Imperial College
London

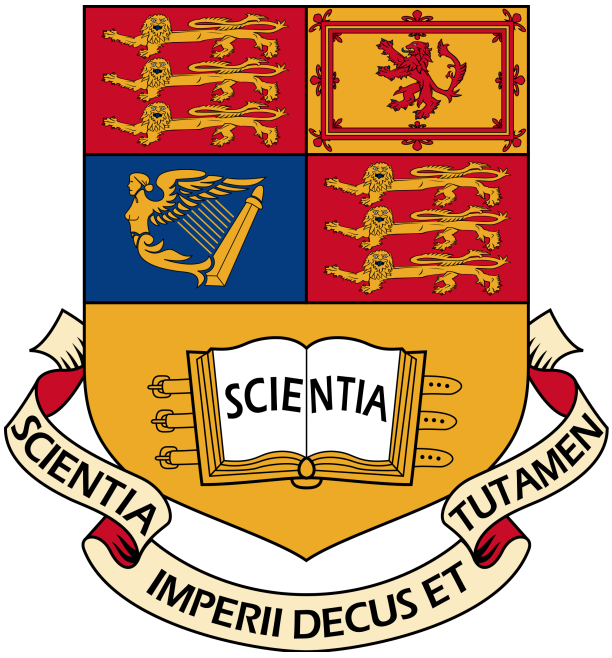
IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

# Automatic Vascular Segmentation and Analysis for Stent Graft Sizing

*Author:*
Kathryn Q. Shea

*Supervisor:*
Dr. Su-lin Lee




Submitted in partial fulfillment of the requirements for the MSc degree in MSc Computing of Imperial College London

September 2016

**Abstract**

Abdominal aortic aneurysms occur when the wall of the aorta weakens and bulges. In some cases, the aorta can be at risk of rupture and must be treated surgically. One method of treatment is fenestrated endovascular aortic repair (FEVAR). This method currently involves the doctor taking CT images, using current software technologies to analyze them, and then gathering measurements of the aorta and surrounding anatomies. This information is passed on to a manufacturer to fabricate a custom stent which takes 4 - 6 weeks. The goal of this project is to automate the measurement process.

Several algorithms were created and used in this report to automatically identify various geometries required for FEVAR stent fabrication. Contrast enhanced CT images from a patient were segmented, the aorta was extracted, and the center line was calculated. Since various arteries can be seen branching off the aorta, these areas were then identified so that the center line could be adjusted accordingly, reducing the bias induced by the presence of the artery. This also allows for the necessary lengths and diameters to also then be calculated. Preliminary diameter calculation experiments were conducted as well.

## Acknowledgments

I'd like to thank my supervisor, Dr. Su-lin Lee, the faculty and staff at Imperial College London who have put in so much time and effort toward the MSc Computing course, CSG for working with me on various hardware issues, my classmates who have helped me along the way, and my family for supporting me throughout the year.

# Contents

# Chapter 1

# Introduction

**Background** Abdominal aortic aneurysms (AAAs) occur when the aorta bulges due to the aortic wall weakening. Once large enough, the risk of rupture is high and the bulging must then be resolved surgically. Open surgery can be invasive and have long recovery periods but endovascular surgery can offer less invasive alternatives. In some cases, though, the best option is a customized fenestrated stent graft which requires the doctor to take images of the patient's aorta. The doctor then uses special software to gather measurements for the stent. These are sent to the manufacturer where fabrication can take 4-6 weeks–a lengthy amount of time in which the patient's aorta can rupture.

**Current Technologies** Current technologies offer 3D imaging of the patient's aorta as well as a few geometries including the center line of the aorta, but the doctor must still click manually to gather the final measurements. Other papers have already successfully segmented the aorta and calculated the various diameters needed as well.

**Goals** The aim of this paper is to take a set of contrast enhanced CT images of a patient with an aortic aneurysm and, with a single click, automatically calculate the center line geometry of the aortic region and identify where the renal arteries are. From there, the appropriate distances and lengths can be calculated so that a customized fenestrated stent graft can be manufactured for the patient.

**Results** The main outcomes of this paper are two algorithms: one for identifying if there is an artery stemming from the aorta in a given image and one for calculating the center line of the aorta. The artery-identifying algorithm was accurate for 716 images out of 737 (97%). The center line algorithm and a circle fitting algorithm were tested and compared. Both were precise and accurate, falling within a few pixels of each other and the manually estimated centroid.

Preliminary diameter calculation experiments were also performed suggesting that the diameter should be calculated by finding the vector from one centroid to an adjacent centroid and using this vector to find the orthogonal plane. Segmented points in the aorta that fall in this plane should be used to calculate the diameter. The diameter should be calculated using a circle fitting algorithm.

In order to implement and test these algorithms, a basic segmentation process also had to be written. This was largely successful and covered enough of the aorta for what was needed but, as a process, had several faults that could be improved.

**Future Works** Segmentation will need to be more robust in the future, perhaps by using model based segmentation. It will also need to follow both femoral arteries as the current method does not. This will lead to better results in the artery identification and center line calculations. The center line will also need to be smoothed through the aortic bifurcation and the diameter algorithm will have to be implemented in code.

One day, with automated measuring, it could also be possible to then automatically fabricate the stents robotically rather than having them woven by hand. This would greatly reduce the wait time and cost of the stents, ultimately saving patients more time and money and reducing mortality rates.

# Chapter 2

# Background & Context

## 2.1 Aortic Aneurysms

An abdominal aortic aneurysm (AAA) occurs when the aorta, the main blood vessel leading away from the heart, swells due to structural weakness in the wall of the aorta as in Figure 2.1. If the wall bulges too much, the aorta can burst causing internal bleeding which normally results in death. While AAAs typically affect men over the age of 65, causes are largely unknown. Smoking, high cholesterol, obesity, and high blood pressure are thought to be risk factors [1].

AAAs are treated in three ways: medical monitoring, open abdominal surgery, and endovascular surgery. Endovascular aortic repair (EVAR) involves placing a graft, a woven tube covered by a metal mesh support, inside the aorta. It is less invasive than open surgery and hence has a faster recovery time with similar survival rates [2]. However, EVAR does have affiliated risks such as endo-leaks, clot embolization, covering of an aortic side branch, infection, and stenosis or occlusion of a stent-graft limb [3].

These complications emphasize the importance of having properly fitting and well-placed grafts [4]. Additionally, the average cost of an EVAR device is estimated at £5000 [5] and can take over a month to manufacture–a period of time in which the aorta is at risk to rupture. In order to handle the various anatomies of individual patients, various methods exist for each case including percutaneous EVAR, fenestrated EVAR, and branched EVAR. This project will focus on fenestrated stent grafts.

## 2.2 Fenestrated Stent Grafts

Fenestrated endovascular repair (FEVAR) is used when the aorta is damaged in areas where it branches into the blood vessels to the kidneys (renal arteries), small bowel, and liver. Fenestrated stent grafts have windows (fenestrations) to reduce the risk of kidney failure by allowing blood to flow through it. These grafts are custom made for each patient resulting in the high costs and long wait for the device. However, advantages of using FEVAR include more people qualifying for grafts, faster recovery time, and lower mortality rates [7]. Figure 2.1 shows an example of a fenestrated graft made by Zenith.

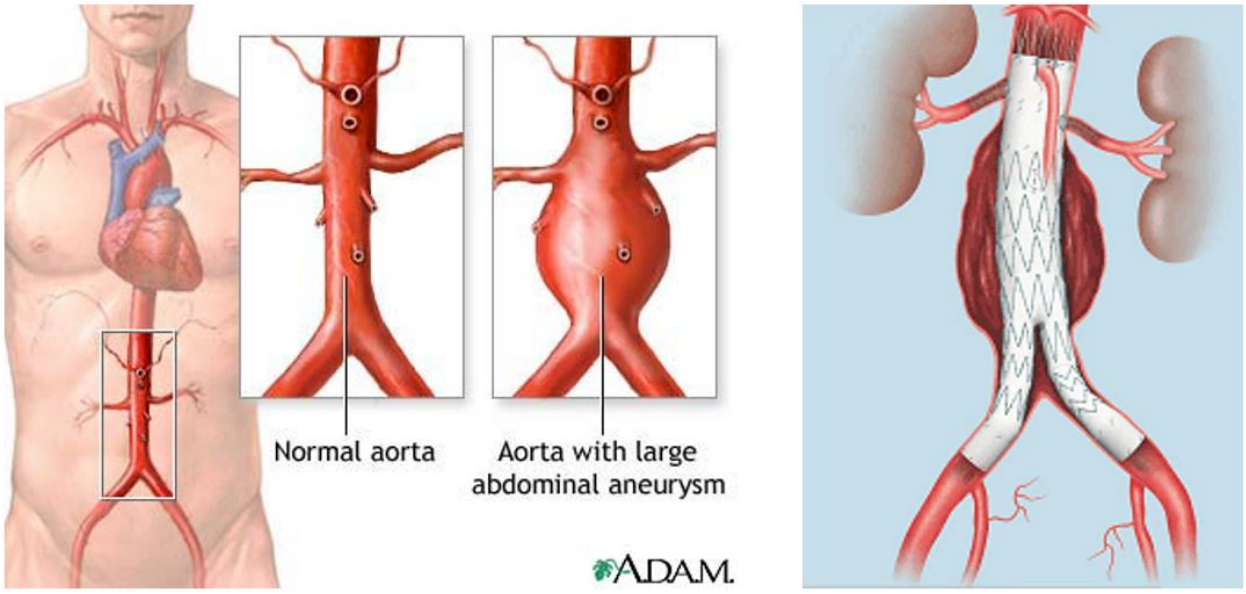



**Figure 2.1:** Left: typical and atypical anatomies of the human aorta [1]; Right: a Zenith fenestrated endovascular graft. The renal arteries can be seen stemming from the aorta to the kidneys [6].

## 2.3 Stent Fabrication

Stent fabrication requires taking measurements, typically via a Computed Tomography (CT) data set, of the patient's aorta and AAA and then sending this information to a company that manufactures the grafts. This process usually takes 4-6 weeks [8]. Measurements from images are typically acquired using various software packages as tools (described in the currently existing technologies section). Grafts are then ordered and fabricated by hand. If these measurements could be taken automatically (and especially if the grafts could be fabricated using robotic technology), this would greatly decrease the wait time and cost for the patients.

**Essentials of EVAR Imaging: Pre-Procedural Assessment** Picel & Kansal summarize the AAA imaging characteristics that must be accurately described for EVAR treatment planning. Aneurysms are described in terms of the proximal landing zone, the aneurysm sac characteristics, the distal landing zone, and the vascular access. Figure 2.2 depicts the important diameters (D) and lengths (L) necessary for EVAR planning [9].

The proximal landing zone is the region between D1 and D2 (L1). This is the region where the stent-graft requires proper fixation to prevent stent-graft migration although fenestrated stent-grafts have the ability to surpass the renal arteries above this zone so they are less limited by the size of this region. The aneurysm sac characteristics are important for evaluating risks such as kinking and endo-leakage.

The distal landing zone (L4 and L5) is typically in the common iliac artery (femoral artery). The diameter of the common iliac artery should not be larger than 25 mm and at least 10 mm in length for an adequate seal. Lastly, the vascular access measurements are required for deploying the device through the femoral artery. A general vascular evaluation is also typically included in such reports but this is out of the scope of this project [9].

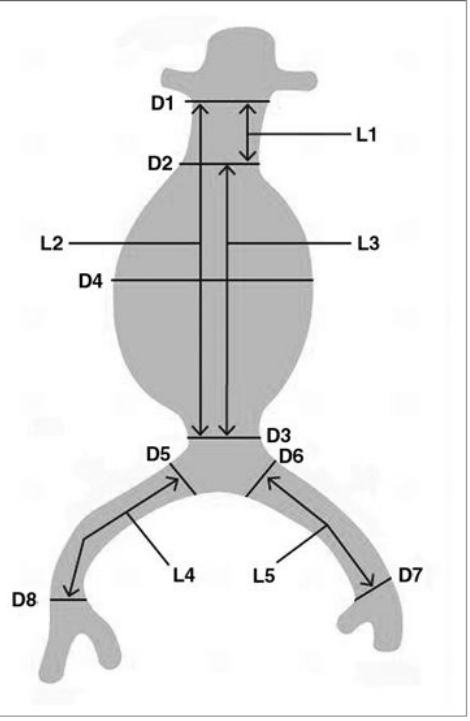
| Label | Description | Diagram |
|---|---|---|
| **D1** | Aorta at the level of the most inferior renal artery |  |
| **D2** | Aortic neck 15 mm distal to the lowest renal artery | |
| **D3** | Aorta at the bifurcation | |
| **D4** | Largest aneurysm sac dimension | |
| **D5** | Common iliac artery (1) | |
| **D6** | Common iliac artery (2) | |
| **D7** | Minimal diameter in distal landing zone (1) | |
| **D8** | Minimal diameter in distal landing zone (2) | |
| **L1** | Length of the aneurysm sac | |
| **L2** | Length from lowest renal artery to aortic bifurcation | |
| **L3** | Length of aneurysm sac | |
| **L4** | Length of distal landing zone (1) | |
| **L5** | Length of distal landing zone (2) | |



**Figure 2.2:** Descriptions of the measurements required and a diagram of the planning template for EVAR measurements [9].

## 2.4 Current Technology

**EVAR Planning Package by Tera Recon** Tera Recon is a company that provides an EVAR Planning Package for assisting doctors in visualizing and measuring the geometries of the aorta. "EVAR Planning Package Provides features including an advanced measurement protocol option, user definable planning template with report output and embedded instructions... By following guided instructions, the user can complete stent-graft planning and generate reports for vendor specific templates." [10].

**Syngo by Siemens** Siemens uses syngo.via in association with SOMATOM Scanners to analyze CT images for EVAR planning. "The Autotracer automatically segments and labels the vessels even before the case is opened. The aorta is displayed in a curved planar reformation and the center line is automatically created providing the basis for important length measurements." [11]. This does not fill out the rest of the measurements–the doctor must do that.

**3mensio by Pie Medical Imaging** 3mensio is used for EVAR, TEVAR, and FEVAR stent placement. The center line can automatically be measured and then relevant measurements must be taken by hand using the software to assist. Appropriate manufacturer stent order sheets are also generated after the measurements are taken [12].

**Endo Size by Therenva SAS** Endo Size uses standard CT scanner data to simplify anatomical measurements along the vessel center line and to guide the operator. It also generates the proper sizing worksheet for ordering stent grafts, among other features [13].

**Automatic Extraction of the Aorta and the Measurement of Its Diameter** Mukai, et al were able to extract the aorta (even when the spine overlaps with it) and measure the diameter of the aorta however some parts of the heart were extracted as the aorta. Images were obtained by CT without contrast [**Mukai422013˙BJMMR˙5050**].

## 2.5 Useful Image Analysis Libraries

Contrast enhanced CT images from three patients have been provided in Digital Imaging and Communications in Medicine (DICOM) format. DICOM is the international standard for medical images. It is used in most medical imaging devices (X-ray, CT, MRI, etc.) and allows for patient information to be embedded in the file itself. Specifically, the DICOM standard contains data elements (attributes) containing the patient information, modality and imaging procedure information, and the image information (e.g. patient name, dpi, etc.) [14]. There are many libraries specifically for dealing with these files. The following libraries were investigated as possible resources for this project.

**Visualization Toolkit** The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, modeling, image processing volume rendering, scientific visualization, and information visualization. VTK is capable of converting DICOM files to VTKs memory data structures so that further processing using other VTK functions (including visualization and image processing) can be performed [15].

**Insight Segmentation and Registration Toolkit** Insight Segmentation and Registration Toolkit (ITK) is an open-source toolkit that operates largely in C++. Segmentation, in the scope of this project, entails identifying the aorta from everything else in the images. Registration is used to bring multiple data sets of the same object into one space [16].

**Medical Imaging Toolkit** Medical Imaging Toolkit (MITK) is an open-source C++ library for integrated medical image processing and analyzing. It has the capability (and tutorials) to create 3D views with or without volume rendering from DICOM files and can handle visualization of multiple views (i.e. axial and sagittal) using Qt horizontal boxes allowing the developer to explore the data. MITK combines the ITK and VTK with an application framework [17].

**Image-based Vascular Analysis Toolkit** Image-based Vascular Analysis Toolkit (IVANTK) is an open-source extension of ITK implemented largely in C++ and focuses on vascular detection, analysis, and modeling in medical imaging. Ivan Oliver has also provided, in his thesis, a segmentation method for both the lumen and thrombus of abdominal aortic aneurysms on CTA images after endovascular intervention and a method for the automatic detention and quantification of endoleaks in the thrombus. [18].

**Medical Image Analysis** Medical Image Analysis (MIA) is a general purpose framework for gray scale medical image processing implemented in C++. IT provides command line tools, plug-ins, and libraries for running image processing tasks interactively in a command a command shell. One valuable command-line ability includes converting a series of 2D images (DICOM) to a volume (3D) [19].

# Chapter 3

# Design & Implementation

The following sections describe the design process and implementation of the software written for this project. The main three processes described include segmentation in Section 3.3, identifying arteries in Section 3.5, and calculating the center line in Section 3.4 and Section 3.6. Other supporting sections discuss the user input, importing the DICOM images, calculating the diameters, and visualizing the data.

## 3.1 User Input

Some pieces of user input are currently hard-coded at the beginning of `thesis.py` (Appendix B.1). These include the path to the DICOM directory, a threshold value for segmenting the aorta, and the initial image number to be presented. In the future, these should be implemented into a graphical user interface.

After importing the DICOM files, in order to initially identify the aorta, the program starts by presenting the user with one of the images from the data set as seen in Figure 3.1. The user then clicks on the aorta in the image and the program takes these coordinates as the starting point. This is done using the `sitk_get_click()` and `onclick()` functions in `thesis.py`.

## 3.2 Importing Data

The data was provided as three folders, one for each patient, containing several hundred DICOM images each. There were many libraries capable of importing these files but it was a struggle to get any of them to perform all of the tasks required. There are many DICOM libraries out there so I chose to narrow my search down to C++ compatible ones (as mentioned in Section 2.5).

After attempting to use several C++ libraries, a tutorial was considered which explained the use of SimpleITK (SITK). SITK was not only easy to install but also easy to use and well documented. This also prompted a shift to using Python instead of C++. Python is likely considerably slower than C++ would be but is perhaps easier for quickly coding up and testing new ideas.

SITK has a function for importing DICOM files as SITK images. The SITK images are 3D arrays with additional information that can be retrieved through other SITK functions such as dpi, patient name, etc. The 3D array is made up of multiple images or "slices" stacked on top of each other. Since contrast is used during the imaging, the aorta typically appears as a bright circle roughly in

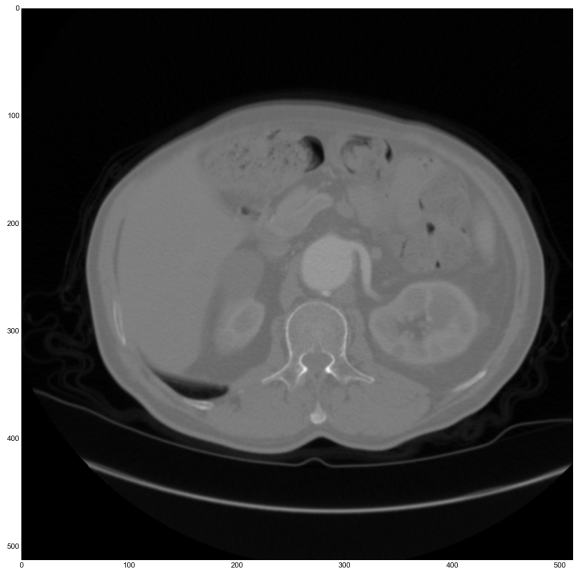



**Figure 3.1:** A DICOM image as it would be presented to the user to acquire initial coordinates. This particular image shows the aorta with a renal artery stemming off and wrapping around the side of it.

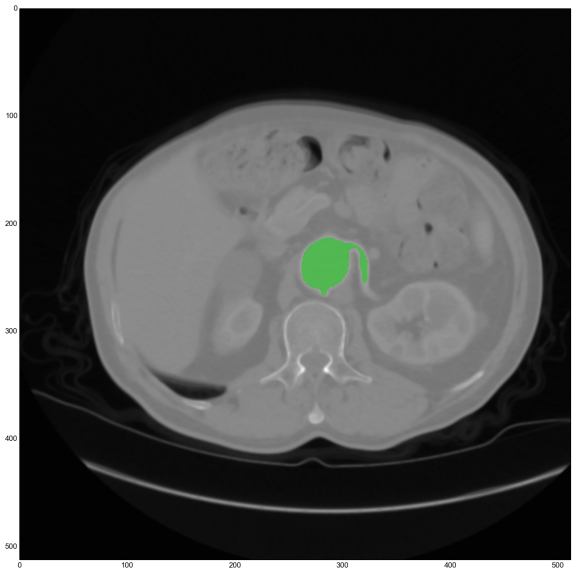



**Figure 3.2:** Segmentation displayed in green and overlaid the original image in Figure 3.1. In this particular example a threshold (leeway) of 40 was used and an artery can be seen stemming off the aorta.

the middle of the image (see Figure 3.1). Images in the given data sets were 512 x 512 pixels and contained transverse images of the torso.

The library also has a function for converting SITK images to NumPy arrays which allowed for faster image processing. The library, however, has a quirk where a coordinate, (x,y), in a SITK image would correspond to (y,x) in a NumPy array. This resulted in quite a few bugs during development and should be well noted for future works.

## 3.3 Segment Aorta

Segmentation, in layman's terms, is not unlike the "paint bucket" effect. It selects a portion of an image based on a coordinate, the color of the pixel at that coordinate, and the threshold for how dissimilar other pixels can be in order to also be selected. Figure 3.2 is the same image as in Figure 3.1 but now with the segmentation on top of it, highlighting the aorta (and the renal artery stemming from it) in green.

Segmentation is the first step required to test center line and artery-identification algorithms. The following sections outline the general issues encountered with segmentation. The main focus of this report was to segment the abdominal aortic region, but the femoral arteries will be needed in future works.

