



Bernhard Frohner, BSc.

Aortic Distensibility Estimation by M-Mode Echocardiographic Data

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn., Christian Baumgartner

Institut für Health Care Engineering

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____
Date Signature

Eidesstattliche Erklärung¹

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____
Datum Unterschrift

¹Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Acknowledgment

I would like to show my gratitude to my supervisor Prof. Christian Baumgartner for his comprehensive, goal-driven and professional supervision and the extensive support, far beyond the thesis over the last year. In addition, I would like to thank Prof. Daniela Baumgartner and her colleagues to be on hand with help and advice for me but also to make the testing phase at LKH-Univ. Klinikum Graz even possible. I personally think that a supervisor showing interest on a master's thesis for seven days a week is certainly not a matter of course.

In addition, I would especially like to thank my mother for her faith in me from the beginning of my studies to this day.

Danksagung

Ich möchte mich bei meinem Betreuer Prof. Christian Baumgartner für die überaus umfassende, zielorientierte und professionelle Betreuung und auch die weit darüber hinausgehende Unterstützung im letzten Jahr bedanken. Weiters gilt mein Dank auch Prof. Daniela Baumgartner und ihren KollegInnen, die mit wissenschaftlichen Rat und Tat zur Seite standen und die Testphase am LKH-Univ. Klinikum Graz überhaupt erst ermöglicht haben. Ich persönlich sehe einen Betreuer/eine Betreuerin, der/die sieben Tage pro Woche Interesse an einer Diplomarbeit zeigt, als alles andere als selbstverständlich an.

Weiters möchte ich mich besonders bei meiner Mutter für das große Vertrauen in mich von Anbeginn meines Studiums bis zum heutigen Tage bedanken.

Abstract

Bestimmung der Aortendehnbarkeit aus M-Mode Echokardiographie

Daten

Da die Ausdehnung und auch die beeinträchtigte Bioelastizität der Aorta für vielerlei kardiale und nicht-kardiale Erkrankungen von Interesse ist, sollten insbesondere KardiologInnen die Möglichkeit haben mechano-elastische Gefäß-Parameter wie beispielsweise die Dehnbarkeit bzw. den Steifigkeits-Index in-vivo und nicht-invasiv erheben zu können. Ziel dieser Arbeit war es, dies mittels einer zu implementierenden medizinischen Bildverarbeitungssoftware basierend auf M-Mode Echokardiographie-Aufnahmen zu ermöglichen. Die Anwendung erfordert dabei die Spezifikation eines M-Mode Ultraschallbildes, das die zeitlich aufgezeichnete Gefäßkontur der auf- oder absteigenden Aorta zeigt (1), die parallel dazu aufgezeichnete EKG-Kurve (2) und oszillometrisch erhobene Blutdruckwerte (3).

Während der Implementierung dieses Programms, war einerseits auf die Anforderungen an die Software als Medizinprodukt gemäß der aktuell gültigen Medizinprodukteverordnung MDR 2017/745 zu achten, andererseits sollte auch der Softwarelebenszyklus nach EN 62304:2006+A1:2015 beachtet werden.

Schlüsselwörter: Aorta, Dehnbarkeit, Ultraschall, Software, Medizinprodukt

Aortic Distensibility Estimation by M-Mode Echocardiographic Data

As the aortic enlargement and impaired bioelasticity are of interest in several cardiac and non-cardiac diseases, especially cardiologists should have the possibility to gain elastic parameters such as distensibility and stiffness-index in-vivo and non-invasively. The goal of this thesis was to facilitate this by implementing a medical image processing software that establishes this by analysis of M-mode echocardiographic images. This application requires the specification of an M-mode ultrasound image, showing the time-based vessel's edges of the ascending- or descending aorta (1), the ECG tracing recorded in parallel (2) as well as oscillometric blood pressure values (3).

During the implementation of this program, requirements on software as a medical

product according to the Medical Device Regulation 2017/745 (MDR 2017/745) had to be taken into account, as well as aspects of the software lifecycle under directive EN 62304:2006:A1:2015.

Keywords: Aorta, Distensibility, Ultrasound, Software, Medical Device

Contents

1	Introduction	1
1.1	Aorta	1
1.1.1	Cardiac Cycle	1
1.1.2	Biomechanical Behaviour	2
1.1.3	Windkessel Model	3
1.2	Echocardiography	4
1.2.1	M-Mode	4
1.2.2	Axial Resolution	5
1.2.3	Temporal Resolution	5
1.3	Image Processing Algorithms	6
1.3.1	Image Filter Kernel	6
1.3.2	Morphological Operators	7
1.3.3	Canny Edge Detection	8
1.3.4	Active Contour Model	9
1.3.5	Morphological GAC	9
1.3.6	Hough Line Transform	12
1.4	Regulatory Aspects of Software as a Medical Product	12
1.4.1	EU Risk Classification	13
1.4.2	Software Lifecycle	14
1.4.3	Demand for Usability	15
2	Scope of Work	17
2.1	Overall Goal	17
2.2	Preliminary Work	17
2.3	Minimum Requirements	17
3	Methods	19
3.1	Calculated Aortic Parameters	19
3.2	Software Requirement Specification	21
3.3	Programming Environment	21
3.4	Software Architecture	22
3.4.1	GUI class	22

3.4.2	UsImage class	24
3.4.3	Backend Classes	25
3.5	Edge Detection Process	26
3.5.1	Load Image	26
3.5.2	Detect Scales	27
3.5.3	Detect Electrocardiogram (ECG)	28
3.5.4	Detect Aorta	29
3.5.5	Detect Edges	30
3.5.6	Calc Parameters	36
3.6	Software Packaging	37
3.7	Software Documentation	38
3.8	Code Reviews	38
3.9	Usability Testing	39
4	Results	41
4.1	AortUs Usage	41
4.1.1	Full Usage Example	41
4.1.2	Manual Scale Detection Usage Example	52
4.1.3	Clinical Patient Reports	55
4.2	Regulatory Aspects of AortUs	56
4.2.1	Traceability and Identification of Software of Unknown Pedigree (SOUP)s	56
4.2.2	Risk Classification	56
4.2.3	System Usability Scale	56
5	Discussion	59
6	Conclusion	63
	References	65
	Appendix	69
	Python Packages and Versions	69
	Software Requirement Specification	71
	Software Usability Scale Form	109

AortUs Report of 21Y Female Marfan Syndrome Patient	111
AortUs Report of 4Y Female Healthy Patient	115
AortUs Report of 48Y Male Healthy Patient	119
User Manual	123
Software Documentation	161

Acronyms

Notation	Description
adventitia	tunica adventitia
AscAo	ascending aorta
BF	Bernhard Frohner, BSc.
BP	blood pressure
CB	Univ.-Prof. Dipl.-Ing. Dr. techn. Christian Baumgartner
CLAHE	contrast limited adaptive histogram equalization
COV	coefficient of variation
CVD	cardiovascular disease
DBP	diastolic blood pressure
DesAo	descending aorta
ECG	Electrocardiogram
GAC	geodesic active contour
GUI	Graphical User Interface
HSV	hue saturation value
IDE	integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
intima	tunica intima
LoS	line of sight
MDA	Medical Device Act
MDD	Medical Device Directive

Notation	Description
MDR 2017/745	Medical Device Regulation 2017/745
media	tunica media
MRI	magnetic resonance imaging
OS	operating system
PDE	partial differential equation
PRP	pulse repetition period
PWV	pulse wave velocity
reST	restructured text
ROI	region of interest
SBP	systolic blood pressure
SOUP	Software of Unknown Pedigree
SPL	spatial pulse length
SRS	Software Requirement Specification
SUS	System Usability Scale
TPR	total peripheral resistance
TTE	transthoracic echocardiography
US	ultrasound

1 Introduction

Despite emerging medical background understanding of etiological factors and improving surgical techniques, cardiovascular disease (CVD) remains the leading cause of death worldwide [1]. One approach of early diagnosis is to study the physiological and pathophysiological characteristics of the aorta during in-vivo diagnostic echocardiography examinations. To illustrate the importance of its mechanical properties, it is essential to give a theoretical overview of the aortic anatomy and physiology with respect to its appearance in transthoracic echocardiography (TTE). Another crucial aspect gaining a sensitive diagnostic marker of impaired arterial bioelasticity is the principle technical and medical background of TTE and how this is related to image processing topics to extract time-based aortic wall distortion curves.

As the overall idea behind this software named *AortUs* is a later commercial use, also regulatory medical device aspects will be outlined in advance.

1.1 Aorta

The aorta is the main artery in the human systemic circulation that supplies oxygenated blood to all parts of the body. It has its origin in the left ventricle of the heart and runs inferiorly down to the aortic bifurcation. Most commonly it is separated into the ascending aorta (AscAo), the aortic arch, the descending aorta (DesAo) and the abdominal aorta, where it splits up into the common iliac and further smaller arteries. The aortic root diameter in healthy adults ranges from 25 – 35mm with a normotensive systolic blood pressure (SBP) of 103 – 139mmHg and diastolic blood pressure (DBP) of 66 – 89mmHg [2].

1.1.1 Cardiac Cycle

Throughout the systolic isovolumic contraction of the heart, the pressure within the aorta increases as soon as it is exceeded by the left ventricular pressure. At this point, the aortic valve opens and a pulsatile ejection supplies the arteries with oxygenated blood until the isovolumic relaxation of the cardiac muscle takes place. This is where the aortic valve closes again, as the ventricular pressure falls below the diastolic aortal pressure like shown in Figure 1.

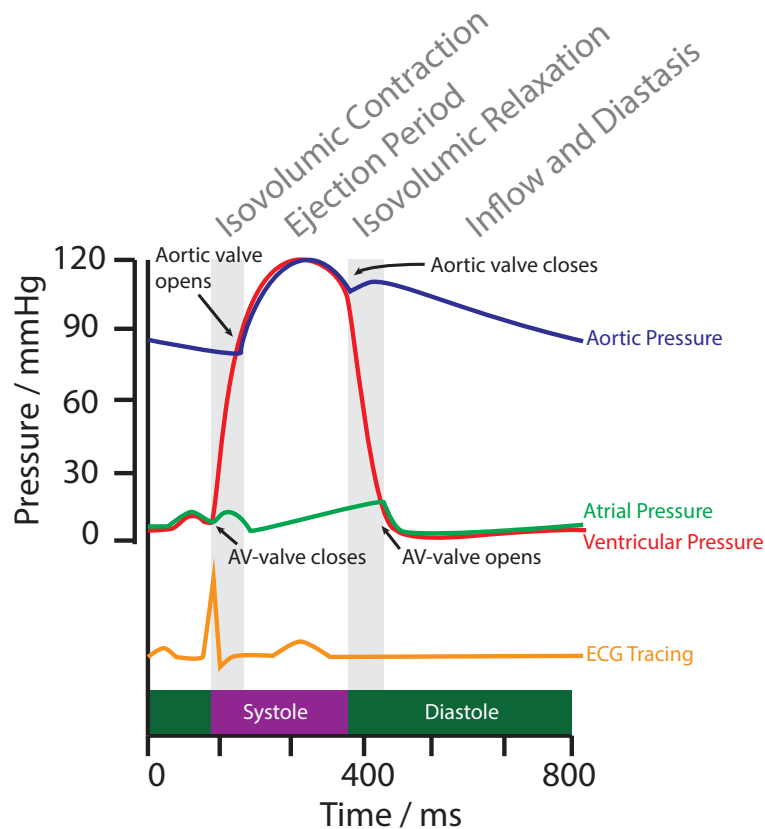


Figure 1: Pressure course of the aorta, left atrium and ventricle as well as ECG tracing shown for one cardiac cycle

1.1.2 Biomechanical Behaviour

During each cardiac cycle, the aortic walls are exposed to pulsatile changes of blood pressure. These walls consist of three layers, the tunica intima (intima), which builds a barrier but stress sensitive smooth inner layer, the comparably thick tunica media (media), contributing active stiffness regulation by its smooth muscle cells and the tunica adventitia (adventitia), which provides nutritional support to outer connective tissue regions. All three layers are composed of different types and quantities of collagen and elastin in which the latter has more presence in large arteries [3]. The main function of collagen is to bear tensile strength, whereas elastin can withstand large strains without breaching [4]. This elastic behaviour is often designated as (arterial) compliance C and can be calculated by the relationship of absolute volume change ΔV per arterial pressure change Δp (1).

The composition of these biological materials leads to a highly nonlinear stress-strain behaviour, like i.e. Zullinger et al. (2004) [5] successfully proved and modelled for

normotensive and hypertensive subjects. Although the arterial wall comprises non-linear, anisotropic and viscoelastic properties, its stress-strain reaction can be roughly characterised by a quantity called “pressure-strain” modulus E_p , defined by equation (2) [3].

$$C = \frac{\Delta V}{\Delta p} \quad (1)$$

$$E_p = R_0 \frac{\Delta p}{\Delta R_0} \quad (2)$$

In equation (2) R_0 denotes the average outer artery radius.

In addition, every vessel builds up a resistance against the passage of blood, commonly summarised for the whole circulatory system as R_{TP} , the total peripheral resistance (TPR) (3), which relies on Δp and the cardiac output Q_H .

$$R_{TP} = \frac{\Delta p}{Q_H} \quad (3)$$

1.1.3 Windkessel Model

One rough but comprehensible approach to model the vascular flow is the two-element Windkessel model, developed by Otto Frank in 1899 [6]. A single chamber (Windkessel) expresses the resilient behaviour of larger vessels to store blood temporarily, formally known as compliance C . This first element is continuously filled by the time-variant cardiac output $Q_H(t)$ from the left ventricle. The second element represents the flow resistance of the peripheral circulation R_{TP} .

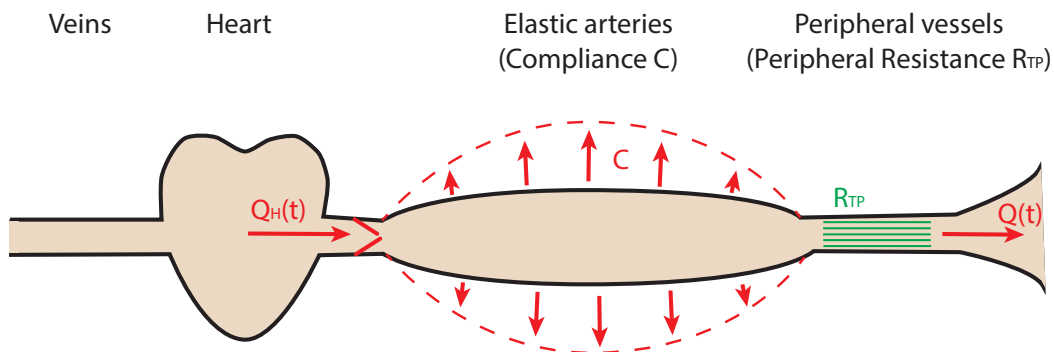


Figure 2: Illustration of two-chamber Windkessel effect

Assuming the incompressibility of blood and disregarding the small back pressure in veins, the arterial blood flow can be expressed as a single ordinary differential equation (4).

$$RC \frac{dQ(t)}{dt} + Q(t) = Q_H(t) \quad (4)$$

1.2 Echocardiography

In echocardiography, each element of a transducer array generates ultrasound pulses that propagate along myocardial structures. These longitudinal mechanical waves with a frequency of 1.5MHz up to 7.5MHz lead to image relevant reflexions at surface boundaries with strongly differing acoustic impedance Z . As a matter of fact, not only reflexions but also refraction and attenuation effects occur and may induce image artefacts, which will not be discussed at this point. Reflected echos from surface boundaries can then be detected by the transducer during the listening time, illustrated in Figure 3. A common simplification is made by the assumption of an overall average speed of sound in tissue, using the speed of sound in water $c_{H_2O} = 1540m/s$.

1.2.1 M-Mode

In TTE a common medical question addresses structure and functionality of heart valves and chambers. This is usually done in B-mode (brightness), showing the reflected amplitudes as different shades of gray encoded pixels in a constantly updated two dimensional image.

In some cases, a grayscale image using one spatial dimension plotted over time is sufficient to prove i.e. the correct opening or closing of the aortic valve. This depth versus time image is also known as M-mode (motion) and uses one transducer element only. The selection of this element recording a single line of sight (LoS) is typically based on the correct positioning of the transducer array in B-mode beforehand.

When using this M-mode for an investigation of the aortic walls, different anatomic landmarks can be used to measure its diameter. In echocardiography the most common method called Leading Edge Technique defines this diameter from the leading edge of the transducer-near to the leading edge of the transducer-far wall.

1.2.2 Axial Resolution

The axial resolution (6) of an ultrasound transducer depends on the spatial pulse length (SPL) (5), in which n is denoted as the number of cycles per pulse whereas λ specifies the wavelength (μm).

$$SPL = n \cdot \lambda \quad (5)$$

$$\Delta x_{ax} = \frac{SPL}{2} \quad (6)$$

When the sonic frequency f (MHz) increases, a better axial resolution can be achieved (7). This equation builds the fundamental relationship between wavelength, speed of sound c and frequency f . The emitted sound intensity $I_0(x)$ (W/m^2) can be adjusted by the examining physician, which certainly affects the intensity of reflected waves $I(x)$. The attenuation of $I_0(x)$ does not just depend on the penetrated distance x , but also on the attenuation coefficient α ($dB/(cm \cdot MHz)$) (8). This coefficient is a function of material properties and sonic frequency.

$$\lambda = \frac{c}{f} \quad (7)$$

$$I(x) = I_0 \cdot e^{-2\alpha(f) \cdot x} \quad (8)$$

1.2.3 Temporal Resolution

The temporal resolution plays a crucial role in M-mode ultrasound imaging. Considering a constant speed of sound c , the minimal temporal resolution, also referred to as minimal pulse repetition period (PRP) depicted in Figure 3, only depends on the penetration depth d set by the examiner (9). This value can be calculated easily for H_2O using d (cm) for depth of view in (10).

$$PRP_{min} = \frac{d}{c} \quad (9)$$

$$PRP_{H_2O} = \frac{d}{154000} \quad (10)$$

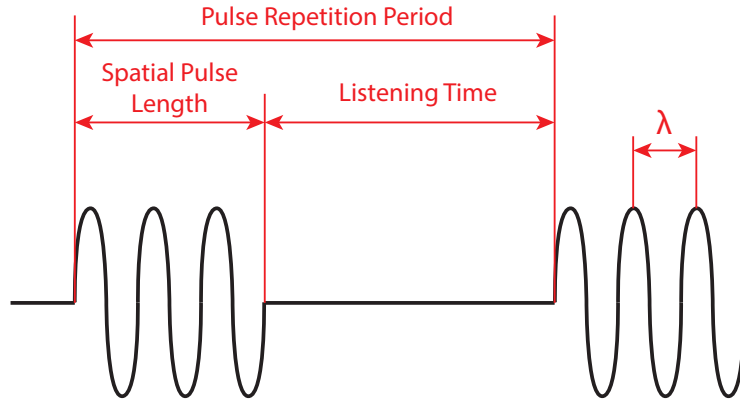


Figure 3: PRP, SPL, listening time and wavelength λ of a ultrasound pulse sequence

1.3 Image Processing Algorithms

The emphasis of object boundaries, but also the reduction of noise are typical operations used in image processing. The following sections will give an overview of image convolution operations, as well as used edge- and line determination techniques used in this work.

1.3.1 Image Filter Kernel

One way to for instance blur or sharpen images in spatial domain is to use symmetric 2D filter kernels. A convolution operation is used in order to calculate the resulting filtered image. This can be imagined as the positioning of the kernel center on each image pixel (neglecting border pixels) and replacing the current pixel value by the sum of each kernel weighted elements.

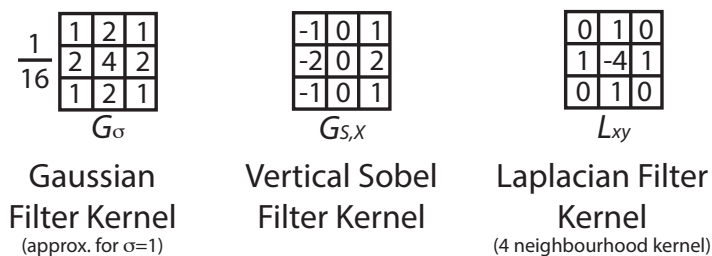


Figure 4: Examples of 3x3 Gaussian, vertical Sobel and Laplacian filter kernels

The Gaussian operator is based on a 2D Gaussian function and can be obtained by a discretisation of $G_\sigma(x, y)$ from (11).

$$G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (11)$$

Its discrete form is widely used to smoothen an image for noise reduction and depends on the variance σ^2 only.

In contrast to that, the Sobel operator G_S is a simple edge detection filter which builds the first order derivative for each pixel in filter direction and additionally smoothen in the orthogonal direction. It is part of the intensity gradient determination for the Canny Edge Detection.

The Laplacian filter aims to find the maximum of the gradients by the determination of the second order derivative and its resulting zero crossings. These found gradient maximum values are considered to be edges.

1.3.2 Morphological Operators

In contrast to the presented kernels in section 1.3.1, morphological operators are non-linear since they set or reset each pixel based on the surrounding morphology and shape. For simplicity, the following operations will be discussed for a 3x3 structuring element with its origin at the center-pixel. This centered structure element is positioned over each non-border pixel of a binary image in cartesian coordinates.

The basic morphological operators used in this work are the dilatation and erosion

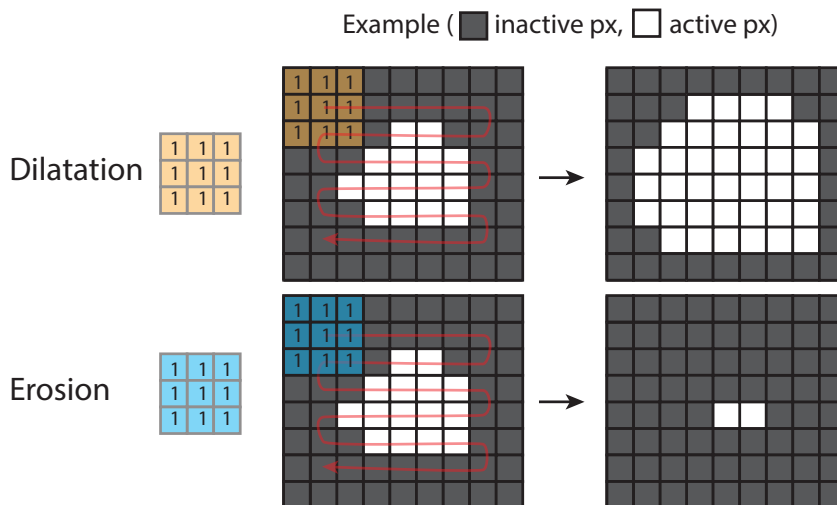


Figure 5: Examples of 3x3 dilatation and erosion operation on a binary image

operator. Dilatation enlarges object boundaries by setting the pixel at the current position active, if at least one pixel in the structuring element matches the covered pixel of the binary image. For a 3×3 structuring element with nine active pixels, the current image pixel is kept inactive only, if all covered image pixels are inactive. Similar to that, the erosion operation reduces the active boundary pixels for kernel positions where at least one of the the structuring element pixels covers an inactive image pixel. The principle of these operations is comprehensible by Figure 5.

1.3.3 Canny Edge Detection

In terms of image processing, an edge can be seen as curve characterised by an intensity gradient perpendicular to its course. The result of its estimation algorithms are binary only, meaning that a pixel can either be an edge pixel or not.

The Canny edge detector is a multi-stage algorithm, published by John F. Canny (1986) [7] that focusses on detecting such edges of an image by passing the following stages:

- **Gaussian blurring** - The pre-implemented algorithm of the OpenCV (open source) library uses a 5×5 Gaussian filter kernel in order to reduce noise.
- **Intensity gradient calculation** - A Sobel operator, in horizontal and vertical direction at a time, is used to calculate the intensity gradient $G(x, y)$ and its direction Θ , denoted by (12) and (13).

$$G(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2} \quad (12)$$

$$\Theta = \arctan \frac{G_y(x, y)}{G_x(x, y)} \quad (13)$$

- **Non maximum suppression** - Each resulting value of $G(x, y)$ is compared with its neighbours in gradient direction. If the calculated intensity gradient of the neighbour is greater than the current pixel's gradient, the current pixel's gradient is set to zero. This results in a gradient image with single-pixel edges only.
- **Hysteresis** - Based on a defined lower and upper gradient threshold, the algorithm finally determines which of the maximum gradient values are accepted. Edges with gradient pixels greater than the upper threshold are certainly accepted, those that are below the lower threshold are rejected. If its value lies in between the lower and upper threshold, it is accepted as edge only if it is

connected to an edge with values greater than the upper threshold.

1.3.4 Active Contour Model

“A snake is an energy-minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges. Snakes are active contour models: they lock onto nearby edges, localizing them accurately” [8]. A snake converges from its initial user defined position iteratively towards an object by minimizing an energy functional E_{snake}^* in (14). Its original formulation by M. Kaas et al. (1988) [8] includes three energy terms.

$$\begin{aligned}
 E_{snake}^* &= \int_0^1 E_{snake}(\mathbf{v}(s))ds \\
 &= \int_0^1 E_{int}(\mathbf{v}(s)) + E_{image}(\mathbf{v}(s)) + E_{cont}(\mathbf{v}(s))ds
 \end{aligned}
 \tag{14}$$

The snake itself is integrated as parametrised curve $\mathbf{v}(s) = (x(s), y(s))$, modelled as a combination of smoothness α and elasticity β parameters by the internal energy term E_{int} . In addition, an external image related energy E_{image} is part of the functional, as well as a constraining energy E_{cont} to specify prior knowledge about the image, which can be disregarded for first approximations.

Main advantages of this computational costly edge detection method is its “insensitivity up to a certain level of noise” [9], as well as the detection of not necessarily connected structures.

1.3.5 Morphological GAC

One of the major problems of the previously presented approach in 1.3.4 is the dependence of the result on the snake’s parametrisation. This issue can be addressed using a morphological implementation of the geodesic active contour (GAC) model, presented by Márquez-Neila et al. (2014) [10]. Instead of specifying the snake values α and β , this level-set approach uses intrinsic geometric features of the image to evolve a geodesic curve along the image content. The initial location and shape of the curve must be defined, which encloses a binary hypersurface $u : \mathbb{Z}^d \rightarrow \{0, 1\}$ that takes the value $u(\mathbf{x}) = 1$ for every point \mathbf{x} inside the hypersurface and $u(\mathbf{x}) = 0$ for outer

points. It can be described using a level-set partial differential equation (PDE) approach including three forces, composed in (15). The smoothing force, expressed by the curve's curvature $|\nabla u| \operatorname{div} \left(\frac{\nabla u}{|\nabla u|} \right)$, the balloon force [11] in order to move the contour along non-informative areas (driven by the balloon force parameter $v \in \mathbb{R}$) and the image attraction force $g(I)\nabla u$.

$$\frac{\partial u}{\partial t} = g(I)|\nabla u|v + g(I)|\nabla u| \operatorname{div} \left(\frac{\nabla u}{|\nabla u|} \right) + g(I)\nabla u \quad (15)$$

$$g(I) = \frac{1}{\sqrt{1 + \alpha|\nabla G_\sigma * I|}} \quad (16)$$

Relation (16) defines the edge attracting image content $g(I)$ by convolving the image I with the derivative of a α weighted Gaussian function G_σ .

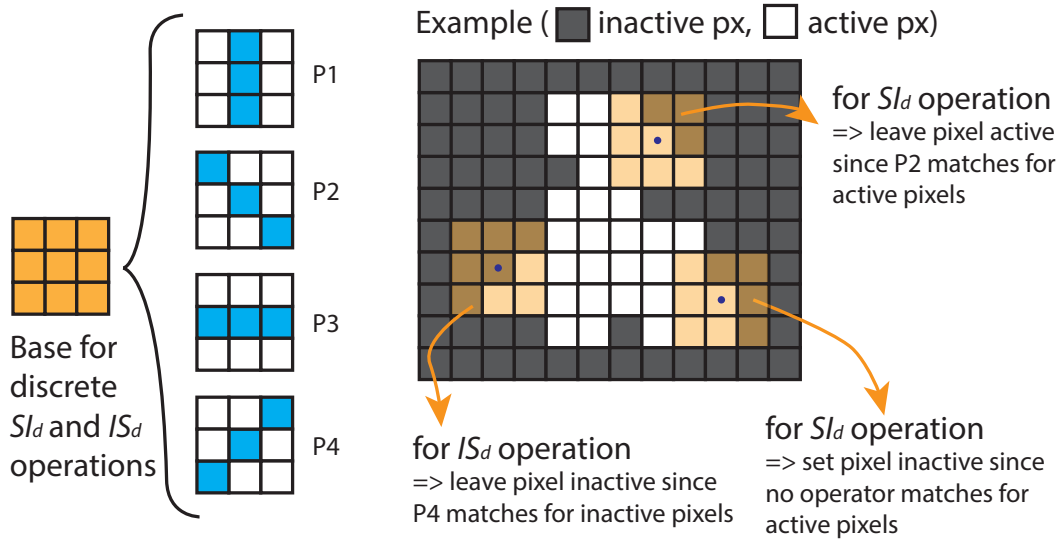


Figure 6: Base configurations for the $SI_d \circ IS_d$ operator (left) and usage example (right)

In order to solve (15), the balloon as well as the smoothing force can be simplified by using binary morphological operations. First of all, the balloon force is required only, when the hypersurface is located far from the target border. Therefore it can be substituted by a discrete dilatation D_d or erosion operation E_d (depending on the sign of v) whenever $g(I)$ exceeds a defined threshold denoted as θ .

Secondly, the actual edge attracting force can be determined straight forward by a

multiplication of the known $g(I)$ with the spatial derivation of the current hypersurface ∇u .

Last of all, the smoothing force can be approximated by the combination of two novel morphological operators denoted as SI_d (supremum-infimum) and IS_d (infimum-supremum). The principle of these operators can be obtained by Figure 6. For any active pixel x_i , the SI_d operator sets this pixel inactive if none of the curvature morphological patterns matches and sets it active for a matching pattern. For the IS_d operator a similar procedure is carried out for inactive pixels. Márquez-Neila et al. defines this composition $SI_d \circ IS_d$ as *discrete curvature morphological operator*, that removes all sharp inactive pixels by using IS_d and repeats this task for active pixels with SI_d afterwards. The number of smoothing repetitions and therefore the strength of smoothing can be adjusted by the parameter $\mu \in \mathbb{N}$. A combination of these simplifications lead to the iterative morphological implementation of GAC snake u^n for iteration n in (17).

$$\begin{aligned}
u^{n+\frac{1}{3}}(\mathbf{x}) &= \begin{cases} (D_d u^n)(\mathbf{x}), & \text{if } g(I)(\mathbf{x}) > \theta \text{ and } v > 0, \\ (E_d u^n)(\mathbf{x}), & \text{if } g(I)(\mathbf{x}) > \theta \text{ and } v < 0, \\ u^n(\mathbf{x}), & \text{otherwise} \end{cases} \\
u^{n+\frac{2}{3}}(\mathbf{x}) &= \begin{cases} 1, & \text{if } \nabla u^{n+\frac{1}{3}} \nabla g(I)(\mathbf{x}) > 0, \\ 0, & \text{if } \nabla u^{n+\frac{1}{3}} \nabla g(I)(\mathbf{x}) < 0, \\ u^{n+\frac{1}{3}}, & \text{if } \nabla u^{n+\frac{1}{3}} \nabla g(I)(\mathbf{x}) = 0 \end{cases} \quad (17) \\
u^{n+1}(\mathbf{x}) &= \left(\left(SI_d \circ IS_d \right)^{\mu} u^{n+\frac{2}{3}} \right) (\mathbf{x})
\end{aligned}$$

In a few words, the method of Márquez-Neila et al. executes the iterative curve evolution process by approximating the solution of a time-dependent smoothing function, using a combination of established morphological operators. This procedure improves not just the stability of the solution, but also reduces the computational effort to solve the underlying PDE.

1.3.6 Hough Line Transform

The Hough transformation can be used to detect any parametriseable figure on a binary edge image, which can be generated by building the 2D-gradient of it. In case of detecting lines, the parametrisation equation (18) can be used to describe any line that crosses a chosen point (x, y) by a variation of α within $[-90, +89]$ degree and d limited to $[-\sqrt{x_{max}^2 + y_{max}^2}, \sqrt{x_{max}^2 + y_{max}^2}]$.

$$d = x \cdot \cos \alpha + y \cdot \sin \alpha \quad (18)$$

The idea is that each point (x, y) on a binary edge image is iterated and the parameters of intersecting lines expressed by (18) with a variation of α are assigned to the 2D Hough space. It is obvious that for fixed edge point x and y , the variation of α will generate a sinusoidal course in Hough space, like shown for the points $P1 - P3$ in Figure 7.

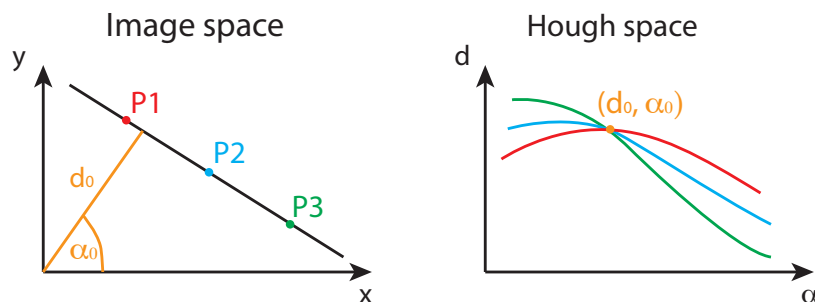


Figure 7: Points P1 - P3 on a line in image space shown as one single intersecting point in Hough space

Each time a sine-curve intersects this point (d_0, α_0) , its value in the Hough space is iterated by one. When each edge point in the image was transformed by equation (18) with all possible α values, high values for discrete d and α positions are indicators for a line with this parametrisation.

1.4 Regulatory Aspects of Software as a Medical Product

The Medical Device Act (MDA) is based on its overall objective, namely to adapt the European law as well as to harmonise national regulations, on European but

also on national principles². This Austrian law incorporates European regulations (90/385/EWG, 93/42/EWG, 98/79/EG) to a national level. Since 25 May 2017, the MDR 2017/745 takes over two of these former legislative regulations of general (MDD 93/42 EWG) and active implantable medical products (AIMD 90/385) to place medical products on the European market. With a transition period of about three years (application of regulation planned for 2020), it includes distinct rules to classify these products into one of four risk classes. The relevant part of the MDR 2017/745 rules as well as essential regulatory demands for this thesis will be outlined in the upcoming sections.

1.4.1 EU Risk Classification

In order to place a medical product on the market, the manufacturer has to proof conformity to be compliant to European directives. One major step in this process of CE-marking is the specification of the medical product's risk class (I, IIa, IIb, III). According to the MDR 2017/745, a set of 22 rules define the classification of a medical product based on its intended use, in which rule eleven specifies the classification of software as follows:

“Software intended to provide information which is used to take decisions with diagnosis or therapeutic purposes is classified as class IIa, except if such decisions have an impact that may cause:

- *death or an irreversible deterioration of a person's state of health, in which case it is in class III; or*
- *a serious deterioration of a person's state of health or a surgical intervention, in which case it is classified as class IIb.*

Software intended to monitor physiological processes is classified as class IIa, except if it is intended for monitoring of vital physiological parameters, where the nature of variations of those parameters is such that it could result in immediate danger to the patient, in which case it

²“Das Medizinproduktegesetz (MPR) basiert entsprechend seiner hauptsächlichen Zielsetzung, nämlich der Anpassung an europäisches Recht sowie der Vereinheitlichung der nationalen Vorschriften, sowohl auf europäischer als auch nationalen Grundlagen” [12]. (translated by the Bernhard Frohner)

is classified as class IIb.

All other software is classified as class I.”

cited from [13]

1.4.2 Software Lifecycle

One major step towards the CE marking of medical products in Europe is a complete technical documentation, authored by the manufacturer. This documentation must be extensive enough to prove essential requirements which can be fulfilled by developing the application along software lifecycle processes. The EN 62304:2006+A1:2015 defines demands on medical software, including phases of software development, maintenance and decommissioning in which this work will focus on the implementational phase. This norm does not prescribe a certain process model for software development, though it specifies mandatory documentation as mentioned in [14], including:

- **Software Requirement Specification (SRS)** - This document defines requirements for software interfaces (user, hardware, software, communication), safety, security, quality, performance, runtime-behaviour, functionality and legal aspects.
- **Architecture of software and detailed design** - The software architecture should identify, describe and classify software components, as well as their interfaces. Other included aspects should be traceability, but also the identification and requirements on SOUPs.
- **Verification of software-units (code review)** - This is not explicitly required, but recommended by EN 62304:2006+A1:2015. As *AortUs* is implemented by one developer only, this aspect will be treated superficially.
- **Integration and system-tests, including test-specification and results** - To ensure correct functionality of single software components but also of their assembly, integration and system-tests are executed and results discussed. This part was omitted due to its extensive effort, in accordance with Univ.-Prof. Dipl.-Ing. Dr. techn. Christian Baumgartner (CB).

The extensiveness of this documentation mainly depends on the software-safety class chosen by the manufacturer. The classes are divided into:

- **Class A** - No injury or damage of health possible

- **Class B** - Non serious injury is possibly
- **Class C** - Death or serious injury is possible

Especially the detailed documentation of software architecture and verification is compulsory for classes B and C only. [14].

1.4.3 Demand for Usability

One part of the expiring Medical Device Directive (MDD) is its demand for usability of medical products, in order to keep risks originating from insufficient usability as low as possible. The EN 62366 [15] conducts this by following a “Usability Engineering Process”, integrating the actual definition of the use specification, a comprehensive statement of the medical device’s usability aspects and a verification and validation of these aspects. [14].

2 Scope of Work

2.1 Overall Goal

The goal of this thesis was to implement a medical image processing software that is capable of estimating the aortic distensibility, the stiffness-index and the systolic diameter increase parameters based on M-mode echocardiography records. Besides the implementational part, the software should also be classified into a risk class according to the MDR 2017/745, with additional respect to the software lifecycle according to EN 62304:2006+A1:2015.

2.2 Preliminary Work

Motivated by the outcome of the study of Baumgartner et al. (2005) [16], processed M-mode images of the aorta turned out to be a powerful diagnostic indicator for young people suffering from Marfan syndrome. It could be shown that the established multiple regression model based on aortic mechanical parameters could achieve a sensitivity of up to 100% and a specificity of 94.7%.

A subset of the AscAo and DesAo M-mode images for this study were provided by CB in order to develop this software (33 images in total). These M-mode recordings used during the implementation of *AortUs* were taken at two ultrasound transducer positions - the proximal AscAo 10 to 20 mm distal to the sinotubular junction (parasternal long-axis view) as well as the abdominal DesAo proximal to the branching off of the celiac trunk (abdominal paramedian longaxis view) [16]. The examiners took care of correct positioning of the line of sight, being perpendicular to the aorta's long axis on the circumferential position of maximal aortic diameter changes.

The images arise from ultrasound devices of two different manufacturers and were provided in "*.JPG" format with 8-bit depth for each RGB-channel.

2.3 Minimum Requirements

The software should facilitate an easy way to calculate the aortic wall distensibility, stiffness-index and systolic diameter increase, based on at least 5 cardiac cycles of M-mode data of AscAo or DesAo. For this computations, it should be possible to process

or enter ECG-data recorded in parallel to the M-mode image as well as oscillometrically established blood pressure values.

Intended users for this application are medical professionals, that control an intuitive process of semiautomatic edge detection. The software should be implemented in a way, that at least an interface for an optional extension for the surveillance of the previously mentioned vascular parameters is integrated. If image artefacts disturb the tracking of the physiological wall distension significantly, the software may also reject this image from being processed.

For more detailed information of functional and non-functional requirements of *AortUs*, the interested reader is referred to the Software Requirement Specification in the Appendix.

3 Methods

In order to understand the assumptions for the calculation of elasticity parameters but also the underlying extraction process, the following sections will describe these topics in detail.

3.1 Calculated Aortic Parameters

One very simple but interesting parameter is the systolic diameter increase d_{inc} in (19)

$$d_{inc} = \frac{d_s - d_d}{d_d} (\%) \quad (19)$$

where d_s denotes the systolic (maximal) and d_d the diastolic (minimal) diameter of the aorta. Based on this relation, the assumption of mainly radial arterial distension and the definition for arterial compliance C in (1), one can also derive a similar relative compliance parameter referred to as distensibility D .

$$D = \frac{A_s - A_d}{A_d \cdot (p_s - p_d) \cdot 1333} \cdot 10^7 (kPa^{-1} \cdot 10^{-3}) \quad (20)$$

Equation (20) considers the systolic aortic lumen A_s as well as the diastolic minimal aortic lumen A_d (both in mm^2) and can therefore be seen as relative analogon to equation (1) at fixed vessel length. Parameters p_s and p_d (both in $mmHg$) denote the systolic and the diastolic blood pressure (BP), respectively. Although the systolic and diastolic BP values from central BP measurement slightly differ from non-invasively determined from the right upper arm [17], the latter is still a good approximation since the brachial artery originates from the first branch of the aortic arch. Therefore, oscillometrically established systolic and diastolic BP values are used in this formula. Another simplification is made, as A_s and A_d are assumed as perfectly circular lumen with radius $d/2$. Since this is not always the case, one could make an error estimation by assuming an elliptically shaped aorta with $A_{ell} = a \cdot b \cdot \pi$, where a and b denote the half width and height of the ellipse. When assuming an ellipse with shortened half height b , the positioning of the LoS along b will result in a calculated area of size A_1 , smaller than the actual area A_2 (*Area failure F1*). Similarly, the calculated area A_1 is

bigger than the actual area A_2 , when the LoS reflects along the direction of the half width a (*Area failure F2*). Both cases *F1* and *F2* lead to an error of 25% like shown in Figure 8.

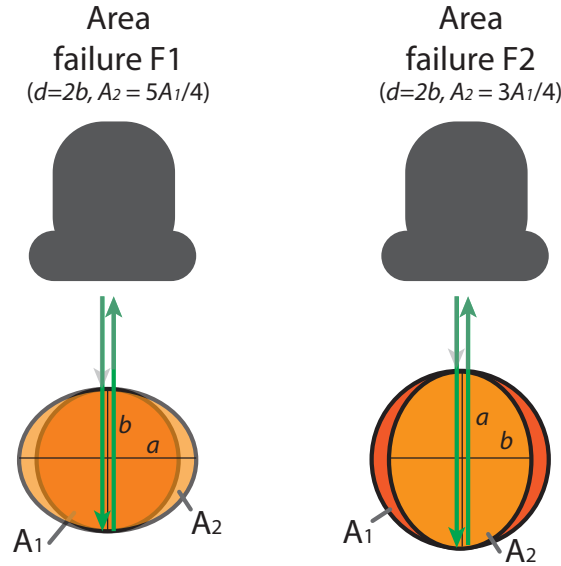


Figure 8: Comparison of circular assumed aortic cross section with two elliptical cross sections

Another parameter similar to distensibility D used to address cardiovascular impairment is the local arterial stiffness-index SI or known as β -index (21).

$$SI = \frac{\ln \frac{p_s}{p_d}}{d_{inc}} (\text{dimensionless}) \quad (21)$$

Typical values for healthy patients younger than forty are for the AscAo (DesAo) a d_{inc} within $18.0 \pm 6.1\%$ ($18.6 \pm 6.1\%$), D within $62 \pm 24 kPa^{-1} \cdot 10^{-3}$ ($65 \pm 30 kPa^{-1} \cdot 10^{-3}$) and SI within 3.4 ± 1.4 (3.2 ± 1) [16]. Nevertheless, the gold standard to estimate the arterial stiffness and distensibility is to measure the pulse wave velocity (PWV) [18, 19], expressed by the length of a vessel segment L , divided by its pulse transmit time of a pressure waveform to pass it. Fortunately the PWV is inversely proportional to the square root of the distensibility, like shown by the Bramwell-Hill [20] equation in (22).

$$PWV \propto \frac{1}{\sqrt{D}} \quad (22)$$

3.2 Software Requirement Specification

One major part of the technical documentation of software developed along the software lifecycle EN 62304:2006+A1:2015 is the SRS. Its objective is to specify all a-priori demands of the system, but also to drive the actual development process and reduce risks in a systematic way.

Requirements regarding *AortUs* were defined on basis of the system specifications elaborated and subscribed by CB and Bernhard Frohner, BSc. (BF), but were also influenced by the goal to maximize the level of automation with whilst keeping the software flexible enough to perform or correct certain image processing steps manually (i.e. scales and resolution detection). The structure of the created SRS document was taken over from the Institute of Electrical and Electronics Engineers (IEEE) 830-1998 standard and describes basically:

- the overall goal of this project and its users
- the layout and functionality of the Graphical User Interface (GUI)
- functional requirements
- demands of performance, safety, security, quality and legal aspects

Due to its extensiveness, the full Software Requirement Specification document can be found in the Appendix. The consequences of these requirements are represented by the sections Software Architecture and Edge Detection Process.

3.3 Programming Environment

Starting at the implementational part, one of the first tasks was to set up the main demands for this software project. When reviewing programming possibilities, the interpreted coding language Python 2.7 combines easy usage with manifold image and math libraries including collaborative support. As the development of the software is done on a Unix based system but most likely to be used on Windows (Microsoft, Redmond, USA) operating system (OS), this programming language also convinces by its platform independence. For a detailed listing of used packages, the interested reader is referred to section Python Packages and Versions in the Appendix.

To simplify the set up of a Python 2.7 project, the integrated development environment (IDE) PyCharm Community 2016.3 (JetBrains, Prague, Czech Republic) was used,

including version control by using the open source software plugin Apache Subversion in combination with the Subversion@TU Graz service.

3.4 Software Architecture

Since functionally good but over-engineered software often remains unused due to its high complexity, the focus of *AortUs* was a comprehensive high level of usability. Considering this, the graphic appearance and user interaction related demands were developed by the combination of unambiguous relatable patient information with functional aspects based on the edge detection process, described in 3.5. This software can therefore be roughly separated into three implemented software layers, depicted in Figure 9.

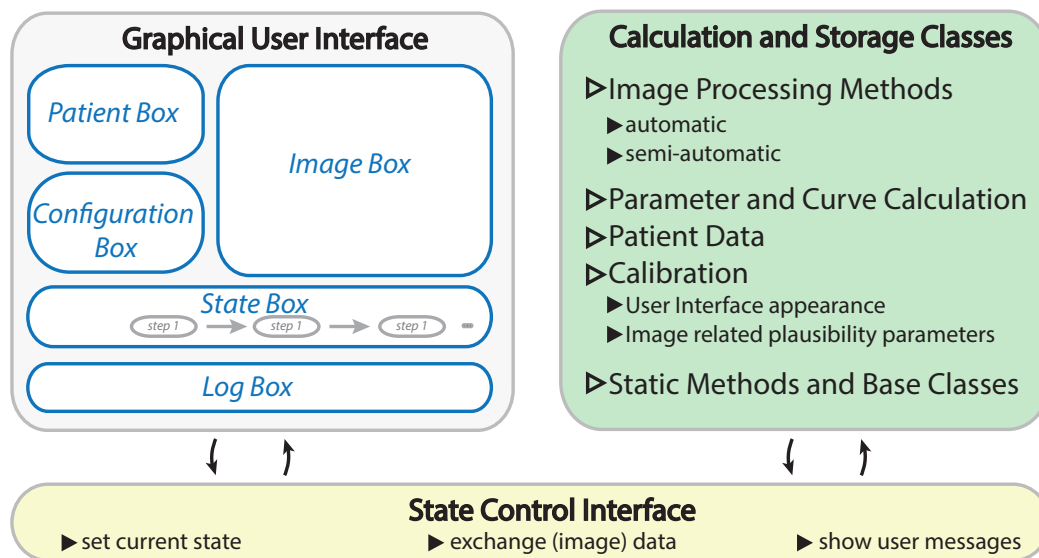


Figure 9: Overview of *AortUs* software architecture layers

These layers are implemented as single (*GUI* and *UsImage*) and multiple backend classes in an object oriented approach.

3.4.1 GUI class

The GUI is implemented as one single class named *GUI*, that creates a window embedding five different kinds of boxes or also called layout containers. These containers

align widgets like push-buttons, textboxes, images, graphs that are can be easily integrated and manipulated by using a well-known cross-platform frontend toolkit called GTK+.

- **Patient Box** - This box embeds input fields for the patient's first and last name, the insurance number, the date of birth and of examination (both in format "DD.MM.YYYY") and the established systolic and the diastolic BP values (in *mmHg*). Entered BP values are averaged and shown as colum label for systolic and for diastolic values respectively.

The transducer position of the investigated body part can be selected in a list with the predefined entries "Ascending Aorta" and "Descending Aorta". It is necessary to set this value correct in order to calibrate the detection algorithm's parameters and plausibility checks. In addition, the sections per centimeter (one per default) as well as per second (five per default) must me calibrated manually since *AortUs* is not capable of extracting this info from the image's inter- and main-scale intersections automatically.

The keyboard-input elements are implemented as input restrictive fields, so that impossible combinations are rejected (i.e. numeric entries in name fields, alphabetic entries in numeric fields, entered diastolic higher than systolic BP). It should be noted, that all of the previously mentioned fields can be accessed by pressing the "TAB" key, including the BP list, that is appended by an empty entry automatically when pressing this key to enter its fields. Only if all input fields are filled, the first state of the edge detection process is enabled.

The bottom of the *Patient Box* offers the actions "Clear Detection", "Clear Patient" and "Export Results". Whenever the user wants to exit the current state of edge detection, a click on "Clear Detection" instantly cancels the process, clears all backend member objects and jumps back to the "Load Image" state. Additionally, the patient-related information can be reseted as well by triggering "Clear Patient". Since the resulting aortic parameters may be of interest for later clinical decisions, the determined aortic parameters as well as the single- and average aortic diameter curves can be exported to a examination report PDF by the usage of "Export Results" pushbutton.

- **Configuration Box** - In almost every state of the edge detection process, manual

corrections by the user are possible. The *Configuration Box* therefore contains interactive pushbuttons and tables but also plain info text to inform and guide the user.

- **Image Box** - Since the user should always be aware of the current state results, the *Image Box* contains a tab to display the loaded M-mode image masked by highlighted found image content (scale axes and ECG in green, boundaries of aortic area in yellow, triggerpoints and wall edges in red). Some of this highlighted components can be manually adjusted by the user at certain states, as described in 3.5.

