CHALMERS
UNIVERSITY OF TECHNOLOGY

Non-functional requirements testing in the Medtech industry

An automated test framework for vascular surgery simulations

Master's thesis in Biomedical Engineering

Marcus Martinsson
Per Nordeman

DEPARTMENT OF MATHEMATICAL SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Non-functional requirements testing in the Medtech industry

An automated test framework for vascular surgery simulations

MARCUS MARTINSSON
PER NORDEMAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Science
Applied Mathematics and Mathematical Statistics
Chalmers University of Technology
Gothenburg, Sweden 2021

Non-functional requirements testing in the Medtech industry
An automated test framework for vascular surgery simulations
MARCUS MARTINSSON
PER NORDEMAN

Supervisor: Richard Torkar, Department of Computer Science, Chalmers University of Technology
Supervisor: Carl Stein, Mentice AB
Examiner: Torbjörn Lundh, Department of Mathematical Science

Master's Thesis 2021
Department of Mathematical Science
Applied Mathematics and Mathematical Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Test output for test detecting tools outside of arteries in simulation image.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Non-functional requirements testing in the Medtech industry
An automated test framework for vascular surgery simulations
MARCUS MARTINSSON
PER NORDEMAN
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Although software testing is widely used during the software development phase of medical technologies (Medtech), there is no common ground regarding identification, separation, and testing of functional and non-functional requirements within the development cycle.

The objective of this thesis was to implement a taxonomy for non-functional requirements within the Medtech industry, and to build an automated test framework including tests for non-functional requirements linked to user experience.

The methodology used for the thesis was action research, iterated through a research, construction and a simulation phase for each test. Information gathered regarding test development was collected through interviews, documents, experience, and prior work.

A taxonomy for identifying, separating, and prioritizing software requirements was developed. Further, an automated test framework was developed which included automated tests evaluating the *reliability, performance, scalability*, and *portability* of the software system. Two *reliability* tests were developed to evaluate the stability and placement of medical tools within the simulation. An additional test was developed evaluating the *performance, scalability*, and *portability* of the software system. It was shown that automated tests can detect and notify software developers and project managers with information regarding non-functional requirements of their software system.

Although non-functional requirements often can be difficult to comprehend, the result within this thesis suggests that there is great value in identifying, classifying, and testing non-functional requirements within the Medtech software development cycle to secure a satisfied end-user.

Keywords: Requirements engineering, Non-functional requirements, Taxonomy, Software development, Medtech software, Automated testing, User experience

Acknowledgements

Firstly, we would like to thank the company Mentice, and more specifically our supervisor Carl Stein, for giving us this opportunity of evaluating their Medtech software throughout this thesis. Furthermore, we would like to thank our supervisor at Chalmers, Richard Torkar, for showing great commitment towards our thesis work and providing us with comprehensive feedback during all phases. Lastly, we would like to thank software developers at Mentice for helping us at all times and interventional radiologist Lars Lönn for providing us with insight into the connection between real-life endovascular surgical procedures and the Mentice simulation platform.

Marcus Martinsson, Per Nordeman, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

The profession of a surgeon is a highly skilled occupation which requires a high level of experience and expertise, generally taking more than two decades to reach the goals of doing advanced surgical procedures. The common approach has been with three guiding phrases: “See one, do one, teach one”. Meaning that a surgeon apprentice needs to see one or a few procedures being done by a more experienced surgeon, followed by performing the whole or parts of a procedure, and lastly teaching the procedure to new apprentices. This approach requires the surgical apprentice early in their career to choose one specific medical field to operate in. Since the procedures are often very difficult and can carry high risk, the training of certain procedures can take up to a whole lifetime to master. This could potentially create a shortage of valid surgeons due to apprentices not meeting the requirements to proceed with the training or there might be few to no opportunities to observe specific surgical procedures. A shortage of skilled surgeons also results in the surgeons being localized around big hospitals and clinics were they have the highest opportunity to proceed with their goals and training. The consequences could create a ripple effect resulting in for example rural hospitals and clinics being shut down, less availability to good healthcare, and longer waiting time for patients.

Moreover, the methods and tools used within the medical field are developed over many years with extensive testing and with strict regulatory requirements. Moving into the digital age, computers have gained more computational power and modern technologies have slowly been embraced and adopted by the medical field. This has lead to new possibilities in aiding the medical personnel in areas such as patient assessment, surgical training, logistics, research, and documentation by increasing efficiency and safety within these areas. Although this change towards new digital solutions offers improved healthcare it also puts the industry against new challenges in securing reliable and safe software in which software testing is an important step in securing these criterion’s. One ability made possible with the use of digital technology is digital reconstruction of the human anatomy which can enable computers to replicate anatomical structures to their finest detail. The ability to replicate the internal function and structure of the body has helped to visualize and create perspective on how organs interact with each other. The reconstruction of human anatomy has also made it possible to simulate surgical procedures. This creates the opportunity for surgeons to train and improve their skills within a certain procedure or with a certain tool more frequently and thus obtaining more experience in a shorter time-frame compared to the old way of practicing on real patients, closing the gap between the “see one, do one” stage. Moreover, creating simulated surgical procedures could help companies that are developing surgical tools with showcasing

why their tool is recommended for certain procedures, pushing the development of surgical technology and better healthcare.

The Mentice simulation provides a realistic simulation platform for which surgeons, or more accurately, interventional radiologists, can practice on. With their hardware and software setup, radiologists are able to feel and see how procedures are done. The possibilities are many but the development of such procedures puts high demand on how the simulation is portrayed by the user and how accurate it is in comparison to a real-life surgical situation. To meet these requirements a lot of testing is necessary when developing new procedures. These tests are often constructed with the help of professional medical personnel since the software and hardware developers often do not have the same medical expertise and experience as a radiologist. This creates a gap between how the tester experience the procedure and how a radiologist experience it, where a tester might not recognize faults in the same way as a radiologist would. Therefore the need for a testing framework is necessary in order to develop robust high-end simulations of surgical procedures where aspects such as user experience are prioritized and where the simulation closely mimics a realistic surgical scenario.

This thesis focuses on how the user experience within medical simulations could be measured and evaluated through non-functional requirements (NFRs). Also, how NFRs for medical software could be broken down and depicted through a taxonomy, in order to help Medtech companies in navigating the field of non-functional testing and securing that their product fulfills certain criteria with respect to NFRs and user experience. NFRs are system requirements which emphasis how the system "should be", i.e., how the system or software should behave or what properties that it should possess. This in relation to functional requirements (FRs), which are system/software requirements that put functional constraints on what the system/ software should be able "to do".

Furthermore, apart from the taxonomy, an automated test framework is developed with tests measuring simulation parameters that mirror specified NFRs and how these requirements correlate with the user experience. The reasoning behind measuring parameters to describe NFRs is that NFRs describe a systems behavior. Meaning that they describe the whole system or a particular aspect of the system and not a specific function.

The NFRs focused on is *reliability, portability, scalability, performance, and efficiency* since these are some requirements that reflect aspects of user experience on the Mentice simulation platform. The *reliability* reflects the tool's behavior within the simulation and the remainder requirements reflect the behavior of the system in general during different procedures. The reasoning behind these requirements is that they reflect different actions that need to work consistently throughout the procedure and if these functions or actions are unable to meet these requirements the user experience would be suffering. Creating tests that measure the grade from which the simulation meets these requirements enables the developer to see how their procedure is behaving within the simulation. The response from the tests grade the current state of the software and its output, informing the developer of what and where the user experience may be inadequate. The tests assist developers during development, closing the gap between how an experienced radiologist interprets the

simulation and how a developer interprets it. Improving the software development time and quality, enabling the company to deliver products faster and securing satisfied customers. The tests could also be used for further research within this type of field by contributing when building a ground truth of how simulated surgical procedures should behave. Also, the tests could help to recognize aspects of NFRs for systems outside the field, creating an overall user experience that is to some degree validated through automated user experience tests.

1.1 Problem Description

There are few research papers and studies in the area of testing NFRs and there is no commonly accepted approach for specification, documentation, and analysis of NFRs within software development, not the least in the medical device industry. This lack of research makes it difficult to navigate through the environment of testing NFRs within the Medtech industry and leaves the industry without a common ground for their testing procedures [1]. According to [1], "It is true that these requirements receive little attention relative to functional requirements during a software development", although their importance in ensuring patient and user security is vital for implementing a successful and safe medical product as well as for following medical device software standards such as IEC 62304 - *Medical device software—Software life cycle process*.

According to an industry study performed in 2015, about 62% of software development projects take NFRs into account when evaluating testing requirements [2]. The main reason for this lack of focus on NFRs is due to the elusiveness of these requirements since they are subjective as well as relative and can easily be scattered across multiple software modules when they are assessed in the software testing procedure [3].

This problem leaves a gap for providing a common ground by implementing new road-maps and methodologies for testing NFRs within the Medtech industry. This is an important step in developing better Medtech products and furthermore to reduce the number of deaths linked to user errors within the medical industry. Additionally, if left unnoticed, this could lead to many lives lost due to insufficient testing of NFRs of medical devices and software [4].

1.2 Purpose and Aim of Study

Figure 1.1 depict the goals of this thesis with their corresponding main questions and metrics. Each main question also includes sub-questions which are shown in 1.3.

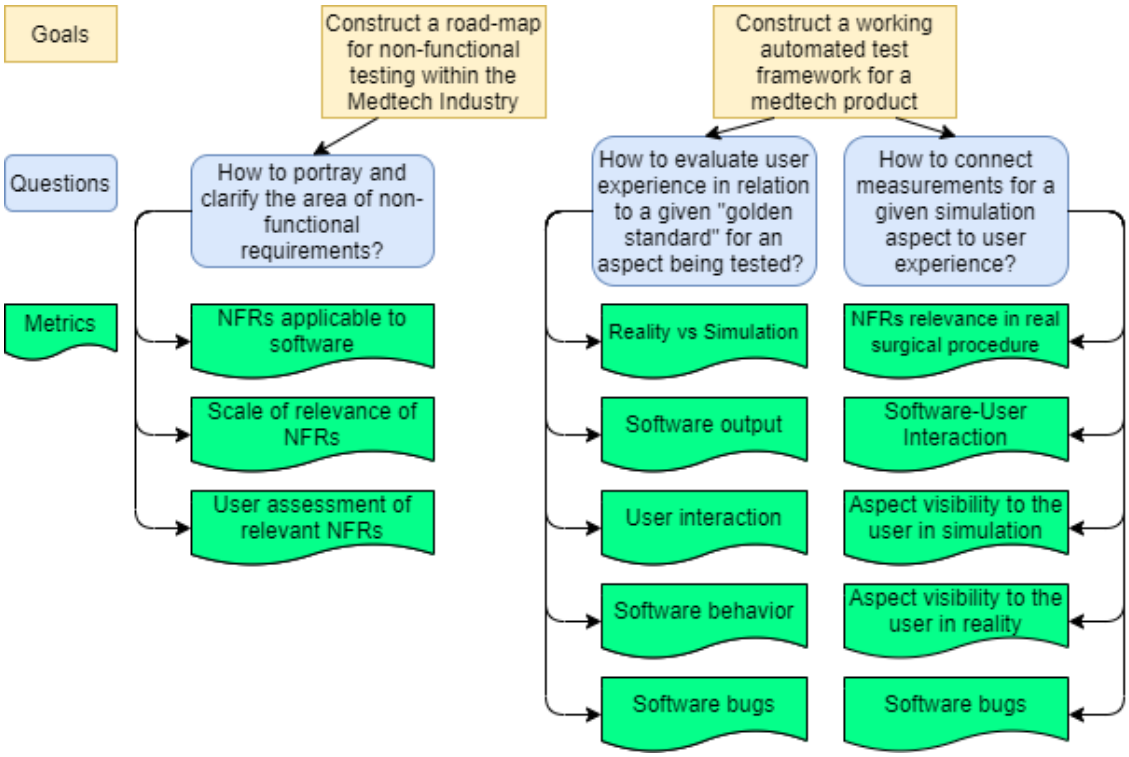


Figure 1.1: Depicts how Goals, Questions and Metrics are linked together. Note that there are two goals in which focus on the road-map/taxonomy and the automated test framework including its tests.

1.2.1 Purpose

The purpose of this study is to lay out a roadmap for non-functional testing in the Medtech industry in order to help future developers navigate the field of non-functional requirements testing. Also to construct an automated test framework that includes a few tests for non-functional requirements with the purpose of providing concrete examples of non-functional tests which match user experience with data collected for a given aspect of Medtech software. In this particular case, the Mentice simulation software.

1.2.2 Aim

The aim of this project is to construct a roadmap that depicts the various non-functional requirements available for software development in the Medtech industry. Additionally, to develop a working automated test framework for evaluating parameters connected to the user experience of a few aspects of the Mentice simulation software. Tests are developed to test some non-functional requirements of the simulations. The test should provide a result of these requirements and thus evaluate the user experience of the simulation.

1.3 Research questions

In order to reach the defined goals the following research questions (RQs) regarding the road-map and test framework needs to be answered:

1. How to portray and clarify the area of non-functional testing in the Medtech industry?
   (a) What non-functional requirements are applicable to the Mentice software?
   (b) How to depict a taxonomy for the specific non-functional requirements?
   (c) How to link the taxonomy to user experience?
2. How to construct an automated test framework for evaluating user experience in relation to a given "golden standard" for the aspect being tested?
   (a) How does the user interact with the software?
   (b) What is the "golden standard" for a specific test within the framework?
   (c) What NFRs have the highest impact on user experience?
3. How to connect measurements for a given simulation aspect being tested to user experience?
   (a) What is the connection between the automated test and manual user interaction?
   (b) How would the user interpret the aspect being tested if manual tests were performed?
   (c) How to link test aspects to real-life endovascular surgical procedures?

1.4 Metrics

In order to answer the RQs presented above multiple metrics needs to be collected regarding:

- Non-functional requirements testing
- Non-functional testing in other industries
- General software testing
- General image analysis
- Mentice simulation software
- Common software bugs within the Mentice simulation
- Construction of non-functional tests
- Real life endovascular surgical procedures
- A "golden standard" for aspects being tested in the testing framework with respect to real-life endovascular procedures.
- Ranking of test outputs with respect to user experience
- Collection of simulation data from Mentice simulation software.

1.5 Scope and Delimitation

Due to the vast complexity of the Mentice simulation system, the scope of this study has to be narrowed down and some simplification has to be made. Firstly, all tests

conducted within this project have been carried out on an emulator instead of the actual physical product delivered by Mentice. This choice results in a few differences compared to the physical product which will be further explained in Sect. 2.1.1. Secondly, due to limited ability at performing medical simulation procedures, procedures tested are just snippets of complete procedures and do not cover all possible test cases within a procedure. Furthermore, tests will only be conducted on a few surgical procedure simulations and will not cover all available procedures within the Mentice software. Research for our project and tests will be carried out in the form of action research.

Another limitation is that tests will only show where and how the user experience is lacking, but they will not cover why the user experience is faulty due to the vast complexity and scale of the Mentice code-base and simulation software. Also, due to the structure being large, only snippets of information of relevant functions in the simulation are gathered in order to deal with the test at hand. Thus aspects of how the whole underlying code may affect the procedure will be left out.

The lack of programming skills in programming languages such as JavaScript and QML is another limitation. Tests have mainly been carried out as external scripts in Python (with a few exceptions) which results in a few drawbacks since an external program is not able to grab all available data from the simulation, this means that some significant parameters affecting the test result might be unknown for our tests. Finally, the use of the record&replay function (described in Sect. 2.1.6) provided by Mentice could be a limitation since using a record&replay file means that the actions and collection of data do not necessarily occur in the same way as if it was run manually by the user. Making the retrieval of data highly reliant on the input-output of the record&replay system. Procedures that require precise actions often fail during a record&replay due to the random nature of the simulation and thus give inaccurate data. Also, for some procedures, only the essential steps are recorded which means that small variations in user interaction (input) might be left out during recording and thus not replayed accurately by the system.

1.5.1 Keywords and inclusion criteria

Below are keywords used for finding related research in the field of non-functional requirements testing as well as the inclusion criteria for this thesis.

Keywords used. “Non-Functional requirements”, “Non-Functional testing”, “Image analysis”, “Fluoroscopic Image quality”, “Catheter mechanics”, “Performance”, “Scalability”, “Portability”, “SIFT”, “SSIM”.

Inclusion criteria. To mainly include articles between 2010-2020 that evaluate or review various types of non-functional testing/requirements and/or mechanical /image properties that are related to this project(see Keywords). Articles should be found on scientific platforms such as Science Direct, ACM Digital Library, Pubmed, IEEE Xplore, and Wiley.

1.6 Outline of thesis

Figure 1.2 depicts the structure of this thesis and how thesis sections are linked together.

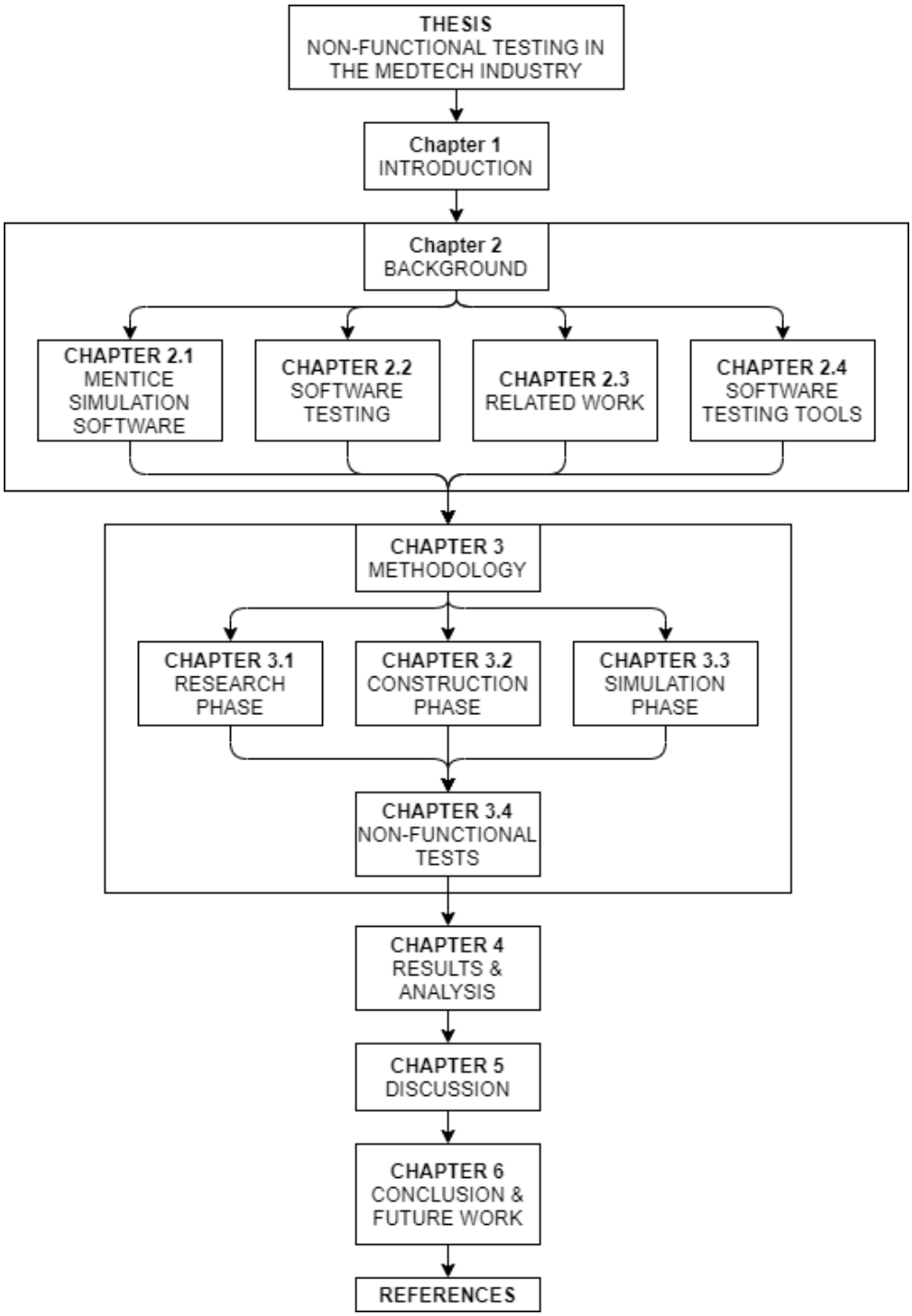


Figure 1.2: Outline of the thesis. Note that BACKGROUND and METHODOLOGY include multiple sub-chapters which cover necessary parts for the following sub-chapters or chapters.

2
Background

An important aspect when developing simulated environments is to test the software, to see if the environment mirrors the preferred realistic situation. Generally, testing NFRs of a system is done manually, either by the developers themselves or by personnel hired explicitly for testing the system. For surgical simulation the available testing personnel is few, resulting in developers testing different features themselves. The consequence could be that aspects are being missed due to lack of time, expertise and experience. A solution could be to implement automated tests, linked to real surgical expertise and experience, which verifies the function of different aspects, the look-and-feel of software, and find bugs that may occur during software development.

One example of how to classify bugs was proposed by Hooper [5], in which bugs were classified as A, B, C (sometimes even D), which specifies the severity of the bug. Class A bugs are of the highest severity and are usually specified as bugs that have an impact on the user's ability to complete a software run or use the software. Class B bugs are less severe but they are still a high priority and are usually connected to the user's experience of the software. The lowest priority bugs are Class C and are usually connected to the aesthetics of the software.

In this thesis, the aim is to build a test framework that mainly focuses on Class A and B bugs, since these are usually the ones having the largest impact on the user experience of the software.

2.1 Mentice simulation software

The Mentice simulation software is a practice tool for surgical apprentices which, to a great extent, can serve as a complement to real endovascular surgical procedure training. The Mentice simulation software is a program featuring user inputs for controlling medical tools, equipment, C-arm (arm for controlling the fluoroscopic camera) while also tracking patient status and realistically displaying the fluoroscopic image of the patient internals.

2.1.1 Emulator

Compared to the actual product delivered by Mentice described in Sect. 2.1, the emulator is based solely on software programs and does not include a physical control module for controlling the surgical table, C-arm and capturing fluoroscopic images. Additionally, there is no physical hardware that the user put surgical tools into but instead, computer software gives the user the ability to operate surgical tools,

Index	Description
1	Fluoroscope monitor
2	Virtual Control Panel
3	Manual Controlbox
4	Vascular interventional system trainer (VIST Device)
5	Radiation Foot padel
6	Indeflator
7	Syringe
8	Stent Release Handle

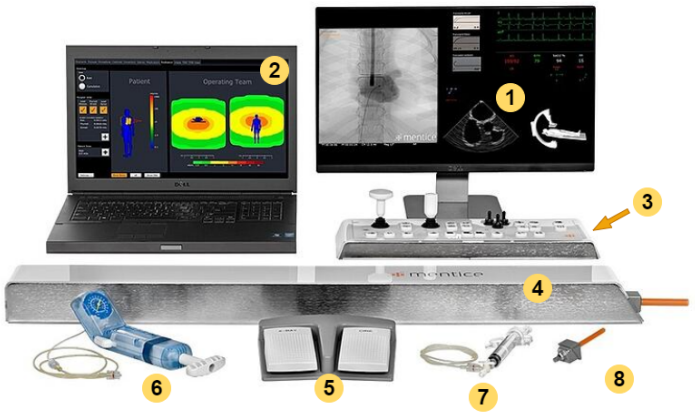


Figure 2.1: Visualization of typical Mentice hardware setup. In reality, there are multiple additional medical tools available which are not presented within this figure.

perform contrast fluid injections and capture cine loops (a re-playable sequence of fluoroscopic images) within the system. Although the emulator closely mimics the actual product there are some slight differences due to the emulator being based solely on software. One difference is that there is no force-feedback system for the tools within the emulator and thus there is no limitation in terms of how tools could be navigated throughout the patient when navigating into narrow vessels.

2.1.2 Program execution cycle

The program execution cycle, as depicted in Fig. 2.2, consists of multiple steps that are performed in each simulation frame of the Mentice software. Ideally, the simulation runs at an approximate 16 frames per second and the physics for the tools and collisions between tools and blood vessels is calculated an approximate 1000 times per second.

During this execution cycle, multiple steps are performed to update the underlying program state which ultimately leads to a visual representation of the simulation. The state also provides force feedback of the surgical tool to the user's hand, representing the inertia experienced when navigating a medical tool into a blood vessel with equal/smaller diameter compared to the tool.

2.1.3 Featherstone algorithm

Calculating the movement of joints and links when an external force is applied to these objects is a computationally difficult thing to do and thus there have been simpler algorithms developed to compute these movements more efficiently. One way to perform these computations is to use the Featherstone algorithm. The Featherstone algorithm, in comparison to Newton's second law $F = ma$, uses the velocity instead of the acceleration of an object. The Featherstone algorithm, given the current joint q with velocity q', a joint torque of G and an external force of F, calculates the joint

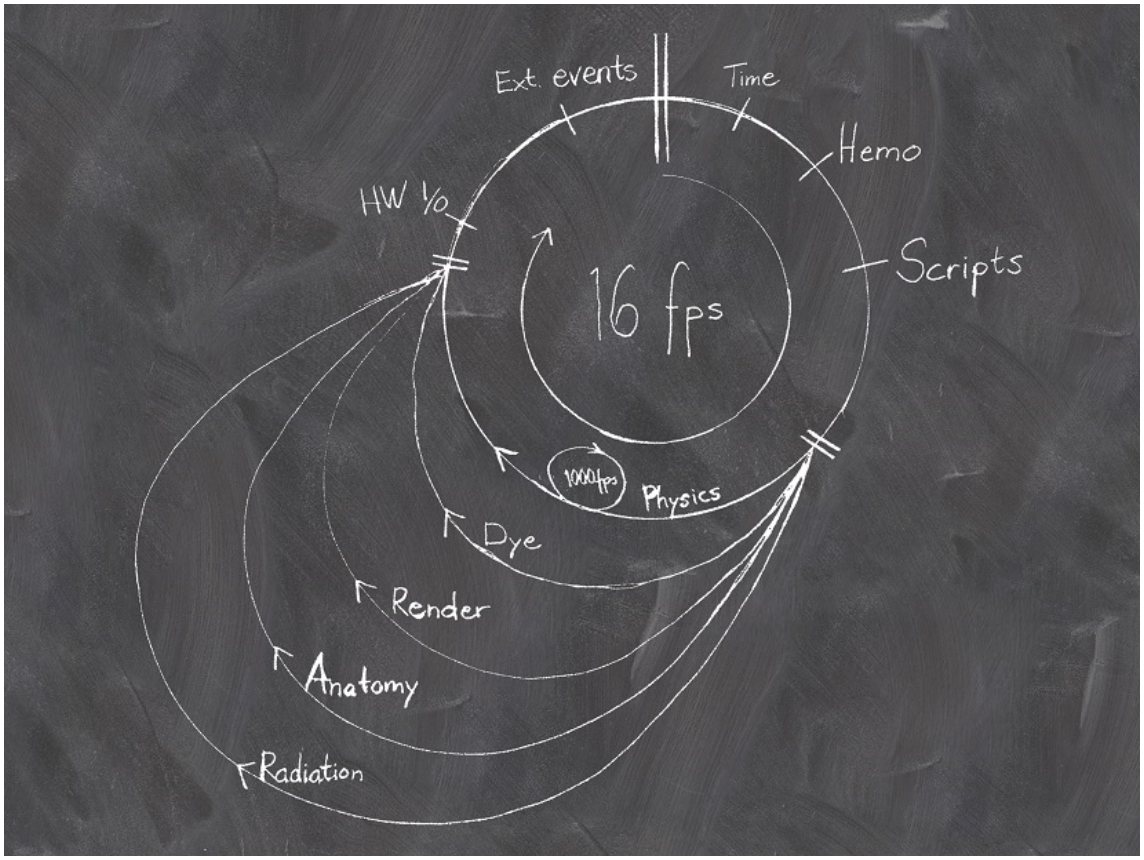


Figure 2.2: Simulation cycle. Note that multiple simulation functionalities are executed in parallel during each simulation frame. Source: Mentice Documentation.

acceleration(q") in linear time:

$$q'' = f(q, q', F, G)$$

By using this algorithm it is possible to convert the force given from the user's hand into a movement within the simulation that is approximate to what would happen in reality. This is, given the explanation above, a simplification of how the mechanics of the system work and how force, given by the user to the surgical tool, is converted into a movement in the simulation. This movement is also scaled upwards within the simulation, i.e, a 5cm move of the physical tool, in reality, will give a larger movement of the tool within the system. This simplifies the architecture of the hardware capturing the movement of the tool within the Mentice simulation platform.

2.1.4 Randomness in simulation

There are factors within the simulation that has an element of randomness and thus makes testing the software more difficult. First of all, for the fluoroscopic image visible to the user there is Gaussian noise added to the image which makes no image from a current simulation identical to a previous run even though everything else within the simulation is set the same. This adds to the difficulty of performing image analysis since this noise will have an impact on the look of a given blood vessel or tool between simulation iterations.

Other than the applied Gaussian noise there is randomness in other aspects such as contrast fluid injection. When performing a contrast fluid injection within the simulation there will be some randomness in how the fluid will spread out across the blood vessels and thus a contrast fluid injection will not look identical to a previous injection even if all other settings are equal. This also adds to the difficulty of performing image analysis across simulation runs since there will never be identical outcomes for identical inputs across simulation runs.

2.1.5 Debug mode

The Mentice simulation software delivers the possibility to add debug options when running a simulation. There are plenty of debug options connected to the tool, blood vessels, physics and fluoroscopic image available to the developer which adds great value when developing tests for the software. As an example, when developing tests that focus on blood vessel—tool interaction there is the possibility to remove all other factors such as skeleton, heart, skin and gaussian noise animation and thus isolate the tool and blood vessels in the fluoroscopic image and assess how these interact with each other. It is important to note that for each aspect that is turned off using the debug mode the simulation diverges further from a realistic scenario. This has to be taken into consideration when designing test cases in order not to remove aspects that will alter the quality of the test itself.

2.1.6 Record&replay function

The Mentice simulation software delivers the possibility of recording each procedure as a list of inputs which was taken by the user as they performed a procedure. This list of inputs, the record file, could be used at a later stage for replaying the same medical procedure. By providing this feature a user can replay the steps of a previous procedure and experience almost identical software behaviour. The word almost is stated here since the software is highly complex and non-deterministic with heavy computations being performed multiple times each second which may alter the result of the procedure slightly due to factors such as CPU speed and current workload on the computer CPU and GPU.

2.2 Software Testing

Software testing is an important step in software development that ensures quality assessment of the software and provides developers with information regarding to which degree certain functionality within the system works. Software testing consists of multiple branches that range from single module tests (component testing) to beta testing which is the final tests conducted before releasing the product to market. In the computer gaming industry, alpha and beta versions of the software/game are usually given out for public testing for the company to get customer feedback in the late stage of the development cycle [6].

2.2.1 Automated testing

Automated testing primarily has one objective, to make manual tests automatic and to automatically assess whether the output/behaviour of the software is according to expected output/behaviour or within guidelines for the software. Since manual software testing is both time consuming and victim to variance in user input an automated software test can give the same input for each test iteration and only assess whether the output is correct or not. Thus making the software testing more robust in comparing input/output but with the drawback of being unable to detect new test scenarios or software bugs which a human being could have detected through creativity and curiosity when performing manual tests on software. Even though there are examples of introducing randomness in automated testing [7][8], it is still a challenge to introduce the nature of human creativity and curiosity into automated testing and thus difficult to test all various interactions with the software which a human would potentially perform.

2.2.2 Testing for user experience

The main objective of software testing could vary depending on the purpose of the test. Some tests focus only on verifying that a certain functionality within the software works as supposed to, while other tests focus on testing how well a certain functionality performs in the software and thus gives a clue regarding how the user will experience the software (given that the main functionality works in the

Table 2.1: List of NFRs with a short description. Requirements highlighted in blue are requirements focused on in this thesis.