### 3.3.1 Simple ITK Toolkit

This software uses the segmentation function included in the SITK library. Before segmentation can be performed, the image must first be smoothed (another function also included in the li-

brary) [20]. The segmentation function returns a binary array that is the same size as the original image where ones indicate the selected area and zeroes fill in the rest. Using the hole-filling SITK function after segmentation also fixes some of the anomalies encountered (e.g. partial segmentation, if the threshold was not set perfectly).

The SITK library was very useful for importing the DICOM images and converting them to NumPy arrays. The segmentation process, however, was not as robust as it could have been. Some segmentations, while they were in the correct locations, did not pick up the aorta very well as in Figure 3.4.

In future works it would be beneficial to try creating a custom segmentation function that does a breadth-first-search (BFS) [21] from the center until an edge is detected rather than using a single color with a tolerance value as the limits (or perhaps using a different library). The edges in the DICOM images tend to be pretty well defined but the aorta itself is not always an incredibly uniform color. This method would likely have its own set of drawbacks but would be worth investigating.

### 3.3.2 Propagation

**Segment Initial Image** The SITK segmentation function only segments a single slice. The next challenge then is to segment the entire data set given only a single point. In order to do this, the initial image is first segmented. This image could be any image in the set– somewhere where it is easy for the user to spot the aorta. Then the centroid of that image is calculated (see Section 3.4) and passed to the neighboring slices.

**Propagate by Overlap** The software then propagates outwards, moving towards either end of the aorta, by taking the prior centroid and using that to perform another segmentation. This process continues until either the program reaches the end of the data set or the new segmented area does not sufficiently overlap with the old segmented area. For example, if the segmentation were c-shaped, then the centroid would fall outside of the segmented area. The next segmentation would select a completely different area and likely not overlap with the prior slice since they would likely be different colors. Figure 3.3 shows a fully segmented aorta that follows one femoral artery. The initial (clicked on) slice is identified by the origin plane toward the end of the artery.

**Propagate by Color** The algorithm used to take the color of the previous image and compare it with the color of the centroid pixel on the next image but since brightness can vary from image to image, the aorta's coloring does not always stay consistent between images. An attempt was made at normalizing the image color range to have a fixed range in hopes that it would normalize the brightness of the aorta but that was also unsuccessful since the brightest object in one image (typically the spine) might disappear in the next image.

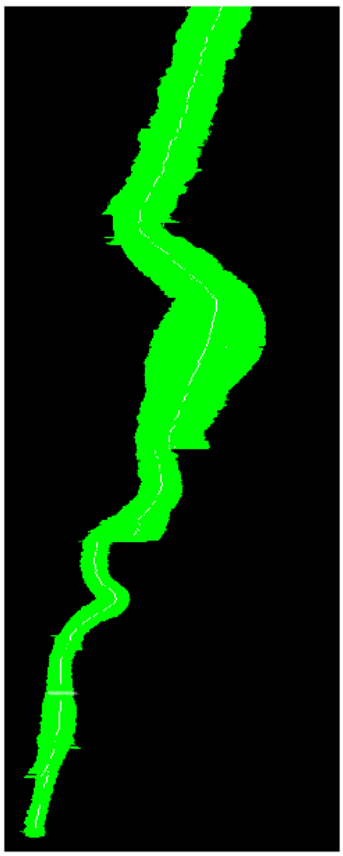


**Figure 3.3:** 3D visualization of full segmentation following one femoral artery. The center line is fragmented at the bifurcations.

**Threshold** The threshold value (called `leeway` in `thesis.py`) impacts how much of the region is highlighted. When selecting the area to segment, the program must have an upper and lower color limit. This limit is determined by taking the color value of the pixel selected and setting the upper limit equal to that value plus the leeway value and setting the lower limit equal to that value minus the leeway value. If the threshold is too high, then other bodies that are close to the aorta and of similar color (the most common culprit being the spine) can also be selected. Having a value that is too low can result in partial segmentations. With the current `leeway` value, both of these events happen.

### 3.3.3 Challenges

The segmentation process has a few challenges to overcome including spine clashing, following both femoral arteries, unusual arterial shapes, and poor segmentation.

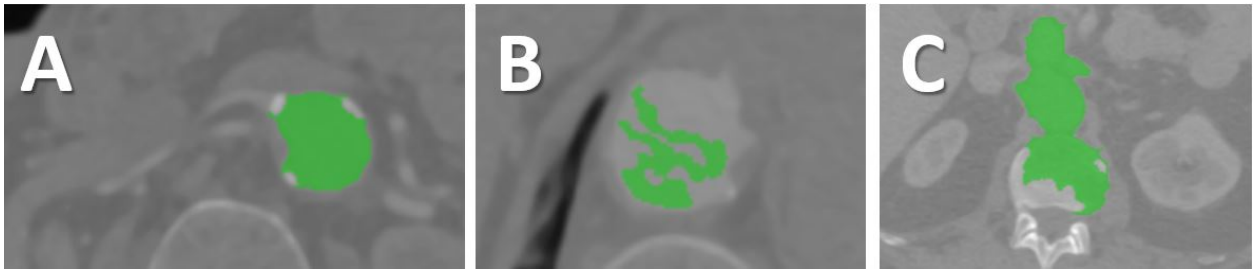



**Figure 3.4:** Examples of poor segmentation. (A) An anomaly blocks the segmentation from the artery. (B) Segmentation did not go everywhere. (C) Segmentation selected spine too.

**Poor Segmentation** Of the several hundred images in a data set, only a few will form a segmentation as in Figure 3.4 (B) where it only fills a small portion of the aorta. This is usually not a problem for continuing propagation but it is a problem for identifying arteries and calculating the diameter of that image.

This could be fixed if a customized segmentation function was written or if another library was used. It might be better to segment based on edge detection rather than a color threshold. If one looks closely at the images, the pixels within the aorta vary by quite a bit in color. It might also be beneficial to try averaging the values of these colors and calculating a threshold using that average.

Lastly, Figure 3.4 (A) shows how a calcification which appears as a bright white color can block the segmentation from filling in all of the space. In this image in particular it is a problem because that would be the image to identify this particular artery in. Adjacent images might not have the artery as well defined and thus might miss it.

**Clashing with the Spine** The spine can get close enough to the aorta and have a similar enough color that it can also be selected as in Figure 3.4 (C). The resulting 3D visualization can also be seen in Figure 3.5 (B).

This is something that was solved by Mukai et. al. [22]. The report focuses on images where the CT image is taken with no contrast. Since the images provided for this project used contrast, it did not seem like this would be a problem and indeed, two of the three data sets were able to fully segment without any spinal collisions. The other data set, however, had considerably less contrast

and merged with the spine in some cases, as in Figure 3.5 (B).

Mukai et. al. solved the problem by first subtracting the spine from the images and then segmenting the aorta. Since the segmentation was adequate in two of the three data sets (and given the time constraints of the project), it did not seem worth the time to perfect this aspect of the software any further.

**Following Both Femoral Arteries** Currently, this algorithm does not follow both femoral arteries, as in Figure 3.5 (E), but it does sometimes follow one. This is something that should be built upon in the future in order to measure both femoral arteries. This report focuses on the aortic center line and geometries instead.

It could be beneficial to have the user start with an image containing both femoral arteries and have them click on both. Then the software could propagate both until they merge and then continue with one. This might also allow for the center line to be followed in a smoother fashion so that the center line does not jump at the aortic-femoral junction as in Figure 3.3. Otherwise pursuing an avenue that involves sampling random points in the last image (rather than just taking the centroid) and segmenting images with those points and then comparing the overlap with the last image. This, though, might be too computationally intensive.

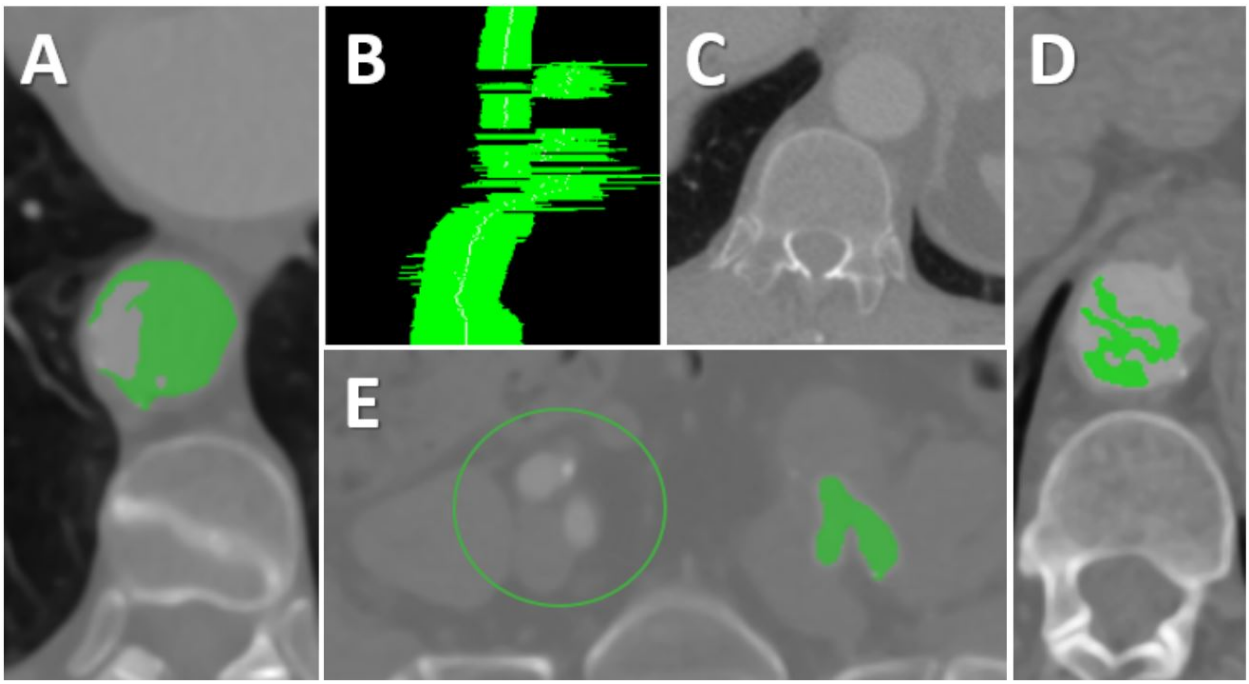



**Figure 3.5:** Common problems with segmentation. (A) Failure to segment whole area. (B) 3D visualization of aorta when segmentation also selects the spine. (C) The spine can be very close to the aorta and quite similar in color. (D) Partial propagation. (E) Unusual segmentation shape where the centroid falls outside the area. Femoral arteries on the other side are not picked up by this method either.

**Centroid Falls Outside Area** When the artery prepares to split into two arteries it sometimes assumes a kidney-bean shape and the centroid no longer falls within the area of the artery. These strange shapes almost always occur in the femoral arteries as in Figure 3.5 (E). This could perhaps

be avoided if, instead of using the centroid to segment the next image, a sample of points were chosen from the last image and then tried in the next image and tested for overlap. The software could also attempt to keep track of multiple shapes rather than just the artery.

### 3.3.4 Working Around Errors

A semi-automatic fix is used when the whole segmentation is needed and either the centroid falls outside the artery or the selected area does not overlap with the prior slice. When one of these events occurs (and the semi-automatic script is being used), the program prompts the user for an additional "initial click" to continue selecting through the entire aorta. While this is not ideal for a final automated product, it was necessary for testing the femoral arteries and for full 3D visualization of the entire data set. It does not, however, solve the problem of the spine being picked up in some images. As a result, the third data set was not used due to the spine collisions.

## 3.4 Calculate Center Line

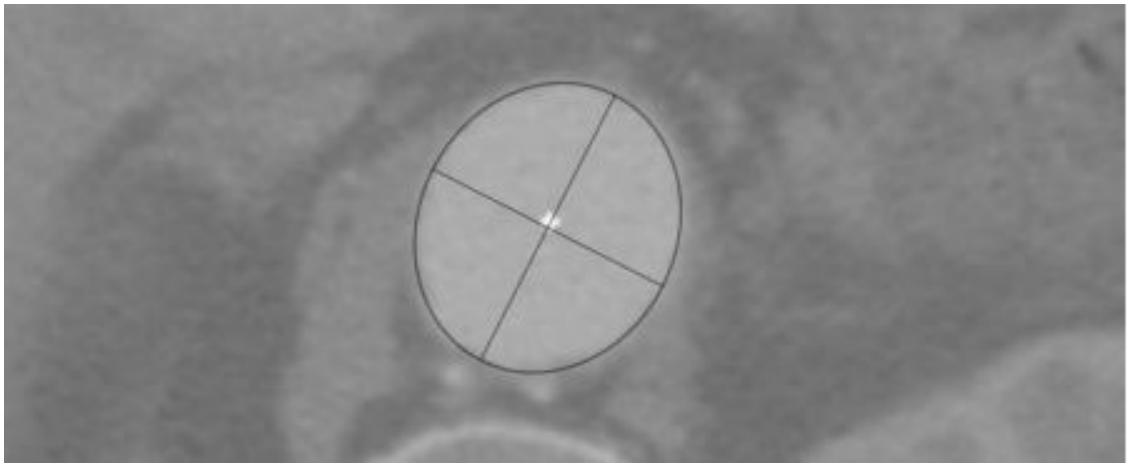


**Figure 3.6:** An aorta with the centroid superimposed and a circle with a cross overlaid for comparison.

In order to calculate the geometries of the aorta, the center line must first be established. While this is fairly simple in sections where the aorta is quite round, it is more difficult when arteries begin to split away from the aorta (at the aortic bifurcation). The calculation used to find the center of a given slice is the same as finding the center of gravity of an object:

$$x_c = \frac{x_1 + x_2 + ... + x_k}{k}$$

$$y_c = \frac{y_1 + y_2 + ... + y_k}{k}$$

where $x_c$ is the x-coordinate of the centroid, $y_c$ is the y-coordinate of the centroid, $k$ is the number of elements in the segmented area, and $x_k$ and $y_k$ are the coordinates of the elements in the segmented area. This calculation checks each element in the array and sums the x-coordinates and y-coordinates of all elements. It then divides the sum by $k$. This results in the centroid x- and y-coordinates. As a side note, this calculation would not be suitable up in the aortic arch where the

aorta is horizontally oriented (though the aortic arch is not necessary for this particular analysis).

Figure 3.6 shows a typical image of the aorta and how nearly-circular it appears. The overlaid ellipse with a cross was superimposed manually and shows that the ideal, approximated centroid is within a few pixels of the calculated centroid.

What happens then, when the image captures an artery stemming off of the aorta, as in the image in Figure 3.2? This would result in the centroid being biased toward the artery. Additionally, the program needs to calculate distances between certain arteries so it would be doubly beneficial to be able to identify which images contain arteries. This is described in the following section.

## 3.5 Identifying Images Containing Arteries

**First Idea** Initially it seemed like a good idea to perform a breadth-first-search (BFS) from the calculated, albeit biased, centroid to the pixels on the edge of the aorta, and record the distance to each pixel. Then, if the maximum distance was much greater than the minimum distance, it would be reasonable to say that there might be an artery present. This algorithm was effective in finding arteries, but also had a significant amount of false positives. Small anomalies and minor segmentation errors caused the algorithm to identify far more arteries than were present.

**Second Idea** Another thought was to start from the centroid and, rather than perform a BFS, traverse in a straight line to the first edge point in various directions (zero, forty-five, ninety, etc. degrees). This worked if the artery happened to be in a straight line and the directions chosen happened to be in the same direction as the artery as in 3.11. This method proved faulty, however, when the arteries wrapped around the side of the aorta as in Figure 3.2. The wrapping drove the algorithm toward a BFS solution rather than a euclidean distance solution.

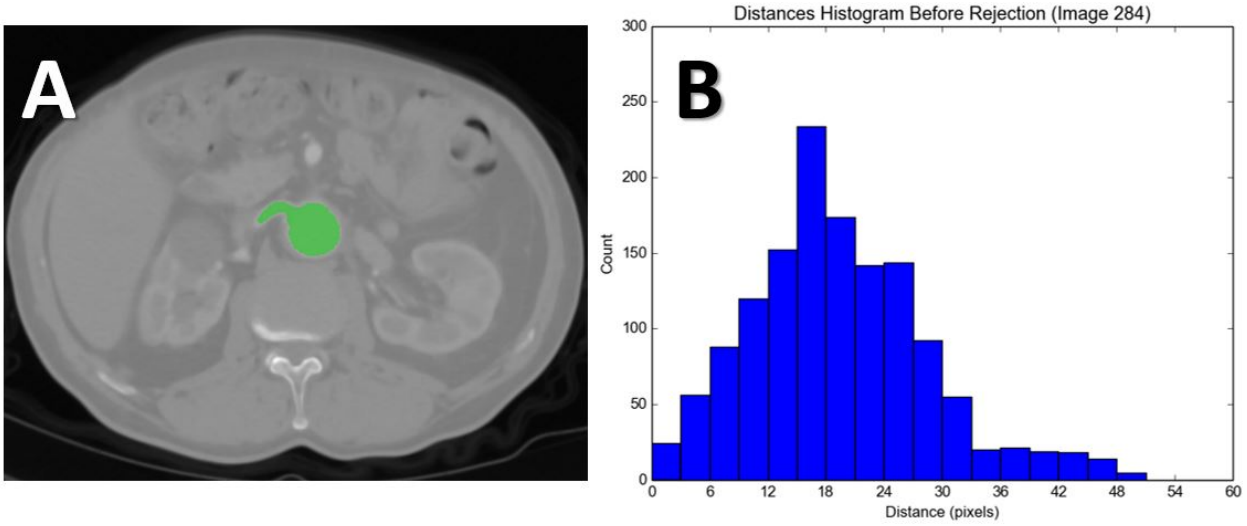



**Figure 3.7:** (A) A segmented aorta with an artery present. (B) A histogram of distances (in pixels) from the centroid to each pixel. Note the outlier values starting at approximately 30.

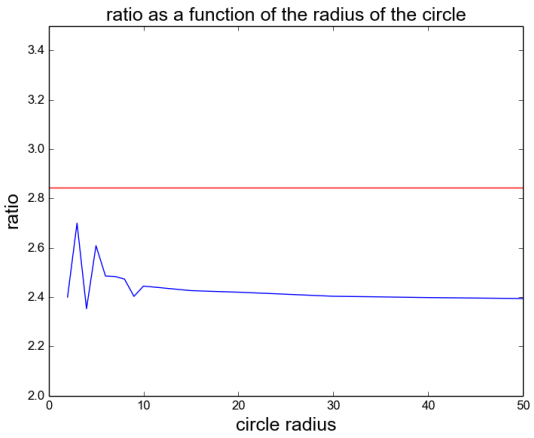



**Figure 3.8:** Ratio as a function of a circle's radius in pixels (blue) which has a max of 2.7 and an average of 2.4. The chosen threshold, 2.845 (red).

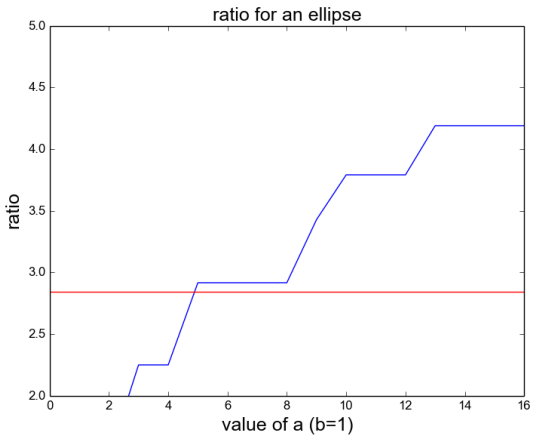



**Figure 3.9:** Ratio as a function of an ellipse's a-value. As 'a' increases, the ellipse becomes less circular and surpasses the threshold just before `a=5`.

### 3.5.1 Artery-Identifying Algorithm

The algorithm chosen involves a BFS implementation. First, the BFS distance from the centroid to every single pixel in the aorta (just the ones in the segmented image and not the zeroes) is calculated (an example of this can be seen on the left of Figure 3.12). These distances are sorted in a list from least to greatest. Then, the bottom third of the distances are taken and averaged (the "lower average"). The same is done for the upper third of the list (the "upper average"). Figure 3.7 (A) shows a segmented image with an artery and (B) shows the resulting histogram of distances. The distance values appearing above approximately thirty are a part of the artery.

**Threshold** A ratio is then calculated by taking the upper average and dividing by the lower average. This ratio is compared with the threshold to determine if the image is roughly a circle or if it likely contains an artery. The ratio for a perfect circle (Figure 3.8) is unstable at low radii values making this method less practical in the really small femoral arteries but the aorta's radius is usually between fifteen and thirty pixels.

In a nearly-perfect circle, this ratio is in the range 2.4 - 2.5. However, in images containing arteries, this ratio is typically above 3.0 and almost always above 2.845. In a very few select cases, an image might falsely fall on either side of the threshold, 2.845. `Circle.py` (Appendix B.4) was used to test the ratio for a circle of various radii as shown in Figure 3.8 and the ratio for an ellipse in Figure 3.9.

**Ellipses** The reason why the line converges on approximately 2.4 but the value for a "nearly-perfect circle" is often closer to 2.5 is because the ratio increases as the shape becomes more elliptical. Figure 3.9 shows how the ratio increases as the circle becomes an ellipse. In this image, the equation used for the ellipse is:

$$(x^2/a) + (y^2/b) = 1$$

where $x$ and $y$ are coordinates along the edge of the shape and `a` and `b` are the constants that change how elliptical the resulting ellipse is. If $a$ and $b$ are equal, the shape will be a circle so $b$ is

set to one while $a$ increases.

The ratio for an ellipse surpasses the threshold when $a$ is approximately five. Figure 3.10 shows a circle of radius one and an ellipse (with $a = 5$ and $b = 1$) overlaid to give an idea as to how elliptical a figure must be to be falsely identified as an image with an artery. While the aorta does not typically take a particularly elliptical shape, the femoral arteries sometimes do.

