Bachelor Thesis in Computer Science

Continuos Diameter Measurement of Vascular Structures in Ultrasound Video Sequences

by Matthias Bremser
Matrikelnr. 9007678
May 03,2011

First advisor: Prof. Dr.-Ing. Rainer Herpers
Second advisor: Dr. rer. nat. Uwe Mittag

Bonn-Rhine-Sieg University of Applied Sciences
Department of Computer Science

German Aerospace Center Institute of Aerospace Medicine

Acknowledgment

I herewith confirm that, I prepared the present thesis independently, by exclusive reliance on the literature and tools indicated therein. This thesis has not been submitted to any other examination authority in its current or an altered form, and it has not been published.

(Place, Date, Signature)

Abstract

Abstract Today, computer aided assessment of ultrasound video sequences is considered the gold standard. Ultrasound analysis forms an integral part of the medical research studies performed at the DLR. Currently, two studies are being performed that obtain ultrasound measurements of vascular structures. In this project, a software is developed that supports these and future studies in providing a framework for ultrasound analysis. Furthermore a method for obtaining assessment of diameter measurements from vascular structures in ultrasound video material is provided. Following the problem analysis based on scenarios and tasks the use cases were constructed and used for an system design. A virtual window was derived from the requirements specification—use cases. After consulting the customer with the virtual window, it was later realized by implementing the GUI. The developed software is clearly structured and focuses on extensibility. To be able to achieve this goal, the softwares main components clearly separated by integrating the model view controller pattern into the software's architecture. The facade pattern is utilized to provide easy access to the frameworks interface models and hides the complexity. In the approach of developing the image processing, the image material was examined in depth and characteristics defined. The approach for determining the edges on the vessel walls for diameter measurement is to differentiate the image with a first order derivate of a Gaussian filter. The edges are then detected by analyzing the differentiated image. Diameter measurement between the edges follows certain assumptions that have been made. The diameter measurement algorithm was later evaluated. The evaluation of the diameter measurement algorithm showed that there is considerable variability between the gold standard of manual measurement and the software results. Furthermore, it was shown that the video quality and the definition of the region of interest have an impact on measurement accuracy. Concluding the software developed here delivers a framework for ultrasound image analysis and enables the further refinement of the measurement algorithm and flexible implementation into future ultrasound applications.

Contents

Chapter 1

Introduction

1.1 Motivation

Nowadays, ultrasound assessment is usually performed on the basis of computer aided assessment. In general, ordinary ultrasound devices are capable of performing such tasks. Also the ultrasound device of the Zentrum für Luft und Raumfahrt (DLR) is able to do so. However, studies that are performed by the DLR perform flow mediated diameter measurement. That means, that the blood flow is kept track of and diameter measurement needs to be obtained. The ultrasound device, however, is not capable of performing both tasks at a time. Therefore, the DLR requires a software that obtains diameter measurements on the basis of video recordings from the particular ultrasound device.

1.2 Objective

The main objective is to develop an extensible software that is used for analyzing video sequences. The aim is to support studies that are performed at the DLR. This project especially supports two studies that are performed at the present time of development. Therefore functionality shall be provided to measure the diameter of present vascular structures on ultrasonic time based image material. For each frame of the video the averaged diameter of the present vessel should be saved into a file with reference to corresponding frame and time.

This work is developed in context of a bachelor thesis. To fit the project into the time frame of a bachelor thesis, certain assumptions have been made. Since processing speed is no major criterion, the requirements for the choice of the development environment were reduced towards speed of software development. Since the author is most experienced in with it, it is decided to develop the software in Java[1].

1.3 Project Environment

The project is developed in context of a bachelor thesis at the Bonn-Rhine-Sieg University of Applied Sciences and for the German Aerospace Center. In the following, both

[1] Java is an object oriented programming language originally developed by Sun Microsystems, Inc. which was later acquired by the Oracle Corporation.

institutions are introduced.

1.3.1 Bonn-Rhine-Sieg University of Applied Sciences

The university was founded in 1995 by an initiative enacted by Bundestag, the federal government, Landtag and federal state government as well as the local region. In the beginning the university had over 5600 students, 131 professors and 156 scientific employees. The aim of the university was to create more academic teaching and research capacity and to promote the regional economic area by cooperation with the industries.

The university has three campuses, in Hennef, Rheinbach and Sankt Augustin. In cooperation with the University of Bonn and the RWTH Aachen University, the university is running the B-IT Academy. Today's university has been released from public sponsorship and is now providing a high standard teaching and research environment being self-governed.

1.3.2 German Aerospace Center

The DLR is the center for national research into aeronautics and space in Germany. The main focus areas encompass the research and development in aeronautics, space, transportation, energy and security. The DLR furthermore invests in networking scientific research and cooperates both on a national as well as international level. Beyond its research activity, the DLR furthermore is given responsibility for the planning and implementation of the German space program and representing Germany's interests. The DLR plays an important role in our modern way of living as it is involved in stabilization of air traffic as well as enabling global communication via satellites. The research undertaken at the DLR furthers your understanding of both our own planet as well our solar system, other planets and even the origin of life. And also other industries such as medicine, material science and software engineering benefit from innovations developed by the DLR. The core elements of the DLR comprise:

1. Exploration of the earth and our solar system
2. Investigation of environment protection measures
3. Advancing global communication and security by creating environmentally- friendly technologies

History

The origin of the DLR can be found in several precursor institutions. One of the first was the Aerodynamics Laboratory (AVA), founded in 1907, which was lead by Ludwig Prandtl, the so called father of modern aerodynamics. Here the first wind-canal testing series were initiated already one year after establishment. Other parallel institutions were founded soon after, such as the German research institute for aviation (DFL) in 1936, which was oriented more towards aviation research.

The beginning of the 1950s marked a shift in focus towards space. Initially within the former German Democratic Republic and only in 1961 in the Federal Republic of Germany. Here, in 1963 the institutions AVA, DVL and DFL were all united within the German Research and Experimentation Institute for aviation and spaceflight (DFVLR) . Finally in 1989 with the reunification of Germany the DLR emerged. Since 1997 the DLR exists in its present structure after a fusion and restructuring with the DARA.

Field of activity of the DLR

The biggest proportion of the DLR's research activity is focused on space travel. Close second is aviation, followed by energy, travel and security.

One of the main topics of research in space travel is concerned with the safety of human existence and the effect of high-performance labor in space, which is relevant for astronauts, pilots, crew and potential passengers. The institute of Aerospace Medicine is the DLR department that generates important contributions to this field and is the only research facility specializing in the aspects vital for human physiology and survival in aviation and space travel. The basis of the work presented here is the incorporation of two studies concerned with the effect of zero gravity on human physiology.

1.4 Strategy

As a strategic approach, this project is organized into different phases. Each phase is processed consecutively. Overall, the project comprises 3 phases:

1. Development of video playback and processing functionality.
2. Implementation of the graphical user interface (GUI) and the software framework.
3. Implementation of image processing algorithms.

In the first phase, video playback and processing functionality is developed. Existing frameworks have to be examined and applied to construct video playback and processing methods. In the second phase, a GUI should be produced and introduced to the client. Furthermore, the framework for image processing methods needs to be prepared during this phase. At last, the third phase deals with the implementation of the image processing methods. Image processing problems are approached in an iterative way. This includes vessel wall detection and diameter measurement methods.

At the end of this project, a software will be at hand providing the ability for the user to play and analyze ultrasound video sequences. An evaluation of the diameter measurement will also be performed.

1.5 Outline

This thesis is divided into eight chapters including the introduction. In the following, each chapter is outlined briefly.

Chapter 2 provides background information about studies that currently are being performed at the DLR and will use the software to evaluate results. Information about the assessment of the diameter of vessels is also given, as well as an overview about the state of the art in the field of computer vision.

Chapter 3 provides additional information about the design patterns that are used in the design of the software. This will be needed to understand certain decisions that are made in the system design.

Chapter 4 explains the system design. A problem analysis is provided, and of the basis of that, the methodical approach for the computer vision task of the software is described. After that, a detailed description of the resultant system design is given.

Chapter 5 details important and problematic parts of the software implementation, and how they are overcome.

Chapter 6 provides the design of the evaluation of results of the diameter measurement algorithm that is developed.

In Chapter 7, the project is shortly summarized, problems are discussed and future prospects are given.

Chapter 2

Background

This chapter provides background information that is required for the understanding of the project process. Herein, the first section will provide the reader with further information about the background of two studies that are performed at the present time. Both studies provide this project with data for development and will use the resultant software.

Then, an introduction to the general assessment processes of ultrasound material displaying vessels, is given. This is important to get a deeper understanding of the general process of diameter assessment.

Following, the last section will provide an overview of the state of the art of, and describe why and which approach is chosen in this project.

2.1 Studies

The human body suffers from muscle atrophy and bone loss within weightless environments. The DLR is researching this effect and counter-measures. For this purpose, two studies are performed:

1. „Molekulare und funktionelle Auswirkungen von Vibrationstraining auf die Skelettmuskulatur (EVE-Studie)" (Molecular and Functional effects of whole body vibration on skeletal muscles) (EVE-Study)
2. „Untersuchung der Muskel-, Knochen- und Kreislaufadaptation an ein neues Immobilisierungsmodell für den Unterschenkel (HEP-Studie)" (Study of muscle, bone and circulation adaptations to a new model of immobilization of the lower leg) (HEP-Study)

Both studies are described in the following two sections. Both studies use the Flow-mediated Dilatation (FMD) technique to measure the proposed effects. This technique is also described in the section three.

2.1.1 EVE-Study

Nowadays there are diverse applications for whole body vibration. It is used during strength training as additional stimulus or as therapeutic procedure during rehabilitation.

Whole body vibration applies mechanical vibration onto the whole body, which causes skeletal muscle to relax and shorten alternating. The energy of the vibration device is transferred to the muscle and is thought to cause the stimulation of sensory receptors resulting in muscle contraction. This effect is described in the literature as Tonic Vibration Reflex (TVR).

The strain on the muscle is dependent on the frequency, magnitude and amplitude of the vibration and can cause a fast succession of concentric and eccentric contractions.

Acute effects Standing on top of a vibration platform has the effect of increasing muscle activity, metabolic rate and body temperature (RF00, CR08). A significant change in EMG activity of extensor muscles of the foot and leg. During previous studies it was shown that whole body vibration training yields an increase in muscle activity as seen in increased speed, muscle strength and jump height (BT99, BV99, CS05).

Long-term effects

Previous studies show that after several weeks of whole body vibration, the jump height of subjects is increased by more than 5% (DV03, RS06, RV04, VB04). This result suggests that muscle performance can be increased both short and long-term using whole body vibration, while the same level is only rarely achieved using conventional strength training (BV07, RV04). The present study examines the effect of strength training (knee bend and tiptoeing). Hereby the subjects are separated into the study group receiving whole body vibration and control group without whole body vibration. Each group consists of six subjects undergoing six weeks of training. The aim is to determine whether there is a significant difference in muscle adaptation between the two training methods. Specifically the vastus lateralis of thequadriceps femoris muscle in the thigh and the musculus soleus of the calf muscle are examined. The analysis is based on anatomic, physiological as well as molecular aspects.

Anatomic level Analysis of muscle volume and fascicle length using sonography.

Physiological level: Various jump tests, strength tests and muscle metabolism analysis.

Molecular level: Taking tissue sample from the muscles m. soleus and m. vastus lateralis.

Additionally, hormone and cytokine levels in the blood are monitored.

2.1.2 HEP-Study

During prolonged stays in zero gravity astronauts face considerable health problems. Zero gravity causes muscle and bone degeneration as well as adaptation of various circulatory parameters. One of the main objectives of space travel research are into the adaptations of the human skeleton.

The DLR considers the conquest of mars as the next realizable milestone. To be able to pursuit this goal astronauts have to be able to spend more than 500 days in weightlessness. But if not overcome this can results in significant health problems. The insight that physical exercise has an effect on health has been paid more attention in recent years. This is a reason why further research into the principles governing the human body and its constant physical construction and deconstruction , is important. On this background targeted counter-measures can be developed.

The study aims to understand the bone metabolism. For this purpose a model is designed that can be subjected to artificial zero gravity to gain essential information about bone resorption and ossification.

Previous studies were unable to identify the cause of bone atrophy associated with periods in space, spinal cord injury or prolonged bed rest. Until today it is only known that the muscle and bone tissues are maintained by carrying the own weight in association with muscle contractions. However the quantification of the respective impact as well as what the effects are, remains unknown. During the study the lower leg of the subject is immobilized for 56 days. The muscle will not be used, while the body weight still weighs on the leg. To make this possible an orthosis was developed that restricts the muscle of the lower leg during walking. It is hypothesized that bone atrophy following wearing of the orthosis is comparable to a stay in space. This should show that maintaining bone tissue is dependent on muscle contraction.

The knowledge that weight bearing itself is not sufficient stimulus to prevent bone loss is important in designing precautions and rehabilitation before and after illness or during spaceflight. Apart from evaluation of the adaptation of the bone furthermore the vascular and muscular degeneration are investigated. This is done by FMD analysis as well as using biochemical blood and urine analysis.

2.1.3 FMD

Flow-mediated dilatation describes the dilatation of the blood vessel wall during blood flow. The phenomenon occurs during increased blood flow.

During the FMD analysis, the blood flow of the subject is restricted using a cuff for roughly 1 minute. During this time the artery constricts. When the cuff is loosened the built-up blood presses into the artery. This stretches the artery wall and increases the diameter. The following dilation of the artery takes around 3-5 minutes. The duration of the relaxation of the vascular wall is dependent on the status of the endothelium which is the inner lining of the vessel wall. Damaged or dysfunctional endothelium is believed to be one of the first events during atherosclerosis, which are lesions of fatty material building up within the arterial wall that can eventually lead to obstruction of the vessel or thrombus formation.

This non-invasive method was first described in 1992 and is facilitated by high-resolution ultrasound devices that allow measurement of the vasodilation of the brachial artery. Due to its rather simple and non-invasive procedure, this analysis method has been used in numerous studies, despite lack of international standardization.

2.2 Assessment Process

This section will supply the reader with information about the process of obtaining diameter measurements from blood vessels in ultrasound images. The general process of assessing vessels displayed on ultrasonic material is described. This is important to get a deeper understanding of how assessment measurement is obtained and what the steps and constraints are that have to be considered in the system design in Chapter 4. In this work, the words diameter measurement and diameter analysis are used as synonyms. The word measurement is not to be confused with the actual obtaining the ultrasound videos.

Manual assessment has two major drawbacks. This applies to the repeated assessment of the same material by the same, or another operator(s). When the same operator assesses the same material multiple times, the measurement results will most likely differ. This sort of deviation is called intra-individual deviation. Furthermore, when there are multiple operators performing measurements on the same material, so called inter-individual deviation appears. Albeit these drawbacks, manual assessment of ultrasound material is still considered a valid technique to obtain diameter measurements from ultrasound material.

Manual assessment, so to say, is the template for automated assessment. Due to the nature of computer automation, major drawbacks are suppressed, possibly even cleared.

The assessment process follows certain steps that are illustrated in figure 2.1. The assessor first previews the video to define an appropriate Region of Interest (ROI) and selection of frames. After that the actual diameter measurement is performed.

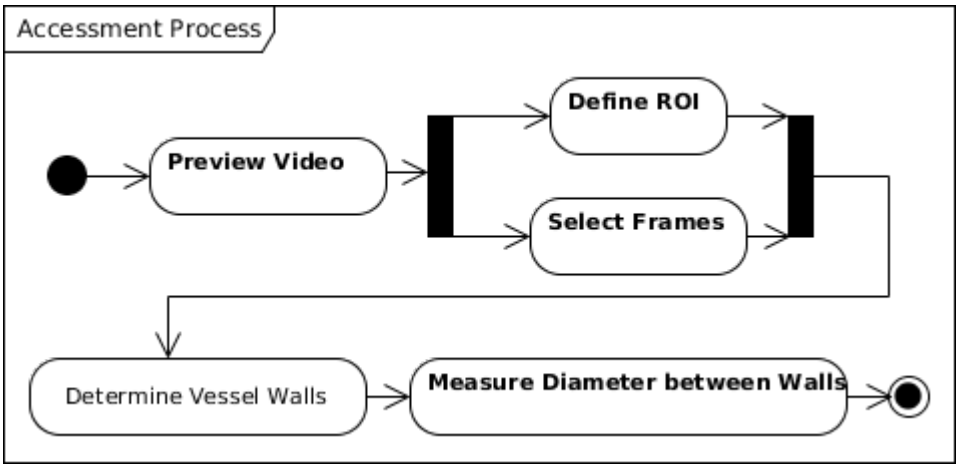


Figure 2.1: The tasks that are performed during the assessment process.

The issues of determining the diameter of a vessel can be distinguished into 2 steps. The determination of the vessel walls and the actual diameter measurement. In the next two paragraphs both steps are described.

Determination of vessel walls A very simple and common method for manual diameter measurement is to fix two markers on the screen of the ultrasound device and measure the distance between them (DMG+07). Figure 2.2 on the following page displays an example of markers on the vessel walls to measure the diameter. In the literature, it is also referred to as the use of electronic caliper (LJR+09). Usually, 3 diameter measurements per image are performed in that way. This work is commonly considered a monotone, interminable, and exhausting task. Thus, only a number of frames are selected and analyzed (SLL02, LJR+09).

A trained operator has prior knowledge about visual appearance and position of the vessel present in the image. When performing manual assessment, the operator makes complex decisions, by integrating this information, intuitively. It is said to measure the internal diameter, although different operators can have different perceptions of what the internal diameter really is. There is no exact specification which position is to be regarded for the internal diameter. However, a reason for this is that most applications for ultrasonic vessel diameter measurement, monitor the relative change in diameter

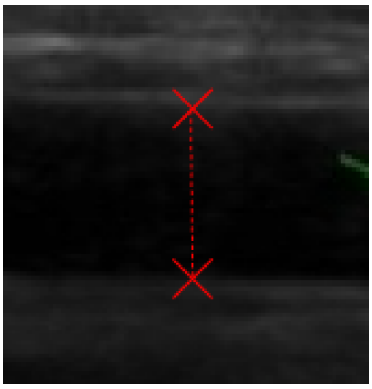


Figure 2.2: Example of markers positioned on the vessel walls to measure the diameter.

over time. The importance here is, that the method that is used to assess the diameter, is consistent for the relevant application. This matter is further discussed in more detail in 4.2.1 on page 27.

Diameter measurement between vessel walls After the vessel walls have been determined, the diameter can be assessed. The diameter is the distance between both vessel walls in the image. Even though, this seem like a trivial task, errors can appear. Yet again, this is due to human inaccuracy. The operator might not always measure the diameter in the correct angle, resulting in different measurements for the same images.

2.3 State of the Art

Computer aided analysis of vascular structures in ultrasonic video material is considered the standard in clinical studies. Various algorithms and techniques for vessel measurement have been proposed. Most techniques require a certain amount of user interaction and are therefore semi-automatic. There are semi-automatic and complete automatic algorithms. The algorithms cover a range of different kinds of techniques. However, most algorithms detect the vessel walls using an active contours technique (cCSTC+99, GPL+02a, GPL+02b, JYX+10), also called snake. The snake model iteratively approaches the contours of the vessel wall. However, earlier and simpler approaches used image differentiation to detect the vessel walls (GAGHL97). This approach is also followed in this project. This is because of the need for a method to provide diameter measurement functionality in a relatively short time. The approach can then be extended or improved later.

Chapter 3

Design Patterns

This chapter provides information about design patterns used in Chapter 4 and Chapter 5. They are used to create a clean structure of the software and help to guarantee that the layout of the software is easily extendable.

