Universiteit Utrecht

UMC Utrecht

Master's Programme in Artificial Intelligence
June 2022
Degree Thesis

Monitoring da Vinci, Safety Managed by a Robot.

A study into Runtime Verification and Monitoring Systems for Robot-Assisted Surgery

Master Thesis

Author:	Y.A. Macrander
Main Supervisor:	dr. N.A. Alechina, Department of Information and Computing Sciences, Utrecht University
Second supervisor:	dr. B.S. Logan, Department of Information and Computing Sciences, Utrecht University
External supervisor:	prof. dr. J.P. Ruurda, R.B. den Boer & C. de Jongh Department of Surgical Oncology, UMC Utrecht

Abstract

Research into robot-assisted surgery (RAS) continues to grow each year. However, RAS is complex and requires extensive surgical training to acquire a sufficient proficiency level to master the skill of accurately controlling the surgical robot that is used for RAS. The most widely used robotic surgical system is named the da Vinci surgical system. Implementing a system that monitors the actions of the surgeon into the digital environment of the da Vinci surgical system could check the correctness of these actions (verification) at the actual time the action is performed (runtime).

The objective of this research is to provide the grounds for a Runtime Verification (RV) system to be used during Robot-Assisted Surgery to improve the patient safety by preventing inadvertent damages and complications during surgery.

To test the workings and the significance of a RV monitoring system in RAS, we present three examples of surgical behaviour that could potentially lead to damages and complications based on the use case of the robot-assisted minimally invasive esophagectomy (RAMIE) procedure. We have simulated these examples in a digital simulation environment and integrated a RV system to monitor the movements and motions of a 3D model of the da Vinci surgical system in a model of the thorax of a patient. We identified suitable formal languages and designed properties to detect undesired behaviour of the da Vinci robot. The monitor checks the behaviour against formal safety properties specified in signal temporal logic and issues a warning in the event a property is likely to be violated, with the idea to alert the surgeon of undesired behavioural activity before complications would arise during surgery.

Our constructed proof-of-concept RV system successfully simulated the monitoring of damage done to the aorta in a real-life surgical video as well as synthetic examples of hasty and hesitant movements of the robot arms. Experimental evaluation of these simulations demonstrate that detection of this undesired behaviour in runtime through the integrated RV system would have given a warning to the surgeon to not come in closer proximity to the aorta and would likely have mitigated and prevented the damage that was caused to the aorta. The ability to monitor movements of the components of the da Vinci surgical system applies to other parameters, like for example the speed of these components, aiding in preventing other undesired scenarios that could happen during RAS procedures and therefore increasing the safety of the patient in general.

Finally, we generalize our results and outline future possibilities for integrating runtime verification into the field of robot-assisted surgery.

Keywords:

Robot-Assisted Surgery, Runtime Verification, Monitoring, da Vinci Surgical System, RAMIE, Signal Temporal Logic

Acknowledgement

First and foremost, this research does not only represent a personal achievement, but also the dedication and hard work of all the people that are working on the adoption of artificial intelligence (AI) in a safe way and using these techniques to advance the medical field to increase patient safety as much as possible and thereby saving many more lives. The experience of working on this interesting and important topics has been amazing and humbling, getting to experience and read about the important surgical procedures that are being done in the field of robot-assisted surgery and getting to think about how AI can help this important field.

I want to thank my dedicated supervisors from the University Utrecht, firstly my main supervisor Dr. Natasha A. Alechina, for handing me the opportunity to work on this magnificent topic and help me progress forward during this research with her support and guidance through the whole research process. Secondly, I also want to thank Dr. Brian Logan for his expertise and additional help through the last 9 months. They helped me a lot in understanding the field of formal logic and in particular runtime verification and helped me from a computer science perspective in trying to bridge the knowledge gap between the technical and clinical field of research.

Furthermore, I would like to thank the surgical team from the UMC Utrecht, Prof. Dr. Jelle P. Ruurda, Robin B. den Boer and Cas de Jongh, for all the help on the clinical side of the research. Working with them was a real honor and their kindness, openness and always offering to help and answer questions made me feel right at home. It was a humble experience that I could be of help in the utmost important job they do.

Finally, I want to thank my family and friends for all the support during the project and providing mental support over the two years I've spent in my Master's degree journey during the COVID-19 pandemic.

Contents

List of Abbreviations

AI **A**rtificial **I**ntelligence

AMBF **A**synchronous and **M**ulti-**B**ody **F**ramework

CPS **C**yber **P**hysical **S**ystem

dVRK **d**a **V**inci **R**esearch **K**it

GUI **G**raphical **U**ser **I**nterface

JHU **J**ohn **H**opkins **U**niversity

JSON **J**ava**S**cript **O**bject**N**otation

LTL **L**inear **T**emporal **L**ogic

MIS **M**inimally **I**nvasive **S**urgery

MTL **M**etric **T**emporal **L**ogic

OR **O**peration **R**oom

PSM **P**atient **S**ide **M**anipulator

RAMIE **R**obot-**A**ssisted **M**inimally **I**nvasive **E**sophagectomy

RAS **R**obot-**A**ssisted **S**urgery

ROS **R**obot **O**perating **S**ystem

RV **R**untime **V**erification

STL **S**ignal **T**emporal **L**ogic

SAW **S**urgical **A**ssistent **W**orkstation

WPI **W**orcester **P**olytechnic **I**nstitute

List of Figures

1 Introduction

Minimally invasive surgery is a surgical approach to reduce surgical trauma in operations by using multiple small incisions (5 – 12mm), instead of inflicting a large incision as in open surgery. Minimally invasive techniques are conventional laparoscopy and robot-assisted surgery (RAS). Conventional laparoscopy is technically limited by suboptimal range of motion of surgical instruments and the two-dimensional camera controlled by an assisting surgeon resulting in unstable view. In contrast, RAS improves dexterity by four robot arms with hand-wrist articulation of instruments with 540 degrees rotation and 7 degrees of freedom, tremor filtering and three-dimensional magnified vision system that is controlled by a stable robot-arm and steered by the operating surgeon instead of an assisting surgeon [23]. These advancements facilitate, among other, an increase in image quality, depth perception, overall comfort, surgical motion precision and complex manoeuvring during suturing, reconstruction techniques and lymph node dissection compared to conventional laparoscopy [55]. The most widely used surgical robot is the four-armed da Vinci Xi surgical system from Intuitive Surgical Inc. [4].

However, RAS is complex and requires extensive surgical training to acquire a sufficient proficiency level, and is therefore associated with a substantial learning curve for novice surgeons [18]. Furthermore, the zoomed-in view of the surgical robot is necessary for highly detailed dissections, but complicates a full overview and proper anatomical orientation, especially for novice surgeons. This can lead to uncontrolled and hesitant movements by surgeons which can be dangerous when dealing with easily damaged anatomical structures [13]. Furthermore, complex robot-assisted surgeries still carry a high risk of complications. For example, robot-assisted minimally invasive esophagectomy (RAMIE) procedures carry complication rate of 59% and mortality rate of 7% [26, 54]. Some components of the da Vinci Xi system are not yet utilized to their maximum abilities. For example, a knowledge gap that arises is that the surgeon is not visually assisted or is provided feedback through the digital interface it interacts with during surgery. Here lie opportunities to increase the safety of robot-assisted surgeries for beginner but also experienced surgeons. Therefore these safety challenges that arise will need to be addressed to improve perioperative outcomes and increase patient safety.

Monitoring of actions performed by the surgeon could be helpful to improve RAS [8]. Implementing a system that monitors the actions of the surgeon into the digital environment of the da Vinci Xi system could check the correctness of these actions (verification) at the actual time the actions is performed (runtime). This runtime verification (RV) of the surgeons actions could result in being able to provide important and helpful feedback to the surgeon during surgery in respect to dangerous moves and help beginner surgeons to learn more efficiently. An example of the feedback such a monitoring system could provide is to alert the surgeon when undesired behavioural activity is being performed in the presence of anatomical structures that can easily be damaged and are vital for the safe completion of the surgery.

Current studies into monitoring systems in other fields mostly focus on autonomous

robotics where such systems are implemented in machinery used in environments like factories or engines in automobiles [15]. For example, a monitoring system implemented in a hybrid engine of an automobile can, among others, check that the average fuel consumption and electric power supply over a given time period does not fall under a certain threshold to avoid unexpected engine troubles. There are also studies where a monitoring system is applied on medical devices such as insulin pumps [27]. In this case, it is important to check that the patient does not become hypoglycemic. So the monitoring system checks if the blood glucose level is above a certain threshold. If it is under this threshold, it alerts the insulin pump to administrate more insulin. However, another knowledge gap is that these medical devices lack the dynamic motions that come into play during RAS [14]. Research done by Bresolin et al. (2017), has experimented in using formal verification to check the correctness of small automatic tasks like puncturing during RAS [8]. However, in addition to the fact this research focuses on autonomous robotic surgery instead of robot-assisted surgery, it covers small simple tasks and does not take into account the runtime aspect that is needed in RAS procedures like RAMIE. The continuous motions of the robot arms and the anatomical structures change dynamically during RAMIE and make for a high amount of sensors and parameters to monitor. This makes monitoring robot-assisted surgery procedures like RAMIE more challenging than less dynamic environments like monitoring an insulin pump. Therefore, the combination of dynamic motion robotics and the high importance of patient healthcare makes this an important research direction.

To address the prior mentioned knowledge gaps, first, literature on these topics has been reviewed. In this literature review, we address the current types of monitoring systems for verifying dynamic motion patterns and the fields they are currently used in and analyse their potential usability in robot-assisted surgery. Furthermore, we have identified the safety and learning curve challenges that arise during RAS, taking robot-assisted minimally invasive esophagectomy (RAMIE) as an example, which is removal of the esophagus and surrounding lymph nodes usually performed for esophageal cancer. These challenges in RAS have been documented through expert knowledge acquired from surgeons with experience with RAS, in this case specifically RAMIE. Based on this review of literature we argue how monitoring systems could tackle these challenges in RAS and what types of monitoring systems could aid in this.

Subsequently, to demonstrate the efficacy and effectiveness of RV-based monitoring systems in RAS, we present simulations of an digital surgical environment that implements a proof-of-concept RV-based monitoring system. We do this by constructing and simulating surgical scenarios (real and synthetic) in the digital simulation environment and check the correctness of the behaviour of a model of the da Vinci surgical system by using a fitting integrated RV monitoring system. By reviewing literature, the Asynchronous Multi-Body Framework (AMBF) simulator was found to be a fitting simulation environment due to it including models of components of the da Vinci surgical system that can be used during simulations [39]. Furthermore, as the AMBF simulator is based on the Robot Operating System (ROS) architecture, we integrated a ROS compatible RV monitoring system, namely ROSMonitoring [43, 17]. Ultimately, based on the literature and demonstrated simulations, we argue how RV-based monitoring systems could tackle the challenges in RAS.

Therefore, this study aims to improve perioperative outcomes and increase patient safety, by constructing a proof-of-concept RV-based monitoring system for demonstrating the effectiveness of verifying the actions performed during RAS in real time. This RV-based monitoring system uses monitoring algorithms and formal specifications to check properties of surgical actions during a procedure for violations, in order to give

feedback or issue warnings during surgery to address dangerous actions. This could potentially increase patient safety by improving perioperative clinical outcomes by warning the surgeon when they come into a dangerous situation. This aim can be summarized into the following research question, which can be divided into 3 sub-questions:

How can a runtime verification (RV) and monitoring system improve the patient safety during robot-assisted surgery (RAS)?

- What types of monitoring systems can be helpful in RAS?
- What challenges arise involving safety of the patient during a RAS?
- What specifications of the da Vinci robot can be monitored?

In the first part of this thesis we extensively discuss relevant literature from both the clinical as well as the technical side of this research. This review of literature will be discussed in chapter 2. Subsequently, we present the research methodology used during the process, including the simulation environment, the patient model, the da Vinci robot model and the RV-based monitoring system which will be discussed in chapter 3. Then, chapter 4 will show the results of our proposed RV system on three simulation examples based on real-life surgical videos and synthetic examples. Chapter 5 will discuss the results of the monitor verdicts and discuss the limitations regarding the research. Lastly, we will conclude this research and discuss future research in chapter 6.

2 Background & Related Work

Having outlined the problem and the knowledge gaps that accompany it, the following chapter maps out the literature found on the technical aspects as well as the clinical aspects of monitoring and verifying RAS. It outlines the findings of an extensive literature review which is needed for laying the foundation for answering the research questions. This chapter is divided into eight sections. The first three sections discuss robot-assisted surgery in general (section 2.1), the da Vinci surgical system that is most often used in RAS (section 2.2) and techniques that are used for improving RAS (section 2.3). Subsequently, we will introduce RV in general, discussing its framework (section 2.4). The general approaches to RV will be discussed (section 2.5) and afterwards we will reduce the scope by focusing on RV in continuous system (section 2.6). To connect the clinical and technical part we will introduce a case study around the RAMIE procedure (section 2.7) and finally discuss suitable RV approaches and techniques to help improving RAS in the RAMIE procedure (section 2.8).

2.1 Robot-Assisted Surgery and its Challenges

Surgical oncology procedures for treating cancer are very complex in nature and therefore require a long learning-curve and often lead to many post-operative complications [18]. Minimally invasive surgery is a surgical approach to reduce surgical trauma in operations by using multiple small incisions (5 – 12mm), instead of inflicting a large incision as in open surgery. Minimally invasive surgical techniques are conventional laparoscopy, thoracoscopy, and robot-assisted surgery (RAS). Notable advantages of minimally invasive surgery compared to open surgery are a shorter hospital stay, lower blood loss, and a faster rehabilitation period. Furthermore, discomfort, pain and trauma is decreased resulting from laparoscopic procedures compared to open surgery. However, conventional laparoscopy is technically challenging due to the suboptimal range of motion of surgical instruments by the lack of wrist movement inside the patient. Moreover, the 2D vision of the surgeon using the surgical camera results in loss of intraoperative depth perception [57]. Related to the 2D vision, the endoscope camera is held by a surgical assistant which can cause unstable video footage and may result in the surgeon becoming fatigued earlier during the procedure. These vision and moveability constraints of traditional laparoscopic surgery result in a poor ergonomic environment for the surgeons during procedures at the expense of surgical performance. According to several studies, these constraints have caused complications during surgery mostly related to abdominal injuries [49, 21].

In contrast, RAS has been developed to overcome the constraints that come with traditional laparoscopic surgery. One addition of RAS to laparoscopic surgery is the capability of teleoperation, which introduces a digital interface that separates the surgeon and the patient, greatly increasing the precision during MIS. The addition of a console for the surgeon and a patient robot in the operation room (OR) increases the ergonomic environment of the surgeon which has been proven to reduce stress of the sur-

geon [7]. It provides a more similar surgical experience for the surgeon to open surgery by, among others, adding more moveability, increasing the accuracy of the surgeon's motions past their natural ability and regaining dexterity through wrist-like movements [3, 37]. Tremor filtering is also a very important feature added by the use of RAS. Uncontrollable hand tremors are filtered when the movements of the surgeon are translated to the robotic arms which helps improving accuracy. Subsequently, RAS results in a lower amount of bedside staff during surgery which also aids in patient safety against infection transfer, reducing inter-personal contacts [16].

Another additional benefit of RAS is the decrease in the post operative hospital stay of the patient as a result of the smaller incisions that can be made when using RAS compared to traditional laparoscopy and open surgery. This increases the amount of free ICU beds for other critically ill cases which is especially preferable during pandemics. There is a caveat of longer OR-time because of extra preparation time for setting up the robot and longer procedure times [31]. However, this does not outweigh the importance of a shorter post operative hospital stay.

To summarize, RAS offers a safer and better surgical procedure compared to traditional laparoscopy and open surgery. As mentioned in chapter 1, RAS has solved issues and complications that arise with traditional laparoscopy in MIS and open surgery and provides the surgeon with beyond human capabilities. Furthermore, the inclusion of a robotic system aids in the training of novice surgeons by collecting data during the procedure and making room for digital supervision by for example artificial intelligence (AI) [56].

However, RAS still has room for improvement. Some components of the robotic systems used during RAS are not yet utilized to their maximum abilities. For example, the surgeon is not visually assisted or is provided feedback through the digital interface it interacts with during surgery. Subsequently, current robotic systems lack haptic feedback, mainly force sensing and reflecting, which could cause unintentionally large forces to be applied to tissue and could lead to damaging this tissue. Lastly, the laparoscopic nature of RAS results in a limited view of the surgical field compared to open surgery. Key anatomical structures can be difficult to recognize through surrounding tissue and therefore can be a challenge for novice surgeons. Here lie opportunities to increase the safety of robot-assisted surgeries for beginner but also experienced surgeons.

The work that is done during this project focus on the potential of providing feedback about and during procedures through the interface of the robotic systems used in RAS. The most widely used surgical robot is the four-armed da Vinci Xi surgical system from Intuitive Surgical Inc.[4]. In the following section we will introduce the workings of the da Vinci surgical system and introduce 3D models of this surgical robot. After which we will discuss some fields of research working on advancements in RAS aiding the surgeons performance as well as increasing patient safety.

2.2 The da Vinci Surgical System

The da Vinci robotic system is designed to facilitate surgery with the help of robotics using a minimally invasive approach to be controlled by the surgeon from a console. The surgical system is developed by Intuitive Surgical Inc. and was approved by the American Food and Drug Administration (FDA) in 2001 [23]. The main goal of the development of this robotic system was to improve minimally invasive surgery using traditional laparoscopy. This mainly involves increasing the handling facility and dexterity of arm and instrument movement compared to traditional laparoscopy. Furthermore, the da

Vinci surgical system introduces a high quality 3D vision system improving the view of the surgeon during procedures [48]. These improvements over traditional laparoscopy aid in providing a more comfortable surgical experience for the surgeon decreasing overall fatigue of the surgeon during procedures. The components of the da Vinci system that enable these improvements are as follows:

- The remote console of the surgeon (see figure 2.1)
- The side car including the image system and additional components connecting the console to the robot
- The patient trolley including the robotic system holding four robot arms (see figure 2.3

Figure 2.1: Console of the da Vinci Surgical System [23]

The console that is operated by the surgeon is most often placed inside the operation room at a distance from the patient area. This console provides a more comfortable ergonomic position compared to traditional laparoscopy in terms of vision, a seated position and moveability of the arms, hands and fingers [29]. The main software operating the console of the surgeon as well as the other two components is situated in the surgeon's console. The vision of the surgeon is provided by a 3D vision system enabling the surgeon to put their head in a viewing space and see the operating field through the endoscope that is held by one of the three robot arms. The 3D vision system consists of binocular stereoscopes and provides a immersive view of the surgical field. During the procedure, the system is transmitting a 12 mm diameter image provided by the endoscope, which contains two 5 mm cameras each providing an image to an eye providing depth perception [29]. Additional to the 3D vision system, the console includes two master instruments to control the robot arms including the cameras (see figure 2.2. These joysticks are controlled by the surgeon and transmits the arm, wrist and hand movements of the surgeon from a 1:1 scale at the console to a 5:1 scale at the robot arms providing large hand motions to be scaled to micro motions in order to be very effective in small regions in the anatomy of the patient. An additional improvement of transmitting the movements of the surgeon over traditional laparoscopy is the filtering of tremors that occur in the hands which could cause injuries to the patient.

The side car is positioned next to the patient and in close proximity of patient-side staff. The side car includes a monitor displaying a 2D view of the surgical field, an

Figure 2.2: Master instruments of the da Vinci Surgical System [23]

insufflator and a light source [29]. The patient trolley consists of four robotic arms. One arm controls the camera that transmits the surgical field to the surgeon and the other three hold the instruments used to perform the procedure. The arms are equipped with so called endowrists at the ends providing 7 degrees of freedom rotations. The endowrists are joints imitating human wrists, therefore greatly improving dexterity compared to traditional laparoscopy. The position of the endoscope and the execution of several cautery types (electricity or chemicals to burn tissue and remove and close wounds or tissue) are controlled by the surgeon through paddles initiated by pressing them by foot.

2.2.1 Robot Design

The two most used da Vinci systems are the Si and Xi version, where the Si version is the older one. Both models consist of four robot arms (also called manipulators), three holding surgical tools and one holding the endoscope [34]. On the Si version, the arms are attached to the "backbone" or central column of the patient trolley through prismatic and revolute joints, as indicated in Figure 2.3.

The Xi version is the most recent model of the da Vinci System and adds new features and an improved design for the manipulators. The main difference between the Si and the Xi model is that the manipulators are not attached to the backbone of the patient trolley anymore. Instead they are attached to an overhead beam which allows the manipulators to rotate as a group which improves the moveability of the manipulators for more flexible movement when inside the patient. Furthermore, where the endoscope was a fixed arm in the Si version, the manipulators in the Xi are all identical to each other and thus the endoscope can be attached to any of the four manipulators and therefore can provide different perspectives of the surgical field [34].

The manipulators of both models consist of links and active/passive joints that make for their change in position and movement. When docked into the patient, only active joints can be actuated and therefore influence the position and movement of the manipulators inside the patient. Passive/non-actuated joints are not controlled by the surgeon through the console but only serve for setting up the robot before the surgery.

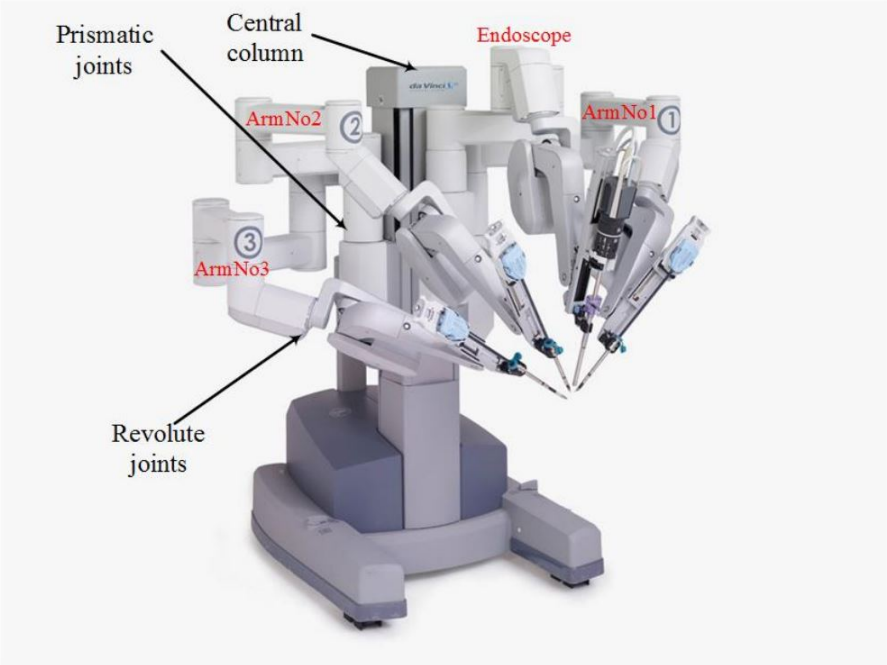

Figure 2.3: The patient trolley of the da Vinci Si Surgical System [34]

The measures of the joints and links of the manipulators are calculated by the kinematic structure of the robot. A kinematic model is a mathematical representation of the physical characteristics of a robot in terms of the joints and links. In the instance of robot-assisted surgery, it specifies the angles and positions the manipulator of the robot can reach when docked into the patient [30].

