TÉCNICO LISBOA

Robotic navigation system for needle positioning in neurosurgery

João Paulo Mendes Oliveira

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisors:
Prof. Jorge Manuel Mateus Martins
Dr. Manuel Herculano Carvalho

November 2019

I have read a thousand books and lived a thousand lives. This thesis represents a portion of mine.

Acknowledgments

For the past five years I have called Instituto Superior Tecnico home, and it is due to this institution that I have grown both as a person and as an engineer. I would like to express my deepest thanks to the institution that has watched me grow and provided the necessary conditions to always push myself, even when a gargantuan task faces me.

Professor Jorge Martins, advisor of this thesis, is an example of the type of professional that I wish to become in the future. It is due to his invaluable guidance provided through this thesis that I have persevered and pushed myself when doubt emerged in my mind.

I would also like to thank Dr. Manuel Herculano for his guidance while developing the strategies used for the neurosurgical procedure and the chance to observe a neurosurgical procedure, that ultimately, gave me precious insight into the workflow inside an operating room that cannot be understood unless experienced first hand.

A special thanks to my partner in life for accompanying me during this challenging time, and throughout the course. It is due to you that I have kept pushing myself more and more each day.

To my family, I have nothing but gratitude. My sister for listening to my ideas, for drawing the gorgeous figures shown in this thesis and for instigating my creative mind. My father and mother for the lessons taught during my life, for the unconditional love and belief in my capabilities.

During the past five years i have enjoyed special friendships that will leave an everlasting mark on me. With you I have experienced both success and failure, and learnt to deal with these experiences along side you all. Thank you for challenging and accompanying me in these chaotic times.

Last, but not least, my labmates: Rui Coelho for all the help provided along this period and the fun squash games that kept me focused and well exercised and Pawel Bujalski for the company, conversation and advice for future endeavors.

Resumo

Com o recente foco em novas soluções tecnológicas em todo o mundo, o campo da neurocirurgia não se mostrou indiferente a esta mudança. Na última década diversos grupos de investigação têm desenvolvido soluções de software que permitem uma integração mais simples ao sistema complexo presente numa sala cirurgica. Um destes avanços é o conhecido protocolo de comunicação OpenIGTLink, que permite que diferentes dispositivos comuniquem entre si, independentemente da arquitectura que cada um possui internamente.

Neste trabalho, o recente robot LBR Med é explorado e uma implementação compatível do protocolo OpenIGTLink é desenvolvida em Java, de forma a permitir que este sistema robótico seja integrado com os restantes dispositivos médicos, tipicamente usados num contexto hospitalar.

De forma a validar os componentes desenvolvidos e criar bases para futuras iterações deste trabalho, é desenvolvida uma plataforma de navegação cirurgica que usa como módulo de visualização o 3D Slicer e a biblioteca de software Plus como modulo de recolha e processamento dos dados obtidos de todos os dispositivos.

O sistema robótico baseia-se numa estrutura de controlo partilhada, através de controlo de impedancia, onde o cirurgião tem um papel ativo no posicionamento do robot, de forma a tornar a transição para um sistema deste tipo mais suave e natural. Com a introdução do registo por ultrasom é possivel criar um link entre as coordenadas internas do robot e as coordenadas internas do paciente, que pode ser classificado como um sistema de registo não evasivo.

Palavras-chave: OpenIGTLink, 3D Slicer,LBR Med, Controlo por impedancia, Localização de ultrasons, Java

Abstract

As the world moves towards more technological solutions, so does the field of neurosurgery. Several research groups have developed software components which add flexibility to the medical system, such as the OpenIGTLink protocol, that establishes connections between medical devices, thus allowing them to communicate between each other.

In this work we try to add functionality to the new LBR Med robot by implementing a compatible version of the OpenIGTLink protocol in Java, thus allowing the robot to communicate with all the medical devices compatible with the widely used protocol.

A prototype platform, to serve as a proof of concept, was developed using the 3D Slicer as a visualization tool and the Plus Toolkit to serve as a data collector. All the software was developed in a modular fashion so that researchers can further improve on top of this work.

The robotic system relies on a shared control structure where the neurosurgeon is an active participant in the control of the robot, thus a seamless interaction between the robot and the physician is achieved. Due to the use of the inexpensive ultrasound modality a link between robot coordinates and physical coordinates of the OR is created, thus creating a non evasive registration procedure.

Keywords: OpenIGTLink, 3D Slicer,LBR Med, Impedance control, Tracked ultrasound,Java

Contents

List of Tables

List of Figures

Nomenclature

κ	Closed inverse kinematics
μ	Average of a set
R	Rotation matrix
S_R	Joint controller
sd	Standard deviation
T_a^b	Homogeneous transformation from frame **a** to frame **b**
x, y, z	Cartesian components
DLR	German Aerospace Center
JVM	Java virtual machine
OR	Operating room
ROI	Region of interest
SRL	Surgical robotics lab from IST
US	Ultrasound

Chapter 1

Introduction

1.1 Motivation

In the past decades a large market has emerged for surgical robotic systems due to their precision, the ability to augment the surgeon capabilities and to compensate human shortcomings, such as concentration, physical and endurance capabilities.

Due to this new market, several companies developed robotic systems specialized in certain surgical fields. By creating robotic systems with physical constraints that eliminate the need of complex control laws, for example physical constraints that force the tool rigidly attached to the robot to only move along a specific point in space, one can develop a robotic system specialized in endoscopic procedures. The same logic can be applied to other surgical procedures.

This strategy creates robotic systems which specialize in niche surgical procedures, thus increasing the vertical capabilities of that specific robot at the cost of horizontal freedom, meaning that one robotic system cannot be used in an workflow different from the one which it was designed to operate in.

The solution to the previous loss of horizontal freedom lies in developing complex robotic systems, capable of implementing different control strategies depending on the operating conditions, thus moving the problem from hardware design to software design. Robots which follow this strategy allow development teams to save precious time, thus allowing them to tackle different surgical procedures.

This thesis explores the applicability of the LBR Med in the neurosurgical field. It explores the software embedded in the robot. By setting performance benchmarks and exploring the limitations of the robot one can both tackle neurosurgery, developing the groundwork and all the necessary additional hardware to complement the final system and also provide guidance in other applications for future developers.

1.2 Objectives

This thesis sets out to achieve three objectives.

The first, and the most important, objective is the exploration of the capabilities of the LBR Med in the context of the neurosurgical field. By pushing the robot to its limits one can hopefully reduce future development time given the benchmarks reported by this work.

The second objective focuses on developing software which facilitates communication with devices typically found in a surgical setting. Although KUKA provides communication structures, they are too rigid to adapt with other medical devices. By developing an implementation of the known medical protocol OpenIGTLink, compatible with the robot's operating system, more flexibility is added to the robot, thus increasing the possible operating conditions in which it can be integrated.

The third objective is the exploration of the known registration problem in neurosurgery. Through the use of ultrasound technology, known to be affordable when compared to other imaging modalities in the medical field, the author hopes to continue the work of previous researchers and probe the potential of the technique in the framework of the registration problem.

1.3 Thesis Outline

This work is divided into four main chapters. The second **chapter** introduces the reader to the neurosurgical field, through its history, how certain milestone technologies were developed and contextualizing the reader on the main problems that define the field. Relevant robotic systems in neurosurgery are introduced, new techniques useful for this work are introduced, as well as work similar to the one developed in this thesis is explored.

The third **chapter** introduces all the software and devices used in this thesis. The LBR Med underlying control architecture is briefly explained in the hopes of simplifying the learning stage of future developers. The OpenIGTLink protocol is explored, some context on the past Java implementations explained and the final communication structure is shown. Last but not least, the integration of all the components is explained.

The fourth **chapter** explains the design choices for the different experiments, the results and relevant analysis for each result and resulting implications. The first component tested is the Java implementation of the OpenIGTLink protocol in the context of this thesis. The second component analysed is the LBR Med performance. Specifically the underlying control law used by the robot's controller and the corresponding higher level control strategies proposed in this thesis. The last hypothesis tested in this chapter is the potential of the ultrasound imaging modality to solve the registration problem by tracking the US image with the LBR Med, thus providing the robot with information about its environment.

The fifth **chapter** briefly reflects on the achievements of the thesis, future developments and possible improvements on the overall system.

Chapter 2

Background

2.1 Neurosurgery

According to the definition provided by the American Board of Neurological Surgeons (ABNS) neurosurgery is defined as

> a medical discipline and surgical specialty that provides care for adult and pediatric patients in the treatment of pain or pathological processes that may modify the function or activity of the central nervous system. For example: (brain, hypophysis and spinal cord), the peripheral nervous system (cranial, spinal and peripheral nerves), the autonomic nervous system, the supporting structures of these systems (meninges, skull and skull base and vertebral column) and their vascular supply (intracranial, extracranial and spinal vasculature). Treatment encompasses both non-operative management (prevention, diagnosis – including image interpretation – and treatments such as, but not limited to neurocritical intensive care and rehabilitation) and operative management with its associated image use and interpretation (endovascular surgery, functional and restorative surgery, stereotactic radiosurgery and spinal fusion) including its instrumentation.

As stated by Barrow and Bendok [1] neurosurgery has shown dramatic developments in its core tools to achieve better and less evasive results. According to Barrow the largest propeller of the neurosurgical specialty in the past decades is the advent of neuroimaging, meaning several non-evasive techniques to probe the brain, usually in the form of image information, allowing for more precise surgical techniques and more information for the neurosurgeon to tackle different anomalies.

2.1.1 The target?

Any project developed in the context of neurosurgery requires a proper understanding of the main target of neurosurgery, the brain.

The brain is the only organ in the human body totally enclosed by bone (Golby [2] p.1) , therefore any intervention to it must deal with this barrier. The second factor that increases dramatically the complexity of any neurosurgery is the non-redundancy of brain tissue, in other words, one region's task is not replicated anywhere else.

In 1908, the first breakthroughs that truly gave birth to the modern definition of neurosurgery were reported [3], and they provided insight on how to bypass the two previous problems.

The first problematic factor, the total enclosure of the brain by bone, could be overcome by a large craniotomy, thus exposing a large section of the brain. Since the exposure of the brain in open craniotomies has been shown to result in higher mortality rates than other less evasive methods, the technique is used in only specific situations where no other alternative is viable [4].

Other solutions rely on burr holes, to only provide an access point to the space inside the skull, but said methods require complex navigation tool (Golby [2] p.3-8).

The second factor, the non-redundancy of brain tissue, requires an extremely precise targeting of the lesioned region, and a large amount of planning is required to define a trajectory that does not damage "eloquent areas" along the path [2].

2.1.2 Tools to probe the brain

As stated in the previous chapter several solutions were developed that bypassed the need for craniotomies. Arguably the most important was the stereotactic frame developed by Horsley and Clarke [5] in 1908.

Their experiments focused on obtaining a brain from a recently deceased animal, gridding it with a cartesian frame of reference and targeting the same point on a living animal, as seen in figure 2.1. The correspondence between the deceased and living animals was established by the use of external features.

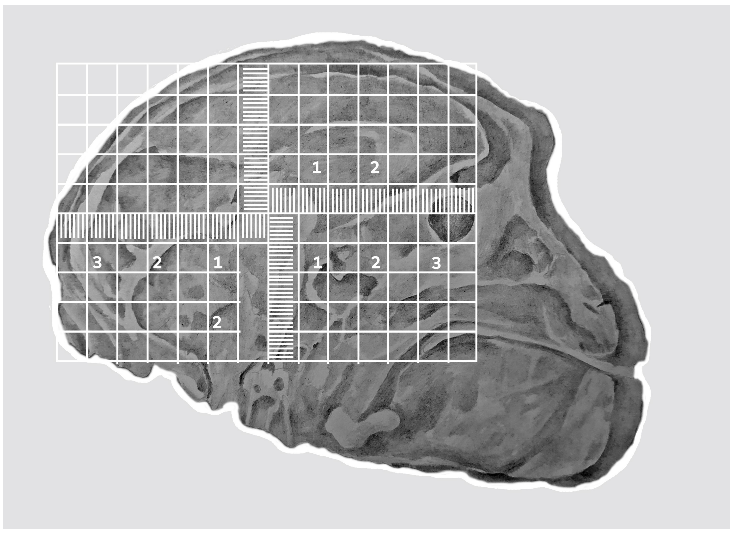


Figure 2.1: Illustration of tracking experiments on dogs performed by Horsley and Clarke (adaptation from Horsley and Clarke [5])

Although they were able to replicate behaviors between different living animals their results had significant variances, thus making them not appropriate for humans [2]. In the end their experiments showed that the inside mapping of the brain is not predictable by external features of the skull.

Given the importance of the mapping so that points inside the brain could be targeted in a systematic fashion independent from the external features of the individual, another 60 years passed until a new revolutionary technology was invented which allowed medical teams to perform the needed mapping.

In 1970 the computed tomography (CT) technique was developed. This imaging technique produces source images that are perpendicular to the long axis of the body (Chen et al. [6], p.4). Each point in the image space is a function of the density of the various targeted tissues measured in Housefield units (HU) [6].

With this revolutionizing technique slices of the CT images are stacked on top of each other to create a volume. From this volume the surgeon can see the structures that compose the brain, thus obtaining valuable 3D information that allows the surgical team to plan the trajectory required to reach the lesioned region.

Although the CT technique propelled neurosurgery to new heights, the contrast between soft tissues is hard to observe. To understand how hard, it is necessary to first understand how the Housefield units are defined. The zero of the Housefield scale is defined by water, while air is defined as -1000. With this scale, the difference between white matter and gray matter in the lies between 30-34 and 37-41 HU, respectively, while the bone is seen in the 700-3000 HU interval (Golby [2], p.46-49).

By computing the relative difference between the three intervals it is easy to understand that in the resulting CT image, the contrast between the soft tissues will be too low to be used without additional filters.

Around 1970, the magnetic resonance imaging (MRI) technique was developed. By subjecting the human body to a strong magnetic field, the free water protons in the body will orient themselves according to magnetic field lines because the proton produces a small magnetic field due to its spin and electric charge. After this alignment two effects occur, the longitudinal relaxation and the transverse dephasing, and by measuring the two effects two different source images with different densities can be retrieved, namely the T1 and T2 weighted images (Golby [2], p.25-28).

The MRI provides a clear contrast between the different types of soft tissue present in the brain, thus improving on the earlier CT technique.

Although the MRI is a powerful tool one must recognize its drawbacks such as image distortion, the metallic incompatibilities with pacemakers, claustrophobia due to the small space that the patient must be kept, and possible weight restrictions on the patients (Golby [2], p.49-51).

2.1.3 Registration

A common mathematical framework between all the imaging modalities is the frame of reference associated with any image. Any pixel that composes a medical image can be described by a vector in the frame of reference of the image. The base of the vector can be changed through a homogeneous transformation, shown in equation 2.1.

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_{14} \\ r_{21} & r_{22} & r_{23} & p_{24} \\ r_{31} & r_{32} & r_{33} & p_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

where R represents a rotation matrix and p represents the translation part of the transformation. In the medical field the change from one frame of reference to another is also called registration.

The coordinate systems typically used in the medical field are:

The image space

The image space is the coordinate system associated with a given image. Any pixel in an image can be uniquely identified by three coordinates [2].

The physical space

The physical space, associated with the patient skull, can be characterized by a coordinate system. Any point inside the patient brain can be uniquely defined by three coordinates [3].

Registration allows the medical team to create a mapping between the image space and the physical space. With this link it is possible to know the position of eloquent areas inside the brain through the image space and navigate between them in the physical space.

Another important use of this transformation is the mapping between different image modalities [7]. In the previous section the MRI and the CT techniques were introduced with certain drawbacks, specifically the image distortion for the MRI modality and the poor soft tissue contrast for the CT modality. By mapping points from the MRI image space to the CT image space it is possible to create an image without the two drawbacks[7].

Precise knowledge of the registration between the involved spaces is a crucial step in the neurosurgery field for both the procedure itself, due to the required mapping between image space and physical space and the correction of deformations of the different imaging modalities.

2.1.4 Stereotactic navigation

The last link to perform any neurosurgical procedure is the physical apparatus used to insert the needle in the brain. According to Henn et al. [3] there are two types of devices called linked interactive localizing devices and non linked interactive devices.

A linked interactive localizing device is classified as such if a physical link between the patient and the tool is necessary. In this set up the patient's skull is rigidly attached to the operating table, removing all degrees of freedom of the patients skull, and the device is also rigidly attached to the operating table which in turn forces the movement of the patient to be congruent with the tool. The physical link allows the transformation associated with with the device and the patient to be fixed and constant. This type of set up is also called frame based stereotactic navigation [2].

A non linked interactive localizing device tracks the position of the patient in relation to the tools used in the surgical procedure. Since this transformation maps the tool in relation to the patient skull, there is no need to establish a physical connection between the two to keep the transformation static.

Usually the technology used by non linked devices relies on either sonic triangulation using spark gap emitters, visual trackers or electromagnetic [3].

Sonic emitters rely on the concept of triangulation to compute the position of the emitter in the physical space. Visual trackers rely on stereo cameras to position the trackers on a frame of reference. Electromagnetic emitters rely of the same concepts as the sonic emitters.

Non linked interactive localizing devices are also classified as frameless stereotactic navigation systems [2].

Frame based systems are know to have excellent mechanical stability and possess accuracies of 0.3 to 0.7 mm [2]. Frameless systems show consistently good results, comparable to frame based systems and usually the choice between the two systems depends on how each group use their resources, what type of training they have been through and not so much on the performance of the tracking system itself [2].

2.1.5 Intraoperative imaging

The previous sections already focused on the revolutionary techniques developed through the use of different imaging modalities. The last degree of freedom in this complex system is called brain shift [2].

As mentioned previously in both frameless and frame based stereotactic navigation systems, the patient maintains a relationship with the tool, either variable or static, but when the surgeon creates an opening in the skull the difference between the intracranial and atmospheric pressure displaces and deforms the brain [8].

Given the unaccounted deformations, as an example tissue displacement, which affects the position on any given point in the brain by up to 10 mm [8], it became apparent since the birth of image guidance that there is a need for real time imaging of the brain to compensate for these effects.

There are some interesting solutions developed to minimize the effects of said disturbances, such as re imaging the patient during the procedure, so that the new image set replaces the old data set, or perform affine transformations on the data set obtained in the preoperative stage through the use of ultrasounds [9].

2.2 Current development efforts

2.2.1 Robotics in the operating room?

Neurosurgery is defined as a field that searches for evermore accuracy and precision. The tools needed to target the lesioned areas inside the brain must respect the required stability and accuracy specifications.

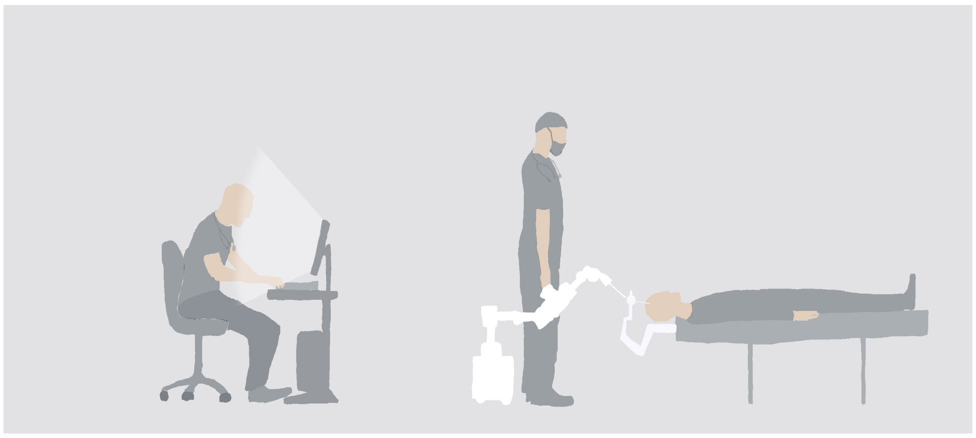


Figure 2.2: **Supervised controlled systems** The surgeon preoperatively plans the trajectory to reach the target and during the surgery the surgeon plays a passive role, one in which he supervises the systems to correct or stop any mistake (adapted from Nathoo et al. [10])

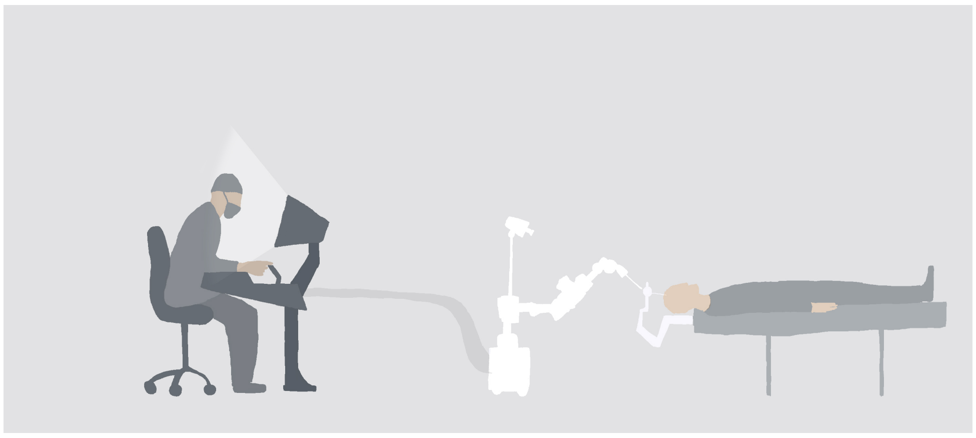


Figure 2.3: **Telesurgical systems** The surgeon controls the robot through a capable interface, either position or force control and receives feedback from the end effector so that he can make real time decisions (adapted from Nathoo et al. [10])

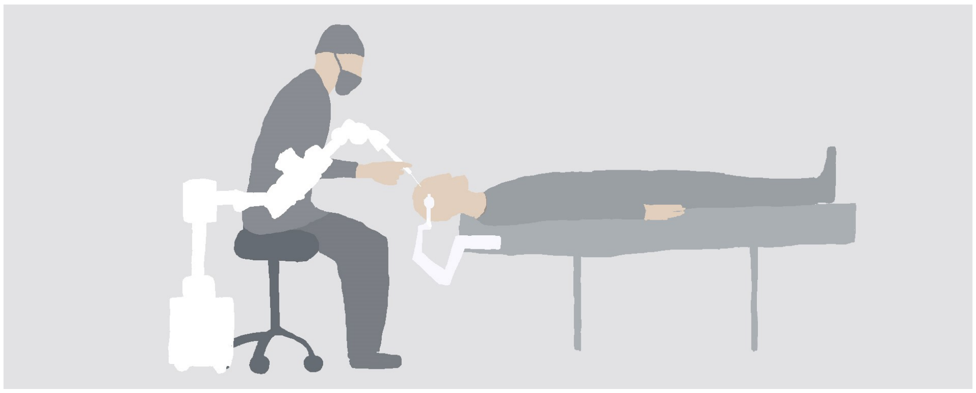


Figure 2.4: **Shared control systems** The surgeon controls the surgical tool but the robot uses its impendance interface so that the forces exerted between the surgeon and the end effector can be modeled as a second order system (adapted from Nathoo et al. [10])

Robotic systems were a promising addition to the neurosurgical field and it still represent one with great potential, since they possess both precision, accuracy and the possibility to be integrated into the different workflows of research groups.

Since 1988, several robotic systems have emerged to respond to the increasing demand of the neurosurgical field. The first of neurosurgical robot was the Programmable Universal Machine for Assembly (PUMA). The PUMA robot showed how safety features could be integrated into the system and gave researchers feedback on how to design the next generations of surgical robots [11].

In 2005, Nathoo et al. [10] classified the different robotic systems depending on the level of interaction with the surgeon. The authors defined three broad categories defined in figures 2.2, 2.3 and 2.4.

Although several robotic systems exist and show promising results there is still one important variable that should be analyzed properly, and that is how much time is required by a neurosurgical group to fully integrate such a system on their workflow. The surgical learning curve is the period of time required for a surgeon to achieve competence in a new procedure [12].

Currently the published literature that tries to quantify the learning curve metric for robotic systems in not heterogeneous and further work is required to provide a good estimate for the metric [12][13].

Due to the lack of information, developers of navigation systems that use surgical robots should try to actively reduce the necessary changes imposed by the robot on the workflow of the surgical teams.

2.2.2 Neurosurgical robots

Several robotic systems are integrated in today's operating rooms (OR) or they are currently being tested by the appropriate authorities.

In chronological order the most relevant robotic systems are:

1. **PUMA 200** The PUMA robot is a six degree robotic system, that uses a CT guided brain biopsy setup. The robot is used for holding a biopsy crannulae navigated by a stereotactic frame attached to the base of the robot[14]. This robot is classified as a supervised control system. PUMA provided a platform to develop many novel instruments in the decades after its introduction.

2. **NeuroMate** The NeuroMate is a stereotactic robot composed by an electromechanical multi joint arm that can be moved in cartesian space with accuracies of $0.7\ mm$ and a paylod of 7 kg[15]. Unlike the previous PUMA robot that uses CT information to guide the stereotactic procedure, NeuroMate uses an ultrasound setup that allows the robot controller to perform the registration between the patients coordinate system and the internal frame of reference of the robot. The robot can be classified as supervisied controlled sytem.

3. **Pathfinder** Pathfinder is a robotic system fused with software to process images from different modalities. This system possesses automatic methods to perform calibrations between the intervening frames of reference in the OR required by the procedure and it possesses several built in safety features [16].

4. **neuroArm** The NeuroArm robot is a seven degree of freedom robot compatible with the MRI imaging modality, built using piezoelectric actuators that can sustain power losses and brake automatically and each joint has a precision of 0.01^o. This robot relies on a human machine interface where the surgeon receives haptic feedback from the tools held by the robot's end effector. This system can be classified as a telesurgical robotic system and currently it represents one of the most promising products on the market[11].

2.2.3 Registration through ultrasound reconstruction

The classical medical procedure to obtain registration between the image space and the patients coordinate system relies on placing the head frame shown in figure 2.5, rigidly attached to the patient's head in four points of contact against the surface of the skull [17]. Afterwards the patient is taken to an CT scanner, as seen in figure 2.6, so that a transformation between the CT scan and the MRI use to plan the path to reach a certain point, can be created. Because the head frame has markers which can be seen in the CT scan, it is possible to map points in the image space to points surrounding the patients skull through the head frame.

