other. When the distance between two loose ends is smaller than or equal to d (calculated by the Euclidean distance), the two ends will not be connected as proposed above, but will be connected to each other. In this research, we will use $d = 2$, so only loose ends that lie on the same horizontal or vertical line, with one pixel in between, will be connected.

Last, when the edge arrives in the 8-connected neighborhood of another loose end, the edge is connected to this loose end, instead of going further until an image point is found. In this case, two loose ends are connected. Important to note is that this connection can also be grown in the reverse direction, starting from the other loose end. This may result in a different edge. We do not want to plot both connections, so to choose the best connection, the length (the number of pixels) of the two edges is compared. The connection with the smallest length is preserved and the longer edge is taken into account. It is necessary to make sure the reverse edge is indeed connected to the the initial loose end of which the edge was first grown.

Result

Figure 3.11a shows the connected edges, where the loose ends are used as shown in Figure 3.9. Note how the most upper edge is now nicely extended and at some places the edge makes a curve, see Figure 3.11b. This image is zoomed in on the top of the right ellipse of Figure 3.11a.

Example

We illustrate the procedure with an example. We will show stepwise how the edge in Figure 3.11b is achieved. Figure 3.12 shows a recreation of this image, with directions of the edge in blue and coordinates at the top and side. The grey points are the background, the white points are already edges and the red points are loose ends, so also edges. The direction of the edge is obtained by applying Sobel edge

(a) All edges that are now connected. (b) Close-up of a completed edge.

Figure 3.11: Edges of the measured Shepp-Logan Phantom are now connected, shown in red.

detection on the original image, in this case the measured Shepp-Logan Phantom.

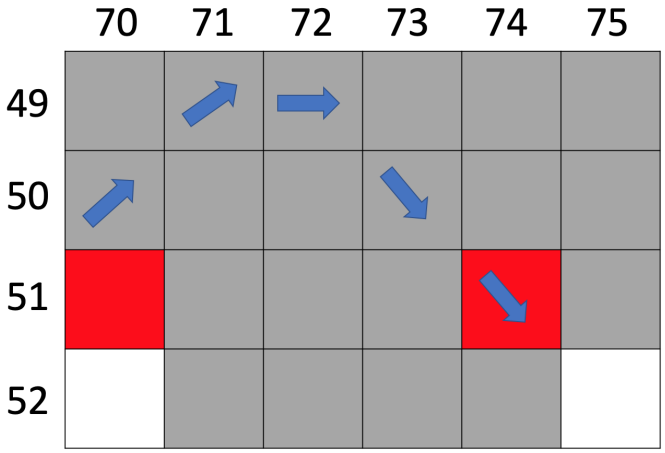


Figure 3.12: Recreation of Figure 3.11b, to show how this edge is achieved step by step.

We start at the loose end located at (51,74)[1]. This is the right red point in Figure 3.12. This is indeed a loose end, since it has only one neighbor. First, we want to know in which direction the edge should be grown, in other words if we have to move backwards or forwards. This step is only done at the beginning, so one time per edge. The direction of the loose end is -30°, so its orientation is reversed anti-diagonal, shown in Figure 3.12 indicated with the blue arrow. We obtain the direction in which the edge is grown, by looking at the values of the pixels before and after. The previous and next pixel are located at (50,73) and (52,75), with values 0 and 1 (dark grey and white in Figure 3.12) respectively. Since the next pixel, on the lower right of the loose end, has value 1, it is already part of an edge, so that means we have to move backwards. In this case that means we go upwards, and the previous pixel becomes part of the edge. Therefore, the previous pixel at (50,73), on the upper left of the loose end, gets the value 1.

[1]The x- and y-coordinates of pixels in an image are reversed with respect to the usual coordinate system. This means that the x-axis is vertical and the y-axis is horizontal.

Note that the direction of the edge (in this case -30°, reversed antidiagonal) is thus not necessarily the direction in which the edge is grown. This depends on the values of the pixels previous and next.

The new loose end is now at coordinate (50,73). This pixel has a direction of -45°, so the orientation is again reversed antidiagonal. Since we already know that we move backwards, we only need the previous pixel, which is located at (49,72). This pixel gets the value 1 and becomes the new loose end.

The direction of pixel (49,72) is 0°, which is a horizontal orientation, therefore the previous pixel lies next to the loose end, at coordinate (49,71). This becomes the new loose end.

The direction of pixel (49,71) is 55°, which means we move diagonally. The previous pixel is located at (50,70), which now becomes the new loose end.

We now note that the new loose end at (50,70) lies in the 8-connected neighborhood of a loose end at coordinate (51,70), since our pixel is located above this loose end. We do not have to look at the direction of this pixel, since we make the connection between the pixel at (50,70) and the loose end at (51,70). This means the edge we have grown is connected and the algorithm stops.

At the last step, we have to grow the edge in opposite direction, to check if this edge is eventually smaller. We grow the edge thus starting from the left loose end at (51,70) in the same way as earlier. The result is shown in Figure 3.13. The edge indeed connects to the loose end at (51,74). The length of this edge is six pixels, while the edge of Figure 3.11b consists of four pixels, so the edge that was grown before (as in Figure 3.11b) is preserved.

Figure 3.13: The edge is grown in the opposite direction, from left to right, between the same loose ends as in Figure 3.11b. This results in a longer connection.

When the edge from one of the loose ends were connected to a white pixel in the image, instead of a loose end, both edges in Figure 3.11b and Figure 3.13 would be shown in the output image (the image containing all the completed edges). Nevertheless, by connecting the loose ends to another loose end, only one edge is shown, reducing unnecessary lines and double edges.

In short, we discussed a method in which unfinished edges can be completed.

The input consists of two objects. First, we have a binary image containing the unfinished edges. Secondly, we need the direction of the edge, which is obtained by applying Sobel edge detection on the original image, so the image of which the edges are found. By using these two, the location of the loose ends can be found and a new connected can be created.
This procedure works best in combination with the Canny edge detection. Namely, Canny edge detection returns an image with edges of (mostly) one pixel wide and the extension also add edges of one pixel wide.

4 K-means clustering

In this chapter, we will discuss the k-means clustering algorithm. This is an algorithm by which all the pixels in an image are divided into k clusters, based on their intensities. First, we will explain how the method works and show some results. Next, in section 4.3 we will elaborate on the choice of the initial cluster centers, which is partly input of the method. Finally, an extension of the method is given, which makes it possible to reassign pixels after the k-means clustering is completed.

4.1 Method description

The k-means clustering algorithm [13] is a method which is used to divide data into k different clusters. It assigns each data point to the nearest cluster. In the case of image segmentation, "nearest" means the smallest difference in gray-scale value. Within each cluster, the variance should be as small as possible.

The input of the k-means clustering algorithm is the image and a list of k cluster centers, which are characteristic pixel values (not necessarily values from the image). The first step is to calculate for each pixel the distance to each cluster center, or more precisely: the difference between its gray-scale value and each of the k centers. The pixel is appended to the cluster for which the difference between the pixel's gray-scale value and the cluster center is minimized. After this is done for each pixel, the cluster centers are updated. This is done by taking the mean of each cluster. If these new centers are different from the centers before updating, we repeat the procedure. If the cluster centers stay the same, the algorithm is done.

The k in the method's name, refers to the amount of clusters and the *means* refers to taking the mean in each step, causing the cluster centers to change.

The pseudocode of the k-means clustering algorithm is described in Algorithm 1.

Algorithm 1 K-means clustering

Input: image I, a list of k initial cluster centers C=$\{c_1, c_2, \ldots, c_k\}$
Output: a list of labels, a list of cluster centers
while *cluster centers do not change* **do**
    for *each pixel in I* **do**
        Assign pixel to cluster for which the distance is the smallest, by giving it a label
    end
    Find new cluster centers by taking the mean of each cluster
end
return list of labels, list of cluster centers

The k-means clustering algorithm returns two objects: a list of labels and a list of cluster centers. First, the list of the labels refers to the cluster each pixel is assigned to. For instance, when we have five clusters, the list of labels consists of the values 0, 1, 2, 3 and 4. The length of this list equals the amount of pixels in the image, since each pixel is labeled.
The second object is a list of the pixel values of each cluster center, so its length is k. Combining these two lists results in an image. For instance, when the first pixel has label 2, it gets the third value in the list of cluster centers. So pixels with the same label, belong to the same cluster and the pixels in each cluster share the same value, which is the mean of the cluster.

