CHASM

Computer-Human Assisted Segmentation of Medical Structures

Manuel Cajada Coelho Lopes

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisor:	Prof. Joaquim Armando Pires Jorge
Supervisor:	Dr. Maurício Lança Tavares de Sousa

Examination Committee:

Chairperson:	Prof. João Miguel Raposo Sanches
Supervisor:	Prof. Joaquim Armando Pires Jorge
Members of the Committee:	Prof. Abel João Padrão Gomes

October 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Preface

The work presented in this dissertation was performed at the Institute for Systems Engineering and Computers of Instituto Superior Técnico (Lisbon, Portugal), during the period March-October 2021, under the supervision of Prof. Joaquim Jorge.

Abstract

This dissertation aims to implement a semi-automatic edge contouring method capable of segmenting high resolution cryosections of a cadaver available at the Visible Human Project, with minimal necessary user input. Initially, different image segmentation algorithms were explored, however, the use of graph cut based algorithms was the preferred method. An algorithm was then developed on top of OpenCV's Grabcut, which receives a manually drawn contour, corrects the segmentation and propagates it across an interval of slices. Pre and post processing methods were developed in order to improve computation time and segmentation performance. The contours were then saved in order to perform a reconstruction of the anatomical structure.

The results obtained were promising, total reconstruction of the femur was possible, along with other structures, such as the aorta and the liver. The time it took to achieve a full reconstruction depends on the size of the structure, computing around four slices per second for the aorta and one slice per two seconds for the liver. The resulting contours could be improved, as the color differentiation between some tissues is minimal. Some solutions were suggested that could enhance the contrast such as the using the additional radiological images available in the data set.

Keywords: Visible Human Project, OpenCv GrabCut, Python, Edge Contouring Algorithm, Computer Vision, Medical Imaging

Resumo

Esta tese propõe e implementa um algoritmo semi-automático de delineação de contornos, capaz de segmentar criosecções de um cadáver disponível através do projecto *Visible Human Project*, minimizando o *input* necessário. Inicialmente, diferentes métodos de segmentação de imagem foram explorados e avaliados, no entanto, o uso de *graph cuts* foi o método escolhido. Um algorítmo foi desenvolvido sobre o *GrabCut* da biblioteca OpenCV, que recebe um contorno desenhado manualmente pelo utilizador, corrige-o, e propaga-o ao longo de um conjunto de fatias. De maneira a melhorar a segmentação e o tempo de computação, foram desenvolvidos métodos de pré e pós processamento de imagem. Os contornos foram gravados de maneira a ser possível uma reconstrução da estrutura anatómica.

Os resultados obtidos foram adequados, foi possível uma reconstrução total do fémur, e de outras estruturas, como o fígado e a aorta. A duração de uma reconstrução total varia com o tamanho da estrutura, sendo que foram processadas quatro fatias por segundo para a aorta e uma fatia a cada dois segundos para o fígado. Os contornos obtidos podem ser melhorados, visto que a diferença na coloração entre alguns tecidos é mínima. Foram sugeridas algumas soluções que visam melhorar o contraste entre tecidos usando as imagens radiológicas disponíveis no *Visible Human Project*.

Palavras-Chave: Visible Human Project, OpenCV GrabCut, Python, Algorítmo de Delineação de Contornos, Visão Computacional, Imagiologia Médica

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

3DRAS	3D Reconstructions from Anatomical Slices. 1
ADC	Apparent Diffusion Coefficient. 4
BITS	Base Information Transfer System. 11
BRATS	Brain Tumor Image Segmentation Benchmark. 2
CNN	Convolutional Neural Network. 60
CPU	Central Processing Unit. 50
CT	Computed Tomography. 11
FLAIR	Fluid-attenuated Inversion Recovery. 4
GAC	Positron Emission Tomography. 27
GE	General Electric. 11
GMM	Gaussian Mixture Model. 33
GUI	Graphical User Interface. 50
GVF	Gradient Vector Flow. 27
I/O	Input/Output. 50
ITK	Insight Toolkit. v, 2

MIVAP	Medical Image Processing Analysis and Visualization. 5
MRI	Magnetic Resonance Imaging. 7, 8, 11
PD	Proton Density. 11
PET	Positron Emission Tomography. 3
PNG	Portable Network Graphics. 12
RGB	Red Green Blue. 12
TIFF	Tag Image File Format. 12
UCHSC	University of Colorado Health Sciences Center. 8
UK	United Kingdom. 32
USA	United States of America. 8
VOI	Volume Of Interest. 5

CHAPTER

1 INTRODUCTION

The use of human bodies for anatomical classes has been present in Europe since Late Middle Ages, and relied on jails and poorhouses as a source for bodies. Only later, in the 1960s, did body donation with informed consent became an alternative and has since been the primary method used for teaching gross anatomy to medical students[1]. It is still considered one of the most important methods of delivering fundamental anatomical knowledge to students in order to ensure safe clinical practice and also to cultivate humanistic values among the students. Nearly all clinicians still remember their first dissection of a human cadaver [2].

However, the whole process can be quite expensive, cadavers require a large laboratory and usually have short shelf lives if not prepared correctly. Body parts are hard to find and are often destroyed in the process. Not only that, but there has also been a shortage in cadavers for medical study due to higher demand. The number of medical programs is growing, organs are being used for transplants, and some bodies may end up in pharmaceutical companies or exhibits. Not only that, but also with the evolution of communication there are fewer unclaimed bodies for use [3][4].

As such, there has been a shift toward other teaching methods recently, such as the use of Virtual Reality. New tools are being developed that allow students to learn anatomy in new innovative ways, with 3D Reconstructions from anatomical slices (3DRAS). This has been made possible due to the large data sets being available, which allow for a total reconstruction of the body[5]. This brings many advantages, such as multiple viewpoints of anatomical structures and re-usability of the model, which were limitations from using real cadavers[6].

1.1 Current Tools

There are already a wide variety of anatomy dissection platforms available and others under development.

1.1.1 Insight Toolkit (ITK)

The ITK is an open software toolkit implemented in C++ for registration and segmentation of images, which is mainly used for medical images. This segmentation can be done in two or more dimensions and consists of identifying and classifying data found. It then establishes correspondence between images, and combines their information. This is usually done with MRI and CT scans[7]. Initially, this was done either manually or fully automated, but recent researches have invested in semi-automatic segmentation and created the ITK-Snap, an application used to segment 3D images that uses active contours. The software assigns an integer value to each voxel in the image in order to create different labels for the objects being segmented. These labels are seen as transparent layers in different colors. The user input are polygons or a mask created with the paintbrush tool, which has an adaptive setting that includes neighbor voxels with similar intensity. There are also tools for 3D input.

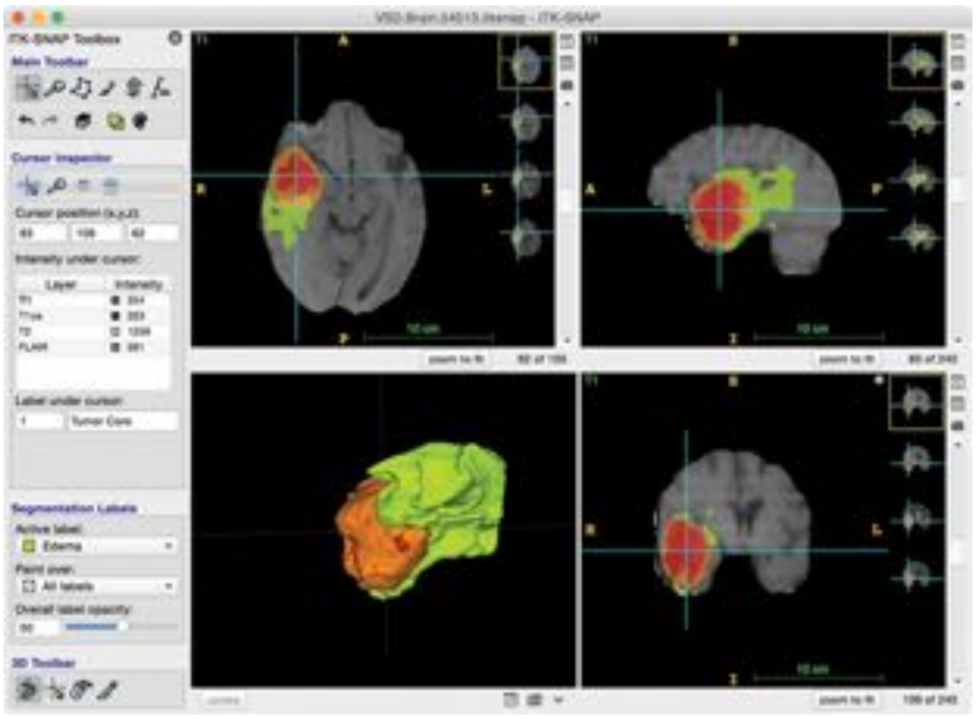


Figure 1.1: Snapshot of the software in action during segmentation of a brain tumor from a multi modality MRI data set. The segmentation illustrated was done using the semi-automatic algorithm in about 15 minutes. The tumor image data is from the Medical Image Computing and Computer Assisted Intervention 2013 Multimodal Brain Tumor Segmentation (BRATS), available in the software's website (http://www.itksnap.org).

The segmentation algorithm consists of a two stage pipeline. The first stage is to compute the probabilities of the object being a structure of interest (foreground) or background. The second stage involves using active contour segmentation. These two stages are then repeated for different structures.

This software is effective, and it is not tissue specific. It manages to segment any type of tissue in the images provided. However it takes some time to compute, with the average segmentation for a brain tumor being around 10-15 minutes. Not only that, but it also focuses on radiological images. [8].

1.1.2 Interactive Brain Atlas

A prototype of a brain atlas was developed using the Visual Human Project data set[9] to demonstrate the techniques and processes that will be later used to develop a complete atlas. The user interface allows for multiple image display, 2D and 3D representations of the brain and volume visualization tools[10].

In this project, only the green channel from anatomical cryosections was used for segmentation, since it provided high contrast between the brain and the skull. The process of segmentation of the brain consisted of manual tracing and semi-automated morphological tools (dilation and erosion). Initially the data went by a threshold in order to have the best representation of the brain with binary data. Then, the image was eroded and dilated until the brain was separated from the background. Manual tracing was used to create the intracranial cavity of the skull. The 3D render was made with an internally-developed tool (Figure 1.2) [10].

Even though the results obtained were accurate, this tool relied heavily on user input.

1.1.3 3D Slicer

3D Slicer is a free segmentation tool available for almost all operating systems. It supports various image modalities, such as CT, PET and MRI scans and allows for visualization in up to four dimensions. The software has modules designed to serve a specific task, with segmentation being one of them. The available module supports automatic and interactive segmentation and uses a wide range of contouring methods (region-based statistical models, level sets, active learning and Expectation-Maximization multi-channel segmentation). The algorithms available can be selected depending the tissue and anatomical site, level of interaction, speed/accuracy trade-off and purpose[11].

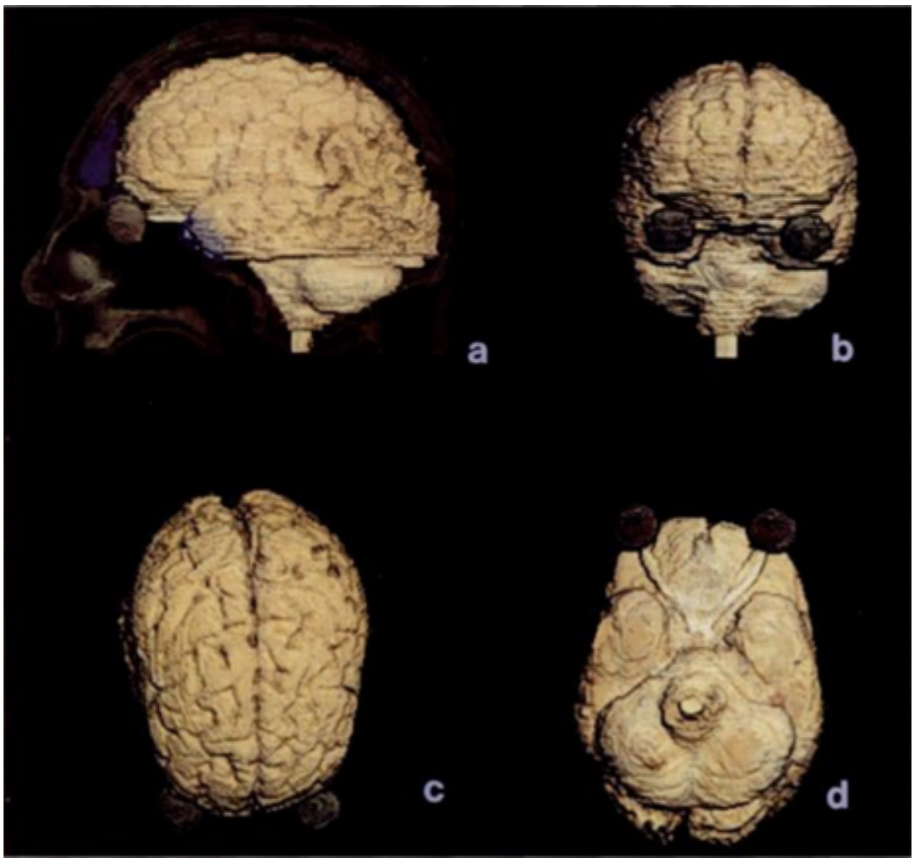


Figure 1.2: 3-D images from the atlas prototype created from the Visual Human Project data set. Four views are shown: side (a), front (b), top (c) and bottom (d)[10].

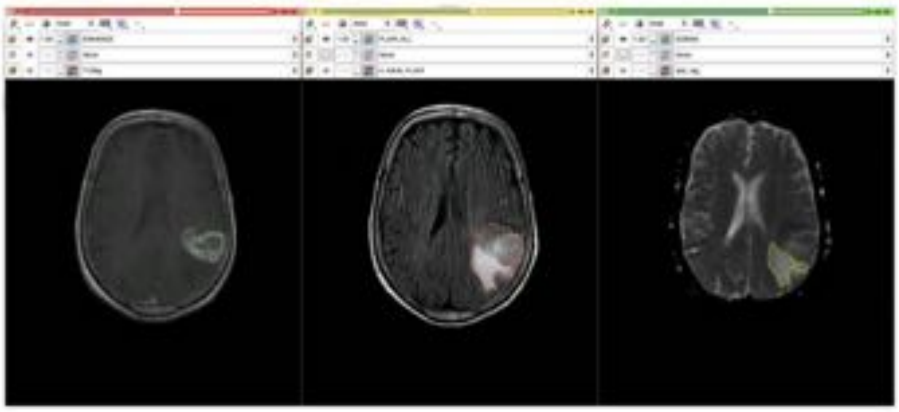


Figure 1.3: Segmentation of T1, FLAIR and ADC maps using the grow-cut segmentation module. On the left, the enhancing region is delineated on a post-contrast T1 image, while in the center and abnormality in the FLAIR is identified. On the right, a vasogenic edema is segmented[11].

1.1.4 Medical Image Processing Analysis and Visualization

The Medical Image Processing Analysis and Visualization, also known as MIPAV, makes use of Java's object oriented features to process and visualize images. It also works in modules, with the algorithms module having over 20 image processing and analysis tools. It supports different data sets, such as microscopy, MRI, CT scans and PET. The image segmentation process used is quite different, connected regions are identified and delineated by Volume of Interest. This process can be automatic, semi-automatic or fully manual. However, automatic segmentations often require correction or the aid of other edge detection algorithms. MIVAP has had a primary focus on radiological images, but is quickly evolving and getting more and more features.

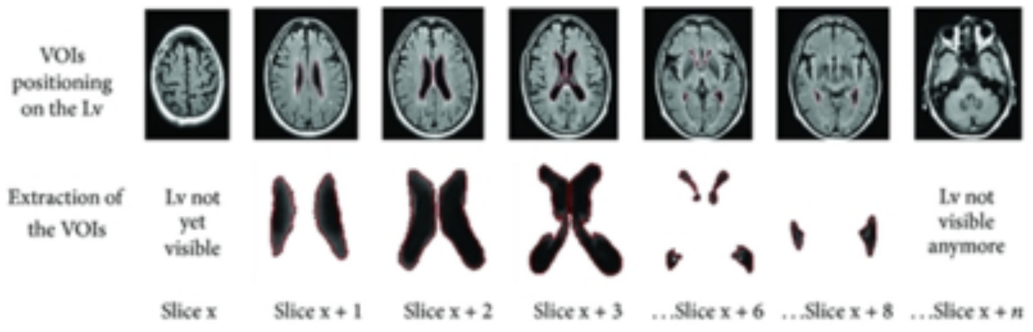


Figure 1.4: Semiautomatic segmentation with MIPAV applied for volume calculation of the left ventricle: In this example, VOIs were defined with a level set contour tool on the left ventricle of some slices of the 2D FLAIR sequence, then extracted in order to calculate the total volume[12].

1.2 Motivation

As stated above, several tools are available that allow the user to manually segment data sets and provide the reconstructed object in 3D. Some use automatic/semi-automatic segmentation with edge contouring algorithms or artificial intelligence, by building and training machines that can perform the segmentation, while others rely heavily on manual delineation. The rising availability of cadaver data sets, such as the Visible Human Project, has also provided important resources for research on image segmentation.

Most user-oriented methods require heavy user interaction. However, pure manual contouring of the structures in a series of images is very time consuming. Moreover, user interfaces to interact with these models are still precarious and often in 2D. Lastly, the results obtained so far can have segmentation inaccuracies and are still not considered good enough to be used as a reliable tool in medical environments. Therefore, there is still a gap in the market for a reliable contouring algorithm that can be used with ease.

In this dissertation, existing segmentation methods are explored and an edge contouring algorithm is implemented in order to facilitate this interaction between the user and the model, removing the need to manually contour every slice of a data set. The algorithm takes a contour and propagates it across cryosections of the Visual Human Project, rebuilding the structure with little input from the user.

1.3 Objectives

This dissertation aims to evaluate existing methods of edge contouring and produce an algorithm capable of reducing the amount of user input needed for a total reconstruction. The algorithm will then be used in Anatomy Studio, a project being developed by INESC-ID in partnership with the Unesco Chair of Digital Anatomy. Anatomy Studio consists of a collaborative Mixed Reality tool for virtual dissection that combines tablets with styli and see-through head-mounted displays to assist anatomists by easing manual tracing and exploring cryosection images.

The goal was to develop an algorithm that was able to take a slice from the Visible Human Project data set, along with a rough contour drawn on the platform and produce an adjusted segmentation of the image. The algorithm should then be able to take this contour and propagate it across a set of slices in order to gather information for a full reconstruction of the anatomical structure.

The developed algorithm should maximize the output with minimal input in order to reduce the amount of interactions required from the user. Several pre processing and post processing techniques were evaluated and applied to the algorithm to achieve better performance and efficiency. The user should be able to have a fully reconstructed object within minutes.

Lastly, since the algorithm will be incorporated in a platform that is under development, several adjustments should be made to accommodate the implementation in the platform.

1.4 Achievements

In conclusion, several different approaches to edge contouring were explored, with some of them not being able to solve the problem at hand.

An edge contouring and propagation algorithm was implemented in Python. This was achieved with the help of OpenCv, a Python library with a wide variety of tools for image manipulation and computer vision.

The adjusting of the contours was done with GrabCut, an algorithm that extracts the foreground of an image using a bounding box or a mask. Several tweaks had to be made in order to transform the inputs and outputs for the project.

First of all, a small algorithm was developed to transform user input into a mask that could be read by GrabCut. Secondly, this contour was propagated across the slices using dilation and erosion tools, along with GrabCut. Lastly, the contour was extracted from the output and saved in a pointcloud for visualization.

The final algorithm was enhanced in order to reduce the computation time and to smooth the final contours, with pre and post processing methods. Finally, parameterization levels were created in order to reduce the amount of input needed from the user.

1.5 Document Organization

Chapter 2 describes the content available in the Visible Human Project, which consist of cryosections and CT and MRI scans of a cadaver. It goes over the process taken to obtain these images and how they were prepared for segmentation with the developed algorithm.

Chapter 3 provides an overview some of the existing segmentation algorithms available, with a focus on techniques used during the execution of this dissertation.

Chapter 4 is divided into two main parts. The first part goes over the OpenCV GrabCut and how it operates. It describes its main functionalities and features. The second part explains the implementation and methods used to build the segmentation algorithm.

Chapter 5 divulges the results obtained from the algorithm described in Chapter 4, when segmenting images from the data set. It presents a comparison of different methods and how the algorithm performed on different tissues.

Chapter 6 concludes with an overview of the dissertation and limitations of the algorithm. It provides a description of what is missing in terms of performance, and what can be done to further improve segmentation.

CHAPTER

2

THE VISIBLE HUMAN PROJECT

The data set used in this dissertation was gathered from The Visible Human Project, available at the National Library of Medicine of the USA. The data set had its origin in 1986 as the need for electronically represented medical images in research grew. It took approximately two and a half years to find suitable cadavers. Currently, two cadavers are available (Figure 2.1), the male cadaver is from 39-year-old Joseph Paul Jernigan who was executed in 1993 by lethal injection and had agreed to donate his body for medical use, whereas the female cadaver is from an unknown donor, who was obese, lived in Maryland, and died of heart disease at the age of 59. A third female cadaver has been donated but has not yet been made available. For consistency purposes only the female body was used, which is also more complete than the male counterpart[13].

The processing of the corpses was done at the University of Colorado Health Sciences Center (UCHSC), in order to obtain the MRI and CT scans before the corpse was frozen. The body was placed in a containment box filled with rapidly hardening foam for stability (Figure 2.2). After the CT images were captured, the entire box was frozen and stored. The cutting process was done with a special cryomacrotome with a large range of motion to be able to cut the whole block. Initially, the cadaver was cut into 4 blocks. These blocks were then isolated, the foam was partially removed and they were set in a 3% gelatin solution. The blocks were then frozen again at -70ºC[14].

