Augmented Reality in Teleoperated Human-Robot Interaction

Filippo Brizzi

Augmented Reality in Teleoperated Human-Robot Interaction

Perceptual Robotics Laboratory, Tecip

Scuola Superiore Sant'Anna

Author:

Filippo Brizzi [illegible]

Supervisor:

Emanuele Ruffaldi [illegible]

Ph.D Course

Emerging Digital Technologies

Academic Year

2016 / 2017

March 12th, 2018

ABSTRACT

Nowadays, robots are becoming more and more complex, providing increasingly functionalities, precision and effectiveness. Yet, the amount and variety of tasks that robots can accomplish autonomously is limited. For this reason, when dealing with complex tasks, usually robots are controlled by a human operator through teleoperation. Teleoperation allows humans to perform tasks in environments that are harsh or not accessible and to reduce travel needs as in the case of telemedicine. The downside of teleoperation comes from a reduced perceptual capability and the control of a robot that has a different kinematic structure or capabilities than the human, requiring specific training.

To overcome these difficulties it is possible to employ Mixed and Augmented Reality display and feedback. The idea is to overlay on the visual feedback received from the remote robot virtual cues providing information about the remote environment or about the task in execution, which are not directly inferable from the video. Virtually generated features can be used not only during the actual task execution, but also for training purposes. In order to recreate a specific scenario scientist and engineers started to create Virtual Environments (VE) that are replicas of real world scenarios. In this case the operator controls a simulated version of the target robot. This thesis describes a set of software approaches and design solutions finalized at the creation of integrated real-time AR/VR applications, with a focus on fast prototyping and reconfigurability.

The first part of the thesis presents the state of the art software solutions created to provide tools that ease the development of AR/VR applications. The software is described and presented along with real examples. The second part, instead, focuses on the study and analysis of the impact of AR and VR in two specific scenarios: industry and medicine.

ACKNOWLEDGMENTS

It has been a wonderful journey and, as all the beautiful things, also this came to an end.

My deepest gratitude goes to Dr. Emanuele Ruffaldi who has guided me through this journey, and convinced me to start it in the first place. I have to thank him for sharing his knowledge and giving me the possibility to do amazing experiences.

Dr. Carlo Alberto Avizzano, who has always supported and encouraged me.

Dr. Alessandro Filippeschi, for always finding time (and patience) to help me with selfless dedication.

Giacomo Dabisias, faithful fellow since the first year of our University studies. We have always helped each other through the toughest moments, and shared the joy and the excitement of learning new things. I wouldn't be here if it wasn't for him.

Conte Lorenzo Landolfi, for his quiet wisdom.

Lorenzo Peppoloni, Massimo Satler, Paolo Tripicchio, Juan Manuel Giacinto Villegas and Alessandro Maddaloni, who contributed in creating a happy and pleasant environment in the laboratory.

The SSSUP canteen staff, who has always provided delicious meals, able to brighten even the gloomiest days.

All the guys at Meta Company, who gave me the possibility to spend six wonderful months in the Silicon Valley.

Most importantly, none of this would have been possible without the constant support of my family, with their love, trust and the wise advices.

Thanks to Anna-Lisa for her love, support and for always being by my side.

My last thanks goes to my grandparents. Even though you are no longer here, this thesis is also yours. For all my life you have taught me the love and passion for studying, the importance of self-commitment, and the joy that comes from hard work. This thesis is dedicated to you.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AR	Augmented Reality
AV	Augmented Virtuality
CV	Computer vision
MR	Mixed Reality
VR	Virtual Reality
VE	Virtual Environment
AI	Artificial Intelligence
SG	Scene Graph
DF	Data Flow
HI	Haptic Interface
HUD	Head-Up Display
HRI	Human-Robot Interaction
HMD	Head Mounted Disply
DOF	Degree of Freedom
DAG	Direct Acyclic Graph
SRG	Spatial Relationship Graph
USG	Ultrasonography
URDF	Universal Robot Description Format
DiagUI	Diagnostician User Interface

1 INTRODUCTION

Augmented Reality (AR) refers to the technique of overlaying, over a live direct or indirect view of the real environment, synthetic data. The purpose of AR can be of several types: recreational, for education, military, navigation or medical. AR's possible applications are theoretically endless. The only limit is the ability of generating fictional images to augment the reality.

The term Augmented Reality was coined in 1990 by researchers Tom Caudell and David Mizell while working at Boing. However, both the first idea of AR and its first implementations can be traced far backward in time.

The concept of AR is very fascinating as it allows to enhance the reality, composing any information or images. This has inspired several fictional writers and scientists. In 1901, L. Frank Baum, author of the Wizard of OZ, in the novel *The Master Key*, imagines the existence of spectacles that provide the capability to see the goodness or evilness of the other people, as a letter overlaid on their forehead. More than one hundred years after Baum's vision, Google presented the Google Glasses, giving to the people the possibility to experience what Baum predicted in his novel.

The history of AR as science starts during the Second World War when a Head-up Display (HUD) was mounted in the English military airplanes to help the pilots finding, in the sky around them, the objects detected by the radar (Figure 1.2). Aviations all over the world have kept investing more and more sophisticated HUDs, also for commercial planes, exploiting the benefits that AR can have in helping the pilots, both during training and real flights. Nowadays HUDs are becoming popular also in cars as a tool to show navigation directions and car status. The limitation of current HUDs is that they don't have any knowledge of what is happing on the other side of the screen, thus they cannot adapt their visual feedback to respond to environmental changes.

The first wearable device with AR capability is the EyeTap, invented in 1980 by Professor Steve Mann, see Figure 1.1. The EyeTap can not only be worn and carried around but, unlike the aforementioned HUD, it uses cameras to sense the world and adapt the visual feedback to respond to the

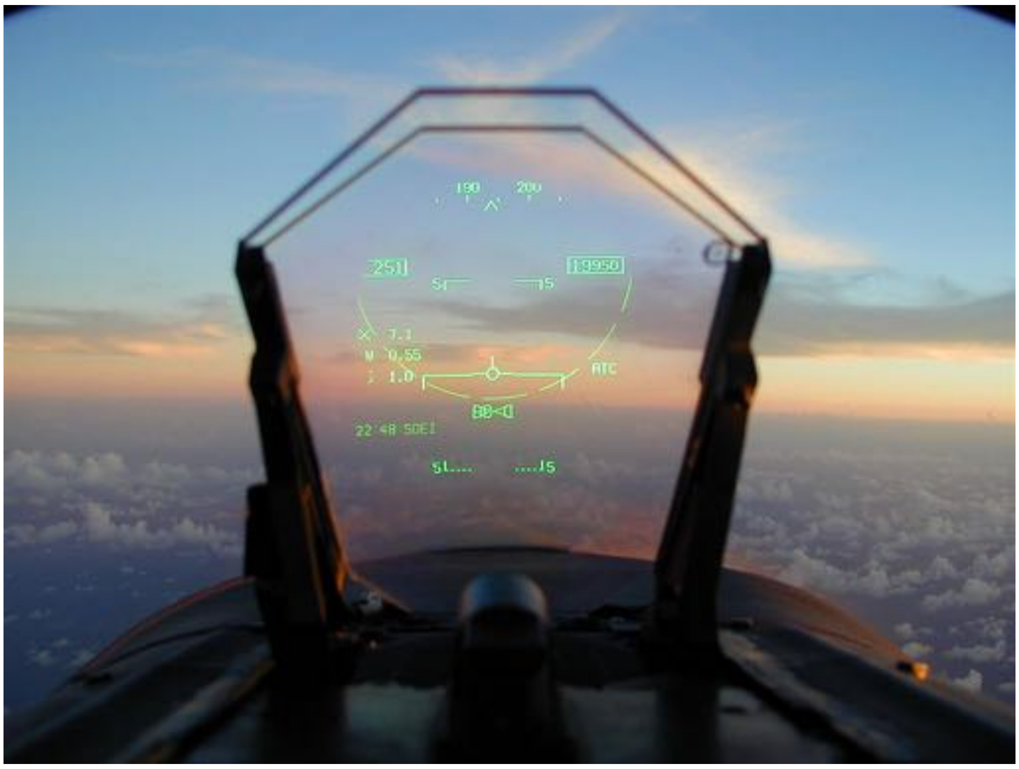


Figure 1.1: An example of a HUD mounted on a plane.

natural scene. EyeTap represents the first milestone in the creation of a complete AR capable device. In 1993, Steven Feiner introduced KARMA [45], an AR system that explains to end-users how to do simple maintenance of a laser printer using a novel see through head mounted display. This is one of the first examples of AR as a tool for supporting the execution of a given task. In 1999 Hirokazu Kato created ARToolKit [67], the first successful open-source library that helps in the creation of AR applications, providing support for rendering and tracking. This was an important step in the evolution of AR as it allowed developers and researchers to speed-up the development of AR applications and contributed to the creation of a shared and common codebase.

Researchers and engineers have kept working and improving AR devices for years and in 2013 we have witnessed the first release of a developer ready AR device: the Google Glasses and the Meta 1 developer kit. Three years later Microsoft released the HoloLens[1] augmented reality headset, which is the first device providing full AR capabilities. In 2017 Meta released its new headset, the Meta 2[2], which promised to change the office as we

[1]https://www.microsoft.com/en-us/hololens
[2]https://www.metavision.com/

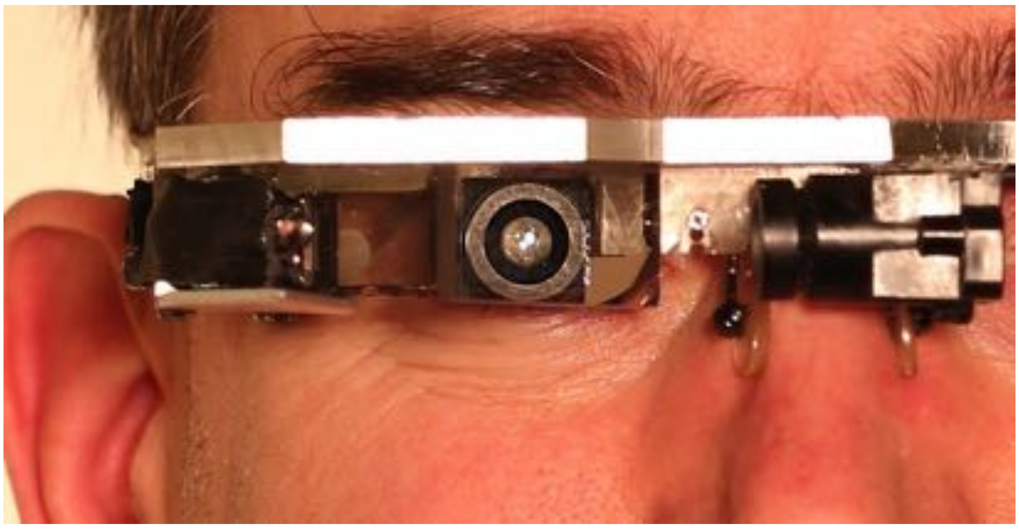


Figure 1.2: One version of the EyeTap device invented by Steve Mann.

know it, replacing all physical monitors with virtual ones. A lot of money is flowing in the AR industry, with companies like Magic Leap[3] that have been founded for over one billion dollars. Apple and Google have released software updates, for iOS[4] and Android[5], giving phones the ability to perceive the world and augment it through the phone camera. In June 2016, with the release of Pokemon GO, augmented reality has reached millions of people becoming one of the most popular trend in Google Search. AR is ready to enter in people's everyday life and it is going to change the world as we know it.

To better understand what AR is, it is interesting to see it in the context of other technologies. AR is part of what Milgram and Kishino [78] call Mixed Reality (MR). MR refers to those applications where real and virtual images are fused together. MR is composed of AR and Augmented Virtuality (AV). While in the case of AR we overlay virtual object in a real scene, in AV a virtual environment is augmented with images of the real world. In other words they differ depending whether there is a predominance of virtual objects (AV) or real ones (AR). To the extremes of MR we have, the Real Environment where no augmentation is done, and Virtual Reality (VR) on the opposite side. A VR environment is one where the visual feedback doesn't contain any live images of the real environment, but it's all either graphically generated or previously recorded. Figure 1.3 shows a diagram of the different level of reality.

[3]https://www.magicleap.com/
[4]https://developer.apple.com/arkit/
[5]https://developers.google.com/ar/

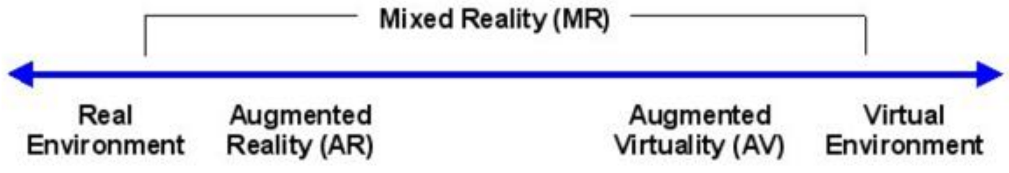


Figure 1.3: Milgram's mixed reality continuum.

AUGMENTED REALITY FOR HUMAN ROBOT INTERACTION One of the fields where AR has proven its strength and utility is in Human-Robot Interaction (HRI) applications. Robotics engineering is evolving very fast producing increasingly sophisticated robots, with more and more capabilities, expanding quickly the range of possible applications. In parallel, also the study of Artificial Intelligence (AI) is growing, providing autonomous capabilities to robotic devices. However, for many tasks involving robots the complexity is too high to be handled by an AI, and a human operator is needed to control the robot. Examples of these tasks can be found in medicine, industrial scenarios and space, where there is the need of highly precise movements and low reaction time to unexpected events. In many of these scenarios the operator cannot even be co-located with the robot, either because the robot is in a hazardous scenario, or because it is difficult to reach, as in space. Teleoperation is the name of the technique of controlling a robot from a remote location. Controlling a robot is not usually a trivial task as the robot may not have an intuitive kinematic structure and may have several degrees of freedom. In addition, in the case of teleoperation, we have the problem that the operator, not being co-located with the robot, may have difficulties in understanding the remote environment. Finally, an additional problem is the latency between the operator commands and the robot actuation, which can be very high in environments like space.

AR has proved to be an effective solution in providing support and guidance in the execution of a task. First of all, with a correct placement of one or multiple cameras on top of a robot, it is possible to give to the operator an egocentric view of the remote scene, enhancing his/her sense of presence and embodiment. On top of the images, coming from the remote environment, it is possible to display a set of virtual cues providing information related to the task. Examples of these virtual features, called in the literature virtual fixture [100], are the highlighting of target objects, showing the optimal trajectory to be followed, or providing the distances from obstacles and from target key objects. Examples of the use of AR in

HRI can be found in space teleoperation [81], medicine [53] and in industrial application [26].

VIRTUAL REALITY FOR HUMAN ROBOT INTERACTION Not only AR, but also VR is used in HRI application. The advantage of VR with respect to AR is that it allows to create virtual replicas of a real scenario, including the physical interaction. This simulation can be used for multiple purposes as training or validation, both for new algorithms and robotic devices. One of the most important fields, where VR and HRI are used together as a tool for training, is medicine. Still nowadays the majority of medicine students practice surgical operations on dead bodies, or spent long time observing highly experienced doctors performing live operations. Thanks to the combination of a robotic haptic device reproducing doctors' surgical instruments and a VE simulating the patient, it is possible to give to the doctor a complete visuo-haptic feedback. In these cases a VE, is not only used to provide the visual feedback, but also to compute the interaction between the robot and its virtual objects extracting the forces that need to be displayed to the doctor. It is possible to find several examples of training application for medicine in the literature, as needle insertion [29] or abdominal palpation [131]. In addition, one advantage of training in a simulated environment is that every session can be recorded and students can compare different executions and measure their performance.

Validation is also an important field where VEs are used. Robots are very expensive, so before testing new algorithms or proceeding to production, a simulation of the robot in VR is used. This allows to make a pre-evaluation, reducing the risk of damages and waste of money. Another example is to validate a robotic console that teleoperates a robot. These kind of consoles are often developed separately from the remote robot, and usually are designed to be able to control several different robots. For this reason a common way to validate the console is to create a VE implementing physics and use the device to control a simulation of a robot in the virtual world. An example is the validation of a haptic device to perform an ultrasonography (USG) examination or the hybrid system used to test the interface to perform abdominal palpation.

