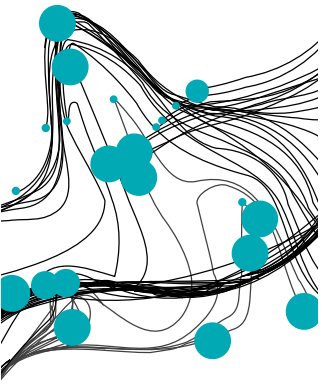
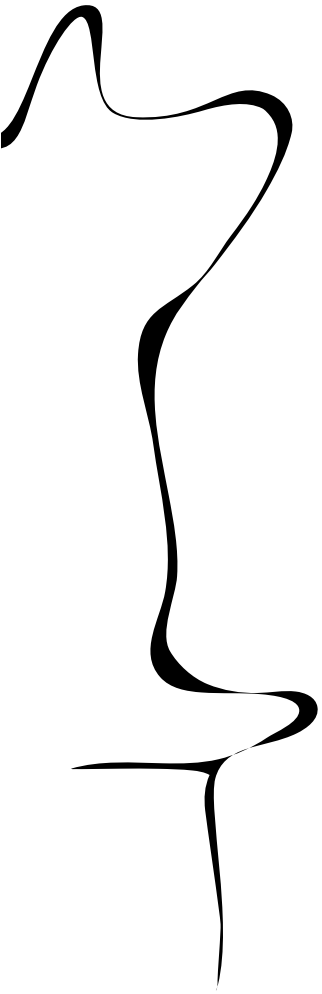


RAM ROBOTICS AND MECHATRONICS



MR NAVIGATION FRAMEWORK FOR ROBOT-ASSISTED ENDOVASCULAR INTERVENTION

J.B. (Jelle) Bijlsma

MSC ASSIGNMENT

Committee:
prof. dr. ir. S. Stramigioli
dr. G. Dagnino
dr. ir. W.M. Brink

November, 2021

073RaM2021
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Abstract

Endovascular diseases are the number one cause of death in the western world. From atherosclerosis to stroke, there are many forms of this disease. Over the years, minimal invasive surgery (MiS) has proven to be superior in many cases to traditional open surgery. For MiS, many robotic platforms have been created to increase the repeatability, accuracy and decrease patient recovery time. Many of these platforms have been proven successful under X-ray fluoroscopy. This imaging method brings danger of ionizing radiation to the patient and the surgeon, thus new imaging methods are researched. In 2021, the Imperial College London developed a robotic platform (CathBot) that is safe to use during Magnetic Resonance Imaging (MRi).

In this thesis, the port of CathBot from X-ray to MRi is approached, and comparing the pro's and cons' of MRi, a framework is created that tries to connect the robot to the MRi device. An imaging strategy is devised to detect one instrument of minimal invasive endovascular intervention (EvI), the guidewire, during cannulation of the abdominal aorta. Using traditional image processing and a combination of low sensitivity matching and blob finding, we are able to detect on average 68% of specific markers on the guidewire, with a standard deviation of 0.74 millimeter, and a bias of 4.16 millimeter, and calculate the distance from the guidewire to the vessel wall. The imaging can be done on a single core in real-time, at 40 frames per second on a mid/high end workstation.

Contents

1 Introduction

Robotic surgery to the layman might seem like a futuristic and theoretical concept. It is however, very real. The history of robotic surgery started as early as the 1970's when NASA was examining concepts of tele-operation for astronauts in need of surgery. The concept of a doctor on earth operating on an astronaut in space however, never came to fruition [1]. The first practical application of robotic surgery was in 1985 when an industrial robot was re-purposed to perform a brain biopsy [2]. The robot was able to outperform human operators in terms of needle positioning accuracy. Since then, the discipline of robotics has seen large growth and has branched into many different areas of medicine. An important distinction to make is between autonomous robotic surgery and assisted robotic surgery. In the former, the motion of the surgical instruments is dictated by a computer. The latter still sees a human operator controlling the instruments, but the instruments are now actuated by a robot. For this thesis, the focus is on assisted robotic surgery. Thus, for the sake of simplicity, these words will be used interchangeably.

In this thesis, **a navigation framework using magnetic resonance imaging for robot-assisted endovascular intervention** is proposed. In this section, it will be explained why there is a need for such a framework, and why endovascular procedures are performed in the first place. For a more in-depth explanation of what these terms mean, the reader is referred to section 2, in which these concepts are explained more thoroughly. Besides that, in this section the goals of the thesis are laid out, an approach to solving the problem is given and the section is concluded with an outline of the report.

1.1 The Need for Endovascular Procedures

Circulatory diseases are the main cause of death in Europe, responsible for 37% of all deaths, followed by different forms of cancer at 26% [3]. Endovascular intervention is a form of minimal invasive surgery to treat different vascular and cardiac diseases. Minimal invasive surgery has a shorter recovery time and produces fewer complications than traditional methods. By virtue of an increased understanding and continuous development, surgery success rate for MIS has increased, and anesthesia use is reduced [4].

1.2 Robotics in Endovascular Procedures

The reason for introducing robotics to endovascular intervention is twofold, the first being an increase in dexterity for the surgeon. Not manipulating instruments correctly can lead to severe consequences for the patient, from increased recovery time, failed surgery, and even death. This increased dexterity can be accomplished in two different forms, namely by lever-effects and inclusion of additional sensory systems. The former principle scales a motion down linearly, which can help to reduce positioning errors and tremors in human operators. The latter is done by interpreting data from sensors and encoding this in forms of different signals. For example, the operator could receive haptic, visual or audible feedback by the system.

The second reason to introduce robotics is tele-operation, which circumvents one of the obstacles of the current form of imaging, fluoroscopy. Fluoroscopy introduces harmful ionizing radiation for both patient and surgeon. Although wearing protective lead-filled aprons, the risk for specific types of cancer and cataracts is three times higher in surgeons who perform endovascular interventions [5]. Using tele-operation, the surgeon can be physically relocated to a position outside of the operating theatre. This reduces the risk of ergonomic problems due to the heavy, lead-filled clothing, and it eliminates the increased cancer rates for interventional surgeons.

Special attention is given to the CathBot [6]. This robot was designed for endovascular Surgery and was developed at Imperial College London. It allows tele-operation and very precise movements of the guide-wire, more information on it is given in section 2.3. A navigational framework has been developed for Cathbot at Imperial College as well. This framework has been proven to work under fluoroscopy in animal testing. It can detect a guidewire tip position, and detect vessel walls. It then is able to calculate the tip distance to the wall and provide haptic feedback to the surgeon.

Further to CathBot, there are several commercial and research robotic platforms for EvI available [7]. Some opt for the use of steerable guidewires [8], others use active tracking with specialized guidewires [9]. Some platforms combine pre-operative MR scanning overlays during fluoroscopy to aid the surgeon [10], however CathBot is unique in using standard guidewires, with a passive tracking and a fully MR-compatible platform. Now by introducing a framework for haptic feedback to the surgeon, CathBot has the potential to be the first robotic platform for EvI under MR.

1.3 Introducing Magnetic Resonance to Endovascular Procedures

During surgery, the patient is subjected to ionizing radiation from the fluoroscopy machine. Advanced techniques in low-dose imaging combined with relatively short exposure time, still make endovascular interventions a more safe approach generally than open surgery. However, fluoroscopy is still contra-indicated in patients with a young age, due to children being more susceptible to the long term effects of ionizing radiation. Besides radiation, a contrast fluid has to be injected into the patients bloodstream to outline the vessels due to the low soft-tissue contrast of fluoroscopy. This contrast fluid only shows the vessels during a short amount of time. The contrast fluid is nephrotoxic [11] and thus can only be injected during critical points in the surgery. For the remainder of the time, the surgeon has to remember the anatomy or use a superimposed image. Due to patient motion, either breathing or the pulsatory motion of the heart, vessel wall location may differ from previously detected locations. Contrast fluid (gadolinium) for MR also exists, but it has been shown that the abdominal aorta and aortic arch are large enough and have enough contrast on their own to allow to be imaged without any contrast agents [6]. It is to be noted that gadolinium remains in the body long after surgery, and the effects on the body are unknown.

For this project, an alternative to the traditional fluoroscopy will be considered. Magnetic resonance imaging (MRI) is based on the excitation of hydrogen atoms in a strong magnetic field by radio frequencies (more detailed information is available in section 2.2). It does not involve ionizing radiation and excellent soft-tissue contrast can be achieved without contrast fluid. Besides the removal of health hazards, MRI is also able to produce 3D images. In MRI, arbitrary image planes can be selected for scanning. Combining multiple image planes creates a 3D view of the scanned object. For endovascular intervention this means that the surgeon can determine the catheter tip position in all three dimensions during MRI, compared to only two in fluoroscopy.

However, there are also downsides to using MRI. MRI is mainly used for imaging and scanning, these procedures involve relatively long acquisition times compared to those needed in interventional procedures. Improving the temporal resolution comes at the cost of spatial resolution, and a compromise needs to be found between these two. The strong magnetic fields produced by the MR scanner also prevent the use of conventional robotic manipulators. Thus, specially designed actuators have to be used, as explained in section 2.3. This is due to the fact that traditional manipulators use electric motors or other magnetic materials, and would thus catastrophically accelerate to the MR-scanner. Finally, the last downside of MRI is the cost compared to fluoroscopy, with the initial purchase price of the machines differing a factor 5.

1.4 Problem Description & Goal

The main reasoning as to why the original statement: *"a navigation framework using magnetic resonance imaging for robot-assisted endovascular intervention"* is relevant, should now have been answered. By asking what this means, the statement is narrowed down and proper development can proceed. The framework will be created with application to the CathBot in mind, however, the framework is general and could be applied to other robots as well. What is meant by endovascular intervention should be narrowed down as well. For this thesis, we are only considering the movement of a guidewire inside of a phantom model of the abdominal aorta. This will be described in more detail in chapter 4. Finally, the magnetic resonance imaging boundaries are set. Within the University of Twente, there is access to a 1.5T Siemens Aera, and thus this device is used to collect images and test the framework.

The goal of this thesis is thus to replicate the framework created for the CathBot by Imperial College, but now under magnetic resonance (MR), since the previous framework works with fluoroscopy. This goal can be broken down into three steps:

- Vessel Segmentation in MR.
- Real-time Pose Estimation of the guidewire in 2D space.
- Calculation of wire-wall distance and tip angle

By knowing the 2D state of the guidewire, as well as that of the vasculature, the framework allows for vision-based haptic guidance in any robotic application under MR.

The original goal was to implement 3D pose estimation as well. This however, required more time than was available and was thus scrapped. The proposed solution has been attached in appendix B: 'COPS'.

1.5 Outline

This concludes the introduction of the thesis. In the following section, a thorough theoretical analysis of all the relevant material is given, this includes endovascular intervention, magnetic resonance imaging, CathBot and navigational frameworks in literature. Following up in section 3, the problem is analyzed in depth. In section 4, the methods and materials are explained. In the next section, the segmentation software is broken down on an abstract level. This is followed up with a quantitative analysis of the obtained results. Section 6.5 provides a discussion on the goals that were removed from the thesis due to them being unattainable within the current time restraints and points to clues for future research. The final section is the conclusion, which summarizes the results and provides a closing note.

2 Background and Resources

The reader experienced in the topics described in the following sections can skip them without missing out on the progression of the thesis. They have been written for readers with different expertise to catch up with the basic required knowledge to understand design considerations and choices made during the coming sections.

2.1 Endovascular Intervention

The concept of Endovascular Intervention (EvI) is to perform surgery from within the arteries, instead of from the outside. Many types of EvI exist, some of the more common procedures are: Aneurysm repair, stent placement, angioplasty and treatment of thrombosis, embolism and stroke. This means that during EvI, an artery is cannulated, depending on what region of the body needs to be accessed. Access to the vasculature is obtained through the axillary artery when the neck or head is concerned, and the iliac artery when dealing with the torso. From here, the surgeon can introduce a guidewire into the artery and navigate this to a specific location. Now, depending on the specific type of surgery, different tools can be slid along the guidewire to perform the surgery. EvI is however a very challenging procedure, even for expert surgeons. When excessive force is applied to the vessel walls by the rigid guidewires, complications can occur, such as inflammation, thrombosis, perforation and internal bleeding [12]. Because of this, multiple robotic platforms have been created to assist and support the surgeon during MIS [7].

2.1.1 Imaging under EvI

In contrast to traditional, open surgery, endovascular intervention requires a method of visualization since the catheters and guidewires are hidden from view. The de-facto method is X-Ray fluoroscopy.

Fluoroscopy provides high resolution, both temporally and spatially [10]. The surgeon has access to the fluoroscopy images in real-time during surgery and navigates the vascular structure based on the current catheter position with respect to the vasculature. As mentioned in section 1.3, to properly visualize the vascular system, a toxic substance has to be injected into the veins. This limits the time in which the surgeon is able to see the vasculature. Otherwise, only the surgical instruments and bones are visible. Placing the patient between the X-ray receiver and sender causes the creation of a 2D image. The intensity at each pixel represents the transmission coefficient of the material above the receiver, as shown in figure 2.1. This implies that under fluoroscopy, it is impossible to tell where a guidewire would be under a bone (radio-opaque), but a stack of translucent materials would be identifiable.

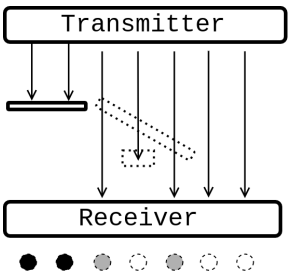
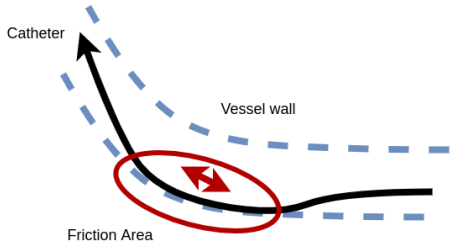


Figure 2.1: 1D X-ray, displaying a 2D space mapped onto a discrete 1D space.

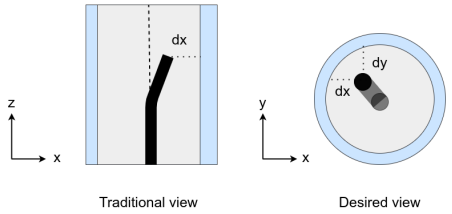
2.1.2 Complications in Navigation under EvI

The main risks specific to EvI are damaging the inner lining of the vessels, caused either by sliding friction of the instruments against the walls (figure 2.2a) or direct perforation of the vessel. Guidewires generally are made out of Nitinol due to their super-elasticity. The super elasticity allows for the catheter to be navigated around a bend, as shown schematically in figure 2.2a, however this sliding contact causes friction during back and forth movement and is a source of damage that can not be avoided. Steps are taken during the construction of the guidewire, such as coating the wire in polyurethane to reduce sliding friction. The surgeon has to keep a

close eye on the position of the tip and the full length of the guidewire. Due to the projective nature of fluoroscopy, the distance to the wall can only be determined in one axis (figure 2.2b). This means that one dimension is 'hidden' from view for the surgeon.



(a) The inner walls (Endothelium) of the vessels are very sensitive to mechanical force.



(b) The left view is the classic, top-down view that the surgeon has during intervention.

Figure 2.2

As displayed in figure 2.2b, the projective method lacks information in the third dimension (Y), which is just as crucial to collision avoidance as the second (X). Methods have been established to counteract this such as c-arm rotation to generate 3D images from 2 perpendicular X-ray scans [13]. Although safer in terms of tip-wall contact, the patient is exposed to a greater amount of radiation and the procedure takes longer due to adjustment of the c-arm.