When the user successfully finishes the final edge detection state, a second tab appears within the *Image Box*. This tab named “Aortic Parameters” contains:

- a plot showing the “Single Aortic Diameter Courses”
 - a plot showing the overall “Averaged Aortic Diameter Course”
 - a box listing the calculated aortic parameters, according to 3.1
 - an interactive treelist to remove outlying single aortic diameter curves from calculation
- **State Box** - Each of the edge detection states can be initiated by a click on the related pushbutton in the *State Box*. Detection states that cannot be reached from the current one, are restricted by disabling these buttons.
 - **Log Box** - The *Log Box* provides the user with additional information about the latest events and results (i.e. name of loaded image, detected axes resolution). This information is not necessarily important for the moment, but may be relevant for later commercial use.

3.4.2 UsImage class

To establish a flexible and clear way of communication between the *GUI* class and the algorithm containing classes, an interface layer is used. The main advantage of this separation of visual components from functional parts is that the former can be replaced easily or even neglected in case of direct programmable control. Similarly to the *GUI* class, this controlling interface is implemented as one single class named *UsImage*. This class embeds basically two functionalities:

- Interface between *GUI* class and algorithm containing classes

- Control of backend classes by following the edge detection process

3.4.3 Backend Classes

The actual scaling axes and ECG extraction, as well as estimation for the aorta's position and the edge- and parameter calculation is processed in backend classes of *AortUs*. The most relevant classes for further understanding are described in the following:

- **Patient and Examination classes** - These classes process and store all patient relevant data including personal data, the loaded and processed image themselves and the extracted diameter curves and parameters. Since *AscAo* as well as *DesAo* images can be loaded, it also references to a list of *TransducerPosTypes*, each element containing a list of *Examinations*. Since it is required to facilitate processing multiple M-mode images of one patient to one parameter set, each entry in this list of *Examinations* is created for one loaded image. Each examination in turn contains a list of diameter curves for this image. The embedded structure can be obtained by Figure 10.

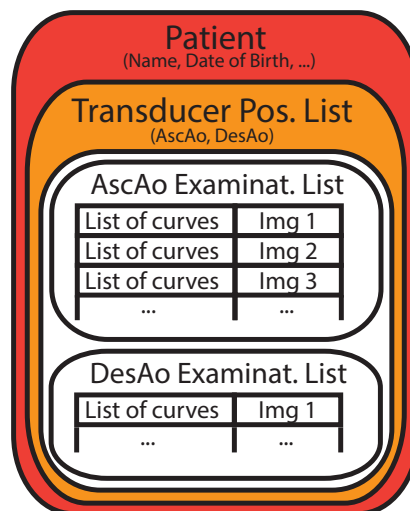


Figure 10: Storage structure of diameter curves in *Patient* class

- **Scale class** - This class handles the automatic detection, but also the manual scale definition process. Its main target is to determine the M-mode resolution in pixel per second, the pixel per centimeter as well as the position and dimension of the actual M-mode region of interest (ROI).

- **ECG and R-peaks classes** - One requirement of loaded aortic images is the parallel record of the patient's ECG tracings in order to obtain the ventricular contraction point at the R-peak. The *ECG* class implements the automatic extraction of this curves and creates a list of determined R-peaks used as triggerpoints. These triggerpoints are used to average the diameter curves over heart-cycles.
- **Aorta class** - The *Aorta* class embeds the estimation of the position of the aorta within the M-mode ROI, but also controls the detection of aortic wall's edges by calling methods of the *Edges* class. It can be seen as the interface between the extracted M-mode image as input and the calculated list diameter curves as output.
- **(static) Edges class** - This class provides several static methods to split, merge, reconnect a passed list of edges, based on certain properties (i.e. geometric distance, gradient of definition direction).
- **Calibration classes** - These classes are used to define general aspects of the software project (flag definitions of debug modes), as well as physiologically useful thresholds like minimal/maximal heart rate and time frames for the detection of minimal and maximal diameters.

For more detailed information about the functionality and structure of the classes, the interested reader is referred to the Software Documentation in the Appendix.

3.5 Edge Detection Process

The edge detection process is the backbone of *AortUs* and separates its aim into six consecutive steps. It starts with loading an M-mode image and ends with the calculation of aortic parameters and triggered aortic diameter curves over time. Each of these steps can be initiated by a click on the according state button in the *State Box* and is further processed by the *UsImage* class.

3.5.1 Load Image

When the user entered all patient relevant information in the *Patient Box*, the initiation of the first state is enabled. The user is asked to pass a `"*.jpg"`, `"*.png"`, `"*.tif"` or `"*.bmp"` M-mode image of the AscAo or DesAo that also includes a coloured ECG

tracing. Since there is no unique standard for M-mode images, no additional input image plausibility check is performed at this point.

Its implementation is a very straight forward usage of the GTK+ *FileChooserDialog*. The image itself is then stored and handled in the previously mentioned administrating *UsImage* class.

3.5.2 Detect Scales

After an M-mode image was successfully loaded, the scale detection step is performed in automatic mode first. It aims to find the main M-mode image axes, the ROI as a result of the found axes that span it and their pixel-scalings (pixels per *cm* and *sec*). Since the M-mode axes in the provided dataset are shown as bright lines on a black background, the first step in the automated mode is to threshold and binarise the image. For the detection of main axes, the resulting lines of the Hough Line Transform are filtered and checked for plausibility. A certain length in relation to the image width and height must be reached on one hand and the orientation must be purely vertical and horizontal, respectively. The pixel resolution is then determined by averaging the pixels used for inter-subsections on each axis.

This automatic algorithm fails whenever one of the axes is interrupted or not clearly detectable (i.e. the user changes transducer-device settings during record which may result in vertical black bar in the image, usage of ultrasound device-specific measurement cursors). In this case, the user will be informed about the unsuccessful automatic mode and guided through the manual scale detection process, based on interactions on the image plane:

1. Selection of a rectangular M-mode image area (M-mode ROI)
2. Definition of a horizontal line, equal to one second
3. Definition of a vertical line, equal to one centimeter

Each of these steps is defined by a click-and-drag interaction in the *Image Box*, that highlights a coloured rectangle or line and must be confirmed by a click on the “Next” button in the *Configuration Box*.

3.5.3 Detect ECG

As already mentioned, the loaded and detected M-mode image area must also contain a coloured ECG tracing. This ECG is used later on to split the consecutive aortic wall curves into a list of cardiac cycle based curves.

The semiautomatic step starts with hue saturation value (HSV) filtering the image to coloured content. Since the ECG tracings of the provided images are partially disturbed by thick vertical coloured cursors, the next step to perform is a vertical line suppression on the remaining image content. After this is done, a fast border following edge detection algorithm is applied, based on the work of Suzuki et al. (1985) [21], to detect the ECG tracing. To make sure that no image content is mistakenly classified as ECG, the found contours are rated by the sum of the tracing's absolute derivation in time domain, as well as the derivation's standard deviation and the relative coverage of the found contours in time domain, compared to the M-mode ROI's width.

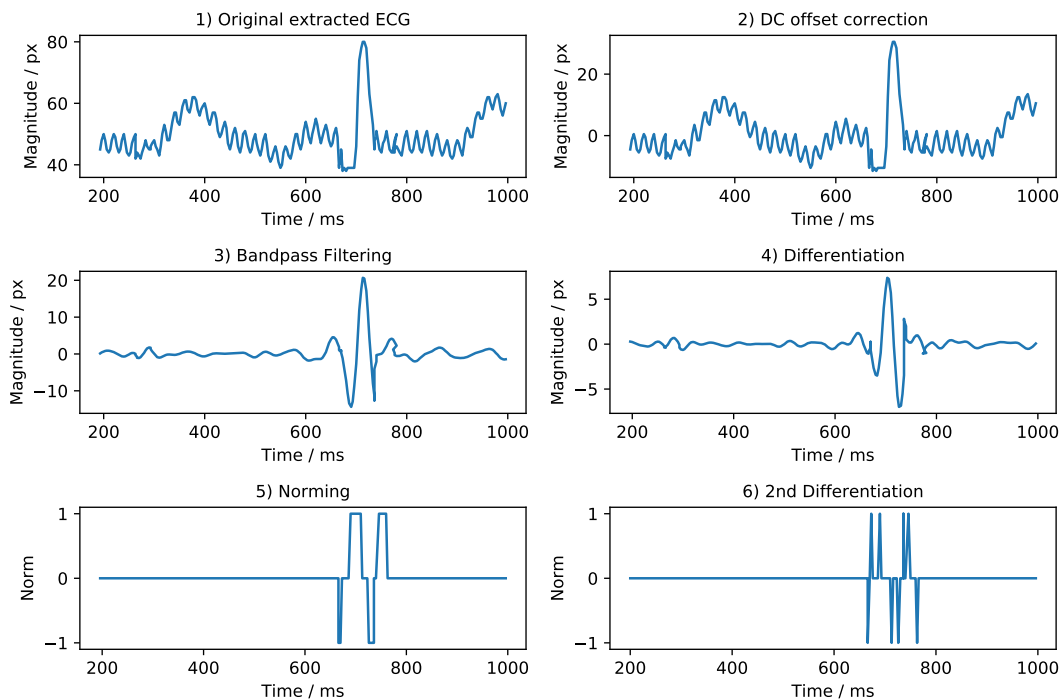


Figure 11: R-peak area detection by using second order derivative calculation, based on a image extracted ECG tracing

After the ECG tracing is extracted, the algorithm basically performs a bandpass filtering of ECG relevant frequencies ($f_{cutoff} = [10, 30]Hz$, sixth order), then calculates

the first order derivative and norms all threshold exceeding data points to one or zero. A second derivation of this rectangular graph is then used to mark start- and end-indices for the expected R-peaks positions by finding the local maximum values. Figure 11 shows this process on a cutout of an image's ECG tracing from the provided dataset.

Found peaks must fulfill the plausibility criteria to fit an average heart rate of $[40, 220]bpm$. By default, the triggerpoints are offsetted to $-50ms$ prior to these detected R-peaks if possible. The result of this step is the masked, green highlighted ECG tracing on the image, including red circles used for estimated triggerpoints.

Due to ECG tracing shifts, too small R-peaks or record artefacts, not all R-peaks may be found correctly. Therefore the user can adjust the position of any red-circled triggerpoint by dragging it along the detected ECG tracing. Additionally the *Configuration Box* offers a listview to interactively add (green "+") or remove (red "-") triggerpoints, but also to set a time-based offset for one or even all of them ("*All" button at the bottom of the list). Any added triggerpoint is created between the "last" triggerpoint in time domain and the right M-mode image border. If at least two triggerpoints were found or defined, the user can proceed to the next state.

3.5.4 Detect Aorta

This step does not detect the walls of the aorta, but its limiting aortic ROI used for detection. It is necessary since the performance of the edge detection algorithm increases for smaller images but also due to an easier error handling.

For an approximation of the aortic ROI, an algorithm was established, starting with the previously mentioned contour finding algorithm [21] on a slightly blurred image (Gaussian filter, $\sigma = 0.33$, *kernel size* = 3). Found contours are then rated and filtered by their length but also their deviation over time. During the development of this approach it turned out, that an image-row summation of rated contour-pixels produces a good first estimation for the region of interest. The principle of this idea can be obtained by Figure 12.

A peak detection is then performed on this projection and the most reliable combination of physiologically possible peaks is chosen to indicate the outer borders of the aortic walls. These borders are highlighted as two yellow horizontal lines on the image.

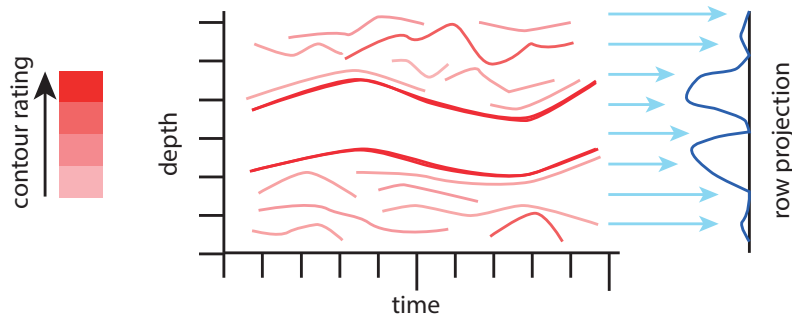


Figure 12: Illustration of rated contours in aortic M-mode image and its projection over rows by summation

Since this approach gives a good estimation for M-mode images with high aortic wall contrast, other image components from connective tissue may disturb this approximation. Therefore, the user is requested to ensure correctness or manually adjust the result by dragging the yellow-lines close to the outer borders of the aortic walls and clicking the “Confirm” button in the *Configuration Box*.

3.5.5 Detect Edges

In the edge detection state, *AortUs* tries to find time-variant distortion of the vessel borders within the aortic ROI. Basically there are a lot of image processing algorithms available for this task, like i.e. the Sobel or Laplace convolution operators, or the well known Canny edge detector. These algorithms could have been used easily in *AortUs* since they are part of the vast OpenCV library. Although it looks like a simple task, it is difficult to implement it this way since M-mode images typically come along with challenging quality aspects:

- low general brightness resolution
- time dependent brightness changes (depending on transducer-body coupling and organ movements)
- reverberation artefacts of ultrasound
- run-time dependent amplitudes of ultrasound reflexions (lead to unsharp contours of deeper reflexions)

These attributes lead to discontinuous wall edges as well as the detection of irrelevant edges for the mentioned algorithms. A comparison of three different edge detection algorithms on an 8-bit grayscale aortic M-mode ROI is shown in Figure 13.

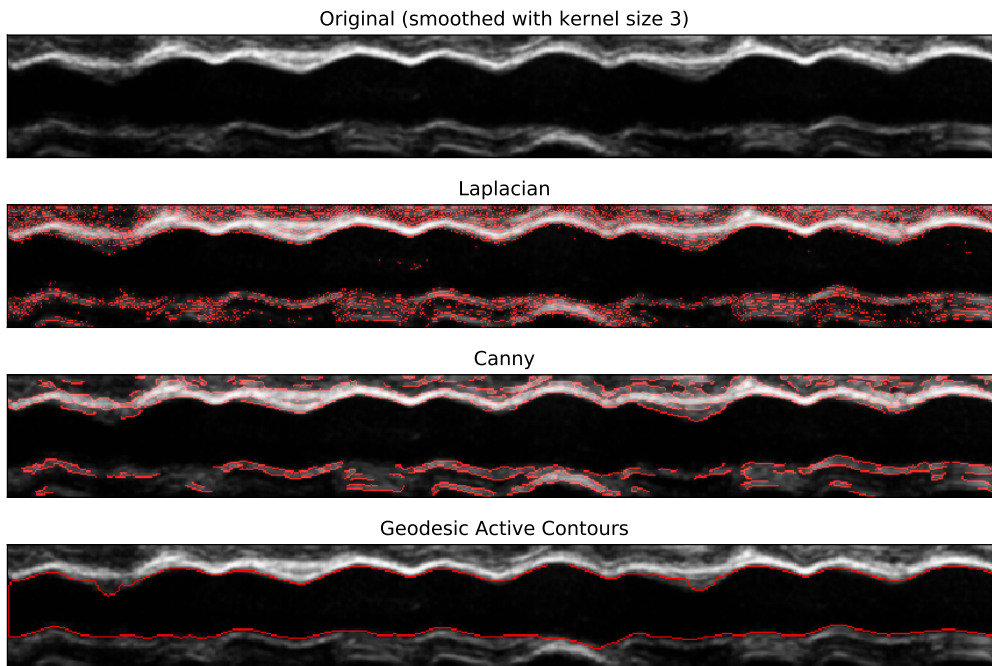
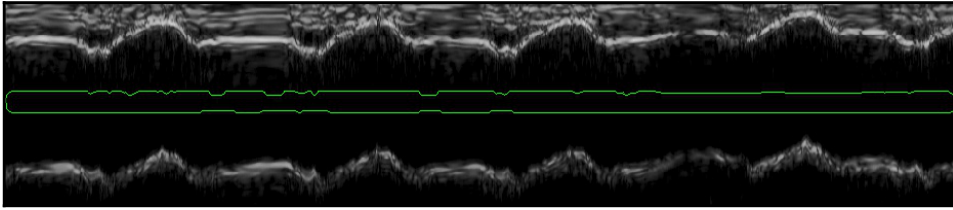


Figure 13: Comparison of a thresholded Laplacian-, Canny- and GAC-edge detection algorithms of a Gaussian blurred image (*kernel size = 3*)

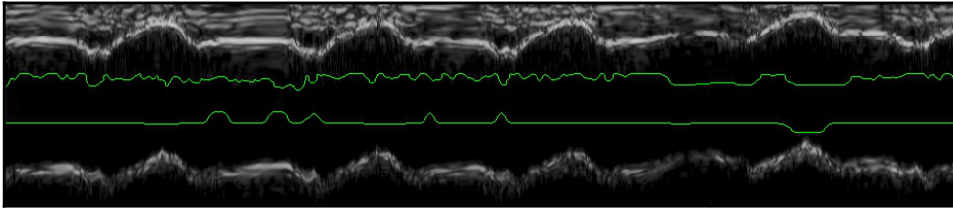
Detecting edges by the usage of a Laplacian image kernel is a fast technique and its resulting edges are independent of the brightness gradient orientations. Unfortunately it is very sensitive to noise, even though it is smoothed by a 5x5 Gaussian kernel and brightness thresholded for the upper 5 bits beforehand, which makes this approach inappropriate for this use case. The Canny edge detector shows good results on clearly detectable time-invariant wall edges, but lacks to find edges with differing brightness. This is comprehensible since the definition of lower and upper thresholds for hysteresis requires a tradeoff between false positive and false negative edges. The usage of the morphological implementation of the GAC algorithm is less sensitive to local brightness discontinuities, since the calculated curve must always be closed. Like already explained in section 1.3.5, a few parameters needed to be estimated empirically which showed overall good results on the given image dataset:

- $\alpha = 1000$ - scaling factor for edge-based image $g(I)$
- $\sigma = 3$ - standard deviation for applied Gaussian kernel in $g(I)$
- $\mu = 1$ - number of iterations for the smoothing step $SI_d \circ IS_d$
- $\theta = 0.31$ - threshold for balloon force effectiveness
- $v = 1$ - (positive) strength for the expanding morphological balloon

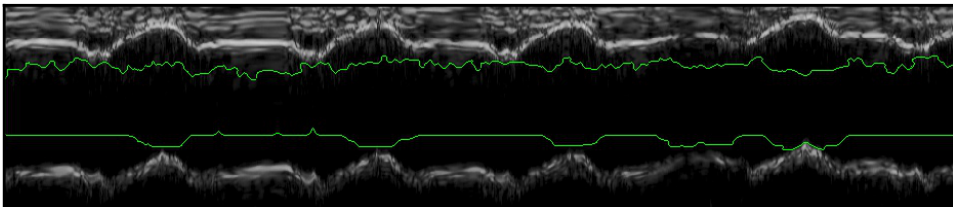
Snake Iteration 1



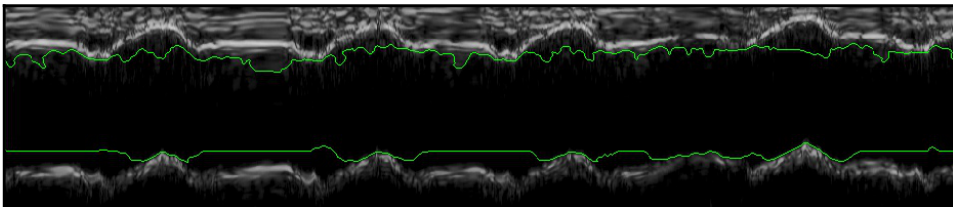
Snake Iteration 10



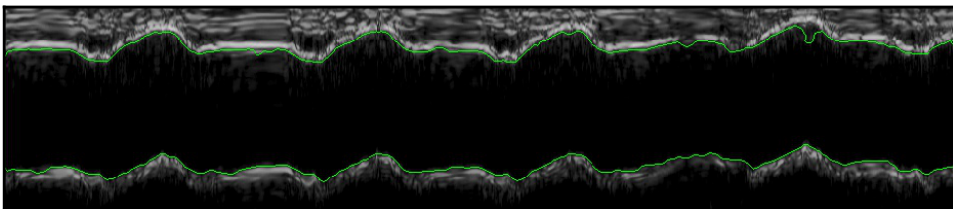
Snake Iteration 20



Snake Iteration 30



Snake Iteration 50



Snake Iteration 70

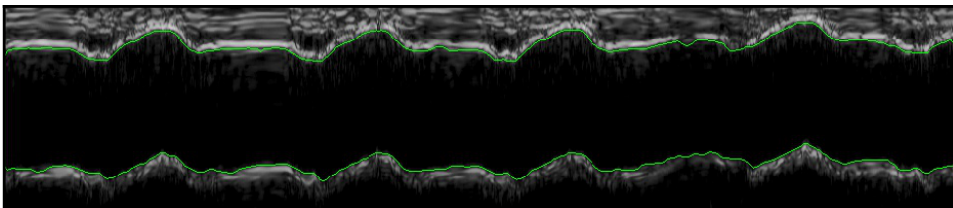


Figure 14: Six example iterations of the active contour approaching in 70 iterations

Based on this parameter set, the “Detect Edges” step first runs this active contour algorithm and then processes the calculated snake to single wall curves of the aortic wall’s leading edges. The snake itself approaches the image’s walls with a maximal number of 350 iteration, in which the iteration’s premature acceptance criteria is fulfilled, if the difference of the snake’s shape over the last 10 iterations did not change significantly. Figure 14 shows an example of the snake, approaching in 70 iterations.

The procedure to split the estimated snake into actual leading edges, can be separated into the following consecutively executed functions:

1. **Run GAC Algorithm** - The presented method in 1.3.5 is implemented in the “Morphsnakes” class and returns the cartesian coordinates of the estimated 1D curves on the aortic region of interest. This step is initialised by an elliptical zero-levelset with $10px$ height and almost the ROI’s width, positioned within the aortic lumen.
2. **Break Snake on Image Borders** - The calculated snake is split at the ROI’s vertical borders and the two most reliable horizontal curves are defined as *wall curves*. Since this morphological GAC approach naturally splits and merges its snakes, this is also the step where small image artefacts between the aortic borders are rejected.
3. **Split Edges at Inflection Points** - In aortic M-mode images, the wall diameter changes are progressive over time. It is physiologically impossible that the wall edges run along the negative time domain. Therefore this step focuses on splitting the determined curves at found inflection points. This function gets a single consecutive edge as input and returns a list of edges, based on the splitted input.
4. **Reject Bulges** - The previously created list is scanned for bulges by first detecting entries whose definition points run against the proceeding time domain in ascending and then descending definition order. These elements are depicted in Figure 15 as the two gray vertically running edges of the already split bulge. If such a combination of bulge elements is found, it will be removed from the list. After this is done, the neighbouring main edges are reconnected by calculating the euclidean distance of their endpoints and using a local endpoint-fixed original active contour algorithm by Kaas et al. [8] (in Figure 15 shown as red edge).

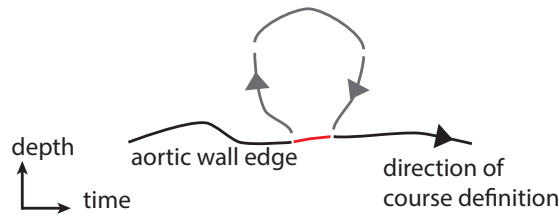


Figure 15: Illustration of estimated aortic wall edge including a already split bulge (gray), the direction of edge definition (gray arrow) and the calculated connecting edge (red)

5. **Clip Edge's Definition Ends** - The list of edges may just contain one element (in best case), but may also include multiple not connected wall curves or even edges from non-aortic objects. To facilitate a later reconnection of single edges that actually focus the same wall in the image, this step clips off endpoints of edges that overlap with ends of other edges in time domain.
6. **Reject Outliers** - In some cases, the GAC algorithm fails to detect the aortic walls and "leaks" out. At this point, the created list of edges is scanned for elements that match certain criterias:
 - edges that contain a straight line, longer than a certain length (calibrated to 10% of the ROI's width)
 - edges that runs against proceeding time domain (not necessarily bulges)
 - edges that have less number of definition points than a defined threshold
 - edges that have an enormous overall gradient in depth domain

If at least one of this criterias matches for a list element, it is rejected.

7. **Correct to Leading Edge** - By default, the zero levelset is initialised within the aortic walls. In ideal case the resulting list of curves represent the true edges of vessel's inner distortion. Since the diameter measurement should be done by using the leading edge method, the transducer near edges must be corrected by the thickness of the aortic wall in depth dimension. This task was accomplished by a method that first convolves the ROI with a horizontal Sobel operator. The result is a smoothed image of horizontal edges, where wall borders tend to have a local minimum (strong negative gradient). The distance from each point of the calculated GAC-edges and its determined horizontal in-line gradient minimum is stored as a list, whose median value is taken as global shift value for the

GAC-edges. Ideally, this value equals the actual thickness of the aortic wall. If an element of the edges list does not have any leading edge, it is being rejected.

8. **Find and Connect Neighbours** - Based on a list of curves that denote the leading edges of the aorta's walls, this method finally connects edges that are neighbours in time dimension, as long as the gap between them does not exceed a certain distance. This is done by finding each two list elements that have a minimal euclidian distance and then again using the endpoint-fixed local active contour algorithm to connect them.

The result of this procedure is one edge element describing the found transducer near wall curve and a second one for the transducer far wall curve. Especially during calibration of the previously explained methods, a quality metric of determined wall curves turned out to be helpful. Established quality parameters were defined as follows:

- Cross correlation of transducer-near and far curves - To accomplish this, the shorter curve is zero padded to equalise in length and the resulting scalar is determined by calculating the correlation of the exactly overlapping curves only, hereinafter referred as cross-correlation coefficient.
- Number of edges in each list
- Percentage of ROI width, covered by the list elements
- Curve's mean distortion deviation value, its standard deviation and the coefficient of variation (COV)
- Average absolute incline of the curve

Since the whole detection procedure is implemented iteratively, the outcome of the quality metric calculation defines wheter another iteration of this detection state will be executed for the whole ROI, for just a part of it or if it is accepted. This inner iteration is especially useful for M-mode images with time variant wall contrast, like exemplified in Figure 16.

In this case, the active contour leaks for the contrast low part on the left side of Figure 16 (red arrow), which is rejected since no leading edge is found for this part. Therefore, the percentage of ROI coverage of the transducer near curve is lower than the calibrated threshold. For this sub-area of the ROI a contrast limited adaptive histogram equalization (CLAHE) algorithm is applied, in order to enhance the contrast



Figure 16: Example of M-mode aortic ROI with time variant transducer near wall contrast including leaking closed-end snake

of transducer near wall encoding pixels, before the inner iteration of the GAC algorithm is executed. The resulting partial wall edge is then connected to the edge of the outer iteration so that most of the ROI's width is covered.

Another criteria for an inner iteration is i.e. a significantly low cross correlation of the transducer-near and far curve. Since the edge between the aortic lumen and the transducer-far wall is typically sharper than the edge to the transducer-near wall, a low cross correlation is used as an indicator for a lacking detection of the transducer-near edge. If the coefficient falls below a defined limit, another iteration is executed in order to determine the leading aortic edge of the transducer near wall with the snake not initialised within the lumen, but at the transducer near outside of the aortic wall.

Whenever the wall edges are accepted, the resulting curves are highlighted on the loaded M-mode image in red. Since the results may need to be manually adapted, the *Config Box* offers the possibility to activate the wall cursor ("Cursor" button), reset the curves to their original "Detect Edges" results ("Undo" button) or confirm the shown curves ("Confirm" button). When the manual cursor is activated, the user has the possibility to click and drag on each of the shown wall curves to reset their depth position to the current mouse cursor location. A click on the "Confirm" button releases the activation of the next state.

3.5.6 Calc Parameters

The last state of the edge detection process finally calculates the heart-cycle based diameter curves. This is done by first finding out which heart cycles are fully covered by a transducer-near and far edge. In order to get a standardised list of heart cycle based diameter curves, each element starts at its foregoing triggerpoint ("red circle") and ends at the determined shortest length of all curve elements. Based on this established list, an average diameter curve is calculated and its found minimal and maximal

values within certain ranges are used for the calculation of defined aortic parameters described in 3.1. The diameter minima is determined within physiologically plausible ranges of $[0, 200]ms$ after the triggerpoint, the maxima within $[100, 500]ms$.

After this calculation, a second tab titled as “Aortic Parameters” will appear within the *Image Box*, which embeds two graphs for single- and averaged-aortic diameter curves on its top. Below those graphs the estimated aortic parameters are printed (“Systolic Diameter”, “Diastolic Diameter”, “Systolic Diameter Increase”, “Distensibility” and “Stiffness Index”), next to a list of named graphs of the single diameter plot. This list offers the possibility to manually reject single curves from the parameter calculation by clicking the red “-” sign next to the named entry. Whenever an entry is removed, the two described plots are updated, as well as the calculated parameters.

At this point, it is possible to launch the edge detection process again from the start (“Load Image”, “Detect Scales”, etc.), in order to merge the curves of the current image with those of another image. To be able to distinguish between curves of multiple images in the “Aortic Parameters” tab, the list of curves also contains a column containing the image name assigned to each curve. It should be noted, that a second run of the edge detection process for the same patient but using a different “Transducer Pos” option will not merge the estimated diameter curves with the results of the first run. It will append entries in the curves list for its according *TransducerPosType* (see Figure 10).

Since the presented aortic parameters may be of clinical interest, the user is able to export the shown edge-highlighted M-mode images, the plots and parameters including personal data of a patient to a PDF report (“Export Results” pushbutton). This report contains separate sections for each “Transducer Pos” (AscAo, DesAo) used to analyse aortic M-mode images of the same patient. An anonymised Exemplary Patient Report of a 20 year old patient suffering Marfan syndrome can be found in the Appendix.

3.6 Software Packaging

One goal of this thesis was to build a simple standalone executable application in order to be able to test and distribute *AortUs*. Since personal computers typically do not have a Python interpreter installed, the implemented software project needed to be

packaged. This was done by a cross-platform Python toolkit called *PyInstaller*, which collects the Python interpreter and all modules required to run the application in a temporal pythonic environment. Finally this is all embedded by the PyInstaller's bootloader, called from a command window whenever the user runs the executable. Since PyInstaller was not capable to find all required modules automatically for a Windows OS, a specification file containing explicit paths and so-called *hidden import* modules needed to be defined in order to successfully package the application. The executables were distributed for 32 but also for 64 bit systems.

3.7 Software Documentation

In general, the documentation of software is not just essential to bear in mind the developed classes and methods' intention, but it is also part of the technical documentation of any medical software product. Since it is a time-consuming task and prone to be forgotten for "quick and dirty" fixes in the code, the related documentation for *AortUs* was not created by hand.

Instead a semi-automatic approach was used to parse the information embedded in the comment headers of each created class, method and member variable. Among others, the used open source tool called *Sphinx* offers the integration of the markup language restructured text (reST), which was used to describe all implemented modules for *AortUs* in their comment headers. These parsed modules are then recombined to the output format, specified by the *Sphinx* related "config.py" configuration file but also by module related layout "*.rst" files. In case of the Software Documentation in the Appendix, the output format was chosen to create a "*.tex" file for the open source text processing software "pdfTeX". The whole procedure of the software documentation build is controlled by a shell script and can be obtained by Figure 17.

3.8 Code Reviews

Since *AortUs* was implemented by one developer, code reviews were part of the daily development process but not explicitly planned. Although, frequent functional review meetings or at least status e-mails were sent in monthly intervals between CB and BF to ensure the integration of functional demands on *AortUs*.

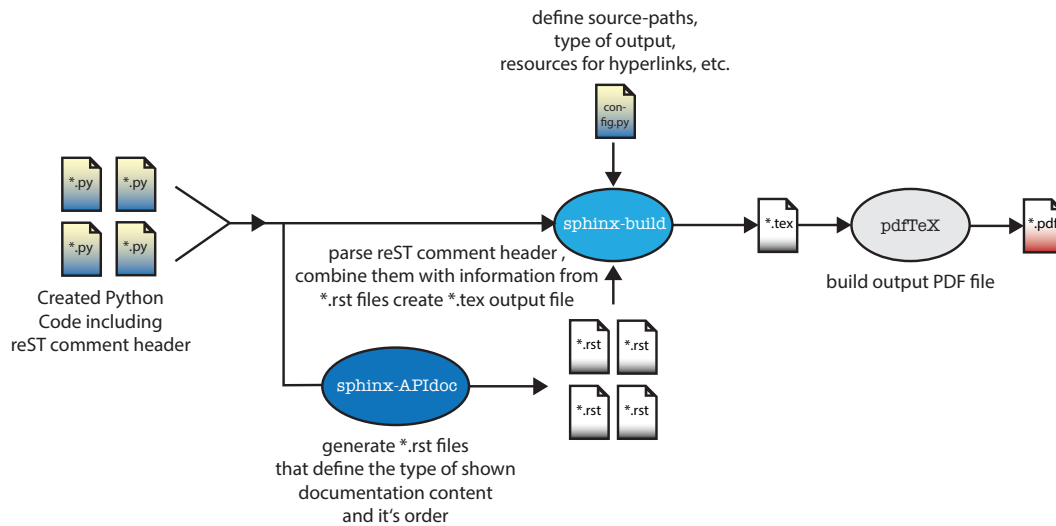


Figure 17: Process of software documentation building for *AortUs*, controlled by a shell script

3.9 Usability Testing

One objective of this project was to integratively consider a high level of usability of the developed application software. To verify the level of usability but also to test the software under realistic conditions, a usability test was performed in collaboration with the clinical department of paediatric cardiology at the LKH-Univ. Klinikum Graz³. This test included four steps:

1. adaption of *AortUs* to the M-mode image standard used at this department
2. informational and applicational briefing of collaborating users
3. testing phase
4. completion of usability questionnaire

Since the recorded M-mode images of the ultrasound (US) device at this department were different to the images provided for this thesis, software changes had to be made beforehand. These changes focus the differing position of detected axes, as well as of the position of the recorded ECG.

After a twenty minutes briefing and a testing phase of approximately one and a half weeks, the completed questionnaires were collected. The usability questionnaire used

³Klinische Abteilung für Pädiatrische Kardiologie, Universitätsklinik für Kinder- und Jugendheilkunde, LKH-Univ. Klinikum Graz

was based on the well known standardised System Usability Scale (SUS) by J. Brooke (1996) [22], slightly modified and translated to German. The used “Feedbackbogen AortUs” can be found in the Appendix. The actual SUS score is calculated by rating each question from 0 to 4, in which each question has a score of 1 for *Strongly disagree* (*Stimme überhaupt nicht zu* in German) to 5 for *Strongly agree* (*Stimme völlig zu* in German). Questions 1,3,5,7,9 contribute by $score - 1$, questions 2,4,6,8,10 by $5 - score$. The resulting scores are summed and multiplied by 2.5 to reach a value within 0 for worst imaginable usability and 100 for best imaginable usability. It has been shown, that this SUS score is highly correlated with an overall question of user-friendliness of the system [23], using the adjectives *Worst Imaginable* (SUS mean of ≈ 13), *Awful* (SUS mean of ≈ 20), *Poor* (SUS mean of ≈ 36), *OK* (SUS mean of ≈ 51), *Good* (SUS mean of 71), *Excellent* (SUS mean of 86) and *Best Imaginable* (SUS mean of 91). The results of this test is listed in section 4.2.3.

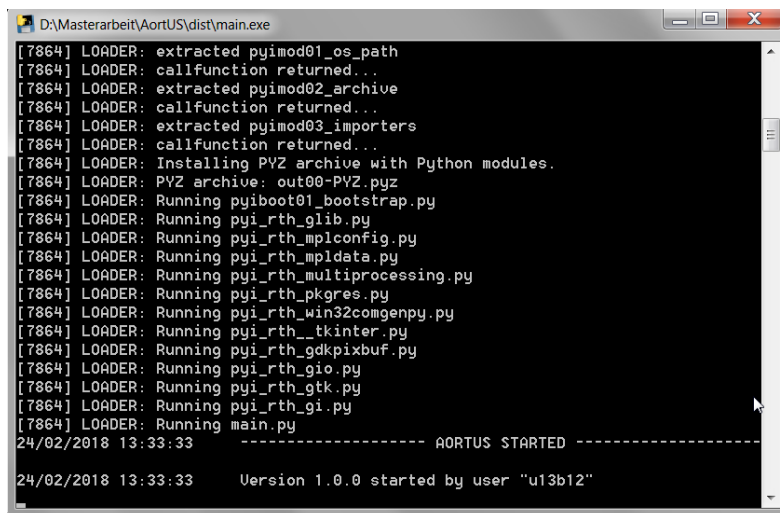
4 Results

The main result of this project is obviously the software itself, packed to be used as a standalone executable for 32bit and 64bit Windows 7 (Microsoft Windows, Redmond, USA) or higher OS. In order to make the previously described approaches more comprehensible, the upcoming section will focus on the usage of the established application and its documentation. In addition, a comparison of possible risk classes will be outlined and the results from usability testing will be presented.

4.1 AortUs Usage

4.1.1 Full Usage Example

The following example is abstracted from the User Manual in the Appendix and will give a usage example for two M-mode images of the AscAo of the same person, processed by *AortUs*. The User Manual also includes more detailed information about an (exemplary) usage intention, the intended user, image rejection criterias and additional notes and warnings in shape of boxes.



```
D:\Masterarbeit\AortUS\dist\main.exe
[7864] LOADER: extracted pyimod01_os_path
[7864] LOADER: callfunction returned..
[7864] LOADER: extracted pyimod02_archive
[7864] LOADER: callfunction returned..
[7864] LOADER: extracted pyimod03_importers
[7864] LOADER: callfunction returned..
[7864] LOADER: Installing PVZ archive with Python modules.
[7864] LOADER: PVZ archive: out00-PVZ.pyz
[7864] LOADER: Running pyiboot01_bootstrap.py
[7864] LOADER: Running pyi_rth_glib.py
[7864] LOADER: Running pyi_rth_mplconfig.py
[7864] LOADER: Running pyi_rth_mpldata.py
[7864] LOADER: Running pyi_rth_multiprocessing.py
[7864] LOADER: Running pyi_rth_pkgres.py
[7864] LOADER: Running pyi_rth_win32comgenpy.py
[7864] LOADER: Running pyi_rth_tkinter.py
[7864] LOADER: Running pyi_rth_gdkpixbuf.py
[7864] LOADER: Running pyi_rth_gio.py
[7864] LOADER: Running pyi_rth_gtk.py
[7864] LOADER: Running pyi_rth_gi.py
[7864] LOADER: Running main.py
24/02/2018 13:33:33 ----- AORTUS STARTED -----
24/02/2018 13:33:33 Version 1.0.0 started by user "u13b12"
```

Figure 18: Bootloader command window opened, when *AortUs* executable is started

AortUs is started with a double click on the executable. Soon the bootloader command window will appear, listing “LOADER” entries of the Python bootloader shown in Figure 18. After all modules are loaded, the software can be operated. To ensure privacy, all input parameters as well as the used images were anonymised.

1. When *AortUs* is started, the main window shown in Figure 19 will appear. On success, the *Log Box* will output the current version as well as the username.

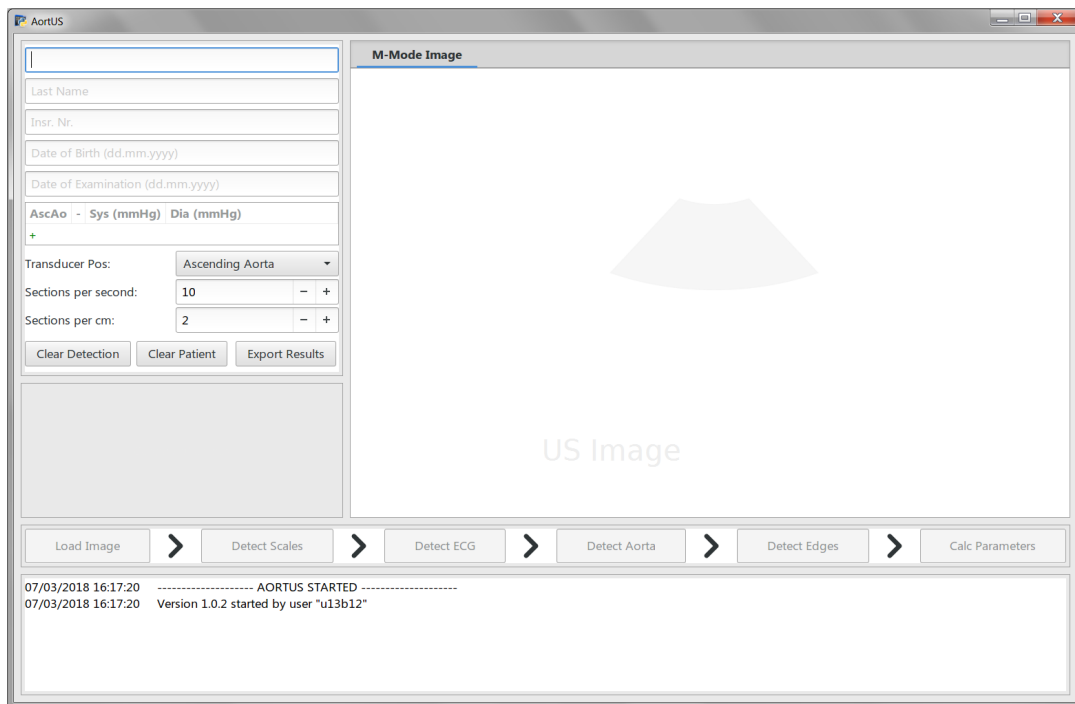


Figure 19: Default appearance of *AortUs* after startup

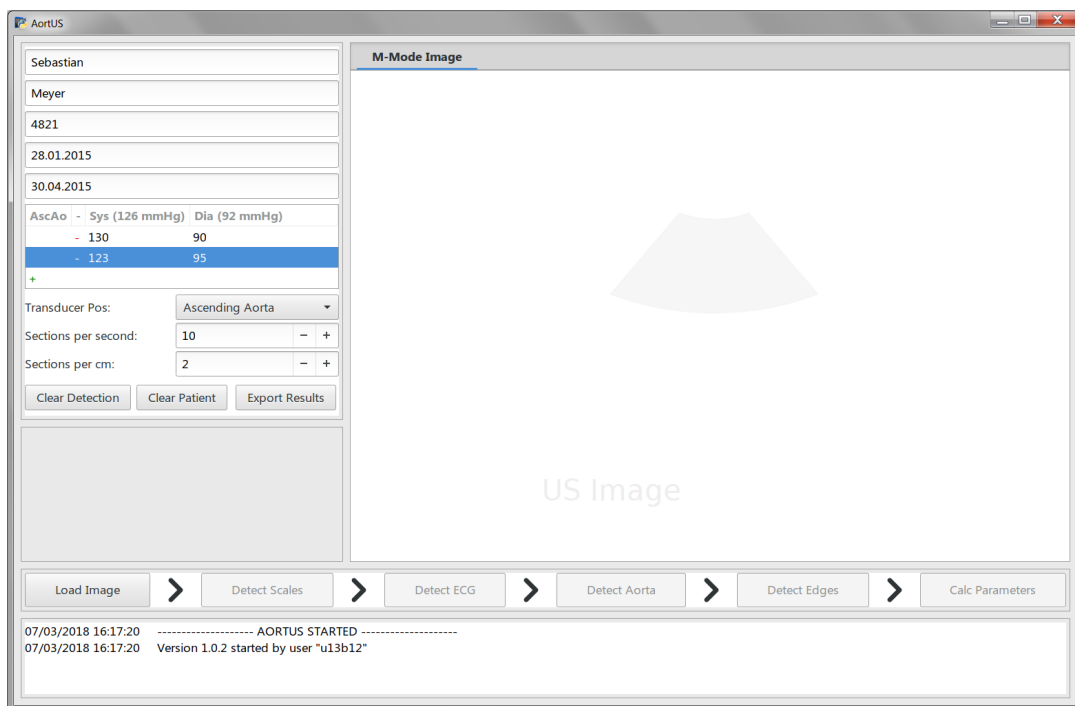


Figure 20: Entered patient data

2. The required first name, last name, insurance number of the patient must be en-

- tered, as well as the date of birth and of examination in format <DD> . <MM> . <YYYY>, like shown in Figure 20. In addition, either the element *Ascending Aorta* or *Descending Aorta* in the *Transducer Pos* list must be adapted to the type of image.
3. When all input fields are set, the first image can be loaded by clicking the “Load Image” button in the *State Box*. A dialogue presented in Figure 21 will appear and the first image to load can be selected and confirmed by clicking “Open”.

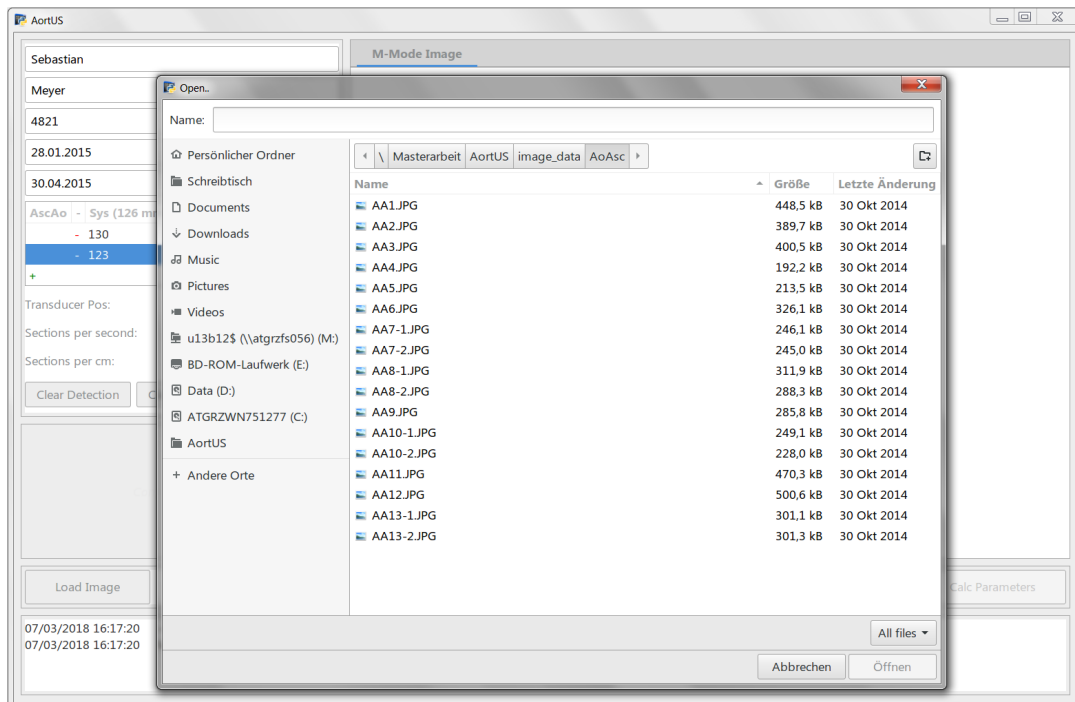


Figure 21: *AortUs* load image dialogue

On success, the selected image will appear within the *Image Box* like shown in Figure 22 and the calibration of “Sections per second” and “Sections per cm” must be adapted if necessary. For the image shown in Figure 22, the settings need to be corrected to 5 vertical sections encoding one second and 1 horizontal section encoding one centimeter.

4. The next step for *AortUs* is to determine the number of pixels for 1cm and 1sec, respectively. This action can be started clicking “Detect Scales”.

On success, the found main axes within the image will be highlighted from the top left of the M-mode’s area in green and the found pixel resolution will be printed within the *Log Box*, shown by Figure 23.

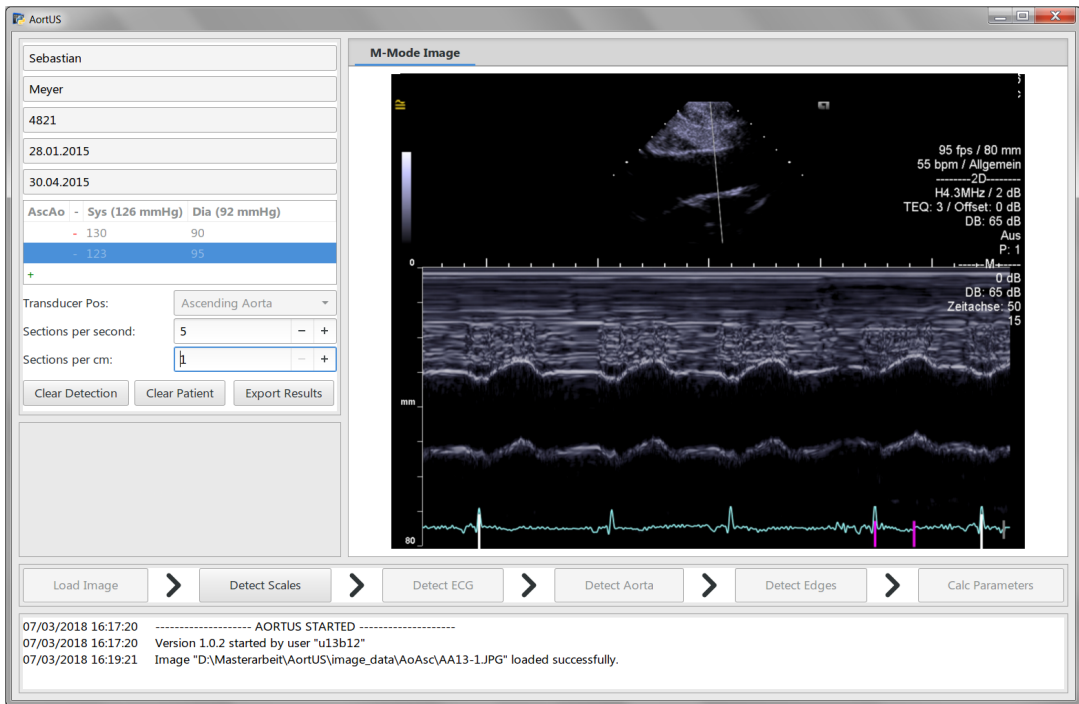


Figure 22: *AortUS* showing the loaded image including adapted vertical and horizontal section settings

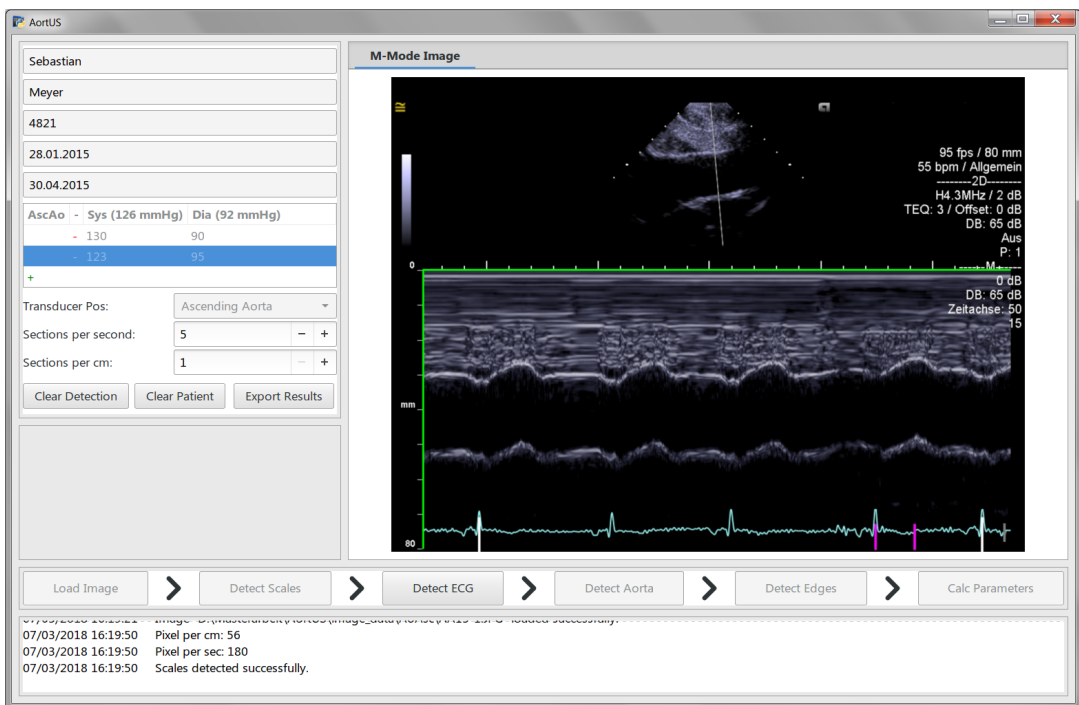


Figure 23: *AortUS* showing the automatically detected scales

In some cases the main axes cannot be detected automatically, i.e. if the axes and their neighbouring ROI content have similar brightness values. For this particular

case, the user can define the region of interest as well as the pixels used for 1sec and 1cm manually like described in section 4.1.2.