Requirement	Explanation
Performance	Speed at which computer executes a program [12]
Scalability	System ability to accumulate loads. [13]
Portability	The degree software can be used in different environments. [14]
Efficiency	Handling of resources during load/activation [15]
Capacity	System volume accommodation when scale-up [16]
Availability	System accessibility at a given point in time [17]
Reliability	Ability to function without failure [18]
Recoverability	Ability to recover from failed states
Maintainability	The ease with which an attribute can be maintained [10]
Supportability	The ease to regulate updates & errors within the system [10]
Security	Protection of system disclosure [10]
Manageability	The ease with which system could be managed [19]
Data Integrity	Ability to maintain data accuracy and consistency [10]
Usability	Capacity of a system to provide good user interaction [10]
Interoperability	Degree of clarity in interface interaction [10]
.....	.....

first place). These two types of tests can usually be divided into *functional*, and *non-functional* testing in which the later type of testing focuses on how a verified functionality within the software will "feel" when used by the user.

2.2.3 Functional and non-functional requirements

Functional requirements (FRs) could best be described as "what a software should *do*". An example of a FR for software would be: "If A is pressed, B should happen". These requirements focus on the functionality of the software and that certain functions works as supposed to. Functional testing is the act of testing these FRs by asserting that various functionalities within the software works as supposed to.

On the other hand, NFRs could be described as "how a software should *be*". An example of such a requirement would be: "If A is pressed, How long does it take until B happens". This requirement is focused on the performance aspect of the functionality rather than the actual functionality itself, as in the scenario for the FR. FRs are usually the most important and mandatory requirements when testing software. Since if the functionality is not working at all there is a clear malfunction of the software which will inhibit the user from proceeding. Even though FRs are important, NFRs determining whether or not a software is reliable, secure, scalable, user friendly and so on, which are all very important aspects of the software and how it will be viewed by its users. Figure 2.3 summarizes the difference between functional and non-functional requirements [9]. Table 2.1 depicts a few examples of NFRs obtained from [10].

Functional vs Non Functional Requirements

Parameters	Functional Requirement	Non-Functional Requirement
What is it?	Verb	Attributes
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case.	It is captured as a quality attribute.
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing, etc.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.
Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties

Figure 2.3: Depicts the difference between functional and non-functional requirements. Note that FRs usually are connected to verbs while NFRs deal with system attributes. Source: [11].

2.2.4 Specific non-functional requirements

To narrow the many different non-functional requirements of the Mentice simulation, the thesis focuses to develop tests for the following types of non-functional requirements: *reliability, portability, scalability, performance and efficiency.*

According to [18], *"Reliability engineering is a sub-discipline of systems engineering that emphasizes the ability of equipment to function without failure"*. Where the function/equipment should work under some condition during a specific time. If the *reliability* requirement is applied to the Mentice simulation, one example could be to view how the medical tools within the simulation and how the motion is perceived during the time of the whole procedure. The motion is sometimes subtle and for an inexperienced person, this movement could seem normal. Thus developing a test to quantify a non-functional requirement like *reliability* is necessary since the simulation is highly dependent on user experience.

The intent for *portability* is to evaluate the system usability through different software/hardware environments [14]. One example of this is to see if all surgical modules of the Mentice simulation perform similar between hardware changes. One approach to this is to see the software reaction of a button press between different hardware. If the underlying software of the button reacts too slow for specific hardware, one could say that the *portability* of the software is weak for this particular hardware.

Scalability is a requirement whose intent is to view how the growth of a system affects itself and how it performs when extra work is added to its architecture [13]. By adding load on the hardware it is possible to view how the simulation performs by measuring the number of calculations over each second. If a specific Mentice procedure is in need of very precise calculations in order to match the user experience aimed for, it would be difficult to achieve if the underlying software took a lot of processing power to execute surrounding operations. Therefor a test to assess the *scalability* of the hardware is needed for each simulation procedure.

Moreover, the *Performance* NFR is widely used when evaluating computer systems. The requirement could be divided into different kinds of measurements, though the general definition is to measure the amount of useful work the system achieves under a certain time period. The performance is measured in terms of efficiency, accuracy and speed of executions [12]. The Mentice simulation is built around heavy computations and therefore needs high performing systems to keep a high level of user experience. For each execution cycle, thousands of calculations need to be computed, which requires a good and reliable system setup. Developing a test for evaluating the performance limit of the system could help in the aspects of setting parameters and thus tune them to receive the best user experience for given hardware.

Efficiency requirement is closely related to the *scalability* requirement. Since *efficiency* is defined as the ability to reach a result with as little waste as possible [15]. In context of the Mentice simulation, one could measure how efficient a simulation procedure is at calculating the physics. If the underlying code is efficient the degree of *scalability* is increased since the code takes a short period of time to execute and leaves room for additional functionality. Knowing the *efficiency* of the code could allow the software developers to tune parameters to increase the user experience.

2.3 Related work

This section aims to describe some related work in the area of requirements engineering (RE) for NFRs. It covers the importance of NFRs, how they could be classified, a few examples of integration and whether or not NFRs really are "non-functional". Sommerville et al. [20] describes RE as *"the process of discovering, documenting and managing the requirements for a computer-based system"*. The goal of RE is to provide a complete and consistent collection of requirements that are relevant for a given system. The idea is that these requirements should reflect what the end-user actually wants and secure that these needs are fulfilled.

2.3.1 Importance of NFRs

Paschali et al. [21] performed a survey on what factors had the highest influence on user satisfaction when playing computer games. The researchers concluded that almost all games share a common key requirement: user satisfaction. They also point out that user satisfaction is not uniform across different computer game genres but dependent on them. Another conclusion was that what influenced the user satisfaction the most was different over time [21]. From a previous study by Ham et al. [22], the authors noted that the greatest driver for user satisfaction was the graphics, gaming controls and the game character. On the other hand, another study from the same year performed by Balkin et al. [23], emphasizes the importance of interactivity between player and game/community and state it as the greatest driver for user satisfaction. The argument is that, for the player, interaction with a community is equally important to real-world interaction. It also enables the player to develop an identity when playing computer games online. In a thesis written by Young, the author tried to determine whether or not the graphics played an important role in user satisfaction. This study shows that the majority (54%) of the participants viewed graphics as not that important when deciding what game to play [24]. This change, compared to Ham et. al., suggests that graphics play a smaller role in user satisfaction than before. Although, Paschali et al. state that it is important not to ignore that the rest (45%) still viewed graphics as an important factor which leads to the conclusion that graphics still play a major role in satisfying customers for computer games.

From a paper written by Maiti et al. [25], the authors state that during software development using an agile methodology, NFRs are not taken into account during the requirement phase. This results in projects needing to revisit their code due to failures and bugs. The paper also states that if NFRs were taken into consideration early on in development, it would increase the stability and reliability of the software. The paper analyzed if the $\alpha\beta\gamma$-framework could work for prioritizing NFRs as good as it prioritised FRs within the "Capture, Elicit and prioritizing" (CEP) methodology. The data consisted of already rated NFRs from other methodologies from which the weighted sum could be derived. The gathered parameters were then applied to the framework. α was collected through the 100-dollar test [26], β was the weighted sum from previous data, and γ was the weights of each requirement divided by the average weight of all requirements. This resulted in different NFRs

gaining higher values, making it easier for Agile processes to consider NFRs with a different priority level [25].

In a later paper, Memon et al. [27] illustrated how two common NFRs in software development correlate with the quality of the software. They also state that there is no unanimous consensus regarding how NFRs are gathered, analyzed and documented in an organized manner. They identified 270 empirical studies in which NFRs is the least explored area, even though NFRs are of great importance and often regarded as the "body and soul" of a software system. Historically this consideration has not been made and most emphasis has been placed on FRs which undermines the importance of NFRs.

The Memon et al. paper focuses on how the NFRs *reliability* and *scalability* affect software quality. The paper explains how these two NFRs are interpreted and used through different stages in the requirement assessment. The paper points out that standalone NFRs cannot be assessed alone and that they often affect each other. Futhermore, it also states that a developer's knowledge of human cognition helps to write quality software since it is these quality attributes that mirror human interaction with the software. They claim that if NFRs are considered more at the beginning of the requirement process, the smoother the following processes will run. Furthermore, the paper points out that today, researchers are trying to find the true meaning and functionality of NFRs, and that this research might lead to a better understanding of what NFRs really are and how these quality attributes can be analyzed and tested during software development.

2.3.2 Classification of NFRs

Mairiza et al. [28] stated that there is no consensus regarding the notion of NFRs and present an extensive and systematic review based on three dimensions of NFRs: *Definition and Terminology*, *Types*, and *Relevance* of NFRs based on system and application. Based on their investigation, the authors present two different perspectives of how NFRs are perceived by the software engineering community: *requirements that describe properties, characteristics or constraints that a system must exhibit* or *requirements that describe quality attributes that the software product must possess.* Also, the authors illustrate the vagueness of NFRs by depicting that more than half (53.51%) of their sample pool (114 NFRs) are without a definition and attribute. This adds to the difficulty in assessing NFRs during software development. Lastly, the authors also list the most frequently considered NFRs based on their investigation which are: *performance*, *reliability*, *usability*, *security*, and *maintainability.*

In later years, the use of text classification algorithms along with machine learning has been tested to create an automated classification test that could classify NFRs and FRs from text. One paper written by Lu et al. [29], applied this technique to analyze NFRs within mobile application reviews. The authors point out that the use of NFRs for mobile app development is crucial and provide good information for developers on what to change during the software development cycle. The motivation for this tool is that going through each review manually is very time consuming and important messages could be missed. They also state that extracting NFRs is especially hard since they are often hidden within the review.

Instead, Lu et al. proposed a way to extract NFRs and FRs from reviews using a combination of classification algorithms together with machine learning. The data used for the paper was from applications such as iBooks (Apple app store) and WhatsApp (Google play store). The data was labelled by the researchers as they tried to extract the information by supervised learning. The best result was given by the setup of "Augmented User Reviews-Bag of Words" (AUR—BoW) algorithm and a Bagging Machine Learning (BML). The setup was able to extract information from reviews that only contained single NFRs or FRs and unable to classify reviews that contained multiple NFRs and FR.

Nurbojatmiko et al. [30] aim to validate and develop a model to obtain application features related to NFR attributes and how these attributes are represented between FR and NFRs. They validate the models commonly used today. The current methods provide a sufficient ground from which the features can be obtained, though there still exists a gap in correctly identifying, classifying, and determining NFRs. The research was done using the "Prism framework" (a systematic review approach), where the idea is to gather documentation and papers from current methods and research. This is followed by sorting the papers by unique domain and topics. After gathering the unique data, analysis was done on the frequency of NFRs and the reasoning behind each NFR. The papers analyzed were divided into five groups: *NFR classification*, *NFR identification*, *NFR Representation*, *Environments-based NFR identification* and *NFR Agile Development*. Within the five groups, the frequency of each NFR was counted which indicated which type of NFR was covered more frequently and which NFR that was covered less. By the analysis, the result presented used NFR trends in software development and how software quality was affected due to the trend.

A later paper, by Majthoub et al. [31], propose a solution for bridging the gap between NFRs on the business side with NFRs on the system side. They claim that there is a strong correlation between quality/NFR actions made on the business side which affect the system side. They also state that currently (as of 2020) there is no classification method indicating the NFR correlation between the two sides and that there only exist models that classify NFRs on each side separately. The paper recognizes that the business side was more affected by NFRs than the system side since the business side had more manual work related to it. Moreover, if new NFRs were implemented on the business side it would lead to increased constraints and load of NFRs on the system side as well. The result of the paper suggests that one could trace and link the NFRs of the business side with the ones on the system side. This would result in developers and stakeholders being able to recognize the effects of implementing and removing NFRs from their development process.

2.3.3 Examples of integration of NFRs

Although this thesis focuses mainly on recent work within the field of NFRs in software development the importance of integrating NFRs into the software development cycle was already brought up over 20 years ago. One paper, written by Franch et al. [32], describe what characterizes NFRs (the 'how') from FRs (the 'what') in software development. The authors point out that traditionally, most

work has been on the FRs. The authors propose a way of putting non-functional information into the software architecture as a way of pinpointing NFRs important for the specific software. The paper suggests that non-functional attributes (such as *efficiency*, *usability*, and *reliability*) should be assigned a certain behaviour that stands as a NFR for a particular software module. The idea is to add value in terms of how NFRs should be attached to each software module.

In a following paper carried out a few years later, Barrett [33], similar to Franch et al., put emphasize on the lack of focus on NFRs within software development. The author describes NFRs as quantitative or qualitative in which the qualitative NFRs are more difficult to measure since they often are connected to the "look-and-feel" of the software and might be overlooked during development. The authors introduce a model for documenting NFRs called "Performance Cases". These "Cases" include a test with a name, goal, and a level of importance as well as what type of test it is. They also include a number of steps to be performed in order to test the NFR. The idea behind these "Performance Cases" is to clarify documentation around NFRs and display the role of a specific NFR during software development.

Dyrkorn et al. [34] provided an open-source toolkit for automated testing of NFRs. The idea was to provide an alternative to current approaches (e.g., testing suites) and to provide developers and project managers with reports from software systems that are under development. The authors suggested, based on experience, that this toolkit could be a way of reducing risk when developing software by not overlooking NFRs.

There has also been some work in trying to increase the traceability of NFRs within software development. One such paper, written by Yrjönen et al. [35], introduce a tooling solution for a "Domain-Specific Modelling" approach, an approach used during the design and development of computer software. The idea is to enable and guide the developers towards non-conflicting requirements by having bi-directional tracing of NFRs within the software development cycle, from models to implementation. The tool also provides an up-to-date view of the state of the NFR and enables the developer or project manager to keep track of the NFR during development.

In the last year, with climate change more frequently on the agenda, some recent articles focus on integrating sustainability as an NFR in their software testing activities. A paper carried out by Raturi et al. [36], discuss how sustainability can be developed as a non-functional requirement and outlines a road-map for how to integrate it in RE. Since sustainability in many cases can be of minor direct importance during software development the authors also discuss indirect consequences linked to consumer behaviour based on how the software is developed.

2.3.4 Are NFRs really non-functional?

There have also been some papers discussing whether or not non-functional requirements really are non-functional. Eckhardt et al. [37], performed an empirical investigation on 530 NFRs extracted from 11 industrial requirement specifications to discern to which extent they described the system behaviour. Their findings suggested that most NFRs were not NFRs and could be handled similarly to FRs.

The paper pointed out, not only that there exists no commonly accepted approach for NFR-specific elicitation, analysis, and documentation, but also that NFRs are usually described quite vaguely. The authors tried to pinpoint NFRs with respect to ISO standards for quality characteristics and found that most NFRs describe the same behaviour as FRs. Another note is that NFRs were specified at different levels of abstraction in which high-level NFRs needs to be refined into low-level FRs, in which there is no common approach

In another paper written by Broy et al. [38], proposes a model for rethinking FRs. The author states that the definition of NFRs often is unsatisfactory and the fuzzy term "nonfunctional" is often reduced to an attribute or constraint of equal fuzziness. Furthermore, the author states that rigorous classification of requirements often lack precision and concreteness, which leads to poor formulation of requirements. The author based his model on a system model which suggests structuring systems by using three levels of abstraction: an architecture structure, a state machine structure, and their interface abstractions. The requirements were categorized based on behavioural and non-behavioural aspects. Furthermore, behaviour aspects were divided into *logical* and *probabilistic* views, in which NFRs often can be described in terms of probabilistic models. These views were then also separated between internal and external views of the software system. Lastly, with the goal of developing a more general notion of functional behaviour the author proposed to redefine FRs based on syntactic interface and logical or probabilistic system interface behaviour.

2.4 Software testing tools and image analysis

There are many ways to perform software testing and it usually depends on the architecture of the software, how various data can be obtained from the software, and what the pros and cons are for testing with a specific programming language with respect to its alternatives. In this section, programming languages that are assumed relevant for the automated tests within this thesis are introduced.

2.4.1 Programming language

Since many software solutions are combinations of many programming languages there is a need to grasp the fundamentals of various programming languages in order to understand how the software works. Needless to say, when software and hardware are integrated into a product, additional programming languages are utilized for their various pros and cons depending on what part of the product is implemented. The Mentice software utilizes multiple programming languages including C++ for hardware and heavy computations, Javascript for simpler application logic and HTML, Qt, batch scripts for additional functionality and ease of work.

C++. The C++ programming language was developed at the beginning of 1980 at Bell Labs by Stroustrup. Today it is one of the most popular programming languages and is used in multiple fields such as computer games, consumer electronics and other software and hardware where there is a need for efficient and fast computations. C++ is based on the C programming language but delivers wider functionality

and supports object-oriented programming [39].

Javascript. JavaScript (JS) was when first implemented, presented as *Mocha*. Due to commercial reasons and the development of the Java programming language, this name was later changed into *Javascript*. Initially, JS was built in order to provide a language for building small applications within the web browser. This functionality was then further developed and today JS supports building more advanced web applications such as games and image analysis [40]. JS is today also used outside of the web browser in combination with *Node.js*. This enables the developer to build large scale JS applications, command-line tools and server-side scripts which are more efficient than the original idea of using JS in the web browser [41].

Python. Python is a high-level, general-purpose programming language. The language is in an interpreted one which means that code is not compiled before execution. Python delivers a wide range of possibilities from process management, hardware communication, image analysis and machine learning. Since one main disadvantage of an interpreted programming language is that it typically runs slower due to the fact that the program instructions are not compiled, many extensions built-in C or C++ are available when dealing with computationally heavy programs in Python [42].

2.4.2 Libraries

There are plenty of programming libraries already supplied by the open-source community that are available to simplify and reduce the development cycle for automated software tests. In this thesis, some specific libraries connected to image analysis have been suggested in order to take advantage of otherwise complex algorithms that are time-consuming to develop from scratch. Additional libraries for handling sub-processes, capturing computer load parameters (CPU, RAM, SWAP, GPU) and data handling.

Scikit-Image. Scikit-Image is an open-source image processing library for the Python programming language, which includes multiple algorithms for processing images efficiently [43]. In this thesis, the SSIM algorithm supplied by Scikit-Image is used.

Open-Cv. Open-Cv is a computer vision library that is open source and delivers multiple simple or advanced algorithms for image analysis, image processing, machine learning, and more. Open-Cv functionalities are available for multiple programming languages (C, C++ and Python) and simplify the development of automated tests using image analysis and processing.

Additional libraries. Some additional libraries used for this thesis that are of importance is the Python subprocess library, psutil library, matplotlib, and pandas. The subprocess library enables the user to spawn new processes from within Python and communicate with those [44]. In this thesis, this library is used for spawning and communicating with the Mentice simulation software from within Python. The psutil library is a library for grabbing information from processes as well as information about system utilization within Python [45]. In this thesis, the psutil library is used for evaluating system utilization (CPU, Memory, Disks). The Pandas library is a well known Python library for data analysis and manipulation [46]. In

this thesis, pandas is used for storing a large amount of data in a structured way. It is also used for performing statistical analysis on large portions of the data efficiently.

Matplotlib. Matplotlib is a visualisation library that is open source and supplies tools similar to the MATLAB interface. The tools provided by the library enables users to visualize their data in a static, animated or interactive way to clarify results and help with analysis.

3
Methods

The methodology for this thesis has mainly surrounded action research. This strategy was chosen due to its structure being well fitted for iterating over research and software development continuously. It also turned out to cover multiple aspects of the Mentice simulation software, software testing and NFRs classification and framework construction which served as a base for answering the RQ 1–3(*abc*) enumerated below.

1. How to portray and clarify the area of non-functional testing in the Medtech industry?
   (a) What non-functional requirements are applicable to the Mentice software?
   (b) How to depict a taxonomy for the specific non-functional requirements?
   (c) How to link the taxonomy to user experience?
2. How to construct an automated test framework for evaluating user experience in relation to a given "golden standard" for the aspect being tested?
   (a) How does the user interact with the software?
   (b) What is the "golden standard" for a specific test within the framework?
   (c) What NFRs have the highest impact on user experience?
3. How to connect measurements for a given simulation aspect being tested to user experience?
   (a) What is the connection between the automated test and manual user interaction?
   (b) How would the user interpret the aspect being tested if manual tests were performed?
   (c) How to link test aspects to real-life endovascular surgical procedures?

Figure 3.1 depicts the different parts for the action research and the workflow of this thesis. This workflow is iterated for each RQ. Furthermore, as the figure depicts, the action research for this thesis is split into three phases: *research, construction* and *simulation* phase.

3.1 Research phase

The action research in the research phase involved qualitative research in the form of interviews with project managers at Mentice and analyzing prior work within the area of automated software testing and NFRs. Information regarding what simulation procedures had bugs or visual defects was collected through these interviews.

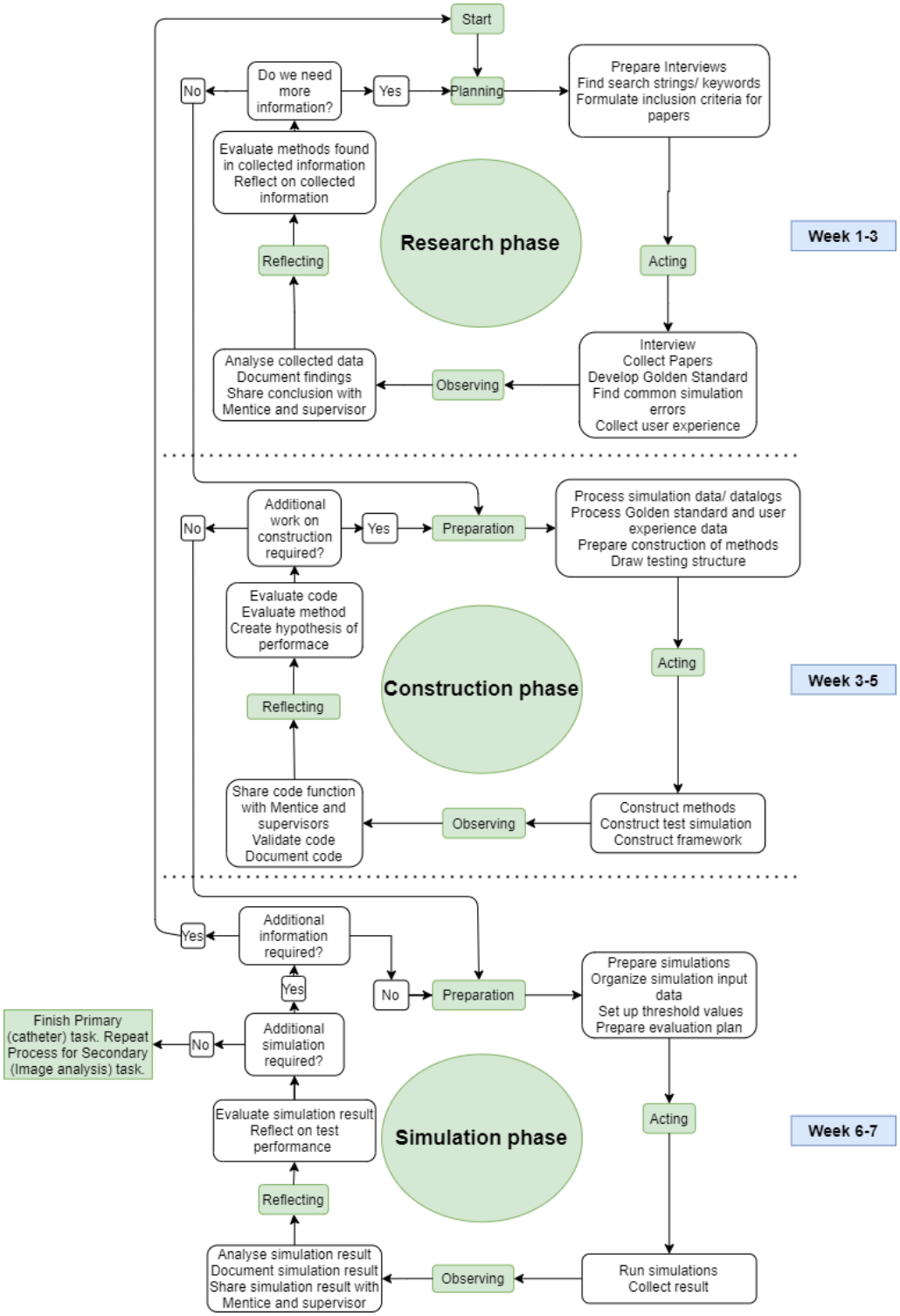


Figure 3.1: Flow chart depicting the structure of the methodology for this thesis. Note that each phase is iterated until satisfactory results are achieved, sometimes leading to previous phases being executed twice or more.

Insight of the Mentice software gathered from interviews with project managers served as a good base for RQ 2*c*, 3*a* and 3*b*.

Through additional discussions with software developers at Mentice, further insight into RQ 2*a* could be achieved. These interviews were conducted in order to get a deeper understanding of how the simulation platform was built, what information could be made available from a simulation, how the user interacted with the software, and what obstacles that were important to take into consideration when constructing an automated test framework. Additional valuable information connected to RQ 2*c* regarding user experience could also be gathered from these interviews. More user experience of the Mentice simulation software was gathered by practising various medical procedures on a Mentice simulation platform. The practice was performed both on the actual product as well as the emulator.

Furthermore, dialogue with a professional interventional radiologist served for increased understanding of how closely the Mentice simulation software mimics reality, and what differences that the radiologist had noted in the software compared to real endovascular procedures. This dialogue gave a better understanding of how automated tests could be linked to real-life endovascular surgical procedures, i.e., it gave insight for RQ 3*c*.

The research phase was performed in an iterative manner with each iteration adding more information of a certain RQ. Information gathered was reflected upon and the phase was iterated until a sufficient amount of information was gathered for a particular RQ.

3.2 Construction phase

The action research within the construction phase involved putting together suitable solutions/ideas found in the research phase (Sect. 3.1). To achieve a novel solution for the RQ being evaluated, it also involved collaboration with developers at Mentice regarding how data or software functionality, beneficial for the specific RQ, could be extracted from or built within the simulation software.

Similar to the research phase, this phase was conducted in an iterative manner for each research question. With each iteration increasing the level of completeness and complexity of the proposed solution. For each iteration, the solution was reflected upon and another iteration was performed if the solution appeared insufficient for answering the RQ.

3.3 Simulation phase

In this last phase, which mainly covers RQ 2–3 (and their sub-questions), simulations of the proposed solution were tested and evaluated against the Mentice simulation software. The desired outcome for a given simulation was outlined and the actual output from the simulation was compared to the desired outcome. Results were shared and discussed with supervisors and Mentice developers. If results turned out satisfying, documentation was finalized and another RQ was brought under

evaluation. On the other hand, if results turned out insufficient or dissatisfying, previous phases were iterated again.

3.4 Non-functional requirements road-map

Apart from the methodology described in Sect. 3.1, in order to answer RQ 1, additional scientific papers were sought out which focused on NFRs and the construction of a framework for such requirements. By studying prior work on constructing NFR classification and frameworks, a better understanding of such could be achieved.

Parallel to this study, NFRs applicable to the Mentice simulation software was identified with the purpose of answering RQ 1*a*. With an emphasis on user experience and the connection between simulation—reality, a few key NFRs linked to common software bugs were identified and ranked.

By studying a collection of frameworks and classification models from prior work, an idea of how to answer RQ 1*b* evolved. A taxonomy based on a combination of this prior work and knowledge of the Mentice Software served as a base for a novel NFRs taxonomy for Medtech software. Different from "regular" software, the Mentice software aims to be used as a training tool and to be perceived by the user as close to reality as possible. Divergence from reality within the software would sap the user experience and thus sap the training of a interventional radiologist, making the simulated training insufficient, or in the worst case, irrelevant. This aim was taken into great consideration when answering RQ 1*c* and constructing the framework since it is the most important aspect in terms of user experience and the development of a successful surgical training product. Figure 3.2 depicts this process.

3.5 An automated test framework

To answer RQ 2 and 3, a few automated tests had to be built, which connected NFRs with the user experience of the Mentice simulation. The NFRs *reliability, performance, scalability, portability* and *efficiency* were explored through automated tests. Close dialogue with supervisors and Mentice personnel made sure that the tests developed were relevant for the user experience of the software and for the NFR framework.

3.6 Non-functional testing: reliability

This section aims to describe the methodology used for developing tests connected to the *reliability* of the system. The methodology is based on the phases described in Figure 3.1 and each test was constructed over 6–7 weeks.

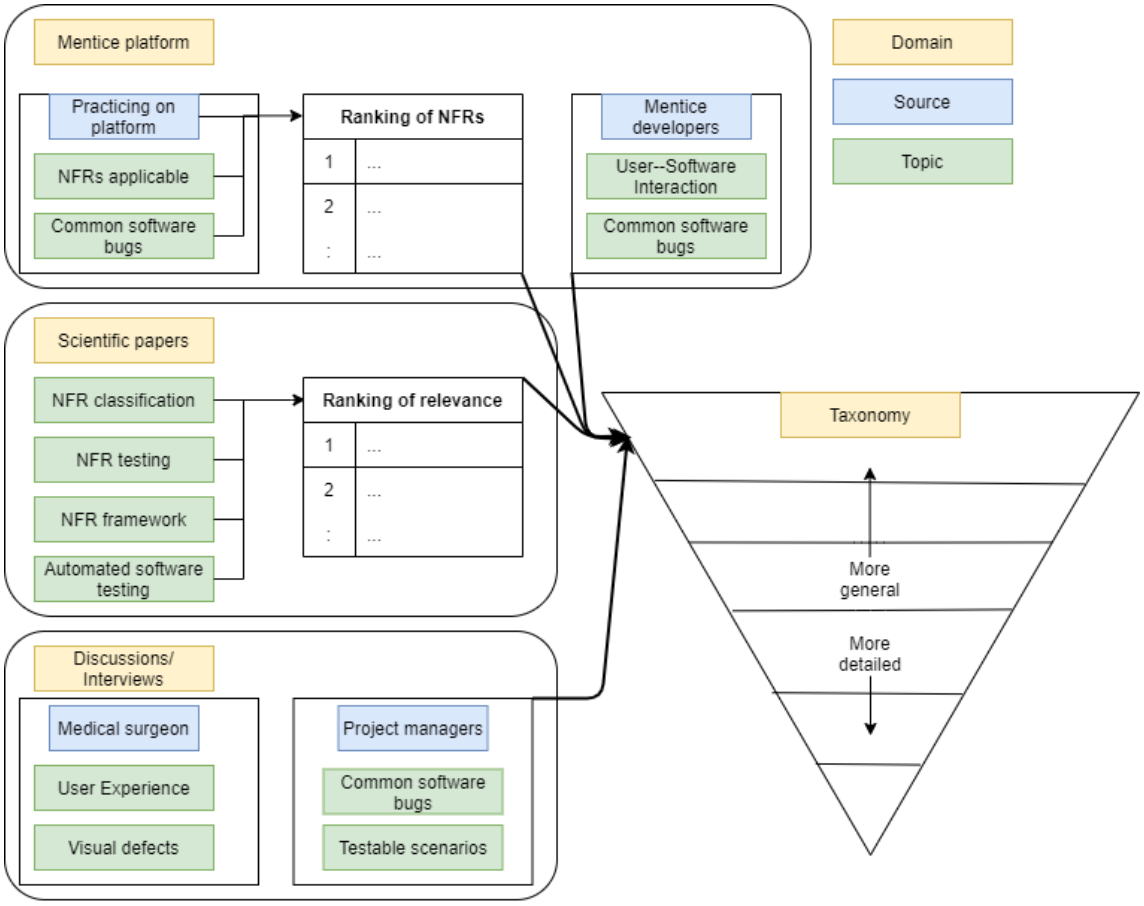


Figure 3.2: Methodology for constructing a taxonomy for NFRs. Note how different sources of information are combined for the solution.

3.6.1 Test for tool stability

The first reliability test evaluated a specific scenario in which the procedure tool started to shake/vibrate when entering some of the arteries within the simulated patient. This defect was proposed as a test case by one of the project managers at Mentice. The main objective for this test was to look at a set of images for a given procedure and assess whether or not the tool was shaking when placed in a specific artery.