In order to simulate motions of the robot we need to model the robot and its kinematics. Simulating a model of the da Vinci system enables testing the correctness of the execution of the robot without the access to the actual hardware and software of the system. Intuitive Surgical does not disclose the technical specifications of the actual da Vinci system. Thus, during this research we need a model of the da Vinci robot and a tool to simulate a particular task in order to test monitoring that task. We introduce a model of the da Vinci system in subsection 2.2.2. and introduce the outline of monitoring RAS later on in section 2.6.

2.2.2 Model of the da Vinci Robot

To initiate more research into the use of the da Vinci system and into RAS in general, a community effort in the form of a research platform was established with the help of Intuitive Surgical to tackle the shortcomings of the da Vinci robot. This initiative is called the da Vinci Research Kit (dVRK). Intuitive Surgical facilitates hardware from old da Vinci surgical systems to universities participating in this research platform [22]. Researchers can use the building blocks provided by Intuitive Surgical to build research prototypes of the da Vinci system to test for multiple purposes and research directions. This makes for a challenge setting-up, running and testing the facilitated pieces of hardware from the da Vinci system because the software connecting the pieces is not provided.

Researchers at John Hopkins University (JHU) have developed controllers for establishing a hardware interface between the components of the dVRK. This way, the

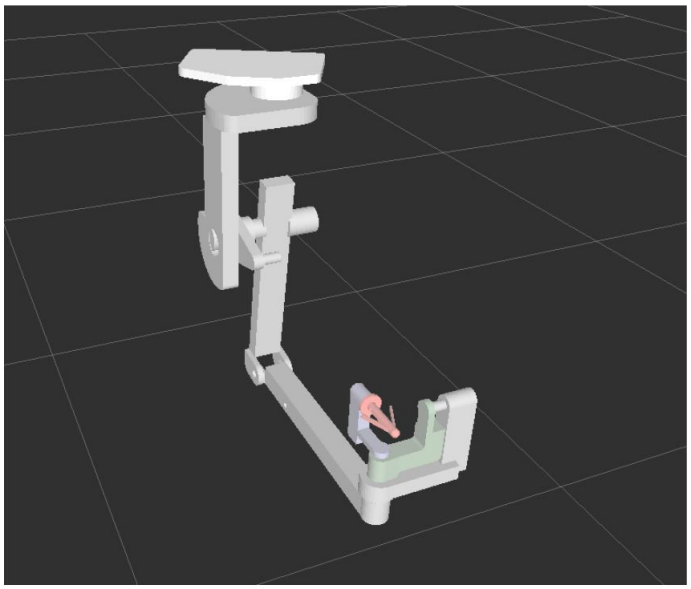

Figure 2.4: Digital 3D model of a component of the da Vinci Si Surgical System [38]

pieces of hardware that are provided with the dVRK can communicate with each other and therefore be used simultaneously. Furthermore, they have developed an open source platform called Surgical Assistant Workstation (SAW), providing a platform to use these controllers and develop applications for different types of medical robots [38].

Researchers at both Worcester Polytechnic Institute (WPI) and JHU have modelled the dVRK to provide models to be used for simulating the workings of the da Vinci robot in a digital environment [38]. This model is not an exact copy of the actual dVRK components. The approach to modelling these components included the disassembling of all dVRK components and manually measuring each component, which is prone to inaccuracies in the measurements. This is a relatively cumbersome approach but necessary as no digital models are provided with the dVRK. Therefore the model relies on mere visual observations and measurements. However, this model of the dVRK is very suitable for testing purposes in simulations. Figure 2.4 shows a model of one of the manipulators provided in the dVRK.

Research by Munawar (2015) extended the control of the dVRK in simulations to be integrated with Robot Operating System (ROS), enabling more capabilities of the dVKR [38]. The motivation behind using ROS is that it is very important to control the dVRK through a platform that is open source and familiar and common to researchers. The Surgical Assistant Workstation (SAW) has been developed solely for the purpose as a platform for controlling the dVRK and requires significant understanding of the platform to experiment and develop with. Therefore building upon the architecture of the dVRK using ROS extends the capabilities that can be used and plays a big role in future research and development of dVRK.

The adoption rate of ROS across the world is increasing rapidly and has become very popular among researchers in the field of robotics [43]. ROS is a framework for building robot applications and allows developers to assemble a complex system by connecting existing solutions. The framework of ROS is easy to use, code can be reused by design and the level of abstraction between the software and hardware allow researchers and developers to adapt quickly to the framework. Furthermore, because of the open source nature of ROS resulting in thousands of available packages to be used, contributes to its

wide spread rapid adoption in the field of robotics. Section 3.5 will go into more detail about ROS.

In this research, we have partially build on the work of Munawar (2015), using their models of the dVRK and its integration into ROS to simulate the da Vinci system in use [38]. Next to the advantages of ROS named above, we will motivate the choice for building on this work further in section 2.8.

2.3 Techniques for Improving Robot-Assisted Surgery

RAS has brought numerous advancements over traditional laparoscopy and open surgery. However, RAS also leaves challenges open from traditional laparoscopy and introduces new challenges. The two main challenges involve the complex system that is needed to be worked with in RAS as well as the limited view of the surgical field that comes with the laparoscopic nature of RAS. In this section, we discuss some techniques that are trying to tackle these challenges.

2.3.1 Computer Vision

A huge amount of data is being produced during RAS and the advancements in computing capabilities over the last decade, like artificial neural networks, are a perfect way to analyze this data to observe surgical care and ultimately improve the safety and effectiveness of RAS. One of the fields that take advantage of these advancements is computer vision. Deep Learning takes advantage of the increase in computing power and enables computer vision applications to improve the results of real-time image recognition tasks. Using these AI-driven models, it is possible to very accurately (depending on the amount of data) recognize things from images and video in real-time.

RAS is a perfect use case that can take advantage of computer vision in order to tackle the challenges that come with it. The challenges that arise in RAS that are related to image analysis mostly concise of limited or obstructed view of the surgical field due to smoke from the cautery tools, blood coverage, blurry vision due to unfocused frames and variation in lighting. Key anatomical structures can be covered by other tissues or organs and only become visible after removing surrounding tissue. Therefore, it would improve the oversight of the surgeon of the surgical field if the anatomical structure could be segmented and annotated and directly imposed on the surgical video feed available to the surgeon during the procedure. Computer vision can be used to address this by using deep learning models to recognize the key anatomical structures that are often hard to recognize by novice surgeons.

Research has showed impressive and promising results in recognizing anatomical structures in the field of MIS using computer vision models based on deep learning [33, 35, 46, 9, 32]. For example, Mascagni et al. (2020) introduce a deep learning model, called DeepCVS, to automatically segment hepatocytic anatomy in laparoscopic cholecystectomy [35]. They used still images from laparoscopic cholecystectomy videos and annotated the key anatomical structures and trained a deep neural network comprising of a segmentation model to highlight these key anatomical structures. Results showed that deep learning models like DeepCVS can be trained to reliably and accurately segment anatomical structures from surgical images. Figure 2.6 shows how these structures are annotated using this deep learning model.

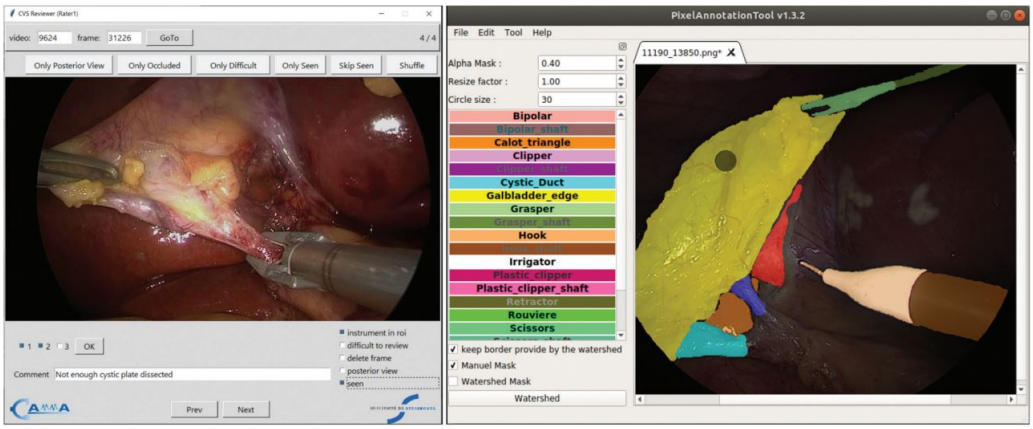

Figure 2.5: Example of DeepCVS: annotation software for segmentation of anatomical structures [35]

2.3.2 Augmented Reality

Another field of research that extends computer vision in terms of improving the field of vision of key anatomical structures in the surgical field during procedures, is augmented reality (AR). We define AR to be a tool to superimpose the digital over the real, meaning enhancing the vision of the real physical world by overlaying it with the use of digital visual elements. In terms of RAS, AR can be used to superimpose computer-generated images and models onto the field of vision of the surgeon during procedures. This way it could help providing 3D models and images onto the surgical field that can be helpful during procedures.

Over the past decade, several studies have developed and proposed techniques to implement AR in for its use during procedures, mainly in hepatic surgery which is a liver resection procedure [12, 41, 19]. For example, the study done by Plantefève et al. (2015) tries to tackle the partial surface view of the liver of the surgeon through the endoscope during the procedure [41]. They developed a biomechanical model to estimate the internal structures of the target organ and used a tracking algorithm to register the correct position of that model with the liver and superimpose it onto the endoscopic video that is perceived by the surgeon. An example of how this would look like is shown in Figure 2.6.

Another way that AR can improve RAS is for training novice surgeons in using the complex robotic systems that come with RAS. Novice surgeons are often supervised when handling the robots used in RAS. AR could help improve the training of novice surgeons by enhancing the visual communication between the instructor and trainee by for example enabling the instructor to be able to point to anatomical structures in the surgical field and virtually demonstrate the use of surgical instruments [24].

The issues that are present in RAS can be tackled by augmented reality and computer vision as we have demonstrated above. But another technique that could be beneficial to robot-assisted surgery in general is rigorously verifying the actions and movements on their correctness to try avoiding any sort of missteps and injuries during and after surgery. The formal method of monitoring a system is called runtime verification (RV) and will be the focus of this project and will subsequently be discussed in section 2.4.

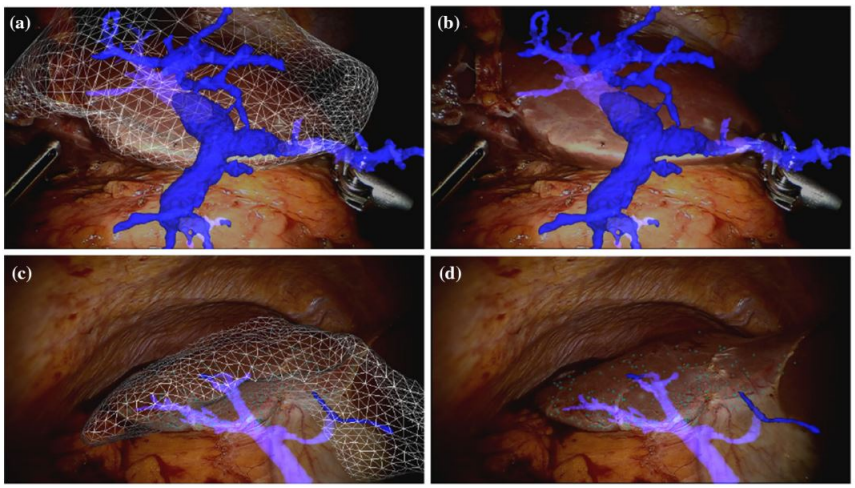

Figure 2.6: Example of Augmented Reality used to superimpose the internal structures of the liver [41]

2.4 Runtime Verification

When designing and developing a system, we want to make sure that the system exhibits the correct behaviour we intended it to have. Observing the correctness of the behaviour of a system by a human is very exhaustive and will take a lot of time. Therefore we want to replace the human observer with a program to monitor the behaviour of this system. One of the ways of doing this is by analysing the dynamic behaviour of systems, may they be software or hardware related. A popular study of methods of analysing the behaviour of systems using formal methods is runtime verification (RV). Such analyses mostly consist of checking if a certain specification holds or is satisfied during a run of the system. There are methods where in the instance of a violation, the run is altered or an action is triggered to rectify this violation. In general, using RV we check the correctness of the behaviour of a system.

When talking about the behaviour of a system in terms of RV, we consider the way it changes over time by going from some state the system is in to the next by performing or taking a certain action, may they be performed by the system itself or externally in their environment. In RV, we describe this behaviour by talking about observations we can make about it. We can either observe the current state the system is in or the performed action or change in its environment, indicating a transition from one state to another.

Before we go deeper into the different components that make for RV, we will introduce a fixed list of RV related terms that will be used for the remainder of this project. This is because terminology is not yet standardized in the field, because it is a very broad field, terms can sometimes focus more on a specific part of the field.

- Monitored system: The system of which its behaviour is being monitored. This can be software, hardware or a combination like cyber-physical system (CPS).
- Events: An event can be any kind of observation about the system. They may vary from simple cases like a traffic light that turns green to something more complex like an event containing data values like the movement speed of a component of a robotic arm. Therefore it can capture many levels of abstraction like for example both internal as well as external behaviour of the system.

- Traces: A sequence of events that capture a behaviour of the monitored system we are interested in to analyse. The extraction of these traces is done by the instrumentation part of the RV system. The order of the events in the sequence can create a qualitative notion of time.
- Properties: Can be described as a set of traces extracted from the monitored system.
- Specifications: Describe properties through a dedicated formalism. Therefore, one property can be described by multiple specifications.
- Monitor: Checks the properties of interest on violations during runtime alongside the monitored system. Observations about the system are received from the instrumentation in the form of events and traces to analyse.
- Instrumentation: The mechanism that extracts events, traces and signals from the monitored system during its execution and communicates these to the monitor.
- RV system: Encompasses all components for checking properties of the monitored system, namely the monitor, the instrumentation and the monitored system. We talk about the framework of the RV system when targeting the specification language, the monitoring algorithms and instrumentation techniques used in the RV system.
- RV tool: A RV framework designed for a particular application (domain), consisting of a often modified specification language, suitable monitoring algorithms and dedicated instrumentation techniques.

There are three main things to be taken into account and which are needed to analyse behaviour of a system using RV. What is desired (specifications), the component that monitors the behaviour and what instrumentation is needed to extract the necessary info for monitoring.

2.4.1 Specifications

The main important thing to specify in a RV framework is what we want to check and validate about the behaviour of the system we are monitoring. Specifications are formulated before monitoring runs of a system and describes key behaviours of the system within the abstraction of its environment.

A specification is a description, written in a certain specification language, of a property, which in turn is an abstract description of a set of traces of a system [5]. A specification can capture many properties. Therefore, a specification describes a key behaviour of a specific system in a dedicated application domain in the form of one or more properties.

A problem with properties is that most specification languages are sometimes not expressive enough to capture properties for a particular application. This is why in the field of RV, a lot of research is being done on designing the right specifications language for specific application domains. Sometimes they consist of little syntactic or semantic changes of conventional languages. This makes designing specifications for properties in a dedicated field of application a difficult task, especially for laymen in the field of logic.

2.4.2 Monitor

One of the key components of a RV system is the monitor. The monitored system that executes a certain task or tasks cannot guarantee the correctness of its behaviour. Therefore we implement a monitor alongside the system to observe and check the correctness of its behaviour. An online monitor checks the behaviour of a monitored system during runtime on satisfaction and violations of the formal specifications that are described beforehand. An offline monitor checks the execution trace on violations of properties after its execution.

When detecting a violation, the monitor often communicates its verdict to the user or a supervising software component [5]. When we are dealing with an online monitor, the monitor can also interfere with the execution trace of the system to rectify the violation that has been made. However, in some application domains, active interventions by the RV system are not desired due to certain constraints (this will be discussed further in section 2.5.1). When we are dealing with an offline monitor, the monitor can only prevent violations by the system in future runs of the execution as it is not checking the monitored system during runtime.

Monitors can often automatically be generated from the prescribed specifications it is supposed to monitor. This is possible when there is an equivalent automaton for the formal language the specification is specified in. For example, specifications written as a linear temporal logic formula can be transformed into a Büchi automaton [47]. Here, an open-source library containing translations from Büchi automaton to linear temporal logic is leveraged to construct a tool for automatically generating monitors.

2.4.3 Instrumentation

The instrumentation is the backbone of the RV system and connects the monitored system with the monitor, enabling the monitor to observe and analyse the execution traces of the system. The foremost thing the instrumentation concerns itself with is determining what part of the systems execution traces and events are visible to the monitor. Only providing the relevant information to the monitor for analysis helps with the computational load and minimizing overhead on the monitor. The instrumentation can also extract previously unobservable aspects of the behaviour of the system which could help with monitorability of the system [5].

Instrumentation can also control how the monitor executes in relation to the system. When dealing with an offline monitor there is only the choice of running the monitor after the execution of a run of a system. However, when dealing with online monitoring, the monitor and the system can execute in a synchronous or asynchronous matter. Asynchronous monitoring is when the monitor and the monitored system do not analyse or execute in sync [15]. An example of this is when the monitored system stops executing after a certain amount of the execution is observed by the monitor and only continues after it is analysed and checked.

Instrumentation can also be software embedded as well as hardware embedded depending on the monitored system we are working with. When we are monitoring a hardware system, the instrumentation will require physical wiring between the monitor component and the sensors that can provide signals to the monitor. When dealing with a software, the instrumentation will mostly comprise of software in the same programming language as the monitored system [5].

2.5 General Approaches to RV

Implementing a RV system can take on many forms and a lot of different approaches can be used depending on the application domain it is used in. For example, it depends on if it is preferable that the RV system initiates a change in execution or if there is a need for the monitor to analyse during runtime in parallel with the monitored system. We will discuss the different approaches and techniques in the following subsections.

First we will give a quick overview of the classical approach to RV, as represented in Figure 2.7.

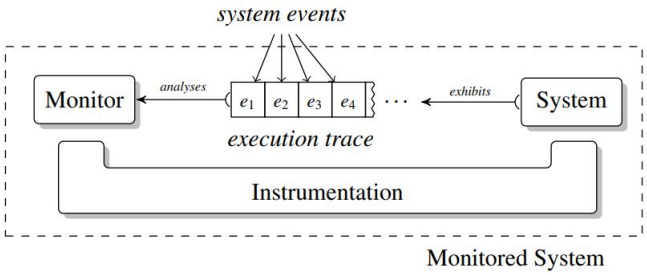

Figure 2.7: The Basic RV Monitoring Setup [5]

This classical approach by Bartocci et al. (2018), depicts the three main components of our described RV system and how they are connected to each other [5]. The system exhibits a certain type of behaviour which is captured as system events and traces with the help of the overarching instrumentation. These execution traces and system events are subsequently analysed by the monitor on correctness of the intended behaviour by the system.

However, this classical and basic approach knows a lot of expansions to allow its implementation into many different applications. We will first discuss a couple of components or actions that can modify the classical approach to RV systems. Furthermore, we will discuss the most often used specification languages for RV system implementations. Lastly, we will discuss examples of different and more dedicated RV approaches.

2.5.1 Level of Invasiveness

Implementing a RV system can serve for multiple purposes in relation to a system [15]. The least invasive functioning of a RV system is letting it only collect information about the behaviour of the system. Capturing things like events and traces from the behaviour of the system and extracting metrics like execution time fall under this category.

Another approach to the functioning of a RV system is performing analyses on the information gathered from the behaviour of the system. This approach has the goal to provide for example safety and security to the interaction of the system with its environment by verifying the correctness of the system its behaviour.

Lastly, RV systems can also be applied to detect issues and faults and fix them [15]. We consider this the highest level of invasiveness as it actively modifies the execution of the monitored systems instead of more passively observing and analysing. An example of an action at the highest level of invasiveness would be to disable a certain action that would violate a particular property.

2.5.2 Offline and Online - Synchronous or Asynchronous

In RV, the time that the monitored system is monitored and analysed can vary and depends on the constraint of the system that is monitored. We say a system is being monitored offline when the analysis of the behaviour of the system is being done after the system execution. This might be preferred when the system does not suffer from low overheads or the monitor needing to be less intrusive.

We say a system is online when the monitored system is being monitored during its execution. The biggest advantage over offline monitoring is that violations of properties are caught during the execution of the system and there is a possibility to rectify that violation of stop the execution entirely. Also, monitoring parts of the execution of the system is helpful when dealing with low overheads requirements.

Carrying out online monitoring in an incremental fashion can be done in a more synchronous way or a more asynchronous way. Here, the notion of synchronous is associated with the system waiting for the verdict of the monitor after each event before proceeding with the systems execution. Conversely, the notion of asynchronous is associated with the system not being dependent on the finishing of the event analysis by the monitor. This approach is less intrusive and make for lower overhead, however at the cost of the possibility of verdicts indicating a violation coming in too late to rectify them [10].

2.5.3 Specification Languages

Temporal Logic: Different approaches to RV systems often begin with different specification languages. The specification language that is used in RV systems is one of the main components and contributes heavily on how and what about the behaviour of the system is needed to be analysed and monitored.

Linear Temporal Logic (LTL), introduced by Pnueli in 1977, was created for the use of formal verification of computer programs [42]. Fast-forwarding a couple of decennia in the future and it has become most popular temporal logic to use in the field of RV. This particular type of logic thrives in its application to formal verification because of its additional temporal operators. These temporal operators make for ability to specify in the future if something should or should not happen. We will quickly present the formal definition of LTL:

Given a set of atomic propositions AP, the set of LTL formulas over AP is defined inductively as follows:

- True and false are LTL formulas
- If p $\in$ AP, then p is an LTL formula
- If ϕ and ψ are LTL formulas, then $\neg\phi$, $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \equiv \psi$, Xϕ, Gϕ, Fϕ and ϕUψ are LTL formulas

In addition to the propositional logic operators, we have the X, G, F and U temporal operators. We say Xϕ means that ϕ should hold in the next state, Fϕ that ϕ should hold at some point in the future, and Gϕ that ϕ should always hold, i.e. in the current state and in the future. Lastly, U corresponds to until and ϕUψ says that ϕ should hold until ψ becomes true.

However, the infinite nature and unfinished traces that can encompass LTL, is not suitable for RV because it concerns with finite execution traces. Therefore, research has been done into adapting LTL and adding finite trace semantics to fit the constraints of RV. One approach to this constraint is the inclusion of the multi-valued verdict domain

and is captured in LTL3 [6]. In this adaption of classical LTL, there is an added notion of impartiality (denoted with the ? verdict). Therefore, in LTL3 there can exist some formulae that can never be satisfied or violated and therefore give an inconclusive verdict on the specification at hand. In this case, we can look at a finite trace to be a finite prefix of some infinite trace.