5. When clicking “Detect ECG”, *AortUs* will try to find a coloured ECG tracing including striking R-peaks in the image.

The updated main window appearance can be obtained by Figure 24, including the found ECG tracing highlighted in green, as well as the R-peaks indicated by red circles. The positions of these circles, also referred to as triggerpoints, define the start- and endpoints of each diameter-period used for averaging.

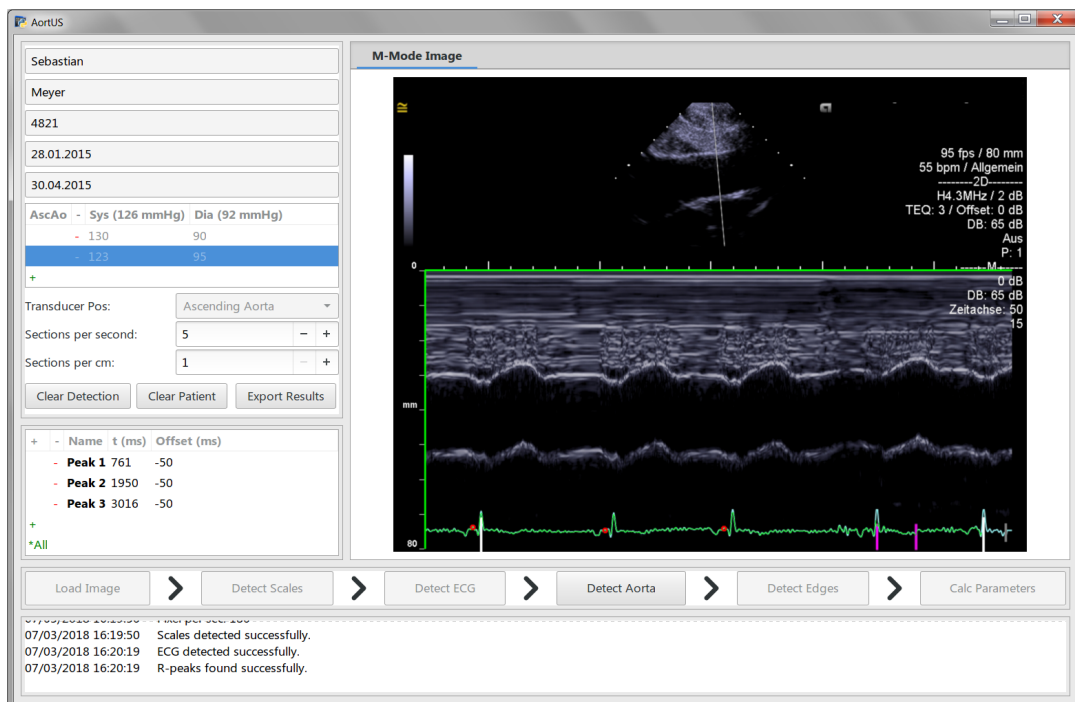


Figure 24: *AortUs* showing the “Detect ECG” state including original locations of found R-peaks

In this case, *AortUs* is not able to find the image related peaks precisely, like shown in Figure 24. Therefore, the user has two options for manipulation:

- click-and-drag on the red circles to modify their position along the found ECG tracing
- remove, add or adjust positions of triggerpoints list items within the *Config Box*.

The list is embedded within a scrollable window and itemises found triggerpoints including their position in time domain, as well as an adjustable offset to this position. This offset is $-50ms$ by default and can be changed by simply clicking

into the field of a triggerpoint item. In addition, the list offers the following functionality:

- + ... add a triggerpoint
- ... remove a triggerpoint
- *All ... set \pm offset of all triggerpoints to the offset of the selected one in *ms*

For the image shown, Peak 4 and Peak 5 needed to be inserted (“+”) and dragged $\approx 50ms$ prior to their missed R-peaks, comprehensible by Figure 25.

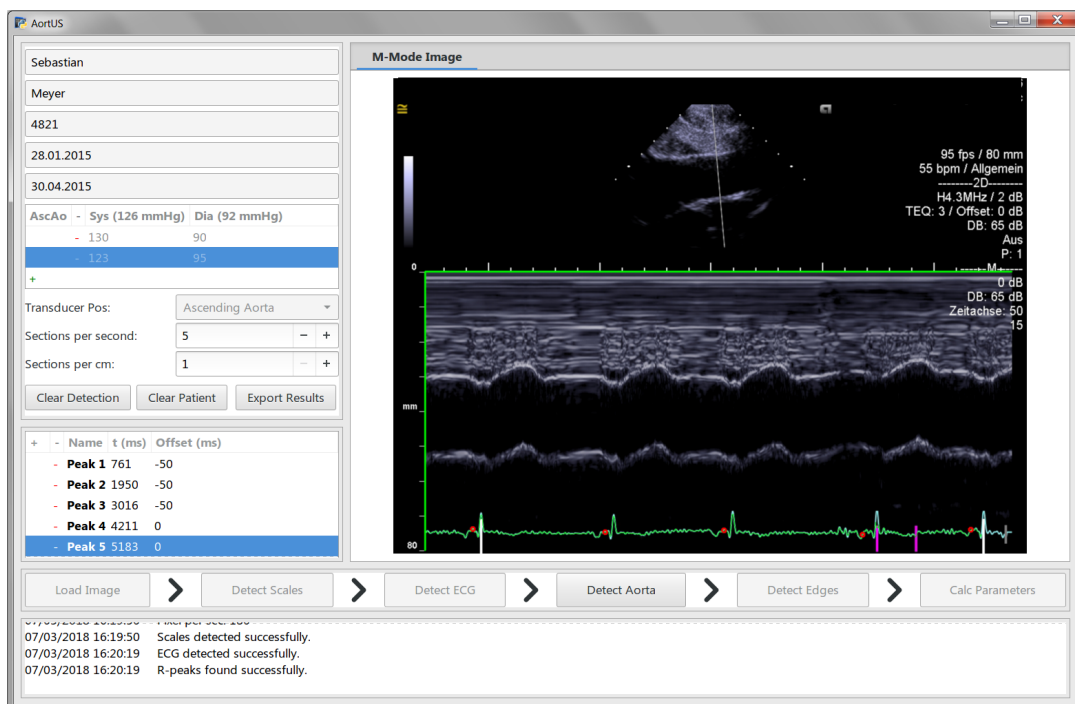


Figure 25: *AortUs* showing the adjusted triggerpoints on found ECG

6. When all triggerpoints are defined correctly, the user should move on to the “Detect Aorta” state. This function aims to scale down the ROI vertically from the whole M-mode region to the minimal image area arrogated by recorded aortic walls. This is symbolised by limiting the ROI to the area between two yellow lines, shown in Figure 26. For images with high wall contrast, the automatically suggested positions of these boundaries closely approximate the outer borders of aortic walls. Although, it is also possible to adjust the positions of these boundary lines close to the outermost points of the aortic wall image, again by a mouse click-and-drag action. In case of the image shown in Figure 26, the detected borders suit the whole aortic representation and can therefore be accepted by

clicking the “Confirm” button inside the *Config Box*.

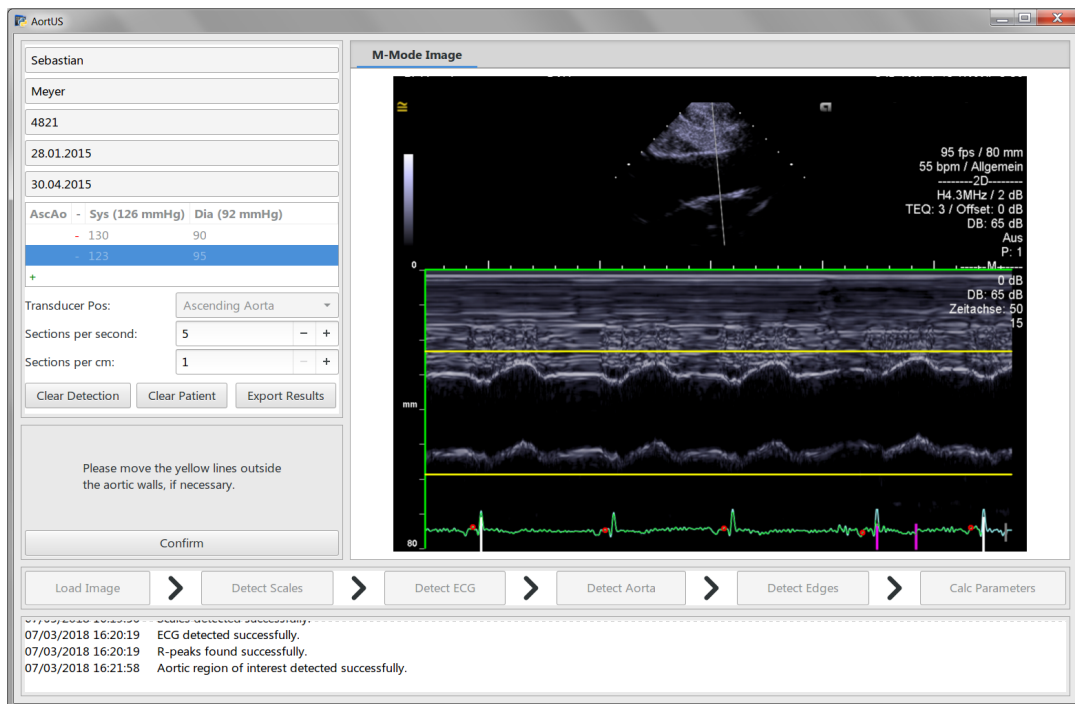


Figure 26: *AortUs* showing the “Detect Aorta” state to scale down ROI vertically to aortic walls

7. The main edge detecting function is the most performance demanding step during a standard procedure of *AortUs*. When clicking “Detect Edges”, it might take a few seconds to determine the leading edges, shown in Figure 27.

Since the internal edge detection algorithm cannot handle all sudden brightness interruptions of the aortic walls, the user is asked to mend these outliers manually.

For this interaction, three mode buttons are shown within the *Config Box*:

- ... Activate manual cursor mode
- ... Reset edges to initial results from “Detect Edges”
- ... Proceed to “Calc Parameters”

When the manual cursor mode is activated, the user can click-and-drag on each of the red shown circles. The positions of the cursor’s nearest edgepoints will be reseted to the positions of cursor’s pathway, without splitting the edge. For the edges shown in Figure 27, the region around the fourth triggerpoint of the transducer-far edge is adjusted by first clicking “Cursor” and then correcting the local transducer-far edge around the fourth triggerpoint from left to right. The resulting image can be obtained by Figure 28. After the curve is corrected, the user may proceed to the final “Calc Parameters” state by clicking the “Confirm

Walls" button.

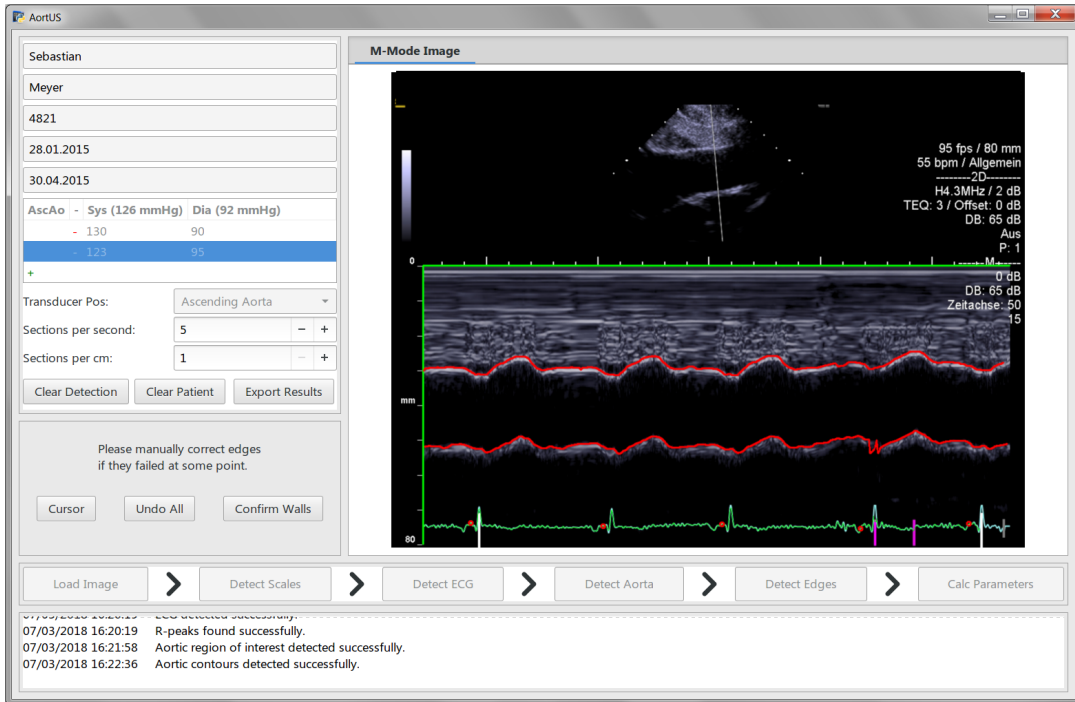


Figure 27: *AortUs* showing the automatically detected aortic walls in “Detect Edges” state

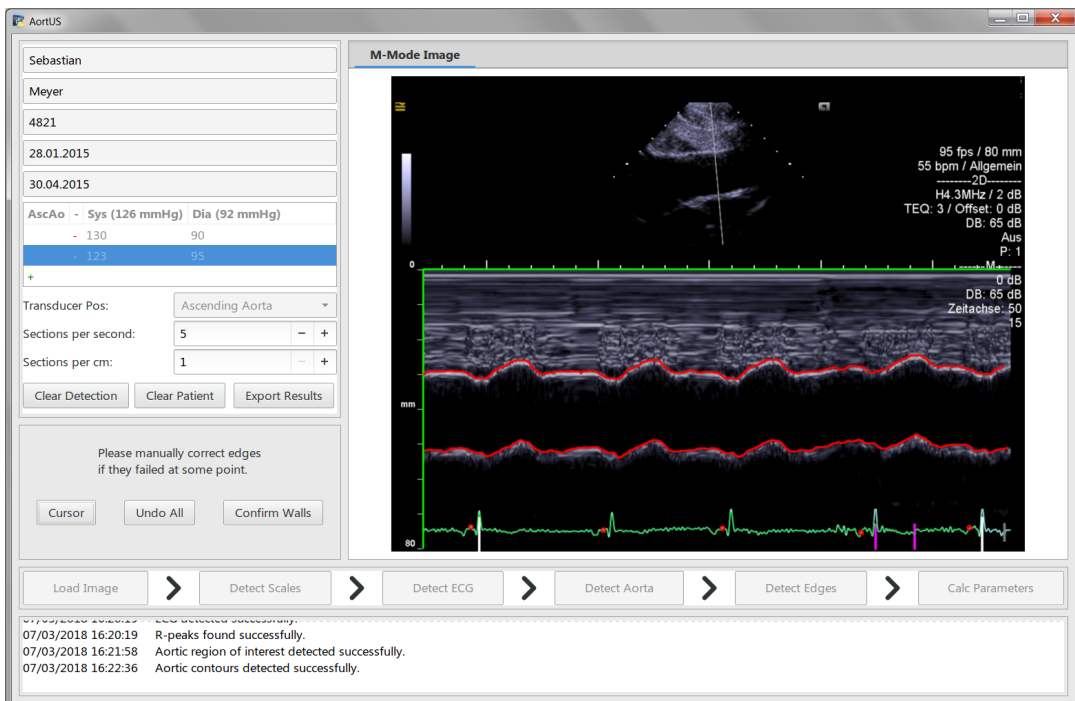


Figure 28: *AortUs* showing the manually adjusted aortic wall edges in “Detect Edges” state

8. Finally *AortUs* will create a second tab within the *Image Box* named *Aortic Parameters*, containing two types of plots, the calculated parameters as well as a

list of curves that contribute to the shown plots. Each curve shown is extracted from the double-edges of previous step only, if both edges fully cover each whole heart cycle.

The left plot named *Single Aortic Diameter Courses* shows all diameter curves that could be extracted, named from course 0 to course $\langle n-1 \rangle$ for edges from “left to right” where n denotes the number of diameter curves. The right plot named *Average Aortic Diameter Course* shows the average diameter at each timepoint, in which the determined *Diast. Diameter* as well as the *Syst. Diameter* are marked by a red dot.

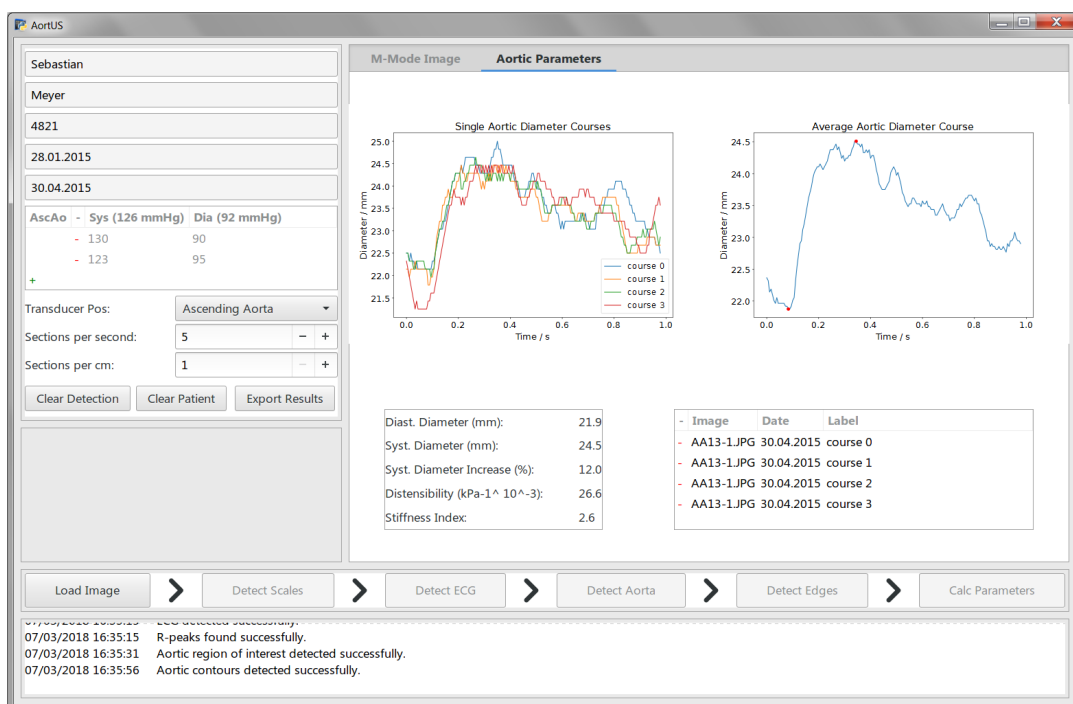


Figure 29: *AortUs* showing the determined curves including the estimated parameters in “Calc Parameters” state

Next to the calculated parameters, a list of the curves contributing to the calculation is shown, including the name of the image(s) used and the date of examination. Similar to the functionality of the BP list, the “-” button can be used to remove a certain curve from the calculation, which might be useful to remove outliers. Whenever this is done, all plots and parameters are updated to the current list of curves.

The *Diast. Diameter* and *Syst. Diameter* are determined by finding the local Minima within $[0, 250]ms$ and the Maxima within $[200, 500]ms$ after the averaged origin of

diameter curves.

- The attentive user will notice, that the “Load Image” button can be used again at this “Calc Parameter” state. For a second M-mode image, steps 3 to 8 can be repeated. In this example, the image AA13-2.JPG is loaded and the combined results are shown in Figure 30.

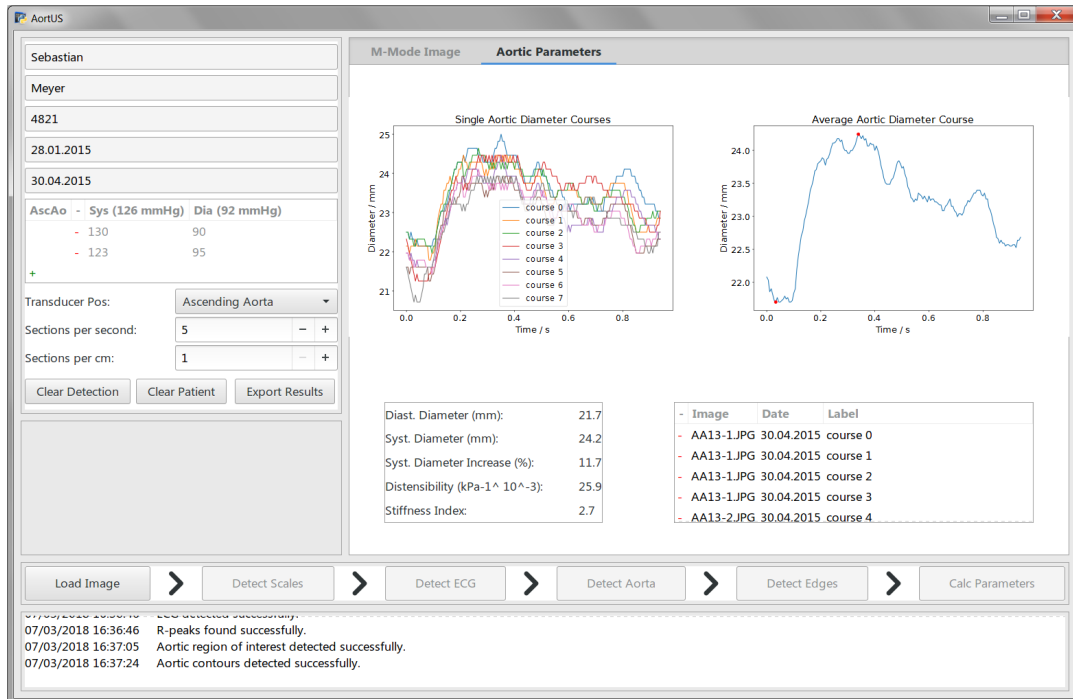


Figure 30: AortUs showing the combined estimated parameter of image AA13-1.JPG and AA13-2.JPG

- In order to append the extracted information for instance to a patient record, the user may want to export or print the shown results. At this point, the shown plots as well as the estimated parameters can be exported to a PDF by clicking the *Export Results* button. A dialogue shown in Figure 31 will open and ask for the user’s credentials. By default, the username of the OS’s user will be proposed. When submitted, a second dialogue will open to select a location for the created PDF, shown in Figure 32. The content of the PDF is self explanatory and includes all used M-mode images including highlighted edge curves, as well as patient related data. The content of this final PDF can be obtained by Figure 33.

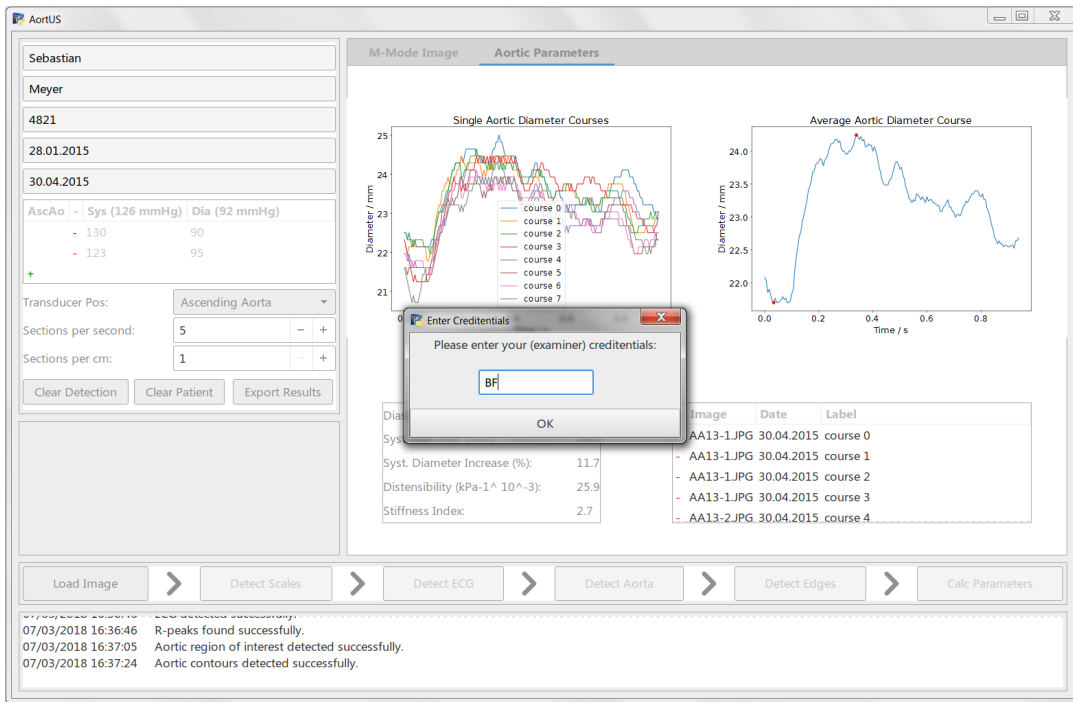


Figure 31: AortUs showing the dialogue to enter the examiner's username

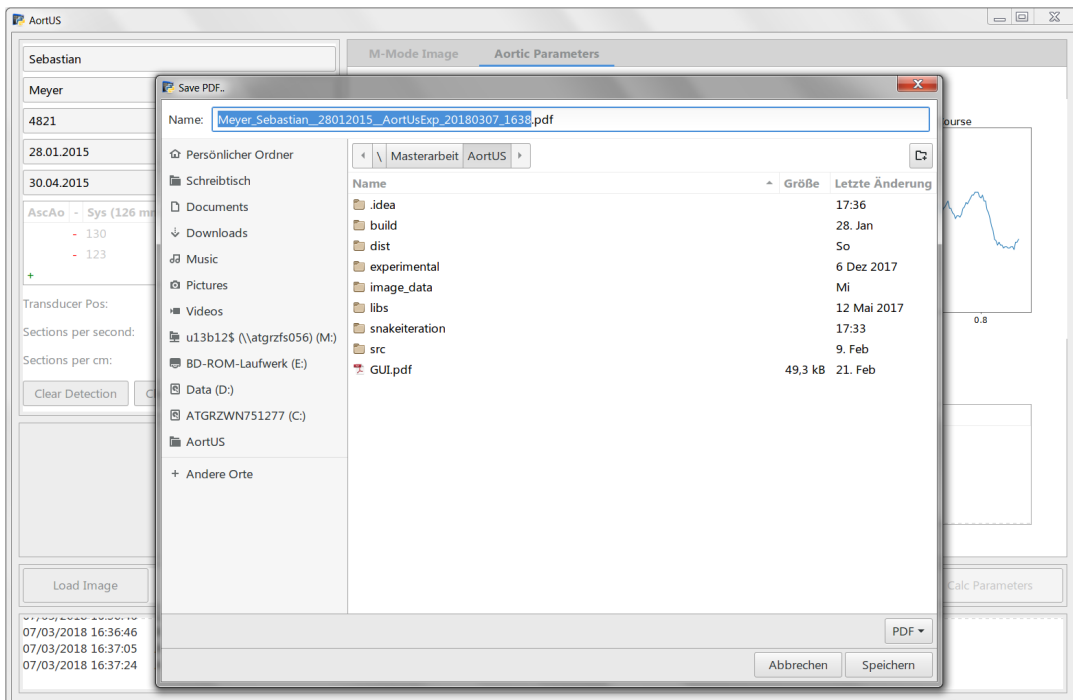


Figure 32: AortUs showing the dialogue to enter the location of the PDF export file

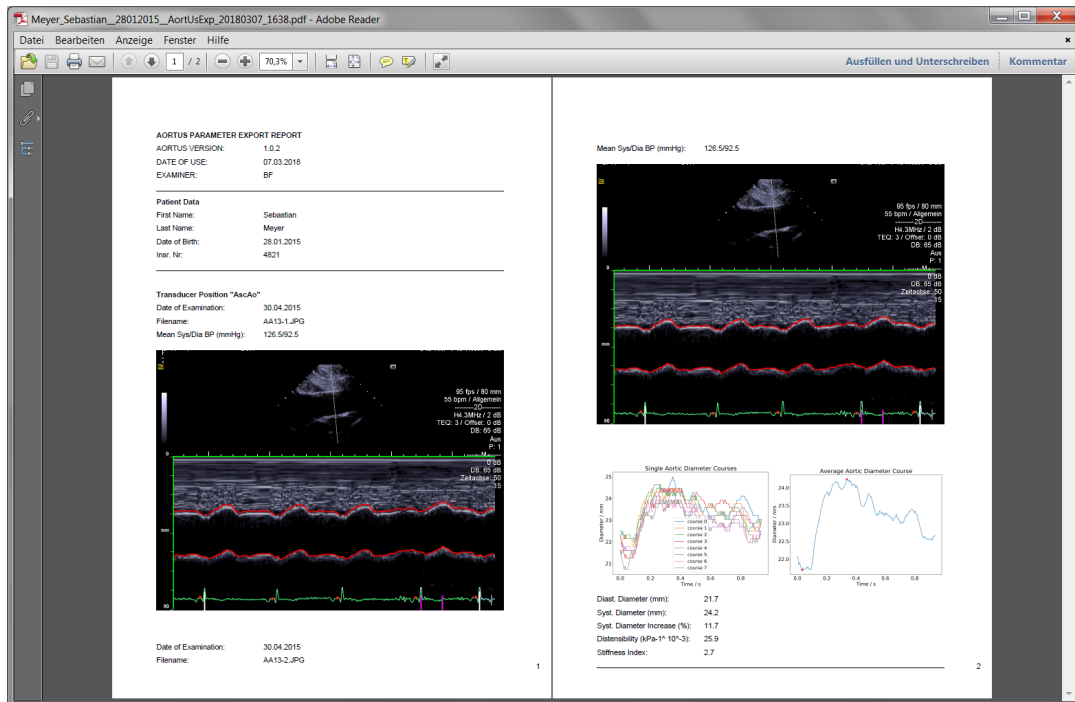


Figure 33: PDF export of the two analysed M-mode images

4.1.2 Manual Scale Detection Usage Example

In some cases, however, the automatic scale detection fails when clicking “Detect Scales”. A *Message Box* will appear, and the Manual Scale Detection is activated automatically. This process will guide the user through a manual definition of the M-mode ROI, the scaling for 1sec and for 1cm respectively.

The following example should help to understand the process of Manual Scale Detection:

1. When clicking “Detect Scales”, a notification will appear to inform the user about the launch of manual mode, like shown in Figure 34. The first step defines the actual M-mode image ROI. To achieve this, it is recommended to click at the very left top of the M-mode image content and drag down the appearing green rectangle to the bottommost right end of the image, like shown in Figure 35. When the left mouse button is released, the defined ROI will be denoted by highlighted green vertical- and horizontal axes, shown in Figure 36. This definition of the M-mode ROI can be repeated until the user is satisfied with the axes positions. If so, a click on the “Next” button within the *Config Box* switches to the time-definition step.

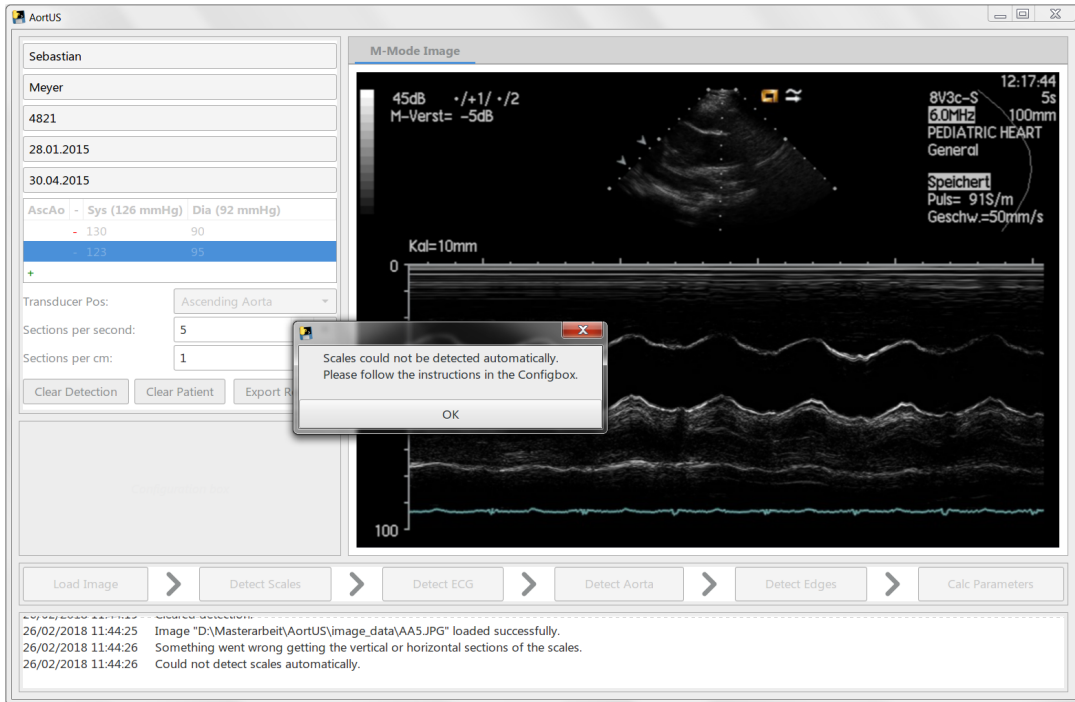


Figure 34: AortUS showing the start of manual scale detection after user clicked “Detect Scales”



Figure 35: AortUS showing the manual definition of M-mode ROI by click-and-drag

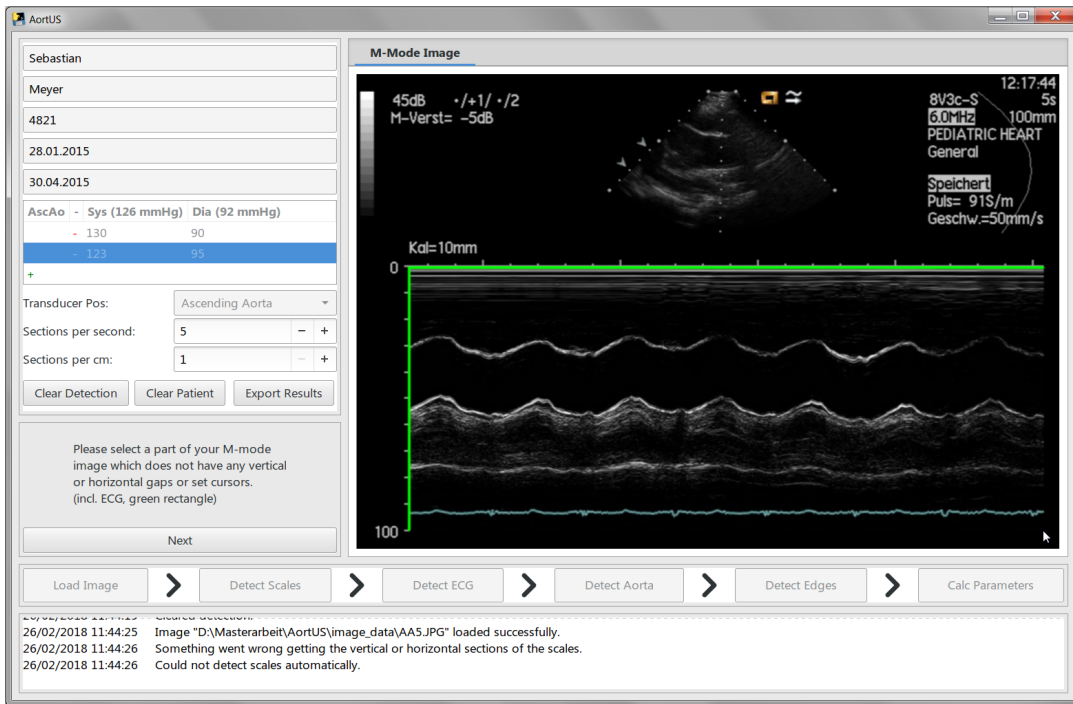


Figure 36: *AortUs* showing the defined M-mode ROI as green axes

2. To define the pixels encoding 1sec, the user is asked to define a horizontal line with the length of 1sec by (again) click-and-drag. It is recommended to define this line based on the visible time axis, like shown in Figure 37.

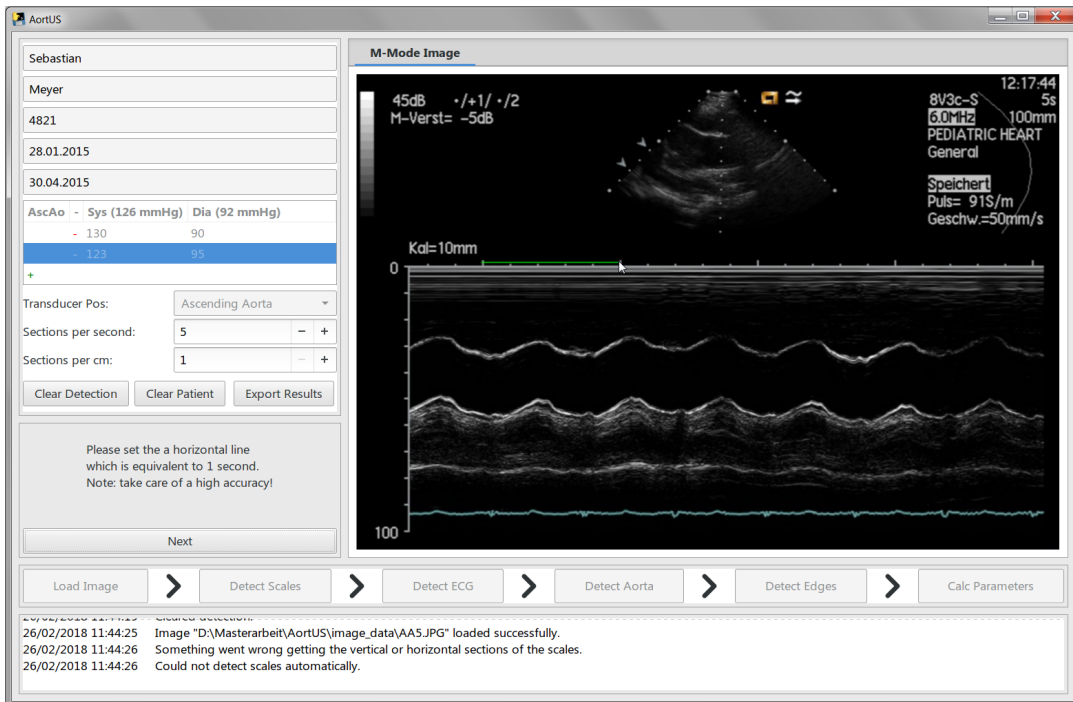


Figure 37: Manual definition of 1sec

Similar to the definition of the M-mode ROI, this step can be repeated and confirmed by clicking “Next”.

3. Equal to the time-definition step, the user is asked to define a vertical line with length of 1cm. The resulting definition is shown in Figure 38.

When this is done (“Next”), the user can finish the Manual Scale Detection process and *AortUs* enables to trigger the next state (“Detect ECG”).



Figure 38: Manual definition of 1cm

4.1.3 Clinical Patient Reports

Since no additional information about the patient’s age, sex and blood pressure values related to examinations of the provided test image set was given, no reliable results were estimated on this dataset.

Instead, three TTE patient recordings were performed at the LKH-Univ. Klinikum Graz by Univ. Prof. Dr. Daniela Baumgartner. The images were processed by *AortUs* and its resulting patient reports in the Appendix include recordings of a 21 year old female patient suffering Marfan syndrome type 1, a 4 year old healthy female patient and a 48 year old male healthy patient.

4.2 Regulatory Aspects of AortUs

4.2.1 Traceability and Identification of SOUPs

Regarding software development, the term “Traceability” describes the verification and documentation of the software architecture being compliant to its requirements. Due to extensiveness, this is not explicitly outlined, but can be related by comparing the functional and non-functional requirements of the SRS with the described architecture and processes in sections 3.4 and 3.5.

Besides this, the EN 62304:2006+A1:2015 requires the identification of all software components used, that are not developed and maintained by this norm’s requirements. The modules used for the development of *AortUs* are Open-source only. Therefore it is obvious, that this project mainly includes SOUP components.

4.2.2 Risk Classification

To roll out medical software on the market, the manufacturer has to approve the conformity of the product by passing a conformity assessment procedure. The path and, as a consequence, the level of regulatory requirements to be fulfilled depend on the risk class chosen by the manufacturer. Since *AortUs* is a standalone application treated as general medical product, one can choose a risk class by using rule eleven of the MDR 2017/745 on the defined intended use. Table 2 compares the selection of risk classes (columns), according to different levels of intended use specifications [14] (rows). It should be noted, that classes IIb and III only differ in the impact of harm resulting from diagnostic or therapeutic decisions for this particular case. Since the therapeutic decisions have a vast range including long-term β -blocker medication up acute surgery for dissecting aortic walls, no more detailed risk analysis was made.

4.2.3 System Usability Scale

The following section will summarise the outcome of the software usability testing, performed at the clinical department of paediatric cardiology at the LKH-Univ. Klinikum Graz. Since only three users participated this test, Table 3 shows their single scores from 1 for *strongly disagree* to 5 for *strongly agree* for used questions. Table 4 shows the scored values of the question related scores from Table 3.

Table 2: Possible classification of risk classes of *AortUs* versus their intended use definition

	I	IIa	IIb	III
Working principle	Standalone application to determine local Distensibility ($kPa^{-1} \times 10^{-3}$), Stiffness-Index and Systolic Diameter Increase (%) of the aorta by combining the average values of entered systolic- and diastolic bloodpressure (mmHg) with average values of systolic minimal- and end-diastolic maximal aortic diameters (mm), based on processed TTE M-mode images of the ascending or descending aorta.			
Medical Indication	The software establishes local physiological parameters of Distensibility, Stiffness-Index and Systolic Diameter Increase of the the AscAo or DescAo.	The software establishes aortic parameters to support cardiologists in questions of qualitative aortic elasticity changes.	The software establishes aortic mechanical parameters that may indicate pathological structural changes of the aortic wall.	
Intended group of patients	All patients whose ascending or descending aorta can be examined with TTE in M-mode.	All patients without previous personal or hereditary cardiovascular disease but with suspect of impaired bioelasticity of the aortic wall, whose ascending or descending aorta can be examined with TTE in M-mode.	All patients with diagnosed cardiovascular disease or with suspicion of impaired arterial elasticity, whose ascending or descending aorta can be examined with TTE in M-mode.	
Intended part of body	The examination focusses on the AscAo and DescAo, which may be related to general elasticity changes of the arterial circulation.			
Intended usage profile	The user groups may be differentiated into three types: <ul style="list-style-type: none"> • Cardiologist (being the examiner, has experience in TTE imaging techniques, fluent in English and German, has experience in handling application software) • Graduated Nurse (has experience in TTE, has an instruction to run the software by a cardiologist, fluent in English and German, experience in handling application software) • Academic User (experience in TTE, based on aortic M-mode images recorded by a cardiologist, fluent in English and German, experience in handling application software) 			
Intended usage environment	The software is intended to be used in daily clinical cardiology practice. The resulting parameters and curves may be attached to a report database for i.e. statistical evaluation purposes.	The software is intended to be used, whenever the medical specialist wants to make a diagnosis of recorded aortic M-mode images.	The software is intended to be used, whenever the medical specialist wants to monitor a cardiovascular therapy or makes a diagnosis of recorded aortic M-mode images.	
	The software is executed on a Windows 7 OS or higher, under usual office working conditions.	The software is executed on a Windows 7 OS or higher, under usual office working conditions.	The software is executed on a Windows 7 OS or higher, under usual office working conditions.	
Usage Exclusion	All M-mode images that do not fulfill at least one of the following conditions: <ul style="list-style-type: none"> • 24bit RGB image, embedding a grayscale M-mode image with clearly visible transducer- near and far aortic walls • the aorta must not be located near the M-mode region of interest's boundaries • coloured ECG tracing recorded synchronous to the wall distortions, not covering wall image content • each image covers at least two cardiac cycles • the LoS representing the M-mode recording must be exactly perpendicular to the aorta • the LoS should represent the position of maximal systolic diameter change • the image must not contain any vertical or horizontal cursors from the TTE-device's measurement tools • the image must not contain any other ROI breaking content, i.e. black areas within the M-mode ROI evoked by changes of TTE-device settings • the axes of the M-mode area must be defined by a scaling of one of the following shapes <ul style="list-style-type: none"> – a purely horizontal line including orthogonal intersection lines on the area's top border as well as a vertical line including orthogonal intersection lines on its left border – no vertical or horizontal axes, only intersection lines on the area's right border (vertical) as well as on the area's bottom border (horizontal) • the image exported from the echocardiography device software must be in format "JPG", "PNG", "TIF" or "BMP" 			

Table 3: Single scores of SUS question scores of each user (in German)

Nr	Questions	User 1	User 2	User 3
1	Ich denke, dass ich die Software gerne häufig benutzen würde.	5	4	3
2	Ich fand die Software unnötig komplex.	1	1	2
3	Ich fand, die Software war einfach zu benutzen.	5	5	3
4	Ich glaube, ich würde die Hilfe einer technisch versierten Person benötigen, um die Software benutzen zu können.	1	1	1
5	Ich fand, die verschiedenen Funktionen der Software waren gut integriert.	5	4	3
6	Ich denke, die Software enthält zu viele Inkonsistenzen.	2	2	-
7	Ich kann mir vorstellen, dass die meisten Menschen den Umgang mit dieser Software sehr schnell lernen.	4	5	4
8	Ich fand die Software sehr umständlich zu nutzen.	1	1	3
9	Ich fühlte mich bei der Benutzung der Software sehr sicher.	5	5	4
10	Ich musste eine Menge lernen, bevor ich mit der Software arbeiten konnte.	2	1	1

Table 4: Summary of usability test using SUS scoring and related adjective for each user

Users	Echocardiography experience (Y)	Images analysed	Sex (m/f)	SUS score	Adjective
User 1	> 20	≈ 8	f	92.5	Best Imaginable
User 2	> 5	-	m	92.5	Best Imaginable
User 3*	> 5	≈ 5	m	69.4	Good

It should be noted that User 3* only answered nine questions out of ten. The omitted question was rejected from the SUS score.

5 Discussion

The presented approach focusses on determining local elastic properties of the aorta by using well known image processing techniques on TTE M-mode images. Although “Pulse wave velocity (PWV), a measure of regional aortic stiffness, is the most widely studied and validated noninvasive method because it is a simple, accurate and reproducible []” [24], it does not reflect arteriosclerotic changes in the aortic wall since it is an indirect method of determining regional stiffness. In contrast to this, a standard echocardiography examination combined with the developed software enhances the physician’s opportunities to gain local distension related parameters but also offers to perform a quantitative investigation of the blood flow by using the Doppler technique. Another method to address changes in local stiffness would be the usage of a multi-slice 2D phase-contrast magnetic resonance imaging (MRI) technique to measure the PWV [25]. Both methods require a synchronous recording of ECG for triggering. Although this technology can produce a fairly good temporal resolution of approximately 10 – 30ms, amongst others the comparably long examination time due to a required preceding angiography, as well as the high costs of an investigation make this method inefficient for clinical use. As a matter of course, impaired bioelasticity can be a result of a genetic mutation and therefore be detected using costly molecular and cytogenetic methods.

Relating to the implementation of *AortUs*, an object oriented approach using Python 2.7 as a programming language turned out to be a promising combination for the development and usage of inherited methods and classes. One example showing this benefit is the development of entry fields, embedded in the GUI (i.e. First Name, Last Name, etc.). The GTK+ framework does by default not offer such properties as the automatic labeling of empty *Gtk.Entry* fields or input character restriction. Nevertheless, the implemented inherited class *MyEntry* does imply these properties and therefore integratively considers error prevention. Another developed method reducing the risk of misuse of *AortUs* whilst keeping time demanding manual tasks as little as possible, is the unidirectional backbone of this software, designated as edge detection process. The automatically extracted image features, such as scale axes, triggerpoints and aortic region boundaries, are presented to the user and can be adapted interactively within the *Image Box* if necessary. Although the used active contour method in general fits the

aortic walls comparably accurately, local brightness blackouts may cause bulbes in the semi-automatically segmented and corrected aortic walls on model based assumptions (i.e. positions of found R-peaks, high cross-correlation of both wall edges, no edges with negative temporal gradient), but may also be corrected manually. The excellent result of the semi-automated wall contour analysis is expressed by the high scores of the performed usability test. Even though the number of participating users was low, preliminary results showed that the participated cardiologists seemed to handle *AortUs* safely with a prior briefing of just twenty minutes.