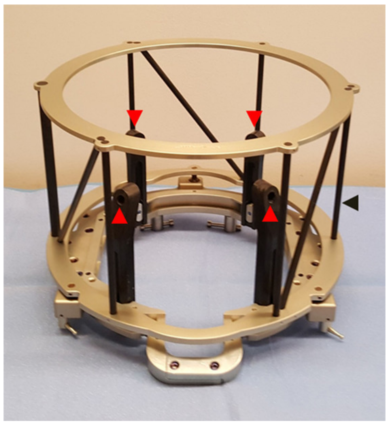


Figure 2.5: Cosman–Roberts–Wells (CRW) head frame used to mantain a constant relation between patient skull and imaged coordinates (from Falconer et al. [17])

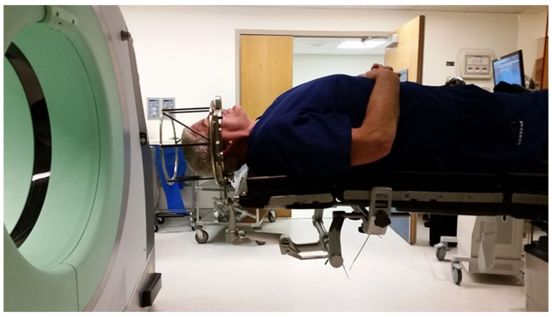


Figure 2.6: CT scan with head frame placed on patient (from Falconer et al. [17])

To introduce a non-evasive registration procedure between the image space and the physical space associated with the patient and compatible with a robotic system, the work of Amin et al. [18] provides a viable option. In summary by tracking the ultrasound probe it is possible to reconstruct the surface of a bone structure and perform a mapping between the reconstructed bone surface and the surface present in the MRI or CT scan, thus obtaining a transformation between the physical space and the image space.

With this technique the head of the patient must still be fixed through the use of mayfield pins [2] seen in figure 2.7.

The reason why this registration procedure is viable rests on the fact that bone structures don't typically show noticeable deformations for a wide range external forces. They possess the correct mechan-

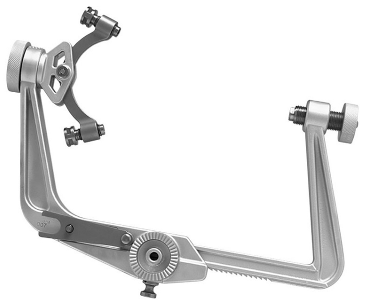


Figure 2.7: Mayfield stabilizing pins with three contact points exist between the skull and the mechanical structure (from Falconer et al. [17])

ical properties for surface identification, because their geometry is kept constant between the MRI/CT scan and the ultrasound scan performed prior to the beginning of the surgical procedure.

The advantage of this technique lies in the reduction from four points of contact to three points of contact, diminishing the evasiveness of the process and US technology does not entail any danger to the patient, unlike the CT scan, which requires the use of radiation.

The most complex step in this non-evasive registration method is the volumetric reconstruction of the bone surface from US images. There are several open source choices available to process US images, such as the public software library for ultrasound (PLUS) toolkit [19], the medical imaging interaction toolkit (MITK) [20], among others.

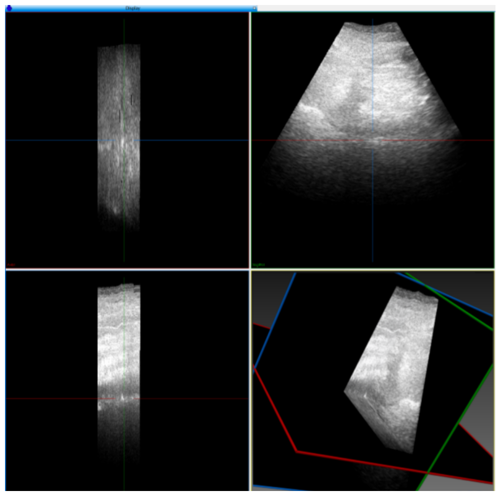


Figure 2.8: Volume reconstruction of US images using MITK (from Seitz [21])

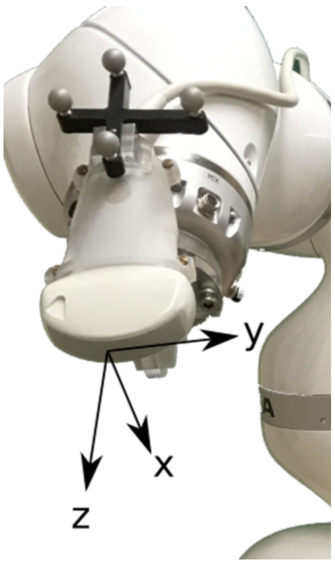
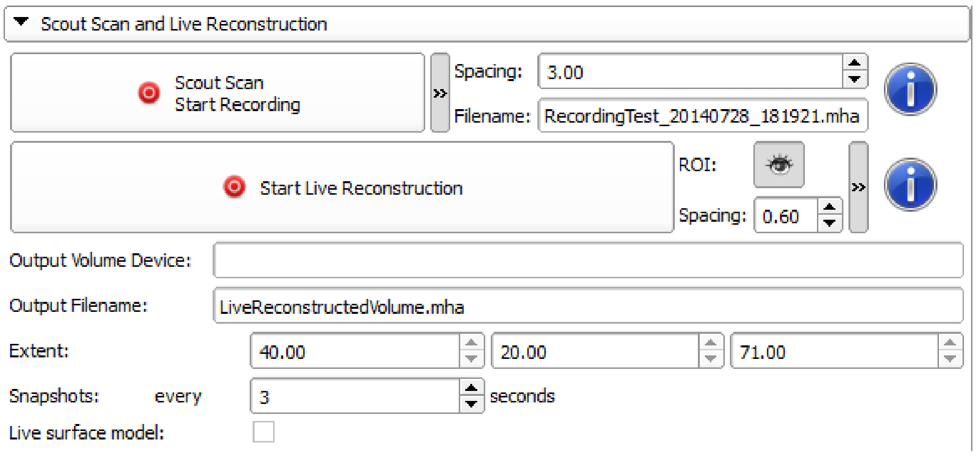


Figure 2.9: LBR Med with US probe rigidly attached to the flange of the robot (from Seitz [21])

The fusion between the LBR Med and an ultrasound scanner to reconstruct the US volume, usign the MITK library, has already achieved by Seitz [21]. In his work the US images retrieved from the US probe rigidly attached to the robot, seen in figure 2.9, are compiled and positioned in the robot's frame of reference, seen in figure 2.8, thus forming voxels containing the US image information.

Plus library has some advantages in relation to the MITK library since the latter is fully integrated into a application programming interface (API) removing flexibility to the source code.

The Plus toolkit can fuse several tracking streams and image stream just like Steitz did with the MITK library. Meyer et al. [22] developed a simple user guided interface (GUI) that uses the plus library, seen in figure 2.10, that allows the surgical team to perform US volumetric reconstructions in a seamless fashion.


Scout Scan and Live Reconstruction

Scout Scan Start Recording

Spacing: 3.00

Filename: RecordingTest_20140728_181921.mha

Start Live Reconstruction

ROI:

Spacing: 0.60

Output Volume Device:

Output Filename: LiveReconstructedVolume.mha

Extent: 40.00 20.00 71.00

Snapshots: every 3 seconds

Live surface model:

Figure 2.10: GUI integrated in 3D Slicer to reconstruct US images into volumes using the Plus toolkit (from Meyer et al. [22])

Using the reconstructed structures obtained through the scout scan, seen in figure 2.11, one can define a region of interest (ROI) and perform a live reconstruction with higher resolution, which can also be seen in figure 2.11.

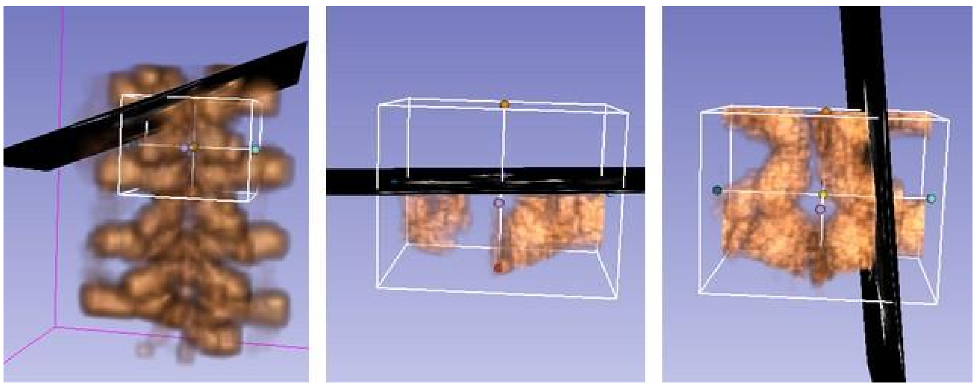


Figure 2.11: Result of spinal scout scan (left) and live ultrasound scan(center,right) using the Plus toolkit and 3D Slicer GUI (from Meyer et al. [22])

With the previous software it is simple to recreate surfaces and thus perform the non-evasive registration procedure.

2.2.4 Platforms for image guided neurosurgery

As seen in the previous sections, there is a real need for software that allows the surgeon to interact with the robot and the medical devices in a seamless fashion. This software is typically bundled as a GUI which provides simplistic representations of the state of the system.

One such navigation platform developed specifically for neuronavigation is the intraoperative brain imaging system (Ibis) image guided platform that integrates most of the functionality of its commercial counterparts. It's GUI can be seen in figure 2.12.

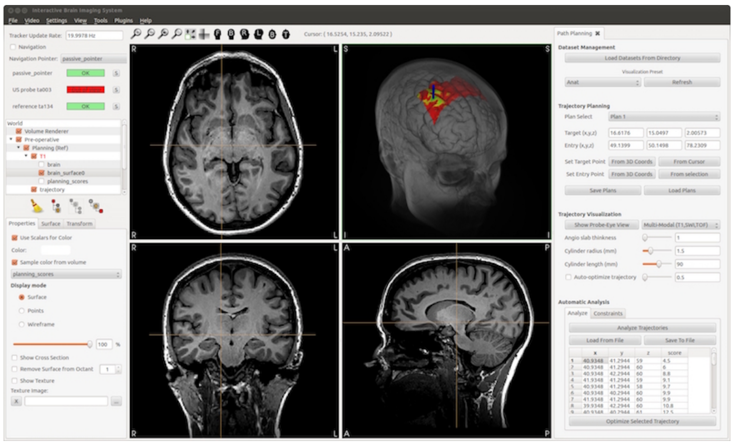


Figure 2.12: GUI of Ibis navigation platform (from Drouin et al. [23])

The Ibis platform was developed with an architecture which allows the community to contribute with new modules. Because the platform relies on known libraries, like the OpenIGTLink protocol, it is compatible with other development tools and with most medical hardware products [23].

Another noteworthy navigation platform is the CustusX platform. It is managed by the Norwegian university of science and technology and it has already experienced use in a surgical setting. The GUI can be seen in figure 2.13. The platform uses the OpenIGTLink protocol, thus making it compatible with most medical devices [24].

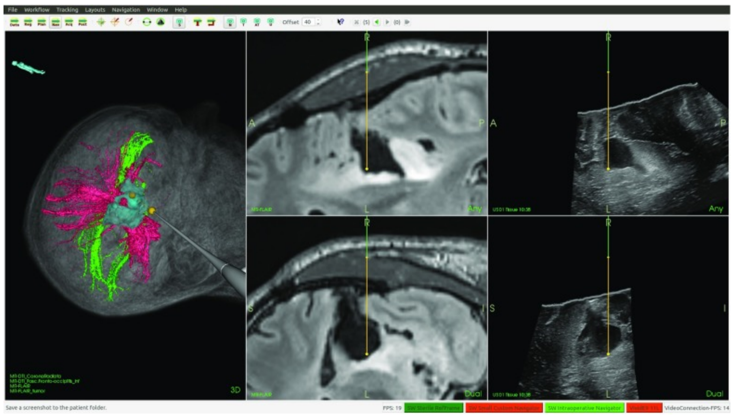


Figure 2.13: GUI of CustusX navigation platform (from Askeland et al. [24])

Both Ibis and CuscusX are the culmination of good implementation practices, being both compatible with open libraries used by research groups and allow the seamless integration of new modules, which can be developed by the communities of the respective navigation platform.

Although these platforms respect good implementation practices the actual integration of these types of guidance systems has been slower than most had initially anticipated [25]. The main reason for this slow adaptation lies in the complexity of implementing from scratch the underlying architecture of a navigation system, which is both safe and flexible.

When a research groups develop such systems, they are usually designed with an architecture which is not compatible with future additions to the system. This incompatibility removes the incentive to change the underlying software thus reducing the potential developments of guidance platforms [25].

2.2.5 Previous work developed at the Surgical Robotics Lab

As with all projects, this thesis is the culmination of previous projects developed at the Surgical Robotics Lab. To understand some design choices in this thesis a closer look at the previous research topics at the SRL is necessary.

In the past several years a number of projects focused on the integration of robotic systems into the OR have been proposed. For the purpose of this thesis two previous projects provide the ground work, inspiration and the also benchmarks necessary to decide where to focus exploratory work and how to develop the navigation system.

The two projects are:

1. ***A co-manipulation robotic system for brain biopsies***

   The work, done by Pedro Miguel Pires, focused on developing the workflow of the robot and the impedance environment as well as improving on the accuracy of the system which was first develop in Tiago Salgueiro's work [26].

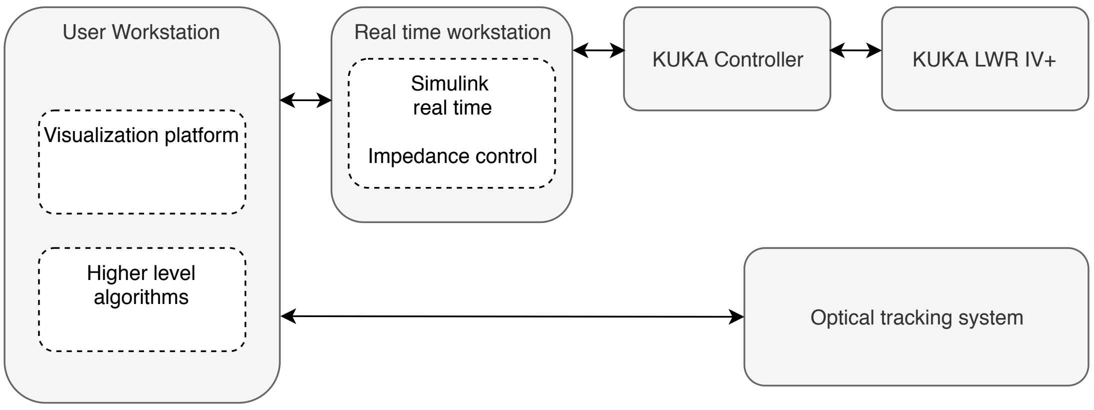


Figure 2.14: The set up uses an optical tracking system to perform the registration between the robots coordinates and the patient coordinates (adapted from Roios [27])

   His set up, seen in figure 2.14, relied on two computers running concurrently. The user workstation possesses an interface that both provides feedback to and receives inputs from the surgeon to control all the devices in the set up [27].

   The real time workstation processes the current pose of the robot and defines the force vector that the robot should impose on the environment.

   The optical tracking system allows the set up to solve the registration problem between the robot and the patient's coordinates.

Although the results obtained with this architecture show promising prospects, there is an underlying problem that could diminish the adoption of the set up. Due to the two workstations and the robot's controller, the whole system loses mobility, which can be a problem for hospitals that do not have the necessary funding to create a single operating room dedicated to surgeries that use the robot abilities.

2. ***Intraoperative registration in neurosurgery using the ultrasound modality***

   The work done by Mariana Elyseu focuses on the problem of registration between the patient's frame of reference and the tools used to perform the surgical procedure [28].

   This thesis showed several important results. The algorithm used to perform the mapping between the surface extracted from the US volume and the MRI/CT surface is the key behind the success or failure of this technique [28].

   Another important factor that influences the accuracy of the system lies in the calibration between all the intervening frame of reference, as it is expected, and the asynchronous data collection of image and pose of the US probe.

2.3 Proposed work

This thesis establishes the ground work required for a neurosurgical navigation platform. Given the size of a project of this nature only a portion of the navigation platform is developed, as seen in figure 2.15.

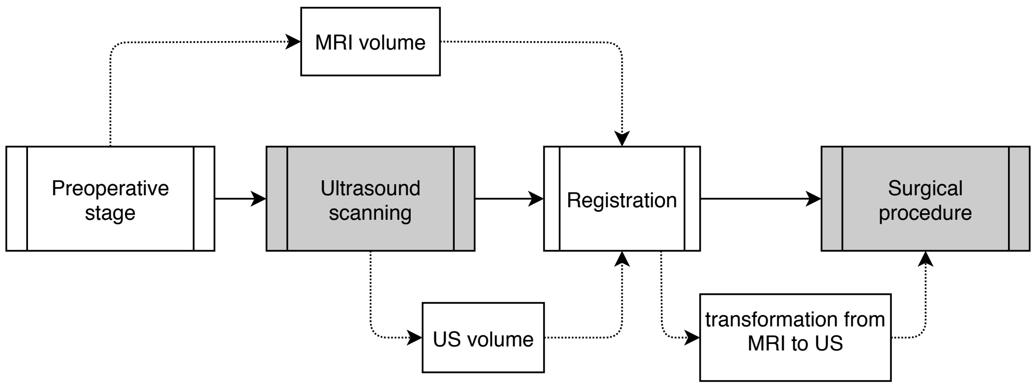


Figure 2.15: Stages of the surgical procedure that were tackled in this thesis (gray) and stages left for future works (white). The dashed arrows represent the objects retrieved from the specific stage and by which stage they are used.

The preoperative stage involves the analysis of the MRI data obtained prior to the surgical procedure, in detail it involves the path planning required to reach the target inside the brain. The ultrasound scanning stage requires that the patient is scanned with an US probe, so that the coordinates of the scanned structure, e.i surface of the skull, are known in the robot's coordinate frame. The registration stage uses the data both from the preoperative stage and the ultrasound scanning to obtain a transformation between the robot's coordinate frame and the image space, where the path planning was defined. The last stage, the actual surgical procedure, uses the transformation between image space and the robots coordinate system to align the end effector of the LBR Med with the trajectory previously defined.

The set up defined for this thesis, seen in figure 2.16, takes some inspiration from the previous architectures shown in 2.14 and reduces the amount of hardware required to perform the surgical procedure, thus increasing the portability of the overall system.

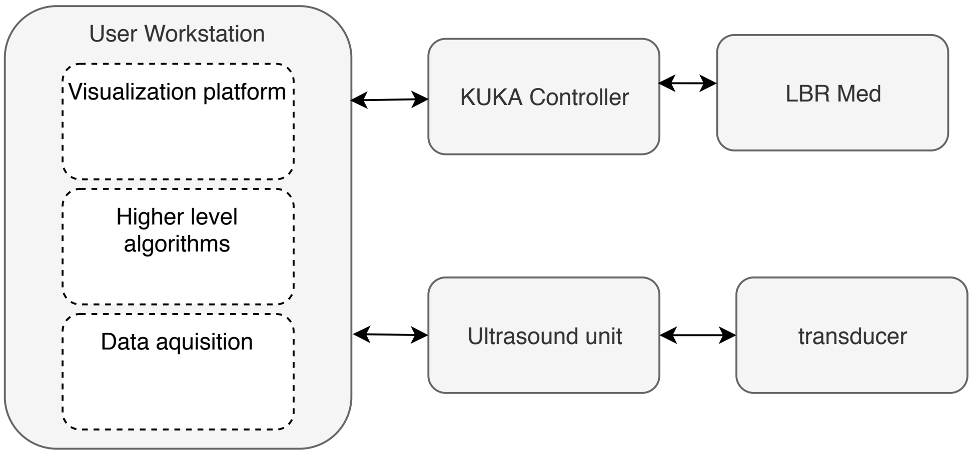


Figure 2.16: The registration is performed using an US system and there is only one computer (excluding the dedicated robot controller) to interact with all the devices.

With only one computer required, the robot could be moved in a small portable unit and the the surgical team only needs a single computer to interact with all the hardware and software.

Chapter 3

Materials and implementation

3.1 KUKA lightweight robot

The KUKA LBR Med lightweight robot is the culmination of all the control strategies developed for the KUKA iiwa robot with the safety features necessary to pass the scrutiny of the medical field. This new robot has five characteristics which make it the ideal tool for this thesis. All the following definitions were retrieved from the documentation provided by KUKA [29].

1. **Precision:** *The LBR Med requires no additional devices for calibration or precise work. Thanks to its integrated mastering sensors it calibrates itself fully autonomously and achieves an outstanding repeatability from ± 0.1 mm to ± 0.15 mm.*
2. **Flexibility:** *The LBR Med is designed as a robot that can be deployed universally. It can be integrated seamlessly into a wide range of different applications. The required interfaces come as standard in large numbers, as the robot is based on the iiwa that has proven its worth in Industries 4.0 settings.*
3. **Safety:** *The LBR Med sets standards with its safety structures. Its safety rated hardware and software's process relevant data.*
4. **Sensitivity:** *The robot has redundant integrated torque sensors. It can detect forces applied externally and react according to the freely programmable system responses you have specified. Benefit from its haptic capabilities for manual guidance, teleoperation with haptic support or gravity compensation... Furthermore the integrated sensors are also used for safe collision detection, thereby enabling human robot collaboration (HRC).*
5. **Certification:** *The KUKA Med is the first lightweight robot wordwide to be certified for integration into a medical product. Its certified is based on the ECEE CB Scheme. Given this certification any further approval process based on the LBR Med is substantially reduced.*

Since the LBR Med was licensed by the German Aerospace Center (DLR)[1], one can understand

[1] According to the document published by the DLR: History of the DLR LWR

the internal control strategies implemented inside the robot controller by analysing the control strategies embedded in the DLR lightweight robot.

The control structure implemented in the DLR is shown in figure 3.1. The joint control is computed in a decentralized manner, where the desired torque and position are required, and the derivative of these signals is computed to create a full state feedback controller [30].

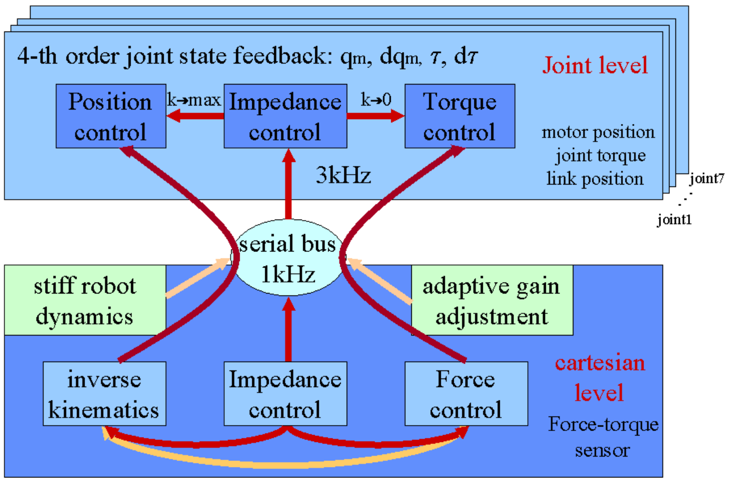


Figure 3.1: DLR control architecture (from Albu-Schaffer and Hirzinger [30])

An extensive analysis of the joint controller is beyond the scope of this thesis. For the intent and purpose of this work, the control law implemented in each joint will only be referenced as the function S_R [31].

The robot dynamics can be modeled by a nonlinear model composed by the links that constitute the DLR, as shown in equation 3.1a, the inertia of the joints by equation 3.1b, and the stiffness of the joints as shown in equation 3.1c [31].

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau + \tau_{ext} \tag{3.1a}$$

$$B\ddot{\theta} + \tau = \tau_m \tag{3.1b}$$

$$\tau = K(\theta - q) \tag{3.1c}$$

where $q \in \mathbb{R}^n$ and $\theta \in \mathbb{R}^n$ represent the link and motor positions respectively. $M(q) \in \mathbb{R}^{n\times n}$, $C(q,\dot{q})\dot{q}$, $g(q) \in \mathbb{R}^n$ represent the inertia matrix of the system, the centripetal and Coriolis vector and the gravity vector. $\tau \in \mathbb{R}^n$ is the joint torque, $\tau_{ext} \in \mathbb{R}^n$ is the external torque exerted by the environment and $\tau_m \in \mathbb{R}^n$ is the motor torque. Finally $K = diag(k_i) \in \mathbb{R}^{n\times n}$ and $B = diag(b_i) \in \mathbb{R}^{n\times n}$ are the diagonal positive definite joint stiffness and joint inertia matrices, respectively.

The control structure followed by the DLR relies on an inner loop controller, the S_R joint loop controller, and the outer loop controller which transforms the desired cartesian behaviour into the desired joint torques and angles [30].

The DLR robot has three potential impedance control laws implemented in the outer loop of the control architecture [30].

The aim of any impedance controller is to recreate a mass damper spring relationship between the positioning error of the end effector, Δx and the force done by the robot on the environment, f, as seen in equation 3.2.

$$f = m\Delta\ddot{x} + b\Delta\dot{x} + k\Delta x \tag{3.2}$$

where m,b and k represent the mass, viscosity and the stiffness respectively.

The first outer loop control strategy is the *impedance controller* which uses directly equation 3.2, where the actual position of the end effector is computed through direct kinematics as $x^{'} = \kappa(q_2)$ and the error, Δx , is fed into equation 3.3.

$$f = K_k\Delta x + D_k\Delta\dot{x} \tag{3.3}$$

The force f is then translated to joint torques by equation 3.4.

$$\tau_d = J^T(q_2)f \tag{3.4}$$

And lastly, through the joint controller S_R, the joint torques are transformed by $\tau_m = S_R\{\tau_q\}$ as motor torques.

The second outer loop impedance control law is denominated as *stiffness controller* and it uses a local mapping between the desired stiffness matrix, K_x and damping matrix D_x and the joint stiffness matrix K_j and damping matrix D_j through equations 3.5a and 3.5b.

$$D_j = \frac{\partial\tau}{\partial\dot{q_2}^T} = \frac{\partial(J(q_2)^T D_k\Delta\dot{x})}{\partial\dot{q_2}^T} = J(q_2)^T D_k J(q_2) \tag{3.5a}$$

$$K_j = \frac{\partial\tau}{\partial q_2{}^T} = \frac{\partial(J(q_2)^T K_k\Delta x)}{\partial q_2{}^T} = J(q_2)^T K_k J(q_2) + \frac{\partial J(q_2)^T}{\partial q_2{}^T} K_k\Delta x \tag{3.5b}$$

The joint impedance matrices are then translated as torque motors by the joint impedance controller S_R as $\tau_m = S_R\{\kappa^{-1}(x), K_j, D_j\}$ where κ^{-1} represents the inverse kinematics computed by the robot's controller.

Since the derivative assumes that the Jacobian is static in position q_2 if the real joint position $q_2{}^{'}$ shows a significant deviation from the equilibrium position, then the mapping proposed in 3.5a and 3.5b will reproduce the stiffness matrix K_k and damping matrix D_k distorted in relation to the desired stiffness and damping matrices.