Regular Expression: A similar and also declarative specification language that is used in RV approaches is regular expression. Specifying system requirements in this language is quite simple and intuitive which makes for clear requirement specifications with little error. Extensions of this language with for example the notion of time have not received much attention in the field of RV [1].

State Machines: Another specification language that is widely used and popular in RV is the use of state machines (in particular finite state machines). In state machines we describe the events of the system and visualize them by labelling transitions from one event to the next. In this way, the execution trace can be visualized as a sequence of transitions. State machines have an advantage over declarative specification languages like temporal logic, of being directly executable whereas temporal logic requires the synthesis of an executable monitor.

Hybrid Systems: Depending on the application of the RV system, a combination of several specification languages could be best for some particular implementation. So some RV approaches consider the use of multiple different formalisms to come to a hybrid best suitable for a particular application. Here, researchers make compromises between the advantages and disadvantages of languages like temporal logic, regular expression, finite state machines, etc.

2.6 RV in Continuous Systems

Going deeper into the implementation of the RV in a variety of systems, the type of behaviour the monitored system exhibits greatly influences the type RV approach that has to be used. The behaviour of the monitored system can be discrete, in the sense that its behaviour is divided into distinct events, organised in clear steps. Examples of such systems are often related to the digital computer domain as all digital computers operate using discrete math since they use a finite, often fixed, number of bits to represent numbers. However, examples can also be found in the real world. For example, traffic lights can be modelled as a finite set of states being either off, red, yellow or green light. Another example is a two-way light switch, either being switched on or off.

The behaviour of the monitored system can also be continuous in nature, involving fluctuating numbers and varying data values measured over a specific time interval. Modelling behaviour of physical systems often presents such continuous behaviour and is typically done through representation of differential equations (for example in the form of a kinematic model). This continuous behaviour can be considered as signals and trajectories [5]. Examples of systems that exhibit behaviour that can be captured as signals or trajectories are related to human condition monitoring, like heart rate, respiratory rate, etc. But also from signals processed through audio for finding useful information.

2.6.1 Cyber-Physical Systems

A prime example of systems that behave in a continuous matter are cyber-physical systems (CPS). Cyber-physical systems are systems consisting of a digital and physical component to control the physical environment around the system [5]. Its continuous

dynamic behaviour is often interleaved with discrete instances of system events. We can conceptually capture the systems behaviour through continuous and discrete variables that vary in value over time during its execution trace and occurrences of events. However, when dealing with complex and/or many CPSs, there is too much information about the systems behaviour to be efficiently and correctly evaluated. We use RV systems to extract the relevant information into system events and execution traces for monitoring requirements and performance indices of these systems. When dealing with safety-critical applications like preventing the crash of an airplane, it is needed to monitor system requirements to check if for example engines in the airplane function correctly. While in less safety-critical instances we would want to monitor the performance of a system by evaluating some system metrics over a certain time period, like queue time or energy consumption.

Designing formalisms and monitoring algorithms for continuous dynamical systems like CPSs is a difficult task, as such systems are the subject of study in a broad spectrum of application domains and tend to change and evolve constantly. These domains have developed a variety of ways to measure and evaluate signals and time series and detect occurring patterns over the years. The challenge in monitoring CPSs is to integrate these measures with those provided by the newly developed formalisms for runtime verification which are more suitable for capturing sequential aspects of behaviours in continuous time.

2.6.2 Specifications Languages for CPS

As mentioned before, CPSs do not exhibit behaviour that can be seen as a trace of discrete sequential events but exhibit behaviour more similar to continuous dynamically changing variables that can be seen as collections of signals. Standard propositional linear temporal logic may not be sufficient to capture the continuous signals that capture the behaviour of CPSs. Therefore, during the last two decades, research has been done to extend LTL for also capturing signal predicates on numerical values in dense instead of discrete time.

The constraint of reasoning over continuous and real time has been present and described in Metric Temporal Logic (MTL). However, this language lacks the numerical predicates that are needed to capture the behaviour of CPSs represented by numerous differential equations.

This extension of linear temporal logic with signal predicates in continuous time is called Signal Temporal Logic (STL). It extends the semantic domain and logic of MTL to real-valued signals but also quantitative form of STL adds the feature that gives an answer to how far away the value of a predicate is from a violation or satisfaction [11]. This type of temporal logics has been suggested to use and implemented in application domains ranging from robotics to biomedical to automotive systems, etc. [45].

Distributed networks of such CPSs have spatial requirements due to their topological placement in relation to each other inside a particular environment like in the Internet-of-Things domain or involving smart grids. One proposed extension of STL has been designed to tackle this constraint in the form of Signal Spatio-Temporal Logic (SSTL), adding the feature of expressing spatio-temporal requirements to be used in application domains where the environment contains multiple CPSs that need to be connected to each other [40].

2.6.3 Monitoring of CPSs

Medical devices and robots also fall under the term CPSs [45]. In the scope of this research, we can therefore identify the da Vinci surgical system as a CPS. Motions and movements by the da Vinci system will be simulated and subsequently monitored as part of the goal of this research. The controlling of the robot in these simulations will be done through ROS. Recently, Ferrando et al. (2020) have designed a RV framework for monitoring models controlled through ROS, called ROSMonitoring [17]. They have developed an accompanying tool that can be integrated with ROS to actually monitor model of CPSs during simulations.

The ROSMonitoring framework consist of three components, the instrumentation, the oracle and the monitor, which resemble the classical main components of traditional RV systems and frameworks. The instrumentation handles the creation and placement of monitors in the ROS environment to intercept messages on relevant topics that we want to check properties against. The oracle is the component that is used for checking a particular observed event conforms to some formal specification or not. One of the specific oracles that are included with ROSMonitoring is the Reelay Oracle. Reelay is used to construct runtime monitors for checking temporal behaviours of systems by using state-of-the-art RV techniques like the previous mentioned LTL, MTL and STL. A more elaborate description of ROSMonitoring will be outlined in section 3.5.

2.7 Case Study: Robot-Assisted Minimally Invasive Esophagectomy

In order to analyse the possible implementation of RV in robot-assisted surgery, we will present a case study derived from the field of robot-assisted surgery, depicting a complex surgical procedure executed with the help of the da Vinci surgical system, namely Robot-Assisted Minimally Invasive Esophagectomy (RAMIE).

The esophagus is a muscular tube, which allows food to travel from your mouth to your stomach. The surgery in which a part of the esophagus is removed as a result from a disease affecting the esophagus, is called a esophagectomy. This procedure consists of a abdominal, thoracic and cervical phase, depending on the surgical technique that is used. The surgical technique depends on the location of the tumour and the comorbidity of the patient. These procedures are most often oncological surgeries and are the standard curative treatment. The need for this operation is a result of growths on or near the esophagus and the enclosing lymph nodes affecting the health of the patient and causing problems. Cancer is a common cause of these growths but not in every instance.

In this procedure, the esophagus and surrounding lymph nodes are removed. In the abdominal phase of the procedure, a gastric conduit is created from the stomach and lymph nodes are dissected. In the thoracic phase, the esophagus is prepared by freeing it from surrounding tissue and lymph nodes which are also dissected here. During the intrathoracic and cervical phase an anastomosis is created depending on the location of the tumour.

Despite traditional surgical advancements, esophagectomy remains a high-risk operation, with a 60% morbidity and a 4.5% mortality and, among all the complications, anastomotic leak remains one of the most feared with an incidence of about 10% [28]. Therefore, the advancements in robot-assisted surgery has gained popularity in the order to tackle issues that arise in traditional esophagectomy. Robot-assisted minimally invasive esophagectomy (RAMIE) has shown to be an excellent improvement over

esophagectomies done by traditional laparoscopy in MIS or open surgery in terms of intraoperative blood loss, postoperative complications, short-term quality of life, and functional recovery. Evidently, a study done by European researchers on 100 patients comparing conventional MIS and RAMIE observed a reduction in the post operative hospital stay when the RAMIE procedure was used [51].

The most widely used surgical robot for RAMIE is the four-armed da Vinci Xi system from Intuitive Surgical Inc. Using the da Vinci robot improves dexterity by four robot-arms with hand-wrist articulation of instruments with 540 degrees rotation and 7 degrees of freedom, tremor filtering and three-dimensional magnified vision system that is controlled by a stable robot-arm and steered by the operating surgeon instead of an assisting surgeon. These advancements facilitate, among other, an increase in image quality, depth perception, overall comfort, surgical motion precision and complex manoeuvring during suturing, reconstruction techniques and lymph node dissection compared to conventional laparoscopy [55].

However, RAMIE remains a very complex procedure in terms of working with the highly technical da Vinci system as well as the complex surgical phases to go through to correctly execute the procedure. One of the obstacles of RAMIE is the learning curve of beginner surgeons. It is indicated that the learning curve of the RAMIE procedure is indicated to be between 24 and 70 cases. Furthermore, numerous variations on the application of RAS in the thoracic, abdominal phases and anastomotic technique exist worldwide [44]. Therefore, it is necessary to form a standardized approach for the RAMIE procedure to aid in lowering the learning curve of novice surgeons when it comes to the RAMIE procedure. One team of surgeons has published their approach to standardizing the thoracic dissection in RAMIE [26].

For the sake of consistency during this project, we will introduce the standardized approach published by Kingma et al. (2020) for describing the procedure of RAMIE [26].

2.7.1 Step-Wise Approach

The goal of the RAMIE procedure is to remove the tumour of the esophagus and the surrounding lymph nodes and to create an anastomosis between the healthy esophagus and the stomach. This procedure can consist of an abdominal phase, a thoracic phase, and a cervical phase, depending on tumour location and patient comorbidities. In this proposal we focus on the thoracic phase, because the anatomy is relatively fixated and there is a close relationship between important anatomical structures.

A step-wise approach to thoracic dissection is represented in Figure 2.8. First, the inferior pulmonary ligament, a ligament related to the lung, is dissected followed by the pericardial sac. Particular care is taken to avoid injuring both the pulmonary vein and right main bronchus during this phase. The parietal pleura is then dissected along the inferior border to the arch of the azygos vein before being dissected free from the vein itself. The pleura is then incised up from the arch.

Subsequently, the superior mediastinum is dissected. This begins with the incision of the parietal pleura along the posterior border of the superior vena cava (SVC). Afterwards, the dissection plane is continued towards the left posterior side, immediately inferior to the right subclavian vein to the level of the superior intercostal vein. Here we need to be cautious not to damage the left subclavian artery which is running from the aortic arch along the left side of the trachea.

Switching over to the right side, the right vagal nerve is then dissected free and lateralized from both the incised pleura and the surrounding tissue. Doing this, while

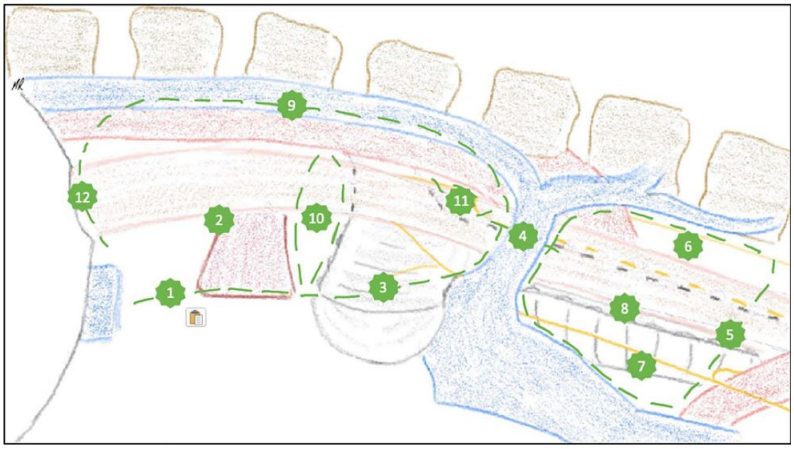

Figure 2.8: A schematic representation of a standardized approach for RAMIE. 1, mobilization of the inferior pulmonary ligament; 2, pericardial dissection; 3, right pleural dissection; 4, division of the arch of the azygous vein; 5, superior mediastinal pleural dissection; 6, right paratracheal dissection; 7, left recurrent laryngeal nerve dissection; 8, tracheesophageal dissection; 9, para-aortic and mesesophageal dissection; 10, subcarinal lymph node dissection; 11, AP lymph node dissection; 12, hiatal dissection. [26]

dissecting close to the subclavian artery, we need to take care to avoid damaging the right recurrent laryngeal nerve. The tissue between the posterior border of the superior caval vein and the right lateral border of the trachea is then dissected. Afterwards, a dissection plane is created between the ventral surface and the membrane of the trachea. When arriving at the left border of the trachea, we identify the left recurrent laryngeal nerve and be cautious not to damage it. Dissection then continues to the point where the trachea divides into to bronchi. Here we arrive back at the right vagus nerve which is then divided. Dissection is continued and when we encounter the mediastinal pleura, it is incised over the length of the azygos vein. This incision ends approximately 5 cm cranial before reaching the esophageal hiatus. After the dissection of the mediastinal pleura, we detach both the azygos vein and the aorta by dividing them.

We continue dissecting until the left sided visceral pleura is laid out. Afterwards, the membrane of the esophagus is then dissected and preserved. At this point, the thoracic duct, is divided between 2 large clips. The esophagus is then retracted dorsally after which the dissection progresses cranially. In order to dissect the lymph node bundle, its enveloping fascia is first released inferiorly. It is then released along the inferior border of the right bronchus before proceeding to the inferior border of the left bronchus. The lymph nodes are then dissected free. To complete the lymphadenectomy, the aorto-pulmonary lymph nodes are also dissected. Here we again need to be cautious to avoid damaging the left recurrent laryngeal nerve originating from under the arch of aorta.

In the last phase, the enfolded fascia is released from the lateral border of the trachea as well as the superior border of the left bronchus. Here we encounter vessels that supply both the esophagus and bronchus with blood. Here, esophageal vessels are dissected while bronchial vessel are preserved. The thoracic dissection is then completed by incising the pleura at the level of the hiatus in order to join both thoracic and abdominal cavities.

2.7.2 Properties to Monitor

The goal of describing this use case it to identify and describe relevant and specific properties that need to be monitored as a result of otherwise dangerous situation during the RAMIE procedure. We describe three types of properties within the RAMIE procedure that are important to be monitored in order to obtain a correct execution of the procedure while minimizing post operative complications or injuries.

The first kind of properties that could be checked is avoiding critical anatomical structures where possible. During the procedure, a lot of critical anatomical structures are exposed from their surrounding tissue that also serve as a protective layer. During the procedure lymph nodes are removed around the anatomical structures to extract cancer cells as well as to get a clear visual of the diseased anatomical structure parts that need to be removed. Critical anatomical structures like the aorta, azygos vein, trachea, and laryngeal recurrent nerve are easily damaged which can lead to serious (post operative) complications and injuries, some even leading to mortality. Therefore, it is needed to keep away from critical anatomical structures when possible during a particular phase in the procedure. An example of a property could be that during a particular phase in the surgery, the robot arms need to keep a certain amount of distance (a safety distance threshold) to critical anatomical structures that are not involved during that phase of the procedure. In this way, the critical anatomical structures that are not involved during this phase have an additional protection against accidental damaging. An example of an English requirement of this property could be:

The distance between the robot arms and critical anatomical structures should never drop below the safety distance threshold.

Another kind of property that is interesting to be monitored is the motions and movements a surgeon makes during the procedure. Hasty or hesitant movements and motions can lead to scraping or touching critical anatomical structures. When a surgeon exhibits this kind of behaviour, alerting them could lead to them reflecting on their execution and slow down and be more precise if necessary. A way to monitor this would be to measure the average movement speed over a series of RAMIE procedures and taking that average as a reference point when checking if the movement speed of a surgeon is too high. We can once more give an example of an English requirement of this property:

There should not be a quick pause in the course of taking an action/movement, while controlling the robot arms.

Lastly, as we introduced above, correctly completing surgical phases before going on to the next is crucial for the correct execution of the RAMIE procedure. Monitoring the full completion of surgical phases can be helpful to surgeons as a memory aid during the procedure. Therefore the chance of missing a step in a surgical phase is decreased.

2.8 Discussion: Suitable RV Approach for Robot-Assisted Surgery

The purpose of this literature review is to outline the state-of-the-art in RAS and identify the possible improvements that can be made to RAS. Here we have focused on the most popular robotic system that is used in operation rooms across the world when it comes to RAS, the da Vinci surgical system. Review of literature has identified that the biggest improvements that can be made involve the digital interface that is introduced by RAS. Little advantage is taken from the digital interface between the surgeon and the patient

in terms of visual assistance through digital annotations or feedback on the execution of the procedure.

Therefore, we have extensively discussed the da Vinci system and its design in order to get a better view on the improvements that could be implemented in the surgical system to help and assist the surgeon during procedures. Most of the research done on improving RAS involves the segmentation of anatomical structure to improve and help the surgeons orientation within the patient by the means of computer vision and augmented reality [41, 35]. However, little research has been done on monitoring of movements and actions of the robotic arms controlled by the surgeon [36].

For that reason, we suggest the use of RV for improving RAS by the means of monitoring the motions and actions of the robotic arms during RAS procedures. We have therefore introduced the notion of RV and discussed general approaches to state-of-the-art RV. However, we need to identify a suitable RV approach to be used for RAS as not every approach or tool can be used to specify and formalize relevant properties for RAS. To identify a RV approach for RAS, we go over the components of RV systems and motivate suitable options for implementation in RAS.

As we have mentioned in section 2.6, we can classify medical devices and robots as a form of cyber-physical systems. We can therefore conclude that we can represent and treat the da Vinci surgical system as a CPS. The behaviour of CPSs can be represented as a collection of metrics or signals over a continuous sequence over time [11]. These metrics and signals correspond to the kinematic model of the robot and change according to the movements and motions these systems exhibit.

Subsequently, when monitoring RAS it would be preferable to receive warnings of violations in real-time to avoid damaging and injuring the patient during the procedure. Additionally, waiting for the monitor to analyse motions and actions and come with a verdict is not preferable. Therefore, the RV system that will be constructed will need to be of an online and synchronous nature, as described in section 2.5.2. Another component that needs to be taken into account is the level of invasiveness of the applied RV system. As the surgeon using the da Vinci system needs to know when a property is violated, it needs to give off a visual warning to the digital interface of the system. Furthermore, because we are dealing with a highly safety critical environment because of the inclusion of a patient, intervention by the system when violating a property would be a viable approach. An example of this applied to RAS could be disabling the movement of the robotic arm when getting too close to a safety critical anatomical structure and would help avoiding damaging the structure. However, from a surgical point of view, surgeons prefer to leave the final decision of an action or motion by themselves. Therefore, in this research, we address the RV system without intervention to represent the effectiveness of the approach for RAS from the perspective of the surgeon. Having identified the nature of the system that we need to monitor, we need to identify a suitable specification language for formalizing properties.

Literature on specification languages for CPSs indicate STL to be the most often used specification language for its application in CPSs like robotics and autonomous systems [45]. This specification language is suitable for monitoring the da Vinci system as it can formalize real-values, adding the feature to represent the kinematic models of CPSs and therefore monitor motions and movements of these systems [12]. Therefore, using STL to formalize properties for the RAMIE procedure, we can specify a preferred type of motions of maintain a particular distance between the robotic arms and anatomical structures. Giving warnings of violations of these properties could potentially help the surgeon during the procedure and improve patient safety. A useful feature of STL is

that it can give an answer to how far away the value of a predicate is from a violation or satisfaction [11]. In this way novice surgeons could be warned when getting too close to critical anatomical structures before violating the actual property. This could potentially help decreasing the learning curve of RAS.

Testing the formalized properties is essential for correctly monitoring CPSs. However, the case study we are dealing with introduces some issues in terms of accessibility to hardware and software of the actual da Vinci system. Therefore, an alternative to these issues is using data obtained from the dVRK, as mentioned in section 2.2.2. Additionally, developing a fully operating RV system that can be used in real-life RAS is out of the scope of this research due to the complexity and accessibility of all components that are involved. Therefore, during this research, we have designed a proof-of-concept of a RV system for RAS and test it with the use of digital 3D model of the da Vinci system and a simplified 3D model of the anatomy of the patient. As found in literature, a 3D model of the dVRK and its implementation in ROS and a simulation environment has been openly provided by researchers from WPI [38]. Therefore, for testing purposes, we have used a digital simulation environment to visualize the models in question and use ROS to control the motions and actions of the da Vinci robot to simulate RAMIE procedure based scenarios. An existing RV framework, called ROSMonitoring and which is specifically designed to be used for and integrated with ROS, has been used to test the correctness of the formalized properties in a simulated environment [17]. In the following chapters we will discuss the process of constructing this proof-of-concept RV system and demonstrate the workings of said system.

3 Research Methodology

For filling the knowledge gaps found in the literature, it is imperative to follow a clear and structured research process to answer the research questions. For that reason, this chapter will elaborate on the method we used during this research to develop and integrate systems and models needed. Firstly, Section 3.1 will clarify and explain the methodological approach that we have taken for the research. Thereafter, Section 3.2 will describe the 3D simulation environment that was used to mimic components of the real da Vinci surgical system. Section 3.3 will describe the system used to operated and control the model of the da Vinci system, namely ROS, and explain its architecture. Furthermore, Section 3.4 will give a detailed description of the models developed and used for this research for simulations. Lastly, section 3.5 describes the ROSMonitoring framework used to monitor the motions simulated in AMBF.

3.1 Methodological Approach

The goal of this research is to improve perioperative outcomes and increase patient safety, by constructing a proof-of-concept of a fitting monitoring system for verifying the surgical actions performed in RAS during runtime. Three methodological approaches have been considered for adoption of this research but two of them had too much constraints for potentially coming to a significant scientific contribution. We will go over them accordingly:

The first methodological approach is directly implementing the RV based monitoring system in the software of the da Vinci surgical system. Subsequently, the RV system would leverage the kinematic data about the da Vinci surgical system to reason about the correctness of the behaviour of the robot during surgery. Furthermore, it would ideally leverage annotated 3D anatomical structures based on CT scans, in order to get a clear view and abstraction of its environment to help reason about the other components within this environment. This way, the RV system can reason based on how the anatomical structures are laid out, which is evidently different per patient. However, this approach could not be adopted due to the fact that no access is granted to researchers (outside Intuitive Surgical and a very small group of participating universities) regarding any software technicalities of the da Vinci surgical system. Furthermore, if access would have been granted, privacy concerns related to real patients would prevent us experimenting with data obtained during patient surgery. as well as prevent us from experimenting with the workings of the RV system during surgery on real patients. Therefore, we needed to reduce the scope of this research to focusing on working with synthetic models and data.

The second approach that was considered was therefore focused on using SimNow, a skill-building simulation and integrated learning that is included with the da Vinci surgical system. Novice surgeons can use this simulator to practice certain tasks before performing them in real surgery. Using a simulator specifically designed to work with,

and based on the software components of the real da Vinci surgical system would provide the most efficient baseline for implementing a RV system specifically for the use in RAS. The future integration of a RV system optimised for the SimNow simulator would subsequently be easier to integrate in the da Vinci surgical system than a RV system optimised and integrated into a third-party simulator system. Such a RV system would be integrated into the SimNow simulator as a plugin leveraging the kinematic data about the models used in the simulator to reason about the correctness of their behaviour. However, a similar constraint as in the first approach prevented us from adopting this approach. Again, no access could be granted to the internal workings of the SimNow simulator because of it being developed by Intuitive Surgical. Only novice surgeons that are officially enrolled into a training program for using the da Vinci surgical may use the training simulator, which already ruled us out from trying out the simulator.

