Medical image analysis - assignment4

Joel Ottosson

December 2020

1 Introduction

In the medical field there are many standards that are important to get used to. One of them are the DICOM file format. It is a way to store medical images with the important information surrounding the image. This assignment will focus on how DICOM files are structured and how to read them. As a final part, we will use the reader on a set of DICOM files to try and make segmentations using the *fast marching method*.

Part 1 - DICOM-reader

The first part of this assignment is about how medical images can be stored and restored. A common method is to use the *Digital Imaging and Communications in Medicine* (DICOM) format. A DICOM-file mainly consists of two parts. The first part stores information about the image in what I will call *data elements*. It can be information like the size of the image, settings that was used when taking the image or information about the patient which the image was taken of. This information is stored binary with each data point consisting of a tag, the type of the data, length of data, and finally the data itself. Following this image information we have the second part, the image itself. The image can then be extracted from the file given the previous set of data elements. The first part of this assignment is how one may go about writing such a DICOM-reader.

Data elements

The first thing that our DICOM-reader needs to do is to be able to read the data elements. For this we need to specify the tag, the data type, and data length of each data element that we are looking for. This will give us a table that the reader will use as a lookup table while reading the file. The table used for this assignment is given in table 1.

Extracting the image

After extracting the data elements we will use them to extract the image. First we need to find the image in our file. This is done via the data element *Start*

Name	Tag	Data type	Default value
Transfer Syntax UID	0002,0010	char	1.2.840.10008.1.2.1.
Start Of Pixel Data	7fe0.0010	uint16	[]
Bits Allocated	0028,0100	uint16	0
Rescale Slope	0028,1053	num	1
Slice Thickness	0018,0050	num	0
Spacing Between Slices	0018,0088	num	0
Rows	0028,0010	uint16	1
Columns	0028,0011	uint16	1
Pixel Spacing	0028,0030	num	1
Bits Stored	0028,0101	uint16	16
Rescale Intercept	0028,1052	num	0
Pixel Data	7ef0,0010	char	'EMPTY'
Image Position Patient	0020,0032	num	[0,0,0]

Table 1: Settings used in the DICOM-reader's lookup table

of Pixel Data. *Bits Allocated* then tells us how many bits each pixel occupies, and *Rows* and *Columns* tells us how they should be organized (remember that DICOM goes row by row). We now have a matrix with the stored values. It is often the case that stored image values and the actual image values are different. The reason for this is that the true image values might be negative or span a wide set of numbers which is difficult to store. To get from the stored values (SV) to the real once (RV), the reader uses the data elements *Rescale Slope* (RS) and *Rescale Intercept* (RI) which then gives RV as $\mathrm{RV} = \mathrm{RS} * \mathrm{SV} + \mathrm{RI}$. We now have our true image. In this assignment, the DICOM-reader was tested on two images which can be seen in figure 1. The intensities in CT images are given in Hounsfield scale (HU). HU for air is -1000 and 100 - 300 for soft tissue. The left image in figure 1 shows the value out side the body (air) and a value from the heart, and both seems to be at least somewhat plausible.

1.1 3D image reader

Sometimes medical images are 3-dimensional. In these cases the image is stored as a stack of DICOM images. To extract the 3-dimensional image we simply extract the image from each file and stack them on top of each other, and also storing the information for each slice in a stack. This gives us our final DICOM reader. A reader which takes a folder with DICOM files as input and outputs a stack of information files (just one if it is a 2D image) and a 3D matrix.

The reader implemented in this assignment is rudimentary and is based on some restrictive assumptions. It assumes that the there are only one set of DICOM images in a folder, and it assumes that those files are in order. This is a problem if the files are indexed as 'image1', 'image2',...,'image10',... since 'image10' is now before 'image2' according to the reader. The reader also defines the dimensions of the 3D image according to the size of the first file. This is