The algorithm aims to minimize the sum squared error of the variance within clusters:

$$\sum_{n=1}^{N} \sum_{k=1}^{K} ||x_n - c_k||^2 \gamma_{nk} \tag{4.1}$$

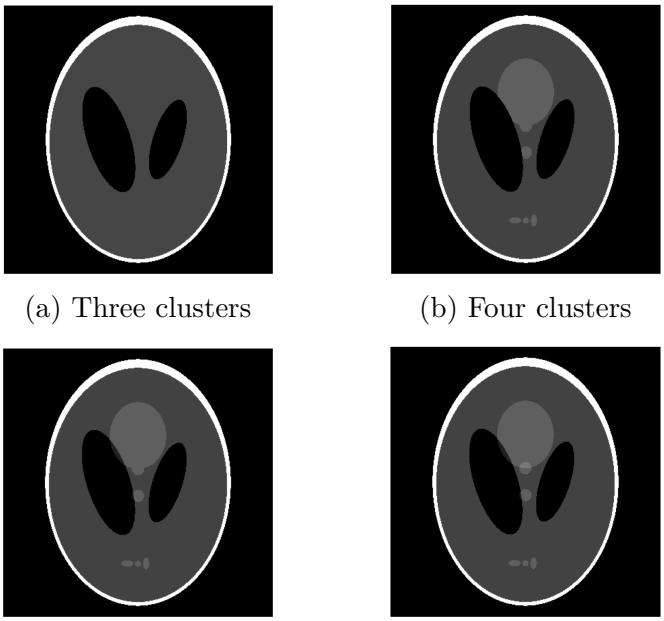
In this equation, x_n is the value of pixel n and c_k is the value of cluster center k. The variables N and K refer to the number of pixels and the number of clusters respectively. The last variable γ_{nk} is an indicator function which is 1 when pixel n belongs to center k and 0 otherwise.

4.2 Result

In Figure 4.1 is the k-means clustering algorithm applied to the simulated Shepp-Logan Phantom, with several numbers of clusters. Since the segments are very clear by eye, it is easy to guess what the algorithm should return. Figure 4.1d is exactly the same as the original Shepp-Logan Phantom, since this image already contains six unique gray-scale values. These unique values are the same values that the k-means clustering algorithm returned.

4.3 Influence of the choice of initial cluster centers

As mentioned in the description of the algorithm, the main input is the initial cluster centers. Each pixel will be assigned to one of the clusters based on the smallest distance to each center. The question arises whether the choice of the initial centers



(a) Three clusters (b) Four clusters

(c) Five clusters (d) Six clusters

Figure 4.1: K-means clustering done on the measured Shepp-Logan Phantom, with different values of k.

influences the output of the algorithm.

There are several ways to choose the initial centers. For example:

- One can inspect the image and select the pixel values of the segments one wants to get out. This can also be done by examination of the image's histogram. This graph shows the frequency of each intensity, so one could choose for instance the values that are common.
- One can create an evenly spaced interval. For example, if the image should be segmented into four clusters, one could use the pixel values 0, 0.33, 0.67 and 1 as initial clusters.
- One could choose the centers based on the Otsu threshold, as proposed in [22]. The Otsu threshold [21] is the gray-value that optimally separates the foreground and the background, by maximizing the variance between the two classes. The initial cluster centers should be chosen such that the difference between the mean m of the centers and the Otsu threshold O is less than a preselected threshold T: $|m - O| < T$.

In some cases, the choice of the initial cluster centers determines the result. For instance, Figure 4.2 shows three different clustered images, where k is equal to four. Each of these images started with different cluster centers that are noted below each image. These different centers result in different outputs and a different sum squared error. Note that it seems like Figure 4.2c has only three clusters, instead of four. However, the fourth segment is quite small and located in the left black ellipse, which makes it rather invisible.
In Figure 4.1 are for each of the figures, the initial cluster centers chosen such that

the sum squared error between the image and the clustered image is minimized. These centers are found by trial and error.

(a) 0, 0.1, 0.4, 1 (b) 0, 0.3, 0.4, 1 (c) 0, 0.1, 0.2, 0.3

Figure 4.2: Different initial cluster centers result in different outcomes.

However, when we use different cluster centers to cluster the measured Shepp-Logan Phantom, the outcome is always the same. This difference may occur since the simulated Shepp-Logan Phantom is even: it consists of only six different gray-scale values and each segment has its own value. Contrary, there are almost no pixels with the same value in the measured Shepp-Logan Phantom. So by my experience, the measured Shepp-Logan Phantom seems to be more robust, because of the many different gray-scale values. The k-means clustering algorithm may be sensitive to different initial cluster centers, but also may not, depending on the character of the image.

4.4 Reassigning pixels to different clusters

We now propose an extension to the k-means clustering algorithm, by which pixels can be transferred to a different cluster than they were originally assigned to.

4.4.1 Method description

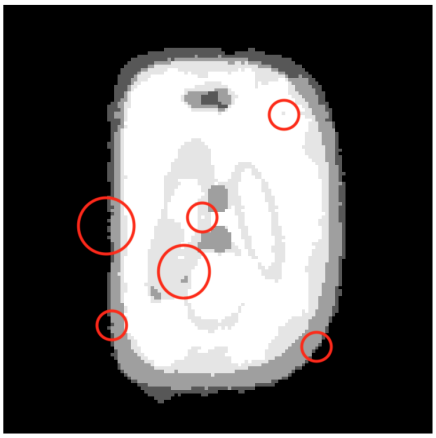
In some cases, the k-means clustering algorithm returns an image in which single pixels or small groups of pixels are assigned to the wrong cluster. For instance, in a white-colored cluster is a gray pixel located. We know, by eye, that this pixel should also be part of the white cluster. How can this pixel be relabeled, and thereby assigning it to a different cluster?

First, after the k-means clustering is done, all single pixels or small groups of pixels, that do not have neighbors of the same color, have to be detected. Then, for each of these pixels, the sum squared error (SSE) between the pixel value in the original image and each of the existing clusters is calculated (except for the cluster it was initially assigned to). The cluster that minimizes the SSE, is the cluster the pixel is assigned to. The cluster that the pixel was originally assigned to is not taken into account when calculating the SSE, since this is of course the cluster that minimizes the SSE. Hence, then taking the smallest SSE will not result in a change of cluster. In short, the pixel is assigned to the cluster for which the distance

between its gray value and the cluster center is second to lowest.
This method returns a new list of labels, while the list of the cluster values stays the same.

4.4.2 Result

Figure 4.3a shows the result after k-means clustering on the measured Shepp-Logan Phantom shown, with five clusters. Figure 4.3b shows the result when groups of smaller than five pixels are reassigned. Since the differences might be hard to find, some groups that are reassigned are marked with a red circle in Figure 4.3a. These are the groups of pixels that are "gone" in Figure 4.3b. Even though the difference might be subtle, it makes the image in general less noisy.



(a) K-means clustering is done on the measured Shepp-Logan Phantom, with five clusters.

(b) Small groups of pixels are reassigned to different clusters.

Figure 4.3: The result of reassigning pixels to new clusters.

5 Region growing

In this chapter, we will discuss the method region growing. This is a method based on the idea that neighboring pixels share the same properties and hence form one connected region. We first explain how the method works, next we show some results.

5.1 Method description

Region growing [1] is a method to obtain a group of pixels of an image, that is connected and share some predefined criteria. During region growing, a region is grown starting with one or more seeds. Such a seed is a coordinate of the image. The neighboring pixels of a seed is checked for a homogeneity criterion. The "neighboring pixels" refer to all the pixels that are contained in the 8-connected neighborhood of a certain pixel (see Figure 3.3a). When a neighboring pixel meets the criterion, it is added to the region. Then for each neighbor that is added to the region, their neighbors should also be checked for homogeneity. Therefore the region is getting bigger and bigger every iteration.

In this research, the homogeneity criterion depends solely on the intensity. Namely, the criterion is met when the difference between the intensities of pixel p and its neighboring pixel n is smaller than a certain threshold T: $|p - n| < T$.

The pseudocode of the region growing method is described in Algorithm 2. Initially, the list S contains only the seed point(s) that are given by the user, but this list grows every iteration. The variable *connected* contains up to eight neighbors of the point that meet the homogeneity criterion. Hence, during each iteration, up to eight points can be added to S. Namely, each point in *connected* is added to the region, consequently each of these points' neighbors may also be part of the region, so these are added to the list S. The list R contains all the pixels that form the region.

Algorithm 2 Region growing

Input: a list with the initial seeds S, threshold T
Output: a list of points R that form the region
while $S \neq \emptyset$ **do**
    point $\leftarrow$ the first element of S
    connected $\leftarrow$ neighbors(point,T)
    for $c \in$ *connected* **do**
        add c to R
        if c *not yet processed* **then**
            add c to S
        end
    end
    Remove point from S
end
return R

5.2 Result

Figure 5.1 shows two results, where region growing is done on the simulated Shepp-Logan Phantom. The initial seed is indicated with a red dot. Since each of the parts in this image share the same value, it is easy to guess beforehand what should come out of the algorithm. Indeed, the outcome meets the expectations.

(a) (b)

Figure 5.1: An example of region growing, done on the simulated Shepp-Logan Phantom, with the initial seed marked in red.

6 Numerical experiments

In this chapter, we evaluate the image segmentation methods on two independent images: the MRI scan of an apple and the CT scan of a brain. First, we evaluate the k-means clustering algorithm combined with region growing. Additionally, we evaluate how the choice of the initial cluster centers influence the result. Furthermore, we discuss the edge based methods and show how these can be combined with k-means clustering and region growing. Finally, the active contour model is evaluated.