In software development, certain problems repeatedly occur. Design patterns are proven and well-established solutions to these problems. They are formalized and documented making them readily available. Furthermore, knowledge of design patterns helps to discover relationships between components or modules. It is often described, that design patterns form a common language amongst software engineers enabling them to quickly and precisely communicate problems and solutions. (ETF04)

Design patterns can be divided into three categories: Creational patterns, Structural patterns, and Behavioral patterns. Creational patters deal with class instantiation. Structural patterns relate to the matter of composition of classes. Behavioral patterns concern the communication between objects of classes. There is also the category of architectural design patterns, that are special in the way that classes are viewed from a higher level. Architectural patterns concern the overall structure of a software to satisfy different requisitions.

Subsequent sections cover patterns in order of:

1. Behavioral patterns
2. Creational patterns
3. Structural patterns
4. Architectural patterns

3.1 Observer

The observer pattern is a behavioral pattern. It provides the capability of informing *observers* about changes in the *observable* object. Instead of polling—that means constantly checking the state—of the observable object, the pattern prescribes that the *observable* notifies all its *observers* on state change. The observer pattern is a widely known concept. There are different names for the objects. The *observable* is often referred to as the *subject*.

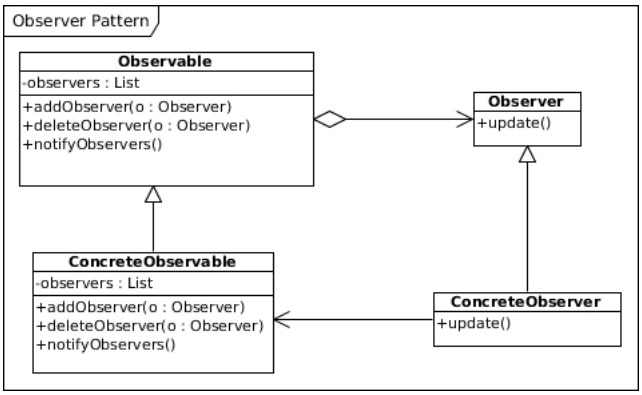


Figure 3.1: UML diagram of the observer pattern. Naming of the entities is based on the Java implementation of the observer pattern.

Figure 3.1 shows the Unified Modeling Language (UML) class diagram of the observer pattern. It prescribes the abstract classes *Observable* and *Observer*. *ConcreteObservable* and *ConcreteObserver* are any desired classes that extend the two abstract classes, meaning, each is taking the role of observable or observer, respectively. The observable holds a list of observer objects. Each of these objects gets notified by *notifyObservers* method by calling the *update* method of the observer. The way how an update is performed depends on a design decision: Whether update should tell that something or what has changed (pull), or update should bring along the new state information (push).

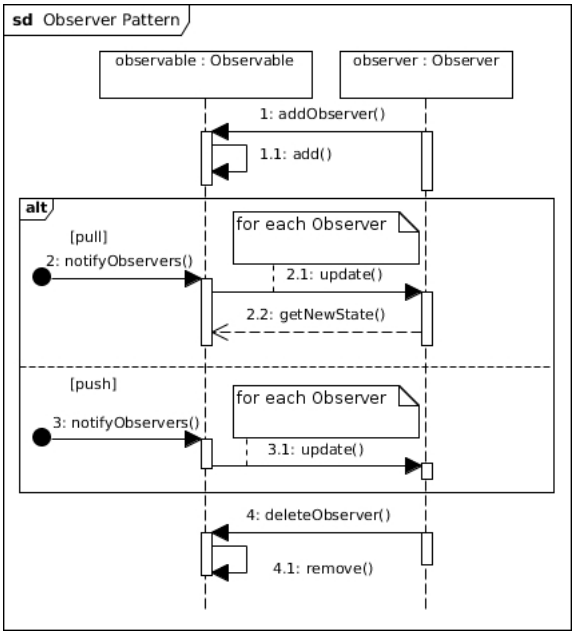


Figure 3.2: UML sequence diagram of observer pattern.

Figure 3.2 shows the sequence diagram of the observer pattern. The diagram represents communication between observable and observer. Leading in and out, there are the *addObserver* and *deleteObserver* methods. These methods cause the observer to be taken on or off the list of observers that observable holds. The alternative combined fragment presents both, the pull or the push method.

3.2 Strategy

The strategy pattern belongs to the group of behavioral design patterns. It facilitates switching algorithms within a context on run-time. The algorithms are called strategies and are used in a certain context. The abstract or interface class *Strategy* is implemented by—and therefore assembling—a number of concrete strategies. The *Context* can refer to all concrete strategies over the interface of the abstract strategy.

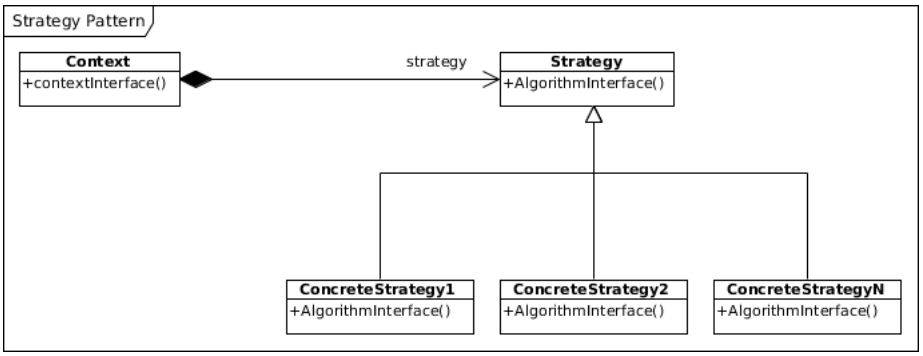


Figure 3.3: UML class diagram of the strategy pattern.

Figure 3.3 displays the pattern as UML class diagram. The *Context* contains an object of type *Strategy*. This can be an instance of *ConcreteStrategy1*, *ConcreteStrategy2*, . . . , or *ConcreteStrategyN*. All strategies implement the *AlgorithmInterface* method, and due to composition relationship between context and strategy, the concrete strategies can be exchanged on run-time.

3.3 Singleton Pattern

The mathematical concept of a singleton is that a *set* contains exactly one element. The pattern converts the mathematical concept to computer science. The singleton pattern is a creational pattern. It provides a way of creating only one instance of a class at a time.

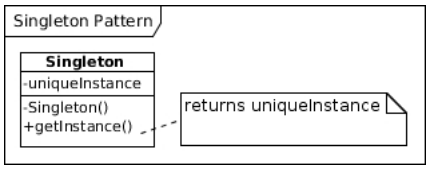


Figure 3.4: UML diagram of the singleton pattern.

In figure 3.4 it can be seen that the singleton holds an private instance of itself. The constructor method is also private, which ensures that external objects can only obtain an instance of the class by using the *getInstance* method. There are two conceptually different variants of implementing the pattern:

Static instantiation *UniqueInstance* is a static field of the singleton class. The singleton class is instantiated by initialization of the field uniqueInstance. This variant should be applied, when the creation of the singleton does not depend on any external context.

Instatiation by getInstance Instantiate the singleton class when needed, that is when the method *getInstance* is called. The method would check if an instance is existent – else create one – and return it. This variant should be applied when instantiation depends on external context. A race condition appears in this variant. Multithreaded use of the method getInstance could accidentally lead to more than one instance of the singleton. This needs to be handled by synchronizing the getInstance method.

3.4 Factory Method

The factory method is a creational design pattern. It was originally described by the Gang of Four (ETF04). Presumably because of the name it was given, there are two different understandings of the pattern.

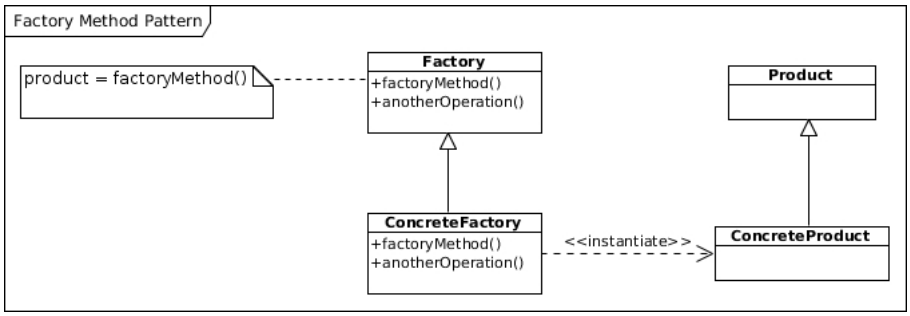


Figure 3.5: UML diagram of the factory method pattern.

The original description of the pattern is shown in figure 3.5. It describes the *Factory* as a class that builds a product with the use of many methods/operations. In the example, creates an instance of a product and applies operations to it. The note in the figure shows the crucial snippet of code. The factory method lets subclasses decide what concrete product is instantiated. This example transfers the responsibility for creating the product to another class in the sense of the design paradigm:

> „Program to an 'interface', not an 'implementation'." (Gang of Four, 1995) (EG94)

In practice, there is often a different understanding by the name of the pattern than it's original definition. Factory method is often used as a static method. The method contains the code that produces a particular product. In example:

```
Product p = new ConcreteProduct();
```

is replaced by:

```
Product p = ConcreteFactory.createConcreteProduct();
```

Other uses are to encapsulate code when creation of a product is very complex, or replacing the creational fragment with a method having a sensible name to increase readability of the program code.

3.5 Facade

The facade pattern is a structural pattern. The name facade comes from the architectural facade, that is a exterior side of a building. In computer science, a facade is

a class that provides interface methods for the client to address the system behind it. The interfaces are high-level interface. The intention is to simplify the use the subsystem for the client. The client only operates on the facade, calls high-level interface methods, that can be translated from the facade into complex subsystem calls. Figure 3.6 gives a visual impression of the facade pattern by presenting the facade in front of the subsystem, as it is the case for architectural facades.

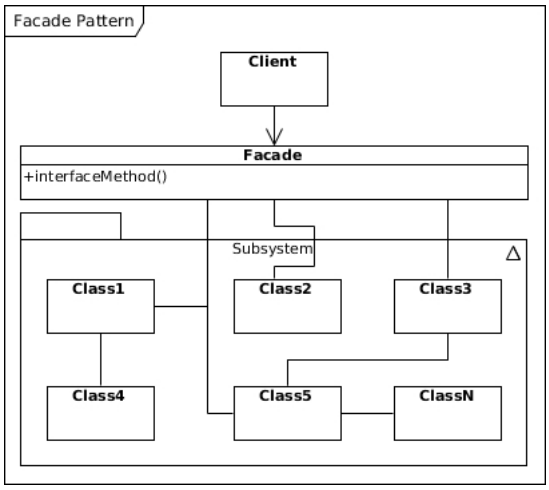


Figure 3.6: UML class diagram of the facade pattern.

3.6 Model View Controller

The model view controller (MVC) pattern is a software architecture pattern. The MVC pattern is popular for GUI applications and web frameworks. The concept is to structure the system into three parts:

1. Model
2. View
3. Controller

The model comprises all internal logic and data. The data is visualized by the view. The controller connects the view with the model in the way that actions originating from the view are forwarded or translated to methods of the model.

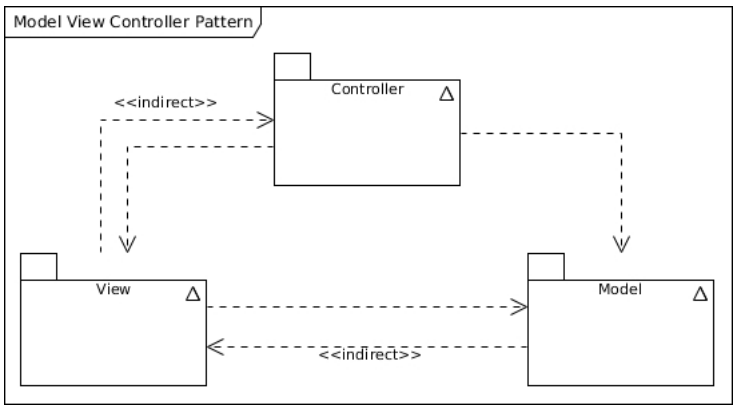


Figure 3.7: UML diagram of the mvc pattern.

There are many different options on how to implement the MVC pattern. However, as one can see in figure 3.7 there is generally the same control flow. The user interacts with the user interface in some way. The controller handles input events or actions and translates those into understandable actions for the model. Regardless of which communication structure, at some point the view queries the model to obtain its data to be presented by the view.

Chapter 4

System Design

There are several definitions of system design. The general idea is outlined by the following quotation:

> „Software Design is the practice of taking a specification of externally observable behavior and adding details needed for actual computer system implementation, including human interaction, task management, and data management details." (Coad and Yourdon) (Tay09)

System design is a procedure of determining a feasible solution to satisfy specified requirements. The result is the layout of a system, defining the system's architecture, components and data. System design follows the problem partitioning principle, which is an adaption of the divide and conquer principle. The process of the system design, therefore, involves two domains: Problem analysis (Section4.1) and problem solving (Section4.2 and 4.3).

Objective of the problem analysis is to determine requirements of the customer and provide groundwork for modeling and implementation. Requirements are examined, problems and sub-problems deriving from that are elaborated and input data is characterized.

Methodical problems—in particular image processing problems—are attended in the methodology 4.2. The methodology deals with the specification of image processing methods that are applied for assessment methods.

In the section design, the system is modeled. Components and modules are defined and put together. to produce the model that is realized in Chapter 5.

4.1 Problem Analysis

This section covers the problem analysis. Gathered information can then be used for decision making (CHK65). The complete system can be seen as the problem at hand. This problem is examined and divided into individual issues. Each issue is then handled successively, eventually resolving the problem. This section is divided into three parts: Requirements analysis techniques, requirements analysis, and data characteristics.

Section 4.1.1 provides an introduction to requirements engineering techniques that are used for acquiring requirements specifications. Brief descriptions of each technique

will provide the required knowledge to understand the approach that is taken in the requirements analysis.

The requirements analysis is addressed in the second part of this section. On the basis of functional requirements presented, further refinement and inspection will provide the groundwork for the requirements specifications.

In the last part of this section, problems that deduce from input data are analyzed and characterized.

4.1.1 Requirements Engineering Techniques

Knowledge about terms and concepts that are being used is essential for understanding the approach that is taken in the requirements analysis 4.1.2. All concepts and methods that are explained in this section are used there to analyze the requirements in detail.

4.1.1.1 Scenarios

Scenarios are an informal description of a planned flow of activities to accomplish a certain goal. The basic idea is:

> „Scenarios are stories. They are stories about people and their activities." (M. Carrol, 1999) (Car99)

Scenarios are written in the language of the customer. This procedure provides a customer-oriented approach of defining functional requirements. Instead of extracting the user requirements directly from customer dialog, the use of scenarios lower the chance of accidentally bypassing essential information.

4.1.1.2 Use Case

The principle of *use cases* comprises two slightly different terms: *Use case* and *business use case*. The meaning of both can be explained with the use of *scenarios*. Each step or action that is done by the user can be described as a use case. The use case itself, formalizes the flow of actions and steps that are executed by the system. The name of a scenario that describes its scope is the business use case under which use cases are assembled. The following quotation outlines the definition of tasks:

> „A use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result to an actor." (GBJ99)

In addition, Lauessen stated:

> „The definition only addresses the actions that the system (computer)—not the user—performs" (Lau03)

Use Case Diagram In the UML use cases are usually illustrated with the use of *use case diagrams*. Figure 4.1 shows an example use case diagram of the UML. Each use case of the system is displayed as a bubble within the frame of the system.

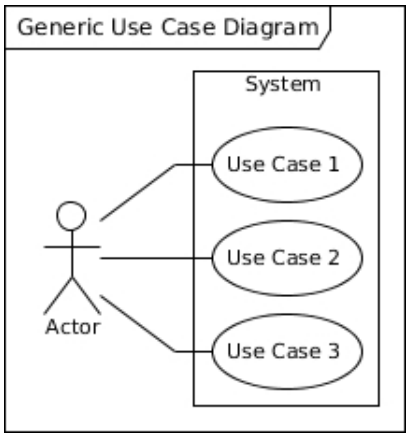


Figure 4.1: A generic use case diagram. Each use case is represented as a bubble.

4.1.1.3 Task

The term *task*—also called *user task*—refers to the concept of *task descriptions* that was developed by Marianne Mathiassen and Soren Lauesen (Lau01). Task descriptions are formalizing tasks that are performed by a user to achieve a certain goal. As with use cases, tasks descriptions can be derived directly from scenarios. However, tasks do not define a certain flow of activities as with use cases. Furthermore, tasks describe the actions that are carried out by the user and the computer/system together. Tasks can be defined on different levels. The term *high-level task* implies that there are other tasks that are part of—or included into—another task. High-level tasks are usually refined, working out "ordinary tasks" (Lau04) and possible sub-tasks. When they are derived from scenarios, high-level tasks are usually the aim of a scenario – as with business use cases. Essential—or say ordinary—tasks are encapsulated within the high-level task.

This differentiates tasks from use cases in the way that:

> „This approach does not divide the labor—that is a design issue to be dealt with later." (Laussen, 2003) (Lau03)

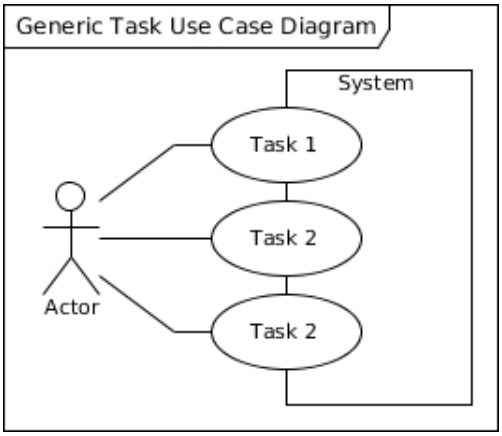


Figure 4.2: Adapted from Laussen (Lau03). Illustrates the difference between use cases and tasks by moving the task only partially within the borders of the system frame.

Figure 4.2 represents the illustration that Laussen used to support his proposition (Lau03). Refinement of tasks loosens the dependency from the user, transforming them into reg-

ular use cases. One can think of that as moving the bubbles from the outside into and within the borders of the system—making it a regular use case. Figure 4.3 illustrates the procedure of a task oriented approach of determining the system functions needed to fulfill the user wishes.

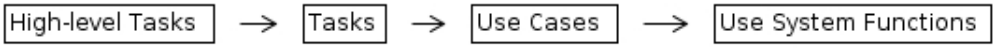


Figure 4.3: Illustration of task oriented approach of determining system functions.

4.1.2 Requirements Analysis

In this section, requirements of the customer are examined to get a basic understanding of the customers believes and wishes. Since the customer usually is not aware of distinct features that are needed, requirements analysis techniques are applied. This project uses a task oriented approach, that is explained in 4.1.1.3.

The customer needs the system for assessing ultrasound video material. Videos are recorded by a particular ultrasound device and saved separately. The customer wants to access the diameter of vessels displayed in each video with the use of the software. A scenario is used to determine the expectations of the customer. From the scenario user tasks can be deduced. In the following, the scenario for the use of the system is presented. The customer needs the system for one particular scenario. Table 4.1 shows the scenario that is considered.

The scenario contains all requirements from the customer to the system. The scenario tells how assessment of ultrasound videos shall be executed. Videos shall be loaded into a playlist. The customer wants then to be able to examine every video and set-up conditions of the measurement run. By that, he defines a ROI, sets a time window, and measurement marks. The *time window* defines the part of the video from which frame numbers and corresponding times shall be recorded. The measurement is taken between two measurement marks and shall be listed along with their frame number. The next step is to derive user tasks from the scenario, and thereby extract from the chronological sequence.

Table 4.2 displays the task derived from the scenario. The user wants to be able to load video files into a playlist. The task T1: Assess Diameter is a high-level task and its sub-tasks are the functional requirements to the system. Each task usually is described with an extra document. However, in context of this work, it is sufficient to simply list the tasks with a short description. The descriptions of tasks can be indicators that the task has sub-tasks. This is true for and task T1.2 and T1.4. Refining these two tasks results in the task list displayed in table 4.3.