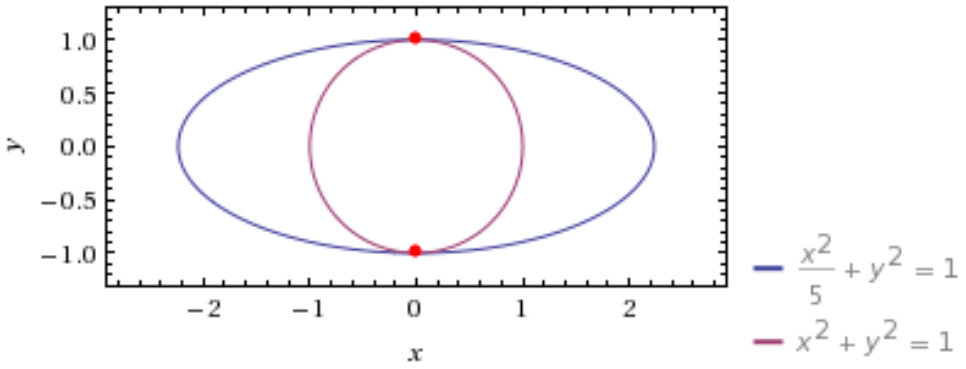



**Figure 3.10:** A circle with radius of one (red). An ellipse on the border of the threshold between having and not having an artery (blue).

## 3.6 Calculate Centroid in Images With Arteries

The centroid of each image can be biased if there is an artery detected. Figure 3.11, for example, shows the original biased centroid edging towards the top of the segmented area where the artery is. Any image identified as having an artery must have the center recalculated so that the center line is more accurate.

**Sampling Edge Points** In order to recalculate the centroid, the algorithm first takes a sample of eight points around the edge of the image (Figure 3.11). Starting from the current estimated centroid, the algorithm proceeds at zero, forty-five, ninety, etc. degrees until it hits an edge point and then checks for any outliers by distance within the eight points. Outliers are removed and then the centroid is calculated based on those remaining points.

**Histogram** After acquiring a rough estimate for the center of the aortic region in the segmentation, the algorithm then calculates (using a BFS) the distance to each pixel from the centroid. An example of what this distance calculation might look like for each element in a perfect circle with radius of four can be seen in Figure 3.12. The histogram for a circle of radius 30 shows that the radius of the circle would be approximately equal to the mode (29). This principal is used to decide which coordinates to eliminate.

**Modal Rejection** Another rejection function runs to remove coordinates with distances that are fifteen units above the mode distance. Fifteen was chosen as the threshold after experimentation with the images. As segmentations become elliptical, the ends can be falsely rejected. Rather than have that, fifteen was chosen to err on the side of leaving a small amount of the artery. In the

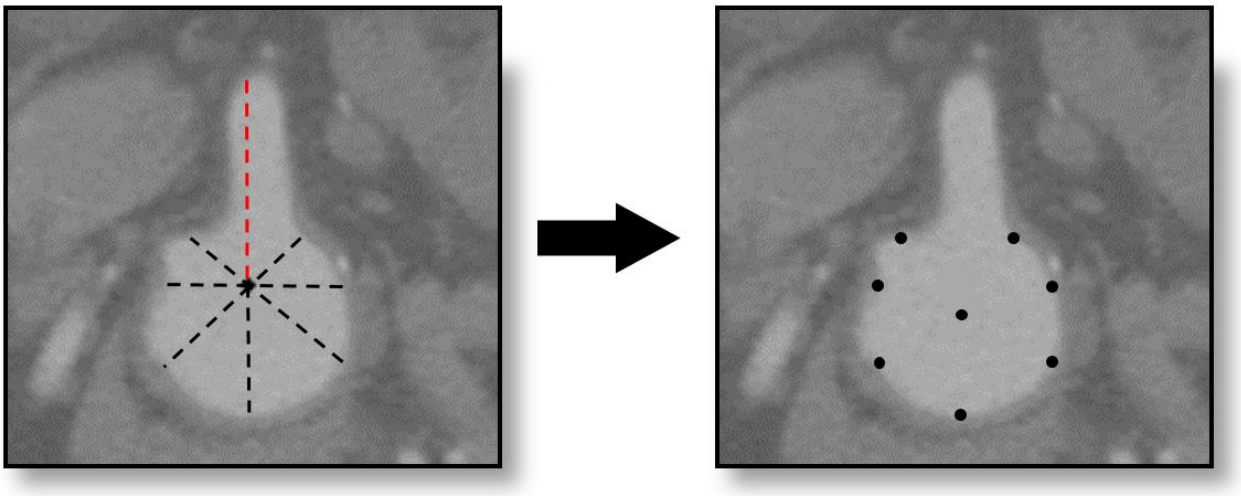


**Figure 3.11:** The red line shows the outlier edge point as determined by distance from the centroid. This point will not be included in the new calculation. The centroid displayed on the right is an approximate center based on the seven sampled points left.

Shape (array, r=4)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 4.0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 3.6 | 3.2 | 3.0 | 3.2 | 3.6 | 0 | 0 |
| 0 | 3.6 | 2.8 | 2.4 | 2.0 | 2.4 | 2.8 | 3.6 | 0 |
| 0 | 3.2 | 2.4 | 1.4 | 1.0 | 1.4 | 2.4 | 3.2 | 0 |
| 4.0 | 3.0 | 2.0 | 1.0 | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 |
| 0 | 3.2 | 2.4 | 1.4 | 1.0 | 1.4 | 2.4 | 3.2 | 0 |
| 0 | 3.6 | 2.8 | 2.4 | 2.0 | 2.4 | 2.8 | 3.6 | 0 |
| 0 | 0 | 3.6 | 3.2 | 3.0 | 3.2 | 3.6 | 0 | 0 |
| 0 | 0 | 0 | 0 | 4.0 | 0 | 0 | 0 | 0 |

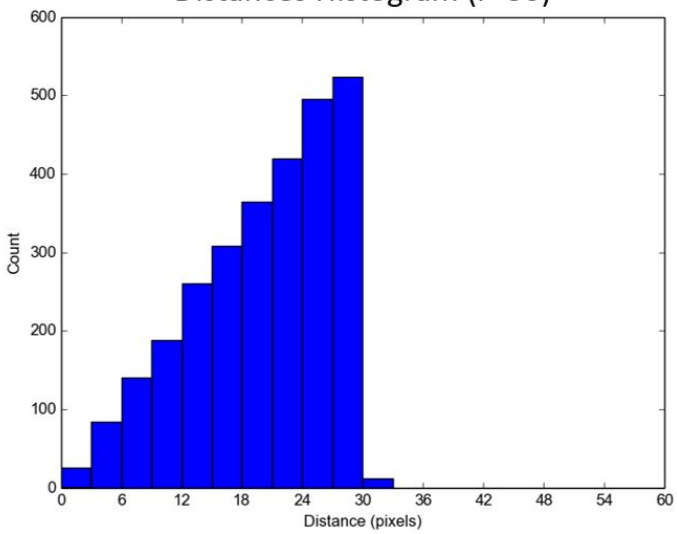



**Figure 3.12:** Left: Calculating the distance (in pixels) from the centroid to each pixel inside a circle with a radius of four. Right: The resulting histogram of distances for a circle with a radius of 30.

future, using a Gaussian method of rejection could be considered for comparison but the implementation should be sure to to unintentionally eliminate the bottom end of the spectrum.

Figure 3.13 (A) shows the original histogram for the segmentation we analyzed in Figure 3.7 (also shown in Figure 3.14 (B)). Now the outliers well above the mode have been removed in Figure 3.13 (B) and the resulting image can be seen in Figure 3.14 (C) where the segmentation has been trimmed.

**Recalculate Centroid** Finally, after trimming the segmented area using the modal rejection, the center of gravity calculation (as discussed in Section 3.4) is used again on the new area. The resulting centroid with an overlaid circle for comparison is shown in Figure 3.14 (A). The cen-

troid coordinates are stored in three files (`clx.txt`, `cly.txt`, and `clz.txt`) to form the overall center line. Steps such as edge sampling and modal rejection can be repeated for a more refined calculation.

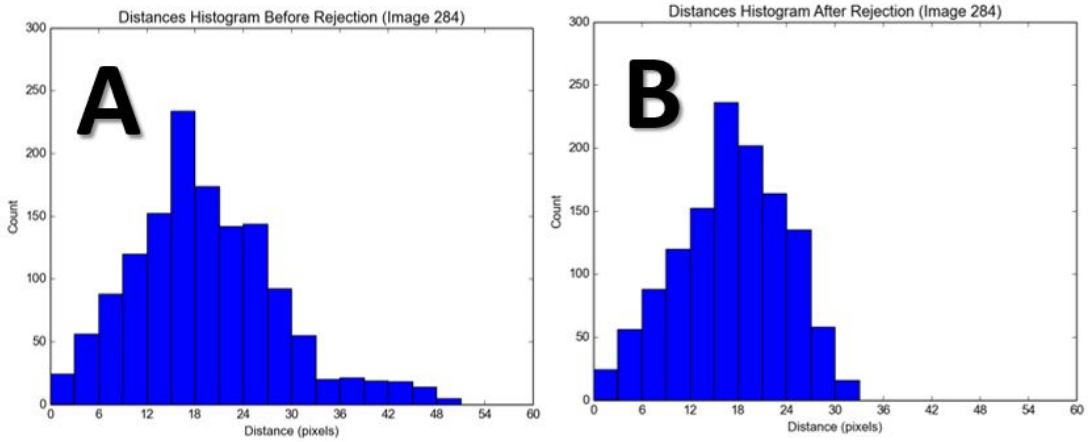



**Figure 3.13:** (A) Histogram of the distances (in pixels) from the centroid of the shape to each pixel. (B) Histogram of the same distances as in A but without the outliers on the upper end of the graph.

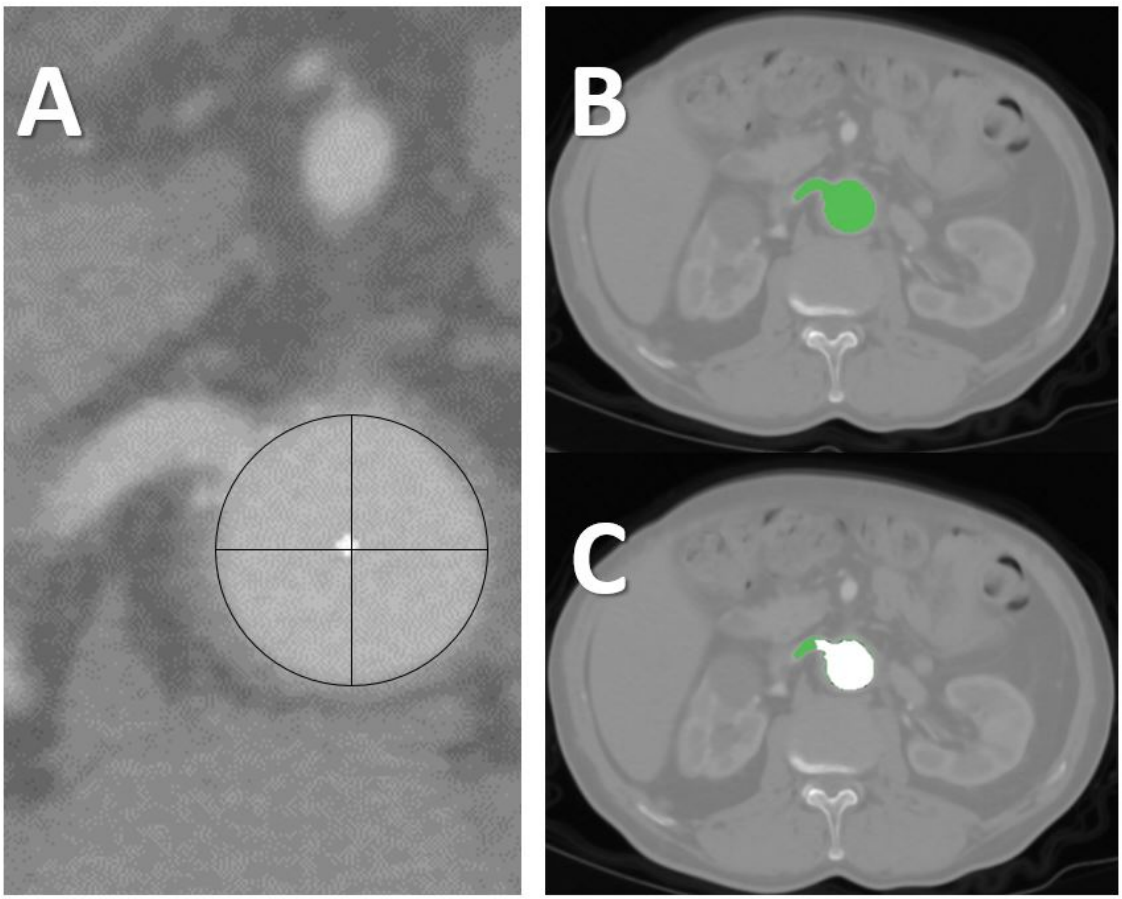



**Figure 3.14:** (A) Original image with centroid marked and an ellipse with a cross superimposed for comparison. (B) Original segmentation of aorta and artery. (C) Segmentation after modal rejection.

### 3.6.1 Circle Fitting

Circle fitting is a process whereby a radius and center point are fit to a set of data points such that the resulting circle is as similar to the data as possible. Code from the SciPy Cookbook [23] which calculates a least squares circle has been modified (see Appendix B.5) to fit a circle to the edge points of the segmented array (the 'ones' in the binary array that are on the edge of the shape). Edge points are defined as any pixel that is touching a zero.

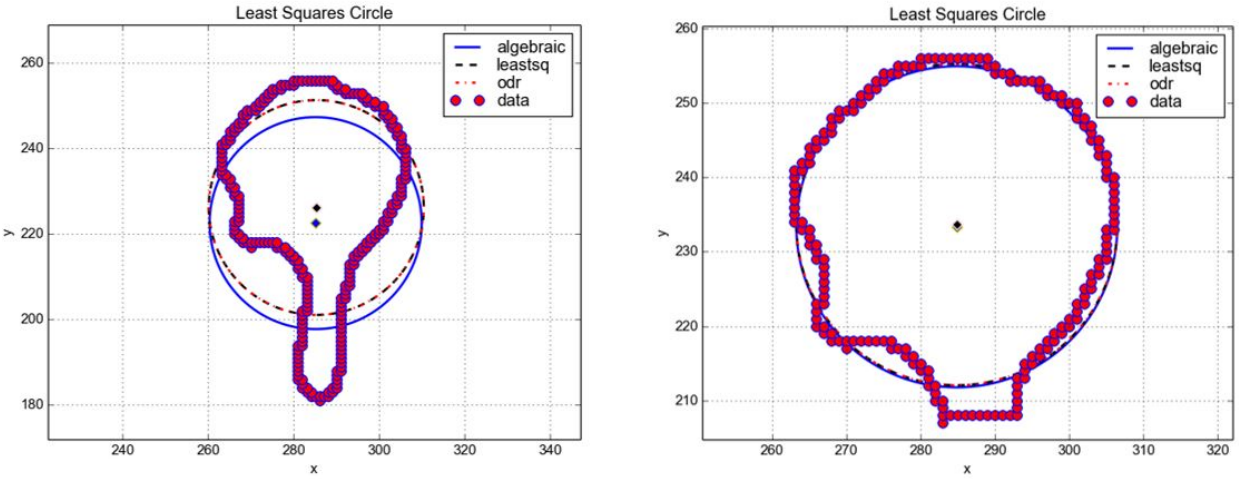



**Figure 3.15:** Left: Circle fitting the aorta edge points ('data') when an artery is present using three different methods. Right: Circle fitting the aorta edge points after the artery has been trimmed.

Before trimming the aorta, the circle fitting is not very practical. After trimming the aorta, the circle fitting works very well. The SciPy Cookbook code generates three circles since the goal of the original code was to test various methods. All three methods create approximately identical center points and radii for the trimmed aorta and the center points are within a pixel of the center-of-gravity calculation. This is exhibited in Figure 3.15. Since the circle fitting also yields a radius approximation, it would likely be practical to the use it while calculating the diameter of the aorta at a given center point. This topic is discussed in the following section.

## 3.7 Calculating Diameters

The diameter must be calculated at the top, middle, and bottom of the aortic aneurysms as well as in the femoral arteries and at the base of the renal arteries. The calculation requires the center line points as well as the circle fitting method. A simple method has been used in this implementation but a more complex method should be used in future works.

At first, a sample of four diameters (the same as in Figure 3.11) were taken for each image and the maximum diameter was chosen after removing outliers. This naive approach was used before the circle fitting was performed but with the circle fitting, this allows for the radius measurement to be used as the diameter of the aorta for that image instead.

A third alternative method of calculating the diameter involves using the area. Since the aorta is not a rigid structure and the purpose of the diameter is to design a cylindrical graft for the aorta, an assumption could be made that the aorta will reform to fit the graft if the aorta isn't perfectly round. The diameter could then be calculated by finding the area of the segmented region and

deriving the diameter from there.

While these methods will succeed at calculating the diameter in the instances where the aorta is truly vertical, it does not bode well when the aorta is at any other angle. Nonetheless, these methods have been tested on the images as they are, regardless of the aorta's orientation, for comparison. This means that the results are not very accurate but should help drive the direction for future works.

The ideal way to perform this calculation would be to first calculate the center line, compare neighboring points to each other, and generate vectors based on these neighboring points. Each vector for a given center point should then be used to calculate a plane for which the vector is the normal of that plane. Now, using the segmented pixels (pixels in the aorta) that are in that plane, circle fitting can be used to calculate the approximate diameter.

## 3.8 Visualize Images & Aorta

Images can be viewed simply using `sitk_show()`, a function in `thesis.py` (Appendix B.1). Images had to be converted to NumPy arrays in order to plot them using the matplotlib library. The `thesis.py` file creates various files containing the coordinates of the aorta, a file with a maximum diameter for each slice, and a list of images that contain arteries. These files can be visualized using `showme.py` (Appendix B.3). Figure 3.3 is an example of what the 3D visualization looks like.

I started this project in a virtual machine environment but realized it wasnt suitable since it didnt have a graphics card for visualization. I ultimately acquired root access on one of the graphic computers in the lab (graphic21) which allowed for a 3D visualization and the ability to show images on the screen. As a side-note for future users, graphic21 was not physically available so I had to ssh into the computer. The computer I ssh-ed from in the lab had to have a NVIDIA graphics card or else the visualizations would not work. Lastly, Aeskulap [24] was used as a DICOM viewer.

# Chapter 4

# Results & Discussion

Once finished running, `thesis.py` generates several output files:

- `Clx.txt`, `cly.txt`, and `clz.txt` each contain the x-, y-, and z- coordinates respectively for the center line of the segmented section.
- `Diameters.txt` contains the maximum calculated diameter for each image that was segmented (by sampling, not circle fitting; circle was implemented in a separate test file for comparison).
- `Out.txt` contains the experiment information and the ratio calculations for each image. A double asterisk means the ratio was above the threshold and picked out as having an artery.
- `Xs.txt`, `ys.txt`, and `zs.txt` contain the respective coordinates for every pixel that was segmented and considered to be inside the aorta.

The following sections may reference these files and will discuss the results of the segmentation, center line calculation, artery detection, and diameter calculation experiments.

## 4.1 Segmentation

Segmentation is typically successful in the aortic region but suffers once the propagation hits the aortic bifurcation. 252 of 275 (92%) required images from patient #1 and 485 of 545 (89%) required images from patient #2 were successfully segmented and the entire aortic region was reached in both. Required images include any images in the aorta and images between the first aortic bifurcation and the next split in each femoral artery. All of the images that were omitted in the automatic segmentation of these two data sets were below the aortic bifurcation.

The reason why one data set 'requires' more images than the other is simply because there were more images available for one set. Realistically, the analysis would not require all of the aortic images but the segmentation would ideally, nonetheless, certainly be capable of segmenting the entire abdominal aortic region.

The data set from patient #3 struggled with the spine in the abdominal aortic region, rendering that data set ineffectual. Only 71 images were automatically segmented from patient #3 in the best attempt (some starting images were more successful than others allowing for further propagation). Figure 4.1 (A), (B), and (C) show the automatic segmentations for each patient.

The best image to start with was often one near the aortic aneurysm since it is, in these data sets, located between the renal arteries and the aortic bifurcation. Since measurements are required from all of these entities, the aortic aneurysm is one of the ideal places to start typically. Starting in the femoral arteries can also be successful since the propagation does not then have to handle the splitting of an artery. The propagation was typically more successful with the joining of two arteries than the splitting of one.

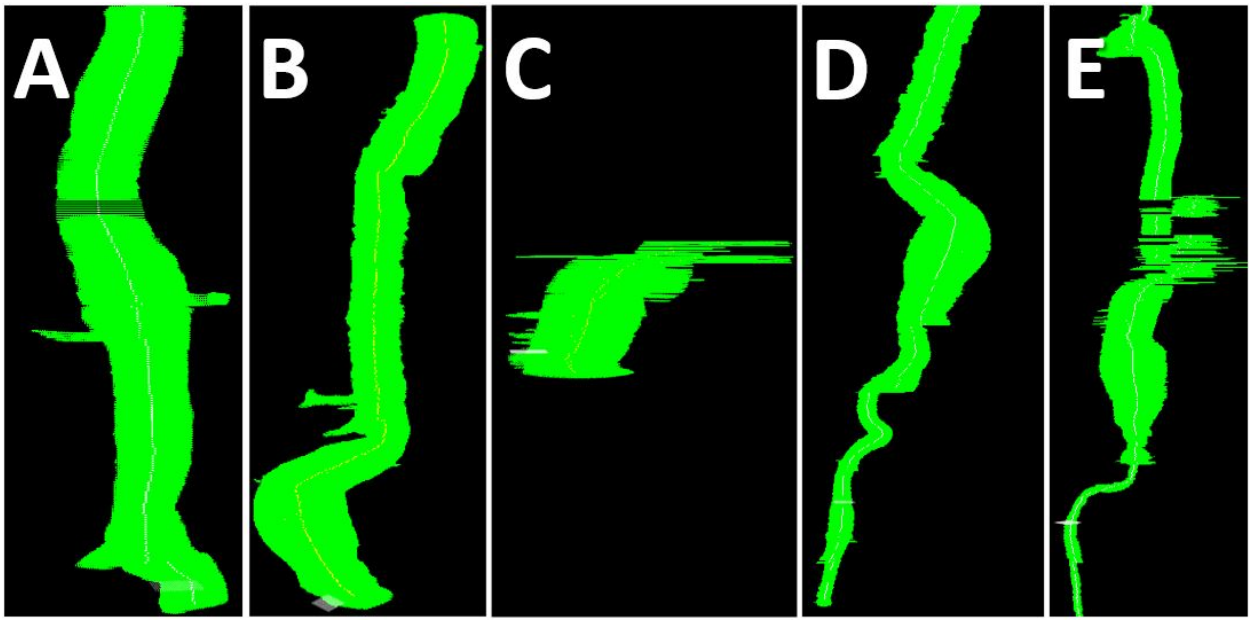



**Figure 4.1:** Automatic segmentation 3D visualizations for (A) patient #1 (B) patient #2 and (C) patient #3. Semi-automatic segmentation 3D visualizations for (D) patient #2 and (E) patient #3.