The risk class of this medical product is defined by applying the rules of the MDR 2017/745 on its intended use. Since *AortUs* could be used as diagnostic decision support tool for suspected impaired bioelasticity, but also as a tool used for comprehensive scientific evaluation purposes of aortic distensibility, there is no uniquely dedicated risk class for this software application as demonstrated by Table 2. This depends, however, on the clearly stated intended use of the software. When used as a direct diagnostic tool, *AortUs* is most likely to be classified to classes IIa or IIb, otherwise class I if *AortUs* only displays aortic parameters that are not used for an explicit diagnostic decision or control of body function. Typically, the decision for or against life-threatening therapeutic or surgical treatment is not based on one single parameter, but on a combination of multiple diagnostic parameters. In addition, it can be stated that risks originating from *AortUs* are limited, since the software delivers 3 – 4 single aortic parameters that are part of an extended catalogue of diagnostic criteria needed for medical decision making and therapeutic patient management.

Examinations of 3 selected individuals, analysed by *AortUs*, showed broad differences in estimated parameters. A subject diagnosed with Marfan syndrome (patient-report 1) showed as expected decreased AscAo distensibility of $9.8kPa^{-1} \cdot 10^{-3}$, compared to two healthy persons with values of $42.4kPa^{-1} \cdot 10^{-3}$ (patient-report 2) and $25.1kPa^{-1} \cdot 10^{-3}$ (patient-report 3). It should be noted that the distensibility of the healthy patient 3 may be also decreased because of his age of 48 years versus her age of 4 years.

One methodical weakness to extract elastic aortic parameters from M-mode images is certainly its dependence of the cardiologist's skills and the US device settings. As shown in section 3.1, the accuracy of the measured diameter and therefore of the gained parameters have a strong dependence on the shape of the cross section

focused by the LoS. Even if the aorta has an ideal circular cross section, a LoS not perfectly perpendicular to the aorta's center position leads to elliptical diameter records. Furthermore the examiner is responsible for recording images with transducer settings that emphasise the representation of distinct aortic wall segments whilst keeping noise and artefacts within the aortic lumen as low as possible. This has a major effect on the manual edge correction required to facilitate proper calculation of diameter curves. For the usage of *AortUs* in clinical routine, the adaption of the parameters *Sections per sec* and *Sections per cm* to a loaded M-mode image are prone to be forgotten since it is unlikely to configure them before loading an image. This is a serious problem since i.e. a default value of five sections per second in an image scaled by ten sections per second and a heart rate of $140bpm$ cannot be rejected by an integrative method, since this incorrect scaling leads to a calculated heart rate of $70bpm$ which is also within plausible boundaries for the ECG detection.

Although the active contour calculation has an overall good performance on the test dataset of AscAo images, there is still room for improvement especially for approaching DesAo images. Besides the optimisation of US device settings by the examiner, the correctness of estimated edges may also be enhanced by further customisation of the snake's parametrisation.

A time-consuming but necessary step to improve the stability and functionality of this software would be the usage of a static code analysis tool to determine all possible fault conditions in the software. This should go hand in hand with the setup, execution and analysis of test cases for unit tests, based on the provided image dataset.

For commercial use, a high acceptance level is an indispensable prerequisite. Besides a possible interface to automatically load the patient's personal data to *AortUs*, another functional extension that certainly limits the variety of possible risk classes would be a module for tracking changes of local aortic stiffness over long time. This is of special interest to monitor the effectiveness of a therapeutic treatment, for instance the regular intake of β -blocker drugs after diagnosed decreased aortic distensibility.

6 Conclusion

The main objective of this thesis was to develop an easy to use medical software along the software lifecycle (EN 62303:2006+A1:2015), that calculates the local aortic parameters systolic diameter increase, distensibility as well as the stiffness-index based on the estimated averaged systolic and diastolic diameter of an TTE M-mode image. The necessary requirements to develop this application named *AortUs* were elaborated in a Software Requirement Specification and the software itself was implemented in Python and bundled to a standalone executable, in order to carry out a limited usability test which was well received. The architecture was described in detail and a highly automated approach was used in order to generate the Software Documentation.

Besides this, the scope of this work was to discuss this medical product in terms of risk classification according to the MDR 2017/745. A comparison of possible intended use definitions was devised and discussed with respect to rule eleven of the MDR 2017/745.

In conclusion, it should be stated that local elasticity parameters of the aorta are, by now, not established in course of routine cardiological examinations, but may be a powerful supportive tool in future, by using an automated image processing approach.

References

- [1] World Health Organization. Global status report on noncommunicable diseases 2014.
- [2] Daniel Levy Ramachandran S. Vasan, Martin G. Larson. Determinants of echocardiographic aortic root size. *Circulation*, 91(3), February 1995.
- [3] C. Ross Ethier and Craig A. Simmons. *Introductory Biomechanics, From Cells to Organisms*. Cambridge University Press, 2007.
- [4] Gerhard A. Holzapfel. The extracellular matrix. Lecture 2 Slides of Class 'Mechanics of Biological Tissues', Graz University of Technology, Winterterm 2014/2015.
- [5] Martin A. Zullinger, Alexander Rachev, and Nikos Stergiopoulos. A constitutive formulation of arterial mechanics including vascular smooth muscle tone. *American Journal of Physiology-Heart and Circulatory Physiology*, 287(3):H1135–H1143, 2004.
- [6] Otto Frank. Die grundform des arteriellen pulses: Mathematische analyse. erste abhandlung. *Zeitschrift für Biologie*, 1899.
- [7] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.
- [8] Michael Kaas, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
- [9] Rudolf Stollberger. Deformable models - active contours. Lecture 'Basics of Image Segmentation' Slides of Class 'Biomedical Image Processing', Graz University of Technology, 12 2014.
- [10] Pablo Márquez-Neila, Luis Baumela, and Luis Alvarez. A morphological approach to curvature-based evolution of curves and surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):2–17, January 2014.
- [11] Laurent D. Cohen. On active contour models and balloons. *CVGIP: Image Understanding*, 53(2):211–218, 1991.

- [12] Christian Baumgartner. Gesetzliche Grundlagen. Lecture 'Medizinprodukte für Europa, Hersteller und Händlersicht' Slides of Class 'Medizinprodukterecht', Graz University of Technology, 2017.
- [13] European Parliament and Council of Europe. Regulation (eu) 2017/745 of the European Parliament and the Council of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC. Official Journal of the European Union, May 2017.
- [14] Christian Baumgartner. Software als Medizinprodukt. Lecture 'Medizinprodukte für Europa, Hersteller und Händlersicht' Slides of Class 'Medizinprodukterecht', Graz University of Technology, 2017.
- [15] European Committee for Electrotechnical Standardization. EN 62366-1, medical devices – part 1: Application of usability engineering to medical devices, August 2017.
- [16] Daniela Baumgartner, Christian Baumgartner, Gabor Mátyás, Beat Steinmann, Judith Löffler-Ragg, Elisabeth Schermer, Ulrich Schweigmann, Ivo Baldissera, Bernhard Frischhut, John Hess, and Ignaz Hammerer. Diagnostic power of aortic elastic properties in young patients with Marfan syndrome. *The Journal of Thoracic and Cardiovascular Surgery*, 129(4):730–739, April 2005.
- [17] Nobuyuki Ohte, Tomoaki Saeki, Hiromichi Miyabe, Seichiro Sakata, Saiji Mukai, Junichiro Hayano, Kiyomi Niki, Motoaki Sugawara, and Genjiro Kimura. Relationship between blood pressure obtained from the upper arm with a cuff-type sphygmomanometer and central blood pressure measured with a catheter-tipped micromanometer. *Heart Vessels*, 22(6):410–415, 2007.
- [18] Roland G. Asmar, Jirar A. Topouchian, Athanase Benetos, Fady A. Sayegh, Jean-Jacques Mourad, and Michael E. Safar. Non-invasive evaluation of arterial abnormalities in hypertensive patients. *Journal of Hypertension. Supplement*, 15(2):99–107, 1997.

- [19] Roland Asmar, Athanase Benetos, Jirar A., Pierre Laurent, Bruno Pannier, Anne-Marie Brisac, Ralph Traget, and Bernd I. Levy. Assessment of arterial distensibility by automatic pulse wave velocity measurement. *Hypertension*, 26(3):485–490, 1995.
- [20] John C. Bramwell, Archibald V. Hill, and Bryan A. MacSwiney. *The Velocity of the Pulse Wave in Man in Relation to Age as Measured by the Hot-wire Sphygmograph*. <https://books.google.at/books?id=rQg0cgAACAAJ>, 1923.
- [21] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitalized binary images by border following. *Computer Vision, Graphics and Image Processing*, 30(1):32–46, 1985.
- [22] John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
- [23] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of Usability Studies*, 4(3):114–123, 2009.
- [24] Jae Yeong Cho and Kye Hun Kim. Evaluation of arterial stiffness by echocardiography: Methodological aspects. *Chonnam Medical Journal*, 52(2):101–106, 2016.
- [25] Andrew L. Wentland, Thomas M. Grist, and Oliver Wieben. Review of mri-based measurements of pulse wave velocity: a biomarker of arterial stiffness. *Cardiovascular Diagnosis & Therapy*, 4(2):193–206, 2014.

Appendix

Python Packages and Versions

Programming language: Python 2.7.13

Installation via pip package manager:

Package Name	Version	Usage
XlsxWriter	1.0.2	create MS Excel (Microsoft Corporation, Redmond, Washington, USA) files for data logging
scipy	0.19.0	morphological operators, image and signal filters
numpy	1.12.1	N-dimensional array operations (algebraic, boolean, etc.)
scikit-image	0.13.0	image processing algorithms
PeakUtils	1.1.0	peakfinding
matplotlib	2.0.0	graph and image plotting
PyPDF2	1.26.0	manipulate PDFs
reportlab	3.4.0	format content to PDF

Separate Installation (on Mac OS X i.e. via homebrew)

Software Requirement Specification

for project AortUs

BERNHARD FROHNER

Version 1.2

Revision History

Version	Date	Author	Description
1.0	09.06.2017	Bernhard Frohner	initial setup of SRS
1.1	05.01.2018	Bernhard Frohner	revision of system feature descriptions and naming conventions
1.2	22.03.2018	Bernhard Frohner	typo mistake fixing after master's thesis correction

Acronyms

AscAo ascending aorta 2, 3, 13, 14, 21, 22

BF Bernhard Frohner, BSc. 5, 9

CB Univ.-Prof. Dipl.-Ing. Dr. techn. Christian Baumgartner 5, 9

DesAo descending aorta 2, 3, 13, 14, 21, 22

ECG Electrocardiogram 11, 12, 16, 17, 20, 23

GUI Graphical User Interface 1, 4, 7, 8, 10, 12–14

LoS line of sight 4

MDR 2017/745 Medical Device Regulation 2017/745 25

MRI magnetic resonance imaging 2

OS operating system 4, 15

PACS Picture Archiving and Communication System 8

PWV pulse wave velocity 2

ROI region of interest 11, 12, 18, 19

SOH state of health 23

SRS Software Requirement Specification 3

TTE transthoracal echocardiography 1, 15

US ultrasound 1, 2, 12, 17, 18, 23

Contents

Acronyms	5
1 Introduction	1
1.1 Purpose	1
1.2 Document Conventions	1
1.3 Intended Audience and Reading Suggestions	1
1.4 Project Scope	1
1.5 References	2
2 Overall Description	3
2.1 Product Perspective	3
2.2 Product Functions	3
2.3 User Classes and Characteristics	3
2.3.1 Cardiologist	4
2.3.2 Academic User	4
2.4 Operating Environment	4
2.5 Design and Implementation Constraints	4
2.6 User Documentation	5
2.7 Assumptions and Dependencies	5
3 External Interface Requirements	7
3.1 User Interfaces	7
3.2 Hardware Interfaces	8
3.3 Software Interfaces	8
3.4 Communications Interfaces	8
4 System Features	9
4.1 System Feature 1	9
4.1.1 Description and Priority	9
4.1.2 Stimulus/Response Sequences	9
4.1.3 Functional Requirements	9
4.2 System Feature 2	10
4.2.1 Description and Priority	10
4.2.2 Stimulus/Response Sequences	10
4.2.3 Functional Requirements	10
4.3 System Feature 3	11
4.3.1 Description and Priority	11
4.3.2 Stimulus/Response Sequences	11
4.3.3 Functional Requirements	12
4.4 System Feature 4	12
4.4.1 Description and Priority	12
4.4.2 Stimulus/Response Sequences	12
4.4.3 Functional Requirements	12

4.5	System Feature 5	12
4.5.1	Description and Priority	13
4.5.2	Stimulus/Response Sequences	13
4.5.3	Functional Requirements	13
4.6	System Feature 6	13
4.6.1	Description and Priority	13
4.6.2	Stimulus/Response Sequences	13
4.6.3	Functional Requirements	13
4.7	System Feature 7	14
4.7.1	Description and Priority	14
4.7.2	Stimulus/Response Sequences	14
4.7.3	Functional Requirements	14
4.8	System Feature 8	14
4.8.1	Description and Priority	14
4.8.2	Stimulus/Response Sequences	15
4.8.3	Functional Requirements	15
4.9	System Feature 9	15
4.9.1	Description and Priority	15
4.9.2	Stimulus/Response Sequences	15
4.9.3	Functional Requirements	15
4.10	System Feature 10	16
4.10.1	Description and Priority	16
4.10.2	Stimulus/Response Sequences	16
4.10.3	Functional Requirements	16
4.11	System Feature 11	16
4.11.1	Description and Priority	16
4.11.2	Stimulus/Response Sequences	16
4.11.3	Functional Requirements	17
4.12	System Feature 12	17
4.12.1	Description and Priority	17
4.12.2	Stimulus/Response Sequences	17
4.12.3	Functional Requirements	17
4.13	System Feature 13	18
4.13.1	Description and Priority	18
4.13.2	Stimulus/Response Sequences	18
4.13.3	Functional Requirements	18
4.14	System Feature 14	18
4.14.1	Description and Priority	18
4.14.2	Stimulus/Response Sequences	19
4.14.3	Functional Requirements	19
4.15	System Feature 15	19
4.15.1	Description and Priority	19
4.15.2	Stimulus/Response Sequences	19
4.15.3	Functional Requirements	19
4.16	System Feature 16	20
4.16.1	Description and Priority	20
4.16.2	Stimulus/Response Sequences	20
4.16.3	Functional Requirements	20
4.17	System Feature 17	20
4.17.1	Description and Priority	20
4.17.2	Stimulus/Response Sequences	21

4.17.3	Functional Requirements	21
4.18	System Feature 18	21
4.18.1	Description and Priority	21
4.18.2	Stimulus/Response Sequences	21
4.18.3	Functional Requirements	21
4.19	System Feature 19	22
4.19.1	Description and Priority	22
4.19.2	Stimulus/Response Sequences	22
4.19.3	Functional Requirements	22
5	Other Nonfunctional Requirements	23
5.1	Performance Requirements	23
5.2	Safety Requirements	23
5.3	Security Requirements	23
5.4	Software Quality Attributes	23
5.5	Business Rules	24
6	Other Requirements	25
6.1	Legal Requirements	25

1 Introduction

1.1 Purpose

This document describes the major software requirements for the ultrasound (US) image post-processing software *AortUs* V 1.2. The objective of this standalone application is to support an echocardiography-examining physician with important additional information of an M-mode image of the aorta.

As the aortic enlargement and impaired bioelasticity are of interest in several cardiac and non-cardiac diseases [1], especially cardiologists show great interest in gaining elastic properties of the aorta. *AortUs* should support cardiology healthcare professionals in therapeutic and diagnostic decisions by estimating these relevant properties like i.e. Distensibility and Stiffness Index fast and intuitively.

1.2 Document Conventions

In the following sections, project-related proper names like *Patient Box*, *Log Box* etc. as well as units like *px-per-second*, *sec* etc. are formatted in *italic* font type. All exemplary or implicitly prescribed program code phrases are shown in **typewriter** font.

Collections of characters are expressed and collected within square brackets, separated by commas [] (i.e. [1-9, ., ;, -, *]), whitespace character is expressed as “ ”.

1.3 Intended Audience and Reading Suggestions

Besides software development aspects, the reader should also be familiar with basic terms of US equipment with respect to echocardiologic applications.

The upcoming chapters are separated into a general description of *AortUs*, its dependencies and constraints (2), an explanation of interface requirements (3), functional requirements (4), non-functional requirements (5) and other requirements (6). The functional requirements are sorted starting from very general prescriptions like certain *Graphical User Interface (GUI)* behaviours up to very accurate descriptions of how a certain manual or automatic state change must respond to user interaction.

1.4 Project Scope

As already mentioned, *AortUs* has its field of application in giving an estimation of aortic distensibility and stiffness of the aorta by processing M-mode echocardiography images. These images can be differentiated in two transthoracic echocardiography (TTE) positions:

- proximal ascending aorta (AscAo) 10 to 20 mm distal to the sinotubular junction (parasternal long-axis view)
- abdominal descending aorta (DesAo) proximal to the branching off of the celiac trunk (abdominal paramedian longaxis view)

The gold standard to estimate the arterial stiffness and distensibility is to measure the pulse wave velocity (PWV) [2, 3]. This is typically done by measuring the time a pressure wavelet needs to pass a vessel of certain length. Although it gives accurate and reproducible results, it is a measure of regional arterial stiffness and does not reflect local mechanical changes [4].

Naturally, one could determine the diameter curves of the aorta by using MRI based techniques. While the temporal resolution for US can easily take values of 50 frames per second¹, real-time magnetic resonance imaging (MRI) techniques with 20ms temporal resolution are prone to geometric distortions or even local signal losses [6]. Besides the high computational effort to reconstruct these MRI images, inexact M-mode images can be recorded again easily with mobile US systems.

Changes of elastic properties of the aorta are observed with advancing age, but may also be related to genetic mutations of a patient. Therefore, also molecular and cytogenetic approaches can be used to identify genetic disorders, accepting time demanding and expensive encoding methods.

1.5 References

No external references are used in this specification document.

¹Temporal resolution depending on the depth of sonic penetration, the number of focal points and the number of scan lines per frame [5].

2 Overall Description

2.1 Product Perspective

After the development of a semi-automatic image processing software used in a clinical trial with Marfan-syndrome patients [7], the approach to use this high diagnostic power features extracted from M-mode images showed great response at cardiologic conferences all over the world. Therefore this self-contained M-mode image processing software should act as general tool to assess questions of aortic diameter changes noninvasively, but also to provide an interface to easily append modules for i.e. therapeutic tracking of treatment or machine learning algorithms based on all ever estimated imaging parameters of healthy/diseased subjects.

2.2 Product Functions

In this section the overview of the main functionality of *AortUs* should be given. First of all, the user must enter valid input to all required patient data fields, in section 3.1 described in detail. After this is done, the user should be permitted to load the first M-mode image and start the actual image processing procedure. These steps involve

1. identification of the actual M-mode image
2. extraction of depth- and temporal-resolution (*px-per-cm*, *px-per-sec*)
3. identification of the ROI within the M-mode image (the aorta incl. its wall)
4. detection of wall edges applying the leading edge technique
5. computation of diameter over time curve
6. averaging and (optional) smoothing the triggered separated edges
7. calculation of aortic parameters (Distensibility, Stiffness-Index, Systolic Diameter Increase)
8. export the established parameters including patient data to a patient record file
9. if demanded by the user, load another M-mode image of the same patient and redo steps 1. - 8. to combine these results with those previously calculated

It should be possible to separate calculations, based on M-mode images of AscAo and DesAo transducer positions of a patient.

2.3 User Classes and Characteristics

The intended user class mainly focusses on cardiologists with broad experience in fields of echocardiography. In some cases also academic usage, i.e. for collecting Stiffness Index data for a study, should be conceived. The upcoming subsections should explain those users more

detailed. In any case, the typical user does not have a signal or image-processing background. Thus, expertise demanding procedures like i.e. manual definition of filter coefficients to use a digital filter should be avoided.

2.3.1 Cardiologist

This is the main user class this product aims on, as the cardiologist typically gains interest in cardiac morphology and function, if a patient complains about cardiovascular problems. This user does have a high expertise in fields of an echocardiography examination and is therefore responsible for recording well suited M-mode images of the aorta. The selected element of the transducer to be used for the M-mode image (in the following referred as line of sight (LoS)) must be placed exactly perpendicular to the long axis of the aorta. Considering this, the view should also show the largest achievable diameter in order to determine the diameter courses accurately. It should be noted that the reliability of the calculated time-diameter curves (which is the basis for further aortic parameters) strongly depends on transducer element position.

2.3.2 Academic User

When doing clinical trials with interest in aortic properties, the extraction of parameters like the systolic or diastolic diameter, the Distensibility or the Stiffness-Index may be of interest - just a few examples: [1, 7, 8]. Since *AortUs* has its focus on a fast and simple estimation of these parameters, it may be an appealing tool for supporting these trials.

2.4 Operating Environment

This software should run on Windows 7 (32 and 64bit) or higher as a single executable file without requiring additional dependencies. Besides this, no further operating system (OS) requirements need to be considered.

2.5 Design and Implementation Constraints

The GUI does not require to be according to a designated corporate design, neither a regulatory policy.

Since *AortUs* is a post-examinatory image processing tool, a fast execution speed of internal processes is not mandatory, but recommended to enhance the usability. It is not planned that *AortUs* requires interfacing to other external applications on the OS or databases, so that no certain communication protocols must be provided.

The unauthorized access to use the application *AortUs* or patient records of it must be restricted by the user or IT-responsible for the computer. The latter are also responsible to install updates of the application provided by the manufacturer.

2.6 User Documentation

A user documentation in form of a PDF file in English must be submitted with emphasis on the intended usage and rejection criterias for M-mode images. A certain format or standard for this user documentation is dispensable.

2.7 Assumptions and Dependencies

Most of the requirements listed in this document are based on a kick-off meeting of this project on the 15th of March 2017, with Univ.-Prof. Dipl.-Ing. Dr. techn. Christian Baumgartner (CB) and Bernhard Frohner, BSc. (BF) at the Institute of Health Care Engineering (University of Technology Graz). These requirements had to be extended by BF in order to fulfill the objectives of this software:

- intuitiveness
- simplicity
- plausibility checking
- error avoiding

Furthermore, these requirements had to be rendered more precisely at further meetings of CB and BF, in order to enhance the clinical acceptance of *AortUs*.

3 External Interface Requirements

3.1 User Interfaces

The GUI is the main interaction window where the user controls the image feature extraction procedure. The backend of the program should interface with one single window with a size that fits the main screen of the computer.

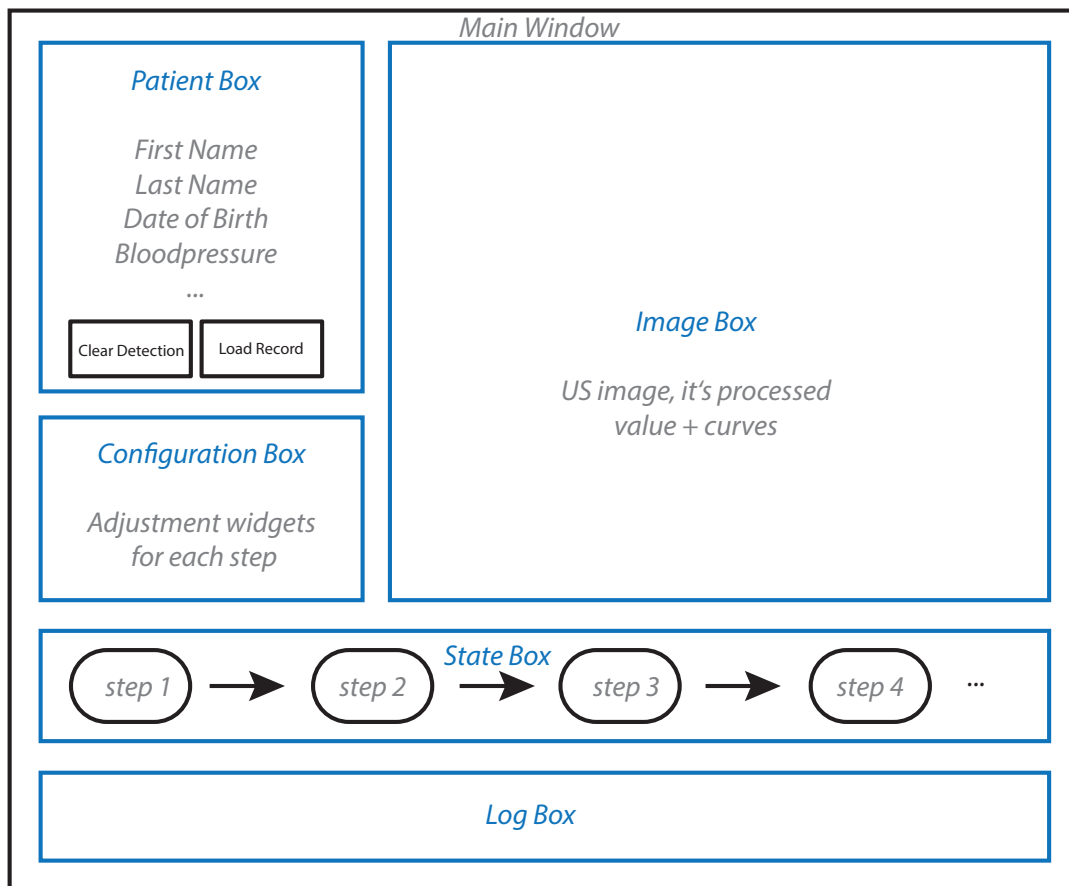


Figure 1: Draft Layout of the GUI main window

As shown in Figure 1, the GUI should contain several layout boxes (rectangles with included blue font box-names) and is not required to be resizable.

The application should start in idle state, waiting for the user to enter patient data within the *Patient Box* and also to load the M-mode image. After this is done, the *State Box* should guide the user through the typical image parameter extraction process. During each step of this automated procedure, the *Configuration Box* should show relevant data and also allow manual correction or interaction if necessary. After each of the state processes has finished, the application should return back to idle state and wait for either manual adaption of the process results

(refer to *Configuration Box*) or for a click event on the next state button.

The *Image Box* should show the M-mode image or details of it. In particular, the aortic wall deflection over time including superposed lines or polygons of detected or processed image details. For the last step of the feature extraction process, the *Image Box* may also contain the averaged diameter curve as well as the discrete values of calculated aortic parameters. The bottom of the GUI should contain the *Log Box* to give feedback during successful or failed process steps. The full content of this logging widget should be accessible by a scrollbar.

Except the case the user clicks or switches through editable fields in the *Patient Box*, File-Save/Open-Dialogues or items within the *Configuration Box*, the application should not respond to keyboard interactions.

If one of the automated image feature extraction processes fails or the user enters invalid data, a warning message should be given to the user by opening a *Message Box* with a meaningful text-content and an OK-button.

3.2 Hardware Interfaces

For this application no external hardware nor related interface is required.

3.3 Software Interfaces

For this software version, no interface to other software component is planned. In further releases there might be an interface to directly share M-mode images and the calculated results of *AortUs* with a Picture Archiving and Communication System (PACS), or an electronic health record.

3.4 Communications Interfaces

For this application no network communication interface is planned.

4 System Features

This chapter describes the main functional requirements, listed as “System Features”. As some of them depend on another, each feature that prerequisites other certain features, will link their direct dependence in the **Dependencies** list.

Each description includes the attributes “**Priority** of consideration” and the related “estimated **Effort**”. These subjective attributes are rated from 1 (low priority/effort) to 9 (high priority/-effort) by the author of this document. The assigned priority is a result of previous meetings of CB and BF, as well as aspects of usability whereas the estimated effort is based on the author’s development experience and alternative possible ways of implementation.

4.1 System Feature 1

Input of Patient Data

4.1.1 Description and Priority

Each usage of the image feature extraction procedure is related to an M-mode image of an echocardiographic examination of a patient.

Priority: 9

Effort: 7

4.1.2 Stimulus/Response Sequences

As long as at least one field of the *Patient Box* does not contain valid content, the image extraction process must not be accessible. Furthermore changes in the *Patient Box* during this state-process can be allowed, but the user shall not be able to proceed if the already valid patient data is changed to invalid patient data.

The user must not be able to enter restricted characters to specific entry-fields. If the user enters semantically incorrect or bad formatted data like i.e. “35.01.1992” in the date of birth field or blood pressure values outside specified boundaries (see SF4.7), the entry-field must be cleared and a *Message Box* must appear (see SF4.3). Each info-string of these *Message Box* calls must also be printed to the *Log Box*.

4.1.3 Functional Requirements

The following patient specific information must be allowed to enter:

- First Name (max. 100 alphabetic characters incl. [, -], restricting numbers, mutated vowels and other special characters)
- Last Name (same restrictions as First Name)
- Date of Birth (in format DD.MM.YYYY, allowing only realistic and past dates)
- Date of Investigation (same Date of Birth incl. today's date)
- Insurance Number (max. 10 digits)
- Systolic and diastolic blood pressure of ascending- and descending aorta (see SF4.7)

To stop the acquisition procedure during any of the image data extraction states and also clear all patient information fields, a “Clear Detection” button must also be available.

When the user successfully finished at least one run of the edge detection process, a “Export Results” button should facilitate to generate a patient report (see SF4.18).

Any other patient specific information can be omitted.

Dependencies: SF4.7

4.2 System Feature 2

State Visualisation

4.2.1 Description and Priority

To ensure intuitive usage, a sequential visualization of the main image processing steps is required.

Priority: 9

Effort: 4

4.2.2 Stimulus/Response Sequences

This state visualisation should permit user interaction to proceed the main process (from step 1 → step 2, step 2 → 3 and so on) but in general prohibit reverse- and “Goto”-step jumps (i.e. step 3 → 2, step 2 → 5), except it is explicitly demanded by any other requirement.

4.2.3 Functional Requirements

The GUI needs to include a *State Box*, embedding a visualisation of the main image processing steps. For detailed information of these steps' objectives, the reader is asked to refer to the list in section 2.2. Except the implementation of the state process's origin by a “Load Image” button, the naming and layout of further steps is up to the developer.

If the user tries to trigger a forbidden (“Goto” or reverse) step, the application should stay in the same state like before the state-trigger event happend.

Dependencies: SF4.3, SF4.1

4.3 System Feature 3

Message Popup Window

4.3.1 Description and Priority

A message window (*Message Box*) pops up if the user triggered an unexpected or restricted event.

Priority: 9

Effort: 1

4.3.2 Stimulus/Response Sequences

Unexpected or restricted events are that

- the user entered an invalid date (see also SF4.1)
- the user entered invalid blood pressure values (see also SF4.7)
- the user passed an invalid image- or patient record file or the passed file could not be opened
- the user tries to move triggerpoints in the Electrocardiogram (ECG) outside the axes limits (see also SF4.11)
- the image does not contain enough information for the extraction algorithm. This is the case if
 - no ECG tracing was found
 - no region of interest (ROI) was found
 - no edges could be extracted
- certain steps of the main process could not be executed automatically. This is the case if
 - the scales could not be detected
 - the ROI could not be detected

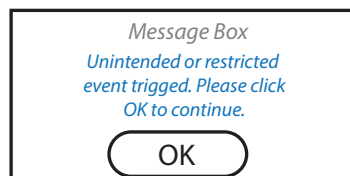


Figure 2: Draft Layout of the *Message Box*

Figure 2 shows how the window of the *Message Box* could look like.

4.3.3 Functional Requirements

The *Message Box* must pop up in front of the GUI to inform the user about unexpected or restricted events. To continue using the main window, the user must click the “OK” button.

Dependencies: SF4.7, SF4.8, SF4.10, SF4.11, SF4.12, SF4.14

4.4 System Feature 4

M-Mode image and estimated *Parameter Box*

4.4.1 Description and Priority

Box for displaying and interacting with loaded M-mode image and for representing calculated results.

Priority: 9

Effort: 8

4.4.2 Stimulus/Response Sequences

The content of this box has to be adapted to the current state of the main image processing procedure. If no image is loaded or the current detection process is cancelled, this box should show a placeholder for the actual M-mode image.

In any other non-terminal state, the box should embed the loaded M-mode or ROI including highlighted detected image elements (scales, ECG, triggerpoints, edges). If the application successfully reached the last state, the box should also embed the established parameters, in particular Distensibility, Stiffness-Index, Systolic Diameter Increase (described in [7]) and the averaged (and optionally smoothed) diameter curve.

4.4.3 Functional Requirements

As already mentioned, highlighting detected image content is mandatory. Since the automatic modes of the state-processes can fail due to deviating image resolution, bit depth or overlaid disturbing US-specific image cursors i.e. from time measurement. Therefore the listed dependencies include single state-processes where manual interaction can be done by click-events within the *Image Box* and the state-specific content of the manual *Configuration Box*.

Dependencies: SF4.10, SF4.12, SF4.13, SF4.14

4.5 System Feature 5

Logging Information

4.5.1 Description and Priority

Information about just triggered GUI-events must be provided within the *Log Box*.

Priority: 8

Effort: 3

4.5.2 Stimulus/Response Sequences

Events that must certainly trigger an entry in the *Log Box* are the same as those from SF4.3 and additionally after

- all image load and clear events
- all state change events

4.5.3 Functional Requirements

The logging information should contain distinct information of the action triggered in *AortUs*, in form of a logging text. Each new entry must be appended to already existing entries in the *Log Box* and start with a new line, followed by the characters ">>" and the current time".

Dependencies: SF4.1

4.6 System Feature 6

Choice between M-mode images of "Transducer Position Types" AscAo and DesAo

4.6.1 Description and Priority

The software should integratively consider the analysis of two different types of examinations for one patient.

Priority: 7

Effort: 5

4.6.2 Stimulus/Response Sequences

It must be possible to define of the transducer position used for an acquired M-mode image, before the edge detection process is started. When this process was finished successfully or cancelled, it should be possible to change the transducer position type

4.6.3 Functional Requirements

To clearly distinguish between AscAo and DesAo parameter extraction, this information of transducer position type must also be part of the exported PDF.

4.7 System Feature 7

Entry Field for Blood Pressure

4.7.1 Description and Priority

Multiple input fields for systolic and diastolic blood pressure records for the AscAo and the DesAo

Priority: 9

Effort: 3

4.7.2 Stimulus/Response Sequences

This input field must be accessible only, when *AortUs* is not in any of the state processes (before loading the first image). As great flexibility and usability is crucial, the user needs to have the possibility to enter at least 5 blood pressure records that are averaged for the calculation of Distensibility, Stiffness-Index and Systolic Diameter Increase. If invalid content is entered, a *Message Box* should appear.

4.7.3 Functional Requirements

The input fields must allow to enter numeric systolic blood pressure values within $50mmHg$ and $300mmHg$, diastolic values should be permitted within $30mmHg$ and $250mmHg$. A *Message Box* should not just appear for non-numeric input, but also for values that are physiologically impossible (values outside defined boundaries, systolic < diastolic values).

As echocardiography examinations can focus the AscAo and the DesAo, input fields for both types must be implemented.

In addition, the GUI must also show the averaged values of the entered systolic or diastolic blood pressure values.

4.8 System Feature 8

Load Image

4.8.1 Description and Priority

A standard open file dialog must be available to pass either an M-mode image to *AortUs*.

Priority: 7

Effort: 1

4.8.2 Stimulus/Response Sequences

The file load dialog must appear if the user either clicks “Load Image” button of the state visualisation. This should not be possible if the application is one of the image data extraction states (every state after an M-mode image is loaded).

Successfully loaded files must lead to an information entry within the *Log Box* about which file was chosen. Files that could not be opened, must also lead to an entry to *Log Box* and additionally to a *Message Box* popup. If the user cancels or closes the dialog, the application should be in the same state as before the dialog was opened.

4.8.3 Functional Requirements

The file dialog itself must offer the possibility to visit all OS available directories.

Dependencies: SF4.2, SF4.5, SF4.3

4.9 System Feature 9

Input M-mode Image

4.9.1 Description and Priority

It must be able to process an M-mode image with defined image format properties.

Priority: 2

Effort: 5

4.9.2 Stimulus/Response Sequences

Whenever the user is able and intends to load an M-mode image, *AortUs* must be able to process an image with certain properties.

4.9.3 Functional Requirements

It must be possible to load an M-mode image from a TTE examination with following properties:

- RGB colouring
- 8bit resolution per channel
- format “jpg”, “png”, “tif” or “bmp”

Dependencies: SF4.8

4.10 System Feature 10

Spatio-temporal M-mode resolution

4.10.1 Description and Priority

To get information about the depth- and time scaling of the M-mode image, *AortUs* should be able to extract this information.

Priority: 8

Effort: 7

4.10.2 Stimulus/Response Sequences

This functionality is necessary to accomplish before the ECG triggerpoints for averaging can be defined.

4.10.3 Functional Requirements

The application should be able to detect the vertical (depth) and horizontal (temporal) scales and its subsections automatically, if the image resolution is high enough and no perturbing image content covers the scales. If this is not the case, the user should be informed in form of a *Message Box* and the application should provide the possibility to enter the vertical and horizontal resolution manually (define *px-per-cm* and *px-per-sec*).

Dependencies: SF4.8

4.11 System Feature 11

Detection of ECG tracing

4.11.1 Description and Priority

In order to enable averaging the wall edge curves over multiple heart cycles, an ECG tracing must be part of the M-mode image.

Priority: 7

Effort: 8

4.11.2 Stimulus/Response Sequences

This functionality depends on the definition of spatio-temporal resolution and is required to furthermore start the edge detection. It is the basis step to facilitate the automatic detection of possible R-peaks in the ECG tracing.

4.11.3 Functional Requirements

The ECG must be continuously present over the whole time axis of the M-mode image, without being covered by US-device specific analysis tools (i.e. vertical bars that cover certain image areas). It must be ensured that the ECG is located at the bottom part of the actual M-Mode image, so that it does not cover any contour relevant parts of the image.

As the M-mode image is usually made up of different shades of grey, the ECG must set oneself apart by using an outstanding color (i.e. cyan, magenta, yellow, red, green, blue). If these prerequisites are fulfilled, an automated image detection algorithm should be able to extract the ECG tracing in order to provide a baseline to set trigger points.

If the ECG cannot be detected automatically, the user must be informed that an edge detection with the loaded image cannot be applied.

Dependencies: SF4.10

4.12 System Feature 12

Detection of R-peaks

4.12.1 Description and Priority

A basic suggestion of triggerpoints to facilitate averaging wall edges over heart-cycles has to be made by the automatic detection of R-peaks within the ECG

Priority: 6

Effort: 5

4.12.2 Stimulus/Response Sequences

After the automated ECG detection was performed successfully, the detection of R-peaks within this curve should take place.

4.12.3 Functional Requirements

The R-peak detection algorithm should be based on a one dimensional numeric datastream so that high flexibility concerning other data sources (i.e. an additional bytestream-file containing the ECG tracing) is provided. Each M-mode image must contain at least two heart-cycles, in order to find discriminable R-peaks. If the automatic peak-detection algorithm fails, the user must have the possibility to create such triggerpoints manually within a supportive tool in the *Configuration Box*. This manual sort of intervention should not just be available if the automatic detection fails, but also when the user wants to define, remove, correct or offset certain triggerpoints.

In addition, the calculated heart rate, based on the *px-per-sec*, must be within $[40, 250]bpm$.

Dependencies: SF4.11

4.13 System Feature 13

Detection of aortic ROI

4.13.1 Description and Priority

To get rid of uninteresting image content, *AortUs* should execute its wall detection on the aortic ROI which is the minimal image section of M-mode image, showing the aorta's wall over time.

Priority: 8

Effort: 5

4.13.2 Stimulus/Response Sequences

This image sectioning functionality should be executed before the detection actual edges is performed.

4.13.3 Functional Requirements

As the usage of the ROI simplifies the implementation of the edge detection algorithm, it is a crucial step. The boundaries are positions in depth dimension of the overall M-mode image, defining the first and the last image row of the aortic ROI like shown in Figure 3. The determi-

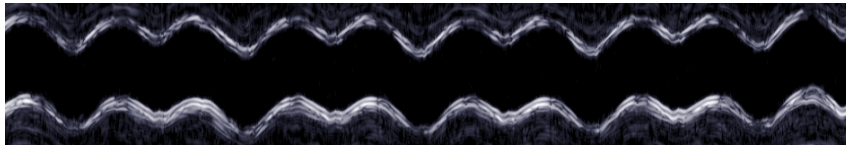


Figure 3: Example for ROI

nation of the ROI can be manual or optionally automated.

It should be noted, that this image section is the only one used to determine the diameter changes of the aorta and thus should not contain any vertical or horizontal cursors from US device specific measurement tools.

4.14 System Feature 14

Aortic wall edge detection

4.14.1 Description and Priority

To estimate the aortic diameter changes for one heart cycle, the aortic wall deflections within the M-mode image must be detected.

Priority: 9

Effort: 9

4.14.2 Stimulus/Response Sequences

The aortic wall edge detection algorithm must be executed after M-mode was reduced to the actual ROI. A precise detection is also the basis for accurate aortic parameters, which are calculated afterwards.

4.14.3 Functional Requirements

The edges should be detected according to the leading edge technique, so that the diameter estimation does consider the wall thickness by measuring from the leading edge of the anterior aortic wall to leading edge of the posterior aortic wall. Since some M-mode images do not show distinct time continuous wall deflections, the user must have the possibility to correct the found edges manually.

If the image quality is not high enough or the visible difference between the aortic wall and its environment is not strong enough, the algorithm can reject the image and inform the user about this by showing a *Message Box*.

Dependencies: SF4.13

4.15 System Feature 15

Calculation and visualisation of extracted parameters

4.15.1 Description and Priority

The aortic elastic parameters must be calculated, based on heart-cycle triggered average diameter curve over time.

Priority: 9

Effort: 4

4.15.2 Stimulus/Response Sequences

This is the very last state that can be reached in the edge extraction process. After this step, the configuration of the “Transducer Position Type” (SF4.6), the export of established parameters as well as loading and processing other images to this patient should be enabled.

4.15.3 Functional Requirements

It must be possible to combine at least 5 previously determined heart-cycle based diameter curves in this step, to calculate reliable results. The systolic- d_s and the diastolic diameter d_d of the aorta as well as their reasoned circular approximated areas $A_{s|d} = \frac{\pi d_{s|d}^2}{4}$ lead to the parameters:

- Distensibility $D = \frac{A_s - A_d}{A_d \cdot (p_s - p_d) \cdot 1333} \cdot 10^7 (kPa^{-1} \cdot 10^{-3})$

- Stiffness-Index $SI = \frac{\ln \frac{p_s}{p_d}}{d_{inc}}$ (*dimensionless*)
- Systolic Diameter Increase $d_{inc} = \frac{d_s - d_d}{d_d}$ (%)

Variables p_s and p_d denote the averaged entered systolic and diastolic blood pressure. Furthermore the visualisation of the averaged aortic diameter changes over time must also be implemented. These results should appear within the *Image Box*.

Dependencies: SF4.14, SF4.12, SF4.1, SF4.7, SF4.17, SF4.18

4.16 System Feature 16

Offset of triggerpoints

4.16.1 Description and Priority

The automatic and manual correction of positions of triggerpoints

4.16.2 Stimulus/Response Sequences

When the ECG was detected and R-peaks were found successfully, the *Configuration Box* should offer a tool for manipulating the temporal position of shown triggerpoints by setting a positive or negative temporal offset.

Priority: 3

Effort: 3

4.16.3 Functional Requirements

By default, triggerpoints must be shifted to $-50ms$ prior to found R-peaks, in order to set correct time ranges to find the systolic and diastolic diameter (see SF4.17). Since the adaption of this automatically defined offset may be required in the future, a function to individually or globally set a temporal offset value must be integrated.

Dependencies: SF4.12

4.17 System Feature 17

Determination of systolic and diastolic diameter

4.17.1 Description and Priority

The extraction of the systolic and diastolic diameter, based on the average diameter curve

Priority: 8

Effort: 1

4.17.2 Stimulus/Response Sequences

After diameter curves are extracted and averaged over heart-cycles, this curves should build the basis for the determination of the systolic maximum and diastolic minimum diameter. By using this extracted values, the actual parameters in SF4.15 can be calculated.

4.17.3 Functional Requirements

These minima and maxima values should be found in the following time ranges:

- Diastolic Minimum Diameter, minimal diameter within $[-50, 200]ms$ around found R-peaks
- Systolic Maximal Diameter, maximal diameter within $[150, 450]ms$ around found R-peaks

It should be noted, that these values change to $[0, 250ms]$ and $[200, 500]ms$ for the default triggerpoint offset of $-50ms$ (see SF4.16).

Dependencies: SF4.16, SF4.14

4.18 System Feature 18

Export of calculated parameters and plots

4.18.1 Description and Priority

It must be possible to export parameters calculated for each edge detection run of AscAo and DesAo to a patient examinations PDF

Priority: 8

Effort: 5

4.18.2 Stimulus/Response Sequences

This action can only be used if the user at least finishes one parameter calculation process successfully. If no set of diameter curves and parameters is available, this function must be restricted.

4.18.3 Functional Requirements

The exported PDF must contain at least the following information:

1. *AortUs* version, date of usage and name of user
2. all content of the patient data fields from SF4.1
3. average blood pressure values
4. processed M-mode images including superposed content (at least the wall edges)

5. calculated single- and averaged diameter curves as well as related calculated parameters

It should be noted that elements 4 - 5 must be separated for analysis of AscAo images from DesAo images.

Dependencies: SF4.14

4.19 System Feature 19

Cancel main edge detection process

4.19.1 Description and Priority

Whenever the user is in one of the main image processing states, it should be possible to stop this process by jumping back to the “Load Image” state.

Priority: 4

Effort: 2

4.19.2 Stimulus/Response Sequences

Except the entry point by loading the M-mode image, the order and actual implementation of further main image processing steps are up to the developer. In any of these steps the user should have the possibility to return to the “Load Image” state.

4.19.3 Functional Requirements

This cancellation must clear all data extracted by the main edge detection process so far. The *Image Box* should show the idle image described in SF4.4.

Dependencies: SF4.2, SF4.4

5 Other Nonfunctional Requirements

5.1 Performance Requirements

In general, a global goal of the software should be to extract these parameters from one image within around 10 minutes, assuming the user is already familiar with *AortUs*. Besides this, other performance requirements can be neglected.

5.2 Safety Requirements

In case of a risk class IIa product, this product may be used as a diagnostic decision support for questions regarding the biomechanical behaviour of the aorta. Thus the software does not have an direct impact on the patient's state of health (SOH), although serious deterioration of the patient's health can be a consequence of mistaken therapy, depending on the intended use. Thus the definition of additional safety requirements can be neglected for the development process.

5.3 Security Requirements

The usage of *AortUs* itself is not restricted to certain users of the machine, as the misuse is not dangerous. Nevertheless, the generated data of this application is sensible and must therefore be protected against unauthorised usage by the user.

5.4 Software Quality Attributes

As *AortUs* is based on M-mode images of different diagnostic US manufacturers and models, especially adaptability, interoperability and maintainability plays a crucial role. For instance, the detection of the ECG should not be based on its specific color or for the detection of the scales the algorithm should not be based on a device specific certain pixel row, but on an certain model independent general morphological conditions which should first of all be elaborated by the different M-mode image examples provided. This conditional parametrisation should always be central available at the top of a method or class code, including comments for the description of its usage if not self-explanatory. To get the point, none of the image detecting algorithms should be strictly fitted to one certain M-mode image type, but on at least 2 morphologically different image types to facilitate a flexible solution.

Other quality aspects to be considered are robustness (in terms of subject an time-varying quality changes) and a high level of usability. Differences in wall edge image quality can have multiple reasons (physiological, technical, examination related) but they must be considered when implementing the detection algorithms. To enhance the likelihood of clinical usage of

AortUs, not just the intuitive state based process is required (refer to SF4.2), but also a well-arranged positioning of entry-fields and buttons must be ensured. It should be intuitive enough so that a user without prior knowledge about this application should be able to step through the main image feature extraction process within 20 minutes.

5.5 Business Rules

As there is just one single user mode, a definition of business rules can be neglected.

6 Other Requirements

6.1 Legal Requirements

The aim of this project is not just to develop the application itself, according to all previously described requirements, but also to consider Medical Device Regulation 2017/745 (MDR 2017/745) classification aspect to bring the software on the market. Based on the intended use described in the user manual, *AortUs* modules will be classified to one of four risk classes.

Bibliography

- [1] I. Voges, M. Jerosch-Herold, J. Hedderich, E. Pardun, C. Hart, D. D. Gabbert, J. H. Hansen, C. Petko, H.-H. Kramer, and C. Rickers, “Normal values of aortic dimensions, distensibility, and pulse wave velocity in children and young adults: a cross-sectional study,” *Journal of Cardiovascular Magnetic Resonance*, no. 14:77, 2012.
- [2] R. Asmar, J. Topouchian, A. Benetos, F. Sayegh, J. Mourad, and M. Safar, “Non-invasive evaluation of arterial abnormalities in hypertensive patients,” *Journal of Hypertension. Supplement*, vol. 15, no. 2, pp. 99–107, 1997.
- [3] R. Asmar, A. Benetos, Jirar A., P. Laurent, B. Pannier, A. Brisac, R. Traget, and B. Levy, “Assessment of arterial distensibility by automatic pulse wave velocity measurement,” *Hypertension*, vol. 26, no. 3, pp. 485–490, 1995.
- [4] J. Cho and K. Kim, “Evaluation of arterial stiffness by echocardiography: Methodological aspects,” *Chonnam Medical Journal*, vol. 52, no. 2, pp. 101–106, 2016.
- [5] A. Ng and J. Swanevelder, “Resolution in ultrasound imaging,” *Continuing Education in Anaesthesia, Critical Care & Pain*, vol. 11, no. 5, 2011.
- [6] M. Ueckera, S. Zhanga, D. Voita, A. Karausa, K. Merboldta, and J. Frahma, “Real-time mri at a resolution of 20 ms,” *NMR in Biomedicine*, vol. 23, no. 8, pp. 986–994, June 2010.
- [7] D. Baumgartner, C. Baumgartner, G. Mátyás, B. Steinmann, J. Löffler-Ragg, E. Schermer, U. Schweigmann, I. Baldissera, B. Frischhut, J. Hess, and I. Hammerer, “Diagnostic power of aortic elastic properties in young patients with marfan syndrome,” *The Journal of Thoracic and Cardiovascular Surgery*, vol. 129, no. 4, pp. 730–739, April 2005.
- [8] A. Redheuil, W. Yu, C. Wu, E. Mousseaux, A. de Cesare, R. Yan, N. Kachenoura, D. Bluekme, and J. Lima, “Reduced ascending aortic strain and distensibility: Earliest manifestations of vascular aging in humans,” *Hypertension*, vol. 55, no. 2, pp. 319–326, 2010.