The task T1.2 Examine video, comprises basic video playing functionality. To clear up the meaning of examine, another scenario could be presented here. This, however, would go beyond the scope this report. The scenario is dealt with in textual form: To examine a video, the customer wants to first play the complete video. It should be possible to pause and jump back and forth within the video for repeated investigation of parts of the video. Frame-wise navigation within the video is considered to be needed for convenient determination of the exact time of „cuff-release". The task list 4.3 is accurate enough for describing use cases and thereby specifying the requirements.

Scenario Name	Assess diameter
Actors	Operator
Flow of Activity	1. Operator starts the software 2. The operator loads one or many ultrasound video recordings 3. The operator examines the first video. 4. The operator uses the mouse to draw a rectangle in the video scene. 5. If the video is not a reference recording", the operator looks for point of „Cuff-Release" and sets start and stop time. 6, The operator looks for parts that are applicable to diameter measurement and sets measurement marks around them. 7. The operator can preview measurement results and obtain instant feedback from the software. 8. The operator repeats steps 3—6 until settings for all videos in the play list are applied. 9. The operator starts automatic measurement run, that analyzes every video in the playlist. 10. The software obtains measurement results, giving the user an extra file. 11. The operator waits until the measurement run has finished. 12. The operator closes the software and views the resultant file that shows three columns: Two columns where each frame number and the corresponding time is listed, and a third column where measurement results of frames—that are in between two measurement marks—are listed with respect to the row of the frame number.

Table 4.1: Scenario of assessing diameter measurements from ultrasound recordings.

4.1.2.1 Requirements Specification

In the previous section, requirements have been worked out and can now be specified as system functions. The typical way of specifying functionality is with the use of use cases. The tasks that have been worked out in the previous section can now be interpreted as use cases. Figure 4.4 displays the central use case diagram for the system. It can be seen that each task is displayed as a use case. Although not required by the customer, two additional use cases are added to the system for convenience reasons, namely: *Stop video* and *Delete video*.

The stop video use case is motivated by the ISO 9241-110[1] *Dialogue principle: Conformity with user expectations* (IOfS06), where it is demanded that a application should behave as a user expects it to do and conforms to well-established standards. As the system in some part behaves like a standard video player, standard video playing methods should be available. Furthermore, since „Stop video" is a special case of „Pause video" it is supposedly quick to be designed and implemented.

The delete video use case is motivated by the Dialogue principle: *Error tolerance* (IOfS06). When errors occur, the user should be able to still reach the intended goal,

[1] ISO stands for International Organization for Standardization. ISO 9241 covers principles for human-system interaction. Part 110 defines principles for demands of usability which apply to the design of visual dialogs of human-system interactions.

Task list		
T1	Assess Diameter.	*Assess diameter of vessel displayed in video*
T1.1	Load video into playlist.	
T1.2	Examine video.	*May arbitrary access every frame.*
T1.3	Define ROI.	*By drawing rectangle in video scene.*
T1.4	Set time window.	*Set start and stop times.*
T1.5	Preview measurement results.	
T1.6	Set measurement mark.	
T1.7	Start automatic measurement.	

Table 4.2: Task list derived from scenario displayed in figure 4.1.

Task list		
T1	Assess Diameter.	*Assess diameter of vessel displayed in video*
T1.1	Load video into playlist.	
T1.2	Examine video.	*May arbitrary access every frame.*
T1.2.1	Play video.	
T1.2.2	Pause video.	
T1.2.3	Go to next frame.	
T1.2.4	Go to previous frame.	
T1.2.5	Go to specific frame/time.	
T1.3	Define ROI.	*By drawing rectangle in video scene.*
T1.4	Set time window.	*Set start and stop times.*
T1.4.1	Set start time.	
T1.4.2	Set stop time.	
T1.5	Preview measurement results.	
T1.6	Set measurement mark.	
T1.7	Start automatic measurement.	

Table 4.3: Refined task list of 4.2. Sub-tasks for tasks T2 and T4 have been added.

either with no – or minimal effort. Suppose the user inadvertently loads video files he didn't want to. Instead of restarting the application and loading the video files correctly, it would be easier for the user to simply delete the wrongly loaded file.

4.1.2.2 Virtual Window

In software engineering, *virtual windows* are used to specify data requirements of use cases. Virtual windows look like mock-ups[2] as they provide a UI. They also can be used to give the customer an early impression of what the software will look like. Virtual windows are useful in the systems design as early visual design decisions are made and can be realized by a GUI.

In figure 4.5, the virtual window is presented that was planned with consultation of the customer and further used for realization. It was used to prevent different understandings of what the software would look like and ensure that the customer receives the software that was envisioned.

[2] A mock-up is a drawing on screen or paper, or a early *prototype* of the User Interface (UI). Prototypes are realized GUI that may provide functionality to some extend. Mock-ups are useful, because implementation of GUI can be a time consuming task. If the customer does not like the GUI is feels some feature missing, changes can be applied, early.

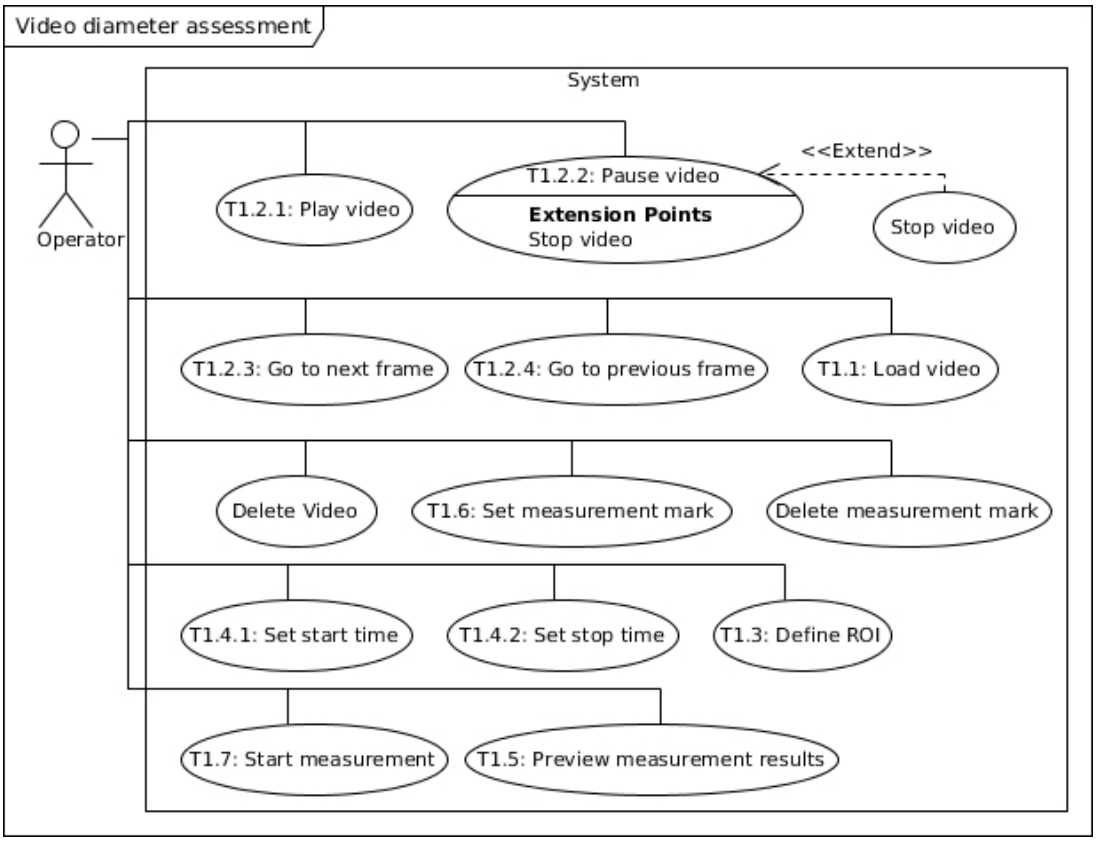


Figure 4.4: Use case diagram displaying all steps that the operator executes for video assessment.

The virtual window realizes the requirements that are worked out in figure 4.4. Use cases are converted into buttons or other interactions. There is a playlist that holds videos as entries. Files can be loaded and deleted using the menu. At the bottom of the window are buttons for most of the use cases. The *Scene* displays the frames of the current video. Below the scene, is the time line of the video. There is a check box next to the buttons. When it is checked, the vessel edge detection is displayed in the video using dots or lines along the vessel walls. Furthermore, the customer specified in the scenario (see figure 4.1) that the ROI is defined by drawing an rectangle in the video. This is also applied by the virtual window. The lower row of buttons are used to set start and stop times of the time window and to set or delete measurement marks. The time window and the measurement marks of the current video are represented along the time line.

4.1.3 Data Characteristics

The purpose of the software is to measure the diameter in ultrasound videos. Every frame of the video needs to be analyzed individually with image processing methods. It is important to examine the image material, to be able to make decisions about the application of image processing methods. Information worked out in this section is used for determination of the image processing procedure in Section 4.2.

Figure 4.6 is a sample video frame of the ultrasound video supplied material. The graphical output of a particular ultrasound device is displayed. It contains the ultrasound image embedded into a user interface. The ultrasound image, in every frame, is located on the same position of the output, at (680, 125). The user interface provides additional

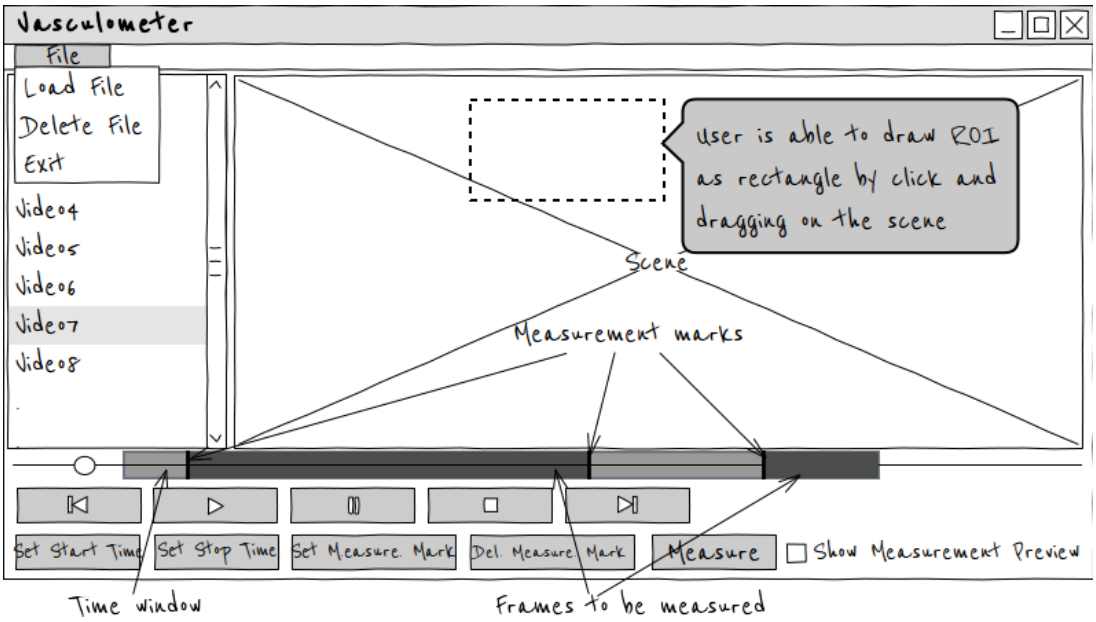


Figure 4.5: Virtual window of system

information about the ultrasound signal. Important parts of the image are marked in the image. The ratio scale of the ultrasound image is framed by a green dashed line. The red arrow points to the uppermost mark of the scale. Although, the ratio scale differs from image to image, it is important to note that this particular mark is always at the same position in all images. This information can be used later for automatic determination of the scale factor. The vessel walls are displayed as two bright nearly horizontal lines in the image. These are marked with two blue dashed frames around them. The upper line is usually referenced as the near wall and the lower line as the far wall. This refers to the relative position to the ultrasound transducer probe. The orange arrow points to the so called lumen. The lumen is the dark structure between the vessel walls—the inner of the vessel. The purple arrow points to the colored cursor, that is used as a reference for the Doppler measurement. The cursor can be moved by the operator. However, to perform the Doppler measurement, it must be positioned in the lumen.

In general, ultrasound images suffer from noise. Ultrasound images differ strongly from subject to subject. A reason for this being the individual habitus of each person. High amount of tissue around the vessel usually leads poor quality of the images of ultrasound scans, since the acoustic waves have to pass through more material. Bad quality of ultrasound images is noticeable by high amounts of noise in the image. Typical and most noted kind of noise is the so called speckle noise (JYX+10, DMG+07, LDoCS05). Speckle noise causes the typical appearance of structures in ultrasound images (see figure 4.7).

Another kind noise appears due to blood backscattering the ultrasound waves. Figure 4.8 and 4.9 display examples of blood backscattering. The backscattering of the blood determines a distribution of gray tones in the vessel lumen (see figure 4.8). In terms of excessively high blood backscattering this may also affect the texture of the far wall, which has a

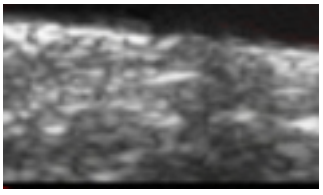


Figure 4.7: Typical ultrasound speckle

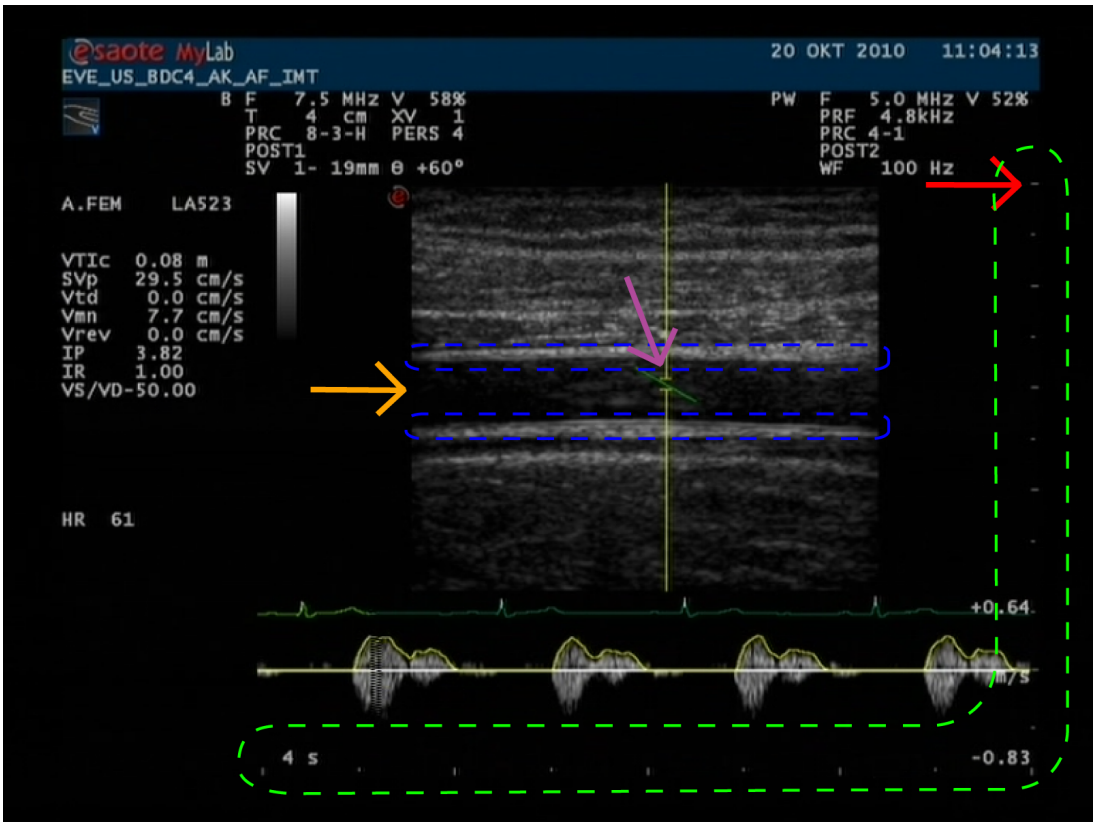


Figure 4.6: Sample image of the output from the ultrasound device. Blue dashed lines enframe vessel walls, green dashed line enframes ratio scale marks. Orange arrow points to lumen, purple to cursor, red arrow points to the uppermost scale mark, that resides on the same position in every video.

lower resolution (see figure 4.9). However,
a trained operator can reduce the amount of backscattering by properly setting up the ultrasound equipment and keeping the amplifiers gain of the acoustic channels as low as reasonably possible (DMG+07).

Other sources of disturbance of the ultrasound signal are echo shadows (JYX+10)(GAGHL97). Echo shadows may appear behind strong reflecting vessel walls or when tissue is absorbing the ultrasound signal (see figure 4.10).

Due to artifacts and noisy appearance of the ultrasound images, determining fine structures—such as vessel walls—poses a difficult task. The gray structure displayed around the vessel walls are different kinds of tissue. There are also tendons displayed in the ultrasound image. When positioned close to the vessel walls and if those do not generate a strong contrast in the image, an untrained observer might not be able to distinguish between tendons and vessel walls. Figure4.11 shows an example of a documented scenario. The vessel wall is hardly visible while the tendons appearance is displayed very clearly.

The red arrow points to the vessel wall, while an untrained operator could easily determine the tendon, that is pointed to by the blue arrow, as the the vessel wall.

To reveal the effects of noise more clearly, figure 4.12 presents a three-dimensional surface plot. The height of a point p at (x, y) represents the intensity of a pixel with the same coordinates (x, y) in the source image. The source image is cropped from the ultrasound interface displayed in figure 4.6.

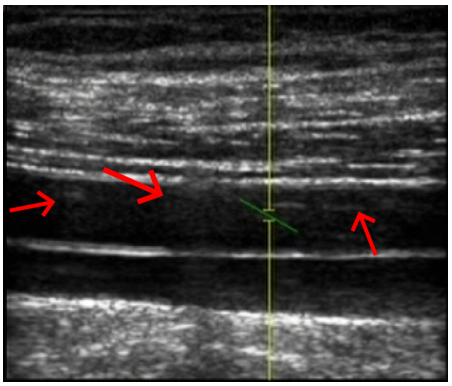


Figure 4.8: Blood backscattering of the ultrasound signal. Red arrows point to various occurrences of blood backscattering.

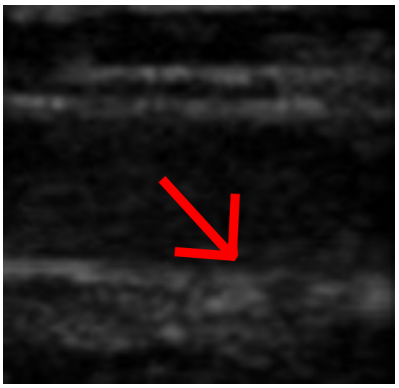


Figure 4.9: Red arrow pointing to intensive blood backscattering that is texturing the vessel wall.

The presented plot can be pictured as a landscape. The typical speckle noise is then observed by the many hillocks in the image. Strong structures, however, appear as mountains. It becomes clear that the lumen itself is an inhomogeneous structure. However, when looked at with respect to the mountains next to it, the lumen is usually „lower"—meaning darker—than the vessel walls. Same can be observed by the intensity profile shown in figure 4.13. The profile is taken from a column to the left of the Doppler cursor. Yet again, the lumen has a lower intensity than the vessel walls. As the intensity of pixels of the lumen is usually lower than the intensity of pixels from the vessel walls and following the typical pattern of the speckles, the noise in the lumen can be distinguished from the vessel walls by the gradient magnitude.