The final possible impedance control strategy in the outer loop is the *impedance controller enhanced by local stiffness control* where three assumptions are made:

1. $J(q_2)$ and $\kappa(q_2)$ are computed in a slower loop.
2. The robot has a fast joint control loop, which is true for the LBR Med, since the inner control loop runs at 3KHz.
3. Equations 3.5a and 3.5b are valid only locally.

If one performs approximations on the real position and Jacobian the joint torques produced by the stiffness factor are given by equation 3.6a and the joint toques produced by the damping factor are given by equation 3.6b.

$$\begin{aligned}\tau_{dK} = J^T(q_{2k})K_k\Delta x_k + J^T(q_{2k})K_kJ(q_{2k})\Delta q_{2j} + \Delta q_{2j}{}^T \left.\frac{\partial J^T(q_2)}{\partial q_2}\right|_{q_{2j}=q_{2k}} K_k\Delta x_k \\ +\Delta q_{2j}{}^T \left.\frac{\partial J^T(q_2)}{\partial q_2}\right|_{q_{2j}=q_{2k}} K_kJ(q_{2k})\Delta q_{2j}\end{aligned} \tag{3.6a}$$

$$\tau_{dD} = J^T(q_{2k})D_kJ(q_{2k})\Delta\dot{q}_{2k} + \Delta q_{2j}{}^T \left.\frac{\partial J^T(q_2)}{\partial q_2}\right|_{q_2=q_{2k}} D_kJ(q_{2k})\Delta\dot{q}_{2j} \tag{3.6b}$$

In the **fourth** chapter, experiments were designed to probe which outer loop controller is implemented in the LBR Med controller.

With the control architecture introduced, we can now classify the robot according to the categories introduced by Nathoo et al. [10].

With the impedance interface embedded in the controller, one can create a shared control system where the surgeon can guide the robot to the desired target.

In the work done by Tauscher et al. [32], using the OpenIGTLink protocol, the LBR Med robot is controlled with a cyclic communication architecture between the robot controller and an outside workstation. Due to the cyclic architecture, this set up can be classified as a telesurgical system.

Lastly, the supervised control system category can be implemented by simple position control with an emergency switch capable of triggering the brakes of the robot.

The potential to integrate the LBR Med in all possible categories shows how versatile the robot is.

3.1.1 Sunrise workbench

The Sunrise OS is the new operating system (OS) designed by KUKA to interact with the LBR Med [29]. The new OS provides the Workbench application programming interface (API) to program the robot using the Java languange.

By merging the real time capabilities of the VXWorks real time OS and the Java Virtual Machine (JVM), it is possible to achieve higher abstraction levels, thus facilitating any kind of application development.

The organization of a project is an important characteristic of the Workbench API. This understanding is beneficial when developing complex applications. Normally a project is divided as seen in figure 3.2.

In any Workbench project, there are four main folders necessary for a safe and well designed application. They are:

1. **src folder:** The source folder contains all the classes that can interact with the LBR med and the robot controller.

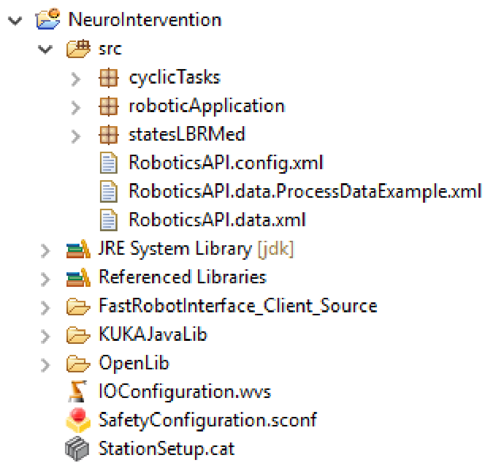
2. **Libraries:** The libraries in Java provide functionality to an application, either through the java run-

time environment (JRE) system library which allows the developer to interact with core hardware components, or the KUKAJAVAlib which provides the necessary classes to manipulate the robot controller or user libraries, as seen in 3.2 called OpenLib.

3. **IOConfiguration:** The IOConfiguration uses the Workvisual software developed by KUKA to interact with external devices. This allows the developer to create a set up without the need of external PLCs to control simple components, like a LED system that signals the internal states of the robot.

4. **SafetyConfiguration:** The safety configuration allows the developer full access to the safety capabilities implemented in the LBR Med.

5. **SationSetup:** The station set up serves as a configuration file that defines the robot and the controller used in the project.



```
NeuroIntervention
  src
    cyclicTasks
    roboticApplication
    statesLBRMed
    RoboticsAPI.config.xml
    RoboticsAPI.data.ProcessDataExample.xml
    RoboticsAPI.data.xml
  JRE System Library [jdk]
  Referenced Libraries
  FastRobotInterface_Client_Source
  KUKAJavaLib
  OpenLib
  IOConfiguration.wvs
  SafetyConfiguration.sconf
  StationSetup.cat
```

Figure 3.2: Folders generated by the Workbench API when a project is created.

Normally any given application will follow the following baseline strucutre:

```
public class ApplicationExample extends RoboticsAPIApplication {

@Inject
private LBRMed lbr;
@Inject
private ITaskLogger logger;
@Inject
private Tool tool;
private MedController medController;

@Override
public void initialize(){
    // Initialize the needed variables to control the robot
}

@Override
public void run() throws Exception {
   // Control the robot to so some action
}

@Override
public void dispose(){
   // dispose the current user objects and threads ...
    super.dispose();
}
}
```

When an application is started, either through the SmartPad or through an external signal, the class that extends the **RoboticsAPIApplication** is instantiated and the *initialize()* is called.

Once the *initialize()* method returns the handle to the calling thread the *run()* method is called.

If the application is terminated, either through forced termination in case of an error or because the programmed task has finished the *run()* method returns the handle to the calling thread and the *dispose()* method is called to clear all the memory allocated during the life time of the application.

3.1.2 Safety Configuration

The safety configuration embedded into the KUKA controller is arguably the most important piece of software that the controller has. It allows the LBR Med to obtain its safety certification according to the IEC 62304 norm[29].

The software running inside the controller can be classified into two categories, class C software and class B software. Class C software means that death or serious injuries are possible in case of failure while class B category is reserved for the software which non serious injures can happen in case of failure.

To receive the IEC 62304 certification the safety controller portion of the LBR controller is classified as class C software, and it has been tested in severe conditions.

The code developed by the software developer working with the Workbench API is categorized as class C software and resides in the motion controller part of the controller [29].

With the previous architecture it is possible to develop safety functions that monitor the LBR Med motions, using the full capabilities of the safety controller, to decrease the risk associated with errors thrown by the motion controller and thus bringing the risk of the motion controller down to class B software.

The Sunrise.OS achieves this by defining two different monitoring mechanism [29]:

- Permanent safety-oriented monitoring

The safety functions of the PSM Mechanism are always active. It is only possible to deactivate individual safety functions by changing the safety configuration. The PSM mechanism is used to constantly monitor the system. It implements basic safety settings which are independent of the process step being carried out.

- Event dependent safety-oriented monitoring

The ESM mechanism (Event-driven Safety Monitoring) defines safe states. It is possible to switch between these in the application. A safe ESM state contains the safety functions required in the corresponding process step. Since switching is carried out by means of non-safety-oriented signals, the defined state must ensure a sufficient degree of safety, regardless of the time or place of activation.

These two abstractions were developed and tested by Haddadin et al. [33] and through the use of their safety observers it is possible to obtain reaction times in small time windows, depending on the type of response programmed by the developer.

3.1.3 Workbench classes for path planning and robot control

To use the full capabilities of robot it is necessary to understand how the Java virtual machine (JVM) interacts with the controller to define trajectories and control laws embedded in the controller, introduced in section 3.1, to follow said trajectories.

Arguably the most important classes to interact with the robot's environment are the classes used to define cartesian frames of reference that allow the robot to move in its surroundings. In figure 3.3, the three types of frames that the robot uses are introduced.

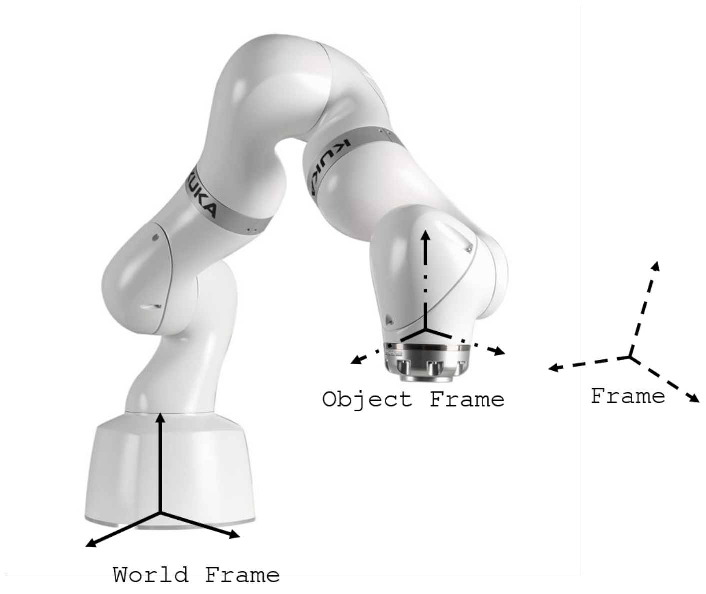


Figure 3.3: Frames of reference that the robot can interact with.

The *World Frame* is the frame of reference that allows the robot to know where it is positioned. All frames used in a Workbench application are frames transformed in relation to the *World Frame*.

Object Frames typically represent tools which can be attached to the flange of the robot. As an example assume that the developer possesses a gripper that has a frame of reference F_1 attached to its extremity. To use F_1 in an application the developer is required to attach said frame of reference to the robot in the application, so that the robot can control the positioning of the frame F_1 in relation to the *World Frame*. In summary, it is a dynamic frame of reference.

The last class that defines cartesian frames of reference is the *Frame* class. Objects of this type represent points in space that the robot can move to. Frames of this type are static and cannot be attached to the robot.

The second important type of classes that the robot uses to move from one frame of reference to another are the motion classes which encode both trajectories and control laws used by the robot to move in the environment. All the classes that the developer can use are introduced in figure 3.4.

All the trajectories extend the *IMotion* interface. To move the robot to a frame the developer can call either the *move()* method of the robot, which blocks the calling thread until the motion is finished, or

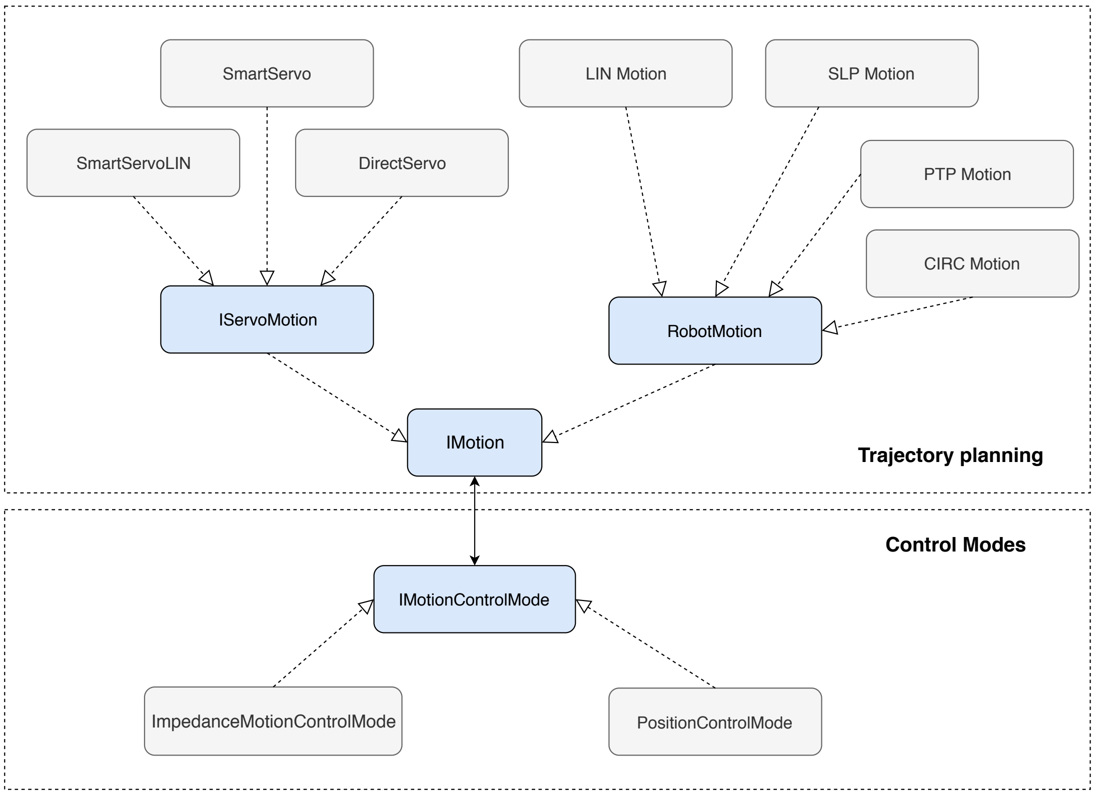


Figure 3.4: Types of intervening classes (in gray) and interfaces (in blue) to generate trajectories and control laws that can follow said trajectories.

the *moveAsync()* method, which is an assyncronous call to the move() method. Notice that any *Object Frame* attached to the robot can call the *move()* method because it is a dynamic frame of reference.

Any call to a *move()* command only accepts a trajectory which implements the *IMotion* interface.

Two main interfaces extend the *IMotion* interface, namely the *RobotMotion* and the *IServoMotion* interface.

The *RobotMotion* interface defines trajectories typically used by industrial robots, i.e. linear or circular motions, called the *LINMotion* and the *CIRCMotion* respectively.

These types of trajectories define the position, velocity, acceleration and jerk profiles, thus some computation time is required by the trajectory interpolator to define said profiles, tipically $100\ ms$. Given this restriction any reaction to external perturbations that the developer wishes to program cannot use motions which implement the *RobotMotion* interface.

The *IServoMotion* allows the developer to change between motions in real time. This in turn allows the robot to perform corrections to the trajectory depending on external perturbations performed by the environment.

Trajectories that implement the *IServoMotion* interface only guarantee that the jerk of the robot is bounded, thus reducing the computation time necessary to transition from one motion to another. A higher bound can be placed on the computation time, with the lowest possible value of $20\ ms$ [29].

To understand the what a real time reaction to external perturbations is lets use an example adapted from the KUKA documentation [34]. Assume that the robot is performing a motion from P_1 to P_3 as seen in figures 3.6 and 3.5.

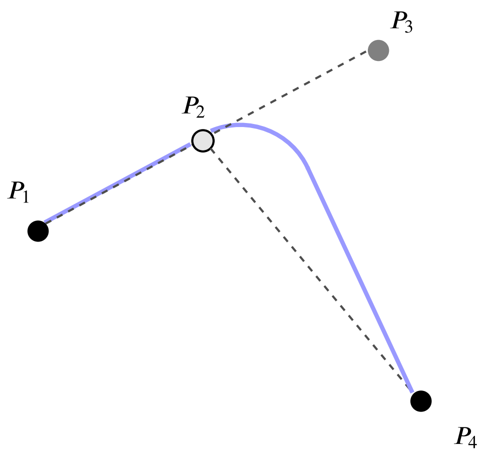


Figure 3.5: Example of real time smooth transition in trajectory from P_3 to P_4 due to an external disturbance in P_2 (adapted from [34])

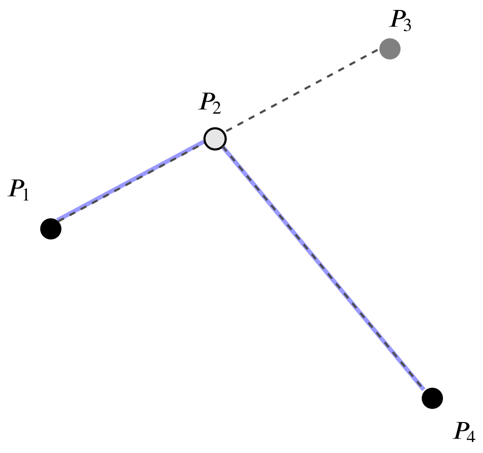


Figure 3.6: Example of real time hard transition in trajectory from P_3 to P_4 due to an external disturbance in P_2 (adapted from [34])

The trajectory interpolator defines the desired trajectory and the motion is ensued with a given control law. During the motion, in point P_2, due to some external perturbation it is necessary to correct the motion of the end effector to point P_4. By using the *IServoMotion* interface the new trajectory from point P_2 to point P_4 is generated with a smooth transition in the velocity and acceleration profile as seen in figure 3.5. When one uses motions which implement the *RobotMotion* this transition cannot happen in real time, so at point P_2 the velocity must be zero to start the new motion as seen in figure 3.6.

The last interface needed to perform a motion is the *IMotionControlMode* which encodes the control law used to follow the trajectory provided by the KUKA interpolator. There are two classes which implement this interface, the *ImpedanceMotionControlMode* which defines an impedance interface as described in section 3.1 and the *PositionControMode* which sets the stiffness of the motor joints to its maximum, thus following as closely as possible the desired trajectory.

When the developer commands the robot to move under a motion that implements the *IMotion* inferface they are required to define which control law should be used which results in the dependency shown in figure 3.4 between the *IMotion* and the *IMotionControlMode* interfaces.

3.1.4 OpenIGTLink

Since the set up that this thesis proposes requires the robot controller to communicate with an external workstation, a real time capable protocol is required between the two devices.

The OpenIGTLink protocol was developed by Tokuda et al. [35] to respond to the challenge posed by the increasing number of commercially available medical products. This increase in proprietary software creates additional complexity when integrating different devices into an already complex system.

By establishing a common language between all the medical devices OpenIGTLink became the standard communication protocol in the medical field [32].

To produce a portable protocol independent of the underlying architecture of each device a 58 byte header is attached to the beginning of each message.

The header is composed of six fields that encode all the necessary information for the receiver to decode the message, namely:

- **Version number:** Defines in 2 bytes the version of the protocol currently implemented in the sender hardware.
- **Data type:** Defines in 12 bytes which type of data is sent to the receiver, so that the message can be decoded once it is received thus making the protocol portable.
- **Device name:** Defines in 20 bytes the name of the sender device, in case one needs to differentiate between different devices.
- **Time stamp:** Defines in 8 bytes the current time in the sender clock since 00:00:0 1 January 1970 UTC. With this information it is possible to perform matching algorithms assuming that the internal differences between the different hardware clocks are properly accounted for.
- **Body size:** Defines in 8 bytes the number of bits pertaining to the current incoming message.
- **CRC:** Defines in 8 bytes the cyclic redundancy check for the body section of the message.

By defining a header in the beginning of each message the receiver can determine which message the incoming bit stream encodes and how to correctly decode it, thus eliminating any compatibility needs between the different hardware architectures.

To solve the problem of bit corruption the protocol relies on the CRC code mentioned earlier. Since all the security of the protocol relies of this 8 byte array it is important the the inner workings of the CRC algorithm are clear.

Although the real CRC algorithm does not work the way the following analogy presents it, the core idea behind the two are similar enough for the purpose of understanding how the algorithm works. In the following example the subscript $[]_x$ represents the *radix* of the number.

Assume that the message to send, called L, is composed of the bits $01100110\ 10010111$. One can translate said bits as $L_{10} = 26263$.

Now we can introduce the number P_{prime}, known both to the sender and receiver, that can be encoded by p bits, for example 11111011 where $p = 8$ and, in base 10, $P_{prime} = 251$. Now the trick used by the CRC algorithm requires that we add p bits with value zero to the initial L message as such:

$$S_2 = 01100110\ 10010111\ 00000000,\ S_{10} = 6723328$$

The sender of the message can now divide S by P_{prime}, obtaining the remainder $R = 42$. By performing the subtraction

$$Dif = P_{prime} - R = 209$$

the sender can now replace the bits that encode Dif for the p bits with value zero in S, obtaining the new $S^{'}$ number which is divisible by P_{prime}. For this example we obtain

$$S^{'}{}_2 = 01100110\ 10010111\ 11010001,\ S^{'}{}_{10} = 6723537$$

Since the receiver knows P_{prime}, he can divide the received decoded number $S^{'}$ by P_{prime} and check if the remainder is zero. If that is the case the receiver can now retrieve the bits that encode the original L number by simply selecting the first $n - p$ bits as such:

$$L = (01100110\ 10010111)\ \ (P_{prime} - R) = (11010000)$$

In case the remainder of the division of $S^{'}$ by P_{prime} is different from zero then some bit encoded in the message has been corrupted.

One detail that the reader might have noticed is that if the corruption of the $S^{'}$ number occurs in such a way that the resulting number is also divisible by P_{prime} then the CRC algorithm will not capture the message distortion. Due to this fact one can increase the size of P_{prime} so that the likelihood of such corruption appearing becomes negligible.

Another important distinguishing characteristic of the protocol is the large array of possible messages that can be sent using the protocol, enlarging the possible application that can use it. Currently there are 20 types of messages defined, and some of the most used are:

1. **Transform message:** Message which encodes a translation and rotation in the form of a 4×4 matrix():

$$T = \begin{pmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{22} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R & p \\ 0 & 1 \end{pmatrix}$$

2. **Position message:** Message which serves the same purpose as the transform message, namely position and orientation through a a 3D Cartesian vector for the position and a quaternion vector relating to the orientation information.

3. **Image message:** Message with 2D or 3D volumetric information of an image.

4. **Capability message:** Message with a list of the types of messages which the sender can decode.

5. **Status message:** Message pertaining to the status of the sender device.

6. **Query mechanism:** Mechanism used to tackle synchronization problems, by which the sender requests one given message, or a stream of messages, and can order the receiver to stop sending said stream.

7. **String message:** Message containing string information. This type of message is typically used when using xml files to configure a certain piece of software.

Given all the previous characteristics it quickly becomes clear why the OpenIGTLink protocol is chosen for the communication between LBR Med and the main workstation.

As introduced in the original paper [35], the protocol is only implemented in the C++ language, thus making the direct use of it incompatible with the JVM inside the robot controller.

Given the previous restriction two hypothesis emerge, as proposed by Tauscher et al. [32], either develop the protocol in JAVA or use the Java Native Interface (JNI) which allows the JVM to call directly native code, i.e. the C++ implementation of the protocol.

The benefits of a full JAVA implementation is the control over the source code and therefore the quick prototyping velocity and possible workarounds to any future problem. The downgrade of this path it that only partial implementations of the protocol exist and any code developed is dependent on which libraries already exist in the JVM.

The benefits of using the JNI to call the library is that one can use the C++ implementation of the protocol, reducing the initial effort required to use the protocol and because one is using the C++ implementation, it would possibly be more optimized, since a team of specialized developers continually upgrade and develop said implementation.

The downgrade of this path lies in the complexity of using the JNI and a slower prototyping speed because one would need to change between the C++ and JAVA language.

Considering the two possible paths, the full JAVA implementation quickly became the better solution, given that working with the JNI would induce direct memory manipulation inside the robot controller, defeating the purpose of the JAVA language with its safe architecture.

In 2008 Andre Charles Legendre [2] began to code the protocol in JAVA, designing the structure of the protocol and the different data types.

In his work he defined three relevant classes that the protocol requires to function:

- **ByteArray:** Because Java was developed in part to tackle the complexity brought forth by the world wide web, some assumptions had to be made, namely increase the security of the language by eliminating memory manipulation using pointers (typical constructs found in C and C++) and standardize different primitive data types, such as the int our double, which instead of being encoded into different byte arrays, as in C or C++, they are defined with a fixed size, namely 16 bytes for the primitive int class and 64 bytes for the double class. This fact turns any byte manipulation extremely difficult in Java, so the Byte Array class is a helper class, which allows one to codify primitive variables with different encoding styles, as needed by the developer.

- **Header:** The header is a class that provides fast methods to manipulate specifically the 58 bytes pertaining to the header attached to each message.

- **OpenIGTMessage:** The OpenIGTMessage is an abstract class, meaning that it cannot be instantiated as an object and can be extended by other classes. This class serves as the base for any data type class, and defines the general common behavior of any data type that is implemented in the protocol.

[2]Public information from https://kalima.io/our-team

Although the message encoding is well structured, the work done by André and latter by the WIP lab [32], relies on a complex socket structure which removes flexibility to future uses.

To make the protocol easy to use, and facilitate the jump from C++ to Java for future developers, the code was created as similar as possible to the original C++ implementation.

The final design for the socket logic of the protocol relies on two classes, shown in figure 3.7.

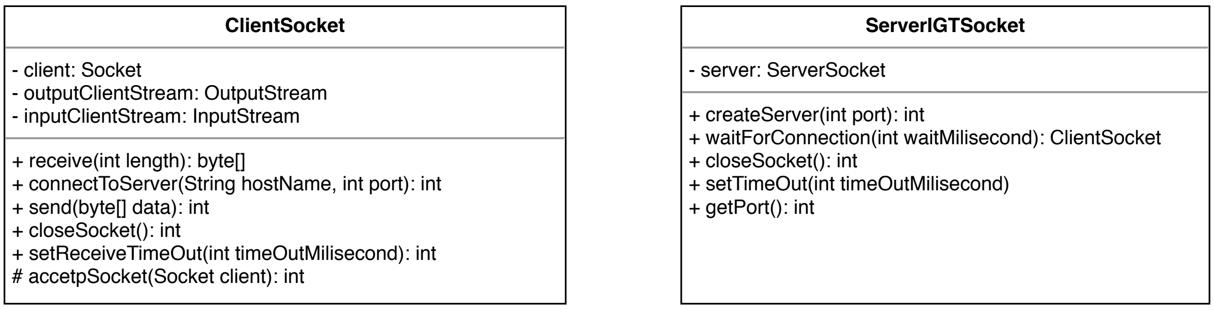

ClientSocket
- client: Socket
- outputClientStream: OutputStream
- inputClientStream: InputStream
+ receive(int length): byte[]
+ connectToServer(String hostName, int port): int
+ send(byte[] data): int
+ closeSocket(): int
+ setReceiveTimeOut(int timeOutMilisecond): int
# accetpSocket(Socket client): int
ServerIGTSocket
- server: ServerSocket
+ createServer(int port): int
+ waitForConnection(int waitMilisecond): ClientSocket
+ closeSocket(): int
+ setTimeOut(int timeOutMilisecond)
+ getPort(): int

Figure 3.7: Class diagram of the simple socket logic used by the Java implementation of the OpenIGTLink protocol.

Once a ServerIGTSocket class is instantiated as an object and bound to a port one can listen for an indefinite amount of time waiting for connections, or a specific waiting time can be defined. If a connection is established then the ServerIGTSocket object return an instance of a ClientSocket and communication can be established.

Most of the message types were coded from scratch to guarantee that the Java implementation and the C++ implementation of the protocol are compatible with each other.