Segmentation was attempted in full on the data from patient #2 and patient #3. One femoral artery was followed all the way through by having the user click on the image when the propagation got stuck. Figure 4.1 (D) and (E) display the extent to which that solution worked despite encountering the spine and aortic bifurcation. It is a temporary solution to gather the data for further testing (i.e. on the femoral arteries) and would not be ideal for a fully automated solution.

Propagation and segmentation really struggled in four situations:

- poor segmentation of the aorta (not filling in the entire shape)
- unusual segmentation shapes that resulted in the centroid falling outside the shape
- following both femoral arteries
- differentiating between the aorta and other bodies of the same color next to the aorta

Figure 3.5 shows examples of all of these cases. However, segmentation was very successful in cases where the shape of the artery was mostly round (even if there were arteries stemming off the side). It worked well in the femoral arteries too when they maintained a relatively circular or elliptical shape and would perform better if starting lower in the artery. Results in Appendix A show which images chosen correctly and incorrectly as having arteries present.

## 4.2 Center Line Calculation

Of the 252 successfully segmented images from patient #1, four arteries were identified (see Section 4.3). Patient #2 only identified two arteries so patient #1 was chosen as the data set for exemplifying the center line results. Patient #3's segmentation was insufficient for analysis.

Each artery of the four arteries found in the patient #1 data set was identified in about four separate images. For example, images 297 - 300 all had an artery present and they were all a part of the same artery. A breakdown of the image ranges and arteries identified can be seen in the raw data output in Figure 4.2 and in Appendix A.

The image numbers on the left are identified as having arteries present and are output by the program in the file `results.txt`. The notes on the right are added in after checking each image manually. The aortic bifurcation was considered successful in a few images since it is in fact an artery with another artery present. Since this is a bit of an ambiguous situation, it would be better if splitting arteries could be identified by a separate algorithm.

```
Images with arteries are on the left, notes are
on the right. If a number is listed on the right,
that means it probably should have been listed as
having an artery.

168 ellipse x
169 ellipse x
170 ellipse x
171 ellipse x
172 ellipse x
173 ellipse x
174 ellipse x

189 (aortic bifurcation)
190 (aortic bifurcation)
191 aorta becoming two femoral arteries x
192 aorta becoming two femoral arteries x
193 aorta becoming two femoral arteries x

283
284
285
286

288
289
290
291
292
293
    missed 294 x
    missed 295 x
    missed 296 x
297
298
299
300

319
320
321
322
323
```

**Figure 4.2:** Numbers on the left were identified as having arteries. Every segmented image was then checked manually and noted on the right. If there are no notes, then it was correctly identified as an artery.

A sample of each of the four arteries have been chosen for in-depth analysis and comparison: 284, 290, 300, and 321. Image 310 was chosen as a control, an example of an image with no artery present for comparison.

**Control Image** The center line for any image that is not identified as having an artery is calculated by taking the center of gravity of the segmented region (see Section 3.4 for more information on this method).

Figure 4.3 (A) shows image #310 with the calculated centroid as a white cross in the center. The ellipse with the cross was generated manually, estimating by eye the approximate, preferred aortic region and the ideal centroid. The center of this cross appears to be within a few pixels of the calculated centroid. Figure 4.3 (B) shows the area that was segmented and (C) shows the histogram of BFS distances from the centroid to each pixel. The shape of this histogram is similar to the histogram for a perfect circle in Figure 3.12.

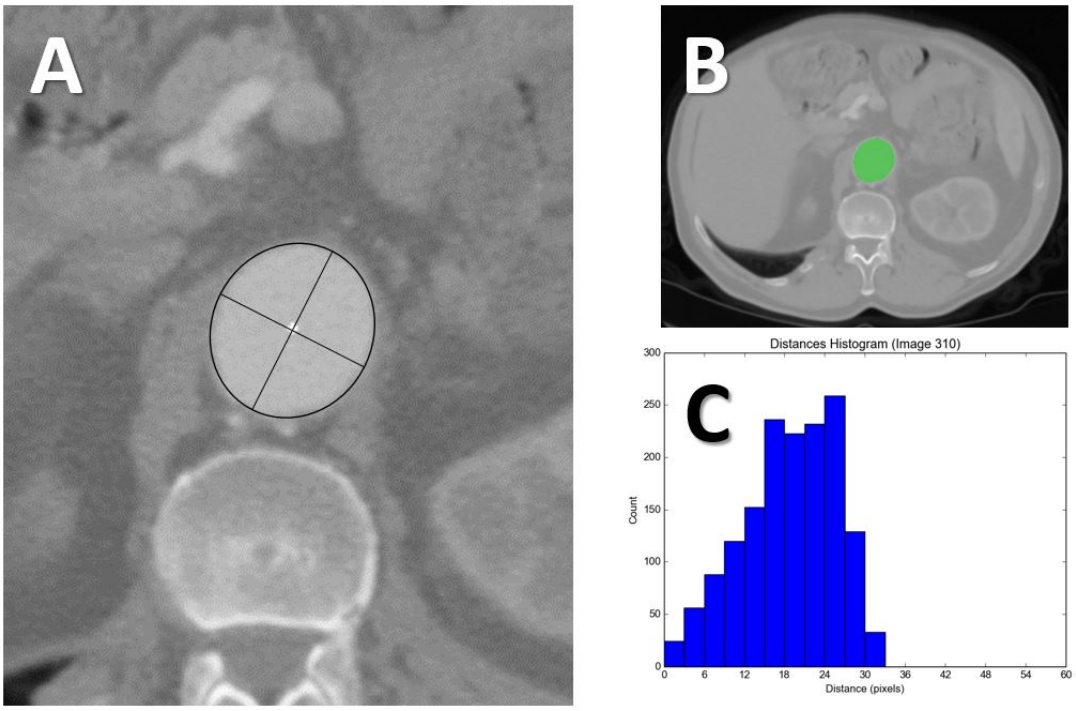



**Figure 4.3:** Image #310 (A) Original image centroid of the the green region in B as a white dot. An ellipse has been overlaid for comparison. (B) The segmented region (green) of the original image. (C) The histogram of BFS distances from the centroid to each pixel for the segmented region in B.

**Fixing the Centroid** If an image contains an artery then the centroid will be biased toward that artery. Thus, images 284, 290, 300, and 321 (and any other images in the data set with arteries) must have their respective centroid adjusted. The method for this was explained in Section 3.6.

The results of this method are shown in Figures 4.4 - 4.7. (B) shows the segmented area in green while (C) shows the trimmed area in white. The original centroid calculated from (B) is shown in (A) as a black dot (note that, in Figure 4.4, the white dot is in the same spot as the black dot and has thus been overwritten) while the centroid from (C) is shown in (A) as a white dot.

In each figure, (A) contains an overlaid ellipse as well which was added manually, estimating by eye the ideal region of the aorta and the ideal centroid. In each figure, the cross appears to be just a few pixels from the calculated centroid. In one of the images (Figure 4.4, image 284), the centroid does not change but still seems accurate based on the overlaid ellipse.

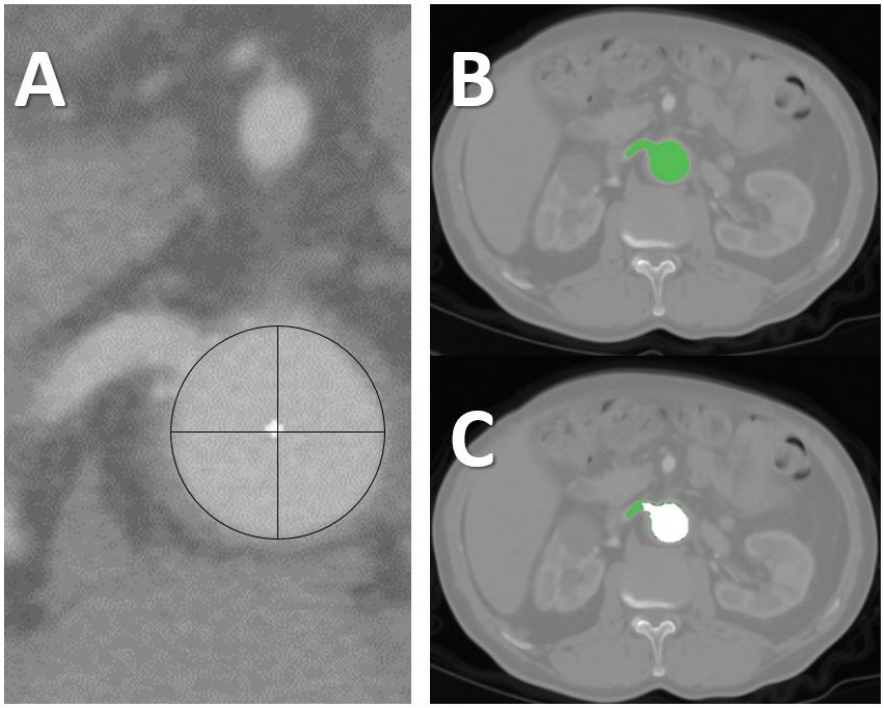



**Figure 4.4:** Image #284 (A) Original image centroid of the the green region in B as a black dot (overwritten by the white dot) and the centroid of the white region in C as a white dot. An ellipse has been overlaid for comparison. (B) The segmented region (green) of the original image. (C) The trimmed down region of the segmentation (white).

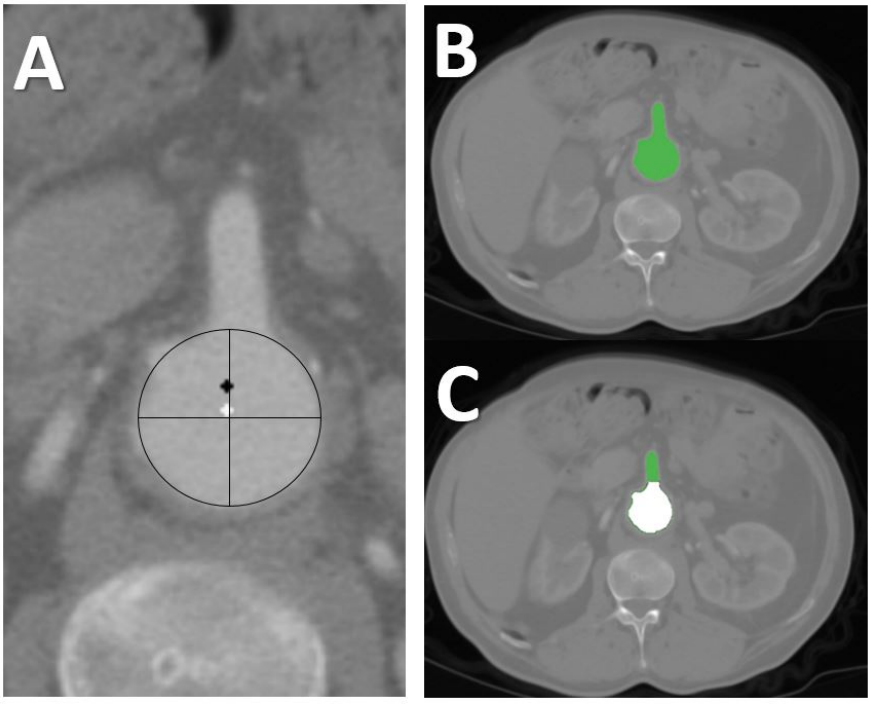



**Figure 4.5:** Image #290 (A) Original image centroid of the the green region in B as a black dot and the centroid of the white region in C as a white dot. An ellipse has been overlaid for comparison. (B) The segmented region (green) of the original image. (C) The trimmed down region of the segmentation (white).

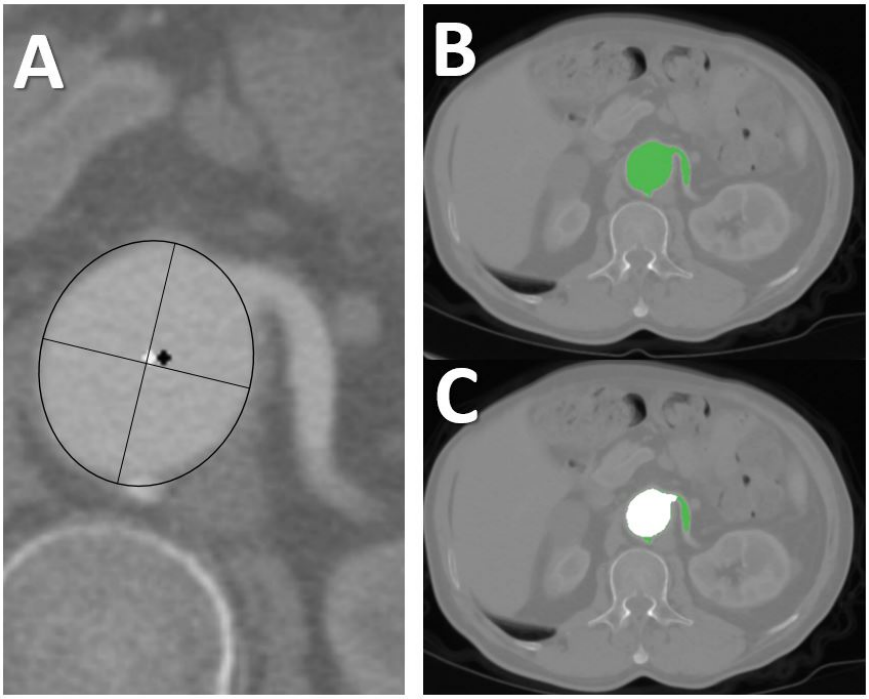



**Figure 4.6:** Image #300 (A) Original image centroid of the the green region in B as a black dot and the centroid of the white region in C as a white dot. An ellipse has been overlaid for comparison. (B) The segmented region (green) of the original image. (C) The trimmed down region of the segmentation (white).

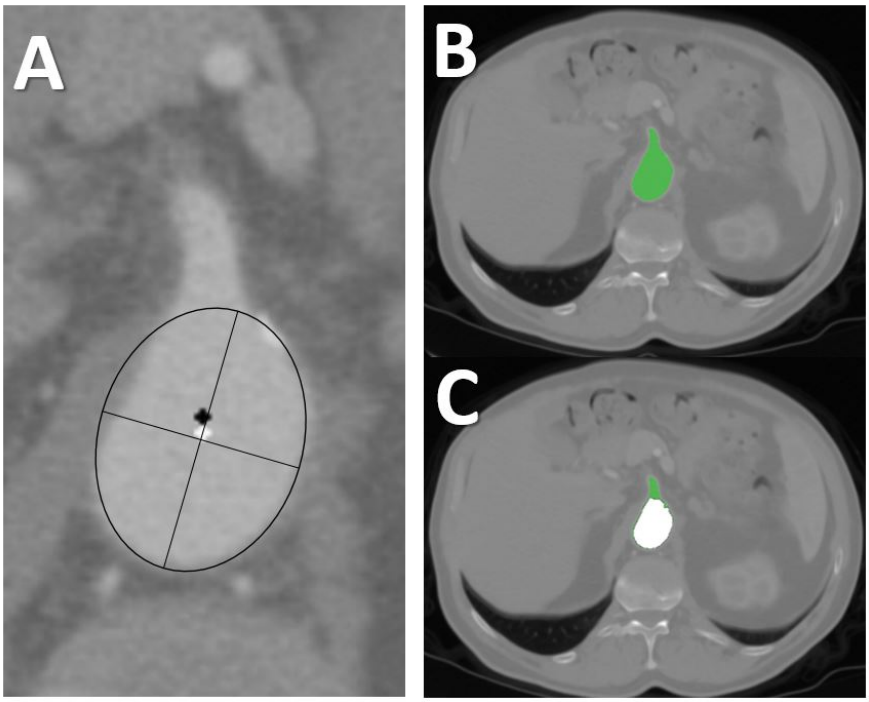



**Figure 4.7:** Image #321 (A) Original image centroid of the the green region in B as a black dot and the centroid of the white region in C as a white dot. An ellipse has been overlaid for comparison. (B) The segmented region (green) of the original image. (C) The trimmed down region of the segmentation (white).

**Circle Fitting** All of these calculations were then assessed a second time by comparing them to a circle fit. Instead of using the entire segmentation, the edge points of the trimmed aorta (if an artery was present, otherwise just the original edge points) were extracted and then circle fitted as explained in Section 3.6.1.

The three methods used to calculate a circle produced identical center points (see Appendix A.1.1 for results). These center points are compared with the center-of-gravity calculation (centroid) in Figure 4.8. A few of these fittings (#300, #321, and #310) are depicted in Figure 4.9. Figures 4.10 and 4.11 show how similar the center points were between the circle fitting and center-of-gravity calculations for patient #1 and patient #2, respectively. The x-coordinates and y-coordinates are plotted separately and are typically within a few pixels of each other.

| Image Number | Center of Gravity | Circle Fitting Center | Difference |
|---|---|---|---|
| **284** | (285, 234) | (284.84, 234.50) | (+0.84, +0.50) |
| **290** | (284, 233) | (284.86, 233.35) | (+0.86, +0.35) |
| **300** | (284, 236) | (284.66, 236.10) | (+0.66, +0.10) |
| **310** | (281, 241) | (281.11, 241.76) | (+0.11, +0.76) |
| **321** | (276, 244) | (276.45, 244.36) | (+0.45, +0.36) |

**Figure 4.8:** A table of five images that were sampled, circle fitted, and compared with the original centroid calculation (using the equation for a center of gravity). All of the images except for 310 had arteries that were trimmed before the calculation. Summarized from Appendix A.1.1.

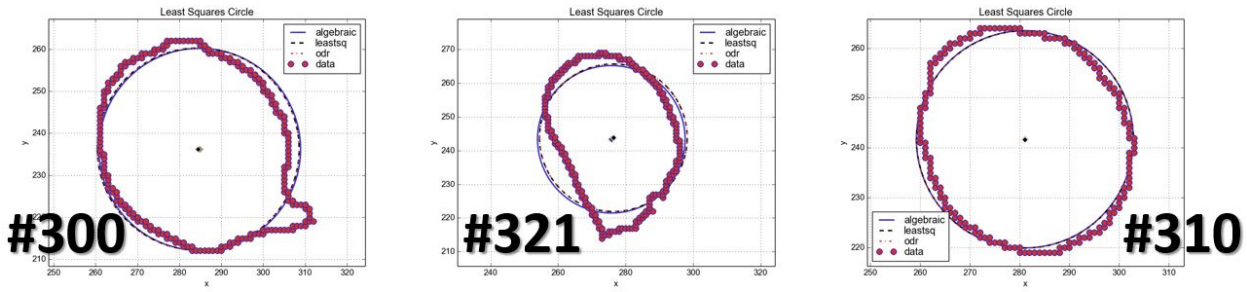



**Figure 4.9:** Three examples from patient #1 of circle fitting the edge of a segmentation. Two images have arteries trimmed, 300 and 321, and one image has no trimming, 310. Edge points are shown in red ('data') while various circle fitting methods are shown as three differently styled lines.

**Visualization** The overall center line is finally plotted in relation to the aorta from patient #1 in Figure 4.12 and Figure 4.13. The two images show the same aorta but one of them is rotated 90 degrees relative to the other one so that all four arteries can be seen. Qualitatively, the line is not perfect but is significantly less biased than it otherwise would be. Patient #2's segmentation can also be seen in Figure 4.1 (B).

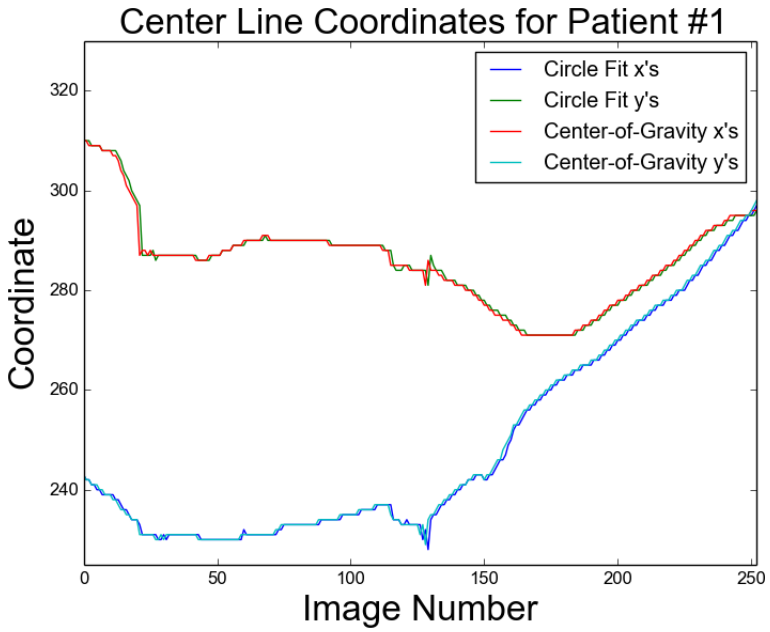



**Figure 4.10:** Centroid coordinates (pixels) for each segmented image from patient #1 as calculated by circle fitting and the center-of-gravity equation. X-coordinates and y-coordinates were plotted separately for comparison. The image number corresponds to the image number of the segmented set (hence why images such as #284, #290, etc. appear to be off the scale). The z-coordinate corresponds to the image number.

