UNIVERSITY OF CENTRAL GREECE
COMPUTER SCIENCE
AND BIOMEDICAL INFORMATICS

Human Torso Modeling for MRI Simulations

Angelos Chalkias

THESIS PROJECT

Supervisor
Anthony H. Aletras, Ph.D.
Professor, Department of Computer Science and Biomedical Informatics University of Central Greece, Greece

Lamia 30-10-2012

Contents

Abstract

MRI simulations require a realistic and accurate human body model that adds to the validity and accuracy of the simulation results. To build such a model it is necessary to start with the creation of an anatomical map. This map was created using image segmentation on images – transverse raw slices –from an in vitro human body (Virtual Human Project). The information contained in the images was extracted using Matlab and reconstructed in space to form the anatomical map. The result was suitable to be used for the design of the motion models in the MRI simulator.

Introduction

Magnetic Resonance Imaging is a diagnostic tool used in radiology to monitor the human body by means of strong magnetic fields and by taking advantage of the large number of hydrogen atoms in the body. MRI has the advantage of not exposing the patient to ionizing radiation. MRI simulators aid in research, education and training of individuals in a controlled environment without the need of an expensive MRI scanner. A simulation though may differ significantly from reality and may result in misleading results. The use of a realistic model of the human body, which includes anatomical structures as well as biological and MRI parameters, can make the MRI simulation more realistic and accurate. In the past, human body models have been used in projects in the field of Imaging and more specifically in MRI. For example, a human body model has been used for addressing issues of electromagnetic interactions between the surroundings and the human body [1]. However, to date, human body models have not been used in MRI simulators for educational or training.

The specific aim of this project was to create a model of the human body and to visualize in three dimensions this model's output for the torso and heart. This Bachelor's Thesis project describes the development of this realistic human body model based on raw data acquired from a cadaver. This model is intended for use in a Magnetic Resonance Imaging simulator and is based on the dataset of the Visible Human Project. The text that follows will describe how spatial information necessary to form this model was extracted from the raw data by means of image processing. The model development methodology is based on identifying tissue borders within neighboring pixels for segmenting different tissues. Each three-dimensional voxel was created so as to be utilized in a compatible MRI simulation. In this manner, a magnetization vector with its characteristic parameters (T_1, T_2) could be assigned to each voxel by the simulator. We hypothesized that segmentation of the Visual Human Project dataset with Matlab could produce a model of the female body that was compatible with an MRI simulator, that is, it would allow for assigning magnetic resonance parameters to at least the heart.

The organization of this thesis is as follows: basic MRI physics are introduced in order to help the reader better grasp this project's concepts and goals. Then, there is a detailed description of the methods that were applied for image processing and data management. Last, the results are presented along with conclusions, limitations and future work.

References:

[1] ***Vogel, M.H.*** and ***Kleihorst, R.P.*** Large-Scale Simulations Including a Human-Body Model for MRI [http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4264083]

Magnetic Resonance Imaging Basics [4]

Magnetic resonance imaging is a diagnostic tomographic technique which acquires quality images of the human body by means of static and varying magnetic fields. MRI is based on the fact that the human body consists of about 75% water. Since water consists of one atom of oxygen and two atoms of hydrogen, there is an abundance of hydrogen atoms in the human body. The nucleus of the hydrogen atom is a positively charged proton and has a spin associated with it, thus resembling a small spinning magnet: a magnetic dipole (Figure 1). When examining more carefully the alignment of a spin with the static magnetic field, one observes that the alignment is not perfect and that the spins is at an angle with respect to the axis of B_0.Therefore, the spin precesses about the static magnetic field. Precessional motion can be best described by the spinning top motion.

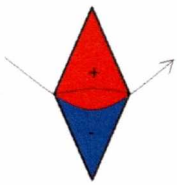


Figure 1: The "spin"

Without the presence of a magnetic field the spins are randomly oriented in the human body (Figure 2 – adapted from [2] and [18]).

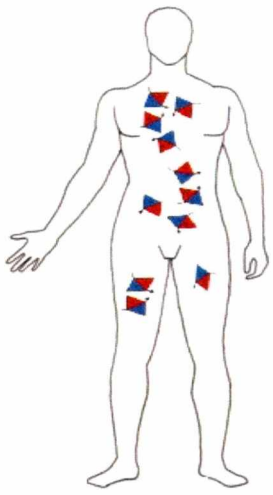


Figure 2: The hydrogen atoms of the human body are randomly oriented inside the body in the absence of an external magnetic field

During an MRI examination the patient is placed lying on his back, on a moving bed (Figure 3 – adapted from [2]) which is inserted inside a large superconducting solenoid, which generates the external magnetic field B_0 with a strength that usually ranges from 1.5 to 7 Tesla. The strength of B_0 depends on the part of the human body we want to image; for example, when the region of interest is the head then a 3 T scanner is preferred. It should be noted that the cost of the MRI scanner is directly proportional of the static field strength. When the patient is subjected to the static external magnetic field, the hydrogen spins align with the direction of the external field in a parallel or anti-parallel manner (Figure 3 – adapted from [2]).This alignment results in a coordinated summation of the magnetic dipoles into a net magnetization vector, M_0 (Figure 4 – adapted from [2]), which can yield an image if properly manipulated. In order to describe magnetic resonance, a Cartesian coordinate system is introduced with its longitudinal z-axis aligned with the field B_0 (Figure 3 – adapted from [3]).

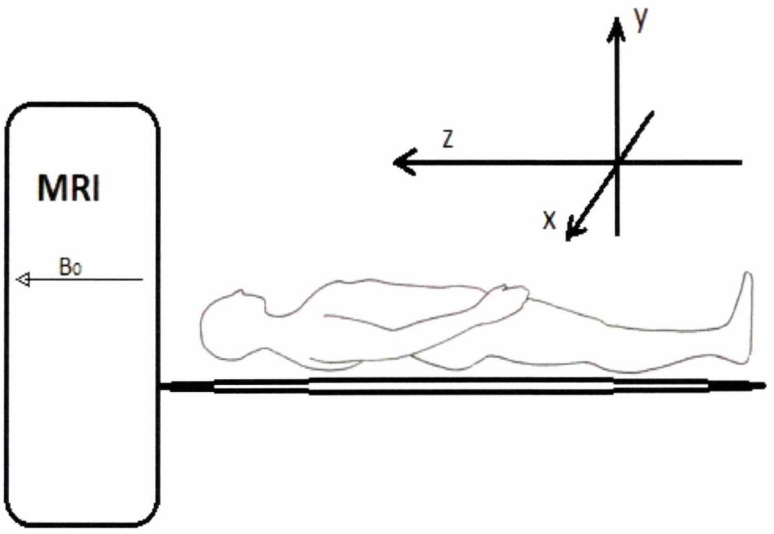


Figure 3: Patient positioning in MRI scanner coordinate system

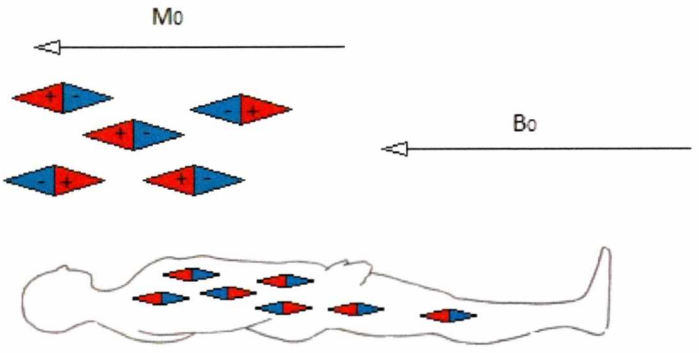


Figure 4: The hydrogen spins align with the static magnetic field B_0.

Inside the static magnetic field, the spins are precessing with a frequency that is calculated via the Larmor equation:

$$f = \gamma_H B_0$$

where f is the frequency, γ_H is for the gyromagnetic ratio of hydrogen (measured in Hz/T), B_0 is the static magnetic field.

The spins not only precess about the static magnetic field B_0 but also precess about any secondary magnetic field that is applied. Such secondary field is used to excite the nuclear spins in order to form the MRI signal. A rotating B_1 magnetic field of a radiofrequency (RF) is applied for a brief time period on the xy-plane so as to rotate ("nutate") the magnetization vector M_0 away from the z-axis and onto the xy-plane. This RF pulse is called a 90^o pulse because it tilts the M_0 vector 90 degrees from its original position. To accomplish this nutation, the B_1 field rotates at the same frequency as the spins precess. When the magnetization vector reaches the xy-plane the RF pulse is removed and M_0 rotates about B_0 onto the xy-plane, thus generating the MRI signal. The spins now rotate being affected at the same time by both fields. If we could visualize their movement in space we would see the magnetization M_0 as a spiral that descends from the z axis extending its diameter, until it reaches the x-y level (Figure 1 – adapted from [3]). This point of view represents the Laboratory Frame of Reference.

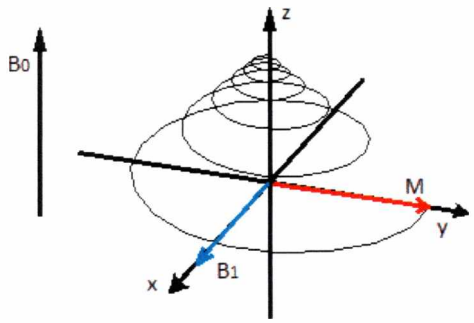


Figure 5: M_0 descent as seen from the Laboratory Frame of Reference

If the observer could somehow sit himself on the B_1 vector and watch the process from that point of view then he would see the B_1 vector being still as he rotates with it and the magnetization vector would be seen descending until it becomes parallel to the y axis (Figure 6A-6B).This is the Rotating Frame of Reference point of view.

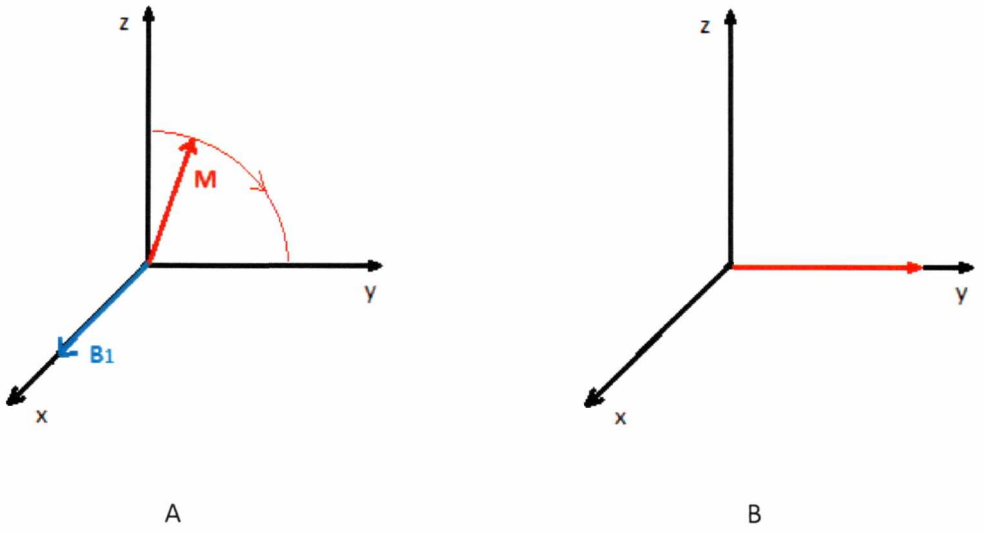


Figure 6: Magnetization descent in the Rotating Frame of Reference

If we want to see how the magnetization vector M rotates about the B1 magnetic field then we use the "left hand rule". The left hand rule dictates that we place our left hand in such a way that our thumb is parallel to the B_1 field and points in the same direction as B_1. The remainder of the fingers will point in the direction in which M will be directed (Figure 8).

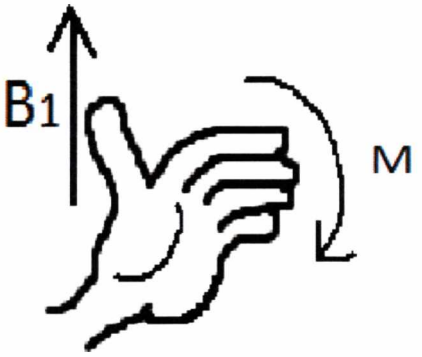


Figure 8: The left hand rule

The signal obtained after the magnetization M is on the x-y level is called Free Induction Decay (FID) because from the moment it is created it starts to diminish until it is completely dissolved. This happens because spins interact with each other gaining different rotational speeds. The decay caused exclusively by these interactions is represented by the T_2 relaxation time of the tissue. There are other causes for the signal to decay though. The magnetic field B_0 is by default not completely homogenous due to design imperfections and

its homogeneity is additionally affected when the patient body is inserted. The signal decay that includes spin to spin interactions as well as the inhomogeneity of the field is described by the T_2^* constant relaxation time. The T_2 and T_2^* times both depend on the body tissue, the myocardium for instance has a T_2^* of about 20 ms and a T_2 of about 50 ms. So, if M_0 is the original magnetization, the magnetization on the x-y level M_{xy} at a given time moment t is calculated by the formula:

$$M_{xy}(t) = M_0 e^{-t/T_2^*}$$

Where M_0 is the magnetization of the initial latitude (magnetization M at its maximum)

The decay due to spin-spin interactions only is calculated likewise using the T_2 constant:

$$M_{xy}(t) = M_0 * e^{-t/T_2}$$

Where M_0 is the magnetization of the initial latitude (magnetization M at its maximum)

After the signal starts to decay the magnetization (vector M) starts to grow along the direction of the z axis. This happens as the spins transfer the energy they have received back to the lattice. This is "spin – lattice" relaxation. The magnetization reformation M_z along the z axis at a given time t is calculated by the formula:

$$M_z(t)=M_0 *(1-e^{-t/T_1})$$

Where Mz(t) is the magnetization formed ay time t and M_0 is the maximum value of the magnetization (i.e. as it was when initially formed) and T_1 is the time constant of the tissue

The *Spin Echo* sequence

The spin-echo sequence is used to calculate the T_2 value since the FID decay includes more than spin-spin interactions. The magnetization is parallel to the y axis due to the 90x pulse that was applied (Figure 7A). After a specific time period TE/2 the spins have dephased due to the field heterogeneities and the magnetization has partially disappeared (Figure 7B). At this point a 180°x pulse is applied and tilts the spins towards –y axis (Figure 7C). After waiting for another TE/2, the spins have rephrased reforming the magnetization parallel to –y axis this time (Figure 7D). The dephasing that occurs now to the newly formed magnetization on the –y is free of field heterogeneities so the T_2 time constant can be calculated.