With this simple socket logic it is possible to create numerous applications. As an example the following code shows a connection between a server and a client, seen from the client side.

```java
String localhost = "localhost";
int port = 18944;
// Client creation
ClientSocket client = new ClientSocket();
int result = client.connectToServer(localhost, port);
if (result == 0) {
   System.out.println("Could not create the client bound to the localhost host on port 18944");
   return;
}
// Start sending messages
TransformMessage transf = new TransformMessage("Example");
double[][] matrix = { { 1, 0.0, 0.0, 0.0 }, { 0.0, 1.0, 0.0, 0.0 }, { 0.0, 0.0, 1.0, 0.0 },{
    0.0, 0.0, 0.0, 1.0 } };
transf.setMatrix(matrix);
int resultOfPacking = transf.PackBody();
if (resultOfPacking == 1) {
   client.send(transf.getBytes());
}
client.closeSocket();
```

The simple logic used in the previous code shows that the Java implementation of the OpenIGTLink is flexible and simple to use.

Another important characteristic of the implementation lies in the simplicity to create a new message types. The developer only needs to extend the base class *OpenIGTMessage* and define the codification type and organization of the bytes that compose the new message.

3.1.5 Observer architecture

To develop an architecture to control the robot it is necessary to follow certain Java guidelines do produce a well designed piece of code. Three considerations where strictly followed while developing the logic of the application:

1. **Modular:** To develop an architecture which can integrate future states of the robot without the need to redesign from scratch the core components of the code.
2. **Fast:** The change from one state to another must be fast, given that the application must be classified as soft real time capable.
3. **Safe:** The architecture must facilitate the use of the safety features of the LBR controller.

To be able to reconstruct ultrasound images as volumes and receive feedback from the robot during the surgery it is necessary to communicate with the external workstation. To separate the communication process from the motion control the architecture shown in figure 3.8 was developed.

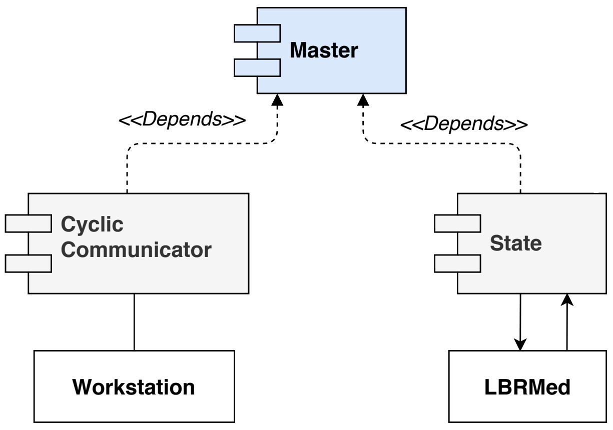


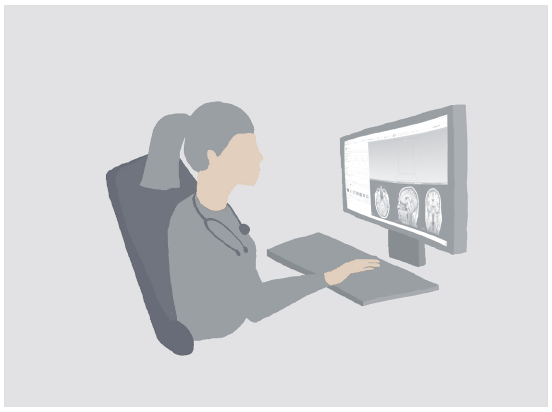
Figure 3.8: Component Diagram of the classes that control the internal responses of the robot.

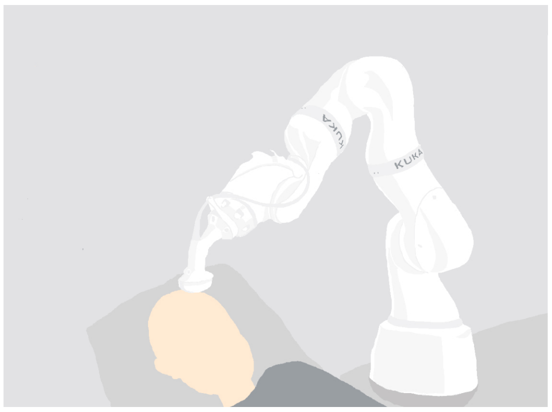
In this set up the *Master* is the class which instantiates the necessary variables so that the application can function and it also controls the transition from one *State* to the other. Another important characteristic is the modularity between the *State* classes which actually interact with the robot. Each *State* does not depend on the previous variables initiated in the previous states. This allows the developer to add new behaviour without the need to rewrite large sections of the underlying software.

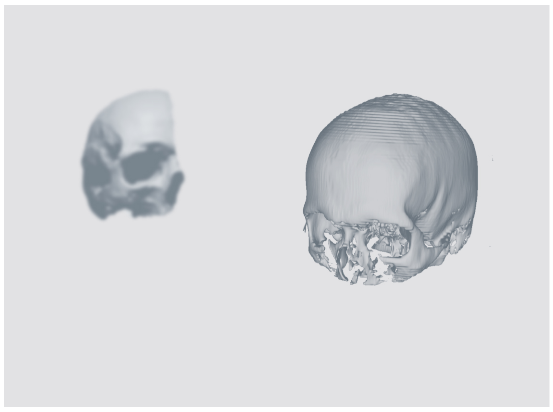
All the *States* share a common interface, called *StateSkeleton* which defines base methods required to interact with the *Master* class. The methods defined in the *StateSkeleton* are: the *initiate()* method defines the ESM safety functions, introduced in section 3.1.2, required to reduce the danger level of each *State*, depending on its function in the neurosurgery. The *run()* method is called once the *initiate()* method returns the handle to the calling thread.

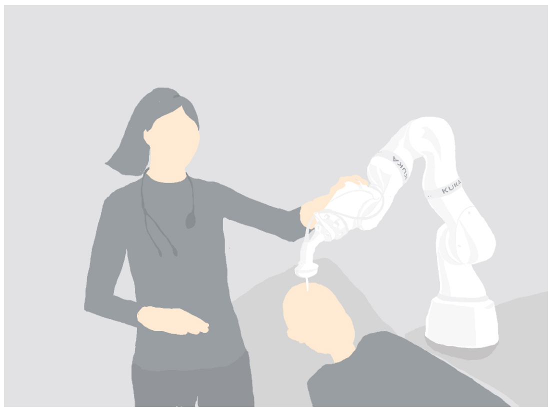
Once the *run()* returns the handle to the calling thread the *dispose()* method is called to dispose of all the classes created in that specific state. This architecture is based on the observer pattern [36].

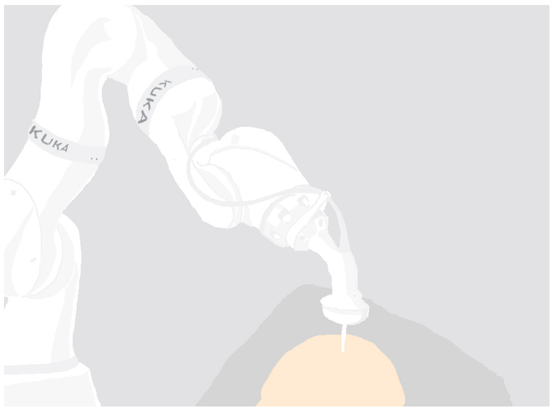
The actual states implemented in the LBR Med depend on the surgical workflow that underlies this thesis. The surgical workflow is shown in figure 3.9.


(a) path planning


(b) US scanning


(c) registration


(d) surgeon guidance


(e) surgical procedure

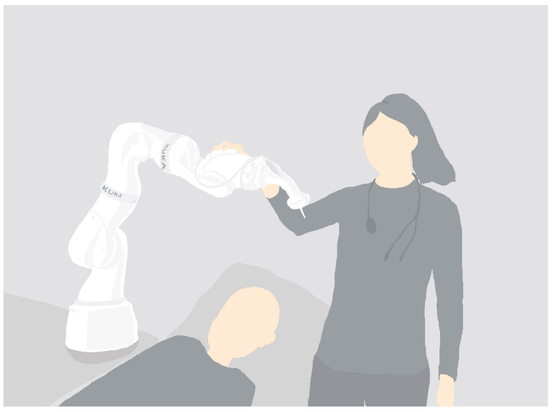

(f) stop robot motion

Figure 3.9: The figure shows all the steps for any surgical produced. (a) The neurosurgical team defines the target in the MRI volume and the path to reach said point, (b) The surgeon ultrasounds the patients skull and reconstructs the skull as a volumetric object from the 2D images, (c) The surgeons performs the registration between the MRI image space and the US image space, (d) The surgeon guides the needle under impedance control to the correct position, (e) The surgeon inserts the tool necessary to perform the surgical procedure inside the patients skull, (f) After the procedure has terminated the surgeon guides the robot into a safe position and blocks all movements.

Since not all the steps shown in 3.9 directly translate to robot *States*, the following enumeration shows the mapping between the actual stage in the surgical procedure and the internal robot *States*.

1. **Start up and verification:** The first state verifies the integrity of the LBR Med brakes and checks the encoders of all the seven joints of the robot.

2. **Ultrasound registration:** The second state allows the robot to track the US probe rigidly attached to the flange of the robot.

3. **Move to safe position:** Once the surgeon decides that the registration between the MRI-US space shows no significant difference between one another the robot moves under impedance control to a point positioned away from the patient and aligned with the desired trajectory.

4. **Impedance control:** The fourth state uses the impedance control of the LBR Med to follow the trajectory defined in the preoperative analysis.

5. **Termination:** Once the procedure is over the surgeon safely terminates the intervention and the robot is stopped.

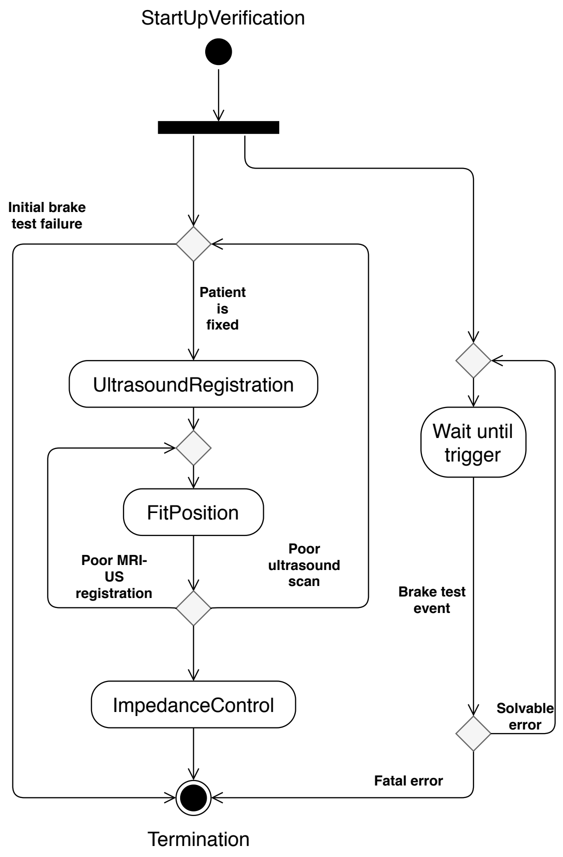


Figure 3.10: State diagram of the internal states of the robot.

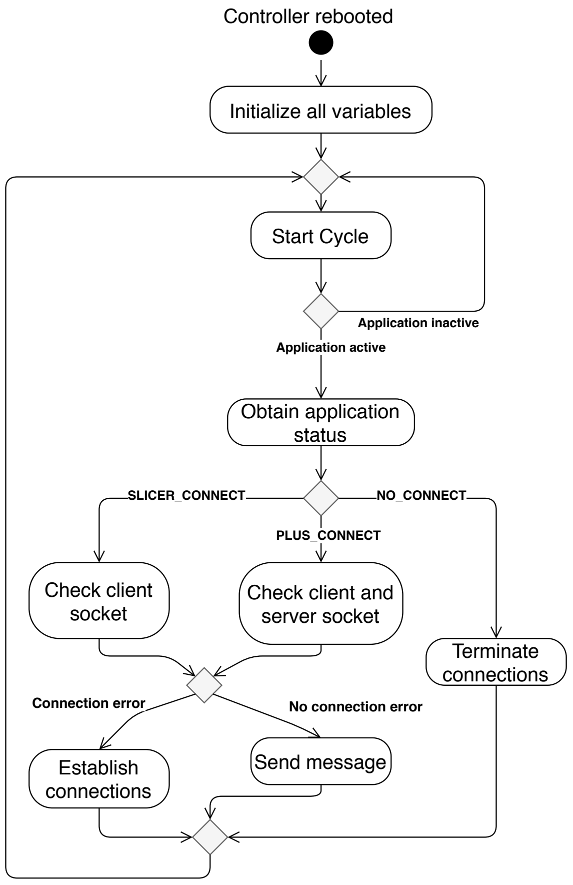


Figure 3.11: State diagram of the communication structure implemented in the robot

To understand how the previous robot *States* interact with each other one can use an activity diagram to show how the system evolves in time.

The activity diagram, seen in figure 3.10, represents the evolution of the robot *States* throughout the life of the application. Once the application is initiated the process begins. Two process are concurrently called once the safety of the robot is assured. The ultrasound scan begins and a listener waits for a brake test trigger in the event of a fault during the lifetime of the application. The rest of the logic followed in figure 3.10 is self explanatory.

The communication structure used by the *CyclicCommunicator*, can be seen in figure 3.11. The relationship between the *Master* class and the *CyclicCommunicator* class is established through an interface which allows the instance of the cyclic communicator to request the current communication status from the *Master* class and act accordingly.

3.1.6 Virtual impedance environment

In the previous projects developed at the SRL the impedance control law always relied on a impedance cone [27][26], that compensates external disturbances, thus correcting any orientation error and positioning error, as the surgeon guides the robot along the desired trajectory.

The approach in this work is slightly different, because the robot does not move once the surgical needle is inserted in the patient's skull, only holding its position when the actual surgery is taking place.

Two strategies were developed to follow the desired trajectory. An impedance cone that increases the stiffness imposed on the end effector as the surgeon approaches the target, and an impedance line that has a constant stiffness along the desired trajectory.

One might question the need for the impedance cone strategy since the impedance line strategy guarantees smaller errors for the same external perturbations along the desired trajectory. The reason for the exploration of this strategy is two fold: *a)* Since the physician guides the surgical needle into position, by performing a smooth increase on the stiffness the procedure feels natural which is important given that the technology is fairly recent and still faces scrutiny be the medical field, *b)* this thesis also aims to probe the full capabilities of the robot and although the final strategy chosen to perform the surgical procedure might not rely on the impedance cone, future work might do so.

The impedance line strategy uses the target frame *Commanded Frame* shown in figure 3.12 as equilibrium position of the robot. The LBR Med performs the required computation to mimic the second order system by itself, as described in section 3.1.

To force the robot to stop once the *Commanded Frame* is reached a class that allows the developer to update the stiffness in real time is required. Since the desired trajectory is linear the *SmartServoLIN* class has the *changeControlModeSettings()* method that allows the developer to change the stiffness of the robot in real time.

Since in a **shared controlled system**, it is the surgeon that guides the robot along the desired trajectory then the stiffness along the x axis is set to zero. With this simple trick the robot controller won't pull the end effector along the x axis by itself.

The stiffness matrix is set as 3.7:

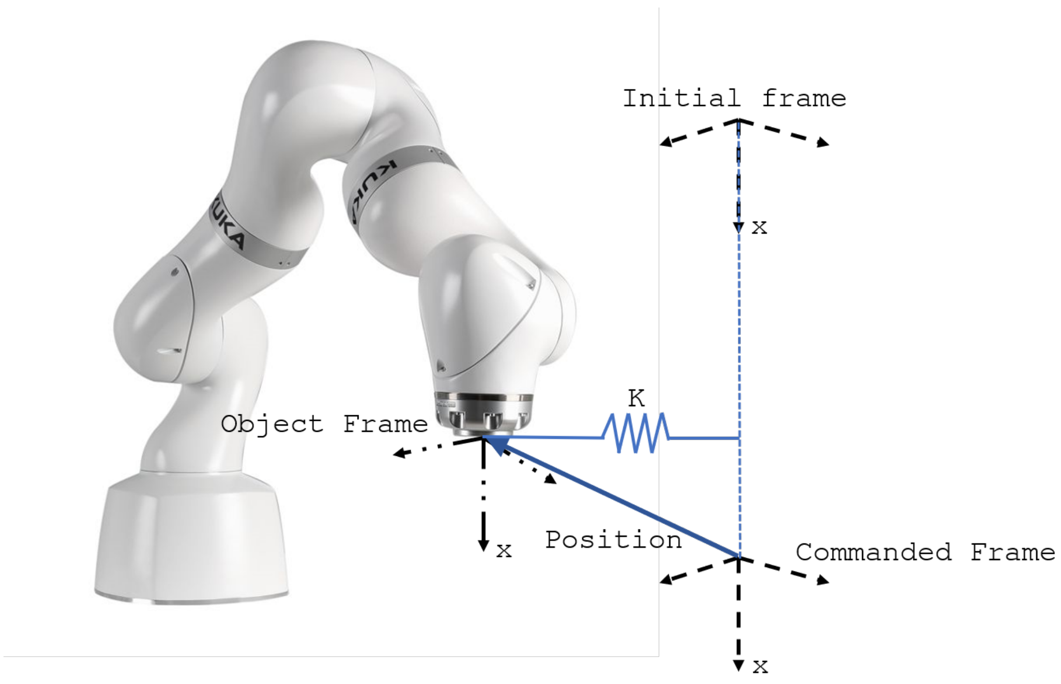


Figure 3.12: Objects used by the robot to create the impedance control environment.

$$K_t = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 5000 & 0 \\ 0 & 0 & 5000 \end{pmatrix} \tag{3.7}$$

By setting the k_{11} component in the stiffness matrix to zero one losses the asymptotic stability property of the controller introduced in section 3.1 because the K_t matrix is not positive definite.

The loss of stability only effects the x axis, because we do not wish for the end effector to move by itself to the target frame. Only in the y and z directions is the asymptotic stability required. The resulting system can be classified as a marginally stable system.

The second control strategy, the impedance cone, uses the same logic as the previous control strategy but updates the stiffness of the y and z directions according to equation 3.1.6.

$$k_{ii} = f(x) \tag{3.8}$$

where k_{ii} represents the stiffness value from the K_t matrix with $i = 2, 3$, and x represents the distance to the target frame.

Since the robot has a maximum allowable stiffness of $5000\ N/m$ [34], the function must respect the specifications introduced in equation 3.9.

$$f(x) \in [0, 5000] \tag{3.9}$$

A simple function that respects this specification, easy to compute, which in turn allows the cycle that updates the stiffness to have a smaller duration, is introduced in equation 3.10.

$$f(x) = \frac{5000}{|x| \times d + 1} \tag{3.10}$$

where d is a constant that controls the rate of change of the function.

Since any fast change in the stiffness will in turn produce large increases in the corresponding motor torques, the function must have small rates of change. The comparison between the function for different values of d is shown in figure 3.13.

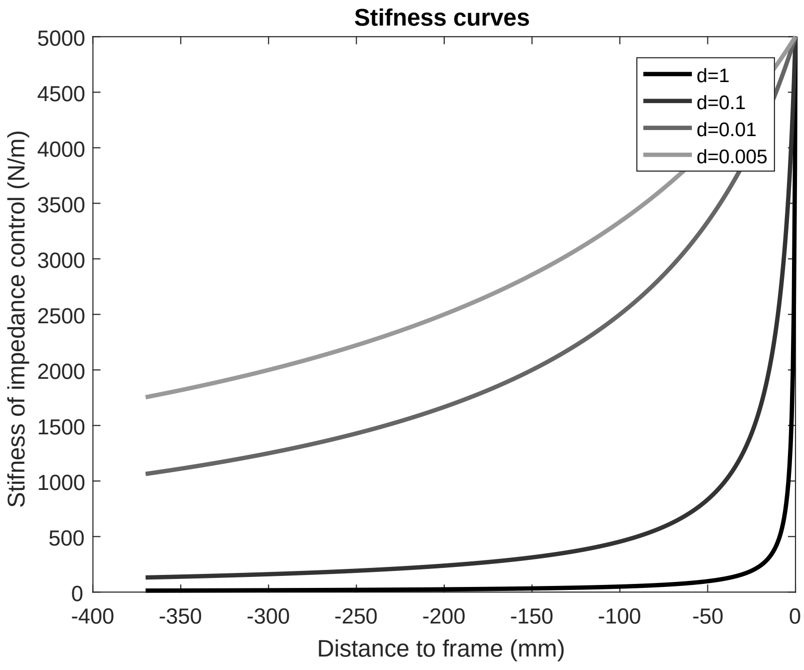


Figure 3.13: Values of the desired stiffness curve depending of the value of the constant d

The final stage of the impedance environment is the robot behaviour when the target is reached by the guidance of the surgeon.

When the *Commanded Frame* is reached the behaviour of the two strategies is equal.

As soon as the distance to the target frame in the x direction is positive the component k_{11} of the K_t matrix is increased in a linear fashion until it reaches the maximum value of $5000\ \frac{N}{m}$.

Since, in a static analysis, the force required to perturb the position of the end effector by more that $2\ mm$ is $10\ N$, as deduced in equation 3.11, then the robot should enter into position control after the stiffness in the x direction is set to $5000\ N/m$ so that the robot's position is fixed and robust against external perturbations.

$$f = m\cancelto{0}{\ddot{\Delta x}} + b\cancelto{0}{\dot{\Delta x}} + 5000 \times \cancelto{0.002}{\Delta x} = 10\ N \tag{3.11}$$

The transition into position control is rather complex, due to the inner workings of the KUKA library. The command used to perform position control in a given point in space is called *PositionHold*. This control scheme can only hold the current position of the robot.

This feature becomes a problem if the robot has some external force applied to it during the transition, either due to incorrect estimation of the tool's mass, or due to some unaccounted force applied on the tool, because the current position of the robot might not correspond to the targeted position to perform the surgical procedure.

To find the solution to this problem one can look at figure 3.14, which shows the current problem in two dimensions, but the same logic applies in three dimensions.

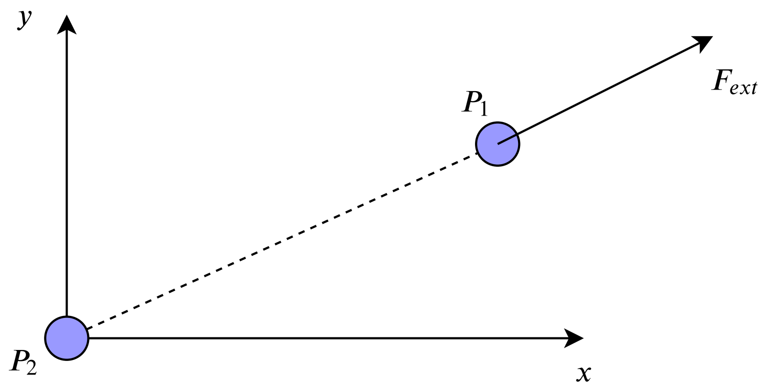


Figure 3.14: Displacement caused by unaccounted force F_{ext}, where P_1 is the real position of the robot and P_2 is the desired position. The dashed line represents the linear motion necessary to correct the position.

Since the unknown F_{ext} force is pushing the robot from the desired position P_2 to the unknown position P_1 the solution lies in performing a linear motion, from P_1, the current pose of the robot, to P_2 under position control.

Afterwards the robot can then transition to the *PositionHold* command, thus eliminating the incorrect pose problem due to unknown external forces imposed on the robot.

3.2 Tracking and ultrasound image fusion

Before tackling the complex problem of volumetric reconstruction of US images it is important that the principles of US imaging and the internal mechanics of a typical US diagnostics probe are completely understood, so that all assumptions made in this work are clear and their advantages, or lack thereof, be explicit.

Sound is one, among many ways, to propagate energy, and this propagation is characterized by certain physical laws. The first characteristic of sound is how the wave is affected by the medium in which it propagates.

Unlike electromagnetic waves, which propagate in vacuum, sound requires a medium to propagate, and the properties of the medium directly affect the characteristics of the sound waves passing through it.

One such characteristic is the speed of the sound wave, and each medium, or material, is defined by a unique speed of sound. The capacity of the medium to propagate sound is inversely proportional to the *acoustic impedance* of the material. The acoustic impedance depends on the density of the material, among other factors [37].

In table 3.1 several wave speeds of different biological materials are shown.

Table 3.1: Sound of speed under different mediums (from Jack et al. [37])

Tissue	Bone	Muscle	Blood	Kidney	Liver	Soft tissue average	Water	Fat	Air
s (m/s)	4080	1580	1570	1560	1550	1540	1480	1450	330

Since the average speed of sound in soft tissues is $1540\ m/s$, by convention all ultrasound devices assume this as the true value of the speed of sound [37].

There are two important physical effects in sound mechanics, *reflection* and *acoustic impedance*. *Reflection* of sound waves is the physical phenomena actually used by the ultrasound probe to create the US images. Reflection in this context is defined as the redirection of a portion of the sound wave due to a change in the value of the *acoustic impedance* when passing through an interface between two different materials. The greater the difference in acoustic impedance the larger the redirected portion of the sound wave.

Typical diagnostic US probes are composed of piezoelectric crystals which can create vibrations when excited by an electrical current, or produce an electric current when excited by vibrations. By emitting pulsed waves for short durations by exciting the piezoelectric crystals, and afterwards measure the reflected sound waves one can determine distances to objects by multiplying the speed of the sound wave by the time taken from the tip of the probe to the reflected wave arrival [37]. As one can assume, by decreasing the pulse duration's the better the resolution of the measured distances.

The first US measurements were obtained in 1960 using the previous introduced concepts, where the actual probe only contained one piezoelectric cristal. This technique is called the A-mode ultrasound and one such result is shown in figure 3.15.

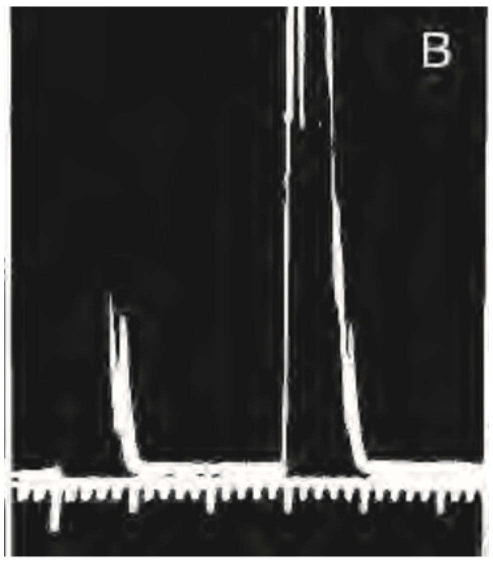


Figure 3.15: Measured US signal showing the size of a cystic nodule (from Jack et al. [37])

Since the A-mode technique only allowed to estimate unidimensional information about the targeted tissues it was soon followed by what is currently used by diagnostic tools, called the B-mode scan.