2.2 Magnetic Resonance Imaging

Volumes have been written on the theory behind magnetic MRI [14], and it would be impossible to give an encompassing description of the physics behind it. In this section, a simplified working principle will be given and the concepts of scan planes are introduced which serve an important purpose later on.

2.2.1 Basic Theory

MR imaging is based on the phenomenon of magnetic resonance which allows manipulation of 1H nuclei in strong magnetic fields. Within this homogeneous field, a 'gradient field' is created. The gradient field makes that the resonance frequency of proton spins become dependent on their position along this gradient. This allows to excite a specific 'slice' by transmitting a radiofrequency pulse with the corresponding frequency content. The same relations are then used to differentiate the spins within the excited slice, by a process called "spatial encoding".

Due to the construction of the gradient field and the spatially and phase encoded signal, the resulting measured signal can be Fourier transformed, which produces a 'picture' that ideally shows spatially the concentration of hydrogen nuclei.

However, this ideal representation can be thrown off by many different processes, such as field inhomogeneities. These inhomogeneities can be caused by implants and are mostly unwanted. However, they can also be introduced on purpose. In MR compliant guidewires, small paramagnetic rings are introduced. They are used to locate the guidewire by creating a large signal void. [15].

2.2.2 Imaging Planes

The gradient field creates the imaging plane which results in the output of the MR scan. In traditional imaging, these images are combined to create a 3D scan. For interventional purposes, due to the strict requirements on temporal resolution, it is custom to rely on a single imaging frame. To still be able to see the whole vasculature, the slice thickness needs to be increased, see figure 2.3.

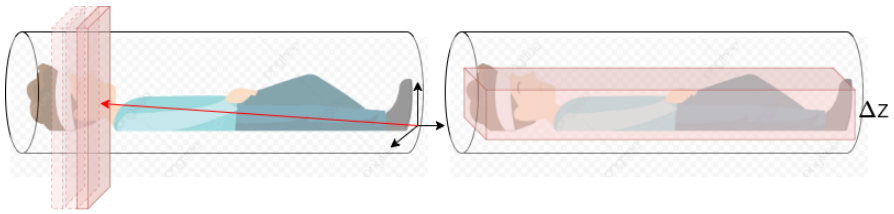


Figure 2.3: The slice thickness, or ΔZ can be adjusted. The left image shows how a 3D brain scan would be performed. By taking multiple subsequent thin slices (images) of the brain, these images are later combined into a 3D model. The picture on the right shows a more interventional view, which is similar to taking an X-ray, by virtue of the larger slice thickness. (Image source 'patient clipart PNG' designed by 浩 and taken from https://pngtree.com/)

By increasing the slice thickness, a larger part of the patient is visible, and since slices can be arbitrarily selected, one can now have access to regions that were first blocked by radiopaque material. Compare figure 2.1 to 2.4.

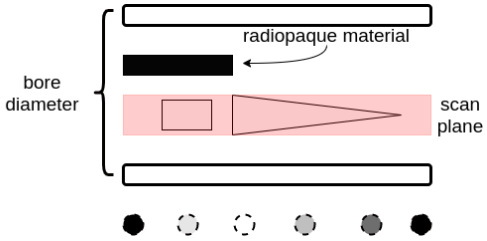


Figure 2.4: A 1D diagram to display imaging in MR. Inside the bore are three objects. A scan plane is constructed in an area that contains a square and a triangle. The radiopaque slab is not visible, since it is outside the scan plane. The rectangle is visible. In the triangle, the amount of signal received depends on the 'amount' of triangle contained in the scan plane.

There is however a trade-off between slice thickness and contrast. The 3d rectangle which consists of the gradient field, needs to be projected to a 2D image. This is done by averaging over the Z direction. A short but strong signal gets averaged out in a thick slice. See figure 2.4.

2.3 Cathbot

A closer look is taken at the Cathbot [6]. This tele-operated type of robot is unique in two ways: The master manipulator is operated in the same manner a surgeon has to perform during a non-robotic procedure. This is in contrast to robots in which the surgeon has to manipulate a joystick or other input device. By utilizing his or her previously trained motor skills, no expensive time is spent on training. Secondly, the robot is equipped with haptic feedback.

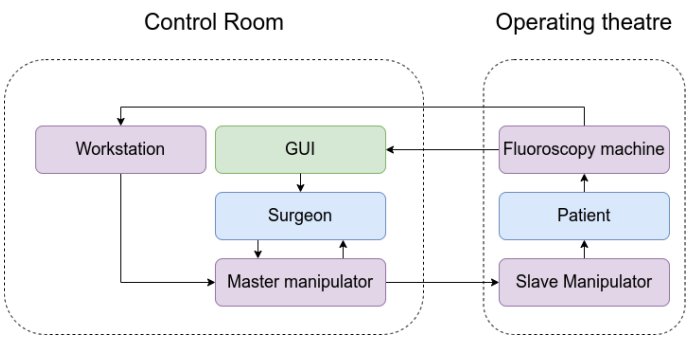


Figure 2.5: By using tele-operation the surgeon can be situated in the control room, minimizing radiation exposure and increasing ergonomic comfort.

By using a tele-operated system, the ease of manipulation is increased, yet there is no tactile feedback, when the catheter tip is in contact with the vessel wall. Surgeons indicate that this is a major limitation of robot operated systems [16]. By introducing haptic feedback, the amount of contact between tip and vessel wall is reduced [17]. The current Cathbot system uses fluoroscopy data to provide haptic feedback to the surgeon. The data is passed to a workstation that can segment the image, and provide the distance between the guidewire tip and vessel wall. This distance is then used to generate a damping force in the master manipulator, providing feedback to the surgeon. The process schematic is shown in figure 2.5.

2.3.1 MR Safe

Due to the strong magnetic fields involved with MRI, ordinary actuators can not be used to manipulate the catheter. Because of this, robotic MR-guided intervention was not possible for a while due to the precision of MR-safe actuators being too low. One would have to use piezo- or pneumatic actuated stepper motors, which could not satisfy the requirements needed for endovascular intervention. This changed with the new pneumatic steppers introduced by [18] at UTwente. Using these, building on the work of [19, 20], Imperial College developed a MR-safe version of the Cathbot [6].

2.4 Guidewire/Catheter Tracking in Literature

Currently, all endovascular interventions are done with the help of fluoroscopy. Most existing commercial applications of introducing robotics into endovascular interventions are also geared towards catheter tracking in fluoroscopy. The general form of the tracking algorithms has moved from classical image processing techniques to learning based approaches in the last years. These learning approaches have higher confidence and accuracy than traditional approaches. The cost of this method is that large datasets are required to train the data. These data sets are much more readily available for fluoroscopy due to fluoroscopy being a more widely accepted surgery method.

For MR, besides the methods of tracking available to fluoroscopy, a unique MR method is available. Referred to as active tracking, it uses a small coil in the guidewire and specialized software to accurately determine the instrument position. It has the benefit of being accurate, fast and reliable, at the cost of increased instrument size [21, 22]. When imaging guidewires, the actual wire is of a smaller size than the MR scanner is able to provide resolution for in a real-time setting [23]. This would mean the guidewire will not be visible during MR. However, MR-compatible guidewires include para-magnetic markers in them. These para-magnetic markers create local signal voids which can be detected, see figure 2.6. These markers have a distinct shape and are much larger than the actual wire, thus allowing for the detection of the guidewire with resolutions below the width of the actual instrument.

(a) **(b)** **(c)**

Figure 2.6: Subfigure a,b & c show the signal voids at different signal intensities. These images were taken from captured data.

3 Analysis

This section of the thesis is dedicated to analyzing the current environment surrounding the research topic. This means the current expertise available at the Robotics and Mechatronics (RaM) lab of the University of Twente, the tools available, and how the project is set up. It also states the requirements for the project and the different goals that are set.

3.1 Endovascular Intervention in RAM

In RAM, there are no previous groups currently working on endovascular intervention under MR. There are groups working on EvI and collecting data using fluoroscopy. No frameworks exist yet for doing the same under MR, and it has to be set up from the ground. This does however, give a lot of freedom in creating a new framework. It is essential to narrow down and not get lost in the excess of possibilities to chase.

Because there is no previous experience with EvI under MR, there are no datasets available of either manual or robotic interventions under MR. This limits the use of learning procedures and material available. Thus, classical image manipulation tools are used.

3.1.1 Cathbot at UTwente

Currently, there is no device available at TechMed for robotic endovascular intervention. However, the construction of a replica of the MR-safe Cathbot is in process. The estimated time of arrival is in January. This means that there is no possibility of integrating the MR-framework with a working replica of the Cathbot at the end of the thesis. Data acquisition will thus be done using manual motion of the guidewire in the phantom.

3.2 Material & Software Requirement

There are two material requirements posed, them being:

- The catheter type is required to be passive.
- No gadolinium contrast fluid.
- Acquire images using TechMed's 1.5T Aera Scanner.

The passive catheter requirement is set to increase the availability of the system and reduce the overall complexity. No gadolinium, as stated in section 1.3, the size of the vessels involved should provide enough signal to be readily observable. As mentioned earlier, the images used for analysis have to be acquired. This will be done by the MR scanner available at TechMed centre at the University of Twente together with the help of the Magnetic Detection and Imaging team (MDi) of the University of Twente. On the software side, the requirements are:

- Usage of Python
- Real-time processing on modern hardware

The choice for Python was the result of a consensus of the people currently working on vessel segmentation within the CathBot framework at RAM. It is open-source, easy to read and has rich image processing libraries and there are possibilities to expand to learning approaches for the segmentation.

3.3 Project Goals

Due to the limited time available with the MR scanner, most of the processing will be done offline. Different sequences will be recorded during the acquisition steps, and these will be saved in memory. The first step is to load these images into the workstation memory at a fixed rate, to emulate actual scanning. Secondly, image segmenting should be performed. This segmenting consists of segmenting the image into two parts, first, the vessel walls and secondly the guidewire and tip. The natural question to arise is, which images are analyzed? To follow routine clinical procedures, the first set of images are taken of the lower abdominal aorta. This coincides with the phantoms currently available at TechMed. From these segmentations, the distance between the full length guidewire and vessel-wall can be calculated.

3.4 Project Setup

A Framework will be created which can segment an MR-safe guidewire from phantom vessel walls and calculate the distance between them. To keep the objective narrowed down within the limited time frame, only the guidewire will be of consideration and no other medical devices are taken into account. The Abdominal artery will be taken as a starting point, with the iliac artery as a point of insertion. The imaging plane used will be along the phantom. This gives an image appearance similar to that obtained in fluoroscopy.

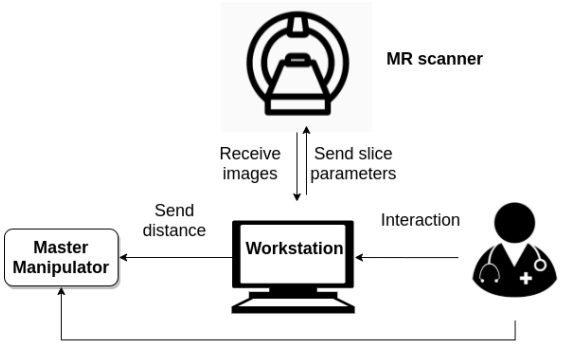


Figure 3.1: Interactive workflow between the different parties and components involved.

It is important to realize that to use the system in a clinical setting, the data created and the segmented images have to be presented in a way the surgeon can interact properly with the system, as shown in figure 3.1. To facilitate this, a graphical user interface (GUI) is created. This GUI is made with PyQT, a python binding for QT. Using this GUI, the surgeon can interact with the framework. Ideally, the workstation should be able to receive images directly from the MR scanner, this however, has not been achieved due to time constraints.

4 Data & Methods

In this section, the materials used in the data acquisition are detailed. Their characteristics and dimensions are given. For this thesis, two separate visits to the Techmed centre were planned for imaging purposes. The settings during these visits are described and detailed.

4.1 Phantom Model

To acquire images that resemble an endovascular intervention, it is ill-advised to ask for test subjects, thus a model of a subset of the arteries is used, referred to as 'the phantom'. The phantom, as seen in figure 4.1, that was used is property of RAM. The phantom has connectors to simulate blood flow, but these were closed off since there was no pump available.

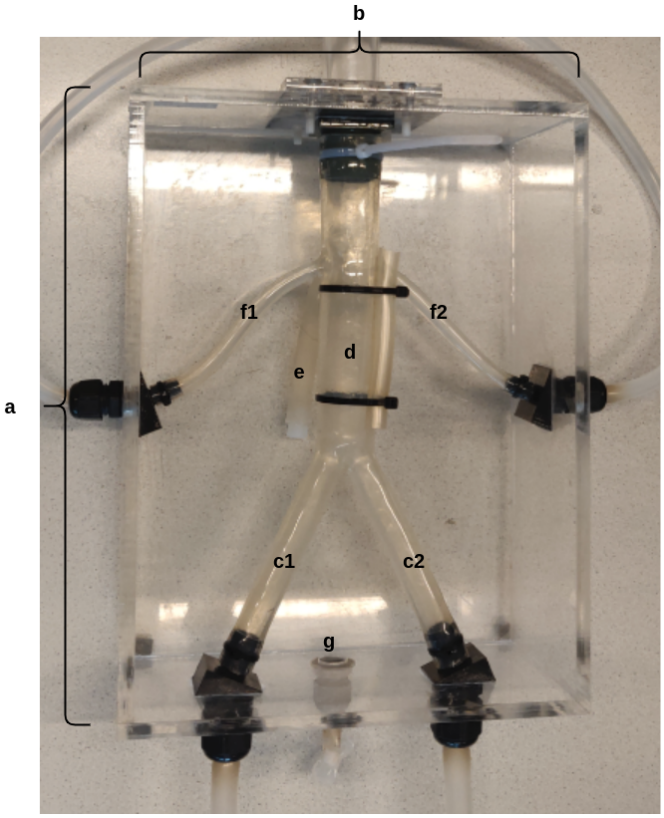


Figure 4.1: The phantom used for data acquisition. The box is 30cm (a) by 23cm (b). The illiac arteries (c1,c2) are 19mm in diameter. The abdominal aorta (d) is 31 mm in diameter. The renal arteries (f1,f2) are also present, as well as the superior mesenteric (e). The surrounding box can be filled through g.

The guidewire used during the scanning is an EP-FLEX mr-line guidewire (EPflex Feinwektechnik GmbH, Dettingen an der Erms, Germany). The guidewire is 0.89mm thick and consists of an inner core of braided fibers, which is coated with composite and PTFE, as seen in figure 4.2. The tip is covered with a large para-magnetic marker, followed by 5 short and evenly spaced markers. The markers afterwards are spaced farther apart. The markers are cylindrical and surround the braided fiber.

The images are acquired using a Siemens 1.5T Aera scanner located at TechMed (figure 4.3. The scanner has an Access-I interface which could later be used to communicate directly with the navigation framework. This is currently not yet researched properly and thus during the acquisitions in this thesis, the images were acquired, and then transported from the MR workstation to the computer running the imaging framework using a pen-drive.

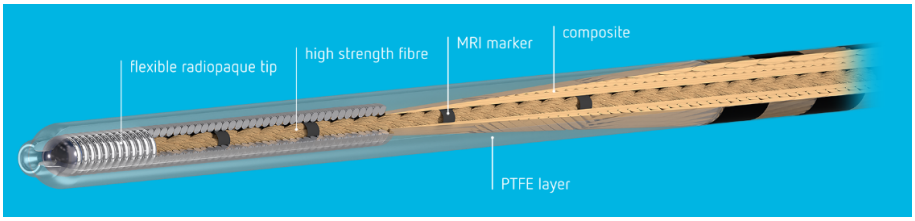


Figure 4.2: The inner structure of the MR-Flex guidewire, taken from https://www.epflex.com/en/products/mrline/

Figure 4.3: The 1.5T Siemens scanner located at TechMed, with the phantom placed on top of the bed.