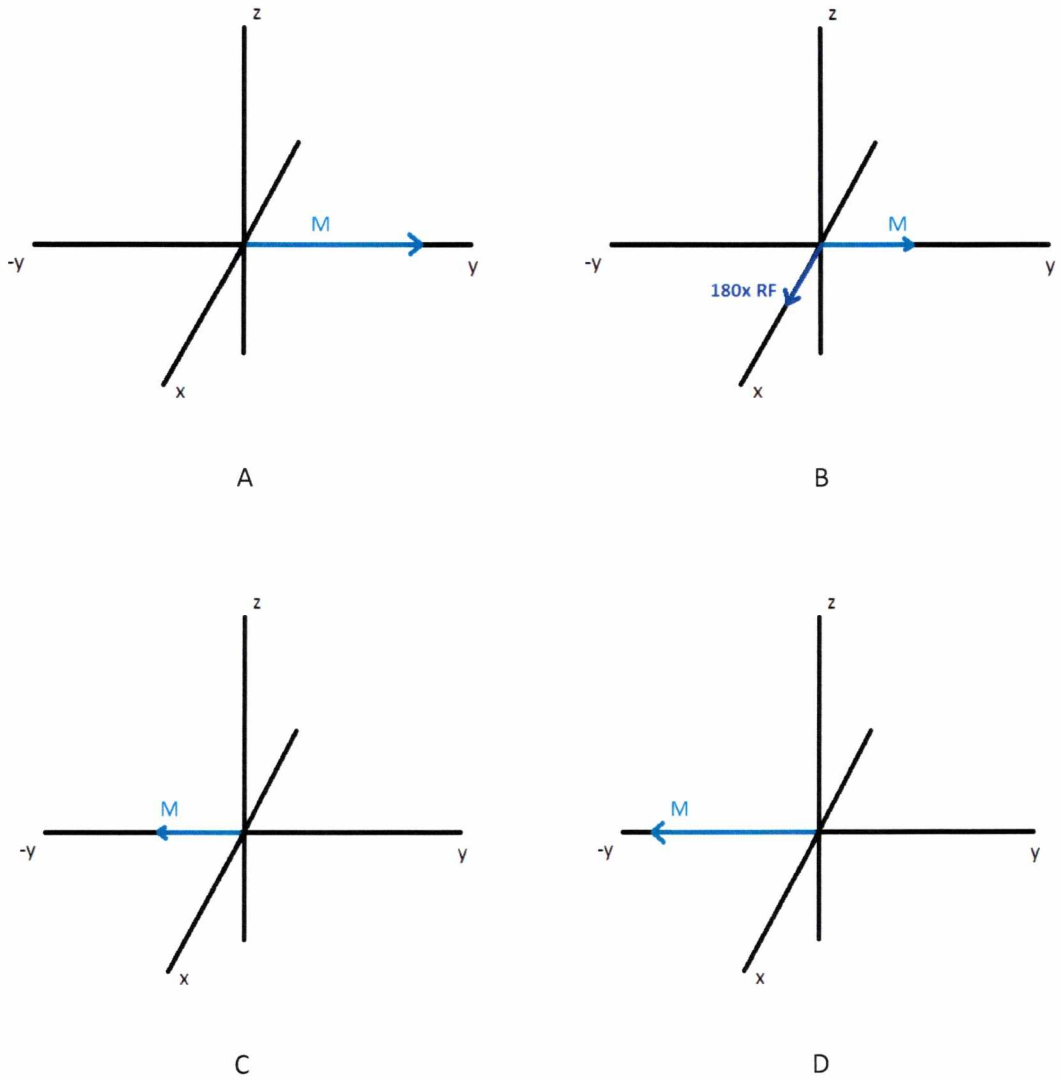


Figure 7: The spin echo sequence

The *Gradient Echo* Imaging Sequence

A gradient is a spatial change in a magnetic field, which is applied in addition to the initial static B_0 field, thus affecting the total at any given position. This has as a result spins at different distances from the center of the magnet to have different precessional speeds according to the Larmor equation. For example, by applying a G_y gradient, spins precess at different frequencies as we move along the y axis. Based on the known strength of the gradient we apply, if we know the precessional frequency of a spin it is possible to calculate its position on the y-axis and vice versa. The precessional frequency of a spin while applying a gradient along the y-axis is calculated in the Laboratory Frame of Reference by the Larmor equation:

$$f=\gamma_H B_o + G_y y$$

where y is the distance from the center of the magnet (isocenter) on the y axis

In general, gradients can be applied in all three axes. So, the Larmor equation becomes:

$$f = \gamma_H (B_0 + G_x x + G_y y + G_z z)$$

where $G_x G_{,y,} C_z$ are the gradients along each axis respectively and x, y, z are the spins distances from the isocenter.

In practice all three gradients are applied in order to obtain an image.

Imaging: Slice Selection

By applying a G_z gradient (Figure 9 A-B) we change the rotational speeds of the spins along the z-axis. This means that we can select a distance at the z axis where the frequency the spins precess to will be equal to the frequency of the B_1 radiofrequency pulse (Figure 9C). Therefore, only the magnetization of the spins that belong to a plane perpendicular to the z-axis at that specific distance on z will be tilted to the xy-plane thus generating a signal (Figure 9D). This is how the "slice" selection is performed.

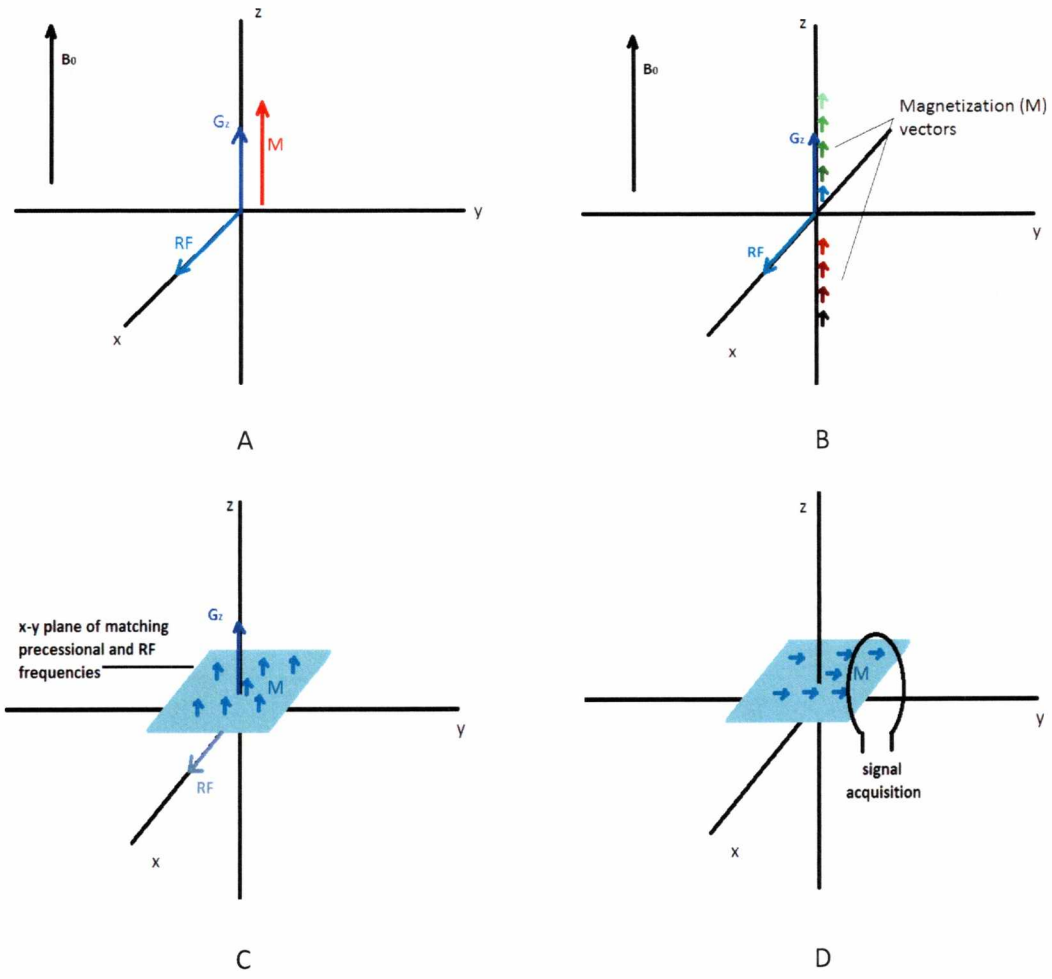


Figure 9: The "slice" selection process: different colors in M represent the different precessional frequencies along the z-axis due to Gz

The G_x gradient is the *frequency encoding* gradient. The application of the G_x gradient results in a frequency differentiation of the spins' precessional frequencies along the x axis (Figure 10A). This means that at a specific distance on the x axis the spins have the same precessional frequency. Since the precessional frequency does not change as a function of the y coordinate, the spins form rows of specific frequencies perpendicular to the x-axis (Figure 10B). The result is having different precessional frequencies in each row while moving along the x-axis. The precessional frequency and the x distance are linearly related as seen by the Larmor equation. Once the signal containing these different frequencies is acquired, these frequencies can be distinguished by means of the Fourier transform. These frequencies can be then translated to location along the x-axis based on the Larmor equation.

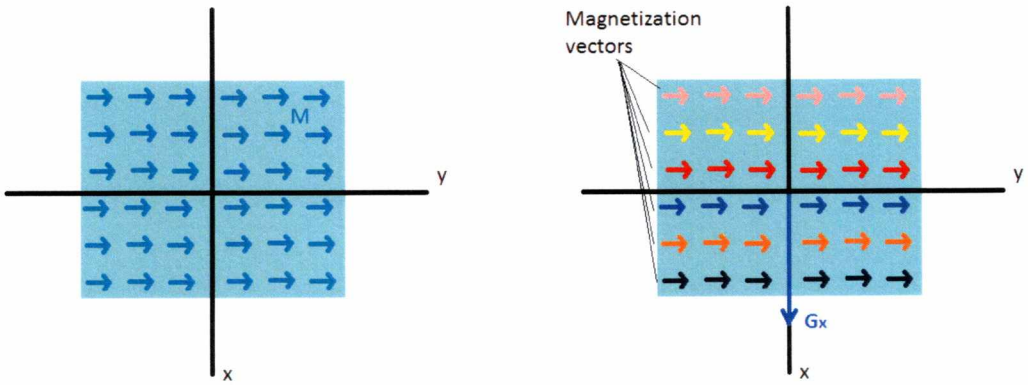


Figure 10.2: Precessional frequency differentiation along the x axis – frequency encoding

Imaging: Phase encoding

To find the position of the spins along the y axis a phase encoding gradient is used. A G_y gradient is applied along the y axis right after the RF pulse has ended and right before the signal acquisition starts. During the application of this G_y gradient pulse the spins along the y axis have different precessional speeds (Figure 11B) which depend on their distance from the isocenter. As a result, at the end of the G_y gradient pulse every spin has the same precessional frequency but a different phase (Figure 11C) depending on its distance from the isocenter. For every location along the y-axis these different spin phases are summed. The Fourier transform cannot extract the original phases from their sum alone and therefore it is not possible to extract y position information from a single experiment such as this. To solve this problem, the experiment is repeated several times with linearly increasing Gy gradient strength. For each experiment, at every y location the phase will be linearly dependent on the Gy gradient strength used for that particular experiment. Moreover, across experiments, at every y location there is a linear dependence of phase to experiment index number. In other words, across experiments there is a different "frequency" for every y location. The Fourier transform can be applied along the "experiment direction" to distinguish these frequencies form one another and provide y position information.

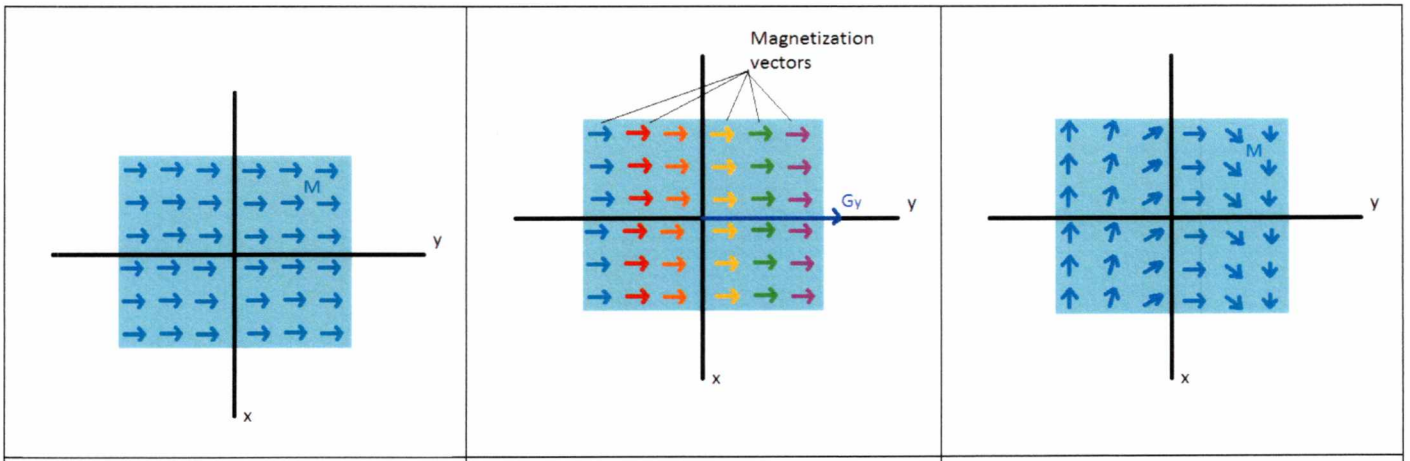


Figure 11: When the Gy gradient pulse stops the magnetization vectors have the same rotational speed but different phases

References:

[2] http://hippie.nu/~unicorn/tut/xhtml/ "Figure 2.1 Proportions of the human body"

[3] http://www.mathematische-basteleien.de/spiral.htm "Conical Helix"

[4] Class Notes in Introduction to Biomedical Engineering, course offered by the Department of Computer Science and Biomedical Informatics, 2010

[18] http://primatologie.revues.org/508 "Figure 1"

Methods

The raw dataset

The dataset used to create this model was provided by the Visible Human Project (VHP) of the United States National Library of Medicine [17]. The Visible Human Project data was generated from male and female cadavers and includes MR, CT and true color high resolution axial anatomical images of the human body. The true color axial images of the female cadaver were used for the model described herein. These images consisted of slices taken at 0.33 mm intervals with a pixel size 0.33 x 0.33mm (i.e. with isotropic voxel). The imaging matrix was 2048x1216 pixels with a color depth of 24 bits (8-bit per color) per pixel and the images covering the entire torso. Each cell in the matrix contained the color intensity of Red, Green and Blue (RGB) corresponding to a spatial pixel location (i.e. the images were supplied in "pixel coordinates"). A total of 1941 images were included in this dataset; however, not all images were used for the purposes of this thesis. Due to the high density of the dataset, only every third image was used to generate the model described below. This decimation of the dataset was dictated by the limited computing power of the laptop computer used for data processing. Using the entire dataset would have resulted in prohibitively long processing times. To account for the loss of slices along the superior-inferior direction, the axial spacing used for data processing was 0.99mm.

Summary of image processing methods

In order to accommodate the needs of the MRI simulation platform, images of the VHP had to be segmented so that, when used by the simulator, individual human organs could be assigned specific magnetic resonance properties, such as proton density, T1, T2, etc. Binary data were generated from the VPH dataset so that the contours delineating the heart and the torso were identified so as to form the 3D objects. These "contour coordinates" were generated by scanning the high resolution VHP images with custom built software in Matlab version R2010a (Mathworks, Natick, Massachusetts, U.S.A.). The conversion took into account the VHP pixel size and the axial slice spatial intervals. The contour coordinates were finally reconstructed to display the 3D human organ. The figure below displays in brief the sequence of the actions just described (Figure 12)

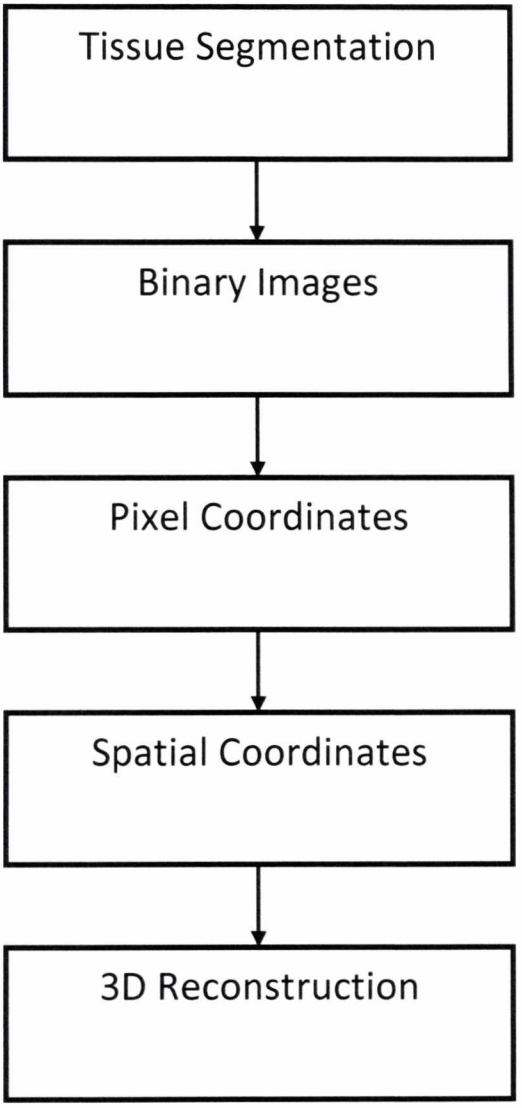


Figure 12: Data processing steps

Segmentation with gray scale images

Gray scale edge detection was first evaluated for detecting the torso boundaries. While gray scale edge detection was not the method utilized for the final product code of this thesis, it is included here for the sake of completeness. In summary, this type of edge detection was based on signal intensity variations that occur throughout the image after the initial dataset has been reduced to 256 shades of gray. The steps used for this type of processing are shown in Figure 13.