6.1 K-means clustering and region growing

We apply the k-means clustering algorithm on the CT scan. By inspecting the image, we determine that it consist of three different gray tones, hence we take $k = 3$. The initial cluster centers that we use as input consists of the values 0, 0.5 and 1. These values correspond approximately to each of the three gray tones. Executing the k-means algorithm results in Figure 6.1a. The method returns the separate segments as expected.
Region growing gives us the opportunity to obtain solely the area of the ventricles and to measure its size. The result is shown in Figure 6.1b and the area consists of 17287 pixels. For comparison, the total image consists of $320 * 320 = 102400$ pixels.

(a) K-means clustering is applied on the CT scan, with three clusters.

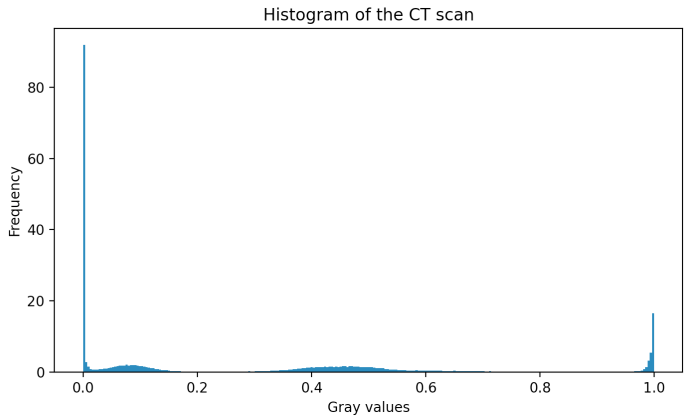
(b) Region growing applied on the ventricles of Figure 6.1a

Figure 6.1: Result of k-means clustering and region growing.

6.1.1 The influence of different cluster centers

When different initial cluster centers are used to cluster the CT scan, for instance 0, 0.1, 0.2, the outcome (the clustered image and the sum squared error) is exactly the same. However, the running time is twice as long: executing the k-means clustering algorithm with 0, 0.5 and 1 as initial centers took approximately 6.5 seconds, while the algorithm took 13.5 seconds with 0, 0.1, 0.2 as initial cluster centers. We can conclude that, even though the outcome with different initial cluster centers is the same, choosing accurate centers is profitable for a smooth execution of the algorithm.

A different way to choose the (amount of) initial cluster centers is by looking at the histogram. Figure 6.2a shows the histogram of the CT scan, with four peaks at the gray values 0, 0.08, 0.45 and 1. Figure 6.2b shows the result when these values are used as initial cluster centers. Note that uneven spots with two black tones in the area of the ventricles are now visible. These arose by adding the value 0.08 to the list of initial cluster centers. The ventricles of the original image consisted of pixels with gray values between 0 and approximately 0.1. Therefore, considering two small value such as 0 and 0.08 as initial cluster centers, results in oversegmentation of the ventricles. With oversegmentation, we refer to the process where objects itself are also segmented, resulting in an segmentation that is too refined [7]. Additionally, note the few black pixels in the gray area and the edge of the ventricles that is now less smooth in comparison with Figure 6.1a.



(a) The histogram of the CT scan, showing four peaks.

(b) The centers are obtained via the histogram, resulting in two black tones.

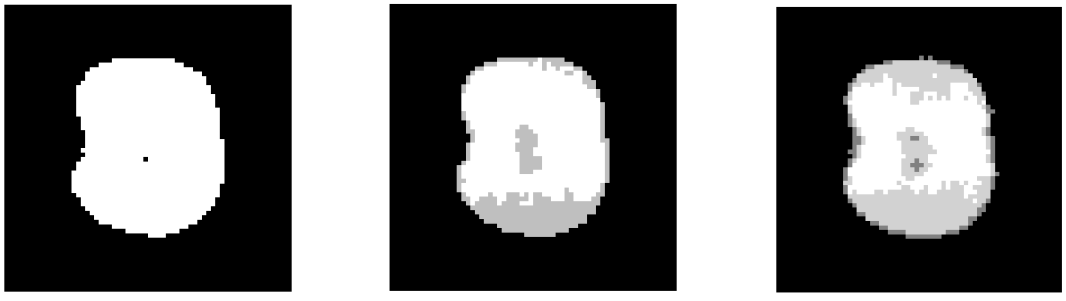
(c) The centers are obtained by inspecting, resulting in two gray tones.

Figure 6.2: Result of k-means clustering with four clusters.

Just like the ventricles, the area of the brain (the dark gray part outside the ventricles) is also uneven in the original image. This area consists of pixels with gray values between approximately 0.4 and 0.6 that also can be found in the histogram. So, executing the k-means algorithm with values 0, 0.4, 0.6 and 1 as initial centers will probably result in the same kind of spots as seen in Figure 6.2b. Indeed, Figure 6.2c shows two gray tones in the area of the brain, and therefore in oversegmentation.

Comparing Figure 6.2 with Figure 6.1a, we conclude that adding a fourth clusters does not result in a more useful representation. For this image, choosing $k = 3$ is therefore the most accurate option.

On the other hand, for the MRI scan of an apple, the choice of k is more relevant. After clustering, it would be ideal to retrieve the shape of the apple and to count the number of pits.
By inspection of the image, we note that it consists of two gray tones: the black background and the pits (a value of 0), versus the pulp of the apple that is a lighter tone (approximately 0.5-0.7). When we apply the k-means clustering on the apple with centers 0 and 0.6, we get Figure 6.3a. The apple and its background are separated and we see the nice shape of the apple. Unfortunately, the result does not show the pit (except for one pixel) that we expected. Using different initial cluster centers does not change the result.
Next we take $k = 3$. By inspecting the histogram, we choose the values 0, 0.4 and 0.6 as initial cluster centers. Figure 6.3b shows the result, where the pit is now visible. Nevertheless, we cannot distinguish between the two pits. Additionally, the border of the apple is shown, though it is quite broad at the bottom and thus not very accurate. Again, different initial cluster centers do not result in a different outcome.
Finally, when doing k-means clustering with four clusters, oversegmentation occurs, see Figure 6.3c. This is obtained by using the gray values 0, 0.4, 0.6 and 0.8 as initial centers. The latter value is added since there is a peak between in the histogram between gray values 0.6 and 0.8. The image is segmented into areas that are not necessarily different parts of the apple, see for instance the border that consists of two gray tones. However, the image does show two groups of pixels representing the pits, while in Figures 6.3a and 6.3b the two pits were only shown as a whole.

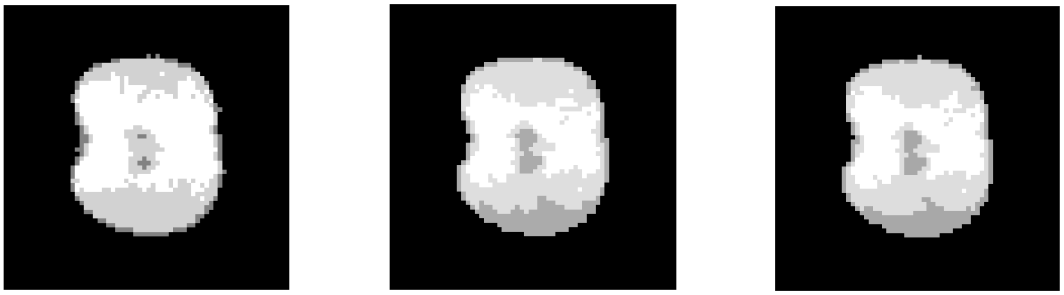


(a) Two clusters (b) Three clusters (c) Four clusters

Figure 6.3: K-means clustering is done on the MRI scan of an apple, with different values of k.

Segmenting the apple with a variation of four initial cluster centers, results in

different outcomes. We show some of the outcomes in Figure 6.4, in addition to Figure 6.3c. The values of the initial cluster centers are listed below the image.
The differences are sometimes subtle, compare for instance Figure 6.3c with Figure 6.4a and Figure 6.4b with Figure 6.4c. Only a couple of pixels seems to have different values. Comparing Figure 6.4a with Figure 6.4b does show a significant change.



(a) 0, 0.1, 0.4, 0.6 (b) 0, 0.6, 0.8, 1 (c) 0, 0.4, 0.6,1

Figure 6.4: The result of the k-means clustering applied on the apple is influenced by the initial cluster centers when using four clusters.

For this image, we can conclude that the choice of k depends on what one wants the get out of the algorithm. When the size of the apple has to be found, Figure 6.3a will be preferred. When the number of pits has to be found, most probably Figure 6.3c will be used. Last, Figure 6.3b best distinguishes between the pit, border, pulp and background.

6.1.2 Reassigning pixels

Next, we evaluate the extension proposed in Section 4.4, where (groups of) pixels can be transferred to different clusters.

When Figure 6.1a is examined more closely, we see that a small gray border consisting of groups of pixels runs along the white border (Figure 6.5b). We can reassign those groups by using the procedure described in Section 4.4. We apply it to groups consisting of three or less pixels. Figure 6.5a shows the result. The thin gray border is now part of the thicker white border (Figure 6.5c) and a group in the dark gray and black part are now gone.
The difference is quite subtle, however it is satisfying to see that the extension works as expected.