The B-mode scan is based on the same principle as the A-mode, with a larger number of piezoelectric crystals used to measure electric signals as seen in figure 3.16.

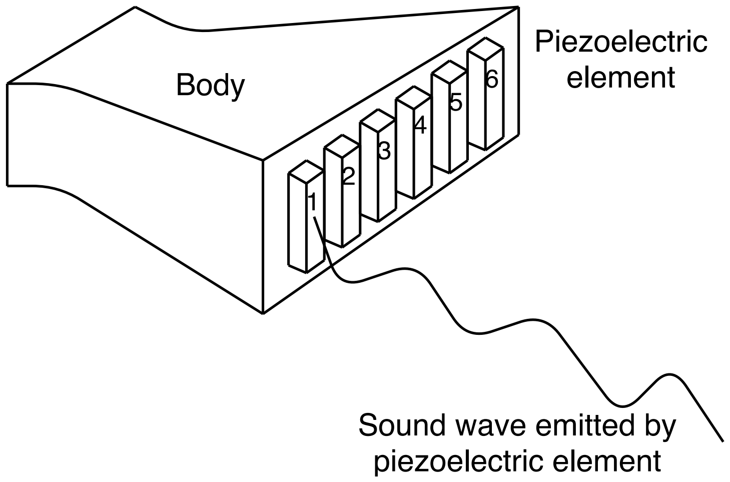


Figure 3.16: The figure shows a typical probe, that fires sequentially the piezoelectric elements and creates a B-mode US image from the measured sequential signals.

Any probe is characterized by the frequency at which it works, in other words, the sequence of piezoelectric excitation's from the first element until the last element of the transducer.

3.2.1 Plus Toolkit

As introduced in section 2.2.3, the Public software Library for Ultrasound (PLUS) toolkit is an open source project to obtain, process and track US images with tracking tools such as the NDI Polaris device.

The toolkit uses an versatile underlying architecture which allows complex algorithms written in C/C++ to compensate for time variations on the incoming OpenIGTLink messages and perform transformations of the incoming image messages. The architecture of the toolkit is shown in figure 3.17.

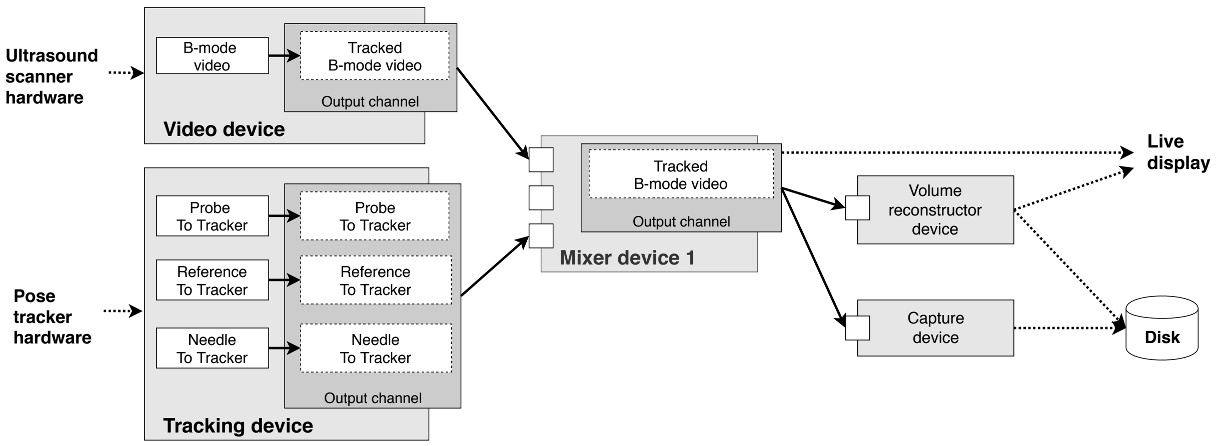


Figure 3.17: Interaction between hardware and the software components used by the plus library (adapted from Lasso et al. [19])

To use the Plus library, only a single video device is allowed to stream images from the hardware, but multiple tracking devices are allowed, so that different components of any set up can be tracked in real time [19]. In figure 3.17 another important component of the plus library is the mixer device, which performs different transformations on the incoming poses of the tracked objects in the OR.

The volume reconstructor component uses the tracked images, with their corresponding transforms, to create a volume where the intensity of the voxels is extrapolated from the intensity of the transformed image pixels.

This volume can be further processed to isolate the required surface to perform the registration between the MRI and the US space.

The library works under two underlying assumptions, and they are:

a) The image is composed of A-mode signals which are not obtained at the same instant, therefore the image only contains information from the same slice of soft tissue if the probe is static, or the movement of the probe is small compared to the frequency of the transducer. The image processed by the library is assumed to contain information from a single slice of the targeted tissue.

b) To correct the delay between the stream of transformations obtained through the tracking system and the stream of images obtained through the ultrasound system only a constant offset is required. This assumption can prove to be damaging if the stability of the connection between the ultrasound system and the main workstation becomes unstable, thus introducing larger delays.

Given the complex mechanics associated with any system which tackles the topic of US image manipulation, and volumetric data extraction, corresponding tools should exist that facilitate the calibrations required to estimate the offset mentioned in b), static transformations that correct distortions on the US images and other miscellaneous deformations.

One advantage associated with Plus is the large array of algorithms, shown in figure 3.18, which reduce the time taken to calibrate the set up when its components are changed.

To actually use the library capabilities the developer is faced with two choices, either integrate the C++ code into his own system, and performing the required changes when any incompatibility emerge between the library and the intended final application or use the PLUS applications introduced in figure 3.18.

The **fCal** application allows one to use the full capabilities of the library, but its intended use lies solely in US image manipulation, whereas the **PLUS Server** is intended to be used as an interface between the hardware and the visualization platform.

Although this thesis does not tackle directly the problem of US-MRI registration, future iterations of this project require a visualization API with flexible processing prowess, thus the capabilities of the **fCal** are used when necessary, but the main focus of this work lies with the **PLUS Server** application.

The last important component that interacts with the library is the actual hardware used in the set up, The developer can create a XML file where all the different devices are specified so that the Plus server can establish the proper connections to the IP and port configured for each specific device.

The XML file is called the configuration file and in appendix A it's explained in detail how to create one with all the intermediary steps.

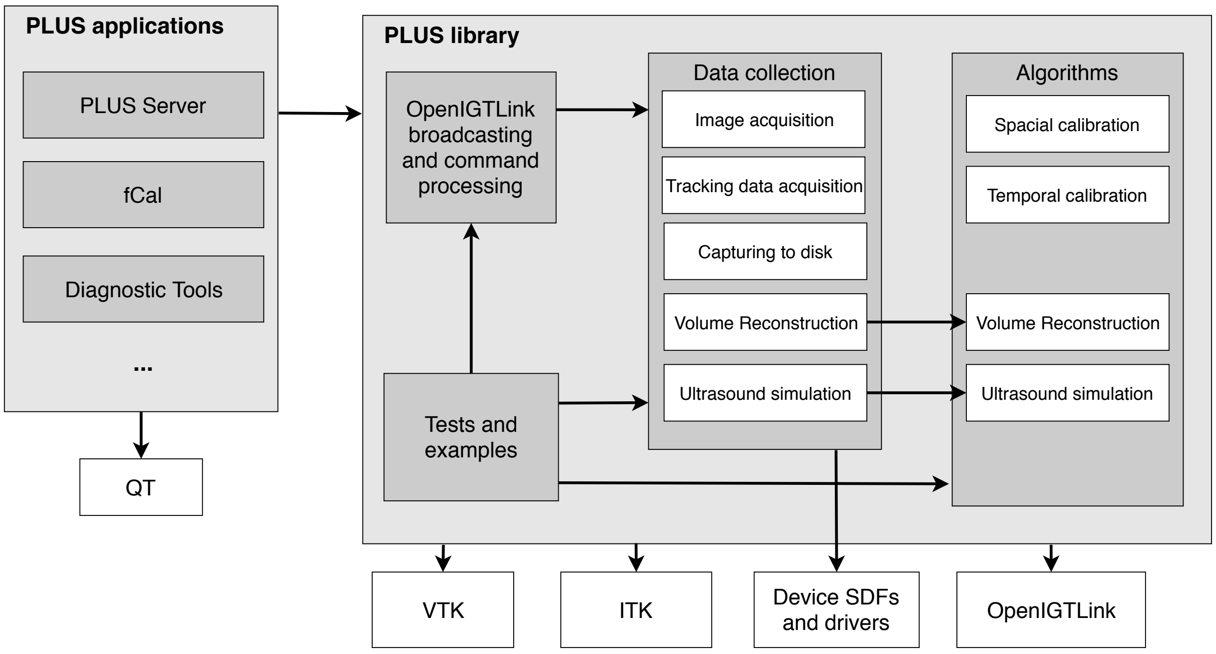


Figure 3.18: Software available in the plus library, to use it either as a standalone application or a data collector in a higher level navigation platform

In summary XML files are formatted as element trees. A XML tree starts at the root element, for in the plus library corresponds to the PlusConfiguration element, and branches from the root the child elements, which can in turn have children of their own. The main elements of a plus XML file are:

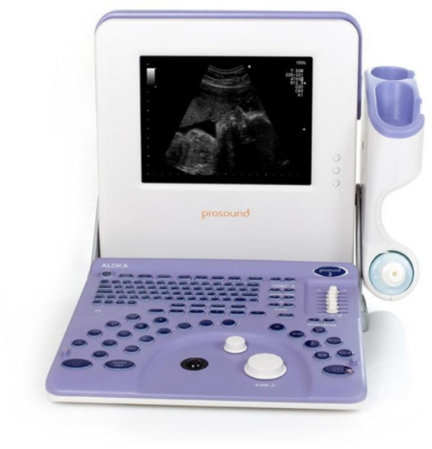
- **DataCollection** Element used to retrieve streams of data from several devices. This element contains two types of child elements inside it, and they are:
  1. **DeviceSet** Element that contains a brief summary about the specific configuration of the current file.
  2. **Device** Element that defines one device that has a server which connects to Plus.
- **CoordinateDefinitions** Element used to define fixed transformations between frames, normally these transformations are the result of calibrations between different devices.
- **PlusOpenIGTLinkServer** The server created by the Plus Server Laucher application which waits until a connection from a client is established and begins the information stream to the visualization platform, or for further data processing.

At the moment most commercially available devices in the market, either ultrasound systems, such as the Ultrasonix, Telmed, Interson, Capistrano and Philips, or tracking devices, such as Ascinsion, Ultrasonix SonixGPS, Atracsys, OptiTrack, NDI Vega, Polaris among others, are preconfigured in the library.

With the large array of commercially available devices already integrated into the plus library it greatly simplifies future work when integrating the set up proposed in this thesis in different OR's with different devices.

3.2.2 Analogic Ultrasound and image capture

The device used to obtain the US images and test the viability of the registration between image space and physical space is the prosound compact unit, seen in figure 3.19. This unit creates b-mode images as described in section 3.2.



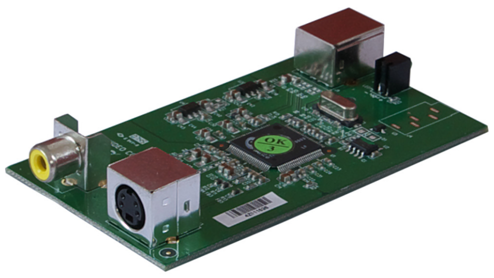


Figure 3.20: DFG/USB2propcb

Figure 3.19: Prosound unit

Although the prosound unit is a user friendly device with several settings to process the US images depending on the types of tissue being observed, the actual US image cannot be obtained through the typical communication protocols. To establish the connection to the main workstation a frame grabber is required to serve as an interface between the two devices.

The frame grabber chosen is the DFG/USB2propcb frame grabber seen in figure 3.20, which is one of the preconfigured frame grabbers in the Plus library. The frame grabber has a maximum frame rate of 30 fps with a maximum resolution of 768×576 pixels.

3.2.3 Slicer 3D

3D Slicer is a open source software designed for medical image processing and visualization purposes. The software is continually improved by teams of professional engineers in close collaborations with physicians all around the globe[2].

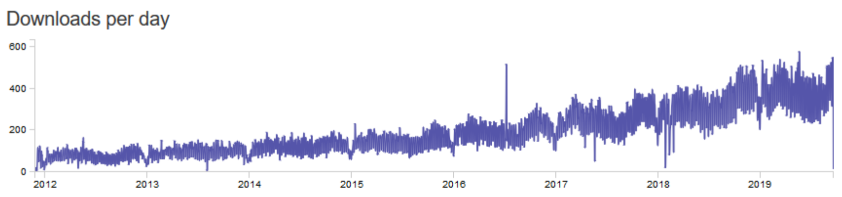


Figure 3.21: Daily downloads of Slicer since its release to the market, showing a consistent growth pattern since its birth.

The software is actively used in neurosurgical planning, guidance and follow up. Acording to the public available information online, Slicer has a total of 562202 downloads and the number of downloads per day is steadily increasing as seen in figure 3.21.

Given that one of the goals of this thesis lies in the developing a set up that could be used by neurosurgeons, then the choice of using slicer as a visualization tool is the logical addition to the set up. The Slicer API, shown in figure 3.22, has a visualization window and an integrated GUI which can be programmed either in python or C++, allowing one to develop applications for specific purposes.

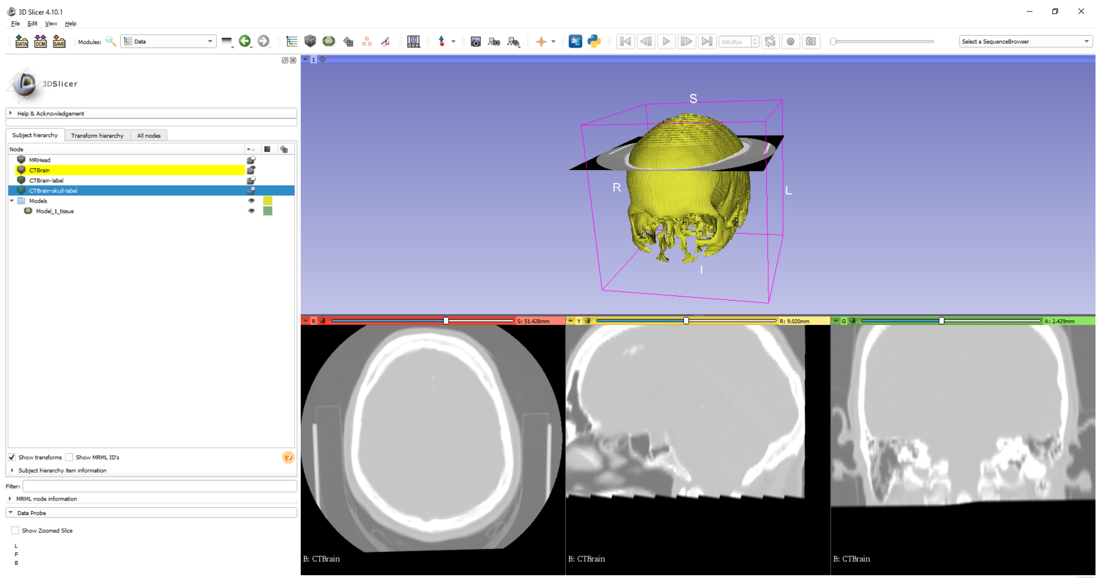


Figure 3.22: The 3D Slicer interface, showing the visualization area (top right), the GUI area (on the left) and the sagital, coronal and horizontal plane of the processed volume (on the bottom right).

Another important consideration lies in the amount of existing extensions already implemented in Slicer. Currently several extensions already exist to process volumetric information, or create models from anatomical data among others. The second advantage of this visualization software component lies in the already existing extensions to interact with the **PLUS Server** application.

The extension that serves as an interface between the **PLUS Server** and the Slicer API is called Plus Remote. It allows the surgical team, or developer, to start the procedure, connect all the devices to the server which processes the information and perform the following actions:

- *Start recording:* Depending on the development stage of the process, the user can record the ultrasound data, so that further experiments and analysis can be performed on the recorded data.
- *Offline reconstruction:* With recorded data, the user can reconstruct the ultrasound images into a volume, thus exploring the capabilities of the Slicer API whilst dealing with the US imaging modality.
- *Start scout scan:* Before creating the surface to perform the registration between the MRI space and the US space it is necessary to scout the patient with the ultrasound probe, with a smaller resolution, usually 3 mm, so that a region of interest (ROI) in the physical space can be defined.
- *Start live reconstruction:* Once the scout scan has defined the ROI one can use the full capabilities

of the plus library to recreate the information contained in the US images with the best possible resolution.

3.2.4 Integration of all the introduced devices

Previous sections hinted how all the devices that compose the set up interchange information between each other. This section formally defines the architecture developed in this work.

Some physical definitions are required so that the reader understands which data is obtained and processed by which devices. The main coordinate systems used in this work are shown in figure 3.23.

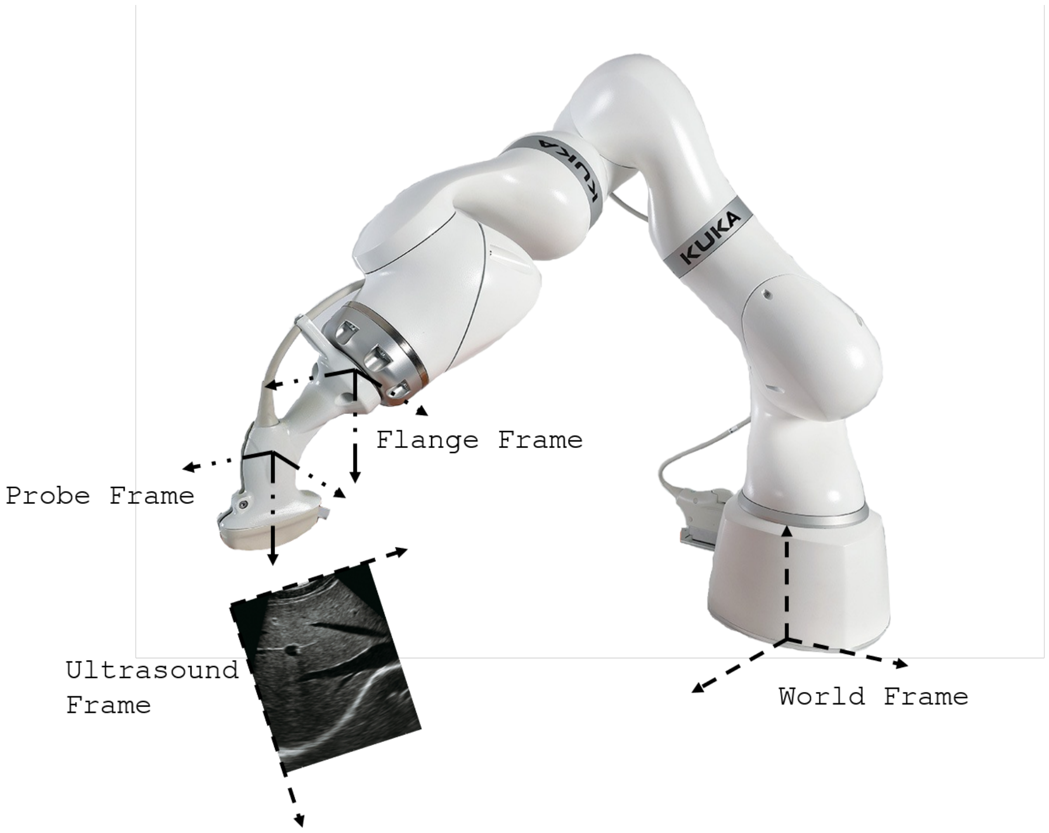


Figure 3.23: Frames of reference used by the US and robot fusion.

The LBR Med has an embedded calibration between the flange frame and the world frame. To obtain the fixed transformation between the probe frame and the flange frame one can either use some calibration method or one can directly calibrate the transformation between the ultrasound frame and the flange frame, thus implicitly both the ultrasound to probe and the probe to flange transformations are contained in this transformation as seen in equation 3.12.

$$T^{Image}_{World} = T^{Flange}_{World}.T^{Probe}_{Flange}.T^{Image}_{Probe} = T^{Flange}_{World}.T^{Image}_{Flange} \tag{3.12}$$

With all the frames in the system defined it is now possible to understand how one can fuse the data streams from all the previous described devices. The two communication stages of this set up are: a) are the volume reconstruction stage, where US images are retrieved from the probe and the position information is retrieved from the robot; b) the data streaming of the position surgical needle attached to the robot, namely the needle tip, inside the patient's skull.

The first communication stage is characterized by its star shape communication where all the data streaming is controled by the central node, the Plus library, as seen in figure 3.24.

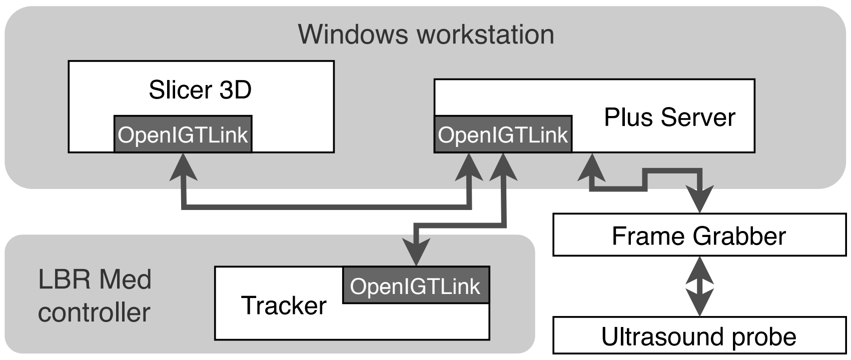


Figure 3.24: Interaction of the devices used in this thesis.

This set up allows one to create some modularity between all the devices, since the visualization platform can be substituted without effecting how the robot and the probe communicate with the Plus library, thus allowing for future replacement without the need for new software.

In this stage some underlying assumptions are made about how the hardware works, which need to be addressed so that the final errors produced by the set up can be properly addressed and the contribution of each assumption to the total error can be quantified and corrected if necessary.

The time delay assumption imposes that between the intervening data streams only exist a constant delay. This assumption ignores that the ultrasound captures a vertical line of pixels instead of an complete image at a constant frequency.

If the velocity of the probe is comparable to the frequency of the transducer, then the final reconstructed volume will show deformations associated to this relative movement.

The second assumption is that by using an affine transformation one can perform all the required geometric calibrations between all the intervening devices. The physical assumption does not entail as many problems as the time delay assumption.

The second communication stage does not rely on the Plus library and the communication is established between the visualization software and the tracker. This stage allows the surgeon the see in real time where the needle is placed and focus his attention on the surgical process.

This stage is not necessary to the procedure and it should be included if requested by the surgical team.

Chapter 4

Experiments and Results

4.1 Problem Description

As introduced in the previous chapter, several components constitute the final set up used for the surgical procedure so their respective interactions must be explored and properly studied. The three sections of this chapter are divided according to:

- **Section 4.2:** The performance of the Java implementation of OpenIGTLink is tested in the context of this thesis. Key parameters are reported as defined by [35] and the limitations of the current measurments are reported and possible solutions are presented.
- **Section 4.3:** The performance of the control strategies introduced in the previous chapter, and their validity according to the underlying control law used by the LBR Med, are tested. Since the actual control law used by the LBR is unknown, experiments are also designed to understand said control law. This section mainly focuses on experiments to explore the capabilities of the LBR Med and prove the validity of the control architecture.
- **Section 4.4:** The validity of the calibration algorithms used by the Plus library, namely the temporal calibration and the spacial calibration, are studied and the errors are analyzed.

4.2 OpenIGTLink in Java

4.2.1 Latency measurement

The latency of a protocol is defined in this work as it was introduced by Tokuda et al. [35], where two instants are measured, t_0, when the message is created and t_f, when the message is decoded by the receiver.

In figure 4.1 the two instants are visualized.

The most important concept that needs to be understood in figure 4.1, is the fact that both workstation 1 and workstation 2 measure the time in relation to the same clock. This becomes a problem because

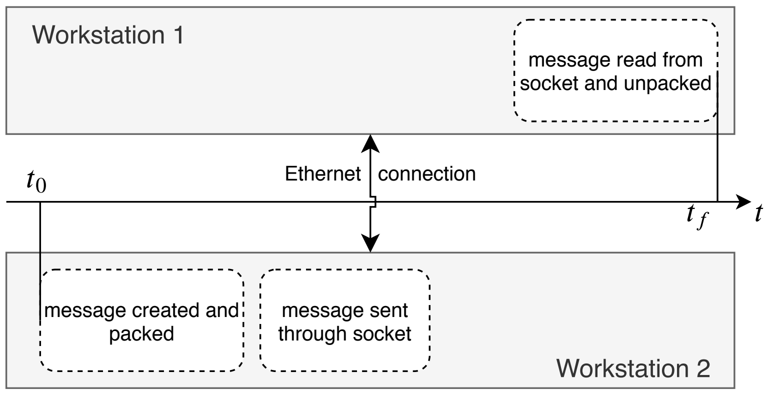

Workstation 1
message read from socket and unpacked
t_0
Ethernet connection
t_f
t
message created and packed
message sent through socket
Workstation 2

Figure 4.1: Logic used to measure the latency of the protocol. With this set up there are two workstations, connected through an Ethernet connection and all measurements refer to the same time frame

the internal clocks of both workstations measure time differently due to the mismatch between clock speeds[38]. This in turn means that t_0 and t_f are measured according to different clocks.

To avoid this problem the communication between the client and the server is established using a single workstation. This in turn means that any time measurement will be in reference to the same clock.

The latency results can be seen in table 4.1.

Table 4.1: Latency of the OpenIGTLink Java implementation

frame rate (fps)	Method used	latency
199044	System.nanotime()	0.00452 ms

Because the results reported by Tokuda et al. [35] only consider the worst case scenario, which is 16 concurrent channels, they should not be compared to the performance obtained in table 4.1.

4.2.2 Jitter measurment

Jitter is defined according to **RFC 4689** as the absolute difference between the delay measurement of two adjacent packets of data, defined as in equation 4.1.

$$J = |D_i - D_{i+1}| \tag{4.1}$$

where J represents the instantaneous jitter, D_i is the timestamp of message i and D_{i+1} is the timestamp of message $i+1$.

Because the Java version of the protocol is only used by the LBR Med, the jitter measurements were only obtained between the cyclic communicator implemented in the LBR controller and the main workstation.

Since the cyclic communicator architecture sends messages at constant intervals the jitter in the incoming messages is measured by sending messages at constant intervals, Δt, from the robot's controller and measuring the incoming messages in the main workstation.