By collecting data from the procedure in the form of images, tool coordinates, and tool translation, the output for multiple procedures could be analyzed and assessed regarding whether or not the behaviour of the tool was correct.

Iteration 1: Research phase. The first iteration focused on achieving a better understanding of the record&replay functionality as well as the Mentice simulation software by practising various medical procedures, replaying these scenarios and assessing the similarity of the replayed procedure to the one performed manually on the Mentice software. Various debug options within the simulation software were explored and assessed, a particular predefined debug option, which enables the user to view the arterial system, was used for further simulations. Furthermore, functionality for executing the replay-files automatically from within Python was added through a script. This script was also built to capture the fluoroscopic image, at a high rate, while replaying the medical procedure. The captured fluoroscopic images were stored as a set of BGR-images.

Iteration 2: Research phase. With the objective of isolating the tool from its surroundings in the fluoroscopic image, multiple image filtering methods were evaluated and compared for the stored image set. The image set was converted into three types of sets: RGB (Red, Green, Blue), BGR (Blue, Green, Red) and HSV (Hue, Saturation, Value). Different filtering ranges for each channel (RGB, BGR, HSV) were tested with the purpose of finding a sufficient filter for isolating the tool within the image, setting all elements, except for the tool, to black colour. By isolating the tool, within the image, from its surroundings, it was possible to assess whether or not the tool was shaking/vibrating.

This method proved dissatisfying since by only assessing the information within an image, it was impossible to draw conclusions regarding the user input for the tool. It was concluded that additional data, such as user input to the tool, was needed in order to make a robust assessment of whether the shaking/vibrations of the tool was based on software bugs or user input.

Iteration 3: Research phase. To achieve a more robust evaluation of the test, this iteration looked at various ways of scraping more data from the simulation. By discussing different approaches with Mentice software developers it was decided that additional information could be scraped and logged from the simulation by implementing a JavaScript (JS) script.

By implementing this script it was possible to grab data for tool coordinates (x, y, z), tool translation, and in which artery the tool was positioned. Based on the translation of the tool, additional data regarding tool coordinates etc. could be logged for the simulation. This new script was also implemented within the record&replay functionality which made it possible to automatically iterate a recorded medical procedure multiple times using a Python script. The result from these it-

erations was then assessed in terms of continuity, accuracy and user perception.

Iteration 4: Construction phase. In this iteration, a complete external Python script was built for assessing the stability of a tool within the software. This script was based on previous research iterations, knowledge of the Mentice simulation platform and its code-base.

It was also decided that the functionality for capturing an image set, built in **Iteration 1**, was redundant and unnecessary for sufficient assessment of the tool behaviour and thus removed from the solution.

Iteration 5: Simulation phase. The JavaScript script built in **Iteration 3** was implemented and tested against the Mentice simulation software. A record&replay file was replayed as input for the simulation, which ensured that the same input was given for each iteration. Result obtained in the log file by the JavaScript script was then evaluated using the Python script developed in **Iteration 4**. Finally, the result given by the Python script was controlled against the user perception concerning in which artery the tool had been unstable during replay. By controlling if the test identified the same true positives as the user would do, the robustness of the test could be evaluated.

Iteration 6: Improvement and finalization In the final iteration for this test, detecting whether a medical tool placed at a location within the virtual patient is shaking or not, the test was improved by grabbing tool coordinates at multiple positions along with the tool. Initially, only a single position along the tool was evaluated for stability and in order to obtain a degree of "shakiness"/instability additional positions were evaluated. This algorithmic change also suggested to reducing the number of false positives for the test result. Additionally, various thresholds for changes in coordinates (x, y, z) of the tool was tried in order to answer RQ 2*b*. A complete description of the finalized solution and algorithm for this test will be covered in Sect. 4.4.1.

3.6.2 Detection of tool outside artery walls

The second *reliability* test focused on the problem of the tool taking shortcuts through/lay outside of, solid veins. Meaning that the tool could jump and skip small sections of the simulated endovascular system. The objective was to identify portions of the tool which lay outside of the arteries. This was done through image analysis. The test could be used to check tool behaviour between software and hardware updates.

Iteration 1: Research phase. Similar to the first iteration in Sect. 3.6.1, the development of the test began by gathering images of the tool taking shortcuts and saving these images into datasets. The Mentice system was put into a specific debug mode with only arteries and tool visible in the fluoroscopic image. This specific debug mode allowed the images to have more defined colours, where the arteries are more defined in the colour red and the tool in blue.

The decision to use the debug mode was made due to the fact that using the standard fluoroscopic image (gray-scale) or predefined debug modes within the simulation resulted in images that were difficult to analyse. By using specific debug settings it was easier to distinguish the arteries from the tool using image analysis

and a simpler and more robust filtering algorithm could be used for the image.

Iteration 2: Research phase. As a first approach to finding parts of the tool outside of an artery, basic colour filtering was used. By using Python library Matplotlib to observe the colour range of each feature (tool and arteries), a notion for which colour range to filter for each colour channel (RGB) could be obtained. By tweaking the filter parameters, while simultaneous watching the resulting image, a good estimate for which range to filter for a given colour channel could be achieved. Filtering was done using Open-CV (Python library) functions.

Furthermore, this filter was applied to the complete image dataset resulting in a filtered image set. This filtered image set was then evaluated against the ground truth (the original image set), in which the user could notice parts of the tool being outside of a given artery, and evaluate if the filtering method proved to detect these areas. This filtering method proved sufficient for single image analysis, but when performed over a set of images, it resulted in high redundancy in the result since multiple succeeding images being detected by the filtering method looked "similar" to the user.

Iteration 3: Research phase. The focus for this iteration was to find a method to avoid redundancy within the resulting image set from the filtering process. Multiple image analysis algorithms were evaluated for detecting equality between images within the filtered image set.

Image analysis algorithms evaluated were: Template Matching, Structural Similarity Index Measurement (SSIM), Scale-Invariant Feature Tracking (SIFT), and Optical flow. The most promising image analysis algorithms were further explored for their use within this test case. To solve the redundancy issue, the orientation of the two most promising algorithms, SSIM and SIFT, were explored and evaluated for finding equality between images in the filtered image set. result is given as a *Structural Similarity Index* which is often set as 0 (no similarity) and 1 (complete similarity) [47]. In the following formulas, the comparison is made between images which are denoted as image x and image y.

Iteration 4: Construction phase. In this iteration, the previously evaluated algorithms (SIFT, SSIM), were combined into one script together with the filtering method evaluated in **Iteration 1** and **2**. This combined script was evaluated and various thresholds regarding SIFT, SSIM, and the amount of tool area that was required to be outside of an artery, within an image, in order to be classified as a problem. By discussing various pros and cons of higher/lower thresholds for the different algorithms, a better sense of an answer for RQ 2*b* could be achieved.

Iteration 5: Simulation phase. After evaluation, the combined script constructed in **Iteration 4** was tested and evaluated against the Mentice simulation software. To achieve a high degree of accuracy in user input between iterations, a record&replay file was used as input. The result from the simulation was evaluated against the user perception of where the tool had travelled outside of an artery. The redundancy in the test result was also investigated in terms of the *uniqueness of test results (images)* and *number of hits (images) in the result*.

Iteration 6: Improvement and finalization. After testing the structure of the method in **Iteration 5**, a decision was made to swap the order of the SIFT and SSIM algorithms within the constructed Python script. This change was made in

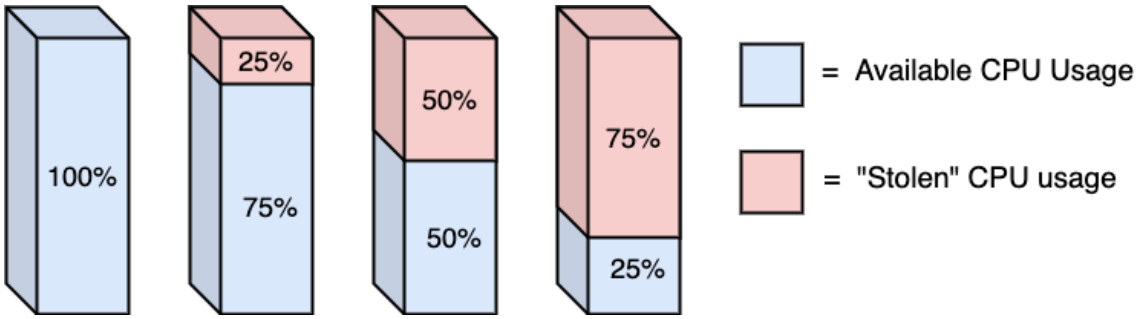


Figure 3.3: Illustration of how the external program *CPUStres* steals CPU power.

order to increase *the number of hits (images) in the result*, but without increasing the redundancy in the result (low image uniqueness). In order to increase the uniqueness of the images in the result, morphological skeletonization was performed on the tool in order to compute its length. The complete algorithm for this case is further explained and illustrated in Sect. 4.4.2.

3.7 Non-functional testing of performance, scalability, portability, and efficiency

The last test for the automated test framework evaluates the NFRs *performance, scalability, portability* and *efficiency*.

Iteration 1: Research phase. In this iteration, it was decided to measure how the computer performed during different procedures and different loads, and how these parameters could affect the user experience. Both existing system utilization programs for Windows and Python libraries for system utilization were explored and evaluated in terms of pros and cons.

Iteration 2: Construction phase. After exploring the available alternatives for measuring system utilization, a script was developed, using Python, for capturing computer performance parameters. The script was evaluated in terms of what system utilization parameters were relevant for the Mentice software as well as had a clear connection to the user experience of the software. The different parameters, measurable by the script, was discussed with supervisors with the aim of finding the most relevant parameters for measurement.

Iteration 3: Simulation phase. The script constructed in **Iteration 2** was tested using two different medical procedure simulations which required more or less from the computer hardware. The script was executed on two different hardware (*DELL PRECISION M6700/M6800*) and a program for applying CPU load (*CPUStres* [48]) was used for finding bottlenecks within these medical procedures. The result was analyzed and system utilization for the different loads (on different hardware) was noted. *CPUStres* made it possible to increase the load by a step of 25% and setting the priority level of the applied load. Figure 3.3 depicts this.

Due to system utilization parameters having a weak correlation with the actual user experience of the software, the research and construction phase had to be reiterated once more to find a suitable measure for user experience and to achieve a

better measure for answering RQ 2*b*.

Iteration 4: Improvement and finalization. By discussing possible ways of measuring user experience linked to the four NFRs assessed in this test and to answer RQ 2*b*, additional parameters were taken into the measurement. A JavaScript script was implemented for logging the frame rate (FPS) of the specific medical procedure simulation. Through dialogue with Mentice developers, additional information regarding the physics engine frame rate (PFPS) was logged through a JS script.

When the test was improved and finalized, simulations were performed for the two given medical procedure simulations. Both procedures were executed 3–5 times each for a specific CPU load using CPUStres. The simulations were performed an equal amount of times for each hardware setup and all external programs were terminated leaving only the test script, CPUStres, and Mentice simulation running. The result received from different hardware and different CPU load was analyzed and compared. The mean and variance of the FPS and PFPS over all iterations for a given CPU load was computed and compared across different hardware setups. The mean simulation time over all iterations of a given procedure and CPU load was also computed and compared across different hardware setups.

4
Results & analysis

This chapter aims to describe the result obtained from the process illustrated in Chap. 3 and to answer the research questions (RQs) described in Sect. 1.3. The result includes an NFR taxonomy/road-map, an automated test framework, and specific test results carried out as part of this automatic test framework. This chapter also involves an analysis of the results obtained for each RQ.

4.1 Research questions repeated

In the list below are research questions mentioned earlier in Chap. 1 and Chap. 3 restated for clarity. This chapter (Results & analysis) aims to answer these questions through the results obtained within this thesis.

1. How to portray and clarify the area of non-functional testing in the Medtech industry?
   (a) What non-functional requirements are applicable to the Mentice software?
   (b) How to depict a taxonomy for the specific non-functional requirements?
   (c) How to link the taxonomy to user experience?
2. How to construct an automated test framework for evaluating user experience in relation to a given "golden standard" for the aspect being tested?
   (a) How does the user interact with the software?
   (b) What is the "golden standard" for a specific test within the framework?
   (c) What NFRs have the highest impact on user experience?
3. How to connect measurements for a given simulation aspect being tested to user experience?
   (a) What is the connection between the automated test and manual user interaction?
   (b) How would the user interpret the aspect being tested if manual tests were performed?
   (c) How to link test aspects to real-life endovascular surgical procedures?

4.2 NFR taxonomy

Based on research findings, gathered through the methodology described in Sect. 3.1 and Sect. 3.4, research question (RQ) 1(a-c) and 2(c) could be answered through a taxonomy. This taxonomy aims to portray the area of NFR testing in the Medtech

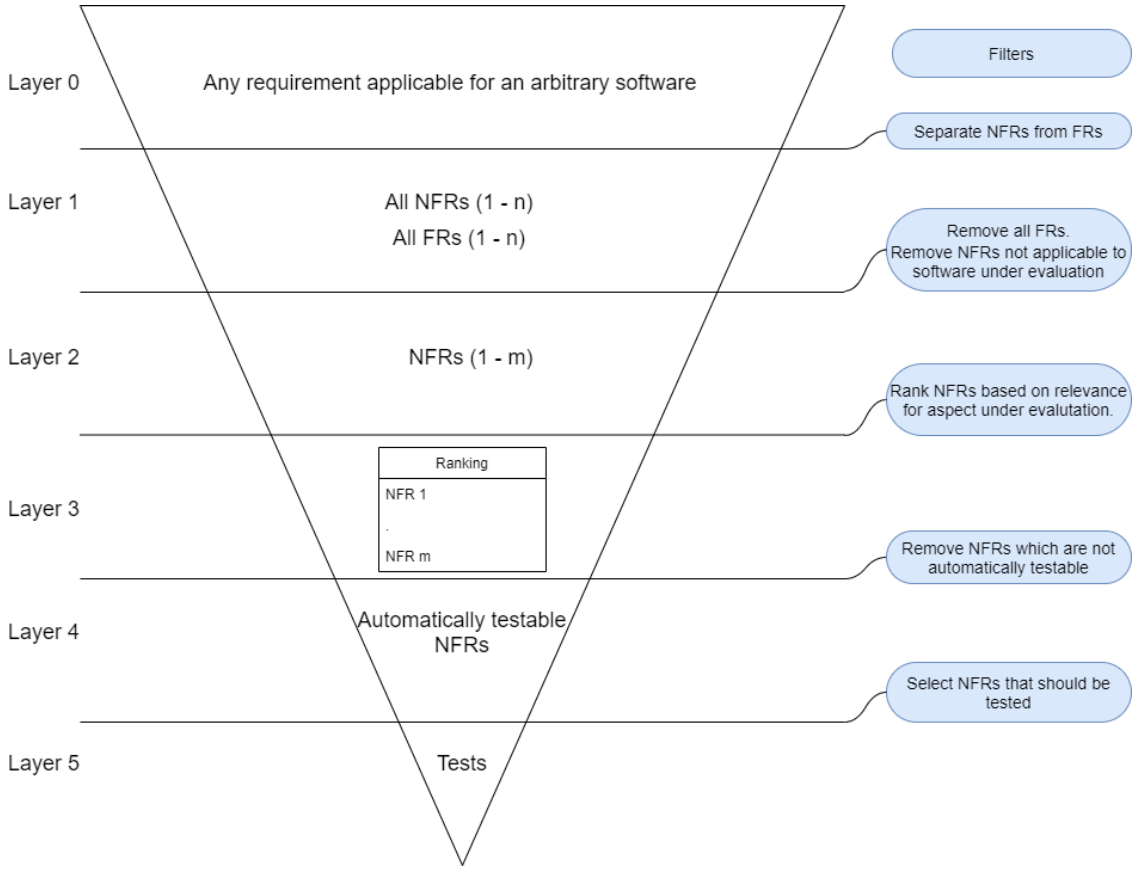


Figure 4.1: Taxonomy for creating testable NFRs. Note how each filter reduce the number of requirements brought into the next layer of the taxonomy thus increasing clarity within the testing phase.

industry and to link testable aspects to user experience. Figure 4.1 describes this taxonomy.

The taxonomy serves as a filter for answering RQ 1*a* by obtaining a clearer view of what NFRs are applicable to a specific product, in this case, the Mentice simulation software. Furthermore, the taxonomy aims to depict how NFRs are integrated into it, i.e., answering RQ 1*b* and how "user experience is related to the taxonomy (RQ 1*c*). Further description of each filter is obtained in Sect. 4.2.1–4.2.5.

4.2.1 NFRs and FRs

The first filter of the taxonomy depicted in Fig. 4.1 separates NFRs from FRs. A basic fundamental property of this filter is given from [10] in which it is stated:

- FRs: "what a software should *do*."
- NFRs: "how a software should *be*."

Based on this property, a requirement can be separated based on if it describes *what* specific functionality the software should *be able to do* (FR) or *how* it should *be able to do* a specific functionality (NFR). A concrete example is:

- FR: "Software should be able to navigate a virtual medical catheter through a virtual patient's arteries through user input from a user interface."

- NFR: "The navigation of a virtual medical catheter through a virtual patient's arteries should be smooth and without interruption."

4.2.2 NFRs applicable to software

The next filter is separating the gathered NFRs based on their level of importance for the particular software being investigated, i.e., answering RQ 1*a* by removing irrelevant NFRs. In the case of this thesis, no NFR linked to network functionality or software sharing passed this filter. This due to no network topology being necessary for using the software and the source code is not being shared between Mentice and a third party agent.

4.2.3 Ranking of NFRs

The following filter aims to rank NFRs based on their relevance. Relevance could be linked to the frequency of which a specific NFR appears during an assessment. Relevance could also be linked to a certain aspect and then the ranking will be based on how much relevance a certain NFR has to the specific aspect.

In this thesis, all applicable NFRs are ranked with respect to the user experience of the software. This ranking links the taxonomy described to user experience and thus answers RQ 1*c*. It is important to note, that although user experience *can* be linked to all NFRs in some way, this filter serves to clarify the degree of importance for a given NFR based on what the user values the most. For the specific software evaluated in this thesis, which is a training tool for interventional radiologists, user experience has been linked to *how closely the software mimics reality*. Although this link between simulation vs. reality is not the only important aspect of user experience, it is important for developing a useful training tool that mimics real-life surgical procedures as closely as possible.

Furthermore, based on [10], sub-categories of NFRs such as ***Maintainability*** where: *Analyzability, Changeability, Testability, Reusability* are the sub-catergories, and the sub-categories of ***Compliance*** such as: *Compliance, Documentation, Legal & Licensing issues* have a weak connection to the user experience defined above and are thus ranked lower.

4.2.4 Automated testing

The last filter of the taxonomy aims to separate the gathered NFRs which are automatically testable and which are not. Due to the previous filter, all NFRs left at this stage of the taxonomy have some link to user experience. The purpose of this filter is to remove NFRs which are difficult to test automatically given today's tools and technology. A few examples of NFRs that could be removed here are *usability*, *documentation*, and *accessibility*.

4.2.5 Tests specific for thesis

At this stage, out of all FRs and NFRs discovered in Sect. 4.2.1, only a few NFRs passed all filters within the taxonomy. Out of these existing NFRs, automated tests

can be built for testing that the software complies with these requirements.

In this thesis, tests developed focused on NFRs linked to *reliability, performance, scalability*, and *portability*. Tests developed and their corresponding result is further explained in Sects. 4.4–4.5.

4.3 An automated test framework

The chosen NFRs (see Sect. 4.2.5), which passed through the filters within the taxonomy, was further investigated and automatic tests were constructed to test these requirements. Tests were performed automatically or semi-automatically and each test developed had its own specific steps and functionality for executing successfully. Figure 4.2 depicts a general overview of the framework used when building tests in this thesis. In this framework, specific medical procedures were chosen for the evaluation of a specific NFR.

Information regarding a specific NFR and how it could be tested was collected through interviews with Mentice developers and by reading documentation about the Mentice software and specific procedures. This information was necessary in order to answer RQ 2–3, in which all questions relating to how the user interacts, interpret and evaluate the Mentice simulation platform. This information was used for building evaluation scripts in Python which assessed these bugs/defects automatically and made sure that the information gathered was assessed automatically by the script.

The results from the automatic tests were further evaluated, manually, to find weaknesses in these tests and their results. The golden standard was picked based on dialogue with a professional radiologist and/or Mentice developers. The golden standard was not always described as a strict measure but more often as a range of results that would serve as sufficient for securing the user experience.

Additionally, by discussing how the Mentice simulation is tested manually with project managers and developers, this information could be used for building evaluation scripts within Python which looked for similar defects within the simulation automatically, thus answering RQ 3*b*.

Through additional discussions with a professional radiologist, more aspects linked to real-life endovascular surgical procedures could be brought into the Python evaluation test script . These additional aspects aimed to answer RQ 3*c*, i.e., “how to link test aspects to real-life endovascular surgical procedures”, and to provide a test that better evaluated the Mentice simulation in relation to reality.

4.4 Non-functional requirement: Reliability

This section presents the results from tests focusing on the *reliability* of the Mentice simulation software. Two tests related to the *reliability* of the software were constructed and their results were analysed.

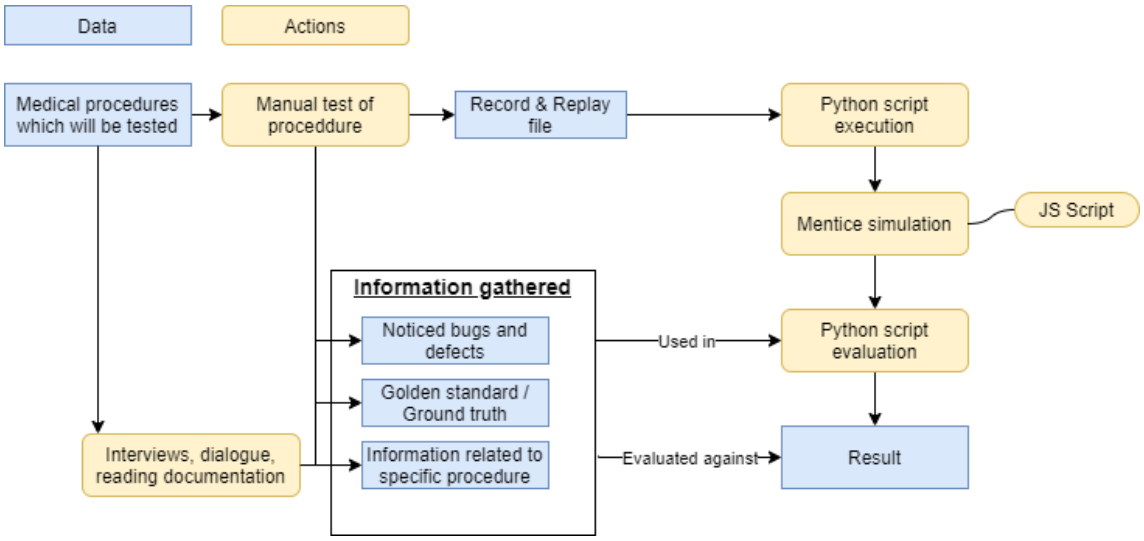


Figure 4.2: Overview of the general approach for all tests developed within this thesis. Note how manual testing and interviews, dialogue, and documentation are combined and brought into test-scripts.

4.4.1 Result: Test for tool stability

The purpose of this test was to detect whether or not a medical tool within the simulation was stable. By following the multiple steps described in Sect. 3.6.1, it was possible to construct a test that could determine whether or not a medical tool was stable within the simulation. By tracking the translation for the tool (user input) and measuring tool coordinates within the simulation, the stability of the tool could be evaluated.

Since the record&replay functionality closely mimics the actions of the user, with exceptions based on the non-deterministic behaviour of the software, the record&replay file was used as input for each result. This made it possible to answer RQ 2*a*, by constructing input manually, which closely mimicked how the user would normally use the software and then replaying this input. Additionally, by using the record&replay functionality, the input could be ensured to be as similar as possible for each test iteration, enabling evaluation of only the output and assuming that the input stayed equal throughout the testing phase. Furthermore, by using the record&replay functionality, the connection between manual and automatic user input could be strengthened since the difference between manual user input and "automatic computer input" (record&replay file) could be minimized (RQ 3*a*).

To answer RQ 2*b* for this test, a threshold was used to identify instability for a tool. Multiple thresholds were tested and a threshold of ± 0.05 coordinate change between frame proved to be sufficient, i.e., if the x, y or z coordinate moved ± 0.05 between two simulation frames, instability of the tool was detected. To deal with the non-deterministic behaviour of the simulation, the thresholds were set to identify **more than** ten instabilities within a specific artery and tool, for the test to detect an unstable tool for that particular artery. There was no significant difference in using a slightly lower/higher threshold since if the tool was unstable it would show a larger movement than ± 0.05, and if not, the movement of the tool would be very

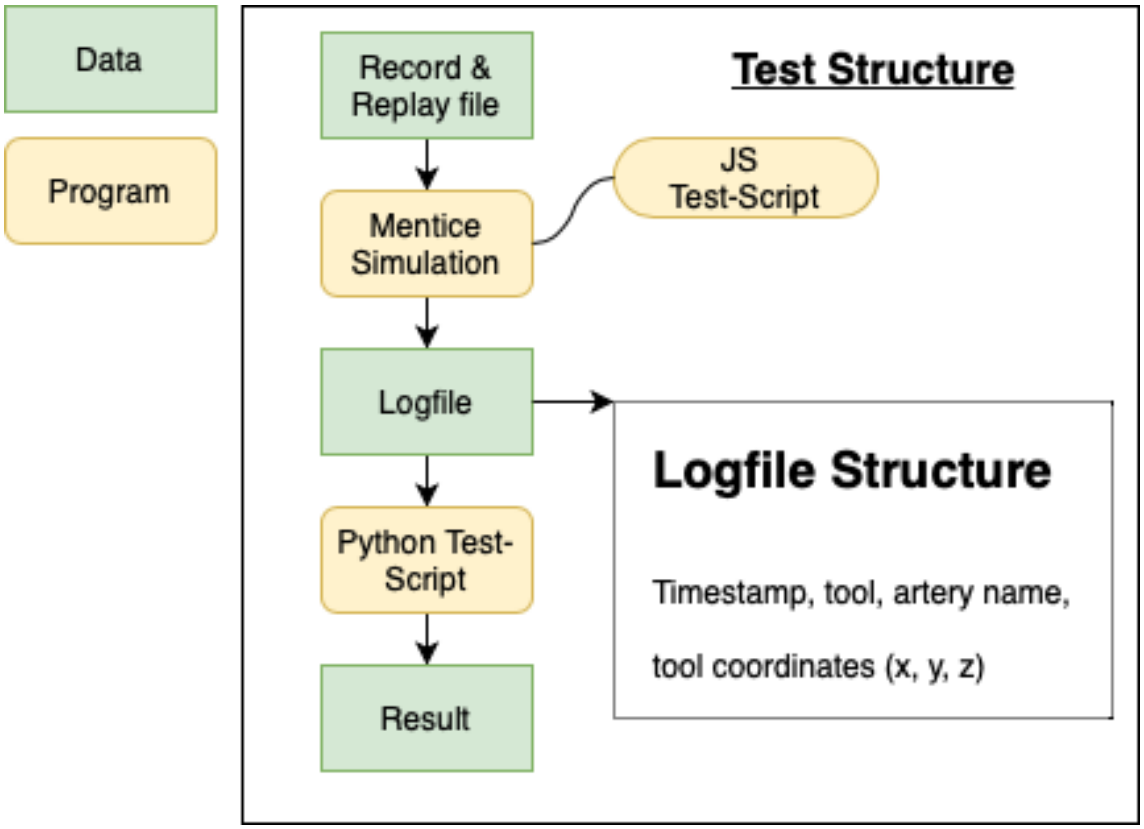


Figure 4.3: Structure of test for detecting shaking tools. Note how an internal JS-script is used to grab additional simulation information into the logfile.

low, or nonexistent, between simulation frames. This with an exception when the tool was located close to the heart, in which the tool experienced more force due to the beating of the heart.

Result 1: Measuring tip of the tool. The first finalized version of the test tracked user input and evaluated the movement of the tip of the tool when no user input was given for a certain time period. Figure 4.3 depicts the structure of the test. Tables 4.1–4.2 show the main steps of the JS and Python scripts.

By executing the test depicted in Fig. 4.3, the following result displayed in Fig. 4.4 could be obtained. The result was obtained on a *DELL PRECISION M6800* computer (denoted as *DELL68*) and code branch *B1*. This result proved different from what the user perceived when watching the simulation run, i.e., the ground truth. The result shows that the test detected two unstable tools at location *"6_40_453_Aortic_root"*. Compared to the ground truth found trough manual testing and interviews (See Fig. 4.2), this result was false since no drastic instability was noticed by the user. This suggested that the test had to be improved further.

Result 2: Measuring multiple positions along with the tool. By measuring multiple coordinates along the tools within the virtual patient, it was possible to achieve a more robust evaluation regarding the stability of each tool. By adding another parameter *relative position* to the logfile in Table 4.1, each *timestamp, tool, artery name, and tool coordinates* also got a *relative position*. This *relative position* was computed from the tip of the tool and towards the source at which the tool was

Table 4.1: JS test-script used for capturing data from Mentice simulation.

JS test-script (once every simulation frame, iterated for each tool within system)

1. Check tool translation and add to buffer: $[tt_0, ..., tt_{99}]$
2. Compute mean-translation from buffer: m_{tt}
3. If buffer is filled more than 50%:
   (a) Pick first element of buffer: tt_0
   (b) If $m_{tt} - 0.02 < f_{tt} < m_{tt} + 0.02$ (i.e., tool translation buffer mean $\approx tt_0$:
      i. write timestamp, tool, artery name in which tool is at and tool coordinates (x, y, z) to logfile.

Table 4.2: Python test-script used for evaluating logfile from simulation.

Python test-script (executed with simulation logifle as input)

1. Parse logfile data into separate tool dataframes $toolT$ where:
   (a) $T = 0, 1, 2$
   (b) Each $toolT$ includes: time+artery, tool, x, y, z (global coordinates).
2. For each $toolT$ dataframe compute for each artery the difference between coordinate x, y, z between samples:
   (a) $toolTdiff[j] = abs(toolT_{i-1} - toolT_i)$, $\forall i \in (1, ..., n-1)$
   (b) If $toolTdiff[j] > t_h$:
      i. number of rapid changes in tool at specific artery: $nrcToolT[artery]$
      ii. $nrcToolT[artery] = nrcToolT[artery] + 1$
3. For each $nrcToolT[artery]$ check whether number of rapid changes exceed threshold nrc_{th}:
   (a) If $nrcToolT[artery] > nrc_{th}$:
      i. $toolTstable[artery]$=False
   (b) else:
      i. $toolTstable[artery]$=True

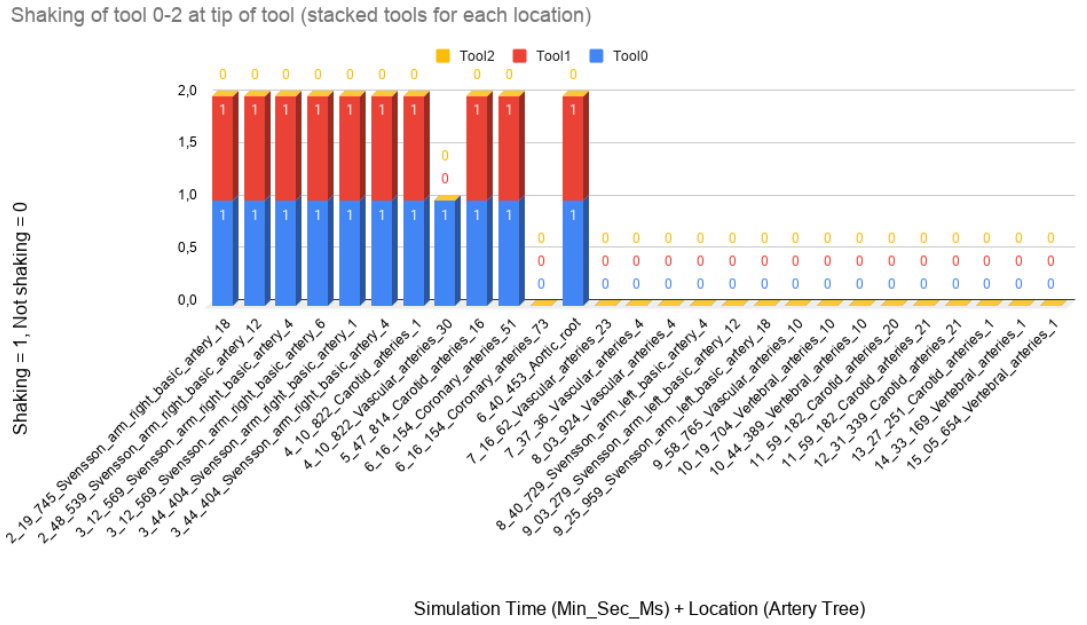


Figure 4.4: Result when measuring tool stability at the tip of the tool on branch *B1* using the *DELL68* computer. Note that the scale is only presented as "unstable" = 1, "stable" = 0.

inserted. Figure 4.5 depicts this change in measurement points for tool coordinates.