4.2 First Scan Opportunity

During the first scanning opportunity, the phantom was filled with tap water (Enschede, Netherlands), and placed in the MR scanner. The 'body' coilset was used. Endovascular intervention was new for the researchers at MDI as well, thus by trial and error the following scan parameters were obtained.

- Scanning Sequence: 'Gradient Recalled'
- Slice Thickness: '8.0mm'
- Repetition Time: '45.0ms'
- Echo Time '1.26ms'
- Flip Angle '67.0'
- Pixel Size: '1.25mm2'
- Field of View: '165*240 mm2'

These scan parameters were chosen to have good contrast between the markers and the background, whilst keeping the temporal resolution high (1 image per second). The aim is to have

a temporal resolution of at least 3 images per second since this is also the speed achieved in interventional fluoroscopy. Two scans were made with these parameters, further referenced as scan 1 and scan 2. Scan 1 is a stationary scan, with no movement of the guidewire. During scan two, the guidewire is being partially retracted from the phantom.

4.3 Second Scan Opportunity

During the second scan opportunity, the phantom was filled with Manganese(II) chloride, to simulate a more realistic signal intensity to blood than with ordinary water. To increase the segmentation challenge, surrounding area was filled with Manganese(II) chloride to, to simulate tissue. The main problems with the scan 1 & 2 were the trailing observed during movement (see section Results), the guidewire moving out of plane, and the low framerate. To combat these problems, the following scan parameters were used.

- Scanning Sequence: 'Gradient Recalled'
- Slice Thickness: '20.0mm'
- Repetition Time: '216.84ms'
- Echo Time '1.22ms'
- Flip Angle '75.0'
- Pixel Size: '1.346mm^2'
- Field of View: '210*280 mm^2'

5 Segmentation Software Breakdown

In this section the navigation framework will be broken down in its individual components and explained on an abstract level. Later in Appendix A, the code is broken down on the 'line-to-line' level.

A general overview of the navigation framework can be found in figure 5.1. The user interacts with the GUI, and through it, has access to the Dicom-processor, which is explained in the next section. The user can also interact with the video-player (following section). The video player consists of two modules, the playback module and the image processing module. There are possibilities to communicate with CathBot and MR-scanner, these however are not implemented.

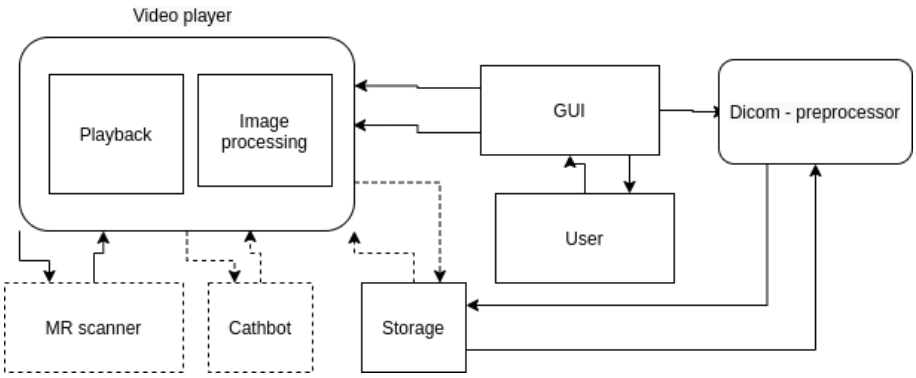


Figure 5.1: Conceptual overview of the different framework elements working together.

The general idea of the image processing is described in figure 5.2. The idea is to first create a mask, which separates the arteries from the rest of the image, then to detect the signal voids created by the para-magnetic markers. The initial image is presented and then preprocessed (gray-level slicing + filter). Then an edge finding algorithm is used (canny). By then applying morphological operations on the image, a closed contour can be created of the arteries. Then, selecting a point manually, the interior of the contour is determined. A flood filling operation is performed, and now we have generated a mask of the arteries. The masked image is now processed again and using a detection algorithm, the signal voids of the guidewire are detected. These points then can be used to calculate the distance from the wall to the marker-point, as well as the wall-tip angle.

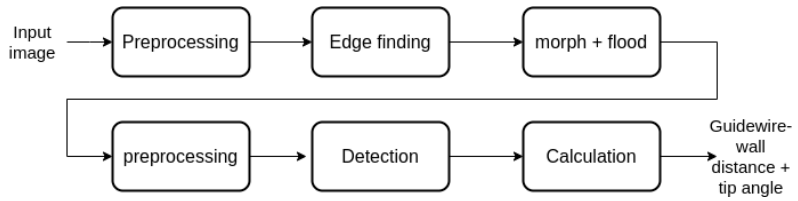


Figure 5.2: Flow diagram summarizing the image processing steps of the framework.

5.1 Dicom to PNG (Dicom Processor)

Each image is generated on the MR-scanner as a .Dicom file. Dicom files are standardized files for medical imaging to facilitate the transfer of files between radiologists, doctors and different devices from different manufacturers. Besides the actual file, the Dicom-header contains information on the patient, the radiologist and the scanner.

The collection of Dicom files is transferred from the MR-station to the general workstation by pen-drive. Using the *pydicom* package, the images are converted into .PNG files. These PNG files can be loaded into memory using conventional python image processing libraries. This procedure has been integrated with a GUI (PyQT) to optimize the workflow. Besides PNG's a .mp4 is also created.

5.2 Video Player

The video player consists of two parts, the playback element and the image processing element. The former facilitates parameter tuning during image processing, as well as provides direct visual feedback. The video-player element loads in all the frames to a specific scan in memory and presents them to the user. The video can be played, paused, speed up, slowed down, and the image processing steps can be applied in real-time. The video player tries to simulate an image-stream from an MR-scanner.

The image processing element takes in user input from the GUI and applies filters and algorithms on the initial image to perform eventual segmenting and distance calculations. The steps are explained in the next section.

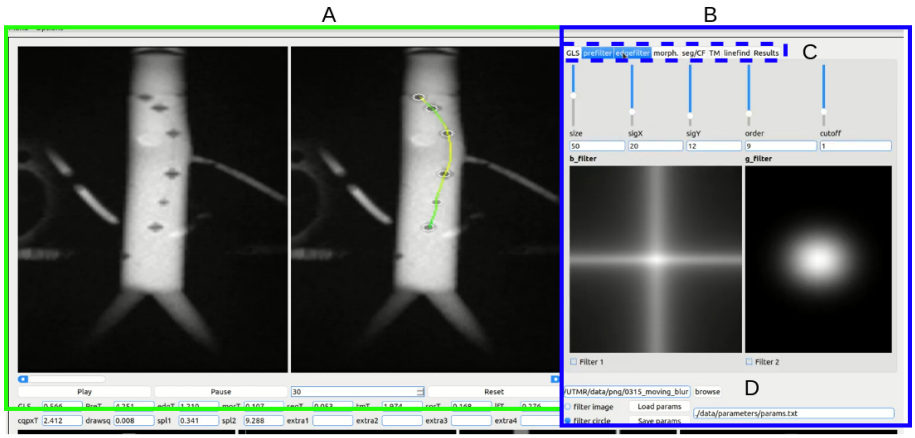


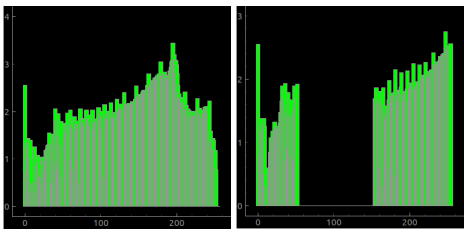
Figure 5.3: The GUI of the framework. (A) is the video player, showing left the original image and right the output image. In this specific picture, the output is fully segmented and the guidewire is found and the distance to the wall is calculated. (B) is the image processing menu, it allows for manipulation of parameters of the selected operations, such as in this case filter size and strength. (C) is the operations tab, in which different operations (explained in the next sections) can be performed.

5.3 Gray-Level Slicing

The first step in the image processing is to edit the histogram. The images acquired by the MR-scanner are monochrome 8bit images. This means they can be represented by a nxm matrix, with values ranging from 0 to 255. It is possible to change the total brightness (adding or subtracting a constant to the entire image), as well as taking a specific range of brightness values and adding or subtracting a constant, these two operations are defined as gray-level slicing. Finding the correct values is trial and error and depends on the general image characteristics of the specific scan.

5.4 Filtering

The next step is to filter the image. The framework uses a Gaussian filter, to smoothen the image and reduce noise in the edge detection step. The filtering is done in the frequency domain for speed considerations and thus allows for more filters to be applied easily later on since the framework supports time to frequency conversion of filters.



(a) **(b)**

Figure 5.4: The effects of gray-level slicing can be observed on the histogram. Before (a) and after general boost -20 and a +100 to the range of 60 to 250.

5.5 Creating a Mask

First, the Canny edge detection algorithm is used, this will provide an outline to the arteries. This turns the monochromatic 8-bit into a binary image. By subsequently applying a dilation and erosion operator, using a square 7x7 pattern (see figure 5.5, remaining holes in outline of the artery can be closed. This now provides a closed contour of the image.

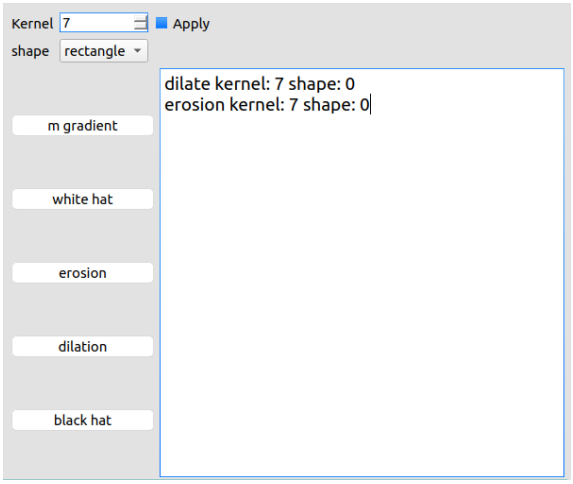


Figure 5.5: The framework provides a custom morphological operations menu in which different morphological operations can be applied subsequently using different kernels.

After obtaining this closed contour, the inside of the contour needs to be determined, this is done manually. Now the inside is known, the closed contour is filled. This results in the mask, a binary image which represents where the arteries (approximately) are. The mask is of interest since it prevents noise/structures outside the artery to be picked up by the detection algorithm, and the closed loop allows us to do guidewire-wall distance calculations.

By multiplying the mask with the original image, we have segmented the image into artery/non-artery. From here, it is possible to re-apply the gray-level slicing and filtering, however, this turned out not to be necessary.

5.6 Template Match & Blob-Finding

On the masked image, template matching is first applied. Using the GUI, the templates can either be selected live, or loaded from previous sessions (see figure 5.6a. The framework supports up to 4 templates. By settings the confidence interval low, many points will be selected around the same para-magnetic marker. By later averaging out over these points, the correct location is determined. This is done due to the low contrast acquired especially in scan 3 and 4. The template matching tab also provides sliders for setting the different 'zones'. These zones refer

to the guidewire-wall distances, and determine the color with which the guidewire is drawn. This is explained in section 5.8.

(a) Template matching menu **(b)** Blob finder menu

Figure 5.6: Parameters can be quickly edited and templates selected in real-time using the GUI.

Next, to supplement the template matching, a blob finding algorithm is used. This is done since the template matching alone is inadequate in spotting all the markers. The blob-finding parameter menu is displayed in figure 5.6b.

5.7 Sorting and Splicing

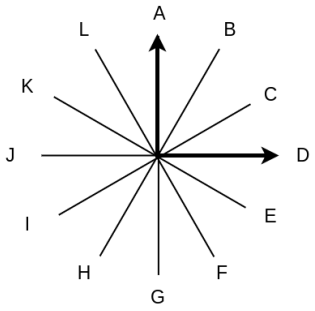
The previous blob-finding and template matching have generated large numbers of potential matches, and to successfully rebuild the guide-wire, the points first need to be collected and grouped per magnetic marker. This is done by:

- List all un-grouped points
- Select random point b_0^n
- Determine the distance to every point
- Every point with distance $< q$ is regarded as grouped belonging to b^n.

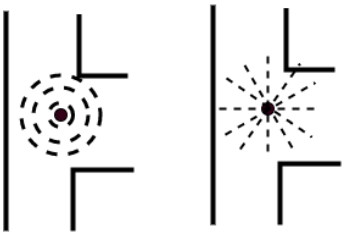
This is repeated until there are no un-grouped points left. Now the groups b^n are replaced by the points q where q^n is the average of each group. This results in a collection of points, representing the guidewire. Now sorting $q^n(x, y)$ on their y value, a quadratic spline can be fit through the points. This allows for interpolation and increases the available points by a factor 10. These interpolated points will be used later when calculating the wire-wall distance and tip-wall angle.

5.8 Wire-wall Distance & Angle Calculation

After the image has been segmented, the pixel distance between the guidewire and the vessel wall can be calculated for each point. This could be done naively by expanding a circle from a point in the guidewire, and accessing the mask at each coordinate to find if the value is equal to one (figure 5.7a). This however, is computationally too heavy to do for all the (interpolated) points of the guidewire. A new strategy is devised, by expanding specific easy-to-calculate points (figure 5.7b). By taking lines at 90, 60, 30, and 0 degrees respectively, only two values need to be calculated for each radius (table 5.1), and the amount of look-ups decreases as well.



(a) The new sparse lines drawn with the points labeled by letters.



(b) The circle algorithm has the highest precision, while the accuracy of the sparse lines drop with increasing radius. However, the accuracy is of most importance when the distance is small, so the decrease in accuracy is negligible.

Figure 5.7

By approximating the square root as well, the calculations result in float multiplications. These steps result in fast enough distance calculations to be able to run in real-time.

Point	A	B	C	D	E	F	G	H	I	J	K	L
X	0	a	b	r	a	b	0	-b	-a	-r	-a	-b
Y	r	b	a	0	-b	-a	-r	-a	-b	0	b	a

Table 5.1: The coordinates of the lines described in figure 18. $a = \sqrt{\frac{3}{2}}r, b = \frac{1}{2}r$. As seen, only two values need to be calculated, when taking r as the required radius.

The wall-tip angle is of importance because it signifies the severity of wire-wall distance is small. The guidewire approaching the wall at a perpendicular angle is at a greater risk of perforation than when they are parallel to each other. The wall-tip angle is calculated as follows:

- Draw a bounding box around the guidewire-tip of bxh. (The framework provides tools to set bxh
- Take the extreme points of the guidewire that lie within the bounding box, (q^1, q^2), as seen in figure 5.8
- Calculate the angle of the wire with $\theta = arctan(\frac{dy}{dx})$
- Do the same for left wall and right wall.

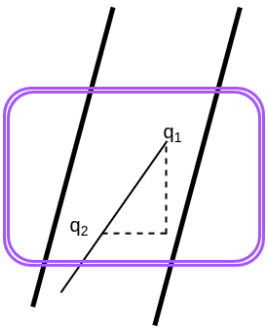


Figure 5.8: Bounding box (purple) delineates the cutoff for the angle calculation. The beginning and end points (q_1, q_2) are used to determine the angle.