The results are shown in figure 4.2.

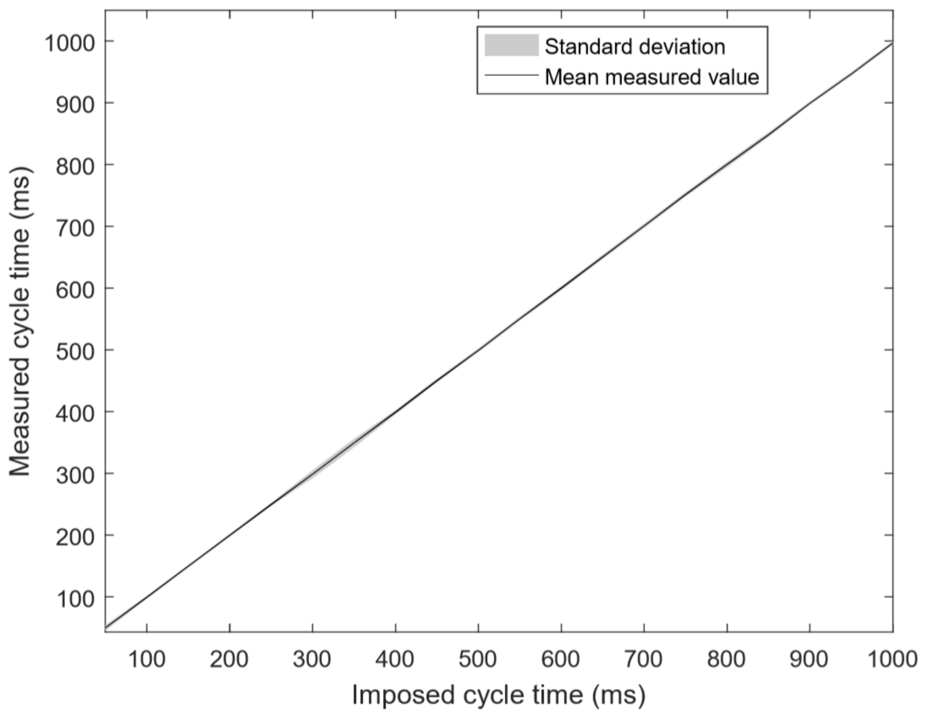


Figure 4.2: Jitter of the connection for different cycle times

4.2.3 Discussion of OpenIGTLink in Java

The Java implementation of the OpenIGTLink protocol meets the requirements necessary to be used in a soft real time application, due to its low latency and low variance in the jitter values.

Although the experiments that Tokuda designed to test the C++ implementation of the protocol are different in nature from the experiments designed in this work, if the latency is compared between the two implementations we can see that Java does not add significant overhead to the protocol in comparison to the latency reported for the C++ implementation.

If one wished to test the Java implementation outside the scope of this project, they would need to compensate the clock drift between the two workstations. To do this, two protocols exist that can correct said clock mismatch. The Network Time Protocol (NTP), introduced by Mills [38], which is typically used over the internet, and it can reasonably maintain the clock drift up to $1\ ms$. The second protocol is the Precision Time Protocol (PTP), explored by Scheiterer et al. [39], which can guarantee synchronizations down to $100\ ns$. The OpenIGTLink protocol, developed in C/C++ used the PTP protocol to reduce the time drift between two workstations thus guarantying the validity of the latency measurements.

The jitter results indicate that a lower bound should be placed on the cycle time of the cyclic communicator. As the cycle time decreases the ratio $jitter/\Delta t$ increases because the variance is close to constant, as seen in figure 4.2, thus the interpolator employed by Plus can show poor performance while trying to process the incoming transform messages.

4.3 Analysis of the proposed control strategies

4.3.1 Analysis the internal controller of the LBR Med

As introduced in section 3.1 there are three possible control strategies implemented in the LBR Med controller. With simple experiments it is possible to probe which control law is implemented inside the robot controller due to the different behaviours they produce. The different controllers are:

1. *Impedance controller:* The impedance control law is well suited for low stiffness a damping values. For high values of these parameters stability problems appear. Around singularities the behaviour of the robot will be smooth but the mapping of the stiffness matrix will be distorted.

2. *Stiffness controller:* The impedance law can map both high a low stiffness values but the mapping is only valid locally. If a perturbation force affects the end effector, the mapping of stiffness and damping matrices will be distorted because the equation assumes that the Jacobian is static and equal to the Jacobian in the equilibrium position.

3. *Impedance controller enhanced by local stiffness control:* The impedance law uses the mathematical background developed for the stiffness controller, applying a conservative congruent transformation to the desired stiffness matrix, and enhances the performance of the stiffness controller with the impedance controller when the perturbation force acting on the robot entails a substantial difference between the real position of the end effector and the equilibrium point.

With the previous different behaviours in mind, two experiments, similar in nature to the ones proposed by Albu-Schaffer and Hirzinger [30], were developed to test the actual behaviour of the LBR Med.

Table 4.2: Commanded values for the desired stiffness matrix.

x	y	z	roll	pitch	yaw
5000 N/m	3000 N/m	5000 N/m	300 Nm/rad	300 Nm/rad	300 Nm/rad

Table 4.3: Equilibrium position of the LBR Med in joint angles.

A_1	A_2	A_3	A_4	A_5	A_6	A_7
-0.7°	25.69°	1.15°	-58.16°	-0.3°	95.25°	11.03°

The first experiment imposes high stiffness values on the end effector to rule out the *impedance controller* as the actual control law used by the LBR Med. The stiffness values used in the experiment can be seen in table 4.2. The equilibrium position is shown in table 4.3.

By performing a constant force on the end effector and measuring the displacement one can compute the actual stiffness imposed by the robot. Table 4.4 shows the results for this static test.

Table 4.4: Results of the static test with high stiffness.

$\overline{y}$	$\overline{f}$	commanded k_{yy}	$\hat{k}_{yy}$
12.10 mm	47.81 N	4000 N/m	3951.2 N/m

Table 4.5: Commanded values for the desired stiffness matrix.

x	y	z	roll	pitch	yaw
3000 N/m	100 N/m	3000 N/m	200 Nm/rad	200 Nm/rad	200 Nm/rad

Table 4.6: Equilibrium position of the LBR Med in joint angles.

A_1	A_2	A_3	A_4	A_5	A_6	A_7
-1.84°	20.84°	4.41°	-43.24°	-0.7°	-65.22°	14.62°

The second experiment uses high stiffness values in two directions and a low stiffness in the remaining direction, as seen in table 4.5. The equilibrium position is shown in table 4.6.

The results of the experiment are shown in figures 4.3 and 4.4 and the forces applied on the flange of the robot are shown in figure 4.5.

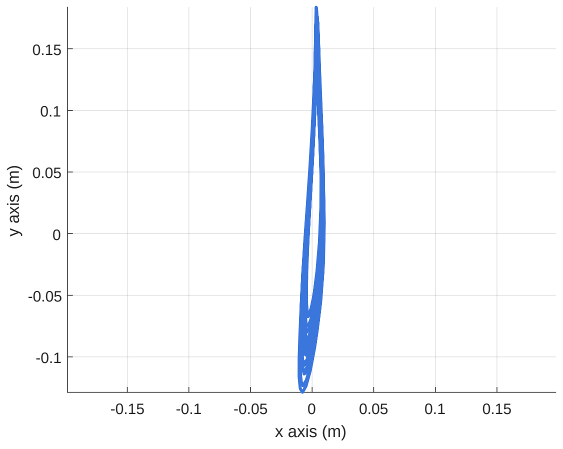


Figure 4.3: Measured displacement of the end effector on the x-y plane.

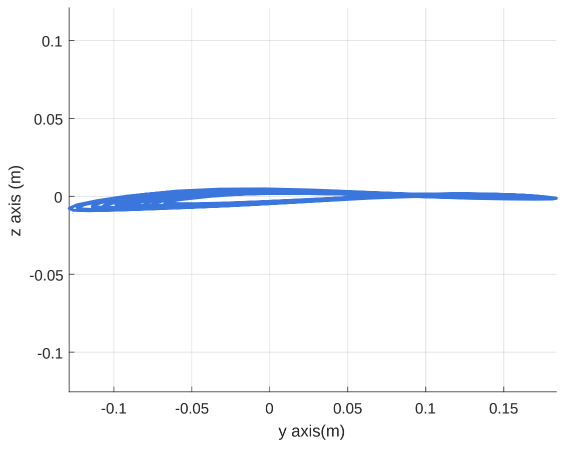


Figure 4.4: Measured displacement of the end effector on the y-z plane.

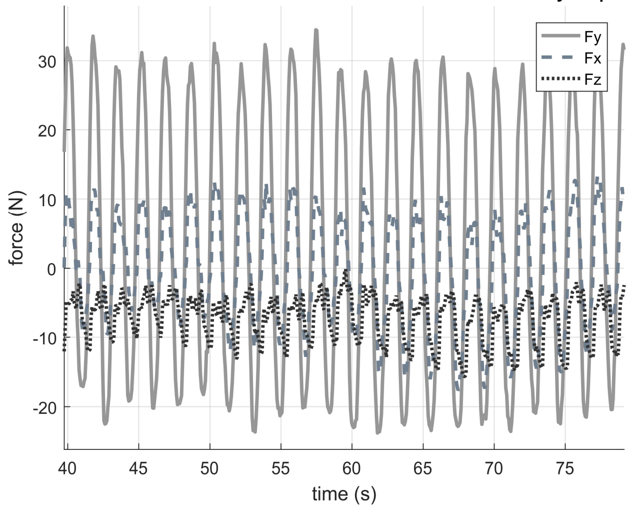


Figure 4.5: Forces applied on the robot's flange.

In the two previous experiments a high rotational stiffness was imposed on the robot. This in turn reduces the rotational effects of the robot to insignificant values.

4.3.2 Stiffness update rate

As introduced in section 3.1.6 both control approaches rely on the constant update of the desired stiffness.

Since the documentation provided by KUKA does not cover how fast or regularly one can update the stiffness imposed on the robot [34], it is necessary to test if the internal socket that serves as an interface between the robot controller and the joints can process the imposed update rate without creating delays in the communication between the controller and the robot joints. If the delay exists then it is necessary to quantify the effect and impose a lower bound on the update rate.

All temporal variables that need a statistical analysis associated with the update of the desired stiffness are shown in figure 4.6.

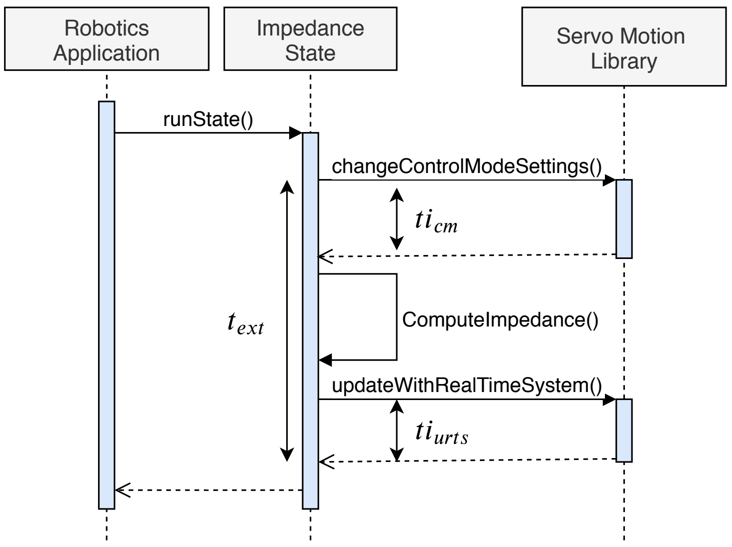


Figure 4.6: Sequence diagram that shown all the relevant elapsed times in the stiffness update cycle.

where ti_{cm} represents the time used by the method *changeControlModeSettings()* to sent to the robot controller the new impedance values, the ti_{urts} represents the time used by the method *updateWithRealTimeSystem()* to retrieve the position information of the robot and t_{ext} represents the total time taken by the cycle to compute the desired stiffness and update all the relevant values.

Since the robot's flange is guided by the surgeon, there is no method to know how long will the desired stiffness be updated, so it is necessary to determine if the time taken by the two methods used in the cycle, *changeControlModeSettings()* and *updateWithRealTimeSystem()*, depend on the number of calls made to them.

To test the hypothesis that the number of calls made to the method is independent from the duration of said method we can design an experiment that provides an accurate statistical analysis of the relationship between ti_{cm} and ti_{urts} with N, where N is the number of cycles.

To keep the sample size constant, so that meaningful comparisons can be made between the four features measured, N is always a multiple of the number 15000, which is the smallest number of cycles that was used in this particular experiment.

Because the largest cycle run reached is 450000 cycles, then the smallest cycle run must be repeated

30 times so that the sample size is equal to $15000\ (\frac{cycles}{run}) \times 30\ (repetitions) = 450000$.

Since both methods only return the average and the standard deviation related to the set of measurements for one repetition, then a formula is necessary to compute the mean and standard deviation from all the repetitions for a given value of N.

Assume that we have k repetitions of size N each. The total set can be written as $S = \{s_1, s_2, ..., s_k\}$, where s_i is the set of a particular repetition composed of the observations $s_i = \{o_1, o_2, ..., o_N\}$.

For any set s_i the average and the variance can be computed by equations 4.2 and 4.3, respectively.

$$\overline{s_i} = \frac{1}{N}\sum_{j=1}^{N} o_j, \quad o_j \in s_i \tag{4.2}$$

$$\sigma^2_{s_i} = \frac{1}{N}\sum_{j=1}^{N}(o_j - \overline{s_i})^2, \quad o_j \in s_i \tag{4.3}$$

The average of the set S is given by equation 4.4.

$$\overline{S} = \frac{1}{N.k}\sum_{p=1}^{N.k} o_p \Leftrightarrow \overline{S}.N.k = \sum_{i_1=1}^{N} o_{i_1} + \sum_{i_2=1}^{N} o_{i_2} + ... + \sum_{i_k=1}^{N} o_{i_k} \Leftrightarrow \overline{S} = \frac{\overline{s_1} + \overline{s_2} + ... + \overline{s_k}}{k} \tag{4.4}$$

To compute the variance of the total set it is useful to perform some algebraic manipulation on equation 4.3 to obtain equation 4.5.

$$\sigma^2_{s_i} = \frac{1}{N}\sum_{j=1}^{N}(o_j - \overline{s_i})^2 = \frac{1}{N}\sum_{j=1}^{N}(o_j^2 - 2\overline{s_i}o_j + \overline{s_i}^2) = \frac{\sum_{j=1}^{N} o_j^2}{N} - 2\overline{s_i}\frac{\sum_{j=1}^{N} o_j}{N} + \overline{s_i}^2 = \frac{\sum_{j=1}^{N} o_j^2}{N} - \overline{s_i}^2 \tag{4.5}$$

By using the formula introduced in 4.5 to the set S we can deduce how the variance of each repetition s_i is related to the variance of S, as it is shown in equation 4.6.

$$\begin{aligned} \sigma^2_S = \frac{\sum_{p=1}^{N.k} o_p^2}{N.k} - \overline{S}^2 = \frac{1}{N.k}(\sum_{i_1=1}^{N} o_{i_1}^2 + \sum_{i_2=1}^{N} o_{i_2}^2 + ... + \sum_{i_k=1}^{N} o_{i_k}^2) - \overline{S}^2 = \\ = \frac{1}{k}((\sigma^2_{s_1} + \overline{s_1}^2) + (\sigma^2_{s_2} + \overline{s_2}^2) + ... + (\sigma^2_{s_k} + \overline{s_k}^2)) - \overline{S}^2 \end{aligned} \tag{4.6}$$

With the previous statistical relations it is possible to compute the total variation and average for each experiment. The results for the duration of the commands *changeControlModeSettings()* and *updateWithRealTimeSystem()*,is shown in table 4.7.

Table 4.7: Results of time measurements for the methods used to update the stiffness and the position of the LBR Med.

Nº cycles	$\overline{ti}_{cm}\ ms$	$sd(ti_{cm})\ ms$	$\overline{ti}_{urts}\ ms$	$sd(ti_{urts})\ ms$
15000	0.9837	0.2385	1.0463	0.2802
50000	0.9822	0.2356	1.0489	0.3047
150000	0.9800	0.2269	1.0500	0.3180
450000	0.9900	0.2200	1.0500	0.3500

Unlike the two previous methods, in order to analyze the time characteristics of the *runState()* method it is necessary to use the *StatisticTimer* provided by KUKA. The class allows one to compute accurate temporal measurements by using the Kernel of the Sunrise.OS.

With this tool the variable t_{ext} introduced in figure 4.6 can be estimated, and the results of this specific experiment are shown in table 4.8.

Table 4.8: Results of the total cycle time needed to update the stiffness value at the joint level.

Nº cycles	min	max	μ	sd
874538	1.74 ms	127.03 ms	2.06 ms	0.42 ms

Since this method is user implemented and it incorporates the previous methods used to update the stiffness and the the robot pose characteristics its duration does not depend on the number of calls made to it if the underlying methods also don't depend on the number of calls.

4.3.3 Impedance characteristics

According to the data reported on the previous section the robot is able to update the desired stiffness, on average, in 2.06 ms but the stiffness simulated by the joints can be distorted due to two main reasons: a) given that our control approach relies on having large deviations from the equilibrium position, the linearization of the Jacobian around the equilibrium point can replicate distorted stiffness matrixes, b) the static joint friction can introduce deviations from the robot's expected behavior thus replicating stiffnesses different from the ones imposed on the flange of the LBR Med.

To test the impedance cone strategy an experiment similar in nature to the previous experiments is proposed. By performing static tests, keeping the value of x as defined in section 3.1.6, close to constant and applying a force in the y direction, one can divide the average displacement by the average disturbance force and compute the stiffness according to hooke's law.

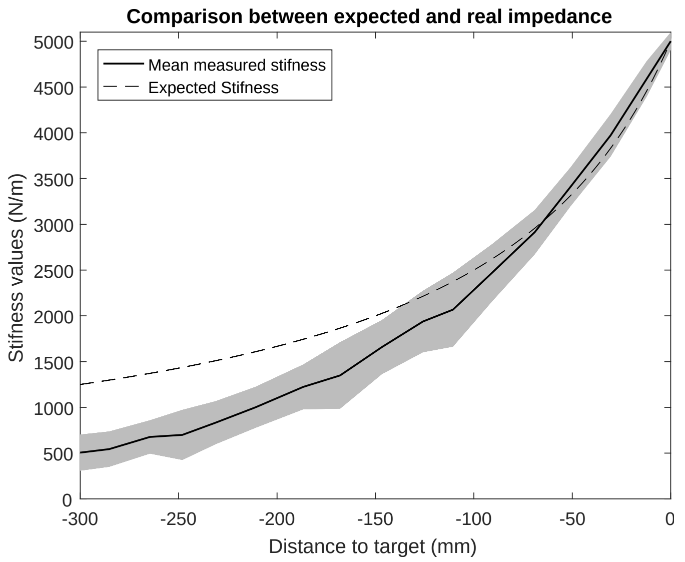


Figure 4.7: Comparison between the commanded stiffness with $d = 0.05$ and estimated stiffness along the trajectory.

There is a loss of performance as the distance to the target increases, as seen in figure 4.7. Because the commanded stiffness decreases as we move away from the target one cannot rule out *a priori* either the friction effect or the geometric effect of the Jacobian on the final replicated stiffness. To quantify the effect of each hypothesis two experiments are proposed.

The first experiment quantifies the joint friction effect. To test this hypothesis the manipulator is placed with a constant displacement with respect to the equilibrium point and different stiffness values are imposed on the end effector.

By applying a constant force on the manipulator and estimating the stiffness simulated by the joint torques one can verify if the difference between the estimated and commanded stiffness is constant as the commanded stiffness changes or if this difference changes. Because the manipulator is perturbed by a constant displacement the Jacobian is close to constant, the only effect that could induce this difference would be the joint friction effect.

The joint positions of this experiment are shown in table 4.9. The manipulator was displaced from the equilibrium position by $20\ mm$ in the x direction and the results of the test are shown in figure 4.8.

Table 4.9: Equilibrium position of the LBR Med in joint angles.

A_1	A_2	A_3	A_4	A_5	A_6	A_7
-1.84°	20.84°	4.41°	-43.24°	-0.7°	-65.22°	14.62°

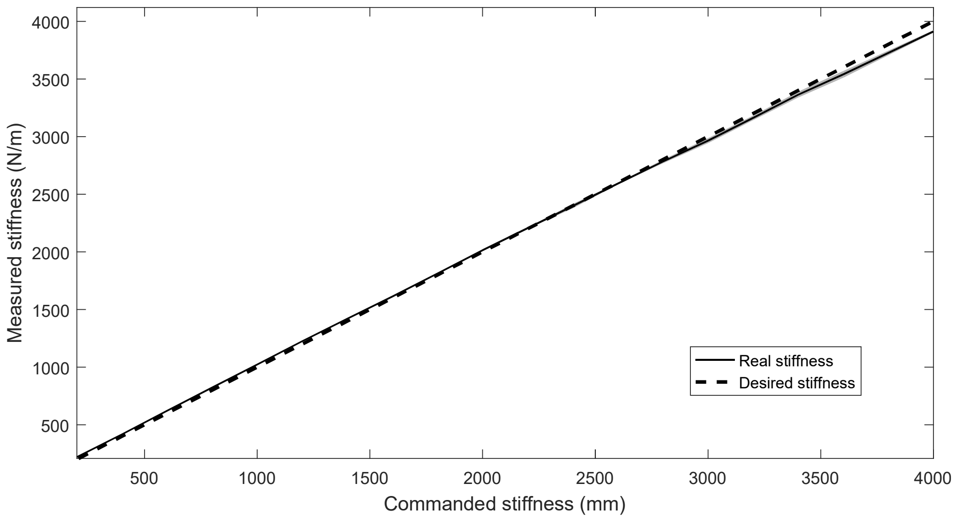


Figure 4.8: Estimated stiffness for different commanded stiffnesses with constant manipulator configuration.

The second experiment tries to probe the geometrical effect on the proposed control strategies. To eliminate the joint friction effect one can select high commanded stiffness values that can overcome any joint friction. By displacing the manipulator from the equilibrium position by known values and performing a static test on each displacement the stiffness can be estimated. Since the stiffness is high the only effect that could explain the discrepancy between the commanded stiffness and the estimated stiffness would be the geometrical error introduced by the changing Jacobian.

For this test the equilibrium position can be computed by the joint positions in table 4.9 and the results are shown in figure 4.9.

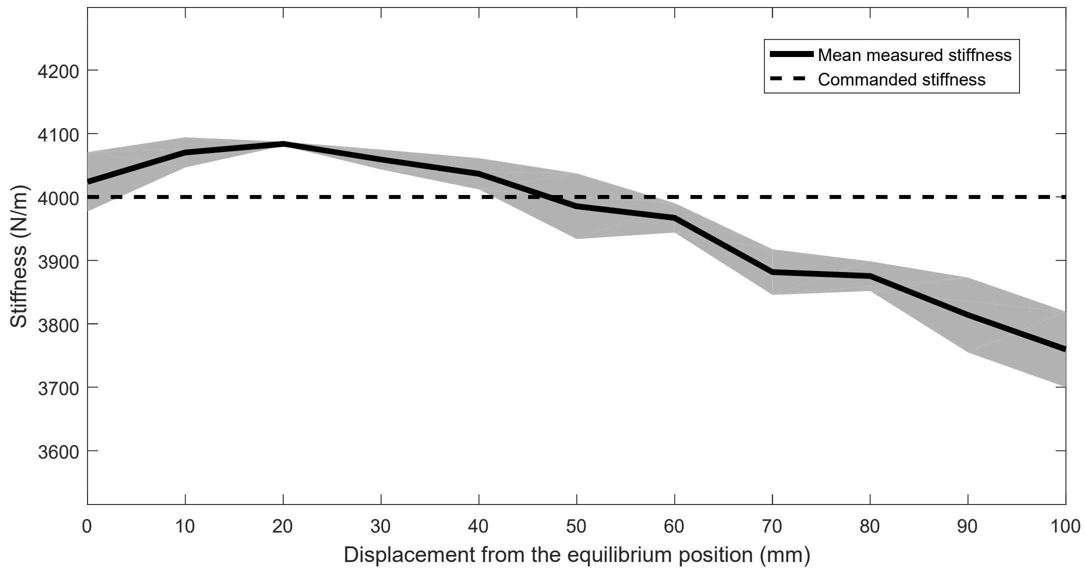


Figure 4.9: Measured stiffness under constant commanded stiffness and changed manipulator configuration.

4.3.4 Discussion of control strategies

In section 4.3.1 the control law used by the LBR Med was probed through two experiments. The first experiment, results shown in table 4.4, eliminates the *Impedance controller* as the underlying control law imposed on the robot, because it cannot replicate high stiffness values as the ones used in the experiment [31].

The results obtained from the second experiment eliminate the *Stiffness controller* as the underlying control law implemented in the LBR Med because there is no noticeable geometric deformation on the trajectory of the end effector around the equilibrium point, as seen in figures 4.3 and 4.4.

It can be concluded that the underlying control law of the LBR Med is the *Impedance controller enhanced by local stiffness control* proposed by Albu-Schaffer et al. [31] given the similarity between their results and the results obtained in this experiment.

Because the control law implemented in the robot is robust against large deviations from the equilibrium position the control strategies used in this thesis are stable and they don't pose stability problems while the surgeon guides the robot along the desired trajectory.

The results shown in table 4.7 show that no correlation exists between N and the time taken by the two methods used to update the stiffness of the robot. The lack of correlation between N and ti_{cmd}/ti_{ulrd} shows that the cone shape impedance strategy is a valid solution to follow the desired trajectory while the surgeon guides the end effector because the desired impedance can be sent to the robot for undetermined amounts of time.

To classify the cone shaped impedance strategy into a soft or hard real time control strategy table 4.8 shows the duration of the method used to update the stiffness of the robot. The results indicate that the cycle has a reliable duration time, with an maximum value of $127\ ms$ and a mean duration of $2.06\ ms$. Because the underlying architecture of the method cannot guarantee a maximum cycle duration, the strategy is classified as a soft real time application [40].

Although the control law implemented in the robot does not pose stability problems, it does not guarantee that the stiffness replicated in the end effector converges to the desired stiffness matrix K_t when the error between the real position and the commanded position is significant error, as explained in section 4.3.1. Figure 4.7 shows that there is indeed a loss in performance of the controller as the error increases.

The first experiment, that maintains the manipulator pose while the desired stiffness is increased, shows that the difference between the desired and replicated stiffness cannot be explained by the low stiffness values send to the robot when the distance to the targeted frame is large. The results can be seen in figure 4.8.