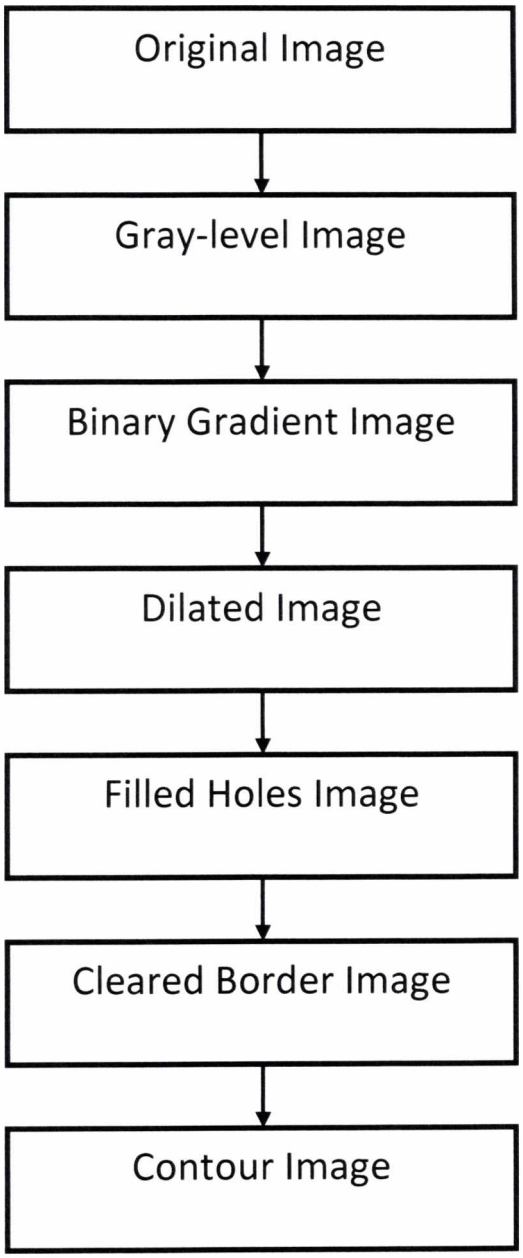


Figure 13: Gray scale image processing

For gray scale segmentation, the boundaries of the torso were initially detected using the image's intensity gradient i.e. using the changes in signal intensity. As mentioned above, first the original 24-bit color image (Figure 14A) was converted to 8-bit grayscale (Figure 14B). This was done using Matlab's "rgb2gray" function [5]. "rgb2gray" converts RGB values to grayscale by using the three components R G B to create a sum. Each component adds to the sum multiplied by a weight value. The weight values are 0.2989 for the R-component 0.5870 for the G component and 0.1140 for the B component. Next, gradient detection was implemented by means of the "edge" Matlab function. An example of a gradient binary image is shown in Figure 14C. The 1's in this image represent the abrupt gradients detected. The "edge" function takes as input the grayscale image and the type of mask we intend to use (for instance sobel, prewitt, roberts) and returns as output a binary image with the edges as pixels with value 1 and the rest with the value 0. In order to later group pixels together and segment the torso, the binary gradient image was dilated. This dilation consisted of assigning to the neighbors of non-zero pixels the value 1. This was done by Matlab's "imdilate" [6] routine which takes as input the binary or gray-level image and a structure element [7] and its output is the dilated image. The structure element is an element that defines the neighborhood of the pixels that are affected during the dilation (for example a disk, a line, or a square). This resulted in a binary image where the white parts are "dilated" as seen in Figure 14D. As a result, the pixels corresponding to the edge of the torso became more connected so that the outer contour could be extracted later on. In segmenting the torso, the next step was to "fill" the gaps and "holes" of the torso gradient image. This was done by using Matlab's "imfill" [8] routine (with parameter "holes") which automatically fills the holes of an image, where a hole in this case is defined in the Matlab documentation as "a set of background pixels that cannot be reached by filling in the background from the edge of the image". The result of this step is shown in Figure 14E: the torso area and the two arms appear now as a solid white object surrounded by a black background, which contains some "salt and pepper" noise.

Prior to identifying the outer contour of the torso and arms, the background noise needed to be cleaned because the binary image background contained non-zero values as a result of signal intensity gradients in the gel that the cadaver was immersed. The image dilation process described earlier exacerbated the situation by increasing the number of non-zero pixels in the background. Removing these background pixels was performed using the "imclearborder"[9] routine. "imclearborder" as described in Matlab's documentation "suppresses structures that are lighter than their surroundings and that are connected to the image border" [9]. Objects that consisted of a number of pixels less than a predefined value were erased. This was implemented by means of the "bwareaopen" [10] function in Matlab, where the user defines the minimum amount of pixels that the objects should have. The resulting image was clear from small objects. The result of this step is shown in Figure 14F where the torso and arms appear as a solid white object over a black background. In this type of binary image the contour of the torso and arms can be more easily obtained as requested for previous steps "by applying an edge detection routine the so called "edge" [11], which now in contrary to its former application only detects a single object resulting in a continuous single-lined contour (Figure 14G).

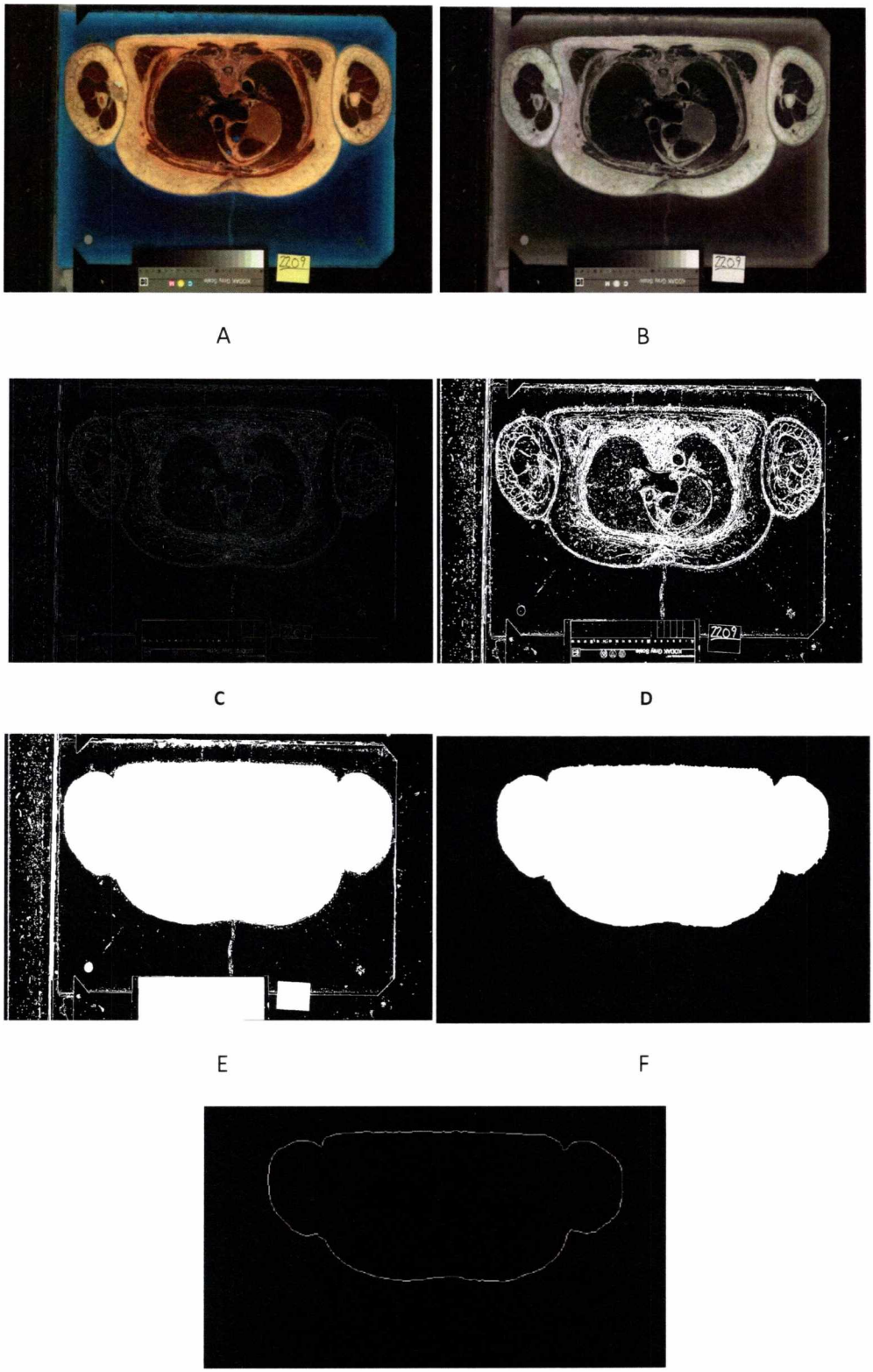


Figure 14: The steps of the initial image processing sequence

A casual viewer of these contours may conclude that this type of gray scale image processing works well. However, a more careful examination of the results indicates that

there are problems with this segmentation method since the torso was not completely separated from the blue background i.e. from the surrounding gel (Figure 15).

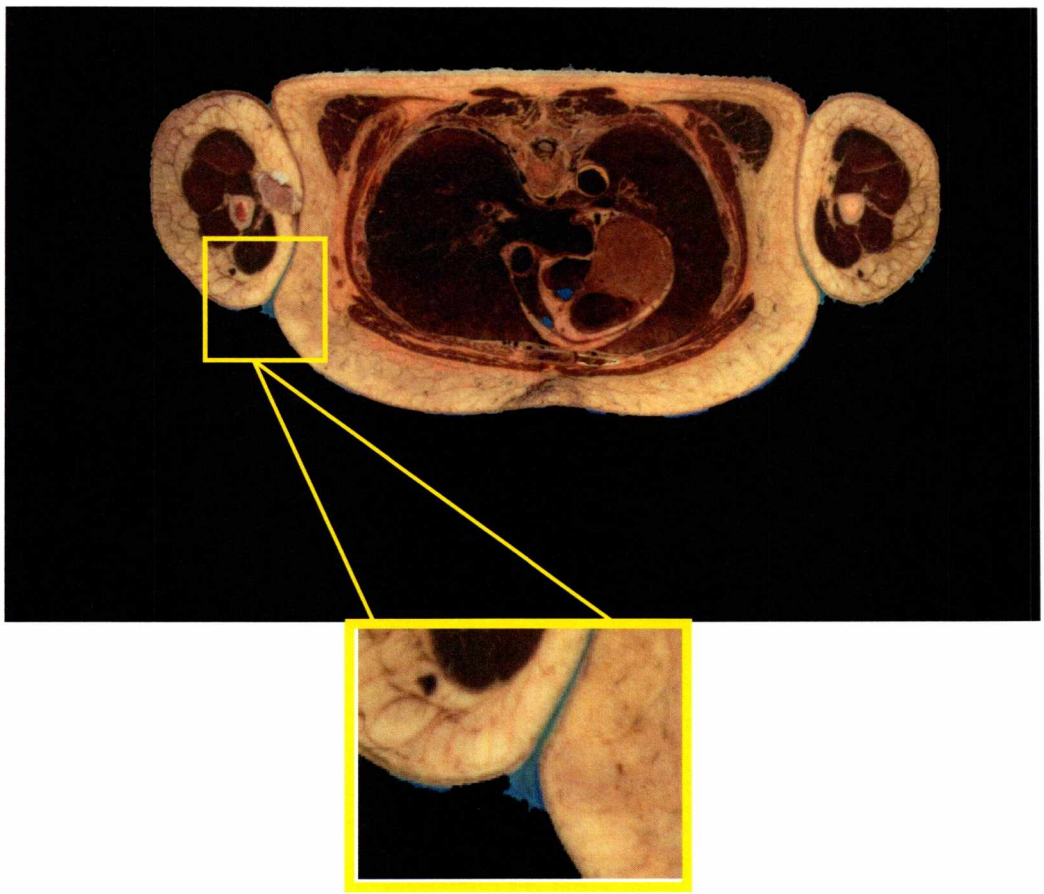


Figure 15: Blue objects being detected as part of the contour of the torso were considered significant errors that pointed to methodological improvements

Segmentation with true color images

To improve the segmentation a new methodology was developed based on a Matlab demo code [12] that discriminates an images different colors using the L*a*b color space for classification.

*The L*a*b color space*

The images that contain the VHP dataset are true-color png images. This allowed for alternative image processing which takes advantage of the different colors so as to detect the different tissues. The colors were classified using the L*a*b color space model (Figure 16). According to the L*a*b color space every color was represented by three components. The L-component stands for luminosity with 0 representing the darker value (black) and as

the values increase the brighter the objects become i.e. approaching white. The a-component indicates where the color is located between the green (negative a values) and the red (positive a values). The b-component indicates the color's position between the blue (negative b values) and the yellow (positive b values).

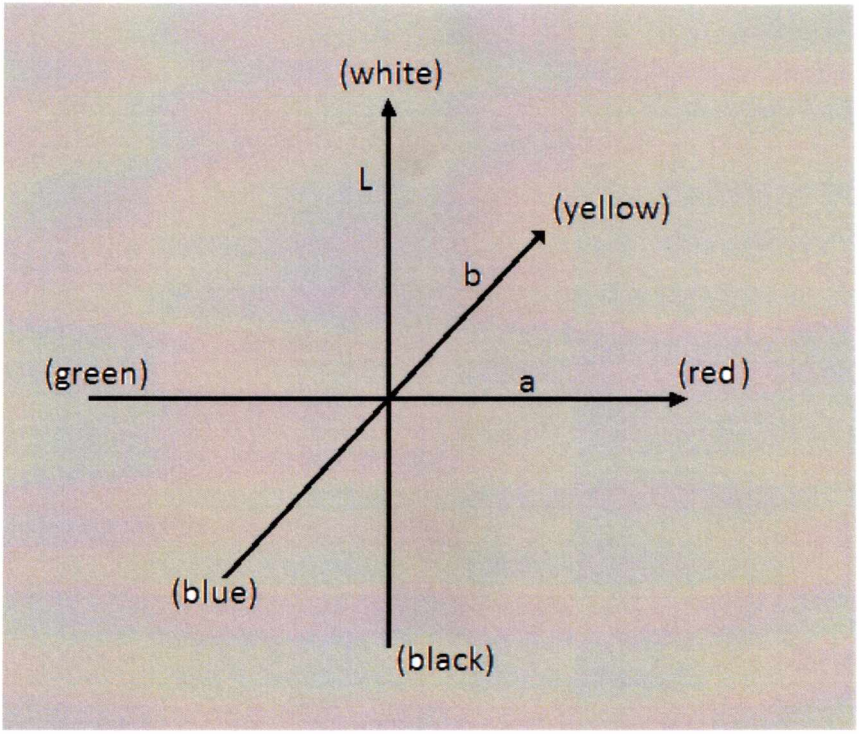


Figure 16: The L*a*b color space coordinates

The use of this color model simplifies the segmentation procedure and fewer steps are needed (Figure 17).

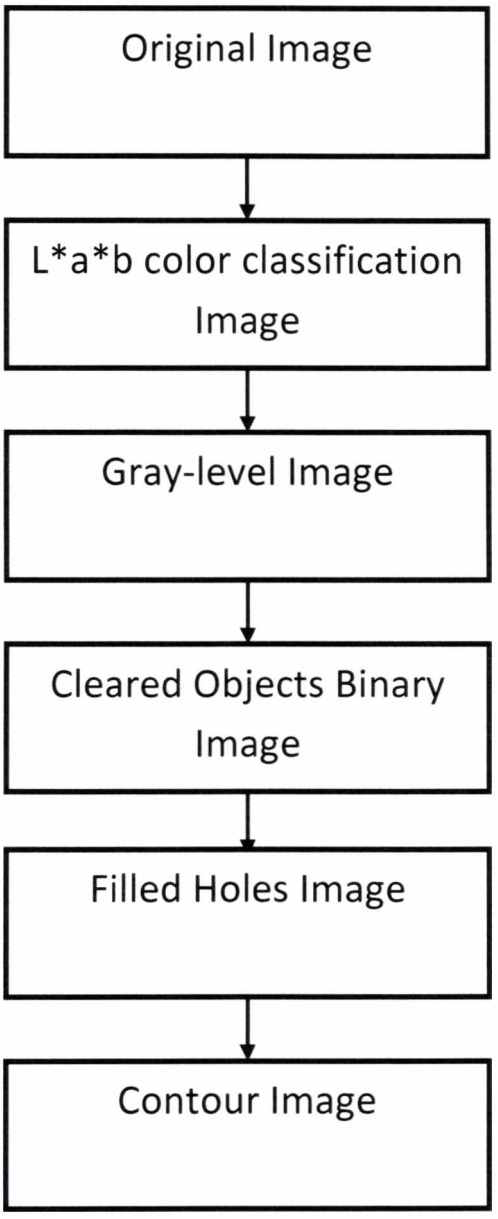


Figure 17: The final algorithm for the detection of the torso

The original Matlab code [12] classifies the image colors in six different classes using predefined areas of the image for sampling in order to create a center for each color class using the coordinates of the L*a*b color space. Following this classification, it calculates for each pixel the distances from each class using the k-nearest neighbors method. The code

was modified to detect only three colors i.e. there were three class centers. The coordinates of the class centers in the L*a*b color space were created just once by manually defining the sampling areas for the colors and then using part of the original Matlab code to create the L*a*b coordinates for the colors of the areas used for sampling. One center was created for the torso for a sampling area that included all the colors of the torso area (fat – yellow, muscle/flesh – red, blood – darker red). One center was created for a sampling area that included the blue gel exclusively. Last, a third center was created using a sampling area with just the black background (outside the blue gel). By applying this methodology three images were obtained, each of them displaying the areas just described. The image of interest was the one that resulted from the center created for the torso area (Figure.18.A) and it could be used to define the torso area. Using this image a binary image was created with the torso area being 1's. The torso's edge was then detected by applying the same steps for removing the unnecessary objects as described for the gray images in the previous section. In brief, the image was transformed to gray-level (Figure 18B) and by applying Matlab's routines [9] [10] mentioned previously, the objects near the border were removed. Objects with a number of pixels less than a certain value were also removed (Figure 18C). Black "gaps" resulted inside the torso because the pixels values of some areas inside the torso (lungs for example) were close to those of pixels in the black background. These black "gaps" of the inner torso area were "filled" (Figure 19D) using "imfill" [9] so that the torso created one continuous area in the resulting binary picture. At this point the contour could be extracted using edge detection (Matlab's "edge"[11] routine) as done in the gray-scale image segmentation (Figure 19E).