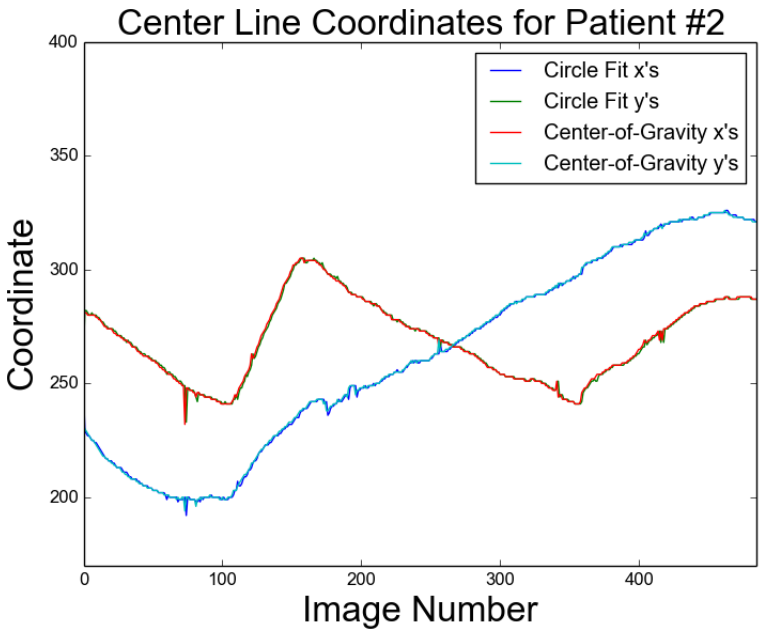



**Figure 4.11:** Centroid coordinates (pixels) for each segmented image from patient #2 as calculated by circle fitting and the center-of-gravity equation. X-coordinates and y-coordinates were plotted separately for comparison. The image number corresponds to the image number of the segmented set. The z-coordinate corresponds to the image number.

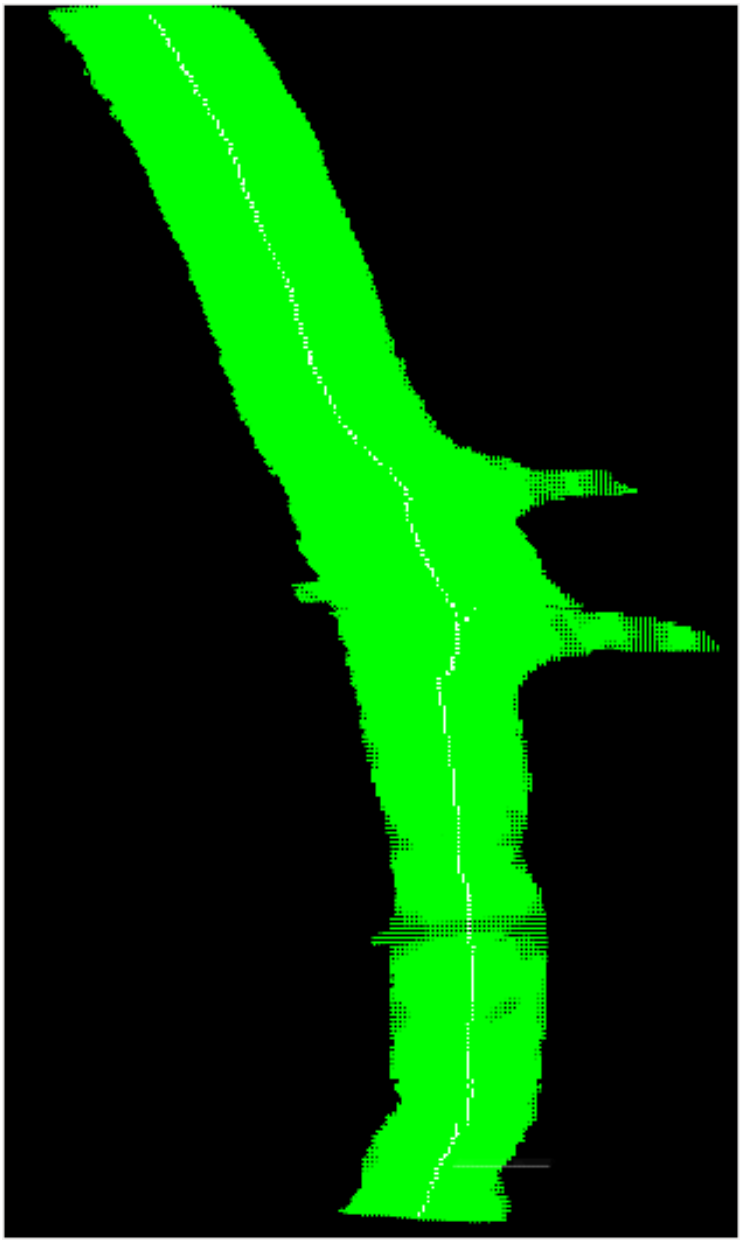


**Figure 4.12:** A 3D visualization of the abdominal aortic region of patient #2. The black shading should be ignored.

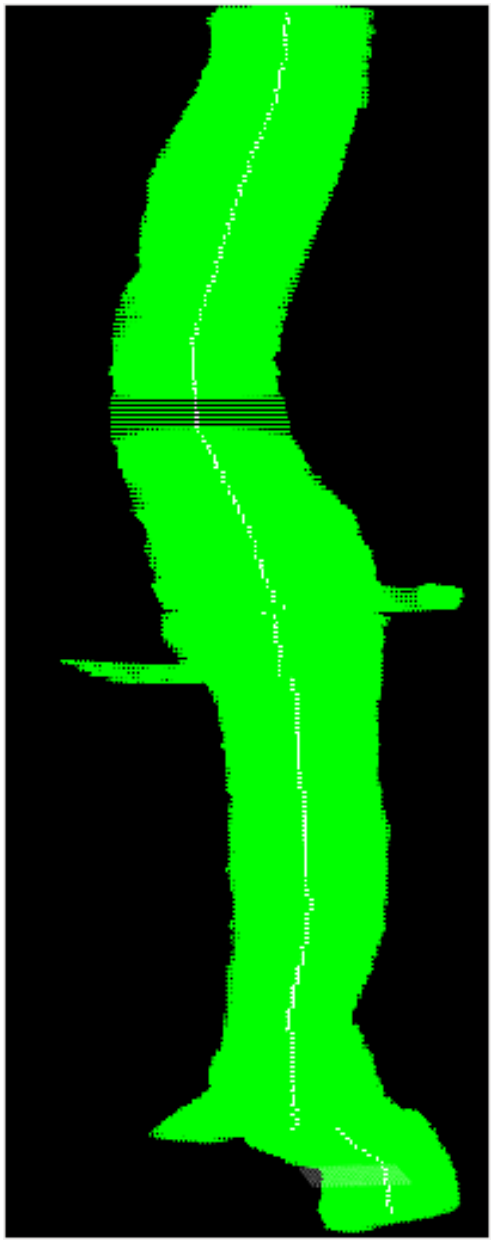


**Figure 4.13:** The visualization in Figure 4.12, rotated ninety degrees.

As a side note, the green area in Figure 4.12 and Figure 4.13 is simply many green dots all put together so the black shading that appears is only there because it's the visualization's way of saying we are looking directly at that plane of dots and we can see the space between the two planes/images/slices. The center line is also not consistent at the aortic bifurcation. It is important to note this pitfall. One way of solving this in the future would be to smooth the center line between the center line in the femoral artery and the center line in the aorta.

## 4.3 Artery Identification

Two of the three given data sets were segmented well enough to test for artery identification. 252 images in the first data set (patient #1) were segmented and 485 images were segmented in the second set (patient #2). Of the 737 images, 716 of them were successfully identified as having/not having an artery present. The results are summarized below.

Patient #1:

- 252 images analyzed in successfully segmented data set
- 95% (239/252 images) were correctly identified
- 5% (13/252 images) identified as having arteries when most of them were, in fact, just ellipse-shaped femoral arteries

Patient #2:

- 485 images analyzed in successfully segmented data set
- 98% (477/485) were correctly identified
- 2% (8 images) were either not identified or falsely identified because:
  - the artery wasnt segmented at all
  - partial segmentation
  - the segmented artery was not large enough to be noticed
- Figure 4.14 exhibits these difficulties

While these results are great for a first attempt at this algorithm, it is important for all of the arteries to be identified. Patient #1 successfully identified the four main arteries (as shown in Figure 4.2 and Figures 4.4 - 4.7) typically seen but patient #2 missed out on two arteries, only identifying the other two. The two missed arteries are shown in Figure 4.14. Segmentations from the two identified arteries from patient #2 can be seen in Figures 4.15 and 4.16. Overall, these failures are more related to the segmentation than the algorithm itself but must nonetheless be mitigated in the future.

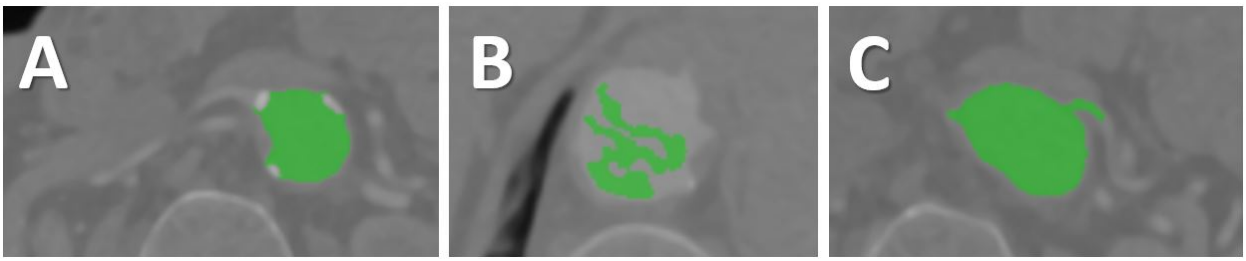



**Figure 4.14:** (A) A white calcification blocks segmentation from the artery. (B) Partial segmentation of the aorta. (C) Artery is not identified because the segmentation does not extend far enough.

The segmentation process for patient #3 was so poor that evaluating the results of the artery detection is futile. Most of the images that were picked out were picked out for having strange shapes due to extraneous bodies being selected in addition to the aorta (as in Figure 3.4 (C)). This emphasizes the importance that should be placed on an accurate and reliable segmentation process. All three patients' results can be found in Appendix A.

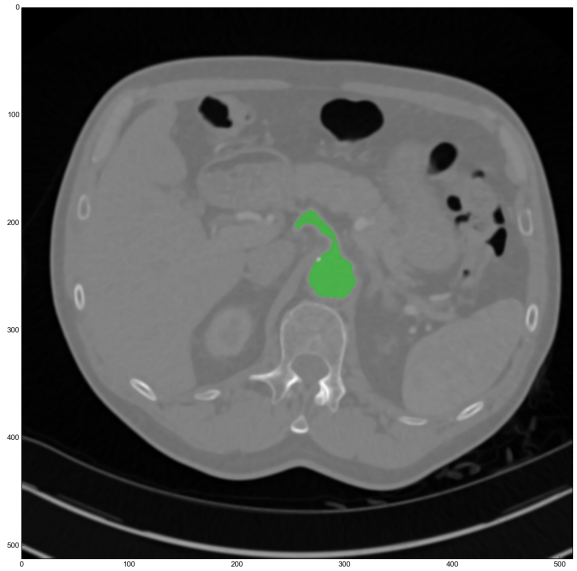


**Figure 4.15:** Successfully identified artery from patient #2 (Image #478).

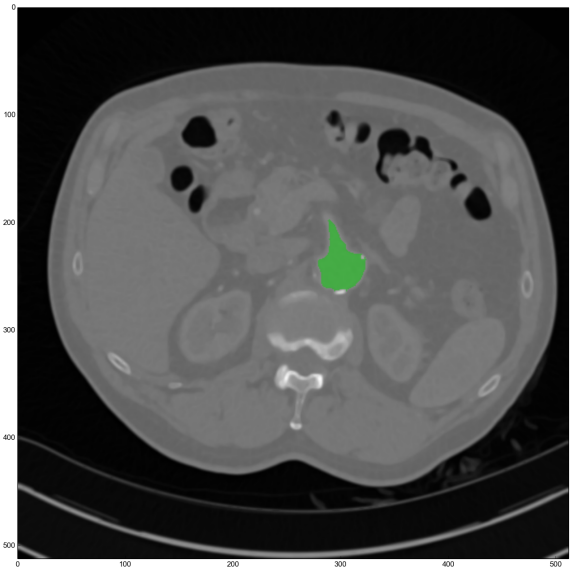


**Figure 4.16:** Successfully identified artery from patient #2 (Image #455).

## 4.4 Diameter Calculation

**Calculation** The diameter of the aorta was calculated for each image using three methods: edge sampling, circle fitting, and by area. These methods are described in Section 3.7. Ideally, this would not be done on each image but rather for each centroid. Each centroid needs to create a vector with the next centroid and then a plane is required whereby the vector is the normal of that plane. Then, the diameter should be calculated using pixels of the aorta that fall within that plane. This was not completed in time for this report. Regardless, the three methods for calculating the diameter were tested and compared on each image.

**Results** Figure 4.17 shows the diameter (in pixels) for each image as calculated by the three discussed methods (circle fitting, area, and edge sampling). Rather than the image number corresponding to the image in the original data set, the first image is now the first in the segmented set. Hence, this is why image #300 and other previously mentioned images are beyond the 252 x-axis limit.

**Discussion** Sudden and extreme troughs in the graph, after investigating the corresponding images, seem to be images that were only partially segmented. This makes sense since the areas of those segmentations are significantly smaller and the edge points are very close to the centroids. This is an interesting way to see how often such an event occurs (severely, once for patient #1 and twice for patient #2).

Other key events have been indicated for patient #1. There is a significant spike in the sampling method in a few locations. One of these was the aortic bifurcation, one was a section of the aorta that began to take a round, square-like shape, and another spike was due to the presence of one or more arteries. The sampling method was consistently higher than the other methods, suggesting it is likely not an ideal method for this calculation. The ‘circle fit’ and ‘area’ lines appear in both

diagrams to be nearly identical.

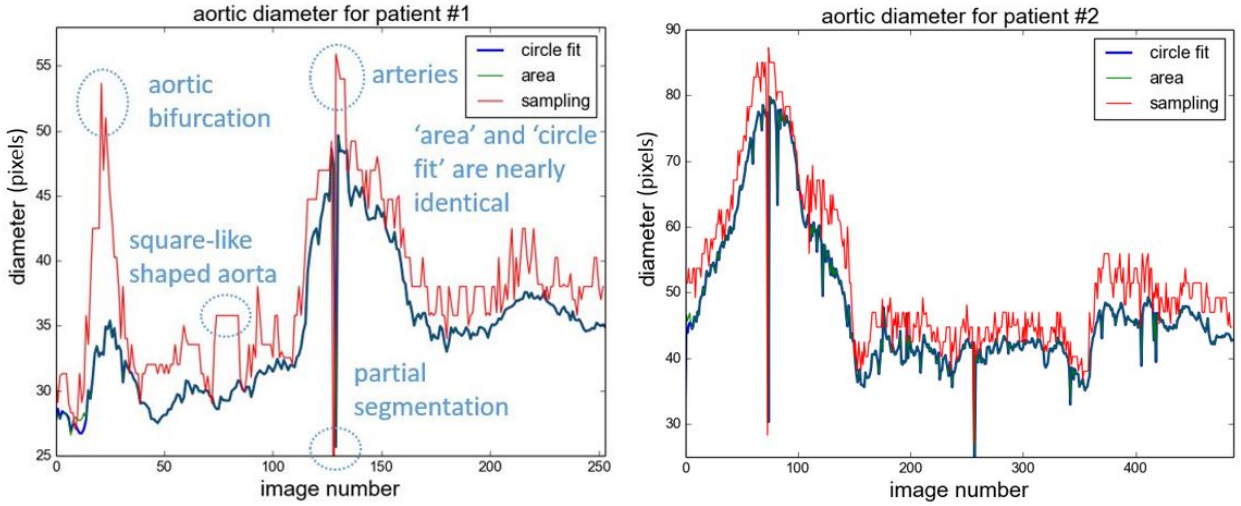



**Figure 4.17:** Diameter as a function of image number with patient #1 on the left and patient #2 on the right. The diameter was calculated using three methods, circle fitting (blue), area (green), and edge sampling (red). Extreme troughs indicate a partial (poor) segmentation. Other notable events have been labeled as well. The blue and green lines tend to nearly overlap. n.b. The x-axis is for image numbers based on the segmented set, not the original set. Additionally, the y-axis limit is larger for patient #2; recall that patient #2 has a larger segmented set of images.

These results suggest that calculating the diameter by circle fitting or by area would both be reasonable methods for future methods. As a next step, a test should be done where instead of calculating the diameter on the image, the plane orthogonal to the vector of the centroid should be used to select the appropriate pixels. Then these methods should be tested on those pixels for each centroid and compared with the above results.

# Chapter 5

# Conclusion & Future Work

## 5.1 Conclusion

### 5.1.1 Segmentation

Segmentation is the process of extracting, in this case, the aorta from the set of raw images. The program begins by taking a click from the user indicating the position of the aorta and calculating the centroid of that slice of the aorta. This centroid is used to propagate to the following images until either the new segmentation does not sufficiently overlap with the prior segmentation or the end of the data set is reached.

This implementation of a segmentation algorithm is successful in the abdominal aortic region if the aorta is sufficiently differently colored. It struggles with the femoral arteries since it can only follow one of them. It also has difficulties when other bodies around the aorta are of similar color and close by. This data set in particular experienced issues with the spine.

Lastly, the SimpleITK segmentation function may not have been the best one to use. A customized edge-finding function could be more practical for more accurate segmentations. The importance of an accurate and reliable segmentation function was realized in the center line and artery identification algorithm tests.

### 5.1.2 Center Line Calculation

The center line is a crucial part of the process for calculating lengths and diameters of the aorta. It is calculated by finding the centroid of each image, deciding if that image contains an artery, and then readjusting the centroid if it is biased by an artery. The centroid is calculated in two different ways for comparison: circle fitting and a center-of-gravity calculation.

In images with an artery present, the centroid is first estimated using the center of gravity calculation. From there a sample of edge points are taken, outliers are removed, and a new centroid is calculated from these points. Then each method can be run to find a more precise centroid.

Both methods produce very similar results suggesting that either method could be used in the future. A sample of these results were tested manually as an additional means of verification. While the center line estimation implemented is reasonably accurate, the overall picture (as seen in Figures 4.12 and 4.13) does show a few stray points which could be perfected.

### 5.1.3 Artery Identification

Artery identification is necessary for calculating the distance between the renal arteries and the aneurysm as well as the distance from the renal arteries to the aortic bifurcation. The method for this begins by calculating the BFS distance from the centroid to each pixel in the segmented region. A list of these distances is compiled into a sorted list.

The lower third of the list is averaged and the upper third of the list is averaged. These two numbers create a ratio that is then compared with the ratio that a perfect circle would produce. A threshold just above the maximum ratio possible for a circle decides whether the calculated value is from an image with an artery or not.

Artery identification works very well in the aorta when the segmentation is reasonably accurate. The main issues with the algorithm involve segmentation. If the segmentation does not pick up the entire or aorta or excludes the artery that is present, then the artery identification algorithm fails.

This algorithm is also not practical in the femoral arteries. Femoral arteries can take very strange shapes and do not frequently have large arteries stemming from them. A different algorithm should ideally be used specifically for identifying when an artery splits in two.

### 5.1.4 Diameter Calculation

Various diameters throughout the aorta and femoral arteries are required for the FEVAR graft fabrication. Diameter calculations in this report were performed using three methods: edge sampling, circle fitting, and calculating by area. Tests were run on each segmented image as it was, without taking the angle of the aorta into account, making the measurements almost certainly false. However, it was, nonetheless, an appropriate test for comparing the three methods.

Circle fitting and calculating by area proved to be very comparable ways of finding the diameter of a given shape as their results were so similar. Edge sampling, however, was rather inconsistent and would not be recommended for the future. Further testing to see if the other two methods are still appropriate when the orientation of the aorta is taken into account is still needed.

## 5.2 Future Work

### 5.2.1 Segmentation

In the future, a bottom-up approach might be best. Start with one of the images that falls after the aorta splits into two femoral arteries–perhaps one with four femoral arteries present (so, after another split). Click on all four and propagate upwards. This would ensure the entire required region is reached and would allow for the center line to be smoothed through the bifurcations.

Another thought would be to propagate from anywhere and use a sampling of points to move from image to image. Then segment each point and test it's segmentation for overlap with the last segmentation. This would make things easier on the user, certainly, and still allow for the center line smoothing idea to be implemented. A separate function would simply have to be used to start from the lowest number image and recalculate the center line from there.

Other types of segmentation that are already out there might also be useful such as model based segmentation. Researchers at University Medical Center Utrecht have, for example, implemented an active-shape-model-based segmentation. This method takes advantage of the sequential slices and uses the prior slice's shape to maximize the fit of the segmentation shape in the next image [25].

In order to prevent the spine collision, it might be necessary to implement the solution developed by Mukai [22] and subtract the spine before segmenting the aorta. Otherwise, it might be necessary to only use data sets with enough of a color difference or enough distance between the aorta and spine that they do not collide.

### 5.2.2 Center Line Calculation

Since circle fitting seemed to work well and the manual way of examining the images was to 'fit' an ellipse over the image, ellipse fitting should be considered to improve the center line calculation. It might also be interesting to see how using a Gaussian distribution to perform the histogram trimming might compare with the current method.

Another problem to overcome is the center line's disruption whenever the artery splits in two. This could be compensated for by finding the center line of each artery and then, as they merge, bringing the centroids together slowly–perhaps by only allowing the centroid to travel a few pixels at a time or by attempting to keep the center line(s) parallel to the outer edges. Or, instead of only allowing a few pixels of travel, a moving average could be used. This would likely remove a lot of the stray centroids. This method could also be used to potentially improve the segmentation results as well.

### 5.2.3 Artery Identification

The key to success with the current algorithm is having an accurate segmentation but the current algorithm could still be built upon. It would be beneficial to have a specialized algorithm for the femoral arteries, capable of knowing when they are splitting. This would likely involve noticing when there are two bodies in one image that both sufficiently overlap with the prior image and were connected.