Predefined boundaries can be set for wire-wall distance and visual feedback to the surgeon. By outlining three different risk regions, *low, medium, high*, a color can be assigned to a line segment. When the distance d is greater than low-risk, it is green. In between low and medium,

it takes the form:

$$\begin{aligned} rc &= -1/(low - medium), \\ b &= 1 - rc * medium, \\ scale &= round((d * rc + b) * 255, \\ RGB &= [scale, 255, 0]. \end{aligned} \tag{5.1}$$

And thus the color scales from green to yellow. A similar trick is done to scale the colors from yellow to red when going from medium to high. When the distance is smaller than high danger, the color will always be red.

5.9 Ground-Truth

To test the results of the framework, a test with ground-truth data is required. This ground truth data is collected through a custom program, which allows the user to zoom in, and select centre points of each signal void. After selecting all the necessary points, the user can move to the next image with the arrow keys or on screen button. During any point in the labelling, the user can save progress and return at any time by loading in the saved progress.

5.10 Data Comparison

After determining the ground-truth data, the results need to be compared with the results determined by the framework. The data will be interpreted in the next section, but in this section it will be explained how the data was generated. The challenge is to effectively sort the points from the ground truth and the points by the framework in the following categories: **Correct, Incorrect, Missed**. The diagram in 5.9 displays and explains the sorting algorithm.

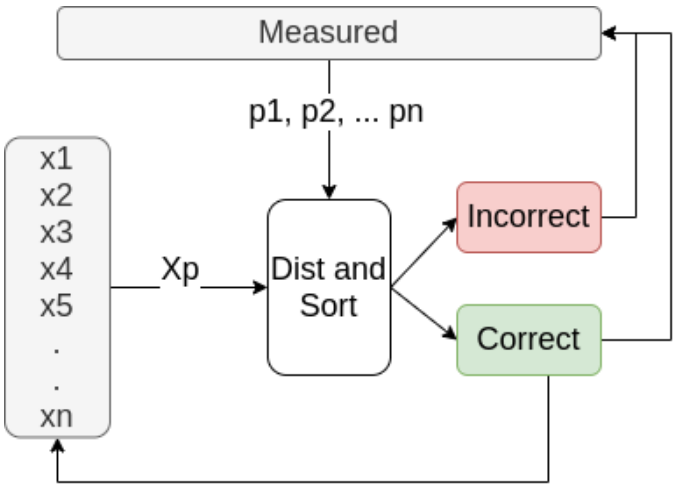


Figure 5.9: From the true points ($x_1 ... x_n$), one point X_p is selected. The distance to each generated point $p_1 ... p_n$ is calculated. If the distance between the two is smaller than a specified cutoff value (correct), the generated point is assigned to the true point, X_p and the assigned point are popped from their respective lists, and they are marked as Correct. If the distance is larger, the true point will be marked as 'Missed'. Then the process is repeated. If the true points run out before the measured points, the remaining measured points are labeled incorrect, vise versa, any remaining true points will be labeled as missed.

5.10.1 Replicating the Results

To replicate the results and run the different programs for yourself, follow the instructions in Appendix C: 'Running from Source'.

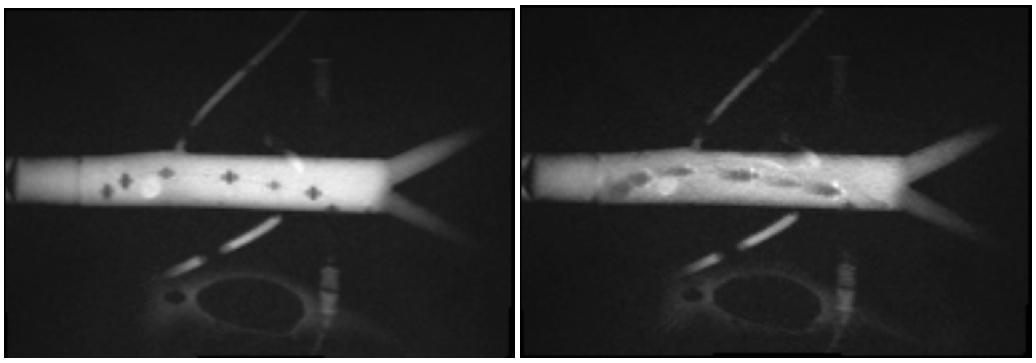
6 Results & Discussion

In this section, the results of the two scanning runs are described. To re-iterate, a total of 4 scans were analyzed for this thesis, they were acquired on two separate occasions, which are referred to as run 1 and run 2. These runs require slightly different image processing parameters for the segmentation to work optimally due to the different scan parameters used in obtaining the images. The image processing parameters are stored in a binary file stored in the code-base. Run 1 uses *run1.pcl*, and run 2 uses *run2.pcl*.

6.1 Run 1 Visual Analysis

To re-iterate, during scan 1 the guidewire was kept stationary, during scan 2 the guidewire was pushed in before being retracted slowly. Inspecting the images generated during run 1 visually, the following observations can be made:

- With the slice thickness being 8mm and the abdominal aorta 31mm in diameter, it is clear that the positioning of the wire will indicate signal strength. For example, looking at figure 6.1a, the 5th MR-marker is less in-plane than the other markers, due to its smaller size and less distinctive size
- Due to incomplete filling and slight angle of the phantom, the renal- and iliac arteries are not properly filled. This causes signal loss and makes the abdominal artery the only fully visible artery
- Strong motion artifacts occur when moving the guidewire.



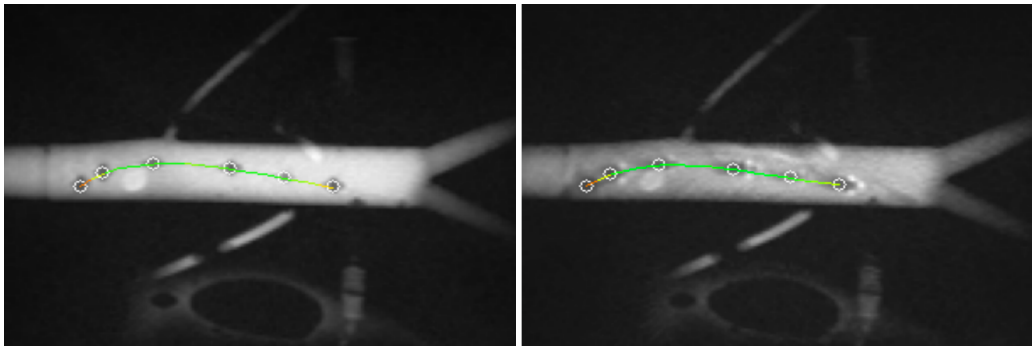
(a) Stationary **(b)** Moving

Figure 6.1: Imagery of the phantom with two different guidewire states, notice the trailing. The object on the bottom of both images is an IV bag filled with water, this was placed there to increase the received signal and bypass low signal errors generated by the MR. This would not be an issue during clinical settings because the volume of the phantom is low, and the container of the phantom was empty.

Now the segmentation software is run on scans 1 and 2. Two screenshots of the results were taken to showcase the working, please refer to figure 6.2. The full frame by frame analysis is presented in section 6.3.

6.2 Run 2 Visual Analysis

During run 2, the trailing effect was completely mitigated by replacing the tap-water with a copper magnate solution. The slice thickness was also increased to reduce the guidewire going out of plane. The rest of the container also was filled with a Manganese(II) chloride solution, which explains the general increase in brightness.



(a) Stationary **(b)** Moving

Figure 6.2: As seen in both images, the color of the guidewire is determined by the distance to the perceived wall. In both stationary and moving situations the markers that are connected with the wall can not be found.

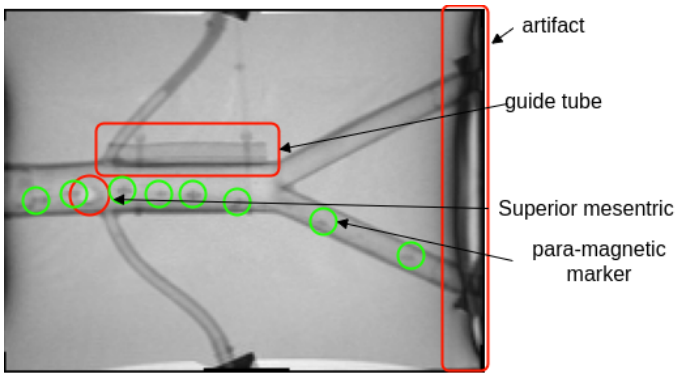


Figure 6.3: During run 2, the general visibility was increased, at the cost of the contrast.

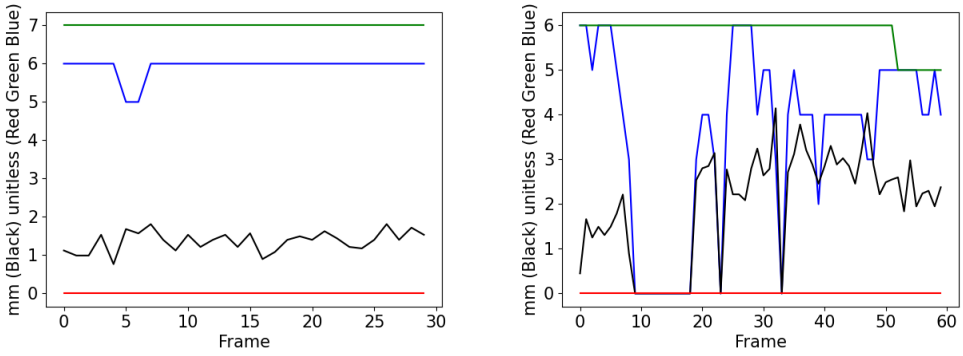
Due to the increased slice thickness and the increase in acquisition time (1 second to 0.25 seconds) the image contrast is greatly reduced. This can be readily seen in figure 6.3. The guidewire is annotated with green circles, and can be hardly distinguished from the background, in contrast with run 1. The superior mesenteric artery also provides a strong imaging artifact, and an aliasing artifact is also visible. Initially, the container and phantom were filled with water. Then, the container was filled with copper magnate solution, and a guide tube (see figure 6.3) was attached to compare the effects of trailing in water and copper magnate.

6.3 Run 1 & 2 Numeric Analysis

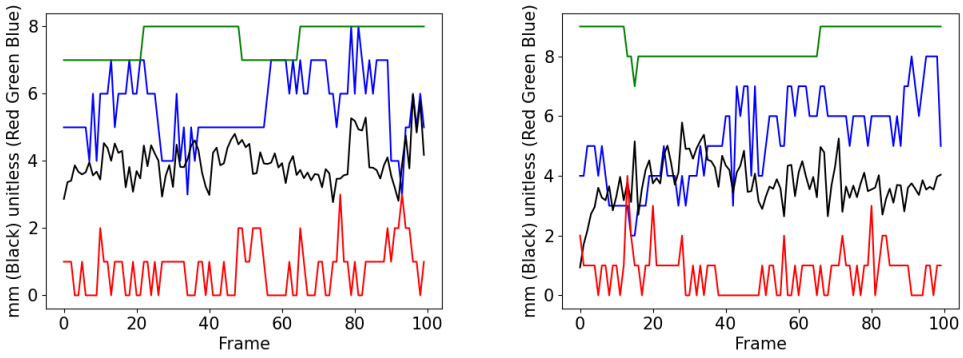
Using the framework, with the sorting algorithm as explained in section 5, the following results are obtained:

Figure 6.4, contains the data on correctly identified points, missed points and faulty points as well as the average distance between the ground truth and the measured point. From this data, the following can be observed. Run 1 had no incorrect labels, and in the static case, a 1-pixel average deviation. During scan 2 however, there was a period of time in which no markers were detected. This happened because the guidewire was in contact with the wall, and thus the signal void became indistinguishable from the wall, as seen in figure 6.5. This caused the guidewire to be temporarily detected as a wall and thus giving zero detections.

Furthermore, when comparing run 1 to 2, it is observed that due to the lower contrast, the sensitivity threshold of both blob finding and template matching had to be lowered and thus the number of false positives increases.



(a) Scan 1 **(b)** Scan 2



(c) Scan 3 **(d)** Scan 4

Figure 6.4: Results from matching the true values to the measured values. **Green line** indicative of the total amount of True points, **blue line** indicates total correctly matched points, **red line** indicates total incorrectly labeled points. The **black line** is the average distance (mm) between the true value and the measured value. The x-axis labels each elapsed frame.

Further data analysis (table 6.1) is done. The standard deviation and mean are calculated from the average pixel error. Also, to check whether or not the measurement samples can be drawn from a normal distribution, we do a Shapiro-Wilkes test on the run. If the errors are not normally distributed, the run is not statistically significant. For extra confirmation of the validity of the run, a T-test is performed. This test can show whether or not the errors in the run are drawn from the same distribution, i.e. repeatability within the measurements exists. The results show that run 1 and 2 should be excluded from analysis since they do not provide enough statistical evidence (Shapiro Wilkes > 0.1). Run 3 and 4 can be used, and they have been drawn from a similar random distribution, as seen from the low score of the t-test. The standard deviations are 0.60mm and 0.78mm, which means the algorithm is precise. The accuracy however is not

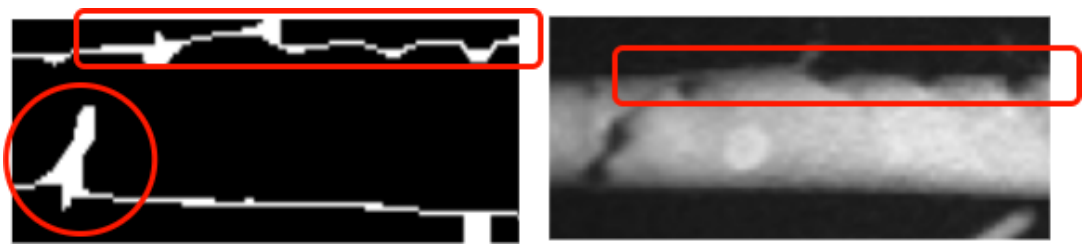


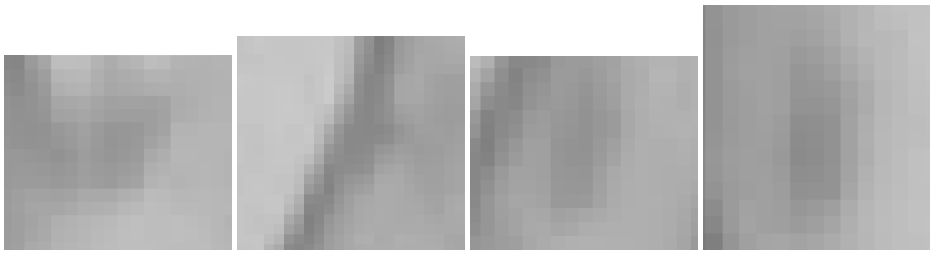
Figure 6.5: Due to heavy trailing and touching of the walls, the edge-finding failed momentarily. On the left, the mask can be seen and on the right, the actual MR image.

as good, due to the mean error of 3.95 and 3.78 mm respectively. There could be a problem with the manual labeling due to the inconsistency of the shapes and the difficulty of selecting the appropriate middle of the marker, as figure 6.6, shows. Current labeling software does not allow for sub-pixel marking. This sub-pixel marking could potentially increase accuracy.

When analyzing scan 3 and 4 the percentage of identified markers is on average lower, and the amount of wrongly identified markers much higher, indicating that the contrast is a determining factor in segmentation accuracy.

Run	Std. dev. (mm)	mean (mm)	RMSQ (mm)	Shapiro Wilkes	T-test	Correct	Wrong
1	0.27	1.36	0.25	0.4170		84.76%	0%
2	1.21	2.00	0.38	0.3822	5e-13	61.7%	0%
3	0.60	3.95	0.40	0.061		74.26%	11.36%
4	0.78	3.78	0.39	0.056	0.085	61.84%	12.66%

Table 6.1: Standard deviation, Mean error, Root mean square error, Shapiro Wilkes test, T-test between run 1-2 & run 3-4.