This procedure can also be used to connect separate regions that belong to the same cluster. For instance, recall Figure 6.3b, where we applied k-means clustering with three clusters. The gray cluster consists, besides the pit, of the apple's border. Nevertheless, by region growing, we determine that this border consists of three separate regions. Hence, we aim to connect these regions to form one region. To achieve this, we will also need some user input and we refer to the procedure of completing edges described in Section 3.5.

First, we want to know which clusters should be connected. We obtain the separate clusters by applying region growing on the clustered image. This results in

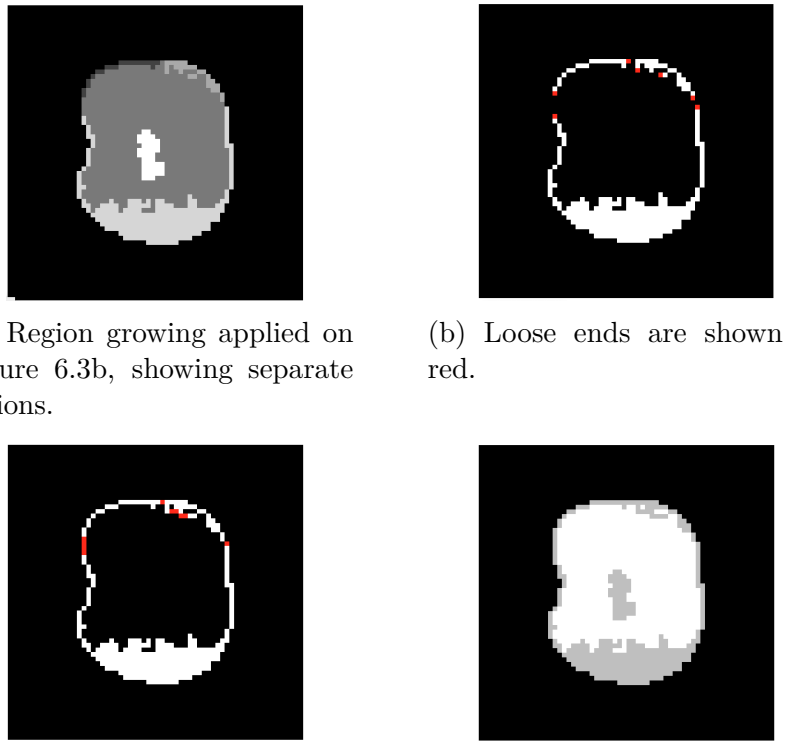
(a) Pixels are assigned to new clusters.

(b) Zoomed-in image of the border of Figure 6.1a.

(c) Zoomed-in image of the border of Figure 6.5a.

Figure 6.5: Groups consisting of three or less pixels got relabelled.

Figure 6.6a, where each region has its unique gray tone. We can now easily select the three clusters that should from the border.
Next, we find the loose ends of these clusters, these are shown in Figure 6.6b. Completing the edges results in Figure 6.6c. Indeed, the border is now connected and now consists of one region. Notice that Figure 6.6b also contains two loose ends in the upper right that are not ends of the border. These ends are connected to each other in Figure 6.6c. Even though this edge does not disturb the new image too much, one may choose to discard these two loose ends, since these are irrelevant and do not contribute to connecting separate regions.



(a) Region growing applied on Figure 6.3b, showing separate regions.

(b) Loose ends are shown in red.

(c) The loose ends are connected, shown in red.

(d) Final result where the border is now one region.

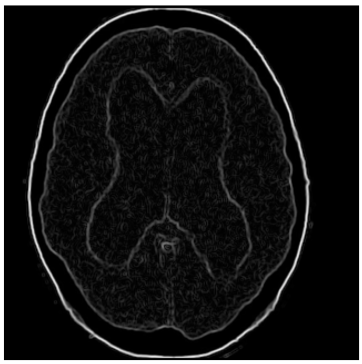
Figure 6.6: An example of how separate regions of a clustered image are connected to form one region.

The three clusters are now connected and form one cluster. However, in the clustered image we did not yet make changes, so for the final step we have to give the pixels that make up the connections (the red pixels in Figure 6.6c) a new value by relabelling. Figure 6.6d shows the result. We see that the gray border is now one connected region and our goal is achieved.

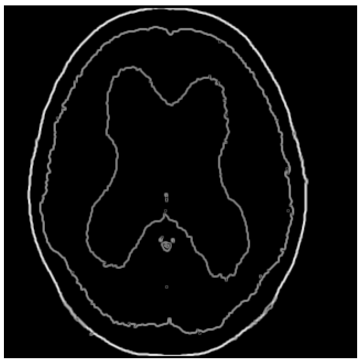
Note that this procedure does need some user input, since only the user can decide which clusters have to be connected and which loose ends are relevant. Furthermore, this is an example where preselected pixels are reassigned instead of all groups of $\leq n$ pixels. Therefore, this method can also be applied to relabel specific clusters or parts of clusters.

6.2 Sobel and Canny edge detection

Furthermore, we demonstrate the edge based methods. In Figure 6.7a is Sobel edge detection applied on the CT scan. The edges are nicely connected, especially on the outside of the head. However, there are also several blobs in the region of the ventricles and the brain since in the original image these areas are also uneven, as previously noted in Section 6.1.1. Consequently, also the edges of these uneven parts are obtained by Sobel edge detection. Even though these edges are not very visible, they are irrelevant.
The idea arises to apply Sobel edge detection on Figure 6.1a. Namely, the clustered image has the same shapes, but within each segment the pixels share the same value. Applying edge detection on this image will thus not result in blobs. Indeed, Figure 6.7b is aesthetically more pleasing than Figure 6.7a. Both share the same edges, but in Figure 6.7b these edges are better visible.



(a) Sobel edge detection applied on the original CT scan.

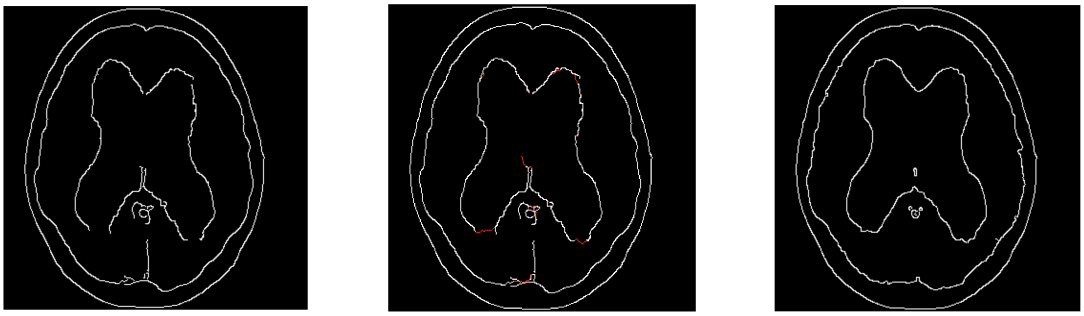


(b) Sobel edge detection applied on Figure 6.1a.

Figure 6.7: Applying Sobel edge detection in two ways.

We continue with Canny edge detection. The result of applying Canny edge detection on the CT scan is shown in Figure 6.8a. The shapes are again preserved, however the edge of the ventricles are not connected, probably because the gray values at these points are too weak. To complete these edges, the procedure of Section 3.5 is used. Figure 6.8b shows the result, with the connected edges in red. Note that the border of the ventricles is now closed and the image overall is not

much disturbed by new formed edges.
Since applying Sobel edge detection on the clustered image returned quite a nice result, we do the same with Canny edge detection. Figure 6.8c shows the outcome, which is almost identical to Figure 6.7b. The result is better than Figure 6.8a and Figure 6.8b, since the edges are connected and the less relevant edges are not shown (for instance, the vertical line at the bottom center of these two figures).



(a) Result of Canny edge detection.

(b) The edges are connected.

(c) Canny edge detection applied on Figure 6.1a.