When cutting, the block was placed in a larger chamber with blocks of dry ice to prevent defrosting during the process (Figure 2.3). The surface of the block went into a photography chamber, the image was captured, and the block was moved back and raised by the desired interval in the z axis and cut again.

This process was then repeated for the four blocks. During the photographic process, the block was flushed with compressed air and examined for defects. Cavities were filled with foam and uncut fragments, such as tendons and fascia, were trimmed. Lastly, the surface was sprayed with absolute alcohol, surrounded with a black mask, slice number and grey scale.

Digital and photographic images were taken with a Leaf camera and examined to assure the quality necessary for film acquisition. The images were then compressed and aligned. The cameras used for the film images were 35mm and 70mm Rollieflex cameras, while the surface of the block was illuminated with Polarized Strobe lights. An aluminium tray with dry ice was then placed over the block to keep the surface frozen between cycles. This process was very long, taking almost a year to complete[14].

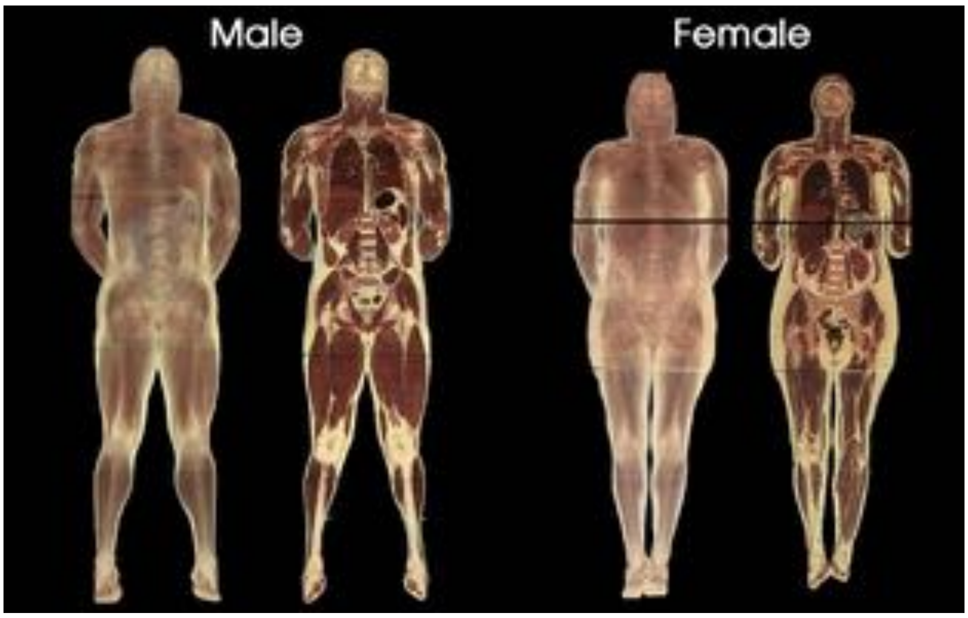


Figure 2.1: Coronal reconstructions of the blocks from the transverse slices (1,878 male and5,189 female) of the Visible Human Data set. The discontinuities indicate block interfaces[14].

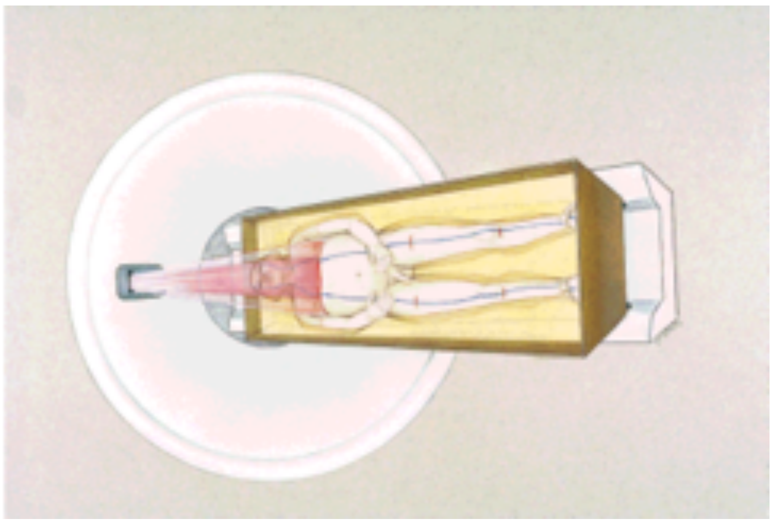


Figure 2.2: The cadaver cocooned in a containment box filled with foam. The laser is indicating the optimal block interface location determined from the CT survey images[14].

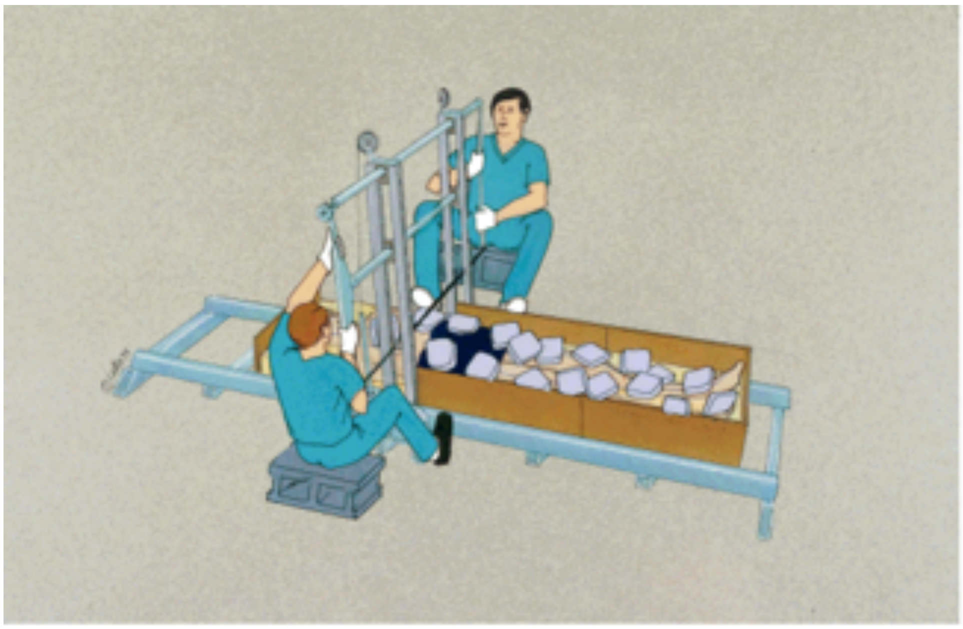
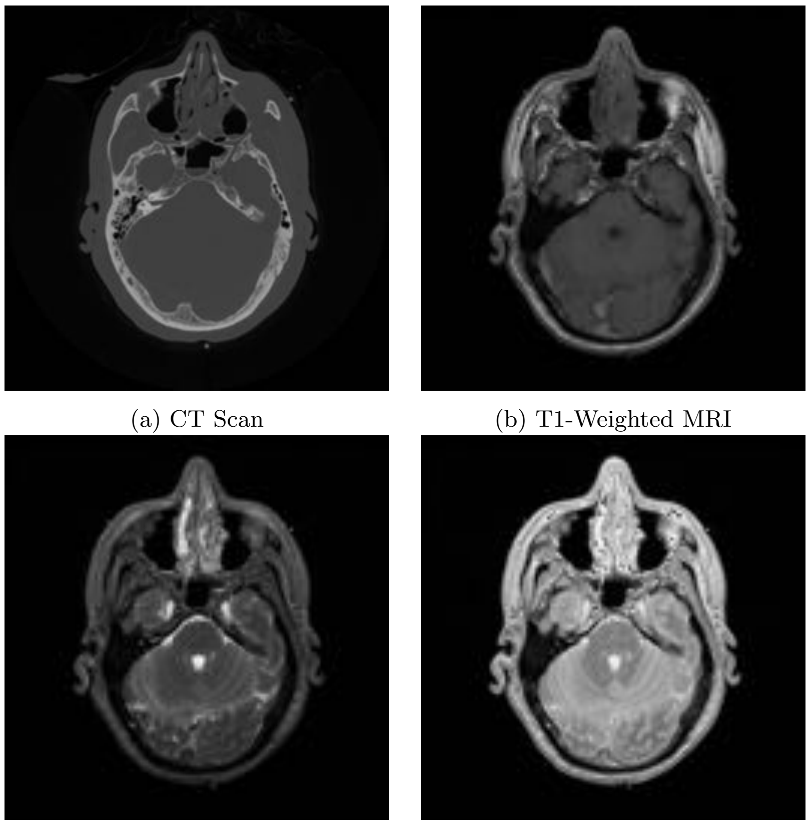


Figure 2.3: The cocoon cadaver was kept in the box with blocks of dry ice to prevent frosting, while segmentation into blocks was done with a backsaw. The blade of the saw was positioned in the marks identified with the laser[14].

2.1 Radiological Images

The radiological images consist of CT and MRI scans. The CT data are axial scans of the body, taken at 1mm intervals with a resolution of 512 by 512 pixels with 12 BITS of grey tone. The MRI data are also axial scans of the head and neck, and longitudinal sections of the rest of the body, but taken at 4mm intervals with 256 by 256 pixel resolution and 12 bits of grey tone. The CT images are formatted in GE 16 BITS (Compressed), while the MRI images are split into 3 formats each representing the three different sequences used: T1-weighted, T2-weighted and Proton Density weighted (PD), which allow for different contrasts between tissues. Sample images of radiological scans of the head can be seen in Figure 2.4.



(a) CT Scan (b) T1-Weighted MRI

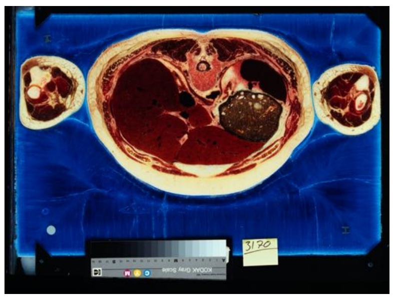
(c) T2-Weighted MRI (d) PD-Weighted MRI

Figure 2.4: Sample images of the CT and MRI scans available on the data set.

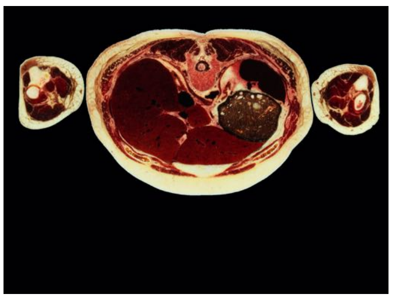
2.2 Cryosections

The cryosections come in different formats, 2K PNG, 4K TIFF and RAW (RGB 24 BIT non interleaved, compressed). There are a total of 5 189 slices, obtained at 0.33 mm intervals in order to match the sizing of the X-Y plane, making it easier to work with voxels. The images are available at a pixel resolution of 2048 pixels by 1216 pixels and 4096 pixels by 2700 pixels[9].

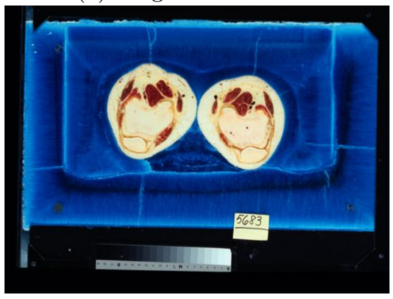
Initially, the original data set was used, but after the algorithm was developed it was used to remove the ice in the background along with other unwanted objects, as seen in Figure 2.5. It is also important to note that some slices were damaged by the cryo-procedure (Figure 2.6) along with others where parts of the body disappear, such as the fingers and/or toes, and the nose cartilage (Figure 2.7). The cadaver also contains several pathologies, including cardiovascular disease and diverticulitis. The damaged slices were not used. The slice numbers from the filenames do not match the numbers on the image. Since the background was removed, when referencing the number of the slice, the filename was used.



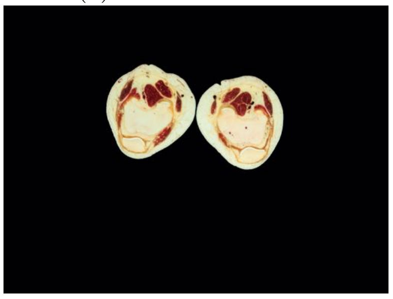
(a) Original Slice 2660



(b) Masked Slice 2660

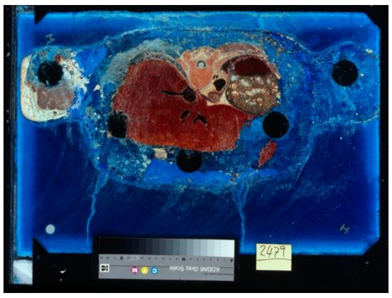


(c) Original Slice 4616

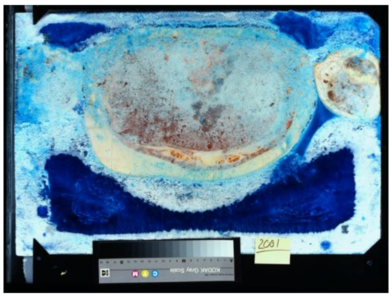


(d) Masked Slice 4616

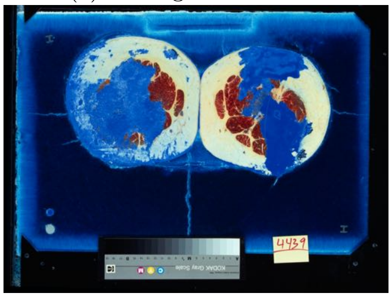
Figure 2.5: Slices 2660 and 4616 before and after background removal using the developed algorithm from this dissertation.



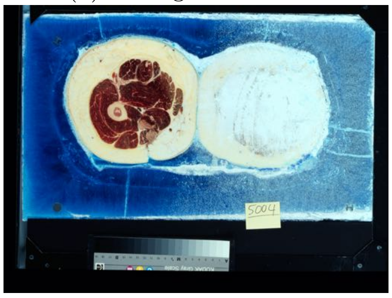
(a) Damaged Slice 2468



(b) Damaged Slice 2491

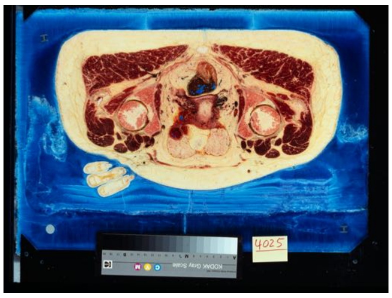


(c) Damaged Slice 3929

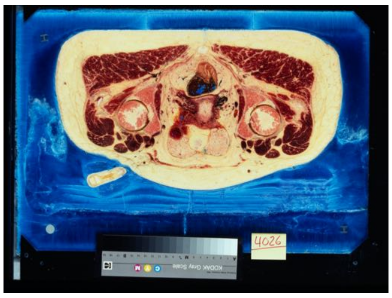


(d) Damaged Slice 3937

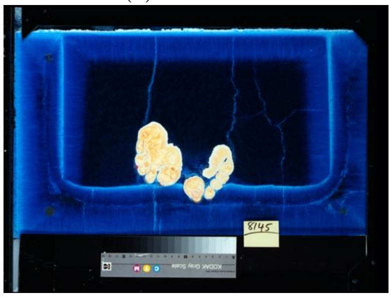
Figure 2.6: Examples of damaged slices in the dataset. The upper torso in (a) and (b) and the upper legs in (c) and (d) are badly damaged making the structures harder to identify.



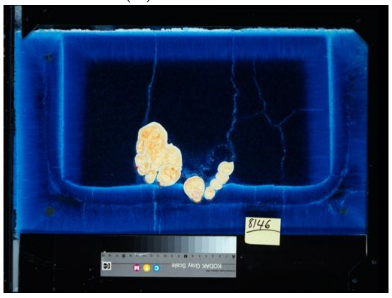
(a) Slice 3515



(b) Slice 3516



(c) Slice 6133



(d) Slice 6134

Figure 2.7: Examples of missing appendages in the dataset. From (a) to (b) two phalanges are missing and from (c) to (d) part of the foot's sole is also missing

CHAPTER

3

EXISTING EDGE DETECTION ALGORITHMS

3.1 Localized Histograms & Region Merging

In this technique, the segmentation is done based on pixel feature values. Clusters are formed with peak-valley analysis of an histogram of pixel intensity. It can also be coupled with region merging to further enhance the segmentation. This has many advantages, such as not being a purely local nor global algorithm, taking all variability into account. This is very important, since some images have huge amounts of features, such as textures, color, shadows, highlights and even fine geometric structures[15]. These systems are usually composed of two algorithms, a localized histogram-segmentation and a region-merging algorithm, as seen in Figure 3.1.

3.1.1 Local Histogram Region-Segmentation Algorithm

This algorithm consists of five phases. The first step is pre-processing the image with edge-preserving smoothing and scaling. Then, the image is divided into sectors, and histograms of pixel feature values are created. The dimension of the histograms is usually 16x16 pixels or 32x32 depending on the sharpness of the image. The sectors are usually expanded to include the boundary and overlap the neighbor sectors.

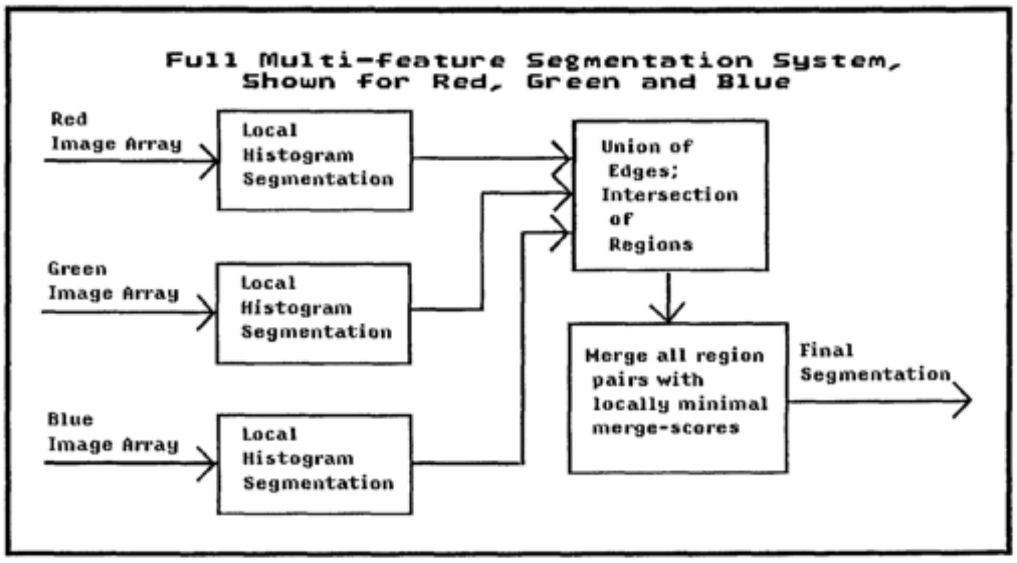

Full Multi-feature Segmentation System, Shown for Red, Green and Blue
Red Image Array
Green Image Array
Blue Image Array
Local Histogram Segmentation
Local Histogram Segmentation
Local Histogram Segmentation
Union of Edges; Intersection of Regions
Merge all region pairs with locally minimal merge-scores
Final Segmentation

Figure 3.1: Primary components of the segmentation system. First different features of an image (typically color)are independently segmented using local histogram segmentation. Then a new segmentation is created by taking the union of all the resulting edges. Finally, this highly fragmented segmentation is refined by a region-merging phase[15].

Clusters are defined by a histogram peak and its two neighboring valleys, and are created for each sector based on peak-valley analysis of the histograms. First, all peaks are marked, using local maxima. Then, the minimum valley between each peak is a classified as a valley. The clusters are then created and named according to the grey level of their peak, as seen in Figure 3.2. The clusters are evaluated according to their significance using a variety of measures: peak height, peak-valley ratio and peak distance. The user provides thresholds for these values in order to select clusters.

Pixels are labeled according to the cluster they belong to and are segmented. Subsequently, missing clusters are added, which might have been missed due to being in adjacent sectors, obscured by a larger cluster, for example. The method used to find these clusters was peak addition, by Kohler [16]. Lastly, the artificial sector boundaries are removed and in post-processing the one- and two-pixel regions are removed. All adjacent regions at a sector boundary are compared using a merge-score:

$$merge - score = \frac{\|\mu_a - \mu_b\|}{max(\sigma_a + \sigma_b, 1)} \tag{3.1}$$

where μ is the mean and σ the standard deviation. This measure is used to evaluate local (pixels at a fixed distance) and global (all pixels in the respective regions) similarity. The regions are only merged if the score is below a pre-defined threshold.