Feedbackbogen AortUs

Danke, dass Sie sich trotz des hektischen Klinikalltags **5 Minuten** für ein kurzes Gebrauchstauglichkeits-Feedback von *AortUs* Zeit genommen haben. ☺

Die Fragebögen werden **anonym** ausgewertet und in die Masterarbeit „Aortic Distensibility Estimation by M-Mode Echocardiographic-Data“ mit aufgenommen.

Anwender

M	W
---	---

Erfahrungslevel Echocardiographie

>5J	>10J	>15J	>20J	30J
-----	------	------	------	-----

Datum

Anzahl ausgew. Bilder mit *AortUs*

ca.

		Stimme überhaupt nicht zu			Stimme völlig zu	
1	Ich denke, dass ich die Software gerne häufig benutzen würde.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	Ich fand die Software unnötig komplex.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	Ich fand, die Software war einfach zu benutzen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4	Ich glaube, ich würde die Hilfe einer technisch versierten Person benötigen, um die Software benutzen zu können.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5	Ich fand, die verschiedenen Funktionen der Software waren gut integriert.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6	Ich denke, die Software enthält zu viele Inkonsistenzen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7	Ich kann mir vorstellen, dass die meisten Menschen den Umgang mit dieser Software sehr schnell lernen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8	Ich fand die Software sehr umständlich zu nutzen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9	Ich fühlte mich bei der Benutzung der Software sehr sicher.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10	Ich musste eine Menge lernen, bevor ich mit der Software arbeiten konnte.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Das kann passieren, wenn AortUs einen Fehler in der Berechnung macht (Folgerisiken):

Das möchte ich noch sagen:

Fragen/Anregungen/
Wünsche?

Bernhard Frohner
frohner@student.tugraz.at
+436604057005

DANKE!

AORTUS PARAMETER EXPORT REPORT

**21 Year old Female
Marfan Syndrome Typ 1 Patient**

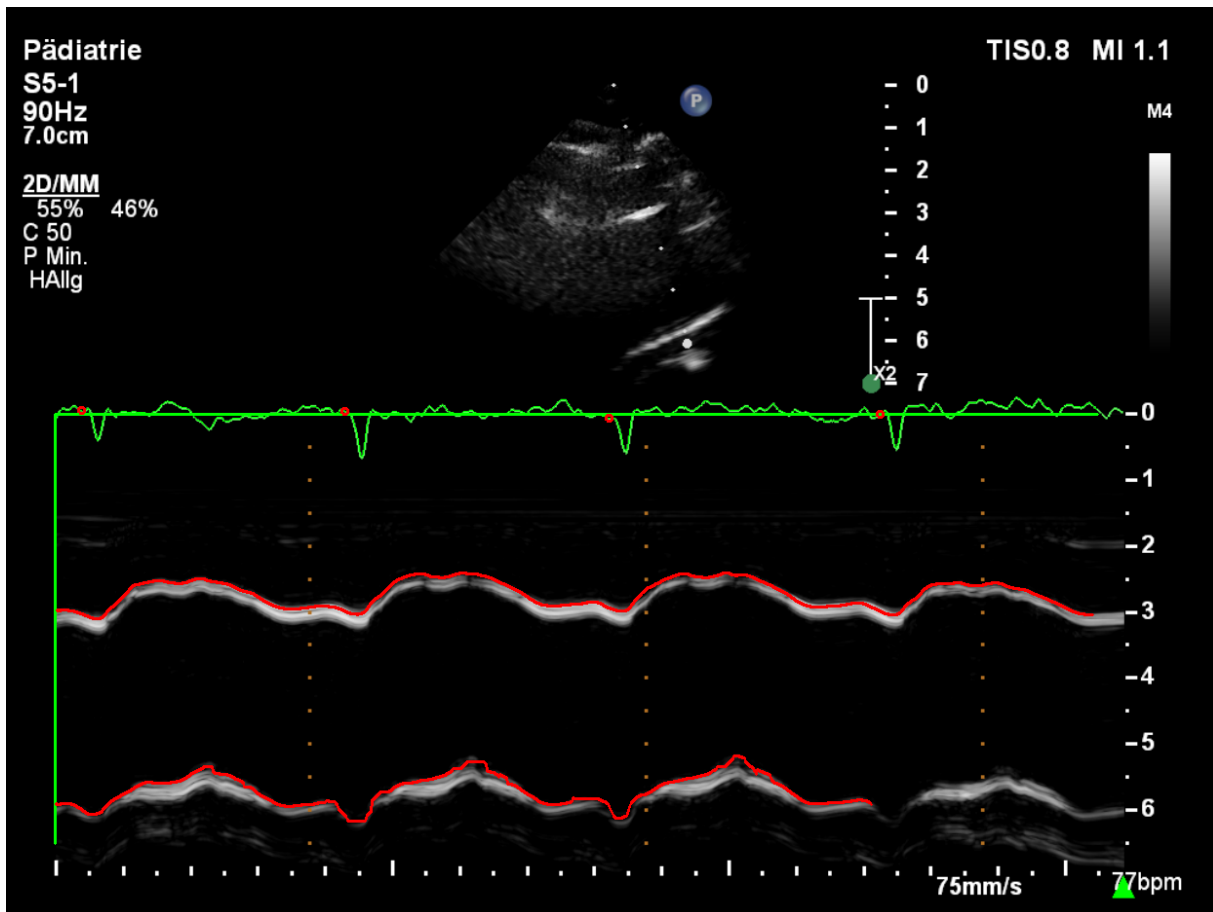
AORTUS VERSION: 1.0.2
DATE OF USE: 04.03.2018
EXAMINER: Daniela

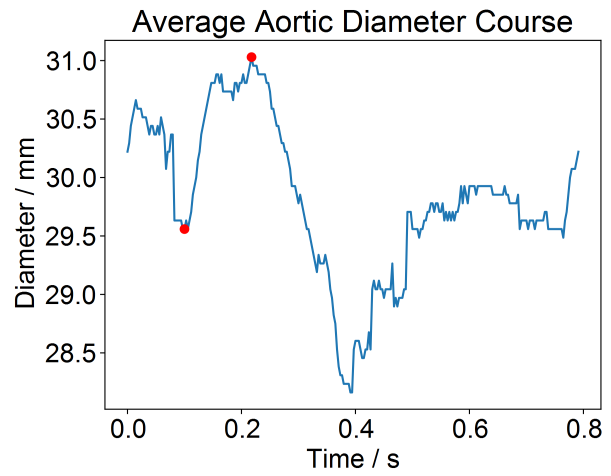
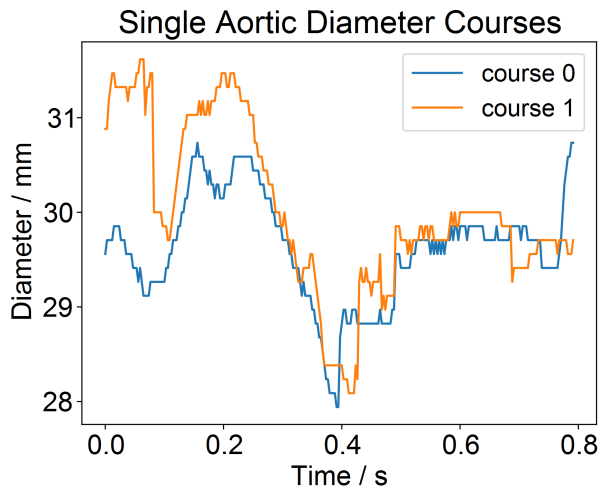
Patient Data

First Name: [REDACTED]
Last Name: [REDACTED]
Date of Birth: [REDACTED] 1997
Insr. Nr: [REDACTED]

Transducer Position "AscAo"

Date of Examination: 28.02.2018
Filename: [REDACTED].bmp
Mean Sys/Dia BP (mmHg): 95.0/57.0

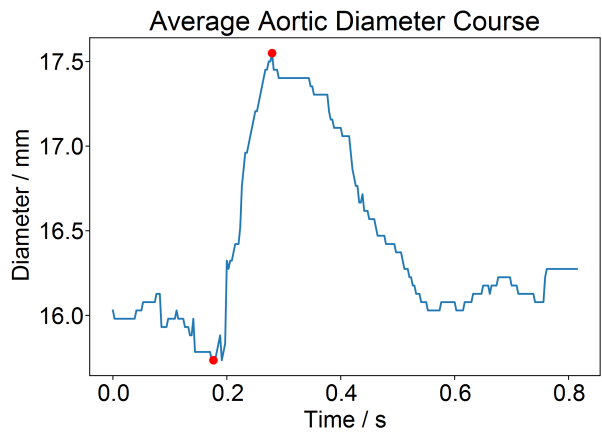
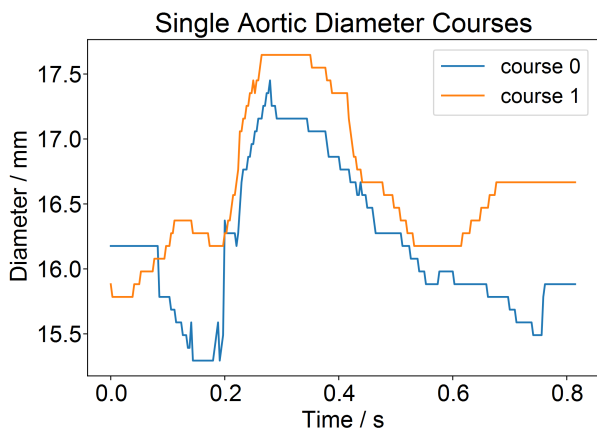
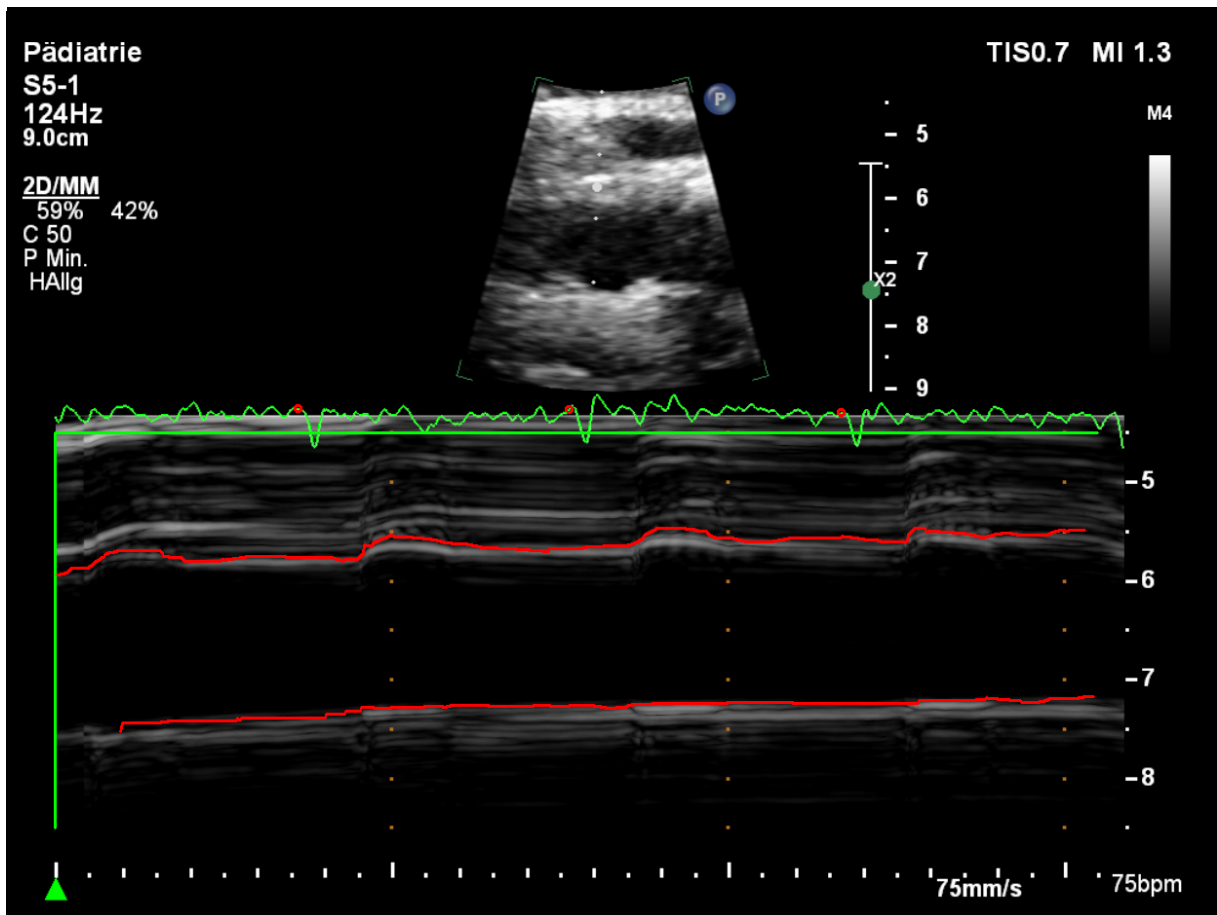




Diast. Diameter (mm): 29.6
 Syst. Diameter (mm): 31.0
 Syst. Diameter Increase (%): 5.0
 Distensibility (kPa⁻¹ 10⁻³): 9.8
 Stiffness Index: 10.3

Transducer Position "DesAo"

Date of Examination: 28.02.2018
 Filename: ██████████.bmp
 Mean Sys/Dia BP (mmHg): 95.0/57.0



Diast. Diameter (mm):	15.7
Syst. Diameter (mm):	17.5
Syst. Diameter Increase (%):	11.5
Distensibility (kPa ⁻¹ 10 ⁻³):	22.8
Stiffness Index:	4.4

AORTUS PARAMETER EXPORT REPORT

**4 Year old Female
Healthy Patient**

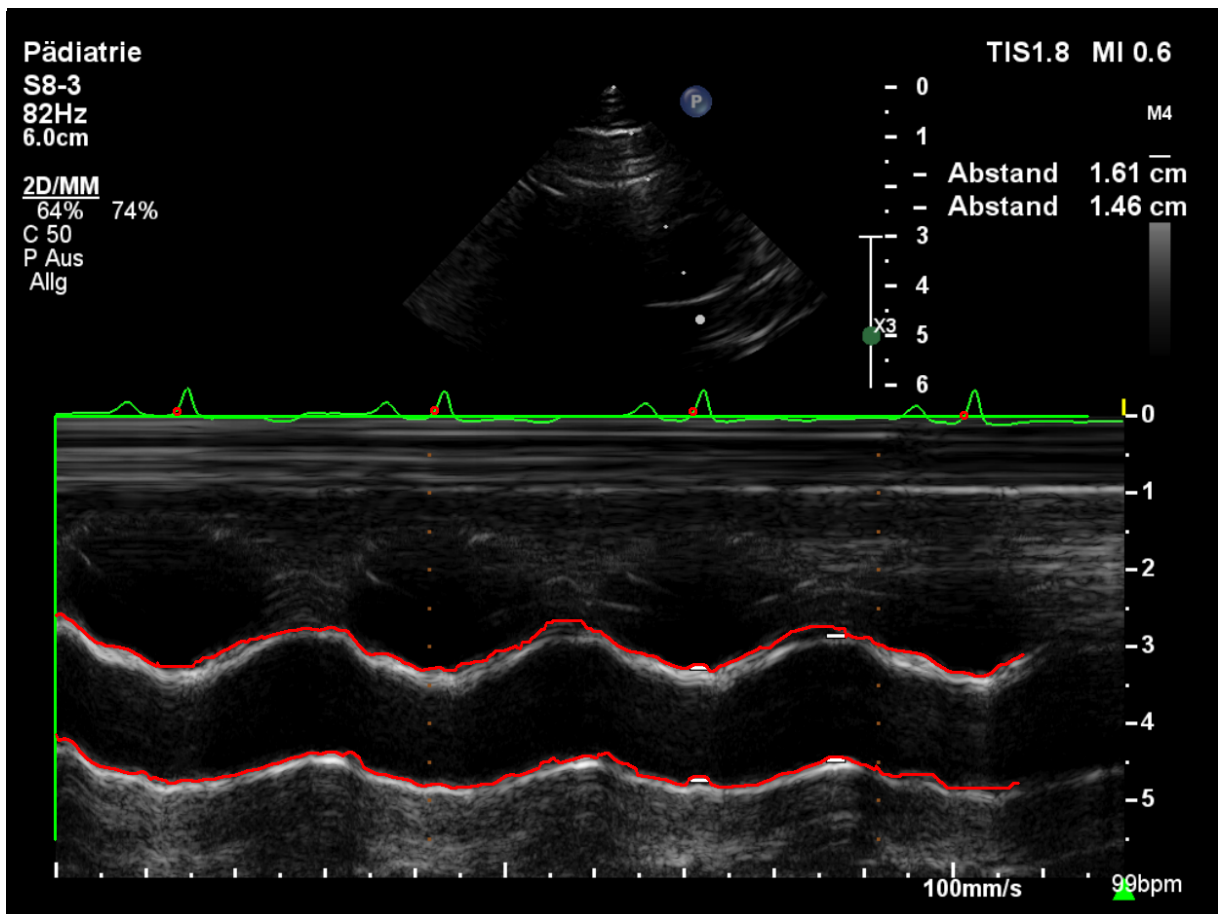
AORTUS VERSION: 1.0.2
DATE OF USE: 04.03.2018
EXAMINER: Daniela

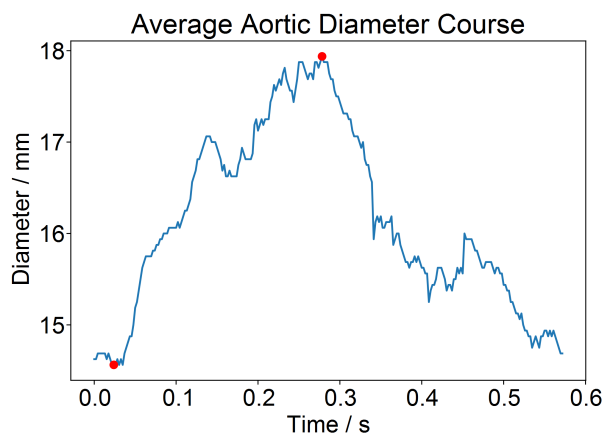
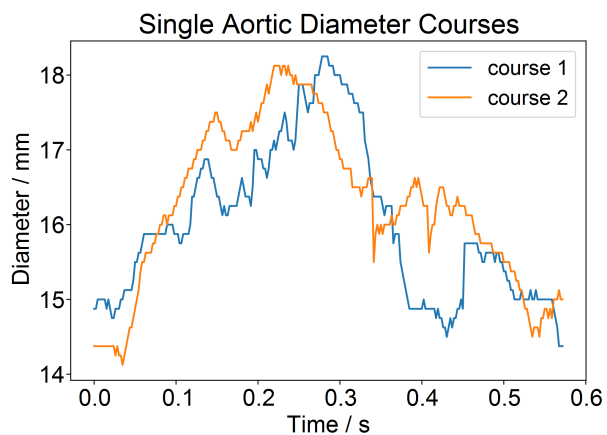
Patient Data

First Name: [REDACTED]
Last Name: [REDACTED]
Date of Birth: [REDACTED] 2013
Insr. Nr: [REDACTED]

Transducer Position "AscAo"

Date of Examination: 04.12.2017
Filename: [REDACTED].bmp
Mean Sys/Dia BP (mmHg): 113.0/72.0

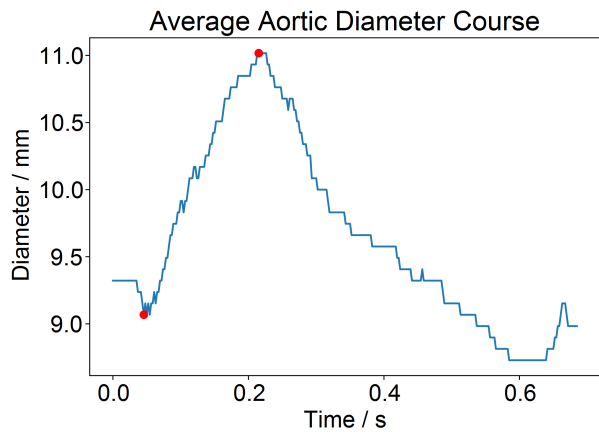
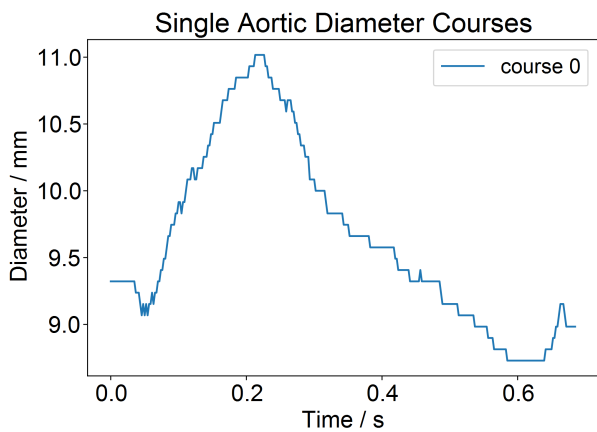
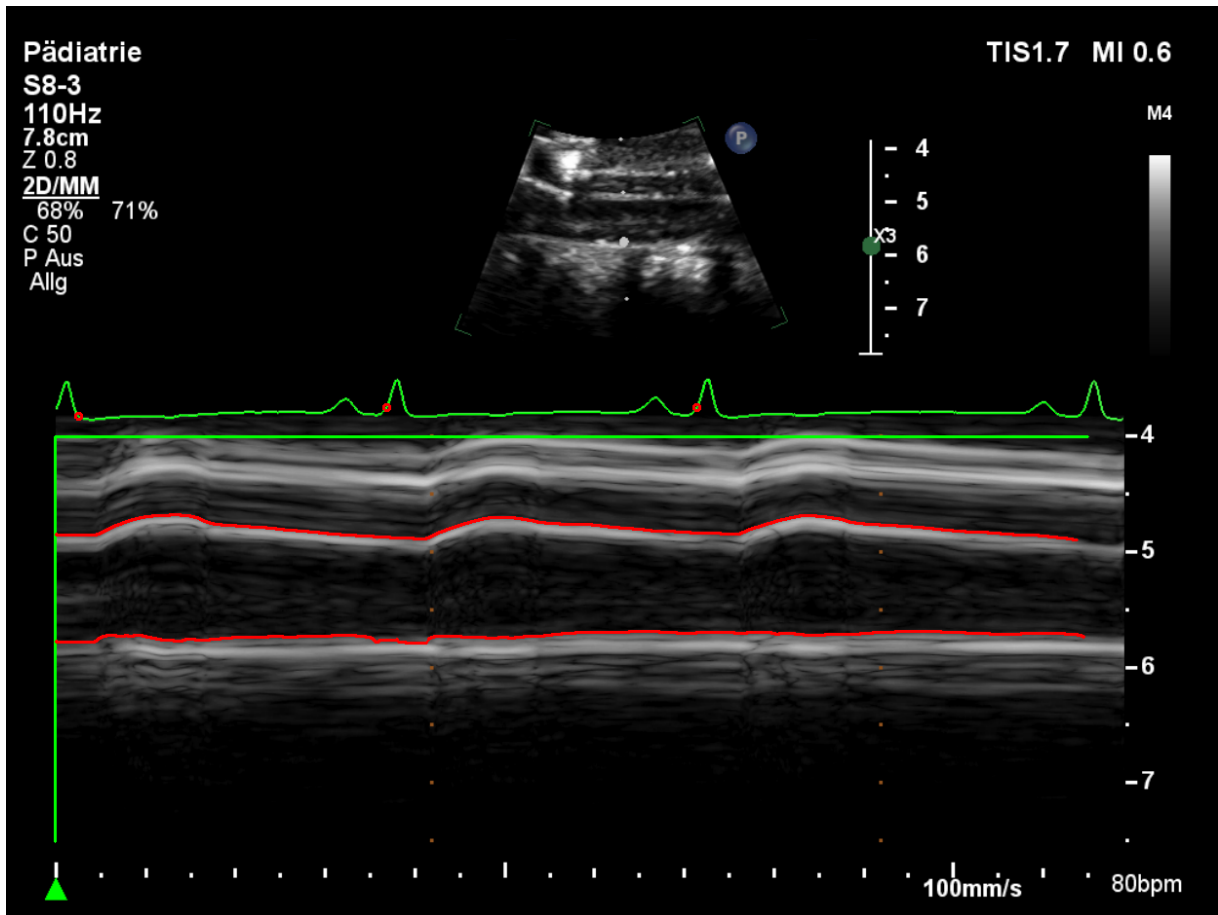




Diast. Diameter (mm): 14.6
 Syst. Diameter (mm): 17.9
 Syst. Diameter Increase (%): 23.2
 Distensibility (kPa⁻¹ 10⁻³): 42.4
 Stiffness Index: 1.9

Transducer Position "DesAo"

Date of Examination: 04.12.2017
 Filename: [REDACTED].bmp
 Mean Sys/Dia BP (mmHg): 113.0/72.0



Diast. Diameter (mm):	9.1
Syst. Diameter (mm):	11.0
Syst. Diameter Increase (%):	21.5
Distensibility (kPa ⁻¹ 10 ⁻³):	39.3
Stiffness Index:	2.1

AORTUS PARAMETER EXPORT REPORT

**48 Year old Male
Healthy Patient**

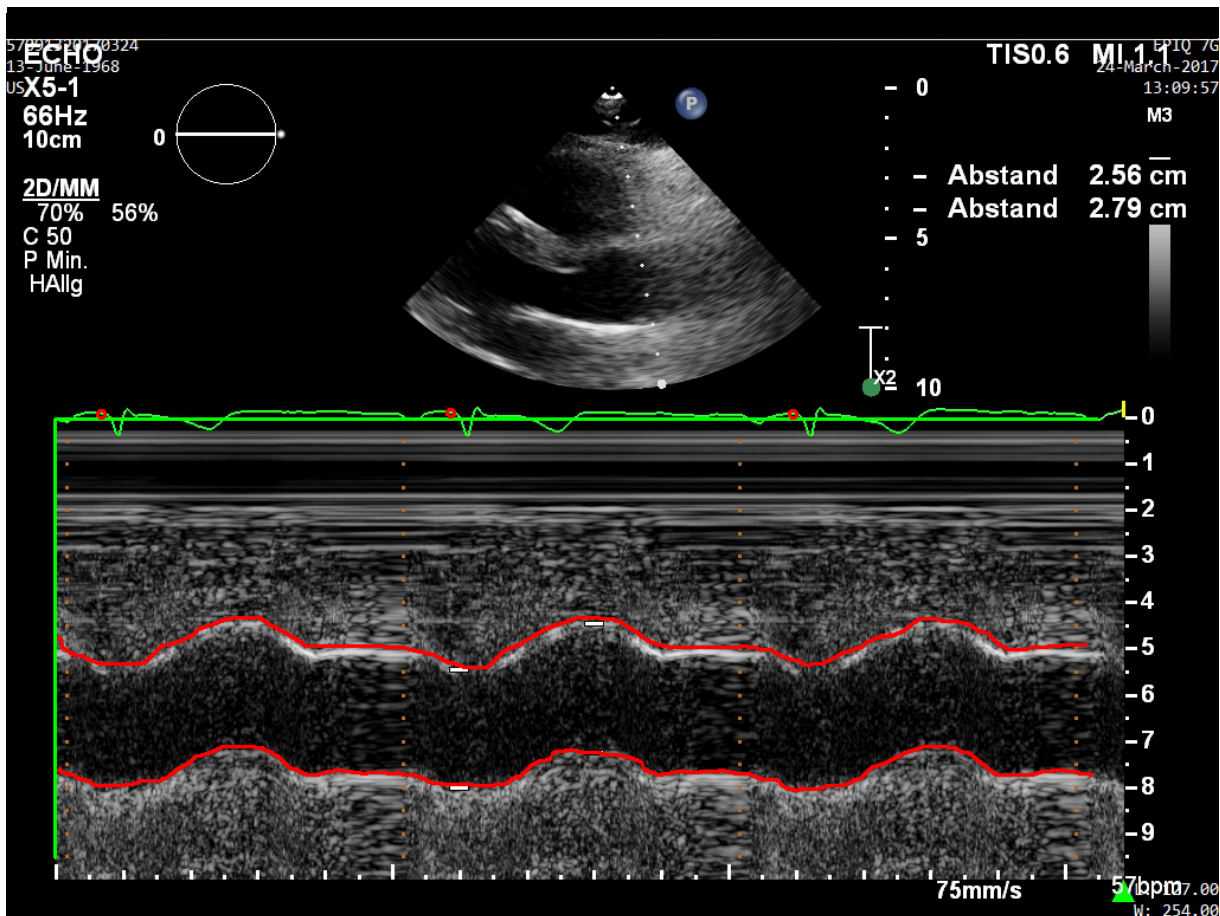
AORTUS VERSION: 1.0.2
DATE OF USE: 14.03.2018
EXAMINER: BF

Patient Data

First Name: [REDACTED]
Last Name: [REDACTED]
Date of Birth: [REDACTED]
Insr. Nr: [REDACTED]

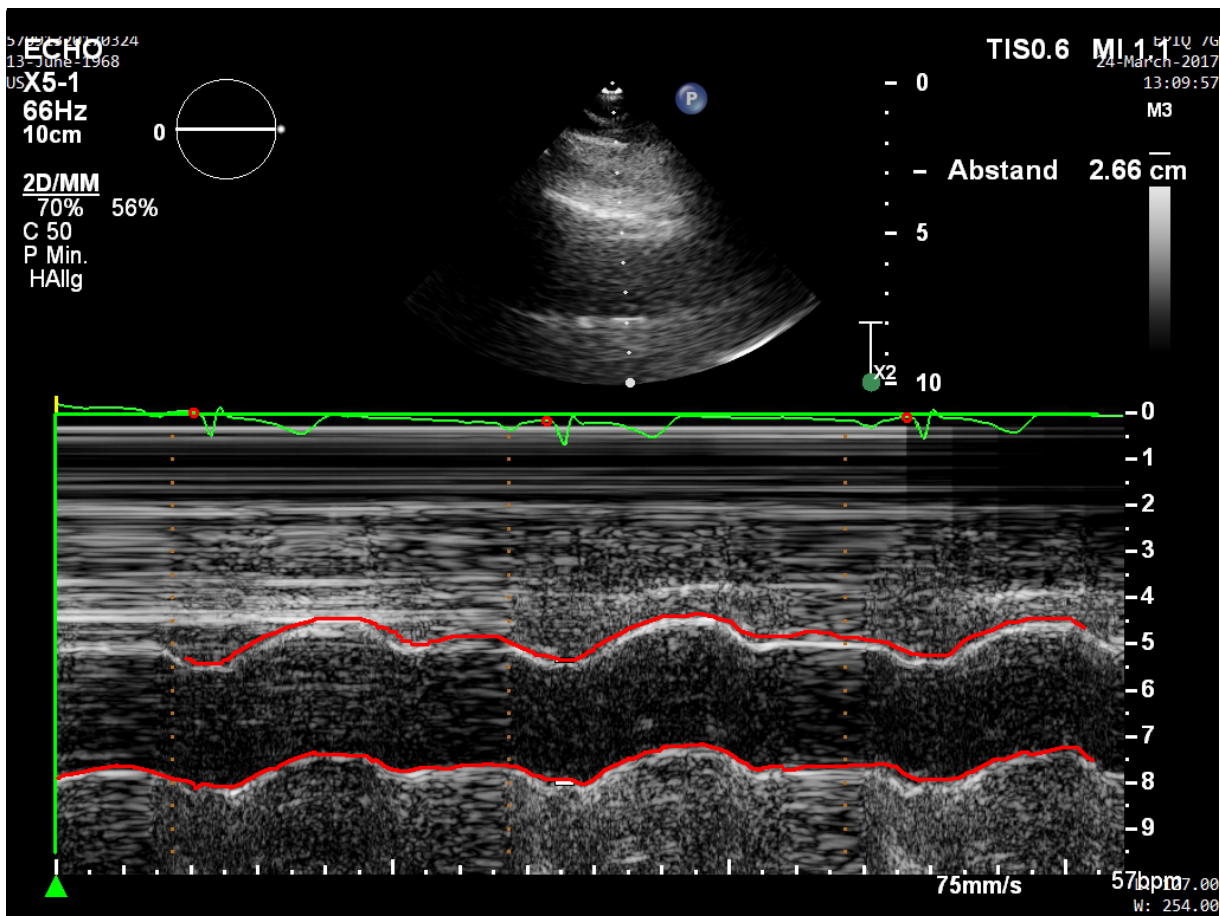
Transducer Position "AscAo"

Date of Examination: 30.04.2017
Filename: AA2_144X76F4Y.jpg
Mean Sys/Dia BP (mmHg): 126.5/92.5

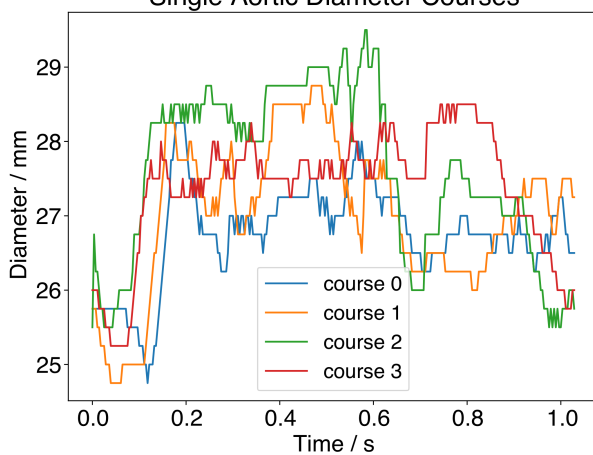


Date of Examination: 30.04.2017
Filename: AA2_244X76F50.jpg

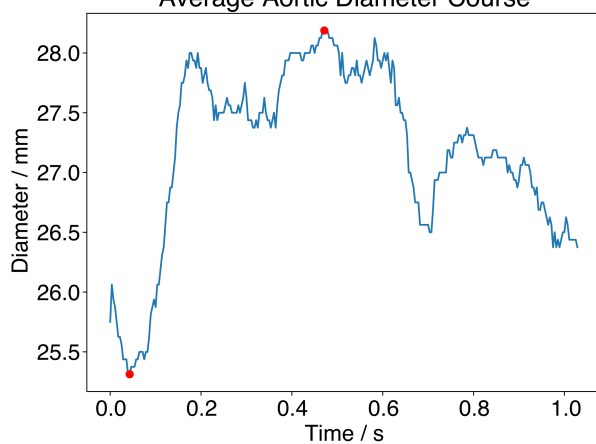
Mean Sys/Dia BP (mmHg): 126.5/92.5



Single Aortic Diameter Courses



Average Aortic Diameter Course



Diast. Diameter (mm): 25.3
 Syst. Diameter (mm): 28.2
 Syst. Diameter Increase (%): 11.4
 Distensibility (kPa-1 $\cdot 10^{-3}$): 25.1

Stiffness Index: 2.8

User Manual

for project AortUs

March 2018

BERNHARD FROHNER

Revision History

Version	Date	Author	Description
1.0	23.02.2018	Bernhard Frohner	initial setup of user manual
1.1	06.03.2018	Bernhard Frohner	adaption of screenshots to current software version V1.1 <ul style="list-style-type: none">• correction of typos• change of <i>Message Box</i> appearance reasons• defining the Intended User Group• adaption of System Requirements
1.2	22.03.2018	Bernhard Frohner	typo mistake fixing after master's thesis correction

Warning and Safety Instructions

This manual contains relevant warnings of the software, in the following referred to as *AortUs*, which must be observed by the user.

The device is only intended for the designated use described in this documentation. This manual will also explain essential prerequisites to ensure the correct, smooth operation of *AortUs*. BF e.U. can not offer warranty nor except any liability if the software is used in applications other than those described.

The software may only be used and operated by personnel, who, due to their qualifications, are capable of understanding the instructions of *AortUs* during use and operation. The operation principle of *AortUs* is such that the accuracy of calculated results depends not only on the fulfillment of operating requirements for *AortUs*, but also on a variety of peripheral conditions beyond the control of the manufacturer. Therefore, the results obtained from this software must be released by an expert before any other diagnostic or therapeutic treatment is taken based on those results.

Bernhard Frohner e.U.

Document Conventions

Typographic Conventions

Warning: Special care needs to be taken regarding the described warning, in order to ensure correct calculated results.

Note: Useful remarks regarding the described topic may be given.

Text Styles and Variables

Bold	Software Parameters; important text
<i>Italic</i>	Software related names of boxes, buttons and other widget elements; math expression
Typewriter	Keyboard input or output
<User>	wildcard for the term within the characters “<” and “>”

Abbreviations

This document contains abbreviations, explained with full name at the very first mention in format <full name> (<abbreviation>) and abbreviated for all other usages. A full list of used acronyms can be obtained in Acronyms on page 7.

Screenshot Examples

The following manual will describe the usage of *AortUs* by using real M-mode images of the aorta. This is supported by screenshots of the software version V1.0.2. It should be noted that supporting text-messages or defined boundaries may vary, depending on the used software version. Furthermore, the patient’s personal data on the used images were anonymised and all entered data in the upcoming examples are fictive.

Acronyms

AscAo ascending aorta 1, 7, 11

BP blood pressure 8, 11, 18, 21

DesAo descending aorta 1

ECG Electrocardiogram 1, 2, 14, 15, 21, 24, 27

LoS line of sight 2

OS operating system 2, 5, 12, 19

ROI region of interest 2, 3, 14–16, 21–24

TTE transthoracic echocardiography 1–3

Contents

- Acronyms** **7**

- 1 General** **1**
 - 1.1 Basic Information 1
 - 1.2 Intended User Group 1
 - 1.3 Definition of Parameters 2
 - 1.4 Examination Prerequisites 2

- 2 Software Launch** **5**
 - 2.1 System Requirements 5
 - 2.2 Logfile 5
 - 2.3 Start Up 5

- 3 User Interface** **7**
 - 3.1 Main Window 7
 - 3.1.1 Patient Box 8
 - 3.1.2 Config Box 9
 - 3.1.3 Image Box 9
 - 3.1.4 State Box 10
 - 3.1.5 Log Box 10

- 4 Usage Example** **11**

- 5 Manual Interventions and Notifications** **21**
 - 5.1 Message Box 21
 - 5.2 Manual Scale Detection 21

- 6 Maintenance and Feedback** **25**

1 General

1.1 Basic Information

The measurement of aortic wall diameters is well used to address medical questions of aortic properties such as distensibility and stiffness. The arterial stiffness does not just increase over age, but can also be affected by the impairment of the connective tissue. A fast and economical way of exploiting such changes, is transthoracic echocardiography (TTE). It is broadly used to detect not just abnormal behaviour of cardiac valves, but also of the temporal changes of the aortic diameters.

AortUs is a standalone software that calculates elastic aortic properties, based on such M-mode echocardiography images of the ascending aorta (AscAo) and descending aorta (DesAo) and on oscillometric blood pressure values. It is capable of processing recorded M-mode images of the AscAo or DesAo in a semiautomatic way, that aims to detect the leading edges of the aortic walls. These edges denote the vessel's diameter over time and can therefore be averaged over covered heart cycles, as long as the M-mode image also contains a coloured Electrocardiogram (ECG) tracing.

In order to extract these heart cycle based curves as simple as possible, the main *Edge Detection Process* of *AortUs* guides the user in six distinct steps to establish the following aortic parameters:

- Diastolic Diameter (*mm*)
- Systolic Diameter (*mm*)
- Systolic Diameter Increase (%)
- Distensibility ($kPa^{-1} \cdot 10^{-3}$)
- Stiffness Index (dimensionless)

It is comprehensible that the M-mode image must at least contain two heart-cycle based diameter curves in order to build the average diameter curves and based on this, derive previously mentioned parameters. Since the accuracy of the estimated diastolic and systolic diameter increases with the number of averaged diameter curves, *AortUs* facilitates to process multiple images to one single aortic parameter set. The resulting curves and quantities can be exported to a patient examination report PDF.

1.2 Intended User Group

AortUs primarily focusses on cardiologists that gain interest in impaired local bioelasticity of the aorta. This user group has expertise in performing TTE examinations and is therefore responsible for recording M-mode images that respect the criterias described in the Examination Prerequisites.

Besides this, also academic users with know-how in terms of aortic M-mode images are intended

to use this software, when instructed by a cardiologist.

In any case, the user has to process M-mode images generated by a cardiologist and additionally needs to be comfortable with the operating system (OS) as well as application software basics.

1.3 Definition of Parameters

The aortic parameters are calculated as follows

- Diastolic Diameter d_d (the minimal diameter in the averaged wall curves)
- Systolic Diameter d_s (the maximal diameter in the averaged wall curves)
- Systolic Diameter Increase d_{inc}

$$d_{inc} = \frac{d_s - d_d}{d_d} (\%) \quad (1.1)$$

- Distensibility D

$$D = \frac{A_s - A_d}{A_d \cdot (p_s - p_d) \cdot 1333} \cdot 10^7 (kPa^{-1} \cdot 10^{-3}) \quad (1.2)$$

- Stiffness Index SI

$$SI = \frac{\ln \frac{p_s}{p_d}}{d_{inc}} \quad (1.3)$$

Formuals (1.1) till (1.3) take use of the approximated circular aortic cross section A_s for systolic and A_d for diastolic diameter. Thus it should be noted that *AortUs* can give rough estimations of the aorta's properties with idealised shape. The parameters p_s and p_d denote the average measured systolic and diastolic blood pressure, respectively (both in *mmHg*).

1.4 Examination Prerequisites

Since there is no universal image standard for TTE aortic M-mode images, the examiner must ensure that the record respects the following conditions:

- 24bit RGB image, embedding a grayscale M-mode image with clearly visible transducer-near and far aortic walls
- maximation of axial transducer resolution
- the aorta must not be located near the M-mode region of interest (ROI)'s boundaries
- coloured ECG tracing recorded synchronous to the wall distortions, not covering wall image content
- each image covers at least two cardiac cycles (three clearly detectable R-peaks)
- the line of sight (LoS) representing the M-mode recording must be exactly perpendicular to the aorta
- the LoS should represent the position of maximal systolic diameter change
- the image must not contain any vertical or horizontal cursors from the TTE-device's measurement tools

- the image must not contain any other ROI breaking content, i.e. black areas within the M-mode ROI evoked by changes of TTE-device settings
- the axes of the M-mode area must be defined by a scaling of one of the following shapes
 - a purely horizontal line including orthogonal intersection lines on the area’s top border as well as a vertical line including orthogonal intersection lines on its left border
 - no vertical or horizontal axes, only intersection lines on the area’s right border (vertical) as well as on the area’s bottom border (horizontal)
- the image exported from the echocardiography device software must be in format “JPG” “PNG”, “TIF” or “BMP”

It should be noted, that the quality of calculate aortic wall edges as well as the accuracy of derived parameter set mainly depends on the quality of loaded TTE images.

2 Software Launch

2.1 System Requirements

Since *AortUs* is distributed as independent executable, no installation of internally used libraries is required. Although, the execution of this software demands (minimum) system requirements:

- Operating system: Windows 7, 32bit/64bit
- Free disk space: > 300MB

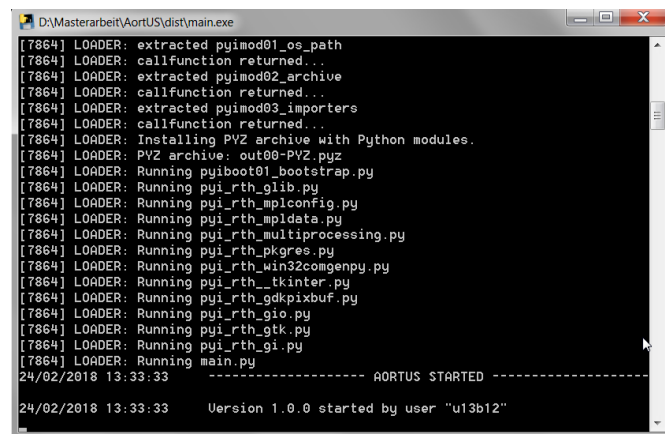
Note: If the main drive of the OS does not provide enough free space, the bootloader will close at startup without running *AortUs*.

2.2 Logfile

In addition to the printed information within the *Loginfo Box* of the main window, each usage of *AortUs* will append this log info content to a temporary Windows user related log file “aortus.log.tex”, created by default in the directory “C:\<Username>\AppData\Local\Temp”. If *AortUs* hangs, terminates or produces incomprehensible results, this file will be of interest for the support by BF e.U..

2.3 Start Up

When launching the executable, a Windows command window referred as *Bootloader* will appear, showing the current state of loaded modules used in *AortUs* (see Figure 1). Depending on the PC's performance, this may take a while until finally the main window of *AortUs* appears.



```
D:\Masterarbeit\AortUS\dist\main.exe
[7864] LOADER: extracted pyimod01_os_path
[7864] LOADER: callfunction returned...
[7864] LOADER: extracted pyimod02_archive
[7864] LOADER: callfunction returned...
[7864] LOADER: extracted pyimod03_importers
[7864] LOADER: callfunction returned...
[7864] LOADER: Installing PVZ archive with Python modules.
[7864] LOADER: PVZ archive: out00-PVZ.pyz
[7864] LOADER: Running pyiboot01_bootstrap.py
[7864] LOADER: Running pyi_rth_glib.py
[7864] LOADER: Running pyi_rth_mplconfig.py
[7864] LOADER: Running pyi_rth_mpldata.py
[7864] LOADER: Running pyi_rth_multiprocessing.py
[7864] LOADER: Running pyi_rth_pkgras.py
[7864] LOADER: Running pyi_rth_win32comgenpy.py
[7864] LOADER: Running pyi_rth_tkinter.py
[7864] LOADER: Running pyi_rth_gdkpixbuf.py
[7864] LOADER: Running pyi_rth_gio.py
[7864] LOADER: Running pyi_rth_gtk.py
[7864] LOADER: Running pyi_rth_gi.py
[7864] LOADER: Running main.py
24/02/2018 13:33:33 ----- AORTUS STARTED -----
24/02/2018 13:33:33 Version 1.0.0 started by user "u13b12"
```

Figure 1: Bootloader of *AortUs*

3 User Interface

The upcoming section will describe the user interface of *AortUs* and will give an example of two loaded images for an AscAo parameter set.

3.1 Main Window

When *AortUs* is started, the bootloader will run and the main window will appear after all modules are loaded, like depicted below (Figure 2).

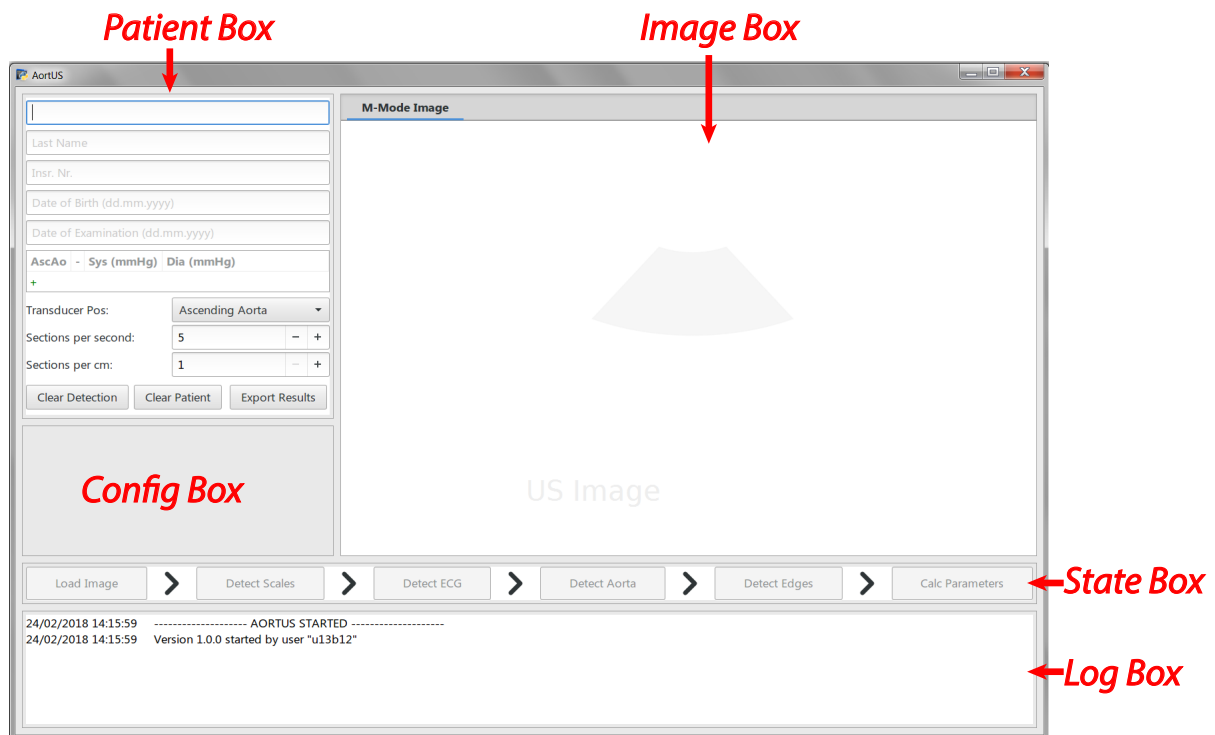


Figure 2: Main Window of *AortUs*

Note: The main window of *AortUs* will be fitted to a certain ratio of the user's main screen size. It should not be resized to fit the whole screen, although it is possible.

Note: The main window will be terminated whenever the user closes the *Bootloader*, the main window will be closed as well.

The main window can be separated into boxes, separated as follows.

3.1.1 Patient Box

Before the first image can be loaded, it is required to enter the patient's personal as well as related examination data for unique identification (first name, last name, insurance nr., date of birth, date of examination). Impossible characters within the name-fields as well as impossible entered dates will be restricted and a *Message Box* will appear. A filled example of the upper part of this box is shown in Figure 3.

Bernhard
Frohner
3065
17.02.1992
02.12.2009

Figure 3: Example filled input fields of *Patient Box*

Right below the input field for “Date of Examination”, a dynamic list to enter the systolic and diastolic blood pressure (BP) values is shown (see Figure 4). Click the leftmost symbols
+ ... to add a row and enter sys./dia. blood pressure values
- ... to remove a row

AscAo	Sys (136 mmHg)	Dia (88 mmHg)
-	132	90
-	141	85
-	135	91
+		

Figure 4: Example of three entries in dynamic BP list

Note: To enhance filling the previously describe entry fields, the user may also use the “TAB” key to switch from one field to the next one. This is not just possible for the fields “First Name”, “Last Name”, “Insurance Nr.”, “Date of Birth” and “Date of Examination”, but also to create entries in the blood pressure list. Pressing “TAB” switches to the next available field (“Sys” → “Dia”, “Dia” → new entry of “Sys”).