Repeating this algorithm for the entire set of images yielded the 3D contour area of the entire torso.

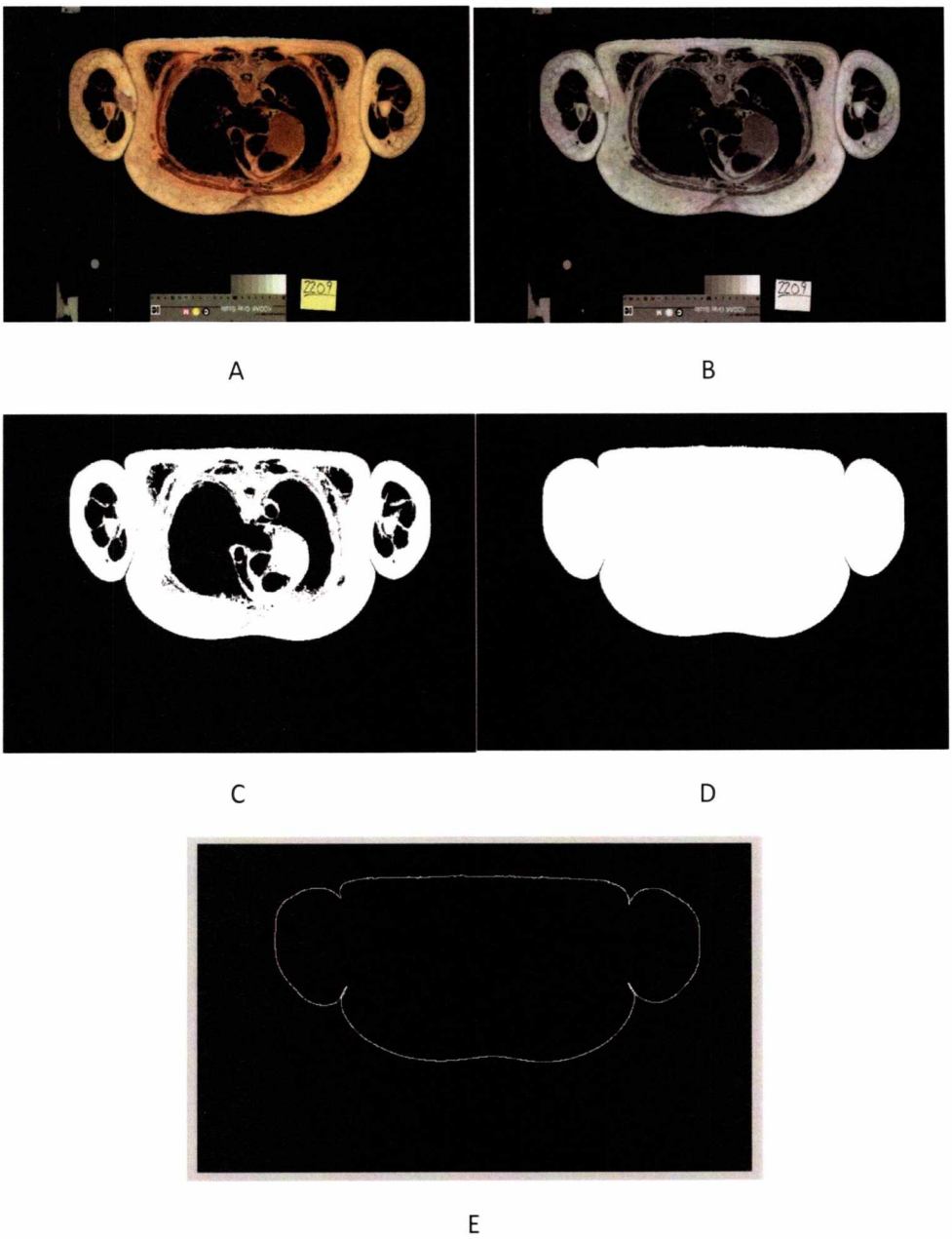


Figure 19: Color classification using the L*a*b color space

Segmenting the heart

The L*a*b color space technique was used for segmenting of the heart as well. This time though there was no blue gel involved and the discrimination considered different colors due to different tissues. As a result different sampling areas for the formation of the class centers in L*a*b spatial coordinates were used. The procedure steps were similar to those of the torso detection algorithm with the exception of engaging a manual frame selection this time (Figure 18).

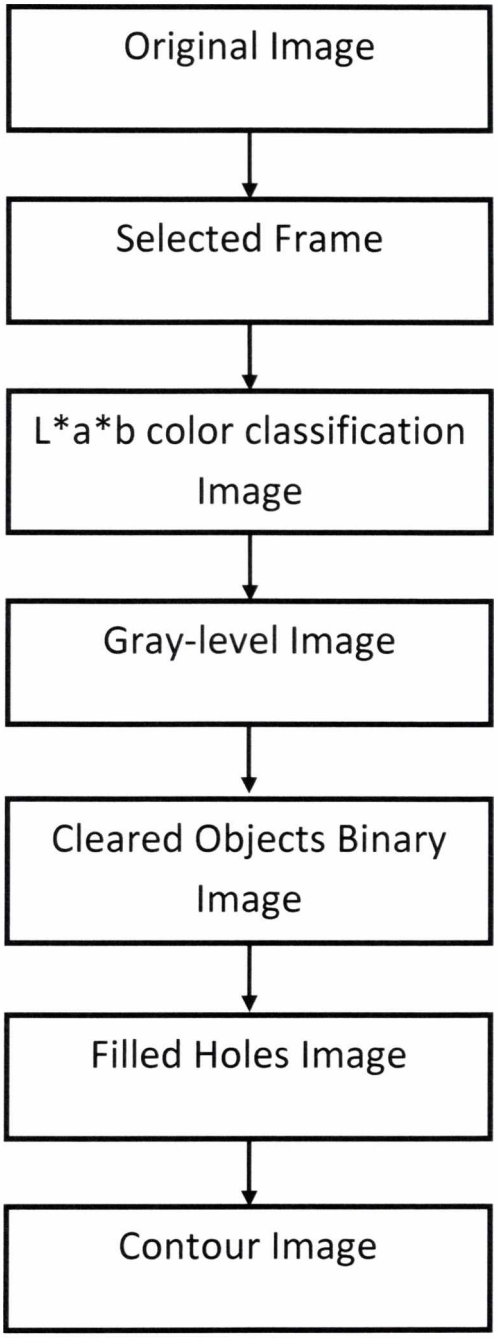


Figure 18: The heart area was segmented using the same method as the one used for the torso

Identifying the heart structures

The segmentation of the heart included the identification of cardiac structures. Those were the Superior Vena Cava, Pulmonary Trunk, Right Pulmonary Artery, Aorta, Left Atrium, Right Atrium, Left Atrium and Left Ventricle areas (Figure 19). Another tissue of the heart was included in the heart components tissues, not visible in the figure below, was the intraventricular septum, which is not easily detected by observing the "slices" and therefore its segmentation was not as accurate.

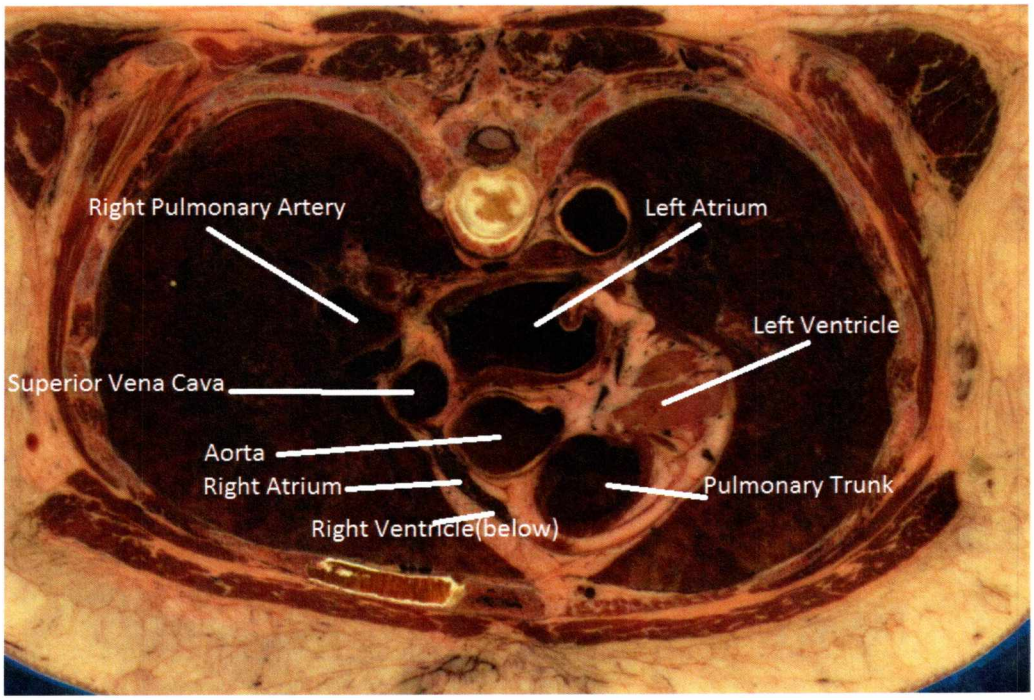
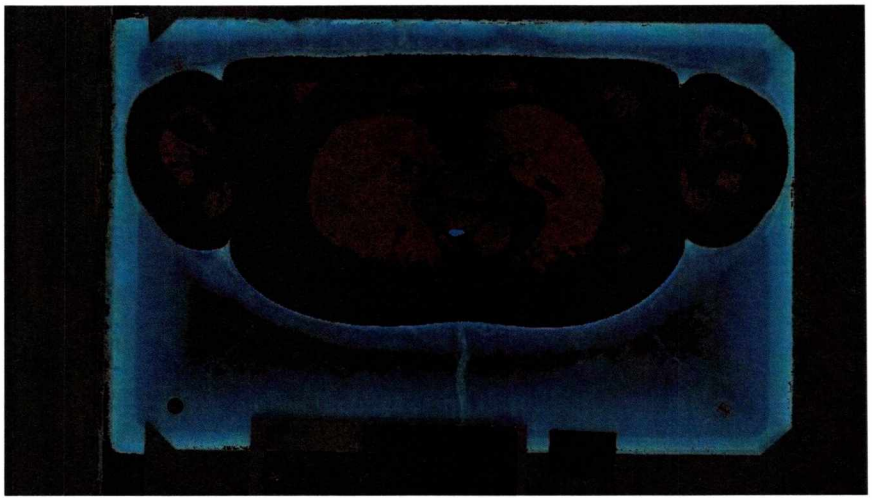


Image 19: The heart structures

There were three class centers formed like before, coming from three sampling areas including a region for fat, a region for flesh (lighter red colored tissues) and a region for the magenta objects (like the inner area of lungs and vessels – darker red colored areas). The execution of this code generated three images (Figures 20A-C) were the colors were classified thus helping identifying the structure of interest.



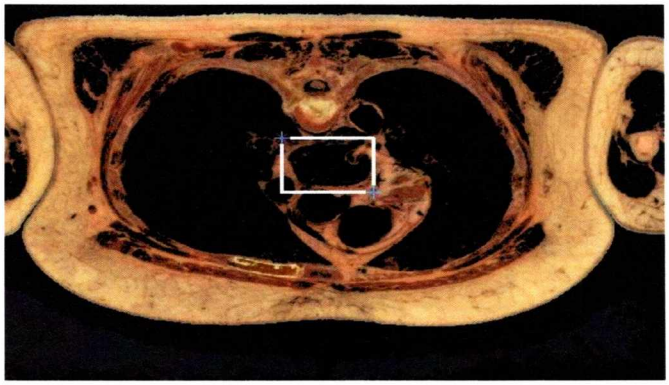
A

B

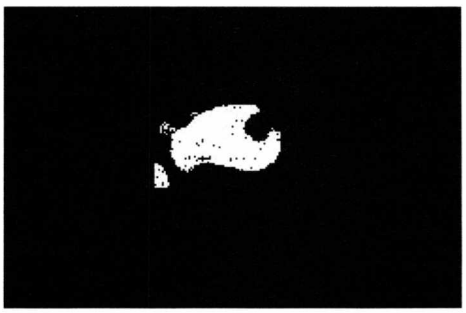
C

Figure 20: The classified colors

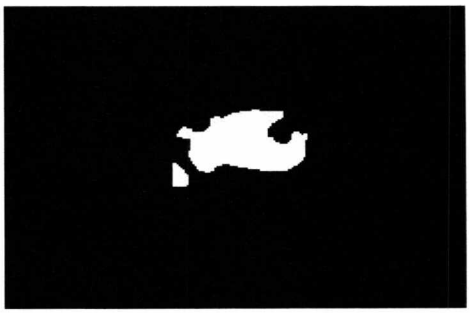
By observing these three images one can see that the colors were not classified completely and exclusively i.e. every picture contained a range of colors instead of just one. This was because the sampling areas that were created happened to contain pixels within a range of values. So, the corresponding L *a*b coordinates represented a range of values. In practice same color values exist in areas of different tissues. The resulting pictures were useful though in detecting the different tissues of the heart. For example the left atrium appeared in the first picture (Figure 20A) as a brown-purple colored area surrounded by a black area. This meant that it could be detected by forming a binary image using its non-zero valued pixels. In addition, it could also be detected using the third picture (Figure 20C) to form a binary image by selecting the zero-valued (i.e. the black pixels) this time. The procedure that was followed included some manual intervention in order to specify a frame that contained the region of interest exclusively. For example, in order to detect the left atrium in the picture in Figure 20A using the third picture, a binary image was created from the zero-valued pixels of the Figure 20C. To achieve that a frame was set to include only the region of interest and erase the rest (Figure 21A). The frame selection was manual and Matlab's "getpts"[13] routine was used to achieve that. "getpts" allows mouse clicking on images for manual point selection. In the binary image that was created, the lower left edge of the frame (Figure 21B) included a disconnected small unnecessary area that needed to be removed. The binary image was at that point dilated and the holes were filled so that potentially unconnected objects became one entire continuous area (Figure 21C). Objects with an amount of pixels less than a predefined value were removed, using the Matlab functions [9] as before so that the unnecessary area was removed (Figure 21D). The application of an edge detection [11] algorithm produced the contour of the left atrium (Figure 21E).



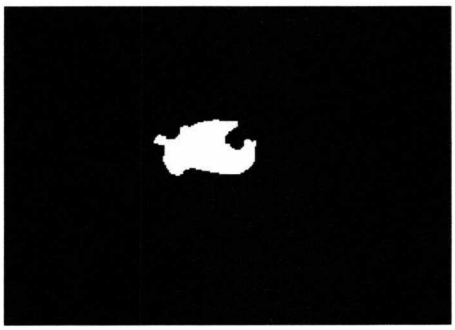
A



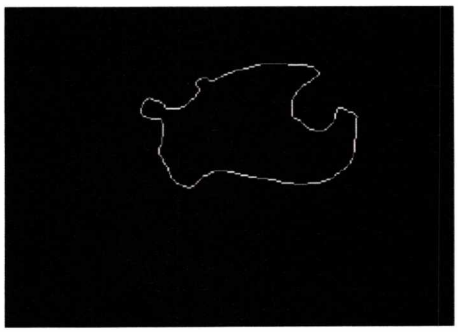
B



C



D



E

Figure 21: Segmenting the heart structures

This procedure was used for all other heart structures as well. Once the contours of some of the heart structures had been identified then these could also be used to help in identifying the remaining ones. For instance, in identifying the aorta, the pulmonary artery had already been identified; a fact that made it easier to find the borders of the aorta on the side of the pulmonary artery. This was done by excluding from the aorta the pixels belonging to the pulmonary artery. This logic was applied when a tissue neighboring to the region of interest had already been identified.

Contour undersampling

Each of the resulting contours consisted of a large number of points that changed while moving from slice to slice (i.e. image to image) in the inferior-superior direction. This makes sense since the diameter of the organ, e.g. the heart, may change along that axis. As expected, a contour originating from the base of the heart will consist of more points than a contour originating from the apex of the heart. The MRI simulator demanded that for purposes of visualization and potential modeling of motion that consecutive contours contained the same small number of points. To achieve this, the existing contours underwent an undersampling process so that in the end the contour of every slice contained a fixed number of points. The algorithm developed to achieve this is described below.