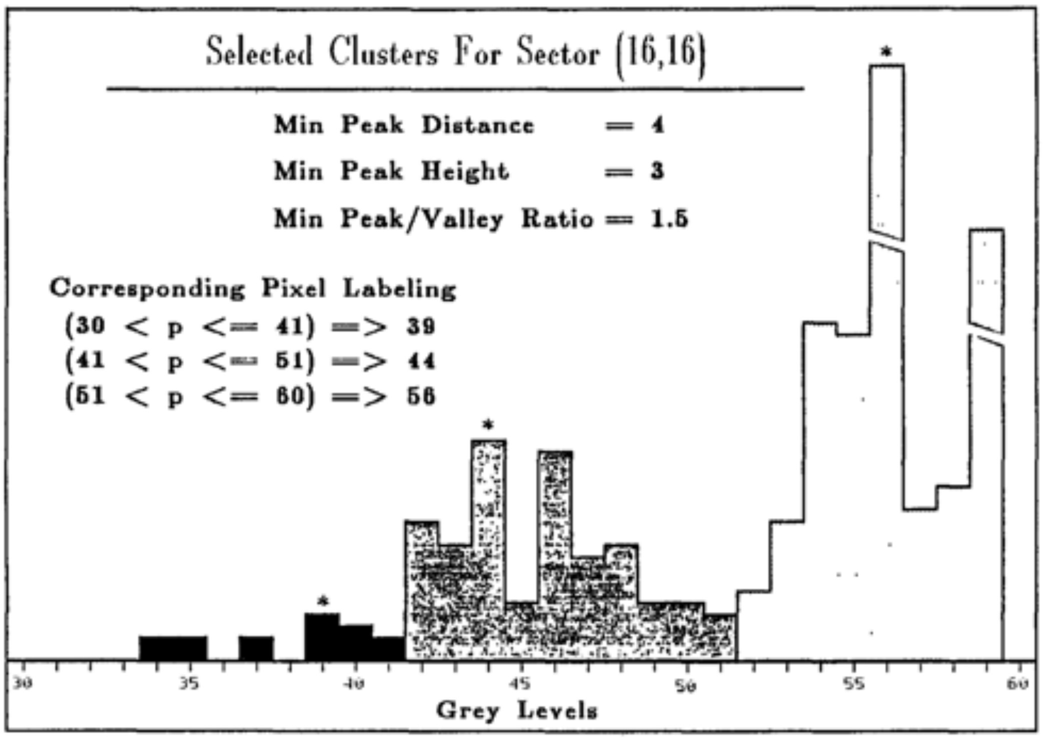


Figure 3.2: Cluster selection example illustrating the clusters selected using a specific set of cluster-filtering parameters. Peaks are noted with an asterisk[15].

3.1.2 Region Merging

This algorithm receives as input a set of fragmented regions, This strategy is a iterative process that also uses the merge-score as measure. Regions with minimal scores are identified, the regions are merged, and the process is repeated until all merge-scores are larger than the global threshold. However, in this algorithm it is also taken into account that similar regions should be merged, larger regions are preferred and regions that have a common boundary should not be merged. There is a merge-score for each of these factor that are combined into a final score.

One of the factors used is **spectral similarity**. using the following measure:

$$S_sim(a,b) = \frac{\|\mu_a - \mu_b\|}{max(1, \sigma_a + \sigma_b)} \tag{3.2}$$

where μ is the mean and σ the standard deviation. The ideal value would be 0, as this means that the mean values for the two regions are equal. If there are multiple features, such as color, the measure can be extended:

$$S_sim(a,b) = \frac{max_k\|\mu_a k - \mu_b k\|}{max(1, \sigma_a + \sigma_b)} \tag{3.3}$$

Size is also taken into account, by encouraging small regions to merge and discouraging large ones. The measure has a limit at 2.0 so that it does not override other measures.

$$S_size(a,b) = min(2.0, \frac{min(c(a), c(b))}{k}) \tag{3.4}$$

where

$$c(x) \equiv \text{the number of pixels in region x}$$

$$k \equiv \text{target number of pixels in a region, user-specified}$$

The last measure used is connectivity, which reflects how large the boundary between regions is. Should the boundary be small, it is less likely that the two regions should be merged. This measure is also bounded between 0.5 and 2.0 to avoid overpowering the other measures. It is defined as follows:

$$\begin{array}{rcll} & = & c(a,b) & \text{if } 0.5 \leq c(a,b \leq 2.0 \\ S_conn(a,b) & = & 0.5 & \text{if } c(a,b) < 0.5 \\ & = & 2.0 & \text{otherwise} \end{array}$$

where

$$c(a,b) \equiv \frac{min(l(a), l(b))}{4 \times l_S(a,b)} \tag{3.5}$$

$$l_s(a,b) \equiv \text{the length of the shared boundary between regions a and b}$$

$$l(x) \equiv \text{the circumference of region x}$$

The final merge-score is the product of these three features. The size-score has a weight of 0.5 as it is less important to the score overall. These formulas were obtained through experimentation and have proved to be effective on a wide variety of images.

Other methods were taken into account, such as similarity of standard deviation, local contrast across the boundary, maximum allowed standard deviation, and mixed pixel region suppression. Even though these added features proved effective in certain cases, the excess in computation was not worth it.

3.1.3 Combining The Algorithms

The junction of these two algorithms counteract the disadvantages of using each of them by themselves. The local-histogram segmentation is very accurate at detecting homogeneous areas. However, it tends to create too much small regions in textured parts of the image. Even though the parameterization can be changed to tackle this problem, significant image boundaries are lost in the process.

On the other hand, the region-merging algorithm is not as global, it is more sensitive to local image characteristics. It only merges regions, it cannot create new boundaries. Therefore, these two algorithms work very well together, where the local-histogram segmentation creates a good starting segmentation and the region merging corrects the posprocess fragmented areas.

3.1.4 Integrating Multiple Color Features

In this algorithm, the use of multiple features is also explored, which is known to improve segmentation quality. Regions that are not found in one feature might be found in others, as seen in Figure 3.3[17][18]. Each feature is independently segmented at high sensitivity, and the result is joined to create a new segmentation, called multispectral-union. Pixels that are separated in one of the colors are also separated in the union. The region merging algorithm is then applied to this oversegmented image.

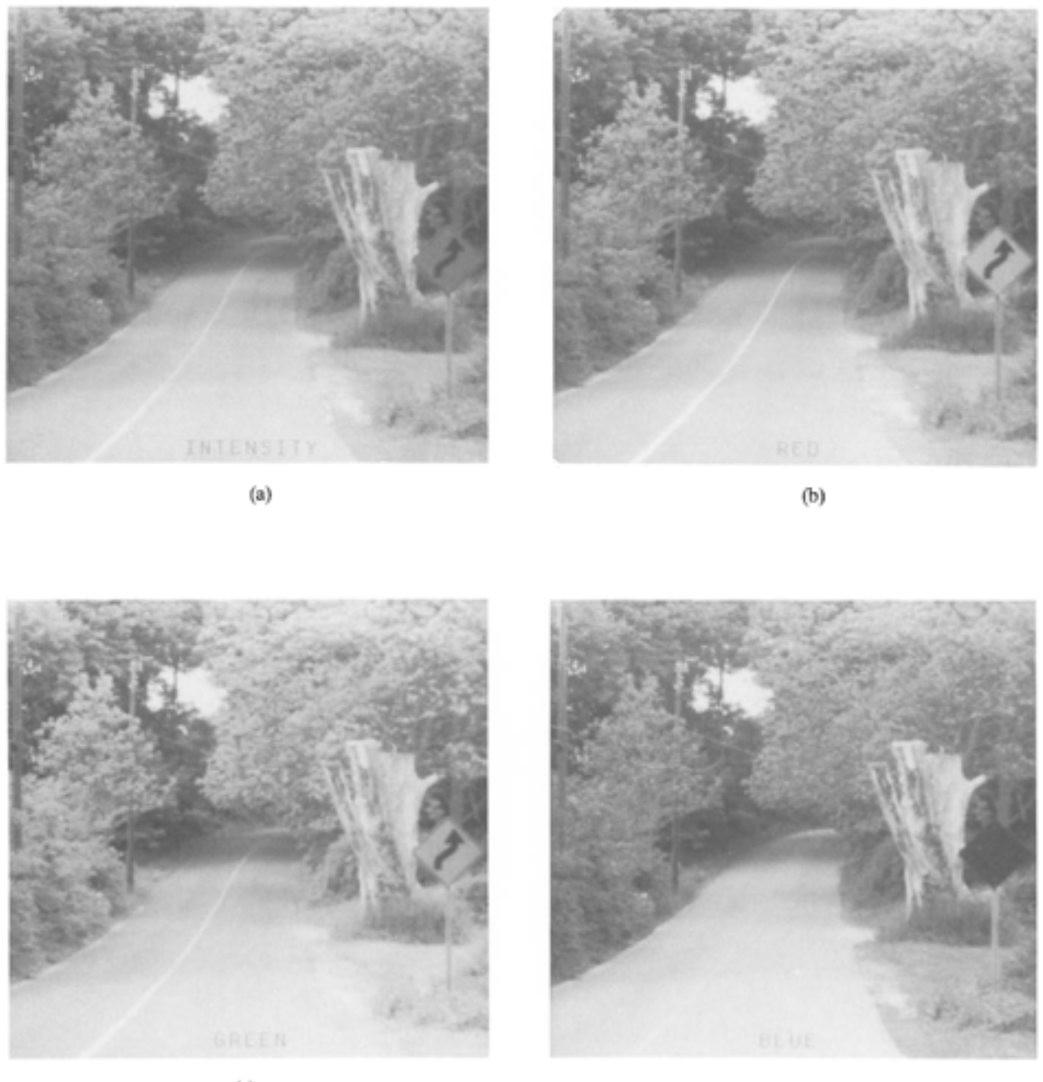


Figure 3.3: Color data for a road scene. (a) Instensity, (b) red, (c) green, and (d) blue components. [15].

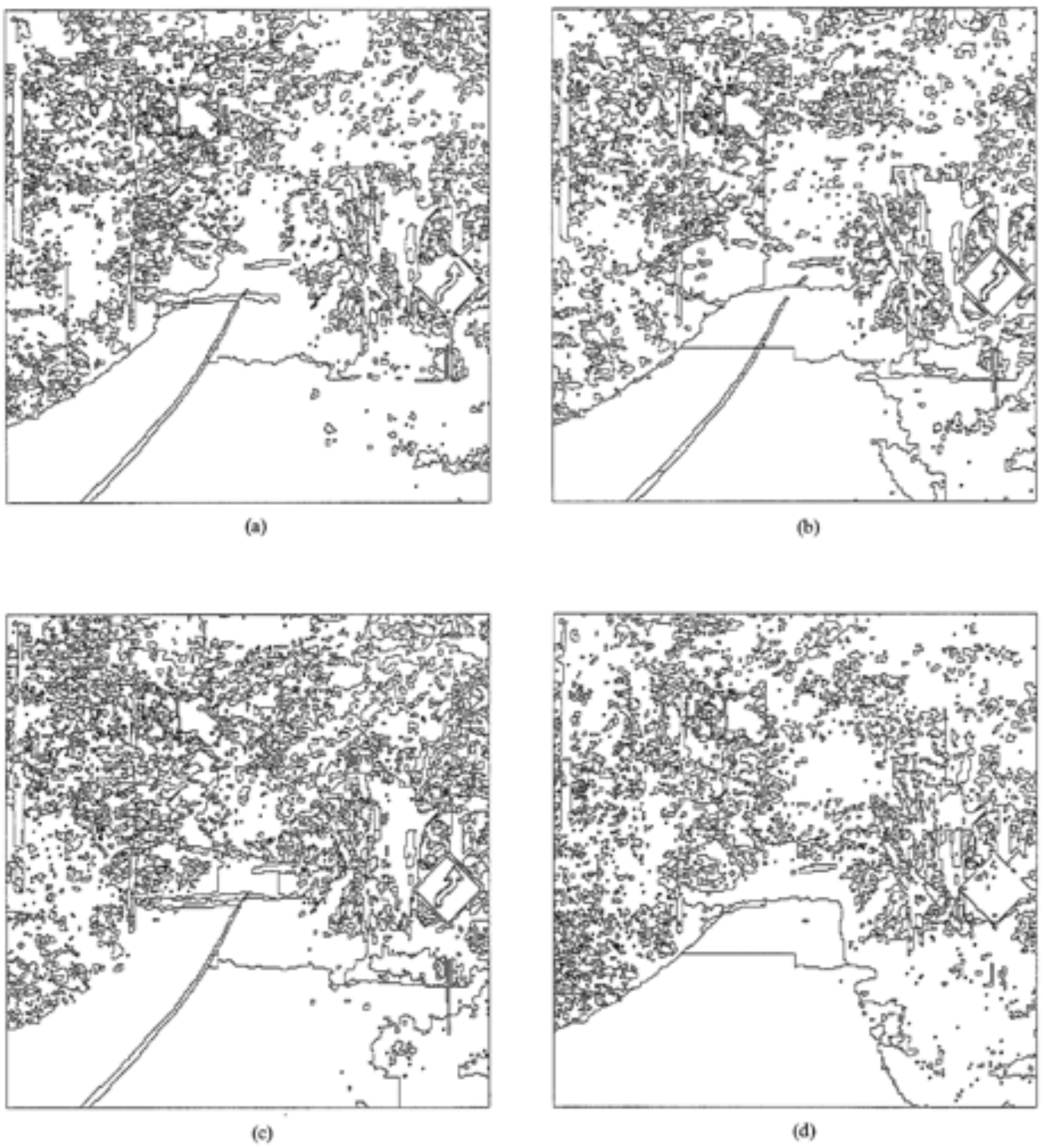


Figure 3.4: Segmentation of the individual color components. High-sensitivity segmentation of the (a) intensity, (b) red, (c) green, and (d) blue data for the road scene shown in Figure 3.3. [15].

3.1.5 Parameterization

Each operation is controlled with a variety of parameters that need to be chosen appropriatly in order to have a successful segmentation. The size of parameter space for all functions is rather large, creating a problem of parameterization for each new image. In order to tackle this problem, the goal was to collapse the parameter space into a much smaller one, so that the algortihm is effective across a variety of image types. The only parameter needed for the localized histogram segmentation is sensitivity (very-low, low, medium, high, and very-high). [15].

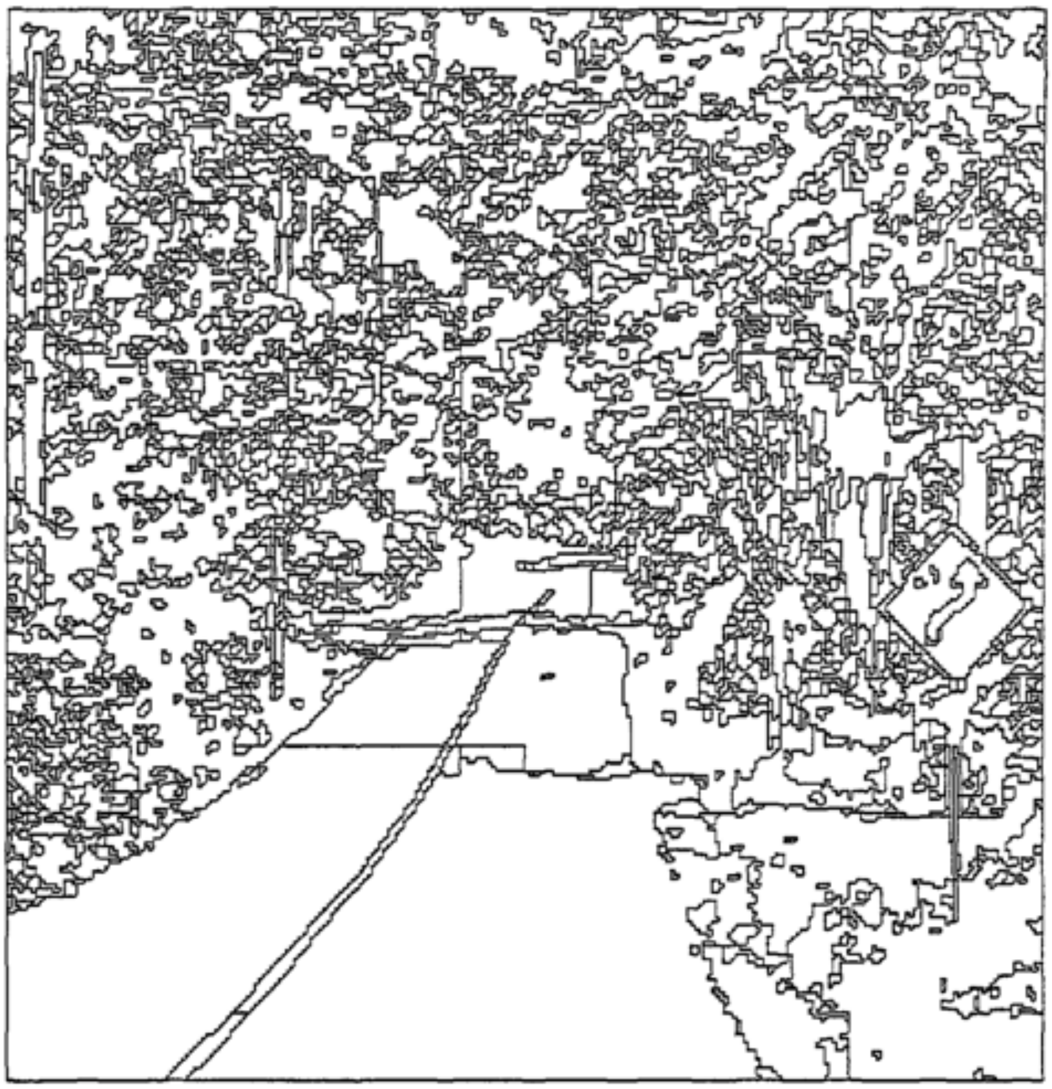


Figure 3.5: Illustration of the three-color-union segmentation at high sensitivity by taking the boundaries between regions in each of the three color segmentations shown in Figure 3.4. [15].

For cluster selection, the parameters (minimum-peak-distance, minimum-peak-height and minimum-peak-valley ratio) are measured using the average height as a reference and a, less complicated, user input. Like the previous parameterizations, the user chooses a sensitivity, where low sensitivity performs a two-class threshold on the image while a high sensitivity setting indentifies each individual peak as a cluster. [15].

Lastly, in order to remove the boundaries, three settings are used: weak, normal and strong sector-boundary removal. A local maximum merge-score threshold of 2.5 is used on all three, while the global maximum varies accordingly from 0.6 to 1.1 and 1.7.

3.2 Gradient Based Algorithms

3.2.1 Canny Edge Detection Based Algorithms

The Canny Edge Detector is widely used to extract structural information in computer vision, reducing the data load that needs to be processed. This algorithm was developed by John F. Canny in 1986 and has since been used due to its good detection, localization and single response to an edge. [19]. It starts by receiving an image and outputs the computed edges of the entire image.

The algorithm consists of a few simple steps. Firstly, some pre-processing is applied to the image, to reduce noise and undesired image details, usually a 2D Gaussian filter.

$$G(x,y) = exp[-(x^2+y^2)/2\sigma^2]/2\pi\sigma^2 \tag{3.6}$$

Secondly, the gradient magnitude and direction is calculated for each pixel. The first order partial derivatives are obtained with the following formulas:

$$E_x[i,j] = (I[i+1,j] - I[i,j] + I[i+1,j+1] - I[i,j+1])/2 \tag{3.7}$$

$$E_x[i,j] = (I[i,j+1] - I[i,j] + I[i+1,j+1] - I[i+1,j])/2 \tag{3.8}$$

The magnitude and direction can then be calculated:

$$\|M(i,j)\| = \sqrt{E_x[i,j]^2 + E_y[i,j]^2} \tag{3.9}$$

$$\theta(i,j) = arctan(E_y[i,j]/E_x[i,j]) \tag{3.10}$$

Should the gradient magnitude be larger than its two neighbor pixels, the pixel is marked as an edge. Otherwise, it is marked as background. Lastly, the weak and undesired edges are removed using the hysteresis thresholding[20].

Even though the Canny Method is the most accurately defined, it is rather simple and often misses simple edges, as seen in Figure 3.6[19]. It cannot detect branching edges and, when the gradient slopes are very small, the computed direction might not be the normal direction to the contour which leads to missing edges[19].

In order to tackle these problems, several adjustments have been developed to try and improve this method, mainly changing the third step of the algorithm - where the pixel is marked as an edge or background.

One approach is to create a new category that consists of pixels whose gradient magnitude is larger than their neighbors, but not necessarily in the gradient direction.

Then, these pixels can be used to connect and fill missing edges. Some extra steps are taken to prevent false loops. This revision adds the following steps: If the gradient magnitude at a pixel is bigger than its neighbors in the gradient direction, then it is marked as a major edge. If this happens in any direction, it is marked as a minor edge. Otherwise, it is marked as background. Then, the minor edges are partitioned at the branch points and all branches which do not contain major edges are removed. The minor edges that are delimited by major edges are then remarked as major edges, and are combined with the previous edges.

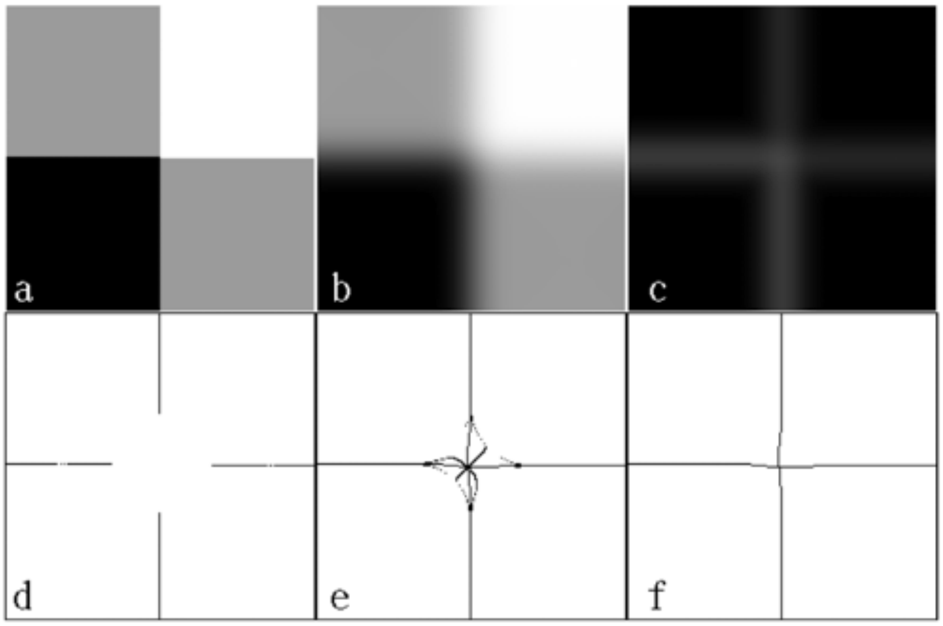


Figure 3.6: (a) An image with four well-defined homogeneous regions. (b) Image after smoothing with a Gaussian kernel of standard deviation 11 pixels. (c) Gradient magnitudes of (b). (d) Edges of (a) determined by the Canny edge detector with a Gaussian smoother of standard deviation 11 pixels. These are the major edges. (e) The minor edges. (f) Edges obtained by the revised Canny edge detector[19].