This thesis comes with two aims, one technical and the other methodological. The first research aim (**RA1**) is to define, design and create a set of software tools for easy and fast development of highly performing AR/VR applications, with a focus on HRI scenarios. The second research aim (**RA2**) is, instead, focused on the analysis and evaluation of the effects of AR in the interaction with robots when accomplishing medical and industrial tasks. These quite general aims have been split into several research questions in order to allow a step-by-step analysis and resolution.

All the work presented in this thesis started with the question: what do I need to implement a working and usable application that uses the power of AR to help doctors in performing remote examinations? In particular, what elements are already provided in the state of the art, and which are the challenges that still need to be tackled? (Research Question 1 (**RQ1**)).

AR applications, in particular in the HRI world, are composed of several components, each one dealing with a different aspect. While there are several tools and studies that deal with specific problems (tracking, rendering, ...), very few software solutions can be found that allows to orchestrate altogether the different components. A software is needed to allow all the components to work together live and in real-time respecting all time constraints. This means that this software needs to provide the possibility to connect the components so that one output can be the input to another one. This brought to the second question: how can we handle in the same application graphics rendering, Computer Vision (CV), haptics and robotic communication, satisfying the real-time constraints dictated by the usability needs? (**RQ2**)

Another important aspect, that had to be taken into account, was that there weren't any exact specifications of the requirements for the AR application that I had to implement. This was originated from the fact that AR is still a young and not well known world, where there isn't a state of the art of user studies showing which is the best possible feedback configuration. This introduced the question: how is it possible to create an application that can be easily customized and adapted, according to the users feedback, for constant improvement? (**RQ3**)

Every component involved in an AR application has its own reference system, graphics, CV algorithms and the robot kinematic. In order to be able to register all these blocks together it is necessary to align all

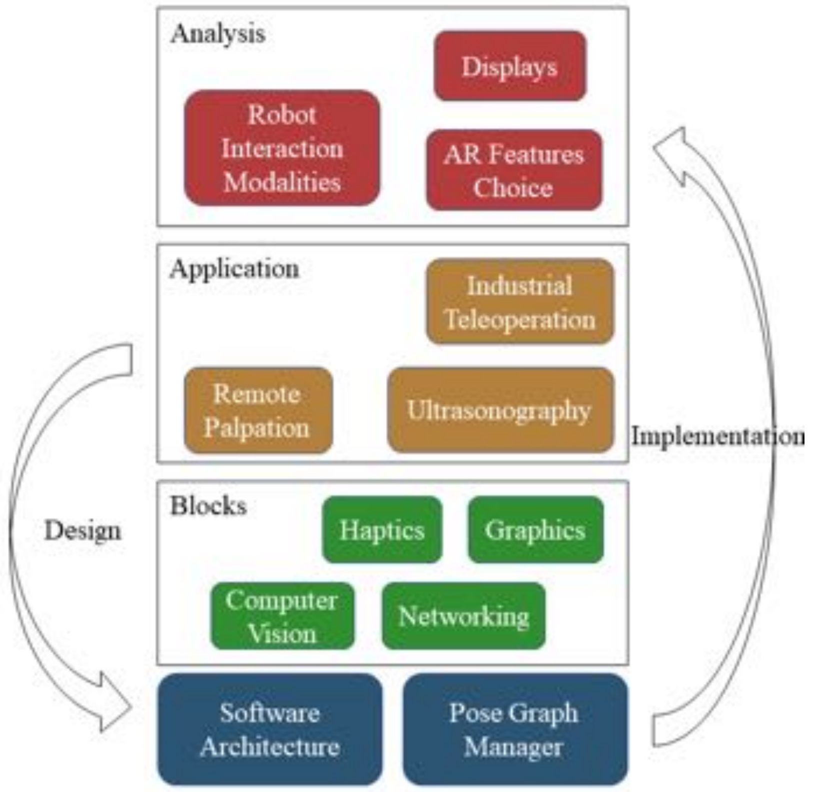


Figure 1.4: This image shows the research path followed in the thesis. Starting from the investigation of the requirements for an **Application**, I moved to the definition of the building **Blocks**. From there the question of how to connect these blocks led me to the design of a new **Software Architecture** and a framework to handle a **Pose Graph**. The implementation started from the lower level going up until I was able to create the applications I needed. At that point I was able to move to the **Analysis** and to perform a user study to evaluate and extract analytics on AR.

of their reference systems. Is it possible to find an integrated approach that allows to ease the calibration of systems and the computation/query of transformations, while preserving the efficiency of use and execution? (**RQ4**)

Finally when all the previous pieces came together, and the problem of how to create an AR application was solved, a last question remained open. What are the best and most effective features to be provided to the users in applications in the medical and industrial scenarios? (**RQ5**)

Figure 1.4 shows the process that brought to the formulation of the above research questions and the path taken to answer them.

1.2 CONTRIBUTIONS

In this section the list of the contributions of this thesis is presented. The research aims/questions answered by each contribution are reported between brackets.

C1: Analysis of the requirements and the state of the art for AR applications in telemedicine and general HRI (Chapters 2). (**RQ1**)

C2: Analysis of the current existing middleware/architectural framework for robotics, computer vision, and graphics (Chapters 2 and 3). (**RQ1**)

C3: Development of a framework for real-time integration of graphics rendering, computer vision, haptics and robotic communication. The framework has also a strong focus on modularity and fast prototyping (Chapters 3). (**RQ2**, **RQ3**)

C4: Development of a library to handle pose graphs. The library supports different types of calibration and efficient query and update of the transformations (Chapters 4). (**RQ4**)

C5: Implementation and evaluation of a user interface to perform remote palpation and one to perform an USG examination. (Chapters 5). (**RQ5**)

C6: Analysis of different AR features and their impact on user experience in industrial scenarios (Chapters 6). (**RQ5**)

1.3 THESIS STRUCTURE

This thesis is organized as follows:

Chapter 1. This chapter introduces the context of Augmented and Virtual Reality and their application in HRI. The list of the research questions we tried to address in the work are presented, together with a summary of the contributions.

Chapter 2. AR is a complex domain with several non trivial requirements that are listed and analyzed in this chapter. In addition a survey of the existing tools and reference applications are presented.

Chapter 3. The software architecture developed for enhancing and improving the development and performance of AR applications is presented in this chapter.

Chapter 4. This chapter describes the framework created to handle, easily and efficiently, generic pose graphs, and the solution implemented to compute reference systems calibration.

Chapter 5. Two applications of telemedicine and AR/VR are described in this chapter. In addition, a user study for the validation of a robotic device is presented.

Chapter 6. This chapter show a possible application of AR in industrial teleoperation, together with a user study to evaluate the impact of different AR features in industrial task execution.

2 BACKGROUND AND CHALLENGES OF AR/VR IN HRI

Augmented reality is a vast world with several different possible applications and developments. It can provide to users information about the environment to help them accomplishing certain tasks, both in an industrial scenario, and in the everyday life, as for example visualizing direction for navigation. AR can help designers, architects and engineers in visualizing 3D models before creating expensive prototypes, and AR can be social since that different people can look at the same hologram at the same time. One additional field where AR is showing its potential is in HRI scenarios. AR has the power to improve and enhance the interaction between humans and robots, cutting down the learning curve, which is often steep due to the complexity of robotics devices. VR also plays an important role in HRI applications, as it provides a tool to create simulated scenarios that can be used for training and testing.

In this chapter we are going to review the state of the art, presenting the existing softwares and tools, some of which have been utilized in this work. It is important to point out that the focus of this thesis is in the integration of AR and VR with robotics devices. This introduces additional complexity that happens to invalidate some of the available solutions and studies, introducing the necessity for doing an adaptation or even re-implementation in few cases.

2.1 AR/VR

One of the most commonly accepted definitions of AR is the one provided by Ron Azuma that states that AR technology has the following three requirements [7]:

1. Combine real and virtual.
2. Is interactive in real time.
3. Is registered in three dimension.

The first point is straightforward, the user needs to have the possibility to visualize, somehow, the real environment augmented with the virtual information. The next two points are less obvious and introduce additional complexity. AR being interactive in real time means that the virtual objects need to respond and adapt to user inputs (e.g. head and body movements) and environment modification in real-time. Finally, the last requirement states that augmented objects need to be placed registered according to the real world. This because, in order to have a faithful and convincing feedback, objects should interact with the real environment as if they were real. If the user sits in front of a desk looking at a 3D model of a car, the model should lay on the table and should respect the physics of the environment, thus not intersecting with other object and following the law of gravity.

For VR the requirements are more relaxed, as there is only the need of visualizing virtual objects and no registration in three dimensions is needed. On the other hand, the interaction requirement remains valid, as the VE needs to modify according to user movement and in the eventuality of an input device, such as a controller.

Starting from these three points we have divided the components that constitute an AR application into three categories that will be presented and discussed in the following.

2.1.1 *Graphics Rendering*

Rendering is, at the moment, the most advanced and complete topic in an AR application. Thanks to the advancement in the video games and movies industry it is possible to find plenty of, either open-source or proprietary, rendering software, each one with its own perks and capabilities. The two most important and famous ones are Unity 3D and Unreal Engine. These editors have evolved to provide to users a simple and intuitive interface that helps in the creation of cross-platform 3D application, abstracting the complex programming parts and the target devices. While these systems work great in standard environments, as in a desktop computers, they have extensibility limitations that makes them hard to use in custom setups. For this reason many other frameworks have been released that operate at a lower level, providing more flexibility and the possibility to be composed easily with other software. This is needed when using novel devices, either displays or cameras, and when integrating third party software.

It is possible to find in the literature two main approaches on how to structure the rendering implementation in an application: Scene Graph (SG) and Data Flow (DF) based approaches. The SG method consists in creating a hierarchy of entities. Each entity has a parent and its properties are the composition of the parent's ones and its own. The hierarchy is used to specify the position and behavior of each graphics component in a tree like structure. The framework organizes the processing in separate loops, each one handling a different property, in particular the graphics, the physics and the collision. The user can customize the behavior of each entity with some flexibility. This approach is found in systems such as OpenSceneGraph [23], XVR [25] or Unity 3D and Unreal Engine. Conversely, in the DF based approach the user describes the application in terms of streams of data and events that are connected in a flow of information floating from one entity to the other. Examples of frameworks using this approach are the ones following the X3D standard like InTml [47] InstantReality [13] and FlowVR [3].

2.1.2 *Displays*

When dealing with displays it is important to distinguish between VR and AR. While for the first type the current technology allows to have convincing experience, for the latter displays are one of the major limitations that prevents AR to be of wider usage.

There are two levels of VR, depending on the engagement of the user in the VE: immersive VE and non-immersive VE. The second is the one provided through a normal 2D or 3D display. The virtual world is only inside the rectangle delimited by the screen, and turning the head around makes it disappear. Immersive VEs instead are always around us, providing a sensation of real presence in the virtual world. There are two kind of technology that can provide an immersive VE: Head Mounted Display (HMD) and CAVE. Oculus Rift and HTC Vive are the most advanced and complete HMDs, providing a high resolution display, precise 6 Degree Of Freedom (DOF) tracking and controllers. The drawback of these two systems is that they require powerful PCs to support the rendering and tracking algorithms, and they are tethered to the workstation. On the other hand several vendors provide HMDs with only the lenses, where the user uses the smartphone as a screen. This solution has the advantage that the HMD is not tethered, and the drawback that the tracking is only 3DOF and the quality of the rendering is limited by the phone capabilities. CAVEs [33] have a completely different approach. A CAVE is a room where the walls and the floor are projector screens. The projection is in 3D and the user movements inside the room are tracked, so that the feedback is adjusted based on the position. Despite CAVEs being much more cumbersome than an HMD and also requiring more computational power to run all the needed projectors, they have the advantage of allowing the users to move freely and to see themselves.

In order to provide the best possible visualization feedback researchers and doctors have studied the human eyes coming up with a set of properties. The human eye has a resolution of about 1 arc minute, meaning that it can resolve up to 60 pixels per degree at the fovea. The most advanced HMDs are still far from this resolution, as the HTC Vive can reach around 11 arc minute. In addition to this, the visual acuity of the human eye is not uniformly distributed among all the retina, but it is more sensitive in the center and decreases moving to the edges. The current rendering systems provide uniform rendering on the entire display, but this is a waste of power as the eye is not able to perceive it. Using eye tracking techniques it is possible to render at different resolutions based on the position of the pupil. Companies like SMI customize HMDs adding the support for eye tracking. Another feature of the human eye is the field of view. Humans have around 200° horizontal field of view, with 114° overlapping between the eyes. The only VR system that can provide these configurations are CAVEs, while most HMDs, like Oculus and Valve, have around 90° of horizontal field of view. The latest important feature of the human eye is the refresh rate that has a lower bound fixed at 30 Hz [21]. Current HMDs are capable of reaching 90 to 120 Hz to improve the motion perception, thus reducing blurring perception.

In the case of AR the situation is more complicated as the user needs to receive as input both images from the real world and from the virtual rendering. To achieve this there are three main possible solutions, video based displays, see-through displays and projector based displays. Video based displays use image processing to combine virtually generated images with a video of the real world obtained from a camera (see Figure 2.1). This approach allows to use VR HMD to do AR, simply by attaching a stereo camera in the front of the device, just at eye height. Figure 2.2 shows an HTC Vive HMD with a ZED stereo camera mounted at eye level. The images coming from the cameras are augmented with the desired virtual

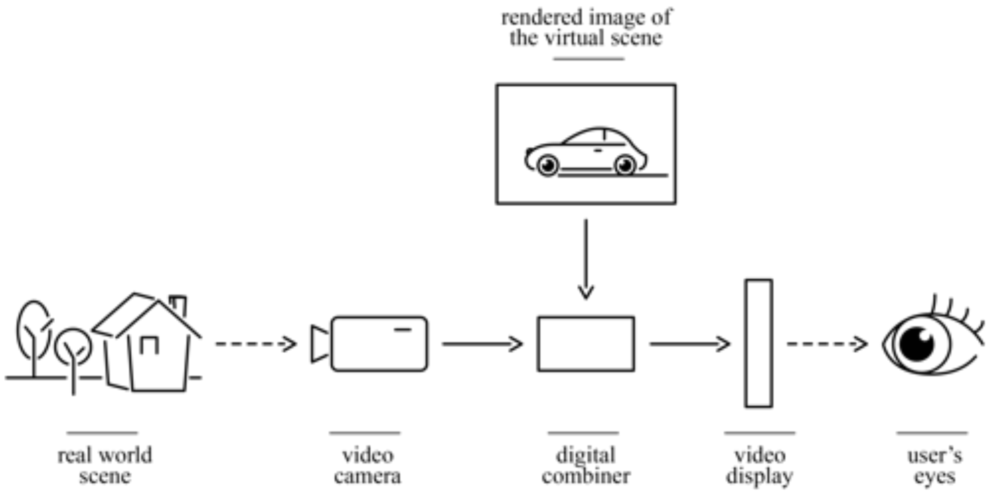


Figure 2.1: Structure of video based AR displays.

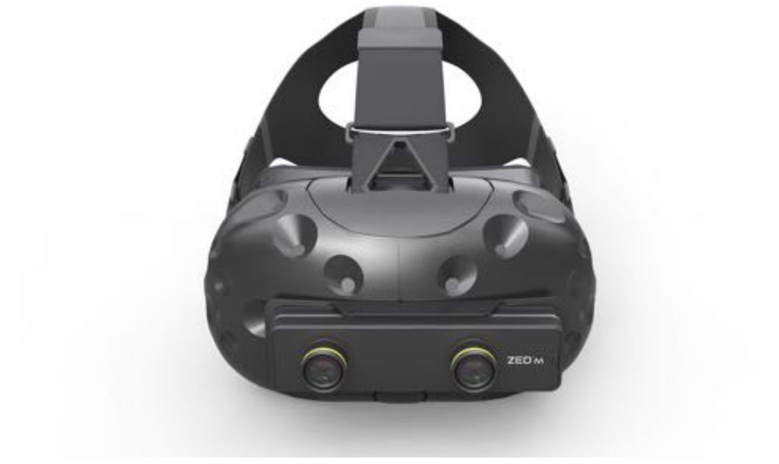


Figure 2.2: A HTC Vive HMD with a ZED stereo camera to create a video based AR display.

objects and then sent back to the HMD and displayed to the user. This approach allows to use mature technology coming from VR to do AR with a high fidelity. However this approach has several drawbacks: first of all the field of view of both the cameras and the display is narrower than the one of a human being, second the resolution of the cameras needs to be very high to give a good perception of the environment. On the other side, increasing the camera's resolution increases the amount of data that has to be streamed to the computer and then back to the headset increasing the latency. As we will see later latency is a key factor when dealing with AR and VR applications.