(a) **(b)** **(c)** **(d)**

Figure 6.6: Para-magnetic markers that have a less cross shape figure and for which determination of the centre is a bit more complicated. In (a,b) the interaction between the marker and vessel wall show how they complicate correct measurement and in (c,d) it is observed that even in more 'free' space, where the marker center is, is not well defined.

6.4 Discussion

The results in the thesis can be hardly compared to other work in the field, since there is not a lot of work done on passive segmentation of guidewires in MR images. Most of the work is on fluoroscopy for interventional surgeries. Examples of techniques used in X-ray fluoroscopy are: Vandini [13]is able to determine guidewire position using priors based on the high quality and invariant imaging, generated by x-ray, as seen in picture 6.7.

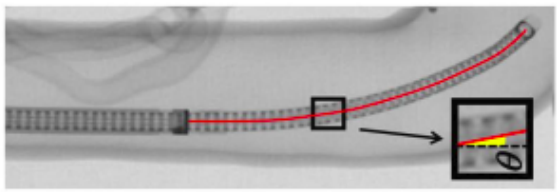


Figure 6.7: The strong contrast and higher resolution of X-ray allows for very sophisticated analysis. Taken from [13]

.

Due to the limited data available in this procedure, learning procedures as suggested by Ma [24] are not possible as Ma user over 10.000 labeled images to train his algorithm. To combat these

limitations, others, such as Bock and Schirra [22][9], use active instruments, i.e. fitted with a coil to locate the device. The closest work available is by Van der Weide [25]. They received a higher accuracy 98%, compared to 68% obtained, however, their acquisition time was 8 times larger and the geometry was chosen such that using a slice thickness of 5mm was enough to encompass the whole vasculature.

By limiting the scope to only guidewire segmentation in the abdominal aorta, the difficulty is greatly reduced compared to the generalized problem. However, given the current limitations of sparse data-points, low resolution and low contrast, the results are still good enough for clinical usage and thus more work has to be done. Recommendations are given in the next section.

6.5 Open Challenges and Future Directions

As the contrast during run 2 was sub-optimal and even for human operators it can be difficult to determine the marker locations, it could be beneficial to find ways to increase the contrast between the vessel and the para-magnetic markers. Another way to increase the understanding is to use the temporal domain. By checking out the difference between frames, I was able to determine whether or not a point was a marker or defect, an example is shown in figure 6.8. As seen, by comparing previous pictures, artifacts can be identified between pictures since they are stationary compared to the moving guidewire. This subtraction imaging is also used by [25], by using the initial frame (with no guidewire) to provide a basic form of vessel segmentation.

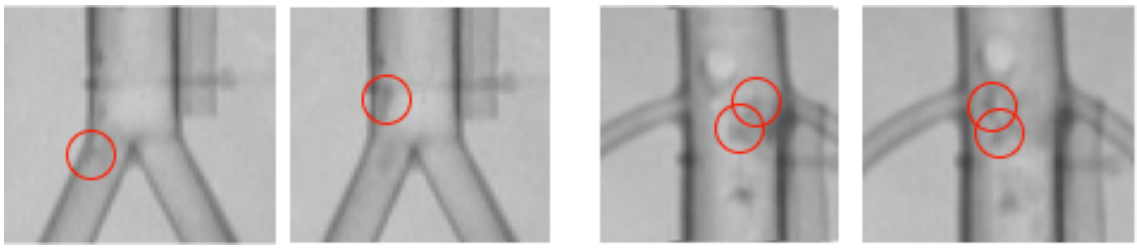


Figure 6.8: By noticing the difference between frames, a difference can be made between image artifacts and signal voids generated by the para-magnetic guidewire markers.

For future research, expansion into the usage of the Access-I interface is greatly encouraged to explore 3D guidewire pose estimation. Applications and methods to explore the possibilities are described in Appendix B.

One thing that can be learned from previous papers is using more involved constraints on the shape of the catheter. For example, by implementing the minimization of a cost function on the full wire interpolation, we can set a few conditions:

- Void-distance: The distance between each signal void is pre-determined by the guidewire manufacturer. Knowing where the tip is, allows for 'guessing' where the other markers will be, based solely on the distance between the points.
- Wire-bend angle: The mechanical properties of the guidewire can be analyzed to determine the maximal bend-angle of two successive points.
- Inter-frame displacement: By comparing the average distance moved of two points in between frames, a penalty (or bonus) can be assigned based on the difference between expected and actual movement is. This ties in with robotic actuation of the guidewire to get an even closer estimate of the displacement.

Using these parameters a specific certainty value can be bound to each point of the guidewire, giving the possibility of predicting whether or not a detected point will be false.

Now that the initial measurements have been made, it will be possible in the future to create a more modern framework with a learning approach, as proposed in FW-net [26], this approach combines spatial with temporal information, and it can decrease the required amount of data by means of data-augmentation.

7 Conclusion

The project was set out to create a **navigation framework using magnetic resonance imaging for robot-assisted endovascular intervention**. A project plan was made, the software written, imaging done, and the data analyzed. Conceptually all the required steps are there.

The framework consists of a DICOM to png conversion tool, a DICOM header reader, an image labeler to create ground truth data, a visualizer tool which can display the labeled points over time and most importantly, the segmentation/detection tool. Finally, a comparison tool is provided to compare the generated data to the ground truth data and recreate the results displayed in the thesis.

Looking at the numbers, of the four scans done, scan 1 & 2 were not qualified for analysis due to their statistical insignificance (Shapiro Wilkes > 0.1). Scan 3 & 4 were significant, and also comparable, due to t-test values < 0.1. When comparing the two, a sub-pixel standard deviation is observed, 0.6mm and 0.78mm. The bias however is relatively large, 3.95mm and 3.755mm, which could possibly be contributed to the less distinguishable shapes of the paramagnetic markers during the second run. The detection rate of the guidewire is 74% and 61.84% for both runs respectively.

This thus concludes the initial step to the passive guidewire tracking for robotic endovascular intervention, to enable haptic feedback under MR imaging.

A Technical software description

In this section, the software will be explained on a more file-to-file basis. To start, everything can be found on GitHub:

```
1000 https://github.com/Jelle-Bijlsma/UIMR
```

An explanation on how to clone the code and do the initial setup is found in C. The repository consists of different folders to provide an overview of the many files. We will go over them initially to provide the users with structure.

- *QT_Gui*: This folder contains everything relating to maintaining the GUI.
- *classes*: Contains files hosting the different classes that were large enough to warrant their own file.
- *data*: Folder structure that should host all the data that is used for analysis, and all the results of that analysis.
- *functions*: Functions that are used throughout the program carefully organized within their own folder

The repository further hosts the separate programs that can be used. These programs include:

- **UTMR_main2.py**: The main file which can be run to perform the segmentation of images, the general image processing and the conversion of Dicom files to .PNG.
- **compare_files.py** Compares the 'measured results' which are obtained by the program, with the 'true results' which are obtained by human selection.
- **image_labeler.py** GUI program which is used to select ground truth points from images.
- **label_visualizer.py** Program which can read and display the results from *image_labeler.py* graphically.
- **read_dicom.py** Small program to acquire important header information with respect to scan parameters from Dicom images.

Now that we have a general understanding of the repository structure before we continue it is important to understand the basics of PyQT. This will be explained in the following section.

A.1 PyQt Introduction

As the name suggests, PyQt is the python 'version' of the Qt framework, which is written for C++. There are many tutorials available and great documentation exists explaining PyQt. The reader versed in C++ can also easily read standard Qt documentation and be able to translate this into Python since the function, class names and methods are exactly the same and the documentation for Qt is of a much greater volume than the Qt documentation. It would be of no use to give a full Qt tutorial in this appendix so the general ideas and concepts that are used will be explained in this section.

A GUI can be created fully in your to-go text editor however a much more convenient method is Qt-Designer. This program provides a visual and simple, point-and-click way to create complex GUI's. In Qt, all the objects you create can be classified as widgets. These widgets are

represented as classes in Python. Making a button would create an instance of the QPushButton class. After creating a GUI in PyQt, this GUI can be loaded in through a Python interpreter as follows:

```
from PyQt5 import QtWidgets
import QTFILE

    class MainWindowz(QtWidgets.QMainWindow, QTFILE.Ui_MainWindow):
    def __init__(self, form):
        super(MainWindowz, self).__init__(parent)
        self.setupUi(MainWindow)

    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        MainWindow = QtWidgets.QMainWindow()
        ui = BuildUp()
        MainWindow.show()
        sys.exit(app.exec_())
```

Where QTFILE is the file that has been created by Qt-Designer. This example itself does not do anything on its own, if the file for example contains a button, nothing will happen on the surface when the button is clicked. All the objects created in Qt-Designer have names and are objects which have many methods and attributes that can be accessed. When a button is clicked, a **signal** is emitted. This signal can be picked up by Qt and allow for a piece of code to be executed. In the $__init__$ function of the $MainWindowz$ class, one could write the following:

```
    self.button1.clicked.connect(self.pressed)
```

Where $button1$ is the object referencing a button on screen. When the button is clicked, $clicked.connect$ will connect the signal emitted from the button to the function $self.pressed$. This function can be defined in the $MainWindowz$ class to perform any arbitrary piece of code. The signals can also carry information regarding the state of the object, as a text input field can for example send a signal whenever a character is inputted. This signal can then carry the inputted text. This construction allows for a complex system of reactions to occur based on different user inputs.

A.2 Program Files Explained

The larger files in the root folder of the repository will be explained in this section. When in doubt about specific statements, please refer to the actual code as well, since it is commented generously and should be understandable on its own. The files *label_visualizer.py & read_dicom.py* are of lesser importance and relatively short that they can be understood from the comments alone.

A.2.1 image_labeler.py

The image labeler is a GUI application but it is relatively simple, so we start explaining here. Once understood we can then move on to the more complicated work in $UTMR\_main.py$. To re-iterate, in this program, the user can select a folder with images, to create a list of ground truth points to be used later.

We first take a look at the (two) GUI files (*.ui*) in the **QT_GUI** folder, *listbox, image_labeler*. This is because the program runs two separate windows. The image labeler is the main one, and listbox just listing the coordinates. First we open $image_labeler.ui$ In the top right of Qt-

Designer the object inspector can be found A.1. It is a useful tool to understand the structure of the program.

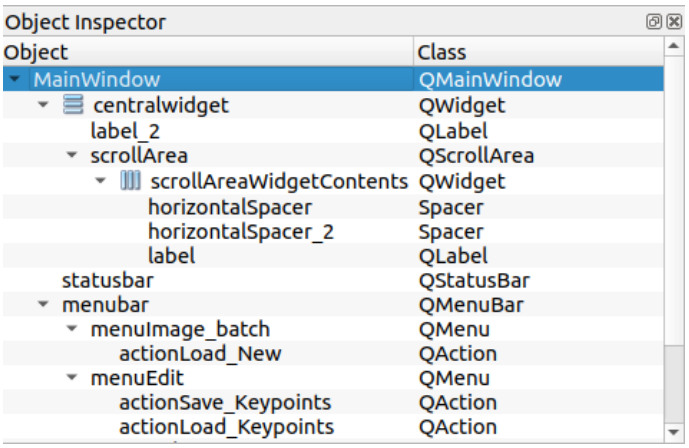


Figure A.1: The object inspector gives many hints to the internal workings of a specific Qt structure.

From figure A.1 it can be observed that the MainWindow consists of two main elements, $centralwidget$ and $menubar$. Menubar refers to the options in the top part of the screen such as **Image batch, Edit, Help**. It is relatively straightforward and skipped now. The central widget contains two elements, a QLabel and a QScrollArea. The instances of the QScrollArea and QLabel class, $scrollArea, label_2$ can be accessed inside the Python script. The scrollarea widget is used for creating a scrollable area, which we will use to zoom in on the loaded pictures to accurately determine the middle of each marker. The Qlabel class is used to put texts and images inside your GUI. The spacers (and as is visible in $listbox.ui$, the layouts) are used for positioning of the elements in case of rescaling of the window.

As stated before, the event handling is done by the main class, in this case $MainWindowz$. We start with initializing an instance of $ScrollClass$, which is responsible for the ability of zooming.

```
1000 self.ScrCls = ScrollClass(image, self.label, self.scrollArea.horizontalScrollBar(),
     self.scrollArea.verticalScrollBar())
1002 self.create_ting(MainWindow, self.ScrCls)
```

Afterwards, the PointWindow is created. This is to initialize $listbox.ui$. Afterwards, we initialize the menubar:

```
1000 self.actionLoad_New.triggered.connect(lambda: self.PtWin.loadem(production=True))
     self.actionSave_Keypoints.triggered.connect(self.PtWin.savepoints)
1002 self.actionLoad_Keypoints.triggered.connect(self.PtWin.loadpoints)
```

As can be seen, the emitted signals of the events are connected to the methods of the main class. Notice in $actionLoad_New$ we utilize a lambda function, this is because we need to pass a reference to a function, and in this case we need the function to already have arguments. $PtWin$ is the PointWindow, and the 'brain' of the process. It will be explained at the end of this section.

```
1000         self.image = cqpx(image)
             self.label.setPixmap(self.image)
1002         self.label.setMouseTracking(True)
```

This codeblock shows $cqpx$, which is a function that will be used a lot throughout the project. It is imported from:

```
1000        from functions.process import change_qpix as cqpx
```

cqpx Converts a Numpy array to a Qpix object, which then can be passed to a QLabel instance, such that it can be displayed. Due to some peculiarities in Qt, we can not 'just' give a Numpy array, it needs to be converted first. Next, we set the label to show the image of which we want to do analysis. Furthermore, we enable mousetracking in the label. This will allow the label to emit a signal every time the mouse moves over the image, and the signal will contain the mouse coordinates, this is used later to select the coordinates of each para-magnetic marker.

```
1000      # scroll override
self.scrollArea.keyPressEvent = lambda event: custom.keyPressEvent(self, event)
1002 self.scrollArea.keyReleaseEvent = lambda event: custom.keyReleaseEvent(self, event)
self.scrollArea.wheelEvent = lambda event: custom.wheelEvent(self, event)
1004
# mouse press override
1006 self.label.mousePressEvent = lambda event: custom.mousePressEvent(self,event)
self.label.mouseMoveEvent = lambda event: custom.mouseMoveEvent(self,event)
```

In this code-block, some events are overwritten. Instead of the usual, we reroute this to specific functions using:

```
1000        import functions.labeler_custom_event as custom
```

Looking at wheelEvent for example, the standard event of moving the mouse wheel would result in the vertical scrollbar moving up or down, however for the custom event we can observe:

```
1000        def wheelEvent(self, event: QtGui.QWheelEvent):
                scr_val = event.angleDelta().y()
1002            skp = self.ScrCls.change_size(scr_val)
                if not skp:
1004                super(QtWidgets.QScrollArea, self.scrollArea).wheelEvent(event)
```

From this code we can observe that we take the $angleDelta().y()$ from the scrollEvent. This is how much the value of the vertical scrollwheel increased from the original position. This value is passed to a method of the ScrollClass, and depending on the return-value of that function, we might run the original event as well. The $change_size$ method checks if the 'Ctrl' button is simultaneously pressed. If this is the case, the image is either zoomed in or zoomed out, depending on whether the scroll wheel was actuated forward or backward, and returns True. If 'Ctrl' was not pressed, it returns False.

```
1000    a_warning = QtWidgets.QMessageBox()
        a_warning.setText("The coordinate transform fails if there are purple boundaries. \n"
1002                      "Please zoom in until no purple is visible")
        a_warning.exec()
```

This is the last part of the $\_\_init\_\_$ script and generates a warning to the user that a certain level of zoom is needed otherwise the program does not work. This is due to a bug in the coordinate transform for which I could not find a solution. Now we will look at the PointWindow class.