Entered BP values must be confirmed by pressing “Enter” or “Tab”. They are automatically averaged and shown in the column's name (i.e. “Sys (133 mmHg)”).

In addition, the examined part of the aorta must be specified within the *Transducer Pos* dropdown list.

Since most M-mode images have axes with separators to define the scaling of sections per 1cm and 1sec, the user is asked to define this scaling (default 5 sections for 1sec, 1 section for 1cm, see Figure 5).

Transducer Pos:	Ascending Aorta	▼
Sections per second:	5	- +
Sections per cm:	1	- +
Clear Detection		
Clear Patient		
Export Results		

Figure 5: Settings for *Transducer Pos* and M-mode image scaling

The bottom of the *Patient Box* offers three actions in form of buttons, described below.

<input type="button" value="Clear Detection"/>	...	Cancel the current state of detection
<input type="button" value="Clear Patient"/>	...	Equal to <i>Clear Detection</i> including a refresh of patient data
<input type="button" value="Export Results"/>	...	Export the results of the <i>Aortic Parameters</i> tab to a PDF

3.1.2 Config Box

The *Config Box* does not contain any content at startup. Whenever the user is in one of the *Edge Detection Process* states, the *Config Box* may contain additional information, interactive lists (list of triggerpoints in *Detect ECG* state) or guidance through semiautomatic steps (manual mode of *Detect Scales*, adjustment of aortic boundaries in *Detect Aorta*, adjustment of found edges after *Detect Edges*).

3.1.3 Image Box

The *Image Box* contains the illustration shown in Figure 6 at startup and is updated in each step of the *Edge Detection Process*. The loaded M-mode image including highlighted image content will be displayed in the tab *M-Mode Image* of this box.

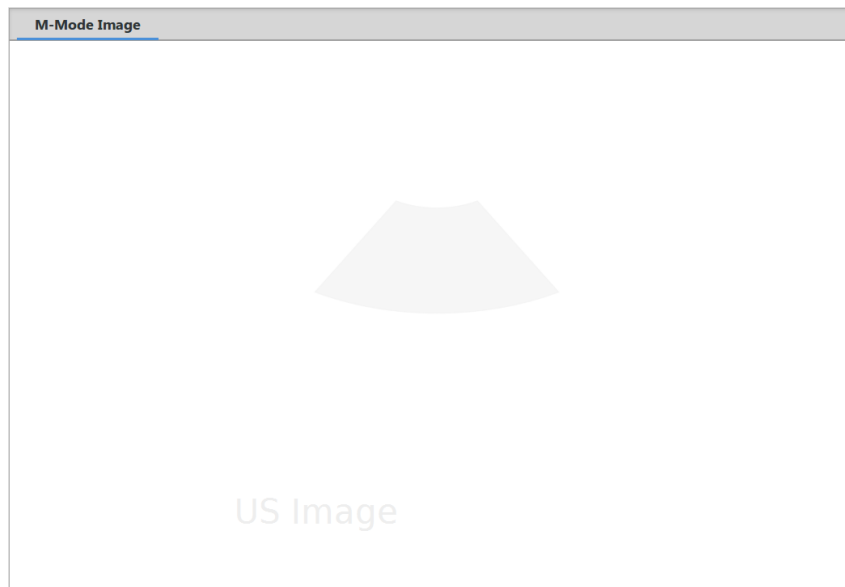


Figure 6: Default appearance of *Image Box*

When the user reaches the *Calc Parameters* state, a second tab named *Aortic Parameters* will appear, containing calculated single- and averaged aortic diameter courses as well as the parameters described in section 1.3.

Note: Whenever *AortUs* is in any interactive mode (i.e. to adjust triggerpoints), interaction referred to as “click” or “click-and-drag” event is related to events of the **left mouse button**.

3.1.4 State Box

The *State Box* guides the user through the *Edge Detection Process*. Each state is represented by one button that can be triggered by click event, whenever *AortUs* has all necessary information to reach the next step.



Figure 7: Default appearance of *State Box*

3.1.5 Log Box

The *Log Box* provides useful information of successful or failing events of *AortUs*. It is embedded within a scrollable window, automatically seeking the latest entries.

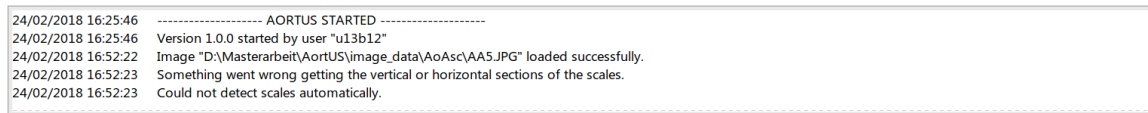


Figure 8: Example of *Log Box*

4 Usage Example

The following section will give a usage example two M-mode images of the AscAo of the same person, processed by *AortUs*.

1. When *AortUs* is started, the *Bootloader* command window opens and the main window shown in Figure 9 will appear when all modules are loaded successfully.

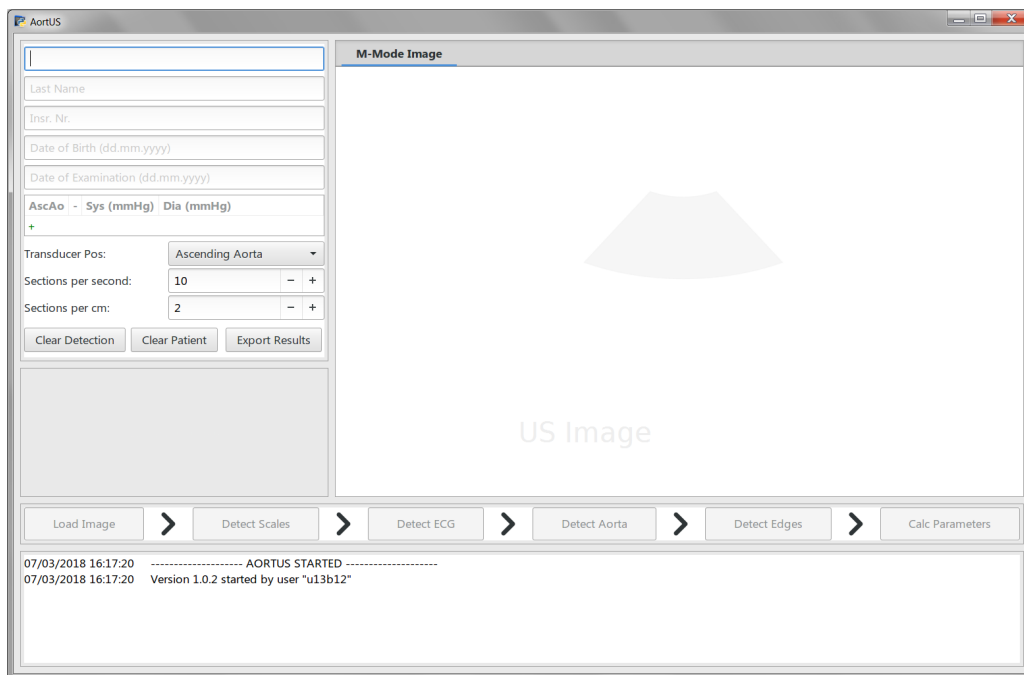


Figure 9: Default appearance of *AortUs* after startup

On success, the *Log Box* will give information of the current software version as well as the applying user.

Note: The personal data of the patient as well as at least one BP-set must be entered before the *Edge Detection Process* can be started by the *State Box* buttons.

2. The required first name, last name, insurance number of the patient must be defined, as well as date of birth and of examination in format <DD>.<MM>.<YYYY>, like shown in Figure 10. In addition, either the element *Ascending Aorta* or *Descending Aorta* in the *Transducer Pos* list must be adapted to the type of image.
3. When all input fields are set, the first image can be loaded by clicking the *Load Image* button in the *State Box*. A dialogue presented in Figure 11 will appear and the first image to load can be selected, and confirmed by clicking "Open". On success, the selected image will appear within the *Image Box* like shown in Figure

12 and the calibration of *Sections per second* and *Sections per cm* must be adapted if necessary. For the image shown in Figure 12, the settings need to be corrected to 5 vertical sections encoding one second and 1 horizontal section encoding one centimeter.

Note: The naming of labels and buttons within the appearing dialogue might vary, depending on the language settings of the OS.

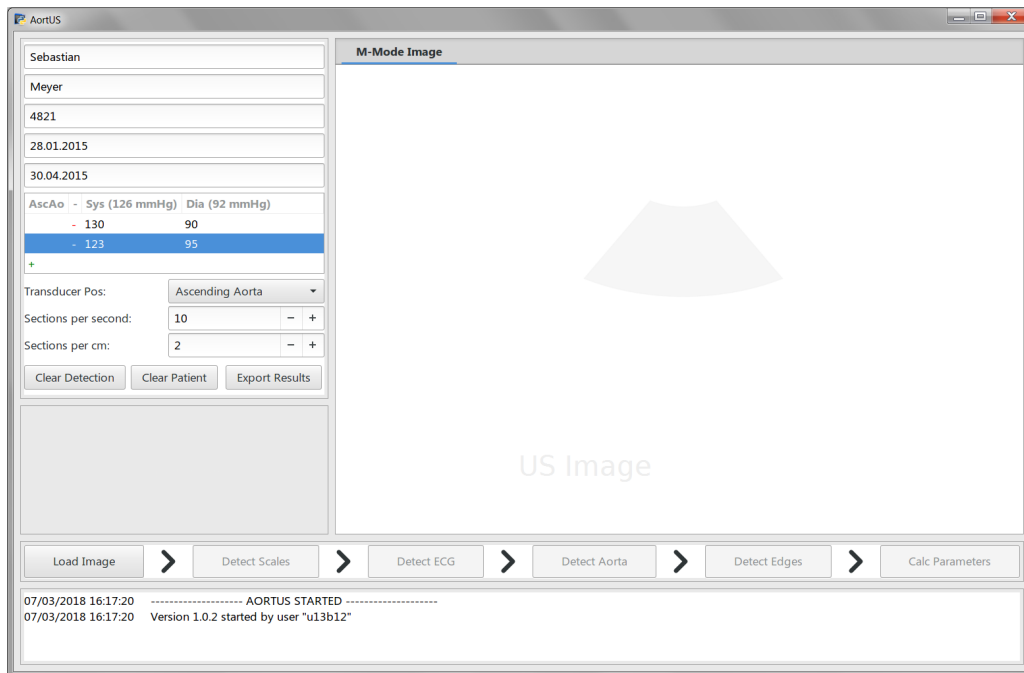


Figure 10: Entered patient data

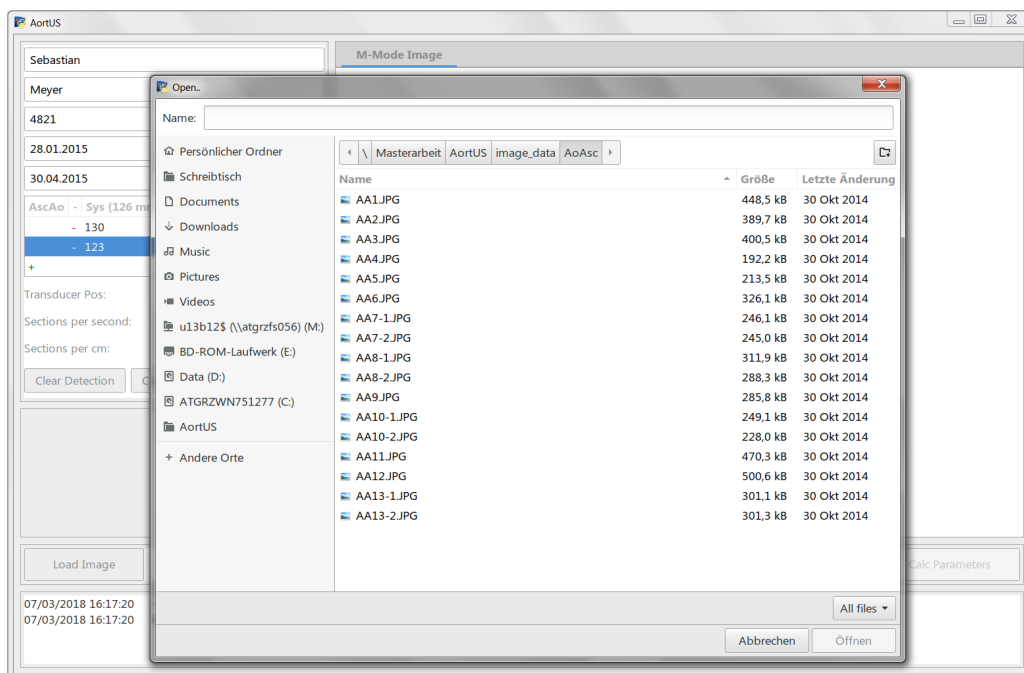


Figure 11: Load image dialogue

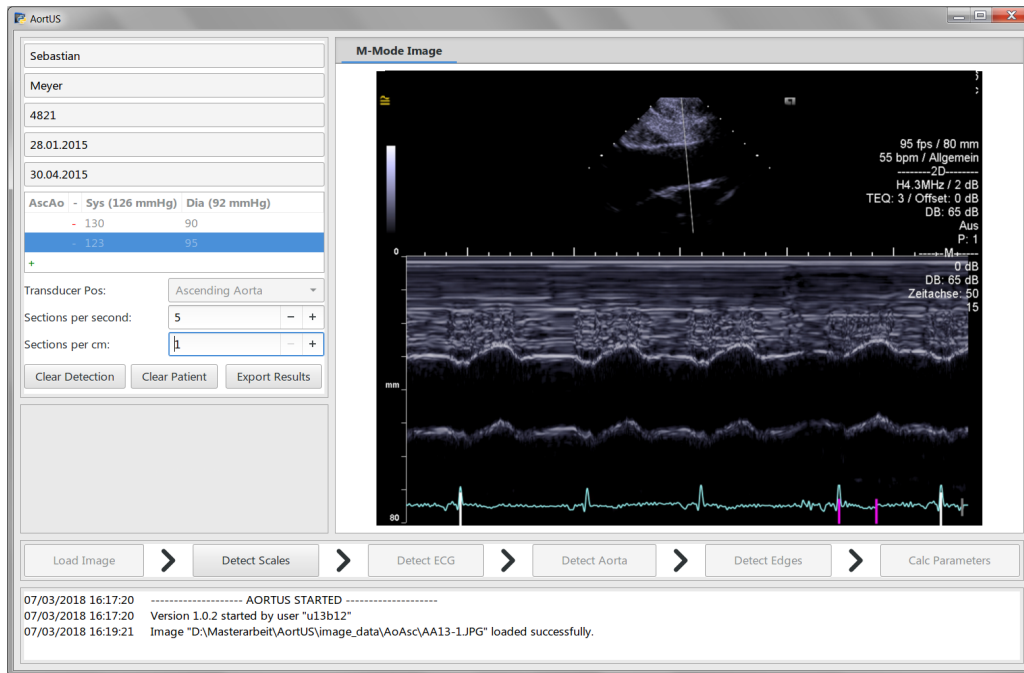


Figure 12: Image loaded

Warning: If necessary, the definition of the correct number of *Sections per cm* and *Sections per second* **must be adapted at this point**. If these definitions are incorrect, *AortUs* may succeed till the *Edge Detection Process*, although the parameters and curves calculated are incorrect!

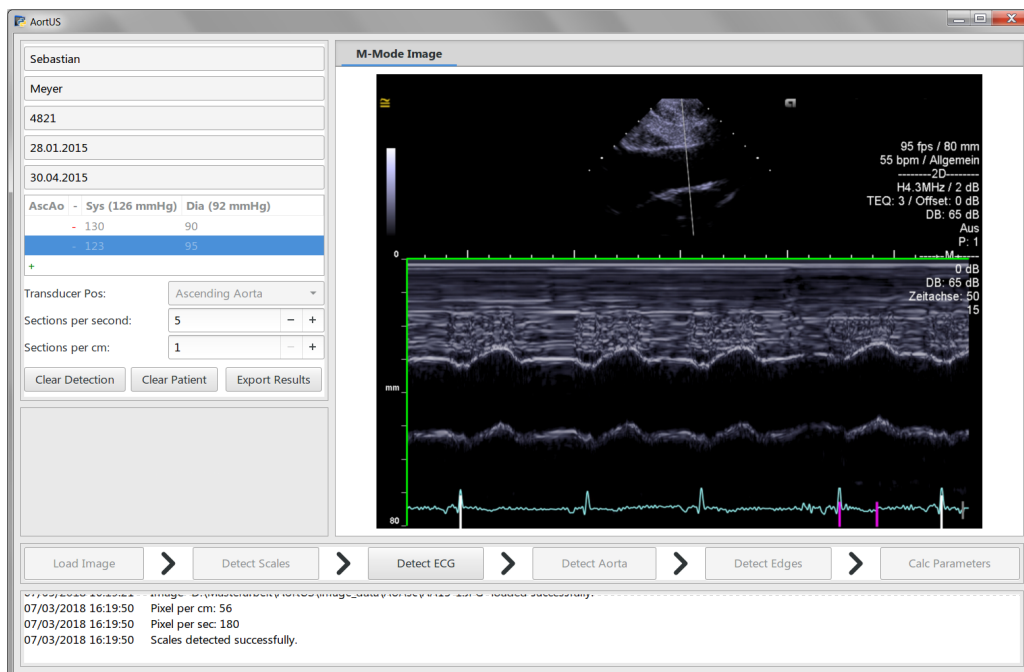


Figure 13: Scales detected automatically

- The next step for *AortUs* is to determine the number of pixels for 1cm and 1sec, respec-

tively. This action can be started clicking *Detect Scales*.

On success, the found main axes within the image will be highlighted from the top left of the M-mode's area in green and the found pixel resolution will be printed within the *Log Box*, shown by Figure 13.

In some cases the main axes cannot be detected automatically, i.e. if the axes and their neighbouring ROI content have similar brightness values. For this particular case, the user can define the region of interest as well as the pixels used for *1sec* and *1cm* manually like described in section 5.2.

5. When clicking *Detect ECG*, *AortUs* will try to find a coloured ECG tracing including striking R-peaks in the image.

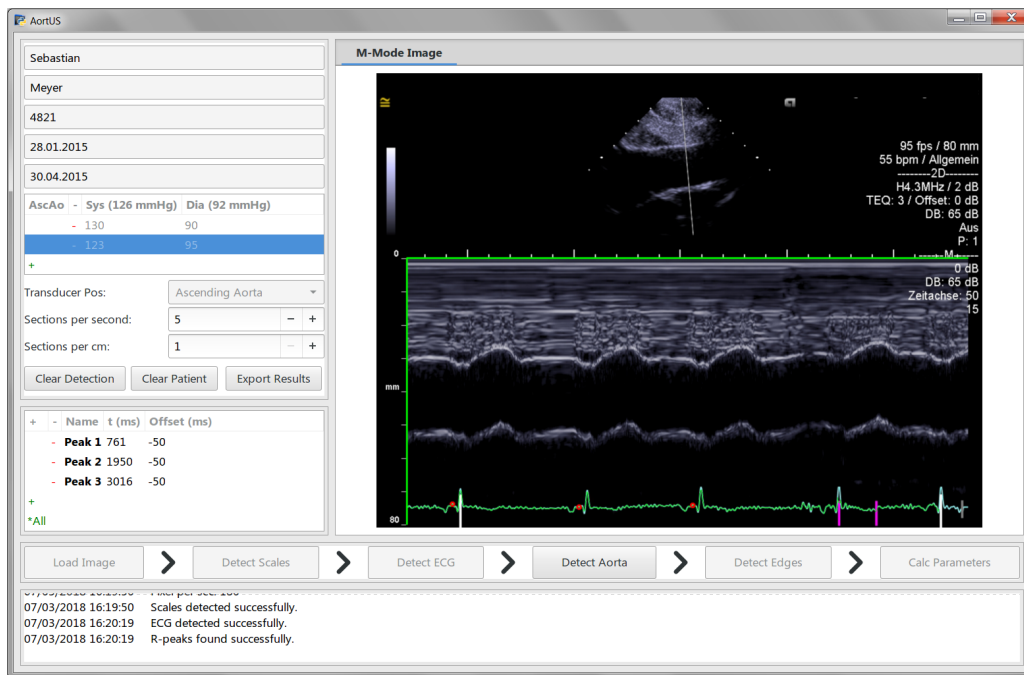


Figure 14: Detect ECG including original locations of found R-peaks

The updated main window appearance can be obtained by Figure 14, including the found ECG tracing highlighted in green, as well as the R-peaks indicated by red circles. The positions of these circles, also referred to as *triggerpoints*, define the start- and endpoints of each diameter-period used for averaging.

Note: Even though vertical cursors cover the ECG in the loaded image, *AortUs* is able to detect the underlying tracing. However, it is not recommended to use vertical cursor in loaded M-mode images.

In this case, *AortUs* is not able to find the image related peaks precisely, like shown in Figure 14. Therefore, the user has two options for manipulation:

- click-and-drag on the red circles to modify their position along the found ECG tracing
- manually remove, add or adjust positions of R-peaks by the shown list inside the *Config Box*.

The list is embedded within a scrollable window and itemises found triggerpoints including their position in time domain, as well as an adjustable offset to this position. This offset is $-50ms$ by default and can be changed by simply clicking into the field of a triggerpoint item. In addition, the list offers the following functionality:

- + ... add a triggerpoint
- ... remove a triggerpoint
- *All ... set \pm offset of all triggerpoints to the offset of the selected one in ms

For the image shown, **Peak 4** and **Peak 5** needed to be inserted (+) and dragged $\approx 50ms$ prior to their missed R-peaks, comprehensible by Figure 15.

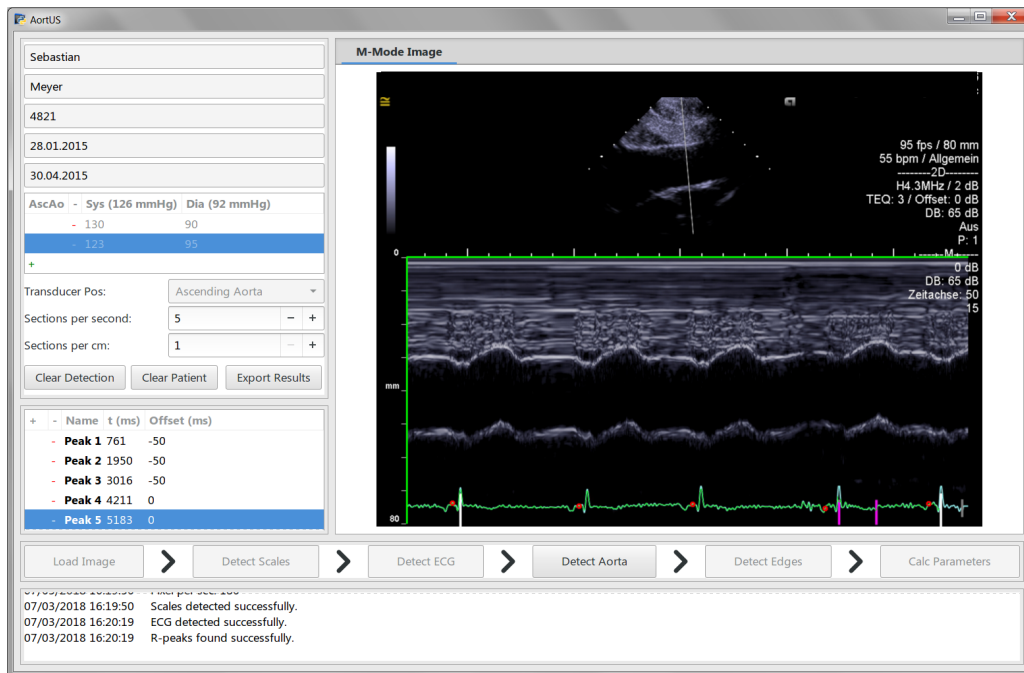


Figure 15: Adjusted triggerpoints on found ECG

Note: If a triggerpoint's offset exceeds the ECG tracing, the user will be notified and the offset is set to 0.

Warning: Incorrect positioning of triggerpoints as well as overlooked ones may lead to in incorrect resulting parameter.

6. When all triggerpoints are defined correctly, the user should move on to the *Detect Aorta* state. This function aims to scale down the ROI vertically from the whole M-mode region to the minimal image area arrogated by recorded aortic walls. This is symbolised by limiting the ROI to the area between two yellow lines, shown in Figure 16. For images with high wall contrast, the automatically suggested positions of these boundaries closely approximate the outer borders of aortic walls. Although, it is also possible to adjust the positions of these boundary lines close to the outermost points of the aortic wall image, again by a mouse click-and-drag action. In case of the image shown in Figure 16, the detected borders suit the whole aortic representation and can therefore be accepted by clicking the *Confirm* button inside the *Config Box*.

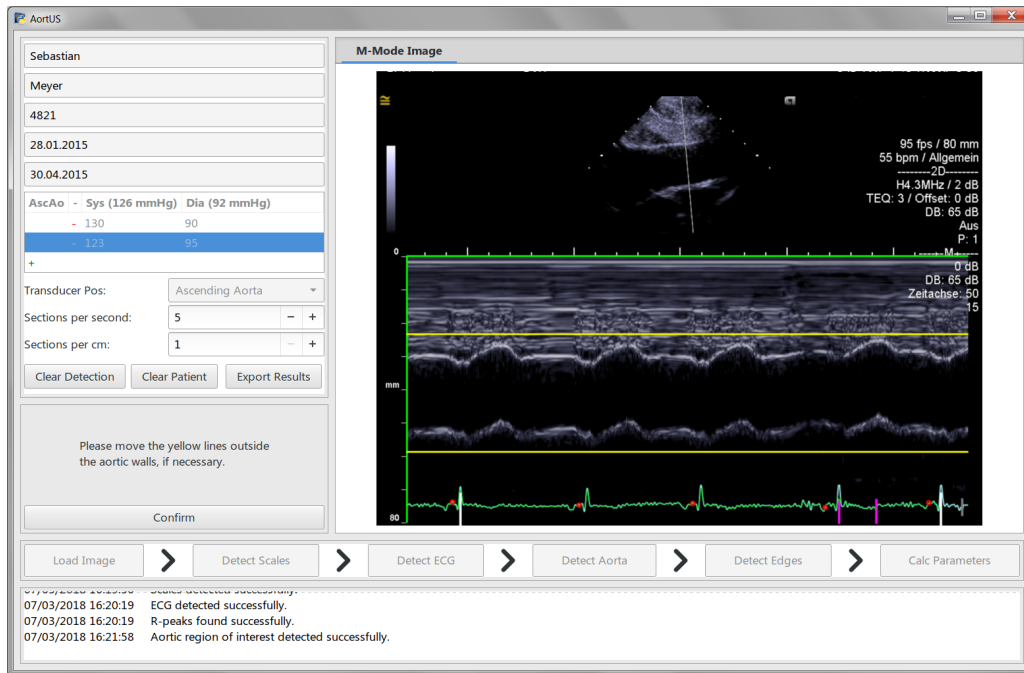


Figure 16: Scale down ROI vertically to aortic walls

7. The main edge detecting function is the most performance demanding step during a standard procedure of *AortUs*. When clicking *Detect Edges*, it might take a few seconds to determine the leading edges, shown in Figure 17.

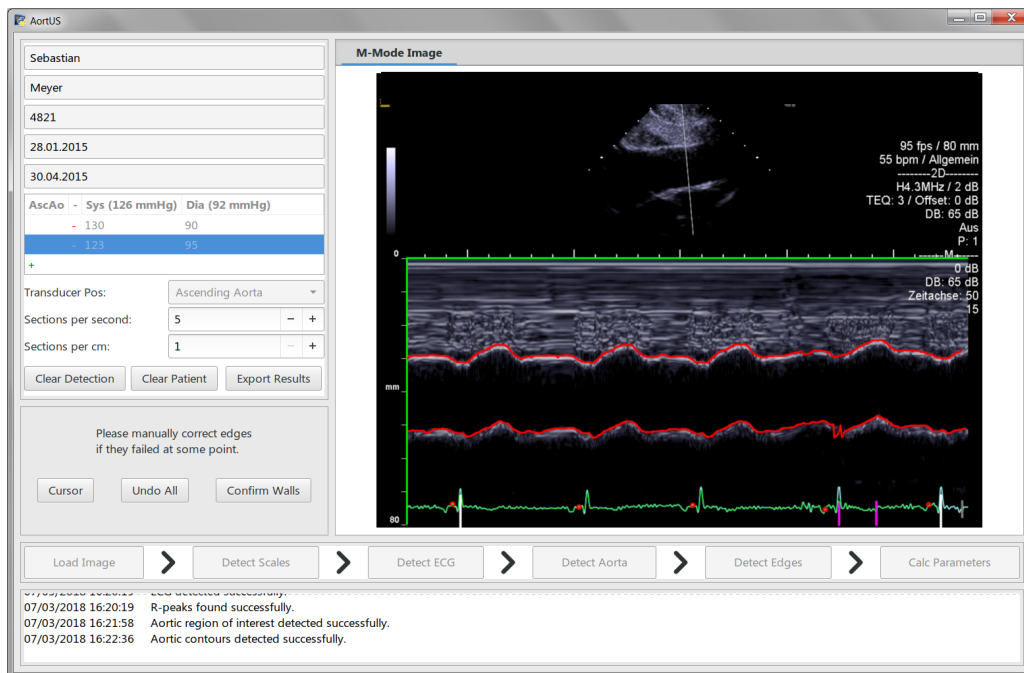


Figure 17: Automatically detected aortic walls

Since the internal edge detection algorithm cannot handle all sudden brightness interruptions of the aortic walls, the user is asked to mend these outliers manually. For this interaction, three mode buttons are shown within the *Config Box*:

- ... Activate manual cursor mode
- ... Reset edges to initial results from *Detect Edges*
- ... Proceed to *Calc Parameters*

When the manual cursor mode is activated, the user can click-and-drag on each of the red shown circles. The positions of the cursor's nearest edgepoints will be reset to the positions of cursor's pathway, without splitting the edge. For the edges shown in Figure 17, the region around the fourth triggerpoint of the transducer-far edge is adjusted by first clicking *Cursor* and then correcting the local transducer-far edge around the fourth triggerpoint from left to right. The resulting image can be obtained by Figure 18.

Warning: Inaccurate or distorted edges may deform the resulting average aortic diameter curves. Therefore it is of great necessity to pay attention to precise edges.

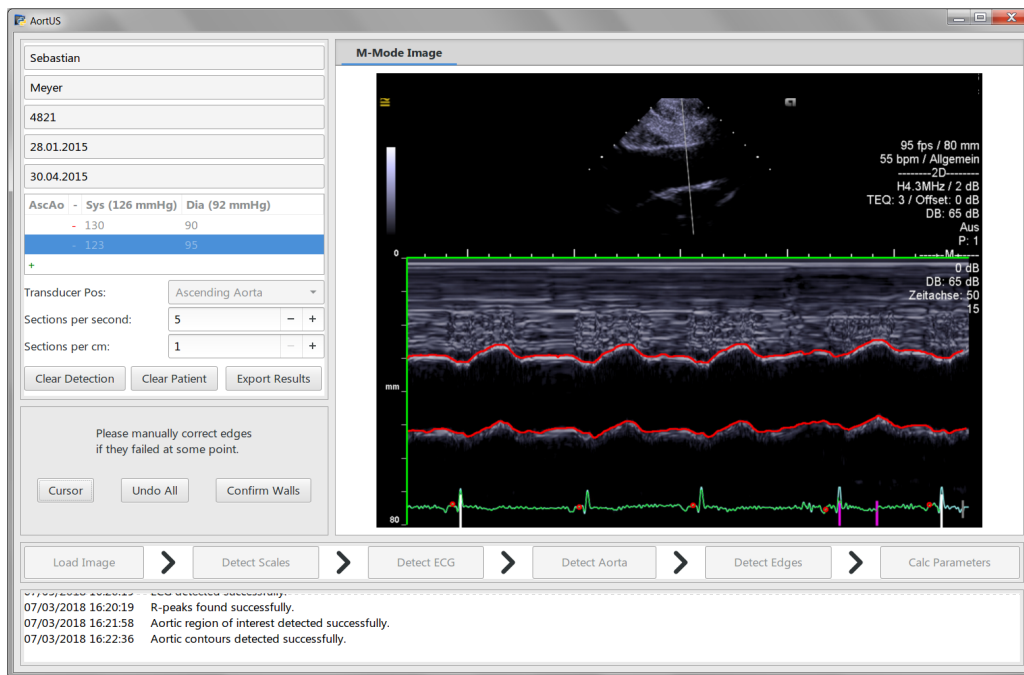


Figure 18: Adjusted aortic wall edges

After the edge is corrected, the user may proceed to the final *Calc Parameters* state by clicking the *Confirm Walls* button.

8. Finally *AortUs* will create a second tab within the *Image Box* named *Aortic Parameters*, containing two types of plots, the calculated parameters as well as a list of curves that contribute to the shown plots. Each curve shown is extracted from the double-edges of previous step only, if both edges fully cover each whole heart cycle. The left plot named *Single Aortic Diameter Courses* shows all diameter curves that could be extracted, named from *course 0* to *course <n-1>* for edges from “left to right” where *n* denotes the number of diameter curves.

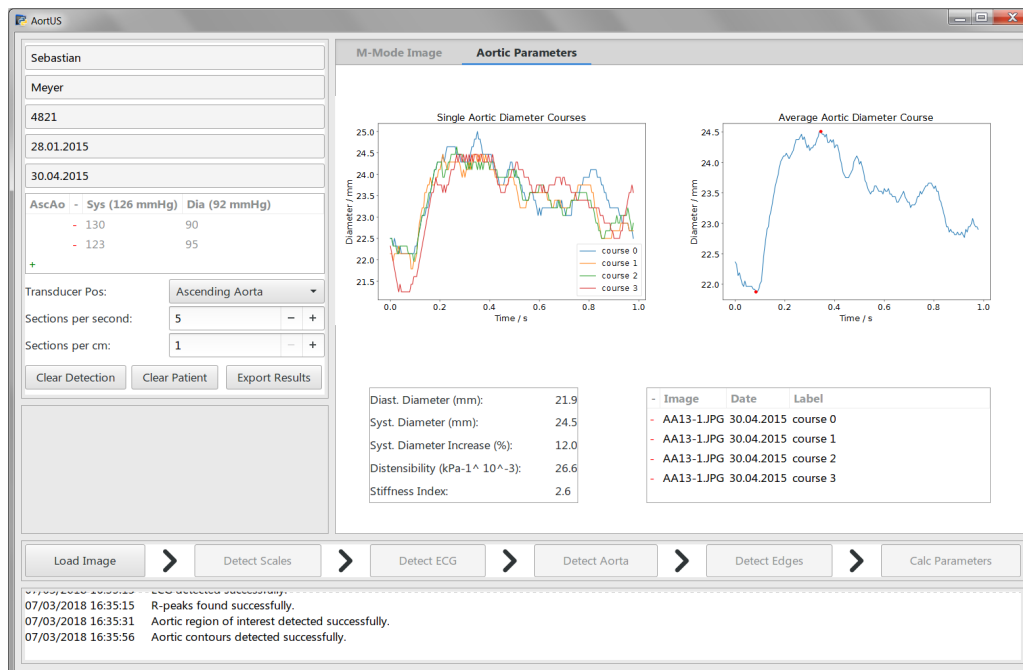


Figure 19: Determined curves including the estimated parameters

The right plot named *Average Aortic Diameter Course* shows the average diameter at each timepoint, in which the determined *Diast. Diameter* as well as the *Syst. Diameter* are marked by a red dot.

Next to the calculated parameters, a list of the curves contributing to the calculation is shown, including the name of the image(s) used and the date of examination. Similar to the functionality of the BP list, the “-” button can be used to remove a certain curve from the calculation, which might be useful to remove outliers. Whenever this is done, all plots and parameters are updated to the current list of curves.

Note: The *Diast. Diameter* and *Syst. Diameter* are determined by finding the local Minima within $[0, 250]ms$ and the Maxima within $[200, 500]ms$ after the averaged origin of diameter curves. Therefore it is importance to define these triggerpoints accurately, so that the incline of each single curve starts approximately at the same point.

Note: The established parameters can serve as a **first indicator for impaired bioelasticity** of the aorta. These values are affected by radial arterial wall changes only and can therefore not fully characterise the aortic behaviour since artery walls are anisotropic, viscoelastic and have a non-linear pressure-radius relationship.

9. The attentive user will notice, that the *Load Image* button can be used again at this *Calc Parameter* state. For a second M-mode image, steps 3 to 8 can be repeated. In this example, the image AA13-2.JPG is loaded and the combined results are shown in Figure 20.

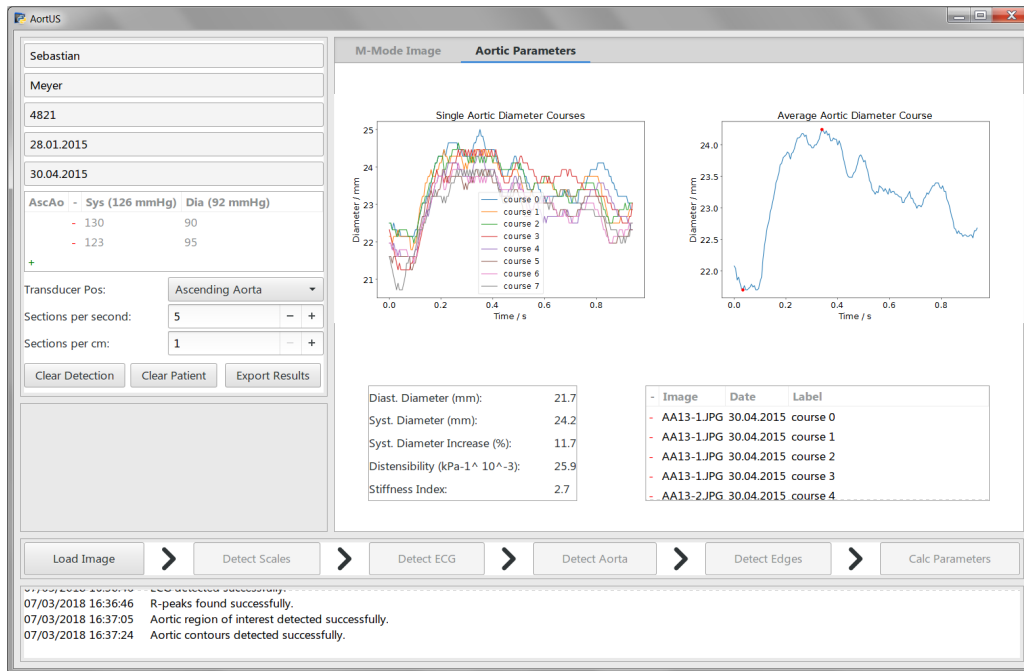


Figure 20: Combined estimated parameter of image AA13-1.JPG and AA13-2.JPG

10. In order to append the extracted information i.e. to a patient record, the user may want to export or print the shown results. At this point, the shown plots as well as the estimated parameters can be exported to a PDF by clicking the *Export Results* button.

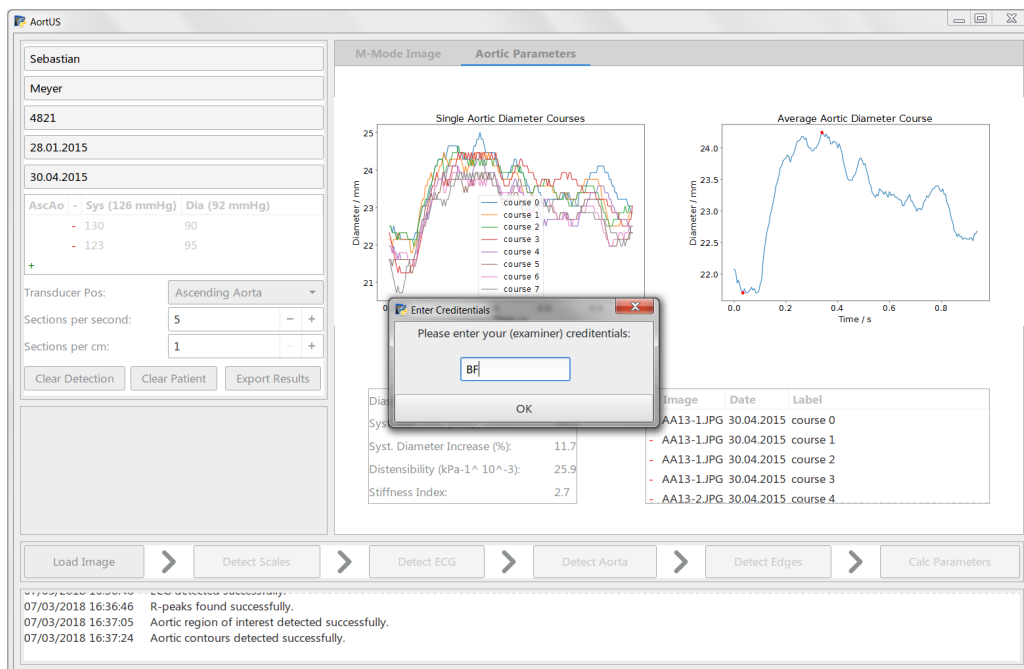


Figure 21: Dialogue to set the examiner's username

A dialogue shown in Figure 21 will open and ask for the user's credentials. By default, the username of the OS's user will be proposed.

When submitted, a second dialogue will open to select a location for the created PDF, shown in Figure 22.

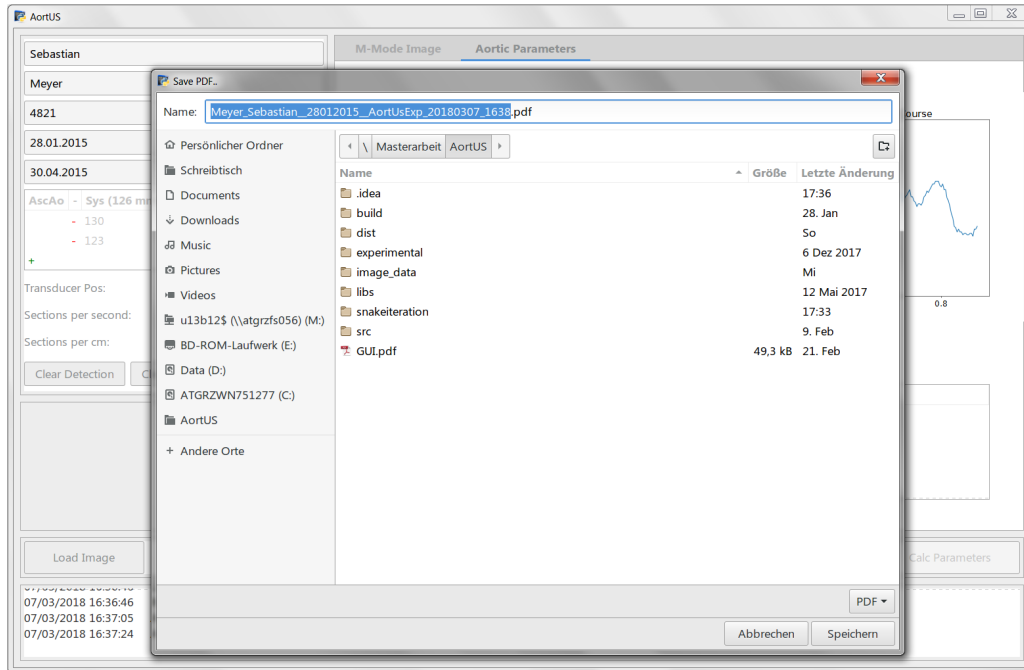


Figure 22: Dialogue to set location of PDF export file

The content of this PDF is self explanatory and includes all used M-mode images including highlighted edge curves, as well as patient related data. The content of this final PDF can be obtained by Figure 23.

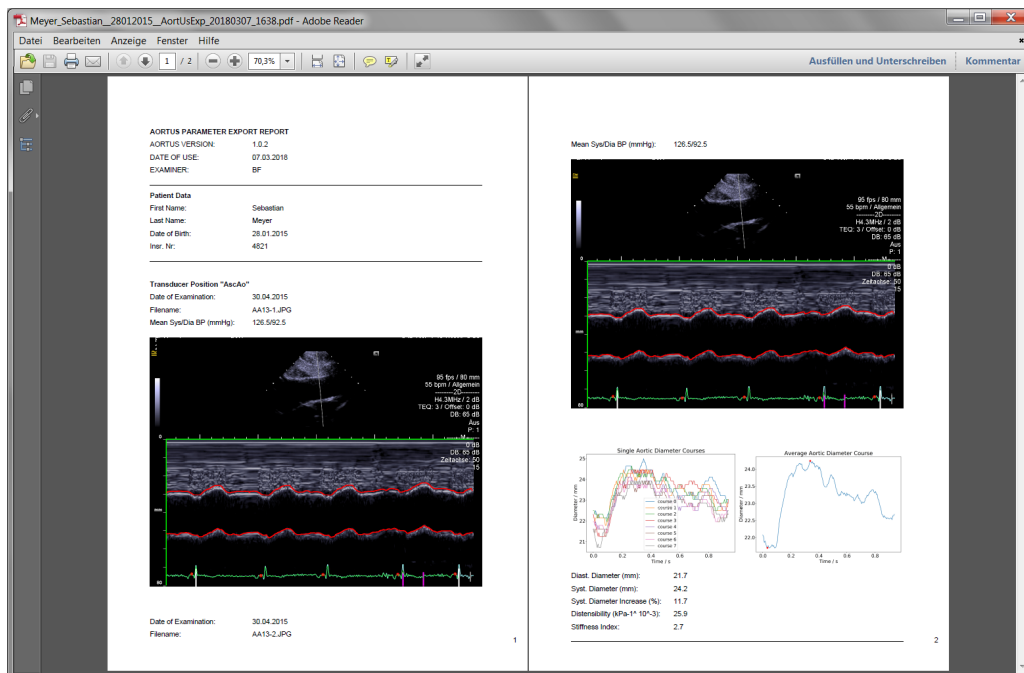


Figure 23: Created PDF

5 Manual Interventions and Notifications

In general, *AortUs* will notify the user about current events by the *Log Box*. In some cases, the user needs to change input data or items of the current state. At this point, a *Message Box* is shown and describes as below.

5.1 Message Box

This popup window will appear i.e. when relevant image content could not be extracted or one of the integrated plausibility checks did not succeed. An example of this window is shown in Figure 24.

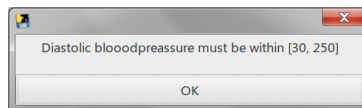


Figure 24: Example for warning popup message

This window will appear, when

- **General** - entered BP is not a valid integer
- **General** - a diastolic exceeds systolic BP for one list-item of blood pressure list
- **General** - systolic BP exceeds plausible limits $[50, 300]mmHg$
- **General** - diastolic BP exceeds plausible limits $[30, 250]mmHg$
- **Load Image** - the input image cannot be opened
- **Detect Scales** - scale axes not found automatically and manual scale detection is active
- **Detect ECG** - no or less than 3 triggerpoints were found
- **Detect ECG** - entered *Offset* for triggerpoint(s) exceeds the time domain
- **Detect Edges** - no edges could be extracted due to bad image quality. At this point the only possible solution is to call *Clear Detection* and load another image.

5.2 Manual Scale Detection

In some cases, however, the automatic scale detection fails when clicking *Detect Scales*. A *Message Box* like described in 5.1 will appear, and the *Manual Scale Detection* is activated automatically. This process will guide the user through a manual definition of the M-mode ROI, the scaling for *1sec* and for *1cm* respectively.

The following example should help to understand the process of *Manual Scale Detection*.

1. When clicking *Detect Scales*, a notification will appear to inform the user about the launch of manual mode, like shown in Figure 25.

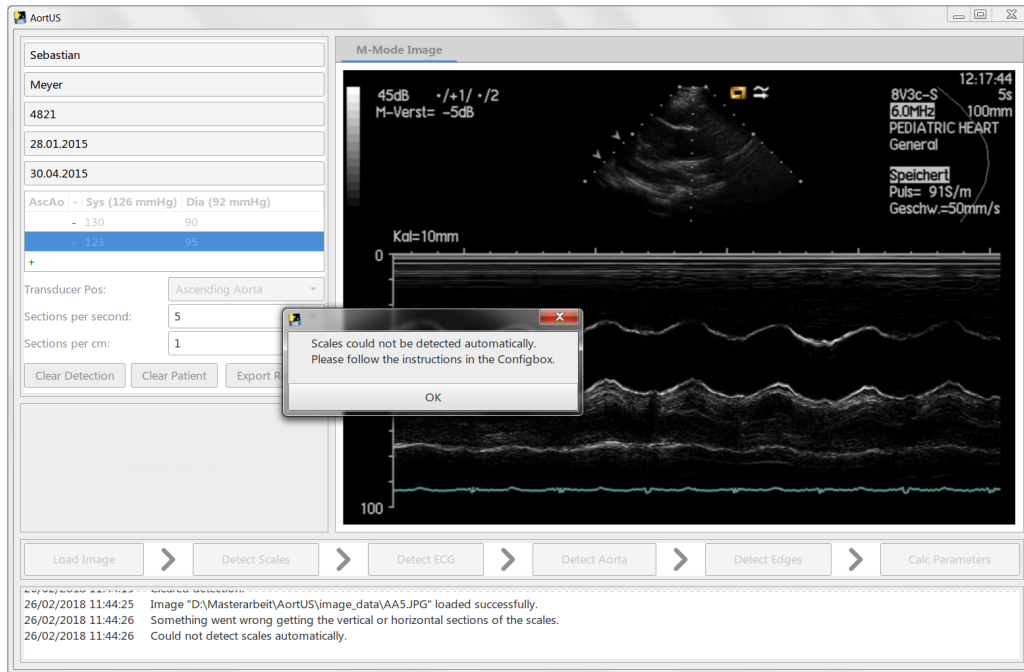


Figure 25: Start of manual scale detection after user clicked *Detect Scales*

The first step defines the actual M-mode image ROI. To achieve this, it is recommended to click at the very left top of the M-mode image content and drag down the appearing green rectangle to the bottommost right end of the image, like shown in Figure 26.