The methodological approach we ended up adopting is based on using a third-party simulator and an open-source robot operating system. The main reason for this choice is because after the difficulties and constraints the first two approaches showed, we opted for components that are of an open source nature and therefore can easily be modified and are accessible in order to implement a RV component for monitoring. As mentioned in section 2.2, the constraints that are visible in the first two approaches are widely known and researchers have been trying to work around them for some years now. Intuitive Surgical encourages these initiatives with the dVRK Intuitive foundation. This research will add to research done based on the dVRK, with the goal to advance the field of RAS with the help of the dVRK.

In order to clearly describe and elaborate on the setup and the components used for this methodological approach, we have presented an overview of the components in figure 3.1. Subsequently, we will briefly describe the use of the components starting with the environment all the components are integrated in.

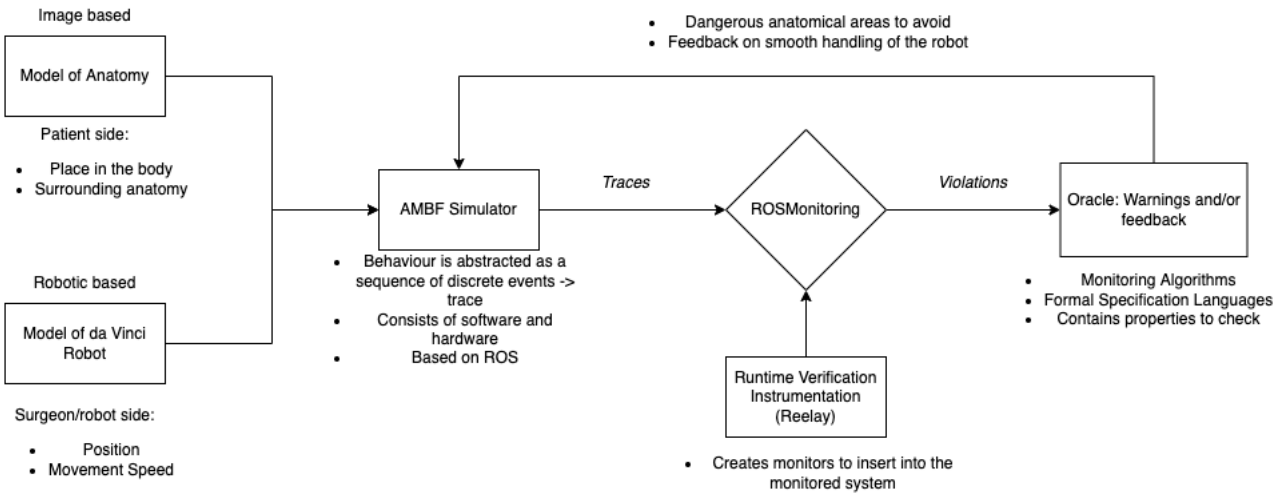

Figure 3.1: A schematic representation of the RV monitor and simulation components.

Asynchronous multi-body framework (AMBF) Simulator: To mimic the environment of the operation room we would be dealing with in real-life surgery, we use a simulation environment called the AMBF simulator. In this environment, we can import digital models of real life systems or objects to simulate motions and behaviour. The AMBF simulator is also included with digital models based on the da Vinci surgical system components included with the dVRK designed by researchers at the JHU and WPI [38].

Section 3.2 will give an elaborate description of the AMBF simulator and its components.

Robot Operating System (ROS): For controlling the digital models of the dVRK, the AMBF simulator is developed based on the Robot Operating System (ROS). ROS is a framework for building robot applications and allows developers to assemble a complex system by connecting existing solutions. The motivation behind using ROS is that it is very important to control the dVRK through a platform that is open source and familiar and common to researchers to encourage future research. Section 3.3 gives a detailed description of the ROS architecture and how it communicates with the digital models that are desired to be controlled.

Simplified model of the patient: Subsequently, we need a model of the patient to complete the models needed to mimic the components present during RAS. It is evident that the model of the thorax that is worked on during the RAMIE procedure is highly complex. Therefore we have used a simplified model of the thorax during this research with the goal to firstly integrate all the components, starting with more simplistic version of them, and importing these components into the simulation environment and make them function correctly. Subsequently, we have then laid a solid base for iteratively improving and making the models more complex to eventually get as close to real-life RAS as possible. How this simplistic digital model of the thorax is designed and constructed will be elaborated on in section 3.4.

Python Client for controlling the da Vinci Robot: In real-life RAS, the surgeon controls the da Vinci surgical system from behind a console using the manipulator instruments. We replace the surgeon for controlling the arms of the da Vinci surgical system in this case by using Python code. With this code, we can set the rotation of the joints of the da Vinci arms in order to create movements in specific directions, therefore being able to control their motions. Subsequently, we can leverage these capabilities to simulate examples of scenarios happening during real-life RAS. Further explanation of how the joint of the da Vinci arms rotate and how they are controlled is presented in section 3.4.

ROSMonitoring for implementing RV system: For implementing a RV system for monitoring the correctness of the behaviour of the da Vinci robot, we use a RV framework designed specifically to be used with the ROS architecture, called ROSMonitoring. In short, this framework leverages the communication system of ROS to intercept and check the movement commands given to the da Vinci robot an return a verdict about the behaviour exhibited based on these commands. This way it can give off warnings when some undesired behaviour is taking place. This way such behaviour could potentially be prevented and remove the risk of collision leading to damage. Section 3.5 will expand on the ROSMonitoring framework, its integration in the AMBF simulator and the ROS architecture and how the properties are specified and the monitors that are generated.

Using this monitoring setup as a basis, this study aims to improve perioperative outcomes and increase patient safety, by constructing a proof-of-concept of a fitting monitoring system for verifying the surgical actions performed in RAS during runtime. This monitoring system is simulated with the use of a digital model of the da Vinci robot as well as a simplified model of the anatomy of the patient, to check properties of surgical actions during a procedure for violations. This will enable the system to give feedback or issue warnings during surgery to address potentially dangerous actions to the surgeon. This could potentially increase patient safety by improving perioperative clinical outcomes and decreasing the learning curve of the surgeon. For the sake of clarity, we will repeat the research question and subquestions mentioned in chapter 1:

How can a runtime verification and monitoring system improve the patient safety

and learning curve during robot-assisted surgery (RAS)?

- What types of monitoring systems can be helpful in RAS?
- What challenges arise involving safety of the patient during a RAS?
- What specifications of the da Vinci robot can be monitored?

3.2 3D Simulation Environment

As the da Vinci Surgical System and its internal software are not widely and openly available to be experimented with, the use of a 3D simulation environment to virtually represent the da Vinci system would be very helpful in experimenting with this system.

The Asynchronous Multi-Body Framework (AMBF) is an open-source 3D simulation environment that offers real-time dynamic simulation of robots, free bodies and multi-link puzzles [39]. It also utilizes external software for providing real-time haptic feedback interaction through several haptic devices, including the da Vinci system manipulators that come with the dVRK. In this way research teams that are in possession of the dVRK can experiment controlling the da Vinci system with the dVRK manipulators in a virtual environment. Because of this feature, the simulator can be used for real-time training of surgical and non-surgical tasks. The simulator also provides a Python Client for controlling models inside the simulated environment without the need for physical equipment. This client can also be used for training neural networks on real-time data obtained from the simulator.

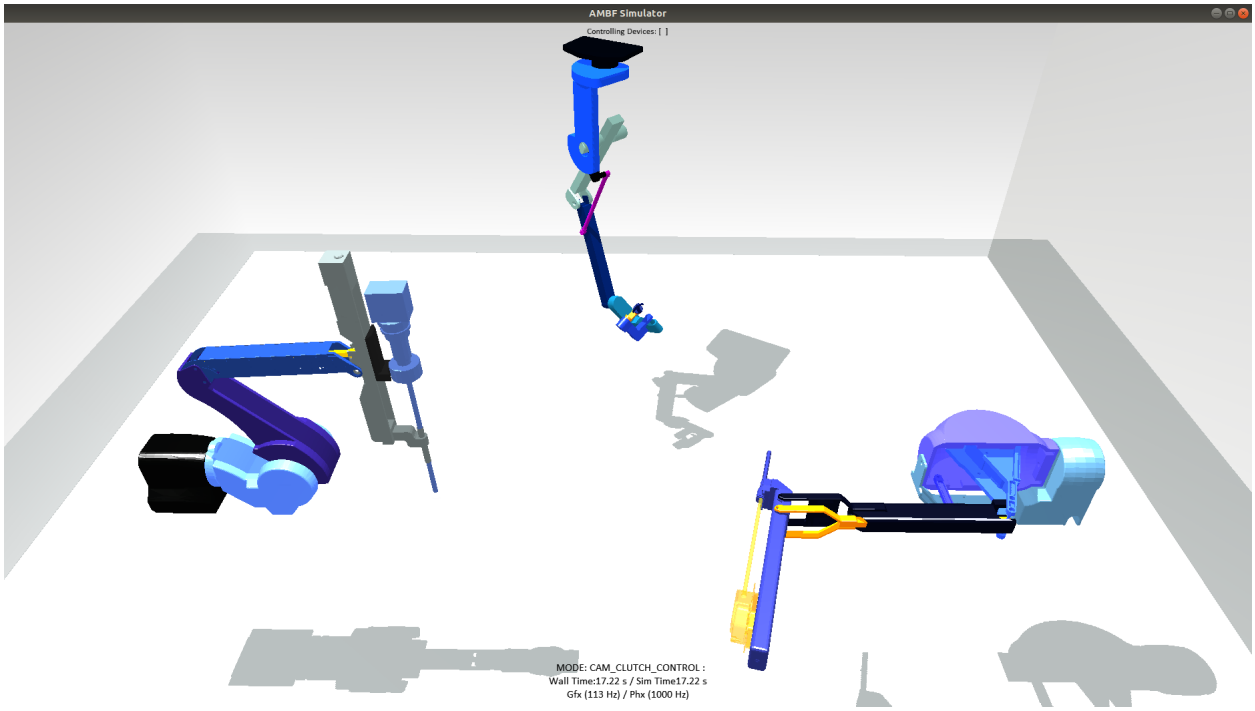

Figure 3.2: 3D Virtual Environment of the AMBF Simulator. [39]

This opens up numerous possibilities for testing different kinds of scenarios happening in real-life surgery. It also opens up the integration of different potential add-ons to the da Vinci system that would improve the overall experience using the system and

ultimately increasing the patient safety during surgery and decrease post-operative recovery time. The simulator is not only applicable to the da Vinci surgical system but can be utilized by many field of robotics related research such as multi-body simulation, manipulation of robotic manipulators and reinforcement learning.

AMBF is integrated with ROS, which handles the communication with robots and allows to control the simulated models through external code. Subsequently, it offers useful plotting and logging features while remaining isolated from the ROS-based runtime mechanics but still being able to leverage its tools. AMBF leverages the topics from the ROS architecture to communicate at high-speeds between the user and the models. Furthermore, the interaction with the simulator can be done either through C++ or Python in order to control robots, multi-bodies, kinematics and visual objects in the simulated environment.

For this project, the Python Client included with AMBF was used to control the dVRK arms and interact with the simulated environment. This is because at the time of conducting this research, we had no dVRK in close proximity and the time that it would take to request one from Intuitive Surgical would supersede the timeline of this research.

The Python Client utilizes ROS to communicate with the dVRK arms in order to control them as well as simultaneously read their state (Pose, Velocity, etc.) using a library of Python functions. In this project, these functions are mostly used for controlling the models inside the simulations environment for setting and getting the position and orientation of the models as well as control the forces and torques acting on the models or get the number of joints connected to it. The main component of the dVRK that being controlled during the simulations is the patient side manipulator (PSM), which is the arm holding the grasping and cutting tools. The Python Client is used to mimic the motions of the PSMs of the da Vinci Robot in the video of a real-life surgery by controlling the PSMs in the simulated environment. This way, we can perform experiments and add features that could improve surgical skills and prevent issues during surgery that stem from flaws in the da Vinci surgical system.

The controlling of the arms using the Python Client is based on the kinematic model of the dVRK. Therefore we need to explain the joints that make the arms move. This will be elaborated on in section 3.4.

3.3 Robot Operating System (ROS)

Robot Operating System (ROS) is a open-source, operating system for robotic systems [43]. For this project, it opens up the opportunity to make an abstraction of the hardware of the da Vinci surgical system and control it virtually. It therefore does not require to experiment with the actual hardware of the real system and make adding and testing commonly-used but also newly developed functionalities to be used during surgery possible.

3.3.1 Architecture & ROS Nodes

Controlling models of robots through ROS is done through mulitple instances of 'nodes' that communicate information about all the components that comprise the model of the robot [43]. Essentially, these nodes are processes that perform some computation or task. This information is past on from one node to the other by means of streaming 'topics', which are essentially topics where data about a particular component is published as a message and can be read and used by other nodes. The use of nodes has the benefit of

fault isolation, which means that if one component or node fails or crashes, the others stay active and working. The system to control a robot often comprises of many nodes. One node controls the motors that drive some rotation of robot legs or wheels that make the robot move in a certain direction, the other controls the sensor that measures the distance to object to prevent it from colliding with that object, another controls planning the path the robot should take and move into, and so on.

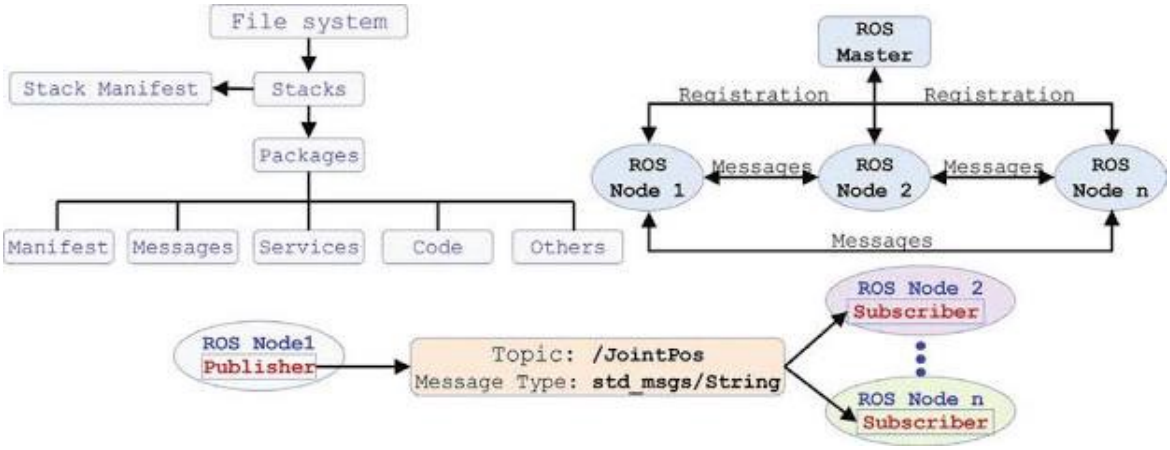

Figure 3.3: A diagram of the ROS architecture. [39]

3.3.2 ROS Topics

These nodes can publish information that can be received by other nodes [43]. An example of this is a camera feed providing the information obtained through the camera to be used by some other entity. A particular kind of information about a component in the system is called a topic. A topic defines the types of messages that will be sent concerning that topic. Nodes transmit data about components of the robot models in ROS. Nodes publish this data to a topic named specifically for the type and name of data is being published. Other nodes can subscribe to a topic and subsequently, the transmitted messages to that topic will be received by that subscribing node. Nodes can publish and subscribe to multiple topics.

To give an example of how a node can publish or subscribe to a topic, let us say that we have a robot with a camera attached to it. This video feed can be published by the node that controls the camera to the topic named "camera/feed". The node that subscribes to the "camera/feed" topic subsequently receives the information that is the video feed. With this information, this subscriber node can for example publish the data on a computer screen if in this case the subscriber node controls the information that is being shown on a computer screen.

3.3.3 ROS Messages

Messages transmitted by nodes are defined by packages that contain the type of message and the data format [43]. For example, the ROS package named "std_msgs" (referring to standards messages) has messages of type "std_msgs/String", which means it consists of a string of characters. There are many different packages that describes messages for different purposes. For example, the package called "sensor_msgs" has message types related to sensor data obtained through sensors that are attached a robot, which often have message type Float64, which consist of numeric characters.

The nodes, topics and messages used during this research will be elaborated on in section 4.3.

3.4 Models of the da Vinci Robot and the Patient

The models of the da Vinci robot that are used in this research are designed based on the dVRK components that are provided by Intuitive Surgical for research purposes [22]. The models of the components of the dVRK are developed by researchers at WPI and JHU and subsequently converted into a AMBF compatible version of those models [39]. Next to the components of the dVRK, a model of the patient is needed to simulate an example to test potential properties on. In section 3.4.3, we will elaborate on the modelling of such a model and the most important parts of the anatomical structures that are needed to be modelled.

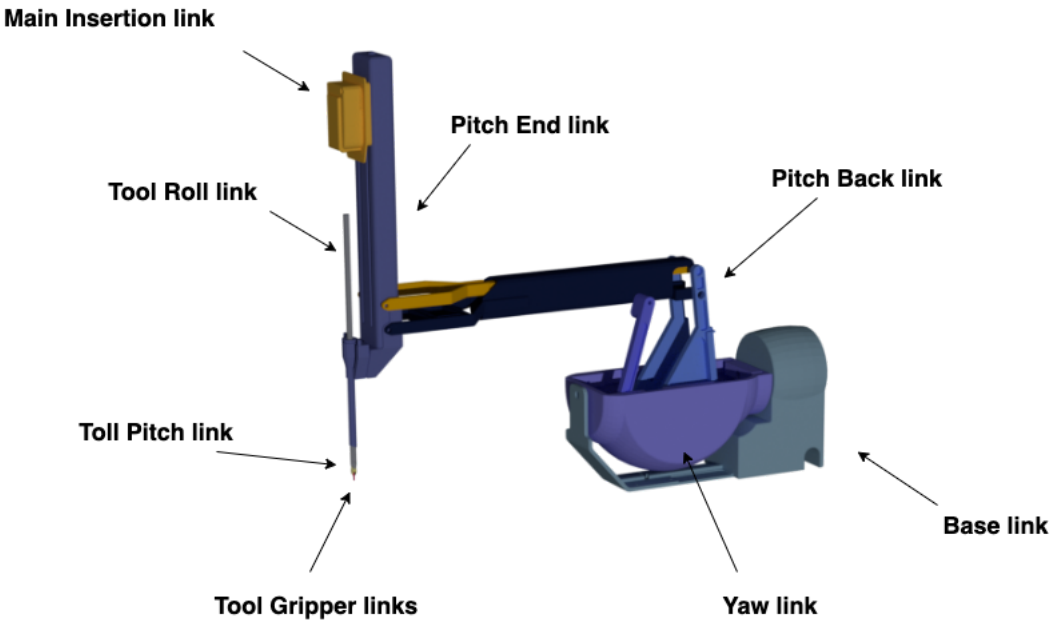

Figure 3.4: 3D model of the PSM included in the dVRK kit.

3.4.1 Kinematic Model da Vinci System

The main model that we are using for this project is the Patient Side Manipulator (PSM). This is because it is the manipulator/arm that holds the gripper and other instrumentals which are the main components used during the procedure and have therefore the highest chance of inflicting damage to anatomical structures. This component of the da Vinci surgical system accounts for human-like movements of hands through the attached gripper at the end of the PSM. To understand how the PSM moves we need to analyze its components and kinematics.

The components of the PSM that are involved in its movement starts at its base, which in AMBF is called the 'baselink' and ends with the two parts of the gripper. We will list the components of the PSM as follows, starting from the baselink:

- Base link
- Yaw link

- Pitch back link
- Pitch end link
- Main insertion link
- Tool roll link
- Tool pitch link
- Tool gripper link (1)
- Tool gripper link (2)

The movements of the gripper depend on all the parent joints it is attached to. The pose of a joint is defined relative to the link it is connected to. Therefore we call this a parent-child relation, which is common in the field of robotics when talking about joints of a robot. The pose of a component of the PSM (also called link, which is a component between two joints) consists of a position (x, y, z coordinates) and a rotation (x, y, z, w coordinates). The joints are named after the two links they are attached to. For example, the joint which moves the pose of everything down from the yaw link is called the 'baselink-yawlink'. The rotation of the joints are represented in radians.

3.4.2 Moving the Arms

We can control the movements of the PSM components by setting the joint rotation between two links. This can be done in a few different ways, but the two ways that were used during this project are:

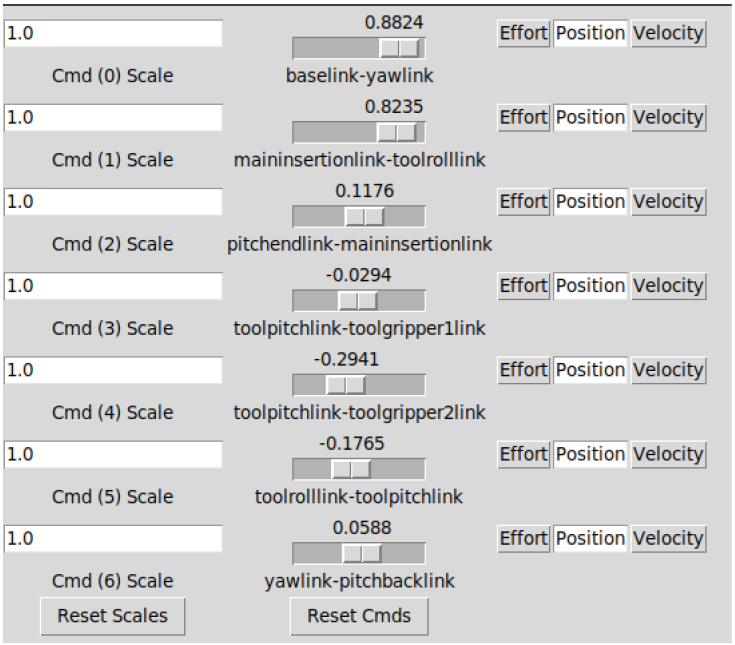

Figure 3.5: GUI to control the rotation of the joints of the PSM in the AMBF Simulator.

- Setting the radian of every joint manually using the GUI provided with the AMBF simulator. This GUI offers sliders to set the radian of the joints. This however is rather cumbersome when trying to simulate motions. Therefore, a Python Client is used that is provided with AMBF to set the position of all joints necessary to

perform the desired motion. The workings of the Python Client will be elaborated on in section 4.2.

- In the simulation environment, links can be grabbed with the mouse/trackpad and moved around. As a result, the other links move around with it relative to the rotation in the joints that result from grabbing and moving a link. However, only one link can be moved at once, therefore accurately simulating a motion of the PSM by only moving one link is not possible. Therefore, we use this approach to only mimic different speeds of the PSM components.