Furthermore, the Python script in Table 4.2 was extended by checking tool stability at each *relative position* instead of only the tip of the tool. Figure 4.6 displays the new result in which a *degree of instability* is presented. This result was gathered on the *DELL68* computer and the same code branch as before (*B1*). As seen in Fig. 4.6, the *"6_40_453_Aortic_root"* has a degree of instability of 1 for *Tool0* and 2 for *Tool1* which better reflects the ground truth, i.e., what the user perceived when looking at the simulation.

Result 3: Tool stability between different code branches and hardware Tool stability was also evaluated for a newer code branch of the Mentice simulation platform denoted as *B2* here (compared to previous *B1*). In Fig. 4.7, the same test, and input, is executed on a newer branch of code. As depicted by the figure, many locations which were evaluated as unstable for branch *B1* are now evaluated as stable. This suggests that the tool stability had been improved between branch *B1–B2*. Although most of the locations visited in Fig. 4.6 are visited in Fig. 4.7, there are some slight differences at which locations the tool visited for the last measurement of this test. This difference could be explained by the non-deterministic behaviour of the Mentice simulation system and the variations between code branches (*B1, B2*).

Tool stability was also evaluated on different hardware for branch *B2*. The result, given the same input (record&replay file), is displayed in Fig. 4.8. This result was obtained on a *DELL PRECISION M6700* computer (denoted as *DELL67*). The output from the test performed on this computer is only slightly different from

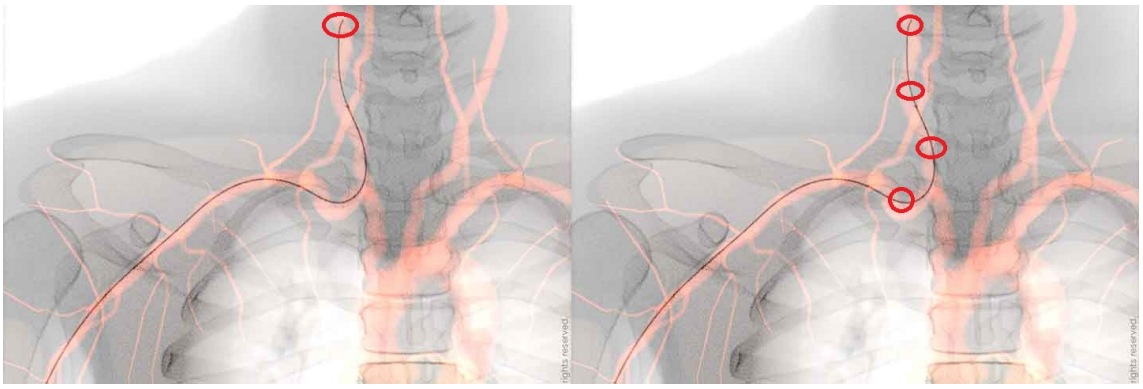


Figure 4.5: Improvement of test algorithm. Measuring, at red circles, tool coordinates (x, y, z) within the simulation.

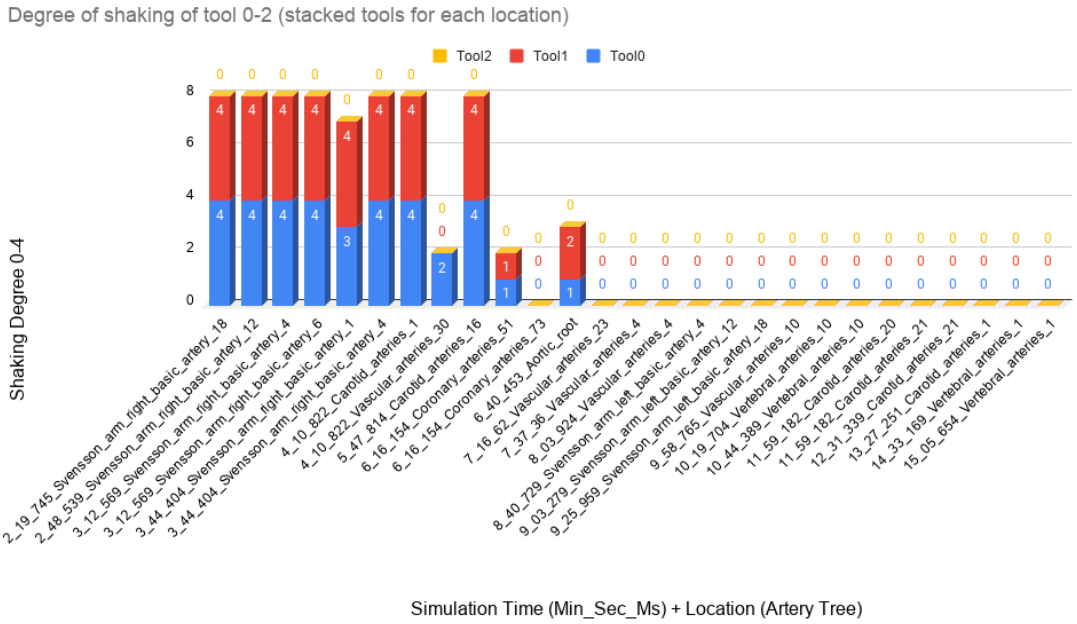


Figure 4.6: Result when measuring tool stability at multiple positions along tool for branch *B1* using the *DELL68* computer. Note how this result, in relation to Fig. 4.4, presents a degree of instability (0–4).

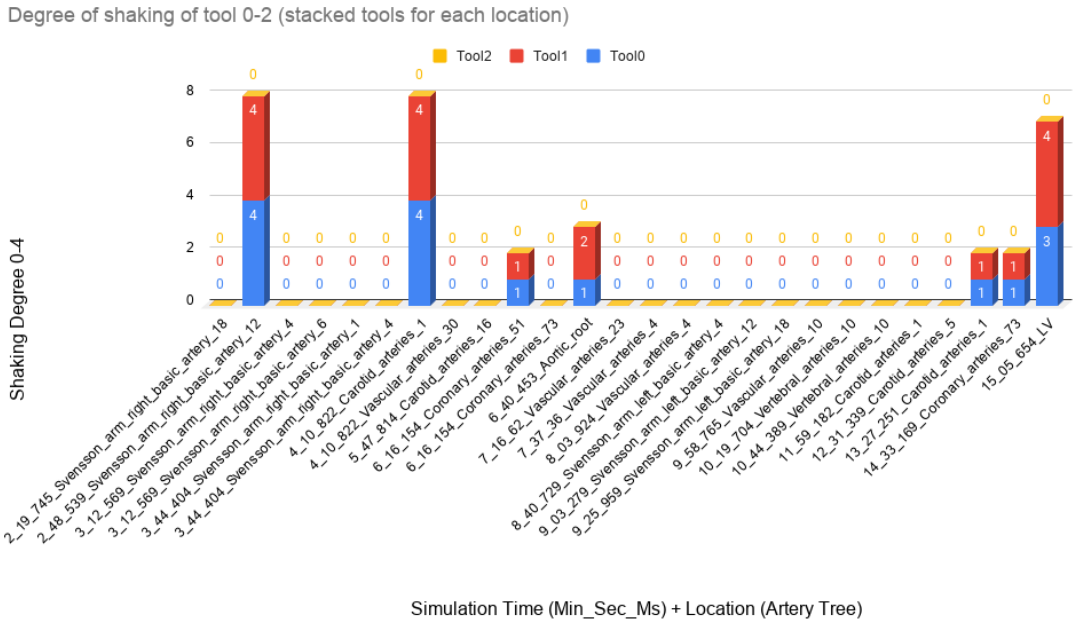


Figure 4.7: Result when running test on a newer code branch *B2* using the *DELL68* computer. Note how the locations visited (x-axis) are slightly different from the ones visited in Fig. 4.6.

the one displayed in Fig. 4.7. The difference can be seen for *"6_16_154_Coronary_arteries_73"* in which the test evaluated the tool as stable for the *DELL68* computer and a degree of instability of 1, for *Tool1*, for the *DELL67* computer. This difference could be explained by the non-deterministic behaviour of the simulation and slight variations in hardware with the *DELL68* being more powerful and thus able to perform more computations for the optimal tool position within the simulation.

Result 4: Physics change of simulation. Lastly, the physical properties of the tools were modified for the simulation to evaluate the behaviour of the tools under these new properties. The link length of the tools was shortened thus increasing the number of collision points and flexibility of the tools within the simulation. Figure 4.9 depicts this physical change of the tools. As shown in the figure, this modification was performed at two different degrees using *shorter links* and *even shorter links*, which were predefined settings within the JS code for the Mentice simulation.

By decreasing the link length and running the test, the result depicted in Figs. 4.10–4.11 could be obtained. As can be seen in these figures, modifying the link length of the tools changes the output from the test quite drastically with the locations being different from the locations visited by the test in Figs. 4.7 and 4.8. This is due to altering the length of the tool will yield a different path of the tool within the simulation although the same input is used. The difference in locations visited by the tool is also noted between Figs. 4.10 and 4.11. This problem makes the comparison difficult between default settings used in the previous test scenarios

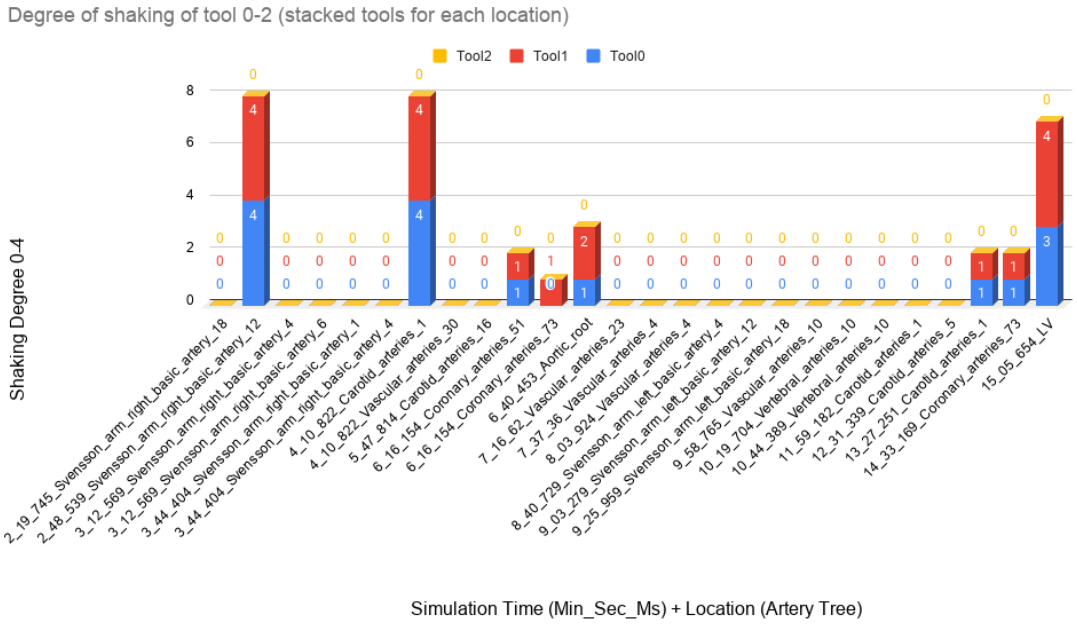


Figure 4.8: Result when running on test on code branch *B2* using the *DELL67* computer. Note that the result is different for *6_16_154_Coronary_arteries_73* compared to Fig. 4.7.

and these new modified settings.

Another difference is that although the tool visits the same location/artery within the simulation, though at different timestamps, the tool will not be placed at the exact same coordinates within these locations and thus the result will be different when assessing the stability of the tool. Compared to previous test comparisons, the computer on the same code branch (Figs. 4.7–4.8) or test algorithm (Figs. 4.4–4.6), there is simply too many parameters such as link length, collision points, and non-deterministic behaviour of the software that makes the comparison of this test output implausible to the previous test result with default link length and number of collision points.

4.4.2 Results: Detection of tool outside artery walls

For the second test, the purpose was to detect areas of the tool being outside of the artery walls. Using the methodology in Sect. 3.6.2, it was possible to detect these areas. The camera, which tracks the tool within the simulation, was set to automatically track the tool of interest. This enables the tool of interest to be centred in the fluoroscopic view.

The approach of the test was to extract unique images in which the tool was outside of the artery walls from a dataset captured during simulation. As described in Fig. 4.17, the images are captured and saved during a replay of a record&replay file, resulting in a dataset of fluoroscopic images ranging from between 100–4000 images. From these datasets, unique images are extracted using a combination of

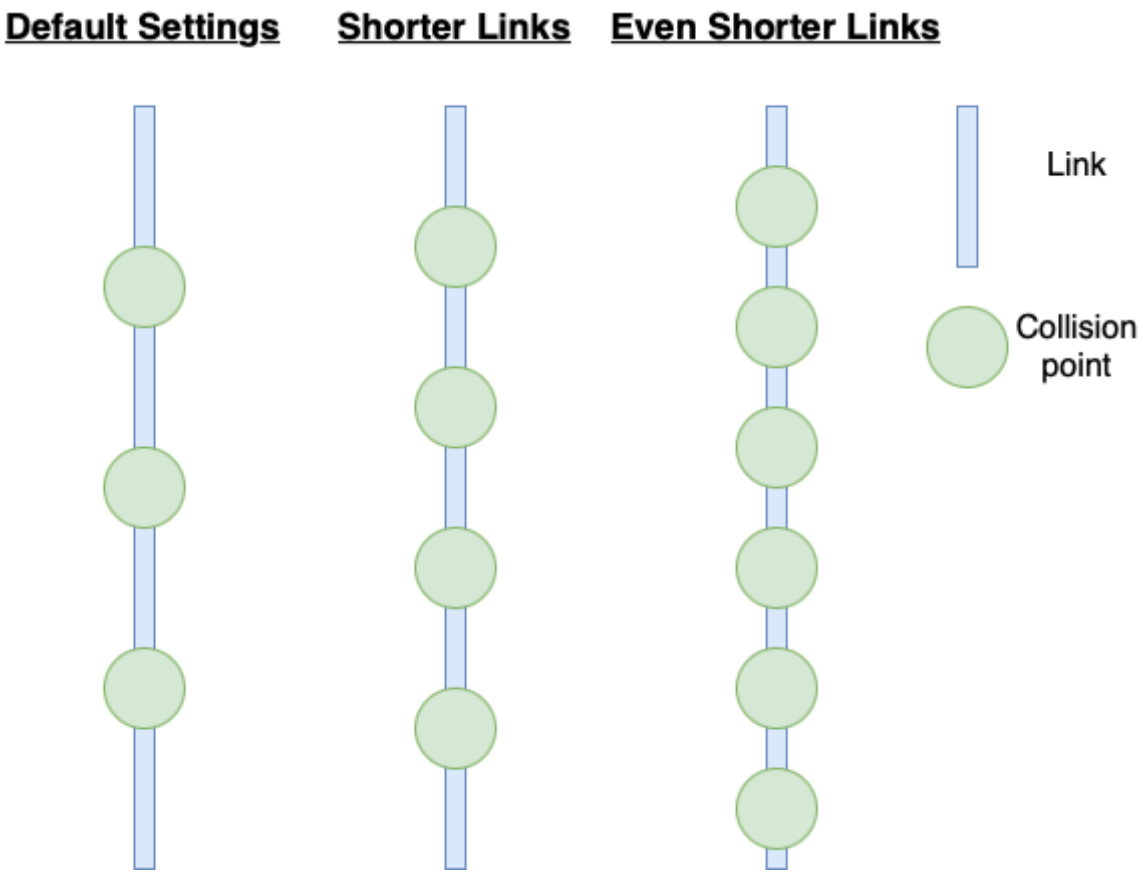


Figure 4.9: Link length modification. Note how the number of collision points for the medical tool increase as the link length decrease.

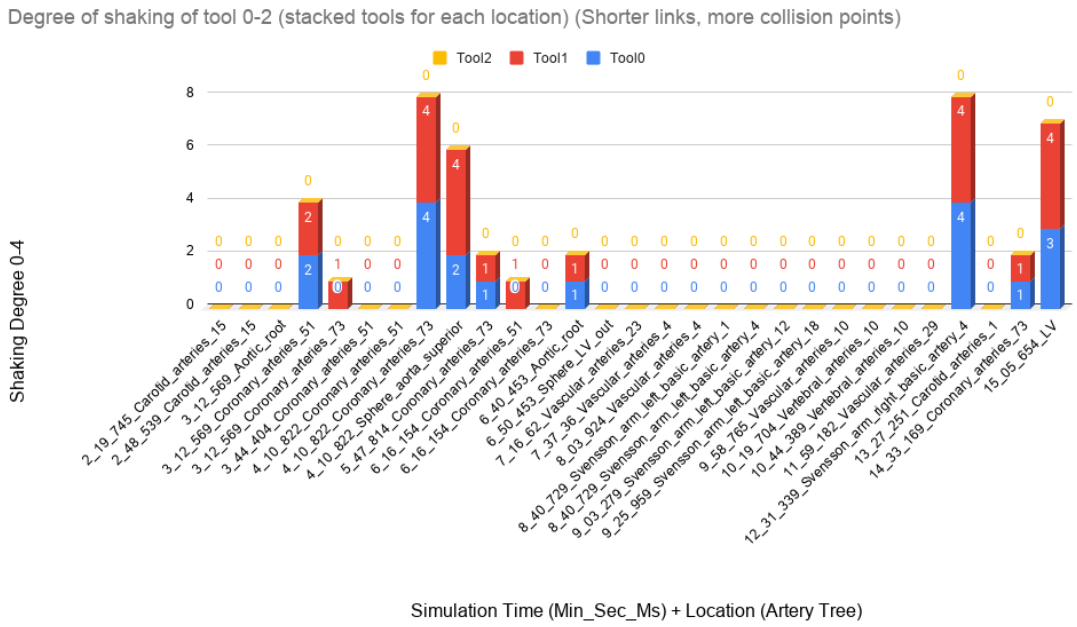


Figure 4.10: Result when shortening the link length on branch *B2* using the *DELL68* computer. Note how locations visited and their timestamps (x-axis) are very different from Fig. 4.7 altough same input and code branch is used.

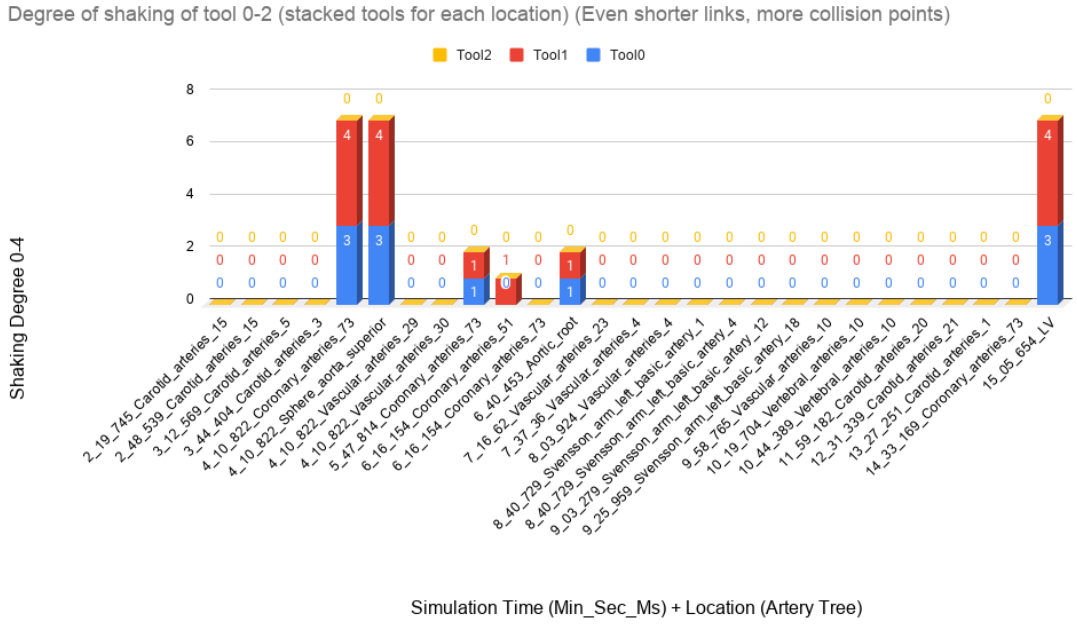
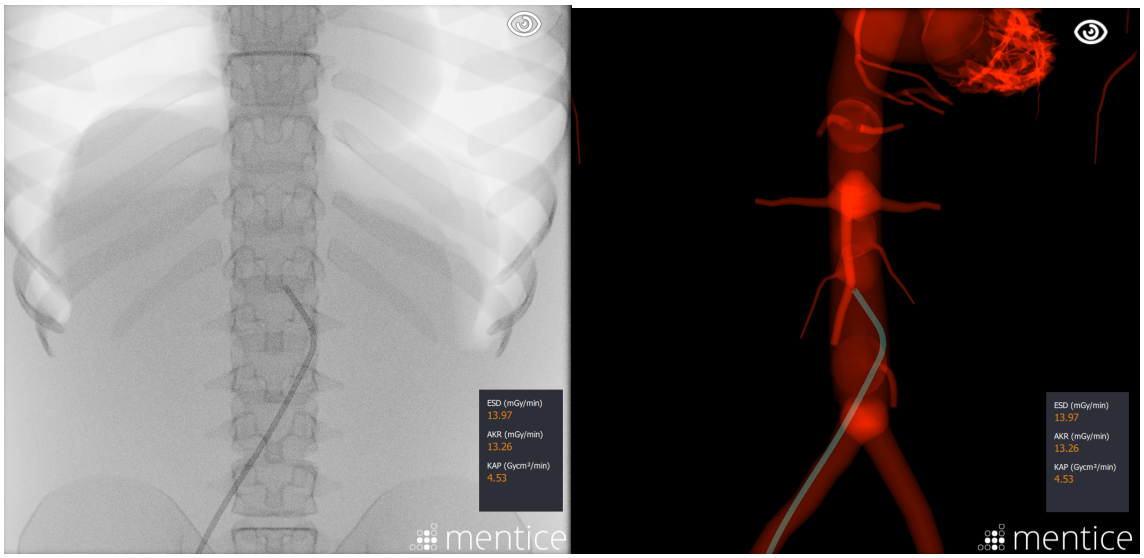


Figure 4.11: Result when shortening the link length even further on branch *B2* using the *DELL68* computer. Note how locations visited (x-axis) are different from the ones visited during short link lengths in Fig. 4.10.



(a) Default mode **(b)** Specific debug mode

Figure 4.12: Visualization of the fluoroscopic image in default mode and in a "specific debug mode". Note how, apart from information and logos, only the arteries and tool are visible within the fluoroscopic image.

available image processing tools. The best approach found was to apply the image processing algorithms in the following order: colour-filtering, image grouping, SIFT, tool movement tracking, and SSIM. The descriptions of these algorithms are not covered to their full extent within this section, but only their functionality within the proposed test. The extensive description of SIFT, SSIM, and tools used for tool movement tracking can be read in Appendix. A.1. Moreover, in order to detect if the tool is outside the artery walls, one must be able to isolate and track the tool.

Color filtering. By applying colour filtering, it is possible to separate the tool from the artery in the fluoroscopic image. The ability to separate the tool within the fluoroscopic image was made possible by the various debug options available in the software. Figure 4.12 depicts the fluoroscopic image, the left image shows the fluoroscopic image in default mode (a), while the image to the right shows the fluoroscopic image in a "specific debug mode" (b). This test has been developed using this "specific debug mode", since separating the tool from the background was proven to be very difficult in the default fluoroscopic image.

The debug mode made it possible to separate the tool from the arteries by splitting the image into red, green and blue channels (RGB). An example of splitting the image into RGB channels can be seen in Fig. 4.13, where the red channel visualize the isolated arteries, and the blue channel visualizes the isolated tool.

Once the channels are separated, the pixel values of each channel were modified to get unitary pixel values, e.g, the blue channel was modified to only contain pixel values of 55 or 0, and the red channel only to contain pixel values of 200 or 0. By adding the red and blue channels (RB image), with unitary pixel value, the areas of which the tool was outside of the artery walls could be found. This since each pixel getting a value of either 0, 55, 200 or 255, where each value describes areas in

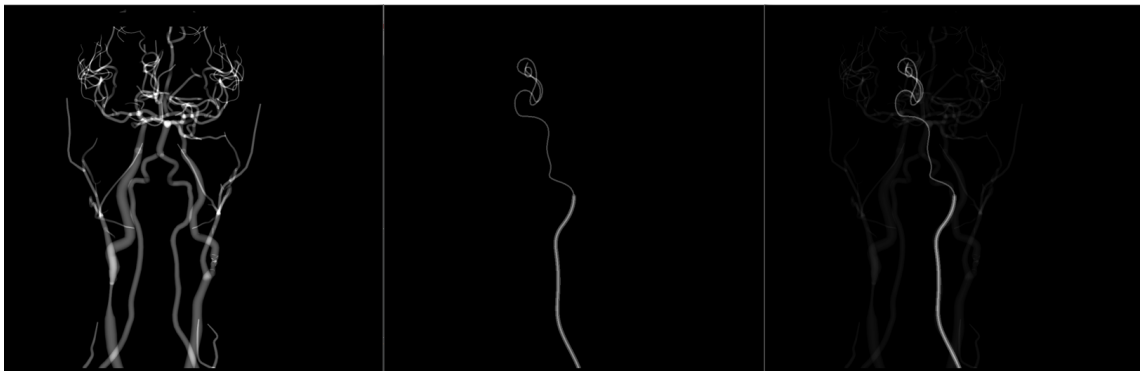


Figure 4.13: Visualisation of RGB channels. Left = red channel, middle = blue channel, right = green channel. Note how information (tool/artery) can be isolated within a colour channel using the "specific debug mode" presented in Fig. 4.12

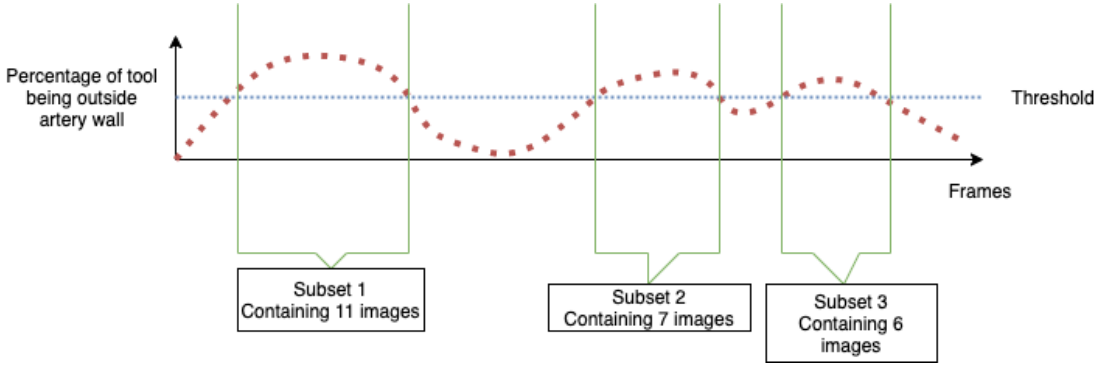


Figure 4.14: Representation of how subsets are collected. Each dot in the red dotted line represents a image in the dataset. Note how the threshold decides when a subset should be obtained.

the image. Pixel values equal to 55 indicated an area in which the tool was outside of the artery wall, meanwhile, pixel values equal to 255 indicated areas of the tool being inside of the artery wall.

The regional information acquired using the colour filtering technique allowed for calculating the percentage of the tool being outside of the artery wall. By counting the number of pixels equal to 55 in the RB image, and dividing that number with the number of pixels equal to 55 in the blue channel, a measure of the percentage of the whole tool being outside the artery wall could be achieved. The method used for colour filtering can be seen in Table 4.3.

By applying a threshold for the percentage of tool outside of an artery, subsets could be created for as long as the percentage of tool outside of an artery was above this threshold. Figure 4.14 depicts this method of creating subsets based on the percentage of the tool being outside of arteries.

SIFT implementation. After gathering the subsets of images where the tool was located outside artery walls, two problems appeared. The first is the redundancy of similar images within each subset and the second problem is high non-similarity within each subset, due to rapid tool movement between frames. The first problem implies that each image within a specific subset looks very similar due to frames

Table 4.3: Iterative representation of color filtering.

Color filtering method

1. Iterate for all fluoroscopic images in the dataset, Z_i = fluoroscopic image
   (a) For Z_i in dataset:
      i. Split Z_i into channels [Red, Green, Blue]
      ii. $R_1 : \forall Red_{pixel} > 0, R_{1pixel} = 200$.
      iii. $B_1 : \forall Blue_{pixel} > 0, B_{1pixel} = 55$.
      iv. Add the red (R_1) and blue (B_1) channels that contain unitary pixel values. $Q = R_1 + B_1$
      v. $Q_c = (\sum_{\forall p \in Q}$ where $Q_p = 55)$, e.g , count pixels where Q = 55.
      vi. Divide $Q_c \div supp(B_1) = \mu$ to get percentage of tool outside.
      vii. If $\mu > \epsilon$, Store Z_i, Where ϵ is preset threshold $0 \leq \epsilon \leq 1$
2. Return subsets of stored images

being captured very frequently. This results in frames displaying more or less the same fluoroscopic image. For the second problem, some subsets contained images that had rapid background changes due to a rapid tool movement, making each of them unique. If only one image is kept from these subsets, relevant locations may be inaccurately discarded.

The main issue with the two problems is that if one removes all redundant images within each subset, images within a subset containing rapid background changes would be lost since all by one image is discarded within each subset. To solve the issue the SIFT algorithm was applied. The features of the algorithm allowed for tracking matching keypoints between two image frames. If the distance between matching keypoints of two frames within a given subset is above a threshold, the "reference" image is seen as unique and therefore stored, while the frame it was compared to is used as a "reference" for further comparison within the subset. The tracking of movement can be seen in Figs. 4.15–4.16, where the matching colours depict the matching keypoints between two frames. The green line illustrates the distance that the key points have travelled between the two frames.

Implementation of tool movement tracking. On some occasions, subsets contained images in which SIFT did not recognize a high enough distance between frames even though there was major tool movement between frames in the subset. Using the method of only storing one image in a redundant subset resulted in frames with high tool movement being discarded. The solution was to track the tool movement between frames within the subset whenever the SIFT algorithm did not recognize any movement.