First, the contour image that was produced by the segmentation procedure was read in Matlab (Figure 22). Then, the contour's center of mass was located (Figure 23) by means of the function "regionprops" [14] routine with the parameter "centroid" set. "regionprops" calculates a set of properties that include shape or pixel value measurements regarding objects of the image. The specific parameter defines that the function should calculate the center of mass of the images object. Once the center had been located, a line of pixels (referred to as "main line" herein) was expanded from the center of mass towards the contour of the image (Figure 24). The main line of pixels was expanded until a certain predefined length of pixels. For every pixel that was added to the main line, a scan was executed from the pixel located 7 pixels back on the x and 7 pixels back on the y axis up to the pixel located 7 pixels forward on the x axis and 7 pixels forward on the y axis. If the pixel of the main line had coordinates (x0,y0) the scan included a square area whose diagonal corners were the points (x0-7,y0-7) i.e. the starting point, (x0+7,y0+7) i.e. the ending point. Whenever a contour pixel was found inside this scanning area, its distance from the pixel of the main line was calculated. If the current distance was less than an already calculated one from a previous iteration, then its value replaced the previously stored distance value and correspondingly the coordinates of the current contour pixel replaced the stored coordinates of the previous contour pixel. So, when the scan was completed for one of the pixels of the main line, its distance from the closest contour pixel had been calculated and if it was less than the correspondingly calculated distance for the pervious pixel of the main line, the new distance value replaced the previous one and the coordinates of the new main line pixel replaced those of the previous one. The distance was given an initialization value in case this was the first iteration and there was no previous value to be replaced. So, when all the scans were executed for all the pixels of the main line, the saved pixel coordinates were those of the expanding main line pixel closer to the contour, if an actual pixel of the contour was not matched. The same procedure was repeated for pixel lines that expanded from the center for different angles until a full 360 degrees rotation was complete. The rotational angle for the heart components was set at 10 degrees, meaning that starting from 0 degrees a main line was expanded from the center of mass of the the contour towards the contour itself every 10 degrees detecting one point of the contour. This way, 36 points were detected in total for each contour. The same procedure was applied for the undersampling of the hearts neighboring vessels such as the pulmonary artery, except that the rotational angle was set at 30 degrees thus resulting in a total of 12 points per contour. In order to rotate the expanding main line of pixels the computation and application of these angles

was necessary. That was achieved by using the "cos" [15] and "sin" [16] functions of Matlab. These functions return the cosine and sine values. Whenever the main line was expanded one pixel inside the loop, the added pixel's x coordinate was multiplied by cos(a) and its y coordinate by sin(a), where a was the current angle. As mentioned above, the angle a changed with a step of 10 or 30 degrees per loop depending on the component until a 360 scan of the contour was performed. This way, the pixel line could be expanded towards various angles. The initial design of this algorithm included expanding the main pixel lines until a contour pixel was met. The coordinates of the pixels belonging to the expanding line did not always coincide with those of a contour pixel, because they did not always form a continuous line (due to the use of angles), meaning that although the expanding line of pixels crossed the contour, there were not always coordinates of a pixel saved. This posed a problem (Figure 25). The scanning around the expanding main line of pixels solved this problem. In a case of non coincidence of the coordinates, the pixel of the main line with the smallest distance from the contour was considered as a contour pixel met and its coordinates were saved. So, it was ensured that every line that will cross the contour will trace and save the coordinates of a pixel.

The aforementioned undersampling came with a slight accuracy loss since a pixel neighbor to the contour pixel may have replaced the actual contour pixel. In case the expanding line of pixels did not find a match to the contour, the pixel with the closest distance to the contour was considered the contour pixel instead of the actual contour pixel itself. The maximum distance of a neighbor pixel which was chosen as a contour pixel instead of an actual contour pixel can be calculated. Assuming the pixel of the expanding main line had coordinates (x_0,y_0) and the scan finds a contour pixel located at the most distanced point possible i.e. either the (x_0+7,y_0+7) or the point (x_0-7,y_0-7) it makes no difference, their distance (d) is d= $\sqrt{(x_0 + 7 - x_0)^2 + (y_0 + 7 - y_0)^2} = \sqrt{49 + 49}$ = 9.89 pixel. Knowing that the pixel size is 0.33 mm we can calculate the spatial distance in mm which is 9.89 x 0.33=3.2637mm. That is the maximum diversion that may occur from this process according to the worse scenario where the most distant pixel has been chosen as a contour pixel. Note that the undersampling process aims in improving the result visually (and potentially enabling the formation of areas which can be modeled for motion),the loss of accuracy takes place exclusively in the visualization process and does not affect the saved data. This accuracy loss is visually undetectable.

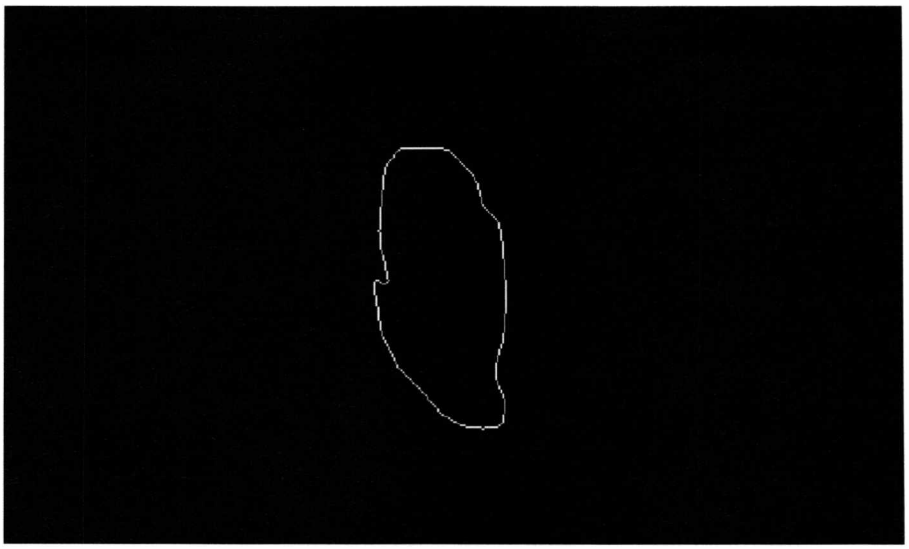


Figure 22: The contour picture of the right atrium (input)

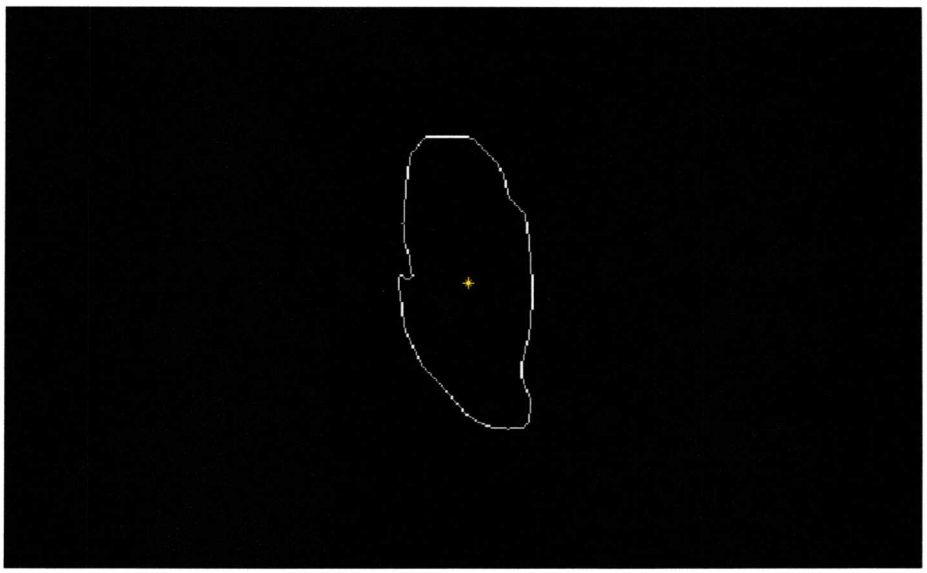


Figure 23: The contour center is detected

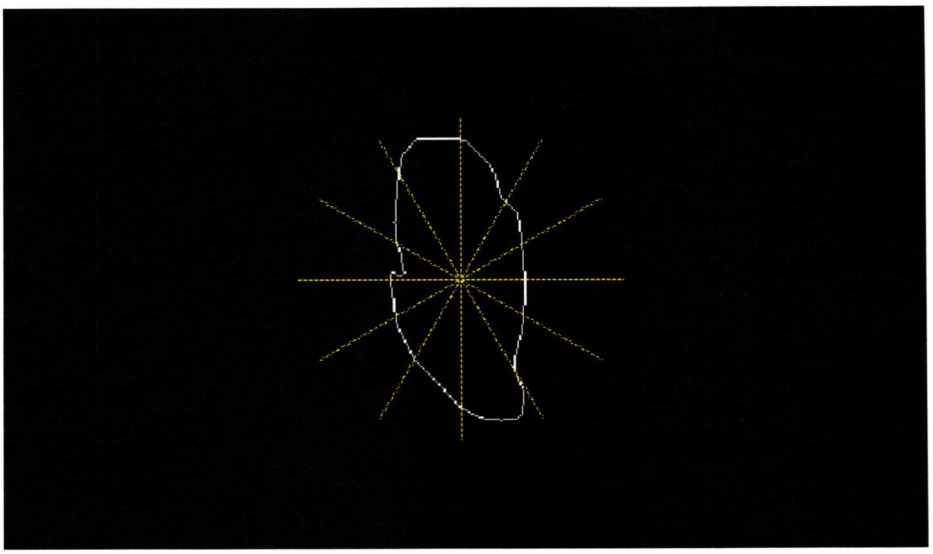


Figure 24: The pixel lines cross the contour while expanding

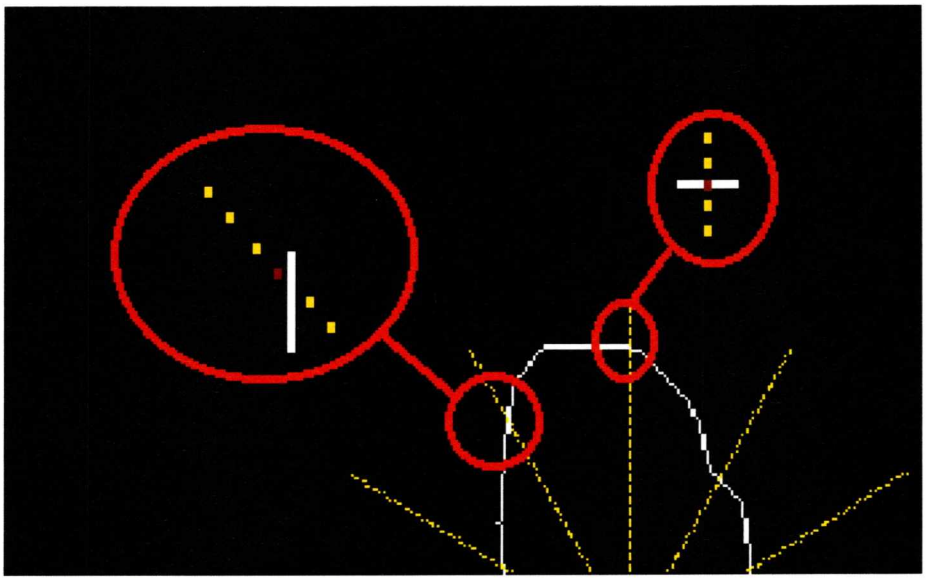


Figure 25: Zooming in to the procedure, yellow pixels: the expanding main line, magenta pixels: the pixels considered as contour pixels met, white pixels: contour pixels.

Data storage and coordinate system

When all of the contours had been successfully created the data were stored in an array with three columns where the first column stored the pixel's x coordinate the second column the pixel's y coordinate and the third column the pixel's z coordinate which changes along the inferior-superior direction i.e. along the slice direction. As the images were read in one after another the z coordinate was reduced, so that the pixels of the current contour were placed below those of the previously read image. This array was then saved as a .mat file. At that point the coordinates in pixel values had been saved, since these coordinates resulted straight from scanning the images. In order to transition to spatial coordinates, the data acquisition parameters, as described in the official webpage of the Visible Human Project, were considered. The pixel dimensions were 0.33x0.33 mm so by multiplying the first two columns (x and y coordinates) of the array with 0.33 these were automatically converted to spatial coordinates in mm. The VHP images had been obtained every 0.33 mm but as mentioned earlier the images used to form the model were not read one after another but for every image used the two following images were skipped instead. That meant that the distance along the z axis was 0.99mm between slices. By multiplying the third column of the array (z coordinates) by 0.99 the transformation into spatial coordinates was completed. The x,y,z spatial coordinate values were also divided by 1000 in order to convert from mm to meters.

References:

[5] rgb2gray: http://www.mathworks.com/help/images/ref/rgb2gray.html

[6] imdilate: http://www.mathworks.com/help/images/ref/imdilate.html

[7] imfill: http://www.mathworks.com/help/images/ref/imdilate.html

[9] bwareaopen: http://www.mathworks.com/help/images/ref/bwareaopen.html

[10] imclearborder : http://www.mathworks.com/help/images/ref/imclearborder.html

[11] edge: http://www.mathworks.com/help/images/ref/edge.html

[12] lab color space: http://www.mathworks.com/help/imaq/examples/color-based-segmentation-of-fabric-using-the-l-a-b-color-space.html

[13] getpts: http://www.mathworks.com/help/images/ref/getpts.html

[14] regionprops: http://www.mathworks.com/help/images/ref/regionprops.html

[15] cos: http://www.mathworks.com/help/matlab/ref/cos.html

[16] sin: http://www.mathworks.com/help/matlab/ref/sin.html

[17] VHP NLM dataset: http://www.nlm.nih.gov/research/visible/visible_human.html

Results

The initial gray scale algorithm

As mentioned in Methods, the gray scale image segmentation approach method was not preferred due to poor results (Figure15). By overlaying the initial image to the binary image resulting from the segmentation one can observe quite a few cases when blue objects remained attached to the torso and thus erroneously considered to be part of the torso.

*The L*a*b color space algorithm*

To verify the results of the L*a*b color space algorithm, the overlay of the initial image to that of the algorithm's resulting binary image shows that the torso area is detected without any additional objects attached to it in error (Figure 26)

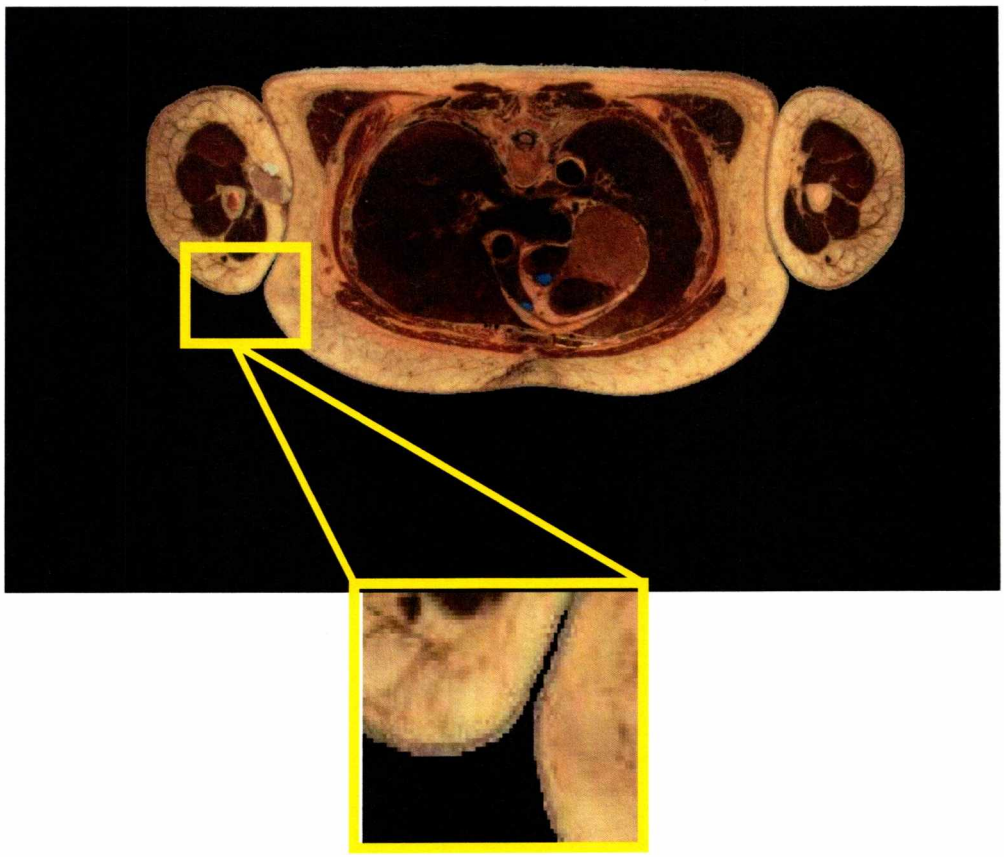


Figure 26: The torso area contains no errors, the improvement is visible in comparison to the previous results seen in Figure 15

Since the binary images of the L*a*b color space algorithm produced successful detection of the boundaries of the torso area, an edge detection function was applied in order to obtain the torso contour (Figure 27).