Next to the position of the arms, an important parameter of the motions exhibited by the da Vinci robot is the speed in which the components move around within the patient. The speed of a link in the AMBF simulator is represented by the velocity of that link in the direction of the x, y and z axis. These parameters are constantly updated and can be subscribed to by the Python Client. What we are interested in is the speed of a link, in particular the tool tip of the PSM. The 3D velocity vector that is published to the Pose topic can be used to calculate the speed of any link with the use of Pythagoras' theorem:

$$|\vec{v}(x, y, z)| = \sqrt{x^2 + y^2 + z^2} \tag{3.1}$$

The Python Client subscribes to the topic of the desired link from which we need the velocity vector from. Subsequently, it calculates the speed of that link using the Pythagoras theorem and publishes it to a topic called "/vel_output" so it can be picked up by monitors.

Ultimately, the most realistic way of controlling the PSM is by using the controllers used to control the real-life da Vinci surgical system. The AMBF simulator leverages the inclusion of such controllers in the dVRK by enabling them to be connected to the simulation environment and control the PSM like it would control it in real-life. For this project however, no access to a dVRK was possible and therefore not a viable controlling options.

3.4.3 Model of the Patient

The anatomical structures present in the thorax are highly complex and therefore difficult to find a 3D digital model of. Furthermore, the integration of ROSMonitoring and AMBF needs to be tested to check if it works properly. This is easier to do with relatively simple models to clearly see the correct execution and workings of the system as a whole. Therefore for this project, we chose an iterative approach in integrating, testing and eventually simulating motions of the PSM to be checked on particular formal properties. With the model of the PSM, the components of the model are relatively big in size compared to the anatomical structures and veins that are present in an accurate model of the thorax. Furthermore, an accurate model of the PSM was included with the AMBF simulator, so we decided to use this included model in this iterative process.

However, for the model of the patient, we decided to model and create a simplified version of the thorax to be used in the beginning of this process. This is because, like mentioned in section 2.7, there are a handful of critical anatomical structures within the thorax that are important not to damage. These are the azygos vein, the aorta (including the aortoesophageal vessels) and the esophagus. For orientation purposes, we have added the diaphragm and spine to the model to more realistically represent a thorax. Furthermore, the basic structure of these organs can be seen as similar to

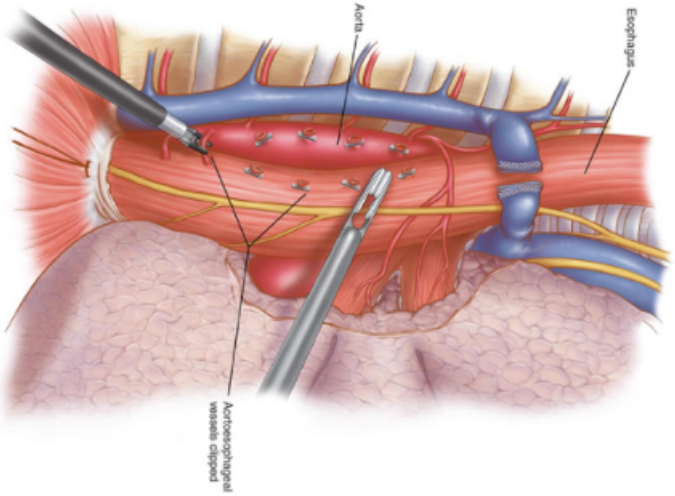

Figure 3.6: Schematic representation of the thorax in Minimally Invasive Esophagectomy. [25]

cylinders, however in real-life these are much more detailed and in certain parts bigger or smaller in size. But for the purpose of making a simplified model we chose to keep the general orientation of these anatomical structures but simplify them by representing them as cylinders. The example that was used to model the thorax can be seen in figure 3.6. In this schematic example, the most important anatomical structures are depicted during the thorax phase of the RAMIE procedure. These structures have subsequently been implemented in the simplified model of the thorax. The decision to omit the lungs from the simplified model is due to visibility during the simulation. The result of the simplified model can be seen in figure 3.7.

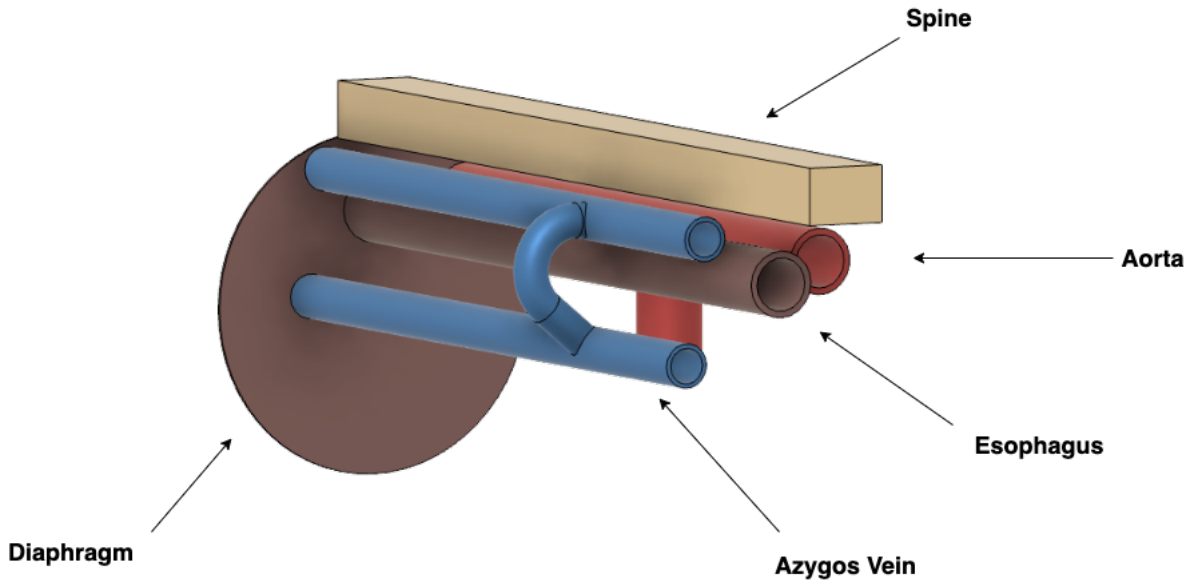

Figure 3.7: Simplified 3D model of the thorax.

The current simplified model is modelled as a rigid body. This means that the model is a solid body and that there is no deformation when colliding with other bodies.

Furthermore, the distance between two given points in the rigid body remains constant in time regardless of any external forces exerted on it. This is evidently not representative to a real-life thorax. Human tissue is predominantly soft tissue, therefore a soft body would be more realistic to use in the simulations. Although the current version of AMBF simulator is able to import and generate soft bodies, soft bodies with complex geometric structure can make pressure and volume constraints (for soft bodies) unstable. This is mostly due to difficulties with the collisions between soft bodies in AMBF, which is a work in progress. Therefore for the scope of this project and based on the concept of a simplified model we chose to model the thorax as a rigid body.

As part of this iterative process of transitioning from a simplified model to a realistic complex model, implementing and simulating motions and interaction with a complex model was out of the scope of this research. In the future, a model generated based on CT-scans could eventually be substituted with the current simplified model. Related work on this topic has been done in for example the superimposing of internal structures of the liver, here a 3D-model of those internal structures is generated for the use in simulations [41].

3.4.4 Importing and Modifying Models

For modifying and importing models in the AMBF simulator, the Blender AMBF Add-on is included with the simulation environment package. This Blender add-on can be used to create and load AMBF yaml config files into Blender. AMBF yaml config files are specifications of multibody systems such as robotic manipulator arms, like the PSM, and are in this particular case used to load models into a simulation environment. Using Blender, these models can be modified to fit some specific application purpose and subsequently easily be exported and again loaded into AMBF. It also provides the ability to create bodies or models in Blender and subsequently use these models in AMBF. Using this add-on, you can manually set all physics related attributes of a model which are then implemented when using the model in AMBF.

The simplified model of the thorax was initially created using Fusion 360 after which it was ported to Blender when discovering the Blender add-on feature the AMBF simulator offers [2, 39]. In Blender, the physics of the model were modified with the help of the AMBF Blender add-on. The model was designed as a static body and had no gravity forced on it to keep it in place within the simulation environment, simulating the patient lying still within the world like in the body of a patient lying on the operating table. Furthermore, the collision margin was reduced to prevent unstable frequency and frame rates during simulation. As mentioned earlier, collision filtering is a work in progress in AMBF and therefore not considered in this project.

The model of the PSM was not modified while being able to port it to Blender using the Blender AMBF add-on. This was due to there being an error when exporting and loading the model back into AMBF which caused issues during the simulation. For the goal of this research, not being able to modify the models of the PSM was no significant loss due to the model being suitable to simulate the motions needed in this project.

3.5 ROSMonitoring

As we are dealing with a robotic system and use ROS as basis for operating it, we decided to go for a RV framework that can leverage the ROS architecture to obtain data about the robotic system and reason about the correctness of the behaviour of this system.

Recently, Ferrando et al. (2020) have designed a RV framework for monitoring robotic systems controlled using ROS, called ROSMonitoring [17]. Subsequently, they have developed a Python implementation for integrating the ROSMonitoring framework with ROS' architecture, therefore being able to verify messages exchanged in the ROS system at runtime. In this way, we can simulate motions of the PSM model with the use of ROS and monitor and verify the correctness or constraints of these motions.

The ROSMonitoring framework consist of three components, the instrumentation, the oracle and the monitor, which resemble the classical main components of traditional RV systems and frameworks. The instrumentation handles the creation and placement of monitors in the ROS environment to intercept messages on relevant topics that we want to check properties against. The oracle is the component that is used for checking whether a particular observed event conforms to some formal specification or not. One of the specific oracles that are included with ROSMonitoring is the Reelay Oracle. Reelay is used to construct runtime monitors for checking temporal behaviours of systems by using state-of-the-art RV techniques like the previous mentioned LTL, MTL and STL.

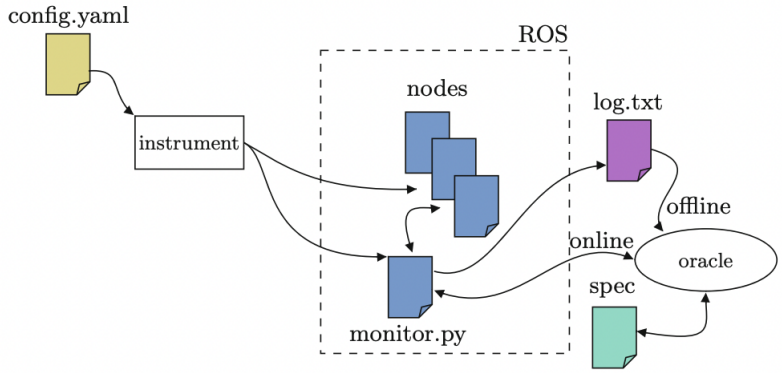

Figure 3.8: A diagram of the ROS and ROSMonitoring integrated architecture. [17]

3.5.1 Integration

The way ROSMonitoring integrates into the ROS architecture is by placing monitors between ROS nodes to intercept messages on topics for checking them against formal properties. Figure 3.8 shows a diagram of the ROSMonitoring framework integrated in and around the ROS architecture. Starting from the left, the monitors that are placed between the ROS nodes are generated through a configuration file, we will elaborate on this file in subsection 3.5.2. For checking the ROS messages against formal properties the oracle is placed outside the ROS architecture. Adding additional new oracles is a task that requires little effort. It leverages the JSON data format to interchange and serialize messages observed by the monitor node. The ROS messages received from a topic are translated into the JSON format [17]. The messages are subsequently sent to the oracle for the verdict. The oracle receives the JSON messages using the WebSocket protocol. WebSocket is used because of its real-time point to point communication between client and server. This way it circumvents the need to continue poll the server for messages. For this research, we chose to use the Reelay oracle for checking messages against formal properties. The conventions of Reelay and its configurations in ROSMonitoring will be discussed further in subsection 3.5.3.

3.5.2 Generation

To generate monitors for a specific application, we need to make a configuration file containing the description of a specific monitor we want to generate. This configuration file produces the Python code for the specified monitors and is automatically generated into a ROS node in Python, which is the native language of ROS and can therefore easily integrate into the ROS architecture. Because of the way ROS works, the monitor subscribes to a topic at interest and gives a verdict on the data in this topic. Furthermore, the monitor perceives the continuous behaviour of the system as a set of events or traces in a discrete manner, making it a hybrid system [8]. While physical processes such like the motion of a robot evolve in a continuous time manner, the monitor gives verdicts on a discrete state space of these physical processes. The rate in which data is published to topics can be modified to suit the application. Modifying this rate will subsequently modify the time points the monitor gives a verdict over. For example, setting the publish rate to 1 means that every second, data is published to a topic. Therefore, the monitor gives a verdict every second of the simulation. Setting the publish rate higher to for example 2 means the monitor will give a verdict every half a second.

By default, the monitors can log or filter messages that are intercepted. We can specify this in the configuration file if we are only interested in logging the messages or we can specify the monitor to intercept the messages and interfere. Another feature we can specify is the monitor giving a warning if a property is violated. When the warning attribute is specified as active in the configuration file, the monitor will publish as much info about the event where a property is being violated and will publish this to a special topic called "/monitor_error" which contains messages of its own type "MonitorError.msg". The info that is published here describes the topic where the event happened, the content of the message that is intercepted, the violated property and the time when this violation occurred during the simulation run.

```
monitors: # here we list the monitors we are going to generate
  - monitor:
      id: dvrk_velocity_monitor
      log: /ambf/log_dvrk_velocity.txt # file where the monitor will log the observed events
      silent: True # we let the monitor to print info during its execution
      warning: 1
      oracle: # the oracle running and ready to check the specification (localhost in this case)
        port: 8080 # the port where it is listening
        url: 127.0.0.1 # the url where it is listening
        action: nothing # the oracle will not change the message
      topics: # the list of topics this monitor is going to intercept
        - name: /vel_output # name of the topic
          type: std_msgs/Float64 # type of the topic
          action: log

  - monitor:
      id: dvrk_distance_monitor
      log: /ambf/log_dvrk_distance.txt # file where the monitor will log the observed events
      silent: True # we let the monitor to print info during its execution
      warning: 1
      oracle: # the oracle running and ready to check the specification (localhost in this case)
        port: 8080 # the port where it is listening
        url: 127.0.0.1 # the url where it is listening
        action: nothing # the oracle will not change the message
      topics: # the list of topics this monitor is going to intercept
        - name: /dis_output # name of the topic
          type: geometry_msgs/Pose # type of the topic
          action: log
```

Figure 3.9: Configuration file for generating monitors.

The generation of monitors for this project depend on the properties we want to monitor. Within the scope of this research, we need a monitor that leverages a topic that is publishing the relative distance from the gripper of the PSM to critical anatomical structures as well as a topic publishing the velocity of the gripper over time. Furthermore, we want the monitor to give out a warning in the event a specified property is violated, therefore setting the warning configuration to 1. These attributes are represented in the example of a configuration file in figure 3.9.

In the configuration file we also set the path to the log file logging the verdict of the monitors, specify the server (URL and port) the oracle needs to listen to in order to receive the messages from the monitor and specify the path to the topic it needs to subscribe to in order to read its messages. After we have set all the desired attributes, running the generator creates a monitor folder containing all the necessary components to run the monitor. For the monitor to be able to read messages from the AMBF simulator topics, the folder needs to be placed inside the AMBF simulator folder before running the monitor.

3.5.3 Properties

The properties used in the ROSMonitoring framework can be formulated in LTL, MTL and STL with the help of Reelay to construct runtime monitors for these properties using its own conventions. For translating in English written requirements, we therefore define the syntax of both STL, which is an extension of LTL and MTL, and define the Reelay conventions.

Signal Temporal Logic: The properties that are specified in this research are based on Linear Temporal Logic (LTL) and Signal Temporal Logic (STL), this is because this formal language extends the semantic domain of classical temporal logic with real-valued signals and presenting how far a predicate is away from being satisfied or violated. We presented the formal definition and syntax of LTL in section 2.5.3 and will give the formal syntax of STL and its past and future operators for the sake of completeness:

The syntax of STL is defined as follows over a set X of real-valued variables:

$$\phi ::\equiv x \sim u | \neg\phi | \phi_1 \vee \phi_2 | \phi_1 U_I \phi_2 | \phi_1 S_I \phi_2 \tag{3.2}$$

where x ∈ X, ∼ ∈ ≥, ¿, =, ¡, ≤, u ∈ Q, I ⊆ [0, ∞) is a non-empty interval [20]. Based on this, we can derive the following standard operators for STL to lay the basis for formulating properties in STL:

- Tautology: true = p ∨ ¬p
- Contradiction: false = ¬ true
- Disjunction: $\phi_1 \wedge \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$
- Implication: $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$
- Eventually: $F_I\phi = \text{true}U_I\phi$
- Always: $G_I\phi = \neg F_I \neg\phi$
- Once: $O_I\phi = \text{true}S_I\phi$
- Historically: $H_I\phi = \neg O_I \neg\phi$

With this base, we can formalize in English written requirements into STL specifications. These specified properties can subsequently be used for verification of CPSs and to generate monitors for checking the behaviour of these systems.

Reelay Conventions: The ROSMonitoring framework implements these formulas according to specific conventions, namely the Reelay conventions. As mentioned before, for this research we use the Reelay tool to implement state-of-the-art RV techniques to construct runtime monitors that check the temporal behaviour of the da Vinci robot against formal properties. Reelay checks specifications received in plain text and verifies these specifications based on the behaviour of the system at runtime. For automatically generating monitors using Reelay and make them execute in runtime, the specified properties have to be written based on a particular syntax, namely the Reelay Expression (Rye) format [53].

We need to keep in mind that the Rye format expresses STL in its past form, as the oracle can only give a verdict over the logged sequence of traces that have been past on by the monitor. However, the past form of STL is as expressive as its normal/future form, only exponentially more succinct. Furthermore, as mentioned in the last section, the behaviour of the da Vinci robot evolves in continuous time. However, the monitor perceives the behaviour in a discrete state space in the form of events and traces (see section 2.4 for an elaborate description of events and traces), making this a hybrid system (see section 2.5.3). The time steps the properties are specified to, correspond to the discrete state space in which the monitor perceives the behaviour of the simulated da Vinci model. For example, when data about the behaviour of the da Vinci robot is published to a particular topic every second, referring to the previous time point means referring to the previous second during the simulation.

The syntax of the Rye format is based on keywords and punctuation, in order to construct a description of specification about the behaviour of the system. This behaviour is of a temporal nature and depicted as a stream of events, or traces, and is formatted in a multiline format like JSON (which is used in the ROSMonitoring framework). We can categorize each type of expression and its syntax as follows:

- Atomic expressions
- Boolean logic expressions
- Temporal logic expressions

Atomic expressions: As with other formal expressions, atomic expressions are the core of Reelay expressions. The syntax used for atomic expressions in Reelay is depicted by wrapping them with curly-brackets {...}. The constraints of the predicates used in the specification are presented within these brackets like in the following example:

{moving: true, speed > 2.0}

which will be evaluated true at time points 4 and 5 in the JSON text lines depicting the behaviour of the system over some time point like below:

```
...
{"time": 4, "moving": true, "speed": 4.05}
{"time": 5, "moving": true, "speed": 3.22}
{"time": 6, "moving": false, "speed": 0.63}
...
```

Reelay can reason over Booleans (true, false), numerical comparisons (<, <=, >, >=, ==, !=) and string equivalences when it comes to atomic expressions [53].

Boolean logic expressions: When it comes to Boolean logic operations over atomic expressions, Reelay applies them in a rather straightforward fashion. We will continue using the same example used in explaining the atomic expressions:

- Negation: The negation of expressions in Reelay are depicted with either 'not' or !. For example, not {moving: true, speed $>$ 2.0}.
- Conjunction: The conjuction of expressions in Reelay is depicted by either 'and' or &&. Which can be either expressed like {moving: true} and {speed $>$ 2.0} or {moving: true, speed $>$ 2.0} because they are functionally equivalent.
- Disjunction: The disjunction of expressions in Reelay is depicted by either 'or' or ||. For example, {moving: true} or {speed $>$ 2.0}.
- Logical implication: The logical implication of expressions in Reelay is depicted by either 'implies' or ->. For example, {moving: true} $\Rightarrow$ {speed $>$ 2.0}.

Temporal logic expressions: As mentioned before, Reelay only support past operators of STL as of this moment. Therefore, we will present the past operators and again give examples of their possible implementation:

- Previously: This unary operation is depicted either using 'pre' or Y, and can only be applied when dealing with untimed or discrete time, not when dealing with dense time. For example, pre{speed $>$ 2.0}, which is true at the current point in time if the speed was above 2.0 at the previous point in time.
- Sometime in the past, Once: This unary operation is depicted either using 'once' or P, and can be extended by adding time constraints. For example, once[a:b]{speed $>$ 2.0}, where a and b are two different time points and is true at the current point in time if lights were on sometime in the past between time point a and b.
- Always in the past, Historically: This unary operation is depicted either using 'historically' or H, and can be extended by adding time constraints. For example, historically[a:b]{speed $>$ 2.0}, where a and b are two different time points and is true in the current point in time if the speed was always above 2.0 between the time points a and b.
- Since: This binary operation is depicted either using 'since' or S between to atomic expressions, and can be extended by adding time constraints. For example, {moving: true} since[a:b] {speed $>$ 2.0}, where a and b are two different time points and is true in the current point in time if moving is always true since the speed was above 2.0 between time point a and b.

We will give a detailed description about the creation process of the specifications used in this research in section 4.3, where we use the syntax of both STL as well as Reelay to derive suitable specifications to be implemented and used in the ROSMonitoring framework.

4 Results & Findings

Through the use of the analysis and review of literature, we found multiple new insights into the research subjects. This chapter will give a structured overview of the results of the monitoring of motions during simulations that have been run with the use of the AMBF simulator and the ROSMonitoring framework and subsequent analyses of the performance of the components used. To present the findings as comprehensively as possible, we structured this chapter around the three examples of undesired behaviour of controlling the da Vinci robot, one real-life example and two synthetic examples. These examples have been simulated and subsequently checked using the ROSMonitoring framework to show the significance of such a RV monitoring system could have as implementation in real-life surgery. Firstly, section 4.1 will give a elaborate description of the examples used for the simulations and monitoring. Subsequently, section 4.2 will describe how the motions present in the examples are translated to be used in the simulations. Furthermore, section 4.3 will outline the properties that were specified for checking the motions present in the simulations. And lastly, in section 4.4 the results of the monitoring system will be presented based on the simulated motions and will be described how the system performed.

4.1 Real-life example from Surgical Video

To test the workings and the significance of a RV monitoring system in RAS, we will represent three examples to build the simulations on. One example is based on a real life surgical video showing a minor bleeding of the aorta during the thoracic part of the RAMIE. While the other two exampled are synthetic but based on a potentially useful and significant property to check during the RAMIE procedure, which is based on hasty and hesitant movements.