Overall, the results are promising and fix the main problem with the Canny Method, which is the missing edges. Even though this problem can be fixed with the zero-crossings of the Laplacian of Gaussian[21], they can sometimes create phantom-edges between objects.

Other approaches seek to tackle the inaccurate localization of the Canny Method and the implication that the optimal edge detector would be infinitely wide - which leads to interference from other edges[22]. Some also change the way the gradient is calculated in order to reduce noise sensitivity, using other methods, such as gravitational edge detection[20].

3.2.2 Edge detection using normal direction

Segmentation of CT scans is quite important and is sometimes done using gradient based algorithms, since contrast in these images is high for some tissues, such as bone. This algorithm is based on the fact that the variation of the intensity of pixels near the edge is greater along the normal direction when compared to the tangent direction. The estimation is usually done using the first derivative of the 2-D symmetrical Gaussian as it estimates the normal direction and also reduces noise at the same time.

$$G_n = \frac{\partial G}{\partial n} = n \cdot \Delta G \tag{3.11}$$

$$G(x) = exp(-\frac{||x||^2}{2\sigma^2}) \tag{3.12}$$

where $x \in R^2, \sigma$ is the estimate of the variance of the noise. The normal direction can be estimated as

$$n(x_0) = \frac{\Delta(G(x - x_0) * I)}{||\Delta(G(x - x_0) * I)||} \tag{3.13}$$

where * is the convolution and I is the image. It is assumed that σ can be estimated and that the normal direction will not change after a coordinate system rotation, as it is calculated on the original coordinate system. The noise variation can be estimated using an auto correlation function, but it is extremely inaccurate and the estimated normal directions can be very different if the system rotates even slightly[23].

There have been many attempts to solve this problem, such as estimating the edge strength in rotated coordinate systems, but it is costly to compute[24]. Another approach is to use the geometrical information available to improve the results obtained. If the edge being contoured is C^2continuous and smooth, which is often the case, then the true edge location of point $i + 1$ and its normal direction can be computed with the detected edge points.

When the contour is a circle, the difference between the angles spanned by two consequential pairs of neighbouring pixels should be 0.

$$\Delta\theta_i^2 = \Delta\theta_i - \Delta\theta_{i-1} = 0 \tag{3.14}$$

where θ_i is the angle spanned by the normal direction at pixel i and the horizontal coordinate and $\Delta\theta_i$ is the angle spanned by the normal directions at two neighbouring pixels $i - 1$ and i. This is not always true, since the contours are not always circles. Therefore, the difference is instead assumed to be a random variable ξ around zero, $\langle\xi\rangle = 0$. Even though the expected value is zero, the variance, on the other hand, can be quite large. Estimation errors can cause a large variance of ξ, so an adjustment was made to correct this:

$$\bar{\theta_i} = q_i\theta_i + (1 - q_i)(\bar{\theta}_{i-1} + \Delta\bar{\theta}_{i-1}) \tag{3.15}$$

where $q_i \in [0, 1]$ is the adjustment/balancing factor, θ_i is the normal direction obtained from equation 3.13, and $\bar{\theta}_{i-1} + \Delta\bar{\theta}_{i-1}$ is the normal direction but estimated through the edge smoothness assumption. Therefore, q_i acts as a way to determine the reliability of these two estimates.

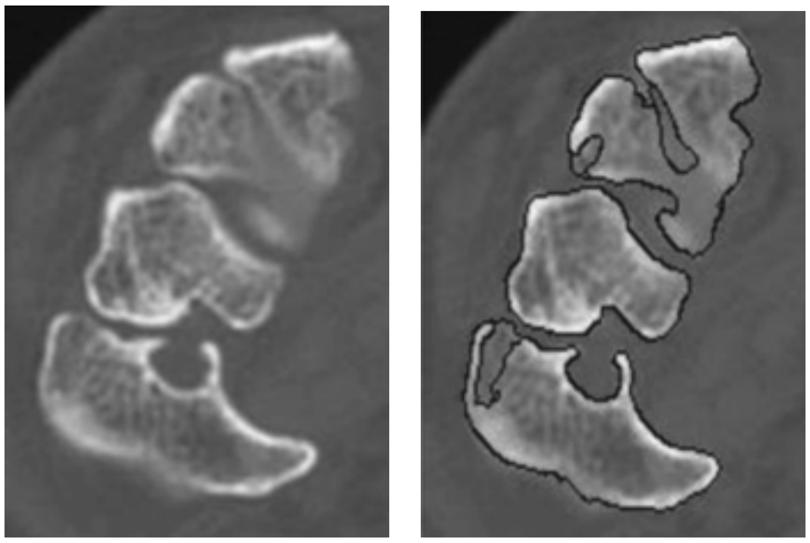
In order to calculate q_i, the reliability of the computed normal direction across different levels of noise is used:

$$q_i = \frac{\sigma_u^i - \sigma_l^i}{\sigma_{max} - \sigma_{min}} \tag{3.16}$$

where $\sigma_u^i \leq \sigma_{max}$ and $\sigma_l^i \geq \sigma_{min}$ are the upper and lower bounds of σ where the discrepancy of the normal direction at pixel i are less than or equal to a defined tolerance, which controls the sensitivity of the estimate. However, the value of q_i relies too much on the tolerance defined, so some adjustments can also be made. A re-scaling factor γ can be added to the equation:

$$\bar{\theta}_i = \gamma q_i \theta_i + (1 - \gamma q_i)(\bar{\theta}_{i-1} + \Delta\bar{\theta}_{i-1}) \tag{3.17}$$

The results from this algorithm are promising (Figures 3.7 and 3.8) and when in comparison with other methods, the results are quite accurate, however require a set of tissue-specific parameters beforehand for the segmentation to be ideal[25].



(a) Wrist CT Scan (b) Segmented CT Scan

Figure 3.7: CT scan of a human wrist from patients who volunteered in a study approved by the Research Ethics Board of Queen's University and Kingston General Hospital. In (a) is the original CT scan and in (b) is the edge map computed with the normal direction algorithm[25].

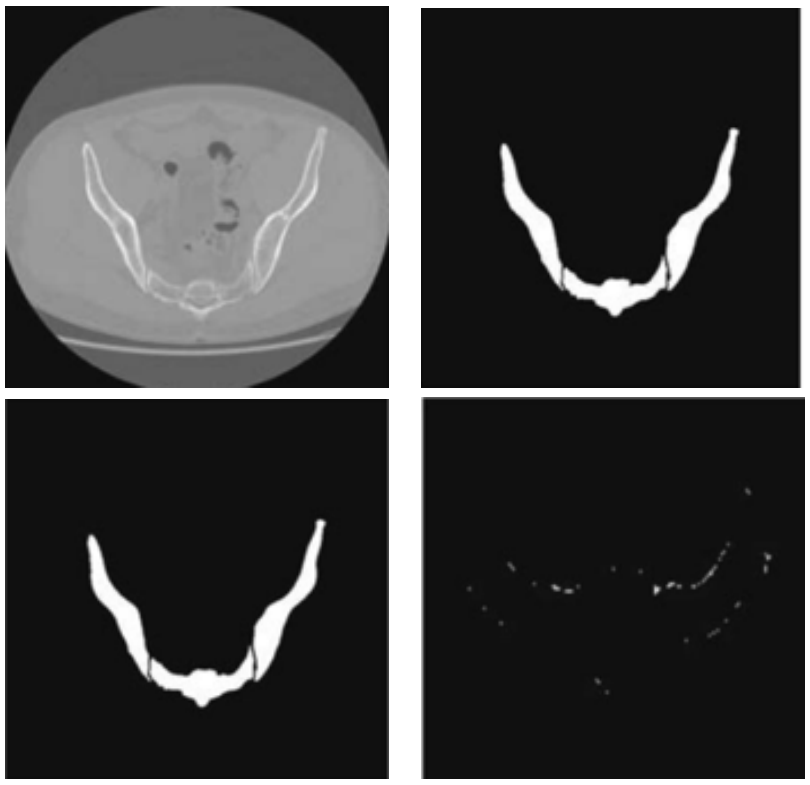


Figure 3.8: CT scan of a human pelvis from patients who volunteered in a study approved by the Research Ethics Board of Queen's University and Kingston General Hospital. In (a) is the original CT scan, in (b) is the manual segmentation of the pelvis, in (c) is the edge map computed by the algorithm, and in (d) is the difference between the two[25]

3.2.3 Active Contouring/Snakes

Active contours, also known as snakes, are widely used in shape recognition, segmentation and edge detection. This method requires previous knowledge of the contour, needing an input. It is very useful when the initial seed is close to the desired edge, resulting almost always in convergence[26].

A snake is defined by a list of points v_i. The deformations of the snake are made according to an internal energy term, $E_{internal}$, and the fitting of the contour to the edge is controlled by an external energy term, $E_{external}$, which is a combination of the image forces, E_{image}, and constraint forces, E_{con}, implied by the user. The energy function of the snake is therefore a sum of these two:

$$E^*_{snake} = \int_1^0 E_{snake}(v(s))ds = \int_1^0 (E_{internal}(v(s)) + E_{image}(v(s)) + E_{con}(v(s)))ds \tag{3.18}$$

The internal energy is defined by the continuity E_{cont} and smoothness E_{curv} of the contour:

$$\begin{aligned} E_{internal} = E_{cont} + E_{curv} &= \frac{1}{2}(\alpha(s)|v_s(s)|^2) + \frac{1}{2}(\beta(s)|v_s s(s)|^2) \\ &= \frac{1}{2}(\alpha(s)||\frac{d\bar{v}}{ds}(s)||^2 + \beta(s)||\frac{d^2\bar{v}}{ds^2}(s)||^2) \end{aligned} \tag{3.19}$$

where $\alpha(s)$ and $\beta(s)$ are user defined weights which control, respectively, the stretch and curvature of the snake.

The image energy consists of a weighted combination of three energy functionals - lines, edges and terminators. Naturally, higher weights mean these features have a larger contribution to the image force.

$$E_{image} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term} \tag{3.20}$$

The line energy is the intensity of the image, represented by:

$$E_{line} = I(x, y) \tag{3.21}$$

Smoothing/Noise reduction filters are usually applied in this step. The edge functional is based on the gradient of the image:

$$E_{edge} = -|G_\theta \cdot \nabla^2 I|^2, \tag{3.22}$$

where G_θ is a Gaussian with standard deviation θ in order to prevent convergence to local minima. The level lines in a smoothed image can be used to find terminators and corners across the image:

$$E_{term} = \frac{\partial\theta}{\partial n_\perp} = \frac{\partial^2 C/\partial n_\perp^2}{\partial C/\partial n} = \frac{C_{yy}C_x^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{(C_x^2 + C_y^2)^{3/2}}, \tag{3.23}$$

where $C(x, y) = G_\theta \cdot I(x, y)$ is the smoothed image, $\theta = arctan(C_y/C_x)$ is the gradient angle, $n = (cos\theta, sin\theta)$ are the unit vectors along the gradient direction and $n_\perp = (-sin\theta, cos\theta)$ are the unit vectors perpendicular to the gradient direction.

The gradient descent method is used in order to optimize the energy minimization process and to help find local minima:

$$\bar{v}_i \leftarrow \bar{v}_i + F_{snake}(\bar{v}_i) \tag{3.24}$$

$$F_{snake}(\bar{v}_i) = -\nabla E_{snake}(\bar{v}_i) \tag{3.25}$$

Images have finite resolution. Therefore, they can only be integrated over discrete time steps.

$$E^*_{snake} \approx \sum_1^n E_{snake}(\bar{v}_i) \tag{3.26}$$

Also, in order to prevent oscillations around minima or converging to a different one, the step size should never be bigger than one pixel due to the image forces[26].

Compared to traditional methods, snakes have many advantages, such as tracking of dynamic objects, the ability to adaptively search for the minimal state and scale sensitivity to the Gaussian smoothing in the energy function. However, the method is very sensitive to local minima, small details on the image are often ignored and the accuracy depends on the convergence policy.

As such, there are many variations of this model that try to solve its limitations, but each of them have their own trade-offs.

GVF Snake Model: This model replaces the default external force. It uses the gradient vectors' diffusion of a gray-level or binary edge map obtained from the image (Figure 3.9). It solves the poor convergence for concave edges and initilizations far from the minimum. However, this causes rounding of the edges of the contour[27].

Geometric Active Contours: GAC takes inspiration from Euclidean curve shortening and level sets to create a model where contours can split and merge as objects are detected on the image. Active contours can handle changes in topology during the gradient descent evolution. This model has been used in medical imaging (Figure 3.10) and can be extended to 3D. [28].

Knowledge-guided Robust Active Contours: This segmentation technique combines a knowledge based segmentation system with the active contours. The knowledge system is built using probability density functions using location, size and image intensity as parameters. However, it is only used for low-level segmentation (Figure 3.11)[29].

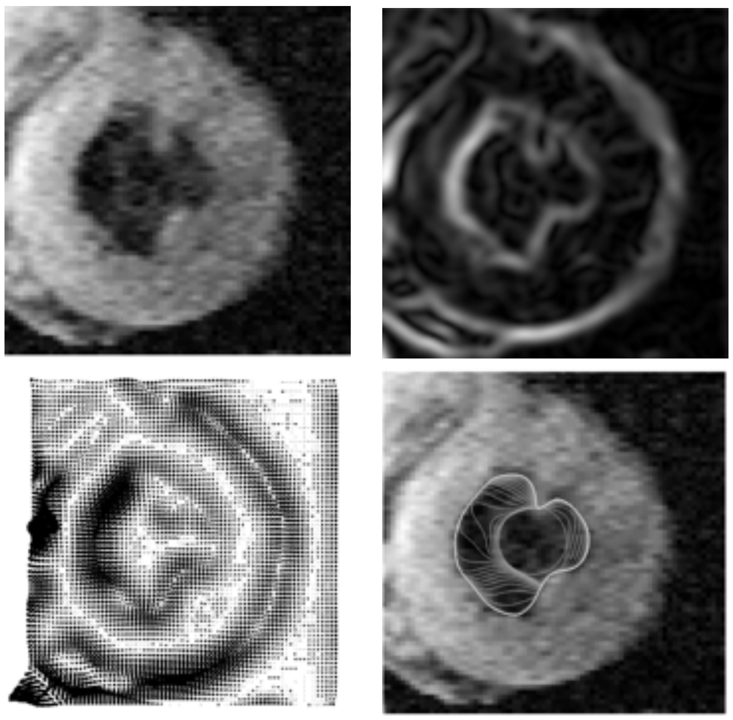


Figure 3.9: Magnetic resonance image of the left ventricle of a human heart (a), the corresponding edge-map (b), the GVF (c) and the initial and intermediate contours (gray) and final contour (white) of the GVF snake (d)[25].

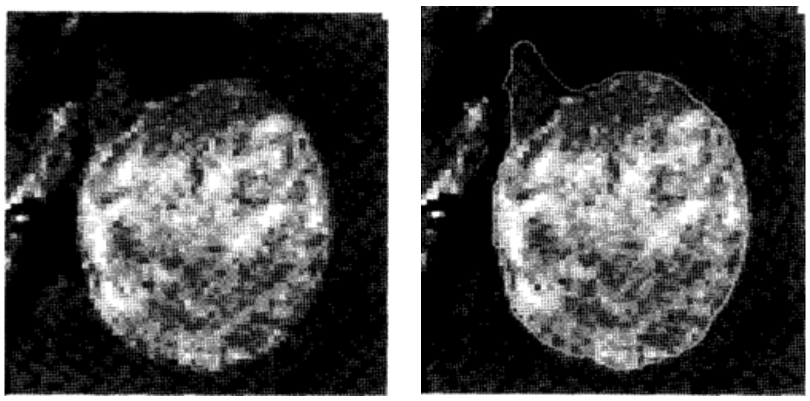


Figure 3.10: Tumor detection using GAC. The tumor in the image is a acousticus neurinoma. In (a) is the original MRI image and in (b) is the segmentation obtained. [28].

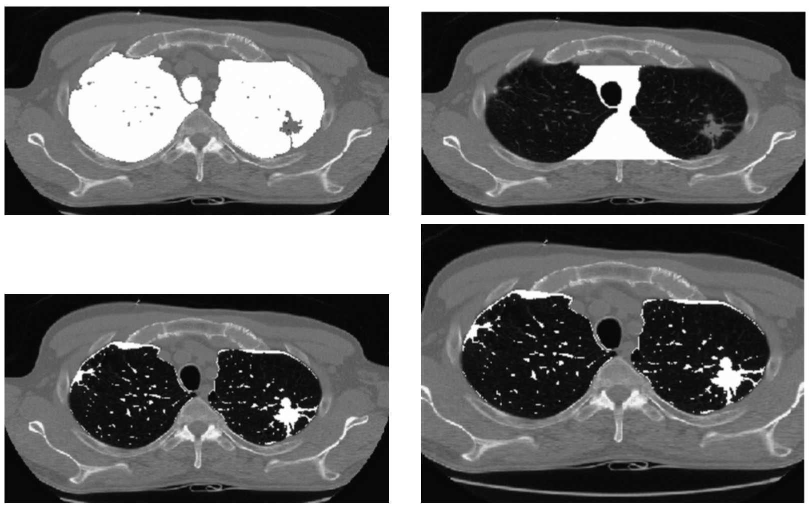


Figure 3.11: Results of using the Knowledge-guided Robust Active Contours on a chest CT scan. Regions are automatically identified without user input. In (a) is the segmented chest wall, in (b) the airspace, in (c) the mediastinal region and in (d) areas of increased attenuation within the airspace[29]

3.3 Graph Cut Based Algorithms

This method is quite different from the ones described above. It was initially developed by Boykov and Jolly in 2001 using a monochromatic image. The segmentation started with a trimap T and an array of grey values, then histograms were built using the respective trimap regions for background and foreground in order to calculate the opacity values and models.

First, let's focus on the concept of graph cut. A graph is defined as $G =< V, E >$, where V are the vertices of the graph and E is the graph edge that connects every pair of neighbor pixels. There can be two different kinds of nodes, there are neighborhood nodes which are the pixels, and terminal nodes (s (source) and t (sink)). In this graph, also called an $s-t$ graph, the s is used to represent the object and t is used to denote the background. There can also be two different types of edges. The first type are the *n-links* which connect the neighboring pixels within the image, and the second type are the *t-links*, which connect the terminal nodes with the neighborhood nodes. Each edge is assigned a non-negative weight w_e, called cost. A cut, which is a subset of edges E, can be denoted as C, $C \subset E$. The cost of a cut is the sum of the weight of the edges:

$$|C| = \sum_{e \in C} w_e \tag{3.27}$$

A min-cut is a cut that has the least cost and can be achieved by finding the max flow[30] With this cut, the graph is divided and the nodes separated into two subsets S and T, where $s \in S, t \in T$ and $S \cup T = V$. The subsets represent the foreground and background, respectively[31].

With this method, to segment an image is to label the pixels as either background or foreground. In order to achieve this, the energy-function is minimized through minimum graph cut. The energy function should be minimum at the projected boundary. The energy function is defined by:

$$E(L) = \alpha R(L) + B(L), \tag{3.28}$$

where $L = l_1, l_2, l_3, ..., l_p$ are the labels, p is the number of pixels in image, $l_i \in 0, 1$, $R(L)$ is the regional term, $B(L)|$ is the boundary term and α is the relative importance factor. The regional term is defined as:

$$R(L) = \sum_{p \in P} R_p(l_p), \tag{3.29}$$

where $R_p(|l_p)$ is the penalty for using the label l_p to pixel p. This weight can be calculated by comparing the histograms of the foreground and background with the pixels' intensity.

$$R_p(0) = -lnP(I_p|'fg') \tag{3.30}$$

$$R_p(1) = -lnP(I_p|'bg') \tag{3.31}$$

Therefore, when a pixel is likely to be in the foreground, the penalty for assigning the foreground label is smaller which in turn reduces the energy in eq. (3.28) If two neighboring pixels have the same label, then the penalty is zero.

On the other hand, the boundary term is defied as:

$$B(L) = \sum_{p,q \in N} B_{<p,q>} \cdot \delta(l_p, l_q), \tag{3.32}$$

where p, q are the neighbor pixels and:

$$\delta(l_p, l_q) = \begin{cases} 1 & \text{if } l_p = l_q \\ 0 & \text{if } l_p \neq l_q \end{cases} \tag{3.33}$$

When the intensity of two neighbor pixels is the same, the penalty is high, and if it the intensity of the pixels is very different, then the penalty is low. This way, the energy function is minimum at the foreground edge. [31].

The weight in the $s - t$ graph is very important, since its value affects the segmentation result. In Boykov and Jolly's method, the weight is defined as:

$$weight = \begin{cases} B_{<p,q>} & p, q \in \text{Neighboring pixel} \\ \alpha \cdot R_p(0) & \text{for edge} p, S \\ \alpha \cdot R_p(1) & \text{for edge} p, T \end{cases} \tag{3.34}$$

When the intensity of the pixel is close to the foreground, the weight between this pixel and the *s-node* will be larger, meaning that the cut will occur at the edge with smaller weight. When two pixels have the same intensity, the weight is very big, therefore, they are not likely to be separated[31].