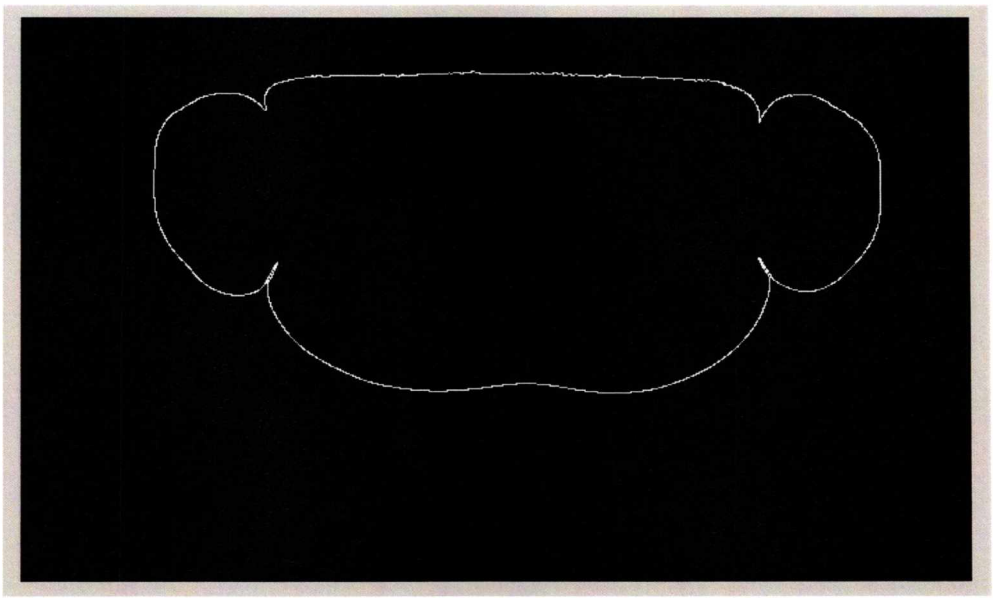
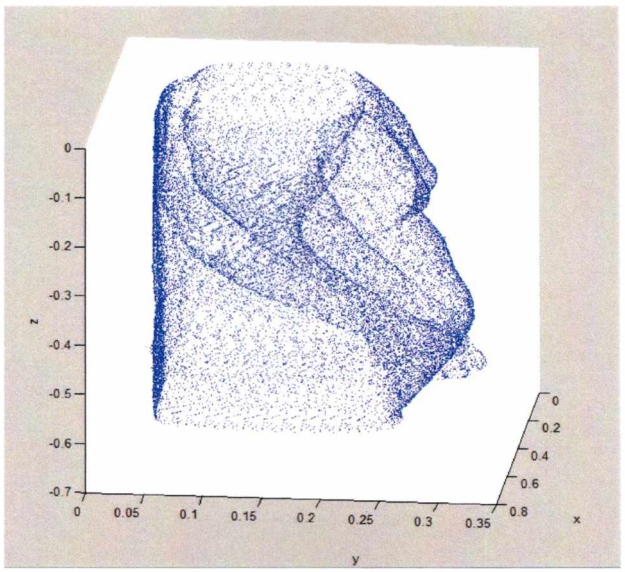


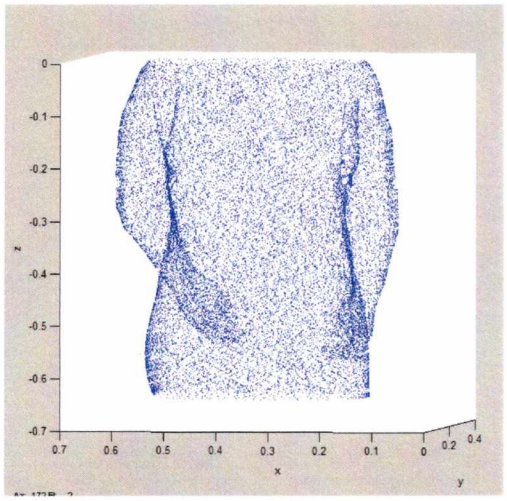
Figure 27: The corresponding contour of the torso area segmented using the L*a*b color space

Visualizing the data

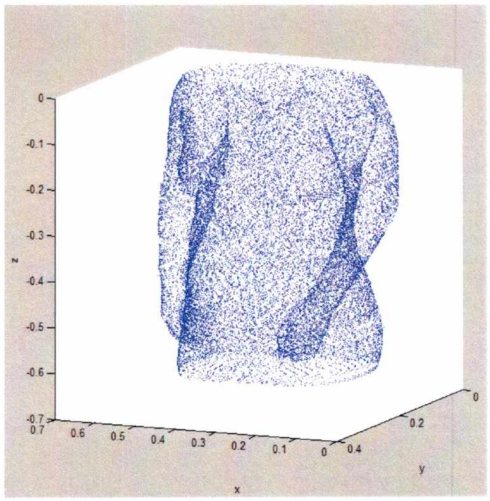
To visualize the data gathered from the images in 3D we read the arrays with the saved coordinates and plotted the points in space (Figure 28) one by one in order to create a 3D visualization.



A



B



C

Figure 28: The torso reconstructed in 3D by plotting the data one point at a time as seen from different angles

The heart was visualized likewise (Figure 29)

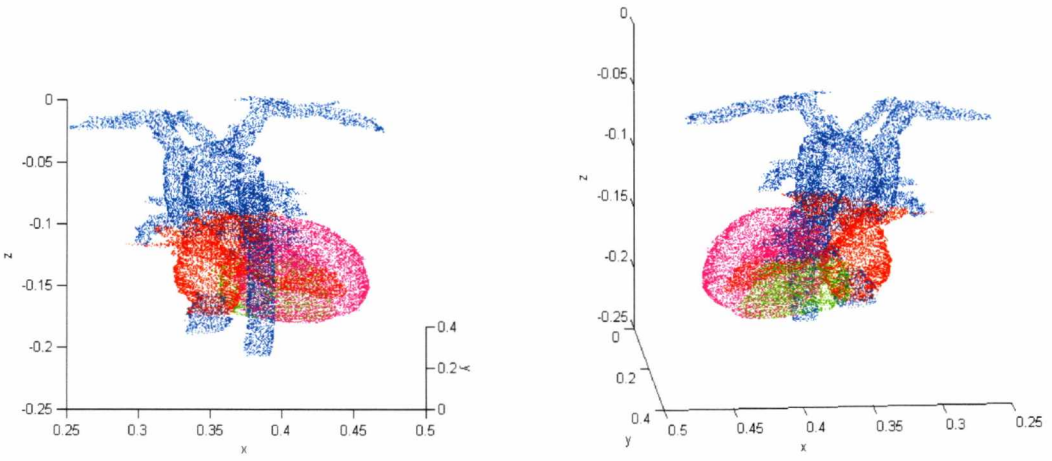


Figure 29: The heart visualized in space by plotting one by one the points that compose it

The above 3D plots could not be manipulated easily since the hardware proved to be inadequate for the volume of data. This was easily noticed when manually trying to rotate the object in space and experiencing a very slow response. By plotting the data that occur from the undersampling process, the visualization result was easier to rotate in real time (Figure 30). The data after the undersampling were less dense, which improved the response during the object rotation in space. In addition, the heart consisted of points that were aligned along the z axis, which allowed for connecting them with each other thus creating the effect of a surface. Some of the details may be lost in this type of visualization. The images below show the spatial visualization of the data resulting from the undersampling process. The 36 or 12 points were connected horizontally per slice and vertically per angle location (angular step during undersampling).

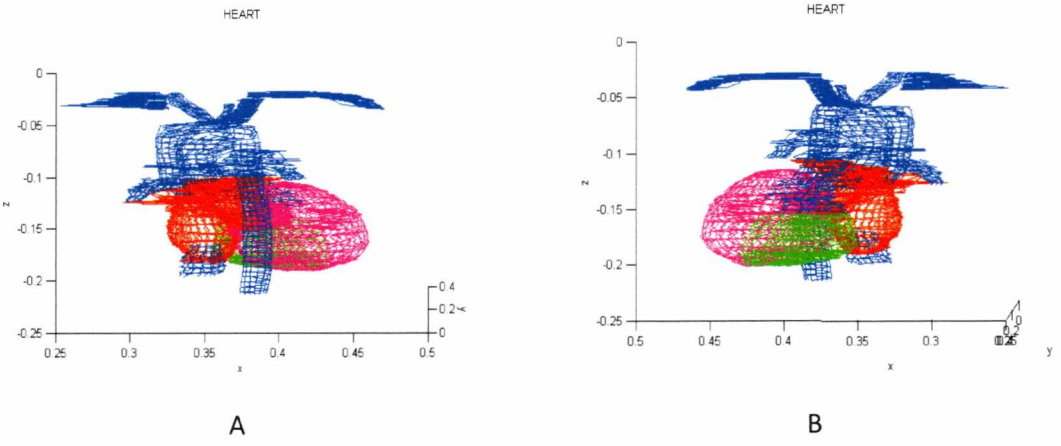
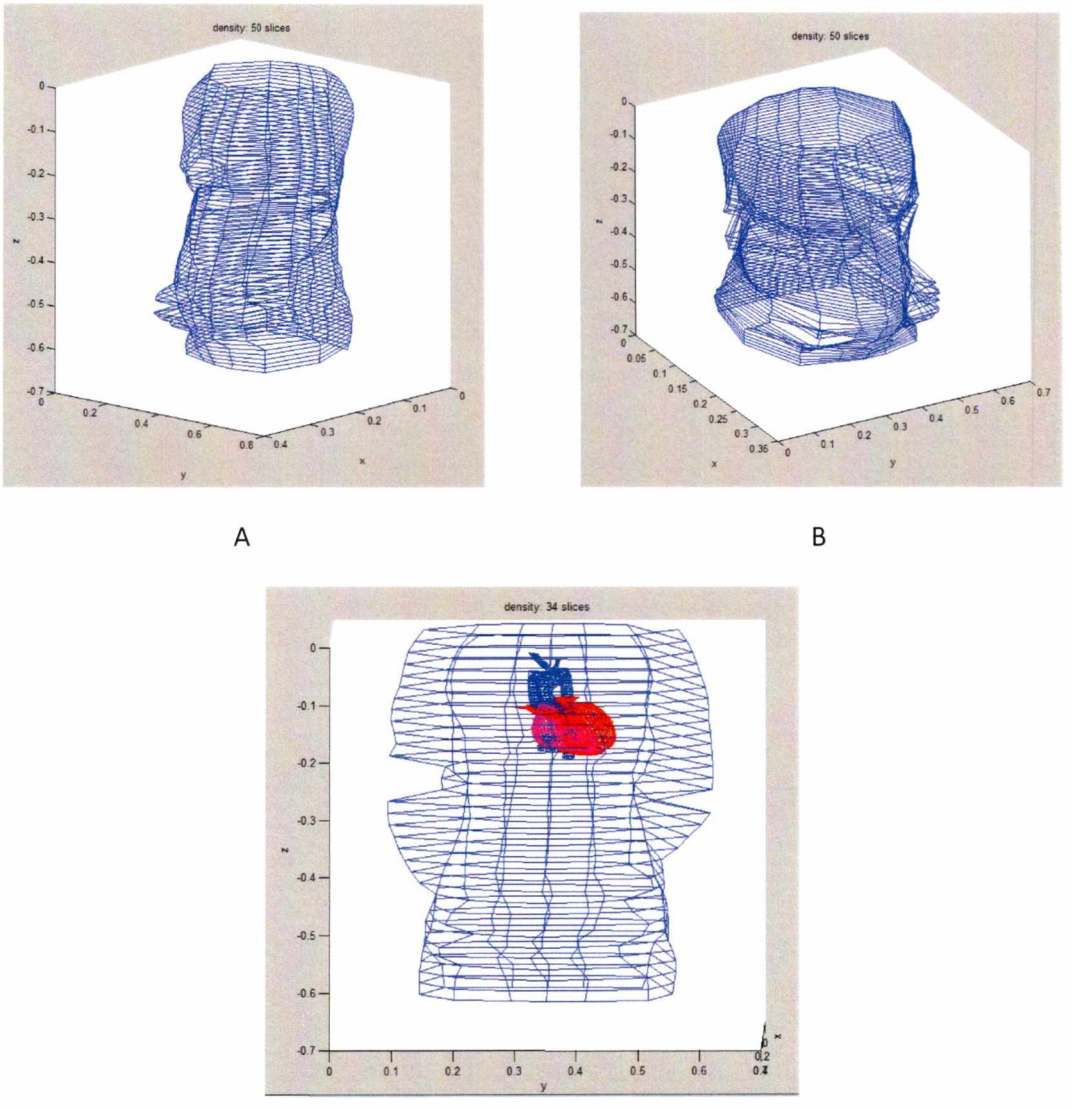


Figure 30: The heart visualized by plotting and connecting in space the points that resulted from the undersampling process (views correspond to those of figure 27)

Undersampling was also applied to the torso data by following the same technique to that of the heart. The data are visualized in Figure 31. The points in space are connected per slice and angle in this visualization as well.



A B

C

Figure 31: The torso formed by the points that remained after the undersampling process was applied, C displays a less dense visualization of the torso containing the heart

Hardware & Execution Times

For the hardware used for this work (Intel Pentium dual core CPU T4300 tuned at 2.1 Ghz and 3Gb RAM), the detection of the contours required an execution time of seconds to one minute (for the case where the user's manual intervention was required) per contour. The process of scanning the binary images in order to store the coordinates in arrays took days due to the data volume, which consisted of 647 binary images each of them containing hundreds of pixels whose coordinates were saved. Also, when trying to visualize 3D objects using the full dataset (without undersampling), the response times to just load the datasets were prohibitively slow.

Conclusions

We have created an anatomical map using images of the human torso, coming from an actual human body (cadaver). The information for constructing this map was extracted from the Virtual Human Project images using image segmentation. This segmentation was implemented using Matlab. The anatomical map can be used to form a model for the simulation of MRI. The project required the careful study of anatomy too, in order to identify the heart's anatomical structures on the raw data images. The execution times were slow when it came to manipulating (visualizing or storing) the resulting data and more advanced hardware should help in the future.

Future Work

Future directions include expanding the model by including more areas, organs or tissues. This would for the current anatomical map, regardless of the method or the algorithm. The L*a*b classification algorithms used herein for the detection of the different tissues of the heart can be used also for the detection of other tissues within the torso, such as the lungs or the fat, by simply selecting different sampling areas thus resulting in different class centers that correspond to different colors and tissues. Another topic suitable for future work is the generation of the tissue motion models. The undersampling process should be the first step towards the design of these models.

Limitations

Hardware

The computer hardware was inadequate for the task and prolonged data processing times to several days. This could be improved by purchasing more advanced hardware so as to reduce execution times. This should also allow for more reasonable data management times during the data storing and visualization.

Partial data corruption

Due to data corruption in several of the original VHP images the algorithm could not detect the torso contours for 14 images (the images are counted from 1262 to 1909 and the corrupted images are from image 1489 to 1502). So, the last contour that was correctly detected was copied and considered to be the contour of the torso for each of these corrupted images. The total number of images whose contours form the torso in 3D was 647. So, the fact that 14 of the contours were corrupted does not affect the results much.