In the first example, one of the PSMs is holding fat tissue away to get a better view of the esophagus. In this motion, the PSM moves out of the of camera view, which can be seen in figure 4.1. A couple of seconds later, a bleeding is observed in the corner where the PSM just disappeared out of side. After adjustment of the camera view and checking where the bleed is coming from, we can see the PSM has touched the aorta during the previously mentioned motion, which can be seen in figure 4.2. This is a minor bleeding and it was fixed using putting pressure with a surgical gauze.

As mentioned in the safety property, a warning when one of the PSMs comes too close to a critical anatomical structure would potentially help preventing such scenarios. To give an example of how such a warning would be implemented and to check if it is possible to monitor such motions, we have simulated similar motions to mimic the scenario that was present in the surgical video. The main motion that was present in the surgical video is that the PSM had come too close to the aorta. Therefore what we want to monitor for, is the distance between the gripper of the PSM and the critical structures within the part of the body that is currently being worked on. How we have

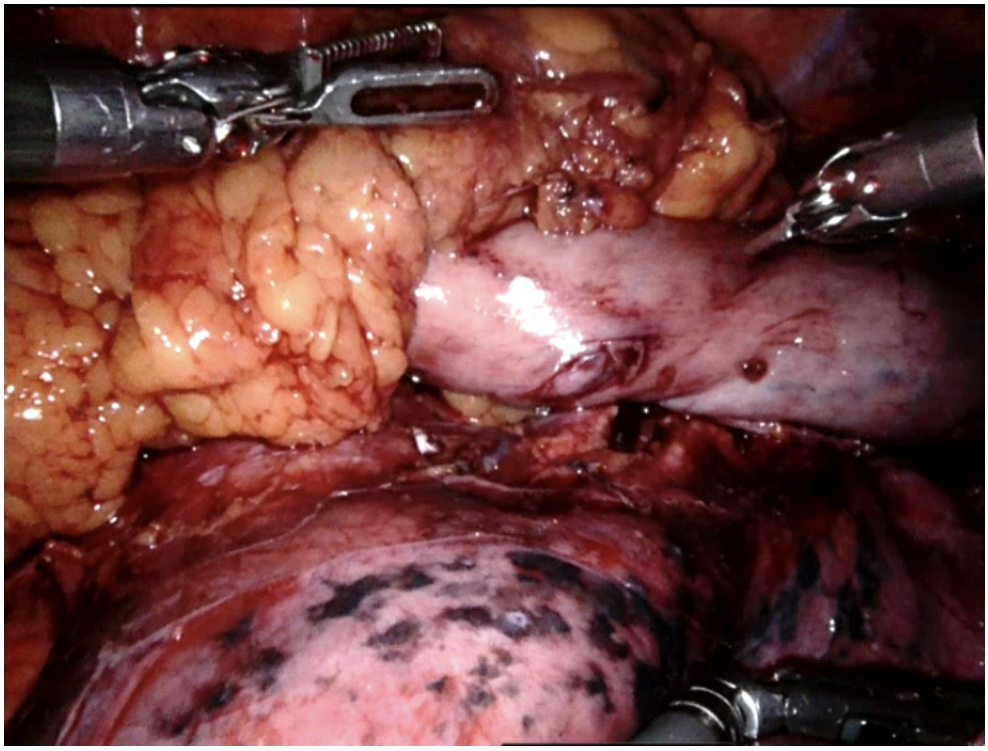

Figure 4.1: Real-life surgical video example.

simulated these motions and specified the properties for this in sections 4.2 and 4.3.

The second and third example are synthetic examples related to how comfortable surgeons handle the da Vinci surgical system controllers. Novice surgeons can be relatively unfamiliar with handling the manipulators in terms of orientation and arm movements. Furthermore, having little experience with using the da Vinci robot in real surgery instead of a simulator can cause tension. This can result in hasty and hesitant movements and motions which can lead to scraping or touching critical anatomical structures. This is especially interesting and important with the surgeries performed using the da Vinci robot, as one of the main reasons of using such a robot for a particular surgery is that the procedures require utmost precision. When a surgeon exhibits this kind of behaviour, alerting them could lead to them reflecting on their execution and slow down and be more precise if necessary. A way to monitor this would be to measure the average movement speed over a series of RAMIE procedures and taking that average as a reference point when checking if the movement speed of a surgeon is too high.

For these examples, searching for such behaviour is a difficult and time-consuming task as the procedures done in the UMC Utrecht are mostly done by highly experienced surgeons. Furthermore, we do not have access to a dVRK kit meaning that we will not be able to synthesize real hesitant motions by for example a surgeon with little to none experience controlling the da Vinci robot. What we can synthesize is simulating motions that are rather quick and uncontrolled. Such motions can be represented by the velocity of the gripper of the PSM over a certain period of time. For showing the correct monitoring of the system we therefore can for example specify a property that is violated when the PSM is moving too fast. This property would be based on hasty movements, where the surgeon is moving the PSM too fast and it would be wise to slow down. The other property we can derive from such behaviour are hesitant movements, where the surgeon pauses or slows down in the course of taking an action or making

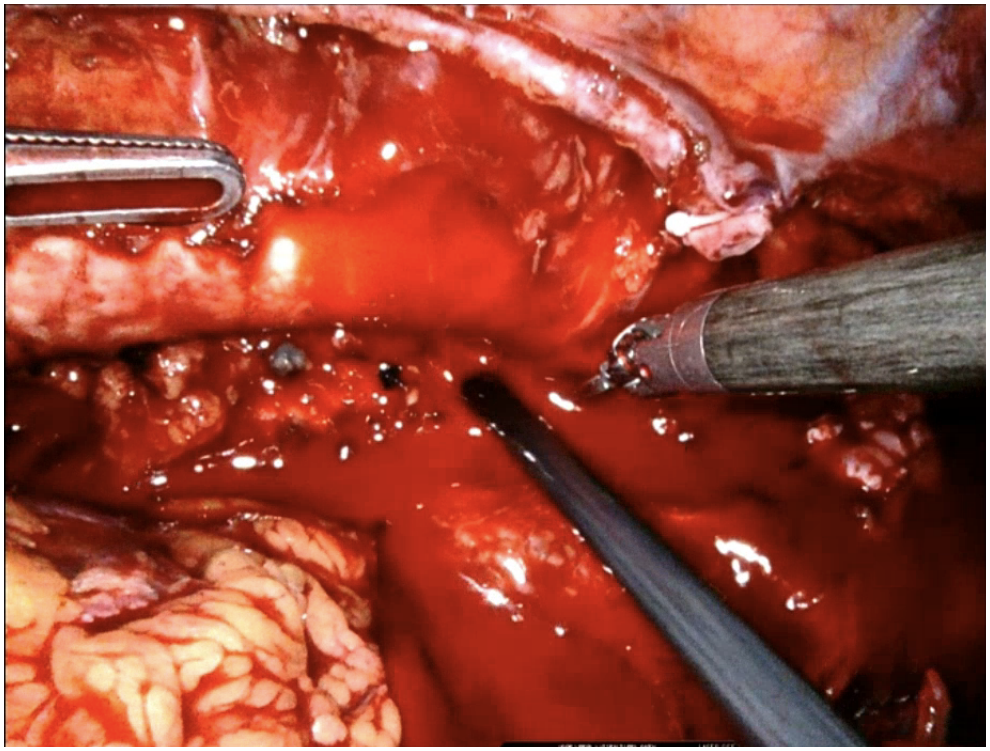

Figure 4.2: Damage to the Aorta.

a movement. Subsequently, if in the future we get access to a dVRK kit and can let surgeons control the PSM models with real da Vinci controllers, the only thing that needs to be adjusted is the specified property in order to check for hasty and hesitant movements.

4.2 Simulated Motions

For implementing a RV monitoring system and use it in the aforementioned examples, we need to simulate the motions using the AMBF simulator and ROS. As mentioned in section 4.4, we use the included AMBF Python Client and computer mouse to control the PSM model.

For the safety distance example we must simulate the PSM to move towards the aorta from a relatively right side of the field of vision moving towards the left. Furthermore, while making this horizontal translation, the PSM should move away from the endoscope towards the aorta, which is situated behind the esophagus in this scenario up till the point where is collides with the aorta.

Using the Python Client, we can set the position of the joints in a way that looks similar to the surgical video. We should set a starting position and a target position of the joints. When running the Python Client, the transition from the starting and target position happens fluently, creating a motion similar to the one in the surgical video. The starting and target position are determined with the help of the GUI for setting joint rotations included with the AMBF simulator. As we do not have real kinematic data from the da Vinci robot during the surgery depicted in the surgical video, we have implemented the motion based on what objectively happened in the video fragment, a motion of the PSM resulting in a collision with the aorta.

The distance between the gripper of the PSM and the aorta is calculated by taking

the relative position of both bodies. The position of the bodies is part of the pose of a body, which is measured through its position and orientation within the simulation environment. The pose info is published by the state topic belonging to the specific body we are interested in. What we need to do to calculate the distance between the two bodies is calculating the relative position by using the Python Client to subscribe to both State topics of the models and then subtract the the two positions from another. Then, in order for the monitor to receive a message that includes this relative position, we need to make a publisher that publishes this relative position to a topic. We call this topic "dis_output", depicting the output of the distance between the two bodies.

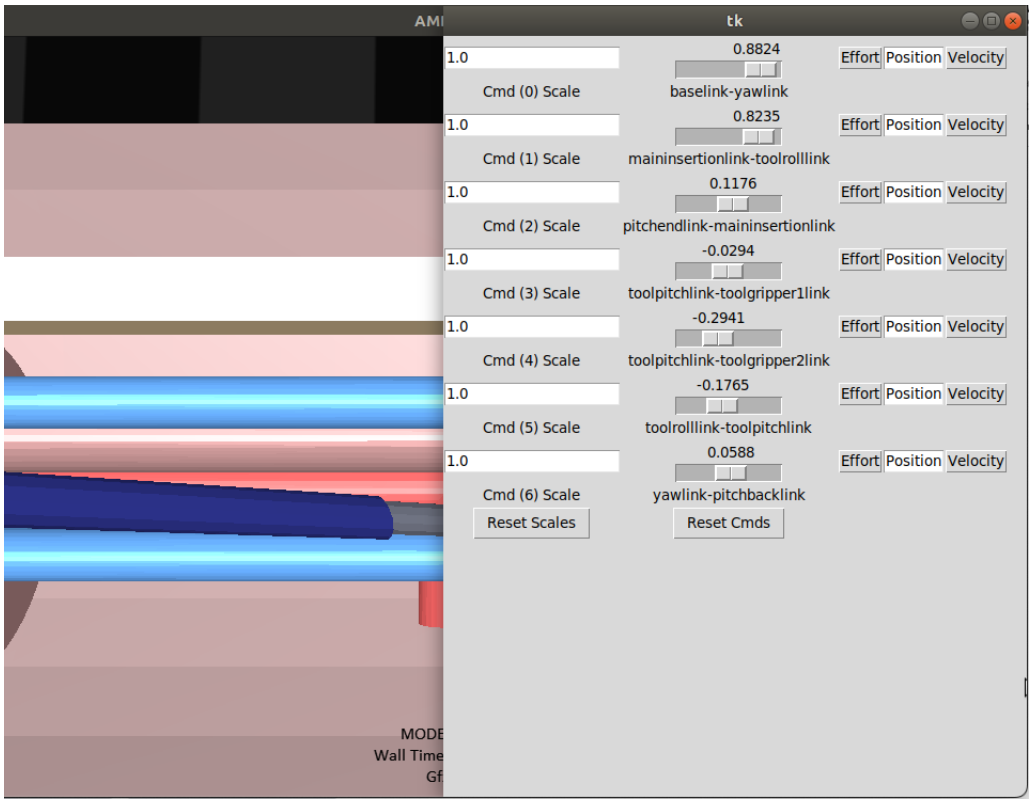

Figure 4.3: Joint positions of the PSM used during simulations.

For the example of hasty and hesitant motions, we do not have an example from a real-life surgical video. Therefore we have created synthetic ones, meaning that we control the PSMs in the simulation environment in such a way that it should trigger the violation of the specific property we are monitoring for. Furthermore, in contrast to the safety distance example, the simulated motions do not have to depend on the positions of the models involved. The main factor we have to look at is the velocity of the PSM.

As mentioned in section 2.2, the da Vinci robot adds enhanced hand-eye coordination and intuitive movement compared to conventional laparoscopy by aligning the motion of the surgical tools with the field of vision of the surgeon and positioning the image of the surgical site atop the hand of the surgeon which provide both spatial and visual alignment [50]. Resulting from this, the da Vinci surgical system can filter out small uncontrollable movements in the hand of the surgeon, providing a steadier and more deliberate approach to surgical tasks. However, the surgical skill difference between an experienced and novice surgeon is still present. This manifests itself, among other things, through the fluency of the motions and movements and handling of the da Vinci controllers, not moving too fast or exhibit hesitant movements when performing a task

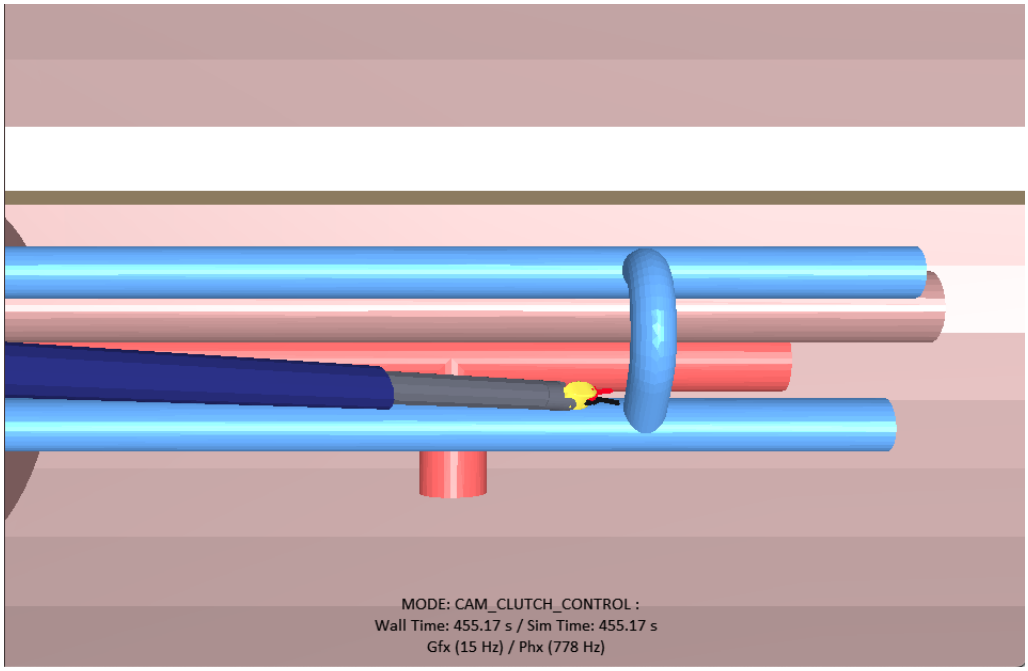

Figure 4.4: Starting position of simulated motion for the safety distance property.

during a particular phase of the surgery.

However, not much research has been done on this subject, hence why it is difficult to specify a realistic property for such movements and motions. Soudan (2020) has tested the influence of velocity of the tool tip on positional error at the point of interest and tested this with velocities of 10 mm/s, 50 mm/s, 100 mm/s and 150 mm/s [52]. Therefore, for this research and these simulations we assume a velocity of the tool tip greater than 150 mm/s as a threshold for exhibiting too fast movements.

For simulating a variety of velocities of the tool tip, we take a different approach to controlling the PSM than in the previous example. Controlling the PSM through the Python client results in a very consistent velocity of the tool tip because it is controlled and moves with a constant force. This will not give very diverse velocity output and is therefore also not representative to real-life. To make it more realistic, we will control the PSM using the computer mouse. This is done by right clicking on the tool tip and moving the mouse in any direction. Moving the mouse quicker results in quicker movements of the PSM and therefore a higher velocity of the tool tip. Furthermore, controlling the PSM by a human makes it evidently more representative to real-life RAS as this is executed by a surgeon rather than a piece of software. However, in the safety distance example, controlling the PSM in this way was not accurate enough to test out the specified properties. In order for the monitor to receive a message that includes the velocity of the tool tip, we need to make a publisher that publishes this velocity to a topic. We call this topic "vel_output", depicting the output of the velocity of the tool tip of the PSM. The velocity as it is represented through the velocity of the tip in the x, y and z axis direction. For the purpose of checking properties, we calculate the average velocity over these three axes as mentioned in section 3.4.2.

The advantage of implementing such a potential monitor is that it will be easily modified when more research will have been done on the topic of checking motions of surgeons. The only thing that is needed is to specify the desired property and the monitor will check it accordingly. This lays a great base for future implementation

and checking of hesitant and hasty movements in the event future research determines a more specific definition of hesitant and hasty movements when using the da Vinci surgical system.

4.3 STL Specifications

For demonstrating the monitoring of motions and movements of the PSM in simulations, we have designed 3 main example STL runtime specifications. We have formulated these specifications based on the requirements and examples of possible properties mentioned in section 2.7.2. These properties can be written in English and were constructed based on first hand knowledge from experts in the field of robot-assisted surgery. This includes parameters and constants mostly regarding the movements of the da Vinci arms and therefore relating to the kinematics of this surgical system. like for example the position of specific components of the PSM and the velocity of the movement of the arm.

When designing the specifications, we considered the temporal nature of the RAS and in particular the RAMIE procedure. For example, during the RAMIE procedure, a distance should be kept to critical anatomical structure but this also depends in which phase of the procedure we are currently in. When a critical anatomical structure is not actively participating in the current surgical phase, the distance that needs to be kept can for example be increased to insure giving an extra early warning to the surgeon and therefore preventing collision more adequate. Therefore the temporal nature of the RAMIE procedure relates to the order of states/phases in which the procedure is executed, which is based on a predictable string of events and time duration.

Specification creation is an iterative process. So to adhere and insure the correctness of the specifications we have validated these specifications during the development of the simulations and the monitoring system. While checking the correct workings of the monitoring system we simultaneously test the specifications, tracing errors back to the STL formulations, parameter definitions or the in English written requirements. We have done this by manually checking the generated traces by the monitoring system for correct and accurate representation of the data obtained through the AMBF simulator and ROS. Furthermore, when running the simulations and the monitoring system, we plot the verdict of the monitoring at each time step and assess analytically if the specification has correctly captured the requirement. This way, we can trace back errors to the STL specifications and adjust them accordingly to fix the bugs causing the incorrect functioning of the monitoring system. After the validation and debugging, the same strategy is used to demonstrate the efficacy and functioning of the RV system. This will be elaborated on in section 4.4.

The properties used for the simulations will be outlined in the subsections below. We will give a detailed description of the translation of the English requirement of the desired property to STL and subsequently to the Reelay conventions for the sake of clearity and reproduceability of the specification creation process.

4.3.1 Safety Distance Property

The values used in the specifications for the safety distance and the max tool tip speed are chosen arbitrarily as no real data is given access to from the original da Vinci surgical system and in particular the data from the example of the surgical video. Therefore, for the safety distance example we have chosen to set the threshold of safe distance between the tool tip and the aorta to be around 1.75mm. This is based relative on the scale of the models used during the simulation and therefore do not accurately represent real

life scales. Furthermore, we have omitted the relative position of on the x axis as the bounds that the PSM needs to stay in are greater than the maximum position the PSM can reach. Therefore, for this property we define the in English requirement to be:

The distance between the PSM tool tip and the aorta should never exceed below 1.75mm.

This English requirement based on expert knowledge should subsequently be translated to a STL specification in order to be implemented into RV systems. The STL translation as follows:

$$G(-0.015 \leq relPositionY \geq 0.015) \wedge (-0.015 \leq relPositionZ \geq 0.015) \tag{4.1}$$

This property will be evaluated true for every point in time where the relative position between the PSM tool tip and aorta stays outside the range of -0.015 and 0.015 and becomes false if it comes into this range. Lastly, we need to formulate this property in the Rye format in order to implement it in the ROSMonitoring system. As the Reelay Oracle currently only supports past operators, the property in Rye format is as follows:

$$\{-0.015 <= relPositionY >= 0.015\} and \{-0.015 <= relPositionZ >= 0.015\} \tag{4.2}$$

which is true if the relative position of the y and z axis is between -0.015 m and 0.015 m (corresponding to a distance 1.75 mm to the aorta) at the current time point. As Reelay only supports temporal operators at the moment, we want it to evaluate every current time point, and not based on past time points as we want to give the warning of the property violation to the surgeon at the moment the violation happens.

4.3.2 Hasty Movements Property

Looking at the threshold for the tool tip speed used for the hasty movements property, the value for this threshold is also arbitrarily chosen due to there being very little data or research on tool tip speeds. As mentioned in section 4.2, we therefore use a threshold of 0.15 m/s (which equals 150 mm/s, as previously stated). The English requirement for this property is defined as follows:

The tool tip speed should always be below or equal to 0.15 m/s.

This English requirement is again translated to the STL language as follows:

$$G(speedToolTip \leq 0.15) \tag{4.3}$$

This property will be evaluated true for every point in time where the speed of the PSM tool tip is below or equal to 0.15 m/s and false if the speed exceeds this value. Lastly, we again need to formulate this property in the Rye format in order to implement it in the ROSMonitoring system. The property in Rye format looks as follows:

$$\{speedToolTip <= 0.15\} \tag{4.4}$$

which is true if the speed of the PSM tool tip is lower or equal to 0.15 m/s at the current time point. Again, we want this property to be evaluated at every time point by the same reason mentioned for the safety distance monitor

4.3.3 Hesitant Movements Property

Hesitant movements are defined in a different way compared to hasty movements. In this research we assume hesitant movements to be pausing in the course of taking an action/movement, according to the definition of hesitant. We can interpret this as stop moving for a small amount of time (for which we have arbitrarily chosen a time span of 1 second) in the course of performing a movement. The value assigned to observe the PSM as moving is 0.01 m/s and is chosen arbitrarily for the sake of demonstrating the correct monitoring of the property clearly. Therefore for this property we define the in English written requirement as follows:

The tool tip should not, be moving, then stop/pause moving and start moving again in the span of 1 second.

This English requirement is translated to the LTL language and is implemented with time steps of a third of a second and looks as follows:

$$\neg((speedToolTip \geq 0.01) \wedge X(speedToolTip < 0.01) \wedge XX(speedToolTip \geq 0.01)) \tag{4.5}$$

This property will be evaluated false if the speed of the PSM tool tip is equal or above 0.01 m/s at the currect time step, below 0.01 m/s at the next time and then above or equal to 0.01 m/s at the time step thereafter within the span of 1 second. This property is true in the event this does happen and the tool tip has a more regular flow of motion. Lastly, we again need to formulate this property in the Rye format in order to implement it in the ROSMonitoring system. The property in Rye format looks as follows:

$$\neg\{speedToolTip >= 0.01\}\&\&pre\{speedToolTip < 0.01\}\&\&pre(pre\{speedToolTip >= 0.01\}) \tag{4.6}$$

which is true when the speed of the PSM tool tip is higher or equal, then below and subsequently higher or equal than 0.01 m/s again, in the last three time points including the current one. Again, we have modified the property to be able to be checked using past-STL.