4.1.3.1 Assumptions and Conditions

For measuring the diameter between parallel lines, the surface normale needs to be calculated. The length between the intersection points at both ends is then the diameter. However, when the ultrasound equipment is not properly positioned, the vessel walls can appear tilted or possibly even bent. Concerning this, obtaining the diameter is a more complex task. For that reason, an assumption is made. The operator has to assure that the tilt of the vessel is within the extent of tolerance of ± 5 degrees. The diameter is measured in vertical direction. Therefore, if the vessel is displayed tilted in the image then the resulting measurement is erroneous (see figure 4.14). After the vessel wall edges have been detected, this assumption can be verified. The expected error

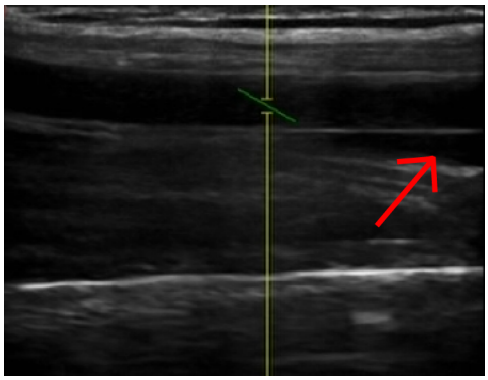


Figure 4.10: Absorbing tissue puts an echo shadow behind the vessel wall (red arrow).

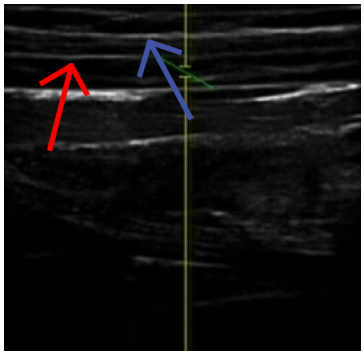


Figure 4.11: Demonstration of confusability of tendons and vessel walls. The red arrow points to the vessel wall and the blue to a tendon.

can be calculated. Given the equation:

$$\cos(\alpha) = \frac{b}{c}$$

Where b is the adjacent side and c the hypotenuse of the triangle, the length of c can be obtained by rearranging for c. The difference to b is then obtained by:

$$\Delta c = \frac{b}{\cos(\alpha)} - b$$

It is guaranteed that the vessel on the ultrasound images will not be angled by more than 5 degrees. The difference to b, which here is normalized, is therefore limited:

$$\lim_{\alpha \to 5} \frac{1}{\cos(\alpha)} - 1 \approx 4 \times 10^{-3}$$

The scale factor on the images is roughly 35—60 pixels per millimeters and the average diameter of the vessel is 3—8 millimeters. This means that one pixel represents 0.09 to 0.13 millimeters. The error in measurement will then be from 0.01 mm up to 0.03 mm and is about 9—4 times smaller than one falsely registered pixel by the edge detection algorithm. Even if the vessel would be angled by 10 degrees, the error in measurement would still be as small as one falsely registered pixel. This error is therefore neglectable.

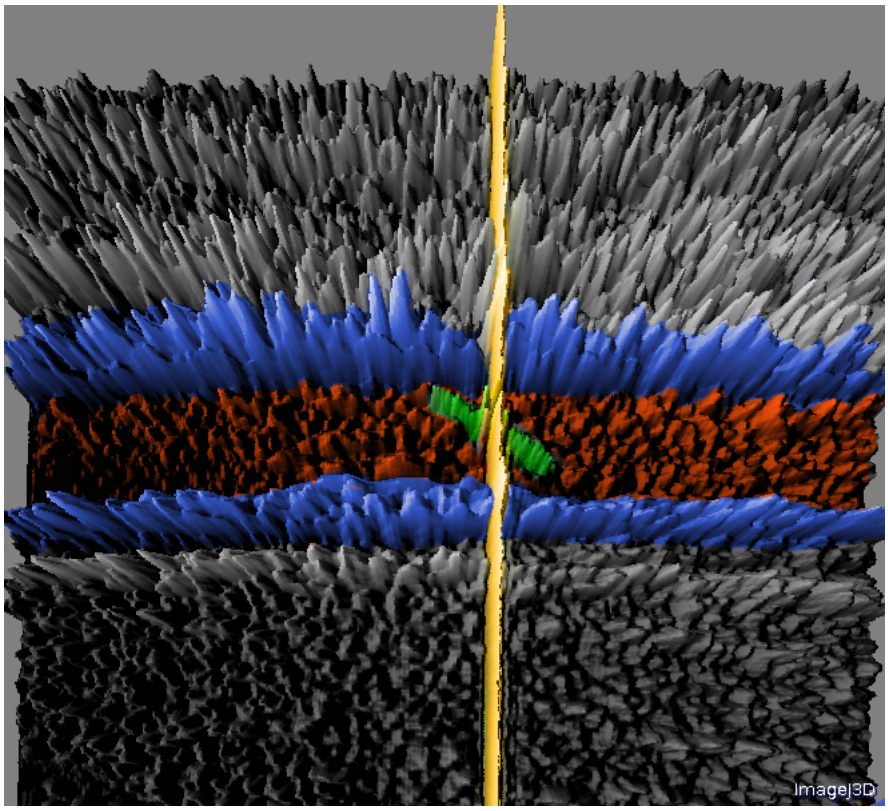


Figure 4.12: Representation of the cropped ultrasound image of 4.6 in three spatial dimensions. The vessel wall is marked blue, and the lumen red. The cursor is well observable due to its original yellow and green color.

4.2 Methodology

In section 4.1.3 the image material is examined and specified from an image processors point of view. On the basis of this information provided, appropriate image processing methods can be determined. The basic idea to perform vessel detection and measurement is to first apply edge detection methods, then determine the position of the edges on the vessel walls and then measure the diameter between the two. The first part of this section describes the methodical approach and techniques that are applied to detect the vessel walls. The second part describes how measurement of the diameter is performed.

4.2.1 Vessel Wall Detection

This section describes the vessel wall detection algorithm that is chosen and how it is applied. The chosen procedure is arranged into three steps:

1. Determine a point in the lumen
2. Apply an appropriate edge detection filter to detect the edges on the vessel walls
3. Locate the edges with a search algorithm from the point in the vessel lumen.

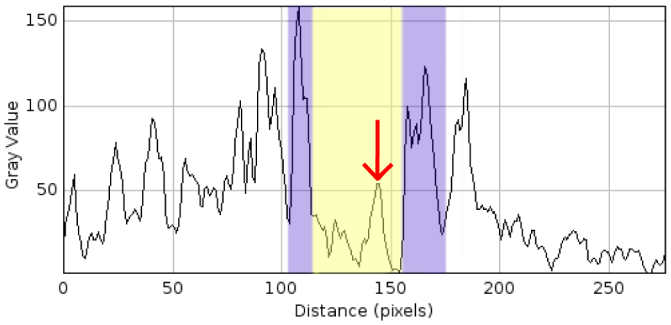


Figure 4.13: Intensity profile of a column left from the Doppler cursor of figure 4.13. The marked purple areas are representing the vessel walls and the yellow area is the vessel lumen.

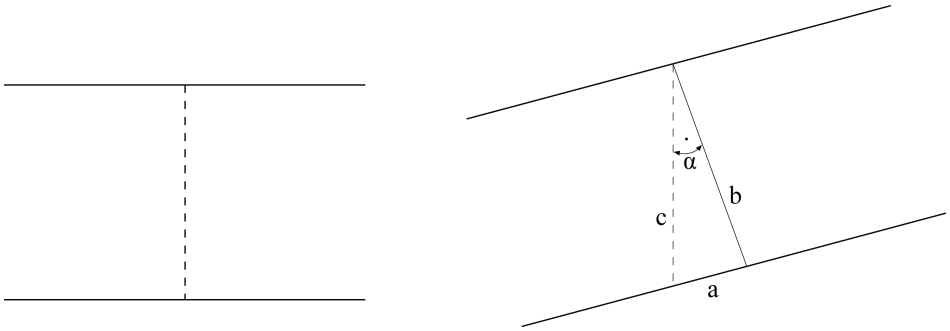


Figure 4.14: Simplified illustration of true (continuous) and measured diameter (dashed).

4.2.1.1 Determination of a point in the lumen

Since, the colored Doppler cursor is positioned by the operator within the vessel lumen, the initial idea was to determine the position of the lumen by looking for the cursor that is displayed in the image. A pixel belonging to the cursor can be determined by its color. Figure 4.15 illustrates the procedure that is used to determine the position of the cursor.

Since the images differ in contrast and brightness, the color of the cursor needs to be approximated; Starting at a certain value and then loosening the limit if nothing was found. For a human, the Hue,Staturation, and Brightness (HSB) color mode is the more natural way of representing colors, the HSB color mode is used for determination whether a pixel is green or not. The pseudo code is displayed in 4.1.

The algorithm checks all pixels in all rows of the image. If the color criterion is matched, the pixel is marked. When both loops are finished, the algorithm checks if pixels have been marked. If not, the criterion is loosened by extending the range of acceptance.

Later in the project it was found that, the cursor is not properly positioned within the lumen in all videos. Therefore, the image processing algorithm produced false results. Because of time pressure, development on this approach was dropped.

Instead the center of the ROI, that is set by the operator, can be used to determine the vessel lumen. Since it enframes a part of both vessel walls, it is almost certain, that the center of the ROI is positioned in the vessel lumen.

Figure 4.16 displays two examples of ROI selections. Selection (1) is an example for a selection of the ROI where this assumption is not true. Even if the operator initially

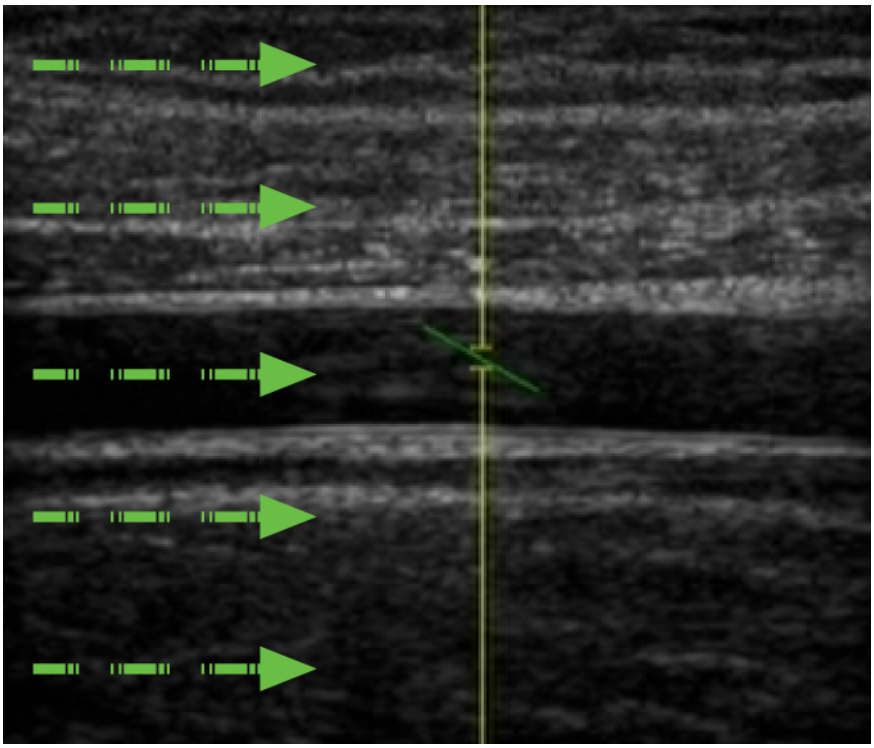


Figure 4.15: Represents the search algorithm for the cursor. The algorithm seeks from the left side of the image for pixels with green colors.

sets a ROI as in (1), since the software provides immediate feedback, the operator will change the selection appropriately. A selection that is then likely to be chosen is as selection (2). This approach is based on the operators selection. Therefore, measurements obtained from different operators or at different times can differ.

4.2.1.2 Edge detection

As only a short time was available for the project it is reasonable to apply an image processing method that matches the authors previous experiences. Hence, a image differentiation method, was chosen, which has previously been used for similar applications.

Image differentiation methods try to mime human vision. It can be assumed, that a human operator determines the edge of a structure at the point of the maximum gradient. There is no sophisticated description of how a vessel wall and its edges are defined. Literature found on this matter only documents the visual appearance of the vessel walls in terms of human vision. Thus, one can assume, that the edge of a wall shall be determined on the point where a human operator would determine the edge. Furthermore, most applications of vessel diameter measurement, compare measurement results among results of the same device or method. As with the assessment of FMD, measurement results are compared over time.

Due to the noisy appearance of ultrasound images, a filter that expects rather smooth edges and has a big filter matrix is advised. For example, when applying the vertical sobel operator (see figure 4.2.1.2), one can easily spot a lot of false positives in the lumen (blue arrow) . Furthermore, the sobel operator responds to a lot of edges, as can be seen by observing the fuzzy edges that are not well detected as a unit, but as individual edges (red arrow). Also not desirable is that the green cursor is also detected.

Algorithm 4.1 findCursor

$lowerThreshold \leftarrow 130$
$upperThreshold \leftarrow 140$
for $i \leftarrow 0$ **to** $height(Image)$
do { **for** $j \leftarrow 0$ **to** $width(Image)$
do { **if** $lowerThreshold \leq HSB(pixel(column, row)) \leq upperThreshold$
then $save(pixel(column, row))$
else $continue$
if $noMarkings()$
then { $lowerThreshold \leftarrow lowerThreshold - 1$
$upperThreshold \leftarrow upperThreshold + 1$

The aim is to produce high responses on the vessel walls while suppressing responses within the lumen. The reason that sobel operator produces such results can be seen from its kernel:

$$Sobel\ operator = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

(WB07)

The sobel operator is sensitive to most kind of edges. It emphasizes the change from pixel to pixel, thereby taking only three pixel rows into account. Detecting the fuzzy edges, as well as suppressing the noise in the lumen and the cursor, can be achieved with the use of a bigger and smoother kernel. In a way, the sobel operator can be thought of a 3×3 approximation of the first order derivative of a Gaussian filter.The derivative of a Gaussian filter is influenced by two variables: the kernel size k, and the standard deviation σ (sigma). A high sigma corresponds to smoothing the gradient of the Gaussian function (see figure 4.18), while the kernel size should be increased appropriately.

To prevent the filter to react sensitive to the noise in the lumen, a rather big kernel and sigma for convolution must be used. In case of this project, a sigma of 7 and the a kernel size of 29 is found appropriate to filter the image. Figure 4.19 shows an example of a resulting image, when such filter is applied.

The big sigma and kernel sizes vanish noise and cursor in the lumen, whereas edges and the vessel walls are emphasized. The convolved image displays the near (upper) edge in black. This is the negative response of the Gaussian filter kernel. The edge on the far wall is displayed in white. The kernel, in this case, generates a positive response. If the orientation of the kernel would be inverted, the resultant image, would also be inverted. The next step is to determine the exact position of the vessel wall edge.

4.2.1.3 Locate edges

Now that the image is convolved, the next task is to locate the edges from the detected point in the lumen. The algorithm that is used to perform this task scans the image, from a certain starting point for the minimum and maximum pixel up- and downwards, respectively. The algorithm also utilizes a threshold as a lower limit to prevent

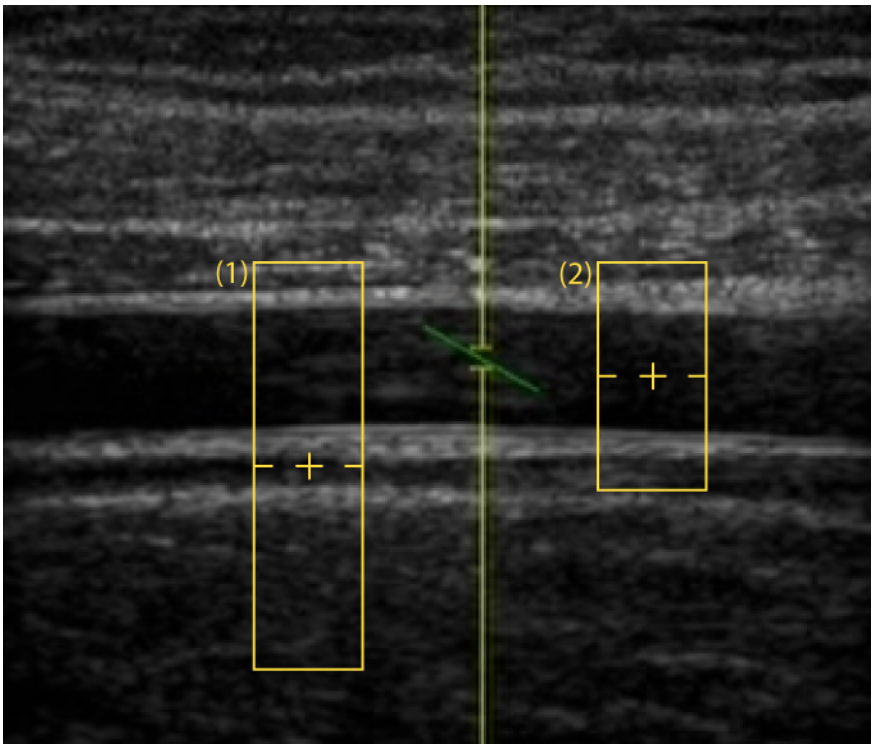


Figure 4.16: Representation of a bad (left) and good selection (right) of ROI.

false positives in the lumen. Figure 4.20 illustrates the procedure. Algorithm 4.2 gives a formal definition of the algorithm in pseudo code.

The algorithm 4.2 operates only in the ROI. It starts from the left of side of the region of interest. A certain height y is given. This is the height of the previously detected lumen point in section 4.2.1.1. The algorithm seeks stepwise in the right direction, and at each step looks upwards and downwards for the edge. In general, the idea is to scan upwards for the darkest pixel and in the opposite direction for the brightest pixel. The algorithm tries to „jump" over noise in the lumen. That means, in case of seeking upwards, the algorithm allows a temporary rising of pixel values. Although, seeking in the opposite direction, this often results in „jumping" the actual edge. Therefore, „jumping" is not allowed when seeking downwards.

Now, that the vessel walls have been detected, and before continuing, the quality of the edges shall be improved by sorting out deviant detections. The standard deviation of the points in an edge is computed and points below and above the mean $\pm$ standard deviation are eliminated. Theoretically, this could result in eliminating all the correct findings of an edge, leaving all the bad ones. However, noise usually appears to accumulate in segments of the image, proper and ROI selection prevents such cases.

The resulting image represents the information and is displayed as it is in the software program. It displays the edge that is detected and, thereby gives immediate feedback. Figure 4.21 displays a result image after the edge detection was applied.

Figure 4.22 displays a malfunction of the algorithm. It can be seen, that the points that are detected, differ from the actual edge on the far vessel wall. This is because of different factors summing up to a bad result. At first, the thin structure of the vessel wall can be observed. Furthermore, and although hardly visible, there is a little bit of noise in the lumen. Another influence is the echo shadow behind the vessel wall. The main influence however is presumably the low contrast of the vessel wall with respect to the

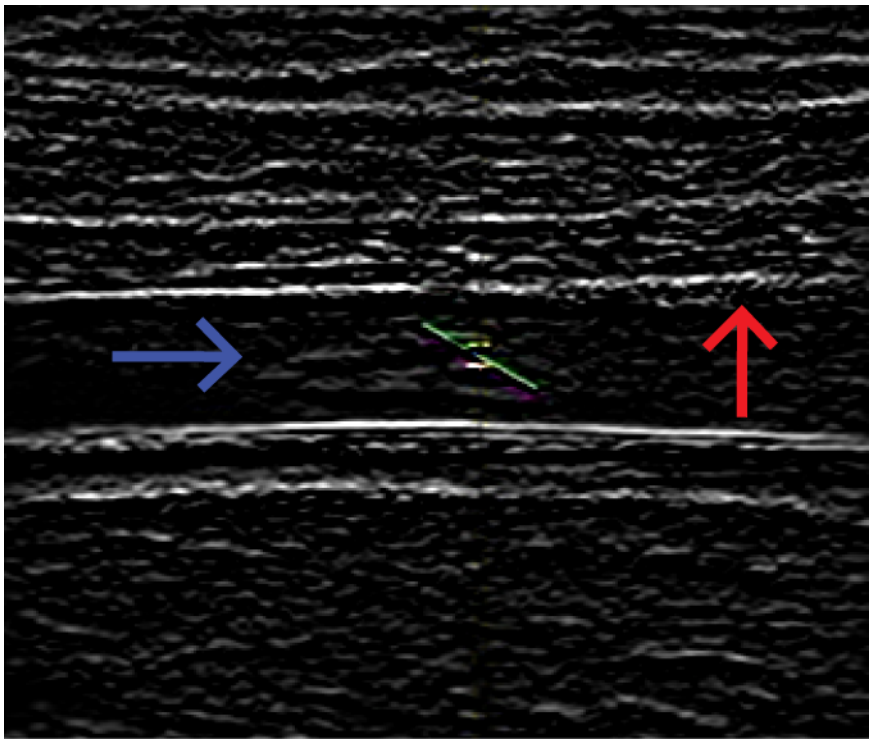


Figure 4.17: Sobel edge filter applied to an ultrasound image.

lumen. This is the main difference to the image in figure 4.21. A solution for such bad detection could be to use a longitudinal filter kernel. Such filter would take the structure to the left and right more into account and thereby suppress the strong effects of the low contrast edge.