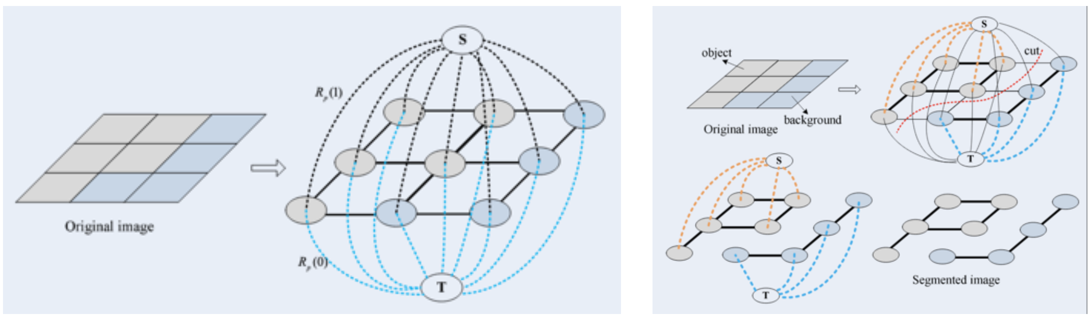


Figure 3.12: In (a) is the illustration of a *s-t* graph, where the pixels correspond to the neighbor nodes in the graph, the solid lines are *n-links* and the dotted lines are *t-links*. In (b) is the illustration of the cut corresponding to the minimal energy[31].

3.4 Comparison

The Canny Edge Detection algorithms while effective, are heavy in computation, rely heavily on proper parameterization and do not use global information. This last aspect is also true for other gradient-based algorithms. These algorithms are best suited for images with high contrast, such as CT and MRI scans.

Localized Histograms and Region Merging make use of global information to detect edges, which is an improvement, and there are multi-feature implementations to segment images of color. However, the results obtained are not always closed contours and the process is fully automatic, which is not what is needed for this project.

As for methods using graph cut, the fact that they can detect boundaries even when the color separation between foreground and background is not very clear makes it very useful to segment the cryosections available in the data set. The main problem with this method is the heavy user interaction needed to produce an image, however, this can be solved with a few bridges between the user and the algorithm.

Out of all the methods presented above, Graph Cut stands out, as later iterations of the algorithm start using RGB color space, the histograms are no longer used and Gaussian Mixture Models have been implemented. Also, based on the results from the other algorithms, the use of Graph Cut is the most suited for the objective of this dissertation.

CHAPTER

4 METHODOLOGY

4.1 OpenCV Grabcut

Initially, the algorithm was being built from scratch. However, this idea was rapidly given up on as there were already plenty of good existing algorithms that use graph cuts to work on top of. One of those was Grabcut.

GrabCut was designed by Carsten Rother, Vladimir Kolmogorov and Andrew Blake from Microsoft Research Cambridge, UK. The motivation was the same, an algorithm was needed to extract the foreground of an image with minimal input.

The segmentation of the image, **z**, is an array of opacity values $\underline{\alpha}$ at each pixel, between 0 and 1, where 0 is background and 1 is foreground. A parameter, θ, is used, that describes the grey-level distribution using histograms from the trimap regions T_B, T_F, as seen below:

$$\theta = \{h(z;a), \alpha = 0, 1\} \tag{4.1}$$

In order to segment the image, a "Gibbs" energy function is used:

$$E(\alpha, \theta, z) = U(\alpha, \theta, z) + V(\alpha, z) \tag{4.2}$$

A good segmentation occurs when the energy is minimum, which means that that it was guided by both grey-level histograms and the opacity is coherent. The term U evaluates the fit of the opacity distribution to the data, given the histogram model:

$$U(\alpha, \theta, z) = \sum - \log h(z_n; \alpha_n) \tag{4.3}$$

As for the smoothness term, it can be written as:

$$V(\alpha, z) = \gamma \sum_{(m,n) \in C} dis(m,n)^{-1} [\alpha_n \neq \alpha_m] exp - \beta(z_m - z_n)^2 \tag{4.4}$$

where **C** represents the pair of neighboring pixels and *dis* the Euclidean distance. Pixels are defined as neighbors if they are adjacent horizontally, vertically or diagonally. This term measures how smooth the label is between these neighbor pixels. The constant β is defined to be:

$$\beta = (2\langle (z_m - z_n)^2 \rangle)^{-1} \tag{4.5}$$

where $\langle\rangle$ denotes expectation over an image sample. This ensures that the smoothness term oscilates between high and low contrast appropriatly. The constant γ was defined as 50, as this value optimized performance over the training set used.

The segmentation can now be estimated by minimisation:

$$\alpha = argmin_{\alpha} E(\alpha, \theta) \tag{4.6}$$

By finding the global minimum the segmentation is estimated[32].

4.1.1 Colour Data Modelling

The original algorithm was made for monochromatic images, with the use of grey-scale histograms. By replacing this step with a Gaussian Mixture Model, GMM, the image can now be in the RGB color space. Two GMMs are needed, one for the background and another for the foreground, and each one of them consists of a full-covariance Gaussian mixture with K components. A unique GMM component is assigned to each pixel, either from the background or the foreground model. An additional vector is needed to implement this, vector $k = \{k_1, \ldots, k_n, \ldots, k_N\}$ with $k_n \in \{1, \ldots, K\}$.

With this implementation, the Gibbs energy is now:

$$E(\alpha, k, \theta, z) = U(\alpha, k, \theta, z) + V(\alpha, z) \tag{4.7}$$

The data term is now changed to take into account the colour GMM models:

$$U(\alpha, k, \theta, z) = \sum_n D(\alpha_n, k_n, \theta, z_n) \tag{4.8}$$

with

$$D(\alpha_n, k_n, \theta, z_n) = -\log h(z_n|\alpha_n, k_n, \theta) - \log \pi(\alpha_n, k_n), \tag{4.9}$$

where *p()* is the Gaussian probability distribution, and $\pi()$ are mixture weighting coefficients. Therefore, the parameters of the model are:

$$\theta = \{\pi(\alpha, k), \mu(\alpha, k), \Sigma(\alpha, k), \alpha = 0, 1, k = 1 \dots K\} \tag{4.10}$$

The weights π, means μ and co-variances Σ of the 2K Gaussian components for the foreground and background are now used. The only change to the smoothness term is the contrast term, that is now calculated using the Euclidean distance in colour space:

$$V(\alpha, z) = \gamma \sum_{(m,n)\in C} dis(m, n)^{-1}[\alpha_n \neq \alpha_m] exp - \beta||z_m - z_n||^2 \tag{4.11}$$

4.1.2 Iterative Energy Minimization

The one-shot minimum cut estimation algorithm was also replaced with a more powerful and effective iterative method. This brings many advantages, such as allowing automatic refinement of the opacities $\underline{\alpha}$ as new pixels from the T_U region are labelled and used to refine the colour GMM parameters.

In the first step, the GMM components are assigned. Then the GMM parameters are learned from the data **z**. The segmentation is estimated using min cut and the process is repeated until convergence is found.

This reduces the amount of user interaction needed to segment an image by allowing the use of incomplete trimaps. The user can now only select the background region, leaving the foreground region empty. The iterative minimization applies provisional labels on pixels in the empty region which can then be retracted[32].

4.1.3 Transparency and Border Matting

Another useful implementation is the use of matting tools to produce continuous alpha values. Border matting starts with a contour C, obtained by fitting the hard segmentation from the previous section to the segmentation boundary. A new trimap is computed where T_U is a window of width $\pm w$ around C.

An index $t(n)$ is assigned to each pixel $n \in T_U$ and the α-profile is calculated as a step-function:

$$\alpha_n = g(r_n; \Delta_{t(n)}, \sigma_{t(n)}) \tag{4.12}$$

where r_n is the distance from the pixel n to the contour C. Parameters Δ and σ represent the centre and width of the transition from 0 to 1 in the α-profile. These values are estimated by minimising the following energy function over t:

$$E = \sum_{n \in T_U} \hat{D_n}(\alpha_n) + \sum_{t=1}^{T} \hat{V}(\Delta_t, \sigma_t, \Delta_{t+1}, \sigma_{t+1}) \tag{4.13}$$

where $\hat{V}$ is a smoothing regulizer:

$$\hat{V}(\Delta, \sigma, \Delta', \sigma') = \lambda_1(\Delta - \Delta')^2 + \Delta_2(\sigma - \sigma')^2 \tag{4.14}$$

The purpose of this regulizer is to let α-values to vary smoothly as t increases, along the contour[32].

4.2 Implementation

In order to implement the algorithm in Python, the OpenCV-Python library was used. This library was developed to solve computer vision problems and supports a wide variety of algorithms, such as the Canny Edge Detection and the Watershed Segmentation Algorithm, along with tools for image processing and manipulation[33].

The GrabCut algorithm works by accepting an image and one of two user inputs: a bounding box with the desired foreground object inside or a mask with the approximated segmentation. Then it iteratively goes through the GMM estimation, the Markov field and the Graph Cut optimization. It receives the following parameters:

- *img*: The input 8-bit, 3-channel image
- *mask*: The input/output mask that can be initialized with the bounding box or by the user, where the values 0,1,2 and 3 are used to represent sure background, sure foreground, probable background and probable foreground respectively. It should be a single channel 8-bit image
- *rect*: The bounding box that contains the foreground region to segment, only used if the mode is set to bounding box initialization (cv2.GC_INIT_WITH_RECT)
- *bgModel*: Array used to model the background
- *fgModel*: Array used to model the foreground
- *iterCount*: The number of iterations the algorithm will perform to generate the background and foreground models.
- *mode*: The initialization method, that is either with a bounding box (cv2.GC_INIT_WITH_RECT) or with a mask (cv2.GC_INIT_WITH_MASK)

The output consists of the modified *mask*, along with the *bgModel* and *fgModel*. The mask is marked with the same flags specified above (0,1,2 or 3) representing the background/foreground. Some adjustments are needed to perform the segmentation: all 0-pixels and 2-pixels are changed to 0 (ie. background pixels) and all 1-pixels and 3-pixels are changed to 1 (ie. foreground pixels). It can then be applied to the initial image to perform the segmentation. The results of the GrabCut algorithm are presented in Figure

Algorithm 1 OpenCV-Python GrabCut

```
1: grabCut(img, mask, rect, bgdModel, fgdModel, iterCount[, mode]) →
2: mask, bgdModel, fgdModel
3: segmented_img = mask * img
```

4.1. The segmentation worked very well, and the desired object (the football player) was full separated from the background.

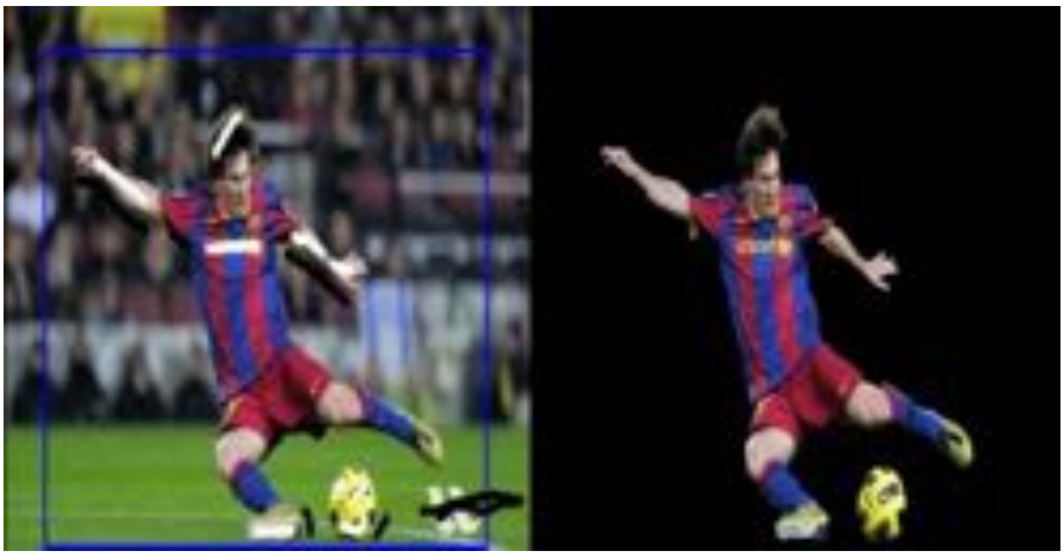


Figure 4.1: On the left is the input image with a bounding box in blue and then touch-ups in white (denoting foreground) and in black (denoting background). The result obtained is on the left.

4.3 Input

Even though the algorithm already has user input functionalities, they do not coincide with the input that will be used. Despite the fact that the algorithm accepts a mask, it needs to be generated in advance. At first, different forms of input were explored 4.2, such as the using polygons and free-hand drawing to label the background and foreground. Since the users will be able to give input through a tablet or in Virtual Reality, which allows for precise drawing, it was decided that a closed contour of the desired edge was the most appropriate for the problem at hand. A small algorithm was developed to solve this, presented below (2):

Algorithm 2 Create mask from closed contour

```
1: for every point in closed contour do
2:     add circle of 3-pixels (probable foreground) around point
3: end for
4: fill inside of generated contour with 1-pixels (sure foreground)
5: fill outside of generated contour with 0-pixels (sure background)
```

The output is a mask with three different pixel labels that can then be used with the OpenCV GrabCut algorithm. Represented in Figure 4.3, the red, yellow and green areas represent the sure background, the probable foreground and the sure foreground respectively. The width of the drawn contour can be adjusted in order to change the size of the yellow area (probable foreground). This allows for less specific inputs to be drawn, as long as the desired edge is inside the yellow zone.

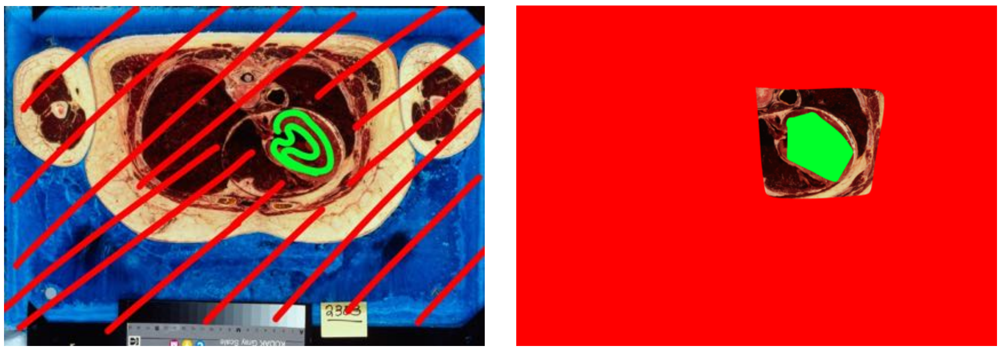
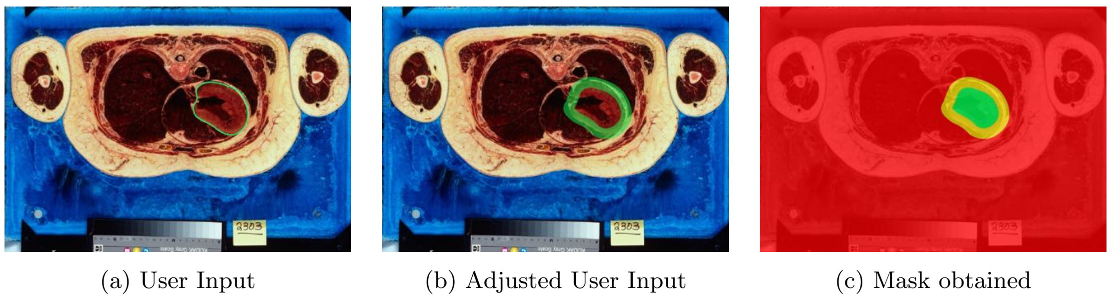


Figure 4.2: Examples of the first inputs explored for the algorithm 2 represented in (a) and (b). The object being contoured is the left ventricle of the heart of slice number 2292. The colors red and green represent sure background and sure foreground respectively.



(a) User Input (b) Adjusted User Input (c) Mask obtained

Figure 4.3: Illustration of the process in Algorithm 2 that starts with a 1-pixel contour in green (a), transforms it into a larger path in green (b) and the final mask obtained (c). In (c), the colors red, yellow and green represent sure background, probable foreground and sure foreground respectively. The object being contoured is the Left Ventricle of the Heart of slice number 2292.

4.4 Editing and Multiple Contours

After obtaining the mask, it may not have converged to the desired edge or contain the entirety of the foreground. Also, some structures might have multiple surfaces, such as hollow objects, where more than one edge is present. As such, some editing tools were added to the algorithm.

Firstly, the ability to directly edit the mask was added. This is done by using two input colors, where the user can draw over the undesired parts in black and unrecognized parts in white. A mask is then created from the input, where white points correspond to sure foreground pixels, and black points to sure background pixels. This mask is used in GrabCut and a new image is produced. The cycle can be repeated until the user is satisfied with the output. This process is represented in Algorithm 3. The illustration of the algorithm can be seen in Figure 4.4, where the left ventricle of the heart was not fully labeled as sure foreground by Grabcut. With the new editing tool, the user can now remove unwanted pixels and make sure the entire object is labeled as foreground.

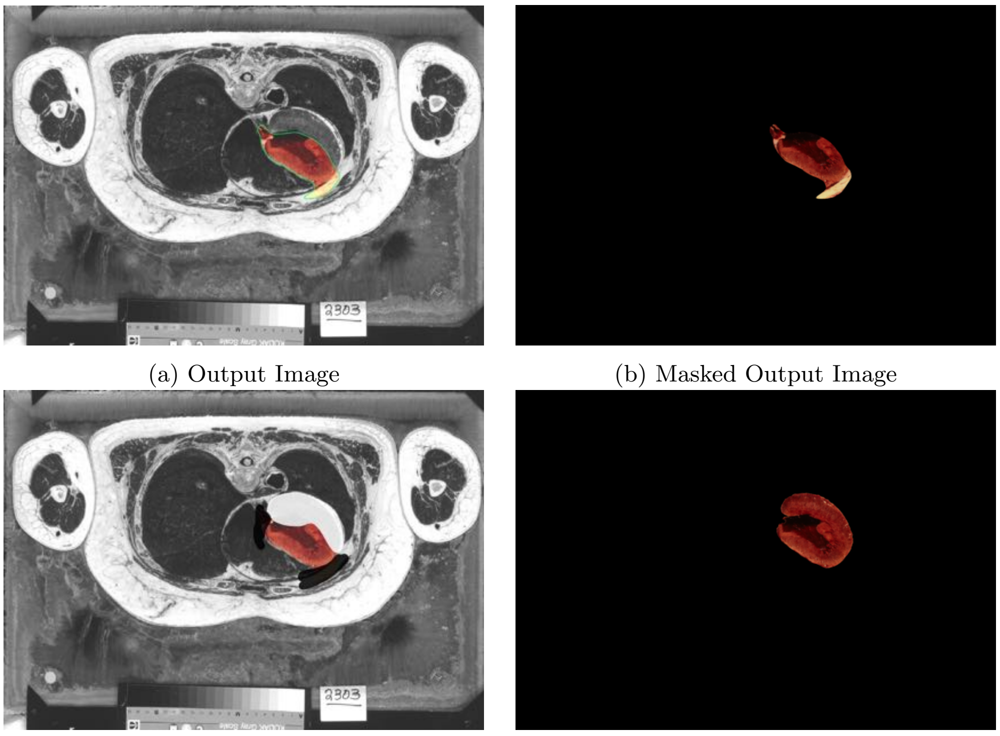
Algorithm 3 Edit mask

```
1: while editing do
2:     open input window
3:     edit_mask[0 where black input]
4:     edit_mask[1 where white input]
5:     grabCut(segmented_img, edit_mask, bgdModel, fgdModel, iterCount, mode)
6: end while
```

Secondly, the ability to add multiple contours was implemented. In order to accomplish this, part of the code was redone to save more than one contour instead. When the user wants to add a new surface, a new window is opened and the process is the same. The user draws the new contour, the algorithm runs and a new mask is computed. Now, the user specifies whether the new object is to be subtracted or added to the original mask. For example, in order to build a hollow bone, the user first draws the entire bone and then the bone marrow, in subtraction mode, as seen in Figure 4.5. On the other hand, if the object the user is contouring splits into two, for example, the user can contour the new structure, in addition mode, represented in Figure 4.6.

Algorithm 4 Adding multiple surfaces

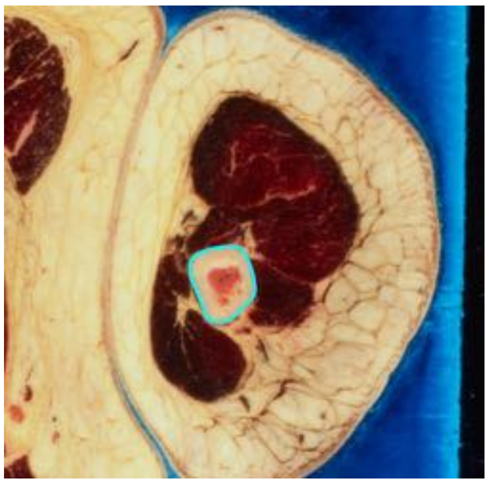
```
1: open input window
2: grabCut(img, new_mask, bgdModel, fgdModel, iterCount, mode)
3: if mode == add then
4:     add(mask, new_mask)
5: else
6:     subtract(mask, new_mask)
7: end if
```

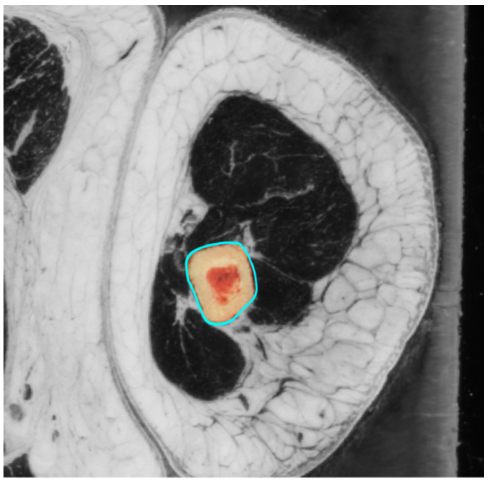


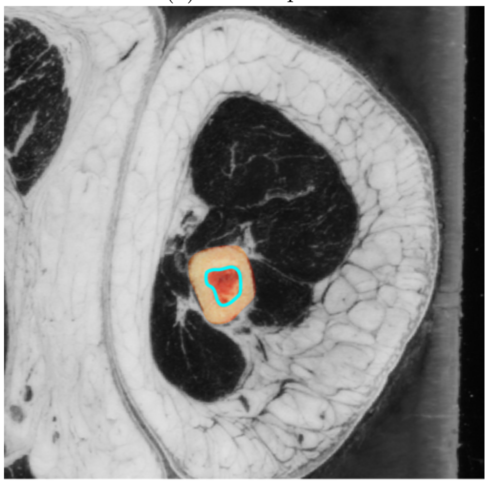
(a) Output Image (b) Masked Output Image

(c) Input (d) Result

Figure 4.4: Illustration of the process in Algorithm 3 that starts with a sub-optimal result, where only part of the left ventricle of the heart in slice 2292 was masked (a)(b). The editing process receives as input white and black strokes (c), that stand for sure foreground and sure background respectively. The input is then processes and the mask is adjusted (d).