The first iteration of tool movement tracking used techniques similar to the colour filtering method. By counting the number of pixels in the isolated tool image between frames, a change could be registered if the number of pixels increased

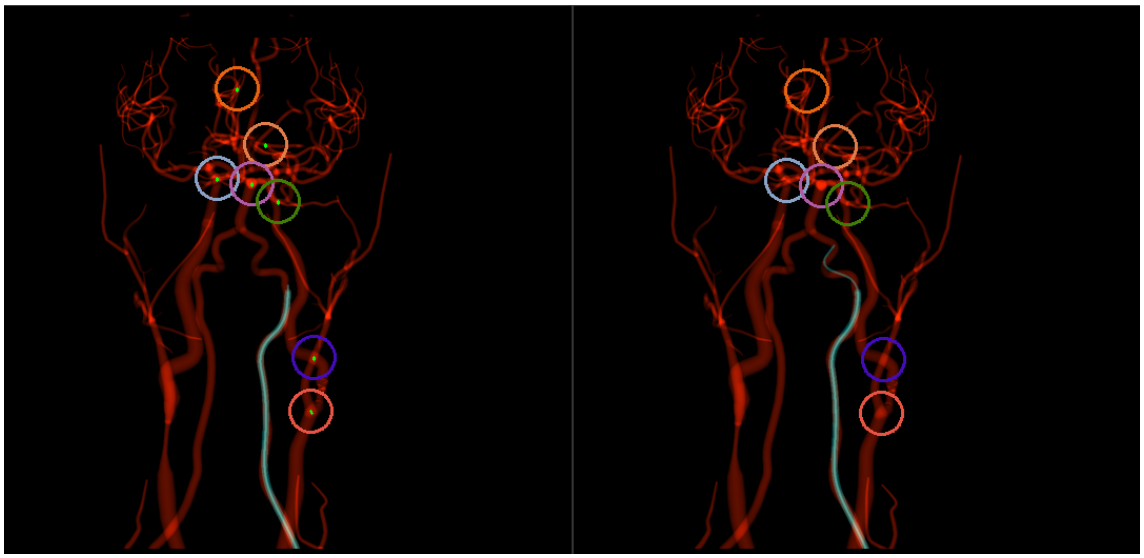


Figure 4.15: Visualization of how keypoints match, where the match is denoted by the color of circles between the two frames. Left image shows frame 1 and the image to the right frame 2. Difference between the two frames can be seen if looked upon the tool movement.

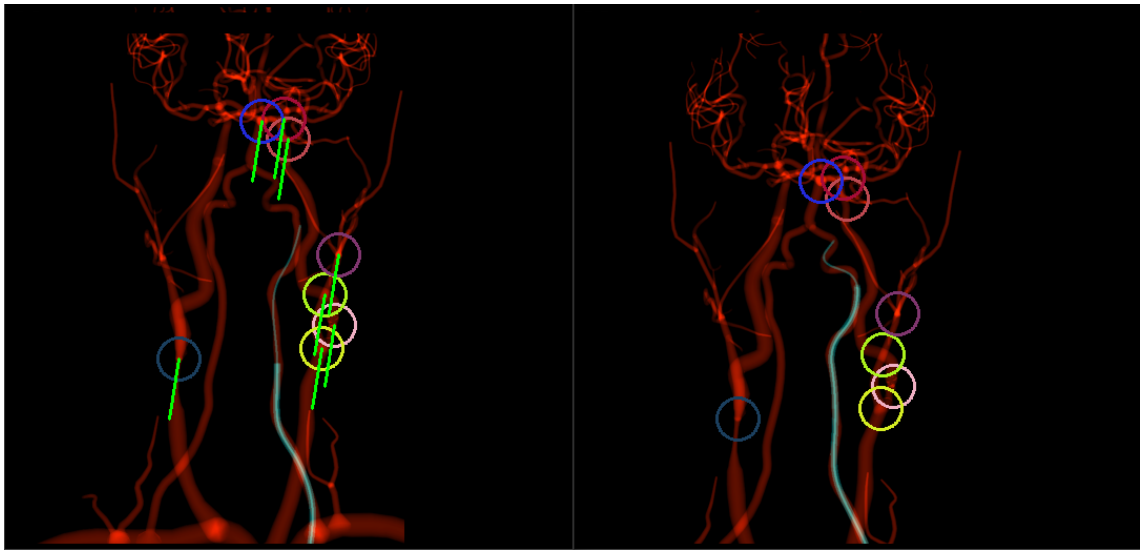


Figure 4.16: Visualization of how keypoints match, where the match is denoted by the color of circles between the two frames. Left image shows frame 1 and the image to the right frame 2. Difference between the two frames can be seen if looked upon the lower portion of the images, where some arteries have moved out of frame.

Table 4.4: Iterative representation of SIFT and tool tracking.

SIFT and tool tracking method

1. Iterate for all fluoroscopic images in each subset of images, S_k = Subset, $I_{(k,j)}$ = image in subset i.
   (A) Parameters:
      i. k = index of subset, j = index of image within subset
      ii. N = #Subset
      iii. η = Distance threshold for SIFT
      iv. λ = Tool area threshold for tool tracking
2. Do for $k = 1, 2, \ldots N$
   (A) For $I_{(k,j)}$ in S_k:
      i. distance = SIFT($I_{(k,j)}$, $I_{(k,j+1)}$)
      ii. if distance $> \eta$:
         a. Return $I_{(k,j)}$
      iii. else:
         a. $F_1 = (\sum_{\forall p \in I_{(k,j)}}$ where $I_{(k,j)} = 55)$, e.g, count pixels in $I_{(k,j)} = 55$
         b. $F_2 = (\sum_{\forall p \in I_{(k,j+1)}}$ where $I_{(k,j+1)} = 55)$, e.g, count pixels in $I_{(k,j+1)} = 55$
         c. if $(F_1 - F_2) > \lambda \rightarrow$ Return $I_{(k,j)}$
3. Store Returned images

or decreased. If the number of tool pixels increased, or decreased, past a certain threshold, the image was seen as unique and thereby stored. The problem with the first iteration of the tool tracking algorithm was that it did not take the number of tools, and their width, into consideration. This resulted in a tool with a smaller width having less to no impact on the tool tracking algorithm while wider tools had a large impact.

For the second iteration, an algorithm of morphological skeletonization was used on the isolated tool image. By morphing the tool into single-pixel width (see Fig. A.4 for explanation), the impact concerning width of the tool was removed, which resulted in a more accurate measurement of the tool movement between image frames. The flow of this process can be seen in Fig. 4.17 and the iterative process can be seen in Table 4.4.

SSIM implementation. The images captured using the colour filtering, grouping, SIFT and tool movement tracking algorithms are saved as a final set of images. As a final step, the SSIM algorithm was used to verify that each image in the final set is unique. This was done by comparing the unique frames, side-by-side, and if the percentage of similarity was above a certain threshold, the image with the most tool area outside of the arteries was stored while the other image was discarded. The whole process from dataset to SSIM can be seen in Fig. 4.17.

Result 1: Test result and physics setting modification. An example

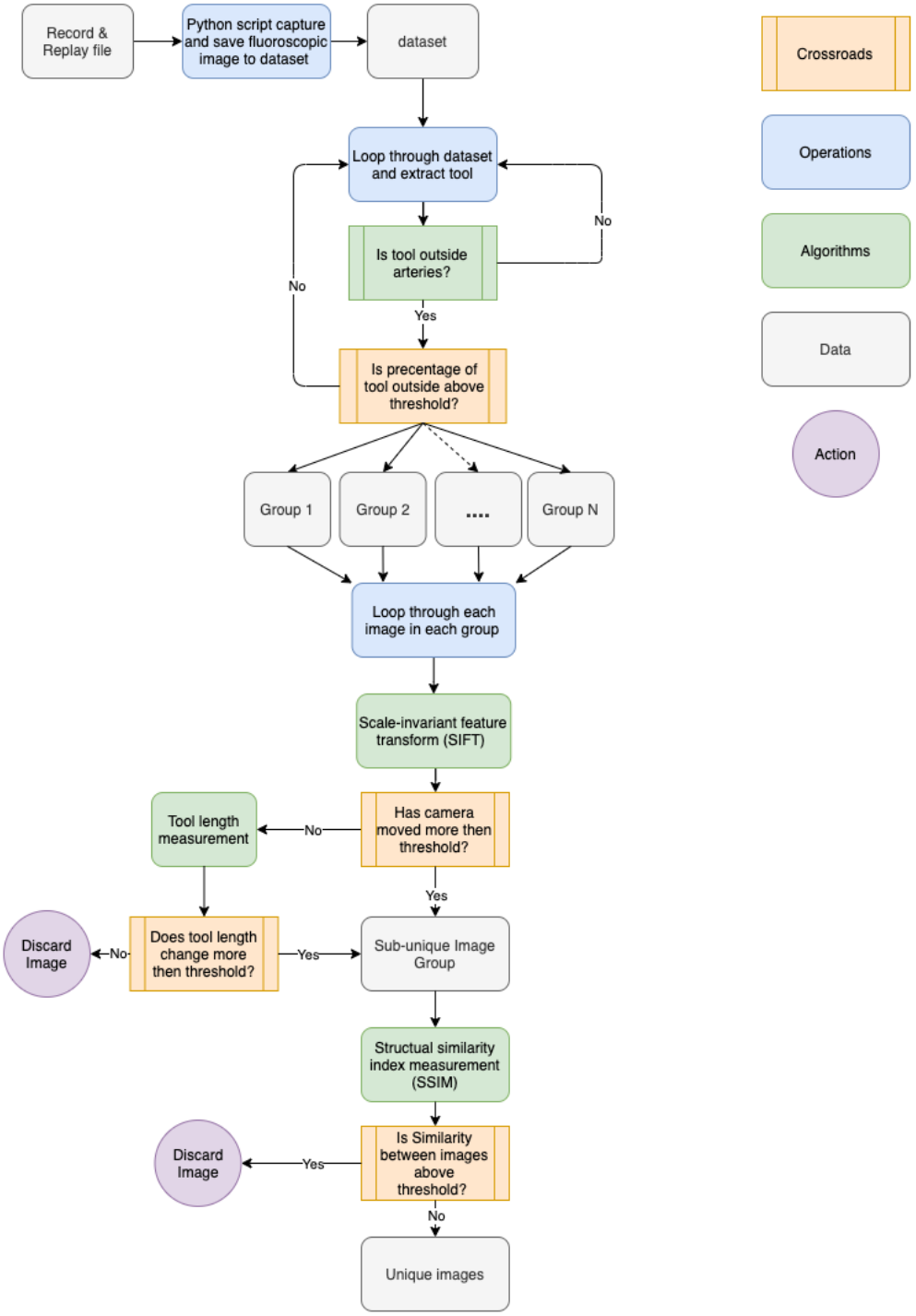


Figure 4.17: Flowchart for test to detect area of tool which is outside artery walls. Note how initial dataset is iterated and frames are put into subgroups (Group 1–N) according to methodology in Fig. 4.14. Furthermore, how SIFT, tool length, and SSIM algorithms are discarding images of high equality compared to the reference image.

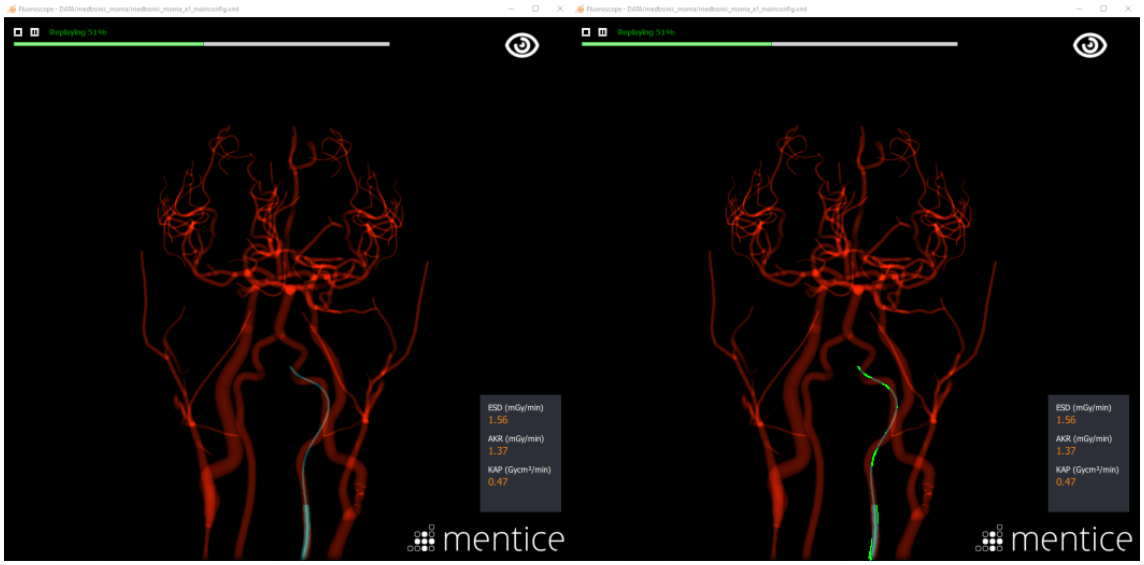


Figure 4.18: Example of a unique image. The left image depicts the original frame. The image to the right depicts the original frame with portions of tool being outside artery wall highlighted in green.

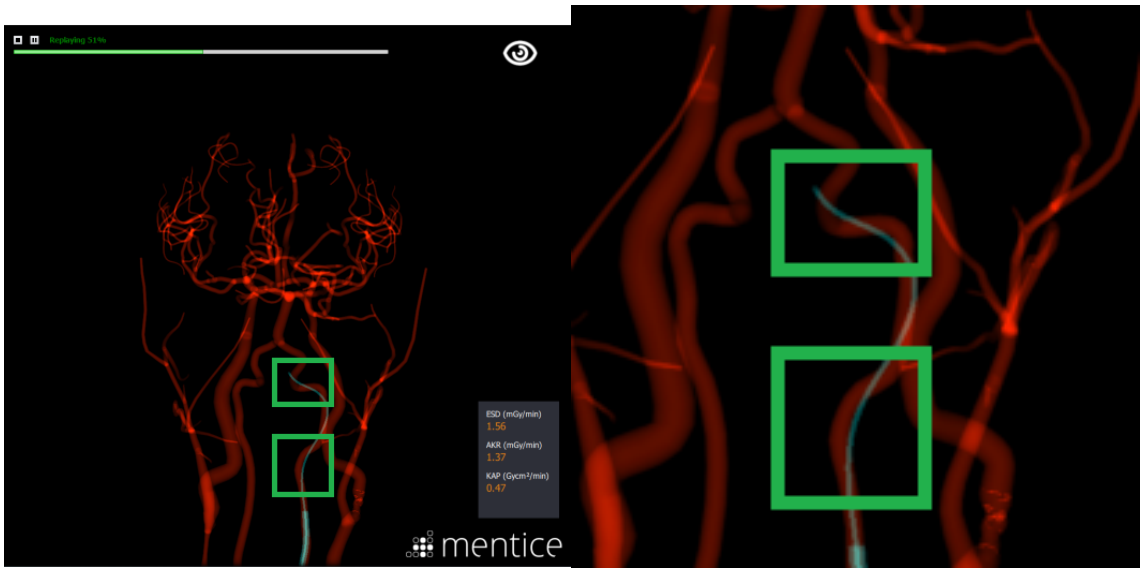
of a unique image can be seen in Fig. 4.18. The image to the left represents the original frame and the image to the right represents the original frame with areas of tool outside of arteries highlighted in green.

By looking closely at the tip of the tool, it is noticeable that the smaller tool is taking a shortcut on the curved artery (see Fig. 4.19). Without the "specific debug mode" toggled on, it would be very difficult for the user to see where the tool takes shortcuts. The possibility of detecting these shortcuts vary between a developer and an experienced interventional radiologist. Where the experienced radiologist may recognize it through the fluoroscopic image using *default mode* due to their well-established understanding of the artery tree (RQ 3b). "Shortcut" behaviour threatens the user experience since the expectations of the tool behaviour is not portrayed accurately in the simulated fluoroscopic image. The behaviour, therefore, needs to be very reliant and realistic in order to meet these user expectations and give an accurate medical simulation procedure.

Simulation physics change & Different hardware. Moreover, two sets of tests were evaluated in order to assess the *reliability* of the tool behaviour. The first set consisted of changing the number of links in the tool (similar to Sect. 4.9). The second was to test the behaviour of the tool between two computers.

Result 1: Simulation physics change. For the first test, changes to the physics of the tool were evaluated. The following settings were used: "default", "short" and "even-shorter", which already existed as predefined settings within the JS code (see Fig. 4.9). The "default" setting is the most common tool physics setting for all of Mentice procedures since these are the settings that reflect the tool behaviour most accurately compared with a real surgical procedure. "Short" and "Even shorter" representing short and more links, striding further away from a realistic tool behaviour and becoming more flexible.

Table 4.5 represents the unique image retrieved from the dataset using the



(a) Default mode **(b)** "Specific debug mode"

Figure 4.19: Magnification of areas that is outside artery walls, the green squares depicts the regions where the tool is outside the walls.

Table 4.5: Resulting break-down of dataset into unique images.

Settings:	Default	"Short"	"Even shorter"
Dataset Images	2269	2107	2104
Unqiue Images	20	16	20

methodology described in Fig. 4.17.

It is important to notice that for "Even shorter" links, the effect on the behaviour of the tool during the replaying procedure is quite significant. As mentioned, using this setting results in the tool having a more flexible type of behaviour and thus does not necessarily travel to the same locations in the endovascular tree. This is true even if the same input was given as for the default settings. The same effects can also be seen on some occasions using the "short" settings. Thus, the resulting unique images when using the setting "short", and "Even shorter", can not be condoned as an improvement for this particular behaviour.

Result 2: Comparison between computers. For the test between two computers, the setup consisted of two computers: *DELL PRECISION M6800* and *DELL PRECISION M6700*. Both computers used the default tool physics setting, the same code branch for the Mentice simulation, the same input (record&replay file) and the same threshold parameters for the Python test script. As described earlier, the difference between the two computers is the *M6800* being more powerful. The computers are here denoted as *DELL67* and *DELL68*.

Note that the number of images in the dataset is different, though this does not affect the evaluation of each image. The capturing of images was set manually and thus the size of the dataset may differ, therefore the ± 100 sign in Table 4.6.

Comparing the procedure between the two computers shows that the *DELL67*

Table 4.6: Resulting break-down of dataset into unique images between *DELL67* and *DELL68* using default tool physic settings.

Computer	*DELL67*	*DELL68*
Dataset Images	2269(±100)	2103(±100)
Unqiue Images	20	18
...	...	...

Figure 4.20: *DELL68*, image with the highest percentage of the tool being outside of arteries. Note how areas outside of the arteries are highlighted in green by the test algorithm.

is able to find 20 unique images while the *DELL68* only found 18 unique images (Table 4.6). This Indicates that *DELL67* computes the tool physics slower than *DELL68*, or that *DELL68* is better at replaying the procedure.

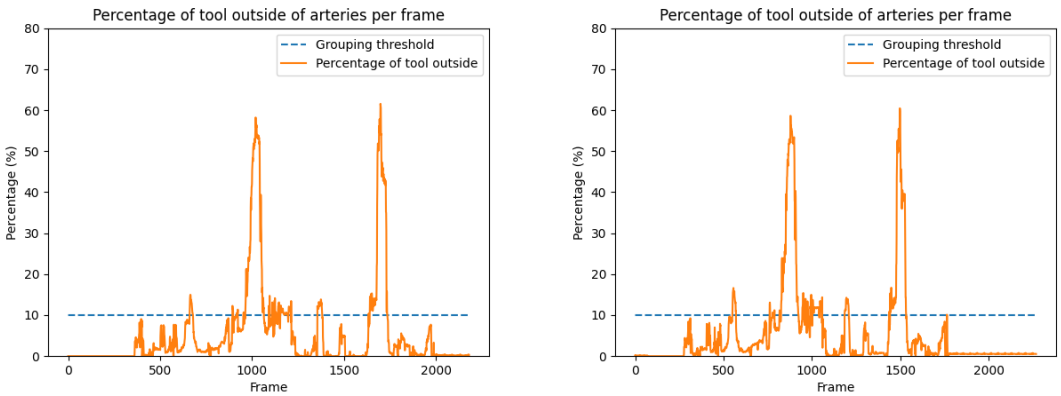
The comparison of the unique images captured by each computer shows that the same location has been discovered by the two computers (besides the two additional images captured by DELL67), see Figs 4.20–4.21. Meaning 18/18 of the images discovered by DELL68 was also discovered by DELL67.

Figure 4.22 depicts the percentage of the tool being outside the the artery walls during the whole procedure. By looking at this graph it is possible to notice regions in which the two graphs are different. Since samples above the threshold are grouped together, the *DELL68* forms more groups from its data compared to the *DELL67*, this explains why the *DELL68* has more unique images than the *DELL67*. Moreover, in addition to the two additional images captured by DELL67, the other images captured between the two computers are identical.

Threshold evaluation. There are four major threshold parameters included in this method that could alter the results. Each threshold gaining its value through trial and error when developing the test. The order of the following threshold depicts the range of impact they have on the outcome of the test. The thresholds are:

1. Percentage of tool outside of artery wall threshold
2. Matching keypoint distance threshold
3. Tool movement distance threshold

Figure 4.21: *DELL67*, image with the highest percentage of the tool being outside. Note how areas outside of the arteries are highlighted in green by the test algorithm.



(a) *DELL68* **(b)** *DELL67*

Figure 4.22: Comparison of percentage of tool outside artery walls during the the replayed procedure. Left figure depicts *DELL68* and the figure to the right show *DELL67*. Note the slightly higher values for the *DELL67* computer.

4. Similarity level (SSIM threshold)

The threshold for percentage of tool being outside of the artery wall is the most important threshold since it is this threshold that determines the degree of accepted inaccurate tool movement/degree of the shortcuts. It also determines the amount of images that will be grouped into subsets and later processed by the succeeding functions. If the threshold is set to low, images where the tool is being recognized as outside may not be of any interest, this since those images will involve very small portions of the tool outside of the artery and thus display irrelevant locations. On the other hand, a too large threshold will miss a lot of interesting images where the tool take intermediate shortcuts or where only the smallest tool take large shortcuts but is small in relation to all tools visible in the fluoroscopic view. A threshold value of 10%, for the whole tool (all tools within the fluoroscopic view), proved sufficient detecting valid inaccuracies within the fluoroscopic view (RQ 2b).

The remaining thresholds do not carry as much impact as the threshold for the tool being outside of artery walls, this since they do not partake in any discoveries of inaccurate tool placement. Their purpose is only to filter out redundant images within the result to provide the developer with locations within the endovascular tree in which the tool behaviour fails. If these thresholds are not set properly, the result will be that the number of redundant images will increase or decrease. Moreover, the best combination of threshold values was discovered to be the following:

1. 10% of the total tool area being outside of artery walls.
2. 50 pixels difference between matched keypoints of two frames
3. 25% movement of tool between two frames
4. Similarity level (SSIM threshold) = 80% similarity between two images

These threshold values are able to detect the most relevant locations where the tool is being outside, without presenting any redundant images. Furthermore, in the ideal case, the graph shown in Fig. 4.22 should be completely flat since if there are any indication of the tool being outside artery walls the consequence in a realistic procedure would mean major or fatal injuries to the patient (RQ 3c). In the simulated case there needs to be an acceptance towards inaccuracies since with the current software and hardware it is not possible to replicate the ideal case. Indeed, the result in Fig. 4.22 does not reflect a normal procedure done by a professional and is only for verification purposes. A normal procedure performed by an experienced radiologist on the Mentice simulation platform may have fewer occasions where the tool is outside of the artery walls. Although, if the percentage of the total tool outside of the artery wall is above 10%, the flaws are clearly visible and thus jeopardizes the user experience.

Note that the two large spikes seen in the Fig. 4.22 reflects actions with the intention of pushing the tool outside of the artery walls and should not occur during a normal procedure. This behaviour is also only possible due to the test being executed on an emulator which does not incorporate force feedback to the user when moving the tool into specific arteries.

4.5 Non-functional requirement: Performance, scalability, and portability

The last test constructed for the automated test framework aims to evaluate performance parameters for the Mentice simulation software. Tests were conducted for two different medical procedures within the simulation, *Procedure A (PA)* and *Procedure B (PB)*, with the latter demanding more CPU power during execution. For each test setup, a record&replay file was used as input with the test script being executed **three** times and the mean being computed for these three runs. Simulation frame rate and physics engine frame rate was measured and compared across different CPU loads and for different hardware setups (*DELL68/DELL67*). A Microsoft program *CPUStres* [48] was used to remove available CPU power from the hardware as described in Sect. **??**. Additionally, "hyperthreading" was evaluated for both procedures on the *DELL67* computer.

As a golden standard for this particular test, a "soft" threshold was set related to how the simulation was perceived by the user during testing, i.e., how the fluoroscopic image and simulation menus were behaving. This in contrast to using a "hard" frame rate threshold for what is an acceptable frame rate for securing user experience (RQ 2*b*).

4.5.1 Result 1: Simulation frame rate

The first test evaluated simulation frame rate for different removals of CPU power and across different hardware. Figure 4.23–4.26 depicts the result when running *PA* on the *DELL68* computer and removing *no, low, medium*, and *high* (0%, 25%, 50%, 75%), power from the CPU. As depicted by these figures, the mean frame rate of the simulation drops as more CPU power is removed. Additionally, variance of the frame rate increase as more CPU power is removed. As can be seen in Fig. 4.26, there are less samples from the simulation frame rate. This is due to the simulation crashing when not sufficient CPU power is available for the simulation.

Figure 4.23–4.26 suggests that, for the procedure *PA*, the simulation runs without any noticeable drop in frame rate and with reasonable variance for *no* and *low*/25% CPU power removal. For higher CPU power removals, i.e., 50–75%, the variance increase to a level which has an impact on the *performance* of the simulation. Although the frame rate graphs can not determine the quality of the simulation in isolation, by inspecting the simulation, and looking for noticeable frame drops, it was concluded that higher CPU removals (50–75%) had an adverse effect on simulation *performance* and were not considered to deliver the user experience seen for *no* or 25% CPU power removal.

Furthermore, since there is no consensus regarding what mean frame rate and frame rate variance that is acceptable for keeping high user experience through the simulation, this inspection of the simulation while removing CPU power suggest that the Mentice simulation software keeps a sufficient user experience with at least 75% (25% removal) of the CPU available for *PA*. This implies that for the *scalability* of *PA*, the procedure could be scaled up in complexity without having an adverse effect on user experience when being executed on the *DELL68* computer.

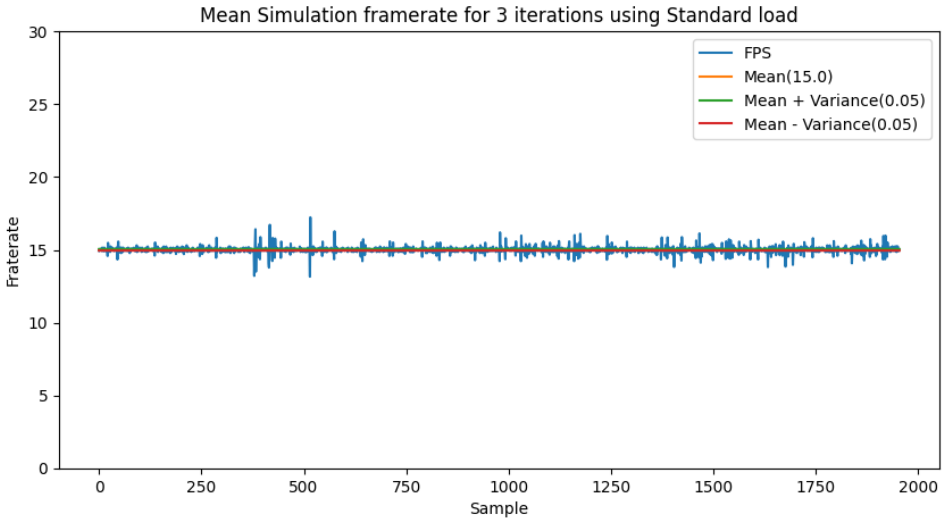


Figure 4.23: Simulation frame rate of *PA* with *no*/0% of CPU power removed (*DELL68*). As depicted, the variance is very low for with *no* load applied to the computer CPU.

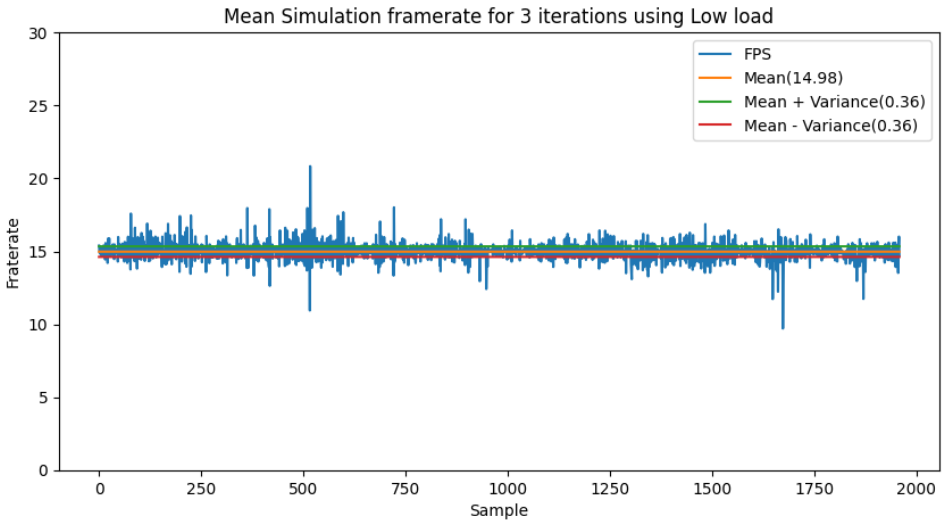


Figure 4.24: Simulation frame rate of *PA* with *low*/25% of CPU power removed (*DELL68*). Note how the variance has increased compared to Fig. 4.23.

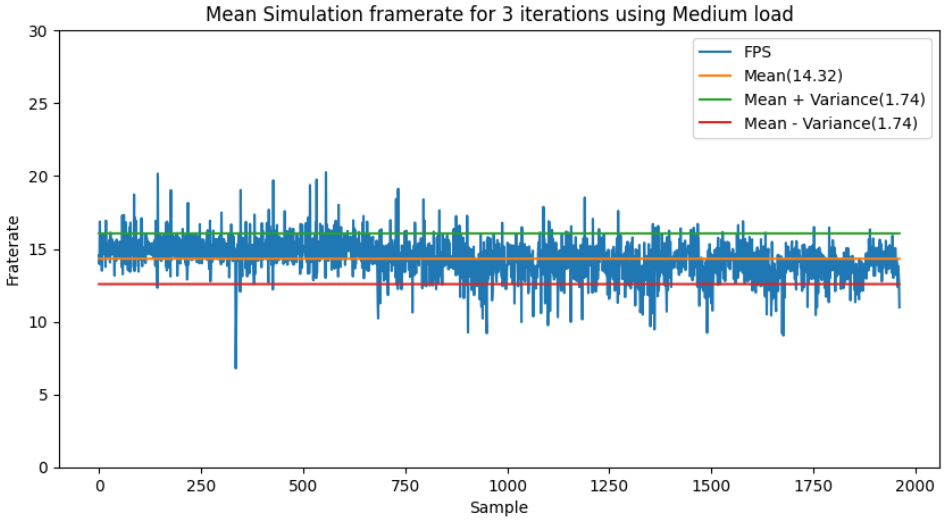


Figure 4.25: Simulation frame rate of *PA* with *medium*/50% of CPU power removed (*DELL68*). Similar to Fig. 4.24, a higher variance. The simulation frame rate also starts to decrease.