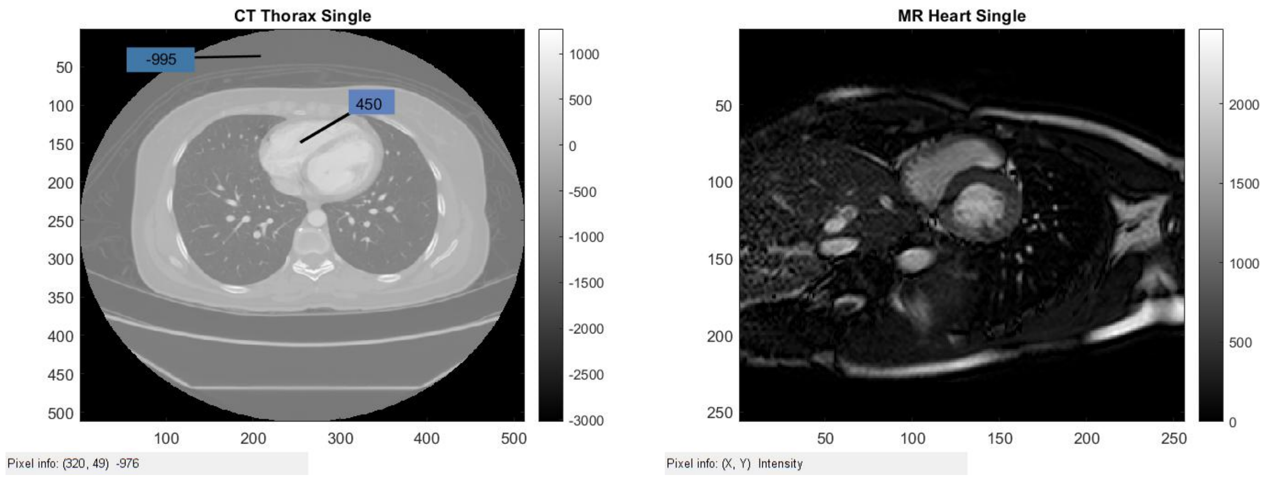


Figure 1: Examples of image extractions done by the DICOM reader. The values displayed in the left image are the calculated real values.

problematic if the files in the folder is of different sizes. Another issue is with the scaling. There are alternative ways of calculating the height of the stack. One way is the way used in this reader when the data element *Image Position Patient* is used, but one can also use the elements *Slice Thickness* and *Spacing Between Slices*. Some files might have one set of elements but not the other and then the reader will fall back to its default values.

Part 2 - Image presentation

We will now look at how medical images can be presented. Since it is not possible to show a 3D image as is, we have to choose a slice of it for visualization. There are three standard views that are being used, *transversal*, *coronal* and *sagitall*. The transversal view is seen from the feet with the chest in the top part of the image. The Coronal view is seen from the front with the head in the top part of the image. Finally, the sagitall view is seen from the left side of the patient with the head in the top part of the image. To be able to use these images, we need to equip them with measurements. Here the tags *Pixel Spacing* and *Image Position Patient* will be of use. Pixel Spacing tells us how far it is between each pixel in mm, and Image position patient tells us witch coordinates the first pixel in the image has. If one take the difference between the first an last slice of the image position patient element, we get the distance between the first and last slice in mm.

In this assignment, two 3D images are at hand. One 3D thorax image witch in figure 2 can be seen visualized in all the three standard planes. The slices shown are simply the center most slice. The second image is of the carotid

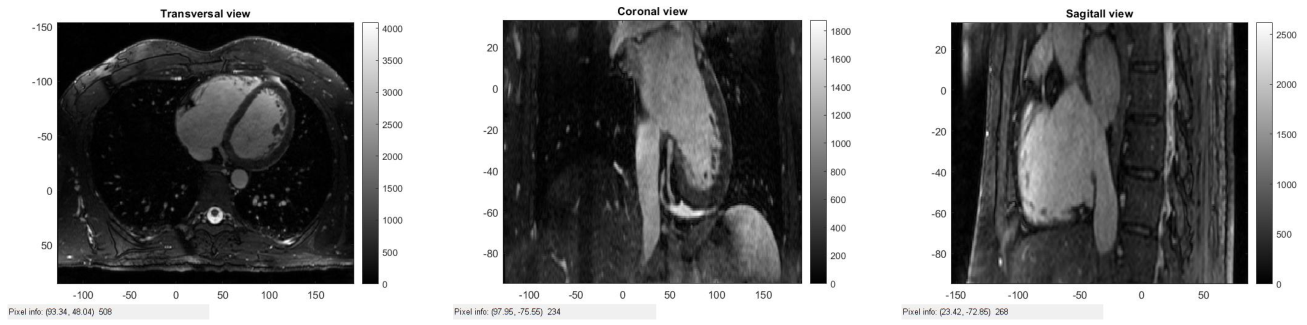

Transversal view
Coronal view
Sagitall view
Pixel info: (93.34, 48.04) 508
Pixel info: (97.95, -75.55) 234
Pixel info: (23.42, -72.85) 268

Figure 2: Transvesal (left), coronal (middle) and sagitall (right) view of the thorax. The axes are in mm.