APPENDIX

Matlab programs

Matlab's demo code: "Color based segmentation Using the L*a*b Color Space"

{

```
% This is the demo code from the Matlab documentation that uses the
L*a*b
% colorspace "Color-Based Segmentation Using the L*a*b* Color Space"
% where the algorithm for the detection of the contours was adapted
from
fabric = imread('fabric.png');
figure(1), imshow(fabric), title('fabric');
%Calculate Sample Colors in L*a*b* Color Space for Each Region
load regioncoordinates;

nColors = 6;
sample_regions = false([size(fabric,1) size(fabric,2) nColors]);

for count = 1:nColors
  sample_regions(:,:,count) =
roipoly(fabric,region_coordinates(:,1,count),...
                                        region_coordinates(:,2,count));
end

imshow(sample_regions(:,:,2)),title('sample region for red');


% Convert your fabric RGB image into an L*a*b* image using makecform
and applycform.
cform = makecform('srgb2lab');
lab_fabric = applycform(fabric,cform);



% Calculate the mean 'a*' and 'b*' value for each area that you
extracted with roipoly. These values serve as your color markers in
'a*b*' space.
a = lab_fabric(:,:,2);
b = lab_fabric(:,:,3);
color_markers = repmat(0, [nColors, 2]);

for count = 1:nColors
  color_markers(count,1) = mean2(a(sample_regions(:,:,count)));
  color_markers(count,2) = mean2(b(sample_regions(:,:,count)));
end



% For example, the average color of the red sample region in 'a*b*'
space is
```

```
disp(sprintf('[%0.3f,%0.3f]',color_markers(2,1),color_markers(2,2)));
% Classify Each Pixel Using the Nearest Neighbor Rule
color_labels = 0:nColors-1;
a = double(a);
b = double(b);
distance = repmat(0,[size(a), nColors]);
for count = 1:nColors
  distance(:,:,count) = ( (a - color_markers(count,1)).^2 + ...
                        (b - color_markers(count,2)).^2 ).^0.5;
end

[value, label] = min(distance,[],3);
label = color_labels(label);
clear value distance;
%Display Results of Nearest Neighbor Classification
rgb_label = repmat(label,[1 1 3]);
segmented_images = repmat(uint8(0),[size(fabric), nColors]);

for count = 1:nColors
  color = fabric;
  color(rgb_label ~= color_labels(count)) = 0;
  segmented_images(:,:,:,count) = color;
end

imshow(segmented_images(:,:,:,2)), title('red objects');
figure;imshow(segmented_images(:,:,:,3)); title('green objects');
figure;imshow(segmented_images(:,:,:,4));title('purple objects');
figure;imshow(segmented_images(:,:,:,5)); title('magenta objects');
figure;imshow(segmented_images(:,:,:,6)); title('yellow objects');
purple = [119/255 73/255 152/255];
plot_labels = {'k', 'r', 'g', purple, 'm', 'y'};

figure;
for count = 1:nColors
  plot(a(label==count-1),b(label==count-1),'.','MarkerEdgeColor', ...
       plot_labels{count}, 'MarkerFaceColor', plot_labels{count});
  hold on;
end

title('Scatterplot of the segmented pixels in ''a*b*'' space');
xlabel('''a*'' values');
ylabel('''b*'' values');



}
```

Creating the color markers

create_color_markers.m

{

```
% this function is used to create the color markers necessary for the
functions that detect the contours
% of the heart components and the torso it requires an image for
sampling
% in the same directory
function [color_markers] =
create_color_markers(image_num)%image_num=the image number, images
are numbered from 1262 to 1909
fn=['avf' num2str(image_num,'%02d') 'a.png']; %create the filename
according to the given image number
I=imread(fn);%read the image of the filename created
figure;imshow(I);title(fn);xlabel('select sampling points for the 1st
area ');%display
[x,y]=getpts();%manually select the points of the sampling area
coor(:,:,1)=[x,y];%save the sampling area coordinates
figure;imshow(I);title(fn);xlabel('select sampling points for the 2nd
area ');%display
[x,y]=getpts();%manually select the points of the sampling area
coor(:,:,2)=[x,y];%save the sampling area coordinates
figure;imshow(I);title(fn);xlabel('select sampling points for the 3rd
area ');%display
[x,y]=getpts();%manually select the points of the sampling area
coor(:,:,3)=[x,y];%save the sampling area coordinates
nColors = 3;%define the amount of colors
sample_regions = false([size(I,1) size(I,2) nColors]);%create array
for sample regions
%set the sample regions from the regions you have defined manually
for count = 1:nColors
  sample_regions(:,:,count) =
roipoly(I,coor(:,1,count),coor(:,2,count));
end

cform = makecform('srgb2lab'); %color transformation structure cform
, srgb2lab = standarRGB->L*a*b
lab_fabric = applycform(I,cform);%converts the color values in fabric
to the color space specified in the color transformation structure
cform i.e. here:RGB->L*a*b
% Calculate the mean 'L', 'a*' and 'b*' value for each area that you
extracted with roipoly.
% These values serve as your color markers in 'L*a*b*' space.
a = lab_fabric(:,:,1);
b = lab_fabric(:,:,2);
L = lab_fabric(:,:,3);
color_markers = repmat(0, [nColors, 2]);

for count = 1:nColors
  color_markers(count,1) = mean2(a(sample_regions(:,:,count)));
  color_markers(count,2) = mean2(b(sample_regions(:,:,count)));
  color_markers(count,3) = mean2(L(sample_regions(:,:,count)));
end
```

}

Scanning contour images for the inner points of a contour

innerpoints.m {

```
% this code was used to obtain the coordinates of the points

% of the cotours and additionally all the points inside the contours
%it requires that the appropriate directories exist
function [data_full] = innerpoints(directory)
cd(directory);%change to directory
cd('binary images');%change to sub-folder "binary images"
ex=exist('fill','dir');
if ex==7 %check if name is a folder
    cd('fill'); %change to "fill" directory
else %change to "tissue" directory otherwise
    cd('tissue');
end
d=dir('*.png');     %list all the .png files of the existing directory
filenames={d.name};%create an array containing the names of the files
start =
str2double(strtok(filenames(1,1),'abcdefghijklmnopqrstuvwxyzABCDEFGHI
JKLMNOPQRSTUVWXYZ'));%isolate the image number from the filename (1st
image )
the_end=str2double(strtok(filenames(1,numel(filenames)),'abcdefghijkl
mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'));%isolate the image number
from the filename (last image)
count=1;%counter that is used for numbering the elements in the array
zet=1262-start; % %the position in the z axis is calculated according
to the numeration of the whole set of images - the first image is
numbered 1262 so it is used as a constant
for i= start:the_end % images range
    filename=char(filenames(1,(i-start+1))); %create a character
variable containing the name of the file to be read
    I=imread(filename);%read an image named after the filename
created
    for i2=1:size(I,1)% scan the image
        for j2=1:size(I,2)
            if ~(I(i2,j2)==0) %whenever a white pixel is met
                data_full(count,1)=i2;%save its coordinates
                data_full(count,2)=j2;
                data_full(count,3)=zet;
                count=count+1;
            end
        end
    end
    zet=zet-1;%change the z axis value every time we pass to the next
image, so that eventually the points of the next slice are placed
under the already
    fprintf('\n %s \n',filename);%saved ones (placing the next slice
under the last one)
end
data_full(:,1)=data_full(:,1)*0.33;%convert from pixel to spatial
coordinates
data_full(:,2)=data_full(:,2)*0.33;
data_full(:,3)=data_full(:,3)*0.99; %replace all the z values with
their negatives so that the values of the z axis are decreasing
(placing the next slice under the last one)
data_full=data_full/1000;%convert to metres
fprintf('\nData created for %s\n',directory);%print
}
```

Save coordinates read from binary images

store_coord.m{

```
% this code is used to scan the coordinates of the binary images
% in the spacified directory, it requires that the directory
definition for
% the area (selected heart component or torso) and the sub folder 'c'
-
% contour, 'f' - fill or 't' - tissue exist

function data=store_coord(directory,ctf)
% mess=wavread('program_complete.wav');%sound message I used to alert
me when work was done
cd(directory);
cd('binary images');%check and switch to the appropriate directory
if ctf=='c'
    cd('contours');
end
if ctf=='t'
    cd('tissue');
end
if ctf=='f'
    cd('fill');
end
d=dir('*.png');    %list all the .png files of the existing directory
filenames={d.name};%create an array containing the names of the files
start =
str2double(strtok(filenames(1,1),'abcdefghijklmnopqrstuvwxyzABCDEFGHI
JKLMNOPQRSTUVWXYZ'));%isolate the image number from the filename (1st
image )
the_end=str2double(strtok(filenames(1,numel(filenames)),'abcdefghijkl
mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'));%isolate the image number
from the filename (last image)
count=1;%counter that is used for numbering the elements in the array
zet=1262-start; % %the position in the z axis is calculated according
to the numeration of the whole set of images - the first image is
numbered 1262 so it is used as a constant
for i= start:the_end %images range
    filename=char(filenames(1,(i-start+1))); %create a charcter
variable containing the name of the file to be read
    I=imread(filename);%read an image named after the filename
created
    for i2=1:size(I,1)% scan the image
        for j2=1:size(I,2)
            if ~(I(i2,j2)==0) %whenever a white pixel is met
                data(count,1)=i2;%save its coordinates
                data(count,2)=j2;
                data(count,3)=zet;
                count=count+1;
            end
        end
    end
    zet=zet-1;%change the z axis value every time we pass to the next
image, so that eventually the points of the next slice are placed
under the already
    fprintf('\n %s \n',filename);%saved ones (placing the next slice
under the last one)
end
data(:,1)=data(:,1)*0.33;%convert from pixel to spatial coordinates
```

```
data(:,2)=data(:,2)*0.33;
data(:,3)=data(:,3)*0.99;
data=data/1000;%convert to metres
% sound(mess);%%sound message I used to alert me when work was done
fprintf('\nData gathered for %s \n',directory);
}
```

Segmentation with gray-scale images

graylevel_image_segmentation.m

{

```
%the current code was adapted from Matlab's "Detecting a Cell Using
Image
%Segmentation" example, note that the contours that form the torso do
not come from this method because the results contain errors
%and additionally this method needs configuration (adding or removing
a dilation or erosion) for every single image
se90 = strel('line', 3, 90);%create structure elements that
se0 = strel('line', 3, 0);%will be used for the image
seD = strel('diamond',1);%dilation & erosion
start=1262;%set the starting image number
for image_num=start:1262 %set the ending image number
fn=['avf' num2str(image_num,'%02d') 'a.png'];%create a variable
bearing the file name following the specific pattern the images are
named after
new_fn=['avf' num2str(image_num,'%02d') 'aContour.png'];%create the
new filename the image will be saved as
I = imread(fn);%read the image named like the filename
figure;imshow(I);title('original image');%display original image -
comment to disable
I2=rgb2gray(I);%transform to gray-level
[junk threshold] = edge(I2,'sobel');%set a threshold for the mask
that will create the binary image
fudgeFactor = .5;%define the segmentation sensitivity
I2 = edge(I2,'sobel', threshold * fudgeFactor);%create binary image
% figure; imshow(I2); title('binary');%uncomment to display binary
image
I2 = imdilate(I2, [se90 se0]);%dilate image
% figure; imshow(I2);title('dilated');%uncomment to display dilated
image
I2 = imfill(I2,'holes');%fill the holes
% figure;imshow(I2);title('filled holes image');%uncomment to display
filled holes image
I2 = imclearborder(I2,8);%clear image border connectivity: 8
neighbors
% figure;imshow(I2);title('cleared border image');%uncomment to
display
I2 = imerode(I2,seD);%erode
I2 = imerode(I2,seD);%the image
I2=bwareaopen(I2,9000);%erase objects sustained by less than 9000
pixels
figure;imshow(I2);title('final');%comment to disable displaying
% I2 = bwperim(I2); %uncomment to display the
% Segout = I;       %contour on the original image
% Segout(I2) = 255; %
% figure;imshow(Segout);title('outlined original image');%
imwrite(I2,new_fn);%save the contour image as defined in the "new_fn"
fprintf( '\nImage %02d \n', image_num);%display the image number on
workspace
end
```

}

Detecting the heart area

heart_detection.m{

```
%This code is based on the demo "Color-Based Segmentation Using the
L*a*b*
%Color Space" and is used for the detection of the heart, it requires
that
%the original images and the appropriate "color_markers" files to be
in the
%current directory


se90 = strel('line', 3, 90);%create the structure
se0 = strel('line', 3, 0);%elements
% load('color_markers4.mat');%the files containing the coordinates
load('color_markers5.mat');%of the colours in L*a*b color space
% load('color_markers.mat');%several of these were created because
% load('color_markers_fat2.mat');%each may work better in a specific
image
for image_num=1464:1465%define the range of images using the
numeration of the images
filename=['avf' num2str(image_num,'%02d') 'a.png'];%read the file
names following the given pattern
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
%THE FOLLOWING PART OF COMMENTED CODE REFFERS TO AREAS THAT HAVE
ALREADY
%BEEN DETECTED AND ARE AREAS NEIGHBORS TO THE CURRENT REGION OF
INTEREST (ROI)
%UNCOMMENT TO READ A SPECIFIC CONTOUR WHEN IT CAN BE USED TO EXCLUDE
A PART
%DURING THE DETECTION OF OUR ROI, CONTINUES ON LINE 116
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
% filename2=['avf' num2str(image_num,'%02d') 'SuperiorVenaCava.png'];
% SVC=imread(filename2);
% SVC=logical(SVC);
% SVC=imfill(SVC,'holes');
% filename3=['avf' num2str(image_num,'%02d') 'Carotide.png'];
% Car=imread(filename3);
% Car=logical(Car);
% Car=imfill(Car,'holes');
% filename4=['avf' num2str(image_num,'%02d') 'Aorta.png'];
% A=imread(filename4);
% A=logical(A);
% A=imfill(A,'holes');
% filename5=['avf' num2str(image_num,'%02d') 'LeftAtrium.png'];
% LA=imread(filename5);
% LA=logical(LA);
% LA=imfill(LA,'holes');
```

```
% filename6=['avf' num2str(image_num,'%02d')
'RightPulmonaryArtery.png'];
% RPA=imread(filename6);
% RPA=logical(RPA);
% RPA=imfill(RPA,'holes');
% filename7=['avf' num2str(image_num,'%02d') 'PulmonaryTrunk.png'];
% PT=imread(filename7);
% PT=logical(PT);
% PT=imfill(PT,'holes');
% filename8=['avf' num2str(image_num,'%02d') 'RightAtrium.png'];
% RA=imread(filename8);
% RA=logical(RA);
% RA=imfill(RA,'holes');
% filename9=['avf' num2str(image_num,'%02d')
'LeftVentricleFlesh.png'];
% LVF=imread(filename9);
% LVF=logical(LVF);
% LVF=imfill(LVF,'holes');
% filename10=['avf' num2str(image_num,'%02d') 'RightVentricle.png'];
% RV=imread(filename10);
% RV=logical(RV);
% RV=imfill(RV,'holes');
I=imread(filename);%read the image name
fabric = I;%save the image read to a new variable
fab2=rgb2gray(fabric);%convert the image to grayscale
% figure; imshow(fabric); title(filename);%uncomment to display
%Calculate Sample Colors in L*a*b* Color Space for Each Region
nColors = 3;%the amount of different colors to be detected
cform = makecform('srgb2lab');%creates the color transformation
structure  that defines the color space conversion specified, here
srgb2lab= from rgb to L*a*b
lab_fabric = applycform(fabric,cform);%convert the color values of
the image to what the "cform" defines i.e. here: rgb to L*a*b
a = lab_fabric(:,:,1);%Calculate the mean 'L ''a*' and 'b*'
b = lab_fabric(:,:,2);%value for each area that you extracted with
roipoly.
L = lab_fabric(:,:,3);% These values serve as your color markers in
'a*b*' space.
%
disp(sprintf('[%0.3f,%0.3f,%0.3f]',color_markers4(2,1),color_markers(
2,2),color_markers(2,3)));
% Classify Each Pixel Using the Nearest Neighbor Rule
color_labels = 0:nColors-1;%Create an array that contains your color
labels
a = double(a);%Initialize matrices to be
b = double(b);%used in the nearest neighbor
L = double(L);%classification.
distance = repmat(0,[size(a), nColors]);
%perform classification
for count = 1:nColors
  distance(:,:,count) = ( (a - color_markers5(count,1)).^2 + ...
                        (b - color_markers5(count,2)).^2 + (L -
color_markers5(count,3)).^2 ).^0.5;
end
[value, label] = min(distance,[],3);
label = color_labels(label);
clear value distance;
%Display Results of Nearest Neighbor Classification
% The label matrix contains a color label for each pixel in the
fabric
% image. Use the label matriz to separate objects in the original
```

```
% fabric image by color.
rgb_label = repmat(label,[1 1 3]);
segmented_images = repmat(uint8(0),[size(fabric), nColors]);

for count = 1:nColors
  color = fabric;
  color(rgb_label ~= color_labels(count)) = 0;
  segmented_images(:,:,:,count) = color;
end
%display the images with the classified colors
figure;imshow(segmented_images(:,:,:,1));
title(filename);xlabel('segimages1');
figure;imshow(segmented_images(:,:,:,2));
title(filename);xlabel('segimages2');
figure;imshow(segmented_images(:,:,:,3));
title(filename);xlabel('segimages3');
%manually set the frame for the ROI of the heart area
[x y]=getpts();
x=round(x);
y=round(y);
% x=[1131;1196];%sometimes the ROI frame needs to stay fixed for
% y=[306;376];%more iterations (i.e. images)
% erase everything outside the ROI boundaries
BWs(1:y(1),1:2048)=0;
BWs(y(2):1216,1:2048)=0;
BWs(1:1216,x(2):2048)=0;
BWs(1:1216,1:x(1))=0;
I3=zeros(size(I,1),size(I,2));%create an empty(black) binary image
%scan the appropriate "segmented_images" image to create the binary
image
for i=y(1):y(2)
    for j=x(1):x(2)
      if   (segmented_images(i,j,:,2)==0)  %%set the appropriate
condition to create the binary image
%       if  ~(segmented_images(i,j,:,2)==0)%%for example by using the
black
                                            %%or not black areas of
the suitable "segmented_images" image

          I3(i,j)=1;
      end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%READ COMMENTS ON LINES 13-16:UNCOMMENT THE CORRESPONDING PART TO
ACTIVATE THE EXCLUSION
%OF A PART OF THE CURRENT ROI USING AN ALREADY DETECTED AREA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(A(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
```

```
% end
% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(RPA(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
% end
% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(PT(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
% end
% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(SVC(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
% end
% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(RA(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
% end
% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(LVF(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
% end
% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(RV(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
% end
% for i=1:1216
%     for j=1:2048
%         if (I3(i,j)==1)&&(LA(i,j)==1)
%             I3(i,j)=0;
%         end
%     end
% end
% I3=imclearborder(I3,4);%clear structures connected to the image
border here:connectivity 4 neighbors
I3=imclearborder(I3,8);%clear structures connected to the image
border here:connectivity 8 neighbors
I3=bwareaopen(I3,300);%erase structures sustained by less than 300
pixels
% figure;imshow(I3);title('Cleared border');%display
I3=imdilate(I3,[se0 se90]);%dilate image
% figure;imshow(I3);title('Dilated');%display
I3=imfill(I3,'holes');%fill holes
figure;imshow(I3);title('filled holes');%display
```

```
I3=imerode(I3,[se0 se90]);%erode image
I3=imerode(I3,[se0 se90]);%may need to add or
I3=imerode(I3,[se0 se90]);%subtract an erosion action
I3=edge(I3,'canny');%apply edge detection
figure;imshow(I3);title(filename);xlabel('final');%display
new_filename=['avf' num2str(image_num,'%02d')
'Heart_Fat.png'];%create the filename the contour will be saved as
% imwrite(I3,new_filename);%%uncomment to save the created contour
fprintf( '\nImage %02d \n', image_num);%display the image number on
the workspace
end


}
```