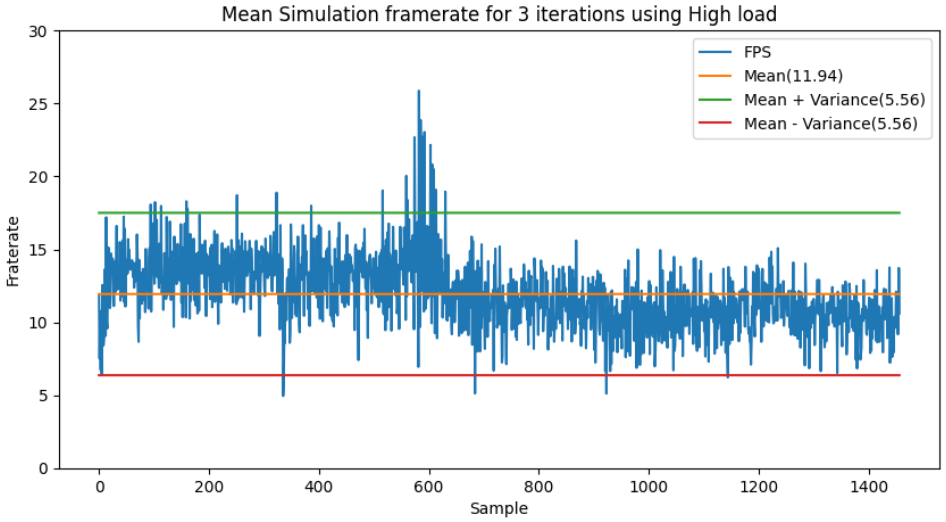
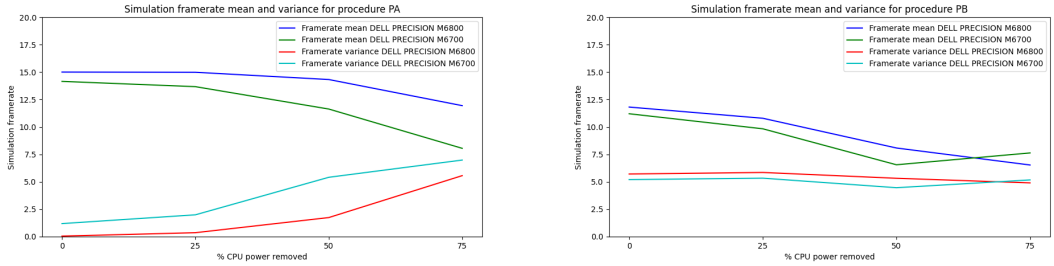


Figure 4.26: Simulation frame rate of *PA* with *high*/75% of CPU power removed (*DELL68*). Similar to Fig. 4.25, a higher variance is noted. Additionally, note how the sampling stops at $\approx$ 1400 samples due to the simulation crashing.



(a) PA **(b)** PB

Figure 4.27: Comparison between *DELL68/67* for *PA* and *PB* for increasing CPU power removal. Note how the behaviour across computers during increased CPU removal is different between procedures.

Simulation frame rate across hardware. Test result similar to Figs. 4.23–4.26 was obtained for different hardware and for different procedures (*PA/PB*). Figure 4.27 depicts the mean simulation frame rate and variance for both hardware setups when removing CPU power, in steps of 25%, as described earlier and in **Iteration 3**.

As depicted by Fig. 4.27, for *PA* the *DELL68* computer has a higher mean frame rate (measured over three iterations) and a lower frame rate variance compared to the *DELL67* computer. For this computer, as more CPU power is removed, the simulation frame rate drops and the variance increases, as depicted earlier by Figs. 4.23–4.26. The same behavior is true for the *DELL67* computer, as seen in Fig. 4.27 (PA), with a higher simulation frame rate drop and a higher variance for increased CPU power removal. This result suggest that for *no* and low/25% CPU removal, the *portability* of *PA* between the two computers is sufficient but for higher CPU removals the *DELL67* computer fails to maintain sufficient frame rate and low frame rate variance.

A more interesting result appears for *PB* in Fig. 4.27. Since this procedure is more computationally heavy than *PA* the mean frame rate is expected to be lower for both computers. As can be seen in the figure, the frame rate is lower when no load is applied, 11–12 compared to previous 14–15, but the variance is high (≈ 5) across all CPU power removals. Another interesting note is that the *DELL67* computer has a higher simulation frame rate for the highest (75%) CPU power removal which was not the result for *PA*. This result implies that for *PB*, the *portability* for the procedure is high across all CPU power removals although the *performance* of the procedure is of low quality for $> 25\%$ power removals for both hardware. See Appendix. A.2.1 for all tests conducted for procedure *PA* and *PB* on both computers (*DELL67/68*).

4.5.2 Result 2: Physics engine frame rate

Physics engine frame rate, the frame rate at which the C++ physics engine computes physics features within the simulation, was also evaluated to see whether the frame rate of these computations was affected when removing CPU power. The idea was

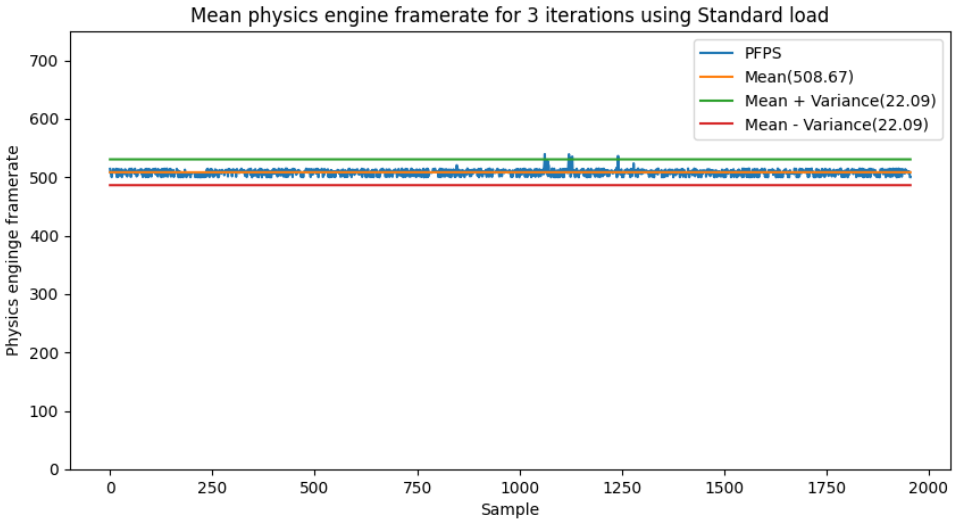


Figure 4.28: Physics engine frame rate with *no* CPU power removal for *PA* on the *DELL68* computer. Note how the frame rate is ≈ 500 for the physics engine.

that this could also be a measure of user experience and used for drawing conclusions about *performance, scalability*, and *portability* of the software.

Figure 4.28 and 4.29 depicts the result for this test performed for *PA*. This test was performed similarly to the previous test regarding simulation frame rate, measuring mean (physics engine) frame rate and variance. As depicted by Fig. 4.29, the physics engine frame rate appeared unaffected by the CPU power removal with mean frame rate and frame rate variance being quite constant. This result was surprising and might be due to the Mentice simulation prioritizing these computations and that the physics engine runs in parallel, and much faster, compared to the other steps performed during a simulation frame (see Sect. 2.1.2).

For *PB*, the more computationally demanding procedure, the test showed a lower overall mean physics engine frame rate and with more variance as well as with multiple outliers which had an adverse effect on the variance of the measurement. Similar to *PA*, this procedure showed no decreasing frame rate when increasing CPU power removed. The reason behind the outliers in Fig. 4.30 was not fully understood. One idea was that the physics engine frame rate spiked or dropped drastically when the user went into the simulation software menus and applied/ removed options within the simulation. Another idea was that this could be due to the tool being still but when compared to the previous result, Fig. 4.23, this idea proved dissatisfying since this procedure also included time intervals in which the tool was not being moved.

Furthermore, *PB*, similar to *PA*, showed no significant difference between the two hardware setups. This result made it implausible to draw any conclusions regarding the *scalability* and *portability* of the software since results were almost identical across different CPU power removals and for different hardware. For graphs displaying more of this result, see Appendix A.2.

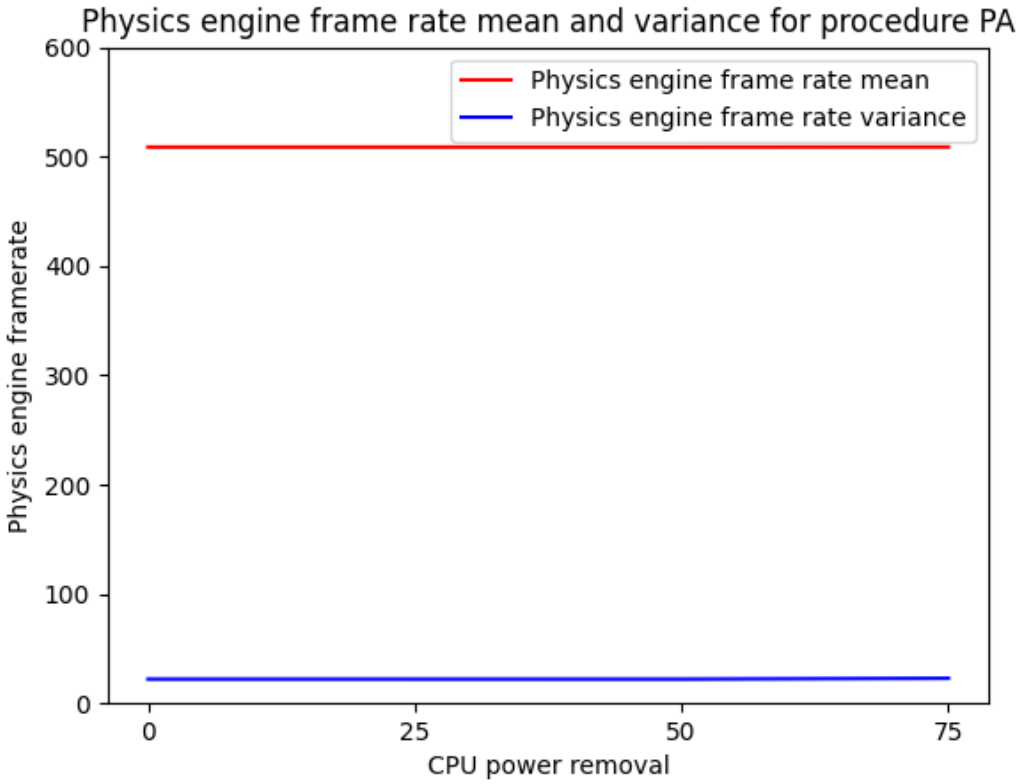


Figure 4.29: Physics engine frame rate with increasing CPU power removal for *PA* on the *DELL68* computer. Note how the frame rate mean and variance is unchanged during increased CPU removal (same as in Fig 4.28).

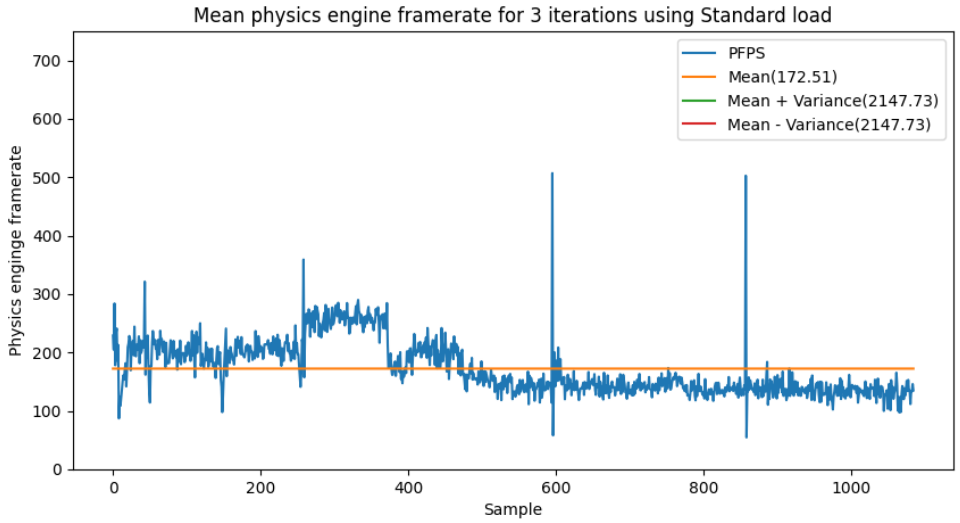


Figure 4.30: Physics engine frame rate with *no* CPU power removal for *PB* on the *DELL68* computer. Note how outliers in data produce a high variance compared to Fig. 4.28. For visibility of outliers, variance lines are not visible.

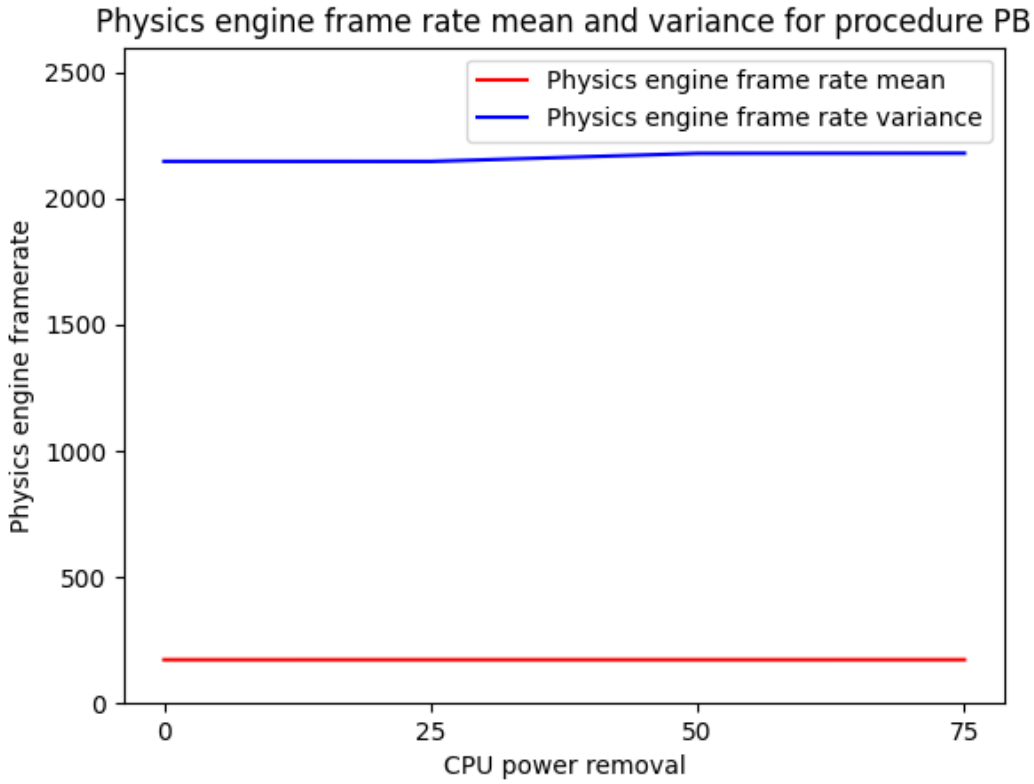


Figure 4.31: Physics engine frame rate with increasing CPU power removal for *PB* on the *DELL68* computer. Note how the frame rate, although stable across CPU removal, is less than half compared to Fig. 4.29. Also, variance is much higher due to outliers. The same stability as seen for the mean frame rate and variance in Fig 4.29 is seen across all load for this procedure as well, i.e., same values as in Fig 4.30 for all loads.

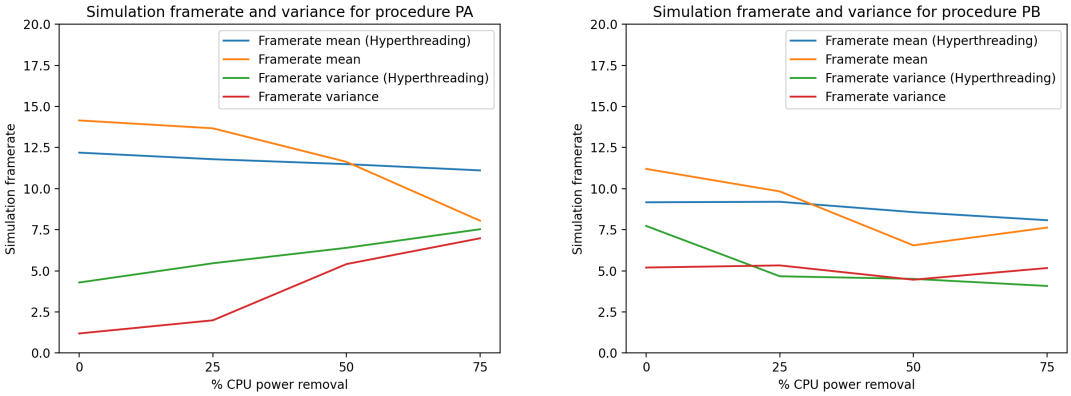
4.5.3 Result 3: Hyperthreading M6700

In the last result for this test, the "hyperthreading" setting was evaluated on the *DELL67* computer. It was suggested that for optimal simulation *performance*, "hyperthreading" should be turned off in the hardware. Test similar to the previous ones were conducted with and without "hyperthreading" being active on the hardware. Figure 4.32 depicts the test result when increasing CPU power removal and having "hyperthreading" active and not active for both procedures (*PA* and *PB*).

For *no* and 25% CPU power removal, the result complies with what was suggested. The simulation *performance* is higher when "hyperthreading" is not active for these CPU removals. This is true for both procedures *PA* and *PB*. A more interesting result is that, as CPU power removal increases, the simulation frame rate stays higher with "hyperthreading" active. This result is true for both procedures investigated. Although frame rate stays higher during increased CPU power removal for both procedures, frame rate variance is different across procedures. As depicted by the figure, *PA* results in higher frame rate variance for increased CPU power removal. Additionally, frame rate variance measures across both test cases for this procedure seemed to converge towards higher percentage removals, though at no CPU power removal, the variance is much lower for the result with no "hyperthreading" active.

Furthermore, for *PB*, the same behaviour is not displayed for the frame rate variance. Similar to *PA*, the variance is higher for *no* CPU power removal but as more CPU power is removed, the variance decreases with "hyperthreading" active while it stays the same for no "hyperthreading".

Lastly, this result suggests that for computers with less CPU power available, activating "hyperthreading" could lead to improved simulation frame rate and *performance* (as seen for both procedures). Although, it is also important to note that the frame rate variance develops differently between these two procedures and thus there is no conclusion whether frame rate variance decrease or increase for higher CPU power removals when having "hyperthreading" activated.



(a) PA **(b)** PB

Figure 4.32: Simulation frame rate mean and variance for *PA* and *PB* using the *DELL67* computer with and without "hyperthreading". Note how the behaviour displayed in *PA* is not similar to the one in *PB*.

5
Discussion

This chapter aims to discuss NFRs and the characterization of such. It will also discuss results presented in Chap.4, what these results mean, why they matter and what conclusions can, and can not, be drawn from those results.

5.1 Non-functional requirements

The NFRs evaluated in this thesis are *reliability, performance, scalability*, and *portability*. As can be seen in the result, no *efficiency* measurement or evaluation was considered. There could have been multiple additional NFRs to evaluate for the specific product (the Mentice simulation software), but these mainly serve as an example of tests connected to NFRs which can be evaluated automatically. Due to the elusiveness of NFRs, these requirements are often picked apart in order to find a parameter that is measurable by a computer script. This is in contrast to manual testing by a human, in which "softer" measurement parameters such as the "look-and-feel" of the software, could be evaluated during testing. The importance of separating NFRs and FRs from each other has to do with their different coverage of the software. FRs are often specific and aims to evaluate specific functionality, while NFRs has more connection to the overlaying software structure, feel, and performance. In this thesis, NFRs evaluated were linked to user experience to evaluate requirements of the software which had an impact on how the user perceived the software.

Needless to say, there are many additional NFRs that can be evaluated in terms of user experience. There are also multiple NFRs connected to the development phase of the software such as *Modifiability, Testability, Extensibility, etc.*, which all are possible to test automatically though through a different automated testing framework and with a different focus compared to this thesis.

5.1.1 Classification

Classification of requirements, into groups of FRs and NFRs, within this thesis was mainly based on if the requirement stated *what the software should do* or *how the software should be*. This classification was used due to its simplicity in separating FRs from NFRs. There are arguable more ways to perform this classification, one of which could be to separate *quality* attributes (NFRs) from *technical constraints* (FRs). Although there are more ways in performing these characterizations, the value is found in the actual separation and not the method used to do it. By

separating FRs from NFRs, it is possible to analyze, prioritize, and specify different kinds of requirements in an effective way. It is also possible to efficiently find the correct test for a specific requirement through this separation.

Furthermore, by separating FRs from NFRs, different values of the software are brought to light during software development. As mentioned in Sect. 2.3, historically, FRs has been the area in which most emphasis has been placed during software development. Although the functionality of the software is critical, this emphasis comes with the risk of overlooking the architectural requirements of the software and the "look-and-feel" requirements of the software.

5.2 NFR taxonomy

There are multiple ways of building a taxonomy for automated testing of NFRs. The taxonomy constructed in this thesis aims to be simple to use by the developer or project manager during software development. As mentioned earlier in Sect.2.3, since there is a lack of a common ground for classification, prioritization, and analysis of NFRs, this taxonomy aims to help software developers within the Medtech industry to better understand how different requirements fit into their testing suite and what tests that should be performed.

5.2.1 Filters

By applying filters, specific to the software and the aspect that will be tested, the idea is to propose an easy path from requirement identification to testable scenarios. These filters aim to reduce noise during requirement classification by prioritizing NFRs which have the largest impact on the aspect that will be tested. This will provide the developer or project manager with a better overview of their requirements during software testing and help build tests with a clear focus. What these filters do not include is layers in which regulatory and compliance parameters are taken into consideration. These parameters are of paramount importance during software development in the Medtech industry but do not fit into this thesis' taxonomy in which user experience is the aspect being evaluated.

5.2.2 Drawbacks of taxonomy

As mentioned in the previous section, regulatory and compliance requirements are not built into this taxonomy. For software development in the Medtech industry, regulatory requirements are important and arguably as important as for hardware development within the industry. In the taxonomy proposed in this thesis, regulatory requirements are not taken into consideration since the aspect under evaluation mainly considered user experience, and more specifically *how closely the simulation mimicked reality*. Additionally, the product being evaluated in this thesis does not possess a medical classification which implies that there are no regulatory requirements based on medical software. Due to this classification and the focus on user experience, no regulatory requirements were considered.

Furthermore, it is arguable that regulatory requirements have an effect on user experience as the user might feel bad, or unsafe when using software that lacks proper testing of regulatory requirements. Because of this, the regulatory requirements could be integrated as an additional filter for the taxonomy if the product subject to evaluation had been of medical classification.

5.2.3 User experience

As mentioned in the previous sections, the aspect being focused on within the taxonomy and throughout tests developed in this thesis focus on user experience. The idea is to prioritize all NFRs applicable to the Medtech software based on their relevance on evaluating user experience connected to the link between simulated medical procedures and real-life medical procedures. Through automated tests, the simulated medical procedure could be evaluated with respect to this link and the software could be brought closer to a realistic scenario thus increasing the user experience and the value of the training for the interventional radiologist.

Furthermore, it is important to note the elusiveness of the term user experience and its fuzziness as a measurement. There is no general consensus regarding what a good user experience is, this since it is highly affected by the actual user evaluating the experience when using the software and what that user prioritizes the most. Due to these drawbacks, the emphasis of user experience within the taxonomy, evaluated as the link between simulated procedures and real-life procedures, aims to bring the simulated software as close to reality as possible through automated tests that evaluate parameters which can be different between simulated, and real-life, procedures.

5.3 An automated test framework

For evaluating NFRs brought to light throughout the taxonomy filtering process, an automated test framework had to be developed. The automated framework is based on a set of key factors which are of importance when developing automated tests. These factors include knowledge of the software, of the medical procedure, how this knowledge could be integrated into the tests, and what the user looks at, i.e., what the test should look at during evaluation.

This framework included a few components that were critical for executing test cases. One component was the record&replay functionality of the software. This ensured that input could be given automatically to the software and that the same input could be given for each iteration of the test as well as for different hardware/ computer setups. Another cornerstone was knowledge of the specific procedure being tested or general knowledge about the Mentice software. This knowledge could then be used for building evaluating scrips in Python which assessed parameters connected to user experience which could be measured within the simulation script or within the fluoroscopic image of the simulation.

5.3.1 Record&replay function

One of the components of the automated test framework is also one of its drawback, i.e., the record&replay functionality. Although the record&replay functionality is good for making sure that equivalent input is given for the test, this functionality is not flexible enough to be considered robust when used across different code branches, hardware setups and software settings. This implies that a lot of manual labour still has to be made in order to construct these record&replay files which often only are sufficiently robust for a specific code branch, software setting or hardware setup. An improvement, or consideration, regarding the input to the simulation is further discussed in Sect. 6.

Furthermore, although these drawbacks are of importance, the record&replay functionality still proved to be sufficient for the tests constructed within this thesis. Its drawbacks were noted in a few test scenarios (see Sects. 4.4.1–4.4.2) but overall the record&replay functionality proved to be robust enough to provide a reliable comparison between test results for different computers and code branches for specific medical procedures within the Mentice simulation. It is important to note that the difference between these code branches and computers was not that large, and due to this, that might have been the case why it worked out well.

5.3.2 Non-deterministic behaviour

Another Mentice simulation aspect that displayed its impact on the result throughout the testing phase was the non-deterministic behaviour of the software. This randomness, partly due to randomly generated functionality within the software, and partly due to hardware acting differently depending on the temperature at which they operate as well as the impact of external programs on the hardware, displayed its impact through minor changes between test iterations. This aspect, the non-deterministic behaviour, is something that is difficult to remove from the testing phase and tests have to be robust enough to handle these difficulties.

5.3.3 Emulator

In this thesis, an emulator of the Mentice simulation hardware was used during testing. This choice was necessary for constructing automatic test input using the record&replay functionality of the software. Although this emulator acts similar to the actual product delivered by Mentice, some drawbacks (also mentioned in Sect. 2.1.1) has to be taken into consideration. Firstly, since the emulator is solely based on software, no defects related to the connection between hardware vs. software will be detected throughout the tests. Additionally, since there is no force-feedback given to the user while creating the record&replay files, some tool placements within the virtual patient would not have been possible using the complete Mentice product. This drawback is mainly noted in Test 2 (Sect. 4.4.2), in which the tools were placed far into small arteries on some occasions, which would simply not be possible when using the actual product.

5.4 Automatic tests

A few automatic tests were presented in Sect. 4. These tests display how certain NFRs could be addressed within the Mentice simulation software or for software testing in general. In the case of this thesis, no major prior tests were constructed for the software and thus these examples stand both as examples for how Mentice can test their software automatically and how software testing, in general, can be executed on Medtech software. As mentioned earlier, tests focused on user experience and thus did not cover the majority of possible aspects that could be tested through NFR testing.

5.4.1 Test for tool stability

This specific test evaluated the stability of medical tools for specific procedures within the Mentice simulation. The test evaluated tool stability and gave a degree of instability for each tool at a specific artery within the virtual patient. This to provide developers and project managers with insight regarding the *reliability* of their software throughout development. Since tools, in real-life endovascular procedures, are stable and do only move slightly due to blood flow and muscle/organ movement, it is important that the Mentice simulation also possess this behaviour. The test was only developed for assessing medical catheters and wires within the virtual patient and did not assess whether other tools, such as medical balloons, stents etc., were stable throughout the simulation run. Further suggestions for tool stability measurement are discussed in Sect. 6.3.

Furthermore, although the test proves sufficient at detecting instability of the medical catheters and wires, it still requires a developer, or project manager, to look at the specific locations and assess whether this instability is problematic or not for delivering a good medical simulation product.

5.4.2 Test for detecting tool outside artery walls

Similar to Sect. 5.4.1 the behaviour of the tool is a vital part to replicate in a simulated procedure. Where the demand to match a realistic procedure is high. With this test, it is possible to locate positions in a procedure where the tool is outside the artery walls. Since this type of behaviour is highly unwanted and unexpected in a realistic situation. If there were to be a penetration of the artery walls in a realistic situation, the action is with intention or a mistake, and not due to faulty projection of the tool in a fluoroscopic image. The test provides the developers and project managers to locate positions in the virtual patients where this type of inaccurate tool movement may occur, improving the process of tweaking the code to achieve a better user experience.

Furthermore, one drawback of this test was that it only “measured” tool outside of arteries in a two-dimensional view. This drawback is not of great significance when looking at a two-dimensional view in which the arteries do not overlap. However, if the C-arm (including the fluoroscopic camera) within the simulation is rotated, i.e., the artery tree is rotated, some arteries may overlap each other with

one artery in the "foreground" of the fluoroscopic image and another one in the "background". In this scenario, the colour filtering method used in this test, to find parts of the tool outside of arteries, will not be able to identify inaccurate tool placement within the simulation.

5.4.3 Performance, Scalability, Portability, Efficiency

This test aimed to evaluate multiple NFRs of the Mentice simulation software. The result suggested that the Mentice simulation software proved efficient at executing different medical procedures across different hardware. Different procedures displayed different performance parameters and the *scalability* of the procedure was dependent on the hardware/computer on which it was executed on.

These tests are important to make sure that the Mentice simulation software maintains high performance, i.e., frame rate, thus deliver a good user experience for the end-user. They also provide insight into how "scalable" a specific medical procedure is and if additional functionality can be applied without jeopardizing the user experience of the medical procedure.

One drawback of this test is that it only evaluates frame rate as the main parameter for user experience. Although frame rate is a decent measure of what the user perceives, it does not fully cover the actual "look-and-feel" of the simulation. Suggestions for further improvement on this NFR test can be seen in Sect. 6.3.

6
Future work

Although this thesis provides a starting point for automated testing within the Medtech industry, further work has to be made to achieve better coverage, robustness and automatization of software tests. Suggested improvements of each result presented in Chap. 4 will be covered in this section.

6.1 Improvement of taxonomy

As mentioned earlier, an improvement of the taxonomy developed in this thesis would be to incorporate regulatory and compliance requirements into the taxonomy. This additional layer would make sure that the testing of Medtech software address these requirements and comply with the necessary medical device standards. Another improvement would be to incorporate a layer in the taxonomy which separates applicable NFRs into buckets depending on what aspect they focus on. This would make sure that the software developers, and project managers, developing these tests, has a clear focus and aim for each specific test.

6.2 Improvement of test framework

Due to the drawback of the record & replay functionality within the Mentice simulation, a future improvement might be to implement randomly generated input for their testing scenarios [7, 8]. This would enable the evaluation of more possible user inputs (automatically) and find software defects that do not commonly occur during "normal" user input. Another way of incorporating test input for the Mentice simulation would be to apply machine learning algorithms. Machine learning could be used to train an agent for multiple medical procedures and then use this "trained" agent as input for a testing suite. It is suggested that this machine learning is performed through reinforcement learning, i.e., letting the algorithm itself detect how to reach its goals within the medical procedure by implementing a reward-structure based on steps within a specific medical procedure. This would, in our view, reflect the closest representation of how a human would learn a medical procedure through "trial-and-error".

Furthermore, it is important to note that the use of machine learning algorithms always comes with the risk of overfitting the input data and overlooking other valuable user inputs, which the agent did not detect when being trained for a specific medical procedure. Although this risk is important, there is still a good reason to implement machine learning models as a complement for building test input.

This with the argument that it could reduce the manual labour necessary for testing the software and increase the test coverage by automatically finding valuable test inputs for different medical procedures. As a suggestion, reinforcement learning can be used together with software tests during the learning process to detect non-trivial software defects that might be overlooked during manual testing, which were also the case for randomly generated test input.

6.3 Improvement of tests

Test of tool stability. As mentioned earlier in Sect. 5.4.1, an improvement of the test for detecting instability of the medical catheters and wires within the simulation would be to expand this test to also include other medical tools.

Stability measures of additional tools such as medical balloons, stents, etc„ could be evaluated in a similar fashion by measuring tool coordinates over time within the simulation and assess whether these are shaking or altering their structure (physical form).

Detection of tool outside artery walls. The test discussed in Sect. 5.4.2 would be possible to improve in multiple ways. Firstly, instead of only applying image analysis of the fluoroscopic image, the test could be extended through measurements within the Mentice simulation JS code. Tool coordinates along the tool and artery coordinates could be grabbed from the simulation and used for analysis regarding if the tool is outside of an artery or not. Though, this improvement would only be slight due to how the tools are constructed within the simulation. Since tools are built upon links and collision points, collision points indicate at which the tool coordinates can be measured and compared to the artery coordinates. This means that the link, in-between two collision points, is not possible to measure directly and thus it is possible that this part of the tool will be outside of an artery. In order to compute whether this link, between collision points, is outside, excessive computations would have to be made while the Mentice simulation is running thus affecting the simulation and test result significantly. Even considering these drawbacks, the test could be improved for identifying tool areas outside of arteries especially when arteries are overlapping within the fluoroscopic view (as discussed in Sect. 5.4.2).