PointWindow

This class is the brain of the program, and contains the methods: *delete_entry*, *savepoints*, *loadpoints*,*delete_all_entry* and *update_text*. They work together to facilitate the data display onto the GUI of the selected points.

```
1000        def savepoints(self):
    try:
1002        name = QtWidgets.QFileDialog.getSaveFileName()
    except FileNotFoundError:
1004        return

1006    print(str(name[0]))
    f = open(name[0], 'wb')
1008    pickle.dump((self.clicklist, self.image_number), f)
    f.close()
```

Looking at the workings of *savepoints* we start with a try-except statement that uses *QFileDialog*. This function opens a fil menu to select how you would like to save the currently generated key-points. It uses Pickle to store a binary containing the amount of selected points per image. The try-except statement is used because it throws an error if the user closes the window prematurely.

```
1000 def click(self, event: QtGui.QMouseEvent):
if event.button() == 1:
1002    brightness = self.scrollclass.nparray[int(self.rescaled_y), int(self.rescaled_x)]
    self.clicklist[self.image_number].append((self.rescaled_x, self.rescaled_y,
    brightness))
1004    self.clicklist[self.image_number] = sorted(self.clicklist[self.image_number], key=
    lambda x: x[1])
self.update_text()
```

The click method is bound to the custom *MousePressEvent*. All the other custom events are written down in *labeler_custom_events*. For every click on the ScrollArea, the method is executed. Notice the IF statement, checking whether it is a left mouse button press and not another mouse key. The function relies heavily on re-scaling, as observed in the *self.rescaled_x*,*rescaled_y*. This re-scaling is from:

```
1000 def mouseMoveEvent(self, event: QtGui.QMouseEvent):
    x, y = self.ScrCls.labelsize  # ORIGINAL
1002    X, Y = self.ScrCls.w, self.ScrCls.h  # current
    xscale = x / X
1004    yscale = y / Y
    x_rescale = event.x() * xscale
1006    y_rescale = event.y() * yscale
    self.PtWin.rescaled_x = np.floor(x_rescale)
1008    self.PtWin.rescaled_y = np.floor(y_rescale)
    # QtWidgets.QToolTip.showText(event.pos(),str(x_rescale)+ str(y_rescale))
1010    self.PtWin.update_coords()
```

It is necessary due to the zooming involved. When zooming in, the resolution increases, yet the display area stays the same. Thus, without re-scaling, one would try to access elements in the original image that do not exist currently. The rest of the methods and rewritten events should be straightforward from here, and the last remaining part is a look at the ScrollClass.

ScrollClass

The general class ScrollClass, or the instance of it used during runtime, ScrCls, manages the window re-scaling. The image has four methods, *change_image*, responsible for loading in a new image, *change_size*, responsible for the zooming action, and the get and set *scroll*, which take and manipulate the positions of the scroll bars. These are used in the *change_size* method, since the original implementation of re-scaling the image puts the scrollbars at (0,0) again, which would deliver an unpleasant zooming experience. By using get and set *scroll* the vertical and horizontal scrollbars are kept in the middle of the image. Let us only have a look at the *change_size* method:

```
1000    def change_size(self, ch_s, override=False):
            if self.ctrl_pressed or (override is True):
1002            if ch_s == 0:
                    pass
1004            elif ch_s == 42069:  # arbitrarily high value to recognize imchange
                    self.sizevar += 0
1006            elif ch_s > 0:
                    self.sizevar += 0.5
1008            elif ch_s < 0:
                    self.sizevar -= 0.5
1010                if self.sizevar == 0:
                        self.sizevar = 0.5
```

The initial part starts with an IF statement that sets the *self.sizevar*, which will do the re-scaling. The first two statements do not produce a change in size but are still used later to do other initialization. Positive values of *sizevar* indicate zooming in, negative zooming out.

```
1000            w = int(self.sizevar * self.qpix_og.width())
                h = int(self.sizevar * self.qpix_og.height())
1002            wold = int(self.qpix.width())
                hold = int(self.qpix.height())
1004            self.qpix = self.qpix_og.scaled(w, h)
                self.w = int(self.qpix.width())
1006            self.h = int(self.qpix.height())
                diff = (self.w - wold, self.h - hold)
1008            self.get_scroll()
                self.image.setPixmap(self.qpix)
1010            if not override:
                    self.set_scroll(diff)
```

The new width (w) and height (h) are saved, together with the current values (wold, hold). Then, using a method of the Qpix class, 'scale' we can re-scale the image, and unless manual scrollbar change is overridden, the scrollbar is set to the middle.

There are other methods and functions in the code that are not discussed, but their workings and meaning should be trivial once the previous sections are understood.

A.2.2 compare_files.py

This file is used to compare the results from *UTMR_main.py* to the results obtained by the image labeler. It does not rely on custom external files and has no PyQt dependency. It creates the plots and tables shown in the Results section of the report.

This program solves two important problems. Number one: matching the values from the image labeler (the ground truth value, or Truth Value, TV for short) to the measured values MV. This is a non-trivial problem since:

$$TV > MV | TV < MV | TV == MV. \tag{A.1}$$

There is also a problem of false MV, i.e. a measured value that has no corresponding truth value.

Second: Providing analysis on the data. As soon as the TV is matched to the MV, the accuracy of the results need to be measured and the significance of the results need to be quantified. Thus, to explain both problems, this subsection is broken up into two sub-subsections, point-matching and results analysis:

Point-Matching

The point matching problem is solved by the DistComp. For each frame, a new instance of the DistComp is created:

```
1000    for image, index in zip(images, range(len(images))):
            dc = DistComp(measured_noff[index], truecoordlist[index])
1002        dc.point_battle()
```

Listing A.1: lines 193:197

In which all the TV & MV are given, and after which the DistComp starts the "point_battle". Before the Point Battle is explained, a word about the point class. the "Point" class is a simple class, of which an instance is created for each TV. The point can bind to a MV using the assign method, and has an attribute called "assigned" to check whether or not it is already bound to an MV.

Given a set of MV and TV, the distcomp calculates the distance between each TV and each MV. If the smallest distance is smaller than a specified cutoff value, the pair is assigned. This removes the specific MV and TV from the list until there are none left that either meet the requirement, or if either the MV or TV list is empty. See figure A.2 for an illustration.

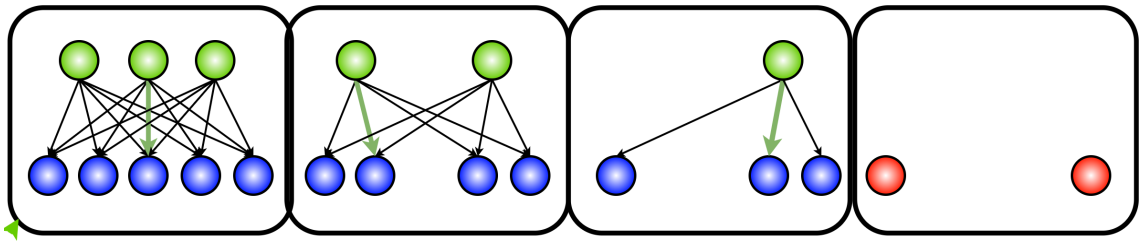


Figure A.2: DistComp pointbattle diagram.

```
1000    pickfirst = lambda x : x[0]
            self.longlist = sorted(self.longlist, key=pickfirst)
1002        while True:
                try:
1004                coords = self.longlist[0]
                except IndexError:
1006                break
                range = coords[0]
1008            if range > self.maxdist:
                    break
1010            mcoord = coords[1]
                pointnum = coords[2]
1012            self.pointlist[pointnum].assign(mcoord)
                clonglist = copy.deepcopy(self.longlist)
1014            for element in clonglist:
                    if (mcoord == element[1]) or (pointnum == element[2]):
1016                    self.longlist.remove(element)
```

Listing A.2: 79:95

The "longlist" is created by the preceding method, "distcalc" which simply calculates the distance between each TV and MV. The rest of the code is straightforward, the only point to notice is that the "clonglist" requires a deepcopy, else due to the way Python references work, removing elements from the original list will cause problems due to items being skipped in the final FOR loop.

Finally, the distcomp will be asked to output each element to a specific group: "ass"igned, "no"t "ass"igned, or faulty. Calling the methods "dc.finalanswer" will return these 3 lists. This process is then repeated over each frame and the results are stored in new lists.

```
1000
        for element in ass:  # assigned
1002        cv2.circle(image, element.coord, 3, green)
        for element in noass:  # not assigned
1004        cv2.circle(image, element.coord, 3, yellow)
        for element in faulty:  # incorrect guess
1006        cv2.circle(image, element, 3, red)

1008    if video is True:
            cv2.imshow('visualizer', image)
1010        cv2.waitKey(0)
```

Finally, the results can be visualized if needed. This was mainly implemented as a debugging tool, to see if the algorithm works as expected, and the variable "video" is False by default.

Results Analysis

The results and analysis are straightforward given the previous conditions and can be easily read from the commented code. The only thing which might be surprising is:

```
1000    for cp, fp, tp in zip(pvar1,pvar3,pvar4):
        perc1.append(cp/tp)  # percentage of points labeled
1002    try:
            perc2.append(fp/(cp+fp))  # percentage of points incorrectly labeled
1004    except ZeroDivisionError:
            print("you tried to divide by zero. This happens when there are no detected"
1006              "points and no fault points. If you expect this, no worries.")
```

Where the TRY EXCEPT statement could be confusing initially. The zero-division error occurs when there are 0 points correctly assigned and also zero faulty assigned points, since:

```
1000    pvar1 = []  # total correctly assigned points
        pvar3 = []  # number of faulty points
```

All the items under:

```
1000    if __name__ == "__main__":
```

are hard-coded, and specific to the scans in this thesis.

A.2.3 UTMR_main.py

Being the main file, it is the most complex file of all three, but the experience gained in the previous two sections allows for a much faster understanding. The program is handled by a

"StackedWidget" which allows for multiple windows to be cycled through. Currently three windows are implemented. The main menu (see figure A.3), which gives access to the other two menus, the Dicom editor and the video editor. The menu also hosts the 'Live Feed' and the 'Surgery' option, two menu's which sadly have not come to fruition due to the unavailability of the Access-I interface. The menu-bar also has a 'Menu' and 'Options' section, which can be used to return or move to the different menus, and to load and save parameters respectively.

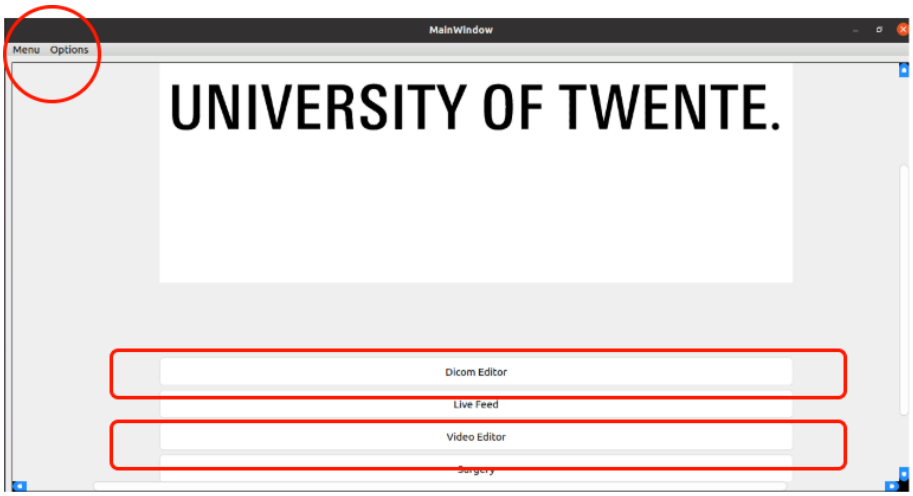


Figure A.3: The main menu and the clickable buttons.

The Dicom editor is hosted, as we expect from Qt, like the other programs, inside the main class; called $BuildUp$. In the $\_\_init\_\_$ we find many variable initializations and a great lot of signal handling, all the different buttons in the different programs linking to specific pieces of code as can be observed in figure A.4.

Figure A.4: Most of the $\_\_init\_\_$ file is dedicated to standard initialisation of variables and the binding of signals.

A relatively larger section yet, is dedicated to $SliderClass$ instances being created. This will be handled later. Let us currently look at the Dicom Editor. The First methods after the init-file are all related to the image manipulator. Starting from line 755, the functions relating to the Dicom Manager are presented. Looking at the Dicom Page in figure A.5, three buttons can be observed. 'Convert', 'Clear Output' and 'Browse'. The browse button works similar to the previously mentioned filebrowser and the Clear Output button clears the logs. Both not very interesting in contrast to the Convert button, which binds to the $convert$ function at line 771. This function contains some user warnings to prevent loss of data, as shown in figure A.5. Besides creating the PNG's using PyDicom, and the video with OpenCV, multithreading is implemented to prevent the GUI from freezing.

```
1000            dcm2pngworker = classes.class_extra.Worker1()
        filelist = os.listdir(dcmpath)
1002    filelist.sort()

1004    path = dcmpath + "/"
. . .
1006      self.threadpool.start(lambda a="./data/png/" + project_name + "/", b=fps, c=
    videopath:
                                  dcm2pngworker.png2avi(a, b, c))
```

Workers are initialised and processes are added to the thread-pool. Since the PNG conversion is quite taxing, especially with an increased amount of files, multi-threading allows for the rest of the GUI to be responsive. This is built on the framework provided by Qt. The worker instance communicates with the GUI thread by means of QT signals emitted when specific events occur, as described in the *class_extra.py* file.

```
1000            self.signals.starting.emit()
        self.a = 0
1002    for element in filelist:
            self.a = self.a + 1
1004        string = path + element
            dicom = dcmread(string)
1006        array = dicom.pixel_array
            plt.imshow(array, cmap="gray")
1008        savestring = "./data/png/" + project_name + "/" + element + ".png"
            plt.savefig(savestring)
1010        self.signals.progress.emit(self.a)
```

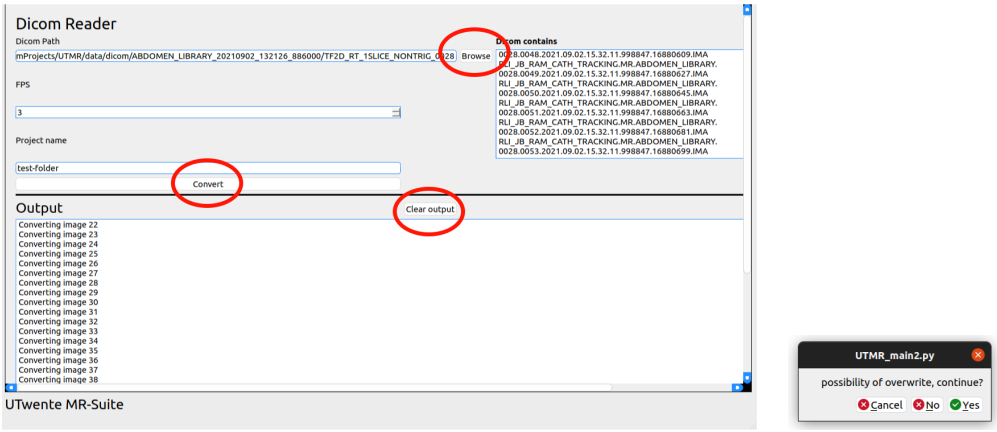


Figure A.5: The three buttons that govern the control of the Dicom Reader. Many warnings and pop-ups notify the user of potential problems during setup as shown on the left.