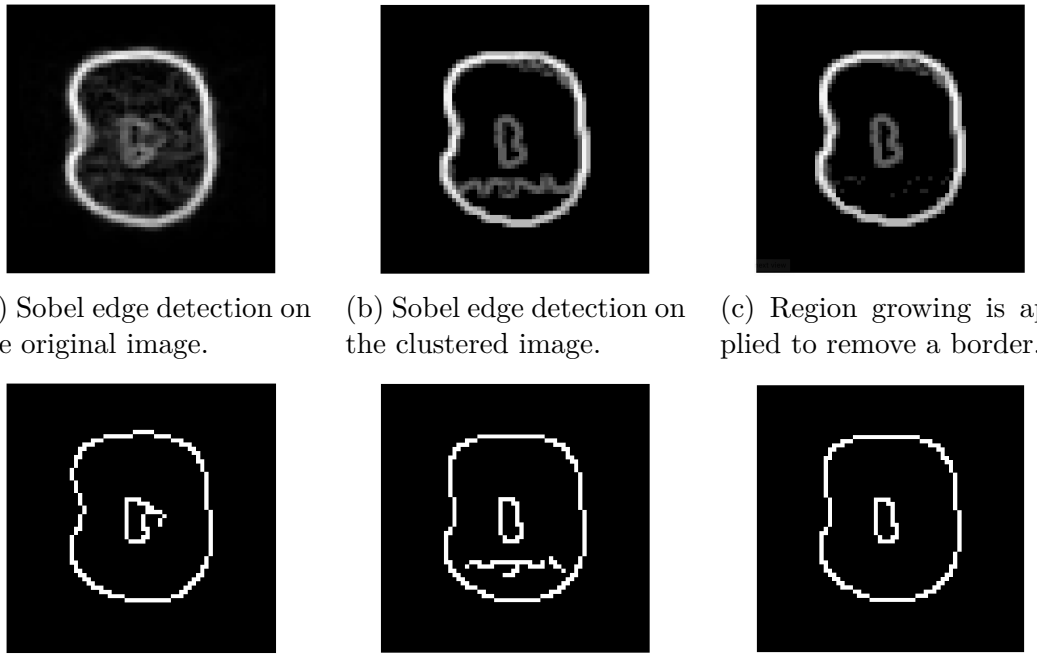
Figure 6.8: Canny edge detection is applied to the CT scan.

We may wonder if this procedure, applying edge detection on the k-means clustered image, always results in a better outcome than applying edge detection on the original image. To investigate this hypothesis we apply edge detection on the MRI scan of an apple and on the clustered image (Figure 6.3b). We choose for the image with three clusters, since Figure 6.3a shows too few features and Figure 6.3c is too frayed.

First, we apply Sobel edge detection on the two images, resulting in Figure 6.9a and Figure 6.9b. Again, we note that Figure 6.9a is messy and uneven inside the apple's border. This issue has been resolved in Figure 6.9b. However, the edge of the thick border at the bottom of the clustered image, is also obtained by Sobel edge detection. One way to get rid of this border is by applying region growing. We take a point on the edge of the thick border, for instance (20,42), and start to grow a region with a threshold of 0.1. Deleting this region from the image, results in Figure 6.9c. Note that for this procedure, the input of the user is necessary to obtain a coordinate of the border and to set a threshold.

Continuing with the Canny edge detection. In Figure 6.9d and Figure 6.9e is Canny edge detection applied on the original image and the clustered image respectively. Again we see that the edge of the border is obtained in Figure 6.9e, while this is not shown in Figure 6.9d. We can get rid of this border by region growing. Since the border consists of two separate parts, we will need one coordinate for each part, for instance (38,42) and (40,41). Deleting these two regions results in Figure 6.9f. On the other hand, the edge of the pit is more accurate when applying edge detection on the clustered image than on the original image. Last, the shape of the apple is slightly different.

We can conclude that applying edge detection on a clustered image results in sharper and more relevant edges, since edges of uneven areas are not obtained.



(a) Sobel edge detection on the original image.

(b) Sobel edge detection on the clustered image.

(c) Region growing is applied to remove a border.

(d) Canny edge detection on the original image.

(e) Canny edge detection on the clustered image.

(f) Region growing is applied to remove a border.

Figure 6.9: Result of combining edge detection with k-means clustering.

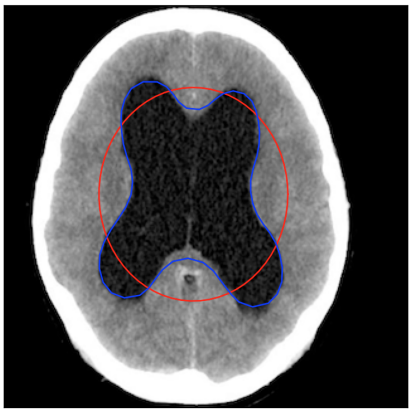
Nevertheless, darker and lighter areas of an image are extra highlighted by clustering. Edge detection also finds the edges of these areas, even though these are often not "real edges" of the original image. Such edges may be removed by region growing, but input of the user is necessary.

6.3 Active contour model

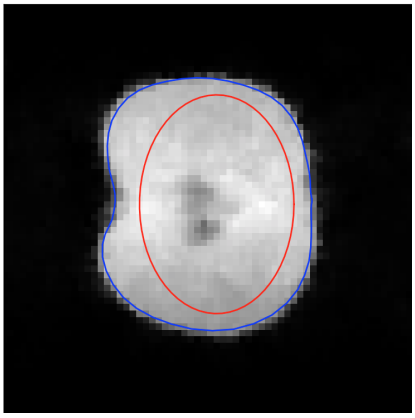
Last, we apply the active contour model on the CT scan and the apple. Figure 6.10 shows the result. The red curve is the initial snake and the blue curve is the final snake, wrapped around the contour of the ventricles and the contour of the apple.

The result of the active contour model is very nice, however it takes a lot of time to fine tune the parameters. As mention in Section 3.4, the following parameters should be fine-tuned to obtain a good result:

- the initial curve $\mathbf{s}(p)$
- α, which controls the continuity of the curve
- β, which controls the curvature of the curve
- the timestep Δt
- the amount of smoothing σ
- the number of iterations n



(a) Active contour model is applied to the CT scan.



(b) Active contour model is applied to the apple.

Figure 6.10: Results of the active contour model.

To obtain Figure 6.10a, the following values were used: $\alpha = 0.001$, $\beta = 0.001$, $\Delta t = 70$, $n = 300$ and $\sigma = 3$. The following parametric representation of the initial snake $\mathbf{s}(p) = (x(p), y(p))^T$ is used:

$$\begin{cases} x(p) = 150 + 75\cos(p) \\ y(p) = 150 + 85\sin(p) \end{cases}$$

for $p \in [0, 2\pi]$.

Similarly, to obtain Figure 6.10b, the following values were used: $\alpha = 0.01$, $\beta = 0.1$, $\Delta t = 15$, $n = 50$ and $\sigma = 3$. The following parametric representation is used:

$$\begin{cases} x(p) = 33 + 12\cos(p) \\ y(p) = 31 + 17\sin(p) \end{cases}$$

for $p \in [0, 2\pi]$.

6.3.1 Sensitivity of the parameters

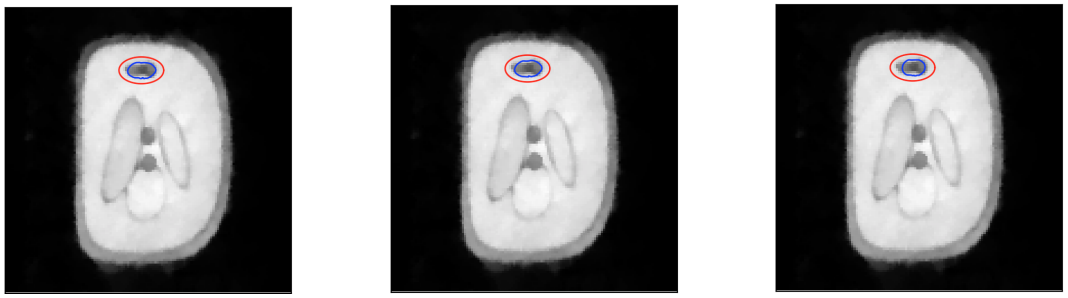
Note that the values of the parameters of the two snakes differ quite a bit. The parameters are unique for every image and have to be found by trial and error. This is a very time consuming task, especially for larger images such as the CT scan, because the method takes about one minute to return a result for this image. So when a lot of different values have to be tried in order to make the snake fit the contour, a lot of waiting time is consequential.
In this section, we show how different values of parameters lead to different results. As the CT scan is a large image and the apple has no significant contours besides its shape, we use the measured Shepp-Logan Phantom to illustrate the meaning of the variables. We aim to find the contour of the so-called eye in the upper middle of the image.

First, we set the parametric representation of the initial snake as follows:

$$\begin{cases} x(p) = 61 + 10\cos(p) \\ y(p) = 28 + 6\sin(p) \end{cases}$$

for $p \in [0, 2\pi]$.