Unfortunately, there was not enough time available during this project to follow this approach. This is something for the future outlook.

4.2.2 Diameter Measurement

Once the vessel walls are detected, the diameter between these walls can be calculated. There are different ways this could be accomplished. The simplest way to calculate the diameter is to calculate the difference in height between each point of the vessel. In section 4.1.3.1 assumptions about the apperance of the vessel walls that are made described. These assumptions have to be verified, before actual measurement can be performed.

Assumptions Verification and quality assurance As stated in the assumptions & conditions in section 4.1.3.1, it is presumed that the vessel walls are represented as two nearly horizontal lines in the image. This assumption is checked with the use of a linear regression analysis. The angle of linear function that is provided by the linear regression is computed. The angles of both edges then can be checked. The edges must not be checked individually. Even if both edge's angle is below 5 degrees, they could sum up to being almost 10 degrees when angled in different directions. If there are angles a and b of the edges, both having an value that is *just* within range of acceptance, that means ± 5 degrees converges towards zero, the result would be ± 10 degrees:

$$\lim_{|a|\to 5} a - \lim_{|b|\to 5} b = \pm 10$$

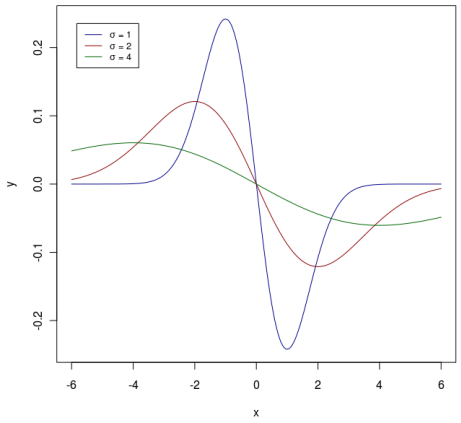


Figure 4.18: Representation of three graphs of the first order derivative of a Gaussian.

Therefore, the assumption must be checked for the difference of the of a and b:

$$a - b \leq 5$$

Apart from checking the assumptions, the linear regression provides additional information about the quality of the edges that are detected. The linear regression provides the correlation coefficient. If the coefficient is near ± 1, then the detected points of an edge are arranged in a straight line. The closer the correlation coefficient magnitude is to zero, the lower is the „correlation" of the points, meaning they are not arranged in a straight line. Since the vessel wall is displayed as a nearly straight line, this coefficient is of interest and gives information about how well an edge has been detected.

Furthermore, the standard deviation can be used to make a similar observation without giving information about the orientation of the appearance of points. The standard deviation of the height of all points is computed and it is decided whether the detection on the edge on the wall was sufficient or not. In this case, it is checked if the standard deviation is ≤ 1.

Measurement Since there are now edge points detected, the measurement of the diameter can be performed. The scale of the image needs to be obtained and then the distance of both edges, the far and the near vessel wall are calculated.

In section 4.1.3, it was stated that the upmost point of the ratio scale marks, that is displayed in the interface output of the ultrasound device, is always at the same position, that is to say at (680, 125). This information is used to obtain the distance from the upper, to the next scale mark. The first two points of the scale define a distance of 5 millimeters. The scale factor that is normalized for one pixel can then

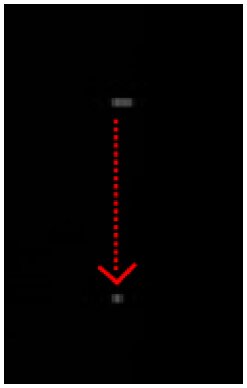


Figure 4.23: Illustration of the procedure to obtain the scale from the image.

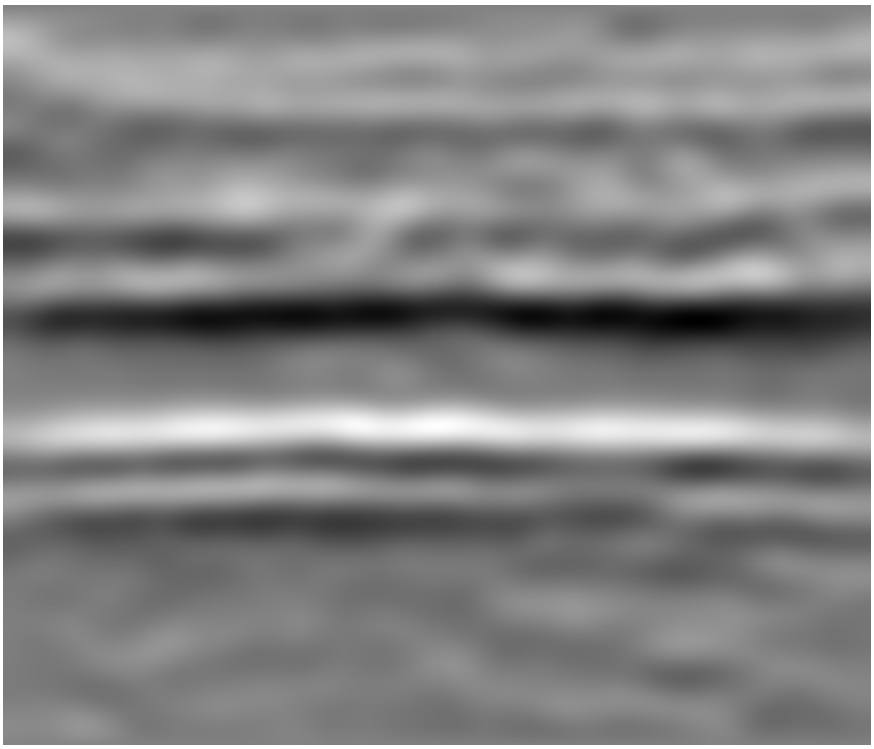


Figure 4.19: First order derivative of a Gaussian applied to an ultrasound image.

be calculated. Figure 4.23 represents the
way, how the ratio scale is obtained.

Algorithm 4.3 shows the pseudo code of how the scale is obtained from the image. Starting from the uppermost point of the scale, the next scale mark is searched and then the scale factor is returned. The distance of the two scale marks represents 5 millimeters. Therefore, the scale factor is calculated by dividing 5 by the distance of the two scale marks. This procedure is done in every frame. This is necessary due to disturbances. Some frames in the video might not display the scale marks and a correct determination of scale factor can not be guaranteed.

In section 4.1.3.1, it is argued, that since the edges are assured to be tilted by less than 5 degrees, the error in measurement when measuring the horizontal distance, can be ignored. This assumption is realized in this step. In order to perform actual diameter measurement two steps are done. First, the average heights of both, all edge points of the near edge and all edge points from the far edge are calculated. This is done in order to achieve sub-pixel accuracy. The difference of both those heights is then calculated and converted into millimeters using the previously determined scale factor.

4.3 Design

This section defines the design of the system. A top-down approach is undertaken. First, the high-level design divides the system into components and describes them. Then the high-level design is used to layout the system model, providing architecture and all modules required of the system.

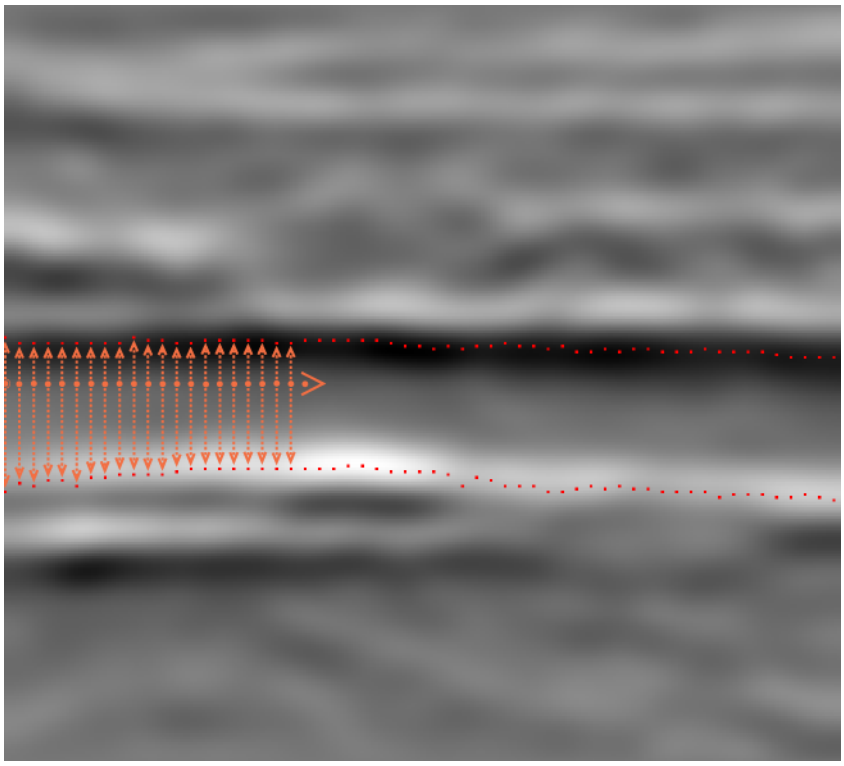


Figure 4.20: Illustration of algorithm 4.2. The red arrows represent the way that the algorithm functions.

4.3.1 High-level Design

This section gives an introduction to the high-level design of the system. Typically, early decisions are made. In this case, decisions about the media frameworks and image processing libraries are made. In the following, the chosen high-level design is presented. After that, components are described and decisions justified. Concluding, the data flow between the components is explained.

Figure 4.24 shows the components of the system and their dependencies. On a whole, the system is divided into two domains: Application and libraries. The blue area represents the application layer. It is made up of the four components: User Interface, Video Player, Data Handling, and Processor. The green area represents the libraries that are used: Java Media Framework (JMF), ImageJ, and FeatureJ. The yellow and red areas signify which components interface both domains.

Java Media Framework

Due to the specification, a video playbackfunctionality is needed. Since Java does not provide functionality to read video material on it's own, a media framework is needed. The framework needs to enable the system to not just play video files, but process them pixel by pixel. This feature requires functionality that is not provided by all frameworks available. This project uses the JMF that supports all required features.

The JMF was originally developed by Sun Microsystems. It provides the ability to decode, play and process time based media, this includes videos. JMF was always criticized for the lag of up-to-date formats and codecs. In 1999 Sun dropped further API development of JMF. Since 2010 the JavaFX Java platform was released. Although

Algorithm 4.2 findEdge

```
threshold ← 3 × pixel(0, y)
i ← 0
while i < width
  do { switchcount ← 0
       for j ← y to 0
         do { current ← pixel(i, j)
              if switchcount = 0 and last ≤ current
                then last ← current
                else if switchcount = 0 and last > current
                then { switchcount ← switchcount + 1
                     { last ← current
                else if switchount = 1 and last ≥ current
                then last ← current
                else if switchount = 1 and current > last and current > threshold
                then break
       for j ← y to height
         do { switchcount ← 0
              last ← 0
              for j ← y to height
                do { current ← pixel(i, j)
                     if current ≥ last
                       then last ← current and continue.
                       else if current > threshold
                       then break
       i ← i + intersection
```

video playback of most modern formats is supported, it is not possible to process the videos. This means, one cannot read each frame to process it pixel by pixel. However, because JMF provides this functionality it used to provide ImageJ with each frame of the videos being processed despite the lag of supported formats. To overcome this issue, the user has to use a script to transcode the video files first, before opening them with the software. A script that uses FFmpeg[3] to enable the user to do so was already provided.

Video Player

The video player component instantiates certain modules from the JMF to access supplied video material. For the use of this application, it is needed to access each frame individually rather than providing a data stream that is handled as a „black box" by other components. When there is a new frame available, it notifies the GUI of its presence, and provides the Frame when asked for.

[3] FFmpeg is an open source project that provides libraries and programs such as ffmpeg (a command-line tool of the same name) for transcoding.

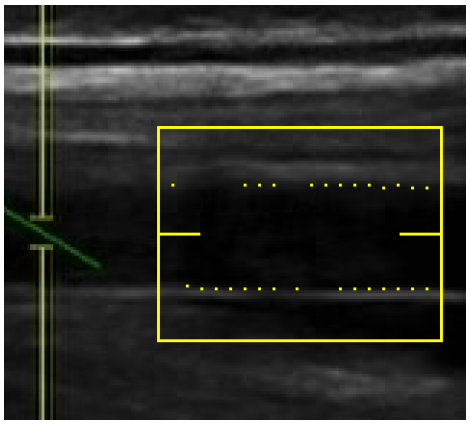


Figure 4.21: Resulting image of vessel wall detection.

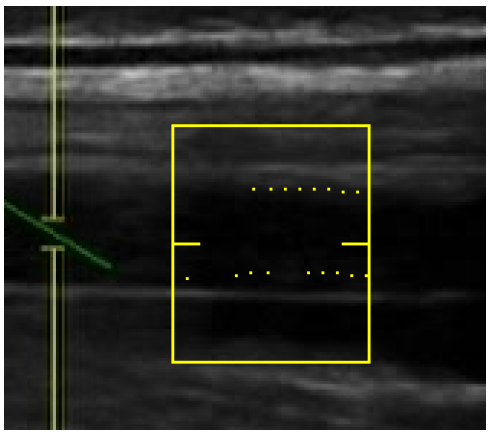


Figure 4.22: Presentation of malfunctioned edge detection.

Processor

The processor is used by the video player and gets forwarded each frame to be analyzed. The processor is one the core component of this project. Image processing methods are applied to determine the blood vessel walls. The processor uses the ImageJ and FeatureJ libraries to process each video frame. Information about both walls is forwarded to the data handling component.

Data Handling

The data handling component comprises modules for different applications. The playlist needs to be stored to be used by the video player and Information about vessel walls submitted by the processor. The data handling component stores measurement results and saves them to an extra file for each video.

Graphical User Interface

The GUI component consists of all user interacting modules. It shall provide the user with the required input methods. Furthermore the video sequences are displayed by the GUI. This requires a certain amount of cooperation with the video player component, although, both components shall be left as independent as possible. The GUI obtains each video frame from the video player and renders it to a video scene.

Algorithm 4.3 getScale

$\leftarrow 680$
$y \leftarrow 125$
$threshold \leftarrow 50$
while $y < height$
do $\begin{cases} \textbf{if } pixelbrightness(x, y) < threshold \\ \quad \textbf{then } \begin{cases} startY \leftarrow y \\ break \end{cases} \end{cases}$
while $y < height$
do $\begin{cases} \textbf{if } pixelbrightness(x, y) \geq threshold \\ \quad \textbf{then } \begin{cases} endY \leftarrow y \\ break \end{cases} \end{cases}$
return $(5 \div (endY - startY))$

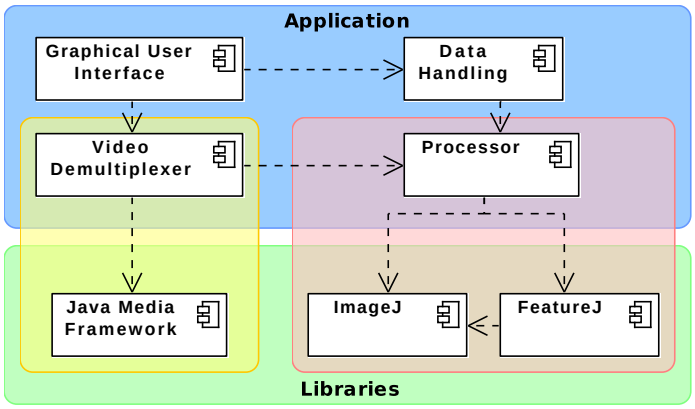


Figure 4.24: High-level design of the software: The system's internal structure (blue area), used libraries (green area), and high coupling between both domains (yellow and red areas).

ImageJ

ImageJ is an image processing software produced by the National Institutes of Health. Because of the plug-in architecture, ImageJ is nowadays often used as a teaching platform for image processing. It allows the use of a wide range of plug-ins implementing common image processing algorithms. On one hand, this decreases the amount of time spent for implementation and on the other hand this allows to try out various image processing techniques before worrying about implementation difficulties.

FeatureJ

FeatureJ was Initially developed by Erik Meijering for die „Biomedical Imaging Group"(BIG) at the Polytechnic University Lausanne, Switzerland. The BIG conducts research to develop new algorithms and mathematical tools to enhance the processing of medical and biological imaging. FeatureJ is a package of plug-ins for ImageJ to provide feature extraction. It provides a collection of implementations of well-known methods.

4.3.2 System Model

The first step towards accomplishing the requirements specification (section 4.1.2.1) is done in the high-level design (section 4.3.1). Components are defined in the high-level design. This section refines these components by defining modules and associations among them. At first, the architecture of the system is defined. After that, the architectures modules will be worked out.

4.3.2.1 Architecture

Architecture design is an important step in system design. Realization of high-level components from scratch can lead to complex and confusing modelling. Furthermore, the system shall be able to be open for future extensions or adaptations to different applications. Since refactoring of a system can be a time consuming task, it is reasonable to avoid design errors in the first place. Architectural design can help with the layout of the system by taking different requisitions in account. The high-level design defined components. In the architecture design these components are examined. Modules are constructed and a structure to fulfill stated requisitions and components is defined.

Since the system is an application providing a GUI that displays data, while a lot of data is being processed in the background, it is a good idea to apply the MVC pattern. The MVC is an architectural design pattern where internal logic and user interface are strictly isolated (see section 3.6). The view consists only of the GUI component. All other components are comprised within the model.

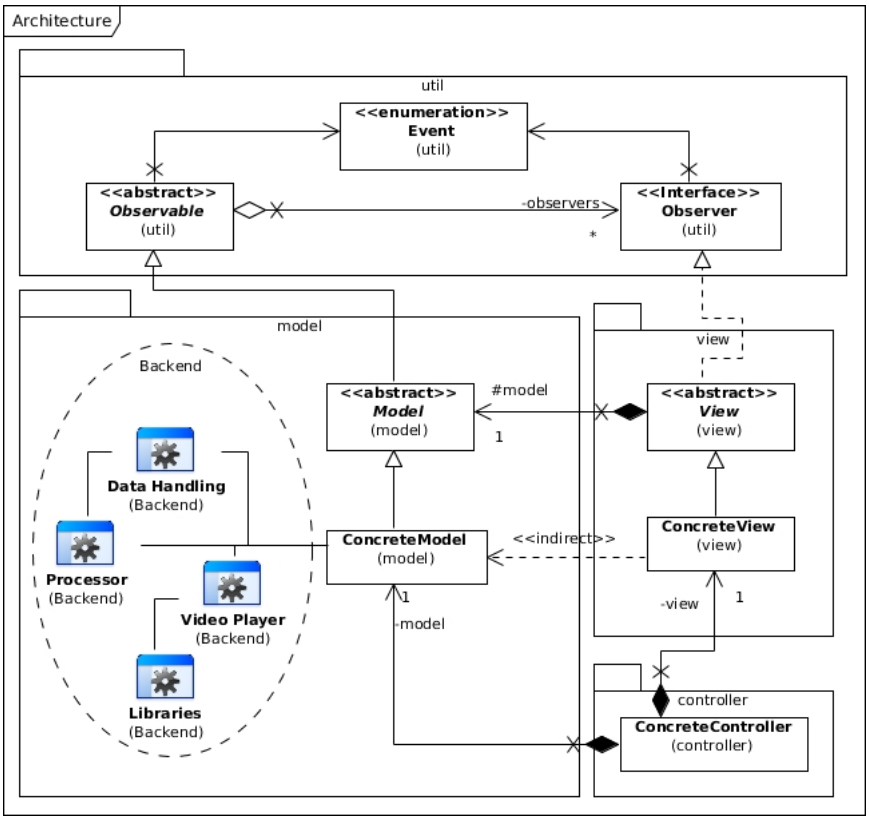


Figure 4.25: Class diagram of the architecture of the system.