Video Editor

The video editor spans the most substantial part of the code and performs the most important task, the segmentation of the vessel and guidewire. This is accomplished by sub-sequentially applying different image processing and segmentation steps. The most important part of the code which is responsible for applying all the operations and displaying the results to the screen is *update_all_things* and depends heavily on the *MovieClass*. First, the *SliderClass* and its usage will be discussed, afterward the MovieClass. During the MovieClass and *update_all*, many branches are made to the functions used by this Class and Method.

In the framework, many GUI elements exist, and much of the parameter tuning relies on quickly finding the right value for the parameters by doing quick experimentation without re-running the program. A nice example of this is found in the Blobfinding method. As seen from figure A.6, there is a multitude of sliders, and changing these sliders instantly changes the values used by the specific algorithms.

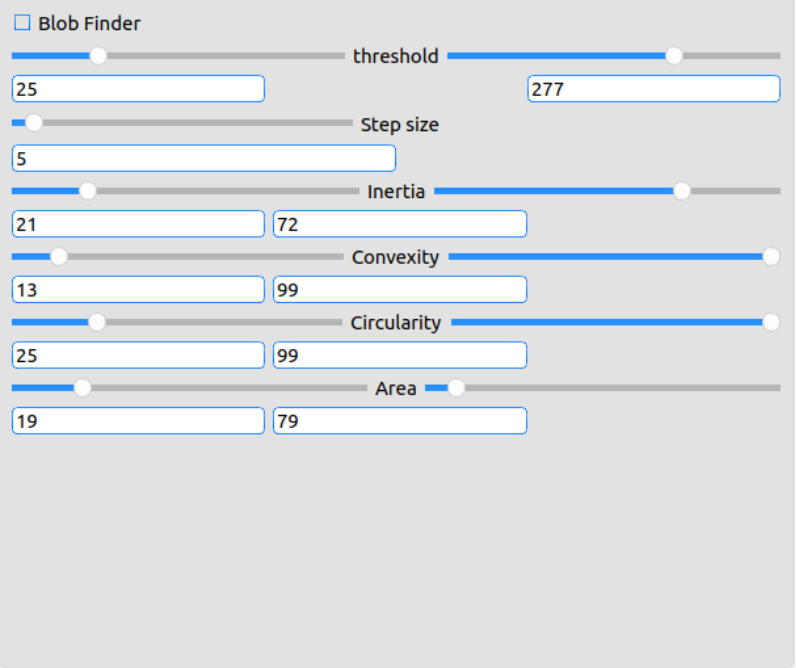


Figure A.6: The blobfinder menu, showcasing a few of the many sliders present.

Besides collecting and accessing the variables in an orderly fashion, SliderClass allows for setting and getting all of the 49 sliders used in the program with ease. Parameters can be loaded in for different settings, as well as new parameter combinations saved, using the methods of the sliderclass. The SliderClass instances are created during the initialization and a typical SliderClass creation looks like this:

```
1000 sl_gfilter = [self.slider_g_size, self.slider_g_sigX, self.slider_g_sigY]
     le_gfilter = [self.lineEdit_g_size, self.lineEdit_g_sigx, self.lineEdit_g_sigY]
1002 self.SC_g_filter = SliderClass(
     slides=sl_gfilter, line_edits=le_gfilter, function=self.pre_value_changed, keyword='
         g_filter',
1004     radiotuple=radiotuple, checklist=[self.check_filter_g])
```

First, the slider objects which belong to the specific group are put in a list, in this case the (sl)iders responsible for the parameter selection of the (G)aussian Filter (sl_gfilter). Next, QLineEdit objects are included to display the value of the slider, as well as to enable numerically setting the sliders to a specific value. The values of the sliders are stored by a dictionary in the MovieClass, thus a function that writes to the MovieClass instance is also passed along, together with the corresponding keyword to retrieve the current slider values. If the effect (Gaussian Filter) can be applied to the masked image too, the parameter $radiotuple$ is set to the corresponding attribute, else it is set to none. If the effect also contains an instance (or more) of the QCheckBox widget, these instances are also passed along. A special extended SliderClass exists too, $ExtSliderClass$, this sliderclass facilitates the usage of 'minimum' and 'maximum' slider pairs, which automatically increase or decrease the corresponding slider if, for example, the minimum value wants to overshoot the maximum value. The internal workings are quite

dense, and will only make sense after careful consideration. They both heavily commented and should not be taken lightly.

MovieClass is responsible for handling the collection of frames and properly edit them according to the GUI. The parameters are stored in a dictionary and can be accessed using the keys given during the creation of the SliderClass. The MovieClass keeps the unprocessed picture saved, and every time a parameter is changed, the update method is called, and by using a "man-link" procedure, each operation is linked to the original image (see figure A.7).

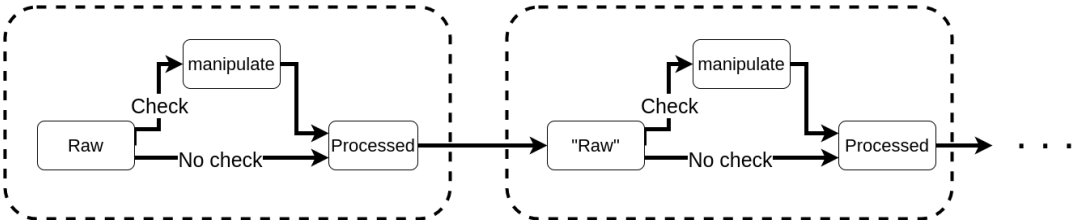


Figure A.7: The raw image, together with the parameters regarding the specific image manipulation are checked. If they pass the test, the raw image is manipulated and becomes the processed image, which is the input to the "raw" image of the next processing step. If the checking fails, either due to the feature being turned off, or the parameters being wrong, the Raw image is taken as the input to the next image manipulation step.

The *update_all_things* function is responsible for displaying the correct information to the user. All the images and values are written to the GUI here. The function starts with a, at first glance, strange command.

```
1000    def update_all_things(self):
            # called whenever the main screen should be updated
1002        self.lineEdit_uaT.start()
```

The QLineEdit object does not have a *.start*() method. However, opening *class_addition.py*, it is observed that QLineEdit has been extended with time-keeping abilities. This has been implemented to check the speed of the process, and has been used a lot in the later stages of the creation process to optimize specific functions which were pointed out to execute relatively slow. The different clocks are shown in figure A.8. By calling the start and stop method of the specific QLineEdit, the time can be displayed in either milliseconds or nanoseconds, depending on the accuracy needed.

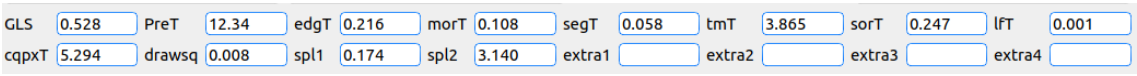


Figure A.8: The time each step takes, calculated using an average of 10 operations, thus initially, the time might fluctuate.

After receiving all the to be displayed information from the movieclass after calling:

```
1000    self.lineEdit_sumT.start()
        output = self.CurMov.update(morph_vars, segment_state, timer_list, self.templates,
1002                                self.checkBox_heq.isChecked())
        self.lineEdit_sumT.stop()
```

And displaying in the various QLabels, the rest of the code should be relatively straight forward, and by reading the comments, many, if not all of the design choices should be clear. One final step to note is how to get readable data from one segmentation session. This is done by first editing the *file* variable in line 550:

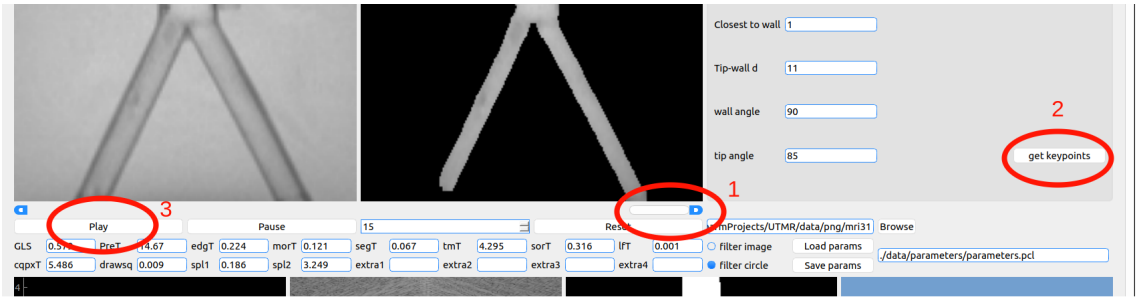


Figure A.9: By pressing the buttons in ascending order, the data-generation is started. As soon as the movie has looped one time, the data-generation is done.

```
1000        if self.CurMov.maxframes == self.CurMov.frame_number:
                file = open('./data/measured_mri1.pcl','wb')
1002            pickle.dump((self.kp,self.internal_dict['crop']),file)
                file.close()
1004            print("written file")
                self.CurMov.getkp = False
1006            self.kp = []
```

Then setting the progress bar to the final frame, hitting the "get keypoints" button in the results section and pressing Play, as explained in figure A.9.

This guide, combined with the comments written throughout the files in the repository, should generate a proper understanding of the code used during this thesis. The Object Inspector is, as mentioned before, an invaluable tool in understanding the workings of the GUI programs.

B COPS

To fully utilize the benefits of MR, a method should be developed to step up from the original imaging to a new form. This proposed viewing method will be called "Corrective Orthogonal Predictive Scanning" (COPS). What constitutes COPS and how to achieve it is explained in this section. In section 2.2, it is explained how every MR-image is a 2D slice. This slice can be in any orientation. Thus, when the slice is taken orthogonal to the classic direction, both x and y catheter-wall distance is immediately visible, as seen in figure B.1. This approach is the most direct and intuitive, in terms of endovascular imaging since it requires no reconstruction techniques using orthogonal scan planes.

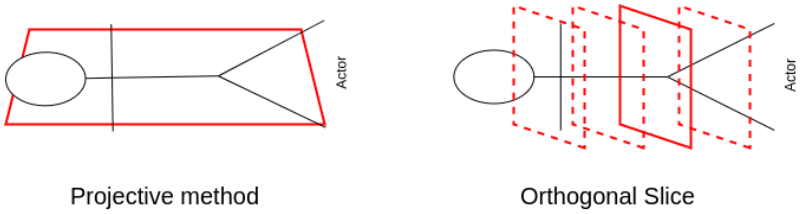


Figure B.1: The red rectangles show the scan plane. In the classic method the slice thickness is large enough to include all the necessary vasculature during surgery. Orthogonal slice works with thinner slices that have to be re-positioned during surgery (dashed lines).

In COPS, the main problem is the variable scan plane. In the classic method, the scan plane is fixed. To make COPS work, the framework should interact with the scanner as well as locate in which plane the catheter tip is. Theoretically, there are four degrees of freedom involved. Movement perpendicular to the plane (**p**), and rotation around x,y & z (ω, ϕ, θ). See figure B.2. The reason why XYZ is not used is due to the fact that the scan plane can be considered infinite and thus, only the orientation of the plane and the perpendicular distance moved is of importance. For moving through the aorta, only **d** is variable when XYZ is set. When looking at the renal artery, a yaw angle needs to be incorporated. The next subsection will focus only on the aortic case.

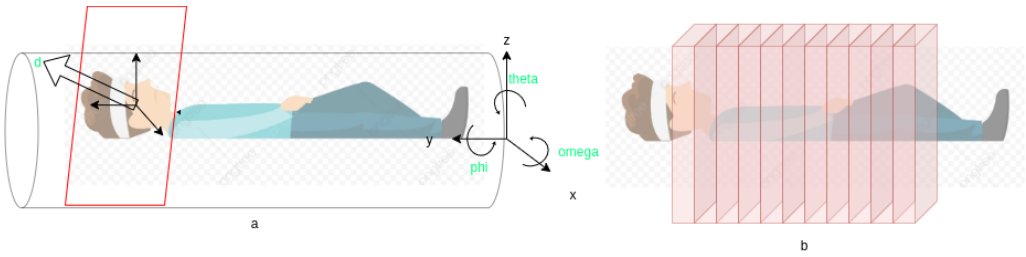


Figure B.2: (a) showing the four degrees of freedom in COPS, and how they relate to the patient frame. (b) shows the discretized scan planes. (image source 'patient clipart PNG' designed by 杨浩and taken from https://pngtree.com/)

Due to the nature of MR, scan planes have a finite thickness, and thus the patient can be 'divided up' into N scan planes. This leads to

$$N = \frac{L}{s}, \tag{B.1}$$

where L is the total length of the MR-tube, and s is the slice thickness. The next step is to predict in which frame the catheter tip will be.

B.0.1 Prediction

The initial position of the catheter tip is determined by the radiologist. The radiologist gives the initial position in terms of the scan parameters x. These scan parameters are what can be read in and out to the MR scanner, while the discretized scan planes (y) are an abstraction to ease calculation. With this knowledge, the following relations can be constructed:

$$\begin{aligned} \dot{x} = x(t) + \mathbf{B}u(t) + \mathbf{K(t)}x(t), \\ y(t) = \mathbf{C}x(t) \end{aligned} \tag{B.2}$$

Where $x(t)$ is a $px1$ matrix where p are the number of scan parameters, and **B** is a $px1$ matrix which maps the control input u to a form suitable for scan parameter manipulation. **K(t)** is the *corrective matrix* and keeps the error of the predictor bounded. The working of the corrective matrix will be explained after, but first, it will be explained why it is needed.

Prediction Errors

The control input mentioned in equation B.2 is a scalar quantity and represents how much the wire is moved forwards or backwards. This quantity is determined according to:

$$u_1 \alpha = u, \tag{B.3}$$

Where u_1 is the linear translation as done by the surgeon on the master manipulator, and α is the slave coefficient and $\alpha \propto \mu\gamma$ where μ is the motor constant and γ the gear ratio of the motor to the extruder. For perfect prediction, it is assumed there is a no-slip scenario. In a 1D case, there would be no need of a correction, however, 2D already shows that there can be a discrepancy between the traveled distance l' and extruder distance l (figure B.3. Assuming l' consists of two line segments joined together, we can approximate:

$$l' = \sqrt{4r^2 + l^2} \tag{B.4}$$

Because of this discrepancy in distance, the selected frame y could either be in front of the catheter tip, and not show anything at all, or lag behind. The initial case is easily discovered.

When no tip is discovered, move back 1 frame. Repeat until a circle is found. The case in which the tracking is lagging behind can be discovered by periodic polling, figure B.4.

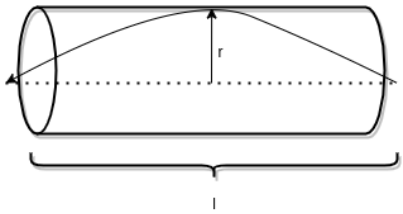


Figure B.3: Besides the travelled catheter path being shorter than the inserted catheter length, rebounding can happen, when the guidewire relaxes and the travelled length becomes equal to the inserted length

Periodic polling works by moving the scan plane, a distance k ahead, where k is the distance between the MR-visible beads which make up the catheter. If the periodic pol plane detects no bead, nothing happens. In case a bead is detected, the periodic pol plane becomes the current scan plane.