It would also be interesting to see how the algorithm works when using a different plane than that of the original image. The algorithm could be tested by using the plane that is perpendicular to the center line vector at that point. Since this algorithm was used to help find the center line, a different method for finding the center line would be needed.

### 5.2.4 Diameter Calculation

Circle fitting and calculating by area produced comparable diameters while edge sampling seemed inconsistent. This suggests that in the future, circle fitting or calculating by area would be reasonable choices. The methods would need to be modified, however, to first find the plane that is orthogonal to the vector of the centroid of that image. The vector would be calculated based on neighboring centroids. Then all of the pixels in that plane would be used for the calculation rather than the plane of the image (unless, of course, the aorta happens to be perfectly vertical at that slice). While the aorta is often somewhat vertical, this method would take into account the orientation of the aorta and be significantly more accurate.

# Appendices

# Appendix A

# Results

## A.1 Patient #1

### A.1.1 Circle Fitting: `out.txt`

`Out.txt` is created by running the main script from the command line like so:

```
python thesis.py > out.txt
```

For the circle fitting tests, the centroids were printed to this file. An excerpt has been shown below where the centroid is first calculated for the untrimmed set of edge points (first table) and then it is calculated for the trimmed set (second table). Each table lists the centroid in the Xc and Yc columns as well as the radius in the Rc column. The final trimmed center of gravity that was calculated without circle fitting is also shown at the bottom of the second table. These files, shown in Figures A.1 - A.5 have been slightly modified and formatted for legibility.

```
     title:  exp
start time:  09:08:52.338900
dicom path:  patient1/SDY00000/SRS00004
    leeway:  60
  idxSlice:  284
 threshold:  2.845
  comments:  circle fitting test

reading dicom series...
getting user input...
Method 2b :
Functions calls : f_2b=9 Df_2b=7

Method 3b :  odr with jacobian

Functions calls : f_3b=24 jacb=23 jacd=23

METHOD                        Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
------------------------------------------------------------------------------------------------
algebraic               278.49648  234.25596   21.71083          1   6.163124 8394.485461 14714421.55
leastsq                 280.84930  233.99228   21.96141         21   5.946001 7813.440265 16217028.26
leastsq with jacobian   280.84930  233.99228   21.96141          9   5.946001 7813.440265 16217027.87
odr                     280.84932  233.99231   21.96141        134   5.946001 7813.440265 16217053.62
odr with jacobian       280.84932  233.99231   21.96141         24   5.946001 7813.440265 16217054.80
Method 2b :
Functions calls : f_2b=8 Df_2b=6

Method 3b :  odr with jacobian

Functions calls : f_3b=19 jacb=18 jacd=18

METHOD                        Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
------------------------------------------------------------------------------------------------
algebraic               283.92775  233.99399   19.98723          1   3.576356 2315.048657 4107457.12
leastsq                 284.84392  234.49978   20.07406         18   3.495055 2210.989659 4316312.77
leastsq with jacobian   284.84392  234.49978   20.07405          8   3.495055 2210.989659 4316312.74
odr                     284.84392  234.49979   20.07406        104   3.495055 2210.989659 4316315.22
odr with jacobian       284.84392  234.49979   20.07406         19   3.495055 2210.989659 4316315.29

center-of-gravity       285        234
```

**Figure A.1:** The output file for patient #1, image #284. Note the Xc and Yc values from various circle fitting methods compared with the center-of-gravity calculation.

```
     title:  exp
start time:  11:43:32.710792
dicom path:  patient1/SDY00000/SRS00004
    leeway:  60
  idxSlice:  290
 threshold:  2.845
  comments:  circle fitting test

reading dicom series...
getting user input...
Method 2b :
Functions calls : f_2b=9 Df_2b=7

Method 3b :  odr with jacobian

Functions calls : f_3b=37 jacb=35 jacd=35

METHOD                        Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
------------------------------------------------------------------------------------------------
algebraic               285.20067  222.64825   25.24906          1   8.524896 17805.094649 46404685.29
leastsq                 285.30249  226.28825   25.55176         21   8.131960 16201.550774 53122081.72
leastsq with jacobian   285.30249  226.28825   25.55176          9   8.131960 16201.550774 53122080.35
odr                     285.30250  226.28835   25.55177        207   8.131960 16201.550773 53122471.34
odr with jacobian       285.30250  226.28835   25.55177         37   8.131960 16201.550773 53122472.16

Method 2b :
Functions calls : f_2b=7 Df_2b=5

Method 3b :  odr with jacobian

Functions calls : f_3b=17 jacb=16 jacd=16

METHOD                        Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
------------------------------------------------------------------------------------------------
algebraic               284.86556  232.99001   21.95834          1   1.984928 744.648205 1502812.27
leastsq                 284.86917  233.35608   21.98101         15   1.967788 731.843635 1529631.90
leastsq with jacobian   284.86917  233.35608   21.98101          7   1.967788 731.843635 1529631.89
odr                     284.86917  233.35608   21.98101         92   1.967788 731.843635 1529632.03
odr with jacobian       284.86917  233.35608   21.98101         17   1.967788 731.843635 1529632.03

center-of-gravity       284        233
```

**Figure A.2:** The output file for patient #1, image #290. Note the Xc and Yc values from various circle fitting methods compared with the center-of-gravity calculation.

```
     title:  exp
start time:  11:45:18.207817
dicom path:  patient1/SDY00000/SRS00004
    leeway:  60
  idxSlice:  300
 threshold:  2.845
  comments:  circle fitting test

reading dicom series...
getting user input...
Method 2b :
Functions calls : f_2b=8 Df_2b=6

Method 3b :  odr with jacobian

Functions calls : f_3b=21 jacb=20 jacd=20

METHOD                         Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
-------------------------------------------------------------------------------------------------------
algebraic               290.78067  238.25644   25.53500          1   5.157543 7740.671990 16969159.41
leastsq                 289.15732  237.09272   25.85042         18   4.985391 7232.548846 18415491.31
leastsq with jacobian   289.15732  237.09272   25.85042          8   4.985391 7232.548846 18415491.57
odr                     289.15730  237.09273   25.85042        116   4.985391 7232.548846 18415517.61
odr with jacobian       289.15730  237.09273   25.85042         21   4.985391 7232.548846 18415517.33

Method 2b :
Functions calls : f_2b=7 Df_2b=5

Method 3b :  odr with jacobian

Functions calls : f_3b=17 jacb=16 jacd=16

METHOD                         Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
-------------------------------------------------------------------------------------------------------
algebraic               285.07661  236.06185   23.39108          1   2.584323 1309.030589 3147401.30
leastsq                 284.66393  236.10319   23.40484         15   2.567862 1292.406889 3185454.20
leastsq with jacobian   284.66393  236.10319   23.40484          7   2.567862 1292.406889 3185454.20
odr                     284.66392  236.10320   23.40484         92   2.567862 1292.406889 3185456.96
odr with jacobian       284.66392  236.10320   23.40484         17   2.567862 1292.406889 3185456.99

center-of-gravity       284        236
```

**Figure A.3:** The output file for patient #1, image #300. Note the Xc and Yc values from various circle fitting methods compared with the center-of-gravity calculation.

```
     title:  exp
start time:  11:47:27.758304
dicom path:  patient1/SDY00000/SRS00004
    leeway:  60
  idxSlice:  310
 threshold:  2.845
  comments:  circle fitting test

reading dicom series...
getting user input...
Method 2b :
Functions calls : f_2b=6 Df_2b=4

Method 3b :  odr with jacobian

Functions calls : f_3b=16 jacb=14 jacd=14

METHOD                         Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
-------------------------------------------------------------------------------------------------------
algebraic               281.09620  241.78357   22.16097          1   1.254157 279.977997  560145.04
leastsq                 281.11937  241.75928   22.16097         12   1.253936 279.879285  560360.61
leastsq with jacobian   281.11937  241.75928   22.16097          6   1.253936 279.879285  560360.61
odr                     281.11938  241.75928   22.16097         81   1.253936 279.879285  560360.65
odr with jacobian       281.11938  241.75928   22.16097         16   1.253936 279.879285  560360.65

center-of-gravity       281        241
```

**Figure A.4:** The output file for patient #1, image #310. Note the Xc and Yc values from various circle fitting methods compared with the center-of-gravity calculation. This image did not need to be trimmed so only one table was calculated.

```
     title:  exp
start time:  11:47:54.641979
dicom path:  patient1/SDY00000/SRS00004
    leeway:  60
  idxSlice:  321
 threshold:  2.845
  comments:  circle fitting test

reading dicom series...
getting user input...
Method 2b :
Functions calls : f_2b=9 Df_2b=7

Method 3b :  odr with jacobian

Functions calls : f_3b=31 jacb=30 jacd=30

METHOD                        Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
-------------------------------------------------------------------------------------------------
algebraic               274.32379  236.54381   23.82808          1   7.165939 11553.904233 26989806.84
leastsq                 276.45359  238.78315   23.94501         21   6.869574 10617.984555 29313503.20
leastsq with jacobian   276.45359  238.78315   23.94501          9   6.869574 10617.984555 29313502.58
odr                     276.45368  238.78325   23.94502        176   6.869574 10617.984555 29313714.84
odr with jacobian       276.45368  238.78325   23.94502         31   6.869574 10617.984555 29313715.14

Method 2b :
Functions calls : f_2b=8 Df_2b=6

Method 3b :  odr with jacobian

Functions calls : f_3b=18 jacb=17 jacd=17

METHOD                        Xc         Yc         Rc   nb_calls    std(Ri)     residu    residu2
-------------------------------------------------------------------------------------------------
algebraic               275.98133  244.08857   21.13127          1   2.956534 1582.138189 2681581.50
leastsq                 276.44892  244.35637   21.14203         18   2.932299 1556.306466 2717457.34
leastsq with jacobian   276.44892  244.35637   21.14203          8   2.932299 1556.306466 2717457.34
odr                     276.44892  244.35638   21.14203         98   2.932299 1556.306466 2717457.54
odr with jacobian       276.44892  244.35638   21.14203         18   2.932299 1556.306466 2717457.52

center-of-gravity       276        244
```

**Figure A.5:** The output file for patient #1, image #321. Note the Xc and Yc values from various circle fitting methods compared with the center-of-gravity calculation.

### A.1.2 Automatic Segmentation: `results.txt`

Figure A.6 shows the artery detection results for patient #1 after automatic segmentation. Images 283-286 are one artery, 288-293 are another, 297-300 are a third, and 319-323 are the last artery. Some ellipses were falsely identified and the aortic bifurcation was identified.

Two of those images were marked successful because it is where the aorta splits into two so it does technically have an artery stemming from the original area. In the future there should either be an algorithm to find the aortic bifurcation specifically or they should perhaps all be accepted.

```
Images with arteries are on the left, notes are
on the right. If a number is listed on the right,
that means it probably should have been listed as
having an artery.

168 ellipse x
169 ellipse x
170 ellipse x
171 ellipse x
172 ellipse x
173 ellipse x
174 ellipse x

189 (aortic bifurcation)
190 (aortic bifurcation)
191 aorta becoming two femoral arteries x
192 aorta becoming two femoral arteries x
193 aorta becoming two femoral arteries x

283
284
285
286

288
289
290
291
292
293
    missed 294 x
    missed 295 x
    missed 296 x
297
298
299
300

319
320
321
322
323
```

**Figure A.6:** Numbers on the left were identified as having arteries. Every segmented image was then checked manually and noted on the right. If there are no notes, then it was correctly identified as an artery.

## A.2 Patient #2

### A.2.1 Automatic Segmentation: `results.txt`

Figure A.7 shows the artery detection results for patient #2 after automatic segmentation. The arteries that were missed were missed because of poor segmentation. An artifact blocked the segmentation from expanding into the artery as in Figure 3.4 (A).

```
Images with arteries are on the left, notes are
on the right. If a number is listed on the right,
that means it probably should have been listed as
having an artery.

353     bad segmentation
        missed 439
453
454
        missed 455
        missed 456
        missed 470
476
477
478
479

621     bad segmentation
695     bad segmentation
697     bad segmentation
```

**Figure A.7:** Numbers on the left were identified as having arteries. Every segmented image was then checked manually and noted on the right. If there are no notes, then it was correctly identified as an artery.

## A.3 Patient #3

### A.3.1 Automatic Segmentation: `results.txt`

Figure A.8 shows the artery detection results for patient #3 after automatic segmentation.

```
Images with arteries are on the left, notes are
on the right. If a number is listed on the right,
that means it probably should have been listed as
having an artery.

        309 is a really bad segmentation
320     really bad segmentation
332     I'm not sure what that blob is
333     I'm not sure what that blob is
334     really bad segmentation
335     I'm not sure what that blob is
336     I'm not sure what that blob is
337     I'm not sure what that blob is
338     I'm not sure what that blob is
339     I'm not sure what that blob is
340     I'm not sure what that blob is
341     I'm not sure what that blob is
342     I'm not sure what that blob is
343     I think this one has an artery
344     I'm not sure what that blob is
345     I'm not sure what that blob is
346     I'm not sure what that blob is
347     I'm not sure what that blob is
348     I'm not sure what that blob is
349     This one might have an artery too
350     I'm not sure what that blob is
351     Really grabbed the spine here
352     Really grabbed the spine here
353     Really grabbed the spine here
        354 probably the worst segmentation
355     I don't know
357     Some spine
358     Some spine
359     Some spine
360     Spine
```

**Figure A.8:** Numbers on the left were identified as having arteries. Every segmented image was then checked manually and noted on the right. If there are no notes, then it was correctly identified as an artery.