The second approach is to use see-through displays to do AR. Figure 2.3

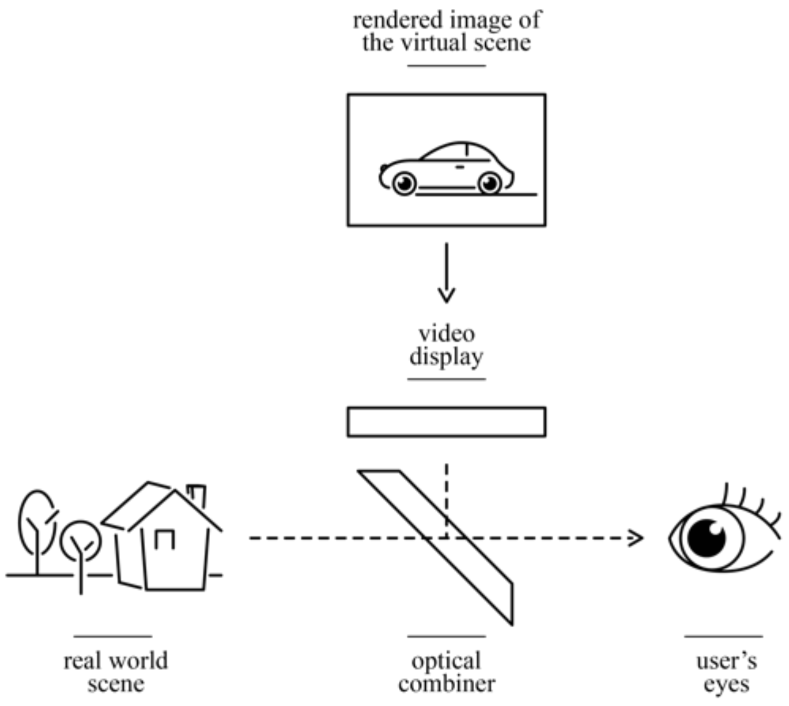


Figure 2.3: Structure of see-through AR displays.

shows a schematic of the functioning of a see-through display system. In this case the display is transparent allowing users to easily see the world around themselves. There are several ways to implement a see-through display. Rendered images can be projected on the display using beam splitter (e.g. half mirrors or combined prism) as in the case of the HUD in airplanes, Google Glasses or Steve Mann's EyeTap (see Figure 1.1). Another approach is the one used in Meta 2, where a display is placed over the forehead and the rendered images reflect on the transparent reflective glass screen placed in front of the user. Hololens use yet another kind of technology, called waveguide, where the hologram is created progressively extracting a collimated image guided by total internal reflection in a waveguide pipe. The pipe is a thin sheet of glass or plastic that the light bounces through.

Projection based AR displays overlay the virtually generated images directly on the surface of the physical objects. Figure 2.4 shows an example of how it is possible to dynamically color a real physical model.

An interesting categorization of AR displays is the one based on the position of the display with respect to the user, as shown in Figure 2.5. Over the years technology is advancing and displays are becoming lighter, while providing wider and brighter images. Despite retinal displays are still not feasible, HMDs are constantly evolving and improving. The main

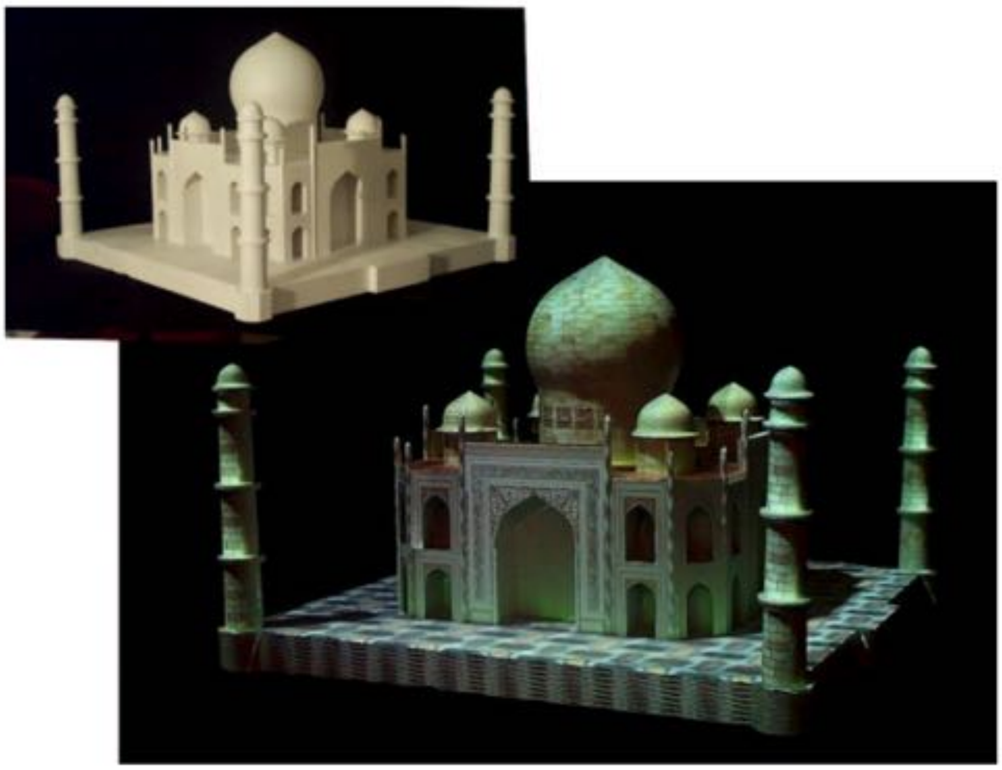


Figure 2.4: A physical model of Taj Mahal augmented with projected texture [98].

advantages of HMDs are the good field of view and that they are always positioned in front of our eyes. The downside is that they may be heavy, causing fatigue, and also, if the tracking and registration is not perfect, they can cause nausea and dizziness. Cellphones and tablets fall in the hand-held display category. They are a great example of video based displays and are becoming very popular in the AR world thanks to the wide diffusion and the relative small price. In addition with the release from Apple and Google of their AR toolkit they have become a complete AR platform. Lastly there are spatial displays that are standalone installations not tighten to the users, and range from displays to projectors [29].

2.1.3 *Tracking*

Tracking is an important part of an AR application as it allows to track the 6DOF movement of the user and of the device. This is mandatory in order to fulfill the second and third point of Azuma's requirements. There are several ways to implement a tracking system.

The simplest way to create a tracking system is by using an Inertial

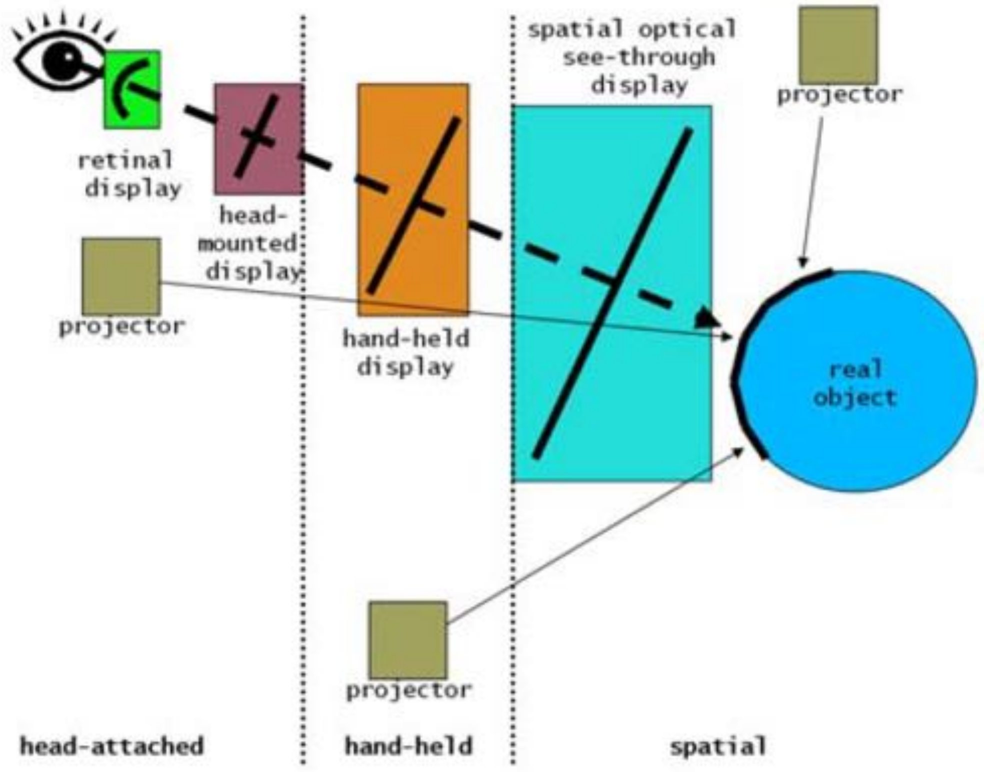


Figure 2.5: AR displays on eye-to-world spectrum [17].

Measurement Unit (IMU) sensor (gyroscope and accelerometer). While IMUs are great at measuring rotations, they have drift issues. For this reason IMU are alway paired with other systems, which use IMUs only for computing the orientation. Both systems use vision to compute the current position, but with an important difference. One, used by the Oculus Rift and the HTC Vive, uses external cameras, while the other approach, used by Hololens, Meta 2 or Apple ARKit, uses the cameras present on the device, in what is called the inside-out technique.

The first approach has the disadvantage of needing external cameras, thus limiting the possible workspace, but has the advantage that it is very precise, fast and reliable. This technique is mainly used by VR headsets that are tethered and can't move too far from the workstation. Even if the technology used by HTC Vive and Oculus seems the same their approach to tracking is diametrically different. In the case of HTC the cameras are not actual cameras, but just infra red projectors that emit light with a certain pattern, constantly. In fact the cameras don't even have to be plugged to the PC. The markers on the headset are photo sensors that trigger every time they are hit by a ray of infrared light coming from the camera. Based on which sensors are triggered at every instant it is possible to compute the position of the headset. Oculus Rift instead has passive markers on the headset that reflect infrared light, and the infrared cameras scan the images looking for the markers to reconstruct the position. In both the systems the position obtained from the infrared system is fused with the one coming from the IMUs.

In the inside-out approach the idea is to use the embedded camera the same way humans use eyes. The software, for every camera frame, looks for points of interest (keypoints), like edges or corners, and associates a descriptor to them, based on feature as orientation and position. Once it has these points, the algorithm tries to match them with the ones in the previous frames to find the transformation between the two camera poses associated with the images. This method is referred to as SLAM (Simultaneous Localization and Mapping) and comes from studies on autonomous robots as it allows them to navigate unknown environments. This system is less precise than the previous one, as it requires to be in an environment with lots of particular points. If the user is facing a plain wall the algorithm has a very hard time trying to detect his movement. It is possible to find in the literature several algorithms for detecting and describing points of interest. The SURF algorithm [12] for example, is very robust, but also computational demanding, while the ORB algorithm [101] is less robust, but also more efficient to compute. Also for SLAM it is possible to find several algorithms in the literature that differs on the algorithm for keypoints detection and the supported cameras, color mono [82], color stereo [88] and depth [69].

ENVIRONMENT DETECTION Detecting user movements is fundamental to have a registered AR application, but in order to have a complete experience there is also the need for the AR system to detect and recognize the environment around the user. This starts with the ability of detecting planes, like tables, walls or the floor, so that it is possible to place objects accordingly. Algorithms to detect planes are present in the major CV open-source softwares as OpenCV and PCL. In addition it is important to be able to detect generic objects and track their pose. This information can be used as a feedback to the user depending on the application that is running. Object detection is still an open research field, which is rapidly evolving.

2.2 HUMAN ROBOT INTERACTION

The interactions targeted in this work belong to two categories: teleoperation of humanoid robots in industrial scenarios and interaction with haptic devices in telemedicine applications.

2.2.1 *Humanoid Robots*

A humanoid robot is a robot with a body shaped to resemble the human one. The choice of the design is for easing their interaction with people and with environments that are designed for humans, like building, construction places and even cities. Humanoid robots usually have a head, a torso, two arms and two legs; sometimes the robots may have only part of the body, for example from the waist up. There are also robots that have highly accurate heads, designed to replicate human facial features and are studied by psychologists to understand interactions. DARPA Robotics Challenge in June 2015 provided a great contribution to the evolution of humanoid robots, which have proven to be able to perform a wide amount of tasks autonomously, like driving a car or using a drill. Despite the fact that AI is constantly improving there are still a whole plethora of complex tasks where these robots need human guidance. Examples of these cases are excavator machines that are controlled by an operator through the use of levers and knobs, up to robot assistants for surgical operations. Operating a robot is not a trivial task, and usually requires long time training for an operator, due to the difficulty of mapping the user's controls with the robot kinematics. For this reason humanoid robots are particularly suitable to be teleoperated, in particular in pair with body-based interfaces, as the straightforward mapping between the human operator's motion and the robot's motion greatly ease the execution.

Several approaches are possible to teleoperate a robot. The first is to use a camera and CV algorithms to track user movements. An example is the work of Song et al. where the teleoperation of a humanoid robot, is done using the motion reconstruction tool provided by the Microsoft Kinect and user movement are mapped to the robot's ones [124]. The advantage of this approach is that the user can move freely without any hardware overhead, and it is very fast to setup. However, the tracking precision is not very high, with a lot of noise, and can fail depending on the lighting condition, or if the user exits from the field of view or assumes strange positions. Another approach is to use an exoskeleton. An exoskeleton, as the name suggests, is a robotic device that is designed to be worn by an operator. In the case of teleoperation the exoskeleton includes a robotic arm where the user inserts his/her own arm and moves it, moving the robot. The movements of the exoskeleton are then mapped to the ones of the remotely controlled robot [16, 27]. The advantage of this approach is the high precision and reliability, and it can be enhanced with haptic feedback. The drawback is that exoskeletons are usually cumbersome and expensive. A third approach consists in using inertial sensing [79]. In this case the user wears a set of IMUs, typically on his arms, that are used to reconstruct his/her movements. The position of the arm is then used to move the arm of the robot. This approach allows to have a less cumbersome and expensive setup than an exoskeleton, and provides a higher precision than a camera based system.

SOFTWARE One of the key factors in the fast evolution of robotics has been the introduction of the Robot Operating System (ROS) [97], which has become the de facto standard software architecture for robotic applications. ROS, since its release, has been used as a base to implement, share and distribute almost all software for robotics, created by researchers and industry. Most of all commercial robots come with ROS support. Several open-source packages are available to compute a plethora of tasks, as inverse-kinematics to move a robotic arm to a given position, or packages to compute the best grasp strategy and many more. ROS has definitely contributed in increasing the evolution speed of robotics, providing a platform where it is possible to share and discuss new works and improvements. ROS has been widely used in this work to interact with several robotic devices and has allowed to speed up the implementation of several applications providing out of the box solutions.

2.2.2 *Haptic Devices*

A haptic device is used to provide a fictional sense of touch or force to the user. The device applies to the user handling it a force, a vibration or simply motions to stimulate tactile receptors and a force feedback. Differently from the eyes, which have an update rate of 30-60Hz, the human's skin and muscles perceive touch and forces at more than 1kHz [19]. In addition it is very sensitive to jitter. Variation in the update period results in a non fluid feedback which is felt as a trembling or shaking, and it drastically impacts the experience, as the human palm can sense movements as low as $0.2\mu m$ [66].

There are several types of haptic devices, each one with its own peculiarities. The first ones are those specialized in displaying forces, and are also the first ones to be studied. These kind of devices can be exoskeletons or handled joystick, like the Phantom from Geomagic or any Delta like device [95]. Handled robotic devices have the advantage of being able to display high forces in 3 to 6 DOFs. For this reason they are often used in teleoperation scenarios that require precise and delicate movements, like demolition machines [62] or medicine [10]. The disadvantage is that, in order to use the device, the user has to constantly keep his/her hand on the device, thus completely distorting the sense of touch. To overcome this limitation it is possible to implement an encountered interface that gives some autonomy to the device. In this scenario the haptic device autonomously comes into contact with the user's hand only when there is a force to be displayed [36].

Vibrotactile devices, instead, aim at simulating the sense of touch via vibration, and are usually very small and wearable. In general, vibrotactile feedback is used as a simple stimulus to which an information is associated. One of the earliest and simplest examples is the cellphone vibration when pressing a key; the feedback is used to notify the user that the key has been correctly pressed. Other examples are in collaborative robotics where robots send information to a user via vibrotactile stimuli [117].

Electrical muscle stimulation is another way of providing a force feedback that is based in stimulating the muscles by passing a current on the skin surface through electrodes. The electric current triggers muscle contractions and generates torque at the joints [138, 72]. The main applications aim at creating free-hand interaction systems, either for games [44] or for public spaces [93].

SOFTWARE The computation of the correct forces to be displayed by a haptic device depends whether the user is interacting with real objects or with a VE. In the first case the controlled robot has sensors to measure forces that are used to compute the stiffness of the remote interacting object. The stiffness is streamed through the network and displayed to the user through the haptic device.

In the second case the real device is interacting in a VE with a virtual surface, where there are no sensors. To compute the force every surface can be considered as a spring, and Hooke's law can be used: $F = kX$, where k is the stiffness constant and X is the depth of the indentation performed in the surface. During the modeling of the virtual world it is possible to statically associate the stiffness value (k) to every mesh. The main problems arise when computing X. Surfaces are usually not flat and they are modified by the interaction. In addition the force changes based on the shape of the tool that is touching the surface and other parameters. Luckily, many algorithms are present in the literature that deal with most of the possible scenarios and also several libraries are available that implement those procedures. The two most important and complete software are CHAI3D [31] and H3DAPI [39].

2.3 TELEOPERATION

Teleoperation, as the name suggest, is the technique of controlling a device without being physically connected to it. This can range from remote controlled toy cars, where the user shares the same environment as the controlled device, to scenarios where the operator and the device are at thousands of kilometers. Remote teleoperation introduces the complexity of establishing a communication protocol between the remote robotic system and the user interface. In this scenario robots are usually equipped with cameras to allow the operator to visualize the environment where the robot operates. This means that the images coming from the cameras, need to be streamed across the network. In addition, if a haptic device is involved, the remote robot needs to stream the values of the forces it is perceiving. In the opposite direction, the user sends the commands to teleoperate the robotic device. Video streaming is a well studied field, with lots of compression algorithms, as H.264, and available libraries and tools. More complex is the streaming of force values and the commands from the operator. There are two main problems, latency and jitter. When a user operates on a haptic device its inputs are collected at least at 1 kHz and sent via the network to the robot. The robot moves its end-effector to measure the forces, and sends back the values to the haptic device. This means that it may happen that, when the force is received, the user has in the mean time already moved the controller, and the feedback is not aligned. In addition, if the network is not stable some packets may be lost causing a non uniform feedback (jitter), and the haptic device may start shaking or jumping. Various approaches are present in the literature to overcome these issues, as the scattering theory [143] or the wave-variable approach [143].

3 SOFTWARE ARCHITECTURE AND INTEGRATION

An AR application, even a basic one, is an orchestration of several components that need to communicate and synchronize with each other. The tracking system provides the movements of the head to the rendering software that in turn uses them to adjust the position of the virtual objects. In addition the images coming from the camera are streamed to the algorithm that detects planes and surfaces, which are then used by the physics system. If the tracking system is implemented using SLAM, the images from the camera will be shared by both the tracking software and the environment detection algorithm. Each one of these modules has precise time constraints, dictated both by the technology and by user requirements. As seen in Chapter 2, tracking, rendering and the control of haptic devices all have strict requirements for update rate, latency and jitter, and in addition, they are not trivial problems. For this reason orchestrating all these tasks is often a daunting problem.

An interesting property of AR systems is that every component is independent from the others, they just need to agree on the format of the data they exchange. The rendering software doesn't depend on how the tracking is computed, as long as the head pose is provided in a known format. The same principle is valid also for the physics system that is not affected by the algorithm used to track planes or do mesh reconstruction. This is an important aspect as it allows re-usability of the code, which can be adapted to different scenarios.

Another characteristic aspect of AR applications is the looped execution structure. Tracking software executes the same algorithm for every frame, as well as the graphics engine that recompute the virtual scene based on the refresh rate of the screen. Figure 3.1 shows an example of the components involved in an AR application, and how these components are connected and interact.

This chapter presents a novel software architecture, Compact Component (CoCo), implemented with the purpose of providing an easy way to create AR applications, in HRI scenarios, with a focus on code re-usability, flexibility and most important performance. The library is open-source and constantly

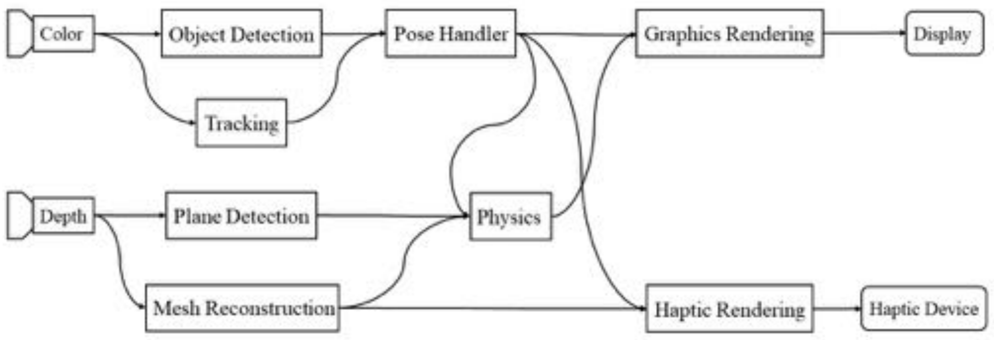


Figure 3.1: Example structure of the components that concur in the creation of a generic AR application. We can see on the left the input sensor devices, like color or depth cameras, which produce images that are then processed by other independent components. On the right the output devices, which are the display(s) and possibly a haptic device.

updated[1].

3.1 RELATED WORK

It is possible to find in the literature several libraries that provide functionalities to implement a specific component, but there is a shortage of software to orchestrate the components all together. During the years several attempts have been made to designe and create a general framework to support the development of AR applications [75, 11, 130], but no one has managed to reach a widespread usage. Nevertheless, interesting solutions, applicable to the AR field, can be found in the robotic world [80].

AR and robotic applications are very similar in the structure and share several requirements. In a robotic application, among all, there is the need to control the motors and the sensors, to compute the planning for the motion or the grasping and the localization. It is easy to find a similarity with the situation described above for AR, given that, also in this case, all these components execute independently from each other, and interact only to exchange data. In the following we are going to analyze the most prominent software architecture for robotics.

[1]https://github.com/cocomr/coco

3.1.1 *ROS*

ROS [97] is a distributed computing platform, with support for data passing. Users can create custom nodes (processes) that can output or receive data in a publisher-subscriber fashion over TCP/IP. Each node is independent from each other and connections between nodes happen at runt-time. Nodes can live on different machines and share the same parameters and configurations. The power of ROS comes from its modularity and simplicity. Every node can be developed stand-alone and combined with whatever other one, providing that they agree on the data they want to exchange. ROS provides also a build system that makes it easy to create and share new nodes. In addition, it provides a long list of support tools that spans from server parameter, graphic debugging tools and simulators, to easy serialization and playback of data.

The main and important drawback of ROS is performance. Despite being a distributed system that allows virtually infinite parallelism, ROS comes with drawbacks. Every node is a process and all the data is shared via TCP/IP, thus requiring copying of data multiple times, even if in the same machine. When dealing with high frame rate (haptics) or big data (images) data copy can become the bottleneck that hijacks real-time execution. ROS has a way to overcome this by using nodelets, which provide multi-thread support, thus allowing for zero-copy shared-memory communications. The problem with nodelets is that they are not easily configurable. Either a piece of code is born as a nodelets or it cannot be instantiated as one. In addition, a third party node can not be run as a nodelet.

For the sake of completion it is important to cite YARP (Yet Another Robotic Platform) [77]. YARP is a software middleware very similar to ROS, as it provide communication support between nodes, in a distributed environment. YARP was developed before ROS, but it wasn't able to find the same space and propagation.

3.1.2 *Orocos*

Orocos [22] is another important middleware framework for robotics, and also one of the first developed and still maintained. It differs from ROS as it is designed to support the implementation of low-level robotic control. Low-level robotic control is characterized by simple algorithms, each one with a specific focus (control of the motors of the arm, wrist, leg, etc.)

that can run in parallel and need to exchange data with each other. The scenario is very similar to the one described for ROS, with a single important difference: these algorithms need to run in strict real-time fashion at high frame rate ($\geq 1kHZ$). This makes a distributed system, as ROS, unfeasible to support this scenario. The problem is that TCP/IP timing cannot be controlled and its latency cannot be always predicted with 100% accuracy. Orocos overcome ROS limitations by providing support to multi-threads shared-memory communication and to hard real-time scheduling running on Linux RTOS. At the same time it also supports a good level of flexibility and reconfigurability, where each node is independent from the others. The drawbacks of Orocos are that, being an old project, it doesn't exploit all the advantages of new software technologies, and it needs to run on a special version of Linux, limiting its integration with other software.

Another difference between ROS and Orocos is that, while the first exploits a publisher/subscriber communication pattern, the second uses a data flow programming approach [125].

3.1.3 *MIRA*

MIRA [41] is another interesting framework developed to support robotics application. It has a structure very similar to ROS with few notable exceptions. ROS uses a centralized system, where each node has to connect to a master to do any information lookup, and this introduces a single point of failure. On the other hand MIRA is a full decentralized system with a reliable fault-tolerance system. In addition MIRA is more flexible than ROS for what concerns the instantiation of nodes in either processes or threads, by just specifying it in the configuration file. Unfortunately it doesn't provide tools for soft real-time execution, using a best-effort approach in the scheduling of the execution. This is mainly due to the publisher-subscriber approach used in the communications that conversely from data-flow doesn't allow for strict management of execution order and predictability.

3.2 THE ARCHITECTURE

The purpose of the CoCo library is to provide stronger timing guarantees than ROS, while being more flexible and easy to use than Orocos. CoCo tries to combine the real-time solutions introduced by Orocos with the flexibility and reconfigurability of ROS. This allows CoCo to be feasible both for supporting prototyping and production execution. This has great value as it speeds up the development, reducing testing time.

CoCo achieves these capabilities by being a component-based system, in which each node is a sequential independent unit of execution capable of exchanging data with other nodes, and of triggering other nodes execution and operations.

CoCo, like Orocos, implements a data flow programming approach. AR applications, as we can see from Figure 3.1, have a logical structure where the data produced by an input component flows through several other components, up to an output device. Using a data flow programming approach it is easier for the developers to map the design of their applications to actual C++ code, speeding up the development.

CoCo has been developed in C++11 exploiting the latest features of the language to be lightweight, easy to use and multi-platform. Components are loosely coupled to increase modularity and reduce development dependencies: in terms of C++ this means that the only common element between components is the exchanged data format. Components are stored in dynamic libraries and they can be instantiated at run-time by name. A CoCo application is typically launched by providing an XML file that specifies which components to load and their configuration parameters, together with the desired connections and execution policies. Thanks to the configuration through the XML file it is possible to quickly and easily update the application, introducing new components or modifying the connections. Programmers can test several different configurations without changing a single line of C++ code, thus without having to recompile the components' code. Meanwhile, the loader operates at C++ level guaranteeing that the performances are not compromised.

3.2.1 *Components*

Components are the building blocks of CoCo applications and represent atomic blocks of execution written in C++. To create a custom component the programmer needs to inherit from the base class *coco::TaskContext*. This allows the user to utilize, inside his/her class, all the CoCo functionalities. A component is characterized by the following elements:

- Callbacks, containing the code that is executed in the main loop.
- Declarative attributes that can be set at run-time via the XML configuration file.
- Operations that can be invoked thread safe and asynchronously.
- Input and output ports that are used to exchange data with other components.

Components can be classified into three categories: producer, worker and sink, depending on their position in the flow of data. Producers are components that, as the name suggests, produce data and don't receive any input information. The most notable examples are camera and sensor drivers. These kind of components are usually executed periodically, based on the refresh rate of the device they are connected to. Workers are pass-through components, they receive data from producers or other workers, and process them, creating new ones. Examples of workers can be image filters, haptic rendering, objects detection, etc.. Mainly they execute only when new data is available and are, hence, synchronized to the producers they are connected to. The last type of components, sinks, consists of those blocks that are the collectors of all the data processed by the other components. In an AR application the most notable example is the graphics rendering component that uses the information computed by the tracking, physics, users input, etc., to create the virtual scene to be rendered to the display. Sink components don't usually output data to other components as they communicate directly with the hardware (displays, robotic devices, etc.).

3.2.1.1 *Callbacks*

Callbacks are the engine that drives the execution and characterizes each component. There are four callbacks, inherited from the *coco::TaskContext* class, which the programmer needs to implement with its custom code. When the component is instantiated, the run-time system calls these functions, executing them according to the policy specified in the XML file and described later in Section 3.2.3.

The first two callbacks, *init()* and *onConfig()*, are used to initialize and configure the component, based on the parameters expressed in the XML file. CoCo run-time guarantees that, before executing any *onConfig()* function, the *init()* function for each component is executed and completed.

The main callback is *onUpdate()*, which is executed in a loop, either periodically with a timer, or triggered by data reception (see Section 3.2.3 and 3.2.1.4). This function is the central part of the component, and contains the code to perform the processing of the data. The *onUpdate()* implementation should not block the execution, by doing for example active waiting on an event.

3.2.1.2 *Attributes*

Attributes are the mechanism that allows a component to expose its internal private variables. This is done to be able to set those variables from the XML file. The programmer can create an *Attribute<T>* object, templated on the type of the variable that will be bind with the Attribute. The Attribute constructor takes the reference of the associated variable, and a unique name within the class's attributes. From outside the class, it is possible to access the attribute by name, and then use a getter and setter to read or update the variable's value. This is used by the XML loader that can read the value specified by the programmer in the file, and set it in the components. In addition, attributes can also be used by other components to read and update each others variables.

3.2.1.3 *Operations*

Operations are equivalent to attributes, but they expose methods instead of variables. Each operation is associated with a member function of the component and a name, which must be unique within the component. The name allows other components to retrieve the operation and call the embedded function with the desired parameters. An operation contains a handle of the function, thus calling the function through the operation doesn't introduce any overhead with respect to a direct invocation. However, it is not recommended to call another component's operation directly, as this is not thread-safe and may produce synchronization problems. This is due to the fact that the operation's method may access component's variables while they are being accessed by the *onUpdate()* callback.

CoCo provides a solution to overcome this problem and guarantee thread-safe execution. The solution is based on the deferred invocation of the operation and its underlying function. The operation can postpone its execution by queuing the call in its component's pending list of operations. Every time a component is triggered, before executing the *onUpdate()*

function, the run-time support checks whether there are pending operations, and in case it calls them one after the other. The task invoking the function to enqueue an operation on another task can add to the call a function to get the return value of the operation.

3.2.1.4 *Ports*

Ports are the key mechanism for data exchange between components inside CoCo, and they have been designed to support different patterns of data exchange and provide an efficient mechanism to transfer large entities such as images or point clouds.

Ports can be either input or output, are templated on a given type and have an unique name within the component that owns them. Input ports can be marked as *event* to specify that the execution of the component should be controlled by the reception of data in that port. In other words, whenever an event port receives data, it will report it to the run-time manager that will wake up the component and start its execution. Input ports provide functions to read the data they receive, while output ports grant the possibility to read from them. The connections of the ports are not specified in component code and are independent from the ports. In other words a port doesn't care from whom it receives the data or to whom it sends them, as the connection is done at run-time.

Any C++ types, with a copy constructor, can be exchanged between ports, and thread-safety is guaranteed. The use of C++'s shared pointers allows to perform zero-copy communications, with the only overhead of copying the pointer and handling the reference counting value. This is very important when dealing with big objects like images or point clouds. The use of ports together with shared pointers is the key feature that allows CoCo to provide (soft) real-time capabilities.

Both reading and writing are non-blocking. This means that when reading from an input port that doesn't contain any data the function will return immediately and it will not wait for new data to be received. The opposite is also true, as writing in a full buffer will cause the data to be dropped and the execution resumes instantly, without waiting for the buffer to free space. Components can have multiple input and output ports. Multiple input event ports can work synchronized so to guarantee that execution happens only when data is present in all of them.

3.2.1.5 *Peers*

To improve the modularity and re-usability it is possible to create sub-components called peers. Peers are used to extend the functionalities of a component preserving code encapsulation and re-usability as they can be instantiated multiple times for different components and the binding is decided at run-time. Peers are components by themselves inheriting all the functionalities such as ports, attributes and operations. The main difference relies in the fact that peers execution is controlled by the owner component, and typically they are used via operations. Thanks to operations there is no limit in the number of peers that can be associated to a component or to another peer, giving the possibility to create a tree of peers with any desired depth or width.

3.2.2 *Connections*

Connections, as the name suggest, are the entities that connect components' ports to each other and allow the flowing of data between components. The only requirements to connect two ports are that: they must belong to different components, one has to be output and the other input and they are templated on the same C++ type. Connections between ports are many-to-many, meaning that a single element, written to an output port, can be received by multiple recipients, and that multiple data, coming from different output port, can be written in the same input port. When the programmer writes in an output port, the message is broadcast by default to all the input ports connected. Conversely, when reading from an input port connected to multiple output ports, data is received one at a time, using a round robin schedule for polling. Alternatively, CoCo provides a special read function that retrieves data from all the incoming connections, and returns them in a vector. Ports manage their several connections with a *ConnectionManager* object that handles all the connections abstracting the complexity to the port and the user.

The data written in an output port and not yet read is stored in the connection that owns the buffer. When creating a connection the programmer specifies the desired policy for it, in particular, the policy for the buffer, for the synchronization and for the type of transport. CoCo supports three different types of buffer:

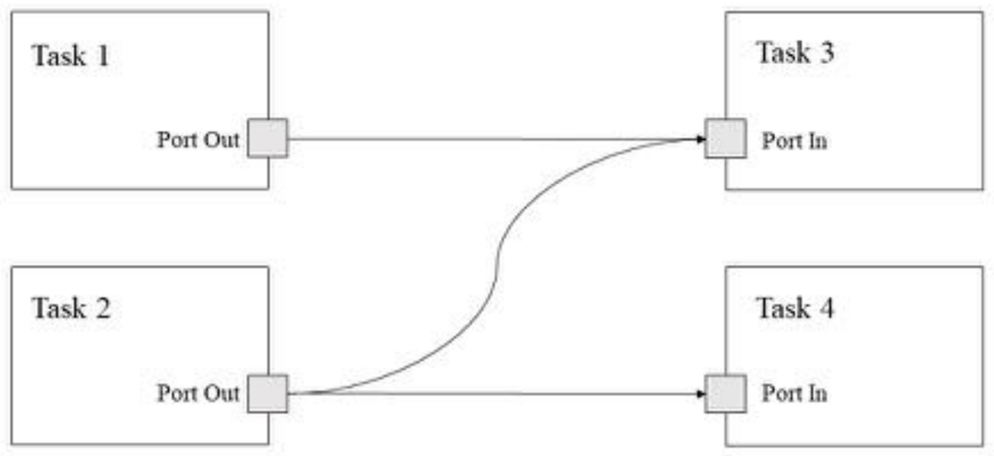


Figure 3.2: Basic example of how ports can be connected with each other in a many-to-many fashion.

- **DATA**: The connection has a buffer of length 1 and new incoming data always overrides existing data even if it has not been read yet.
- **BUFFER**: The connection has a FIFO buffer of length as specified in the configuration file. If the buffer is full new incoming data is discarded without blocking.
- **CIRCULAR**: The connection has a circular FIFO buffer of length as specified in the configuration file. If the buffer is full new incoming data overrides the oldest one.

A DATA buffer is equivalent to a CIRCULAR one with length 1 except much more efficient, given that it doesn't rely on any data structure.

The connections support two synchronization policies: LOCKED and UN-SYNC. In the first case data access is regulated by mutexes while in the second case there is no resource access control policy. A lockless policy could be added providing high-efficient data access. The un-sync policy applies for the connections between components inside the same thread.

CoCo provides a common interface for all the connections allowing to easily create a custom one. It is possible to easily add a new buffer type and also change the type of mutex used. For example, when passing small data through a connection, like shared pointers or primitive types, the mutex from the standard library introduces too much overhead, and can be substituted with a spin lock.

3.2.3 *Execution*

The execution takes place inside a container called activity that can hold multiple components. Every activity has a system thread associated to it. An activity executes in loop, and during every cycle all the components are executed sequentially. Periodicity or triggering are specified at the level of activity: a triggered activity is activated when any of the contained components receive some data in a event input port. Periodic activities execute at a constant rate specified by the user in milliseconds.

The XML configuration file of an application contains the list of activities, each one with its list of components and its configuration. The activity's configuration comprises the periodic nature, as period in milliseconds or triggering. In addition, one of the activities can be marked as "main" for being associated to the main thread of the CoCo application.

The component-activity separation allows the developer to reconfigure the flow of execution at run-time, without the need of customizing or recompiling the components. An improvement of the model discussed above relies on increasing the granularity of components activation, which is supporting multiple rates inside an activity or controlling the triggering.

The component-activity separation allows the developer to reconfigure the flow of execution at run-time, without the need to customize or recompile the components. An improvement of the model discussed above relies on increasing the granularity of components activation to support multiple rates inside an activity based on different triggering.

3.2.3.1 *Metrics*

There are two metrics to measure the performance of a task based, loop application: service time (T_S) and latency ($\mathcal{L}$). The first one measures the time between two successful completion of a round of execution. The lower bound for this value is, of course, the time between the generation of two inputs (T_P). If this value is higher it means that periodically one of the input is dropped. For example if a camera goes at 30Hz, it provides a new frame every $33.\overline{3}$ms. An algorithm processing the image needs to complete in less than $33.\overline{3}$ms to be able to be ready when a new frame arrives. In any case the processing software cannot have a service time of less than $33.\overline{3}$ms, as it is synchronized with the time of the component that produces the information.

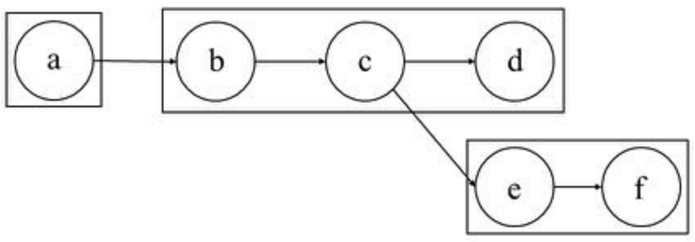
The latency, instead, measures the time for an input to be processed and become output. It is constituted by the sum of the computation time (T_C) of all the tasks concurring in producing the result. In a sequential unit of execution the latency should be less than the service time, otherwise, what happens is that, while the block is still processing the data at time T_n, the new data at time T_{n+1} arrives, and its processing is delayed or even canceled.

3.2.3.2 *Scheduling and Partition of Components*

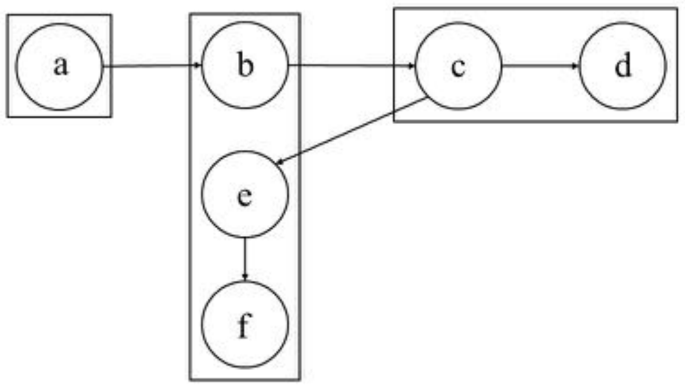
The graph created by the connections, namely the data-flow graph, specifies the dependencies of each component, providing constraints on the order of execution. The triggering system guarantees that the dependencies, specified by the data-flow graph, are satisfied during the execution. So, the schedule comes for free once the user has specified the connections.

The scheduling sequence is not enough in a parallel application, as also the partition of the tasks into threads needs to be provided. This corresponds in placing CoCo components into activities. The easiest solution is to instantiate one activity for every component, so that each component runs in a different thread. However, this introduces a lot of overhead if there are lots of components with short execution time, due to the context switch time. The best solution would be to instantiate a number of threads equal to the number of cores, so that no context switch happens. This may not be feasible as dependencies between tasks may need more activities, not to create dead-locks. Another limitation is that any component inside an activity can trigger it, and once the activity is awoke it will execute all the components it manages, regardless whether all of them have received the needed trigger. This is a strong requirement necessary to reduce control flow checks and simplify the run-time support to reduce the overall overhead. In addition this allows more predictability in the system and ease the testing and debugging. On the other hand it limits the possible partitions as shown in Figure 3.3.

From a performance point of view, a feasible partition is the one that allows to obtain a service time equal to the one of the producer components. In addition, an optimal partition is the one that minimizes the latency. The partition of tasks in activities is a NP-complete problem, as to find the optimal solutions, all the possible combinations need to be tested. To be able to evaluate the feasibility of a schedule/partition system it is necessary to have timing statistics for each component. For this reason the application



(a) A feasible partition schema.



(b) A non feasible partition schema.

Figure 3.3: Circles represent components, arrows connections , and rectangles are activities. Figure (a) shows a feasible partition, while Figure (b) shows an unfeasible one. In the second case activity components f may be executed before receiving the data from component c, resulting in undefined behavior or skipping one iteration.

needs to be profiled, instantiating one activity per component, so to collect timing statistics for each component. Note that for a given schedule there may be no possible partition that guarantees the desired service time.

3.2.3.3 *Parallel Execution Patterns*

In order to improve service time and latency it is possible to parallelize the computation applying known patterns, studied for this purpose. There are two general ways to parallelize an application: task or data based. In the first case the parallelization happens between tasks operating on different data. In the second case, instead, the parallelization is done by splitting the data in independent chunks that are processed by multiple parallel instantiation of the same task.

Data parallelism is the most desired type of parallelism as it reduces the computation time of a single task, thus the latency. Reducing the latency has also the benefit of contributing in keeping the service time optimal. The problem with data parallelism is that it is not always possible, as it requires data to be splittable in independent chunks. There are, however, several examples of possible applications in CV and image processing.

Task parallelism is characterized by two main patterns, pipeline and farm.

PIPELINE Pipeline is the simplest one and consists in concatenating tasks one after the other, each one taking input from the previous and outputting to the following. Each task executes in parallel to the others, and this allows that, when a task is executing on data at time T_n, the previous task can execute on data at time T_{n+1}, and the following on data at time T_{n-1}. This means that the service time of the whole pipeline is not the sum of the T_C of all the tasks, as it would be if run sequentially, but is the maximum between the T_P and the highest T_C among all the tasks. Parallel pipeline, however, doesn't improve the latency, as each data is processed sequentially, by one task after the other. Figure 3.4 shows a simple example, where each task is associated with a T_C and the first with T_P. Pipeline are supported in CoCo by default; through the use of connections and activities it is possible to create a pipeline with the desired degree of parallelism.

FARM Pipeline has the limitation that, if a T_C^i is higher than T_P, a bottleneck is introduced that causes the T_S to increase. Pipelines are feasible only if all the components T_C is lower than the T_P. When this is

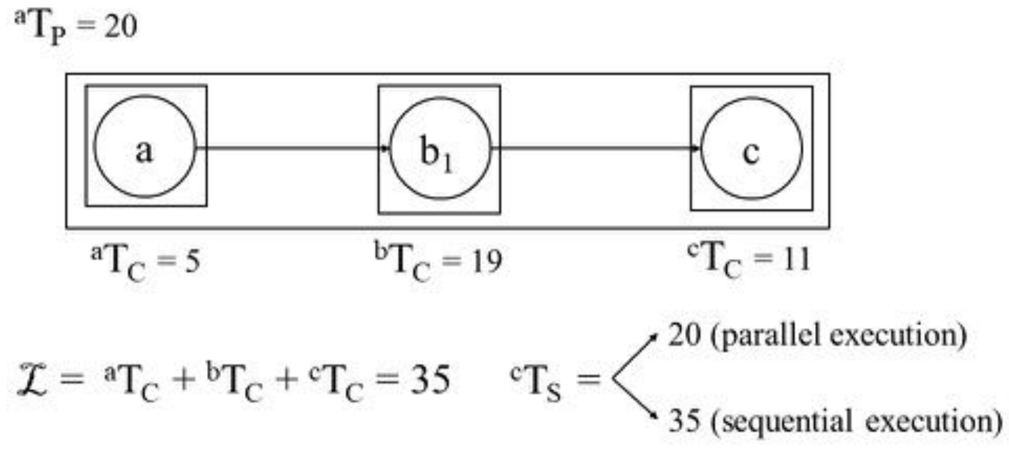


Figure 3.4: Simple example of a pipeline composed of three tasks (*a, b, c*), each one with its computation time and the initial production time.

not the case the only solution is to use the farm paradigm. The idea behind a farm is to take the task that causes the bottleneck and duplicate it, so that, when new data arrives, and the task is still executing the previous one, this is redirected to the duplicated task. In the farm paradigm tasks are called workers and their number must be $N_w \geq \lceil \frac{T_C}{T_P} \rceil$, in order to guarantee that, when a new data arrives, there is at least one free worker that can start executing with the new data right away. Differently from pipelines, farms need additional support to be executed, in the form of two special tasks, the scatter at the beginning, and the gather at the end. The scatter receives the data, look for a free worker, and forward the data to it. The gather instead receives data from the workers, and forward it to the next connected tasks. Scatter and gather are used to mask the farm from the other components that see the farm as two standard nodes.

In CoCo it is possible to easily instantiate a component as a farm that is simply a special type of activity. The programmer, in the XML file, specifies the number of workers, the component and port that will provide data to the scatter and the component and port that will get data from the gather. The loader will modify the *ConnectionManager* object associated to the scatter port, so that, instead of broadcasting every message, it checks for an available worker and sends the message only to it. All of this is done automatically, at run-time, during initialization, and doesn't require any modification in the source code. In addition, it is even possible to create farms where the workers are a pipeline of multiple components, providing more flexibility.

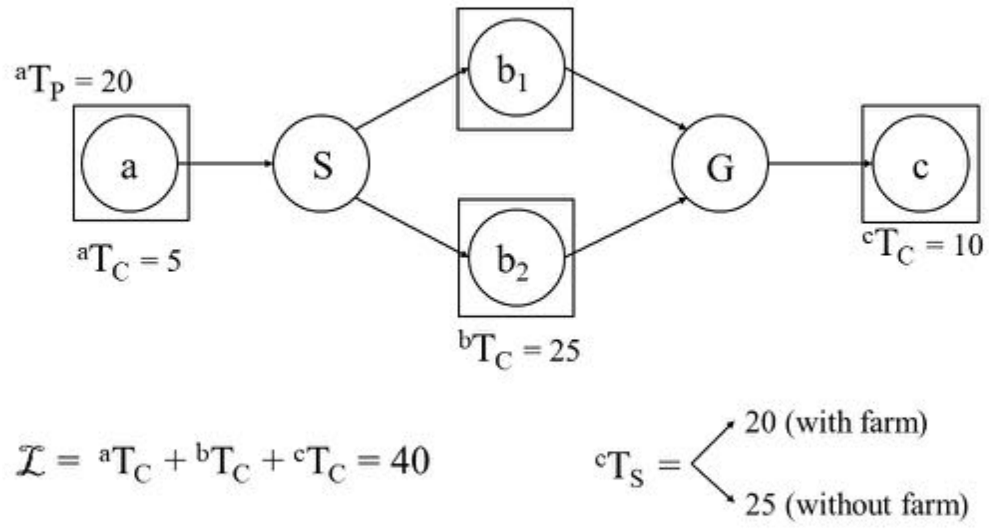


Figure 3.5: An example of how a farm can help reducing the service time of a pipeline, when one of the node computation time is greater than the production time.

Farm allows to prevent possible stall due to problems in the execution. Suppose to have a pipeline of few nodes, each one with $T_C < T_P$, so that $T_S = T_P$. Unfortunately, only in real-time OS, and with very deterministic algorithm, the computation time is guaranteed to be constant; what happens normally is that for certain input, or due to unpredictable OS scheduling policies, a task can take long time to complete, slowing down the service time of the overall application. To overcome this, a solution is to put the pipeline in a farm, duplicating it. Each pipeline execute sequentially, and every new data is redirected to a free pipeline. Figure 3.6 shows an example where the three tasks (a, b, c) execute on the same thread, but there are several instantiation of them. If, unfortunately, one of them gets stuck, the execution can continue on the other instantiation.

3.2.3.4 *Real-Time Execution*

The applications targeted by the CoCo framework fall in the so called soft real-time domain. A real-time application is characterized by several independent components, each one with a computation time, a period and a deadline. There are several levels of constraints in a real-time application that goes from hard to soft real-time. In the hard case it is enough for a component to miss its deadline to cause catastrophic consequences. Examples are airplane or car control. In order to guarantee hard real-time constraints, components need to have constant computation time and run in a special

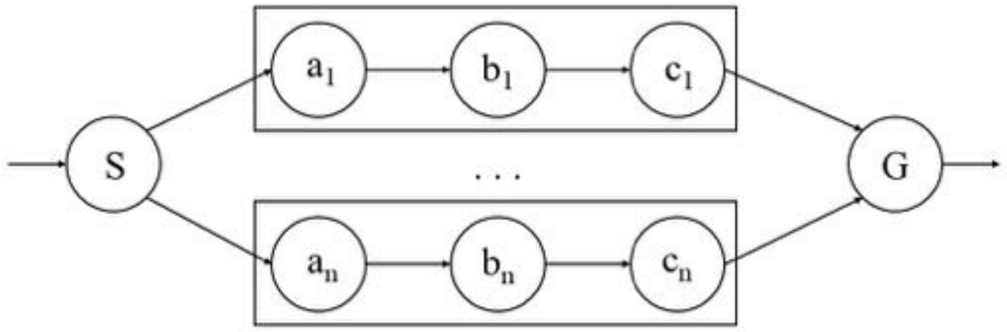


Figure 3.6: A farm of pipelines. Instead of parallelizing within the pipeline, the parallelization happens duplicating it. This allows to prevent possible stall in the computation guaranteeing the same best performance.

OS. Orocos is an example of a framework that supports hard real-time applications. In AR applications there are no such constraints, as skipping a frame will simply cause the tracking to have a jump, which is, yes annoying, but not dangerous. In addition, standard consumer OSs, like Linux or Windows manage the scheduling of threads and process in a best effort manner, with heuristics that don't guarantee execution at the desired time. Nevertheless there are few approaches that can improve the performance of an application improving the predictability and reducing overhead. First of all, OSs provide tools to change the priority of user threads, thus giving them precedence over system or background processes. In addition it is possible to pin each thread to a specific physical core (affinity). This allows to reduce context switch time, also exploiting cache locality. In the latest years the standard Linux kernel has added the possibility to use the earliest deadline first (EDF) scheduling algorithm [126] for any application. EDF is an optimal scheduling algorithm as it guarantees to provide a feasible schedule if this exists. The problem with EDF is that it requires each thread to provide its computation time as constant parameter, and this is not possible for the majority of AR components.

Another way to improve performance is to manage memory wisely, avoiding continuous allocations and deallocations of memory blocks. Every time we allocate or deallocate memory a thread needs to communicate with the OS, and this introduces overhead. A possible solution is to use a memory pool allocator, which firstly allocate a big chunk of memory that is never deallocated, but redistributed to the threads when needed. However, this approach cannot be enforced by CoCo itself as memory is managed by components' developer.

3.2.4 *Performance Measurements*

The CoCo library provides also a utility to easily calculate execution time of blocks of code. This functionality can be used by the user inside its components to quickly evaluate the computational load, and it has been inserted inside the core of the library to obtain precise statistics on the components' performance. The profiling step is activated passing a specific flag to the launcher, and statistics of the execution are provided with a certain time interval. The measurements include, for each component, the number of executions and the total execution time, and mean and variance of the average computation and service time. This last value is used to evaluate the feasibility of the application scheduling because it represents the mean time between two activations of a component and should be equal to the period for periodic components.

3.2.5 *ROS Interoperability*

The integration and interoperability with ROS is a mandatory requirement when developing robotic applications. ROS has become the de facto standard in robotics and many vendors provides the control software of their devices directly as ROS nodes. Thanks to the simplicity and versatility of CoCo it is very straightforward to transform a component so that it can be used as a bridge between ROS and CoCo. To do so the user has to simply create a CoCo component inside a standard ROS package and compile it as a library. In this way the component can declare a *ros::NodeHandle* object and use it to register or publish topics. Received data can be transformed to be exchanged through CoCo ports. When an application contains a ROS component a different launcher has to be used that is a ROS node embedding the same functionalities and the same behavior of the standard launcher.

3.2.6 *Data Manager Component*

A CoCo component can be used as a data manager, e.g. a component that contains information that are updated and queried asynchronously. Examples of data manager components are the transformation manager (see Section 4), a sensors' parameter manager, or an application specific data handler. Data manager components are used to store information that can be retrieved easily at any time, and they need to provide an interface for updating and querying. In CoCo there are two ways for components to communicate with each other: ports and operations. When using ports, any component that wants to update the data manager needs to connect one of its ports to the data manager update port, and write the update data in it. This approach has the advantage that the data manager can manage the update as it prefers, without blocking the source component. Conversely it has the disadvantage that the updating component has no easy way to know if the update was successful. Another approach is to use operations; operations can be invoked directly, by the updating component, and block until the update is complete. Alternately, the operation can be enqueued with a return callback, so that it is performed in a non blocking manner, but the result can be still received. A data manager query is done via an operation, in order to get the result immediately, even if there is the need to block for the query to complete. In addition, it would even be possible to use an observer pattern for querying. Every task that wants to connect to the data manager registers a callback (via an operation for example) that is called when the data manager is updated. The choice of the best policy often depends on the update/query time, and the size of the exchanged objects.

3.3 COMPONENTS LOADER

There are two ways to instantiate a CoCo application. The first one is to manually (in C++) create an executable where the desired components are instantiated, connected and launched. This has the enormous drawback that every time the user wants to change an attribute value or a connection s/he must recompile all the application. Conversely, CoCo comes with a loader that takes in input an XML configuration file containing the specifications of the application in terms of components, connections and activities. In this way, to update an existing application it is enough to change the XML file and reload it. In addition, it is possible, with an independent loading system, to perform optimization on the application and check for inconsistencies.

However, XML is very verbose to express complex semantic, and this causes the configuration file to increase exponentially when adding new components or connections. Furthermore, the XML syntax doesn't have an intuitive mapping with the semantic of a CoCo application. All of this makes it difficult for an user to write and maintain the XML file.

For these reasons, we have introduced a new formalism to define the components data flow graph and the activities partition. This formalism is based on a Domain Specific Language[2] (DSL) that in the future will be possible to parse and use to instantiate a CoCo application. The advantage of using a DSL is that it allows to create a syntax customized for a specific use case, thus reducing verbosity and increasing readability.

Below the DSL syntax is presented.

Listing 3.1: The DSL syntax.

```
# Component and peer instantiation.
a1 = MyComponent()
# Multiple instantiation of the same component.
a2 = MyComponent()
b = AnotherComponent()
p = MyPeer()

# Attribute initialization.
a1.an_attribute = a_value

# Binding a peer to a component.
a1 << p

# Connecting ports.
conn_policy = ConnectionPolicy(buffer, size,
                               synchronization,
                               transport)
a1.out_port >> b.in_port + conn_policy
# Ports can be connected many-to-many.
a2.in_port << a1.out_port + conn_policy

# Definition of a macro virtual component that
# compose # multiple already instantiated components
# in a sub graph.
MyVirtualComponent = Compose(a1, a2, b)
# Bind a port of a sub-component.
MyVirtualComponent::out_port = b.out_port
# Bind an attribute of a sub-component.
MyVirtualComponent::some_attribute = a1.an_attribute
```

[2]https://en.wikipedia.org/wiki/Domain-specific_language

```
# Allocate a copy of a1, a2, b and their connections.
av = MyVirtualComponent()
# This connects av.a1.out_port to c.in_port
av.out_port >> c.in_port

# Activity instantiation
exe_policy = ExecutionPolicy(periodicity, period)
activity = Activity(policy, a1, a2, ...)
# Farm. A farm is created specifying a component
# (or a virtual one), an input and output port
# and the number of workers.
# The loader will look that there is one and only
# one connection between the worker input port
# and any other component output port.
# Otherwise an error is raised.
farm = Farm(a1, a.in_port, a.out_port, num_worker)
```

Listing 3.2 contains the code to create a connection in the XML configuration file, while Listing 3.3 shows how to create the same connection with the new DSL. This comparison clearly shows the benefit of the DSL, which allows to have a more compact and intuitive definition.

Listing 3.2: The XML code to create a connection between two components.

```
<connection data="BUFFER" policy="LOCK"
        transport="LOCAL" buffersize="10">
        <src task="task1" port="OUT"/>
        <dest task="task2" port="IN"/>
</connection>
```

Listing 3.3: The DSL code to create the same connection as in Listing 3.2

```
policy = ConnectionPolicy(BUFFER, 10, LOCK, LOCAL)
task1.OUT >> task2.IN + policy
```

3.4 VISUO-HAPTICS MODULES FOR MIXED REALITY

The CoCo framework has been used as the infrastructure for the realization of a set of libraries, called CoCo Mixed Reality (CoCoMR), targeting the creation of visuo-haptics applications for MR scenarios. One of the complexities in this kind of applications stands in the different rates at which each component executes. A standard visuo-haptics application can be composed of a module reading frames from a camera at 30 Hz or more, the

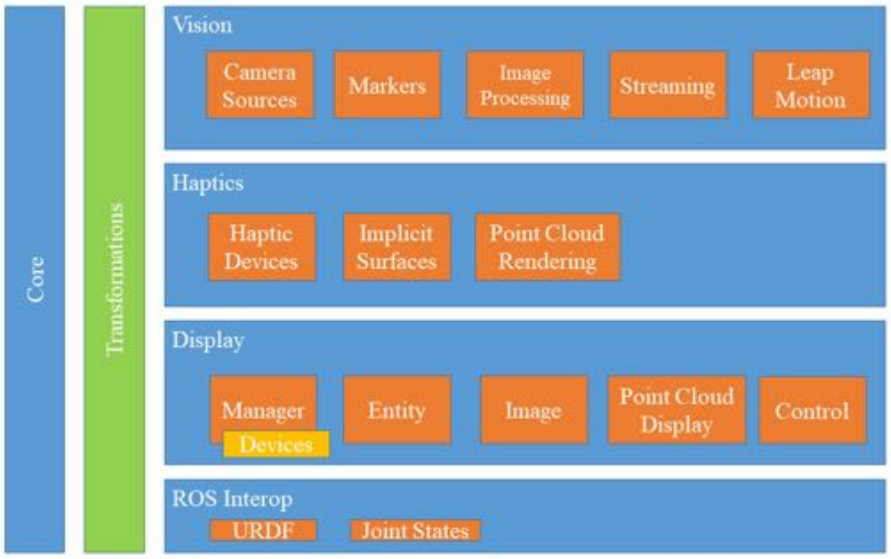


Figure 3.7: Overall view of modules and components

graphics renderer module that runs between 60 to 120 Hz depending on the visualization nature and a module controlling the haptic device running at least at 1 kHz. Three modules sets (Vision, Haptic and Display) have been developed to target each specific scenario, and thanks to the CoCo features they can be combined and customized as desired at run-time. CoCoMR contains also common utilities and a shared interface to allow different components to exchange data through the CoCo ports. An overview of the core modules is shown in Figure 3.7.

3.4.1 *Vision Module*

The vision module provides the services for the computer vision part of an MR application that are the acquisition of image sources, the tracking of features or fiducial markers, and, in the case of tele-presence applications, the streaming.

Components in the vision module exchange data structures that correspond to images, RGB-D images, and camera parameters (intrinsics matrix and distortion). In particular color images are encoded with the possibility of using several color formats (grayscale, RGBA, YUYV and YUV420) with the aim of limiting the conversion from sensors (typically producing images in YUYV) and toward computer vision algorithms that employ often grayscale images. The YUV420 is instead the layout used by video compressors. YUV420 is a planar representation in which Y, U and V are separated with U and V halved in resolution. This means that the per pixel size is 3/2 bytes instead of 3. RGB-D data is stored as a combination of the color image and the depth image (float or signed int16).

Producer components are the following:

- *CameraReader*: using gstreamer or OpenCV captures camera frames and shares them with the other CoCo components.
- *RgbdCameraReader*: same as *CameraReader*, but captures also the depth buffer. The component supports multiple cameras such as Kinect 360 or One, Asus Xtion and Intel R200 directly through libfreenect, OpenNI or libfreesense. For each camera there is a specific CoCo peer embedding the different APIs for each vendor. If provided by the driver, cameras are associated with the intrinsics.
- *LeapReader*: it interfaces the LeapMotion API with CoCo providing to the other components position, orientation and fingers pose of the hands.
- *StreamingReceiver*: it is used to receive via TCP and decodes image and depth streams in case of applications for tele-operation.

Filtering and sink components are:

- *MarkerTracker*: receives a camera buffer and produces in output the pose of eventual markers present in the image.
- *MeshReconstructor*: receives the image and the depth buffers from *RgbdCameraReader* and creates a mesh interpolating the missing points.
- *StreamingServer*: receives an image and/or a depth buffer and sends it via TCP. Must be coupled with *StreamingReceiver* at the client side.

3.4.2 *Haptic Module*

The haptic module provides *haptic rendering* that is the generation of force feedback following the convention of computer graphics rendering [112]. There are several challenges in haptic rendering tasks mainly relating to the high rate of the update loop, 1kHz, and especially when the interacting

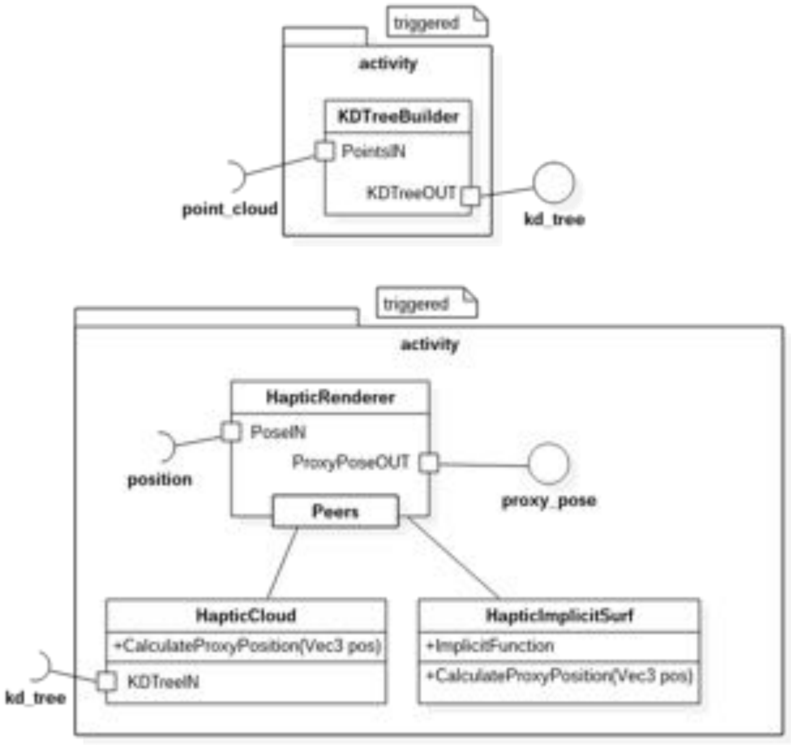


Figure 3.8: General structure of the haptic module.

surface is dynamically updated as coming from a RGB-D camera. CoCo provides internally the support for the haptic rendering of implicit surfaces or point clouds. Other techniques can be integrated such as volumetric voxel models [102] or the 3DOF spherical proxy algorithm [109] for triangular meshes to external libraries such a CHAI3D.

The rendering of implicit surfaces is based on the Salisbury algorithm [113] that updates a contact proxy based on an implicit surface described as a distance to the surface and the local gradient. CoCo supports the creation of procedural implicit surfaces expressed on constructive solid geometry over building blocks such as cylinders, planes and spheres. The *HapticImplicitSurf* component is configured over a functional description of the surface, and it tracks the proxy over the surface, generating force feedback with friction parameters.

Contact rendering of live point clouds, or point-sampled meshes, is based on the identification of the points around the proxy via k-d tree and the creation of a local surface from such points. The *KDTreeBuilder* component is responsible for the creation of the k-d tree either from a point cloud or the vertices of a meshes provided by the adapter *MeshReconstruction* component. The *HapticCloud* component provides haptic rendering over these points using the k-d tree emitted by the *KDTreeBuilder*. This is an example of the large resource management of CoCo: the KDTreeBuilder has an input port with the new point cloud and inside its loop it performs a slow update, while, at the same time, the last value of the k-d tree is available over the output port. The port mechanism allows the reuse of the values in the ports, much like happens in a classical front-back buffer, but only in a more general way.

For a complex scene with multiple surfaces (or layered materials) it is possible to coordinate different rendering components, or aggregate them inside a single component, called *HapticRenderer*, which invokes the various renderers using the peer mechanism. As discussed, anyway, the peers share the same activity, meaning the same OS thread. The module structure and its connections are shown in Figure 3.8.

3.4.3 *Display Module*

The Display module has been developed to provide a reasonably good visualization capability for MR applications with the main idea of displaying images or point clouds produced by a set of cameras looking more at performance than display visual effects. There is no intention to replicate features found in more sophisticated 3D engines, such as shadows or large complex models.

The display module is based on OpenGL 3.3 and it is responsible for the rendering of 3D objects and the eventual images obtained from cameras. It is composed of a single component and several peers, mainly due to the single-threaded nature of the OpenGL API. Multi-threaded OpenGL could be an option but it is known to impact the performance of the overall 3D rendering, and multi-threading can be exploited only for the memory transfer between CPU and GPU buffers, e.g. for uploading point cloud or texture data. The recent introduction of the Vulkan API[3] has opened the way for multi-threading with GPU and it could be an interesting enhancement for the graphics part of CoCo. The component (*GLManager*) is the graphics manager and it is in charge of initializing the OpenGL context and the rendering window across a variety of devices. The OpenGL camera and every element that has to be rendered are associated to a peer. *GLManager*

[3]https://www.khronos.org/vulkan/

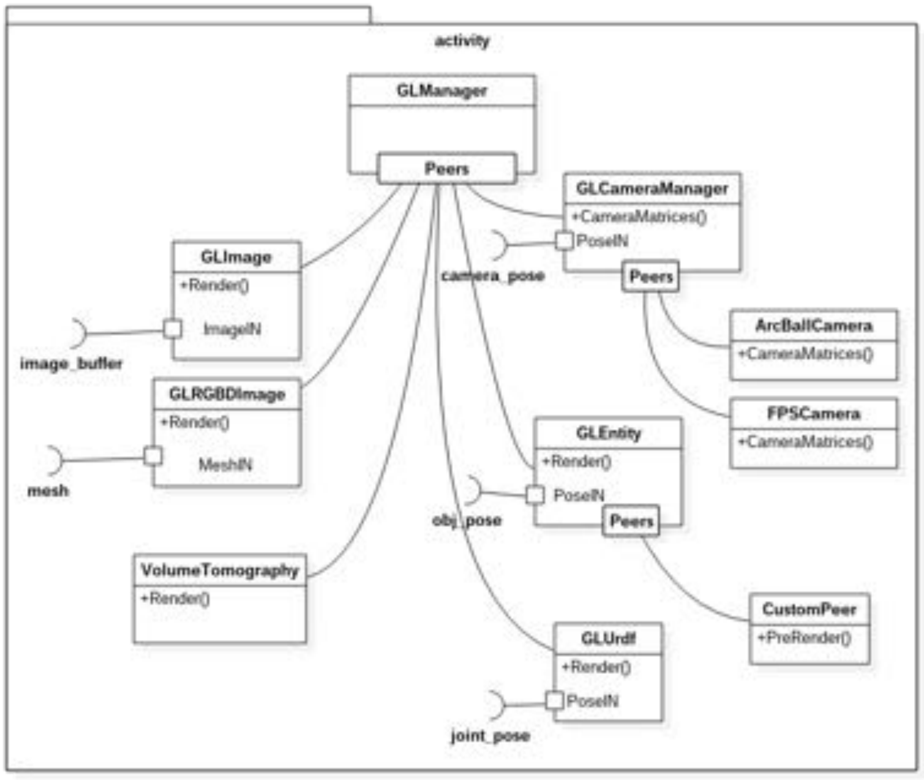


Figure 3.9: General structure of the display module.

queries the camera peer for the projection/view matrices and then iterates over all the other peers calling their rendering function.

When instantiating the *GLManager* component it is possible to specify the frame rate, setting the desired period in the activity containing it, the window resolution and the visualization type covering 2D, 3D stereo and Oculus Rift DK2. CoCoMR supports the creation of multiple visualization windows in Linux by instantiating at run-time one *GLManager* component for each desired display. Furthermore *GLManager* can render on texture and produce the result through a port allowing the streaming of the visualization scene or using CoCoMR as the input for CV algorithms that require the synthetic rendering of the estimated entity (e.g. hand's pose).

The decision of creating a custom graphics module, instead of using the several existing ones, was due to strict requirements of AR application in HRI, such as the compatibility with Linux and the possibility to communicate fast and efficiently with custom and commercial sensors.

3.4.3.1 *Camera*

The *GLCameraManager* peer, one per *GLManager*, is in charge of managing the OpenGL camera, specifying the initial position of the camera through CoCo attributes and the type of camera through additional peers. Camera controllers are also expressed as peers: first person shooter style camera (*FPSCamera* peer) and arcball camera (*ArcBallCamera* peer). Camera can be moved either using mouse and keyboard or by sending the desired position to the *GLCameraManager* dedicated port.

3.4.3.2 *Camera Images*

Two peers are available to render images provided by cameras. *GLImage*: renders a 2D image in the background of the virtual world. The image is scaled to fit the resolution of the window. *GLRGBDImage* renders the 3D scene obtained from the *MeshReconstructor* component. In case of very noisy meshes it supports the possibility to average the position of the mesh points among multiple sequential frames. Furthermore with the support of a geometric shader it is possible to clean the scene removing the big triangles that connect objects far from each others, which are introduced by the triangulation algorithm in *MeshReconstructor*.

3.4.3.3 *3D Objects*

The *GLEntity* peer is used to render any mesh in the 3D world. It supports all the standard formats and provides a set of basic shaders to support textures and lights. *GLEntity* exposes several attributes to set the object's initial pose and scale, the eventual color if not present in the mesh file and the possibility to run a subdivision algorithm on the object's surface. *GLEntity* can be further specialized associating custom peers to it. A peer, to be supported, has to expose an operation named *preRender* that takes as input a pointer to the *GLEntity* object. The operation is called by the component before the OpenGL draw function and can be used to modify the standard behavior of a virtual object. For example it could be possible to alter its color according to external information, to change the mesh shape or to modify the pose.

The display module also supports the rendering of Universal Robot Description Format (URDF) objects from ROS through the *GLUrdf* peer. This feature is very useful when performing robot teleoperation because it allows to easily check that the camera mounted on the robot and the robot itself are correctly registered. Details of registration are provided in the following section. Furthermore it allows to have a clear vision of the robot pose when the camera only focuses on the end-effectors. It can also be used to simulate a robot in a pure virtual environment.

3.4.4 *Components Classification*

Components can be classified according to different parameters. The first one is the activation type that can be either periodic or triggered. In the first case the component is executed according to a timer, while in the second case upon reception of data. The second parameter for components' classification is based on their category. The category can be producer, worker or sink, as we have seen in Section 3.2.1. Activation type and category are strictly related, as a producer is always periodic, and a worker is triggered. Producer components don't usually have input, with the notable exception of the *HapticRender* task. In this case the component takes as input the k-d tree generated from *KDTreeBuilder* task, and a position, to compute the proxy. However, both the k-d tree and the pose don't trigger the execution, as the proxy needs to be computed at a constant rate to avoid jitter and delays. Every time the rendering task is activated it checks if a new k-d tree is available and queries the transformation manager for the latest desired pose, and then computes the position of the proxy, and outputs it. The last classification parameter is the type of the exchanged data. These types can be big, as images, meshes and k-d tree, or small as poses and camera parameters.

Table 3.1: Components classification.

Group	Component	Activation	Category	Output Data
Vision	*CameraReader*	Periodic (30/60 Hz)	Producer	Big (RGB image)
	RgbdCameraReader	Periodic (30 Hz)	Producer	Big (RGB + depth image)
	LeapMotionReader	Periodic (100 Hz)	Producer	Small (Hand Pose)
	StreamingReceiver	Periodic (30/60 Hz)	Producer	Big (RGB + depth image)
	MarkerTracker	Triggered	Worker	Small (Pose)
	MeshReconstructor	Triggered	Worker	Big (Mesh)
	StreamingServer	Triggered	Sink	-
Haptic	*KDTreeBuilder*	Triggered	Worker	Big (k-d tree)
	HapticRenderer	Periodic (1 kHz)	Producer	Small (Proxy Pose)
	HapticDeviceDriver	Triggered	Sink	-
Display	*GLManager*	Periodic (60/120 Hz)	Sink	-

3.5 CONCLUSION AND FUTURE WORK

The diffusion of multicore platforms is requiring new approaches for organizing computation, in particular for demanding tasks such as visuo-haptic mixed reality applications. The presented CoCo framework provides a solution to these challenges. The organization based on the two independent graphs (connections and execution) allows to tackle these challenges by providing flexibility and control, while hiding several low-level aspects.

There are several aspects that can be investigated starting from the present work. One aspect is related to the analysis and optimization of the scheduling resulting from the data flow and the user-defined schedule. This could give space for the identification and reuse of common patterns and the adaptation to a new machine with a different number of cores. In addition, studies could be made to automate the partition problem, in such a way that the framework could decide autonomously the number of running threads, and how to group the components. This will provide even more abstraction to the user, while maximizing performance and minimizing human error.

The second aspect is instead related to the support of GPUs in the data flows. The most promising solution is based on CUDA[4] mainly due to the number of libraries in the vision and simulation world that provide optimization for such library. The port mechanism could be easily extended for supporting CUDA pointers, but it is necessary to introduce an automatic mechanism for transferring the content from/to the GPU when a connection is created between a GPU-bound port and a CPU-bound port. This proposed solution is clearly limited to single-process architecture and it is also subject, in terms of scheduling, to the policies of the CUDA driver.

CoCo could also be extended to support multiprocess, providing an efficient mechanism for data exchange between components in different processes. ZeroMQ is an optimal candidate, while reduced or no-serialization could be employed for efficient data exchange.

3.6 PUBLICATIONS

Emanuele Ruffaldi and Filippo Brizzi. “Coco-a framework for multicore visuo-haptics in mixed reality”. In: *International Conference on Augmented*

[4]https://developer.nvidia.com/cuda-zone

Reality, Virtual Reality and Computer Graphics. Springer International Publishing. 2016, pp. 339–357

Emanuele Ruffaldi, Filippo Brizzi, Giacomo Dabisias, and Giorgio Buttazzo. “SOMA: an OpenMP toolchain for multicore partitioning”. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing.* ACM. 2016, pp. 1231–1237

4 TRANSFORMATION GRAPHS AND CALIBRATION

Transformations between reference frames are a fundamental element of any MR application in particular when multiple image sources are used together with other tracking and robotics devices. Transformations management performs the estimation of unknown transformations (calibration), the efficient propagation of dynamic transformations in a complex setup, the management of uncertainty when these transformations are generated by noisy sensors, and the integration of multiple sensors concurring in the same measure (fusion). Calibration is particularly important in MR for HRI, as the robotic system and the camera(s) setup must be aligned and connected to provide a correct feedback [40, 59].

This chapter presents the work done to create an easy to use and efficient tool for managing transformation, calibration and fusion in transformation graphs. This is constituted by a general model to express transformations and their uncertainty, and a software framework for designing, debugging and supporting at run-time the execution of MR applications.

The choice of using a graph to represent frame is a natural choice and common in the robotic field in particular when combining virtual entities with robots joints it is easy to reach several tens of frames. Graphs are by their own nature modular, allowing to easily combine together sub-graphs coming from different sources (graphics entities, CV and robot kinematics). This modularity allows also to reuse sub-graphs in different applications, which can be plugged together in different configuration. Furthermore, graphs are easy to visualize, helping to debug and monitor applications.

Graph operations such as transformation queries and calibrations can be expressed as algebraic problems in the Lie Algebra with observations as inputs and known transformations as constraints. Another possible approach would be to treat transformations query and calibrations as a global. These algebraic problems can be solved as least square optimization problems using non-linear sparse Levenberg Marquardt (LM) optimization over manifold [60]. The LM formulation should take into account uncertainties associated to observations and known transformations. In this work we choose to focus on specific calibration patterns solved using graph traversal and specific settings, but they could be generalized to a general optimization problem.

Examples of specific calibration tasks can be found in literature for AR [40, 59].

4.1 RELATED WORK

As we have seen in Section 2.1.1, the two most common models used for creating VEs are SG and DF. In both cases the transformations of reference systems are embedded in the application structure and sometimes mixed with other entities. This is specifically relevant for DF structures in which the final organization is not self explanatory. Instead, a clear schema of the transformations and how they are related is useful not only for debugging and analysis purposes, but also for dealing with unknown transformations that can be solved via a calibration procedure.

The OSGAR system [28] presents an adaptation of the Open Scene Graph library for supporting uncertainty in AR applications. The system aims at managing the transformations present in the AR application, considering transformations as generic $4x4$ matrices, with a noise model that is based on a purely additive $16x16$ covariance matrix. In this way, the semantic of the noise model, behind the affine transformation, can be discarded. Transformations comprising uncertainty are propagated over the graph, and specialized nodes support fusion of multiple sources.

In the robotics domain the tf2 package [51] of ROS is an excellent example of transformation graph framework. The tf2 package provides a service for structuring transformations in a graph, allowing the developers to perform queries between any pair of nodes of the graph. The supported transformations correspond to rigid body motions (rotation and translation) without uncertainty. When the tf2 is paired with the ROS visualization tool RViz[1] it allows to visualize the reference systems together with other information such as robotic models, visual markers and point cloud data.

4.2 SPATIAL RELATIONAL GRAPH

Spatial transformations and reference systems in VE are based on the concepts of rigid body transformations in physics and robotics. Each rigid body, or in general virtual entity, in the VE is associated to a reference frame, and two bodies are related by a six-dimensional displacement, three

[1]http://wiki.ros.org/rviz

values for rotation and three for translation. In VE the spatial displacement is not sufficient and in some cases entities are related by a scaling factor or a more general transformation such as a projection or distortion.

The Spatial Relationship Graph (SRG) [84] is a model that allows to describe the relationships between spatial entities in AR and MR applications. The original authors of SRG have extended the concept identifying patterns in the Spatial Relationships for a variety of AR and MR scenarios [96] and later integrated them in a general management framework [68].

In SRG the reference frames associated to the entities in the VE can be organized in a Direct Acyclic Graph (DAG) in which the nodes represent spatial frames (F). The directed edges of the graph represent the relative poses, or more general, the transformations between nodes (T). A sequence of contiguous nodes in the SRG is called a kinematic chain.

The nodes are built from the key entities of the VE, extracting them from the underlying VE structure, being it SG or DF. The directed edges correspond to the transformations that are expressed by the VE structures, and they can be considered as semantic constraints between node pairs. An alternative representation to this frames-transformation graph is the one of multi-graphs in which two type of nodes are present: reference frames and transformations. In a multi-graph the edges connect frames and transformations (F-T) or couples of transformations (T-T). In this work we chose to use the single graph notation to keep the graph more compact.

In the adopted SRG model the transformations expressed by the edges are associated to a series of properties that are used to support design, documentation and debugging as discussed later.

- Transformation type: the nature of the transformation, a relative pose between rigid bodies or a single robotic-joint.
- Constraint: a constraint over the transformation such as limiting the transformation to an axis.
- Uncertainty: the nature of the probabilistic model of the transformation.
- Temporal Semantics: whether the transformation may change and how over time.

The transformation type is internally expressed as a general 3D pose, but it can be specified as a single prismatic or revolute joint along an axis, since it is a common pattern found in robotics. Uncertainty expresses the confidence level of the transformation spanning from totally known transformations to lack of knowledge as discussed later.

Temporal semantics is important because it is the cornerstone of optimization, calibration and sensor fusion. As will be discussed in the specific sessions, an edge can be: Fixed, Dynamic, Calibration, Fusion or Temporary.

4.3 GEOMETRICAL REPRESENTATION

In this section we are going to discuss how transformations have been represented, together with the techniques to compute uncertainty and composition of multiple transformations.