The second experiment shows that the loss in performance is indeed explained by the geometrical deformation introduced by the difference between the Jacobian computed in the equilibrium position and the real Jacobian of the robot. The results can be seen in figure 4.9.

To correct this geometrical error, in theory, the equilibrium position could be changed to the projection of the robot's position onto the desired trajectory as the surgeon moves the end effector. This solution in practice can only be implemented in the impedance line strategy because the robot controller requires $20\ ms$ to change the commanded position of the robot.

If we wished to apply the same logic to the cone shaped impedance strategy, the time required to update the stiffness would go from $2.06\ ms$ to on average $22.06\ ms$ which results in sudden changes in the force done by the flange of the robot on the surgeon's hand.

Given that in the context of the impedance cone strategy when the robot is far away from the commanded frame the desired stiffness is low, then the loss in performance does not pose a significant problem.

4.4 Volume reconstruction performance

4.4.1 Frame calibration

To correctly position the pixels of the US image in the $World\ frame$ of the robot is is necessary to obtain the transformation introduced in equation 3.12. The transformation T_{Flange}^{World} is obtained by the LBR Med. The transformation T_{Flange}^{Image} is a static transformation due to the rigid physical link between the flange and the ultrasound probe.

To estimate the static transformation it is necessary to calibrate the set up. Several calibration techniques are reviewed by Mercier et al. [41]. In this thesis a stylus is used to perform the match between the *World frame* and the *US frame*. This technique was explored in the context of the KUKA light weight + by Rui Coelho [42].

The mathematics behind the calibration algorithm assume that one has two point clouds $P =$

$\{p_1, p_2, ..., p_n\}$ and $Q = \{q_1, q_2, ..., q_n\}$ where p_i and q_i represent the same point in two different coordinate systems, with some noise associated with their measurement. The mapping between two point clouds can be written according to equation 4.7.

$$p_i = R \times q_i + T + \epsilon \tag{4.7}$$

Where R represents the rotation, T the translation and ϵ represents a 3×1 vector which encodes the noise measurements.

By formulating the previous problem in a least squares framework one can estimate the unknown rotation matrix and translation vector. This formulation is shown in equation 4.8.

$$\Sigma = \sum_{i=1}^{n} ||R \times q_i + T - p_i||^2 \tag{4.8}$$

If the rotation matrix is known then the translation can be obtained by taking the derivative of equation 4.8 with respect to T and equal the derivative to zero to obtain the translation which results in the minimum squared error. The result is shown in equation 4.9.

$$\frac{\partial \Sigma}{\partial T} = 0 \longrightarrow \sum_{i=1}^{n} 2 \times ||R \times q_i + T - p_i|| = 0 \longrightarrow R \sum_{i=1}^{n} q_i + nT - \sum_{i=1}^{n} p_i = 0 \longrightarrow \hat{T} = \overline{p} - \hat{R}\overline{q} \tag{4.9}$$

To estimate the rotation matrix, $\hat{R}$, the algorithm proposed by Arun et al. [43] can minimize the cost function of the least squares formulation. The algorithm possesses four steps:

1. Compute the new vectors $p_i^{'}$ and $q_i^{'}$ by subtracting the centroids of the point clouds by taking $p_i^{'} = p_i - \overline{p}$ and $q_i^{'} = q_i - \overline{q}$ where $p = \frac{1}{N}\sum_{i=1}^{n} p_i$ and $q = \frac{1}{N}\sum_{i=1}^{n} q_i$.
2. Compute the 3×3 matrix $H = \sum_{i=1}^{n} p_i^{'} . q_i^{'T}$
3. Find the singular value decomposition of H as $H = U \Lambda V^T$
4. Compute $X = VU^T$
5. Check the determinant of X. If $det(X) = 1$ then $\hat{R} = X$, if $det(X) = -1$ then the algorithm has failed.

With the mathematical background completed the algorithm specific for the proposed set up can now be developed.

All the intervening frames of reference in the calibration set up are introduced in figure 4.10.

By positioning the tip of the stylus in such a way that it appears on the US image one can know its position by both the polaris device and the probe rigidly attached to the robot as it is shown in equation 4.10.

$$p_{Tool}^{World} = T_{Flange}^{World} . T_{Ultrasound}^{Flange} . p_{Tool}^{Ultrasound} = T_{Polaris}^{World} . p_{Tool}^{Polaris} \tag{4.10}$$

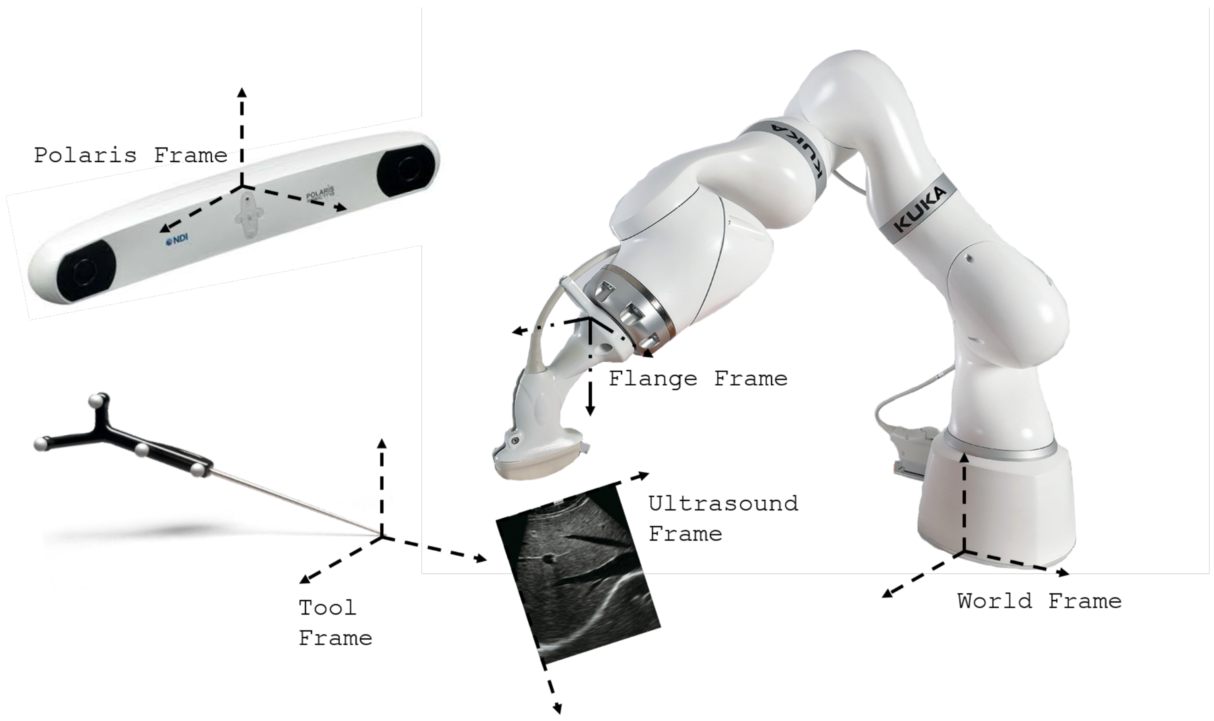


Figure 4.10: Frames of reference required to perform the spacial calibration between the US frame and the Flange frame

To retrieve the transform $T_{Ultrasound}^{Flange}$ equation 4.10 can be algebraically manipulated into equation 4.11.

$$T_{Ultrasound}^{Flange}.p_{Tool}^{Ultrasound} = T_{World}^{Flange}.T_{Polaris}^{World}.p_{Tool}^{Polaris} \tag{4.11}$$

On the left hand side of equation 4.11 one can measure the disturbance on the US image due to the interference generated by the stylus. On the right hand side the position of the tool provided by the Polaris is obtained.

To estimate the $T_{Ultrasound}^{Flange}$ transformation it is necessary to perform the mapping described earlier between the point cloud $p_{Tool}^{Ultrasound}$ to the point cloud p_{Tool}^{Flange}.

To compute the translation p_{Tool}^{Flange} the algorithm requires that the transformation $T_{Polaris}^{World}$ is known. To obtain this transformation we can position the robot in different configurations and extract the position of the LBR Med flange both from the robot software and the polaris tracking device. With at least three points we can use the algorithm introduced earlier to compute the rigid transformation between the polaris frame and the world frame.

Since the algorithm requires intermediary calibrations it is not possible to estimate the error of the calibration from the root mean squared error between the two point clouds. To obtain a proper evaluation of the calibration one can compute the image position and the position of the stylus in *World frame* and compare the shift between the perturbation in the US image and the stylus tip.

In figures 4.11 and 4.12 this validation is shown. The characteristic cross pattern of the stylus interference in the US image can also be observed.

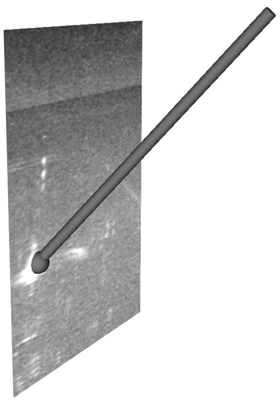


Figure 4.11: Cross perturbation created by the needle on the tracked US image.

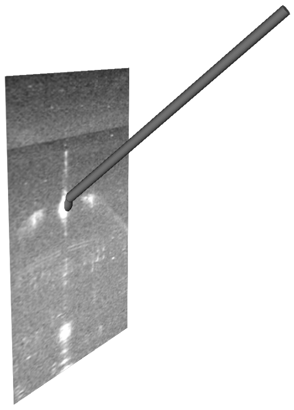


Figure 4.12: Cross perturbation created by the needle on the tracked US image.

The error between the stylus tip and the cross pattern in the US image is a vector. To obtain the average error several points from the US image are obtained, from which the norm of the corresponding error vectors is computed and the average of the norms obtained.

In table 4.10 the results of this experiment are shown for different rotations of the US image in relation to the position used to calibrate the set up.

Table 4.10: Calibration error between the *Ultrasound Image* and *Flange frame*

Rotation	μ	sd	max
0°	0.7307	0.4763	2.1364
5°	1.0266	0.7341	2.4931
10°	1.2788	0.3885	2.0841
15°	1.2906	0.5984	2.8113

There are two effects that influence the results shown in table 4.10. The error introduced from the identification of the stylus tip on the US image and the bias introduced by the developer when selecting the points where the stylus seen in figure 4.12 is over the US image.

To mitigate the bias added by the human selection the US image was evaluated without visual information from the stylus position.

Another important question is: *are the errors reported on table 4.10 the minimum achievable errors using this calibration algorithm?* To answer this question figure 4.19 shows the orientation of the error vectors for the different angles of the US image.

If the error obtained with this calibration algorithm is indeed the minimum achievable error then one would expect that the orientation of the error vector to be closely approximated by a random distribution. Contrarily, if there was an unaccounted offset that the rigid transformation between US image and the robot flange did not capture, then the error vectors should be closely aligned in a given orientation.

The results of this qualitative analysis are shown in figure 4.13.

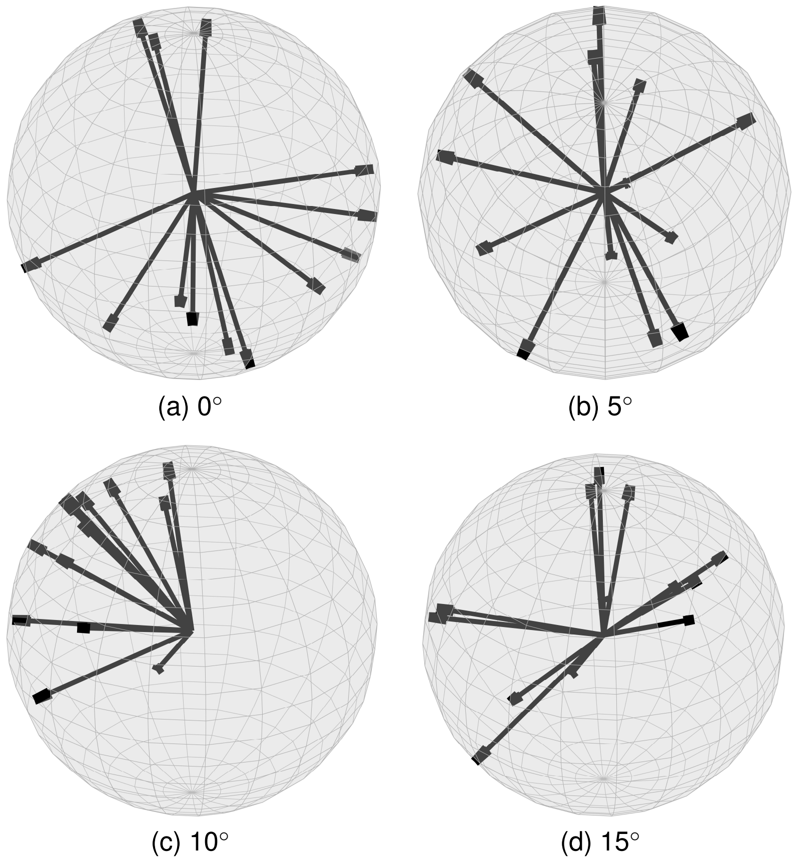

(a) 0°
(b) 5°
(c) 10°
(d) 15°

Figure 4.13: Normalized error vectors for different US image poses.

4.4.2 Temporal calibration

The communication between the robot and the main workstation is classified as a soft real time application, meaning that the processing time taken between each message sent by the robot varies. This variation can introduce a positioning error, because the US image captured by the frame grabber can be incorrectly matched with a transformation sent by the robot controller.

To correct this variable delay, one can use the embedded calibration algorithm provided by the plus library.

By positioning the transducer of the robot over a water tank and performing a sinusoidal movement in the perpendicular direction of the bottom of the water tank we can compute the time shift between the image stream and the tracking stream [44].

This time lag is assumed to be constant, which for low probe velocities produces a small error on the position of the US image on the world frame of the robot.

To extract the position of the bottom of the water tank in the image stream the plus library divides the image in columns and applies a Laplacian filter on each column, afterwards the position of the bottom is extracted from all the columns. In figure 4.14 we can see the end result of this segmentation.

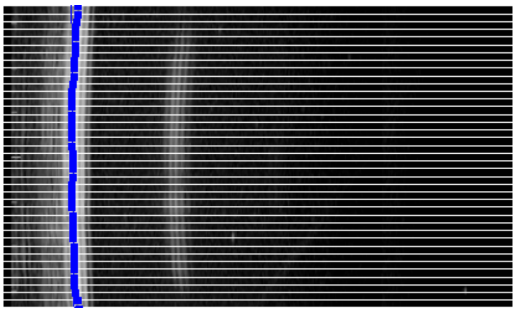


Figure 4.14: Bottom of the water tank extracted using the Plus library.

The unprocessed signals obtained from the tracking signal and the image signal is shown in figure 4.15. The x axis represents the time stamps of the messages and the y axis represents the normalized sinusoidal movement.

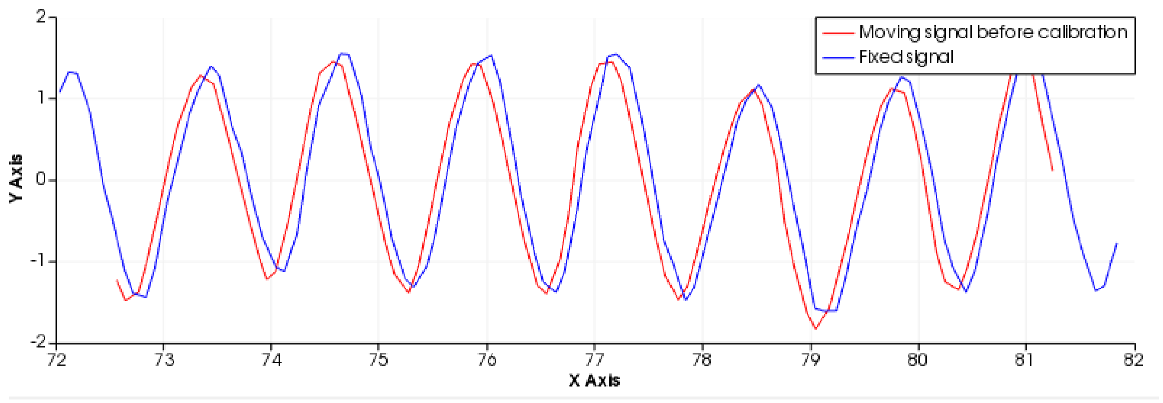


Figure 4.15: Uncalibrated signals obtain from the US image stream (fixed signal) and the tracking signal (moving signal).

To obtain the time shift that corrects the delay between the two signals Plus computes their cross correlation and selects the delay that shows the highest correlation between the two signals.

In the proposed set up of this thesis the correlation between the two signals is the highest with a delay between the imaging device and the tracking device of $80\ ms$. In figure 4.16 we can see the that the two signals overlap with each other once the delay between the two systems is considered.

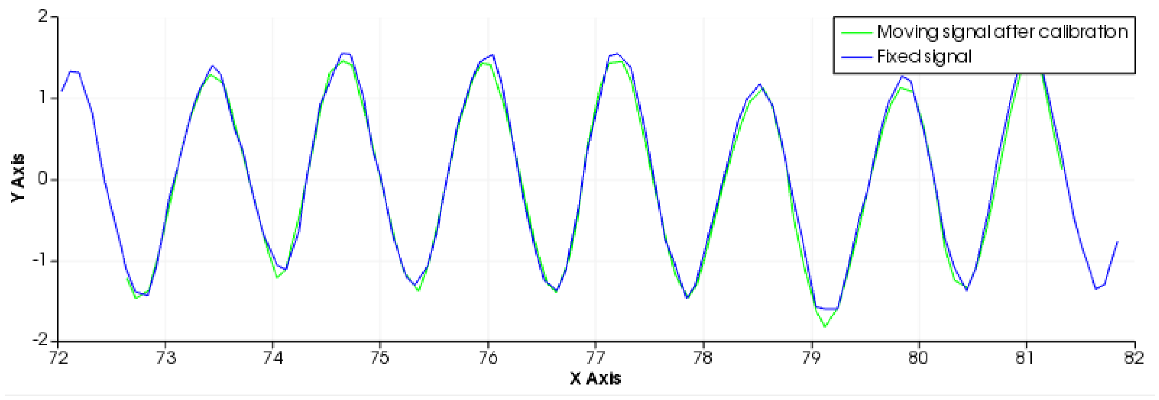
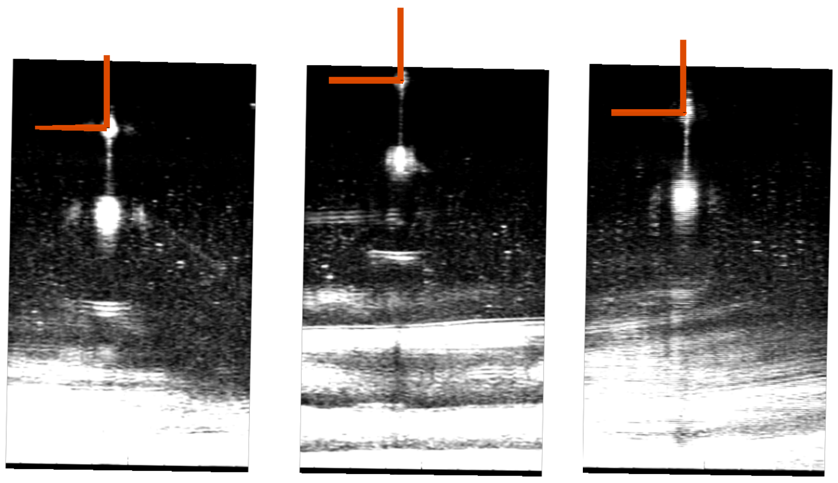


Figure 4.16: Calibrated signals after the correction applied through the temporal calibration.

To validate the temporal calibration one can place a stylus in a fixed position inside a water tank and compute the movement of the cross perturbation caused by the stylus on the US image as the robot's flange is moved under a sinusoidal force.

If the calibration reduces the delay between the tracking data stream and the image data stream to insignificant levels then one would expect that the position of the stylus disturbance would remain constant in the US tracked image in relation to the *World frame*.

In figure 4.17 the static stylus effect can be seen on the three images obtained sequentially.



a) US image at t_0 b) US image at t_1 c) US image at t_2

Figure 4.17: The sequence of US images shows the stylus fixed in space as the US image moves under sinusoidal movement.

To obtain a quantitative measure of the temporal calibration several points can be marked in different US images, and their position can be extracted. Afterwards one can compute the average of all the points and the average of the norm from each point to the mean, thus obtaining information about the relative movement of the stylus is space. The results to this experiment are shown in table 4.11.

Table 4.11: Temporal Calibration error

	μ	sd	max
Calibrated	1.53 mm	0.73 mm	2.73 mm
Uncalibrated	3.81 mm	2.36 mm	8.01 mm

4.4.3 Effects of velocity on the reconstructed volume

The last important parameter in the set up that has not been analyzed is the effect of the probe velocity on the final reconstructed volume.

To test the parameter a simple experiment is proposed. By placing a coin with known dimensions under a silicone plate and commanding the robot to scan the coin with a desired velocity it is possible to analyze each volume and extract the final dimensions of the reconstructed coin.

The dimensions of the scanned coin are shown in table 4.12.

Table 4.12: Two euro coin dimensions (from *fleur-de-coin*).

Diameter	25.75 mm
Thickness	2.20 mm
Weight	8.50 g
Shape	round

In figure 4.18 a isometric projection is shown of the reconstructed coin under a probe velocity of 5 mm/s.

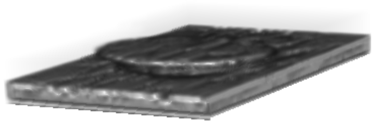
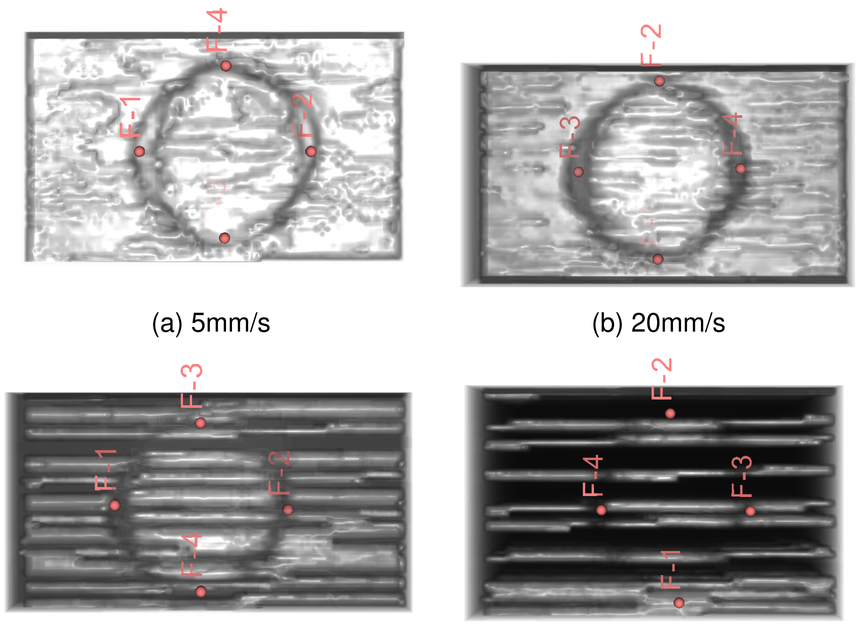


Figure 4.18: Isometric view of reconstructed coin with a probe velocity of 5 mm/s.

The acoustic impedance of the coin is significantly larger than water, around 30 times larger, which causes the sound waves to create an US imaging artifact known as echo, which in turn enlarges the size of the reconstructed coin. In figure 4.18 the estimated diameter of the coin from the reconstructed volume is close to 31.5 mm while the actual coin diameter is 25.75 mm.

The echo artifact renders any proper evaluation of the spacial calibration unachievable with this experiment, therefore only the synchronization between the frame grabber and the tracker can be evaluated with this experiment.

In figure 4.19 the reconstructed coin is shown for different probe velocities.



(a) 5mm/s

(b) 20mm/s

(c) 40mm/s

(d) 80mm/s

Figure 4.19: Top view of the reconstructed coin under different probe velocities

To quantify the distortion of the reconstructed volumes shown in figure 4.19, one can measure the diameter of the coin the along the direction of motion and compare it to the diameter of the coin in the perpendicular direction to the first diameter. The results of this experiment are shown in table 4.13.

Table 4.13: Distortion of the coin dimensions under different probe speeds.

probe speed (mm/s)	$diameter_{\parallel}$ (mm)	$diameter_{\perp}$ (mm)
5	31.01	32.20
20	30.17	32.88
40	29.48	30.63
80	36.89	28.62

Where the dimension of the diameter in the direction of motion is called $diameter_{\parallel}$ and the diameter perpendicular to $diameter_{\parallel}$ is called $diameter_{\perp}$.

One interesting effect which was predicted in section 3.2.4 is the deformation of the reconstructed coin due to the relative motion of the probe while the transducer fires the sequential sound waves measured by the piezoelectric crystals. To visualize the effect figure 4.20 shows the probe moving while the US probe obtains the B-mode US images.

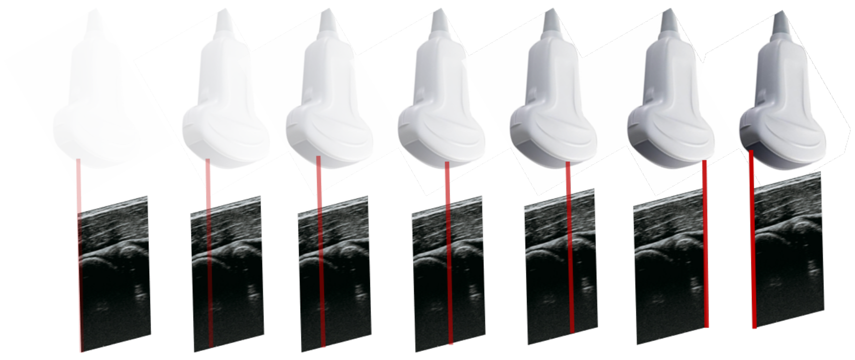


Figure 4.20: Transducer sequential activation while probe is subjected to a linear motion (from left to the right in the figure)

The red lines in figure 4.20 represent the trigger of the piezoelectric element. If the velocity of the probe has a considerable proportion to the frequency of the transducer, then the red lines that compose a single US image will encode information from different slices of the scanned tissue.

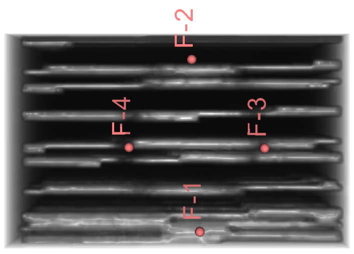


Figure 4.21: Deformation of the reconstructed volume due to relative motions.