# Appendix B

# Code

## B.1 thesis.py

```
# How to use SimpleITK to segment a DICOM image
# https://pyscience.wordpress.com/2014/10/19/image-segmentation-with-python-and-simpleitk/

import os, sys, SimpleITK, scipy.misc, dicom, png, math, datetime, operator
import helpers, calculate
import fit_circle as fc
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import trapz
from mpl_toolkits.mplot3d import Axes3D
from skimage import io, exposure, img_as_uint, img_as_float

# displays an sitk image and takes the coords of the user's click on the image
# http://stackoverflow.com/questions/25521120/store-mouse-click-event-coordinates-with-matplotlib
def sitk_get_click(img, title=None, margin=0.05, dpi=40):
    print ('getting user input...')
    nda = SimpleITK.GetArrayFromImage(img)
    figsize = (1 + margin) * nda.shape[0]/dpi, (1 + margin) * nda.shape[1]/dpi
    extent = (0, nda.shape[1], nda.shape[0], 0)
    fig = plt.figure(figsize=figsize, dpi=dpi)
    ax = fig.add_axes([margin, margin, 1 - 2*margin, 1 - 2*margin])
    cid = fig.canvas.mpl_connect('button_press_event', onclick)
    plt.set_cmap("gray")
    ax.imshow(nda,extent=extent,interpolation=None)
    if title:
        plt.title(title)

    plt.show()

# displays an sitk image
def sitk_show(img, title=None, margin=0.05, dpi=40):
    nda = SimpleITK.GetArrayFromImage(img)
    figsize = (1 + margin) * nda.shape[0]/dpi, (1 + margin) * nda.shape[1]/dpi
    extent = (0, nda.shape[1], nda.shape[0], 0)
    fig = plt.figure(figsize=figsize, dpi=dpi)
    ax = fig.add_axes([margin, margin, 1 - 2*margin, 1 - 2*margin])
    plt.set_cmap("gray")
    ax.imshow(nda,extent=extent,interpolation=None)
    if title:
```

```
40          plt.title(title)
41
42      plt.show()
43
44  # fills the holes in a segmented sitk image
45  def fill_holes(img,radius=[3]*4):
46      return SimpleITK.VotingBinaryHoleFilling(image1=img,
47                                               radius=[2]*3,
48                                               majorityThreshold=1,
49                                               backgroundValue=0,
50                                               foregroundValue=1)
51
52  # smooths an sitk image, pretty essential in order to segment the image
53  def smooth_img(img):
54      imgSmooth = SimpleITK.CurvatureFlow(image1=img,
55                                          timeStep=0.125,
56                                          numberOfIterations=5)
57      return imgSmooth
58
59  # segments an sitk image
60  def segment_img(img, lstSeeds, low=190, up=310):
61      try:
62          segmentImg = SimpleITK.ConnectedThreshold(image1=img,
63                                                    seedList=lstSeeds,
64                                                    lower=low,
65                                                    upper=up,
66                                                    replaceValue=1)
67      except:
68          sitk_show(img)
69      return segmentImg
70
71  # returns a list of coordinates (x,y,distance) where distance is the distance
72  # from xc,yc to x,y. The list is also sorted by distance. Distances is defined
73  # by a BFS (not the hypotenuse)
74  def calculate_distances(xc,yc,aorta):
75      coords = []      # list of coords (x,y,dist)
76      visited = []     # list of visited coords
77      q = [(xc,yc,0)] # queue
78      visited.append((xc,yc))
79
80      # bfs
81      while(q):
82          (x,y,dist) = q[0]
83          q.pop(0)
84          if (dist != 0.):
85              coords.append((x,y,dist))
86          for i in range(-1,2):
87              for j in range (-1,2):
88                  if (aorta[x+i,y+j] == 1 and (x+i,y+j) not in visited):
89                      if (abs(i) + abs(j) == 2):
90                          q.append((x+i, y+j, round(dist+1.414,3)))
91                      else:
92                          q.append((x+i, y+j, round(dist+1.000,3)))
93                  visited.append((x+i, y+j))
94      coords.sort(key=lambda x: float(x[2]))
95      return coords
96
97  # returns true if the image (aorta) contains an artery
98  def artery(xc, yc, aorta, threshold, z):
99      coords = calculate_distances(xc, yc, aorta)
```

```
100     dists = map(operator.itemgetter(2), coords)
101     dists.sort()
102     lower_dists = dists[:len(dists)/3]
103     upper_dists = dists[len(dists)*2/3:]
104 
105     if (lower_dists and upper_dists):
106         lower = float(sum(lower_dists))/float(len(lower_dists))
107         upper = float(sum(upper_dists))/float(len(upper_dists))
108         if (upper/lower > threshold):
109             print ('image ' + str(z) + '**   ' +
110                str(round(upper,2))+ '/' +
111                str(round(lower,2)) + ' = ' +
112                str(round(upper/lower, 4)))
113             return True
114         else:
115             print ('image ' + str(z) + '     ' +
116                    str(round(upper,2))+ '/' +
117                    str(round(lower,2)) + ' = ' +
118                    str(round(upper/lower, 4)))
119     return False
120 
121 # http://stackoverflow.com/questions/11686720/is-there-a-numpy-builtin-to-reject-
    outliers-from-a-list
122 # rejects outliers; code was modified from the above link
123 def reject_outliers(circle, new_aorta, m = 2.):
124     dists = map(operator.itemgetter(2), circle)
125     d = np.abs(dists - np.median(dists))
126     mdev = np.median(d)
127     s = d/mdev if mdev else 0.
128     new_dists = []
129     for i, num in enumerate(s):
130         if num < m:
131             new_dists.append(dists[i])
132     new_coords = []
133     for (x,y,r) in circle:
134        if r in new_dists:
135            new_coords.append((x,y,r))
136        else:
137            new_aorta[x,y] = 0
138     return new_coords
139 
140 # delete distance values that are greater than (mode + leeway) of the dists
141 def reject_over_mode(coords, new_aorta, leeway=15.):
142     dists = map(operator.itemgetter(2), coords)
143     counts = np.bincount(dists)
144     mode = np.argmax(counts)
145     new_coords = []
146     for (x,y,d) in coords:
147         if d < mode+leeway:
148             new_coords.append((x,y,d))
149         else:
150             new_aorta[x,y] = 0
151     return new_coords
152 
153 # calcualte the center of gravity of the points in coords (x,y,dist)
154 def calculate_cog(coords):
155     xc, yc, count = 0, 0, 0
156     for (x,y,_) in coords:
157         xc += x
158         yc += y
```

```
159             count+= 1
160         if (not count):
161             return -1,-1
162         xc /= count
163         yc /= count
164         return xc, yc
165 
166 # returns the centroid of the aorta with an artery as if there was no artery
167 def fix_center(xc, yc, aorta):
168 
169         new_aorta = np.copy(aorta)
170         circle = helpers.sample_edge_points(xc, yc, new_aorta)
171         coords = reject_outliers(circle, new_aorta, 2.8)
172         xc, yc = calculate_cog(coords)
173 
174         coords = calculate_distances(xc, yc, new_aorta)
175         new_coords = reject_over_mode(coords, new_aorta)
176         xc, yc = calculate_cog(new_coords)
177 
178         coords = calculate_distances(xc, yc, new_aorta)
179         new_coords = reject_over_mode(coords, new_aorta)
180         xc, yc = calculate_cog(new_coords)
181 
182         '''
183         xse, yse = [], []
184         for i in xrange(len(new_aorta[:,0])):
185             for j in xrange(len(new_aorta[0,:])):
186                 if new_aorta[i,j] == 1 and helpers.on_edge(i,j,new_aorta):
187                     xse.append(j)
188                     yse.append(i)
189 
190         fc.fit_it(xse,yse)
191         '''
192         return xc, yc
193 
194 # returns the centroid of the 1's in the image
195 def find_cog(image, z, threshold): #, img
196         aorta = SimpleITK.GetArrayFromImage(image)
197         xc, yc = 0, 0
198         xs, ys = [], []
199         xse, yse = [], []
200 
201         count = (aorta == 1).sum()
202         if (not count):
203             return -1,-1,xs,ys
204 
205         for i in xrange(len(aorta[:,0])):
206             for j in xrange(len(aorta[0,:])):
207                 if aorta[i,j] == 1:
208                     xc += i
209                     yc += j
210                     xs.append(j)
211                     ys.append(i)
212                     if (helpers.on_edge(i,j,aorta)):
213                         xse.append(j)
214                         yse.append(i)
215 
216         xc /= count
217         yc /= count
218         #print ('my cog is: ' + str(yc) + ', ' + str(xc))
```

```
219
220     #fc.fit_it(xse,yse)
221
222     if (artery(xc, yc, aorta, threshold, z)):
223         artery_zs.append(z)
224         xc, yc = fix_center(xc, yc, aorta)
225         #print ('my new cog is: ' + str(yc) + ', ' + str(xc))
226
227     # the numpy array and the sitk_image have reversed coordinates
228     # so we return yc and xc below 'backwards' intentionally
229     return int(yc), int(xc), xs, ys
230
231 # write an sitk image to file
232 def write_image_to_file(image, filename):
233     fn = '/homes/ks815/thesis/' + exp + '/images/' + filename + '.dcm'
234     writer = SimpleITK.ImageFileWriter()
235     writer.SetFileName(fn)
236     writer.Execute(image)
237
238 # mouse click function to store coordinates
239 def onclick(event):
240     global ix, iy
241     global coords
242     ix, iy = event.xdata, event.ydata
243     coords = []
244     coords.append(ix)
245     coords.append(iy)
246     if len(coords) == 2:
247         plt.close('all')
248     return
249
250 # reads in the initial set of DICOM images from the user given the path to
251 # the folder and returns a 3D array, imgOriginal, of the images (not binary)
252 def readSeries(pathDicom):
253     print ('\nreading dicom series... ')
254     reader = SimpleITK.ImageSeriesReader()
255     filenamesDICOM = reader.GetGDCMSeriesFileNames(pathDicom)
256     reader.SetFileNames(filenamesDICOM)
257     imgOriginal = reader.Execute()
258     return imgOriginal
259
260 # calculates the maximum diameter by sampling four diameters
261 def max_diameter(xc, yc, aorta, m=2.8):
262     diameters = helpers.sample_diameters(xc, yc, aorta)
263     d = np.abs(diameters - np.median(diameters))
264     mdev = np.median(d)
265     s = d/mdev if mdev else 0.
266     good_dias = []
267     for i, num in enumerate(s):
268         if num < m:
269             good_dias.append(diameters[i]) # removing outliers
270     return max(good_dias)
271
272 # returns the percentage of overlap between aorta1 and aorta2
273 def overlap(aorta1, aorta2):
274     diff = np.add(aorta1,aorta2)
275     count1 = (diff == 1).sum()
276     count2 = (diff == 2).sum()
277
278     if (count1 == 0 and count2 > 1):
```

```
279         return 1.0
280     if (count2 == 0):
281         return 0.0
282 
283     return 1.0*count2/count1
284 
285 # displays an image for the user to click on, gets the coordinates, and shows the
        segmentation
286 def get_user_input(image):
287     sitk_get_click(image)
288     x,y = int(coords[0]), int(coords[1])
289     coord = [(x,y)]
290 
291     # Segment the aorta and fill in holes (the segmented image is binary)
292     smooth_image = smooth_img(image)
293     lower = image[x,y] - leeway
294     upper = image[x,y] + leeway
295     segmented_image = segment_img(smooth_image, coord, lower, upper)
296     segmented_image_no_holes = fill_holes(segmented_image)
297 
298     # Check to make sure it's correct before proceding
299     smooth_image_int = SimpleITK.Cast(SimpleITK.RescaleIntensity(smooth_image),
300                                       segmented_image.GetPixelID())
301     sitk_show(SimpleITK.LabelOverlay(smooth_image_int, segmented_image_no_holes))
302     return segmented_image_no_holes
303 
304 ''' ------------------------------------------------------------------------ '''
305 ''' ------------------------------- MAIN ----------------------------------- '''
306 ''' ------------------------------------------------------------------------ '''
307 current_time = datetime.datetime.now().time()
308 
309 # The following variables are what the user should be able to modify as needed
310 n = 4                   # 2 = patient1, 3 = patient2, 4 = patient3
311 leeway = 60                 # tolerance for segmentation
312 threshold = 2.845               # tolerance for artery selection
313 idxSlice = 300                  # starting image
314 paths = ["patient1/SDY00000/SRS00004",  # 420 images
315          "patient2/SDY00000/SRS00001",  # 766 images
316          "patient3/SRS00003"]           # 594 images
317 
318 pathDicom = paths[n-2]          # dicom directory
319 exp = 'exp' + str(n)            # experiment title
320 comments  = 'circle fitting test'
321 
322 print('     title:  ' + exp)
323 print('start time:  ' + str(current_time.isoformat()))
324 print('dicom path:  ' + str(pathDicom))
325 print('    leeway:  ' + str(leeway))
326 print('  idxSlice:  ' + str(idxSlice))
327 print(' threshold:  ' + str(threshold))
328 print('  comments:  ' + comments)
329 
330 # Read DICOM series
331 imgOriginal = readSeries(pathDicom)
332 
333 # Z slice of the DICOM files to process first.
334 image = imgOriginal[:,:,idxSlice]
335 
336 # Have user click on idxSlice and get coordinates of aorta
337 segmented_image_no_holes = get_user_input(image)
```

```
338
339 # Find the center point of the 2D binary array where 1's indicate the aorta
340 global artery_zs
341 artery_zs = []
342 x1, y1, _, _ = find_cog(segmented_image_no_holes, idxSlice, threshold) #, image
343 x2, y2 = x1, y1
344 idxBelow, idxAbove = idxSlice, idxSlice
345 (_, _, z) = imgOriginal.GetSize()
346 center_line_x, center_line_y, center_line_z = [], [], []
347 aorta_xs, aorta_ys, aorta_zs, diameters = [], [], [], []
348 image = imgOriginal[:,:,idxSlice]
349 lower = image[x1,y1] - leeway
350 upper = image[x1,y1] + leeway
351 prev_aorta1 = SimpleITK.GetArrayFromImage(segmented_image_no_holes)
352 prev_aorta2 = SimpleITK.GetArrayFromImage(segmented_image_no_holes)
353
354 # Use the prior center point as the segmentation point and propogate
355 i = 1
356 keep_going1, keep_going2 = True, True
357 print ('gathering geometries...\n')
358 print ('** = image contains an artery\n')
359
360 while (idxBelow > 0 or idxAbove < z):
361     idxBelow = idxSlice-i+1
362     if (idxBelow >= 0 and keep_going1):
363         image = imgOriginal[:,:,idxBelow]
364         coord = [(x1, y1)]
365         lower = image[x1,y1] - leeway
366         upper = image[x1,y1] + leeway
367         smooth_image = smooth_img(image)
368         segmented_image = segment_img(smooth_image, coord, lower, upper)
369         smooth_image_int = SimpleITK.Cast(SimpleITK.RescaleIntensity(smooth_image),
370                                           segmented_image.GetPixelID())
371         segmented_image_no_holes = fill_holes(segmented_image)
372         filename = 'img_' + str(idxBelow)
373         aorta = SimpleITK.GetArrayFromImage(segmented_image_no_holes)
374         x1, y1, xs1, ys1 = find_cog(segmented_image_no_holes, idxBelow, threshold)
375         if (x1 != -1 and overlap(prev_aorta2, aorta) > 0.2):
376             try:
377                 d1 = max_diameter(y1, x1, aorta) # (the y1, x1 is intentional)
378                 diameters.append((idxBelow, d1))
379                 center_line_x.append((idxBelow,x1))
380                 center_line_y.append((idxBelow,y1))
381                 center_line_z.append(idxBelow)
382                 aorta_xs = aorta_xs + xs1
383                 aorta_ys = aorta_ys + ys1
384                 for _ in xs1:
385                     aorta_zs.append(idxBelow)
386                 write_image_to_file(SimpleITK.LabelOverlay(smooth_image_int,
387                                                            segmented_image_no_holes),
    filename)
388                 prev_aorta1 = np.copy(aorta)
389             except:
390                 keep_going1 = False
391         else:
392             keep_going1 = False
393
394     idxAbove = idxSlice+i
395     if (idxAbove < z and keep_going2):
396         image = imgOriginal[:,:,idxAbove]
```

```
397         coord = [(x2, y2)]
398         lower = image[x2,y2] - leeway
399         upper = image[x2,y2] + leeway
400         smooth_image = smooth_img(image)
401         segmented_image = segment_img(smooth_image, coord, lower, upper)
402         smooth_image_int = SimpleITK.Cast(
403                                         SimpleITK.RescaleIntensity(smooth_image),
404                                         segmented_image.GetPixelID())
405         segmented_image_no_holes = fill_holes(segmented_image)
406         aorta = SimpleITK.GetArrayFromImage(segmented_image_no_holes)
407         if (x2 != -1 and overlap(prev_aorta2, aorta) > 0.2):
408             x2, y2, xs2, ys2 = find_cog(segmented_image_no_holes, idxAbove, threshold)
409             try:
410                 d2 = max_diameter(y2, x2, aorta) # (the y2, x2 is intentional)
411                 diameters.append((idxAbove, d2))
412                 center_line_x.append((idxAbove,x2))
413                 center_line_y.append((idxAbove,y2))
414                 center_line_z.append(idxAbove)
415                 aorta_xs = aorta_xs + xs2
416                 aorta_ys = aorta_ys + ys2
417                 for _ in xs2:
418                     aorta_zs.append(idxAbove)
419                 filename = 'img_' + str(idxAbove)
420                 write_image_to_file(SimpleITK.LabelOverlay(smooth_image_int,
421                                                            segmented_image_no_holes),
    filename)
422                 prev_aorta2 = np.copy(aorta)
423             except:
424                 keep_going2 = False
425         else:
426             keep_going2 = False
427 
428     i = i + 1
429 
430 # Save the data to files so that we can plot it using showme.py
431 print('\nsaving data in txt files...')
432 
433 the_filename = exp + '/xs.txt'
434 with open(the_filename, 'w') as f:
435     for s in aorta_xs:
436         f.write(str(s) + '\n')
437 
438 the_filename = exp + '/ys.txt'
439 with open(the_filename, 'w') as f:
440     for s in aorta_ys:
441         f.write(str(s) + '\n')
442 
443 the_filename = exp + '/zs.txt'
444 with open(the_filename, 'w') as f:
445     for s in aorta_zs:
446         f.write(str(s) + '\n')
447 
448 center_line_x.sort(key=lambda x: int(x[0]))
449 clx = map(operator.itemgetter(1), center_line_x)
450 the_filename = exp + '/clx.txt'
451 with open(the_filename, 'w') as f:
452     for s in clx:
453         f.write(str(s) + '\n')
454 
455 center_line_y.sort(key=lambda x: int(x[0]))
```

```
cly = map(operator.itemgetter(1), center_line_y)
the_filename = exp + '/cly.txt'
with open(the_filename, 'w') as f:
    for s in cly:
        f.write(str(s) + '\n')

center_line_z.sort()
the_filename = exp + '/clz.txt'
with open(the_filename, 'w') as f:
    for s in center_line_z:
        f.write(str(s) + '\n')

artery_zs.sort()
the_filename = exp + '/artery_imgs.txt'
with open(the_filename, 'w') as f:
    for s in artery_zs:
        f.write(str(s) + '\n')

diameters.sort(key=lambda x: float(x[0]))
diams = map(operator.itemgetter(1), diameters)
the_filename = exp + '/diameters.txt'
with open(the_filename, 'w') as f:
    for s in diams:
        f.write(str(s) + '\n')

calculate.run_calculations(imgOriginal, exp)

current_time = datetime.datetime.now().time()
print('end time: ' + str(current_time.isoformat()))
print('finished running thesis.py')
''' ———————————————————— END OF PROGRAM ———————————————————— '''
```

## B.2 helpers.py

```
''' helpers.py: helper functions for thesis.py. '''
import math, sys, SimpleITK
import numpy as np
import matplotlib.pyplot as plt

# show the image and change the center to a (color=white) cross
def sitk_show_pt(img, x, y, color=-1000):
    img[x,y] = color
    img[x+1,y] = color
    img[x-1,y] = color
    img[x,y+1] = color
    img[x,y-1] = color
    sitk_show(img)

# displays an sitk image
def sitk_show(img, title=None, margin=0.05, dpi=40):
    nda = SimpleITK.GetArrayFromImage(img)
    figsize = (1 + margin) * nda.shape[0]/dpi, (1 + margin) * nda.shape[1]/dpi
    extent = (0, nda.shape[1], nda.shape[0], 0)
    fig = plt.figure(figsize=figsize, dpi=dpi)
    ax = fig.add_axes([margin, margin, 1 - 2*margin, 1 - 2*margin])
    plt.set_cmap("gray")
    ax.imshow(nda,extent=extent,interpolation=None)
    if title:
        plt.title(title)
```

```
26
27      plt.show()
28
29  # returns true if (x,y) is on the edge of the aorta
30  def on_edge(x,y,aorta):
31      for i in range(-1,2):
32          for j in range(-1,2):
33              if (aorta[x+i,y+j] == 0):
34                  return True
35      return False
36
37  # print a numpy array (nda) to the terminal
38  # I reccomend using: print_nda_to_terminal(aorta[200:350, 200:350])
39  def print_nda_to_terminal(nda):
40      for i in xrange(len(nda[:,0])):
41          for j in xrange(len(nda[0,:])):
42              sys.stdout.write(str(nda[i,j]))
43              sys.stdout.flush()
44          print(' ')
45
46  # normalizes the color range of an image to be between new_min (white) and new_max (
        black)
47  def normalize_brightness(image, new_min=-1024, new_max=976):
48      aorta = SimpleITK.GetArrayFromImage(image)
49      max_color = np.amax(aorta)
50      min_color = np.amin(aorta)
51      x,y = image.GetSize()
52      for i in xrange(x):
53          for j in xrange(y):
54              image[i,j] = int((new_max - new_min) * (image[i,j] - min_color) /
55                               (max_color - min_color) + min_color)
56
57      return image
58
59  # get 16 points on the edge of the aorta in the form (x, y, r)
60  def sample_edge_points(xc, yc, aorta):
61      circle = []
62      i = 1
63      while (aorta[xc+i][yc] == 1):
64          i+=1
65      i-=1
66      circle.append((xc+i,yc,float(i)))
67      i = 1
68      while (aorta[xc-i][yc] == 1):
69          i+=1
70      i-=1
71      circle.append((xc-i,yc,float(i)))
72      i = 1
73      while (aorta[xc][yc+i] == 1):
74          i+=1
75      i-=1
76      circle.append((xc,yc+i,float(i)))
77      i = 1
78      while (aorta[xc][yc-i] == 1):
79          i+=1
80      i-=1
81      circle.append((xc,yc-i,float(i)))
82      i = 1
83      while (aorta[xc+i][yc+i] == 1):
84          i+=1
```

```
85      i-=1
86      circle.append((xc+i,yc+i,round(math.sqrt((i)**2+(i)**2),2)))
87      i = 1
88      while (aorta[xc-i][yc+i] == 1):
89          i+=1
90      i-=1
91      circle.append((xc-i,yc+i,round(math.sqrt((i)**2+(i)**2),2)))
92      i = 1
93      while (aorta[xc-i][yc-i] == 1):
94          i+=1
95      i-=1
96      circle.append((xc-i,yc-i,round(math.sqrt((i)**2+(i)**2),2)))
97      i = 1
98      while (aorta[xc+i][yc-i] == 1):
99          i+=1
100     i-=1
101     circle.append((xc+i,yc-i,round(math.sqrt((i)**2+(i)**2),2)))
102     i = 1
103     while (aorta[xc+i*2][yc+i] == 1):
104         i+=1
105     i -= 1
106     circle.append((xc+i*2,yc+i,round(math.sqrt((i*2)**2+(i)**2),2)))
107     i = 1
108     while (aorta[xc-i*2][yc+i] == 1):
109         i+=1
110     i-=1
111     circle.append((xc-i*2,yc+i,round(math.sqrt((i*2)**2+(i)**2),2)))
112     i = 1
113     while (aorta[xc-i*2][yc-i] == 1):
114         i+=1
115     i-=1
116     circle.append((xc-i*2,yc-i,round(math.sqrt((i*2)**2+(i)**2),2)))
117     i = 1
118     while (aorta[xc+i*2][yc-i] == 1):
119         i+=1
120     i-=1
121     circle.append((xc+i*2,yc-i,round(math.sqrt((i*2)**2+(i)**2),2)))
122     i = 1
123     while (aorta[xc+i][yc+i*2] == 1):
124         i+=1
125     circle.append((xc+i,yc+i*2,round(math.sqrt((i)**2+(i*2)**2),2)))
126     i = 1
127     while (aorta[xc-i][yc+i*2] == 1):
128         i+=1
129     circle.append((xc-i,yc+i*2,round(math.sqrt((i)**2+(i*2)**2),2)))
130     i = 1
131     while (aorta[xc-i][yc-i*2] == 1):
132         i+=1
133     circle.append((xc-i,yc-i*2,round(math.sqrt((i)**2+(i*2)**2),2)))
134     i = 1
135     while (aorta[xc+i][yc-i*2] == 1):
136         i+=1
137     circle.append((xc+i-1,yc-i*2,round(math.sqrt((i)**2+(i*2)**2),2)))
138
139     return circle
140
141 # Sample four diameters
142 def sample_diameters(xc, yc, aorta):
143     diameters = []
144     #temp_aorta = np.copy(aorta) # testing
```

```
145
146     i = 1
147     while (aorta[xc+i][yc] == 1):
148         i+=1
149     i-=1
150     diameters.append(float(i))
151     #temp_aorta[xc+i][yc] = 3
152     i = 1
153     while (aorta[xc-i][yc] == 1):
154         i+=1
155     i-=1
156     diameters[-1] += float(i)
157     #temp_aorta[xc-i][yc] = 3
158
159     i = 1
160     while (aorta[xc][yc+i] == 1):
161         i+=1
162     i-=1
163     diameters.append(float(i))
164     #temp_aorta[xc][yc+i] = 3
165     i = 1
166     while (aorta[xc][yc-i] == 1):
167         i+=1
168     i-=1
169     diameters[-1] += float(i)
170     #temp_aorta[xc][yc-i] = 3
171
172     i = 1
173     while (aorta[xc+i][yc+i] == 1):
174         i+=1
175     i-=1
176     diameters.append(round(math.sqrt((i)**2+(i)**2),2))
177     #temp_aorta[xc+i][yc+i] = 3
178     i = 1
179     while (aorta[xc-i][yc+i] == 1):
180         i+=1
181     i-=1
182     diameters[-1] += round(math.sqrt((i)**2+(i)**2),2)
183     #temp_aorta[xc-i][yc+i] = 3
184
185     i = 1
186     while (aorta[xc-i][yc-i] == 1):
187         i+=1
188     i-=1
189     diameters.append(round(math.sqrt((i)**2+(i)**2),2))
190     #temp_aorta[xc-i][yc-i] = 3
191     i = 1
192     while (aorta[xc+i][yc-i] == 1):
193         i+=1
194     i-=1
195     diameters[-1] += round(math.sqrt((i)**2+(i)**2),2)
196     #temp_aorta[xc+i][yc-i] = 3
197
198     i = 1
199     while (aorta[xc+i*2][yc+i] == 1):
200         i+=1
201     i -= 1
202     diameters.append(round(math.sqrt((i*2)**2+(i)**2),2))
203     #temp_aorta[xc+i*2][yc+i] = 3
204     i = 1
```

```
205     while (aorta[xc-i*2][yc+i] == 1):
206         i+=1
207     i-=1
208     diameters[-1] += round(math.sqrt((i*2)**2+(i)**2),2)
209     #temp_aorta[xc+i*2][yc+i] = 3
210 
211     i = 1
212     while (aorta[xc-i*2][yc-i] == 1):
213         i+=1
214     i-=1
215     diameters.append(round(math.sqrt((i*2)**2+(i)**2),2))
216     #temp_aorta[xc-i*2][yc-i] = 3
217     i = 1
218     while (aorta[xc+i*2][yc-i] == 1):
219         i+=1
220     i-=1
221     diameters[-1] += round(math.sqrt((i*2)**2+(i)**2),2)
222     #temp_aorta[xc+i*2][yc-i] = 3
223 
224 
225     i = 1
226     while (aorta[xc+i][yc+i*2] == 1):
227         i+=1
228     diameters.append(round(math.sqrt((i)**2+(i*2)**2),2))
229     #temp_aorta[xc+i][yc+i*2] = 3
230     i = 1
231     while (aorta[xc-i][yc+i*2] == 1):
232         i+=1
233     diameters[-1] += round(math.sqrt((i)**2+(i*2)**2),2)
234     #temp_aorta[xc-i][yc+i*2] = 3
235 
236 
237     i = 1
238     while (aorta[xc-i][yc-i*2] == 1):
239         i+=1
240     diameters.append(round(math.sqrt((i)**2+(i*2)**2),2))
241     #temp_aorta[xc-i][yc-i*2] = 3
242     i = 1
243     while (aorta[xc+i][yc-i*2] == 1):
244         i+=1
245     diameters[-1] += round(math.sqrt((i)**2+(i*2)**2),2)
246     #temp_aorta[xc+i][yc-i*2] = 3
247 
248     #print_nda_to_terminal(temp_aorta[200:350, 200:350])
249     #sys.exit()
250     return diameters
```

## B.3 showme.py

```
1  """
2  Plots xs, ys, and zs and the center line in a 3D scatterplot.
3  Must be run in a folder with xs.txt, ys.txt, zs.txt, clx.txt, cly.txt, and clz.txt
4  
5  """
6  
7  from pyqtgraph.Qt import QtCore, QtGui
8  import pyqtgraph.opengl as gl
9  import numpy as np
10 
```

```
11 # get data from a file where the data are floats
12 def get_data_float(filename):
13     s = []
14     with open(filename) as f:
15         for line in f:
16             data = line.split()
17             s.append(float(data[0]))
18     return s
19
20 # get data from a file where the data are ints
21 def get_data_int(filename):
22     s = []
23     with open(filename) as f:
24         for line in f:
25             data = line.split()
26             s.append(int(data[0]))
27     return s
28
29 ############## MAIN ##############
30 app = QtGui.QApplication([])
31 w = gl.GLViewWidget()
32 w.opts['distance'] = 20
33 w.show()
34 w.setWindowTitle('Aorta Visualization')
35
36 g = gl.GLGridItem()
37 w.addItem(g)
38
39 pos, size, color = [], [], []
40
41 xs = get_data_int('xs.txt')
42 ys = get_data_int('ys.txt')
43 zs = get_data_int('zs.txt')
44 clx = get_data_int('clx.txt')
45 cly = get_data_int('cly.txt')
46 clz = get_data_int('clz.txt')
47
48 for i,item in enumerate(xs):
49     pos.append((xs[i]-xs[0], ys[i]-ys[0], zs[i]-zs[0]))
50     size.append(0.8)                    # aorta data points size
51     color.append((0.0, 1.0, 0.0, .3))   # (color, color, color, opacity) green
52
53 for i,item in enumerate(clx):
54     pos.append((clx[i]-xs[0], cly[i]-ys[0], clz[i]-zs[0]))
55     size.append(0.8)                    # center line points size
56     color.append((1.0, 0.0, 0.0, 1))
57
58 pos = np.array(pos)
59 size = np.array(size)
60 color = np.array(color)
61
62 sp1 = gl.GLScatterPlotItem(pos=pos, size=size, color=color, pxMode=False)
63 w.addItem(sp1)
64
65 sp2 = gl.GLScatterPlotItem(pos=pos, size=size, color=color, pxMode=False)
66 w.addItem(sp2)
67
68 ## Start Qt event loop unless running in interactive mode.
69 if __name__ == '__main__':
70     import sys
```

```
71      if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
72          QtGui.QApplication.instance().exec_()
```