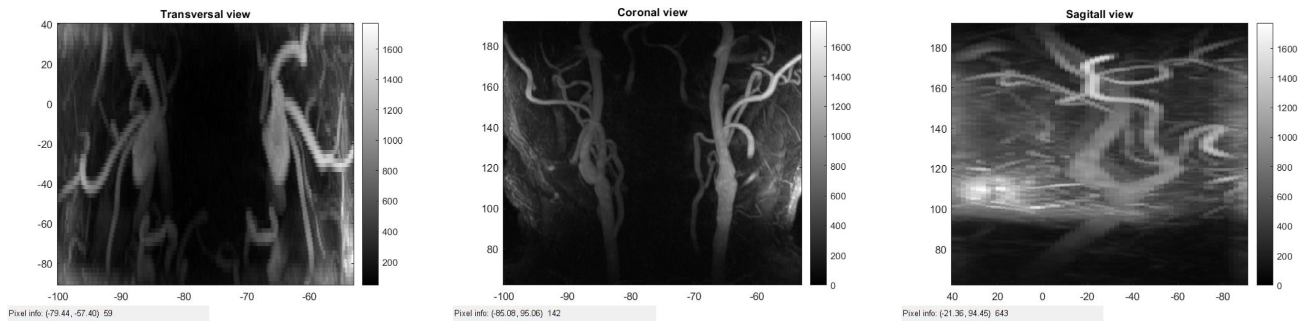

Transversal view
Coronal view
Sagitall view
Pixel info: (-79.44, -57.40) 59
Pixel info: (-85.08, 95.06) 142
Pixel info: (-21.36, 94.45) 643

Figure 3: Transversal (left), coronal (middle) and sagitall (right) view of the carotid arteries. The axes are in mm.

arteries and can be seen in figure 3. Again, the image is shown in all of the standard planes but this time it is not a slice that is shown but the maximal pixel intensity, where the maximum are taken over all pixels in the normal direction of the current plane. For reference the thorax image is 127 mm from bottom to the top, 320 mm from left to right, and 239 mm back to front. The same measurements for the carotid image are 129 mm, 47 mm, and 129 mm.

Part 3 - Segmentation

Now when we have a functioning DICOM reader we want to use it. A common thing when dealing with images is image segmentation. There are many segmentation algorithms but in this assignment we are focusing on the *Fast Marching method*. A simplified way of thinking of the method is to imagine a set of marching soldiers. Given a point in the image, the soldiers march radially outwards. The segmentation is then set as the region that the soldiers manage to cover within a set *arrival time*. However, the image is in a sense a landscape were it is slower to march in some regions and faster in others, or in other words, there is a speed corresponding to each pixel, a *speed map*. This means that we have a region (a segment) which encompass all points reached given a starting point, the speed map and an arrival time. The problem then is just to find a suitable speed map.

It is a good idea to use the knowledge about the starting pixel in our speed map creation, the intensity being the most obvious one. We could then say that we want a speed map that has a heigh speed for pixels with similar intensity and a slow speed for pixels with a very different intensity. My first approach with this knowledge was to use the logistic function given in equation 1.

$$\mathrm{SPEED}(d = |I_s - I_p|) = a_1 \mathrm{Logistic}\big(-a_2(d - a_2)\big) \quad \mathrm{Logistic}(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Here I_s is the intensity of the starting pixel, I_p is the intensity of the pixel which speed is being calculated, d is the absolute distance in terms of intensities between the start and current pixel, and a_i are parameters that can be altered (in my code they are scaled to be useful in the interactive app). The logistic function creates two plateaus that are smoothly connected where in this case pixels with a similar intensity as the starting pixel will be on the higher plateau (high speed) and pixels with a different intensity will be at the lower plateau. The parameter a_1 adjusts the height difference between the plateaus, a_2 adjusts the slope connecting the plateaus, and a_3 adjusts at what intensity distance the current pixel can be before ending up on the lower plateau. In the extreme, an infinite a_2 makes the function work as a threshold filter with threshold a_3. The final segmentation will be all the points that are connected with the starting pixel after the filtering. Conversely, if a_2 is very small, the function works similar to a linear function. Applying this speed image to the images results in the segmentation in figure 4.