This effect can be seen in figure 4.21, where the US images that compose the reconstructed volume show a small, but noticeable, inclination, due to the relative motion of the probe in relation to the fired US signal by the piezoelectric crystals.

4.4.4 Discussion of volumetric reconstruction

The spacial calibration method used to obtain the transformation between the US image and the robot's coordinate system produces small errors, as seen in table 4.10.

The average error of 1.2 mm means that, on average, the real position of a pixel in the US image lies on a sphere with a radius of 1.2 mm of the reported position by the tracker. Depending on the performance of the registration method used to obtain the mapping between the US image space and the MRI space this error can be acceptable.

Regarding the important question posed in section 4.4.1, by looking at the orientation error shown in figure 4.13, we can conclude that there is no static bias unaccounted by the transformation between the US image and the flange of the robot, thus the average error of 1.2 mm is probably the smallest error achievable with this calibration method.

If the error is too large, several additional methods have been proposed that might bring the positioning error to lower values [45].

Regarding the temporal calibration algorithm used to calibrate the set up we obtain a reduction of 60%, as seen in table 4.11, when looking at the relative motion of the static stylus on the US image.

While without calibration, the fixed stylus shows an apparent motion with a maximum deviation from the average measurement of 8.01 mm, when the set up is temporally calibrated the maximum deviation from the average measurements is 2.73 mm which is a significant reduction, thus proving the hypothesis proposed in section 3.2.1.

If the probe is only subjected to low velocity values, then this apparent movement can be reduced to even lower values.

The effects of the probe velocity on the reconstructed volume are explored in section 4.4.3. The first apparent disturbance seen in the reconstructed volume is explicit in figure 4.19, where the reconstructed coin shows gaps of empty voxels as the probe velocity increases.

These gaps can be explained by looking at the frame rate of the frame grabber used to capture US images. If one scans a coin with a diameter of 27.55 mm with a probe velocity of 80 mm/s the necessary time to scan the coin is 0.325 seconds. During this time the frame grabber, which obtains images at 25 frames per second, can only obtain 8 images. The regions between the 8 obtained images will be filled with empty voxels.

It is clear from this experiment, that to avoid said gaps, which will influence the registration algorithm used to map the US space to the MRI space, an upper bound on the probe velocity must be placed.

In regards to the distortion caused by the increase in velocity, table 4.13 provides the necessary data to understand how the velocity influences the distorted volume.

For low probe speeds, under 40 mm/s, an almost constant shift exists between the parallel and perpendicular diameter of the reconstructed coin. This shift can be accounted by the asymmetry introduced by the echo artifact present on the US images. For higher probe velocities the shift grows larger, and this affect can no longer be explained solely by the echo artifact.

The only explanation for this distortion found by the author of this work is that the assumption of the constant delay between the tracker and the frame grabber breaks down at high velocities.

The last effect observed in figure 4.21 is the relative motion of the US probe in relation to the transducer signal. Under low probe velocities this effect does not pose significant problems because the frequency of the transducer lies in the gama of 15 MHz [9], meaning that the relative movement of the red line in the direction along the trajectory is irrelevant compared to the motion along the US image.

Chapter 5

Conclusions

5.1 Achievements

In the previous chapters all three abstract high level objectives, proposed in section 1.2, were achieved up to different degrees of completion.

The first high level objective of this thesis is the exploration and proper decoding of the internal mechanics of the LBR Med. Since the API of the robot is fairly recent, there is still a noticeable lack of resources to inform developers on future code architectures.

With all the resources shown throughout the thesis the author hopes to provide some guiding principles on future software architectures. Another important resource introduced in the thesis are the performance benchmarks for certain key classes embedded API of KUKA, which allow developers to design control strategies for different projects.

The second high level objective, and arguably the most important contribution of this thesis to the medical field, is the Java implementation of the OpenIGTLink protocol. The library allows any Workbench developer to design complex communication structures, due to the underlying flexibility of the protocol, to manipulate the LBR Med.

Although the library already enjoys several Java tutorials created by the author to facilitate the prototyping stage of future developers, the actual limitations of the library have not been probed. These unknown performance parameters, such as the latency with several clients connected to one server, can undermine the work of future developers when using the library in operating conditions significantly different from the ones reported in this thesis.

The last high level objective was the exploration of the registration problem in the context of the neurosurgical field. Several important results, both the spacial and temporal calibration algorithms and the limitations of this set up in terms of maximum probe speed, were studied to understand the limitation of the set up.

Although this work already introduces important results, further analysis is required to actually implement the data fusion between the MRI space and the US space.

The set up proposed in section 2.3 was developed with the flexibility to add new devices to the navigation platform, through the use of the Plus toolkit, and the set up does not rely on either the imaging device or the visualisation platform.

The independence of the different components of the navigation platform allows any research group to implement a similar set up with their proprietary software.

Another interesting result that can be extracted from the work developed in this thesis is the flexibility of the LBR Med under different operating conditions.

Due to the simple Workbench API, any developer can design systems specialized for different areas because the complex control laws implemented in the robot can be easily adapted to different conditions.

5.2 Future Work

As stated in section 2.3 this thesis focuses on key stages of the neurosurgical procedure. Of the four main surgical stages this thesis focuses on two stages, the ultrasound scanning and the surgical procedure.

To create a fully fledged navigation platform the registration and preoperative stage must be studied and integrated with the set up proposed in this work. Both of these stages can be tackled with an GUI embedded in 3D Slicer, developed in C++, so that the performance of the complex segmentation algorithms do not represent a bottleneck for the medical market.

One important problem in the previous set up lies on the clock drift between the intervening workstations, namely the main workstation and the robot controller. To tackle this problem one could use the NTP or PTP protocols to correct the time mismatch.

The problem with both of these solutions is the fact that the robot controller is a closed system which possesses safety features valid as long as the user does not tamper with the controller. The author of this work could not find a workable solution to this problem.

Although the time drift problem does not pose a problem in the set up proposed by this thesis it limits the potential of the navigation platform on important areas such as safety features that guarantee that the data being visualized in Slicer is received in some constant frame rate.

Another area of development should be the implementation of latter versions of the OpenIGTLink protocol in Java. This would in turn allow the LBR Med to be integrated into even more complex networks.

Bibliography

[1] D. L. Barrow and B. R. Bendok. Introduction: What is neurosurgery? *Operative Neurosurgery*, 17 (Supplement1):1–2, 05 2019. doi: 10.1093/ons/opz071.

[2] A. Golby. *Image-Guided Neurosurgery*. Elsevier Science, 2015. ISBN 9780128011898. URL `https://books.google.pt/books?id=2v7IBAAAQBAJ`.

[3] J. S. Henn, G. M. Lemole, M. Gerber, and R. F. Spetzler. Theory and development of frameless stereotaxy. *Barrow Quaterly*, 17, 2001.

[4] K. H. Manwaring, M. L. Manwaring, and S. D. Moss. Magnetic field guided endoscopic dissection through a burr hole may avoid more invasive craniotomies a preliminary report. In B. L. Bauer and D. Hellwig, editors, *Minimally Invasive Neurosurgery II*, pages 34–39, Vienna, 1994. Springer Vienna. ISBN 978-3-7091-6908-7.

[5] V. Horsley and R. H. Clarke. The structure and functions of the cerebellum examined by a new method. *Brain*, 31(1):45–124, 05 1908. ISSN 0006-8950. doi: 10.1093/brain/31.1.45.

[6] M. Y. Chen, T. L. Pope, and D. J. Ott. *Basic Radiology, Second Edition*. McGraw-Hill Education - Europe, Oct 2010.

[7] P. Faulhaber, A. Nelson, L. Mehta, and J. O'Donnell. 24. The Fusion of Anatomic and Physiologic Tomographic Images to Enhance Accurate Interpretation.

[8] D. W. Roberts, A. Hartov, F. E. Kennedy, M. I. Miga, and K. D. Paulsen. Intraoperative brain shift and deformation: a quantitative analysis of cortical displacement in 28 cases. *Neurosurgery*, 43(4): 749–758, Oct 1998.

[9] S. Ji, D. W. Roberts, A. Hartov, and K. D. Paulsen. Combining multiple true 3D ultrasound image volumes through re-registration and rasterization. *Med Image Comput Comput Assist Interv*, 12(Pt 1):795–802, 2009.

[10] N. Nathoo, M. C. Cavusoglu, M. A. Vogelbaum, and D. M. Peereboom. In touch with robotics: neurosurgery for the future. *Neurosurgery*, 56 3:421–33; discussion 421–33, 2005.

[11] J. W. Motkoski and G. R. Sutherland. *Progress in Neurosurgical Robotics*, pages 601–612. Springer New York, New York, NY, 2014. ISBN 978-1-4614-7657-3. doi: 10.1007/978-1-4614-7657-3_46. URL `https://doi.org/10.1007/978-1-4614-7657-3_46`.

[12] G. Mazzon, A. Sridhar, G. Busuttil, J. Thompson, S. Nathan, T. Briggs, J. Kelly, and G. Shaw. Learning curves for robotic surgery: a review of the recent literature", journal="current urology reports. 18(11):89, Sep 2017. ISSN 1534-6285. doi: 10.1007/s11934-017-0738-z. URL `https://doi.org/10.1007/s11934-017-0738-z`.

[13] L. I. M. Pernar, F. C. Robertson, A. Tavakkoli, E. G. Sheu, D. C. Brooks, and D. S. Smink. An appraisal of the learning curve in robotic general surgery", journal="surgical endoscopy. 31(11): 4583–4596, Nov 2017. ISSN 1432-2218. doi: 10.1007/s00464-017-5520-2. URL `https://doi.org/10.1007/s00464-017-5520-2`.

[14] Takacs, Árpad, Nagy, Denes, Rudas, Imre, and T. Haidegger. Origins of surgical robotics: From space to the operating room. 13:13–30, 01 2016.

[15] T. R. K. Varma and P. Eldridge. Use of the neuromate stereotactic robot in a frameless mode for functional neurosurgery. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 2(2):107–113, 6 2006. ISSN 1478-596X. doi: 10.1002/rcs.88. URL `https://doi.org/10.1002/rcs.88`.

[16] P. Finlay and P. Morgan. Pathfinder image guided robot for neurosurgery. *Industrial Robot-an International Journal - IND ROBOT*, 30:30–34, 02 2003. doi: 10.1108/01439910310457689.

[17] R. Falconer, S. Rogers, C. Brewer, F. Piscitani, and M. Shenai. Presurgical rehearsals for patients considering "awake" deep brain stimulation. *Frontiers in Surgery*, 3, 08 2016. doi: 10.3389/fsurg.2016.00044.

[18] D. Amin, T. Kanade, A. DiGioia, and B. Jaramaz. Ultrasound registration of the bone surface for surgical navigation. *Computer aided surgery : official journal of the International Society for Computer Aided Surgery*, 8:1–16, 02 2003. doi: 10.3109/10929080309146097.

[19] A. Lasso, T. Heffter, A. Rankin, C. Pinter, T. Ungi, and G. Fichtinger. Plus: Open-source toolkit for ultrasound-guided intervention systems. *IEEE Transactions on Biomedical Engineering*, 61(10): 2527–2537, Oct 2014. doi: 10.1109/TBME.2014.2322864.

[20] J. Neuhaus, J. Kast, I. Wegner, M. Baumhauer, A. Seitel, I. Gergel, M. Nolden, D. Maleike, I. Wolf, H.-P. Meinzer, and L. Maier-Hein. Building image guided therapy applications with the medical imaging interaction toolkit. 01 2009.

[21] P. K. Seitz. *Konzeption und Entwicklung einer robotergestützten und ultraschallbasierten Lokalisationskontrolle*. Master thesis, 2018. URL `http://nbn-resolving.de/urn:nbn:de:bsz:840-opus4-1508`.

[22] A. Meyer, A. Lasso, T. Ungi, and G. Fichtinger. Live ultrasound volume reconstruction using scout scanning. *Proceedings of SPIE–the International Society for Optical Engineering*, 9415, 02 2015. doi: 10.1117/12.2081488.

[23] S. Drouin, A. Kochanowska, M. Kersten-Oertel, I. J. Gerard, R. Zelmann, D. De Nigris, S. Bériault, T. Arbel, D. Sirhan, A. F. Sadikot, J. A. Hall, D. S. Sinclair, K. Petrecca, R. F. DelMaestro, and D. L. Collins. Ibis: an or ready open-source platform for image-guided neurosurgery. *International Journal of Computer Assisted Radiology and Surgery*, 12(3):363–378, Mar 2017. ISSN 1861-6429. doi: 10.1007/s11548-016-1478-0. URL `https://doi.org/10.1007/s11548-016-1478-0`.

[24] C. Askeland, O. V. Solberg, J. B. L. Bakeng, I. Reinertsen, G. A. Tangen, E. F. Hofstad, D. H. Iversen, C. Våpenstad, T. Selbekk, T. Langø, T. A. N. Hernes, H. Olav Leira, G. Unsgård, and F. Lindseth. Custusx: an open-source research platform for image-guided therapy. *International Journal of Computer Assisted Radiology and Surgery*, 11(4):505–519, Apr 2016. ISSN 1861-6429. doi: 10.1007/s11548-015-1292-0. URL `https://doi.org/10.1007/s11548-015-1292-0`.

[25] T. Ungi, A. Lasso, and G. Fichtinger. Open-source platforms for navigated image-guided interventions. *Medical Image Analysis*, 33, 06 2016. doi: 10.1016/j.media.2016.06.011.

[26] T. Salgueiro. *Robot assisted needle interventions for brain surgery*. Master thesis, 12 2014.

[27] P. P. Roios. *A Co-manipulation Robotic System for Brain Biopsies*. Master thesis, 06 2016.

[28] M. Elyseu. *Registo Intraoperatório com Recurso a Ultrassons em Neurocirurgia*. Master thesis, 06 2018.

[29] *Operating and Programming Instructions for System Integrators Instructions for Use*. KUKA Deutschland GmbH, ga kuka sunrise.os med 1.15 v2 edition, 10 2018.

[30] A. Albu-Schaffer and G. Hirzinger. Cartesian impedance control techniques for torque controlled light-weight robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 657–663 vol.1, May 2002. doi: 10.1109/ROBOT.2002.1013433.

[31] A. Albu-Schaffer, C. Ott, and G. Hirzinger. A passivity based cartesian impedance controller for flexible joint robots - part ii: full state feedback, impedance design and experiments. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 3, pages 2666–2672 Vol.3, April 2004. doi: 10.1109/ROBOT.2004.1307463.

[32] S. Tauscher, J. Tokuda, G. Schreiber, T. Neff, N. Hata, and T. Ortmaier. Openigtlink interface for state control and visualisation of a robot for image-guided therapy systems. *International Journal of Computer Assisted Radiology and Surgery*, 10(3):285–292, Mar 2015. ISSN 1861-6429. doi: 10.1007/s11548-014-1081-1. URL `https://doi.org/10.1007/s11548-014-1081-1`.

[33] S. Haddadin, A. Albu-Schäffer, and G. Hirzinger. Requirements for safe robots: Measurements, analysis and new insights. *The International Journal of Robotics Research*, 28(11-12):1507–1527, 2009. doi: 10.1177/0278364909343970. URL `https://doi.org/10.1177/0278364909343970`.

[34] *KUKA Sunrise.SmartServo 1.9*. KUKA Deutschland GmbH, v1 edition, 01 2016.

[35] J. Tokuda, G. S. Fischer, X. Papademetris, Z. Yaniv, L. Ibanez, P. Cheng, H. Liu, J. Blevins, J. Arata, A. J. Golby, T. Kapur, S. Pieper, E. C. Burdette, G. Fichtinger, C. M. Tempany, and N. Hata. Openigtlink: an open network protocol for image-guided therapy environment. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 5(4):423–434, 2009. doi: 10.1002/rcs.274. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rcs.274`.

[36] H. Mu and S. Jiang. Design patterns in software development. In *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, pages 322–325, July 2011. doi: 10.1109/ICSESS.2011.5982228.

[37] B. Jack, D. S. Duick, and R. A. Levin. *Thyroid Ultrasound and Ultrasound-Guided FNA*. Springer, Nov 2013.

[38] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, Oct 1991. doi: 10.1109/26.103043.

[39] R. L. Scheiterer, C. Na, D. Obradovic, G. Steindl, and F. Goetz. Synchronization performance of the precision time protocol in the face of slave clock frequency drift. In *2008 IEEE International Conference on Automation Science and Engineering*, pages 554–559, Aug 2008. doi: 10.1109/COASE.2008.4626480.

[40] G. Lipari and L. Palopoli. Real-time scheduling: from hard to soft real-time systems. *CoRR*, abs/1512.01978, 2015. URL `http://arxiv.org/abs/1512.01978`.

[41] L. Mercier, T. Langø, F. Lindseth, and D. L. Collins. A review of calibration techniques for freehand 3-d ultrasound systems. *Ultrasound in Medicine and Biology*, 31, 2005. doi: 10.1016/j.ultrasmedbio.2004.11.015. URL `https://doi.org/10.1016/j.ultrasmedbio.2004.11.015`.

[42] R. Coelho. Ultrasound image acquisition integration with kuka lwr for hard real-time synchronization. Internal report from SRL, July 2016.

[43] K. Arun, T. Huang, and S. Blostein. Least-squares fitting of two 3-d point sets. ieee t pattern anal. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9:698 – 700, 10 1987. doi: 10.1109/TPAMI.1987.4767965.

[44] E. Moult, A. Lasso, T. Ungi, C. Pinter, M. Welch, and G. Fichtinger. Improved temporal calibration of tracked ultrasound: An open-source solution. *Journal of Medical Robotics Research*, 02:1–8, 08 2017. doi: 10.1142/S2424905X17500088.

[45] A. Lasso, T. Heffter, C. Pinter, T. Ungi, T. Chen, A. Boucharin, and G. Fichtinger. Plus: An open-source toolkit for developing ultrasound-guided intervention systems. *4th NCIGT and NIH Image Guided Therapy Workshop*, 4:103, 01 2011.

Appendix A

Plus configuration

The XML file used by the plus library is critical step if one wishes to use the library capabilities.

The community that develops the library has a large array of examples for different purposes, but the we found them lacking when trying to actually use the library capabilities with our set up.

To increase the learning curve slope of future developers this section tries to provide some insight into the structure of a configuration file.

The following sections create the configuration file of the proposed set up of this thesis step by step.

A.1 Base structure

The base structure of any configuration file is shown in the following piece of code.

```
<PlusConfiguration version="2.0">
  <DataCollection StartupDelaySec="1.0" >
    <DeviceSet
      Name="Example"
      Description="Example to show the structure of the required configuration file."
    />
    <Device
      ...
    </Device>
    ...
  </DataCollection>

  <CoordinateDefinitions>
    ...
  </CoordinateDefinitions>

  <PlusOpenIGTLinkServer
    ...
  </PlusOpenIGTLinkServer>

</PlusConfiguration>
```

In the set up of the thesis we have two devices, the LBR Med that communicates with the plus library through the OpenIGTLink protocol and the frame grabber DFG/USB2propcb which retrieves the US images from the Prosound unit.

Since the frame grabber has a software development kit (SDK) which is compatible with the plus library we can simply use the preconfigured device. In the following piece of code the configuration of the DFG/USB2propcb device is shown.

```
<Device
  Id="VideoDevice"
  Type="ICCapturing"
  DeviceName="DFG/USB2pro"
  VideoNorm="PAL_B"
  VideoFormat="Y800"
  FrameSize="768 576"
  InputChannel="00 Video: Composite" >
  <DataSources>
    <DataSource
      Type="Video"
      Id="Video"
      PortUsImageOrientation="MF"
      ClipRectangleOrigin="120 90"
      ClipRectangleSize="490 422"
      />
  </DataSources>
  <OutputChannels>
    <OutputChannel Id="VideoStream" VideoDataSourceId="Video" />
  </OutputChannels>
</Device>
```

The **Type** of any device tells Plus if the device sends images or transformations. The **VideoNorm**, **VideoFormat**, **InputChannel** are specific to this specific device. They define the protocol that the device uses to retrieve images, the video format and the channel where the frame grabber is streaming information from.

In table A.1 the characteristics of the device are shown.

Table A.1: Characteristics of the DFG/USB2propcb frame grabber.

Parameters	Values
Video formats @ frame rate (maximum)	PAL: 768×576 (0.4 MP) @ 25 fps NTSC: 640×480 (0.3 MP) @ 30 fps
Video standards	PAL/NTSC, RS-170/CCIR
Analog inputs	RCA (Cinch): 1 input Y/C (S-Video): 1 input Multiplexed
Bus interface	USB 2.0, Type B
Color formats	UYVY, Y800
Square pixels	yes

The **ClipRectangleOrigin** and the **ClipRectangleSize** parameters allow us to cut any portion of the image in real time that does not contain information pertinent to the application.

The **OutputChannel** is a internal component of plus that allows different modules to use the images retrieved from the hardware for different purposes, be it, storing directly the information into a file, or stream the data in real time to a visualization platform.

The second device that needs to be configured is the LBR Med. Since the plus library is not compatible with the LBR Med, the pose of the robot must be sent to plus through the OpenIGTLink protocol. To stream the data a server must be running inside the robot's controller, so that when the data capture beggins the library can establish communication via the so called client-server frame work.

The following code shows the configuration of this device.

```
<Device
    Id="ProbeToReference"
    Type="OpenIGTLinkTracker"
    MessageType="TRANSFORM"
    ToolReferenceFrame="Reference"
    ServerAddress="172.31.1.147"
    ServerPort="30000"
    IgtlMessageCrcCheckEnabled="true"
    AcquisitionRate="20"
    UseReceivedTimestamps="false"
    SendTimeoutSec="0.2"
    UseLastTransformsOnReceiveTimeout="true"
    ReconnectOnReceiveTimeout="true"
    ReceiveTimeoutSec ="0.5">
    <DataSources>
        <DataSource Type="Tool" Id="Probe" />
    </DataSources>
    <OutputChannels>
        <OutputChannel Id="ProbeStream">
            <DataSource Id="Probe" />
    </OutputChannel>
    </OutputChannels>
</Device>
```

The **Id** of the device uniquely identifies the device in the network and it must this string be equal to the name of the device packed in the transform message sent by the LBR Med. The **Type** parameters tells plus that this device tracks the US image in real time.

The **ToolReferenceFrame** defines the reference frame that the transformation refers to.

The **ServerAddress** and **ServerPort** define the server IP and port running inside the LBR software.

The **DataSource** component defines the frame that the transform represents. The last relevant software component is the **OutputChannel** which, like the previous device, creates a data stream that the plus library can use for different purposes.

Now the two data streams, the image and the transform stream, must be fused so that the actual position of the US image is known the the required frame of reference. The device required for this fusion is called a *VirtualMixer* and the code to create this virtual component is shown next.

```
<Device
    Id="TrackedVideoDevice"
    Type="VirtualMixer" >
    <InputChannels>
        <InputChannel Id="VideoStream" />
        <InputChannel Id="ProbeStream" />
    </InputChannels>
    <OutputChannels>
        <OutputChannel Id="ProbeVideoStream"/>
    </OutputChannels>
</Device>
```

This device takes the output streams of the imaging device and the tracker as input channels and defines an output channel with the fused data.

If the reader is attentive, there is still one missing information required to perform this data fusion. To obtain the image frame in the reference frame the transformations shown in equation A.1 is necessary to the set up and it has yet to be defined.

$$T^{Reference}_{Image} = T^{Probe}_{Reference}.T^{Probe}_{Image} \tag{A.1}$$

The transformation $T_{Reference}^{Probe}$ is provided by the robot but the transformation T_{Image}^{Probe} is static. If the probe is rigidly attached to the flange of the robot then, it must be obtained experimentally.

After the static transformation is known the reader needs to define the transformation in plus as it is done in the following code:

```
<CoordinateDefinitions>
  <Transform From="Image" To="Probe"
    Matrix="
      0.2 0.0 0.0 123.0
      0.0 0.2 0.0 −23.4
      0.0 0.0 0.2 1203.4
      0 0 0 1" />
</CoordinateDefinitions>
```

Notice that in the previous transformation the rotation's determinant is not equal to 1 because the scaling factor needed to resize the US image is embedded in this rotation as a multiplication by the matrix seen in equation A.2.

$$R = \begin{pmatrix} r_{xx} & 0 & 0 \\ 0 & r_{yy} & 0 \\ 0 & 0 & r_{zz} \end{pmatrix} \tag{A.2}$$

Where r_{xx},r_{yy} and r_{zz} represent the scaling factors in the x, y and z directions respectively.

With all the data fusion performed it is now time to stream the information to the visualization platform by creating a server that can send OpenIGTLink messages with the transformed US image as it is done in the following code

```
<PlusOpenIGTLinkServer
  MaxNumberOfIgtlMessagesToSend="1"
  MaxTimeSpentWithProcessingMs="50"
  ListeningPort="18944"
  SendValidTransformsOnly="true"
  OutputChannelId="ProbeVideoStream" >
  <DefaultClientInfo>
    <MessageTypes>
      <Message Type="IMAGE" />
    </MessageTypes>
    <ImageNames>
      <Image Name="Image" EmbeddedTransformToFrame="Reference" />
    </ImageNames>
  </DefaultClientInfo>
</PlusOpenIGTLinkServer>
```

This component requires that we define the port that the plus server uses to stream information, e.i **ListeningPort**. It requires that we defined the data stream that we wish to send to the visualization platform, in this case we want to send the fused data so we need to used the output channel *ProbeVideoStream* defined in the previous **VirtualMixer** device.

Notice that we define the frame of reference that we wish to show our image in, in this example called *Reference* frame in the **ImageNames** property under the **EmbeddedTransformToFrame** parameter.

Now we wish to tell plus how to reconstruct the volumetric information from the transformed US images. To do this we need do add an extra software device with all the parameters required to performe this reconstruction.

A typical **VirtualVolumeReconstructor** software component is shown in the following code:

```
<Device
    Id="VolumeReconstructorDevice"
    Type="VirtualVolumeReconstructor">
    <InputChannels>
        <InputChannel Id="ProbeVideoStream" />
    </InputChannels>
    <VolumeReconstruction
        ImageCoordinateFrame="Image" ReferenceCoordinateFrame="Reference"
        Interpolation="LINEAR" Optimization="FULL" Compounding="On" FillHoles="Off" NumberOfThreads="2"
        OutputSpacing="0.5 0.5 0.5" >
        <HoleFilling>
            <HoleFillingElement Type="GAUSSIAN" Size="5" Stdev="0.6667" MinimumKnownVoxelsRatio="
                0.50001" />
            <HoleFillingElement Type="STICK" StickLengthLimit="9" NumberOfSticksToUse="1" />
        </HoleFilling>
    </VolumeReconstruction>
</Device>
```

This device has a large array of option that affect the properties of the final volume. Although these parameters are of the utmost importance, their affect are outside the scope of this appendix.