4.3.4 Overview and Implementation

To give an overview of possible properties that can be specified, table 4.1 presents the three properties used for the example simulations These additional properties are examples that could be used and elaborated on in future studies into STL specifications for RAS.

Example	Abbreviation	STL Specification
Safety Distance	SD1	$G(\text{-}0.015 \leq relPositionY \geq 0.015) \wedge$ $(\text{-}0.015 \leq relPositionZ \geq 0.015)$
Hasty Movements	HA1	$G(speedToolTip \leq 0.15)$
Hesitant Movements	HE1	$\neg((speedToolTip \geq 0.01) \wedge X(speedToolTip < 0.01) \wedge$ $XX(speedToolTip \geq 0.01))$

Table 4.1: STL Specifications

For demonstrating how the properties are implemented in the Reelay oracle, figure 4.5 shows how the SD1 specification is represented in order for the Oracle to check the messages received from the monitor on. Here we convert the messages to the corresponding predicates used inside the desired property. We begin with stating the property to be verified, after which we define the predicates used in the property, including initializing the time to 0. Next, we also need to define how to translate the Json messages received from the monitor to the predicates. We do this by using a function to abstract a dictionary, which is obtained from the Json message, into a list of predicates. In this example, we abstract the position values from the "dis_output" topic and subsequently define the thresholds corresponding to the desired property.

```
PROPERTY = "{safety_distance}"
predicates = dict(
    time = 0,
    safety_distance = True
)

def abstract_message(message):
    if message['topic'] == '/dis_output' and -0.015 <= message['position']['y']\
            <= 0.015 and -0.015 <= message['position']['z'] >= -0.015:
        predicates['safety_distance'] = False
    else:
        predicates['safety_distance'] = True
    predicates['time'] = message['time']
    return predicates
```

Figure 4.5: Implementation of the safety distance property in the Reelay Oracle.

4.4 Functioning of the RV System

As mentioned before, for checking the correct workings of the RV system to be used for the RAMIE procedure, we have designed and run simulations to analyse the functioning of the RV system. These simulations are based on real-life surgical videos as well as synthetic examples for allowing a wider range of simulations to initially test the system on. What we expect and want to see is a warning when a property is violated during the execution of the simulation. We will discuss simulations of the examples mentioned in section 4.1 by simulating the motions described in section 4.2.

To demonstrate the efficacy of our approach, we examine the STL specifications based on these three examples and the resulting output of the ROSMonitoring system from the kinematic data obtained during the simulations. We do this by analyzing the visual simulation and the log file outputted by the monitoring system which we plotted to visualize the values of relevant parameters at each time step and the corresponding verdict.

Firstly, we will discuss the example of monitoring safe distance held between the tool tip of the PSM and the aorta, being a critical anatomical structure which should not be damaged during surgery. In figure 4.6 we can see the starting position of the PSM and its tool tip, resembling the moment in the surgical video where the distance between the tool tip and the aorta is great enough to be safe. On the top right we see the instance of the Python Client that operates the movements of the PSM. Below this we the relative position between the PSM tool tip and the aorta. Lastly, we see the output of the

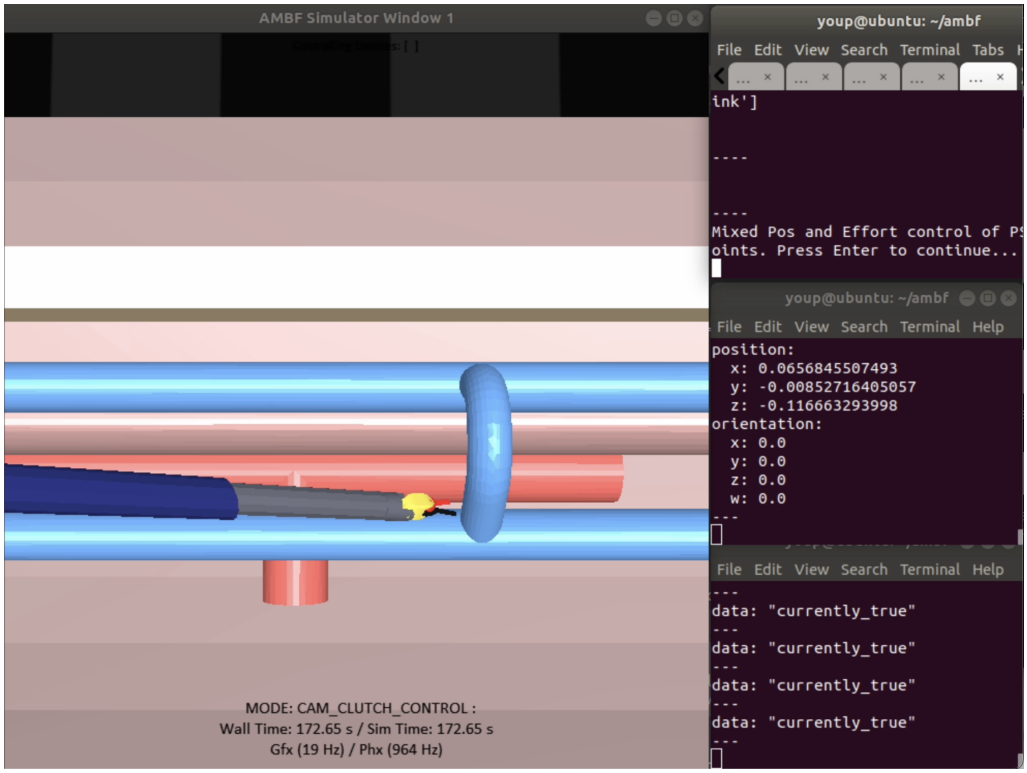

Figure 4.6: ROS Topic and Monitor Verdict Output.

monitor giving the verdict of the safety distance property at different points in time. According to the property specified in section 4.3, the monitor should come to the verdict that the safety distance is currently true based on the current distance between the tool tip and the aorta. As can be seen in figure 4.4 this is indeed the case. Furthermore we can see the same in the content of the log file plotted in figure 4.7 and 4.8, where the output of the monitoring system is logged during the execution of the simulation.

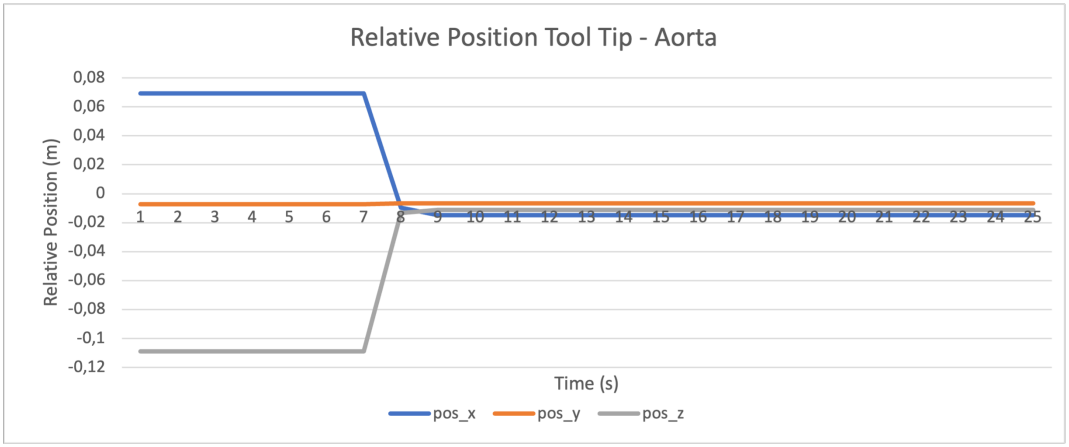

Figure 4.7: Relative Position between the PSM's Tool Tip and the Aorta.

The next movement of the PSM is a motion towards the left side of the field of vision, towards the aorta. The end of the motion takes the tool tip of the PSM so close to the aorta that they come in contact with each other, resembling the graze that happened in the surgical video causing the damage and bleeding of the aorta. We can now see in the log files that the verdict of the monitoring system turns to currently false, indicating

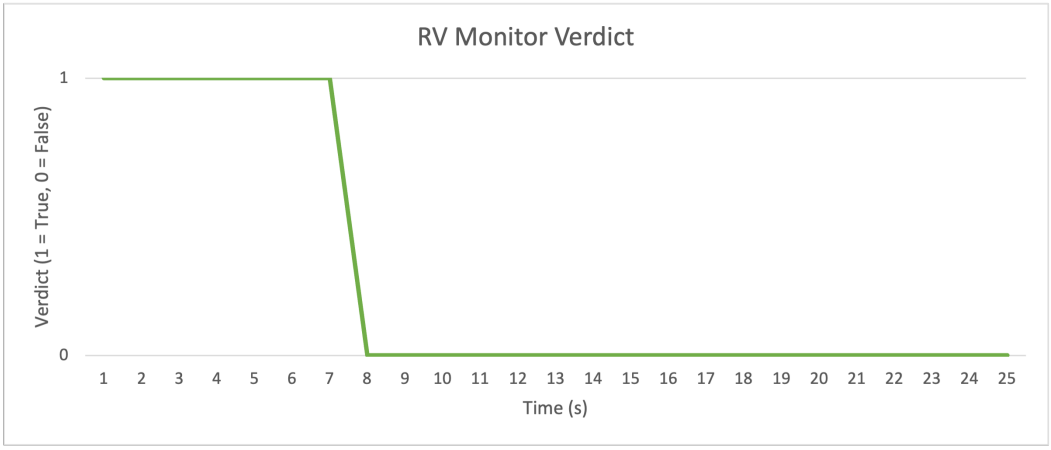

Figure 4.8: RV Monitor Verdict of Safety Distance STL Specification.

that the property of holding a safe distance is violated, as expected. We show this by the plot shown in figure 4.7 and 4.8, which accurately shows the monitors verdict changing accordingly following the safety distance STL specification, 8 seconds into the simulation. This is also noticeable by the warning the monitoring system gives when the verdict becomes currently false, giving out a warning in the topic "/monitor_error".

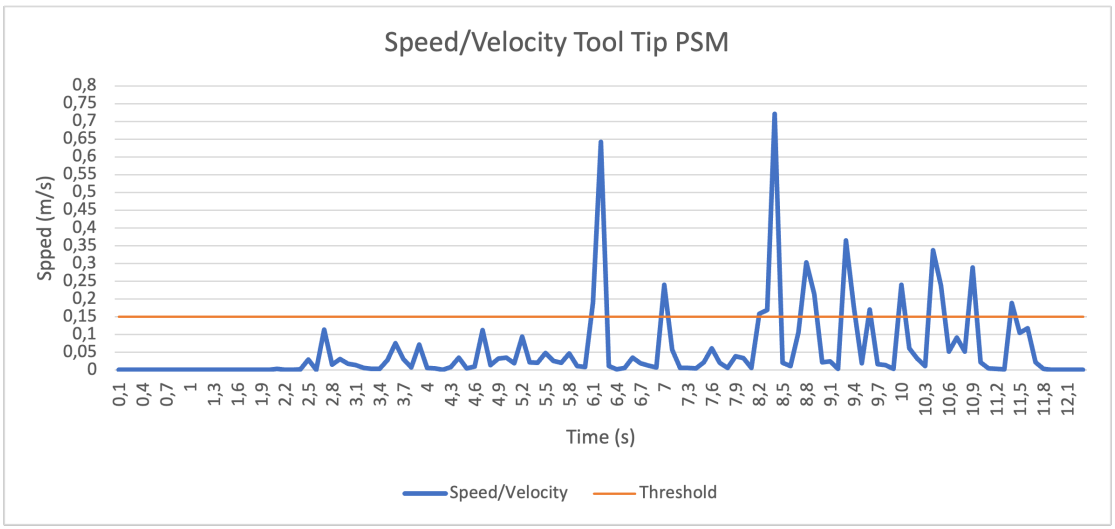

Figure 4.9: Relative Speed of the PSM's Tool Tip.

The next simulation we will run to test the workings of the RV system is testing hasty movements. As mentioned before, we do this by monitoring the velocity of the tool tip during motions. We use the same simulation setup as the one in figure 4.4 but with the tool tip being moved using the computer mouse. This is done by right clicking and holding the body that is desired to be moved and move the mouse. Here we do not assume a starting position of the PSM, as position is not monitored in this example. According to the property specified in section 4.3, the monitor should come to the verdict that the velocity property is currently true based on the current velocity of the tool tip of the PSM. As we begin in a rest state exhibiting no motions, the velocity should relatively be 0 and therefore the verdict should be currently true. We can check this again by looking at the plots in figure 4.9 and 4.10 based on the log file, which confirms our expectation.

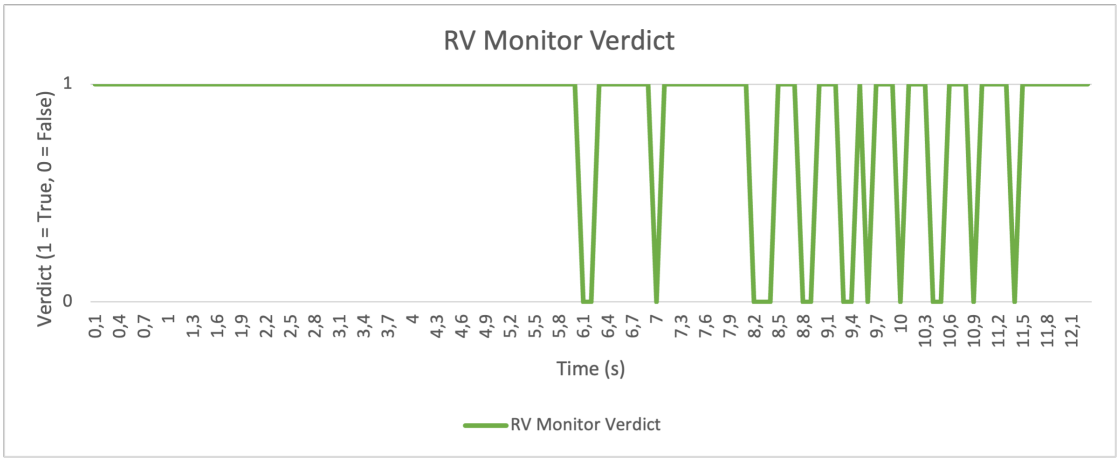

Figure 4.10: RV Monitor Verdict of Hasty STL Specification.

To check the correct workings of the monitoring system we simulate random motions at different velocities. These random motions are purposefully exaggerated to clearly demonstrate the RV system giving out warnings when the specified property in section 4.3 is violated. One of the most important reasons for this is that the PSM is not controlled with any representative controllers used in real-life surgery. Therefore trying to mimic representative hasty movements would not result in a significant result as it would not reflect and represent how the PSMs would move in real-life surgery or like mentioned before, using controllers similar to the ones used in real-life surgery. This again is represented in the log file which we plotted in figure 4.9 and 4.10, to check if the correct verdict was given during a particular velocity exhibited by the tool tip. We expect the verdict becoming currently false when the velocity becomes higher than 150 mm/s. Inspecting the plots confirms this expectation at multiple time points starting around 6 seconds and ending around 12 seconds.

Lastly, we will simulate hesitant movements to demonstrate the monitoring of the hesitant movement property and check the correct workings of the RV system on this particular example. For this simulation we use the same approach as to the hasty movements example, controlling the PSM with the computer mouse. According to the specified property mentioned in section 4.3, the verdict of the monitor should become false in the event that a hesitant movement is exhibited. This is specified as the PSM tool tip moving, then stop moving and start moving again in the time span of 1 second. If no hesitant movements are exhibited, then the verdict of the monitor should be currently true.

For checking the correct workings of the RV monitoring system we simulate the example mentioned in section 4.1. Inspecting the log file we have plotted in figure 4.11 and 4.12, we see that the verdict of the monitoring system was false for 2 times at different time steps. Manually checking the velocity plot of the PSM to the hesitant movement property, confirms the correct witnessing of the 2 violations during the simulations at 7 seconds and 8.67 seconds by the RV system.

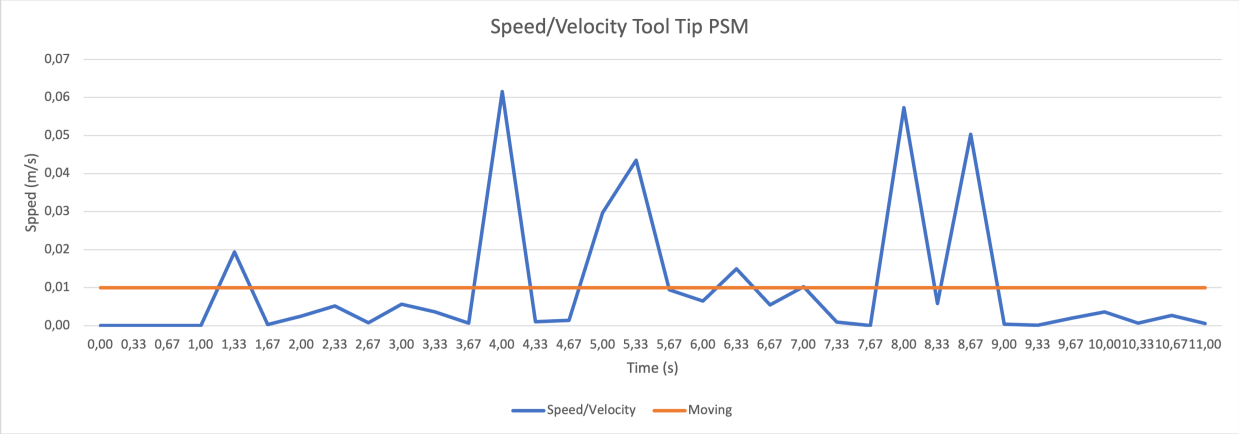

Figure 4.11: Relative Speed of the PSM's Tool Tip with the threshold for when we assume the PSM is moving.

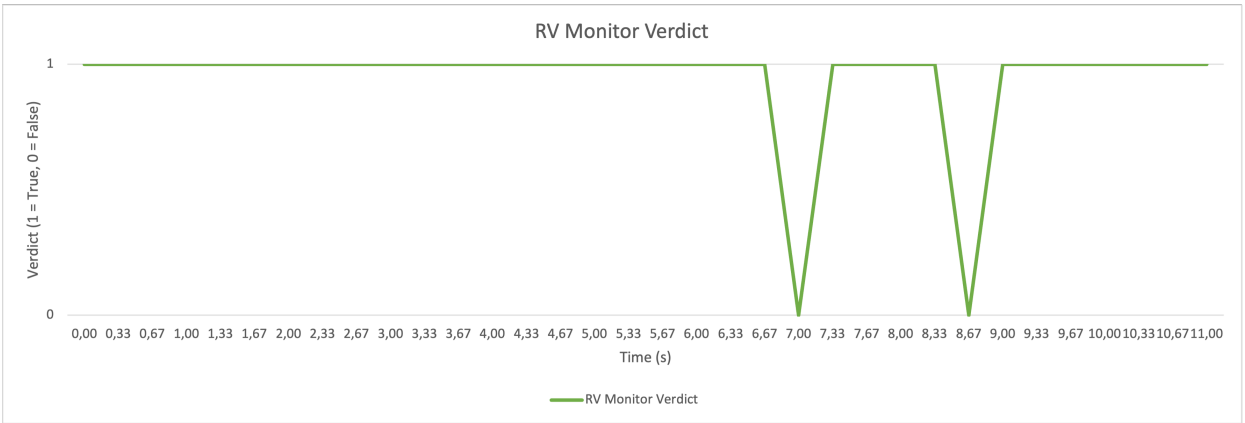

Figure 4.12: RV Monitor Verdict of Hesitant STL Specification.

5 Discussion

In the following chapter, we will discuss and interpret the findings resulting from the simulations run to check the functioning of the ROSMonitoring system. Subsequently, we will evaluate the relation between the results, the literature review and the research questions. Furthermore, we will elaborately discuss future improvements and opportunities of our designed system after we have identified and evaluated the limitations that are present in this research. Section 5.1 will go into the interpretation of the results of the simulations. Lastly, Section 5.2 will evaluate the limitations of the research based on the choices made during the research and discuss future improvements.

5.1 Interpretation of Findings

In this research, we have simulated motions and movements of the dVRK partly based on real-life surgical videos in order to mimic the scenario where we would use the real da Vinci surgical system on a real patient. Through the use of a runtime verification system that we have integrated into the simulator, we were able to monitor the simulated motions and movements of the dVRK components and therefore check the correct executions of these motions and movements. We have evaluated the correct functioning of the RV system on one real-life example, based on surgical videos, and on two synthetic examples. Results of the RV system showed that while monitoring these movements, we observe that the system accurately gives warnings on undesired behaviour. This is evident from the plots presented in section 4.4, where we observe that the verdict of the RV monitoring system becomes false and gives of a warning at the points in time where the specified property that is being checked is violated. These findings are consistent with the literature on the workings of the ROSMonitoring framework, indicating the correct integration of the framework into the simulator as well as the correct specification of the properties we wanted to have checked.

Our constructed proof-of-concept system is build from three main components. Namely the AMBF simulator, the dVRK and patients models and the ROSMonitoring framework for implementing the RV system. This construction enables an easily modified tool for checking the correctness of the behaviour of the dVRK components. From our result section, we can see that properties are easily specified and checked without modifying major simulation components. While the examples used in this research focus on the RAMIE procedure, this simulator integrated with a RV system can be extended to be used for simulating and monitoring any procedure that uses the da Vinci surgical system. How the properties are specified and the dVRK models inside the simulator are controlled remains the same. The component that needs to be replaced is the model of the patient. Models can easily be exported through Blender to be easily implemented in the AMBF simulator using the Blender AMBF add-on, which results in a versatile construction to be deployed in numerous other procedures done with the da Vinci surgical system.

In regards to what type of specifications and monitoring systems fit the da Vinci surgical system, we can observe from both the literature review as well as the research methodology into parameters for checking the behaviour of the robot that the parameters based on the kinematic model of the robot, like for example the position and speed of the components in which they move, are the most important parameters to design specifications from. As we are dealing with fundamentally a cyber-physical system, the behaviour of CPSs can be represented as a collection of metrics or signals over a continuous sequence over time [11]. These metrics and signals correspond to the kinematic model of the robot and change according to the movements and motions these systems exhibit. Therefore, the type of RV monitoring systems that fit the da Vinci surgical system are based on formal languages that can capture and represent these dynamic signals over time. The formal language suggested for the application of RAS is therefore signal temporal logic. The use of STL can subsequently be observed from the specified properties and the example simulations that were run, as described in section 4.3 and section 4.4. For monitoring the desired specified properties in these examples, we need access to the kinematic data of the models of the da Vinci surgical system, in particular the PSM, as this includes the position of the tool tip of the PSM as well as the the parameters with which we can calculate the movement speed of the tool tip. This confirms the evident need for the kinematic model of the da Vinci surgical system and STL as the formal language in order to monitor the da Vinci surgical system effectively. As we have observed from the results, the properties designed in this research can accurately monitor the simulated examples. Both the property based on real-life surgical videos as well as the synthetic examples lay a great base for future implementation for different examples and scenarios based on a variety of procedures in RAS. The arbitrary values in the synthetic example based properties can easily be replaced with realistic values once they are measured in real-life surgical procedures while the overall basis of the property is already significant. The same goes for the property based on the real-life surgical video, when interested in implementing different examples or scenarios, the values of the parameters can easily be modified to suit the new example of interest.