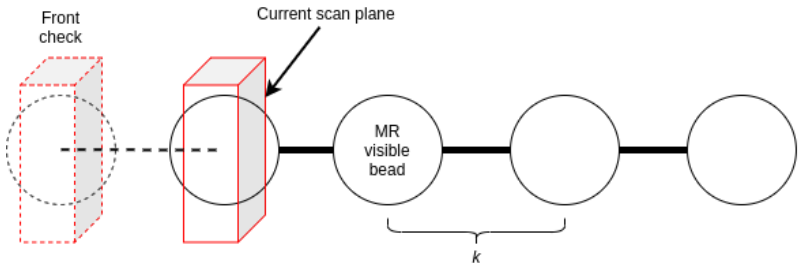


Figure B.4: Periodic Polling to determine whether or not the current plane is actually the tip of the catheter. Distance k is the distance between each bead.

Thus, when dealing with the corrective action, two distinct situations occur. **Lagging** of the scan plane, when the catheter is in front of the scan plane. **Rebounding** of the scan plane, which happens when the catheter is behind the scan plan. This is shown again in figure B.5.

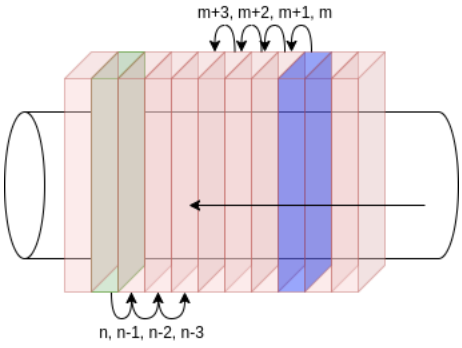


Figure B.5: During lagging (green frame), the scan frame will be lowered each frame subsequently until the tip frame is reached. Similarly, during rebounding, the scan frame will be increased until the tip frame is in position.

Besides discrepancy due to bulging and/or other forms of 2D travel, slipping in the extruder and imperfections in the translational model also cause discrepancies. In conclusion, by using an initial position given by the radiologist, prediction based on surgeon input through the master manipulator and periodic corrective checking, a sustainable proper 2D vision of the catheter tip can be achieved.

B.0.2 General Position Checking

One issue with the COPS algorithm is that the general spatial awareness is lost when COPS is implemented solely. Therefore it is proposed to use COPS in conjunction with the classic imaging. Different modes of operation are possible. The most simple method is to simply do one classic scan before the surgery starts. This scan can be overlaid in a corner of the GUI and based on the current slice that is being scanned, a marker can be displayed where in the vasculature the tip is located. Figure B.6 shows a schematic approach to the GUI. After this initial scan, the position can thus be inferred from the scan and no radical change in gradient field is needed.

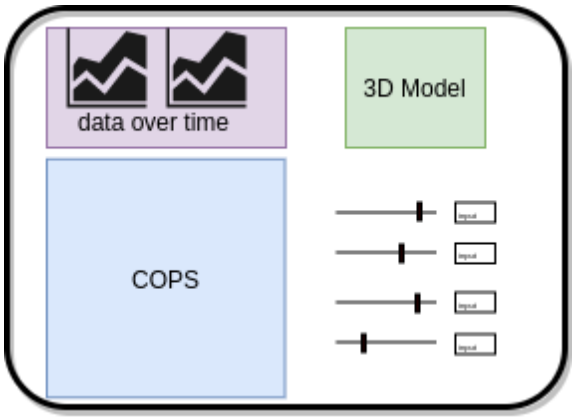


Figure B.6: A view of how the GUI could look like. One window displaying COPS, another window to show the position of the catheter in 3D. Left top corner shows data varying over time which would be of importance to the surgeon, and to the right are sliders to change parameters.

Depending on the speed at which the gradient can be switched, intermittent gradient switching (IGS) could be or not be possible. Due to eddy currents in the gradient coils, the speed at which the gradient field can be altered is limited. Especially for large and abrupt field fluctuations, switching speed has to be taken into account. Either automatically, (every x seconds) or by demand of the surgeon (if the field can not be switched fast enough for automatic sampling), the scan plane will be oriented from COPS to classic. This has as added benefit that now too the entire catheter pose can be tracked and not just the tip.

Finally, if there are enough resources available, instead of doing a preoperative classic scan, a full 3D scan can be done before surgery. This could increase the surgeon's spatial awareness during surgery. Besides bends to left and right, elevations and depressions can be anticipated. It is probably infeasible to get to this step (or IGS), however this does present a goal to strive towards or a stepping stone for future projects.

Alternatives

Instead of automatic tracking, the surgeon could also manually manipulate the scan plane with a device such as a joystick. As explained in section B, there are 4 degrees of freedom. Using a simple joystick with thumb-drive, the surgeon can reach every pose, while keeping one hand free to work on the master manipulator.

Adding to the complexity of the situation, yet giving the surgeon more control over the entire procedure and a richer and more personalized way of conducting the surgery.

B.0.3 Circular Catheter Tracking

Due to the new geometry, the image processing part differs slightly from the 2D case. Instead of relying on image segmentation by means of morphological relations and edge finding algorithms, due to the relative constant nature of vessel geometry over the succession of images

in COPS, active contours would be a good frame to work with. By constant nature of vessel geometry it is implied that the vessel wall does not change abruptly in time. In each frame, the vessel wall will be similar to the previous one, and have a well defined (although noisy), circular shape. With an initial estimation by the radiologist, future shapes can be estimated based on the location and shape of the current contour. For tracking the catheter, using a circle finder within the section which was determined previously to be inside the vessel, the catheter itself can be found.

This subsection is rather brief since no interventional cross sectional images have been made for endovascular procedures in MR. This limits the scope of analysis. However, active contours, or snakes could have potential to solve the image segmentation based on the previously mentioned arguments.

C Running UTMR from Scratch

To run the program successfully, the following steps should be taken. This guides assumes Ubuntu operating system. Open a terminal and type the following:

1. $ mkdir UTMR
2. $ git clone https://github.com/Jelle-Bijlsma/UTMR UTMR
3. $ cd UTMR
4. $ mkdir venv
5. $ python3 -m venv ./venv
6. $ source venv/bin/activate
7. $ pip3 install -r requirements.txt
8. $ python3 UTMR_main2.py

Step 1 creates an empty directory for the files to be put in. In step 2 we acquire the project. in step 3 we enter the folder. Step 4 creates a folder for the virtual environment (venv). Step 5 creates the venv. Step 6 activates the venv. Step 7 installs the requirements. Step 8 runs the program. The following steps lead you through the initialization. See also figure

- Select the folder you want to inspect.
- Press 'load parameters' (1)
- Press 'apply' (2).
- Press the 'TM' (3) tab and press 'load' (4)
- Press the 'results' (5) tab to see the segmentation.

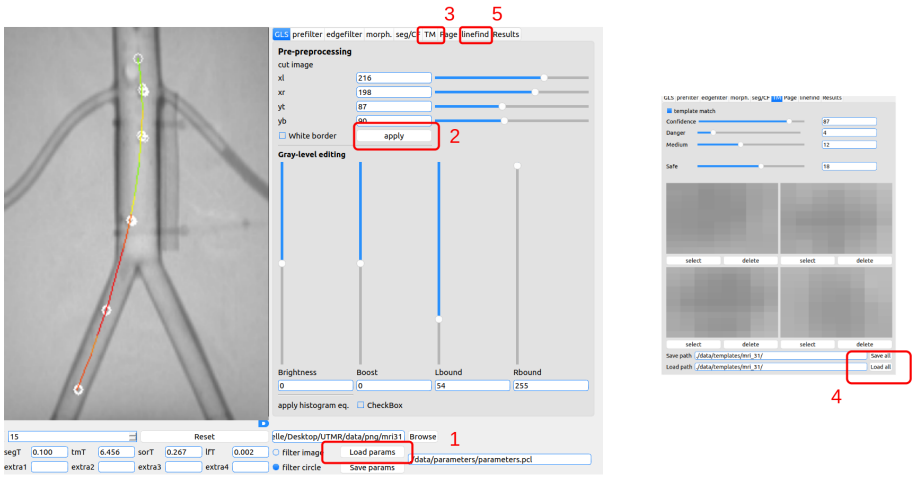


Figure C.1: Steps labeled to obtain a full inspection routine.

Bibliography

[1] E. J. Moore. Robotic surgery. *Encyclopedia Britannica*, 2015. https://www.britannica.com/science/robotic-surgery Accessed: 2021-10-04.

[2] Y.S. Kwoh, J. Hou, E.A. Jonckheere, and S. Hayati. A robot with improved absolute positioning accuracy for ct guided stereotactic brain surgery. *IEEE Transactions on Biomedical Engineering*, 35(2):153–160, 1988.

[3] OECD and European Union. *Health at a Glance: Europe 2020*. 2020.

[4] G. Dagnino, J. Liu, M. E. M. K. Abdelaziz, W. Chi, C. Riga, and G.-Z. Yang. Haptic Feedback and Dynamic Active Constraints for Robot-Assisted Endovascular Catheterization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1770–1775, Madrid, October 2018. IEEE.

[5] Timo Heidt, Simon Reiss, Axel J. Krafft, Ali Caglar Özen, Thomas Lottner, Christoph Hehrlein, Roland Galmbacher, Gian Kayser, Ingo Hilgendorf, Peter Stachon, Dennis Wolf, Andreas Zirlik, Klaus Düring, Manfred Zehender, Stephan Meckel, Dominik von Elverfeldt, Christoph Bode, Michael Bock, and Constantin von zur Mühlen. Real-time magnetic resonance imaging – guided coronary intervention in a porcine model. *Scientific Reports*, 9(1):8663, December 2019. Number: 1.

[6] Dennis Kundrat, Giulio Dagnino, Trevor M. Y. Kwok, Mohamed E. M. K. Abdelaziz, Wenqiang Chi, Anh Nguyen, C. V. Riga, and Guang-Zhong Yang. An MR-Safe Endovascular Robotic Platform: Design, Control, and Ex-Vivo Evaluation. *IEEE Transactions on Biomedical Engineering*, pages 1–1, 2021.

[7] Hedyeh Rafii-Tari, Christopher J Payne, and Guang-Zhong Yang. Current and Emerging Robot-Assisted Endovascular Catheterization Technologies: A Review. page 19, 2013.

[8] Yusof Ganji, Farrokh Janabi-Sharifi, and Asim N. Cheema. Robot-assisted catheter manipulation for intracardiac navigation. *International Journal of Computer Assisted Radiology and Surgery*, 4(4):307–315, 2009.

[9] C.O. Schirra, S. Weiss, S. Krueger, S.F. Pedersen, R. Razavi, T. Schaeffter, and S. Kozerke. Toward true 3D visualization of active catheters using compressed sensing: Active Catheter Visualization Using CS. *Magnetic Resonance in Medicine*, 62(2):341–347, August 2009. Number: 2.

[10] K.S. Rhode, M. Sermesant, D. Brogan, S. Hegde, J. Hipwell, P. Lambiase, E. Rosenthal, C. Bucknall, S.A. Qureshi, J.S. Gill, R. Razavi, and D.L.G. Hill. A system for real-time XMR guided cardiovascular intervention. *IEEE Transactions on Medical Imaging*, 24(11):1428–1440, November 2005. Number: 11.

[11] Carlo Briguori, Davide Tavano, and Antonio Colombo. Contrast agent-associated nephrotoxicity. *Progress in Cardiovascular Diseases*, 45(6):493–503, 2003. Issues on Smoking and Cardiovascular Disease, Part II.

[12] Christopher J. Payne, Hedyeh Rafii-Tari, and Guang-Zhong Yang. A force feedback system for endovascular catheterisation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1298–1304, Vilamoura-Algarve, Portugal, October 2012. IEEE.

[13] Alessandro Vandini, Stamatia Giannarou, Su-Lin Lee, and Guang-Zhong Yang. 3D Robotic Catheter Shape Reconstruction and Localisation Using Appearance Priors and Adaptive C-Arm Positioning. In *Augmented Reality Environments for Medical Imaging and Computer-Assisted Interventions*, volume 8090, pages 172–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[14] Jerry L. Prince and Jonathan M. Links. *Medical Imaging Signals and Systems*. Pearson Education, 2015.

[15] R. van der Weide, C.J.G. Bakker, and M.A. Viergever. Localization of intravascular devices with paramagnetic markers in mr images. *IEEE Transactions on Medical Imaging*, 20(10):1061–1071, 2001.

[16] Allison M. Okamura. Haptic feedback in robot-assisted minimally invasive surgery. *Current Opinion in Urology*, 19(1):102–107, 2009. Number: 1.

[17] Miguel Benavente Molinero, Giulio Dagnino, Jindong Liu, and Wenqiang Chi. Haptic Guidance for Robot-Assisted Endovascular Procedures: Implementation and Evaluation on Surgical Simulator. page 6, 2019.

[18] Vincent Groenhuis and Stefano Stramigioli. Rapid prototyping high-performance mr safe pneumatic stepper motors. *IEEE/ASME Transactions on Mechatronics*, PP:1–1, 05 2018.

[19] Wenqiang Chi, Jindong Liu, Mohamed E. M. K. Abdelaziz, Giulio Dagnino, Celia Riga, Colin Bicknell, and Guang-Zhong Yang. Trajectory Optimization of Robot-Assisted Endovascular Catheterization with Reinforcement Learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3875–3881, Madrid, October 2018. IEEE.

[20] Mohamed E. M. K. Abdelaziz, Stefano Stramigioli, Guang-Zhong Yang, Dennis Kundrat, Marco Pupillo, Giulio Dagnino, Trevor M. Y. Kwok, Wenqiang Chi, Vincent Groenhuis, Francoise J. Siepel, and Celia Riga. Toward a Versatile Robotic Platform for Fluoroscopy and MRI-Guided Endovascular Interventions: A Pre-Clinical Study. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5411–5418, Macau, China, November 2019. IEEE.

[21] Dana C. Peters, Robert J. Lederman, Alexander J. Dick, Venkatesh K. Raman, Michael A. Guttman, J. Andrew Derbyshire, and Elliot R. McVeigh. Undersampled projection reconstruction for active catheter imaging with adaptable temporal resolution and catheter-only views. *Magnetic Resonance in Medicine*, 49(2):216–222, February 2003. Number: 2.

[22] Michael Bock, Steffen Volz, Sven Zühlsdorff, Reiner Umathum, Christian Fink, Peter Hallscheidt, and Wolfhard Semmler. MR-guided intravascular procedures: Real-time parameter control and automated slice positioning with active tracking coils. *Journal of Magnetic Resonance Imaging*, 19(5):580–589, 2004. Number: 5 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jmri.20044.

[23] Krishna S. Nayak, Yongwan Lim, Adrienne E. Campbell-Washburn, and Jennifer Steeden. Real-Time Magnetic Resonance Imaging. *Journal of Magnetic Resonance Imaging*, page jmri.27411, December 2020.

[24] YingLiang Ma, Mazen Alhrishy, Srinivas Ananth Narayan, Peter Mountney, and Kawal S. Rhode. A novel real-time computational framework for detecting catheters and rigid guidewires in cardiac catheterization procedures. *Medical Physics*, 45(11):5066–5079, November 2018. Number: 11.

[25] R. van der Weide, C.J.G. Bakker, and M.A. Viergever. Localization of intravascular devices with paramagnetic markers in MR images. *IEEE Transactions on Medical Imaging*, 20(10):1061–1071, October 2001.

[26] Anh Nguyen, Dennis Kundrat, Giulio Dagnino, Wenqiang Chi, Mohamed E. M. K. Abdelaziz, Yao Guo, YingLiang Ma, Trevor M. Y. Kwok, Celia Riga, and Guang-Zhong Yang. End-to-End Real-time Catheter Segmentation with Optical Flow-Guided Warping during Endovascular Intervention. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9967–9973, Paris, France, May 2020. IEEE.