Performance, scalability, and portability. As an improvement for the test discussed in Sect. 5.4.3, additional measures which better reflects the user experience could be incorporated into the test. One could incorporate image analysis and analyse whether the fluoroscopic image is lagging during higher performance testing. This additional measure, together with the frame rate measure, would give a better overall measurement of the user experience of the medical procedure and comply more accurately with what the user perceives during the simulation.

6.4 Other

Additional improvements which were not explored in this thesis would be to construct a fully automated test framework that can be executed automatically once every night or every time the master branch of the software is updated. This function-

ality could be implemented by using cloud applications such as Microsoft Azure [49]. Although this would enable fully automatic testing of the Medtech software, it is important to note the various drawbacks connected to the testing framework and the use of the emulator (Sects. 5.3.1–5.3.3).

7
Conclusion

Non-functional requirements (NFRs) are often overlooked during software development and the importance of identifying and classifying those within requirements engineering is paid little attention to. This leads to the architecture and properties of the software being left out during the testing phase of the software. The reason may be that NFRs are difficult to comprehend and apply to the software testing suite. Research points out that if NFRs are not taken into consideration early on in development, the problems often remain and needs to be fixed later on during software development (i.e., technical debt).

In applications such as simulated surgical training the demand for an accurate user experience is essential to provide a realistic training scenario that closely mimics real-life endovascular surgical procedures. The user experience could often be measured or explained using an NFR or a combination of NFRs, thus if NFRs are taken more into account during software development, the companies delivering those products will be able to comply with the expectations of the end-user. Furthermore, the easier it is to associate a certain NFR with the software, the easier it becomes to develop tests around it. If more tests are developed around a certain NFR, the easier it becomes to measure these requirements and thus set the level of user experience that is expected.

This thesis aimed to provide the reader with a taxonomy for NFR identification and classification within Medtech software development. In this thesis, a set of NFRs were identified and classified based on common bugs or defects of the Mentice endovascular surgery training platform. The aim was to find NFRs connected to the user experience of the platform and to develop an automated test framework for those. This framework included tests that were meant to evaluate those particular NFRs with respect to user experience. As a simulation-based training platform for interventional radiologists, the simulation software needs to provide a quality user experience that closely mimics realistic surgical scenarios, therefor user experience was set as the main objective for each test within the framework.

The NFRs selected after using the taxonomy on the Mentice software platform were *reliability, performance, portability*, and *scalability*. These requirements were picked since if these were not met, the user experience of the platform would be impaired.

Tests that evaluated the *reliability* of the platform mainly surrounded medical tool movement within the simulation. These tests were a focus since Mentice developers and collaborating Mentice interventional radiologists had noticed some inaccurate behaviour of the tools within the simulation, more particular, unstable tools and tools which were traversing outside of arteries within the virtual patient.

Since the tool behaviour is one of the more vital parts of the simulation, tests were built in order to detect and notify the Mentice developers and project managers about these inaccurate behaviours. If these inaccurate tool behaviours were not detected and treated, the user experience of the platform would be damaged resulting in decreased value for surgical simulation training.

Additionally, inaccurate behaviour of the simulation, in relation to reality, would also result in deficient training and, thus, provide less skilled interventional radiologists. It was displayed, through these tests, that inaccurate tool behaviour is possible to detect automatically by measuring parameters linked to these behaviours. Additionally, by using image analysis it was possible to detect inaccuracies in the fluoroscopic images and notify developers of these inaccuracies through graphs and concrete examples (i.e., screenshots).

Moreover, the remaining NFRs, *performance, portability and scalability*, were selected from the taxonomy since they cover the overall quality of the software and inhibit characteristics that are important for delivering a good user experience. The tests designed for these NFRs verify that the software is within an acceptable region of performance and that the software delivers a good user experience on multiple hardware setups. These tests also provide developers with insight regarding which medical procedures of the software could be changed or extended in order to achieve a better overall user experience. By measuring the frame rate of the simulation as a parameter of user experience, when executing specific medical procedures on different hardware, it was possible to inform the software developers of the overall *performance* and *portability* of a particular medical procedure within the Mentice simulation. Furthermore, these results could then be used by developers to evaluate the *scalability* of the software and whether or not a particular medical procedure should be modified for increased user experience. It could also inform the developers whether there was room for extending the functionalities within the procedure without losing out on the user experience of the procedure.

This thesis provides examples of how NFRs can be identified and classified during the software development of Medtech software. Furthermore, it provides examples of how an automated test framework can be built in order to evaluate NFRs with respect to certain aspects, in this case, the user experience of the software. Tests developed in this thesis suggest that it is possible to successfully evaluate NFRs through automated tests, and thus reduce the workload of manual testing for software developers and project managers. It is suggested, through these results, that NFRs should not be overlooked during software development and that it is possible to obtain great value from software testing by incorporating tests of NFRs that evaluate properties of the software which are of great importance in delivering a good user experience.

Bibliography

[1] J. Defranco, M. Kassab, P. Laplante, and N. Laplante, "The nonfunctional requirement focus in medical device software: A systematic mapping study and taxonomy," *Innov. Syst. Softw. Eng.*, vol. 13, p. 81–100, Sept. 2017.

[2] M. Kassab, "The changing landscape of requirements engineering practices over the past decade," in *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pp. 1–8, 2015.

[3] B. Y. E. M. J. Chung, L. Nixon, *Non-Functional Requirements in Software Engineering*. 2000.

[4] M. A. Makary and M. Daniel, "Medical error—the third leading cause of death in the us," *BMJ*, vol. 353, 2016.

[5] S. Hooper, "Automated Testing and Validation of Computer Graphics Implementations for Cross-platform Game Development," 2017.

[6] "Software testing." `https://en.wikipedia.org/wiki/Software_testing`. Accessed: 2021-05-03.

[7] T. Wetzlmaier and R. Ramler, "Hybrid monkey testing: Enhancing automated gui tests with random test generation," in *Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing*, A-TEST 2017, (New York, NY, USA), p. 5–10, Association for Computing Machinery, 2017.

[8] J. Yan, H. Zhou, X. Deng, P. Wang, R. Yan, J. Yan, and J. Zhang, "Efficient testing of gui applications by event sequence reduction," *Science of Computer Programming*, vol. 201, p. 102522, 2021.

[9] "Why is the difference between functional and Non-functional requirements important?." `https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/`. Accessed: 2021-05-03.

[10] "Non-functional requirement." `https://en.wikipedia.org/wiki/Non-functional_requirement`. Accessed: 2021-05-03.

[11] "Functional vs non-functional requirements." `https://www.guru99.com/functional-vs-non-functional-requirements.html`.

[12] "Non-functional requirement - performance." `https://en.wikipedia.org/wiki/Computer_performance`. Accessed: 2021-05-03.

[13] "Non-Functional Requirement - Scalability." `https://en.wikipedia.org/wiki/Scalability`. Accessed: 2021-05-03.

[14] "Software Portability." `https://en.wikipedia.org/wiki/Software_portability`. Accessed: 2021-05-03.

[15] "Non-Functional Requirement - Efficiency." `https://www.altexsoft.com/blog/non-functional-requirements/`. Accessed: 2021-05-03.

[16] "Non-functional requirement - capacity." https://www.oreilly.com/library/view/mastering-non-functional-requirements/9781788299237/5fa609e1-8dde-43fd-9e53-8b7d972e8325.xhtml. Accessed: 2021-05-03.

[17] "Non-functional requirement - availability." https://www.altexsoft.com/blog/non-functional-requirements/. Accessed: 2021-05-03.

[18] "Reliability engineering." https://en.wikipedia.org/wiki/Reliability_engineerin. Accessed: 2021-05-03.

[19] "Non-functional requirement - manageability." https://www.oreilly.com/library/view/mastering-non-functional-requirements/9781788299237/6c43449b-b923-40b5-810f-1710544d9de1.xhtmll. Accessed: 2021-05-03.

[20] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*. USA: John Wiley & Sons, Inc., 1st ed., 1997.

[21] M. E. Paschali, A. Ampatzoglou, A. Chatzigeorgiou, and I. Stamelos, "Non-functional requirements that influence gaming experience: A survey on gamers satisfaction factors," in *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, AcademicMindTrek '14, (New York, NY, USA), p. 208–215, Association for Computing Machinery, 2014.

[22] H. Ham and Y. Lee, "An empirical study for quantitative evaluation of game satisfaction," vol. 2, pp. 724 – 729, 12 2006.

[23] J. Balkin and B. Noveck, "The state of play : law, games, and virtual worlds," 2006.

[24] T. Young, "A Usability Analysis of Video Games: The Development of Assessment Standards," (Ball State University Muncie, Indiana), 2011.

[25] R. R. Maiti and F. J. Mitropoulos, "Prioritizing non-functional requirements in agile software engineering," in *Proceedings of the SouthEast Conference*, ACM SE '17, (New York, NY, USA), p. 212–214, Association for Computing Machinery, 2017.

[26] K. Riņķevičs and R. Torkar, "Equality in cumulative voting: A systematic review with an improvement proposal," *Information and Software Technology*, vol. 55, no. 2, pp. 267–287, 2013.

[27] K. A. Memon, X. Xiaoling, and H. Halepoto, "Some serious deliberations and reflections on the significance of non-functional requirements for improving the quality of software," in *Proceedings of the 2019 8th International Conference on Software and Information Engineering*, ICSIE '19, (New York, NY, USA), p. 38–41, Association for Computing Machinery, 2019.

[28] D. Mairiza, D. Zowghi, and N. Nurmuliani, "An investigation into the notion of non-functional requirements," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (New York, NY, USA), p. 311–317, Association for Computing Machinery, 2010.

[29] M. Lu and P. Liang, "Automatic classification of non-functional requirements from augmented app user reviews," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, (New York, NY, USA), p. 344–353, Association for Computing Machinery, 2017.

[30] Nurbojatmiko, E. K. Budiardjo, and W. C. Wibowo, "SLR on identification & classification of non-functional requirements attributes, and its representation in functional requirements," in *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, CSAI '18, (New York, NY, USA), p. 151–157, Association for Computing Machinery, 2018.

[31] M. Majthoub, Y. Odeh, and M. Hijjawi, "Non-functional requirements classification for aligning business with information systems," in *Proceedings of the 2020 International Conference on Big Data in Management*, ICBDM 2020, (New York, NY, USA), p. 84–89, Association for Computing Machinery, 2020.

[32] X. Franch and P. Botella, "Putting non-functional requirements into software architecture," in *Proceedings Ninth International Workshop on Software Specification and Design*, pp. 60–67, 1998.

[33] M. L. Barrett, "Putting non-functional requirements to good use," *J. Comput. Sci. Coll.*, vol. 18, p. 271–277, Dec. 2002.

[34] K. Dyrkorn and F. Wathne, "Automated testing of non-functional requirements," in *Companion to the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*, OOPSLA Companion '08, (New York, NY, USA), p. 719–720, Association for Computing Machinery, 2008.

[35] A. Yrjönen and J. Merilinna, "Tooling for the full traceability of non-functional requirements within model-driven development," in *Proceedings of the 6th ECMFA Traceability Workshop*, ECMFA-TW '10, (New York, NY, USA), p. 15–22, Association for Computing Machinery, 2010.

[36] A. Raturi, B. Penzenstadler, B. Tomlinson, and D. Richardson, "Developing a sustainability non-functional requirements framework," in *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, GREENS 2014, (New York, NY, USA), p. 1–8, Association for Computing Machinery, 2014.

[37] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are "non-functional" requirements really non-functional? an investigation of non-functional requirements in practice," in *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, (New York, NY, USA), p. 832–842, Association for Computing Machinery, 2016.

[38] M. Broy, "Rethinking nonfunctional software requirements," *Computer*, vol. 48, pp. 96–99, 2015.

[39] "C++." `https://en.wikipedia.org/wiki/C%2B%2B`. Accessed: 2021-05-03.

[40] "Javascript." `https://en.wikipedia.org/wiki/JavaScript`. Accessed: 2021-05-03.

[41] "Node.js." `https://en.wikipedia.org/wiki/Node.js`. Accessed: 2021-05-03.

[42] "Python." `https://en.wikipedia.org/wiki/Python_(programming_language)`. Accessed: 2021-05-03.

[43] "Image processing in Python." `https://scikit-image.org`. Accessed: 2021-05-04.

[44] "subprocess - Subprocess management." `https://docs.python.org/3/library/subprocess.html`. Accessed: 2021-05-03.

[45] "Psutil documentation." `https://psutil.readthedocs.io/en/latest/#`. Accessed: 2021-05-03.

[46] "Pandas." `https://pandas.pydata.org`. Accessed: 2021-05-03.
[47] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
[48] "CpuStres v2.0." `https://docs.microsoft.com/en-us/sysinternals/downloads/cpustres`. Accessed: 2021-05-05.
[49] "Microsoft Azure." `https://azure.microsoft.com/en-us/`. Accessed: 2021-05-14.
[50] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 2004.
[51] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image Analysis Using Mathematical Morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1987.

A

Appendix 1

A.1 Technical description of algorithms used

Here follows an explanation of algorithms used for test: "Detection of tool outside artery walls". Algorithms used are scale-invariant feature detection, structural similarity index measurement, and morphological skeletonization.

A.1.1 SIFT algorithm

SIFT is a feature detection algorithm that is used in computer vision to detect and describe local features within an image. These features could then be compared between two images and thus detect movement of objects or if equal features are present in both images and located in each of the images. The algorithm is computed by performing *Scale-space Peak Selection, Keypoint Localization, Orientation Assignment, Keypoint description, and Keypoint matching* [50].

Scale-space peak selection is performed by first separating the image into octaves which, for each octave, halves the size of the original image. The number of octaves obtained depends on the size of the input image. Next, each octave (sub-image) is progressively blurred with a Gaussian (blur) kernel for every pixel resulting in multiple images within each octave being blurred at different grades [50]. The blurred image is computed as follows:

$$\mathrm{L}(x, y, \sigma) = \mathrm{G}(x, y, \sigma) * \mathrm{I}(x, y)$$

Where I is an image, G is the Gaussian Kernel, and x, y are image coordinates. σ is a scale parameter that could be described as the amount of blurring.

$$\mathrm{G}(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2-y^2)/2\sigma^2}$$

Next, the Difference of Gaussian (DoG) Kernel is computed by calculating the difference between each pair of Gaussian blurred images within an octave, i.e., for a given octave:

$$\mathrm{DoG} = \mathrm{L}(x, y, \sigma) - \mathrm{L}(x, y, k\sigma)$$

resulting in $n - 1$ DoG images from a set of n blurred images within an octave [50].

Next, the DoG is used to calculate Laplacian of Gaussian approximations that are scale-invariant, i.e., a pixel in an image in an octave is compared with its 8

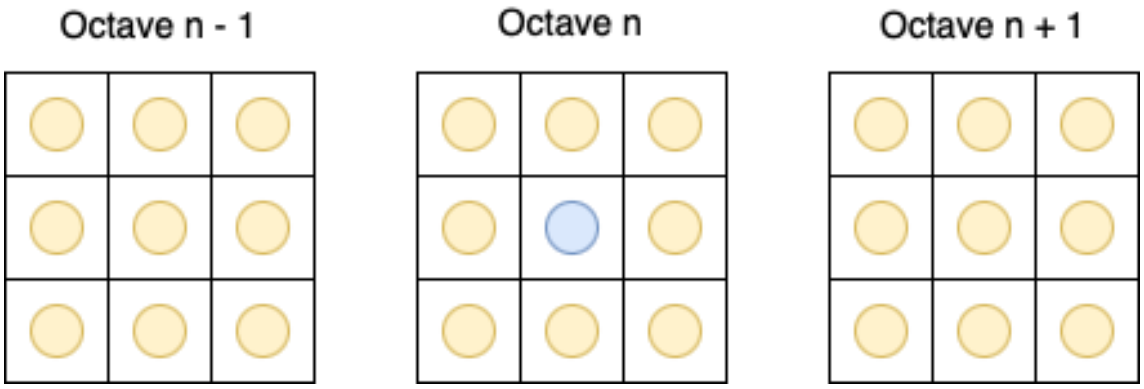


Figure A.1: Blue pixel in octave n is compared to its 8 yellow neighbours and 9 yellow neighbours in octave n ± 1 to find local extrema.

neighbours as well as compared to its 9 neighbours in the previous and next octave. By performing a total of 26 checks it is possible to draw a conclusion whether this pixel in this octave is a local extreme, i.e., a potential keypoint for that specific octave (scale) (see Fig. A.1) [50].

The next step is to perform *Keypoint localization* which purpose is to remove non-useful features detected in the previous step. This step makes use of the Taylor series expansion of the scale-space to get a more accurate localization of the extrema. A threshold value is used and if the intensity in the extrema is less than this threshold the keypoint is rejected. The DoG approach used above has a higher hit rate for edges within the image that needs to be removed. A Hessian matrix (H) is used to compute the principal curvature (how the image intensity bends) in a keypoint [50].

$$\text{Threshold for low intensities: } |D(\hat{x})| < 0.03$$

$$\text{Edge rejection: } \mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

α = eigenvalue with larger magnitude, β = eigenvalue with smaller magnitude

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta$$

$$\text{Tr}(\mathbf{H}) = D_{xx}D_{yy} - D_{xy}^2 = \alpha\beta$$

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}, \text{where r} = \alpha\beta \text{ and r} < 10$$

After this step, a better representation of the keypoints within an image is obtained. The scale in which the keypoints was detected is also known resulting in a scale-invariant measurement [50].

The next step is to perform an *Orientation Assignment* of the keypoints in order to make them invariant to rotation. A neighbourhood is taken around the keypoint location, this neighbourhood is dependent on the scale of the octave. Gradient magnitude and direction are calculated for each pixel in this region resulting in an orientation histogram (360 °) for all pixels in this neighbourhood. The highest peak of the histogram bins, and all peaks 20% below this peak, are taken into

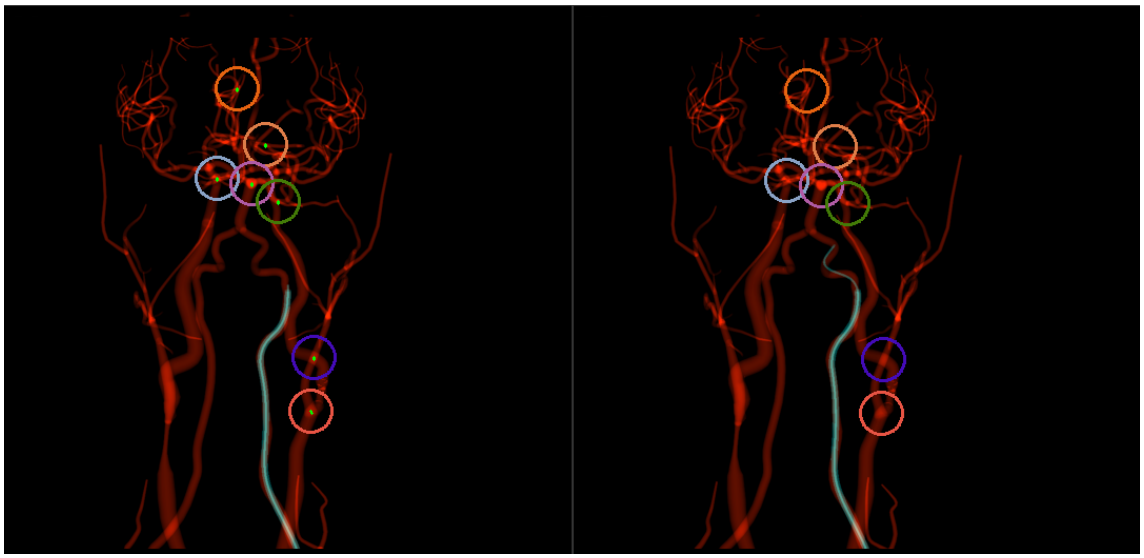


Figure A.2: Visualization of how keypoints match, where the match is denoted by the color of circles between the two frames. Left image shows frame 1 and the image to the right frame 2. Difference between the two frames can be seen if looked upon the tool movement.

consideration when calculating the orientation of the keypoint [50]. After this step, all keypoints found has a location, scale and orientation [50].

Followed by *Orientation Assignment*, next is to calculate a *Keypoint Descriptor* for the keypoint which is highly distinctive and as invariant as possible with respect to viewpoint and illumination. In order to perform this step a window of pixels around each keypoint is taken, this window is then divided into sub-windows and for each sub-window an orientation histogram is calculated which describes the feature orientation of the pixels surrounding a specific keypoint. In order to deal with rotation invariance and achieve rotation independence all orientations within this sub-window are subtracted by the keypoints rotation given in the *Orientation Assignment* step. This gives a relative measure of the feature orientation with respect to the feature orientation. Also, in order to achieve illumination dependence, each feature orientation magnitude within the sub-window is thresholded in order to achieve a normalized feature orientation vector of the pixels in the window surrounding the keypoint [50].

The last step is to use these features obtained from the steps above for two images and try to match these features to one another, also known as *Keypoint matching* [50]. Keypoints between two images are matched by identifying their nearest neighbours. There might be multiple matches for the second closest-match for a given keypoint in another image and in this case, the ratio of the closest-distance to second-closest is taken, if this ratio is too large the keypoint match is rejected. According to [50], by performing this step the algorithm eliminates 90% of false matches while only discarding 5% of correct matches.

A.1.2 SSIM algorithm

Structural similarity index measure (SSIM) is a method used for comparing two images. Compared to the Mean-Square Error (MSE) algorithm, which quantifies the difference in values between each corresponding pixel in the sample and reference image, the SSIM algorithm is a perception-based model. This means that the algorithm evaluates the perceived change in structural information within the image and thus give an image comparison algorithm that closely mimics the human visual perception system. SSIM extracts three key features from an image: *luminance, contrast* and *structure*. These three features are then compared separately between two images and then combined to a result. The result is given as a *Structural Similarity Index* which is often set as 0 (no similarity) and 1 (complete similarity) [47]. In the following formulas, the comparison is made between images which are denoted as image x and image y.

Luminance (μ) of an image (x) is measured by averaging over all pixels. This computation is performed for both images, and compared, by the function $\mathrm{L}(x, y)$.

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\mathrm{L}(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2\mu_y^2 + C_1}$$

where C_1 is a constant ensuring stability by avoiding division by zero:

$$C_1 = (K_1 L)^2$$

and L is the dynamic range for the pixel values (8-bit = 0–255), K_1 is a constant. *Contrast* (σ) of an image (x) is measured as the standard deviation of all pixel values. Same as for luminance, a function $\mathrm{C}(x, y)$, computes and compares two images:

$$\sigma_x = (\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2)^{\frac{1}{2}}$$

$$\mathrm{C}(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

where:

$$C_2 = (K_2 L)^2$$

The structure of an image is measured by dividing the image signal by its standard deviation giving a result of unit standard deviation. This standard deviation is computed and compared between two images by a function $\mathrm{S}(x, y)$,

$$\frac{x - \hat{\mu_x}}{\sigma_x}$$

$$\mathrm{S}(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

where the covariance betweeen image x and y is computed as:

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

Lastly, the SSIM score between image x and y is calculated by multiplying the luminance, contrast and structure functions:

$$\mathrm{SSIM}(x,y) = [\mathrm{L}(x,y)]^{\alpha} \cdot [\mathrm{C}(x,y)]^{\beta} \cdot [\mathrm{S}(x,y)]^{\gamma}$$

α, β and γ denotes the relative importance of each aspect of the comparison and if set equal to 1, and $C_3 = \frac{C_2}{2}$ the SSIM score can be simplified as follows:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Another note is that most SSIM algorithms apply these formulas locally in the image instead of globally in order to achieve a better image comparison. In this thesis, the SSIM algorithm used does exactly this. By applying the steps formulated above locally (over smaller regions) for each image and then calculating the mean for all regions, the algorithm instead computes the *Mean Structural Similarity Index* and the formulas are altered as follows according to [47]:

$$\mu_x = \sum_{i=1}^{N} w_i x_i$$

$$\sigma_x = (\sum_{i=1}^{N} w_i (x_i - \mu_x)^2)^{\frac{1}{2}}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y)$$

where, according to [47], w is an 11×11 Gaussian Weighting function in which values are derived from a Gaussian distribution derived pixel-by-pixel across the image.

Finally, once all computations are made across the complete image the mean is taken given the *Mean Structural Similarity Index* measurement [47]:

$$\mathrm{MSSIM}(x,y) = \frac{1}{M}\sum_{j=1}\mathrm{MSSIM}(x_j, y_j)$$

A.1.3 Morphological skeleton

Mathematical morphology is an approach of image processing in which morphological operations simplify image data while preserving the shape characteristics of the objects within the image [51]. There is both morphological dilation, which adds pixels to the boundaries of an object within an image, and morphological erosion, which removes pixels at the boundaries of an object within the image. Both operations are performed on binary (0 or 1 pixel value) images. The number of pixels that are added or removed depending on the size and shape of the structuring element used

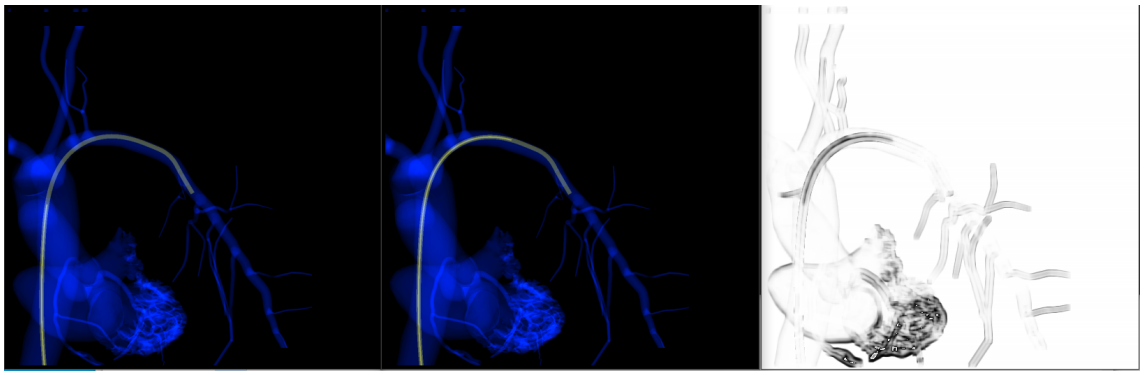


Figure A.3: Visualization of SSIM algorithm. The comparison between the middle image and the left image results in the image on the right. Darker areas represent larger structural differences. The result shows darker areas around the tool and heart since the tool has moved between images and the heart is moving during continuous animation.

to process the image. Pixels are added (dilation) or removed (erosion) depending on the values of the surrounding pixels, i.e., if, for a specific pixel in the input image, **any** surrounding pixel is equal to 1, the specific pixel is set to 1 in the output image when performing dilation. For erosion the opposite is true, if any surrounding pixel is set to 0, the specific pixel in the output image is set as 0. These operations have the drawback of filling (dilation) or removing (erosion) objects within an image if the operation is iterated multiple times which leads to an output image missing certain objects or having objects connected that were not connected in the original image [51].

The morphological skeleton is achieved by performing erosion but adding a certain rule to it. The rule is that pixel connectivity has to be maintained. By eroding an image object **as long as** the connectivity between the pixels is maintained an object within an image could be eroded to its maximum without losing the shape of the object. This results in a 'skeleton' representing the shape of the original object. It is important to note that this action is irreversible, i.e., there is no information stored of the original shape, and thus impossible to reproduce the original shape from the skeleton shape.

A.2 Additional graphs from performance, scalability and portability test

This sections include graphs from performance measurements during test: "Non-functional requirement: Performance, Scalability, and Portability" found in Sect. 4.5.

A.2.1 Simulation frame rate graphs

Following graphs, Fig. A.5–A.15, depict the test result for procedure *PA* on the *DELL PRECISION M6700* computer (*DELL67*) and test result for both computers, *DELL PRECISION M6700/M6800*, for procedure *PB*.

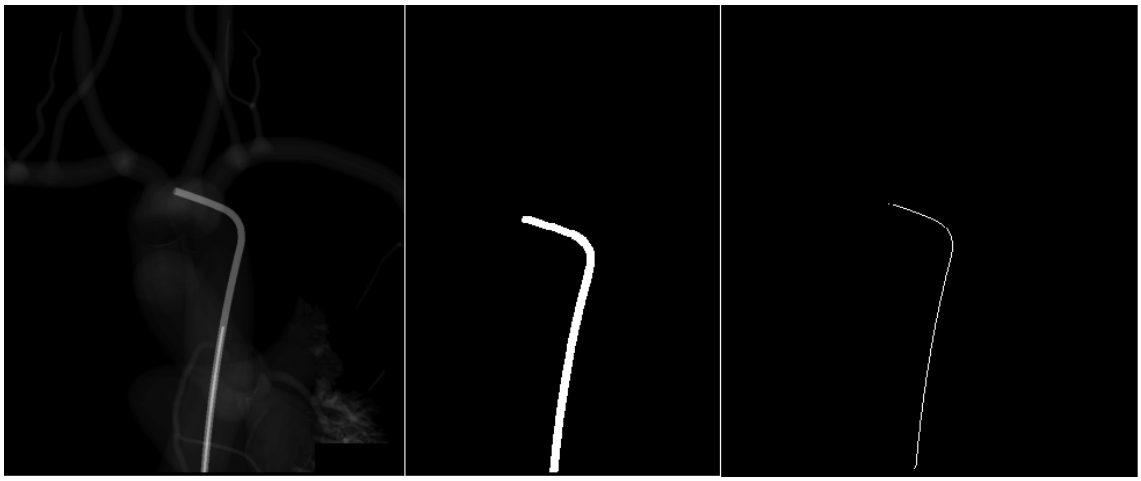


Figure A.4: Visualization of morphological skeleton, from left to right illustrates the process of filtering the image and removing pixels image object until the shape only has one pixel width left.

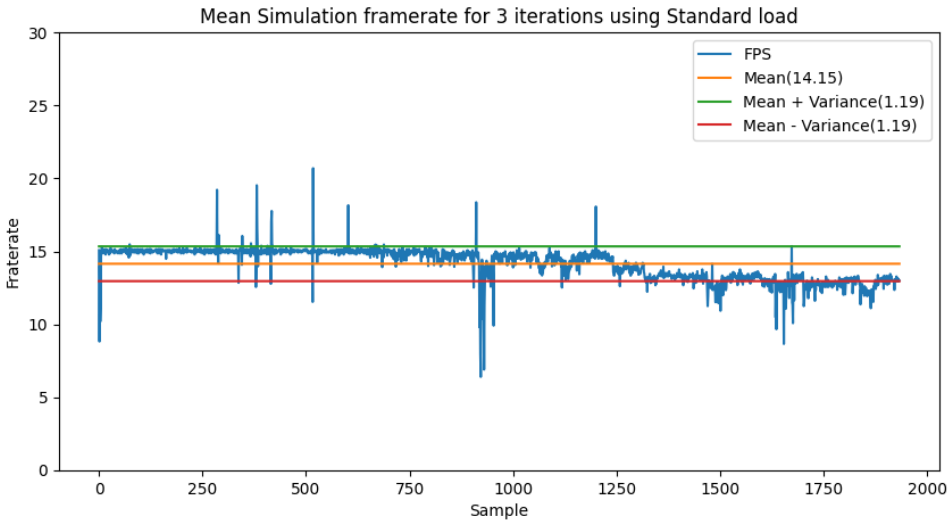


Figure A.5: Simulation frame rate of *PA* with *no*/0% of CPU power removed (*DELL67*).