The three first segmentations seem to have gone quite well, note also how they all have the almost exact settings. The forth one, however, is still quite off. If you compare the segmentations to the corresponding speed images you can see why. As mentioned before, this type of speed image works like a threshold filter, and each of the segmentations seems to align quite well with one of the red segments in the speed image. This is including the knee segmentation which is darker in the lower part of the bone and hence the speed there is slow which is the reason for the bad segmentation.

Another approach I tried was to use the normal distribution as my speed map. The equation for this approach can be found in equation 2. Here the adjustable parameters are the scaling of the normal distribution as well as the standard deviation. This method had very similar results as the logistic version, including the bad segmentation of the knee's femur bone.

$$\mathrm{SPEED}(d = |I_s - I_p|) = a_1 \frac{1}{\sqrt{2\pi a_2^2}} \exp\left(-\frac{d^2}{2a_2^2}\right) \tag{2}$$

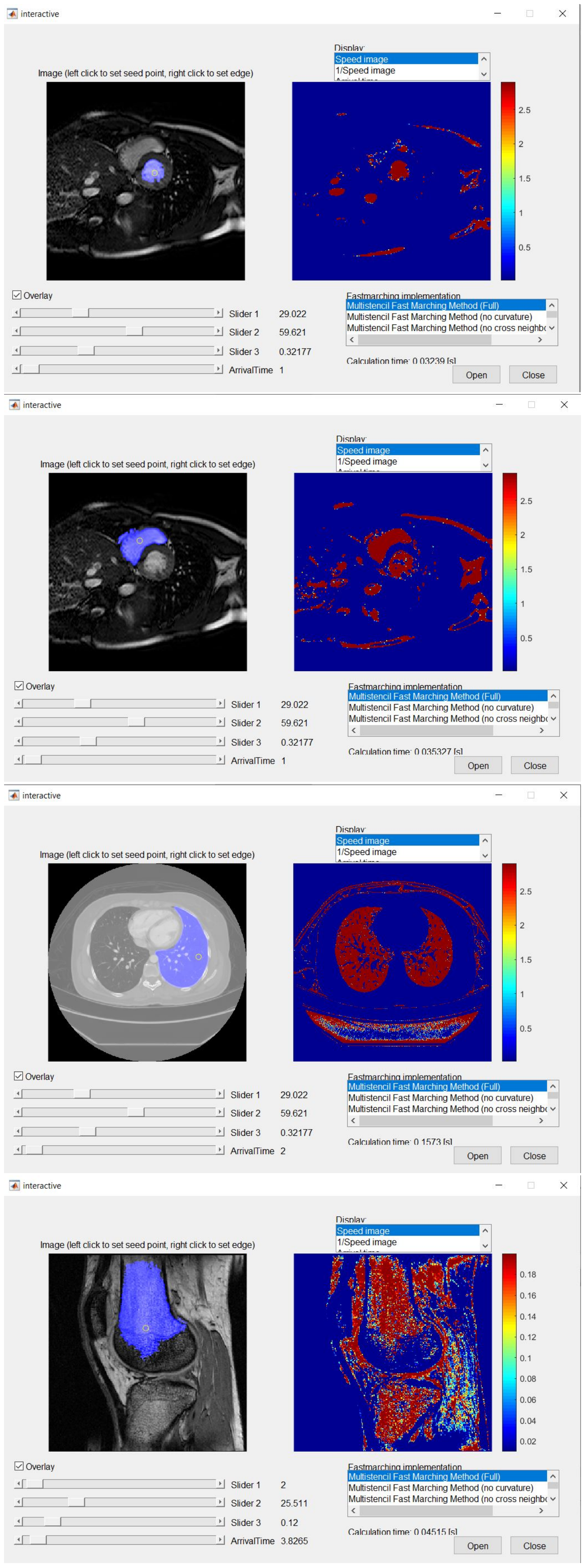
My next attempt was to involve derivatives. The idea was that the dark edge that surrounds the bone should work as a wall. The equation used for this was the simply $\left(\frac{a_1}{a_2+a_3D}\right)^2$ where D is the derivative in that pixel. The problem with this method was that there is a part of the bone which does not end with a black boarder and the hence the segment started spreading out through that small passage. This can be seen in Figure 5 which shows the progression of the segment with increasing arrival time.

I tried to combine some of the methods above, as well as blurring the original image. In the end I was not able to improve the segmentation further.