(a) First Input


(b) First Segmentation


(c) Second Input

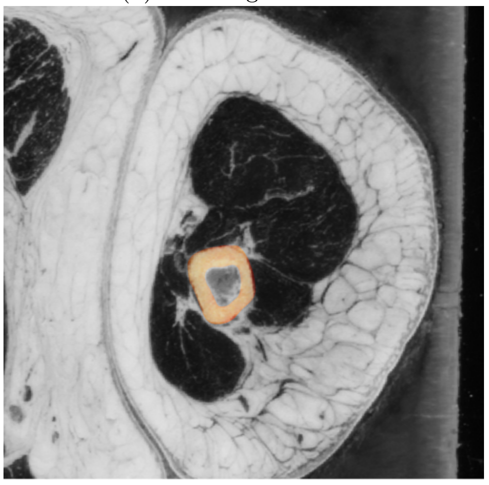
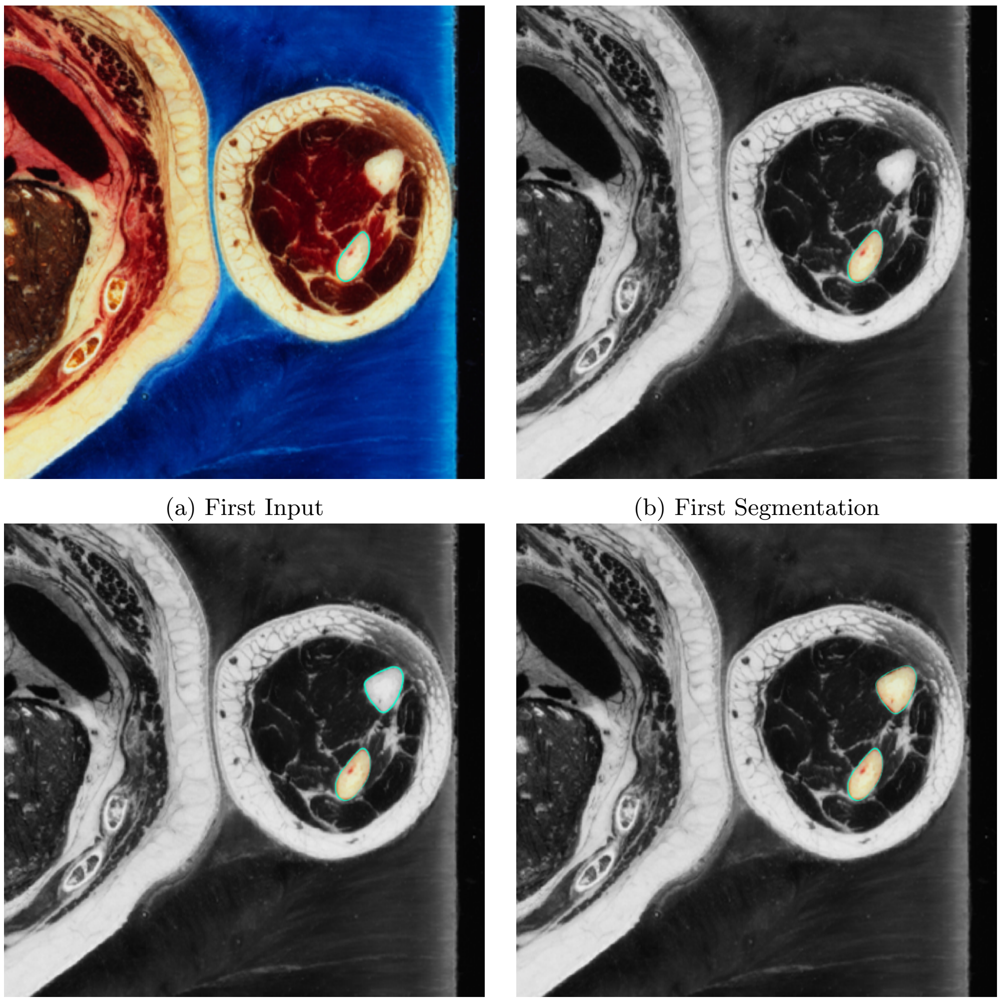

(d) Second Segmentation

Figure 4.5: Illustration of the process in Algorithm 4 of incorporating multiple contours, using subtraction. The objective is to segment the cortical bone of the humerus in slice number 2132. Firstly, the entire bone is contoured by the user (a) and then segmented by the algorithm (b). Then, the internal cavity is contoured (c), the algorithm computes it and the internal cavity mask is subtracted to the entire bone (d). The user input is in blue, the computed foreground is in color and the computed background in black and white.


(a) First Input (b) First Segmentation

(c) Second Input (d) Second Segmentation

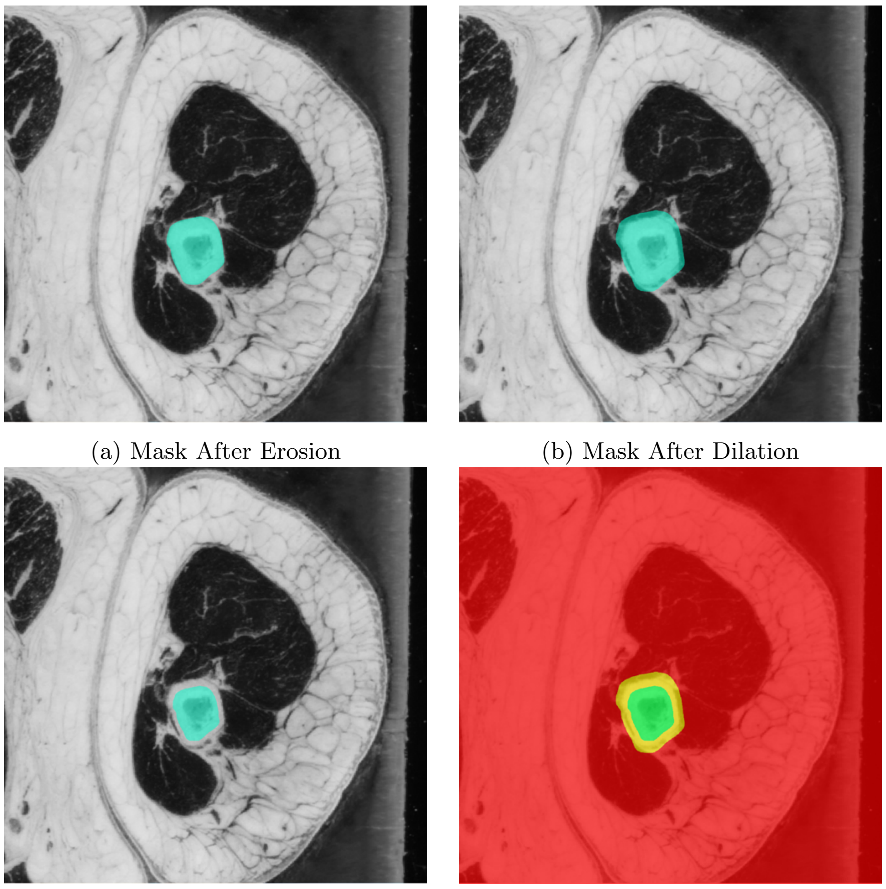
Figure 4.6: Illustration of the process in Algorithm 4 of incorporating multiple contours, using addition. The objective is to segment both ulna and radius of the left arm of slice number 2729. Firstly, the radius is contoured by the user (a) and then segmented by the algorithm (b). Then, the ulna is contoured (c), the algorithm computes it and both masks are added (d). The user input is in blue, the computed foreground is in color and the computed background in black and white.

4.5 Propagation

The main purpose of the algorithm is to reconstruct the entire object from a single contour. In order to do this, the algorithm is applied to an interval of slices, using the output from a slice and converting it into input for the next. This process consists of creating a dilated mask and a eroded mask. The area between the dilated and eroded masks is the read as probable foreground, the eroded mask is read as sure foreground and anything outside the dilated mask is sure background, as seen in Figure 4.7.

Algorithm 5 Propagate Contour across an interval of slices

```
1: input: mask, slice_interval
2: initiate bgdModel and fgdModel
3: for slice in slice_interval do
4:     dilated_mask = dilate(mask, threshold)
5:     eroded_mask = erode(mask, threshold)
6:     new_mask = np.zeros
7:     new_mask[dilated_mask == 1] = 3
8:     new_mask[eroded_mask == 1] = 1
9:     grabCut(img, new_mask, bgdModel, fgdModel, iterCount, mode)
10: end for
```



(a) Mask After Erosion

(b) Mask After Dilation

(c) Original Mask

(d) Propagation Input

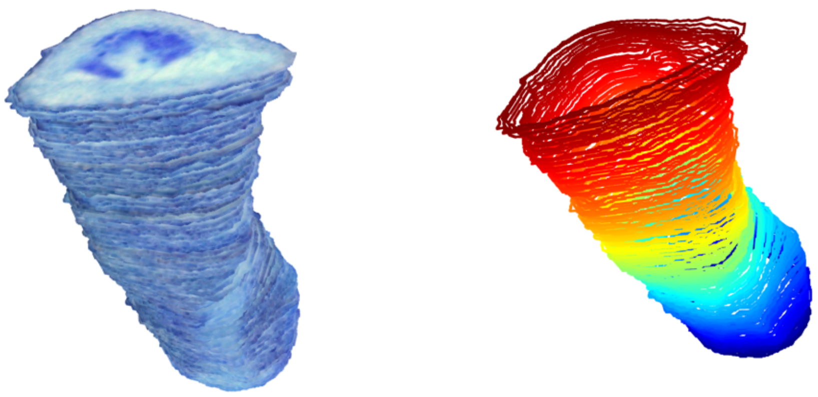
Figure 4.7: Illustration of the process in Algorithm 5 of propagating a contour to the adjacent slices. First, the original mask (a) is dilated (b) and then eroded (c) using a user defined threshold. Then, using these two masks, the input for propagation is created (d). In (a), (b) and (c), the blue area represents the mask, and in (d), the red, yellow and green areas represent, respectively, sure background, probable foreground and sure foreground.

4.6 Output

In an attempt to visualize the reconstructed object, the contours were saved. Even though the reconstruction will be done later using the triangulation method (when implemented in the server), during development, the contours were saved in point clouds to create the object in 3D. This was done using the Open3D Python library, which creates files in the Polygon File Format. This format saves a description of the object as a list of flat polygons. Other properties such as color and transparency can also be stored. This was used to create two point clouds, one with the contours and another one with the whole object, as seen in Figure 4.8. This makes it easier to evaluate the efficacy and performance of the algorithm.

Algorithm 6 Use output

```
1: zz = 0
2: for slice in slice_interval do
3:     grabCut(img, new_mask, bgdModel, fgdModel, iterCount, mode) →
4:     mask, bgdModel, fgdModel
5:     countours = extract_contours(mask)
6:     save_contour(contourss, zz)
7:     zz += slice_thickness
8: end for
```



(a) 3D Foreground Reconstruction (b) 3D Contour Reconstruction

Figure 4.8: 3D Reconstruction of the generated contours. The example used is a set of contours generated from the Left Humerus from slices 2000 to 2400.

4.7 Improving Segmentation

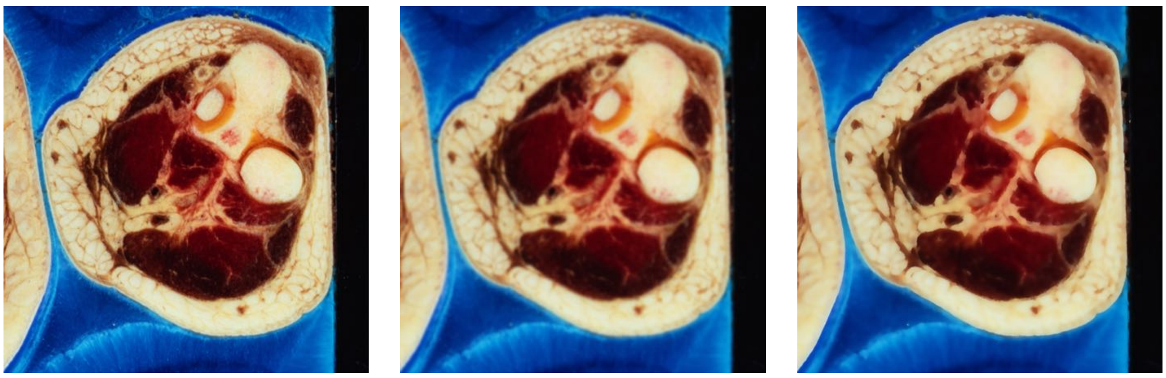
The next step is to enhance the segmentation of the algorithm. One of the most visible problems is that the contours generated are rough. Also, the time it takes to compute the propagation of the contour along the slice interval is quite long and has room for improvement.

4.7.1 Preprocessing

The first obvious step in preprocessing is to reduce the noise in the original image. There are a wide variety of filters that can achieve this: convolving the image using a normalized box filter, Median Blur, bilateral filtering and Gaussian Blur. Each of them are good at removing a special type of noise, and some are faster than others. The most promising filters are Bilateral and Gaussian Blur, which remove gaussian noise. The main difference between the two is edge preservation, because both use Gaussian filtering. While the Gaussian Blur finds the weighted average of the neighbouring pixels and does not take into consideration if the pixel is and edge pixel or not, the Bilateral filter uses an additional multiplicative Gaussian filter applied in the intensity domain that ensures that only pixels with similar intensities are used to the blur the intensity value, therefore preserving the edges. This comparison is done below, in Figure 4.9.

The chosen filter was Bilateral, which enhanced the smoothness of the final contour obtained from the algorithm while keeping the edges mostly intact. The original image was convolved using the OpenCV Bilateral Filter.

When comparing the segmentation with and without the Blur the difference is slightly noticeable, the contours are now smoother and have less spikes.



(a) Original Image (b) Gaussian Blur (c) Bilateral Blur

Figure 4.9: Illustration of the differences between the blurring filters used.

4.7.2 Postprocessing

On the other hand, the post processing implemented was to reduce the number of points in the contour. The reason behind this was not only to reduce the memory needed to store all the contours but also to reduce the complexity of the contour and in turn make it smoother.

The simplest way to achieve this is to reduce the number of points by a factor and using the mean to compute the substitution. This can be done with a few lines of code, using numpy functionalities. The reshape function transforms the list into an array where the number of columns is the factor of reduction. Then, the mean function flattens the array back into a smaller list using the mean of each line. The results can be seen below, in Figure 4.10

Algorithm 7 Reducing number of points using the Mean

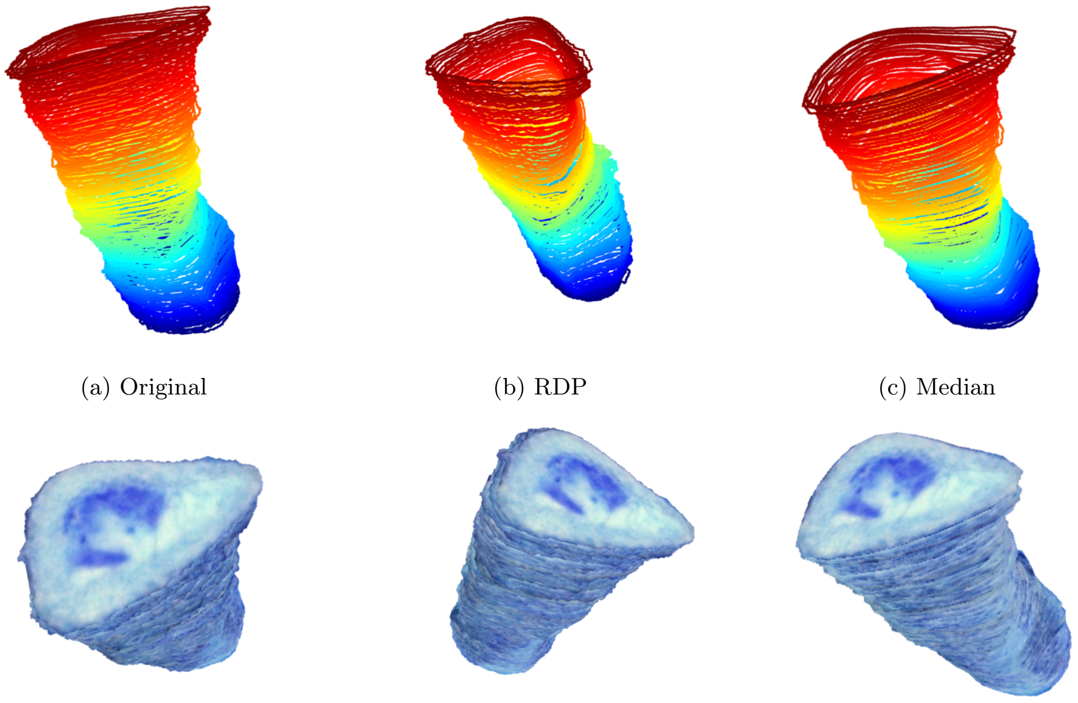
```
1: input: PointList[], factor
2: xx = PointList[x_coordinates]
3: yy = PointList[y_coordinates]
4: xx.reshape(-1, factor).mean(axis = 1)
5: yy.reshape(-1, factor).mean(axis = 1)
6: ResultList[] = combine(xx,yy)
7: return: ResultList[]
```

Algorithm 8 Implementeation of the Ramer–Douglas–Peucker algorithm

```
1: input: PointList[], $\epsilon$
2: dmax = 0
3: index = 0
4: end = length(PointList)
5: for i = 2 to (end - 1) do
6:     d = perpendicularDistance(PointList[i], Line(PointList[1], PointList[end]))
7:     if  d > dmax then
8:         index = i
9:         dmax = d
10:    end if
11: end for
12: ResultList[] = empty
13: if dmax > $\epsilon$ then
14:     recResults1[] = DouglasPeucker(PointList[1...index], $\epsilon$)
15:     recResults2[] = DouglasPeucker(PointList[index...end], $\epsilon$)
16:     ResultList[] = recResults1[1...length(recResults1) - 1], recResults2[1...length(recResults2)]
17: else
18:     ResultList[] = PointList[1], PointList[end]
19: end if
20: return: ResultList[]
```

Another way to achieve this is using the Ramer–Douglas–Peucker algorithm developed for cartographic generalization. Its main purpose is to recursively reduce the number of points of a polyline using the maximum distance between the original curve and the simplified one as criteria. The first and last point are kept, and then it finds the farthest point from the line between the start and end. If the distance is smaller than epsilon, then any points that aren't marked can be discarded. On the other hand, if the distance is bigger, the point is kept. The algorithm then repeats with the first point and farthest point, and the farthest point and the end point. The result is a simplified line consisting only of points that were marked. To use this algorithm in Python, the shapely library was used. The results can be seen in Figure 4.10

The chosen method was the simpler version, Algorithm 7, as using the Ramer–Douglas–Peucker algorithm caused a significant increase in the computation time, not compensating the improved output.



(a) Original (b) RDP (c) Median

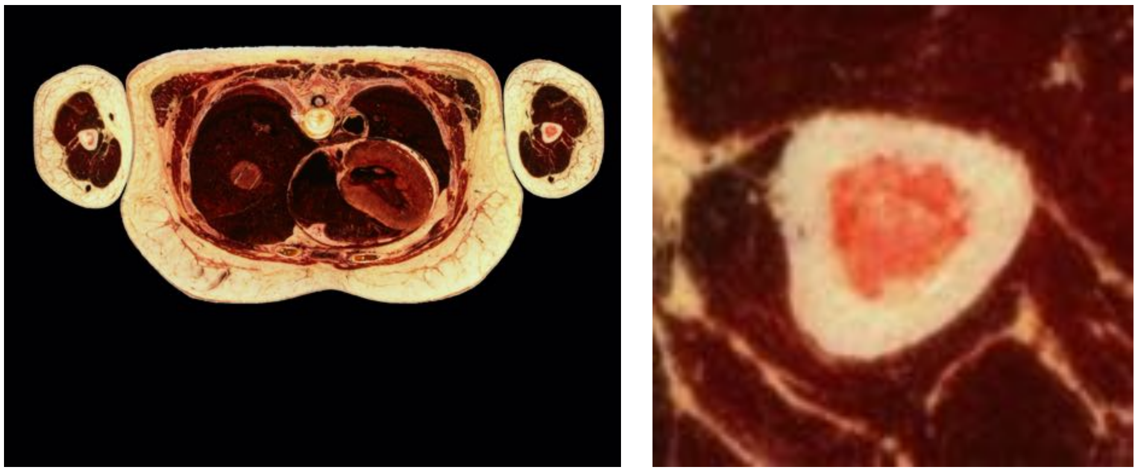
(d) Original (e) RDP (f) Median

Figure 4.10: Illustration of the differences between the blurring filters used.

4.7.3 Efficiency

After the algorithm was more robust and was producing good results, there was still one problem at hand. The images in the data set have a large resolution and the algorithm takes a long time to compute. At this state, the algorithm would compute around 5 slices per minute.