## B.4 circle.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy import stats
4  import math, sys
5
6  # calculate the euclidean distance from (x1, y1) to (x2, y2)
7  def dist(x1, y1, x2, y2):
8      return int(math.sqrt( (x2 - x1)**2 + (y2 - y1)**2 ))
9
10 # plot a histogram for a set of distances (ds)
11 def plot_hist(ds):
12     plt.hist(ds, bins=[0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,51,54,57,60])
13     plt.xticks([0,6,12,18,24,30,36,42,48,54,60])
14     plt.ylabel('Count')
15     plt.xlabel('Distance (pixels)')
16     plt.axis([0,60,0,600])
17     plt.title('Distances Histogram (r=30)')
18     plt.show()
19     m = stats.mode(ds)
20     print ('\nmode(r=' + str(r) + '):\t' + str(int(m[0])))
21
22 # plot ratios as a function of radii and plot a theshold line at y=2.845
23 def plot_rads(rads, ratios, x, y):
24     plt.plot(rads, ratios)
25     plt.axhline(y=2.845, color='r')
26     plt.axis([0, x, 2, y])
27     plt.ylabel('ratio', fontsize=18)
28     plt.xlabel('circle radius', fontsize=18)
29     plt.title('ratio as a function of the radius of the circle', fontsize=18)
30     plt.show()
31
32 # calculate the ratio for a given shape (r = radius; ellipse variables: a & b)
33 def calc_ratio(r, a=1, b=1):
34     n = 3000
35     xc = n/2
36     yc = n/2
37     ds = []
38     arr = np.zeros((n,n))
39
40     for i in xrange(n):
41         for j in xrange(n):
42             if (((xc-i)**2)/a+((yc-j)**2)/b) <= r**2: # equation
43                 arr[i][j] = dist(i,j,xc,yc)
44                 ds.append(dist(i,j,xc,yc))
45
46     ds.sort()
47
48     lower_dists = ds[:len(ds)/3]
49     upper_dists = ds[len(ds)*2/3:]
50     lower = float(sum(lower_dists))/float(len(lower_dists))
51     upper = float(sum(upper_dists))/float(len(upper_dists))
52     return upper/lower
53
54 # MAIN SCRIPT #
```

```
''' CIRCLE '''
rads = [2,3,4,5,6,7,8,9,10,15,20,30,40,50,100,200,300,1500]
ratios = []
i = 1
for rad in rads:
    ratios.append(round(calc_ratio(rad),4))
    prog = (i*1.0)/(len(rads)*1.0)*100
    sys.stdout.write("\r%d%%" % prog)
    sys.stdout.flush()
    i+=1
print('finished circle')

''' ELLIPSE '''
a_s = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
ratios2 = []
i = 1
for a in a_s:
    ratios2.append(round(calc_ratio(1, a, 1),4))
    prog = (i*1.0)/(len(a_s)*1.0)*100
    sys.stdout.write("\r%d%%" % prog)
    sys.stdout.flush()
    i+=1

plot_rads(a_s, ratios2, max(a_s), 5)
plot_rads(rads, ratios, 50, 3.5)

sys.exit()
```

## B.5 fit_circle.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# http://yotamgingold.com/code/fitellipse.py
"""
http://www.scipy.org/Cookbook/Least_Squares_Circle
"""

from numpy import *

# Coordinates of the 2D points
def fit_it(xs,ys):
    x = asarray(xs)
    y = asarray(ys)
    basename = 'circle'

    # == METHOD 1 ==
    method_1 = 'algebraic'

    # coordinates of the barycenter
    x_m = mean(x)
    y_m = mean(y)

    # calculation of the reduced coordinates
    u = x - x_m
    v = y - y_m

    # linear system defining the center in reduced coordinates (uc, vc):
    #    Suu * uc +  Suv * vc = (Suuu + Suvv)/2
    #    Suv * uc +  Svv * vc = (Suuv + Svvv)/2
```

```
30      Suv  = sum(u*v)
31      Suu  = sum(u**2)
32      Svv  = sum(v**2)
33      Suuv = sum(u**2 * v)
34      Suvv = sum(u * v**2)
35      Suuu = sum(u**3)
36      Svvv = sum(v**3)
37
38      # Solving the linear system
39      A = array([ [ Suu, Suv ], [Suv, Svv]])
40      B = array([ Suuu + Suvv, Svvv + Suuv ])/2.0
41      uc, vc = linalg.solve(A, B)
42
43      xc_1 = x_m + uc
44      yc_1 = y_m + vc
45
46      # Calculation of all distances from the center (xc_1, yc_1)
47      Ri_1      = sqrt((x-xc_1)**2 + (y-yc_1)**2)
48      R_1       = mean(Ri_1)
49      residu_1  = sum((Ri_1-R_1)**2)
50      residu2_1 = sum((Ri_1**2-R_1**2)**2)
51
52      # Decorator to count functions calls
53      import functools
54      def countcalls(fn):
55          "decorator function count function calls "
56
57          @functools.wraps(fn)
58          def wrapped(*args):
59              wrapped.ncalls +=1
60              return fn(*args)
61
62          wrapped.ncalls = 0
63          return wrapped
64
65      #  == METHOD 2 ==
66      # Basic usage of optimize.leastsq
67      from scipy      import optimize
68
69      method_2  = "leastsq"
70
71      def calc_R(xc, yc):
72          """ calculate the distance of each 2D points from the center (xc, yc) """
73          return sqrt((x-xc)**2 + (y-yc)**2)
74
75      @countcalls
76      def f_2(c):
77          """ calculate the algebraic distance between the 2D points and the mean circle
78          centered at c=(xc, yc) """
79          Ri = calc_R(*c)
80          return Ri - Ri.mean()
81
82      center_estimate = x_m, y_m
83      center_2, ier = optimize.leastsq(f_2, center_estimate)
84
85      xc_2, yc_2 = center_2
86      Ri_2       = calc_R(xc_2, yc_2)
87      R_2        = Ri_2.mean()
88      residu_2   = sum((Ri_2 - R_2)**2)
89      residu2_2  = sum((Ri_2**2-R_2**2)**2)
```

```
90      ncalls_2    = f_2.ncalls
91
92      # == METHOD 2b ==
93      # Advanced usage, with jacobian
94      method_2b   = "leastsq with jacobian"
95
96      def calc_R(xc, yc):
97          """ calculate the distance of each 2D points from the center c=(xc, yc) """
98          return sqrt((x-xc)**2 + (y-yc)**2)
99
100     @countcalls
101     def f_2b(c):
102         """ calculate the algebraic distance between the 2D points and the mean circle
103         centered at c=(xc, yc) """
104         Ri = calc_R(*c)
105         return Ri - Ri.mean()
106
107     @countcalls
108     def Df_2b(c):
109         """ Jacobian of f_2b
110         The axis corresponding to derivatives must be coherent with the col_deriv
        option
111         of leastsq"""
112         xc, yc     = c
113         df2b_dc    = empty((len(c), x.size))
114
115         Ri = calc_R(xc, yc)
116         df2b_dc[ 0] = (xc - x)/Ri                   # dR/dxc
117         df2b_dc[ 1] = (yc - y)/Ri                   # dR/dyc
118         df2b_dc       = df2b_dc - df2b_dc.mean(axis=1)[:, newaxis]
119
120         return df2b_dc
121
122     center_estimate = x_m, y_m
123     center_2b, ier = optimize.leastsq(f_2b, center_estimate, Dfun=Df_2b, col_deriv=
        True)
124
125     xc_2b, yc_2b = center_2b
126     Ri_2b        = calc_R(xc_2b, yc_2b)
127     R_2b         = Ri_2b.mean()
128     residu_2b    = sum((Ri_2b - R_2b)**2)
129     residu2_2b   = sum((Ri_2b**2-R_2b**2)**2)
130     ncalls_2b    = f_2b.ncalls
131
132     print "Method 2b :"
133     print "Functions calls : f_2b=%d Df_2b=%d" % ( f_2b.ncalls, Df_2b.ncalls)
134
135     # == METHOD 3 ==
136     # Basic usage of odr with an implicit function definition
137     from scipy      import  odr
138
139     method_3  = "odr"
140
141     @countcalls
142     def f_3(beta, x):
143         """ implicit definition of the circle """
144         return (x[0]-beta[0])**2 + (x[1]-beta[1])**2 -beta[2]**2
145
146     # initial guess for parameters
147     R_m = calc_R(x_m, y_m).mean()
```

```
148     beta0 = [ x_m, y_m, R_m]
149
150     # for implicit function :
151     #       data.x contains both coordinates of the points
152     #       data.y is the dimensionality of the response
153     lsc_data    = odr.Data(row_stack([x, y]), y=1)
154     lsc_model   = odr.Model(f_3, implicit=True)
155     lsc_odr     = odr.ODR(lsc_data, lsc_model, beta0)
156     lsc_out     = lsc_odr.run()
157
158     xc_3, yc_3, R_3 = lsc_out.beta
159     Ri_3        = calc_R(xc_3, yc_3)
160     residu_3    = sum((Ri_3 - R_3)**2)
161     residu2_3   = sum((Ri_3**2-R_3**2)**2)
162     ncalls_3    = f_3.ncalls
163
164     # == METHOD 3b ==
165     # Advanced usage, with jacobian
166     method_3b   = "odr with jacobian"
167     print "\nMethod 3b : ", method_3b
168
169     @countcalls
170     def f_3b(beta, x):
171         """ implicit definition of the circle """
172         return (x[0]-beta[0])**2 + (x[1]-beta[1])**2 -beta[2]**2
173
174     @countcalls
175     def jacb(beta, x):
176         """ Jacobian function with respect to the parameters beta.
177         return df_3b/dbeta
178         """
179         xc, yc, r = beta
180         xi, yi    = x
181
182         df_db    = empty((beta.size, x.shape[1]))
183         df_db[0] =  2*(xc-xi)                     # d_f/dxc
184         df_db[1] =  2*(yc-yi)                     # d_f/dyc
185         df_db[2] = -2*r                           # d_f/dr
186
187         return df_db
188
189     @countcalls
190     def jacd(beta, x):
191         """ Jacobian function with respect to the input x.
192         return df_3b/dx
193         """
194         xc, yc, r = beta
195         xi, yi    = x
196
197         df_dx    = empty_like(x)
198         df_dx[0] =  2*(xi-xc)                     # d_f/dxi
199         df_dx[1] =  2*(yi-yc)                     # d_f/dyi
200
201         return df_dx
202
203
204     def calc_estimate(data):
205         """ Return a first estimation on the parameter from the data  """
206         xc0, yc0 = data.x.mean(axis=1)
207         r0 = sqrt((data.x[0]-xc0)**2 +(data.x[1] -yc0)**2).mean()
```

```
208         return xc0, yc0, r0
209
210     # for implicit function :
211     #       data.x contains both coordinates of the points
212     #       data.y is the dimensionality of the response
213     lsc_data  = odr.Data(row_stack([x, y]), y=1)
214     lsc_model = odr.Model(f_3b, implicit=True, estimate=calc_estimate, fjacd=jacd,
        fjacb=jacb)
215     lsc_odr   = odr.ODR(lsc_data, lsc_model)     # beta0 has been replaced by an
        estimate function
216     lsc_odr.set_job(deriv=3)                     # use user derivatives function
        without checking
217     lsc_odr.set_iprint(iter=1, iter_step=1)      # print details for each iteration
218     lsc_out   = lsc_odr.run()
219
220     xc_3b, yc_3b, R_3b = lsc_out.beta
221     Ri_3b       = calc_R(xc_3b, yc_3b)
222     residu_3b   = sum((Ri_3b - R_3b)**2)
223     residu2_3b  = sum((Ri_3b**2-R_3b**2)**2)
224     ncalls_3b   = f_3b.ncalls
225
226     print "\nFunctions calls : f_3b=%d jacb=%d jacd=%d" % (f_3b.ncalls, jacb.ncalls,
        jacd.ncalls)
227
228
229     # Summary
230     fmt = '%-22s %10.5f %10.5f %10.5f %10d %10.6f %10.6f %10.2f'
231     print ('\n%-22s' +' %10s'*7) % tuple('METHOD Xc Yc Rc nb_calls std(Ri) residu
        residu2'.split())
232     print '-'*(22 +7*(10+1))
233     print  fmt % (method_1 , xc_1 , yc_1 , R_1 ,        1 , Ri_1.std() , residu_1 ,
        residu2_1 )
234     print  fmt % (method_2 , xc_2 , yc_2 , R_2 , ncalls_2 , Ri_2.std() , residu_2 ,
        residu2_2 )
235     print  fmt % (method_2b, xc_2b, yc_2b, R_2b, ncalls_2b, Ri_2b.std(), residu_2b,
        residu2_2b)
236     print  fmt % (method_3 , xc_3 , yc_3 , R_3 , ncalls_3 , Ri_3.std() , residu_3 ,
        residu2_3 )
237     print  fmt % (method_3b, xc_3b, yc_3b, R_3b, ncalls_3b, Ri_3b.std(), residu_3b,
        residu2_3b)
238
239     # plotting functions
240     from matplotlib                 import pyplot as p, cm, colors
241     p.close('all')
242
243     def plot_all(residu2=False):
244         """ Draw data points, best fit circles and center for the three methods,
245         and adds the iso contours corresponding to the fiel residu or residu2
246         """
247
248         f = p.figure( facecolor='white')  #figsize=(7, 5.4), dpi=72,
249         p.axis('equal')
250
251         theta_fit = linspace(-pi, pi, 180)
252
253         x_fit1 = xc_1 + R_1*cos(theta_fit)
254         y_fit1 = yc_1 + R_1*sin(theta_fit)
255         p.plot(x_fit1, y_fit1, 'b-' , label=method_1, lw=2)
256
257         x_fit2 = xc_2 + R_2*cos(theta_fit)
```

```
258         y_fit2  = yc_2 + R_2*sin(theta_fit)
259         p.plot(x_fit2, y_fit2, 'k--', label=method_2, lw=2)
260 
261         x_fit3  = xc_3 + R_3*cos(theta_fit)
262         y_fit3  = yc_3 + R_3*sin(theta_fit)
263         p.plot(x_fit3, y_fit3, 'r-.', label=method_3, lw=2)
264 
265         p.plot([xc_1], [yc_1], 'bD', mec='y', mew=1)
266         p.plot([xc_2], [yc_2], 'gD', mec='r', mew=1)
267         p.plot([xc_3], [yc_3], 'kD', mec='w', mew=1)
268 
269         # draw
270         p.xlabel('x')
271         p.ylabel('y')
272 
273         # plot the residu fields
274         nb_pts = 100
275 
276         p.draw()
277         xmin, xmax = p.xlim([200,400])
278         ymin, ymax = p.ylim([100,300])
279 
280         vmin = min(xmin, ymin)
281         vmax = max(xmax, ymax)
282         '''
283         xg, yg = ogrid[vmin:vmax:nb_pts*1j, vmin:vmax:nb_pts*1j]
284         xg = xg[..., newaxis]
285         yg = yg[..., newaxis]
286 
287         Rig    = sqrt( (xg - x)**2 + (yg - y)**2 )
288         Rig_m  = Rig.mean(axis=2)[..., newaxis]
289 
290         if residu2 : residu = sum( (Rig**2 - Rig_m**2)**2 ,axis=2)
291         else       : residu = sum( (Rig-Rig_m)**2 ,axis=2)
292 
293         lvl = exp(linspace(log(residu.min()), log(residu.max()), 15))
294 
295         p.contourf(xg.flat, yg.flat, residu.T, lvl, alpha=0.4, cmap=cm.Purples_r) # ,
    norm=colors.LogNorm())
296         cbar = p.colorbar(fraction=0.175, format='%.f')
297         p.contour (xg.flat, yg.flat, residu.T, lvl, alpha=0.8, colors="lightblue")
298 
299         if residu2 : cbar.set_label('Residu_2 - algebraic approximation')
300         else       : cbar.set_label('Residu')
301         '''
302         # plot data
303         p.plot(x, y, 'ro', label='data', ms=8, mec='b', mew=1)
304         p.legend(loc='best',labelspacing=0.1 )
305 
306         p.xlim(xmin=vmin, xmax=vmax)
307         p.ylim(ymin=vmin, ymax=vmax)
308 
309         p.grid()
310         p.title('Least Squares Circle')
311         #p.savefig('%s_residu%d.png' % (basename, 2 if residu2 else 1))
312 
313     plot_all(residu2=False)
314     plot_all(residu2=True )
315 
316     p.show()
```

# Bibliography


[1] NHS, *Abdominal aortic aneurysm*. [Online]. Available: http://www.nhs.uk/conditions/repairofabdominalaneurysm/pages/introduction.aspx.

[2] Mayo Clinic Staff, *Treatment abdominal aortic aneurysm*. [Online]. Available: http://www.mayoclinic.org/diseases-conditions/abdominal-aortic-aneurysm/diagnosis-treatment/treatment/txc-20197872.

[3] G. Maleux, M. Koolen, and S. Heye, "Complications after endovascular aneurysm repair", *Seminars in Interventional Radiology*, pp. 3–9, 2009.

[4] A. H. Malwaki, T. A. Resch, M. J. Brown, B. J. Manning, J. D. Poloniecki, I. M. Nordon, and R. J. Hinchliffe, "Sizing fenestrated aortic stent - grafts", *European Journal of Vascular and Endovascular Surgery*, pp. 311–316, 2011.

[5] NHS, *Endovascular stent - grafts for the treatment of abdominal aortic aneurysms*, 2009. [Online]. Available: https://www.nice.org.uk/guidance/ta167/resources.

[6] U.S. Food and Drug Administration, *Zenith® fenestrated aaa endovascular graft (with the adjunctive zenith alignment stent) - p020018/s040*, 2012. [Online]. Available: http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/DeviceApprovalsandClearances/Recently-ApprovedDevices/ucm300704.htm.

[7] J. R. Scurr and McWilliams R. G., "Fenestrated aortic stent grafts", *Eminars in Interventional Radiology*, pp. 211–220, 2007.

[8] J. Black, *Fenestrated endovascular aortic aneurysm repair*, 2012. [Online]. Available: https://www.youtube.com/watch?v=A2vbFhYGwd0.

[9] A. C. Picel and N. Kansal, "Essentials of endovascular abdominal aortic aneurysm repair imaging: preprocedural assessment", *American Journal of Roentgenology*, pp. 347–357, 2014.

[10] Tera Recon, *Evar planning*. [Online]. Available: http://www.terarecon.com/advanced-visualization/evar-planning-package.

[11] P. Stenner, *Faster abdominal aortic stent planning with syngo.via and the ct cardio-vascular engine*, 2011. [Online]. Available: https://health.siemens.com/ct%7B%5C_%7Dapplications/somatomsessions/index.php/faster-abdominal-aortic-stent-planning-with-syngo-via-and-the-ct-cardio-vascular-engine/.

[12] Pie Medical Imaging., *3mensio vascular*. [Online]. Available: http://www.piemedicalimaging.com/product/3mensio-vascular/.

[13] Therenva SAS, *Reliable sizing. made easy*, 2014. [Online]. Available: https://www.endosize.com/.

[14] DICOM, *About dicom*. [Online]. Available: http://dicom.nema.org/Dicom/about-DICOM.html.

[15] VTK, *Overview vtk*. [Online]. Available: http://www.vtk.org/overview/.

[16] National Library of Medicine, *Itk*. [Online]. Available: https://itk.org/.

[17] The Intelligent Medical Research Center, *Mitk*, 2013. [Online]. Available: ttp://www.mitk.net/.

[18] I. M. Oliver, *The image based vascular anaylsis toolkit*, 2014. [Online]. Available: https://github.com/imacia/ivantk.

[19] G. Wollny, "Mia, a free and open source software for gray scale medical image analysis", *BioMed Central*, pp. 8–20, 2013. [Online]. Available: http://mia.sourceforge.net/.

[20] A. Kyriakou, *Image segmentation with python and simpleitk*, 2014. [Online]. Available: https://pyscience.wordpress.com/2014/10/19/image-segmentation-with-python-and-simpleitk/.

[21] D. Eppstein, *Breadth first search and depth first search*, 1996. [Online]. Available: https://www.ics.uci.edu/%7B%5C~%7B%7D%7Deppstein/161/960215.html.

[22] N. Mukai, Y. Tatefuku, Y. Chang, N. Kiyomi, and T. Shuichiro, "Automatic extraction of the aorta and the measurement of its diameter", *British Journal of Medicine & Medical Research*, pp. 671–682, 2014.

[23] SciPy Cookbook, *Least squares circle*, 2015. [Online]. Available: http://scipy-cookbook.readthedocs.io/items/Least%7B%5C_%7DSquares%7B%5C_%7DCircle.html.

[24] *Aeskulap - dicom viewer*, 2007. [Online]. Available: http://aeskulap.nongnu.org/.

[25] M. de Bruijne, B. van Ginneken, W. J. Niessen, J. B. A. Maintz, and M. A. Viergever, "Active-shape-model-based segmentation of abdominal aortic aneurysms in cta images", *SPIE Proceedings*, vol. 4684, 2002. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=879759.