Segmentation with true color images

torso_detection.m{

```
se90 = strel('line', 3, 90);%create structure elements
se0 = strel('line', 3, 0);%for dilation/erosion
load('color_markers.mat');%load the "color_markers" file
for image_num=1262:1262%selec the images range
filename=['avf' num2str(image_num,'%02d') 'a.png'];%create the name
to read file following the  pattern
I=imread(filename);%read file
fabric = I;%assign the image read to a new variable ("fabric")
fab2=rgb2gray(fabric);%convert to graylevel
nColors = 3;%amount of colors to be detected
cform = makecform('srgb2lab');%color transformation structure cform ,
srgb2lab = standarRGB->L*a*b
lab_fabric = applycform(fabric,cform);%converts the color values in
fabric to the color space specified in the color transformation
structure cform i.e. here:RGB->L*a*b
% Calculate the mean 'L', 'a*' and 'b*' value for each area that you
extracted with roipoly.
% These values serve as your color markers in 'L*a*b*' space.
a = lab_fabric(:,:,1);%Initialize matrices to be
b = lab_fabric(:,:,2);%used in the nearest neighbor
L = lab_fabric(:,:,3);%classification.
% Classify Each Pixel Using the Nearest Neighbor Rule
color_labels = 0:nColors-1;%Create an array that contains your color
labels
a = double(a);
b = double(b);
L = double(L);
distance = repmat(0,[size(a), nColors]);
%perform classification
for count = 1:nColors
  distance(:,:,count) = ( (a - color_markers(count,1)).^2 + ...
                        (b - color_markers(count,2)).^2 +( L-
color_markers(count,3)).^2 );
end

[value, label] = min(distance,[],3);
label = color_labels(label);
clear value distance;
%Display Results of Nearest Neighbor Classification
% The label matrix contains a color label for each pixel in the
fabric
% image. Use the label matrix to separate objects in the original
% fabric image by color.
rgb_label = repmat(label,[1 1 3]);
segmented_images = repmat(uint8(0),[size(fabric), nColors]);

for count = 1:nColors
  color = fabric;
  color(rgb_label ~= color_labels(count)) = 0;
  segmented_images(:,:,:,count) = color;
end
%display the images with the classified colors %(comment to skip)
figure;imshow(segmented_images(:,:,:,1));
title(filename);xlabel('segimages1');
figure;imshow(segmented_images(:,:,:,2));
title(filename);xlabel('segimages2');
```

```
figure;imshow(segmented_images(:,:,:,3));
title(filename);xlabel('segimages3');
V=segmented_images(:,:,:,3);%select one of the images containing the
classified colors
V=rgb2gray(V);%transform to graylevel
V=imclearborder(V,8);%clear objects connected to the image border
(connectivity 8 neighbors)
V(1020:1216,1:2048)=0;%erase a specific area corresponding to the
lower part of the image that contains numeration & labels
V=bwareaopen(V,10000);%erase objects sustained by less than 10000
pixels
V=imfill(V,'holes');%fill holes
%scan original image to erase the parts that are not included in the
%detected torso area  (projection of corresponding original image)
for ti=1:1216
    for tj=1:2048
        if V(ti,tj)==0
            I(ti,tj,:)=0;
        end
    end
end
figure;imshow(I);title('projection of original image');%project
contour results corresponding on the original image %(comment to
skip)
BWoutline = bwperim(V);
V=edge(V,'canny');%obtain the contour of the area
new_filename=['avf' num2str(image_num,'%02d') 'aIC.png'];%create the
filename the contour image will be saved as
% imwrite(V,new_filename);%save image %(uncomment for usage)
fprintf(filename);fprintf('\n done\n'); %display message on workspace

end



}
```

Undersampling – code version without parameters

undersV2.m{

```
% this function is used to execute an under sampling to the specifed
binary
% images (the user must select by changing the code), it requires the
% binary images exist in the same directory


count=1;%counter used in the array numeration
start=1406;%starting image
zet=1262-start; % z axis is defined by using the number of the first
image as a constant so that all resulting data has a common point of
reference
% message=wavread('program_complete.wav');%alert message
for i= start:1406% images range
    amount=0; % counter
    filename=['avf' num2str(i,'%02d') 'RightAtrium.png'];%create a
file name according to the pattern depending on the directory (change
for other tissues)
    I=imread(filename);%read filename
%     I(1:430,1:2048)=0;
    figure;imshow(I);title(filename);%display
%     uncomment following to erase a manually selected frame
%     zoom(2);
%     [x1,y1]=getpts();
%     I(y1(1):y1(2),x1(1):x1(2))=0;
%     hold on;imshow(I);xlabel('erased');
    s=regionprops(I,'centroid'); %locate centroid - mass center
    x=round(s(numel(s),1).Centroid(1,1));%round the centroid
coordinates to match a pixel's
    y=round(s(numel(s),1).Centroid(1,2));
    clear('s');%absolve variable
%     hold on;plot(x,y,'y*');hold on; %display - plot center
(uncomment to use)
    fprintf('\n Center found : %d %d',x,y);%display message on
workspace
    d=16;   %initialize distance variable
    a=30;   %set angle of rotation (change to redefine)
    gtp=180/a; %convert from degrees to find the corresponding arch
distance step
    for a=0:pi/gtp:(2*pi-(pi/gtp))  %arch distance step
%         figure;imshow(I);title(filename);title(a);
%display(uncomment to
%         activate)
       dis=15;%initialize variable
        h1=0;%initialize variable
        h2=0;%initialize variable
        for r=1:90 % radius length for the ezpanding pixel lines
            nx=x+r*cos(a);%apply angles
            ny=y+r*sin(a);%for executing rotation
            nx=round(nx); %round to match
            ny=round(ny); %pixel coordinates
%             hold on; plot(nx,ny,'y');hold on;%uncomment to activate
%             display
            for ii=ny-7:ny+7 %inner loop (for every pixel of the
expanding lines)
                for jj=nx-7:nx+7
```

```
                    if ~(I(ii,jj)==0) %calculate the distance of
every white pixel met
                    d=distance([ii,jj],[ny,nx]);
                    end
                    if (d<=dis) %the undersampling saves the
coordinates
                        dis=d; %of the pixel that abstains the least
from
                        h1=ny; %the contour pizel (for the case they
do not match)
                        h2=nx;
                    end
                end
            end
%             hold on;plot(h2,h1,'y');hold on;%uncomment to plot
        end
%         hold on;plot(h2,h1,'y');hold on;%ncomment to plot
        amount=amount+1; %counter for the iterations executed
        fprintf('\n Image No %d found %d point(s) %d
%d\n',i,amount,h2,h1);%display message on workspace
       Pulmonay12_1_Sp(count,1)=h1;%strore coordinates in an array
       Pulmonay12_1_Sp(count,2)=h2;
       Pulmonay12_1_Sp(count,3)=zet;
        count=count+1;
    end
    zet=zet-1;%set the z axis positioning for the nezt contour
(slice)
    fprintf('\nImage No %d found %d points\n',i,amount);%display
message on workspace
end
Pulmonay12_1_Sp(:,1)=Pulmonay12_1_Sp(:,1)*0.33;%convert from pixel to
spatial coordinates
    Pulmonay12_1_Sp(:,2)=Pulmonay12_1_Sp(:,2)*0.33;
    Pulmonay12_1_Sp(:,3)=Pulmonay12_1_Sp(:,3)*0.99;
    Pulmonay12_1_Sp=Pulmonay12_1_Sp/1000; %convert to metres
% sound(message);%sound alert message


}
```

Undersampling – code version with parameters (function)

unders.m{

```
% This function takes as inputs the name of the direcory that
contains
% the binary images (contours) that have resulted from previous
% procedures, the angular step according to which the expanding lines
% will be extended, the display activation, the automatic (or) not
center
% locating and optionally the starting and ending image (range) for
the
% execution in case the user wants a custom execution range instead
of
% the whole set.It executes an undersampling so that the final output
% contains the contour formed by a number of points less than the
% initial. The user defines an angular step and straight pixel lines
% are extended from the center inside of the contour until they meet
a
% pixel belonging to the contour. This procedure is repeated for
every
% angle until a 360 degrees scan is completed for every picture of
the
% directory (or the defined range).
% Example: data=intrplte('Aorta',30,0,0);
% Explanation: execution in the "Aorta" folder, for an angular step
of 30 degrees, display off, locating center automatically.
% Example2: data=intrplte('Aorta',30,1,0);
% Explanation: execution in the "Aorta" folder, for an angular step
of 30 degrees, display on, locating center automatically.
% Example3: data=intrplte('Aorta',30,1,1);
% Explanation: execution in the "Aorta" folder, for an angular step
of 30 degrees, display on, locating center manually (user).
% Example4: data=intrplte('Aorta',10,0,0,1440,1450);
% Explanation: execution in the "Aorta" folder, for an angular step
of 10 degrees, display off, locating center automatically,
% from the image numbered 1440 until image 1450 and additionally the
user must manually set a frame for erasing on the image
% so that only one contour remains (this is aplied to images
containing more than one contours so that the automatic center
% locating is executed correctly)
```

```
function [data] =
unders(directory,angle,displ,autoloc,starting_im,ending_im)
fprintf('\n Executing undersampling in "%s" folder for %d degrees
angle\n',directory,angle);%display the execution directory and
angular step
if ~(autoloc==1)%check if the user wants to locate automatically the
center
    manual=0;
    fprintf('\n Locating center automatically\n');
end
if (autoloc==1)%check if the user wants to locate manually the center
```

```
    manual=1;
    fprintf('\n Engaged manual center locating\n');
end
if (nargin>4)%check if the user wants to manually remove a frame
   manual=2;
   fprintf('\nEngaged manual frame removal for images %d -
%d\n',starting_im,ending_im);
end
if (nargin>6)%check number of arguments
    error('\nNo more than 6 arguments required\n');
end
    cd(directory);      %change to directory
    cd('binary images');
    cd('contours');% change to the folder containing the contour
images
d=dir('*.png');          % list the names of the directory ending to
."png" i.e. the images
filenames={d.name};      %create an array with the list
start =
str2double(strtok(filenames(1,1),'abcdefghijklmnopqrstuvwxyzABCDEFGHI
JKLMNOPQRSTUVWXYZ'));%extract the starting image number from the file
name
the_end=str2double(strtok(filenames(1,numel(filenames)),'abcdefghijkl
mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'));%extract the ending image
number from the file name
count=1;
if (manual==2) %usecase where the image range is set by the user
    range=starting_im-start; %calculate a variable that is used to
skip to the right filename in the list (array) according to the range
set
    start=starting_im;  %in this usecase the starting image is set by
the user as an argument
    the_end=ending_im;  %also the ending image
end
zet=1262-start; %the position in the z axis is calculated according
to the numeration of the whole set of images - the first image is
numbered 1262 so it is used as a constant
for i= start:the_end % range
    amount=0; % just a counter
    filename=char(filenames(1,i-start+1)); %create a charcter
variable containing the name of the file to be read
    I=imread(filename); %read an image named after the filename
created
    if (manual==1)  %case when the manual intervetion for the center
locating is engaged
        figure;imshow(I);title(filename);xlabel('Right clic on the
center of the contour');%display
        %manually select the center
        zoom(2);
        [x,y]=getpts();
        x=round(x);
        y=round(y);
    elseif (manual==2)  %case when the manual intervetion for the
removal of an area is engaged
        filename=char(filenames(1,i-start+range+1));%get file name
from the containing struct
        figure;imshow(I);title(filename);xlabel('Select the frame you
want to erase');%display
        %manually select a frame to remove
        zoom(2);
        [x1,y1]=getpts();
```

```
        I(y1(1):y1(2),x1(1):x1(2))=0;
        hold on;imshow(I);xlabel('erased');
        s=regionprops(I,'centroid'); %locates the centrer
("centroid") of a given binary image
        x=round(s(numel(s),1).Centroid(1,1));%acquire the x
coordinate from the structure element created by "regionprops"
        y=round(s(numel(s),1).Centroid(1,2));%acquire the y
coordinate from the structure element created by "regionprops"
        clear('s');%absolve variable
    else
        s=regionprops(I,'centroid'); %locates the centrer
("centroid") of a given binary image
        x=round(s(numel(s),1).Centroid(1,1));%acquire the x
coordinate from the structure element created by "regionprops"
        y=round(s(numel(s),1).Centroid(1,2));%acquire the y
coordinate from the structure element created by "regionprops"
        clear('s');%absolve variable
    end
    if (displ==1) %activated when the display option is activated
        figure;imshow(I);title(filename);%display
        hold on;plot(x,y,'o');hold on;%display
    end
    fprintf('\n Center found : %d %d\n',x,y);
    d=16;   %initialize with a value
    gtp=180/angle;
    for a=0:pi/gtp:(2*pi-(pi/gtp))  %angular step
    if (displ==1)%activated when the display option is activated
       figure;imshow(I);title(filename);title(a);
    end
        dis=15;%initial value
        h1=0;%initial value
        h2=0;%initial value

            pr=180;
        for r=1:pr % pixel radius
            nx=x+r*cos(a);% is used to
            ny=y+r*sin(a);%change the angle
            nx=round(nx); %the values resulting from the "cos" and
"sin"
            ny=round(ny); %functions need to be rounded in order to
match values valid for a pixel
            if (displ==1)   %activated when the display option is
activated
                hold on; plot(nx,ny,'y');hold on;
            end
                for ii=ny-7:ny+7 %inner loop (for every single point
of the extending pixel lines)
                for jj=nx-7:nx+7
                    if ~(I(ii,jj)==0)
                    d=distance([ii,jj],[ny,nx]);
                    end
                    if (d<=dis) %the undersampling process saves the
points
                        dis=d; %with the less distance from the
extending pixel line
                        h1=ny; %so in case there is not a complete
match the nearest point
                        h2=nx;  %to the contour is saved
                    end
                end
                end
```

```
            if (displ==1)%activated when the display option is
activated
                hold on;plot(h2,h1,'y');hold on;
            end
        end
            if (displ==1)%activated when the display option is
activated
                hold on;plot(h2,h1,'y');hold on;
            end
        amount=amount+1; %counter
       data(count,1)=h1;%create array with the points coordiantes
       data(count,2)=h2;
       data(count,3)=zet;
        count=count+1;
    end
    zet=zet-1;  %change the z axis value every time we pass to the
next image, so that eventually the points of the next slice are
placed under the already
    fprintf('\nImage No %d found %d points\n',i,amount);%saved ones
(placing the next slice under the last one)
end
    if(displ==1)%activated when the display option is activated
    close all;
    end
    data(:,1)=data(:,1)*0.33;%convert from pixel to spatial
coordinates
    data(:,2)=data(:,2)*0.33;
    data(:,3)=data(:,3)*0.99;
    data=data/1000; %convert to metres
fprintf('\nData created for %s for %d degrees
angle\n',directory,angle);%print



}
```