One of the steps that takes a lot of time is using GrabCut. When propagating the contour, the entire image does not need to be read. The image can be cut around the contour and only include the area where the mask has foreground pixels (Figure 4.11) by defining a region of interest. The only thing to take into account is to make sure that when propagated, the window always contains the necessary information. A new window is computed with every cycle using the limits of the contour. This managed to drastically reduce the time it takes to perform a full segmentation of small structures. However, if the tissue/organ being segmented covers a big part of the slice, then the time reduction is less noticeable.



(a) Original (b) Computed Window

Figure 4.11: Comparison of the original images being computed and the cropped image when segmenting the humerus. The original image is 4096 by 2700 pixels whereas the cropped image is 265 by 250 pixels.

4.7.4 Parallel Programming

There are different techniques that make better use of the CPU cores. Python has some tools that allow the use of parallel programming - multi-threading and multiprocessing. However, they are very different and have different areas of application.

Threads are a unit of execution within a process. Multiple threads can run inside a process and they use and share its memory space, which makes it easier, faster and safer to share data. Python has a Global Interpreter Lock that prevents threads from running simultaneously, in order to prevent racing, where the system's behaviour is dependent on the sequence and timing of events. This is where multi-threading comes into play. Multi-Threading allows for multiple threads to do different tasks simultaneously, one after the other. This gives the illusion that the program is running in parallel when it is really running concurrently. This is useful for intensive I/O tasks, that rely on input from the user or from another system. If the system is experiencing a bottleneck not in execution, but in a third-party call, this method manages to speed-up the running time. The algorithm at hand does not have any bottlenecks related to I/O tasks or GUI yet, so this might not be the approach to be taken.

On the other hand, processes have their own memory space and are loaded along with all the resources they need to function. They are independent and run in a processor core, and are scheduled by the operating system. True parallelism can only be achieved using multiprocessing. This implies that two or more jobs are being executed simultaneously. This technique makes use of multiple CPU cores, that do not share resources among them. Each of this processes have their own threads running in the memory space and their own instance of Python interpreter executing the instructions. Independent operations can be divided into separate tasks as long as their data is different. The number of processes running in parallel should be the same as the amount of cores in the system. The Visual Human consists of massive amounts of data that is loaded in memory and processed. Multiprocessing will considerably speed-up the algorithm and prevent bottlenecks in time and resources.

There are different approaches that can be taken to implement multiprocessing in the algorithm. When propagating the contour through the slices, the interval can be split into different processes. The three methods used were:

Alternative A: first, the algorithm runs through the interval with a step of 30 slices and then the spaces between are computed by the processes

Alternative B: one process works on the even slices and the other on the odd ones

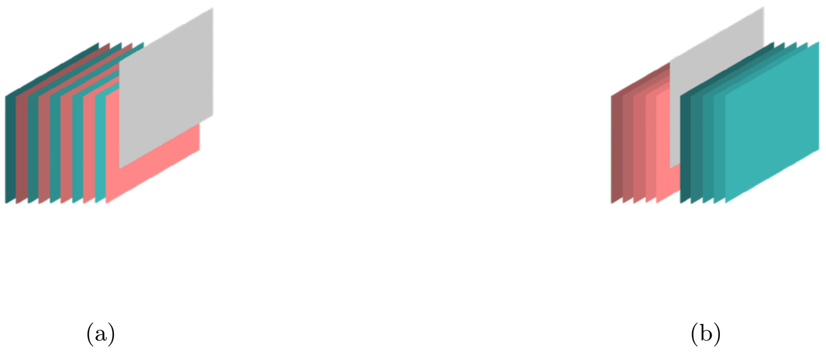
Alternative C: one process for each direction of propagation

Each of these methods have their downsides. Since the propagation relies on moving from slice to slice, using a system of blocks can lead to discontinuities between them. The algorithm may not converge the same way in each block.

This does not happen when alternating slices in the processes. Even though the processes run completely independent of one another, since they run through similar sets of slices, they tend to converge the same way. The only downside of this method is that it can only be used if the interval step is of one slice. The bigger the step, bigger disparity between the sets of slices and a higher risk of the processes converging in different ways.

Lastly, using one process for each direction of propagation does not encounter problems of convergence. The two processes work on independent sets of slices and the data between the two does not cross. The only disadvantage is that this only allows the user to reconstruct an object from the middle outwards.

The method implemented was Alternative B, since it had good results and works in more scenarios, not being limited to two-way propagation. After implementing parallel programming into the algorithm, the running time went down by almost a half. For example, when contouring the humerus, before parallel programming, the process took 260 seconds, whereas after implementation it was reduced to 140 seconds.



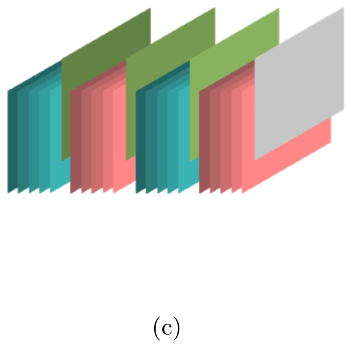


Figure 4.12: Illustration of the different approaches of multiprocessing considered. Alternatives A, B and C are represented in (a), (b), and (c) respectively. The grey slice represents the user input, and red, green and blue represent slices computed by different processes.

4.7.5 Parameters

The last thing to be taken into consideration is the parameters of the algorithm. So far, there are 6 parameters:

width: the width of the input contour, used in Algorithm 2

step: used in propagation, defines the jump between slices

window: defines the distance between the edge of the mask and the window created in propagation

threshold: threshold is used in Algorithm 5 and dictates how the area to compute enlarges between propagation cycles

gaussian: the size of the kernel when applying the Gaussian filter

smoothing: defines the parameter for the smoothing algorithm

iterations: number of iterations of GrabCut in each cycle

Each of these parameters affect the algorithm in their own way.

Lower width is used when the input contour is similar to the desired edge, leaving less area for the algorithm to compute. Naturally, this reduces computation time, but is only used on the first cycle. A larger width is useful for less accurate input but might in turn not converge to the desired edge.

The step size affects the propagation and dictates how many slices are skipped. Naturally, a high step size drastically decreases running time, however, this only works if the structure being contoured is homogeneous and does not change too much between slices.

The window also affects the propagation as it defines the area of the image that is read. A smaller window means less running time but if the structure increases a lot in size between slices then the window might not contain it entirely.

The threshold is the most important parameter for the propagation to work. It cannot be too low or the algorithm will not be able to track the structure through the slices, but at the same time it cannot be too high or the contour will diverge into other structures. The optimal interval for the threshold is between 5-50 pixels.

The number of iterations does not affect the algorithm too much, as by the end the propagation the color models are well defined. However, the number should not be high otherwise it will increase the running time. The number of iterations used varies from 3-7.

Lastly, the Gaussian and Smoothing parameters do not affect the running time, but they do affect the output. The Bilateral filter was optimal at a gaussian kernel size of around 15 pixels, while the smoothing depends on the structure itself and what the user wants to obtain.

There are too many parameters for the user to calibrate. Width is defined at the moment of input, and so is step and smoothing. These two should be easily changed as they rely on the users' intention. As for the rest, levels were defined in order to make the process easier and more user-friendly. They are defined below:

level	window	threshold	iterations
1	20	20	3
2	30	35	5
3	50	50	7

Table 4.1: Parameters for each of the defined levels.

The levels stand for different sensitivities which directly affect the running time of the algorithm. The lower levels have a smaller threshold and are less sensitive to changes in structure, while the higher levels have a bigger threshold in order to track rapidly changing structures. The difference between each level can be seen in Figure 4.13. Each level has a different computation time. For the reconstructions in Figure 4.13, the first level took 140 seconds, the second level took 175 seconds and level three 227 seconds. This change is mainly due to the different window used, bigger window means more data to read. The biggest changes seen in the 3D models are due to the different threshold used. Larger threshold causes the algorithm to label more pixels as probable foreground which in turn affects the color models generated. Naturally, there is more risk of pixels from neighbor tissues being carried into the segmentation.

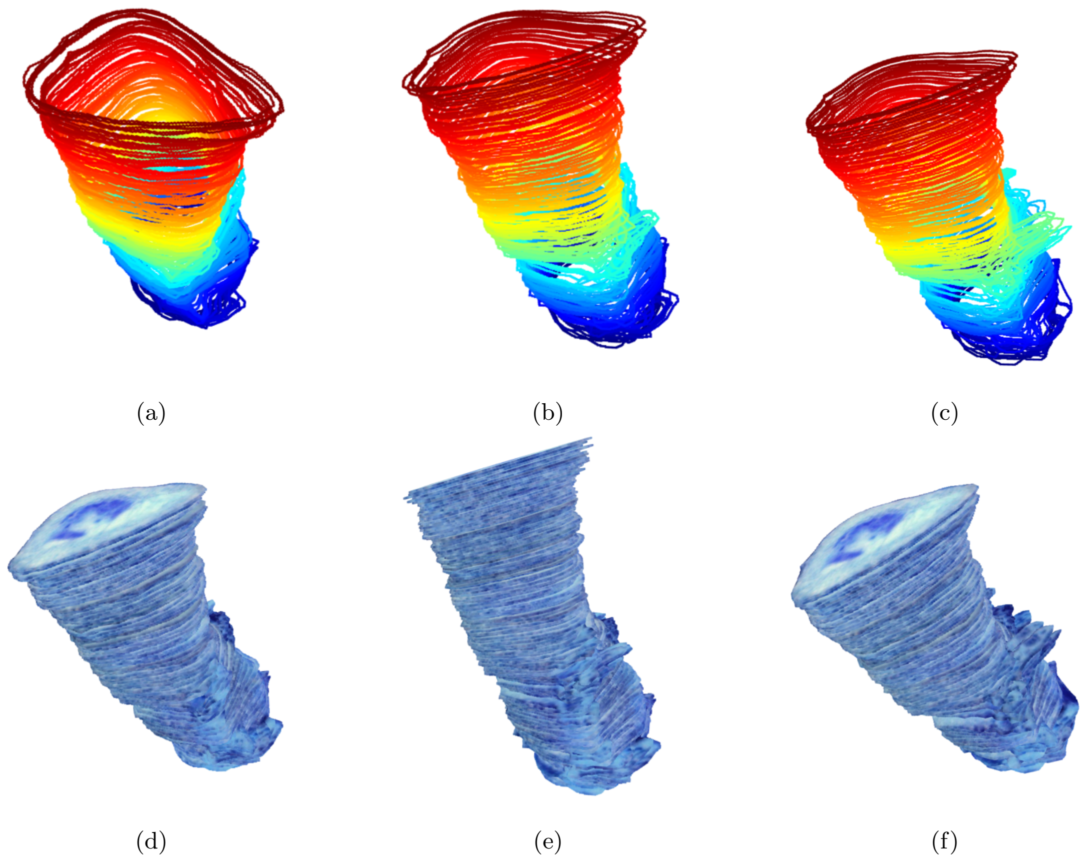


Figure 4.13: 3D reconstructions of part of the femur with different levels of parameters. In the first row are the reconstructed contours and on the second row are the 3D masked slices. In (a) and (d) the level 1 was used, in (b) and (e) level 2, and in (c) and (f) level 3.

4.8 Experimental Results

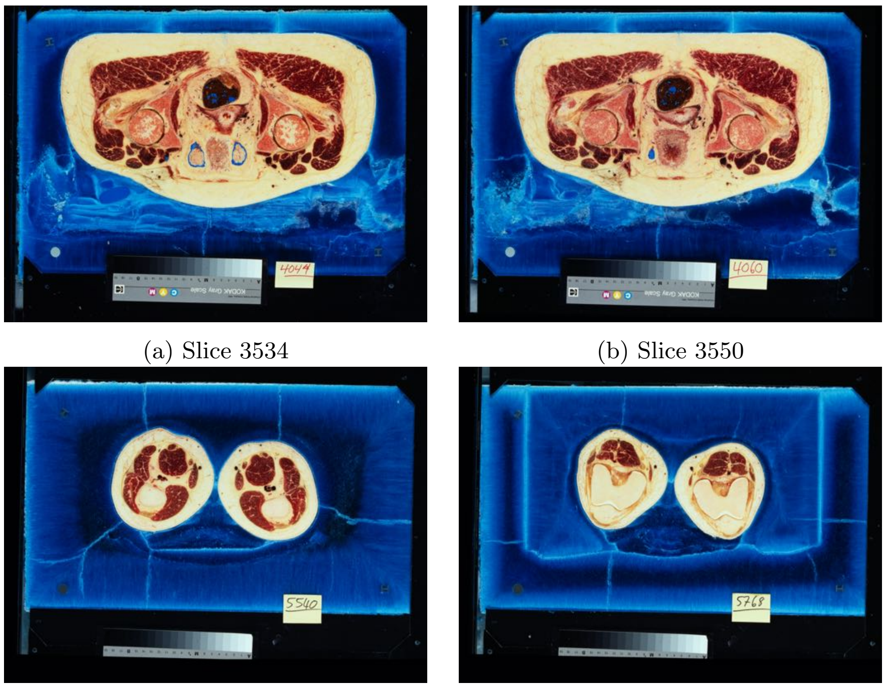
In order to evaluate the results, segmentation of different structures was attempted.

First of all, the whole data set was segmented in order to the remove the ice background (Figure 2.5, as mentioned in chapter 2). The whole process took some time to compute (around 6 hours), since the full resolution (4096 x 2700 pixels) images needed to be processed. The whole body was segmented with only one input on slice 1023. However, some body parts were missing due to the discontinuities on the data set, which were then segmented and incorporated.

Secondly, a complete segmentation of the femur was achieved (Figure 4.15). The femur spans across an interval of 1248 slices, from slice 3477 to slice 4725. A total of 9 inputs were provided during the process, at slices 3477, 3481, 3557, 3571, 3601, 3669, 3839, 4449 and 4577. The whole process took around 12 minutes to compute. The most problematic areas, which required additional user input, were the extremities, and the break on the data set between slices 3924 and 3942. An additional contour was also added in slice 3550, as both the femur's head and greater trochanter were separated in some slices. As seen in Figure 4.14, the knee area is hard to segment due to the lack of clear borders between the bone and the surrounding tissue.

Different levels of parameters were used. For the shaft, level 1 was used, since the structure is cylindrical, does not change drastically across the slices, and to prevent propagation to the brachial fascia, the membranous sheath of the arms' muscles. As for the proximal and distal epiphysis, level 2 was used in order to better track the variation in form.

Part of the aorta was also segmented, but since it is very homogeneous, it was successfully segmented in one go. The interval of slices computed was from slice to slice 2035 to slice 2650 and it took around 160 seconds to achieve the final result, which can be seen in Figure 4.16.



(a) Slice 3534 (b) Slice 3550

(c) Slice 4473 (d) Slice 4701

Figure 4.14: Some of the slices that required additional input in order to completely segment the femur. Images (a), (b), (c), and (d) correspond, respectively, to the femurs' proximal epiphysis, head and greater trochanter, shaft and distal epiphysis.

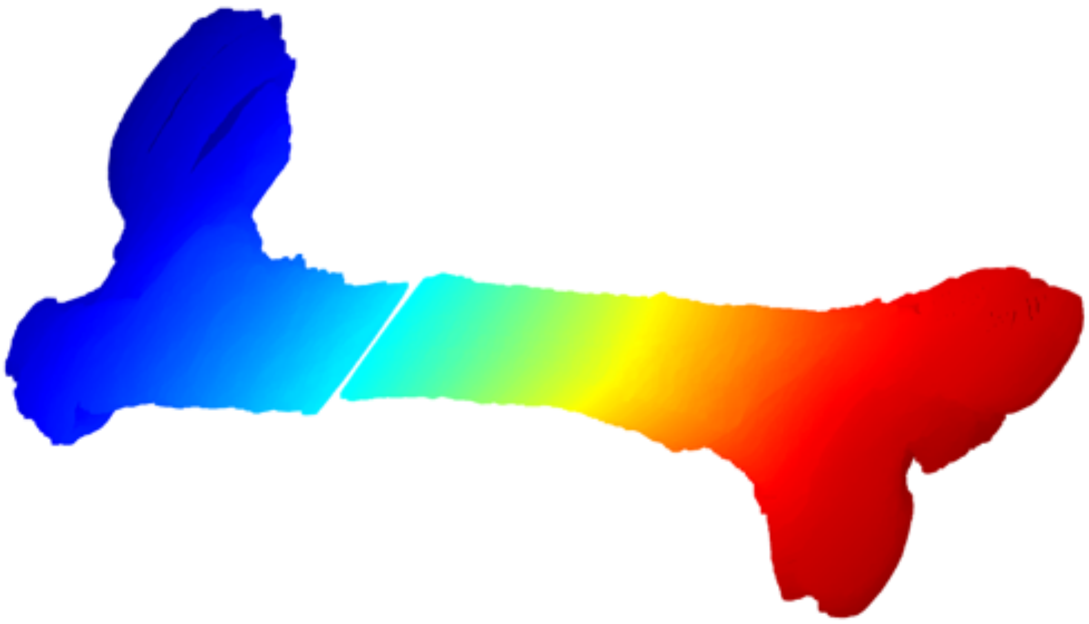


Figure 4.15: Side view of the reconstruction of the femur using the developed algorithm. The discontinuity in the bone is due to the damaged slices present in the data set that were ignored.

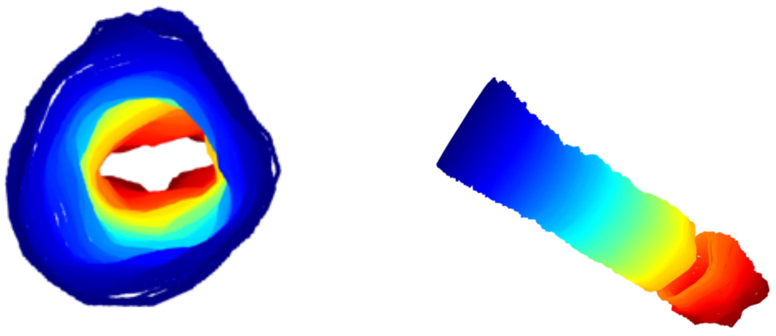

(a) (b)

Figure 4.16: 3D reconstruction of part of the aorta. The discontinuity in the structure is due to the damaged slices present in the data set that were ignored. The top view is represented in (a) and a side view in (b).

Lastly, a partial reconstruction of the liver was made from a single input. This structure is more heterogeneous and has multiple arteries and veins connected. The algorithm ran for 13 minutes in order to propagate the contour through 400 slices (from slice 2400 to slice 2800) with a window of around 1 500 by 1 000 pixels. The final 3D model can be seen in Figure 4.17.

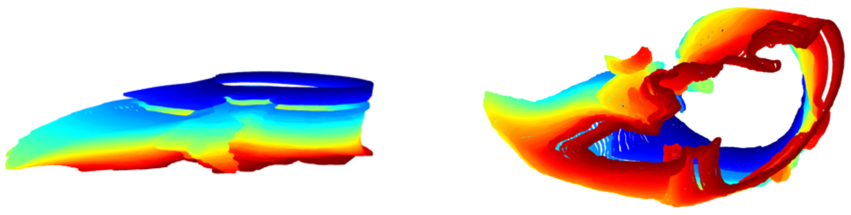
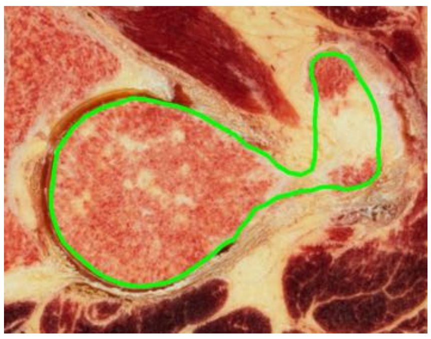

(a) (b)

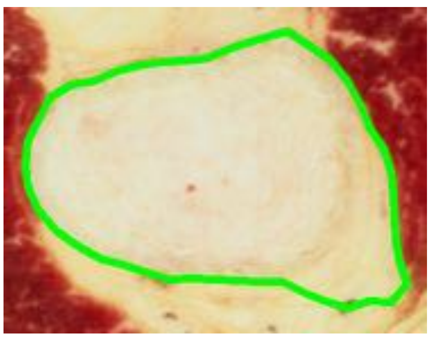
Figure 4.17: 3D reconstruction of part of the liver. The discontinuity in the structure is due to the damaged slices present in the data set that were ignored. The side view is represented in (a) and a top view in (b).

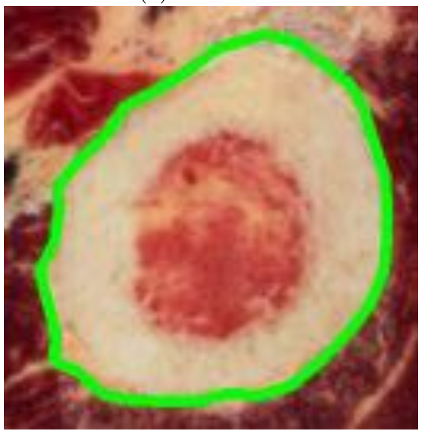
4.9 Discussion

Overall, the algorithm performs well when segmenting structures that are well defined and very different in color from the surrounding tissue, as expected.

As seen in Figure 4.15, the femur was fully reconstructed and the result is quite good. The bone is identifiable and both epiphysis are complete. The running time was adequate and the number of inputs was minimal. However, the contours could still be perfected. In some areas of the bone, where the color differentiation from the surrounding tissue is not big enough, which caused unwanted pixels to be present in the segmentation. As seen in Figure 4.18, in slices 3550 and 4701, the contour included part of the surrounding tissue, and in slice 3534 the computed contour is ambiguous.