Ultimately, if we would reconstruct the example from the surgical video in real life, using the da Vinci surgical system with an integrated RV monitoring system based on the requirements, properties and specifications mentioned in this research, the RV system would give a warning to the surgeon before the damage to the aorta was done. Therefore, it would have most likely prevented the tool tip from damaging the aorta. The same could be done with examples of hasty and hesitant movements, which would produce the same warnings in the event the surgeon would be moving too quick or with too much hesitancy. This type of feedback could be of much help to experienced and novice surgeons in guiding them through complex robot-assisted surgeries.

5.2 Limitations and future improvements

Although we have successfully constructed a system architecture that is able to simulate simplified examples based on real-life robot-assisted surgery and check the motions of the components used in these simulations on the correctness of the execution, we need to address the limitations that are present in this research and advice on future improvements.

This constructed system architecture is intended to be the base on which can be build towards a more realistic and complex representation of simulating and monitoring the correct execution of tasks in RAS. Therefore, the components used in this research

are still relatively simplistic of nature. For example, the simulations run during this research have been done with some controllability constraints in regards to the dVRK components present in the simulation environment. Because of the fact that during this research we had no access to the physical components of the dVRK, the way that the models were controlled was restricted to using Python code to set rotations of the joint of the dVRK components mimicking multiple different positions where the transition from one position to another during the simulations resulted into motions. The other way the dVRK components were controlled is by moving them with the computer mouse. Both control options do not accurately represent the way a surgeon would control the da Vinci surgical system in real-life RAS, which is therefore a limitation of this research. However, in the event that in future research we do have access to the physical dVRK components, we can use the real manipulator instruments to control for example the PSMs in the simulation environment. In this way, controlling the da Vinci arms in the simulation environment would significantly increase the representation of how the surgeon controls these arms in real-life RAS, making monitoring the movements of real surgeons even more significant.

Another limitation of this research is the simplified digital model of the patient used in the simulations. The reason for the simplification was to first clearly present the correct functioning of the RV monitoring system. With a much more complex model, this would be more difficult to clearly present. Another reason was the lack of availability of an accurate digital model of the thorax in a suitable format to be easily imported into the AMBF simulator, this was deemed to be out of the scope of this research. As mentioned before, we have designed a simplified model of the thorax based on a schematic representation of the main anatomical structures present during the thorax phase of the RAMIE procedure. These anatomical structures are designed mostly out of cylinder-like shapes and only represent the real-life version of these anatomical structures in a simplistic way (the way they are situated amongst one another, relative scale, relatively big outgoing vessels/veins, etc.). A more complex model of the thorax would lead to a more realistic representation of simulating a real-life RAMIE procedure. Furthermore, the anatomical structures visible and worked on during the thorax phase are evidently soft tissue. Therefore expanding the current simplified model to be a soft body instead of rigid would again help in making the simulations more realistic. As mentioned in section , grasping and collisions with soft bodies in the AMBF simulator is still in an experimental phase, therefore we omitted the use of soft bodies in the simulations.

A limitation regarding the RV system, ROSMonitoring currently uses Reelay to generate monitors for checking LTL, MTL and STL properties. However, currently Reelay only supports the checking of past operators of these formal languages. Including the future operators would make for a more versatile use of Reelay. Currently, the implementation of future operators of STL in ROSMonitoring is in an experimental phase but is being worked on for future updates of the framework.

Furthermore, the amount of real-life examples from surgical videos is a limitation to this study. Ideally we want examples where tissue is damaged during some movement of motion exhibited by the da Vinci arms, however, there was no way to filter such damages for in the database of the UMC Utrecht containing surgical videos (only major complications get flagged and can be filtered on). Therefore manually looking over every surgical video for any damages made was too time consuming for the scope of this research. More real-life examples of damages in surgical videos would give a wider representation of how a RV-based monitoring system could aid in preventing such complications and help increasing patient safety.

Lastly, one of the constraints when thinking of implementing such a RV system

into the practice in real-life robot-assisted surgery is that we are missing the kinematic data about the anatomical structures of the patient. As is evident, the inside of every patient is slightly different then the other, meaning that placement or position of the anatomical structures are never the same. Furthermore, soft tissue moves around a lot during surgery. Predetermining a fixed position of the anatomical structures to avoid, will result in a highly inaccurate estimate of structures to avoid which is highly inefficient and makes the use of a monitoring system somewhat useless. So what needs to be accounted for when thinking of eventually implementing such a system in real-life is something like real-time recognition of the critical anatomical structures, as mentioned in section 2.3. Based on the annotations of the critical structures, the RV system can reason about what structures to avoid through knowing how far away they are from the da Vinci arms. Through this we can provide similar data to the kinematic data available in the simulation environment to make the monitoring system work in real-life RAS.

To conclude, with this research we attempt to increase patient safety by constructing a system architecture integrating a simulation environment, models of the da Vinci surgical system and the patient and a runtime verification system that introduces monitoring the behaviour of the da Vinci surgical system and checking the correctness of its execution. Results from this research provide a helpful and solid base that can be used for innovating the subject of robot-assisted surgery further and benefit other research into increasing the patient safety. Not only could the result of this research help when implemented in RAS controlled by a surgeon, it could also highly benefit research into automation of robot-assisted surgery.

6 Conclusion

Research into robot-assisted surgery continues to grow each year. With this trend, university hospitals and institutions are likely to join this trend by adopting the dVRK to expand the innovations in robot-assisted surgery. To get a clear view of the requirements of the surgeons that work with the da Vinci surgical system, it is important to also have a good connection with medical centers working with these systems. To help the practitioners as much as possible during surgery with ensuring patient safety, checking the correct execution and movements of the da Vinci robot during surgery will provide an extra layer of safety against damages or other complications. To achieve this goal, we can implement a runtime verification system to monitor the behaviour of the da Vinci robot. As getting access to the real da Vinci surgical system is not possible, we first needed to implement such a system with the use of the dVRK to test its functioning and efficacy.

The purpose of this research was evidently to provide the grounds for a Runtime Verification system to be used during Robot-Assisted Surgery to improve the patient safety by preventing inadvertent damages and complications during surgery. This led to the formulation of the main research question:

How can a runtime verification and monitoring system improve the patient safety during robot-assisted-surgery (RAS)?

Our constructed system architecture successfully simulated the monitoring of damage done to the aorta in the real-life surgical video and simulated the speedy movements of the PSM trying to resemble hasty or hesitant movements during RAS. The detection of this undesired behaviour in runtime through the integrated RV system would have given a warning to the surgeon to not come in closer proximity to the aorta and would potentially have mitigated and prevented the damage that was caused to the aorta. This is due to the fact that the surgeon would have known beforehand that he was going to hit and therefore damage the aorta. The ability to monitor movements of the components of the da Vinci surgical system applies to other parameters, like for example the speed of these components, aiding in preventing other undesired scenarios that could happen during RAS procedures and therefore increasing the safety of the patient in general. Resulting from the simulations we have seen that the RV system can indeed be used to check the speed and velocity of the da Vinci arms. Meaning that it can potentially be used to check certain motion behaviour like hasty or hesitant movements exhibited during RAS.

The primary contributions of this research to the topic of Runtime Verification and Robot-Assisted Surgery can be separated into three components. The first contribution is that at the moment of conducting, this research likely presents the first runtime verification system to be used in simulation systems, in particular the AMBF simulator, to check motions and movements of the components of the da Vinci surgical system. Therefore, this research lays a solid base to be build upon in future research into monitoring

motions and movements of the da Vinci surgical system using runtime verification.

Subsequently, we have integrated the ROSMonitoring framework into the AMBF simulator. Thereby enabling the feature to easily modify formal specifications for the behaviour of the da Vinci robot and generate monitors for them that can be applied to any kind of ROS-based robotic application. Therefore it can be used for all kinds of surgical procedures done with the da Vinci robot.

Lastly, this study identified specifications that can be monitored during robot-assisted surgery, by taking the RAMIE procedure as an example. Using these example specifications, future research can expand and complexify these RAS specifications to more accurately monitor the behaviour of the da Vinci robot and ultimately increase the efficacy of the RV system in increasing patient safety in general.

6.1 Future Research

To reiterate, the ultimate goal of this research is to construct a runtime verification system that eventually can be used and implemented in the real da Vinci surgical system in order to help surgeons during surgery and improve the patient safety. Now that a solid base is constructed to build upon towards this goal and has proven to be able to warn surgeons before collisions between tissue and the da Vinci system happens, the next steps can be taken towards this goal.

Firstly, getting access to a dVRK would be very beneficial in terms of controlling the PSMs and simulating different types of motions and movements inside the AMBF simulator. Having access to a dVRK means enabling the ability to connect the manipulator instruments directly to the AMBF simulator and controlling the PSMs using the same controllers as surgeons would use in real-life RAS. This way, it will be possible to conduct experiments using real experienced and novice surgeons performing example tasks and monitoring their motions and movements during runtime. Furthermore, how effective surgeons perceive the warnings that are giving off by the RV system can help potentially detect new properties to specify and check. This can be the first step to initiate adoption and presenting the usefulness to the practitioners of RAS.

Another future improvement of this research would be modifying or replacing the model of the patient used during simulations. As mentioned before, the model used in this research is simplified for the purpose to test the correct functioning of the proposed RV system. Modifying and improving the current model would aid in the representation of how the system could be used in real-life RAS. Additions like realistic structured surface and forms of the anatomical structures present in the thorax and expanding the different amounts of vessels and branches would all make the model more realistic and complex, aiding in complexity of cases the RV system could be used in. The other option is replacing the synthetic model of the thorax by a digital model generated from for example CT-scans, as mentioned in section 3.4. Implementing a digital model based on CT-scans will result in the most realistic model to be used in the simulations. However, we advice to iteratively make the model more complex to prevent issues importing the model in the AMBF simulator. Furthermore, using a CT-scan based model might be overkill to use during simulations as it might include irrelevant anatomical structures, potentially decreasing the performance of the simulations.

Lastly, for implementing the RV system in real-life RAS, we need some way to determine the position and other parameters of the anatomical structures present during surgery. As mentioned in section 5.2, when thinking of implementing such a RV system into the practice in real-life robot-assisted surgery is that we are missing the kinematic

data about the anatomical structures of the patient. A possible future implementation to solve this is integrating real-time anatomical recognition software to segment and annotate anatomical structures during RAS.

Bibliography

[1] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of the ACM*, 49(2):172–206, 2002.

[2] Autodesk. Fusion 360 — cloud powered 3d cad/cam software for product design — autodesk. https://www.autodesk.eu/products/fusion-360/overview. (Accessed on 02/2/2022).

[3] G. H. Ballantyne. Robotic surgery, telerobotic surgery, telepresence, and telementoring. *Surgical Endoscopy and Other Interventional Techniques*, 16(10):1389–1402, 2002.

[4] S. J. Baron, S. Mick, P. S. Shekar, and L. Mauri. Chapter 11 - advances in coronary revascularization. In E. M. Antman and M. S. Sabatine, editors, *Cardiovascular Therapeutics: A Companion to Braunwald's Heart Disease (Fourth Edition)*, pages 214–239. W.B. Saunders, Philadelphia, fourth edition edition, 2013.

[5] E. Bartocci and Y. Falcone. *Lectures on Runtime Verification: Introductory and Advanced Topics*, volume 10457. 2018.

[6] A. Bauer, M. Leucker, and C. Schallhart. Runtime Verification for LTL and TLTL. 20(4):1–64, 2011.

[7] R. Berguer and W. Smith. An ergonomic comparison of robotic and laparoscopic technique: the influence of surgeon experience and task complexity. *Journal of Surgical Research*, 134(1):87–92, 2006.

[8] D. Bresolin, L. Geretti, R. Muradore, and P. Fiorini. Chapter 40 Formal Verification Applied to Robotic Surgery. (July 2017), 2015.

[9] K. G. Caballas, H. J. M. Bolingot, N. J. C. Libatique, and G. L. Tangonan. Development of a visual guidance system for laparoscopic surgical palpation using computer vision. In *2020 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pages 88–93. IEEE, 2021.

[10] I. Cassar and A. Francalanza. On synchronous and asynchronous monitor instrumentation for actor-based systems. *arXiv preprint arXiv:1502.03514*, 2015.

[11] A. Donzé. On Signal Temporal Logic 40. 2014.

[12] T. R. dos Santos, A. Seitel, T. Kilgus, S. Suwelack, A.-L. Wekerle, H. Kenngott, S. Speidel, H.-P. Schlemmer, H.-P. Meinzer, T. Heimann, et al. Pose-independent surface matching for intra-operative soft-tissue marker-less registration. *Medical image analysis*, 18(7):1101–1114, 2014.

[13] S. Eslamian, L. A. Reisner, and A. K. Pandya. Development and evaluation of an autonomous camera control algorithm on the da vinci surgical system. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 16(2):e2036, 2020.

[14] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[15] Y. Falcone, S. Krstić, G. Reger, and D. Traytel. A taxonomy for classifying runtime verification tools. *International Journal on Software Tools for Technology Transfer*, 23(2):255–284, 2021.

[16] N. Feizi, M. Tavakoli, R. V. Patel, and S. F. Atashzar. Robotics and AI for Teleoperation, Tele-Assessment, and Tele-Training for Surgery in the Era of COVID-19: Existing Challenges, and Future Vision. *Frontiers in Robotics and AI*, 8(April):1–9, 2021.

[17] A. Ferrando, R. C. Cardoso, M. Fisher, D. Ancona, L. Franceschini, and V. Mascardi. Rosmonitoring: A runtime verification framework for ros. In A. Mohammad, X. Dong, and M. Russo, editors, *Towards Autonomous Robotic Systems*, pages 387–399, Cham, 2020. Springer International Publishing.

[18] A. Gallioli, A. Territo, R. Boissier, R. Campi, G. Vignolini, M. Musquera, A. Alcaraz, K. Decaestecker, V. Tugcu, D. Vanacore, S. Serni, and A. Breda. Learning curve in robot-assisted kidney transplantation: Results from the european robotic urological society working group. *European Urology*, 78(2):239–247, 2020.

[19] N. Haouchine, J. Dequidt, I. Peterlik, E. Kerrien, M.-O. Berger, and S. Cotin. Image-guided simulation of heterogeneous tissue deformation for augmented reality during hepatic surgery. In *2013 IEEE international symposium on mixed and augmented reality (ISMAR)*, pages 199–208. IEEE, 2013.

[20] J. He, E. Bartocci, D. Ničković, H. Isakovic, and R. Grosu. Deepstl – from english requirements to signal temporal logic, 2021.

[21] D. A. Hendrickson. Complications of laparoscopic surgery. *Veterinary Clinics of North America: Equine Practice*, 24(3):557–571, 2008. Surgical Complications and Management Strategies.

[22] IntuitiveFoundation. The da vinci research kit. https://www.intuitive-foundation.org/dvrk/.

[23] IntuitiveSurgical. Robotic surgical systems — da vinci — ion — intuitive. https://www.intuitive.com/en-us. (Accessed on 19/1/2022).

[24] A. M. Jarc, S. H. Shah, T. Adebar, E. Hwang, M. Aron, I. S. Gill, and A. J. Hung. Beyond 2d telestration: an evaluation of novel proctoring tools for robot-assisted minimally invasive surgery. *Journal of Robotic Surgery*, 10(2):103–109, 2016.

[25] M. Kent and J. Luketich. Minimally invasive esophagectomy. *Frantzides CT, Carlson MA, eds. Atlas of Minimally Invasive Surgery.*, 2009.

[26] B. F. Kingma, M. F. G. de Maat, S. van der Horst, P. C. van der Sluis, J. P. Ruurda, and R. van Hillegersberg. Robot-assisted minimally invasive esophagectomy (ramie) improves perioperative outcomes: a review. *Journal of Thoracic Disease*, 1(1), 2018.

[27] A. Kowalski. Pathway to Artificial Pancreas Systems Revisited: Moving Downstream. *Diabetes Care*, 38(6):1036–1043, 05 2015.

[28] C. C. Krasnoff, A. Grigorian, B. R. Smith, Z. Jutric, N. T. Nguyen, S. Daly, M. E. Lekawa, and J. Nahmias. Predictors of anastomotic leak after esophagectomy for cancer: Not all leaks increase mortality. *The American Surgeon*, 87(6):864–871, 2021. PMID: 33233922.

[29] P. Kumar and B. Ravi. A comparative study of robots in laparoscopic surgeries. In *Proceedings of the Advances in Robotics 2019*, pages 1–6. 2019.

[30] C.-H. Kuo, J. S. Dai, and P. Dasgupta. Kinematic design considerations for minimally invasive surgical robots: an overview. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 8(2):127–145, 2012.

[31] A. Lindfors, Å. Åkesson, C. Staf, P. Sjöli, K. Sundfeldt, and P. Dahm-Kähler. Robotic vs open surgery for endometrial cancer in elderly patients: surgical outcome, survival, and cost analysis. *International Journal of Gynecologic Cancer*, 28(4), 2018.

[32] J. A. Lott, P. Moser, M. Gebski, M. Dems, M. Wasiak, and T. Czyszanowski. Energy-efficient vcsels for integrated optoelectronic and photonic systems. In *2016 IEEE 6th International Conference on Photonics (ICP)*, pages 1–3. IEEE, 2016.

[33] A. Madani, B. Namazi, M. S. Altieri, D. A. Hashimoto, A. M. Rivera, P. H. Pucher, A. Navarrete-Welton, G. Sankaranarayanan, L. M. Brunt, A. Okrainec, et al. Artificial intelligence for intraoperative guidance: using semantic segmentation to identify surgical anatomy during laparoscopic cholecystectomy. *Annals of Surgery*, 2021.

[34] M. R. Maddah. 3D visualization and interactive image manipulation for surgical planning in robot-assisted surgery Mohammad Reza Maddah To cite this version : HAL Id : tel-02007359 Mohammad Reza MADDAH 3D Visualization and Interactive Image Manipulation for Surgical. 2019.

[35] P. Mascagni, A. Vardazaryan, D. Alapatt, T. Urade, T. Emre, C. Fiorillo, P. Pessaux, D. Mutter, J. Marescaux, G. Costamagna, et al. Artificial intelligence for surgical safety: automatic assessment of the critical view of safety in laparoscopic cholecystectomy using deep learning. *Annals of Surgery*, 2021.

[36] G. N. Moawad, J. Elkhalil, J. S. Klebanoff, S. Rahman, N. Habib, and I. Alkatout. Augmented realities, artificial intelligence, and machine learning: clinical implications and how technology is shaping the future of medicine. *Journal of Clinical Medicine*, 9(12):3811, 2020.

[37] K. Moorthy, Y. Munz, A. Dosis, J. Hernandez, S. Martin, F. Bello, T. Rockall, and A. Darzi. Dexterity enhancement with robotic surgery. *Surgical Endoscopy and Other Interventional Techniques*, 18(5):790–795, 2004.

[38] A. Munawar. *Implementation of a Surgical Robot Dynamical Simulation and Motion Planning Framework*. PhD thesis, Worcester Polytechnic Institute, 2015.

[39] A. Munawar, Y. Wang, R. Gondokaryono, and G. S. Fischer. A real-time dynamic simulator and an associated front-end representation format for simulating complex robots and environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1875–1882, Nov 2019.

[40] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In *Runtime Verification*, pages 21–37. Springer, 2015.

[41] R. Plantefeve, I. Peterlik, N. Haouchine, and S. Cotin. Patient-specific biomechanical modeling for guidance during minimally-invasive hepatic surgery. *Annals of biomedical engineering*, 44(1):139–153, 2016.

[42] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[43] ROS. Ros: Home. https://www.ros.org/. (Accessed on 02/23/2022).

[44] J. Ruurda, P. C. van der Sluis, S. van der Horst, and R. van Hilllegersberg. Robot-assisted minimally invasive esophagectomy for esophageal cancer: A systematic review. *Journal of Surgical Oncology*, 112(3):257–265, 2015.

[45] C. Sánchez, G. Schneider, W. Ahrendt, E. Bartocci, D. Bianculli, C. Colombo, Y. Falcone, A. Francalanza, S. Krstić, J. M. Lourenço, D. Nickovic, G. J. Pace, J. Rufino, J. Signoles, D. Traytel, and A. Weiss. A survey of challenges for runtime verification from advanced application domains (beyond software). *Formal Methods in System Design*, 54(3):279–335, 2019.

[46] P. M. Scheikl, S. Laschewski, A. Kisilenko, T. Davitashvili, B. Müller, M. Capek, B. P. Müller-Stich, M. Wagner, and F. Mathis-Ullrich. Deep learning for semantic segmentation of organs and tissues in laparoscopic surgery. *Current Directions in Biomedical Engineering*, 6(1), 2020.

[47] A. Schimpf, S. Merz, and J.-G. Smaus. Construction of büchi automata for ltl model checking verified in isabelle/hol. In *International Conference on Theorem Proving in Higher Order Logics*, pages 424–439. Springer, 2009.

[48] C. Science. Kinematic calibration for da vinci surgical robot. 2019.

[49] D. L. Shettko. Complications in laparoscopic surgery. *Veterinary Clinics of North America: Equine Practice*, 16(2):377–383, 2000. Endoscopic Surgery.

[50] K. M. Siddiqui and D. M. Albala. Robotic-assisted surgery and treatment of urolithiasis. *International Journal of Surgery*, 36:673–675, 2016. Renal Stones: Causation, Investigation and Modern Management.

[51] P. Sluis, E. Tagkalos, E. Hadzijusufoviç, B. Babic, E. Uzun, R. Hillegersberg, H. Lang, and P. Grimminger. Robot-assisted minimally invasive esophagectomy with intrathoracic anastomosis (ivor lewis): Promising results in 100 consecutive patients (the european experience). *Journal of Gastrointestinal Surgery*, 25, 02 2020.

[52] M. Soudan. Improving the trajectory planning, velocity control, and position accuracy of the da vinci robotic minimally invasive surgical system for material testing. *ZSR Library*, 2020.

[53] D. Ulus. Online monitoring of metric temporal logic using sequential networks, 2019.

[54] S. van der Horst, T. J. Weijs, J. P. Ruurda, N. H. Mohammad, S. Mook, L. A. A. Brosens, and R. van Hillegersberg. Robot-assisted minimally invasive thoracolaparoscopic esophagectomy for esophageal cancer in the upper mediastinum. *Journal of Thoracic Disease*, 9(8), 2017.

[55] J. A. Van Koughnett, S. Jayaraman, R. Eagleson, D. Quan, A. van Wynsberghe, and C. M. Schlachta. Are there advantages to robotic-assisted surgery over laparoscopy from the surgeon's perspective? *Journal of robotic surgery*, 3(2):79–82, 2009.

[56] A. Zemmar, A. M. Lozano, and B. J. Nelson. The rise of robots in surgical environments during covid-19. *Nature Machine Intelligence*, 2(10):566–572, 2020.

[57] M. Zhou and C. G. L. Cao. Vibrotactile feedback improves laparoscopic palpation skills. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 53(11):738–742, 2009.