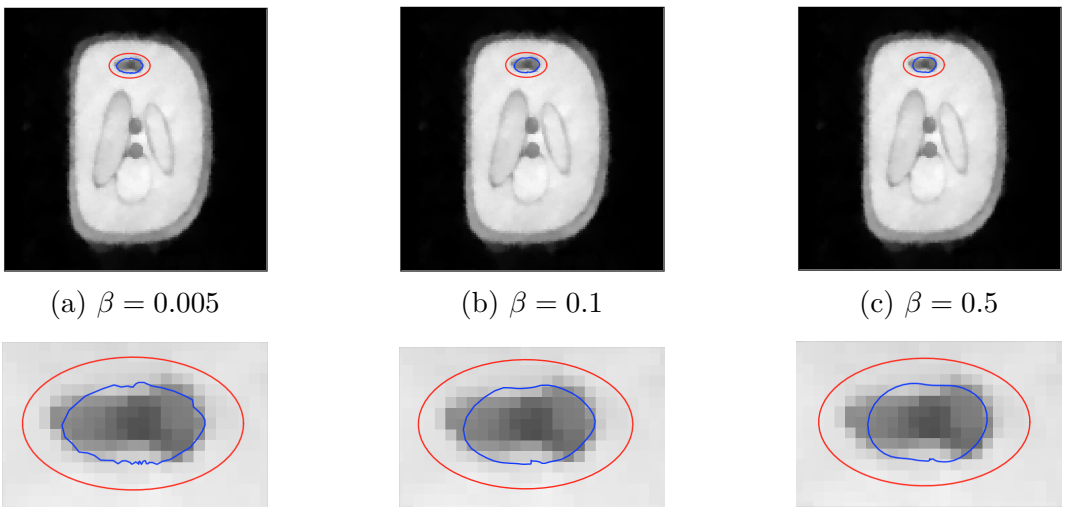
To illustrate the meaning of α, we choose respectively 0.001, 0.01 and 0.1. We set the remaining parameters as follows: $\beta = 0.1$, $\Delta t = 5$, $n = 50$ and $\sigma = 3$. Figure 6.11 shows the result. Comparing Figure 6.11a and 6.11b, we do not see remarkable differences, except that the curve is slightly pulled inwards in Figure 6.11b. This effect is emphasized in Figure 6.11c, where the curve is notably smaller. We conclude that a larger value of α shrinks the snake, which corresponds to the explanation in Section 3.4.1 where we stated that a lower α makes the snake more elastic.



(a) $\alpha = 0.001$ (b) $\alpha = 0.01$ (c) $\alpha = 0.1$

Figure 6.11: Different values of α

Next, we will show how β contributes to the shape of the snake. We set β equal to 0.005, 0.1 and 0.5 and set the remaining values as follows: $\alpha = 0.01$, $\Delta t = 5$, $n = 50$, $\sigma = 3$. Figure 6.12 shows the result. Note how in Figure 6.12d the contour is quite frayed, while for a larger β, the contour is much smoother.

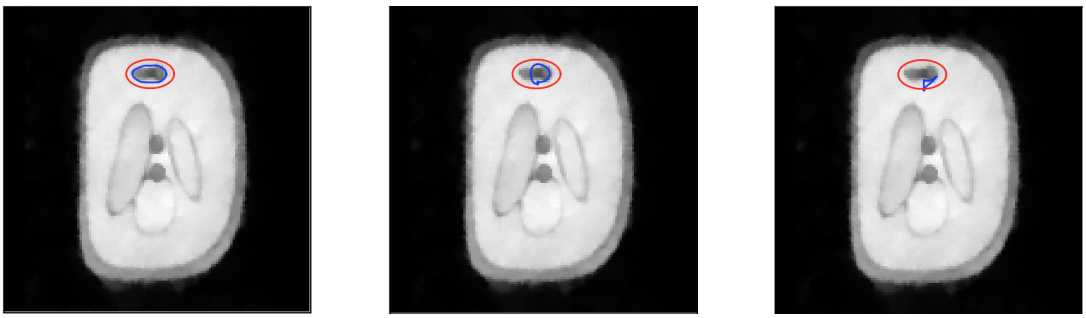


(a) $\beta = 0.005$ (b) $\beta = 0.1$ (c) $\beta = 0.5$

(d) $\beta = 0.005$ (e) $\beta = 0.1$ (f) $\beta = 0.5$

Figure 6.12: Different values of β, the images below are zoomed in on the snakes above.

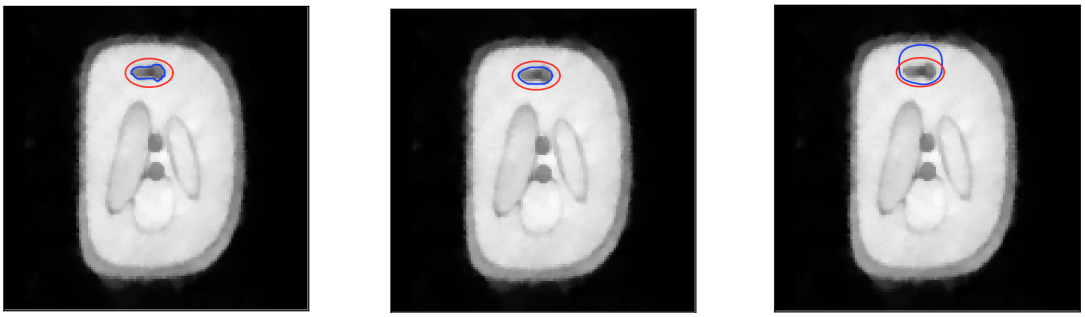
Next, consider the following three values for Δt: 1, 15 and 30. The remaining parameters are set $\alpha = 0.01$, $\beta = 0.1$, $n = 50$ and $\sigma = 3$. Note how something strange occurs for larger values of Δt, the contour is not fitting the eye at all.



(a) $\Delta t = 1$ (b) $\Delta t = 15$ (c) $\Delta t = 30$

Figure 6.13: Different values of Δt

Finally, we consider the value of σ. In Figure 6.14, the values are set to 1, 3 and 5 respectively. The other values are set as follows: $\alpha = 0.01$, $\beta = 0.1$, $\Delta t = 1$ and $n = 50$. Note how in Figure 6.14a, the contour is nicely wrapped around the eye. In contrast, for larger values of σ, the contour spreads out. This makes sense, since a higher σ means more smoothing, hence the edge is not sharp but more blurry and thicker. In Figure 6.14c, the snake is attracted to the contour of the Shepp-Logan Phantom, instead of the contour of the eye. This probably occurred since both edges are blurred in such a way, they come near each other.



(a) $\sigma = 1$ (b) $\sigma = 3$ (c) $\sigma = 5$

Figure 6.14: Different values of σ

We have seen that each parameter contributes to different outcomes, hence it is a quite time-consuming task to pick the right parameters. On the other hand, this method is developed to find one contour at a time, while by the edge based methods, one obtains all edges in an image. Using the active contour model is therefore fortunate when for instance only the shape of the ventricles have to be found. One does not have to worry about frayed, irrelevant or disconnected edges.

7 Conclusion and discussion

In this chapter, we will recapitulate the methods and give some recommendations for future work. Last, we evaluate of the running time of the methods.

7.1 Conclusion

In this research, we implemented Sobel and Canny edge detection, active contour model, k-means clustering and region growing. In addition, two extensions of these methods were proposed, namely completing edges and reassigning pixels as addition to the k-means clustering algorithm. Testing (combinations of) these methods gave us an overview of how these methods can be applied. We will review some of the results.

Sobel and Canny edge detection are both methods to find all edges in an image. The main difference is that Canny edge detection returns a binary image, containing only the relevant edges of one pixel wide. On the other hand, Sobel edge detection returns all edges, whether they are strong or not. Therefore Canny edge detection is often preferred, since the result is rather minimalistic. The extension of completing edges is a nice addition, since it often accurately completes edges. Nevertheless, it may also create non-existent or irrelevant edges. This issue can be avoided if the user selects only the edges that should be completed.
The active contour model is a way to retrieve the contour of an object in an image, without obtaining unnecessary or frayed edges. The result is often nice and accurate, however it is a time-consuming job to fine tune the many parameters. This is not done in one step, while the other edge based methods return a result with one click.

The k-means clustering algorithm is a method that is useful to separate an object from the background or to obtain different areas of an image. Some user input is necessary, to give the initial cluster centers. These centers can be found simply by inspecting the image or its histogram, so it is not a difficult or time-consuming task. However, sometimes different centers may lead to different results or longer running time. In these cases, several centers should be tried after which the user picks the center for which the outcome is preferred.
The extension where pixels are transferred between clusters based on the sum squared error works really well. Even though the changes are often subtle, this

is a way to reduce noise and connect regions that belong to the same cluster.

Combining edge based methods with the k-means clustering algorithm also returns sharp and often connected edges. This procedure is mainly useful when the clustering contains the same shapes as the original image. Otherwise, also edges of non-existing areas will be found.

Last, we implemented region growing. This method is especially useful in combination with other methods. Namely, the user should give a threshold for the homogeneity criterion, but in an image with many different intensities, the ideal threshold is sometimes hard to find. When the image consists of only a few unique values (such as after clustering or edge detection), setting a right threshold is much easier.
For instance, combining region growing with k-means clustering gives the opportunity to determine whether clusters are connected and to compute the size of regions. Region growing can also be used to remove parts of edges that are irrelevant and disconnected.

7.2 Recommendations

In this research, we considered several image segmentation methods. In addition to these methods, there exist many more techniques that may be interesting to implement and test.
First, we have another region based method next to region growing, namely region splitting and merging [8]. These are two separate techniques that are often used together. Region splitting is a way to divide an image into square regions that each meet a certain homogeneity condition. When a region does not meet the condition, it is splitted into four disjoint quadrants. Then for each of these four quadrants, the condition is again checked. This procedure is done until no more splitting is possible. Next, region merging can be applied, where two or more adjacent quadrants are merged when a condition holds for the union of these quadrants. A beginning was made with the splitting, but this method was discarded, since the choice was made to focus on the edge based methods.
A different image segmentation method that may be interesting is morphological watershedding [2]. The image is interpreted as a topographic image, where the gray-scale values represent the height. The topography is flooded with water from below, to find so-called "watershed lines", that represent edges of the image.