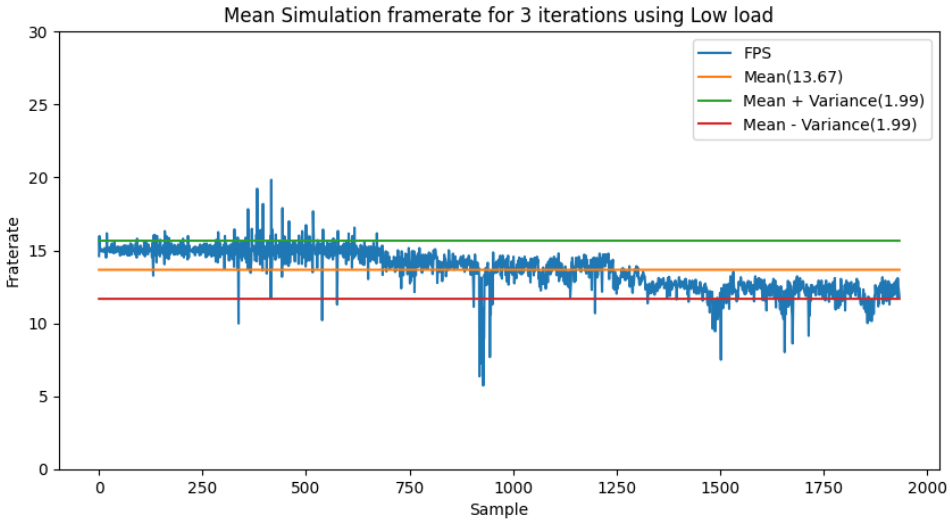


Figure A.6: Simulation frame rate of *PA* with *low*/25% of CPU power removed (*DELL67*).

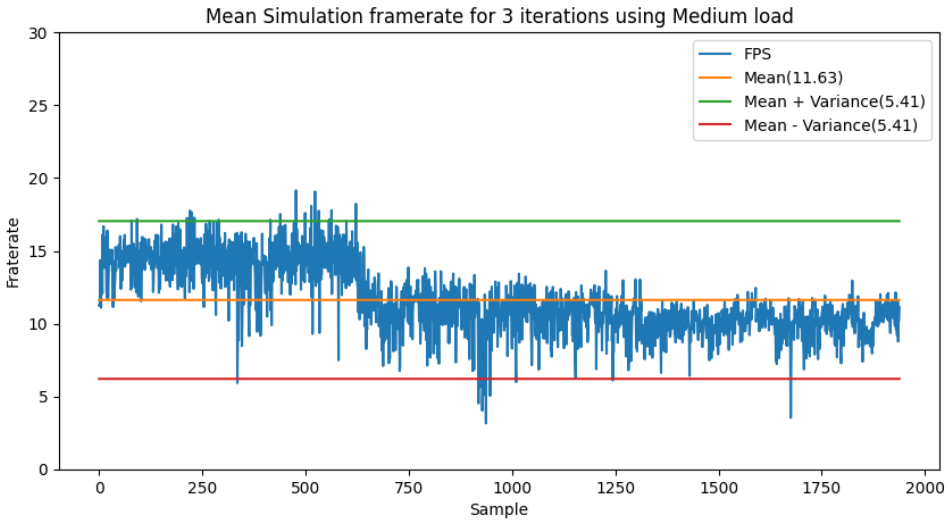


Figure A.7: Simulation frame rate of *PA* with *medium*/50% of CPU power removed (*DELL67*).

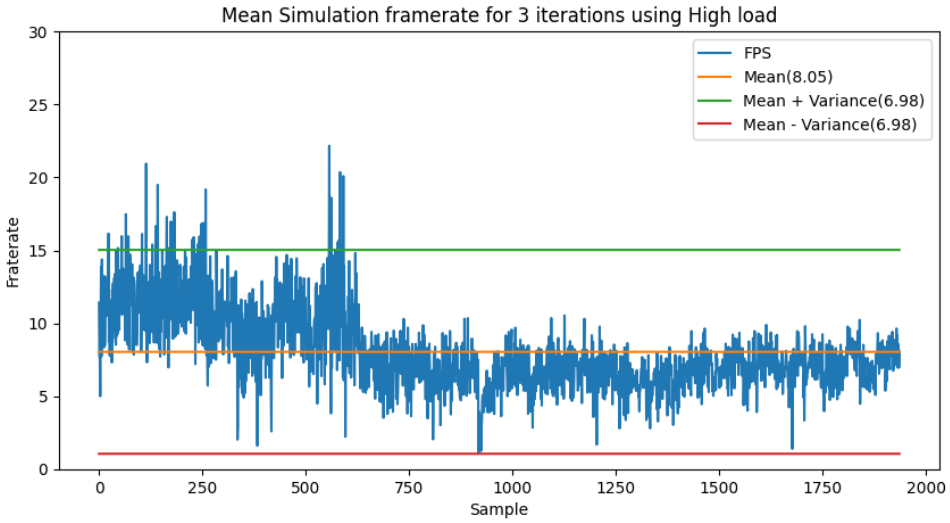
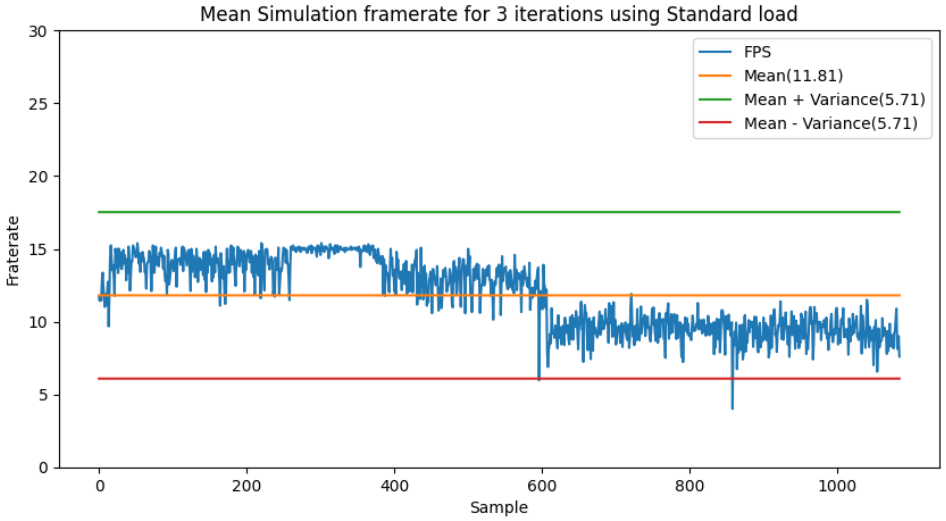


Figure A.8: Simulation frame rate of *PA* with *high*/75% of CPU power removed (*DELL67*).



[Result: Simulation frame rate, Procedure: PB, CPU removal: 0% , DELL68]Simulation frame rate of *PB* with *no*/0% of CPU power removed (*DELL68*).

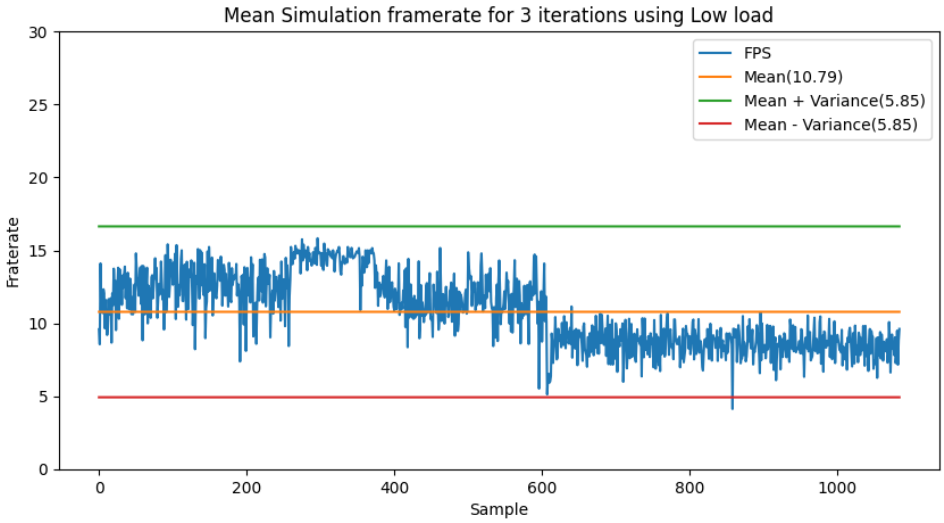


Figure A.9: Simulation frame rate of *PB* with *low*/25% of CPU power removed (*DELL68*).

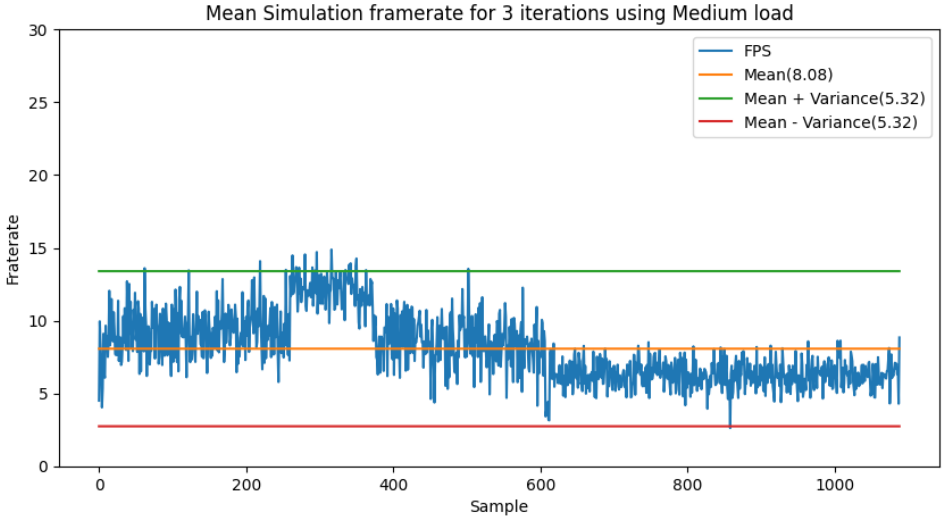


Figure A.10: Simulation frame rate of *PB* with *medium*/50% of CPU power removed (*DELL68*).

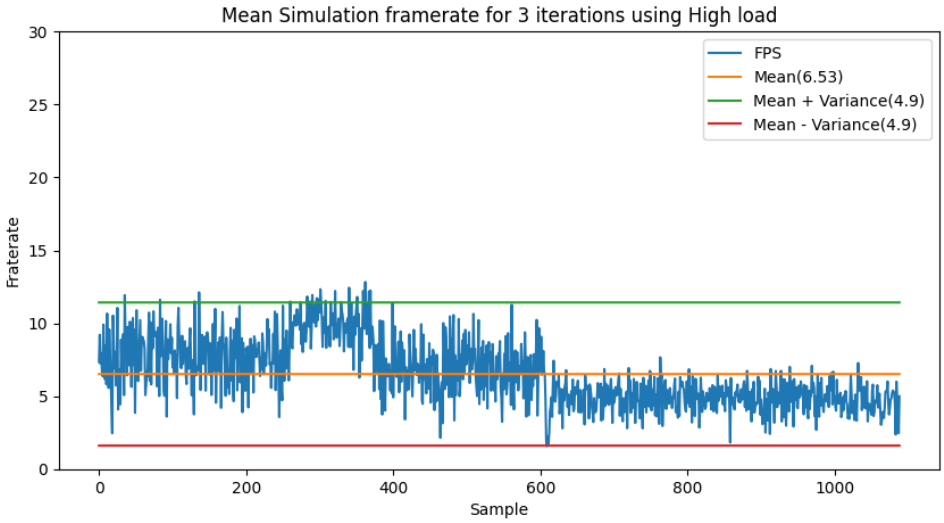


Figure A.11: Simulation frame rate of *PB* with *high*/75% of CPU power removed (*DELL68*).

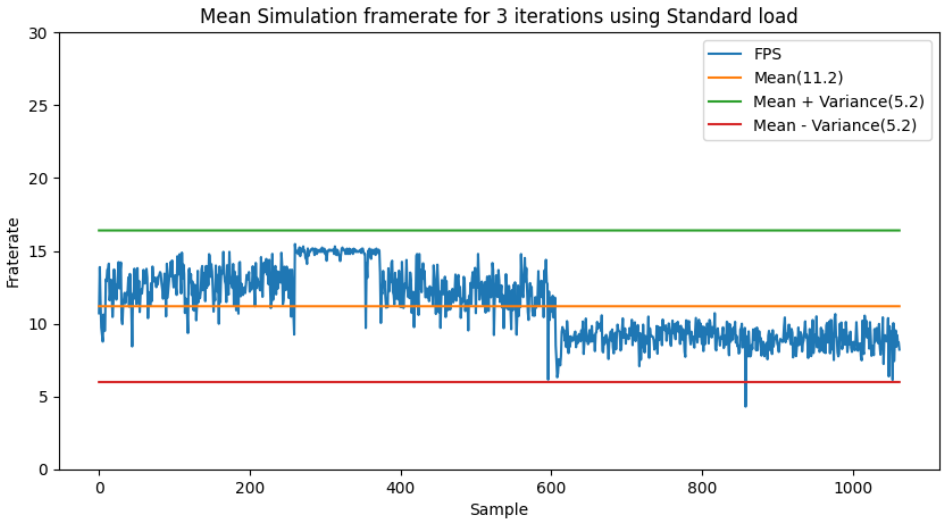


Figure A.12: Simulation frame rate of *PB* with *no*/0% of CPU power removed (*DELL67*).

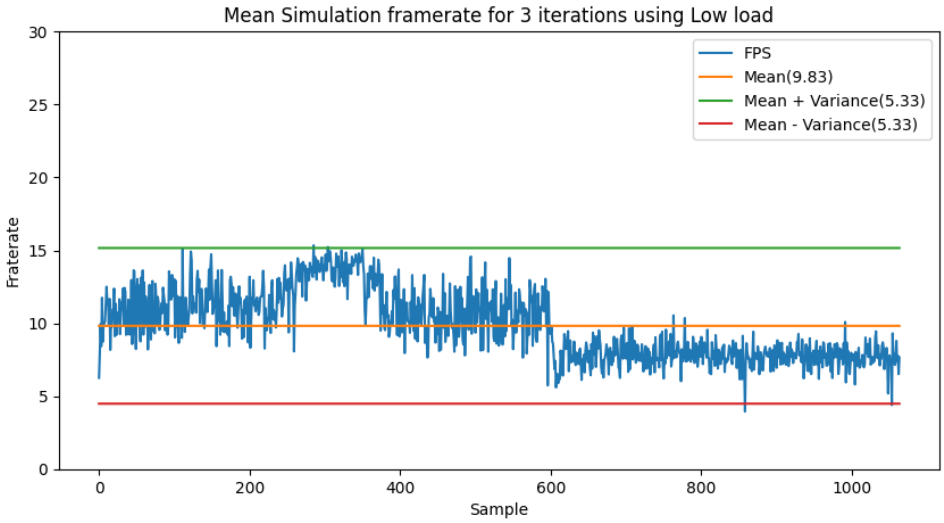


Figure A.13: Simulation frame rate of *PB* with *low*/25% of CPU power removed (*DELL67*).

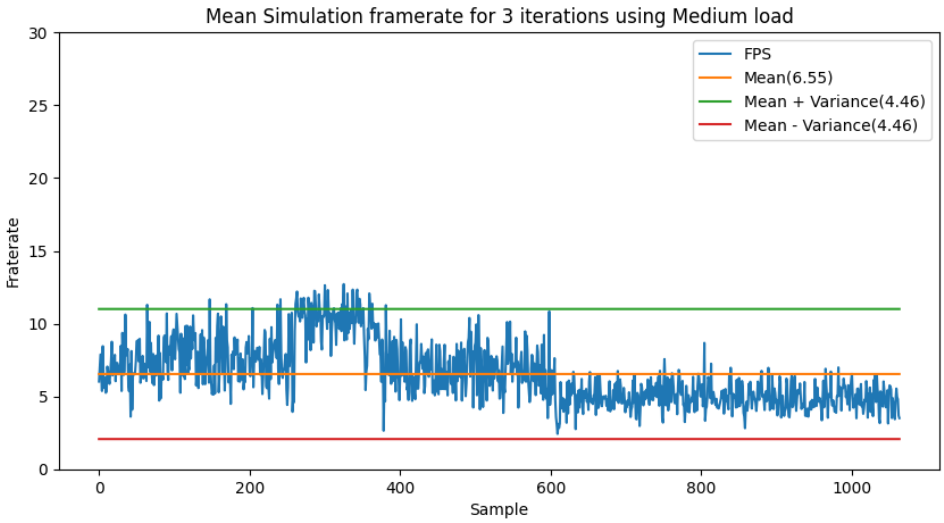


Figure A.14: Simulation frame rate of *PB* with *medium*/50% of CPU power removed (*DELL67*).

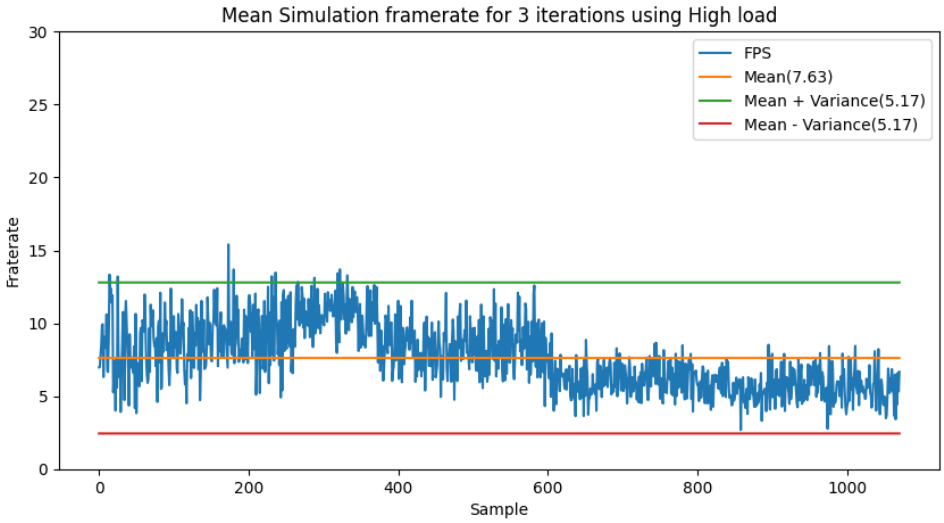


Figure A.15: Simulation frame rate of *PB* with *high*/75% of CPU power removed (*DELL67*).

A.2.2 Physics engine frame rate graphs

Below are graphs displaying physics engine frame rate over different CPU power removals. Results displayed in Chap. 4, Sect. 4.5.2, Fig. 4.29/4.31 were based on these graphs.

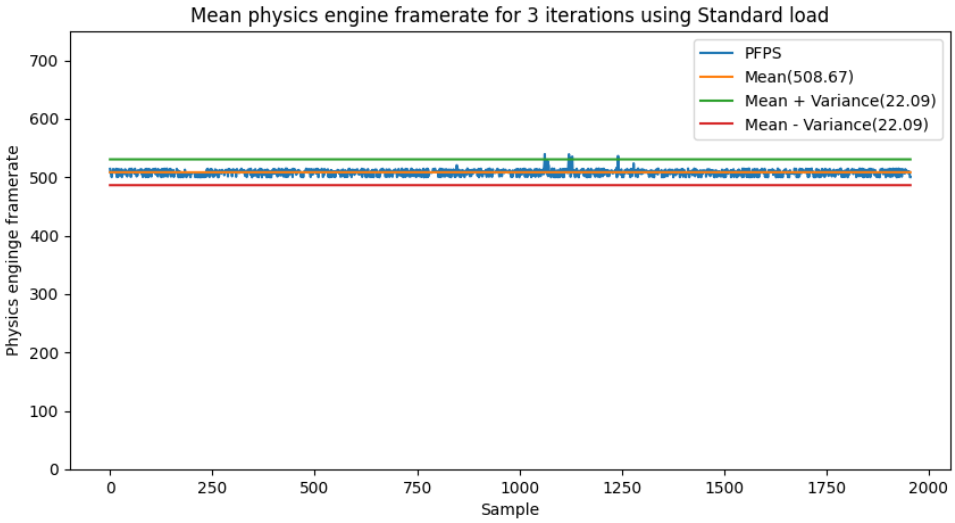


Figure A.16: Physics engine frame rate of *PA* with *no*/0% of CPU power removed (*DELL68*).

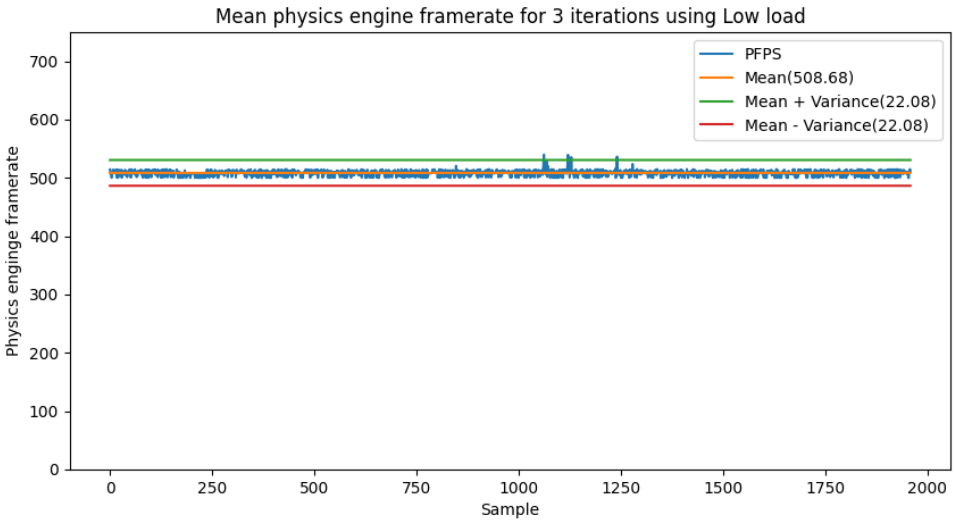


Figure A.17: Physics engine frame rate of *PA* with *low*/25% of CPU power removed (*DELL68*).

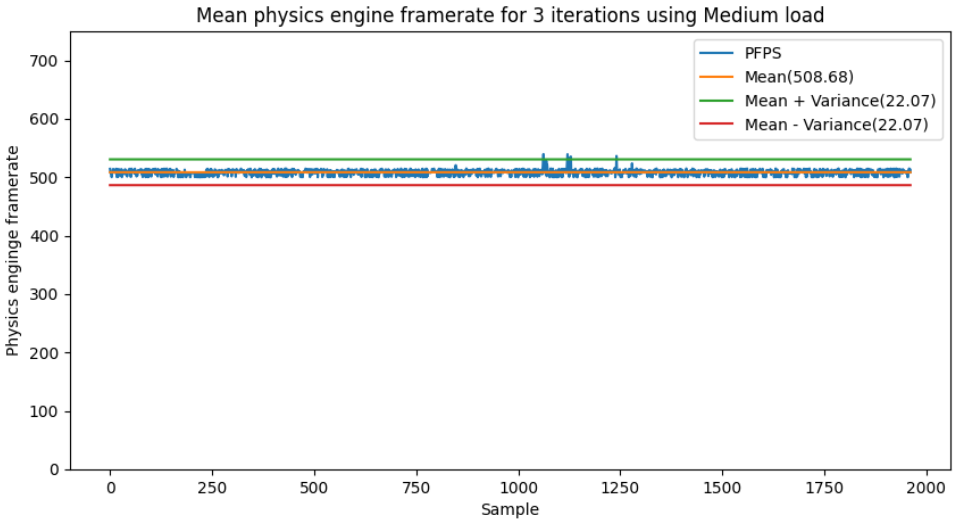


Figure A.18: Physics engine frame rate of *PA* with *medium*/50% of CPU power removed (*DELL68*).

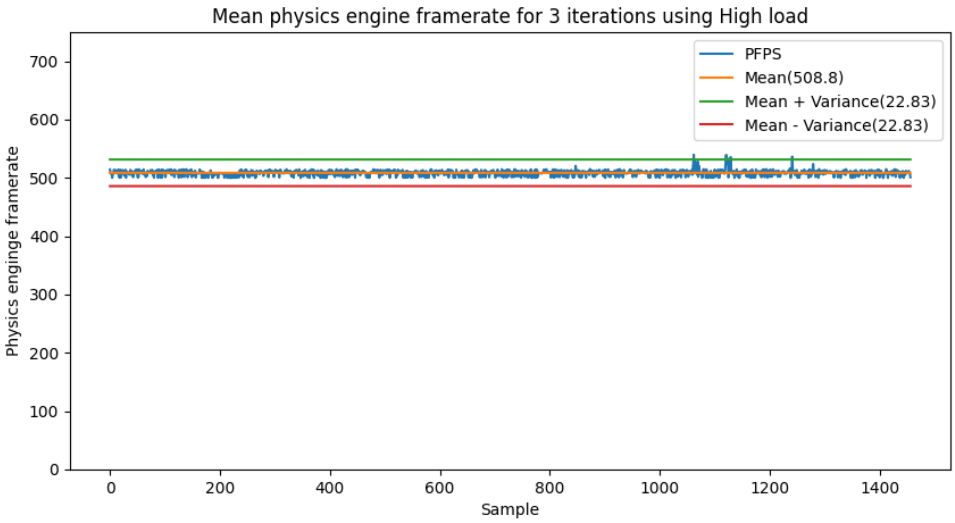


Figure A.19: Physics engine frame rate of *PA* with *high*/75% of CPU power removed (*DELL68*).

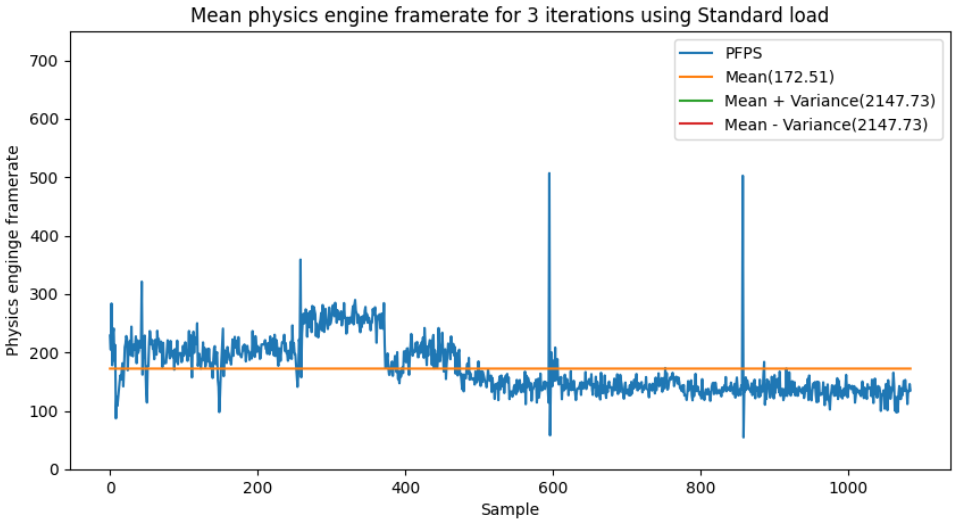


Figure A.20: Physics engine frame rate of *PB* with *no*/0% of CPU power removed (*DELL68*).

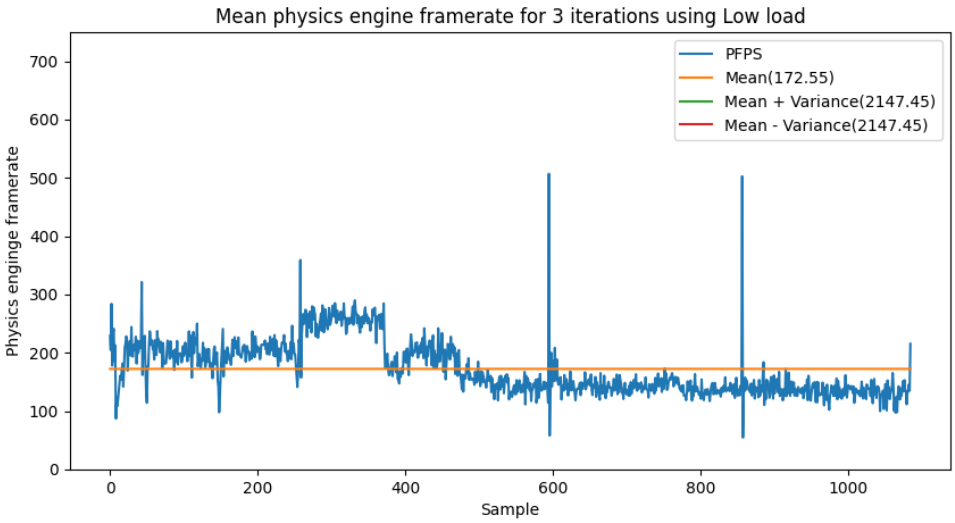


Figure A.21: Physics engine frame rate of *PB* with *low*/25% of CPU power removed (*DELL68*).

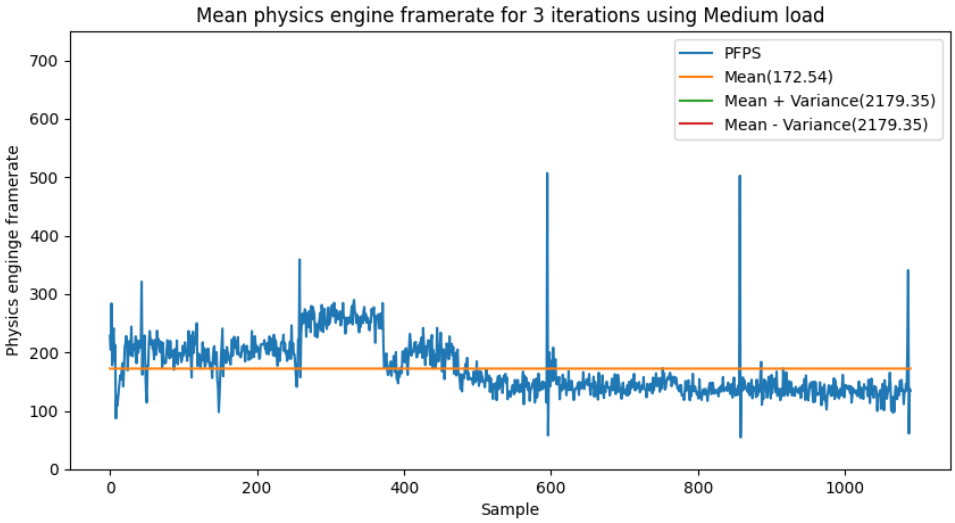


Figure A.22: Physics engine frame rate of *PB* with *medium*/50% of CPU power removed (*DELL68*).

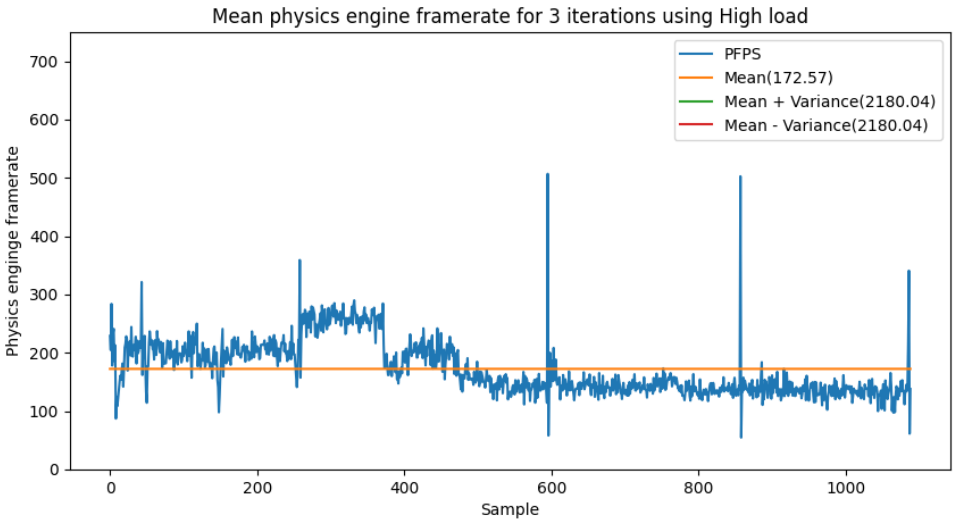


Figure A.23: Physics engine frame rate of *PB* with *high*/75% of CPU power removed (*DELL68*).

DEPARTMENT OF MATHEMATICAL SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY