Robotic navigation system for needle positioning in neurosurgery

João Paulo Mendes Oliveira
joao.m.oliveira@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

November 2019

Abstract

Several surgical navigation platforms are available on the market to allow surgical teams to use the full capabilities of their medical devices. Most navigation platforms suffer from lack of access to their software development kits which in turn impede rapid prototyping and new additions to the surgical workflow. This paper focuses on the integration of a robotic system, the LBR Med, with the known plus toolkit as a data collector, and 3D Slicer as a visualization tool to create a functional neurosurgical navigation platform that provides horizontal mobility to the system. To communicate with the robot's controller, which uses the Java virtual machine, the OpenIGTLink protocol was implemented in Java. With the reliance in open source software, we hope to develop a set up which can operate not only in a neurosurgical context, but also in a varied array of operating conditions, be it pedicle screw placement, neurosurgical biopsies among others.

The critical problem of registration between the patient's coordinates and the robot's coordinates is explored through the use of the ultrasound imaging modality to extract information from the robot's environment in its frame of reference. To extract meaningful information from the environment several tests were performed to place lower and higher bounds on the characteristics of the velocity profile of the robot and the communication structure between the navigation platform and the LBR Med controller.

Keywords: LBR Med, Plus toolkit, Java, 3D Slicer, OpenIGTLink

1. Introduction

With the advent of robotic systems into the operating room in 1970 the neurosurgical field has searched for a robotic system which integrates itself in the workflow of a neurosurgical procedure without introducing more complexity to an already complex intervention.

This work bases itself in the thesis that instead of developing different robotics systems for different operating conditions one can develop a robot which can restrict the end effector movements through the use of complex control schemes, thus increasing the potential areas of applicability of the robotic system. This shift from hardware to software can liberate development teams by allowing them to focus their attention on other key components in their medical solutions, instead of wasting their time on adapting their software to different robotic systems that operate in niche conditions.

Several navigation platforms have been developed in the last decade, among them are: the image guided platform Ibis, that focuses on neurosurgical procedures and it is built with open source software, that can be freely changed by the research community; the CustusX is an image guided navigation platform that shares a similar philosophy in its development to the Ibis platform; the 3D Slicer navigation platform enjoys a differentiating characteristic that makes it stand out in relation to its competitors, namely the flexibility of the underlying software, allowing developers to integrate different software components into the set up they desire to build.

One of the characteristic problems associated with neurosurgery is the registration between different frames of reference. Since 1970, with the advent of computer tomography (CT), neurosurgeons have realized that minimal evasive procedures represent the future of the field. The added complexity of this technique lies in the inherent reliance on information present in the image space, defined by a frame of reference that characterizes every voxel present in that image space. The process to determine the mapping between any given image space and the physical coordinates associated with the patient in the operating room is called registration.

In an operating room a robot is only useful if a mapping between the frame of reference of the robot and the reference frame associated with the patient's skull exists. Several solutions to obtain this mapping exist, such as: a) use markers that the robot can identify in its coordinate system; b) use an external imaging system to identify the tools in the OR and provide feedback to the robot in real time; c) use the ultrasound imaging modality to retrieve identifiable structures (surface of the patients skull) in the robot's coordinate frame.

All the possible solutions to the registration problem, require a two way communication channel between the robot and the main workstation.

In the work developed by Tauscher et al. [5], using the LBR Med robot, the communication problem was tackled in the context of a telesurgical system, as defined by Narendra Nathoo [3]. To control the robot in real time Tauscher used 3D Slicer as both the data collector and the visualization platform. To establish a communication channel between the robot and the main workstation the OpenIGTLink protocol was used as an interface between the two devices. Currently, the OpenIGTLink protocol is only implemented in C/C++ [6], which in turn makes its use by the LBR Med more complex because the robot controller uses the Java virtual machine to interact with the robot. Tauscher identified two possible solutions to this incompatibility, either use the C++ implementation of the protocol through the Java native interface (JNI) to call compiled machine code or develop a full implementation of the protocol in Java.

The communication structure developed by Tauscher is well designed and possesses strong embedded security checks which are positive characteristics of any communication design. The problem with his implementation is the reliance on complex interactions between all the classes used by his library, thus reducing the possible number of application of the library by the medical community.

To replace the communication structure proposed by Tauscher we can look at the set up proposed by Seitz [4] where the LBR Med was integrated with the medical imaging interaction toolkit (MITK) to create a fully functional navigation platform where the Java implementation of the OpenIGTLink protocol, maintained by the WIP lab, serves as an interface between the robot and the main workstation. Although the set up developed by Seitz enjoys more flexibility than the set up proposed by Tauscher, due to the full Java implementation of OpenIGTLink, it still lacks horizontal mobility due to the use of MITK as the data collector between all the intervening devices. The lack of mobility comes from the dependence between the visualization module and the data collector module which forces developers to implement software in C++ to communicate with new additions to the set up when changing the operating conditions of the robot, for example, changing the set up from a neurosurgical intervention to a laparoscopic intervention. To develop an architecture that increases the potential operating conditions of the LBR Med we can look at the public software library for ultrasound (Plus) toolkit developed by Lasso et al. [2]. The Plus toolkit library is an open source library that allows any developer to create a navigation system where the visualization module can be separated from the data collection module. This modularity by itself is not newsworthy, since Seitz uses the same logic in his MITK module. The characteristic of the plus library that increases the horizontal mobility of any set up which uses it, is the configuration file, written as a xml file, that allows a developer to add any device configured in the library to the set up without the need to create new communication structures.

This paper focuses on the development of a navigation platform that uses the LBR Med to perform the surgical procedure, in a shared control structure [3]. To provide horizontal mobility to the set up the robot is integrated with the visualization module through the plus library. To communicate with the plus toolkit the robot uses a new implementation of the OpenIGTLink protocol in Java, with significant improvements compared to other Java implementations, to serve as an interface between the robot and plus. The mapping between robot coordinates and the patient coordinates is performed through MRI-US registration. Specifically by using the robot as a tracking device it is possible to reconstruct the structures scanned by the US probe into a volume, where surfaces can then be extracted and mapped into the MRI volume, thus creating a link between the robots coordinates the the MRI space.

2. Methodology

2.1. LBR Med

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. The robot can detect impacts performed by its surroundings, through the torque sensors integrated in all its seven joints, thus giving the LBR Med human robot collaboration capabilities.

The LBR Med is programmed through the Workbench API which allows developers to interact with the different hardware components through several Java libraries developed by KUKA. To give the LBR flexibility and increase the potential applications of the robot several libraries exist that implement different control laws on the robot's controller such as: a) the trocar handguiding library, that restricts the motion of the LBR Med end effector through a single point in space; b) the direct and smart servo motion libraries which allow the developers to correct the robot's trajectory in real time, allowing the robot to be used as a telesurgical system or use it in the context of visual servoing applications; c) the manual guidance library which allows the robot to be hand guided by the surgeon in its environment.

To reduce the effort required from each development team to meet regulatory standards, the robot was developed in accordance with the ECEE CB Scheme. In simple terms, the robot's controller is divided into two main components: the safety controller and the motion controller. Both of these components are classified as a class C type software in accordance with the IEC 62304 norm. Under the IEC 62304 norm a class A type software entails minimum risks for the patient; class B type software entails injury risks for the patient and class C type software poses significant injury and death risks to the patient. The important characteristic designed by KUKA is that the developers code is used by the motion controller and the safety controller serves as an observer to correct and react to any error detected in the motion controller. With this observer architecture it is possible to reduce the risk level of the code implemented by the developer to level B, thus reducing the effort required by the developer to verify the software integrity under the broad possible situations that the robot can face in an operating room (OR).

2.1.1 OpenIGTLink in Java

To establish a communication channel between the robot controller and the main workstation a Java implementation of the OpenIGTLink protocol was developed. Unlike the implementation maintained by the WIP Lab, which was built on top of a complex socket logic, the implementation we developed relies on a simpler socket logic, with several tutorials, which allow developers to increase the steepness of their learning curve while using the library.

The Java language was developed to tackle specific problems with web applications. This focus translates into certain characteristics such as: the fixed allocation of memory for primitive variables, such as integers, doubles and longs, with fixed byte array sizes of 16, 64 and 64 bytes, respectively; private access to class methods, that increase the security of the language.

These design choices that underline the Java language require that any byte manipulation, required by the OpenIGTLink protocol, to use external classes that add said functionality. The two utility classes implemented in the protocol are:

1. ByteArray - the class provides methods to both encode and decode java variables as byte arrays with variable sizes, which can be defined by the developer and it allows other classes to access the byte arrays as necessary by the library.
2. Header- the class allows the library to encode and decode the byte array attached to the beginning of each message in an efficient manner. This class allows the library to reduce its latency, thus increasing the possible application of the protocol.

The current Java implementation of the protocol has all the message types of OpenIGTLink, which have been tested against the C++ implementation of the protocol.

Another important characteristic of the protocol is the ability to encode new message types, as the need emerges by the developer. In order to do so the developer needs to extend the abstract class OpenIGTMessage which provides low level methods that must be common to all messages implemented in the library.

The socket logic of the protocol relies on two classes, shown in figures 1 and 2.

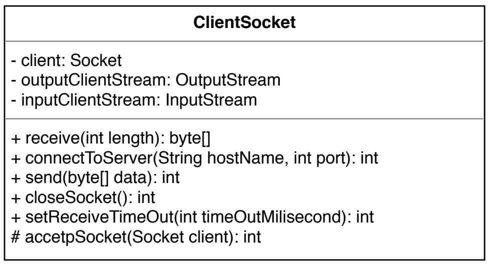

ClientSocket
- client: Socket
- outputClientStream: OutputStream
- inputClientStream: InputStream
+ receive(int length): byte[]
+ connectToServer(String hostName, int port): int
+ send(byte[] data): int
+ closeSocket(): int
+ setReceiveTimeOut(int timeOutMilisecond): int
# accetpSocket(Socket client): int

Figure 1: Class diagram of the simple socket logic used by the Java implementation of the OpenIGTLink protocol.

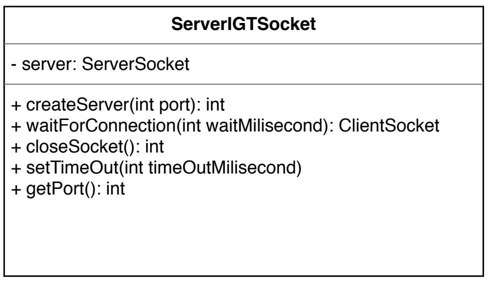

ServerIGTSocket
- server: ServerSocket
+ createServer(int port): int
+ waitForConnection(int waitMilisecond): ClientSocket
+ closeSocket(): int
+ setTimeOut(int timeOutMilisecond)
+ getPort(): int

Figure 2: Class diagram of the simple socket logic used by the Java implementation of the OpenIGTLink protocol.

The developer can connect the workstation to another node in the network by setting up a server connection which returns a client connection so that the workstation can send and read bytes from the client socket by calling the *receive()* method.

2.1.2 Observer architecture

To provide horizontal mobility to the set up, the robot's software must be: a) modular - so that future functionality can be added to the system without the need to rewrite the code for the entire application b) fast- so that transitions from one robot state to another happen in a small time windows c) safe- the transitions from one state to another must dealt with unexpected situations imposed by dynamics of the environment which were not predicted.

All the previous features can be embedded in the architecture of the software by adopting the observer pattern commonly used by Java developers. The pattern uses observers to react to changes in states of the system. The architecture of the software running inside the LBR Med controller is shown in figure 3.

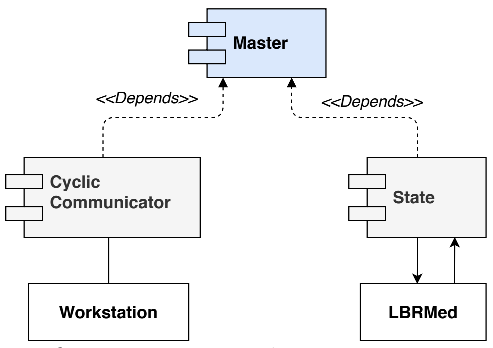


Figure 3: Component Diagram of the classes that control the internal responses of the robot.

The set up divides the communication with the external workstation from the motion characteristics of the robot. The master state controls both the communication class and it handles the state changes of the system. In this context a state is defined according to the stage of the surgical intervention.

The workflow of the surgical procedure follows the following stages:

1. Path planning - The surgical team defines the target in the MRI imaging space and the path to reach said point.

2. US scanning - The surgeon scans the patient skull with an US probe and uses the robot as a tracker device to reconstruct the 2D structure seen in the US images as a 3D structure.

3. Registration - In order to know the rigid transformation between the robot coordinates and the MRI space we can map a surface extracted from the US reconstructed volume to the MRI surface.

4. Surgeon guidance - The surgeon guides the robot under impedance control along the desired trajectory.

5. Surgical procedure - The surgeon performs the surgical procedure with the robot under position control in a fixed point in space.

6. Stop robot motion - When the surgery is finished the robot is moved under manual guidance to a safe position and the robot motion is stopped.

To map the surgical workflow into meaningful robot states the following mapping is proposed:

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

2. Ultrasound registration - The second state uses hand guidance support provided by KUKA to track the rigidly attached US probe while the surgeon scans the patient.

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 robot moves along the desired trajectory under impedance control guided by the surgeon. Once the target is reached to robot is commanded to hold its position.

5. Termination - Once the surgical procedure is over the surgeon safely moves away the robot from the patient and terminates the robot's application.

2.1.3 Virtual impedance environment

All the robot states have an obvious implementation with the exception of the *Impedance Control* state. Because the robot interacts with the surgeons it needs to follow the desired trajectory under impedance control. The robot should not pull the end effector towards the patient, since it should be guided by the surgeon along the desired trajectory. The solution proposed by the authors relies on sending a commanded position to the robot, also called equilibrium position, and force the robot to mimic a second order system that does not posses stiffness along the desired trajectory.

To translate the previous idea into an actual control strategy two hypothesis are proposed, they are: a) An impedance cone that increases the the stiffness imposed on the end effector as the surgeon

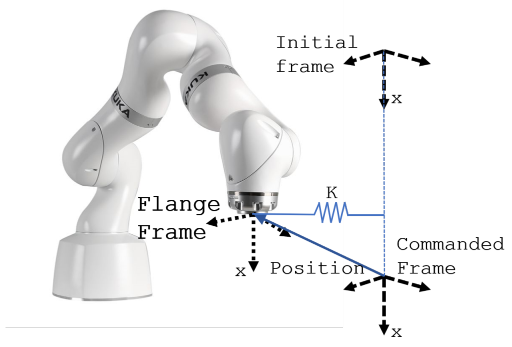


Figure 4: Objects used by the robot to create the impedance control environment.

approaches the target in the perpendicular directions to the motion; b) an impedance line that has a constant stiffness along the desired trajectory.

The impedance line strategy uses the target frame, *Commanded Frame*, shown in figure 4 as the commanded position to reach. The LBR Med performs the required computation to mimic the second order system autonomously through its impedance controller.

To force the robot to stop once the Commanded Frame has been reached one can call the method *changeControlModeSettings()* that allows the developer to change the stiffness of the robot in real time. As soon as the commanded position is reached the robot increases the stiffness in the parallel direction to the desired motion until it reaches the maximum allowable stiffness of 5000 N/m.

The impedance cone strategy relies on the same logic of the impedance line strategy with the simple difference that the stiffness in the perpendicular directions to the desired trajectory are updated as a function of the distance to the commanded frame in real time.

2.2. Analogic ultrasound and frame grabber

The US unit used to provide the US images from the patient's skull is the prosound unit. The prosound device can extract B-mode images from its transducer. Because we need the US images and the device is analogic a frame grabber is required to extract them. Because the Plus library serves as the connecting node between all the devices the frame grabber chosen to be part of the final set up is the DFG/USB2propcd. It can extract images up to 30 frames per second, depending on the protocol used to stream the images, and with a maximum image resolution of 720/576 pixels.

2.3. Visualization module

To visualize data in real time, 3D Slicer was chosen as the visualization platform of the set up. Given the added flexibility of the GUI in comparison to the other commercially available options and the embedded compatibility of the platform with the plus library it became the obvious choice.

With the support already embedded in the platform developers can: a) record data - so that the US tracking by the robot can be evaluated at a latter date, or other tests related to the set up; b) Offline reconstruction - with previously recorded files we can reconstruct volumes from the US tracked images; c) Scout scan - To identify the structure which we want to scan with high definition we can first perform a scout scan to define a region of interest (ROI); d) Live reconstrcution - After the ROI has been identified we can now reconstruct the volumetric information with the highest possible resolution.

2.4. Optical Tracker

To obtain the US image in the reference frame of the robot, lets call it the *world frame*, the US image must be positioned in space by the transformations seen in equation 1.

$$T_{World}^{Image} = T_{World}^{Flange} . T_{Flange}^{Image} \tag{1}$$

The robot is calibrated to obtain the T_{World}^{Flange} transformation directly from the controller, but to obtain the T_{Flange}^{Image} transformation, it is necessary to use some calibration algorithm to obtain the necessary rigid transformation.

To perform said calibration algorithm the Polaris tracker is required, seen in figure 5, to obtain from an external device the mapping from the US image frame to the flange frame of the robot.

Figure 5: Polaris device used in the set up to perform the needed spacial calibrations.

2.5. Set up integration

With all the previous devices, the architecture that connects all of them into a single set up can be seen in figure 6. Plus serves as the connection node where all devices must go through it.

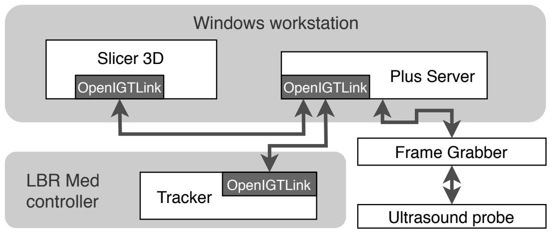


Figure 6: Interaction of the devices used in this paper.

Plus assumes that between the intervening data streams only exists a constant delay. This assumption ignores that the ultrasound captures a vertical line of pixels from each image at a constant frequency, and one can compute the velocity of the line in the physical space.If the velocity of the probe is comparable to the velocity of the probe, then the final reconstructed volume will show deformations associated with 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

3. Results & discussion

3.1. Performance of OpenIGTLink implementation

Given that the Java implementation is new, we cannot guarantee that it will enjoy the same performance characteristics as its C++ counterpart implementation. Although the implementation should be tested for all potential uses, that work will be left for future developers for two reasons: a) windows is not a real time capable system, which means that time measurements are not precise to less than ms measurements; b) we require a java implementation of the network time protocol which is still not completed. Instead of validating the protocol we choose to validate its performance in the context of our set up. This in turn means that any deviation from this set up requires that the performance of the protocol be verified again.

3.1.1 Latency measurement

The latency of a protocol, as defined by Tokuda et al. [6], is the delay between the packing of the message in the sender, call it instant t_0, and the unpacking of the encoded message by the receiver, call it instant t_1.

Given that there are only two workstations in our set up, the latency was measured only between two workstations. The results of the measurements are shown in table .

Table 1: Latency of the OpenIGTLink Java implementation

frame rate (fps)	latency (ms)
199044	0.00452

To obtain time measurements from inside the Java virtual machine the *System.nanotime()* method was used.

3.1.2 Jitter measurement

Jitter is defined according to RFC 4689 as the absolute difference between the measured delay of two adjacent packets of data, defined as the difference between the time instant $|D_i - D_{i+1}|$ where D_i is the instant when the packet i is received and D_{i+1} is the instant when the packet $i{+}1$ is received. The average jitter is defined as the average of the instantaneous jitters measured during a interval of time, Δt.

Since we do not know if the jitter of the connection is related to the frame rate of the communication between the robot and the main workstation figure 7 shows the measurements of the jitter versus the frame rate.

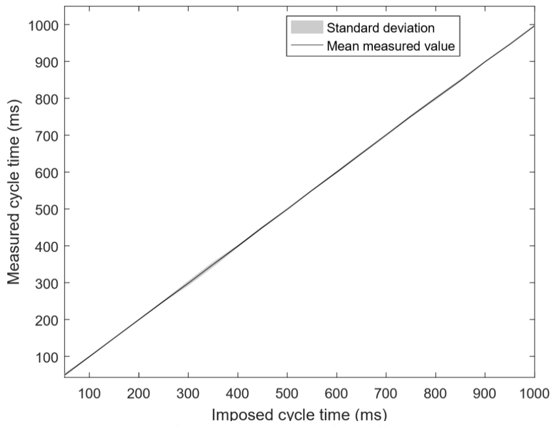


Figure 7: Jitter of the connection for different cycle times

3.1.3 Discussion

As it can be seen in table 1 the latency of the implementation in Java is small, when compared in absolute terms with the C++ implementation of the protocol. Although the conditions in which they were tested are different this tells us that the protocol is efficient. The jitter measurements, seen in figure 7, suggest that the jitter is independent from the frame rate, which indicates that, except with some constraints, the developer can choose the frame rate that best suits their needs.

3.2. Control strategies

The control strategies proposed in section 2.1.3 can become unviable under two situations: a) if the update rate at which the developer can send the stiffness to the controller is too low the surgeon will feel sudden variations in the force exerted by the flange of the robot; b) because the robots position and the robots commanded position is different, the robot can map the force required to mimic a second order system in the Cartesian space with distortions when mapped to the joint space of the robot, thus effectively the system that the robot mimics is not of second order system.

3.2.1 Update rate

The cycle required to update the stiffness of the robot in real time is shown in figure 8. There are three relevant variables that interfere with the stiffness update and they are: the time taken by the *changeControlModeSetting()* method, called ti_{cm}; the time taken by the *updateWithRealTimeSystem()* method which updates the values of position, and force felt by the robot, identified by ti_{urts}; the computation time required to compute the desired stiffness value. These three variables impact the total cycle time taken to update the stiffness of the robot, called t_{ext}.

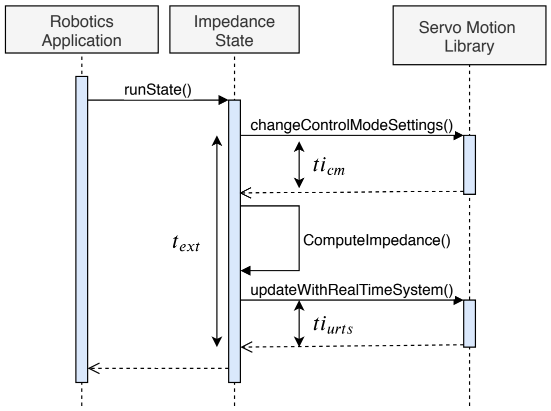


Figure 8: Sequence diagram that shown all the relevant elapsed times in the stiffness update cycle.

To evaluate if the time taken by the *changeControlModeSetting()* method is influenced by the number of calls made to it table 2 shows the average duration and standard deviation of the $\overline{ti}_{cm}$ variable.

Table 2: Duration of the changeControlModeSetting() method

N° cycles	$\overline{ti}_{cm}\ ms$	$std(ti_{cm})\ ms$
15000	0.9837	0.2385
50000	0.9822	0.2356
150000	0.9800	0.2269
450000	0.9900	0.2200

In table 3 the same data is showed about the ti_{urls}, the time taken by the call to the *updateWithRealTimeSystem()* method, under different consecutive calls to the method.

Table 3: Duration of the updateWithRealTimeSystem() method

N° cycles	$\overline{ti}_{cm}\ ms$	$std(ti_{cm})\ ms$
15000	1.0463	0.2802
50000	1.0489	0.3047
150000	1.0500	0.3180
450000	1.0500	0.3500

The total duration of the *runState()* method call is shown in table 4. To evaluate the duration of the method in a reliable way the class *StatisticTimer* provided by KUKA, was used.

3.2.2 Discussion

Given that no visible correlation exists between the number of calls and the time taken by the methods to update the stiffness of the robot, as seen in tables 2 and 3, then the control strategies developed to control the LBR Med are valid and they can operate for undetermined amounts of time. Given the total time taken by the cycle to compute and update the stiffness, as seen in table 4, we can conclude that the control strategies can be classified as a soft real time application.

Table 4: Total cycle time needed to update the desired stiffness.

N° cycles	min	max	mean	SD
874538	1.74	127.03	2.06	0.42

3.2.3 Stiffness validation

According to the data reported in the previous section the robot is able to update the desired stiffness, on average, in 2.06 ms but the actual stiffness simulated by the torque joints can be distorted due to two hypothesis:

a) Geometric error - if the error between the commanded position and real position of the robot is significant, the difference in the manipulator configuration can deform the Jacobian used to map the stiffness matrix in Cartesian space to joint space.

b) Joint Stiffness - depending on the commanded stiffness the simulated force might not overcome the effects of joint friction

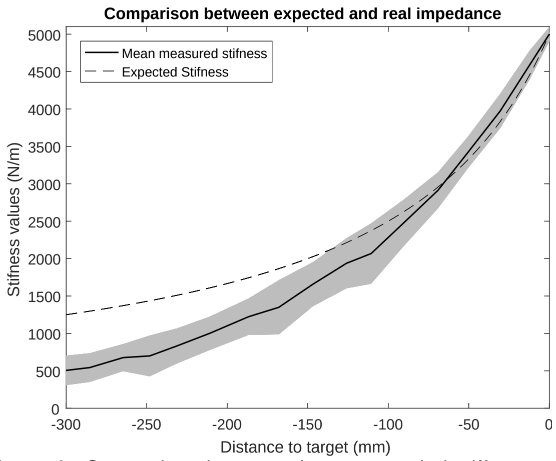


Figure 9: Comparison between the commanded stiffness and estimated stiffness along the trajectory.

To test these effects produced by the controller, one can perform a static test, thus eliminating the effects of the acceleration and velocity, leaving only the effect of the stiffness mimicked by the controller. In figure 9 the static test is shown along the desired trajectory.

3.2.4 Discussion

As seen in figure 9, there is a difference between the desired stiffness and the actual mimicked stiffness performed by the robot. To test the hypothesis proposed in a) and b) two test were performed on the robot. The first test requires that the robot be commanded with high stiffness values, that can surpass the friction of the joints, and the distance to the equilibrium point is increased, thus changing the Jacobian. Because the mapping error increases, this means that the difference between the commanded stiffness and the mimicked stiffness is justified by geometrical distortions.

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 20ms 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 22.06 ms which results in sudden changes in the force done by the flange of the robot on the surgeon's hand.

3.3. Spacial calibration

To obtain the transformation mentioned in equation 1, T_{Image}^{Flange}, we can position the stylus in a water tank, which perturbs the US image. The position of the stylus can now be measured by two paths. To understand which paths, the reader can look at figure 10.

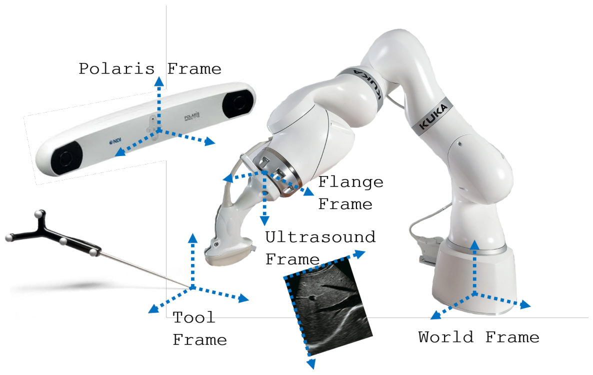


Figure 10: Frames of reference required to perform the spacial calibration between the US frame and the Flange frame

The position of the stylus tip can be measured by the Polaris tracking system and by the robot tracked US images, as it is written in equation 2.

$$p_{Tool}^{World} = T_{Flange}^{World} \cdot T_{Ultrasound}^{Flange} \cdot p_{Tool}^{Ultrasound} = T_{Polaris}^{World} \cdot p_{Tool}^{Polaris} \tag{2}$$

By isolating the transform $T_{Ultrasound}^{Flange}$ equation 2 should be algebraically manipulated to obtain equation 3.

$$T_{Ultrasound}^{Flange} \cdot p_{Tool}^{Ultrasound} = T_{World}^{Flange} \cdot T_{Polaris}^{World} \cdot p_{Tool}^{Polaris} \tag{3}$$

By recording a point cloud we can uncover the transform $T_{Ultrasound}^{Flange}$ using the algorithm proposed by Arun et al. [1] which uses the SVD decomposition to obtain the rotation and translation from one point cloud to another. To evaluate the error of the calibration we can check the position of the stylus, and the perturbation caused by the stylus tip on the US image and measure the error between the two points measured in the *World frame*. This validation is shown in figure 11.

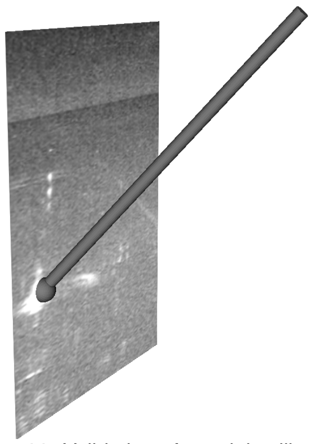


Figure 11: Validation of spacial calibration.

The average of the error measured with this method is shown in table 5.

Table 5: 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

3.3.1 Discussion

The average error of the spacial calibration algorithm of 1.2 mm, as seen in table 5, 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. If on the other hand, the error is unacceptable, several additional methods have been proposed that might bring the positioning error to lower values [2].

3.4. Temporal calibration

Given that the tracking stream sent by the robot is independent from the imaging stream sent by the frame grabber, the computation time used by the two components to send messages is different, given that images are larger in size than transform messages. This computation time can be approximated by a constant offset[]. To estimate this constant offset the US probe can be used to scan the bottom of a water tank. By estimating the position of the bottom from the US image while imposing on the US probe a sinusoidal force we can measure the difference between the two signals. In figure 12 the two uncalibrated signals are shown.

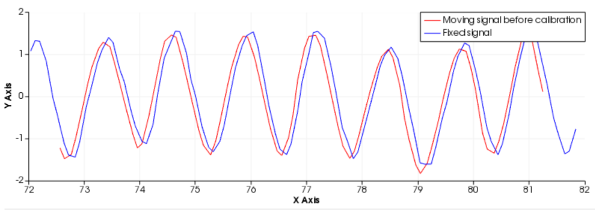


Figure 12: Uncalibrated signals obtain from the US image stream (fixed signal) and the tracking signal(moving signal).

By measuring the correlation between the two signals it is possible to estimate the delay which superimposes them. For this specific set up, with a delay of 80 ms we obtain the overlap between the two signals shown in figure 13.

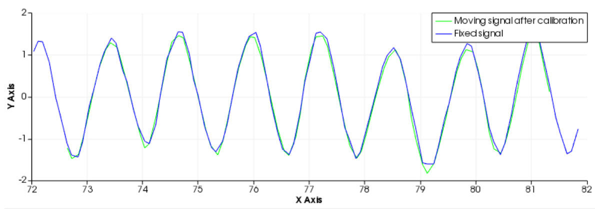


Figure 13: Calibrated signals after the correction applied through the temporal calibration.

To test the temporal calibration we designed a simple experiment where the stylus previously mentioned was placed in the water tank, and the same sinusoidal signal was imposed on the US probe. If the calibration corrects the error between the image stream and the tracking stream the stylus perturbation on the US image should remain static throughout the duration of the sinusoidal movement. The results are shown in table 6.

Table 6: Temporal Calibration error

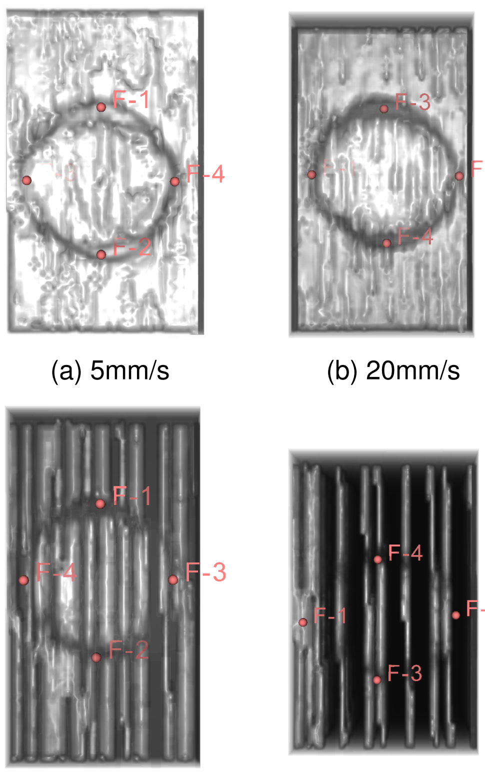
	μ	sd	max
Calibrated	1.53 mm	0.73 mm	2.73 mm
Uncalibrated	3.81 mm	2.36 mm	8.01 mm

3.4.1 Discussion

With the temporal calibration algorithm used to calibrate the set up we obtained a reduction of 60%, as seen in table 6, 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, contrary to the maximum deviation of 2.73 mm when the set up is temporaly calibrated. This represents a significant reduction, thus proving the hypothesis assumed by the Plus library. If the US probe is only subjected to low velocity values, then this apparent movement can be reduced to even lower values.

3.5. Volume reconstruction limitations

To demonstrate capabilities of the volume reconstruction algorithm embedded in the Plus library and test the limitations of the tracking system, so that higher bounds can be placed on the robot's path characteristics when doing the volume reconstruction, figure 14 shows the reconstruction of a coin placed under a silicon plate with different probe velocities.



(a) 5mm/s (b) 20mm/s (c) 40mm/s (d) 80mm/s

Figure 14: Top view of the reconstructed coin under different probe velocities.

To verify if the probe velocity deforms the shape of the probed coin we can measure the diameter of the coin in the perpendicular direction to the motion of the probe and compare it to the diameter of the coin parallel to the direction of the probe. The results are shown in table 7.

Table 7: Results of the total cycle time needed to update the stiffness value at the joint level.

Speed mm/s	d_1 mm	d_2 mm
5	31.0097	32.1972
20	30.1727	32.8800
40	29.4824	30.6348
80	28.6201	36.892

3.5.1 Discussion

The first apparent distortion in the reconstructed volume, seen in figure 14, are the increase in the 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 of80 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 7 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 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 we found is that the assumption of the constant delay between the tracker and the frame grabber breaks down at high probe velocities.

4. Conclusions

In this paper we were able to create a set up which is modular, and relies on public libraries, which in theory should create a robotic system with large horizontal mobility. By developing the implementation of the OpenIGTLink protocol in Java we were able to increase the potential areas of applicability of the LBR Med.

References

[1] K.S. Arun, T.S. Huang, and Steven 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.

[2] 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.

[3] Michael A. Vogelbaum David M. Peereboom Narendra Nathoo, Murat Cenk Cavusoglu. In touch with robotics: neurosurgery for the future. *Neurosurgery*, 56 3:421–33; discussion 421–33, 2005.

[4] Peter Karl Seitz. *Konzeption und Entwicklung einer robotergestützten und ultraschallbasierten Lokalisationskontrolle*. Master thesis, 2018.

[5] Sebastian Tauscher, Junichi Tokuda, Günter Schreiber, Thomas Neff, Nobuhiko Hata, and Tobias 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`.

[6] Junichi Tokuda, Gregory S. Fischer, Xenophon Papademetris, Ziv Yaniv, Luis Ibanez, Patrick Cheng, Haiying Liu, Jack Blevins, Jumpei Arata, Alexandra J. Golby, Tina Kapur, Steve Pieper, Everette C. Burdette, Gabor Fichtinger, Clare M. Tempany, and Nobuhiko 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.