Secondly, as mentioned several times, the active contour model contains many different parameters that may take some time to choose. Further research should demonstrate whether there exists manners to find these parameters more easily.

Another interesting idea is to extend the methods discussed in this thesis from 2D to 3D. Since MRI scans are three dimensional, it makes sense to apply image segmentation on the 3D scan, instead of just a slice, as done with the apple. Of course, a 3D scan can be seen as a stack of 2D images. Since there are three directions to view the MRI scan (from left to right, from the bottom to the top or

from the front to the back), the direction in which the images are stacked, should be considered. We will discuss how a few methods can be adjusted for a 3D image. For the k-means clustering, no changes are necessary. For instance, the distance to the nearest cluster corresponds to calculating the differences between a pixel intensity and the cluster centers, which is not different for three dimensions. Furthermore, in the implementation the image with its pixel values is considered as a one-dimensional list instead of a 2D- or 3D-array, so the shape or size of the image does not matter.
Considering region growing, where in the 2D case, the eight neighbors of each pixels are checked for a homogeneity criteria. In the case of a 3D image, the number of neighbors increases to 26, since we should now consider a $3 \times 3 \times 3$ cube around a pixel. This will be the main adjustment when extending to 3D, since the rest of the procedure is mainly adding points to a region or not.
Finally, we can extend the Sobel edge detection. For the 2D case, two 2D masks are used to calculate the gradient in the x- and y-direction. Thus for the 3D case, we can add a third mask to calculate the gradient in the z-direction and extend each mask to 3D, transforming it into a cube.

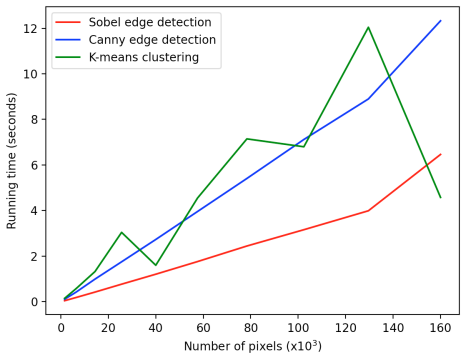
7.3 Running time

Extending the methods to three dimensions, will increase the number of pixels. Hence, it is necessary to elaborate on the running time of the methods, since more pixels take longer to process and methods that take long (say, a couple of minutes) to execute are not efficient nor realistic for future use.

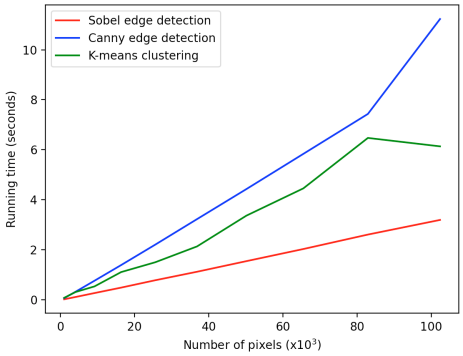
During this research it already became clear that for larger images (the simulated Shepp-Logan Phantom of size 400×400 and the CT scan of size 320×320) methods take noticeably longer than for the smaller images (the apple of size 64×64, and the measured Shepp-Logan Phantom of size 128×128). For the k-means clustering, and edge detection, it takes only a couple of seconds longer. However, region growing (over the whole image) and reassigning pixels as extension of the k-means clustering algorithm, take at least 45 minutes to one hour, which is an unusually long time. Last, the active contour model takes about one second for the apple, but over a minute for the CT scan.
The 2D MRI scans are usually smaller in size, but extending to 3D increases the number of pixels rapidly. For comparison, a 3D MRI scan of size $64 \times 64 \times 64$ contains 262.144 pixels and the simulated Shepp-Logan Phantom of size 400×400 contains 160.000. So considering a 3D scan may even take longer than a large 2D image. Therefore, the more complicated methods should be adjusted to make the running time feasible for 3D scans.

Figure 7.1 shows the running time for the Sobel edge detection, Canny edge detection and k-means clustering as function of the number of pixels. For Figure 7.1a, the simulated Shepp-Logan Phantom was resized to obtain different sizes of the same image (10%, 20% to 100% of the original size). The same is done for the CT scan to obtain Figure 7.1b. Note that overall, the running time increases proportionally. The running time of the k-means clustering algorithm applied to

the Shepp-Logan Phantom seems quite off. The reason is perhaps that initially the image consists of only six unique gray values, which makes the k-means clustering not too difficult. However, during resizing, the gray values change slightly resulting in more than six unique values.
For the simulated Shepp-Logan Phantom, the six unique values of this image were used. For the CT scan, the k-means clustering was done each time with the three values 0, 0.5 and 1.



(a) The simulated Shepp-Logan Phantom is resized.



(b) The CT scan is resized.

Figure 7.1: The running time as function of the number of pixels for several methods.

A Manual for the Python code

In this appendix, we explain how to use the functions implemented in Python in order to create the images. In all cases, the in- and output images are two-dimensional NumPy arrays.

A.1 K-means clustering

The implementation of the k-means clustering algorithm consists of three functions: `kmeans_self`, `smallestdistance` and `kmeans_image`, where the latter two are used in `kmeans_self`.
The function `kmeans_self(img,centers)` is the function that executes the algorithm itself. It takes two objects, first we have the image `img` that should be clustered. Secondly, we have a list (not a NumPy array) containing initial cluster centers of length k, named `centers`. The three-piece output consists of `centers`, `clusterlabels` and `segm`. These refer to the list of final cluster centers, a NumPy array containing labels and finally the clustered image.
The function `smallestdistance(pixel,lst_centers)` is used to compute the smallest distance for a given pixel `pixel` and the cluster centers `lst_centers`. It returns a value between 0 and k, which is in fact the label referring to the cluster the pixel is assigned to.
Finally, the function `kmeans_image(centers,clusterlabels,shp)` is used to create the image. The three input objects are respectively the final cluster centers, the pixel labels and the shape of the original image noted as `(N,M)`.
In the function `kmeans_self` one can see the print statement `"empty points, choose new centers"` in the for-loop that is used to compute new cluster centers. This is done by taking the mean of the current centers. However, it may occur that one of these centers does not contain any pixels. In that case, the functions terminates and one should choose new cluster centers.

In addition to these functions, we have a separate function to calculate the sum squared error of the clustered image, called `sse(img,centers,labels)`.

A.2 Reassigning pixels

We extended the k-means clustering algorithm with a method that aims to reassign pixels to different clusters. To achieve this, we introduce the function `new_labels(img,val,labs,to_change)`. The object `img` is the original image, the objects `val` and `labs` are the final cluster centers and labels obtained by the function `kmeans_self`. Finally, `to_change` is a list of coordinates of all the pixels that should be relabelled. This can either be one coordinate (`[y,x]`), a list of more coordinates (`[[`y_1`,`x_1`],[`y_2`,`x_2`],...]`) or a list of lists of coordinates (`[[[`y_1`,`x_1`],[`y_2`,`x_2`]],[[`y_3`,`x_3`],[`y_4`,`x_4`]],...]`).
This function uses the function `sse` to calculate the sum squared errors. In addition, it uses the function `kmeans_image` to make a new clustered image `seg2`. This, in combination with a new list of labels `l2`, is the output of the function `new_labels`.

A.3 Region growing

The region growing implementation consists of two functions: `regiongrowing` and `neighbors`.
The function `regiongrowing(seeds,img,T)` takes three arguments. First, `seeds` contains one or more starting seeds, which are coordinates of the image. These can be a tuple or a list, and should be given as `(y,x)` or `[y,x]` for one seed and for instance (`((`y_1`,`x_1`),(`y_2`,`x_2`),...)` or `[(`y_1`,`x_1`),(`y_2`,`x_2`),...]` for more seeds. Next, the argument `img` is the original image and `T` is the threshold that is used for the homogeneity criterion. This can be a float or integer.
The function `neighbors(img,y,x,T)` is used to obtain the pixels that are connected to the pixel at coordinate `(y,x)` and meet the homogeneity criterion using a threshold `T`. The default value of `T` is 1000, which means that when no threshold is set by the user, all the neighbors in the 8-connected neighborhood are returned, regardless of their values. The function `neighbors` returns a list of the coordinates of the neighbors and a list of the pixel values of the neighbors. For the function `regiongrowing`, only the first list is used.
The function `regiongrowing` returns two objects. First, we have the segmented image `segm`, showing the region(s). The gray values of the pixels in the region(s) are the same as in the original image. The rest of the image is black. The second object is `regions`, which is a list that contains all the coordinates of each region. For instance, when two seeds are given that result in two separate regions, the length of `regions` is also two. When two seeds are given that make up one region, the length of `regions` is just one.
Note that when a black region is grown, the output is just a black image, since also the background is black. Therefore, the list `regions` can be used to make an image of the region with a white background for instance.