(a) Slice 3534


(b) Slice 3550


(c) Slice 4473

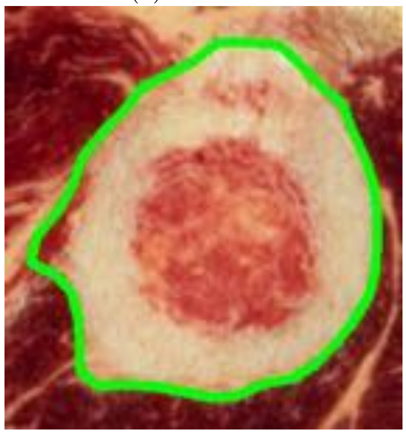

(d) Slice 4701

Figure 4.18: Propagated contours in green from the femur reconstruction at slices 3534, 3550, 4473 and 4701.

As for the aorta, the computed time was around 160 seconds for an interval of 630 slices, which is around 4 slices per second. This is possible due to the aorta being small in size, close to 200 by 200 pixels. The performed segmentation converged and managed to track the aorta with only one input, which is to be expected. However, the generated contours did not fully adapt to the rapid changes in the aortic wall, as seen in Figure 4.19.

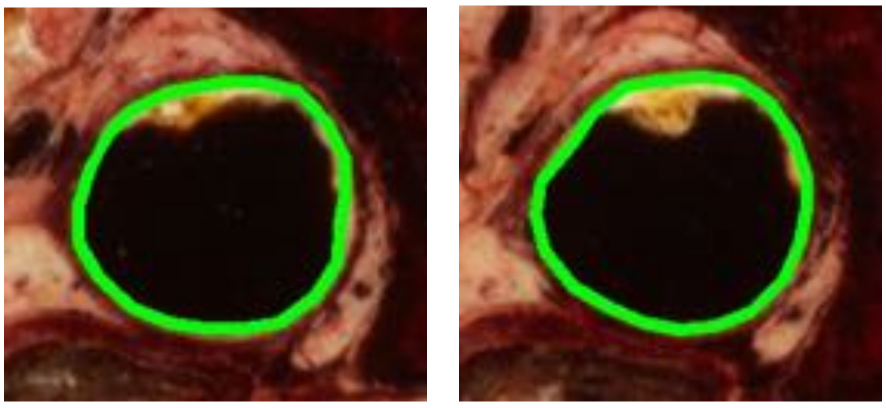

(a) Slice 2323 (b) Slice 2339

Figure 4.19: Propagated contours in green from the aorta reconstruction at slices 2323 and 2339.

Last of all, the liver reconstruction looks promising, since the algorithm managed to contour most of the liver's ramifications from only one input, at the first slice. However, the model is sub-optimal, some veins were not included, as seen in Figure 4.20, and sections of the liver were left out. The liver itself is a difficult structure to contour automatically as it changes a lot from slice to slice, so one solution would be to provide more input across slices. It is also important to note that the propagation took 13 minutes to compute, around two seconds per slice, which is substantially longer than other structures.

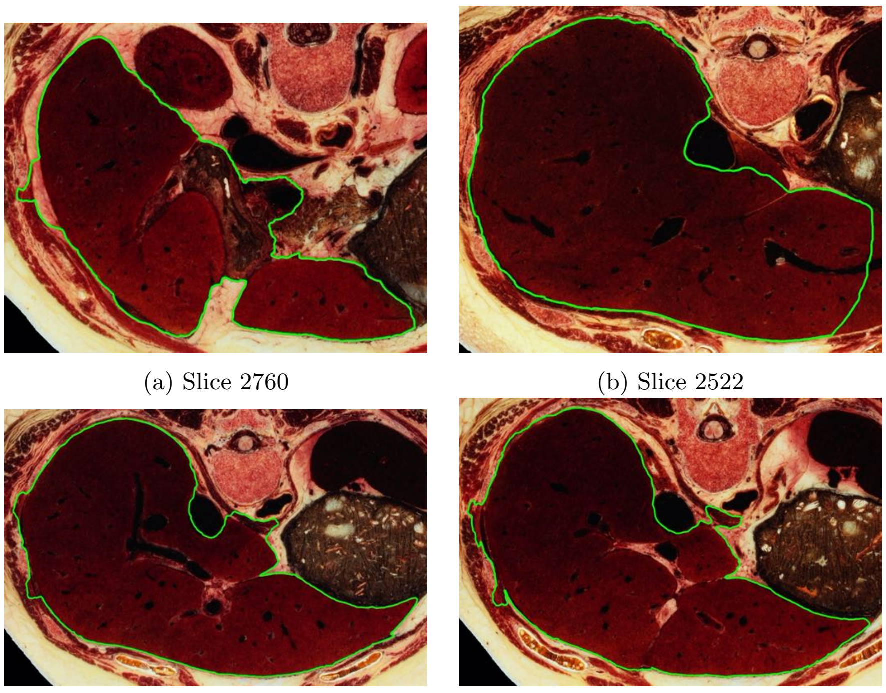

(a) Slice 2760 (b) Slice 2522
(c) Slice 2608 (d) Slice 2658

Figure 4.20: Propagated contours in green from the liver reconstruction at slices 2760, 2522, 2608 and 2658.

CHAPTER

5 CONCLUSIONS

The availability of reliable 3D visualization and segmentation tools is currently a major goal in the medical field, not only for surgical planning but also for teaching anatomy. With the rise of anatomical data sets of cadavers, more information has been available for researchers to investigate and use.

In this dissertation, current existing segmentation platforms were evaluated. While there is a wide variety of tools already available, most of them are focused on black and white radiological scans, and are quite time consuming.

In order to create a new algorithm capable of segmenting cryosections of the Visible Human Project Female data, a study of edge contouring methods was made, such as the canny edge detector, localized histograms and region merging and gradient and graph cut based algorithms. After careful evaluation of the methods, the latter was chosen as it was best suited to segment images of color with the desired user input.

Python code was written to create a semi-automatic segmentation algorithm, which was built on top of the OpenCV GrabCut. First of all, a method was developed to transform user drawn contours into a mask to be received by GrabCut, along with tools to edit the generated image. Moreover, the ability to add multiple contours was integrated in order to allow reconstruction of structures with multiple surfaces or ramifications. Secondly, a function was written to convert the output of GrabCut as input for and the adjacent slices, so that the contour can be propagated. Lastly, a small script was created to reconstruct the contours and resulting structure in three dimensions in order to provide the needed visualization of the model created.

After the reconstruction was done, the segmentation obtained took a long time to compute and the results were not optimal. Therefore, parallel programming was implemented to speed up the computation, and only the area containing the structure was read from each slice. A Bilateral filter was also applied to images before processing to reduce noise and to smooth unwanted color differences on tissues. Lastly, the contours obtained went through a smoothing process.

The final algorithm was implemented, which was capable of fully removing the background from the whole set of slices. Lastly, a full reconstruction of the femur, the aorta and part of the liver was made in order to evaluate performance.

5.1 Limitations and Future Work

Current limitations of the algorithm are tied to the lack of contrast between some tissues and the use of high resolution images. GrabCut relies on its color models to segment an image, when the segmented tissue and the background overlap in color-space, the propagation can diverge. This was observed in some structures, such as the lungs, where the segmentation would not converge after propagation.

One solution could be to use of the CT images and MRI for added contrast. MRI images are particularly useful for this, as there are three different MRI sequences available in the data set: T1-weighted, T2-weighted and Proton Density (PD) weighted. T1, T2 and PD are tissue characteristics and if it is difficult to distinguish two tissues with one characteristic, it is probable that that difference will be visible by using an image from another characteristic. While in PD Images the signal intensity is determined by proton content of the tissue, in T1 Images and T2 images it is determined by the longitudinal relaxation time and transverse relaxation time of the tissue. These differences are very noticeable on the brain, as seen in Figure 2.4. On the other hand, in CT scans, image contrast depends on the differential X-Ray attenuation of tissues due to absorption and scattering. Bone tissue and internal organs are easier to segment in these images.

This is can be done with Image Fusion, which is a process that combines two or more images. There has been a lot of research done on this method in medical imaging recently, using different techniques. These include convolutional neural networks (CNN)[34], stacked autoencoders [35], pixel-level fusion, convolutional sparse representation[36], and many others. The results from these methods look promising, and could be a solution to the segmentation problem.

The only setback when applying this to the algorithm is the fact that the scans were taken at different intervals, the cryosections were done with 0.33mm intervals, the CT scans with 1mm and the MRI scans with 4mm. Also, the axial MRI data is only available for the head and neck, however, these can be reconstructed for the rest of the body using the longitudinal scans.

Another way to improve segmentation quality would be to implement another edge segmentation algorithm. SnakeCut is a technique that integrates the output of both active contours and GrabCut and uses a probabilistic model to integrate both results[38]. Another technique that takes GrabCut one step further is the integration of multi-scale decomposition to extract features and integrate into the segmentation process [39]. However, these methods make the computation time longer, which is not ideal.

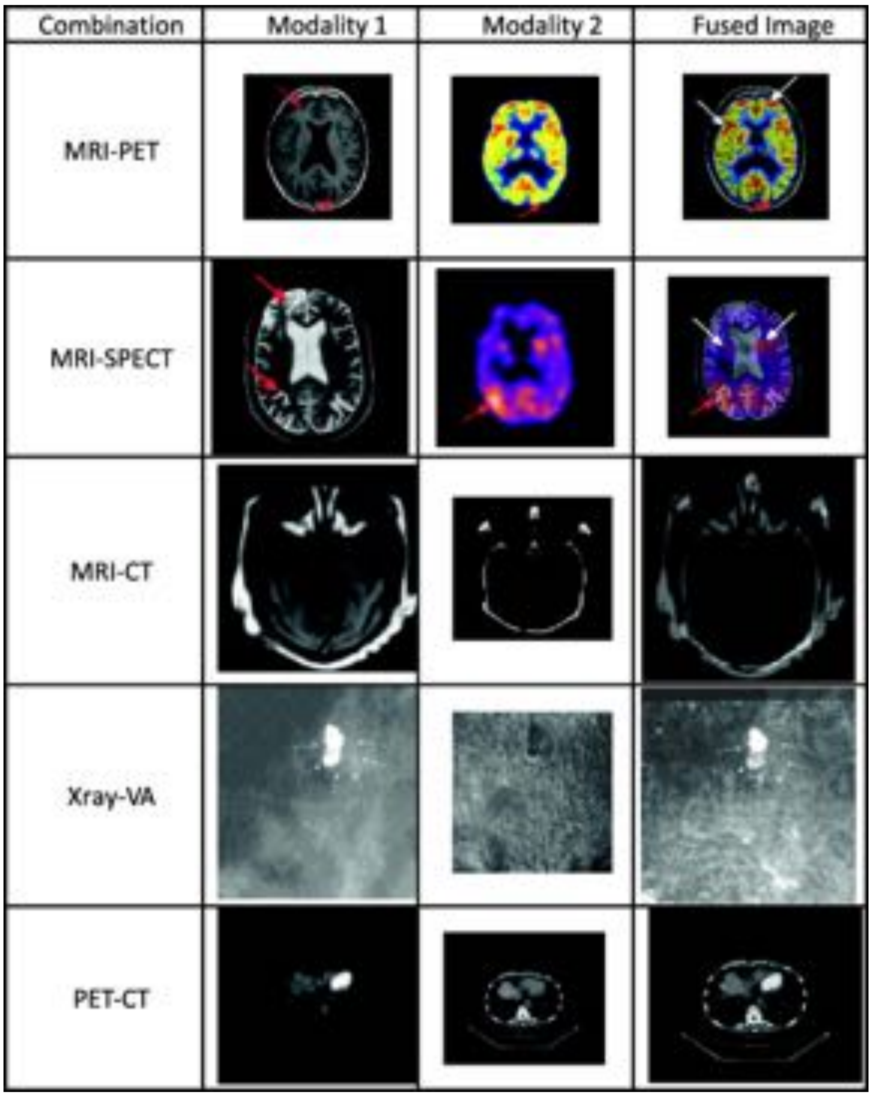


Figure 5.1: Examples of medical image fusion using different modalities resulting in improved visibility and contrast of tissues[37].

Additionally, the algorithm lacks sensitivity to small structures, such as veins and arteries that can be found on the data set. A second algorithm could be used as a separate tool for detailed segmentations. Also, if the data set used is always the same, then a deep learning machine could be implemented to learn features and improve segmentation.

Lastly, the computation time still has room to improve. Even though the algorithm is computing around 2 slices per second for small structures, when contouring large organs (such as muscles or skin) the velocity can decrease drastically. One solution could be a trade-off between quality and velocity, scaling down the images in order to speed up the process.

BIBLIOGRAPHY

[1] Ghosh Sanjib Kumar. ≪Human cadaveric dissection: a historical account from ancient Greece to the modern era≫. In: *ACB* 48.3 (2015), pp. 153–169. DOI: 10.5115/acb.2015.48.3.153. eprint: http://www.e-sciencecentral.org/articles/?scid=1071811. URL: http://www.e-sciencecentral.org/articles/?scid=1071811.

[2] Lawrence J. Rizzolo. ≪Human dissection: An approach to interweaving the traditional and humanistic goals of medical education≫. In: *The Anatomical Record* 269.6 (2002), pp. 242–248. DOI: https://doi.org/10.1002/ar.10188. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ar.10188. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ar.10188.

[3] Ghosh Sanjib Kumar. ≪Bodies for Anatomy Education in Medical Schools: An Overview of the Sources of Cadavers Worldwide≫. In: *Academic medicine : journal of the Association of American Medical Colleges* 93.9 (2018), pp. 1293–130. DOI: 10.1097/ACM.0000000000002227.

[4] R.McS. ≪Why there is a shortage of cadavers≫. In: *The Economist* (Jan. 19, 2014). URL: https://www.economist.com/the-economist-explains/2014/01/19/why-there-is-a-shortage-of-cadavers (visited on 10/10/2021).

[5] M.J. Ackerman. ≪The visible human project: a resource for education.≫ In: *Acad Med J Assoc Am Med Coll,* 74.6 (1999), pp. 667–670. DOI: 10.1097/00001888-199906000-00012.

[6] Jannat Falah et al. ≪Virtual Reality medical training system for anatomy education≫. In: *2014 Science and Information Conference*. 2014, pp. 752–758. DOI: 10.1109/SAI.2014.6918271.

[7] Michael Ackerman. ≪The visible human project®: From body to bits≫. In: vol. 2016. Aug. 2016, pp. 3338–3341. DOI: 10.1109/EMBC.2016.7591442.

[8] Paul A. Yushkevich, Yang Gao, and Guido Gerig. ≪ITK-SNAP: An interactive tool for semi-automatic segmentation of multi-modality biomedical images≫. In: *2016 38th Annual International*

Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). 2016, pp. 3342–3345. DOI: 10.1109/EMBC.2016.7591443.

[9] National Library of Medicine. *The Visible Human Project*. URL: https://www.nlm.nih.gov/research/visible/visible_human.html. (accessed: 05.10.2021).

[10] M Y Toh, R B Falk, and J S Main. «Interactive brain atlas with the Visible Human Project data: development methods and techniques.» In: *RadioGraphics* 16.5 (1996). PMID: 8888399, pp. 1201–1206. DOI: 10.1148/radiographics.16.5.8888399. eprint: https://doi.org/10.1148/radiographics.16.5.8888399. URL: https://doi.org/10.1148/radiographics.16.5.8888399.

[11] Andriy Fedorov et al. «3D Slicer as an image computing platform for the Quantitative Imaging Network». In: *Magnetic resonance imaging* 30.9 (2012), pp. 1323–1341.

[12] M.J. McAuliffe et al. «Medical Image Processing, Analysis and Visualization in clinical research». In: *Proceedings 14th IEEE Symposium on Computer-Based Medical Systems. CBMS 2001*. 2001, pp. 381–386. DOI: 10.1109/CBMS.2001.941749.

[13] Roeggla M Roeggla G Landesmann U. «Ethics of executed person on Internet.» In: 345.8944 (Jan. 28, 1995). DOI: 10.1016/s0140-6736(95)90257-0.

[14] Victor M. Spitzer and David G. Whitlock. «The visible human dataset: The anatomical platform for human simulation». In: *The Anatomical Record* 253.2 (1998), pp. 49–57. DOI: https://doi.org/10.1002/(SICI)1097-0185(199804)253:2<49::AID-AR8>3.0.CO;2-9. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-0185%28199804%29253%3A2%3C49%3A%3AAID-AR8%3E3.0.CO%3B2-9. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0185%28199804%29253%3A2%3C49%3A%3AAID-AR8%3E3.0.CO%3B2-9.

[15] Beveridge et al. «Segmenting images using localized histograms and region merging». In: *International Journal Of Computer Vision* 2 (1989), pp. 311–347. DOI: https://doi.org/10.1007/BF00158168.

[16] R.R. Kohler. «Integrating Non-Semantic Knowledge into Image Segmentation Processes». In: (1984).

[17] K.E. Price R.B. Ohlander and D.R. Reddy. «Picture segmentation using a recursive region splitting method». In: *Computer Graphics and Image Processing* 8.3 (1978), pp. 313–333. DOI: https://doi.org/10.1016/0146-664X(78)90060-6.

[18] Takeo Kanade Yu-Ichi Ohta and Toshiyuki Sakai. «Color information for region segmentation». In: *Computer Graphics and Image Processing* 13.3 (1980), pp. 222–241. DOI: https://doi.org/10.1016/0146-664X(80)90047-7.

[19] Lijun Ding and Ardeshir Goshtasby. «The Canny edge detector revisited». In: *Pattern Recognition* 34.3 (2001), pp. 721–725. DOI: https://doi.org/10.1109/TMI.2005.850541.

[20] Wei Zhang Weibin Rong Zhanjing Li and Lining Sun. «An Improved Canny Edge Detection Algorithm». In: *IEEE* (2014). DOI: https://doi.org/10.1109/ICMA.2014.6885761.

[21] D. Marr and E. Hildreth. ≪Theory of Edge Detection≫. In: *Proc. Royal Society of London* B-207 (1980), pp. 187–217. DOI: https://doi.org/10.1098/rspb.1980.0020.

[22] McIlhagga and William H. ≪The Canny edge detector revisited≫. In: *International Journal of Computer Vision* 91.3 (2005), pp. 251–261. DOI: https://doi.org/10.1109/TMI.2005.850541.

[23] WG Yao and Randy Ellis. ≪A piecewise-linear map for finding edges in images≫. In: Jan. 2003, pp. 117–120.

[24] Andrew Paplinski. ≪Directional filtering in edge detection≫. In: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 7 (Feb. 1998), pp. 611–5. DOI: 10.1109/83.663510.

[25] Greenspan M Yao W Abolmaesumi P and Ellis RE. ≪An estimation/correction algorithm for detecting bone edges in CT images≫. In: 2005, pp. 997–1010. DOI: 10.1109/TMI.2005.850541.

[26] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. ≪Snakes: Active contour models≫. In: *International journal of computer vision* 1.4 (1988), pp. 321–331.

[27] Chenyang Xu and Jerry Prince. ≪Gradient Vector Flow: A New External Force for Snakes≫. In: July 1997, pp. 66–71. ISBN: 0-8186-7822-4. DOI: 10.1109/CVPR.1997.609299.

[28] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. *Geodesic Active Contours.* 1997.

[29] Riccardo Boscolo, Matthew S. Brown, and Michael F. McNitt-Gray. ≪Medical Image Segmentation with Knowledge-guided Robust Active Contours≫. In: *RadioGraphics* 22.2 (2002). PMID: 11896232, pp. 437–448. DOI: 10.1148/radiographics.22.2.g02mr26437. eprint: https://doi.org/10.1148/radiographics.22.2.g02mr26437. URL: https://doi.org/10.1148/radiographics.22.2.g02mr26437.

[30] Zeyun Yu, Ming Xu, and Zhanheng Gao. ≪Biomedical image segmentation via constrained graph cuts and pre-segmentation≫. In: *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society.* IEEE. 2011, pp. 5714–5717.

[31] Faliu Yi and Inkyu Moon. ≪Image segmentation: A survey of graph-cut methods≫. In: *2012 International Conference on Systems and Informatics (ICSAI2012).* 2012, pp. 1936–1941. DOI: 10.1109/ICSAI.2012.6223428.

[32] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. ≪"GrabCut" — Interactive Foreground Extraction using Iterated Graph Cuts≫. In: *ACM Trans. Graph.* 23.3 (2004), pp. 309–314. DOI: https://doi.org/10.1145/1015706.1015720.

[33] *OpenCV: Introduction to OpenCV-Python Tutorials.* 2021. URL: https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html. (accessed: 05.10.2021).

[34] Yu Liu et al. ≪Multi-focus image fusion with a deep convolutional neural network≫. In: *Information Fusion* 36 (2017), pp. 191–207. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2016.12.001. URL: https://www.sciencedirect.com/science/article/pii/S1566253516302081.

[35] Zhuyun Chen and Weihua Li. ≪Multisensor feature fusion for bearing fault diagnosis using sparse autoencoder and deep belief network≫. In: *IEEE Transactions on Instrumentation and Measurement* 66.7 (2017), pp. 1693–1702.

[36] Yu Liu et al. ≪Medical image fusion via convolutional sparsity based morphological component analysis≫. In: *IEEE Signal Processing Letters* 26.3 (2019), pp. 485–489.

[37] Alex Pappachen James and Belur V. Dasarathy. ≪Medical image fusion: A survey of the state of the art≫. In: *Information Fusion* 19 (2014). Special Issue on Information Fusion in Medical Image Computing and Systems, pp. 4–19. ISSN: 1566-2535. DOI: `https://doi.org/10.1016/j.inffus.2013.12.002`. URL: `https://www.sciencedirect.com/science/article/pii/S1566253513001450`.

[38] Surya Prakash, R Abhilash, and Sukhendu Das. ≪Snakecut: An integrated approach based on active contour and grabcut for automatic foreground object segmentation≫. In: *Progress In Computer Vision And Image Analysis*. World Scientific, 2010, pp. 535–555.

[39] Kun He et al. ≪An improved GrabCut on multiscale features≫. In: *Pattern Recognition* 103 (2020), p. 107292.