The system architecture is displayed in figure 4.25. The domains model, view, and controller are clearly separated.

There are four packages. Each package correlates to a domain in the system. Because of the use of the MVC pattern, there is the model, the view, and the controller. It is decided to have one package for each domain, to decrease coupling. Furthermore, there is the package util. The util package behaves like a library, meaning it consist of classes that supply all other packages and classes with general methods and functionality.

In section 3.6 it is mentioned, that there are different flavors of the MVC-pattern. The important information that the pattern carries is: dividing the system into model, view, and controller providing a low degree of coupling. This system's MVC architecture utilizes the observer and the facade pattern. The observer pattern is used to abstract communication between model and view to a higher level. The facade pattern is used by the model to hide the Backend behind the interface of the model.

Model

The model is the heavyweight backbone of the system. It is made up of an abstract class *Model*, a *ConcreteModel*, and the *Backend*. The concrete model is the actual model. It addresses other internal classes in backend that realize video player, processor, and data handling components that are specified in the high-level design. The abstract model provides interface methods for the view, abstracting program internals to interface methods. This abstract class acts based on the facade pattern. It hides internal logic and complex processes behind its interface methods. When a state of the model changes, it makes use of the observer pattern to notify it's observers – the views. Because of the further abstraction through the observer pattern, the model is only indirectly associated with the view. Thus, it only knows the view as an instance *Observer*, decreasing the coupling level. Moreover, the abstract observable class is the only class that provides *active* methods to communicate with the „outside world". Both, abstract and concrete model classes only provide methods allowing „others" to pull data.

View

The view is less complex than the model. Although, the view also is abstracted through an abstract class. The reason for that being, that implementation of the concrete view should not be concerned about integrating into the system. By extending the abstract view, the concrete view establishes a „connection" to the model in the way that the abstract view registers with the a model and the concrete view hence will be notified of updates from the model. The view is designed to „pull" data from the model on notification from the model. Therefore, the view has to know the model and has a direct association to it. This is a design decision of the observer-observable framework provided in the package util. There are different ways to design the observer pattern (see section 3.1). The decision is made for the „pull"-technique to keep the observer-observable framework in the util package more general for other applications, thus making the system provide better extendibility.

Controller

The controller is the connecting part between model and view. The design prescribes that the view should only work on interface methods from the abstract model. The abstract model provides getter-methods, only. Purpose of the model view controller

pattern is the strict isolation of view and model. Consequently, the view does not know of the concrete model. However, since the view provides interaction functionality with the user, it needs to be able to perform actions of the concrete model to let the user control the system. This is where the controller comes in place. There is a controller for each concrete view and concrete model. Events from the view are handled by the controller and forwarded to the concrete model.

4.3.2.2 View

The first step of modeling the system in detail is to realize the use cases and the virtual window that are specified in sections 4.1.2.1 and 4.1.2.2. The approach of modeling the view is realizing the virtual window, since it comprised all specified use cases. By extending the abstract class *View* a concrete view can be defined and inherits all interface methods for interaction with the model. Due to the adopted observer pattern, the view is notified when something in the model changes and can then *pull* new data. The only obligation of the concrete view is to present the data according the the specification in the virtual window.

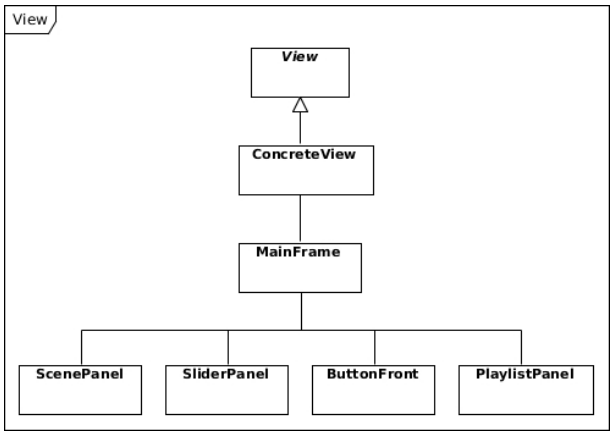


Figure 4.26: UML diagram of the concrete view.

Figure4.26 presents the design of the concrete view. According to the virtual window, the concrete view is a window frame that contains a scene, playlist, buttons, and a time line of the video. To achieve a higher degree of cohesion , each of these elements is realized in an extra class. The *ScenePanel* displays the scene that draws the video and the ROI in it. The *SliderPanel* shows the time line as specified in the virtual window. Furthermore, the *ButtonFront* deals with all buttons and the checkbox that are needed to interact with the system. The last class is the *PlaylistPanel*. As one can tell by the name of the class, this class displays the playlist and handles interactions with it.

4.3.2.3 Model

The model comprises the logic behind the view. Although, It does not only produce the data that is displayed by the view. There is also a file output, that saves the measurement results into a file. The model is made up of three components: *Video Player*, *Processor*, and *Data Handling*. The components *Java Media Framework*, *ImageJ*, and *FeatureJ* are used as libraries.

Video Player Component

The video player itself is realized using the JMF. A class instantiates appropriate classes of the JMF. The video player component covers almost all specified use cases, besides *Start measurement* and *Preview measurement results*.

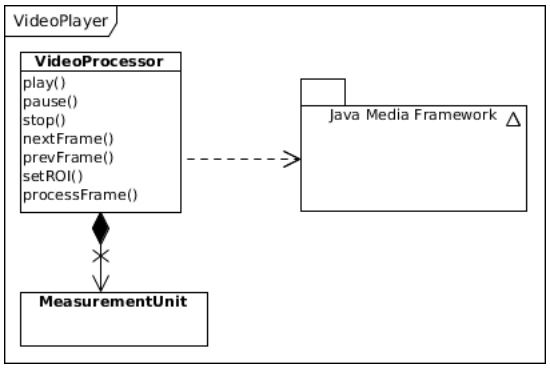


Figure 4.27: UML diagram of the video player component.

The video player provides typical video playback functionality. Moreover, the video player is used to supply the processor component with each frame of a video by calling the processFrame method. It is also specified that the user should be able to draw a rectangle in the video, giving immediate feedback. The feedback is the visualization of the rectangle in the video. That means, that the video visualization is actually being changed by the video player. The class is therefore, and due to naming conventions in the JMF API[4], named video processor, since it is already—besides from playing—processing the videos.

Processor Component

Every frame from a video is analyzed by the processor component. The objective of the processing is to assess the frames and obtain diameter measurement from the vessel present in the image. The assessment and diameter measurement is conducted by the class MeasurementUnit.

Figure 4.27 showed that the *MeasurementUnit* is supplied with every frame of the video. In figure 4.28 the associations of the MeasurementUnit are displayed. Section 4.2.1 and 4.2.2 explained how image processing and diameter measurement is conducted. Therefore, the class MeasurementUnit has the methods findEdges, getScale, and measureDiameter. The actual image processing—the detection of the vessel walls—is transferred into the class *VesselWallsDetectionAlgorithm*. This behaviour applies the strategy pattern (see section 3.2) to make the detection algorithm extendable. The vessel walls detection algorithm uses ImageJ and FeatureJ. Furthermore, the measureDiameter method makes use of the Mathematics and LinearRegression classes to check assumptions and obtain diameter measurement between the vessel walls.

Data Handling

The data handling component handles all data that is input to or output form the system. References to videos files need to be stored in a playlist and measurement

[4] Application Programming Interface (API); is a particular set of specifications that other software can use to communicate with the application behind the API.

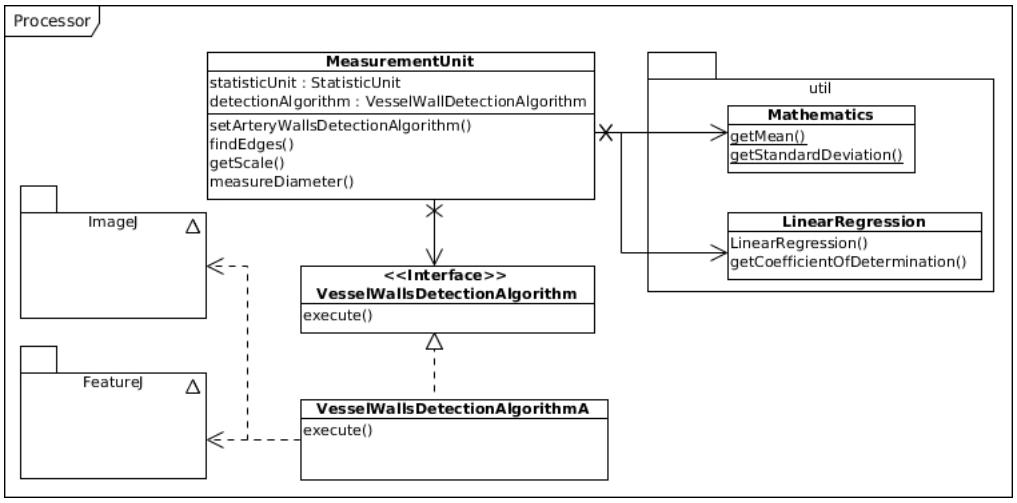


Figure 4.28: UML diagram of the processor component.

results are output to an extra file per video. The data handling component therefore handles two modules: The playlist and the measurement saving module.

Playlist Due to the specifications, the playlist needs to be able to load and delete video files. The measurement of each video can be configured by the user. Measurement marks can be set and deleted; The time window is defined by start and stop time; And the ROI that is drawn on the video scene. Figure 4.29 shows how this information is organized.

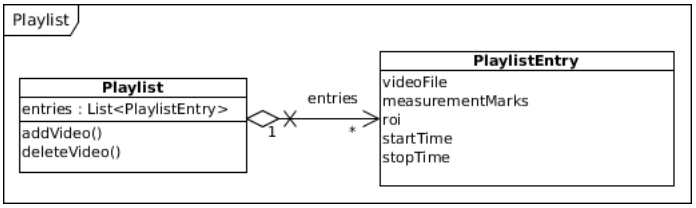


Figure 4.29: UML diagram of the playlist.

Figure 4.29 shows the classes *Playlist* and *PlaylistEntry*. The playlist stores many playlist entries, which holds the corresponding video file, measurement marks, the ROI, and start and stop time. The method addVideo creates a new entry for the supplied video. Consequently, deleteVideo deletes from the entries list.

Measurement Saving Module This module is made up of two classes: The *StatisticUnit* and the *FileSaver*. Figure 4.30 shows both classes and their uses.

The statistic unit class receives results from the measurement unit and stores them with reference to the video. When the concrete model calls the method saveStatisticInfo of the file saver class, it obtains measurement results from the statistic unit and saves them to an extra file per video in the demanded tabular form (see section 4.1.2).

4.3.2.4 Controller

In the systems architecture (section 4.3.2.1), it is already mentioned, that the controller is the connecting part between the model and the view. The design of the system pre-

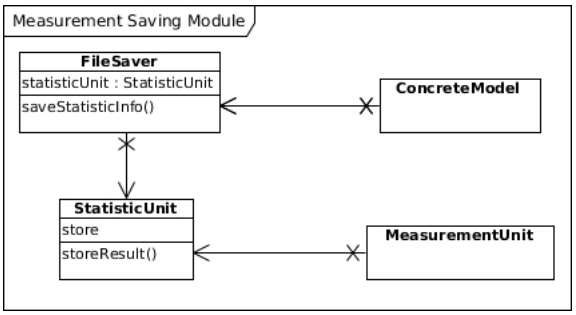


Figure 4.30: UML diagram of the measurement saving module.

scribes that the model should have no knowledge of the view, nor the view knowledge of any internal behavior of the concrete model. Figure 4.31 shows the associations between model and view, concrete model and concrete view, and the controller. The view only operates with getter and setter methods of the abstract model. However, since the system utilizes user input, certain events of the view need to be forwarded to the model.

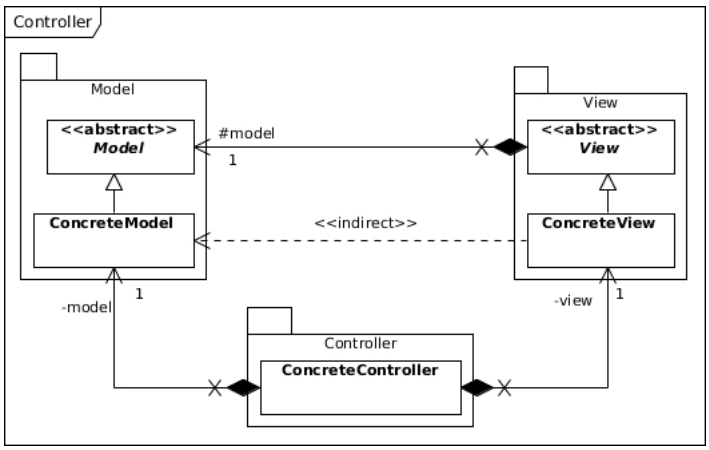


Figure 4.31: UML diagram of the connecting the concrete view indirectly with the concrete model, while the view interface has a direct connection to the model interface.

There are MVC architectures, where the concrete view calls methods of the controller, instead of calling them on the concrete model. This behavior can be seen as taking a circuit by the view. However, the architecture of this system uses one specially built concrete controller for the concrete view and concrete model. The system is put together by the controller. The controller has instances of concrete model and concrete view. Since the concrete controller is built especially for the concrete model, it creates action listeners that, when fired call certain methods of the concrete model, and adds them to the concrete view. By firing an event, the concrete view does actually not know what method is called. In figure 4.31, this behavior is presented as an indirect association.

Chapter 5

Implementation

The software is developed in Java and named „Vasculometer". In order to comply naming conventions of Java, the program resides in the package hierarchy de/dlr/me/vasculometer. The system design in 4 on page 16 prescribes the structure and modules of the software. This section provides details of important or problematic parts of the software implementation.

5.1 Model-View-Controller Architecture

The structure of the software is designed to use the MVC pattern. The aim of the MVC pattern is to isolate the model from the view. To achieve this goal effectively, the observer and facade patterns are made use of.

The first step to implement the MVC architecture was to define interfaces for model and view and by that, define how they interact. The only direct communication between model and view is designed to be when the view acquires displayable data from the model. All other communication happens indirectly.

The interface of the model is realized as an abstract class. It extends the class *Observable* and by that inherits methods to add, remove, and update observers. The other methods that are specified by the model are „getter" methods that can be called by the view to obtain data that can be displayed.

The interface of the view is also realized as a abstract class. It defines the update method, that is inherited from the observer interface. This method is called by the model when it updates all it's observers. The interface view also predetermines the *setModel* method, that is called by the controller to that assigned the view to a model.

For both, the concrete model and the concrete view, there is a controller connecting the two. The controller knows all interface methods of the concrete view and the concrete model and by that establishes communication between the two.

5.2 Controller

A special controller class that knows both, concrete model and concrete view is then used to connect the two. This is shown in the following lines of source code:

```
public class ConcreteController {
        private final ConcreteModel model;
        private final ConcreteView view;

        public ConcreteController(ConcreteModel model, ConcreteView view) {
                this.model = model;
                this.view = view;
                addListenersToView();
        }
        // ...
}
```

Furthermore, it can be seen, that the controller calls the *addListenersToView*. By invoking this method it calls methods on the concrete view to put *ActionListeners* to it. Since the controller knows the concrete model these ActionListeners—when fired—call methods of the concrete model to control the flow of activity. The view itself however, does not know what happens when it fires an ActionListener. Because some method calls of the concrete model do not spawn from ActionListeners, the controller makes use of the reflection programming technique that allows the processsing of methods as data structures. The following source code shows how the reflecting programming technique is used in this program:

try view.setAddFileMethod(model.getClass().getMethod("addFile", File.class), model); catch (Exception ex) Logger.getLogger(ConcreteController.class.getName()).log(Level.SEVERE, null, ex);

The Method object is obtained by calling the *getMethod* function of the class ConcreteModel. This class is determined by model.getClass(). The function getMethod then searches the class ConcreteModel for a Method with the name addFile that takes an object from type File as parameter.

5.3 Model

The relation between *ConcreteModel* and the interface *Model* is described in the previous section. Despite implementing „getter" interface methods from the Model, the ConcreteModel realizes the facade design pattern in providing all methods that are needed to address the backend conveniently by the view. Those methods for example are *play*, *pause*, and *stop*. By invoking these methods, the ConcreteModule performs all necessary tasks to fulfill the command and by that, hides the complexity of the system behind it.

The model's most important modules are the VideoProcessor, the MeasurementUnit and the StatisticUnit. In the following the implementation of each of them is described.

5.3.1 VideoProcessor

The *VideoProcessor* is a class that uses the JMF to perform video playback. JMF provides a functionality to play and process time based media. Without into too much detail, the JMF provides a hierarchical order of *Player* and *Processor*. With a player and render video files. For each media file, a new Player or Processor has to be instantiated. A Processor extends a Player with processing methods. So called *Codecs* or *Effects* can be applied to a codec chain that is supplied with each video frame while playing. Using a Codec or Effect for processing was the initial idea of implementing

the *VideoProcessor*. Each frame could be read pixel-by-pixel and changed by drawing markings for user feedback on the frame. Another possibility of accessing each frame is to obtain a *FrameGrabbingControl* object. Such objects provide the method *grabFrame* that returns the *Buffer* of the current frame. This *Buffer* can then be converted to *Image* with the use of the *BufferToImage* class. Hence, the actual processing of each frame—in terms of edge detection and diameter measurement—should be outsourced to a different class, a Player that utilizes a FrameGrabbingControl. This basically provides the same functionality but with less complexity than a Processor. Therefore, a Player is instantiated, and a FrameGrabbingControl is acquired. This provides the required features for processing the video media. It is also possible to obtain a *FramePositioningControl* instance from a player. With the use of this object, one can achieve effective processing of the video by neglecting the time base defined by the video frame rate.

To provide concurrent operation of the view—the GUI—and video playback or processing, a *videoProcessingThread* is implemented. This thread is used to iterate through the frames of the video using the frame grabbing control. The fabricateVideoProcessingThread method is a realization of the factory method design pattern. As described in section 3.4, this refers to simply replacing the source code with a method invocation to produce clearer source code. The important snippets of code in this matter are shown below:

```
private Thread fabricateVideoProcessingThread() {
        Thread thread = new Thread() {
                // ...
            public void run() {
                        while (!destroy && currentFrameIndex < totalFrames) {
                                // ...
                                updateCurrentFrame();
        // ...

void updateCurrentFrame() {
        // ...
                Buffer buffer = frameGrabbingControl.grabFrame();
        // ...
        img = bufferToImage.createImage(buffer);
        currentFrame = processFrame(img);
        parent.notifyObservers(de.dlr.me.vasculometer.util.Event.UPDATE_FRAME);
        // ...

private Image processFrame(Image img) {
        // ...
        if (measuring || showMeasurement) {
                img = measurementUnit.processImage(img, roi, measuring,
                    showMeasurement);
        }
        // ...
```

The source code that is shown above is only a very short segment of the actual code. Still it is possible to observe, that the thread „runs" through all frames and calls the *updateCurrentFrame* as long as it is flagged to be destroyed. The updateCurrentFrame method grabs the current frame, passes it to the *processFrame* method and then instructs the model to notify its observers that there is a new frame available. The processFrame method passws the image to the *MeasurementUnit* and then draws information, such as the ROI on the it.