Additionally, the function `regiongrowing_everywhere(img,T)` can be used to apply region growing over the whole image, so no initial seeds have to be given.

A.4 Sobel edge detection

The function that is used to execute Sobel edge detection is called `edges(img,filterx,filtery)`. This function takes three arguments: the image `img` and the two filters `filterx` and `filtery` that are used to compute the gradient in the x- and y-direction. The default is set to the Sobel filters shown in Figure 3.3. The output consists of five NumPy arrays: the gradient in the x- and y-direction (called `X` and `Y`), the magnitude of the gradient `magn`, the direction of the edge `edgedir` and the direction of the gradient `graddir`. The latter two are given in radians.

A.5 Canny edge detection

The Canny edge detection consists of multiple steps. In the implementation, each step, except the blurring, has its own function. The main function is `canny(img,blur,sigma,low,high)`. It takes the original image `img`, the amount of blurring `sigma` and the low and high threshold `low`, `high`. The default values of `sigma` is 1 and the default values of `low` and `high` are 0.1 and 0.2. The variable `blur` is a boolean value, which indicates whether Gaussian blurring will be applied (set the value to True) or not (set the value to False).
In the first step is the function `edges`, mentioned in the previous section, used to calculate the magnitude of the gradient and the direction of the edge.
Next, the non-maximum suppression is executed in the function `nonmaxsup(magn,direct)`. The output of this function is an image containing edges of one pixel wide, where the pixel values equal the values in the magnitude image.
Furthermore, the function `double_thresh(img,lower,upper)` is used to apply double thresholding, returning an image with only medium or strong edges. The values `lower`, `upper` refer to the two thresholds. The argument `img` is the image after non-maximum suppression.
The final step of Canny edge detection is hysteresis, which is executed by the function `hysteresis(img,lower)`. The object `img` is the output of `double_thr` and the value of `lower` is the same as used previously. The output of this function is the final result of the Canny edge detection.
The function `canny` returns the result after each step, excluding the blurring. Therefore, the output consists of four images: the magnitude of the gradient and the result after non-maximum suppression, double thresholding and hysteresis.

A.6 Completing edges

We proposed an extension that aims to detect and connect incomplete edges. We will explain the implementation of finding loose ends and connecting edges separately.

A.6.1 Finding loose ends

The implementation uses the function called `loose_ends(img,direct)` to detect the endpoints of edges. The argument `img` is a binary image of which the edges

should be connected. The second argument `direct` contains the direction of the edges, obtained by Sobel edge detection. It returns a list of the coordinates of all the loose ends `l` and an image where the loose ends are indicated with gray, named `out`.
The function `loose_ends` uses three functions, of which one is the `neighbors` function mentioned in Section A.3. This is used to get the amount of neighbors. Next, the function `before_after` is used to obtain the values and coordinates of the pixels before and after a given pixel. Finally, the function `condition` is used to obtain the gray values of the pixels that are connected to the suppressed before or after pixel. The latter two functions are used to check condition 2b, mentioned in Section 3.5.2.

A.6.2 Connecting edges

The function `multiple_edges(img,direc,loose_ends)` is used to connect the loose ends. It takes the original image `img`, the direction of the edges `direc` and a list of the coordinates of the loose ends, called `loose_ends`. The first two arguments are the same as used in the function `loose_ends` and the latter argument is the output of `loose_ends`. It returns a list of the coordinates of each edge `edges`, so it is a list of lists. Furthermore, it returns an image `out` with the new connections in gray.

Two functions are used within `multiple_edges`. First, we have the function `facetoface` that checks for all loose ends whether two of them lie within a distance $d = 2$ from each other (in other words: on the same horizontal or vertical line with one pixel in between). It returns two lists, the first one contains the pixels that lie in between two loose ends and hence make a connection. Secondly, it returns a list containing sets of coordinates of the pixels that are connected. This last list is needed since a record is kept of points that are already connected.
The second function, named `connect_edges`, is the function in which the connections are actually made. It returns an image with the connection and a list of coordinates that make up the connection. Thus, this function is used to make one single connection given a starting point, where the function `multiple_edges` is used to connect all loose ends.

A.7 Active contour model

The implementation of the active contour model consists of three functions. The main function is `iterate_snake`. This function takes the original image `img`, the initial curve, given by `x` and `y` and all the parameters α, β, Δt, n, σ and h, where the latter is set to 1 by default. The output of `iterate_snake` is the final snake, consisting of two arrays with the x- and y-coordinates separately.
It uses the two functions `create_A` and `external_forces`. The first function is used to create the matrix A, which is called once at the beginning. The second function is used to calculate the external force and it returns the values of the external force at the x- and y-coordinates of the curve.

Bibliography

[1] Adams, R., & Bischof, L. (1994). Seeded Region Growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16*(6), 641–647. URL: https://pdfs.semanticscholar.org/db44/31b2a552d0f3d250df38b2c60959f404536f.pdf

[2] Beucher, S., & Meyer, F. (1993). The Morphological Approach to Segmentation: The Watershed Transformation. *Mathematical Morphology in Image Processing, 43*, 433–481. DOI: https://doi.org/10.1201/9781482277234-12

[3] Braun, B. (2017, February 16). *Do It Yourself MRI*. Consulted on May 27 2020, via http://archief.mareonline.nl/archive/2017/02/16/do-it-yourself-mri

[4] Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8*(6), 679–698. DOI: https://doi.org/10.1109/tpami.1986.4767851

[5] Coste, A. (2012). *Image Processing Final Project Active Contours Models*. URL: http://www.sci.utah.edu/~acoste/uou/Image/final_project/ArthurCOSTE_final_project.pdf

[6] Deriglazov, A. (2016). *Classical Mechanics* (2nd edition). DOI: https://doi.org/10.1007/978-3-319-44147-4

[7] Eggleston, P. (1998, December 1). *Understanding oversegmentation and region merging*. Consulted on June 9 2020, via https://www.vision-systems.com/non-factory/security-surveillance-transportation/article/16739494/understanding-oversegmentation-and-region-merging

[8] Gonzalez, R. C., & Woods, R. E. (2002). *Digital Image Processing* (2nd edition). URL: http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/Digital_Image_Processing_2ndEd.pdf

[9] Hancock, M. (2015, May 30). *Notmatthancock/snakes*. Consulted on June 26 2020, via https://github.com/notmatthancock/snakes

[10] Ivins, J., & Porrill, J. (1993). *Everything You Always Wanted to Know About Snakes (But Were Afraid To Ask)*. URL: https://web.mat.upc.edu/toni.susin/files/SnakesAivru86c.pdf

[11] Jana, A. (2019, May 20). *Implement Canny edge detector using Python from scratch*. Consulted on May 1 2020, via http:

//www.adeveloperdiary.com/data-science/computer-vision/implement-canny-edge-detector-using-python-from-scratch/

[12] Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active Contour Models. *International Journal of Computer Vision, 1*(4), 321–331. URL: http://www.cs.ait.ac.th/~mdailey/cvreadings/Kass-Snakes.pdf

[13] Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory, 28*(2), 129–137. URL: https://cs.nyu.edu/~roweis/csc2515-2006/readings/lloyd57.pdf

[14] Medtronic. (n.d.). *Wat is een waterhoofd (Hydrocephalus)?* Consulted on May 27 2020, via https://www.medtronic.com/nl-nl/patienten/aandoeningen/waterhoofd-hydrocephalus.html

[15] Schaefer, A. (2020). *The Watermelon Diet: Fact or Fiction?* [Photo]. URL: https://www.healthline.com/health/diet-weight-loss/watermelon-diet-fact-or-fiction

[16] Shepp, L. A., & Logan, B. F. (1974). The Fourier reconstruction of a head section. *IEEE Transactions on Nuclear Science, 21*(3), 21–43. DOI: https://doi.org/10.1109/tns.1974.6499235

[17] Simkin, P. (n.d.). *Obstructive hydrocephalus* [Photo]. URL: https://radiopaedia.org/cases/obstructive-hydrocephalus

[18] Sobel, I., & Feldman, G. (1973). A 3x3 Isotropic Gradient Operator for Image Processing. In *Pattern Classification and Scene Analysis* (pp. 271–272). Hoboken, NJ, Verenigde Staten: Wiley.

[19] Tiilikainen, N. P. (2007). *A Comparative Study of Active Contour Snakes*. URL: http://home.iitj.ac.in/~manpreet.bedi/btp/rmaterial/Snakes.pdf

[20] O'Reilly, T., Teeuwisse, W. M., & Webb, A. G. (2019). Three-dimensional MRI in a homogenous 27 cm diameter bore Halbach array magnet. *Journal of Magnetic Resonance, 307*, 106578. DOI: https://doi.org/10.1016/j.jmr.2019.106578

[21] Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics, 9*(1), 62–66. DOI: https://doi.org/10.1109/tsmc.1979.4310076

[22] Yao, H., Duan, Q., Li, D., & Wang, J. (2012). An improved K-means clustering algorithm for fish image segmentation. *Mathematical and Computer Modelling, 58*(3–4), 790–798. DOI: https://doi.org/10.1016/j.mcm.2012.12.025