Figure 26: Manual definition of M-mode ROI by click-and-drag

When the left mouse button is released, the defined ROI will be denoted by highlighted green vertical- and horizontal axes, shown in Figure 27.

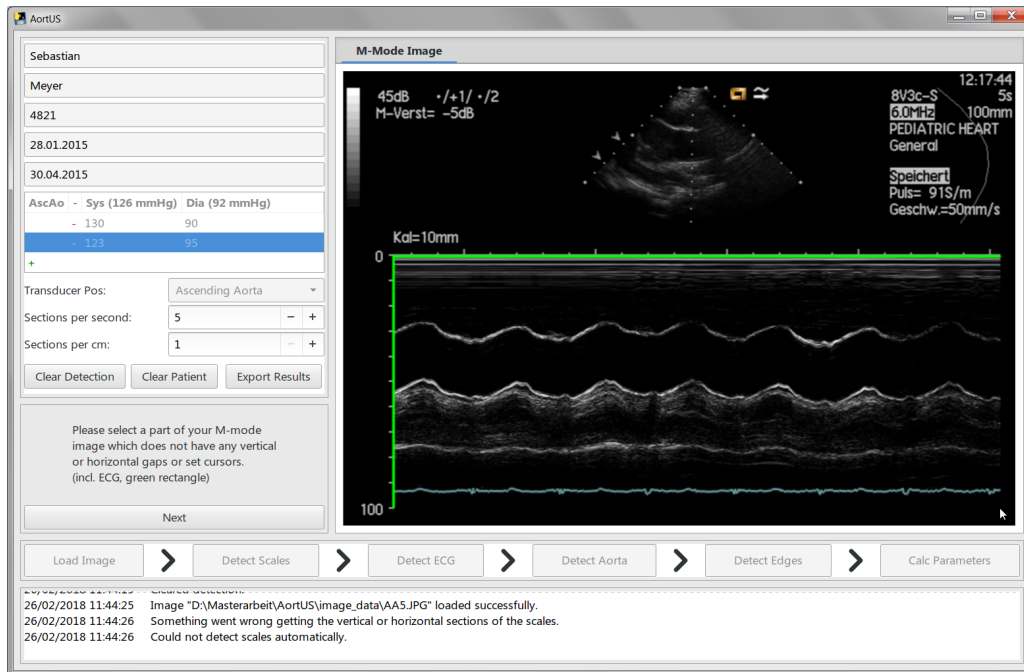


Figure 27: Defined M-mode ROI

This definition of the M-mode ROI can be repeated until the user is satisfied with the axes positions. If so, a click on the *Next* button within the *Config Box* switches to the time-definition step.

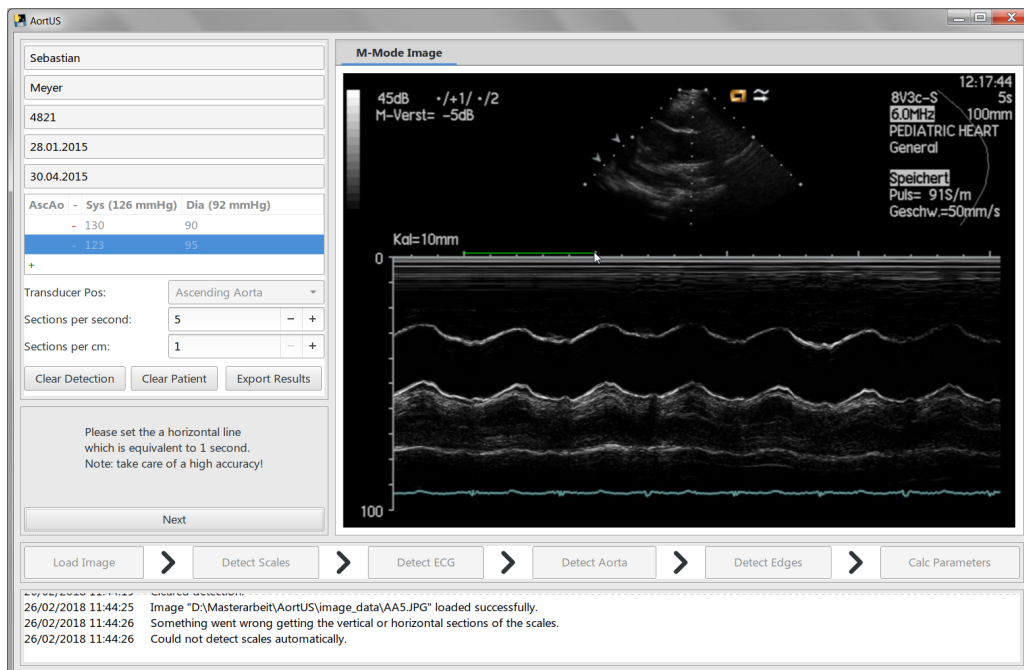


Figure 28: Manual definition of 1sec

2. To define the pixels encoding 1sec , the user is asked to define a horizontal line with the length of 1sec by (again) click-and-drag. It is recommended to define this line based on the visible time axis, like shown in Figure 28. Similar to the definition of the M-mode ROI, this step can be repeated and confirmed by clicking *Next*.
3. Equal to the time-definition step, the user is asked to define a vertical line with length of 1cm . The resulting definition is shown in Figure 29.

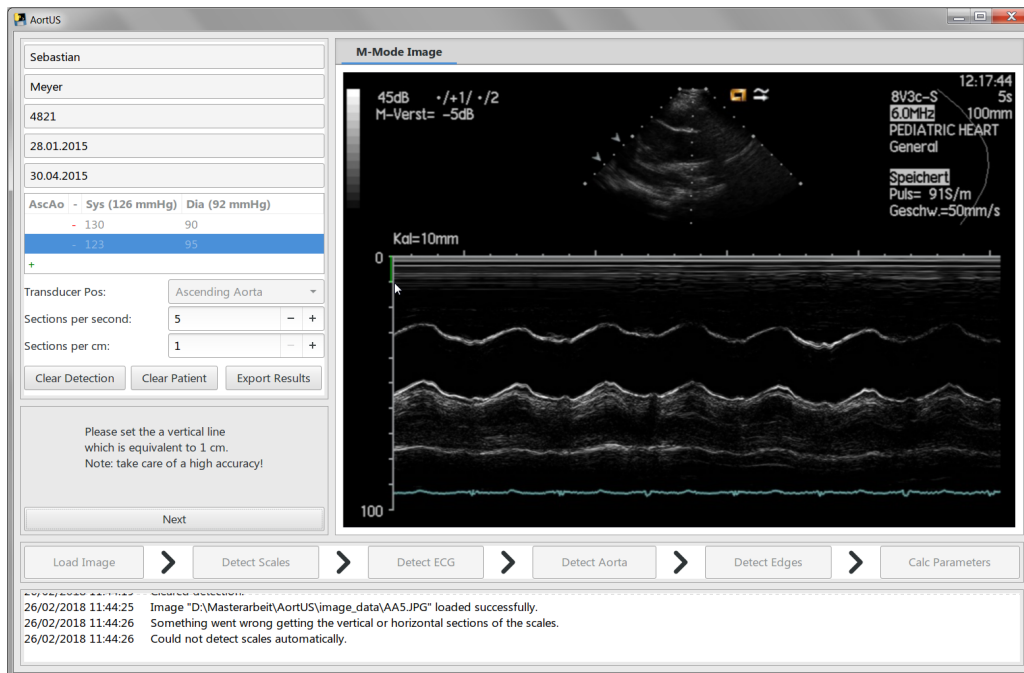


Figure 29: Manual definition of 1cm

When this is done (*Next*), the user can finish the *Manual Scale Detection* process and *AortUs* enables to trigger the next state (*Detect ECG*).

6 Maintenance and Feedback

In order to improve *AortUs*'s stability and integrate performant functionality, this software is updated by BF e.U.. A change log for these versions can be obtained by the appended Software Version History on page 27.

To get in the mailing list for updates as well as for further improvement suggestions or information, BF e.U. is looking forward to get in touch on the contact details below.

BF e.U.
Uhlandgasse 5
8010 Graz
A-Austria
frohner@student.tugraz.at

Software Version History

Version & Date	Author	Description
1.0.0 25.02.2018	Bernhard Frohner	initial version
1.0.1 27.02.2018	Bernhard Frohner	<p>Bugs fixed</p> <ul style="list-style-type: none"> • click on <i>Export Results</i>, led to error message when no patient is loaded • complete reimplementation of ECG-detection in order to be able to process images of LKH Graz's Philips ultrasound device M-mode images <p>Features added</p> <ul style="list-style-type: none"> • click on <i>Clear Detection</i>, also produces a <i>Log</i> if no image is loaded. • <i>Detect Scales</i> function, extended by a detection function to also enable auto detection of axes for files provided by the department of pediatric cardiology LKH Graz (<code>QUICKNDIRTY_getScalesAuto()</code>-method) • implemented accessibility and auto-appending functionality of blood pressures list by pressing "TAB" key <p>Calibration values changed</p> <ul style="list-style-type: none"> • <i>Detect ECG</i>, adapted HSV-filtering thresholds ("Value" threshold changes to 0.2) • <i>Detect ECG</i>, change max. heart rate to <i>220bpm</i> • ECG triggerpoints automatically offsetted to $-50ms$ • implemented thresholds to find local <i>diastolic minimal diameters</i> ($[0, 250]ms$ from triggerpoint) and local <i>systolic maximal diameters</i> $[200, 500]ms$ from triggerpoint)
1.0.2 04.03.2018	Bernhard Frohner	<p>Bugs fixed</p> <ul style="list-style-type: none"> • varying font size in exported PDF fixed <p>Features added</p> <ul style="list-style-type: none"> • multiple examinations for different transducer positions exportable for one patient

1.0.3 open	Bernhard Frohner	<p>Bugs fixed</p> <ul style="list-style-type: none">• None <p>Features added</p> <ul style="list-style-type: none">• integrated auto-break for iterative snake-algorithm, stops if the the last 10 differences of averaged 4 levelsets is lower than a defined coefficient of variation $COV_THRESH = 0.1$
------------	------------------	--

Software Documentation

for project AortUs

BERNHARD FROHNER

Version 1.0.0

Table of Content

1	Introduction	1
2	Algebra module	3
3	Aorta module	4
4	Calibration module	9
5	Configuration module	13
6	Ecg module	16
7	Edges module	18
8	Examination module	28
9	GUI module	29
10	Logging module	38
11	Morphsnakes module	39
12	MySubclasses module	43
13	Patient module	54
14	Rpeaks module	56
15	Scale module	60
16	UsFile module	64
17	UsImage module	65

1 Introduction

This Software Documentation should help the programmer to understand the functionality of the *AortUs* implementation in Python 2.7.13. Each of the following sections will describe one module implemented and used for *AortUs*. These modules integrate classes, that are described by their

- general class definition
- member variables
- methods

It should be noted, that most datatype definitions are integrated as hyperlink, so that the reader can easily access a more elaborated explanation. If no parameter or return type is documented for a variable of the described method, the datatype is unknown or the function is void. Member variables are always listed including their default value (i.e. *fullname_ = None*).

2 Algebra module

`class Algebra.Algebra`

This class implements static algebraic math functions as well as signal processing methods.

`static FWHM(X, Y, x_peak)`

Find full width half maximum of a passed peak. If half maximum on one side of the peak is not in the vector any more, the first/last element of the vector will be taken for calculation

Parameters

- `X` (`ndarray`, `Nx1`) – x-values of function
- `Y` (`ndarray`, `Nx1`) – y-values of function
- `x_peak` (`int64`) – x-position of peak

Returns calculated full width half maximum

Return type `int64`

`static butterBandpassFilter(data, lowcut, highcut, f_samp, order=5)`

Bandpass filtering the input data with zero phase. If `highcut` is above the nyquist frequency, a lowpass filter is used instead

Parameters

- `data` (`ndarray`, `Nx1`) – input data vector
- `lowcut` (`float`) – lower cutoff-frequency
- `highcut` (`float`) – upper cutoff frequency
- `f_samp` (`int`) – sampling frequency
- `order` (`int`) – order of bandpass filter

Returns bandpass filtered input vector

Return type `ndarray`, `Nx1`

`static butterLowpassFilter(data, f_cut, f_samp, order=5)`

Lowpass filtering the input data with zero phase.

Parameters

- `data` (`ndarray`, `Nx1`) – input data vector
- `f_cut` (`float`) – cutoff-frequency
- `f_samp` (`int`) – sampling frequency
- `order` (`int`) – order of lowpass filter

Returns lowpass filtered input vector

Return type ndarray, Nx1

```
static rejectOutliers(data, m=10.0)
```

Function to reject outliers which deviate too much from the median

Parameters

- *data* (ndarray) – input data vector
- *m* (*float*) – deviation factor for rejection

Returns outlier filtered input vector

Return type ndarray

```
static splitConsecutiveData(data, stepsize=1)
```

Splits an array or list of data by the stepsize of it's content.

Parameters

- *data* (list, ndarray) – array/list to be split
- *stepsize* (*int*) – Deviation that two consecutive values in *data* can have in order to be splitted. If *stepsize* is 1, i.e. the list [1,2,3,5,6,7] would be splitted into [[1,2,3][4,5,6]].

Returns splitted array

3 Aorta module

```
class Aorta.Aorta(updateFcn, img_shown, img_obj, mmode_obj, mmode_dx,  
                 mmode_dy, rpeaks, px_per_cm, px_per_sec)
```

This class describes the ultrasound M-mode-image related properties of the aorta.

It handles

- the first estimation where the aorta could be (`getAorticBoundaries()`)
- the automatic extraction process to get the edges (`getEdges()`)
- the callback functions to let the user adapt the found edges

Standard constructor

Parameters

- *updateFcn* (*function*) – function to call after the US-image has been updated
- *img_shown* (ndarray, NxMx3) – currently shown US-image

- `img_obj` (`ndarray`, `NxMx3`) – original image used as ROI without markups on the image
- `mmode_obj` (`ndarray`, `NxM`) – M-Mode image
- `mmode_dx` (`int`) – M-mode image offset within overall us-image object in x-direction
- `mmode_dy` (`int`) – M-mode image offset within overall us-image object in y-direction
- `rpeaks` (`Rpeaks`) – Rpeak object
- `px_per_cm` (`int64`) – nr. of pixels per cm
- `px_per_sec` (`int64`) – nr. of pixels per second

`adjustEdgesManually(config_box_main, btn_calculate)`

Set the GUI's `:obj:~GUI.GUI.config_box_main` content in order to provide three functions:

- adjusting current edges (activate cursor by click on “Cursor” pushbutton)
- undo manipulated edges to initial found edge-curves (click on “Undo” button)
- confirm the currently shown edge-curves in order to calculate the parameters afterwards

Parameters

- `config_box_main` (`VBox`) – configuration box where state-related content is shown
- `btn_calculate` (`Button`) – button of next state to enable when the user confirms the shown edges

Returns

`calcDiameterOverHeartcycles()`

Builds two internal lists of transducer contours (`tnear_hc`, `tfar_hc`) that are triggered by the heart-cycle and finally calculates and stores the resulting diameter-vs-time courses in members `diam_courses_` and `time_course_` (in mm over sec).

Returns True if at least one fully defined heart-cycle was found

Return type bool

`confirmSelCB(btn, btn_calculate)`

User clicks “Confirm” button of `config_box_main` and therefore and enables next state

Parameters

- `btn` (`Button`) – clicked button object
- `btn_calculate` (`Button`) – next state to enable after this step was confirmed by the user

Returns

`drawWallImage(tnear_wall=None, tfar_wall=None, thickness=2)`

Draw aortic walls on the original markup image

Parameters

- `tnear_wall` (`list`) – list of `Nx2 ndarray` edge definitions of transducer near wall
- `tfar_wall` (`list`) – list of `Nx2 ndarray` edge definitions of transducer far wall
- `thickness` (`int`) – thickness of markup contour

Returns the currently shown RGB image, highlighted with passed edges

Return type `ndarray, NxMx3`

`getAorticBoundaries(y_livals, cur_transd_pos_type, blur_kernel=3)`

Estimate the aortic region of interest by calculating boundary values that are likely to embed the aortic walls. This is basically done by

1. Canny Edge detection (`canny()`)
2. Weighting for contours with high y-deviation per dx
3. Rejection of small-weighted contours (by histogram-classification)
4. Building a sum function along y-axis where each datapoint represents the sum of contour px in this image-row
5. Detection of peaks of this mapping function
6. Weighting of peaks (height, FWHM - see `FWHM()`, physiologically realistic diameter, centrality)
7. Selection of most reliable two peaks

Parameters

- `y_livals` (`ndarray, Nx1`) – y-values of the ECG tracing within the `mmode_obj_`
- `cur_transd_pos_type` (`TransducerPosType`) – currently selected transducer type
- `blur_kernel` (`int`) – sigma of gaussian smoothing before Canny Edge detection is applied

Returns the already `mmode_obj_-offsetted` y-values of the estimated aortic boundaries

Return type (int, int)

`getEdges(image_fullfile=None, waitFcn=None)`

Create the local `Edges` object in order to find the `roi_`'s edges in a second thread. Found edges are then shifted to match shown image positions and interpolated to lpx-courses.

Parameters

- `image_fullfile` (*string*) – full path of currently loaded image
- `waitFcn` (*function*) – function called during execution of extensive edges thread

Returns true if edges could be found

Return type bool

`mouseBtnPressPainterCB(x_pos, y_pos)`

Callback function for pressed mouse button in “PAINTER_SEL” `DrawState` when user wants to adjust the wall edges - starts cursor routine.

Parameters

- `x_pos` (*int*) – x-position of cursor on image pane
- `y_pos` (*int*) – y-position of cursor on image pane

Returns

`mouseBtnReleasePainterCB()`

Callback function to stop the cursor routine

Returns

`mouseMovePainterCB(x_pos, y_pos)`

Callback function executed, when user clicks and holds on one of the edges (‘tnear’ or ‘tfar’ edge). Basically it calculates a linear interpolation between the point of last click or the last end-of-interpolation.

Parameters

- `x_pos` (*int*) – current x-position of cursor in us-image
- `y_pos` (*int*) – current y-position of cursor in us-image

Returns

`selDrawState(btn, draw_state)`

Callback function to select the state of manual intervention of edges (select

`Aorta.DrawState)`

Parameters

- `btn` (`Button`) – button that called method
- `draw_state` (`DrawState`) – state that should be set

Returns

`setAorticBoundaries((y_0, y_1), color=(255, 255, 0))`

Sets the aortic ROI (`roi_`) by cropping the M-mode roi from from `y_0_` (lower boundary) to `y_1_` (upper boundary)

Parameters

- `y_0` (`int`) – first vertical boundary position within `mmode_obj_`
- `y_1` (`int`) – second vertical boundary position within `mmode_obj_`
- `color` (`(int, int, int)`) – tuple of 8Bit RGB values for color of horizontal line boundary

`undoSelCB(btn)`

User clicked “Undo” button of `config_box_main` to reset edges to those originally detected.

Parameters `btn` (`Button`) – clicked button object

Returns

`advise_label_ = None`

information message shown for confirmation of set wall-boundaries

`confirm_btn_ = None`

confirm button object

`diam_courses_ = None`

NxM array of heart cycle-based diameter courses (end product of this class)

`draw_state_ = None`

state of drawing interaction after edges were found successfully

`img_obj_ = None`

original image is used as ROI (without possible markups)

`img_shown_ = None`

currently shown US-image

`mmode_dx_ = 0`

passed `mmode_obj_` image offset in x-direction

`mmode_dy_ = 0`

passed `mmode_obj_` image offset in y-direction

```

mmode_obj_ = None
    M-mode image (which is a subimage of image_obj_ )

px_per_cm_ = None
    pixel per centimeter

px_per_sec_ = None
    pixel per second

roi_ = None
    aortic subimage (which is a subimage of mmode_obj_ )

tfar_wall_ = None
    transducer far object edge

tfar_wall_corr_ = None
    manually adapted far wall edge

time_course_ = None
    1xM array of timepoints, related to diam_courses

tnear_wall_ = None
    transducer near object edge

tnear_wall_corr_ = None
    manually adapted near wall edge

updateFcn = None
    update function to call after the US-image has been updated

y_0_ = 0
    lower boundary line for ROI of the used M-mode image

y_1_ = 0
    upper boundary line for ROI of the used M-mode image

class Aorta.DrawState
    Bases: enum.Enum

    CONFIRM_SEL = 4
        state where edges are accepted

    NO_SEL = 1
        state where no tool action or tool is selected

    PAINTER_SEL = 2
        state where painter is selected

    UNDO_SEL = 3
        state where edges should be reseted to initial edges

```

4 Calibration module

```

class Calibration.AortaCalibration
    This class implements an object, that prescribes physiological parameters of different

```

patient ages and US-transducer positions. Its object is globally used for

- parametrisation of image-related detection-thresholds
- storing all possible `TransducerPosType` including their sub-list of `Examination`

```
static getAorticParameters(diam_course=None, time_course=None,  
                           pres_sys=None, pres_dias=None)
```

Calculates aortic elasticity related parameters, if *diam_course* is not empty.

This is done by searching the local minimum within the time defined in `DIA_MIN_START` and `DIA_MIN_END` and the local maximum in `SYS_MAX_START` and `SYS_MAX_END`.

Parameters

- `diam_course` (`ndarray`) – heart-cycle average diameter curves over time
- `time_course` (`ndarray`) – hear-cycle average time curves
- `pres_sys` (`float`) – averaged systolic pressure (mmHg)
- `pres_dias` (`float`) – averaged diastolic pressure (mmHg)

Returns a ordered dictionary containing all calculated aortic parameters, index of minimal and maximal value within *diam_course*

Return type (`OrderedDict`, `int`, `int`)

```
getTransducerAbbreviations()
```

Getter for all defined transducer position abbreviations

Returns all abbreviations of transducer positions

Return type list

```
getTransducerFullNames()
```

Getter for all defined Transducer position names

Returns all full names of transducer-positions

Return type list

```
DIA_MIN_END = 0.25
```

diastolic minima location endpoint, referring to triggerpoint (ms) -> diastolic minimum should be around [0, 250]ms after the triggerpoint which is offsetted by “-50ms” from R-peak

```
DIA_MIN_START = 0
```

diastolic minimum location startpoint, referring to triggerpoint (ms)

```
SYS_MAX_END = 0.5
```

systolic maximum location endpoint (ms)

`SYS_MAX_START = 0.2`
systolic maximum location startpoint (ms)

`cur_transducer_calib_ = <Calibration.TransducerPosType instance>`
currently selected transducer calibration

`sect_per_cm_ = 2`
default nr. of sections per cm

`sect_per_sec_ = 10`
default nr. of sections per second

```
class Calibration.TransducerPosType(fullname, abbrev,  
                                     MIN_DIAM, MAX_DIAM,  
                                     THRESH_ROWSUM PEAKHEIGHT)
```

This class defines the position of the ultrasound transducer used for this examination. Although its not an elegant solution, this object contains the list of Examination objects for each TransducerPosType object. It is the main class to

- manage add/removal of an Examination
- set the current Examination
- calculate the list of “Single and Average Diameter Courses”, based on the passed diameter curves

Standard constructor

Parameters

- `fullname` (*string*) – full name of transducer position
- `abbrev` (*string*) – abbreviation of transducer position
- `MIN_DIAM` (*float*) – minimal aortic diameter for this transducer position type (in cm)
- `MAX_DIAM` (*float*) – maximal aortic diameter for this transducer position type (in cm)
- `THRESH_ROWSUM_PEAKHEIGHT` (*float*) – float threshold within normed [0.0, 1.0], that defines the minimal height of peaks for the row-sum of M-mode image edges

```
addExaminations(diam_courses, time_course, date_of_exam, img_shown,  
                filename, bp_sys_mean, bp_dia_mean)
```

Adds a new Examination to the list of Examination and sets the current Examination to index this list element.

Parameters

- `diam_courses` (ndarray of ndarray) – list of Nx2 diameter curves
- `time_course` (ndarray) – time vector for *diam_courses*

- `date_of_exam` (`datetime`) – date of examination
- `img_shown` (`ndarray`) – markup RGB image of this Examination, NxMx3
- `filename` (`string`) – name of image file
- `bp_sys_mean` (`float64`) – average systolic blood pressure (mmHg)
- `bp_dia_mean` (`float64`) – average diastolic blood pressure (mmHg)

Returns

`clearTransducerPosExams()`

Function to clean up all transducer position Examination, plots and parameters, related to a patient.

Returns

`getExaminationData()`

Generates a resampled list of courses of different Examination on one single time-curve.

Returns the numpy list of average- and single-diameter curves (from possibly different images), their filenames, dates and time vector

Return type list, list, list, list, ndarray

`removeExaminationData(remove_idx)`

Remove the examination element on a certain index

Parameters `remove_idx` (`int`) – element in `examinations_` that should be removed

Returns

`storeCurrentPlotsNParams(norm_diam_plt, avg_diam_plt, param_set)`

Function to store current plots as pngs in the internal `cur_norm_diam_plt_` and `cur_avg_diam_plt_` buffer, as well as strings of calculated parameters in `cur_param_set_`

Parameters

- `norm_diam_plt` (`Figure`) – plot object showing the single aortic diameter curves
- `avg_diam_plt` (`Figure`) – plot object showing the average aortic diameter curves

- `param_set` (*OrderedDict*) – dictionary containing the calculated aortic elasticity parameters

Returns

`MAX_DIAM_ = 0`
maximal aortic diameter for this transducer position type (in cm)

`MIN_DIAM_ = 0`
minimal aortic diameter for this transducer position type (in cm)

`THRESH_ROWSUM_PEAKHEIGHT_ = 0`
float threshold within normed [0.0, 1.0] that defines the minimal height of peaks for the row-sum of M-mode image edges (“Detect Aorta” state)

`abbrev_ = None`
abbreviation of transducer position (i.e. “AscAo”)

`cur_avg_diam_plt_ = None`
current average diameters plot for this transducer pos

`cur_norm_diam_plt_ = None`
current normal diameters plot for this transducer pos

`cur_param_set_ = None`
current calculated parameter set for this transducer pos

`examinations_ = []`
list of different `Examination` with this transducer position of a patient (=super-object)

`fullname_ = None`
full name of transducer position (i.e. “Ascending Aorta”)

`i_examination_ = 0`
index of current `Examination`

5 Configuration module

```
class Configuration.AortaQualityParam
```

```
    Bases: object
```

Helper class that generates an Excel file, containing established quality parameters of the edge detection process. It operates on one single xlsx worksheet containing the quality parameters as columns and the processed images for each recursive iteration as a row. In addition, the edge-highlighted image can be exported for each iteration using the `exportContourMarkupImg()` method.

```
addQualityParams(recursion_cnt, coeff_xcorr, tnear_coeff_rcorr, tn-  

                 ear_cont_metr, tfar_coeff_rcorr, tfar_cont_metr)
```

Write the contour metrics of the current analyzed file into a list of lists, which

is used in `exportXlsxData()` to write into the created Excel-file. Note that this is only done if the file was created with `initConfigurationPaths()` first.

Parameters

- `recursion_cnt` (*int*) – recursion of calling `sepSnakeToWall()` function
- `coeff_xcorr` (*float*) – scalar correlation coefficient of transducer near with transducer far curve
- `tnear_coeff_rcorr` (*float*) – correlation coefficient within [0.0, 1.0] of transducer near course with R-peaks
- `tnear_cont_metr` (`OrderedDict`) – Dict containing quality metrics of transducer near edge generated by `calcContourQuality()`
- `tfar_coeff_rcorr` (*float*) – correlation coefficient within [0.0, 1.0] of transducer far curve with R-peaks
- `tfar_cont_metr` (`OrderedDict`) – Dict containing quality metrics of transducer far edge generated by `calcContourQuality()`

Returns

`exportContourMarkupImg(img, cont, recursion_cnt, color=(0, 255, 0), thickness=1)`

Write the passed contours on the passed image in `image_parampath_` and name the file including the `recursion_cnt`.

Parameters

- `img` (`ndarray`, NxM) – aortic ROI image the edge is based on
- `cont` (list of `ndarray`, NxM) – list of contours
- `recursion_cnt` (*int*) – recursion of calling `sepSnakeToWall()` function
- `color` (*(int, int, int)*) – color in (R,G,B)
- `thickness` (*int*) – thickness of markup line in px

Returns

`exportXlsxData()`

Initialises the content of the excel worksheet to write in with correct column names.

Returns

`initConfigurationPaths(image_parampath, image_fullfile='No file specified')`

This function initialises the paths for the xlsx-file and the contour-markup

images. It must be called before `addQualityParams()` and `exportXlsxData()` can be called.

Parameters

- `image_parampath` (*string*) – name of directory where parametrisation images and xlsx-sheet should be created
- `image_fullfile` (*string*) – name of xlsx sheet

Returns

`b_write_headline = False`

Flag indicating if column headers should be part of the CSV

`image_fullfile_`

`image_parampath_ = ''`

Path of parametrisation files (contour-markup images, csv-files etc.)

`class Configuration.Config`

The `Config` class defines global project configuration settings (i.e. debugging modes)

`static getResourcePath(relative_path)`

Function returns the relative path to the executable. Note that

“`sys._MEIPASS`” only exists as virtual environment, generated by PyInstaller.

Parameters `relative_path` (*string*) – path of related file, relative to the exe

Returns full path related file

`b_aorta_calib = False`

if true, metrics from the aortic edge detection quality estimation are exported to calibration xlsx-file

`b_debug = False`

global debug mode flag - if True, each debugging step of AortUs will open a separate Gtk-window for visualisation

`b_export_snakeiteration = False`

if True, each iteration of the MorphSnakes algorithm will be exported as jpg-markup file incl. a movie in a subdir “~/snakeiteration”

`b_extra_info = False`

if true, some extra information i.e. position of M-mode ROI-points, found R-peak indices, y-position of aortic boundaries etc. will be printed in the command line

`b_logfile = True`

flag for creating/appendng a logfile in tempdir

`b_logimages = False`

flag to save all files loaded by this AortUs session in the “tempdir/aortus_imgs”

```

bit_depth = 255
    standard bit depth of each channel for loaded images

fig_cnt = 0
    figure counter used for figures generated in debug mode

mouse_delta_det = 4
    mouse detection value in px from the actual sensitivity value (i.e. a line is also
    dragged if the cursor is +/- 4 px away from it)

mouse_radius = 3
    mouse detection radius in px for image-interfacing functionalities

version = '1.0.2'
    AortUs version

```

6 Ecg module

```
class Ecg.Ecg(img)
```

This class implements a single ECG tracing, recorded in parallel to the M-mode image. It can be generated by using the `getEcgCourse()` functions. Its objective is to find positions of R-peaks within its tracing, represented by entries of the `Rpeaks` class.

Standard Constructor

Parameters `img` (ndarray, NxMx3) – overall loaded image used to find the ECG on

```
DEBUG_drawContours(img, contours)
```

Debug function to draw and show contours on the given image

Parameters

- `img` (ndarray, NxMx3) – overall used US-image
- `contours` (list of ndarray) – list of contours to plot

Returns

```
DEBUG_plot2ndOrderPeakDetection(y_compare, t_compare, max_time,
                                fontsize=10)
```

Debug function to plot an overview of different states of the second order peak detection on the ECG. It operates on a list of ECG tracings (different states) including their related time vector.

Parameters

- `y_compare` (list of ndarray) – list of ECG-courses in different states of 2nd order ECG detection

- `t_compare` (list of ndarray) – list of related time vectors
- `max_time` (*int*) – range of time values to take (ms)
- `fontsize` (*int*) – size of used titles, axes labels

Returns

`contourDoubleToSingleDef(l_contours)`

This function addresses the problem that the `cv2.findContours()` algorithm typically returns the outer contours of a single contour. In terms of ECG, we want to reduce this to one single y-value per x-value for each passed contour. Note that this function also unifies the direction in ascending x-dimension.

Parameters `l_contours` (list of ndarray) – list of double sided contours

Returns the list of single sided contours

Return type list of ndarray

`filterNCalcEcgEdges(contours, img, px_per_sec, cover_thresh=0.7)`

This function filters the list of passed contours and returns those, that cover at least `cover_thresh` width of the image, but has also got a huge overall x-gradient in its definition (consecutively defined in x-direction).

Parameters

- `contours` (list of ndarray) – unfiltered list of contours
- `img` (ndarray, NxMx3) – overall used US-image
- `px_per_sec` (*int*) – pixels per second
- `cover_thresh` (*float*) – ECG course has to cover at least `cover_thresh` width of image

Returns

`getEcgCourse(px_per_sec, x_axis=None)`

Function to extract the ECG tracing from overall US-image `img_`. It saves the found ECG tracing values (datapoints) within its members `x_livals_` and `y_livals_`. Note: This function requires a bit depth of 8bit per channel.

Parameters

- `px_per_sec` (*int*) – used px-per-sec
- `x_axis` (*(int, int, int, int)*) – defining location of the x-scale axis (p0x, p0y, p1x, p1y). When passed, the ECG will be cropped to the range of the x-axis

Returns

`getRPeaks()`

Method to retrieve the position of R-peak values on an already found ECG tracing. It basically uses a second derivation zero-crossing method to determine the regions where possible R-peaks can be found.

Returns the list of indices within `x_linvals_`, that indicate an R-peak

Return type list

`plotEcgCourse(img, color=(0, 255, 0), thickness=1)`

Function to plot the already extracted ECG tracing (from `getEcgCourse()`)

Parameters

- `img` (ndarray, NxMx3) – overall used US-image
- `color` ((*int*, *int*, *int*)) – color of contour in plot
- `thickness` (*int*) – thickness of contour in plot

Returns

`f_samp_ = 0`

equals px-per-pec

`img_ = None`

overall used loaded US-image

`max_hr_ = 200`

maximal heart rate in BPS

`min_hr_ = 40`

minimal heart rate in BPS

`t_ = None`

time vector with length of `x_linvals_`

`x_linvals_ = array([], dtype=float64)`

x-values of ECG, relative to the overall loaded image

`y_linvals_ = array([], dtype=float64)`

y-values of ECG, relative to the overall loaded image

7 Edges module

`class Edges.Edges(ready, snake_pos, img, rpeaks, recursion_cnt)`

This class represents the actual methods used to split, merge and reject edges, based on the returned closed contour resulting object type `MorphGAC`. It basically splits this closed curve to a transducer near- and far list of edges and calculates quality

measurements of their relation. Its main procedure is started by calling the `run()` method, which is the required function to start this class as an own thread.

Standard Constructor

Parameters

- `ready` (`Event`) – event used to init this thread, `None` if no second thread should be started
- `snake_pos` (`string`) – either “within”, “above” or “under” to define the origin of the active contour in the `img`
- `img` (`ndarray`, `NxM`) – used image for active contour
- `rpeaks` (`Rpeaks`) – triggerpoints object
- `recursion_cnt` (`int`) – internal recursion counter for `sepSnakesToWall()` method

`static calcContourQuality(l_contours, img_shape)`

Calculate quality measurements of a passed list of contours:

1. Nr. of contours in list
2. (merged) Total coverage of contours in x-direction, based on the image-width
3. (merged) mean y-value of all contours
4. (merged) standard deviation of y-values of all contours
5. (merged) coefficient of variation of y-values of all contours
6. (merged) averaged y-deviation per dx of all contours

Parameters

- `l_contours` – contours to calculate the quality (mostly quality criterias of merged list)
- `img_shape` (`int`, `int`) – width and height of related `Aorta`. `Aorta.mmode_obj_`

Returns dictionary of calculated quality criterias

Return type `OrderedDict`

`static calcCorrelation(l_cont1, l_cont2, shape, rpeaks=None)`

This function calculates the cross correlation of two contours (not necessarily equal length → zero padded), but also the position of the contour’s extrema (*Minima/Maxima*). These extrema must lie near the passed `Rpeaks` and between the passed `Rpeaks`, respectively.

To check these extrema positions, a sub-contour (= windowed contour) is used for determination, with the size of an average heart cycle in pixels. Therefore

we by default look around $-\frac{px-per-heart-cycle}{2}$ up to $+\frac{px-per-heart-cycle}{2}$ to the right (balance = 50%) of an R-peak. As we don't always have pixels-of-heart-cycle/2 contour pixels left/right to an R-peak, we need to move window and therefore the position of the R-peak within it to the very left/right contour borders (near contour borders, balance != 50%).

Parameters

- `l_cont1` (list of ndarray) – first list of contours that will be merged to one contour
- `l_cont2` (list of ndarray) – second list of contours that will be merged to one contour
- `shape ((int, int))` – shape of ROI
- `rpeaks` (Rpeaks) – position of the R-peaks within the M-mode image

Return type (float, float, float, float)

Returns

1. cross correlation coefficient of first and second (merged) contour
2. ratio **found** Minima/Maxima at/between R-peaks to **all** Minima/Maxima of `l_cont1`
3. ratio **found** Minima/Maxima at/between R-peaks to **all** Minima/Maxima of `l_cont2`

`static calcXCoverage(l_contours, img_shape)`

Calculate the percentual coverage of image-width by the x-values of a list of contours

Parameters

- `l_contours` (list of ndarray) – contours used for calculation of used x-values
- `img_shape (int, int)` – width and height of related `mmode_obj_`

Returns percentual value of x-coverage

Return type float

`static clipoffAllContourEnds(l_contours)`

Iterates through all contours and checks, if the ends of each contour may interfere with ends of other contours (= are double defined in x-dimension). If so, the end of the *smaller* contour gets cut off. This function is especially helpful as `findAndConnectNeighbours()` can only connect right contour neighbours.

Parameters `l_contours` (list of ndarray) – list where double-defined ends should get cut off

Returns list of short-ended contours

Return type list of ndarray

```
static clipoffContourEndsByTemplate(templ_cont, l_contours)
```

Iterates through a list of contours and checks if each one of its element's ends may interfere with ends the *templ_cont* (= are double defined in x-dimension). If so, and if the found element is smaller than *templ_cont*, it gets cropped.

Parameters

- *templ_cont* (ndarray, Nx2) – contour that is used to cross check ends with elements of *l_contours*
- *l_contours* (list of ndarray) – list where double-defined ends of *templ_cont* should get cut off

Returns list of short-ended contours, regarding the ends of *templ_cont*

Return type list of ndarray

```
static euclidDist((x0, y0), (x1, y1))
```

Calculates the euclidian distance between two points

Parameters

- *x0* (*int*) – startpoint x-position
- *y0* (*int*) – startpoint y-position
- *x1* (*int*) – endpoint x-position
- *y1* (*int*) – endpoint y-position

Returns euclidican distance between startpoint and endpoint

Return type float

```
static findAndConnectNeighbours(l_contours, img, max_x_gap,  
                                max_y_gap, course_direction=0,  
                                cont_clip=0)
```

Wrapper function to iteratively find and connect nearest neighbours with a maximal px-gap distance in x- and y-direction

Parameters

- *l_contours* (list of ndarray) – contours to scan for internal neighbours
- *img* (ndarray, NxM) – image the contours operate on
- *max_x_gap* (*int*) – maximal px-size a gap of two neighbouring contours may has in x-direction

- `max_y_gap` (*int*) – maximal px-size a gap of two neighbouring contours may has in y-direction
- `course_direction` (*int*) –
 - 1 for courses whose definition ranges from $x=img_width$ to $x=0$ (descending)
 - -1 for courses whose definition ranges from $x=0$ to $x=img_width$ (ascending)
 - 0 if it should be automatically detected (if `tnear` and `tfar` is not known yet)
- `cont_clip` (*int*) – nr of pixels that a contour should be reduced at its endings, that will be connected

Returns list of connected contours

Return type list of ndarray

```
static findNeighbourContours(l_contours, max_x_gap=7,
                             max_y_gap=7, value_region=0.2)
```

Find and return the indices of possibly neighbouring contours within

l_contours

Parameters

- `l_contours` (list of ndarray) – list where neighbours to the right should be identified
- `max_x_gap` (*int*) – minimum number of pixels that possibly neighbouring contours are dislocated in x-direction
- `max_y_gap` (*int*) – minimum number of pixels that possibly neighbouring contours are dislocated in y-direction
- `value_region` (*float*) – percentage factor that possibly neighbouring contours may differ in y-mean/y-sd value

Returns

- a list of tuples containing (*index of current contour*, *index of its right neighbour*)
- empty if no neighbour was found

Return type list of (int, int)

```
static getNearAndFarContour(l_contours, snake_pos)
```

Function to return the two largest (`tnear`, `tfar`) contours in a list of contours, if possible. If only 1 or no contour is found, this one contour and [] or ([], []) is returned. In addition, the returned contour curve directions are corrected (`tnear` must be defined with ascending x-values, `tfar` with descending x-values) if the snake was not started from the center of the contour.

Parameters

- `l_contours` (list of ndarray) – splitted snake contours
- `snake_pos` (*string*) – string that is either
 - "within"
 - "above" or
 - "below",indicating where the snake has started within the `mmode_obj_`.

Returns tuple of near- and far contour

Return type (ndarray, ndarray)

```
static linkContours(left_cont, right_cont, img, course_direction,  
                  cont_clip)
```

Links the left contour with the right contour (vertically seen) by using fix-point active contour. Note that this function takes care of the passed course direction (1 -> ascending x, -1 -> descending x). If not passed, `left_cont` and `right_cont` indicates this sorting and inserts the interpolated snake in between the correctly concatenated courses.

Parameters

- `left_cont` (ndarray) – left contour that should be joined to the right
- `right_cont` (ndarray) – right contour that should be joined to the left
- `img` (ndarray, NxM) – `mmode_obj_` where the active contour operates on
- `course_direction` (*int*) –
 - 1 for courses whose definition ranges from $x=img-width$ to $x=0$ (descending)
 - -1 for courses whose definition ranges from $x=0$ to $x=img-width$ (ascending)
 - 0 if it should be automatically detected (if `tnear` and `tfar` is not known yet)
- `cont_clip` (*int*) – nr of pixels that a contour should be reduced at it's endings, that will be connected are clipped before, in ratio to full contour length. This is only valuable for contours that are *not* direct pixel-neighbours (= active contour has to be applied)

Returns joined contour

Return type ndarray

```
static mergeOldAndNewContours(l_new_contours, l_old_contours, img)
```

This function connects two “same contours” found at two different active contour iterations for a “too short” old contour. Therefore following steps are executed:

1. Check on which side of *l_old_contours* are less contour points within the image-width
2. Find end-value on this side of *l_old_contours*
3. Find contour in list *l_new_contours* that includes this end-value
4. Cut the found contour of *l_new_contours* at this point and link it to contour in *l_old_contours*

Parameters

- *l_new_contours* (list of ndarray) – contours found at new active contour run
- *l_old_contours* (list of ndarray) – contours found at previous active contour run. This must be a list containing one contour only!
- *img* (ndarray, NxM) – image where active contour was used on

Returns the merged contour

Return type list of ndarray

```
static moveNearContourToLeadingEdge(l_conoturs, img,  
                                   pct_disting_walldet)
```

Moves each element of the passed list of contours to the median of the determined aortic-wall thickness (= next edge in y-direction). This is done by using `cv2.Sobel()`-operator in y-direction. If a contour does **NOT** have a distinct median minima, the contour is rejected.

Parameters

- *l_conoturs* (list of ndarray) – contours that should be moved to the leading edge
- *img* (ndarray, NxM) – M-mode image where *l_cont* operates on
- *pct_disting_walldet* (*float*) – percentage of px of a conotur that must have a distinctly found Minima (=border) to not reject the contour

Returns leading-edge-corrected contours

Return type list of ndarray

```
static rejectBulges(l_contours, course_direction, max_bulge_width)
```

This function analyses the (already split) list of contours for expanding bulges (= bulges that are convex to the original snake) and removes them. It is assumed, that `l_contours` is sorted along course of the found and splitted snake.

Parameters

- `l_contours` (list of ndarray) – already splitted contour (into list) that should be analyzed for bulges
- `course_direction` (*int*) – must be +1 for ascending and -1 for descending course definition in x-direction
- `max_bulge_width` (*int*) – maximal width of a gap that connects to the bulge, in px

Returns The list of contours, excluding those who represent bulges

Return type list of ndarray

```
static rejectContourOutliers(l_contours,           course_direction,
                             max_line_length,    min_def_length=5,
                             max_y_grad=1)
```

Rejects contours whose main course

1. contains lines longer than `max_line_length`
2. is in negative x-direction
3. reject possibly correct contours within two rejected wrong-course contours
4. is smaller than `min_def_length`
5. has a y-gradient > `max_y_grad`

from a list of contours

Parameters

- `l_contours` – already splitted list of contours where outliers should be detected and the related contour should be rejected
- `course_direction` (*int*) –
 - 1 for courses whose definition ranges from $x=img-width$ to $x=0$ (descending)
 - -1 for courses whose definition ranges from $x=0$ to $x=img-width$ (ascending)
- `max_line_length` (*int*) – maximal length of a definition gap so that a line is plotted in this gap (using polylines)
- `min_def_length` (*int*) – minimal number of points a contour needs to have
- `max_y_grad` – upper threshold for max. average gradient of contour in y-direction

Returns list of outlier-rejected contours

Return type list of ndarray

`run()`

Starts this thread event

Returns

```
static runActiveContour(img, top_left, bottom_right, smoothing=1,  
                       threshold=0.31, num_iters=350)
```

Initialises the zero-level-set of the start-object (rectangle with rounded edges) and runs the geodesic active contours approach (see Morphological Snakes)

Parameters

- `img` (ndarray, NxM) – black-white image to operate on
- `top_left` (*int*, *int*) – x/y-top-left position of rounded rectangle within *img*
- `bottom_right` (*int*, *int*) – x/y-bottom-right position of rounded rectangle within *img*
- `smoothing` (*float*) – strength of Gaussian pre-smoothing step before active contour starts
- `threshold` (*float*) – determines which areas are affected by the morphological balloon
- `num_iters` (*int*) – maximal nr of iterations for active contours convergence

Returns found contours

Return type list of ndarray

```
static sepSnakeToWall(snake_pos, img, rpeaks, recursion_cnt,  
                    p_tnear_cont_split=[], p_tfar_cont_split=[])
```

Splits one continuous snake to aortic wall edges

1. split snake on image borders
2. find possibly already splitted (but neighbouring) contours and join them
3. separate into transducer near- and far contours
4. split each of them at inflection points (where $dx < 0$)
5. reject bulges in contours list and contour outliers
6. reconnect contours at positions of rejected bulges
7. correct contour position to “leading edge”

8. merge contours of this recursion with possibly passed contours of outer recursions
9. calculate quality parameters (correlation of contours to each other, to R-peaks, avg. deviation in y-direction,...)
10. check for acceptance or launch an inner iteration with contrast-highlighted sub-images (= decision tree)

Parameters

- `snake_pos` (*string*) – either "within", "above" or "under" to define the origin of the active contour in the *img*
- `img` (*ndarray*, NxM) – used image for active contour
- `rpeaks` (*Rpeaks*) – triggerpoints object
- `recursion_cnt` (*int*) – internal recursion counter for this method
- `p_tnear_cont_split` (*list of ndarray*) – list of transducer near contours from outer recursion, empty if near contour should be recalculated from scratch
- `p_tfar_cont_split` (*list of ndarray*) – list of transducer far contours from outer recursion, empty if far contour should be recalculated from scratch

Returns list of transducer near contours, transducer far contours, and their cross correlation

Return type (list of ndarray, list of ndarray, float)

```
static splitContourAtInflection(l_contours,           course_direction,
                               min_dev_pts=1)
```

Used to split a contour into a list of subcontours by calculating the consecutive ascending or descending x-value course.

Parameters

- `cont` (*ndarray*) – contour that could be splitted into subcontours if it runs against the preliminary direction (at some point).
- `course_direction` (*int*) – defining the predominant course direction (1 -> ascending x, -1 -> descending x)
- `min_dev_pts` (*int*) – defining the number of points that must run against the predominant direction to split the contour at this position

Returns the list of splitted contour

Return type list of ndarray

`static splitSnakeOnImgBorders(snake, img)`

Splits the biggest found snake (= x- and y-values in *snake*) at the horizontal image boundaries and returns the found list of x-axis-sorted contours.

Parameters

- *snake* (list of ndarray) – list of closed-contours (snakes)
- *img* (ndarray, NxM) – related `mmode_obj_` the active contour operated on

Returns splitted contours with open ends on the image borders

Return type list of ndarray

`static uniformContourDirection(l_contours)`

Uniforms the direction of contours to *from-left-to-right*.

Parameters *l_contours* (list of ndarray) – contours that should uniquely be defined with ascending x-values

Returns contours all defined with generally incrementing x-values

Return type list of ndarray

`coeff_xcorr = None`

cross correlation of `tnear_cont_split` and `tfar_cont_split`

`tfar_cont_split = None`

list of transducer far edges

`tnear_cont_split = None`

list of transducer near edges

8 Examination module

```
class Examination.Examination(date_of_exam, diam_courses, time_course,  
                               img_shown, filename, bp_sys_mean,  
                               bp_dia_mean)
```

This class implements the most basic element - a single M-mode image and the found diameter curves on it. It is typically used for listing, so that a list may contains multiple analysed M-mode images. Note that the list of `diam_courses_` do not necessarily have the same length!

Standard constructor

Parameters

- *diam_courses* (list of ndarray) – list of single diameter curves of different lengths

- `time_course` (`ndarray`) – single time curve, related to longest element in `diam_courses`
- `date_of_exam` (`datetime`) – date of examination
- `img_shown` (`ndarray`, `NxMx3`) – image with markup of axes, triggerpoints and edges used for this examination
- `filename` (`string`) – name of image file
- `bp_sys_mean` (`float`) – average systolic blood pressure (mmHg)
- `bp_dia_mean` (`float`) – average diastolic blood pressure (mmHg)

`bp_dia_mean_ = 0`
diastolic mean blood pressure (mmHg)

`bp_sys_mean_ = 0`
systolic mean blood pressure (mmHg)

`date_of_exam_ = None`
date of investigation

`diam_courses_ = None`
NxM array of heart-cycle-splitted diameter curves

`filename_ = None`
M-mode US-file of this examination

`img_ = None`
M-mode image of this examination

`time_course_ = None`
1xM array of heart-cycle-based time vector

9 GUI module

`class GUI.GUI`

Bases: `gi.overrides.Gtk.Window`

This class implements the graphical user interface of AortUs. When called, it generates a window of type `Window` that embeds boxes initialised by the `initEasyLayout()` function.

- Patient Box - initialised by `showPatientBox()`
- Config Box - initialised by `showConfigBox()`
- State Box - initialised by `showStateBox()`
- Log Box - initialised by `showLogBox()`
- Image Box - initialised by `showImageBox()`

It should be noted, that the Image Box embeds an object of type `Notebook`, showing two tabs:

1. “M-mode image”
2. “Aortic Parameters”

`DEBUG_loadTestDataset()`

Debug function to automatically start AortUs including an already entered patient and optionally load and process one or more images. This function is especially helpful to debug “late” functions in the edge detection process like i.e. the “Calc Parameters” methods, since instead each run of AortUs would require to enter all the data to get into this “Calc Parameters” state, manually.