The VideoProcessor manages all playback tasks by controlling the thread with various boolean flags, Therefore, the VideoProcessor basically provides own playback imple-

mentation of a video player with only Player object in the background to provide random access to the media. Whenever the VideoProcessor stops playing or processing, the videoProcessingThread ends.

In general, threads only run once and then „die". Therefore, when instructed to start playing the media, a new thread—if one is not already one running—has to be instantiated.

```
void startVideoPlayingThread() {
        synchronized (syncLock) {
                if (videoProcessingThread != null) {
                        return;
                }
                videoProcessingThread = fabricateVideoProcessingThread();
                videoProcessingThread.start();
        }
}
```

When working with threads, race conditions can appear. The VideoProcessor therefore uses the startVideoPlayingThread to create and start a new thread. The source code above shows that the *startVideoPlayingThread* method prevents such race condition by synchronizing the specific lines of code with a *syncLock* object, that is a instantiation of *Object* especially for this task.

5.3.2 MeasurementUnit

The *MeasurementUnit*is the core of the video processing task of the software. The method *processImage* is called by the *VideoProcessor*. This causes the MeasurementUnit to start four methods:

1. getScale
2. findEdges
3. cleanEdges
4. measureDiamter

The MeasurementUnit handles the images with the use of *ImageJ* and creates an *ImageProcessor* object for the image. The scale ratio is determined from the scale marks in the getScale method. The procedure of this task implements the algorithm 4.3.

After the scale factor is obtained, the ImageProcessor is used to crop the image according to the ROI. This happens in preparation for the findEdges method, that calles the *VesselWallDetectionAlgorithm*. As described in section 4.3.2.3 the MeasurementUnit utilizes the strategy design pattern to select a certain VesselWallDetectionAlgorithm. The VesselWallDetectionAlgorithm is an interface that defines the method *execute*. The signature of the method execute is shown below:

```
public interface VesselWallsDetectionAlgorithm {
        // ...
        public List<List<Point>> execute(ByteProcessor bp, int startY, int
            interspace);
}
```

The method takes three parameters. A *ByteProcessor*, the starting height within the image of ByteProcessor, and the interspace that should be used for the detection algorithm as defined in section 4.2.1.3. Returned is a *List* of *List* of *Points* for each edge on the vessel wall. To return a List of Lists looks unpleasant since there are always exactly

two Lists of points returned, however, it was deemed unnecessary to write an extra class to wrap two lists in an object.

The VesselWallAlgorithm that is described in 4.2.1 uses *FeatureJ* to perform the convolution of the image. The following code executes the necessary steps to perform this task:

```
// ...
ImagePlus ip = new ImagePlus("", bp);
imagescience.image.Image img = imagescience.image.Image.wrap(ip);
double sigma = 7;
int xorder = 0;
int yorder = 1;
int zorder = 0;
new Differentiator().run(img, sigma, xorder, yorder, zorder);
ip = img.imageplus();
bp = new ByteProcessor(ip.getProcessor().createImage());
// ...
```

FeatureJ provides its own *Image* class. The existing ByteProcessor needs to be converted to an ImagePlus in order to be wrapped into the special Image class. After that, the *Differentiator* class of FeatureJ can be instantiated and run with parameters that were documented in the methodology except for the kernel size. The Differentiator calculates the kernel size on its own, and it cannot be defined. Although, it would be possible to derive the kernel size, which is 29 in both directions in this case from the source code fo FeatureJ. After convolution of the Differentiator is completed, the wrapped Image is converted back into the ByteProcessor.

The edges are then cleaned by statistical analysis using the standard deviation. These are obtained with the use of the class *Mathematics* from the package util. After the edges have been cleaned, the diameter shall be measured. This is done by the measureDiameter method. In this method, a linear regression is computed and the standard deviation is computed once again. The linear regression is obtained with the use of the class *LinearRegression*, that also resides in the package util. The diameter is then measured as described in section 4.2.2.

5.4 View

The concrete view displays data of the concrete model. The concrete view is made up of sub-components that are used to build the GUI. Each sub-component has the responsibility of displaying a certain package of data. The concrete view organizes data update calls and updates relevant sub-components.

Whenever the state of the model changes , the view is notified and the update method gets called, delivering the Event object that determines the kind of change that happend in the model. The following source code gives an out-take of how the concrete view reacts on an update call:

```
public void update(Event e) {
        switch (e) {
                case UPDATE_PLAYLIST:
                        mainFrame.getPlaylistPanel().update(model.getPlaylist());
                        mainFrame.getPlaylistPanel().update(model.
                            getPlaylistSelection());
                        break;
                case UPDATE_FRAME:
                        mainFrame.getScenePanel().update(model.getCurrentFrame());
```

```
            mainFrame.getSliderPanel().update(
                        model.getCurrentFrameIndex());
            mainFrame.getInfoPanel().updateNames(
                        model.getStatisticalInfoNames());
            mainFrame.getInfoPanel().updateValues(
                        model.getStatisticalInfoValues());
            mainFrame.getInfoPanel().updateText();
            break;
        // ...
```

Chapter 6

Evaluation

Evaluation is needed to estimate the value of the software (SW). In this case, this refers to the output results in two manners:

- Validity
- Reliability

The validity specifies how the output from the SW corresponds to the real world. In most biomedical applications, this poses a difficulty, as it is impossible to know the ground truth, that is, the real world (DMG+07). The common strategy is to use manual reference samples to approximate the ground truth. These results can then be compared with the results generated by the SW.

The reliability of a measuring instrument can be rated by several characteristics. However, in this case only two characteristics apply:

- Inter individual deviation
- Intra individual deviation

The inter individual deviation defines the consistency of results when analyzed from different testers. The intra individual deviation on the other hand describes the variation of results when the same test is applied multiple times by the same tester. These two characteristics also apply to the real world. Manual assessment of ultrasound images results in different findings from tester to tester and also when performed multiple times by the same tester.

The next section will describe in depth, how evaluation was performed and presents the obtained results for this SW.

6.1 Evaluation Design

The procedure of evaluating the diameter measurement algorithm is to obtain diameter measurements by two different trained testers. Both testers performed two types of measurements:

Manual measurement was obtained with the use of the software ImageJ by manually drawing three lines from vessel wall to vessel wall for each image. The frames that were measured were previously selected by tester 1 and used for all manual

measurements—This is aimed at matching the manual measurements frame by frame to ensure correct comparability.

Software measurement refers to using the developed software for obtaining diameter measurement of the vessel.

For each type, the measurement series were performed in duplicate by the testers.

To evaluate the software, the following statistical criteria are used:

1. Arithmetic mean of the differences
2. Standard deviation

The evaluation is designed to give information about:

1. Inter-tester-manual deviation
2. Inter-tester-software deviation
3. Intra-tester-manual deviation
4. Intra-tester-software deviation
5. Inter-video variation
6. Independence of the algorithm

„Inter-video variation" investigates how diameter measurement results vary from video to video. „Independence of the algorithm" analyzes the independence of the software measurements from specific user input for the video.

The data pool that is provided for evaluation and development of software and algorithm was grouped before start of the project. One data set was then exclusively used for development and the other for the evaluation, only. The data set that is used for evaluation comprises 27 videos.

Generally, there are between 4 to 7 videos available for each subject.. The The data set for evaluation covers only a few of videos per subject, except in two cases where all material will be used. The structure of the data set is as shown in table 6.1.

Evaluation data set		
Test Subjects	Videos per Subject	Sum
9	2	18
2	complete	9
		27

Table 6.1: Evaluation data set comprises 2 videos of 9 subjects and the compete sequence of videos of 2 subjects.

The data set for evaluation is organized into two blocks—Block 1 (B1) and Block 2 (B2)—both listed in table 6.2.

B1	Evaluation data set
B2	B2 $\subset$ B1: 6 randomly selected videos of B1

Table 6.2: Definition of blocks that are used to organize the evaluation set.

Measurements results of B2 provided the set underlying for all statistical analysis, except for one special case: Tester 1 obtained manual diameter measurement of all videos

of B1 and noted the ROI he used. The same ROI was then used by another tester—tester 3—to obtain diameter measurements with the software. This was done to provide manual reference measurements for the „Independence of the algorithm" analysis. Concluding, table 6.3 summarizes all measurements that were performed.

	Manual measurement		Software Measurement	
	Tester 1	Tester 2	Tester 1	Tester 2
Series 1	B1			
Series 2	B2		B2	

	Fixed ROI Software Measurement
	Tester 3
Series 1	B1

Table 6.3: Evaluation plan: Tester 1 and 2 provided manual and software measurements from B2 in both series. Tester 1 obtained manual measurement results from B1 in series 1 and thereby provided measurements of B2 and manual reference measurements for „Independence of the algorithm" analysis.

6.2 Evaluation Results

The evaluation of the software provides information about the quality of the diameter measurement algorithm. Apart from analyzing statistical data, the evaluation provides data that can be analyzed to describe characteristics and qualities of the diameter measurement algorithm.

	Inter-tester deviation manual	Inter-tester deviation software
Arithmetic mean	0.335	0.329
Standard deviation	0.288	0.370

Table 6.4: Presentation of results of inter-tester deviation of manual and software measurements.

Table 6.4 presents results of inter-tester deviation of manually obtained diameter measurements and matches them with inter-tester deviation of measurements that were generated by the software. The values were obtained by calculating the differences of results from tester 1 and tester 2 for each frame and obtaining the mean and standard deviation from these differences. It can be seen, that the software generates results with a higher standard deviation than obtained by manual measurement, which indicates that there is a greater variability within the results of the SW. The mean of the deviations from tester to tester however, is about the same. between the manual and software measurements, showing that both techniques have generally a similar variation.

	Intra-tester1 deviation manual	Intra-tester2 deviation manual
Arithmetic mean	0,350	0,072
Standard deviation	0,406	0,059

Table 6.5: Presentation of results of intra-tester deviation of tester 1 and tester 2 of manual measurements.

Table 6.5 shows the deviations from intra-tester1 and intra-tester2 by performing manual measurement. It can be seen, that Tester 1 has a significatly higher standard deviation and mean of differences in his measurement than tester 2. This impressively demonstrates the different individual consistencies and illustrates the problem of reproducibility of results obtained in this way.

	Intra-Tester1 Deviation software	Intra-Tester2 Deviation software
Arithmetic mean	0,271	0,361
Standard Deviation	0,332	0,369

Table 6.6: Presentation of results of intra-tester deviation of tester 1 and tester 2 of software measurements.

In comparison to table 6.5, table 6.6 shows the intra-tester deviation while using the software. The standard deviation and mean are relatively highcompared to the standard deviation and mean of the intra-tester2 deviation manual from table 6.5. Still, it can also be seen, that the deviation is slightly lower when measuring with the software than manually.

The following table 6.7 presents the „Independence of the algorithm". As described in the evaluation design, manual measurements of tester 1 were matched with results from the software with the exact same ROI. The deviations that were measured, describe how well—or close—the diameter measurement algorithm performs when it is not configured separately for each specific video.

	Inter Tester fixed ROI Deviation to software
Arithmetic mean	0,775
Standard Deviation	0,826

Table 6.7: Presentation of the independence of the algorithm from specific user input.

The following graph in figure 6.1 provides a deeper understanding of how the results of both measurements differ. The graph shows measurement results from all video files that were examined. The results were compared frame by frame. When considering the results of tester 1 as the ground truth, it can be seen, that the software suffers mainly from outliners—or say—outlining videos. The algorithm often seems to produce similar results to tester 1, but in some passages the results differ strongly, resulting in the high deviation.

The proposition made before that there are strong differences between the videos is underlined by the chart shown in figure 6.2. The chart displays the intra individual software measurement variation of a specific tester of 6 videos of B2. The chart is an example of how the measurement behaves and how deviations occur. Hence the graph shows that software measurements are highly video specific—meaning, there are some videos that are very problematic for the algorithm.

Additionally the graph in figure 6.3 shows that strongly varying software measurements does not imply, that those measurements are particularly inaccurate. The graph presents concrete diameter measurement values for one tester and video. It can be seen that the diameter measurements of the software lie between the two manual measurements—within the intra-tester variation. As discussed in the introduction, the diameter measurement is dependent on the definition of the vessel wall. Hence this graph further illustrates a sysemic difference between the results, which is despite variation, in itself

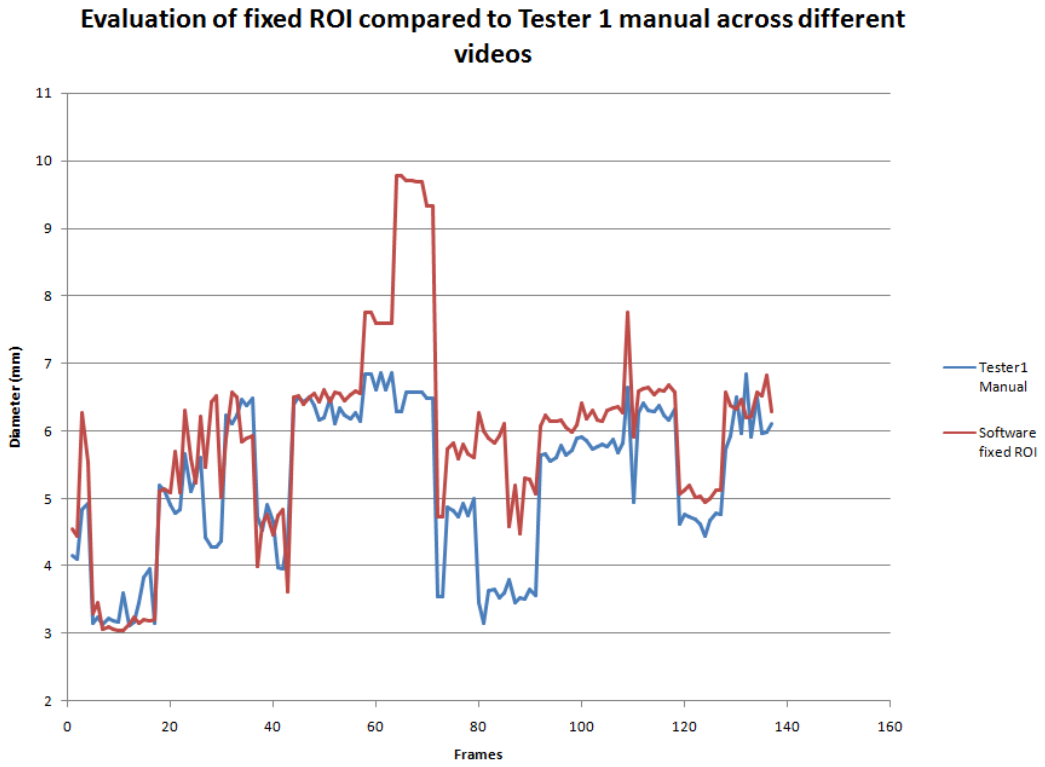


Figure 6.1: Demonstration of deviation of the software without specific configuration for a video. The graph shows all results of videos from left to right.

consisten. This systemic variation between the algorithm measurements is further likely to be caused by different selection of the ROI.

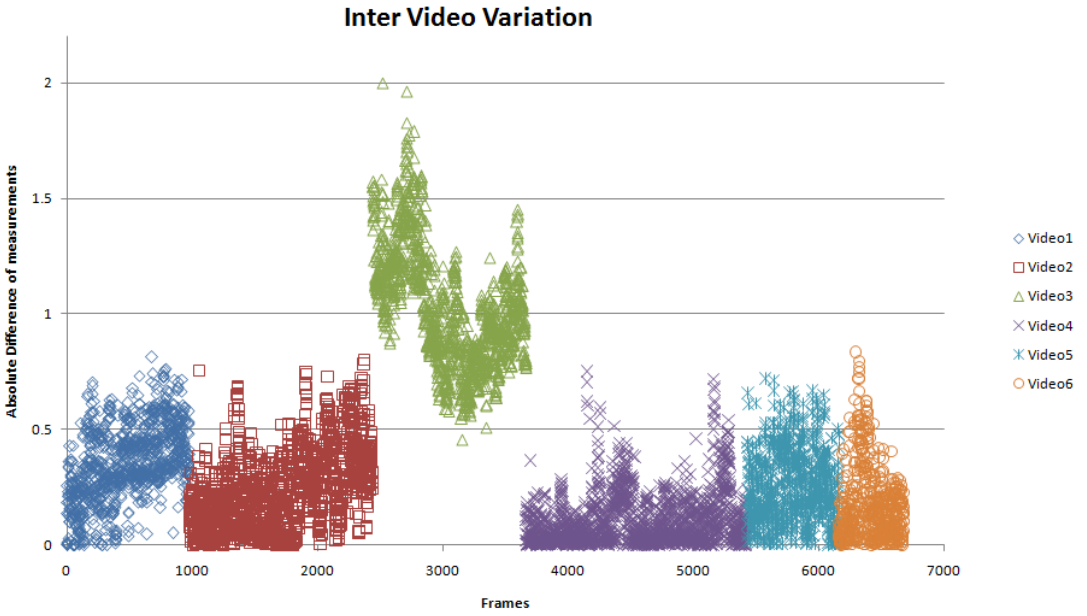


Figure 6.2: Intra individual software measurement variation of a specific tester of 6 videos of B2.

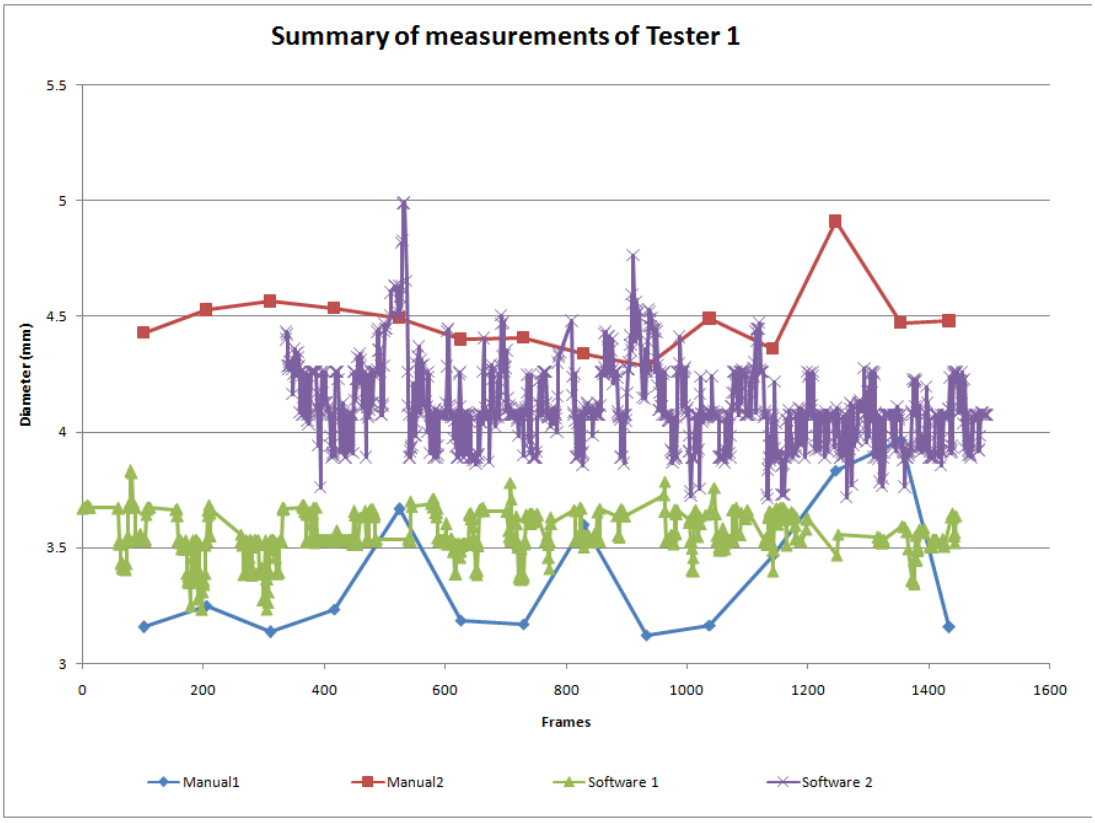


Figure 6.3: The graph compares all measurements (2 manual, 2 software) of tester 1.