Some final words on this algorithm regarding speed. The fast marching method has a time complexity of $N\log(N)$ and is hence scaling fairly well. You could also making the algorithm faster to further push down the speed by changing some of the options. I timed the algorithm for some of the images, the results can be found in table . You can see that it starts to get quite slow at around 2000x2000 pixels (4 mega pixels). A few seconds make the algorithm impractical to use in practise for finding good values for the adjustable parameters but still not bad enough to not be used as a segmenter.

Image name	size	Time (full)	Time (limited)	Time (simplest)
MR-heart-single	256,256	0.04	0.3	0.01
MR-knee-single	256,256	0.05	0.04	0.01
CT-thorax-single	512,512	0.2	0.1	0.04
CR-pacem	1965,1531	3	2	0.5

Table 2: Averaged time for each algorithm on the different images. Time unit is seconds. The column 'Limited' is refering to the algorithm without the cross and neighbouring options set to false.


interactive
Image (left click to set seed point, right click to set edge)
Display
Speed image
1/Speed image
Overlay
Fastmarching implementation
Open
Close

Multistencil Fast Marching Method (Full)
Multistencil Fast Marching Method (no curvature)
Multistencil Fast Marching Method (no cross neighbo

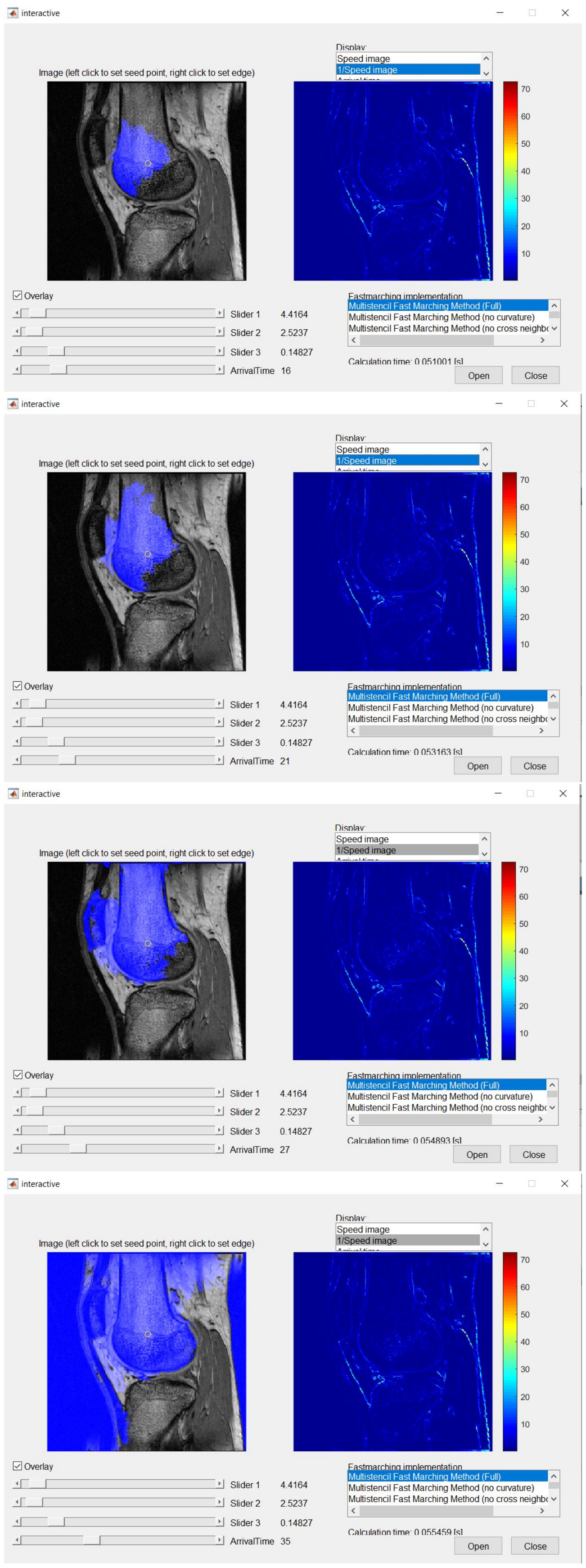
Slider 1 29.022
Slider 2 59.621
Slider 3 0.32177
ArrivalTime 1
Calculation time: 0.03239 [s]

Slider 1 29.022
Slider 2 59.621
Slider 3 0.32177
ArrivalTime 1
Calculation time: 0.035327 [s]

Slider 1 29.022
Slider 2 59.621
Slider 3 0.32177
ArrivalTime 2
Calculation time: 0.1573 [s]

Slider 1 2
Slider 2 25.511
Slider 3 0.12
ArrivalTime 3.8265
Calculation time: 0.04515 [s]

Figure 4: Segmentation using the logistic function to construct the speed image. Attempt at segmenting part (from top to bottom): Left ventricle, right ventricle, left lung, femur bone.