Returns

`autoscrollCB(*args)`

Callback function to ensure that the scrollable-textview is always scrolled down when a new entry is set

Parameters `args` – optional args

Returns

`calculateParametersCB(btn)`

Wrapping callback function for `getAorticParametersHandler()` method

Parameters `btn` (Button) – calling object

Returns

`changeComboboxCB(box)`

Sets active `TrasducerPostype` calibration of us-image (-> patient) object and the name of the blood pressure list

Parameters `box` (ComboBox) – calling object

Returns

`checkPatientDataCB(widget=None, lst_str_row=None, lst_str_txt=None)`

This callback function is connected to each “changed” event of casual patient-fields (`MyEntry`) including the changed-event of blood pressure (`MyBloodpressure`). Parameters `lst_str_row` and `lst_str_txt` are related to the blood pressure changes.

Parameters

- `widget` (Widget) – calling widget
- `lst_str_row` (*string*) – row of changed cell in liststore of `MyBloodpressure`
- `lst_str_txt` (*string*) – text of changed cell in liststore of `MyBloodpressure`

Returns

`clearDetectionCB(btn)`

Clear loaded image from GUI and display idle image

Parameters `btn` (Button) – calling object

Returns

`clearPatientDataCB(btn)`

Clear detection (return to idle state) but also clear all Patient Box fields.

Parameters `btn` (Button) – calling object

Returns

`closeAortus(widget, event, data=None)`

Callback function emitted, when task manager closes the window (= user wants to close the window)

Parameters

- `event` (Event) – triggered event
- `data` – optional data

Returns

`confirmAortaDetectionCB(btn)`

Function to end the state of automatic aortic wall region detection and enable the next state (edge detection)

Parameters `btn` (Button) – calling object

Returns

`createDefaultConfigBoxCont()`

Initialises the Config Box with a default content.

Returns

`detectAortaCB(btn)`

Wrapping callback function for `getAorticBoundariesHandler()` method

Parameters `btn` (Button) – calling object

Returns

`detectEcgCB(btn)`

Wrapping callback function for `getEcgHandler()` method

Parameters `btn` (`Button`) – calling object

Returns

`detectEdgesCB(btn)`

Wrapping callback function for `getEdgesHandler()` method

Parameters `btn` (`Button`) – calling object

Returns

`detectScalesCB(btn)`

Wrapping callback function for `getScalesHandler()` method of `UsImage`

Parameters `btn` (`Button`) – calling object

Returns

`enablePatientBox(b_sensitivity, type='all')`

Enables or disables the accessibility of Patient Box fields whenever the user enters or leaves the edge detection process.

Parameters

- `b_sensitivity` (`bool`) – True when it is intended to enable the Patient Box widgets
- `type` (`string`) – Either “all” or “settings” to enable/disable just personal data fields of the patient, or also enable/disable the transducer- and section scale settings.

Returns

`exportResults(btn)`

Export all examinations of all non-empty `TransducerPosType` objects to an PDF.

Parameters `btn` (`Button`) – calling object

Returns

`focusBPEntryCB(widget, entry, data=None)`

Function to automatically create a new blood pressure value, when user unfocusses the prior entry field in the Patient Box. (date of examination)

Parameters

- `entry` (`Event`) – calling event
- `widget` (`Widget`) – widget that received this signal
- `data` – optionally passed data

Returns

`initEasyLayout()`

Initialises the default layout of the main AortUs window, including all embedded boxes:

- Patient Box
- Config Box
- State Box
- Image Box
- Log Box

Returns

`loadImageCB(btn)`

Callback function which opens a file dialogue to select an US-image

Parameters `btn` (Button) – calling object

Returns

`mouseBtnPressCB(event)`

Callback function for pressed mouse button in GUI. It can roughly be differentiated to the following functions:

- select the correct triggerpoint in *Detect ECG* mode
- set startpoint for manual mode in *Detect Scales*
- select the correct y-boundary line in *Detect Aorta* mode
- call aortic manual edge correction

Parameters `event` (Event) – calling event

Returns

`mouseBtnReleaseCB(event)`

Callback function for released mouse button in GUI. It can roughly be differentiated to the following functions:

- reset the selected triggerpoint in *Detect ECG* mode
- accept scaling values (roi, px-per-cm, px-per-sec) in manual mode in *Detect Scales*
- unselect the selected y-boundary line in *Detect Aorta* mode
- call aortic manual edge correction

Parameters event (Event) – calling event

Returns

`mouseMoveCB(event)`

Callback function for mouse move in GUI. It can roughly be differentiated to the following functions:

- redraw the currently selected triggerpoint in *Detect ECG* mode
- draw the rectangle or line in the manual mode in *Detect Scales*
- reset boundary lines in *Detect Aorta* mode
- call aortic manual edge correction

Parameters event (Event) – calling event

Returns

`resizeCB(window)`

Resize window callback function (currently not connected to the callback)

Parameters window (Window) – window to resize

Returns

`runSpinner(flag)`

Function opening a frameless window, including a spinner object only. (currently unused)

Parameters flag (*boolean*) – true if window should be shown, false if it should be hidden

Returns

`secDepthChangeCB(btn)`

Callback function for changed “Sections per cm”

Parameters btn (Button) – calling +/- button

Returns

`secTimeChangeCB(btn)`

Callback function for changed “Sections per sec”

Parameters btn (Button) – calling +/- button

Returns

`setPatientBoxData(patient, date_of_exam, sys_bp, dia_bp)`

Manually sets GUI patient data fields by passed patient object and examination data.

Parameters

- `patient` (*Patient*) – patient object
- `date_of_exam` (*datetime*) – string containing examination data (i.e. 20.09.2009)
- `sys_bp` (*float*) – systolic blood pressure
- `dia_bp` (*float*) – diastolic blood pressure

Returns

`showAorticParameterBox()`

(Late) Sub-Initfunction to create and arrange a second tab within the Image

Box and arrange the setup of plots, parameters and list to dynamically remove outlying diameter curves.

Returns

`showConfigBox(inter_box_spacing=10, within_box_border=5,
box_size=(None, None))`

Sub-Initfunction to create and arrange Config Box objects

Parameters

- `inter_box_spacing` (*int*) – spacing between widgets of this box
- `within_box_border` (*int*) – inner border of this box to its widgets
- `box_size` (*(float, float)*) – tuple of size values to setup the size of this box, relative to the window size (between [0.0, 1.0]). By default, this size is not explicitly requested.

Returns

`showImageBox(box_size=(None, None))`

Sub-Initfunction to create and arrange Image Box objects

Parameters `box_size` (*(float, float)*) – tuple of size values to setup the size of this box, relative to the window size (between [0.0, 1.0]). By default, this size is not explicitly requested.

Returns

`showLogBox(within_box_border=5)`

Sub-Initfunction to create and arrange Log Box objects. Log info is filled by the global Logging object of type `Logging`.

Parameters `within_box_border` (*int*) – inner border of this box to its widgets

Returns

`showPatientBox(within_box_border=5, box_size=(None, None))`

Sub-Initfunction to create and arrange Patient Box objects

Parameters

- `within_box_border` (*int*) – inner border of this box to its widgets
- `box_size` (*(float, float)*) – tuple of size values to setup the size of this box, relative to the window size (between [0.0, 1.0]). By default, this size is not explicitly requested.

Returns

`showStateBox(within_box_border=5)`

Sub-Initfunction to create and arrange State Box objects

Parameters `within_box_border` (*int*) – inner border of this box to its widgets

Returns

`updateAorticParameterBox(diam_courses=None, avg_diam_course=None, time_course=None, course_lbls=None, bp_sys_mean=None, bp_dia_mean=None)`

Update already initialised aortic parameter box's curves and values by adding new curves.

Parameters

- `diam_courses` (list of ndarray) – list of single diameter curves of same lengths
- `avg_diam_course` (ndarray) – average diameter curves
- `time_course` (ndarray) – time curves with appropriate length of single- as well as average diameter curves
- `course_lbls` (*list*) – labels of `diam_courses`
- `bp_sys_mean` (*float*) – average systolic blood pressure (mmHg)
- `bp_dia_mean` (*float*) – average diastolic blood pressure (mmHg)

Returns

`ao_param_names_ = None`
names of the aortic parameters

```

ao_params_ = None
    values of aortic parameters

ao_params_lbls_ = []
    aortic parameter labels that are assigned when using method
    showAorticParameterBox

avg_diam_plt = None
    plot object containing average diameter curve

b_aorta_manual_ = False
    True as long as manual aortic area correction is active

b_edge_manual_ = False
    True as long as detected edges can be corrected

b_img_loaded_ = False
    True when image was loaded successfully

b_rpeak_manual_ = False
    True as long as manual R-peak correction is active

b_scales_manual_ = False
    True as long as manual scale procedure is active

box_color = Gdk.Color(red=65535, green=65535, blue=65535)
    standard filling color for framed boxes

canvas = None
    canvas of us-image

export_path_ = ''
    (default) export path for examination PDFs

frame_color_ = Gdk.Color(red=60000, green=60000, blue=60000)
    standard frame color for (i.e.) boxes

frame_thickness = 5
    thickness of frames

image_box_frame = None
    interactive image box

img_height_ = 0
    height of image -> for flipping by 180deg

img_r_move_ = None
    image template for moving one peaks along the ECG

img_scale_det_ = None
    temporary image for i.e. drawing the rectangle/line during manual scale de-
    tection

import_path_ = ''
    (default) import path for images

```

`norm_diam_plt = None`
 plot object containing all single diameter curves

`p0_ = (None, None)`
 x- and y-values of click-and-hold in image (start point) and for aortic-ROI movements

`p1_ = (None, None)`
 x- and y-values of click-and-hold in image (end point)

`patient_box_frame = None`
 patient data box

`sel_area = 4`
 area in px around R-peaks that is considered for the cursor moving R-peaks

`sel_ind_ = -1`
 index of selected R-peak within the R-peaks liststore

`sel_x_ = -1`
 x-value of currently selected R-peak

`sel_y_ = -1`
 y-value of currently selected R-peak

`us_image_ = None`
 globally used `UsImage` object

`wait_spinner_ = None`
 spinner window which is activated for excessive calculation operations (when multithreading is used)

10 Logging module

```
class Logging.Logging(timestamp_spacing=5, timestamp_format='%d/%m/%Y
%H:%M:%S')
```

Bases: `gi.overrides.Gtk.TextBuffer`

This class implements the the textbuffer used to fill the Log Box on one hand, and the log-file in the temporary user related hidden directory on the other hand. This is typically “C:\Users\<<Username>\AppData\Local\Temp” for Windows operating systems.

Standard constructor

Parameters

- `timestamp_spacing` (*int*) – number of spaces used between the timestamp (left) and the actual message (right)
- `timestamp_format` (*string*) – format of timestamp

`DEBUG_saveImg(name, img)`

Debug function to save each originally loaded M-mode in tempdir, if flag is true. This function is especially useful for debugging cases on computers, different to the own.

Parameters

- `img` (`ndarray`, $N \times M \times 3$) – image to save to the tempdir
- `name` (`string`) – name of file

Returns

`endLogFile()`

Function called when AortUs is closed, in order to close the logfile.

Returns

`logfile_ = None`

logfile id

`logimg_path_ = None`

logged images path (string)

`new_log_`

`timestamp_format_ = ''`

date format for timestamp creation

`timestamp_spacing_ = 0`

number of character from beginning of new line till start of text-entry (= includes timestamp length!)

11 Morphsnakes module

`class Morphsnakes.MorphGAC(data, smoothing=1, threshold=0, balloon=0)`

Bases: `object`

Morphological GAC based on the Geodesic Active Contours.

Create a Morphological GAC solver

Parameters

- `data` (`ndarray`) – The stopping criterion $g(I)$, see functions `gborders()`
- `smoothing` (`float`) – The number of repetitions of the smoothing step in each iteration. This is the parameter *my*.

- **threshold** (*float*) – The threshold that determines which areas are affected by the morphological balloon. This is the parameter *theta*.
- **balloon** (*float*) – The strength of the morphological balloon. *This is the parameter v.*

set_balloon(*v*)

Set balloon force direction

Parameters *v* (*float*) – The strength of the morphological balloon

Returns

set_data(*data*)

Set the data that controls the snake evolution (the image or $g(I)$)

Parameters *data* (*ndarray*) – passed data

Returns

set_levelset(*u*)

Set current levelset

Parameters *u* (*ndarray*) – passed levelset

Returns

set_threshold(*theta*)

Set threshold for balloon force

Parameters *theta* (*float*) – threshold

Returns

step()

Perform a single step of the morphological snake evolution.

Returns

balloon

The morphological balloon parameter (*nu*, not *v*).

data

The data that controls the snake evolution (the image or $g(I)$).

levelset

The level set embedding function (*u*).

threshold

The threshold value (*theta*).

`class Morphsnakes.fcycle(iterable)`

Bases: `object`

Call functions from the iterable each time it is called.

`Morphsnakes.IS(u)`

IS operator

Parameters `u (ndarray)` – levelset

Returns

`Morphsnakes.ISoSI(u)`

ISoSI operator

`Morphsnakes.SI(u)`

SI operator

Parameters `u (ndarray)` – levelset

Returns

`Morphsnakes.SIoIS(u)`

SIoIS operator

`Morphsnakes.ellipse_levelset(shape, top_left, bottom_right, l_width=5,
rad_corner=5)`

Initialise filled rounded rectangle levelset to passed positions

Parameters

- `shape ((int, int))` – shape of behindlaying image
- `top_left ((int, int))` – top left position of rounded rectangle
- `bottom_right ((int, int))` – bottom right position of rounded rectangle
- `l_width (int)` – line width of rounded rectangle in px
- `rad_corner (int)` – corner radius of rounded rectangle in px

Returns 2D mask with zeros, except the filled rounded rectangle with ones

Return type `ndarray`

`Morphsnakes.evolve(msnake, levelset=None, num_iters=20, DE-
BUG_img=None)`

Evolution of a morphological snake.

Parameters

- `msnake` (MorphGAC or MorphACWE) – MorphGAC or MorphACWE instance, the morphological snake solver
- `levelset` (ndarray, NxM) – If given, the levelset of the solver is initialised to this. If not given, the evolution will use the levelset already set in `msnake`.
- `num_iters` (*int*) – The number of iterations.
- `DEBUG_img` (ndarray, NxM) – The image to print each iteration of the snake

Returns

`Morphsnakes.gborders(img, alpha=1.0, sigma=1.0)`

Calculate stopping criterion for image borders

Parameters

- `img` (ndarray, NxM) – image whose edges should act as stopping criteria
- `alpha` (*float*) – weighting of smoothed image
- `sigma` (*float*) – gaussian smoothing standard deviation

Returns

`Morphsnakes.rounded_rectangle(img, top_left, bottom_right, line_color, l_width, rad_corner)`

Create rounded rectangle on a passed image.

Parameters

- `img` – passed image to create the rounded rectangle on
- `top_left` (*(int, int)*) – top left position of rounded rectangle
- `bottom_right` (*((int, int))*) – bottom right position of rounded rectangle
- `l_width` (*int*) – line width of rounded rectangle in px
- `rad_corner` (*int*) – corner radius of rounded rectangle in px

Returns `img` including rounded rectangle

Return type ndarray

`Morphsnakes.curvop = <Morphsnakes.fcycle object>`
curvature operator

12 MySubclasses module

```
class MySubclasses.MyBloodpressure(parent, def_str_sys, def_str_dia,  
                                   def_abbrev, valueChangedCB=None)
```

Bases: `MySubclasses.MyDynamicTreeview`

Class implementing a TreeView list that automatically averages and stores its columns (systolic blood pressure, diastolic blood pressure) entries. It also checks its content for consistency (within defined boundaries) and is per default appendable.

Standard constructor

Parameters

- *parent* (Window) – parent window
- *def_str_sys* (*string*) – default column title for systolic column
- *def_str_dia* (*string*) – default column title for diastolic column
- *def_abbrev* (*string*) – abbreviation, shown as title of the “+” column
- *valueChangedCB* (*function*) – callback function being called at the very end of other callbacks, after the content was checked for plausibility

```
addBPValues(sys_bp_lst=[], dia_bp_lst=[])
```

Function to add blood pressure values generically via passed lists.

Parameters

- *sys_bp_lst* (*list of float*) – list of systolic blood pressure values to add
- *dia_bp_lst* (*list of float*) – list of diastolic blood pressure values to add

Returns

```
clearList()
```

Call actual clear liststore function and remove member values.

Returns

```
editConfirmCB(_, row, entered_text, model, column)
```

Callback function when user entered and confirmed an BP-field with Enter button. This function also checks the entered value for plausibility and calculates the current mean value for the used column.

Parameters

- `_` – omitted
- `row` (*int*) – selected row
- `entered_text` (*string*) – entered text
- `model` (`TreeModel`) – currently used model
- `column` (*int*) – selected column

Returns

`updateMeanValues(*kwargs)`

Function to update the mean values of the *Systolic* and *Diastolic* column.

The mean value will be integrated in the column title. If the mean cannot be calculated, the default string will be used instead.

`MAX_DIA_BP = 250`

maximal diastolic blood pressure (mmHg)

`MAX_SYS_BP = 300`

maximal systolic blood pressure (mmHg)

`MIN_DIA_BP = 30`

minimal diastolic blood pressure (mmHg)

`MIN_SYS_BP = 50`

minimal systolic blood pressure (mmHg)

`dia_mean = None`

diastolic mean blood pressure (mmHg)

`sys_mean = None`

systolic mean blood pressure (mmHg)

`type_`

Get abbreviation name of BP-Treeview

Returns title of BP-Treeview

Return type string

`valueChangedCB = None`

callback function called at the very end of other callbacks, after entry was checked for plausibility

`class MySubclasses.MyCV`

This class mainly embeds static helper functions for `OpenCv`.

`static check_opencv_version(major, lib=None)`

Function to check the currently installed `OpenCV` version

Parameters

- `major` (*str*) – major version (i.e. “2.”)

- `lib` – used library

Returns true if major is “2.” and used OpenCV version is i.e. 2.1

```
static findContours(image, mode, methods, contours=None, hierar-
                   chy=None, offset=None)
```

OpenCv version independent implementation of `findContours()`. Please refer to the original documentation for more details regarding the datatypes.

Parameters

- `image` (`ndarray`, `NxMx3`) – image to find contours on
- `mode` (`int`) – contour retrieval mode
- `methods` (`int`) – contour approximation method
- `contours` (list of `ndarray`) – (output) detected contours
- `hierarchy` – (output) vector containing information about the contour’s hierarchy
- `offset` (`(int, int)`) – offset by which every contour point is shifted

Returns

```
static is_cv2()
```

Checks if OpenCV is of major version 2

Returns true if version is 2

```
static is_cv3()
```

Checks if OpenCV is of major version 3

Returns true if version is 3

Returns

```
class MySubclasses.MyDiameterPlt(x_vals=None, y_vals=None, title=",
                                dpi=50, box_size=(None, None),
                                font_size_title=22, font_size_ticks=19,
                                font_size_legend=17)
```

Bases: `gi.repository.Gtk.VBox`

Creates a diameter-time plot with a certain size (in inches). This plot is used within a Notebook tab to show the single- and average diameter curves over time.

Standard constructor

Parameters

- `x_vals` (`ndarray`, `Nx1`) – array of x-values

- `y_vals` (`ndarray`) – array of y-values for at least one curve, NxM
- `title` (`string`) – title of plot
- `box_size` (`(int, int)`) – requested size of plot-box in inches
- `dpi` (`int`) – dots per inch - required for high resolution plots
- `font_size_title` (`int`) – font size of plot title
- `font_size_ticks` (`int`) – font size of axis ticks and label
- `font_size_legend` (`int`) – font size of legend entries

`addDataCourses(x_vals, y_vals, ind_highlight=None, course_lbls=None)`

Add data-courses to this axes

Parameters

- `x_vals` (`ndarray`, Nx1) – array of x-values
- `y_vals` (`ndarray`) – array of y-values for at least one course, NxM
- `ind_highlight` (`list`) – indices of element that should be highlighted in the plot
- `course_lbls` (`list`, M) – labels of courses

Returns

`removeDatacourse(remove_idx)`

Tries to remove datacourse from plot

Parameters `remove_idx` (`int`) – remove index

Returns

`setDataCourses(x_vals, y_vals, ind_highlight=None, course_lbls=None)`

First clears the axis-courses, then adds new axis-courses

Parameters

- `x_vals` (`ndarray`, Nx1) – array of x-values
- `y_vals` (`ndarray`) – array of y-values for at least one curve, NxM
- `ind_highlight` (`list`) – indices of element that should be highlighted in the plot
- `course_lbls` (`list`, M) – labels of curves

Returns

`ax_ = None`
main axes object


```

fig_ = None
    figure object

font_size_legend_ = 0
    font size of legend entries

font_size_ticks_ = 0
    font size of ticks and x/y-labels

font_size_title_ = 0
    font size of title

title_ = ''
    title label

```

```

class MySubclasses.MyDynamicTreeView(parent, n_columns,
                                       b_appendable=True)

```

Bases: `gi.overrides.Gtk.TreeView`

Base class of liststore treeview. This class implements a flat list of items stored in its member `bp_liststore` that is appendable if `b_appendable` is true. It always has at least 2 visible columns:

1. “+” column, to append an entry to this list
2. “-” column, to remove a certain element from this list

Additionally added columns therefore have at least the column index 2 and are accessible by a single click in their field, or by pressing “TAB” (if *editable*).

Standard constructor

Parameters

- `parent` (*Widget*) – parent widget
- `n_columns` (*int*) – number of columns the user wants to create (excluding “+”/“-” columns)
- `b_appendable` (*bool*) – true if Dynamic-Treelist is appendable

```

addListRow(l_entries, row=0)

```

Adds a row to list if enough columns are specified.

Parameters

- `l_entries` (*list of string*) – list of string items that should be added to the column’s cells
- `row` (*int*) – row where entries should be inserted

Returns

```

clearList()

```

Clear all items in the list and restore to default view (incl. possible appendable row)

Returns

`clickPlusMinus(treeview, event)`

Callback function for click-event on treeview. This function wraps the call of three types of subfunctions:

- click on “+” (first column)
- click on “-” (second column)
- click on any other visible column

Parameters

- `treeview (TreeView)` – overall treeview
- `event (Event)` – calling event

Returns

`getColVals(col_ind)`

Function to retrieve all visible row values of a defined column in form of a list

Parameters `col_ind (int)` – column index

Returns list of row values, None if extraction is not possible

`jumpToNextEditableCell(treeview, event)`

Function to jump through `MyDynamicTreeview` editable cells by simply pressing “TAB” key. For appendable `MyDynamicTreeview` objects, a new row is created when the user is currently in the last row and presses “TAB”.

Parameters

- `treeview (TreeView)` – overall treeview
- `event (EventKey)` – calling event

Returns

`pmCallback(row, col, cb_fcn)`

Helper function to call the passed callback function with two args of integer row and column index

Parameters

- `row (int)` – clicked row index
- `col (int)` – clicked column index
- `cb_fcn (function)` – function to call

Returns

`selectionChangedCB(sel)`

Callback function for changing the selection of row in BP treeview. The function basically restricts selecting the last row (which is actually just used for putting buttons within the liststore).

Parameters `sel (TreeSelection)` – current selection

Returns

`aux_cb_fcn = None`

function called for click events on click events on non-empty entries in “+”/”-” columns, except the actual “+”/”-” signs

`b_appendable_ = False`

flag indicating if dynamic-treeview-list should have appendable elements

`b_sel_col = False`

boolean flag indicating if user is currently in edit mode

`bp_liststore = None`

liststore object which is basically the table shown in this treeview

`col_ind = -1`

current column index

`minus_cb_fcn = None`

function called when “-” is clicked

`minus_col_idx = -1`

column index of “-” sign

`n_columns = 0`

number of data-columns to create

`parent_ = None`

parent object of this treeview

`plus_cb_fcn = None`

function called when “+” is clicked

`plus_col_idx = -1`

column index of “+” sign

`sel_row_ = None`

iterator of currently selected row

```
class MySubclasses.MyEntry(parent, default_str, type="", changedCB=None,
                           max_chars=100)
```

Bases: `gi.repository.Gtk.Entry`

Subclass to create a restricting Entry-field for (i.e.) upper+lowercase letters only

Standard constructor

Parameters

- `parent` (`Window`) – parent window
- `default_str` (`string`) – displayed in the entry field in light gray as long as the user gives no input
- `type` (`string`) – either “name”, “number” or “date”; defines the charset that is allowed in this field
- `changedCB` (`function`) – function to call whenever input of this field changed
- `max_chars` (`int`) – maximum characters allowed in this field

`clear()`

Reset Entry text to `default_str`

`enterEntryCB(entry, widget)`

Callback function being called when the user focusses this Entry

Parameters

- `entry` (`Entry`) – entered entry field
- `widget` (`Widget`) – calling widget

Returns

`enterText(entry)`

Fill Entry with text programmatically

Parameters `entry` (`string`, `int`, `datetime`) – data which should be stored in Entry

Returns

`entry2Int(entry)`

Convert entry to integer

Parameters `gentry` (`string`) – number which should be stored in Entry

Returns

`exitEntryCB(entry, widget)`

Callback function being called when the user unfocusses from this Entry

Parameters

- `entry` (`Entry`) – exited entry field
- `widget` (`Widget`) – calling widget

Returns

`isEmpty()`

Checks if the Entry is empty or contains the default string

`restrictToDate(entry)`

Callback function checking if the entry is a valid date, according to defined

`dateformats (date_formats_)`

Parameters `entry (Entry)` – entered entry field

Returns

`restrictToName(entry, charset)`

Callback function checking and correcting if input of entry is not within the charset or more than `maxchars`.

Parameters

- `entry (Entry)` – entered entry field
- `charset (string)` – defined charset, either `name_chars_`, `number_chars_` or `date_chars_`

Returns

`date_chars_ = '0123456789.-/ '`
chars allowed in date fields

`date_formats_ = ['%d.%m.%Y', '%d/%m/%Y', '%d-%m-%Y', '%d %m %Y', '%d%m%Y']`
date formats that are allowed

`default_str = ''`
default string which is shown in grey chars and vanishes when user clicks into entry-field

`entry_cont_ = ''`
content which is either a (valid) string, a (valid) date or empty.

`max_chars = 0`
maximal characters allowed in this field

`name_chars_ = 'abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ - '`
chars allowed in name fields

`number_chars_ = '0123456789'`
chars allowed in numeric fields

`parent = None`
parent class object

`type = None`
(optional) either one of (“name”, “number”, “date”) - defines the charset that is allowed in this field

```
class MySubclasses.MyGtk
```

This class mainly embeds static helper functions for GTK.

```
static getNotebookTabLbIs(notebook)
```

Returns a list of all `Label` texts of notebook tab pages.

Parameters `notebook` (`Notebook`) – notebook whose pages should be iterated

Returns list of strings of tab-labels of notebook, “None” if tab is not a `Label`

Return type list

```
static removeAllNotebookTabs(notebook)
```

Removes all existing tabs of the passed `Notebook`

Parameters `notebook` (`Notebook`) – notebook to clear

Returns

```
static removeBoxElements(box)
```

Static method removing all `Widget` s of a `Box`

Parameters `box` (`Box`) – `Box` whose widgets should be removed

Returns `Box` without any children

Return type `Box`

```
static setCurrentNotebookTab(notebook, tabname)
```

Trys to set the current notebook tab by its `tabname`

Parameters

- `notebook` (`Notebook`) – notebook whose current tab should be changed
- `tabname` (`string`) – name of tab

Returns

```
class MySubclasses.MyInputDialog(txt, title, content=None, parent=None)
```

Bases: `gi.overrides.Gtk.MessageDialog`

Standardised input dialog that contains an `Entry` field and an “OK” button to confirm.

Standard constructor

Parameters

- `txt` (`string`) – information text in this dialogue

- `title` (*string*) – title of this dialogue
- `content` (*string*) – default content of the `Entry` field
- `parent` (`Window`) – parent window

`onClickOK(btn)`

Set entered text

Parameters `btn` (`Button`) – calling button

Returns

`content_ = None`
entered text as string

```
class MySubclasses.MyMessagePopup(txt, parent=None, type=<enum
    GTK_MESSAGE_WARNING of type
    Gtk.MessageType>)
```

Bases: `gi.overrides.Gtk.MessageDialog`

Subclass to create standardised message popup windows.

Standard Constructor

Parameters

- `txt` (*string*) – information string to show
- `parent` (`Window`) – parent window
- `type` (`MessageType`) – message type

```
class MySubclasses.MyStateArrow(type=<enum GTK_ARROW_RIGHT of
    type Gtk.ArrowType>, shadow=<enum
    GTK_SHADOW_OUT of type
    Gtk.ShadowType>)
```

Bases: `gi.overrides.Gtk.Arrow`

Class implementing the default layout of state arrows used within the *State Box*

Standard Constructor

Parameters

- `type` (`ArrowType`) – showing from left to right
- `shadow` (`ShadowType`) – shadowing

```
class MySubclasses.MyStateButton(label="", cb_fcn=None, width=70,
    height=50)
```

Bases: `gi.overrides.Gtk.Button`

Class implementing the default layout of state buttons used within the *State Box*

Standard Constructor

Parameters

- `label` (*string*) – label of this button
- `cb_fcn` (*function*) – callback function of this button
- `width` (*int*) – desired width of button
- `height` (*int*) – desired height of button

`btn_ = None`
button object within the HBox

```
class MySubclasses.MyWaitWindow(parent)
```

Bases: `gi.overrides.Gtk.Window`

Window containing a spinner

Standard Constructor

Parameters `parent` (Window) – parent object

`b_waiting_ = False`
flag indicating if the spinner is turning

`spinner_ = None`
spinner object

13 Patient module

```
class Patient.Patient(first_name, last_name, insurance_nr, date_of_birth,  
                      updatePlotsNParams, gui=None)
```

Bases: `MySubclasses.MyDynamicTreeview`

Class implementing the patient relevant data, as well as the related list of Examination (diameter curves).

Standard constructor

Parameters

- `first_name` (*string*) – first name
- `last_name` (*string*) – last name
- `insurance_nr` (*string*) – insurance number
- `date_of_birth` (*datetime*) – date of birth
- `updatePlotsNParams` (*function*) – function to call after examinations have been added to the patient
- `gui` (Window) – parent window


```
addTransducerExamination(diam_courses, time_course, date_of_exam,  
                          img_shown, filename, bp_sys_mean,  
                          bp_dia_mean)
```

Add a new Examination to the currently selected global `TransducerPosType`.

Parameters

- `diam_courses` (list of ndarray) – list of single diameter curves of different lengths
- `time_course` (ndarray) – single time curve, related to longest element in `diam_courses`
- `date_of_exam` (datetime) – date of examination
- `img_shown` (ndarray, NxMx3) – image with markup of axes, triggerpoints and edges used for this examination
- `filename` (*string*) – name of image file
- `bp_sys_mean` (*float*) – average systolic blood pressure (mmHg)
- `bp_dia_mean` (*float*) – average diastolic blood pressure (mmHg)

Returns

```
initCourseList(gui)
```

Initialise the scrollable list of diameter curves with columns “Image”, “Date” and “Label” (of image). Note that this list is *not* appendable.

Parameters `gui` (Window) – parent window

Returns

```
rowRemovedCB(row, col)
```

Remove a certain course from the examination and update plots and calculations by changed items in this `MyDynamicTreeView`.

Parameters

- `row` (*int*) – row index to remove
- `col` (*int*) – column index to remove

Returns

```
cur_examination_
```

Get current Examination

Returns current Examination object if existing, None if not

```
cur_transd_pos_type_
```

Get current `TransducerPosType`

Returns currently selected `TransducerPosType` object

`date_of_birth_ = None`
 date of birth as type `datetime`

`first_name_ = ''`
 first name of patient

`i_transd_pos_ = 0`
 index of current `TransducerPosType`

`insurance_nr_ = 0`
 full insurance nr.

`last_name_ = ''`
 last name of patient

`transd_pos_ = []`
 list of different `TransducerPosType` that contain the list of `Examination` of this patient

14 Rpeaks module

```
class Rpeaks.Rpeaks(parent, img, updateFcn, ecg_x, ecg_y, ecg_t, ecg_r_ind=[],
                  next_state_btn=None)
```

Bases: `MySubclasses.MyDynamicTreeview`

Triggerpoints list object, basically storing and managing the indices within the ECG tracing, where triggerpoints are set.

Standard constructor

Parameters

- `parent` (`Window`) – parent window
- `img` (`ndarray`, `NxMx3`) – overall image
- `updateFcn` – update function being called whenever an R-peak position is initialised or changed
- `ecg_x` (`ndarray`) – x values of Ecg tracing
- `ecg_y` (`ndarray`) – y values of Ecg tracing
- `ecg_t` (`ndarray`) – time values of x values
- `ecg_r_ind` (*list*) – indices of found R-peaks within the `ecg_x`, `ecg_y` and `ecg_t` vectors
- `next_state_btn` (`Button`) – next state button to set sensitive

```
addRemoveListItemCB(row, col)
```

Callback function called, when user clicks on the ‘+’, ‘-’ or “*All” in the treeview

- “+” creates new entry at the end of the list
- “-” removes current entry
- “*All” resets all entry’s offsets to the currently selected one

Parameters

- row (*int*) – row index
- col (*int*) – column index

Returns

`applyToAllCB(treeview, event, offset=None)`

Apply Offset to

1. reset all rows to this *offset*
2. if *offset* is *None*, use selected row to set offset

Parameters

- treeview (TreeView) – used treeview object
- event – calling event
- offset (*string*) – offset to set all values to

Returns

`correctIndByOffset(iter)`

Aux function telling if t +/- offset is within time vector

Parameters iter (TreeIter) – iterator of R-peak that should be corrected by offset

Returns the index of the closest ECG-value of t +/- offset, -1 if time exceeds time-vector

`editConfirmCB(_, row, entered_text, model, column)`

Callback function called, when user entered and confirmed an offset with enter button

Parameters

- _ – omitted
- row (*int*) – selected row
- entered_text (*string*) – entered text
- model (TreeModel) – used treemodel
- column (*int*) – selected column

Returns

`getAvgHeartRate()`

Calculates the average heartrate in px, if more than 2 R-peaks are defined.

Returns 0 if less than 2 R-peaks are defined, else the average nr. of pixel per heart-cycle

`initTreeView()`

Initialise default appendable `MyDynamicTreeview` object with three columns

- Name of triggerpoint
- Time of triggerpoint
- Offset value (editable) of triggerpoint

Returns

`plotRPeaks(img=None, x=None, y=None, color=(255, 0, 0), thickness=2, radius=3)`

Plot the passed x- and y-values (array of list) or (if not passed) the already defined member x- and y-values

Parameters

- `img` (ndarray, NxMx3) – overall image used
- `x` (ndarray or list) – x-values of peaks to plot
- `y` (ndarray or list) – y-values of peaks to plot
- `color` ((*int*, *int*, *int*)) – RGB color
- `thickness` (*int*) – thickness of circle border line
- `radius` (*int*) – radius of circles

Returns

`selectionChangedCB(sel)`

Callback function for changing the selection of row in R-peaks treeview. The function basically restricts selecting the last row.

Parameters `sel` (`TreeSelection`) – current selection

Returns

`updateList(sort_idx)`

Function to update the order of R-peak items in list (according to peaks from “left to right” in the image)

Parameters `sort_idx` (ndarray) – array of sort indices

Returns

`DEF_OFFSET = -50`
default offset of triggerpoints is 50ms

`MIN_NR_RPEAKS = 3`
minimal nr. of R-peaks that need to be found in order to proceed to the next state

`b_ecg_peaks_ = False`
flag indicating at least one R-peak element within the liststore

`columns = ['', '', 'Name', 't (ms)', 'Offset (ms)']`
column titles

`ecg_end_ = None`
Ecg end position (x,y)

`ecg_r_ind_ = None`
indices for R-peaks in Ecg vectors

`ecg_start_ = None`
Ecg start position (x,y)

`ecg_t_ = None`
Ecg time values of x-values

`ecg_x_vals_ = None`
x-values of Ecg course

`ecg_y_vals_ = None`
y-values of Ecg

`img_ = None`
image with scales

`name = None`
list of names that are a combination of `peak_str` and the index of the triggerpoint

`next_state_btn_ = None`
pushbutton to enable/disable next state after “Detect ECG”

`offset = None`
list of offsets of triggerpoints

`peak_col_ = (255, 0, 0)`
color of peak circle

`peak_rad_ = 3`
radius of peak circle in image

`peak_str = 'Peak '`
peak entry name

```

peak_thckn_ = 2
    thickness of circle-border

sel_row_ = None
    iterator of currently selected row

selection_ = None
    current TreeSelection

```

15 Scale module

```
class Scale.Scale(img, updateFcn)
```

This class implements the scale objects within the shown image. It embeds

- the position of the detected or defined x-axis
- the position of the detected or defined y-axis
- the resulting position and dimension of the M-mode ROI
- the scaling in px-per-second and px-per-cm

Standard constructor

Parameters

- *img* (ndarray, NxM) – overall image
- *updateFcn* (*function*) – update function to call after *image_obj_* was manipulated

```
QUICKNDIRTY_getScalesAuto(n_horiz_sections, n_vert_sections)
```

This is a subfunction of `getScalesAuto()` to also enable the scaling-extraction for Philipps ultrasound devices. It also operates on a thresholded image but does not use the Hough lines, but the find contours approach to extract the subscales of the (not existing) axes. These contours are detected as subscales, if they all have the same start position in x-direction (for vertical scaling) or y-direction (for horizontal scaling). If so, their orthogonal distance is averaged and used as scaling. Note, that the axes start and end-definition depend on the position of the first and last found subscale of this image.

Parameters

- *n_horiz_sections* (*int*) – number of horizontal sections per second (defined by the user)
- *n_vert_sections* (*int*) – number of vertical sections per cm

Returns true, if scales and resolution could be extracted

Return type bool

`findLines(img, cmp_type='equal x', max_linewidth=5)`

This function aims to calculate the average distance between subscales of the current axes. This is done by calculating the mean distance in the specified direction (x/y) of non-zero elements (with certain min. length) in the passed subimage of the overall image.

Parameters

- `img` (`ndarray`, NxM) – subimage of overall used image, containing vertical/horizontal subscales only
- `cmp_type` (`string`) – “equal x” to calculate mean vertical distance in the passed `img`, “equal y” to calculate mean horizontal distance in the passed `img`
- `max_linewidth` (`int`) – maximal width of subscale lines in px

Returns

`getScalesAuto(n_horiz_sections, n_vert_sections)`

This function automatically estimates the axis on a given US-image and also computes the used region of interest (= M-mode image pane). This is basically done by the following steps:

- First threshold the image by suppressing content with a brightness value lower than `thresh`. Then the goal is to find horizontal main axis (must cover at least 90% of image width) as well as the vertical main axis (must cover at least 50% of image height).
- If found, the distances of axes-orthogonal sub-lines are calculated and therefore px-per-sec and px-per-cm are defined, as well as the positions of the main axes and the M-mode ROI.
- If not found, the algorithm tries to find it in another way (for Philipps devices) where no main axes are defined, but sub-sections for the horizontal and vertical scaling exist on the right and bottom end of the M-mode ROI. This is where `QUICKNDIRTY_getScalesAuto()` is called.

Parameters

- `n_horiz_sections` (`int`) – number of horizontal sections per second (defined by the user)
- `n_vert_sections` (`int`) – number of vertical sections per cm

Returns True if both axes were found, False if not.

Return type bool

`getScalesManual(btn, config_box_main, updateWindowFcn, btn_det_ecg)`

This function will be called if the scales could not be found automatically. It

is a procedure that helps to set the most important parameters for this class by switching through the `ScaleState`.

1. Definition of M-mode ROI
2. Definition of time-resolution
3. Definition of depth resolution

Each of these states can be confirmed (when definition succeeded) by clicking the created “Next” button in the Config Box.

Parameters

- `btn` (`Button`) – calling “Next” button
- `config_box_main` (`VBox`) – Config Box container
- `updateWindowFcn` (*function*) – function to call after the content of the Config Box was manipulated
- `btn_det_ecg` (`Button`) – next state button which is enabled, after this scale definition succeeded

Returns

`plotScales(color=(0, 255, 0), thickness=2)`

Function to plot the already found scales

Parameters

- `color` (*(int, int, int)*) – RGB color of axes
- `thickness` (*uint*) – thickness of lines as axes

Returns

`setMMode((p0_x, p0_y), (p1_x, p1_y))`

Sets the definitions for x- and y-axis, as well as the M-mode image object and its dx/dy-offset. This is especially useful for API mode.

Parameters

- `p0_x` (*int*) – x-start position of roi
- `p0_y` (*int*) – y-start position of roi
- `p1_x` (*int*) – x-end position of roi
- `p1_y` (*int*) – y-end position of roi

Returns

`setResolution(px_per_sec, px_per_cm)`

Sets the current pixel resolution

Parameters

- `px_per_sec` (*int*) – pixel per second
- `px_per_cm` (*int*) – pixel per centimeter

Returns

```

advise_label_ = None
    Label for advises in Config Box of GUI

b_interaction = False
    true, if user is in manual interaction mode

image_obj_ = None
    overall image object

mmode_dx_ = 0
    x-offset of M-mode image within overall image

mmode_dy_ = 0
    y-offset of M-mode image within overall image

mmode_obj_ = None
    M-mode image object of type ndarray

next_btn_ = None

px_per_cm_ = 0
    determined or defined pixel per centimeter

px_per_sec_ = 0
    determined or defined pixel per second

st_manual_scale_ = None
    State of type ScaleState in Manual State Detection process

updateImageFcn = None
    update function to call after image_obj_ was manipulated

x_axis_ = None
    definition of x-axis (p0x, p0y, plx, ply)

y_axis_ = None
    definition of y-axis (p0x, p0y, plx, ply)

class Scale.ScaleState
    Bases: enum.Enum

    Enum defining states for manual scale detection

    FINISH = 4

    HORIZ_RESOLUTION_DEF = 3

    ROI_DEF = 1

    VERT_RESOLUTION_DEF = 2

```

16 UsFile module

```
class UsFile.UsFile(filename, mmode_settings=(None, None, None, None),
                    px_per_sec=None, px_per_cm=None, rpeak_idx=None,
                    y_roi=None)
```

The UsFile class implements the actually loaded ultrasound file and possible already known properties for AortUs. This class is especially useful for testing in API mode (refer to `loadTestDataset()`) or even for a later usage as save-file in a proprietary format. It mainly contains properties used to bypass the time-consuming extraction in the GUI by “already known” i.e. position of scales.

Standard Constructor

Parameters

- `filename` (*string*) – name of loaded image file
- `mmode_settings` (*(int, int, int, int)*) – position of M-mode image area (`p0_x`, `p0_y`, `p1_x`, `p1_y`)
- `px_per_sec` (*int*) – pixels per second
- `px_per_cm` (*int*) – pixels per centimeter
- `rpeak_idx` (*list*) – list of indices within the `Rpeaks`
- `y_roi` (*(int, int)*) – tuple of y-positions of aortic boundaries in M-mode image

`filename_ = None`
name of loaded ultrasound file

`p0_x_ = None`
(already known) M-mode startpoint x-position

`p0_y_ = None`
(already known) M-mode startpoint y-position

`p1_x_ = None`
(already known) M-mode endpoint x-position

`p1_y_ = None`
(already known) M-mode endpoint y-position

`px_per_cm_ = None`
(already known) pixels per centimeter

`px_per_sec_ = None`
(already known) pixels per second

`rpeak_idx_ = []`
(already known) list of indices within the `Rpeaks`

`y_roi_ = (None, None)`
(already known) tuple of y-positions of aortic boundaries in M-mode image

17 UsImage module

```
class UsImage.UsImage(parent, config_box_main, show_all, win_update,  
                     win_size, image_path=None)
```

Bases: `object`

This class implements the interface between the GUI class and the backend classes (`Aorta`, `Ecg`, `Patient`, `Scales`, `Rpeaks`). It handles input from the GUI, (in)activates its widgets if necessary and calls `MyMessagePopup` if something notable happens.

Standard constructor

Parameters

- `parent` (`GUI`) – parent window
- `show_all` (*function*) – Show all update function for whole GUI
- `win_update` (*function*) – Update function to trigger after image has been modified
- `win_size` (*(int, int)*) – Size of GUI window so that US-image canvas will be created with the correct size
- `image_path` (*string*) – path+filename of image to (optionally) load at this point

```
clearStateMembers()
```

Clear all object related state members (image name, image objects, `Scale`, `Ecg`, `Rpeaks`, `Aorta`)

Returns

```
createIdleImage()
```

Creates idle image for the GUI, as long as no US-image is loaded.

Returns

```
createPatientExamination(first_name, last_name, insurance_nr,  
                        date_of_birth, updatePlotsNParams, gui)
```

Wrapper function to create a `Patient`

Parameters

- `first_name` (*string*) – first name
- `last_name` (*string*) – last name
- `insurance_nr` (*string*) – insurance number
- `date_of_birth` (*datetime*) – date of birth

- `updatePlotsNParams` (*function*) – function to call after examinations have been added to the patient
- `gui` (*Window*) – parent window

Returns

`exportResultsHandler(fn, examiner, ao_params_names)`

This function exports the current list of `Examination` elements for used `TransducerPosType` elements, used in the current session of a patient to a PDF. This PDF includes general information of AortUs and its version, all personal data of the patient, processed and highlighted M-mode images as well as the single- and averaged diameter curves of each `TransducerPosType` and its related extracted parameters.

Parameters

- `fn` (*string*) – filename+path of PDF file to export (combination of personal data)
- `examiner` (*string*) – credentials of examiner
- `ao_params_names` (`OrderedDict`) – names of parameters to export

Returns

`getAorticBoundariesHandler(y_roi=None)`

Interface function from GUI to `Aorta` class. Its optional input parameter `y_roi` makes it possible to skip the detection of aortic boundaries by using the tuple y-values.

Parameters `y_roi` (*(int, int)*) – y-values of boundaries of used to embed the aortic walls

Returns true if boundaries were found or set successfully

Return type bool

`getAorticParametersHandler(date_of_exam, bp_sys_mean, bp_dia_mean)`

Handler function to call after curves were found successfully. Basically it binds the calculated diameter vs time curves to the correct `TransducerPosType Examination`.

Parameters `updatePlotsNParams` (*function*) – function to call after diameter curves list has changed

Returns

`getEcgHandler(rpeak_idx=None, btn_det_aorta=None)`

Interface function from GUI to `Ecg` class. Its optional input parameter

rpeak_idx makes it possible to skip the detection of R-peaks by using the indices from the passed list.

Parameters

- *rpeak_idx* (*list*) – indices of triggerpoints within the ECG course
- *btn_det_aorta* (*Button*) – next button to enable

Returns the ECG and R-peak highlighted image if both could be extracted, *None* instead

Return type *ndarray*

getEdgesHandler(btn_calculate, waitFcn=None)

Interface function from GUI to Edges class. It first tries to retrieve these edges and then plots them in the GUI on success.

Parameters

- *btn_calculate* (*Button*) – next button to enable
- *waitFcn* (*function*) – optional wait function to call during the Edges object calculates the active contour in a second thread.

Returns *true* if edges were found

Return type *bool*

getFigureCanvas(win_size)

Get figure canvas of current image plot area and try to resize it to the GUI's window size

Parameters *win_size* (*(int, int)*) – tuple size of GUI's window size

Returns the figure canvas

Return type *FigureCanvasGTK3Cairo*

static getPdfYPos(canvas, y_pos, dist, y_0, y_1, font_type='Helvetica', font_size=10)

Helper function to get the current y-position within the used PDF file by calculating by the passed current position and the demanded distance to this position. If the content does not fit the current page any more, a new page is created and the position is updated to the top of this page.

Parameters

- *canvas* (*canvas*) – used pdf file
- *y_pos* (*int*) – current y-position within the file in pt
- *dist* (*int*) – desired distance from y-position in pt

- `y_0` (*int*) – top-border for a A4 page in pt
- `y_1` (*int*) – bottom-border for a A4 page in pt
- `font_type` (*string*) – font type
- `font_size` (*int*) – font size

Returns the new calculated y-position

Return type int

```
getScalesHandler(btn_det_ecg, (p0_x, p0_y), (p1_x, p1_y),
                  px_per_sec=None, px_per_cm=None,
                  n_horiz_sections=None, n_vert_sections=None)
```

Interface function from GUI to Scale class. Its optional input parameters

make it possible to skip the detection of scales by using the passed M-mode area position points and the scalings per cm/sec.

Parameters

- `btn_det_ecg` (Button) – next state button to enable
- `p0_x` (*int*) – M-mode startpoint x-position
- `p0_y` (*int*) – M-mode startpoint y-position
- `p1_x` (*int*) – M-mode endpoint x-position
- `p1_y` (*int*) – M-mode endpoint y-position
- `px_per_cm` – pixel per centimeter
- `px_per_sec` – pixel per second
- `n_horiz_sections` (*int*) – number of horizontal sections per second
- `n_vert_sections` (*int*) – number of vertical sections per second

Returns True if Scales could be detected automatically, False if Scales could not be found automatically and Manual Scale Detection got started

Return type bool

```
loadUsImage(full_img_path=None)
```

Load an image from a passed image path and shows it in the Image Box. This function is especially useful for API mode.

Parameters `full_img_path` (*string*) – path+filename of image to load

Returns true, if success

Return type bool

`updateShownImage(img=None)`

Update the current loaded image with a manipulated version of the

`image_shown_`

Parameters `img` (ndarray, NxMx3) – image that should be used to update the GUI

Returns

`aorta_ = None`

Aorta object

`ax_ = None`

basic top-layer image axis

`config_box_main_ = None`

Config Box

`dpi_ = 50`

DPI used to optimize the shown image

`ecg_ = None`

Ecg object

`examiner_ = ''`

(default) examiner credentials

`i_transd_pos_`

`image_file_`

`image_fullfile_`

`image_obj_ = None`

original source image

`image_path_`

`image_shown_ = None`

source image + (i.e.) coloured axes

`parent_ = None`

GUI window object

`patient_ = None`

Patient object

`rpeaks_ = None`

Rpeaks object

`scales_ = None`

Scale object

`showAll_ = None`

Show All update function for whole GUI

`winUpdate_ = None`

Update function for Us-image to trigger after image has been modified

```
win_size_ = None
    Size of GUI window (width in px, height in px)
```