Chapter 7

Conclusion

In this thesis a software for playing and analyzing ultrasound video files sequences is presented. The software developed uses the Java Media Framework to play and process video. The focus of the system design is to provide an extensible framework for image processing methods. A vessel wall detection algorithm and diameter measurement methods have been implemented. This algorithm is the first approach in providing such task within the framework. Work on other computer vision methods was approached, but exceeded the scope of this project.

The provided vessel wall detection method has drawbacks and compromises are made in order to produce a complete functionality of vessel diameter measurement. The measurement is dependent on user input, such as ROI, and unstabe. Noise in the lumen has a big effect on certain videos.

There are a number of enhancements that can be applied to the vessel wall detection algorithm. An idea that was currently worked on, is to apply longitudinal filter kernels instead of quadratic ones. This would take more horizontal pixels into account and would make the convolution less sensitive to noise in the lumen. Another approach might be to apply an active contours model, that takes even more parameters into account. These possibilities will be continued to be explored by the author even after completion of this project. Furthermore the open-end design of the software allows limitless future modifications. This faciliates future improvement of this algorithm or even implementation or different algorithms to suit new research questions.

Acronyms

DLR Zentrum für Luft und Raumfahrt

JMF Java Media Framework

SW software

UI User Interface

GUI graphical user interface

MVC model view controller

ROI Region of Interest

UML Unified Modeling Language

FMD Flow-mediated Dilatation

AVA Aerodynamics Laboratory

DFL German research institute for aviation

DFVLR German Research and Experimentation Institute for aviation and spaceflight

TVR Tonic Vibration Reflex

HSB Hue,Staturation, and Brightness

Bibliography

(Boe86) Barry William Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11, Issue 4:14–24, 1986.

(BT99) Cardinale M. Bosco, C. and O. Tsarpela. Influence of vibration on mechanical power and electromyogram activity in human arm flexor muscles. *Eur. J Appl Physiol Occup. Physiol 79*, pages 306–311, 1999.

(BV99) Colli R. Introini E. Cardinale M. Tsarpela O. Madella A. Tihanyi J. Bosco, C. and A. Viru. Adaptive responses of human skeletal muscle to vibration exposure. Clin Physiol 19:183–187, 1999.

(BV07) Delecluse C. Claessens A.L. Coudyzer W. Boonen S. Bogaerts, A. and S.M. Verschueren. Impact of whole-body vibration training versus fitness training on muscle strength and muscle mass in older men: a 1-year randomized controlled trial. *J Gerontol. A Biol Sci. Med Sci. 62*, pages 630–635, 2007.

(Car99) John M. Carroll. Five reasons for scenario-based design. 1999.

(cCSTC+99) Da chuan Cheng, A. Schmidt-Trucksass, Kuo-Sheng Cheng, M. Sandrock, Qin Pu, and H. Burkhardt. Automatic detection of the intimal and the adventitial layers of the common carotid artery wall in ultrasound b-mode images using snakes. In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, 1999.

(CHK65) Benjamin B. Tregoe Charles H. Kepner. *The Rational Manager: A Systematic Approach to Problem Solving and Decision-Making.* McGraw-Hill, 1965.

(CR08) Stannard S.R. Sargeant A.J. Cochrane, D.J. and J. Rittweger. The rate of muscle temperature increase during acute whole-body vibration exercise. *Eur. J Appl. Physiol 103*, pages 441–448, 2008.

(CS05) D.J. Cochrane and S.R. Stannard. Acute whole body vibration training increases vertical jump and flexibility performance in elite female field hockey players. *Br J Sports Med 39*, pages 860–865, 2005.

(DMG+07) S. Delsanto, F. Molinari, P. Giustetto, W. Liboni, S. Badalamenti, and J.S. Suri. Characterization of a completely user-independent algorithm for carotid

artery segmentation in 2-d ultrasound images. *Instrumentation and Measurement, IEEE Transactions on*, 56(4):1265 –1274, 2007.

(DV03) Roelants M. Delecluse, C. and S. Verschueren. Strength increase after whole-body vibration compared with resistance training. *Med Sci. Sports Exerc. 35*, pages 1033–1041, 2003.

(EG94) Ralph Johnson John Vlissides Erich Gamma, Richard Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

(ETF04) Bert Bates Kathy Sierra Eric T Freeman, Elisabeth Robson. *Head First Design Patterns*. O'Reilly Media, 2004.

(GAGHL97) T. Gustavsson, R. Abu-Gharbieh, G. Hamarneh, and Q. Liang. Implementation and comparison of four different boundary detection algorithms for quantitative ultrasonic measurements of the human carotid artery. In *Computers in Cardiology 1997*, pages 69 –72, September 1997.

(GBJ99) J. Rumbaugh G. Booch and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

(GPL+02a) M.A. Gutierrez, P.E. Pilon, S.G. Lage, L. Kopel, R.T. Carvalho, and S.S. Furuie. Automatic measurement of carotid diameter and wall thickness in ultrasound images. In *Computers in Cardiology, 2002*, pages 359 – 362, sept. 2002.

(GPL+02b) M.A. Gutierrez, P.E. Pilon, S.G. Lage, L. Kopel, R.T. Carvalho, and S.S. Furuie. Automatic measurement of carotid diameter and wall thickness in ultrasound images. In *Computers in Cardiology, 2002*, pages 359 – 362, 2002.

(IOfS06) Switzerland. International Organization for Standardization, Geneva. Ergonomics of human-system interaction – part 110: Dialogue principles, 2006.

(JYX+10) Jin Jing, Wang Yan, Gao Xin, Shen Yi, and Wang Qi. Automatic measurement of the artery intima-media thickness with image empirical mode decomposition. In *Imaging Systems and Techniques (IST), 2010 IEEE International Conference on*, pages 306 –310, 2010.

(Lau01) Soren Lauesen. *Software Requirements: Styles & Techniques*. Addison-Wesley, 2001.

(Lau03) Soren Lauesen. Task descriptions as functional requirements. *IEEE Software*, 2003.

(Lau04) Soren Lauesen. *User Interface Design: A Software Engineering Perspective*. Addison-Wesley, 2004.

(LDoCS05) C.S. Christodoulou Loizou, C.P. Pattichis and Limassol Dept. of Comput. Sci., Intercollege. Comparative evaluation of despeckle filtering in ultrasound imaging of the carotid artery. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 52(10):1653 – 1669, 19 December 2005.

(LJR+09) Jaesung Lee, Artit Jirapatnakul, Anthony Reeves, William Crowe, and Ingrid Sarelius. Vessel diameter measurement from intravital microscopy. *Annals of Biomedical Engineering*, 37:913–926, 2009. 10.1007/s10439-009-9666-5.

(LL07) Jochen Ludewig and Jochen Lichter. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. Dpunkt, 2007.

(RF00) Beller G. Rittweger, J. and D. Felsenberg. Acute physiological effects of exhaustive whole-body vibration exercise in man. *Clin. Physiol 20*, pages 134–142, 2000.

(Roy70) Winston Royce. Managing the development of large software systems. *Proceedings of IEEE WESCON*, 26:1–9, 1970.

(RS06) Verschueren S.M. Delecluse C. Levin O. Roelants, M. and V. Stijnen. Whole-body-vibration-induced increase in leg muscle activity during different squat exercises. *J Strength Cond. Res 20*, pages 124–129, 2006.

(RV04) Delecluse C. Roelants, M. and S.M. Verschueren. Whole-body-vibration training increases knee extension strength and speed of movement in older women. *J Am Geriatr. Soc 52*, pages 901–908, 2004.

(SLL02) M. Sonka, Weidong Liang, and R.M. Lauer. Automated analysis of brachial ultrasound image sequences: early detection of cardiovascular disease via surrogates of endothelial function. *Medical Imaging, IEEE Transactions on*, 21(10):1271 –1279, 2002.

(Tay09) B.B. Agarwal; M. Gupta; S.P. Tayal. *Software Engineering & Testing*. Jones & Bartlett Learning, 2009.

(VB04) Roelants M. Delecluse C. Swinnen S. Vanderschueren D. Verschueren, S.M. and S. Boonen. Effect of 6-month whole body vibration training on hip density, muscle strength, and postural control in postmenopausal women: a randomized controlled pilot study. *J Bone Miner Res 19*, pages 352–359, 2004.

(WB07) J. Burge W. Burger. *Digital Image Processing, An Algorithmic Introduction using Java*. Springer, 2007.

List of Figures

List of Tables

List of Algorithms

Appendix A

Fundamentals

A.1 Software Development Processes

Process models are generic concepts that describe the way of working on a project to achieve a certain goal. In this project, certain principles of process models were adopted to schedule the development of the software. This section roughly provides fundamentals of development process models to understand why and which process concept was chosen. All process models can be discriminated into two different kinds of models, that is, linear and non-linear models.

A.2 Linear Process Models

Linear, or sequential, process models were the first process models that had been proposed. Linear process models define a fixed step-by-step schedule in the software development plan. The typical and most common example for this is the waterfall model. The name comes from the concept of the model that the progress is seen as flowing steadily downwards through each step. There is no way backwards. Steps in a concrete waterfall model could be as follows, chronologically: System analysis, Software specification, Architecture layout, Design, Construction/Implementation, Verification, Integration, and Maintenance.

A disadvantage of this models is that mistakes in planning a project can be very expensive. For the project leader, it is a complex task to correctly estimate all processes. There are various factors which influence the difficulty of this crucial task. Furthermore, not only errors in scheduling are a risk factor. The customer and client does not always know, straightaway, what features of the software he requires. The client changing his requirements during the process of development, can be a more or less costly factor. However, when the project is small and easy to schedule, linear process models can be beneficial. Since everything is planned from the beginning, costs can also be planned, giving the customer a certain amount of financial safety.

A.3 Non-Linear Process Models

Over the time one has come to the conclusion, that strict linear planning comprises disadvantages that are hazardous to software development. The waterfall model came

up due to a historic misinterpretation of the ideas by Royce, that he published in an article in 1970 (LL07). In that he actually described well aimed iterations.

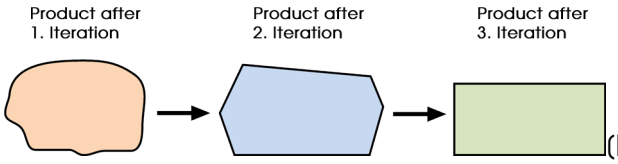
> „Attempt to do the job twice - the first result provides an early simulation of the final product." (Roy70, page 7)

The quotation points out, that the original idea of Royce was in a iterative, rather than a sequential fashion. In 1984 Boehm defined the spiral model (Boe86). From then on, iterative and incremental development paradigms gained influence. Today three different types of paradigms define the structure of software development processes. Those paradigms are: Evolutionary, Iterative, and Increment Development (LL07). The iterative development paradigm is explained in the next paragraph.

Iterative Development The software development process is scheduled in controlled iterations. An outline of the basic idea of this paradigm is given by the following quotation:

> „We get things wrong before we get them right."
> „We make things badly before we make them well." (Cockburn, 1993)(LL07)

Typically, in the first two iterations, a prototype is developed and refined. In the next iterations, faults that have come up in practice, are corrected. Each next iteration is usually planned out with feedback from the customer. The objective is approached in an iterative way. Ludewig and Lichter illustrated this concept with the graphic that is shown in figure A.1.



(LL07)

Figure A.1: Illustration of iterative approach to the product.

Appendix B

Screenshots

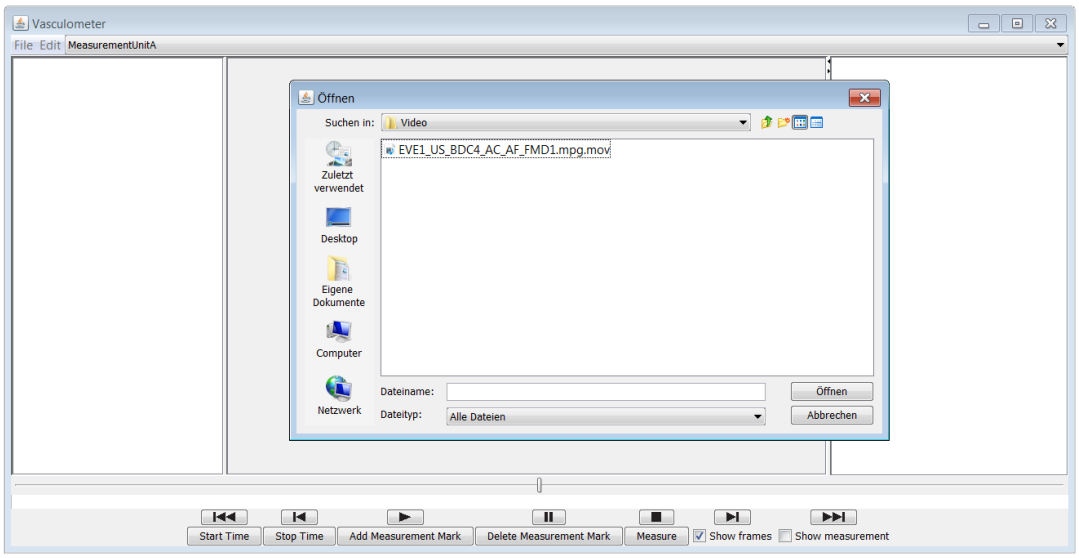


Figure B.1: Dialog to add a video file.

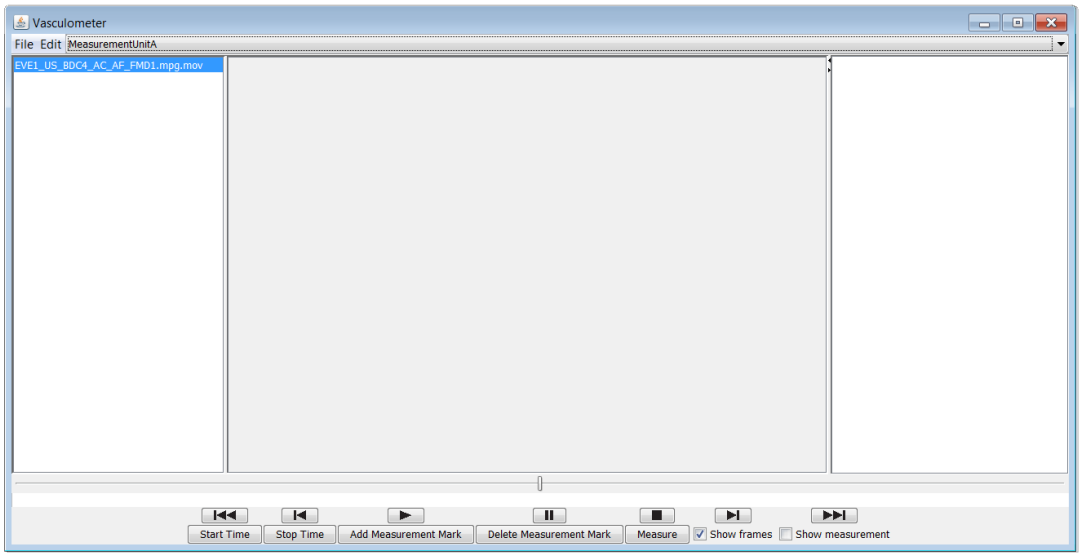


Figure B.2: Main window; The playlist holds a added file.

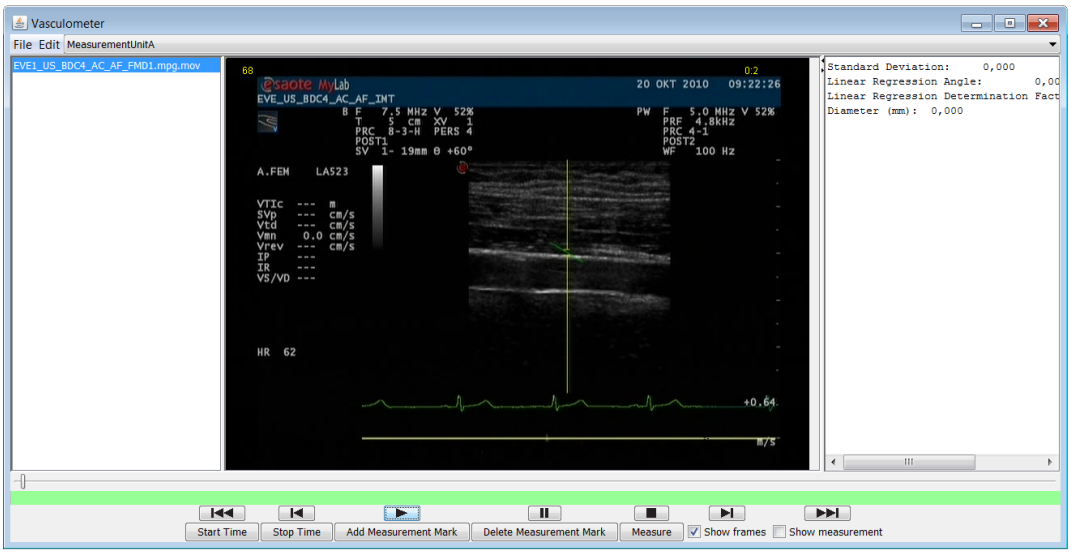


Figure B.3: A video is being played.

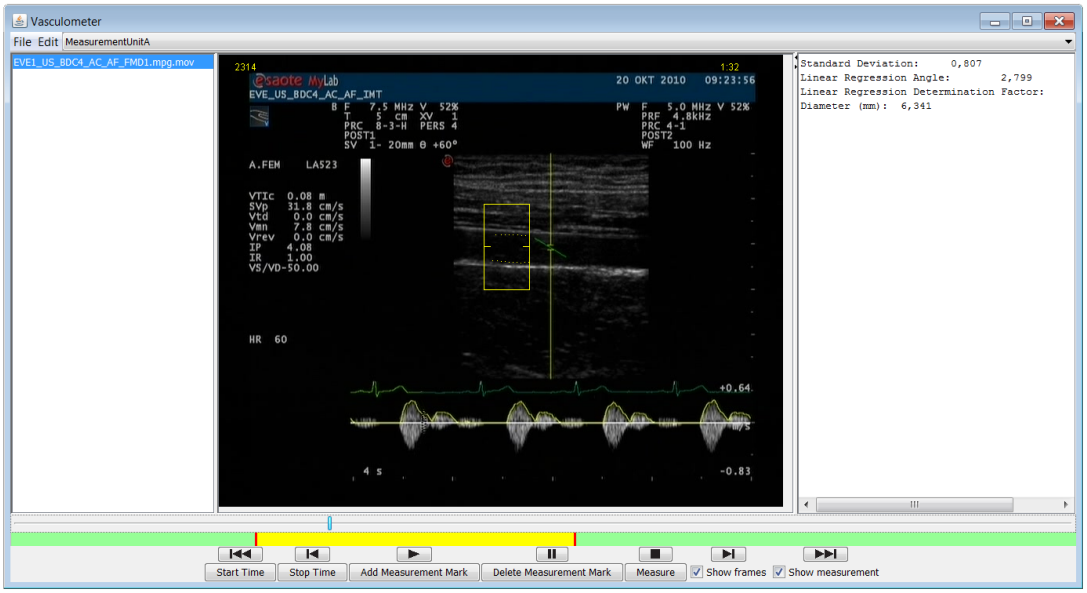


Figure B.4: Measurement marks and a region of interest have been defined and measurement results are displayed.