☑ Overlay
Slider 1 4.4164
Slider 2 2.5237
Slider 3 0.14827
ArrivalTime 16
Fastmarching implementation
Multistencil Fast Marching Method (Full)
Multistencil Fast Marching Method (no curvature)
Multistencil Fast Marching Method (no cross neighb
Calculation time: 0.051001 [s]

☑ Overlay
Slider 1 4.4164
Slider 2 2.5237
Slider 3 0.14827
ArrivalTime 21
Fastmarching implementation
Multistencil Fast Marching Method (Full)
Multistencil Fast Marching Method (no curvature)
Multistencil Fast Marching Method (no cross neighb
Calculation time: 0.053163 [s]

☑ Overlay
Slider 1 4.4164
Slider 2 2.5237
Slider 3 0.14827
ArrivalTime 27
Fastmarching implementation
Multistencil Fast Marching Method (Full)
Multistencil Fast Marching Method (no curvature)
Multistencil Fast Marching Method (no cross neighb
Calculation time: 0.054893 [s]

☑ Overlay
Slider 1 4.4164
Slider 2 2.5237
Slider 3 0.14827
ArrivalTime 35
Fastmarching implementation
Multistencil Fast Marching Method (Full)
Multistencil Fast Marching Method (no curvature)
Multistencil Fast Marching Method (no cross neighb
Calculation time: 0.055459 [s]

Figure 5: Progression of the segmentation using the derivative as a speed measure

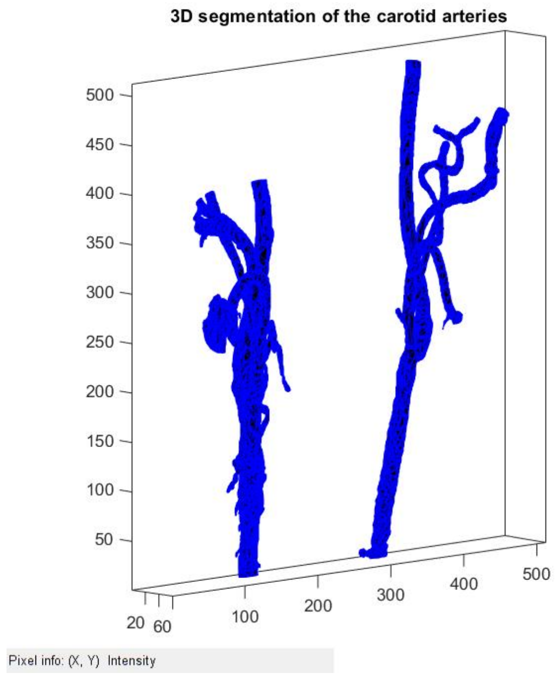

3D segmentation of the carotid arteries
500
450
400
350
300
250
200
150
100
50
20
60
100
200
300
400
500
Pixel info: (X, Y) Intensity

Figure 6: Segmentation of the carotid arteries

Part 4 - Marching in 3D

The fast marching method also works for images in 3D, and in this section we are going to try and segment the carotid arteries. Since the carotid file is in 3D it may be difficult to visualize how our speed map looks, the application will hence not be used. Instead I transformed the image from a coronal view to a transversal view. Since the arteries goes from the bottom up, each slice of the image will have (at least) two disks where the arteries are in the current slice. The arteries are in general brighter then the surrounding pixels which I used by thresholding each slice in such away that only the 8 % of the slice's brightest pixels remained. I set the speed of these pixels to 1 and the remaining pixels got a default value of 0.001. We now have our 3D speed image. I went through the transversal slices to find two starting point for our marching algorithm. I then set of the marching algorithm in the two starting points and found a suitable arrival time (about 300 for me). The method is quite simple, and it was easy to investigate the image to find the 8 % used as threshold and the 300 used for arrival time. Other then that, there was not really any parameters to find. The resulting segmentation can be found in figure 6.