4.3.1 *Lie Group*

The $SE(3)$ Lie group [18] has been used to express transformation. $SE(3)$ denotes the group constituted by the set of all the $4x4$ matrices with the following structure:

$$T = \left(\begin{array}{c|c} R & t \\ \hline 0_{1x3} & 1 \end{array}\right) \tag{4.1}$$

with $R \in \mathbb{R}^{3\times3}$ orthogonal matrix with determinant of ±1 (i.e. $RR^T = R^T R = I_3$) representing the rotation, and $t = [t_x t_y t_z]^T \in \mathbb{R}^3$ for the translation.

If we consider this set of matrices with the binary operation defined by matrix multiplication, it is easy to see that $SE(3)$ satisfies the four axioms that must be satisfied by the elements of an algebraic group:

1. **Closure.** The set is closed under the binary operation; for any $A, B \in SE(3)$, we have $AB \in SE(3)$.
2. **Associativity.** For $A, B, C \in SE(3)$, $(AB)C = A(BC)$.
3. **Identity element.** There must exists an identity element $I \in SE(3)$, such as $IA = AI = A$ for each $A \in SE(3)$.
4. **Inverse.** $\forall A \in SE(3)$ there must exist an inverse element A^{-1} such that $AA^{-1} = A^{-1}A = I$.

In addition $SE(3)$ is a Lie group as it fulfills:

1. $SE(3)$ is a group in $\mathbb{R}^N$.
2. $SE(3)$ is a manifold in $\mathbb{R}^N$.
3. Both, the group product operation and its inverse are smooth functions.

A manifold in $\mathbb{R}^N$ is a topological space where the neighborhood of every point p is homeomorphic to $\mathbb{R}^N$.

4.3.2 *Lie Algebra*

A Lie algebra is an algebra $\mathfrak{g}$ together with a binary operation $[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$ called the Lie bracket that satisfies the following axioms:

1. **Jacobi identity.** $[x,[y,z]] + [z,[x,y]] + [y,[z,x]] = 0$ for all x, y, z in $\mathfrak{g}$.
2. **Anticommutativity.** $[x,y] = -[y,x]$ for all elements x, y in $\mathfrak{g}$.

The Lie algebra $\mathfrak{g}$ associated to a Lie group G happens to be the tangent space at the identity element I.

Associated to a Lie group G and its Lie algebra $\mathfrak{g}$ there are two functions that allow to convert from one to the other:

- **Exponential map.** It allows to map element from the algebra to the manifold and determines the local structure of the manifold: $exp : \mathfrak{g} \to G$.
- **Logarithm map.** It maps elements from the manifold to the algebra: $ln : G \to \mathfrak{g}$.

In the case of the Lie algebra $\mathfrak{se}(3)$ let

$$v = \begin{pmatrix} t \\ \omega \end{pmatrix} \tag{4.2}$$

denote the 6-vector of coordinates in the Lie algebra $\mathfrak{se}(3)$, comprising two separate 3-vectors: ω, the vector that determine the rotation, and t which determines the translation.

We introduce the skew matrix operator $[\cdot]_\times$, which is defined as:

$$\left[\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right]_\times = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \tag{4.3}$$

and its inverse operation $[\cdot]_\nabla$:

$$\left[\begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}\right]_\nabla = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.4}$$

Furthermore, we define the 4×4 matrix:

$$A(v) = \begin{pmatrix} [\omega]_\times & t \\ 0 & 0 \end{pmatrix} \tag{4.5}$$

Then, the map, $exp : \mathfrak{se}(3) \to SE(3)$, is well-defined, surjective, and has the closed form:

$$e^v \equiv e^{A(v)} = \begin{pmatrix} e^{[\omega]_\times} & Vt \\ 0 & 1 \end{pmatrix} \tag{4.6}$$

$$V = I_3 + \frac{1-\cos\theta}{\theta^2}[\omega]_\times + \frac{\theta - \sin\theta}{\theta^3}[\omega]_\times^2 \tag{4.7}$$

with $\theta = |\omega|$ and

$$e^{[\omega]_\times} = I_3 + \frac{\sin\theta}{\theta}[\omega]_\times + \frac{1-\cos\theta}{\theta^2}[\omega]_\times^2. \tag{4.8}$$

The logarithm map:

$$\begin{aligned} ln : SE(3) &\to \mathfrak{se}(3) \\ A(v) &\to v = \begin{pmatrix} t' \\ \omega \end{pmatrix} \end{aligned} \tag{4.9}$$

is well-defined and can be computed as [137]:

$$t' = V^{-1}t \qquad \text{(with V in Eq. 4.7)} \tag{4.10}$$

$$\begin{aligned} \omega &= [ln(R)]_\nabla \\ ln(R) &= \frac{\theta}{2\sin\theta}(R - R^T) \end{aligned} \tag{4.11}$$

The exponential form allows for a compact representation of the poses based on the 6 values of the associated $\mathfrak{se}(3)$ algebra. This representation is interesting because it easily supports interpolation among different relative poses, generalizing the properties of quaternion spherical interpolation to a generic Lie group [121].

4.3.3 *Uncertainty*

There are three main sources of uncertainty in typical mixed-reality and robotic applications: robot's joint precision, camera based tracking, and relative placement of sensors. Being able to measure it provides a way to evaluate the correctness of algorithms and the possibility to compare them.

The choice of representing transformation using the $SE(3)$ Lie group was driven also by the possibility of extending the exponential representation to a Gaussian distribution that is centered in the given average transformation and has a covariance that is expressed in the 6-dimensional algebrical space of the $SE(3)$ Lie group.

This probabilistic representation allows to perform the operations of the group such as inversion and composition. In addition, it has a compact form for the combination of multiple measures as happen in the cases of sensor fusion or calibration.

In the following we will use the notation $\tilde{q} \equiv N\left(\mu_q, \Sigma_q\right)$ for defining a Gaussian variable q with covariance Σ_q and mean μ_q, where the mean is expressed in $SE(3)$ and the covariance in $\mathfrak{se}(3)$.

The composition of two transformations (product) is:

$$\tilde{z} \equiv \tilde{x}\tilde{y} = N\left(\mu_x\mu_y, \Sigma_x + \mathrm{Adj}_{\mu_y}\Sigma_x\mathrm{Adj}^T_{\mu_y}\right) \tag{4.12}$$

here Adj_G is the adjoint representation of the Lie group G [118].

The distribution of the inverse transformation of the uncertain transformation x is the following:

$$\tilde{x}^{-1} = N\left(\mu_x^{-1}, \mathrm{Adj}_{\mu_x^{-1}}\Sigma_x\mathrm{Adj}^T_{\mu_x^{-1}}\right) \tag{4.13}$$

In addition it is possible to compute the estimation of the average pose from a set of measurements not affected by noise, which is an iterative operation that has some connections with the mean shift algorithm. The algorithm starts from a prior (e.g. first element of the series) and computes the distance to any element of the series in algebraical space. Be, $x_i \in SE(3)$ the transformation of the set with index $i = 1 \ldots N$, k the iteration index, μ_k the resultant mean transformation at iteration k with covariance Σ_k, and $\mu_0 = x_0$, we have:

$$v_{i,k} = ln\left(x_i \mu_k^{-1}\right) \tag{4.14}$$

$$\Sigma_k \leftarrow \frac{1}{N}\sum_i^N v_{i,k} v_{i,k}^T \tag{4.15}$$

$$\mu_{k+1} \leftarrow exp\left(\frac{1}{N}\sum_i^N v_{i,k}\right)\mu_k \tag{4.16}$$

The fusion of two different sources of transformation with uncertainty can be expressed as follows [8]:

$$\tilde{z} \equiv \tilde{x} \oplus \tilde{y} = N(\mu_z, \Sigma_z) \tag{4.17}$$

where

$$\Sigma_z = \left(\Sigma_x^{-1} + \Sigma_y^{-1}\right)^{-1} \tag{4.18}$$

and

$$\mu_z = exp\left[\Sigma_z \Sigma_x^{-1} log(\mu_y \mu_x^{-1})\right]\mu_x \tag{4.19}$$

4.4 CALIBRATION

Calibration is the task of computing an unknown transformation via a series of measurements done for this purpose. Examples are the one of multiple cameras that look at the same scene, the calibration between a projector and a camera in a projective AR system, a robot equipped with a camera located at the end-effector of the arm, or instead a camera placed in the head of the robot. For a robot the reference frames of an arm are connected with each other via transformations computed thanks to the measurement of the joint angles.

The calibration task in the SRG has the objective of computing the unknown fixed transformation edge that connects two reference frames, which stay in two disjoint sub-graphs. This can be done by artificially creating a temporary connection between the two sub-graphs with the help of a fiducial marker, for which we can compute the transformation from two frames in the two sub-graphs. Once we have the temporary connection we can compute the exact transformation between the two desired frames and store it in the graph. SRG supports two calibration approaches that will be presented in the following two sections.

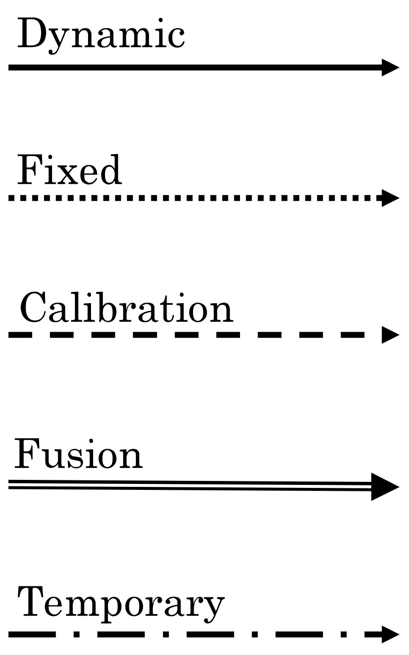


Figure 4.1: The figure shows the convention that will be used in the representation of the edges based on the temporal semantic of the associated transformation.

Figure 4.1 shows the convention used to represent each edge based on the temporal semantics of the associated transformation. This convention will be used in all of the following transformation graphs representation.

4.4.1 *Helper-Based Calibration*

The first approach relies on introducing a temporary transformation that allows to connect the two frames objective of the calibration and via repeated measures computes the missing edge. More formally given two frames R_A and R_B we have a calibration problem when there is no path in the graph between them, so we introduce a new reference frame R_C that is connected via "temporary" transformations to both R_A and R_B. The accumulated measures T_{AB} are used to compute an estimate transformation, and then the R_C frame is being removed.

An explicative scenario is the one of a robot equipped with a camera whose transformation to the rest of the body is unknown due to construction issues and lack of information about the optical center of the camera. We have the robot base R_R, the camera frame R_C and the robot arm end-

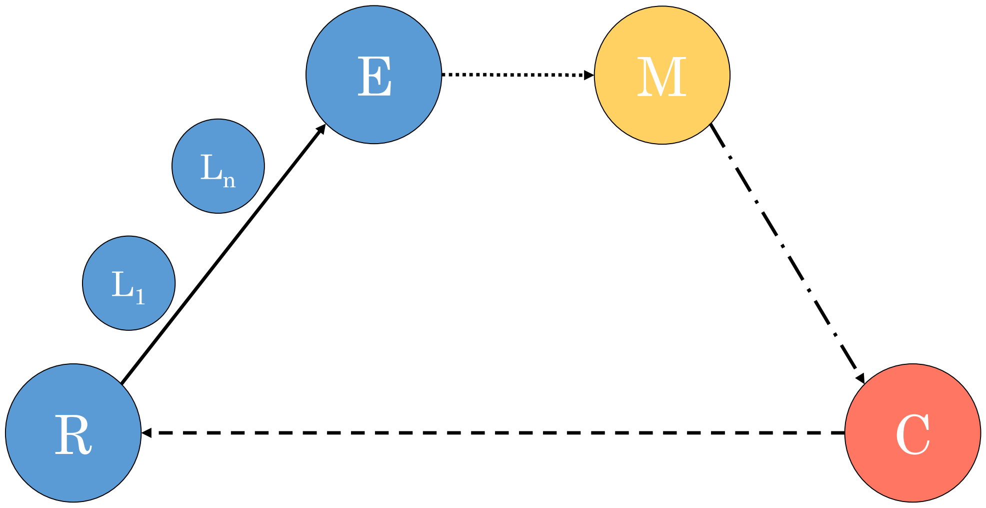


Figure 4.2: Diagram of the first form of calibration that closes the loop with a temporary fixed edge corresponding to a marker attached in a position nearby the end-effector. The diagram shows the target calibration edge as dashed, the end-effector transformation that depends on the other links, and the measured marker position.

effector R_E whose transformation to the R_R is given by the joint measures connecting a series of intermediated frames R_{L_i}. The calibration can be obtained by applying a fiducial marker (e.g. Aruco [54]) nearby the end effector and then move the end effector. This means that we introduce a R_M temporary frame and two temporary transformations: T_{CM}, computed by the vision algorithm and T_{ME}, manually measured and constant (see Figure 4.2). The result is that we can temporarily compute the path between the root and the camera T_{RC}, and when the calibration process is completed the temporary frame is removed from the SRG. Thanks to the probabilistic framework the partial knowledge of T_{RC} from the mechanical design can be added as a prior and as a boundary to the calibration.

In this framework the calibration is expressed by using the following edge types:

- Calibration(E,C): fixed edge, objective of the task.
- Temporary(C,M): dynamic temporary edge used for closing the loop.

The algorithm is described in Algorithm 1.

Algorithm 1 Helper Calibration

```
1: procedure CALIBRATE
2:     Input: $T_{RC}$ calibration
3:     Input: $T_{CM}$ temporary
4:     repeat
5:         Trigger on new data $T_{CM}$
6:         Accumulate transformations $T_{RC}$ via temporary
7:     until convergence
8:     Output $T_{RC}$ as fixed and remove $T_{CM}$
9: end procedure
```

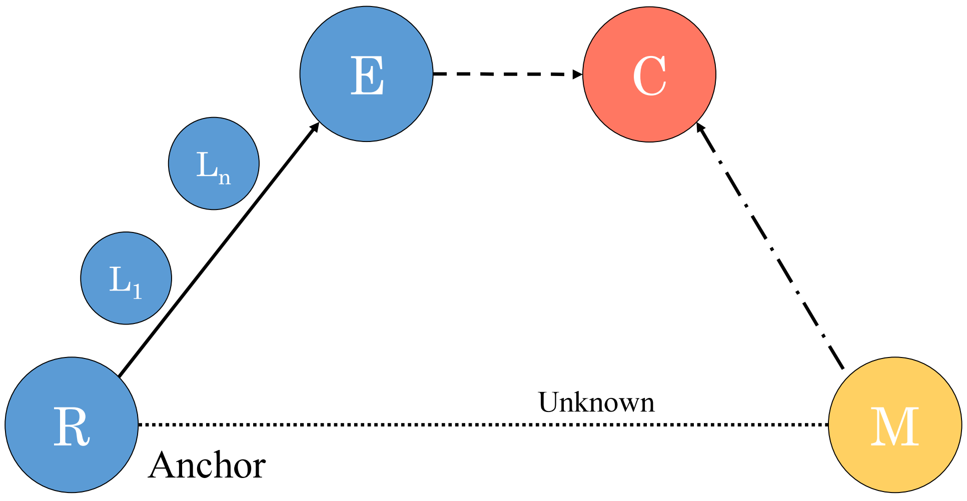


Figure 4.3: Diagram of hand-eye calibration that is based on measurements at different time instants. This operation involves two dynamic edges T_E^R and T_M^C and computes the two fixed unknowns

4.4.2 *Motion-Based calibration*

The hand-eye calibration can be employed when there is no way to artificially create a loop in the graph for obtaining the calibration data. An example is when the camera is placed at the end-effector of a robot's arm, and it cannot frame any robot's part. The approach has been originally introduced by Tsai [129] and then extended to Dual Quaternions by Daniilidis [34], and it relies on creating a series of robot's arm poses during which the camera, mounted on the robot, looks at an external fixed fiducial marker. For each pair of poses of the robot's arm it is possible to formulate an equation with the form $AX = XB$ where X is the unknown transformation and A, B are two transformations obtained from two different instants of time.

The formulation in terms of SRG is straightforward: the unknown is the fixed pose of the camera related to the end-effector T_{EC} with robot kinematics providing T_{RE}. The algorithm employs an external fixed fiducial marker (or chessboard) called M for which we obtain the transformation T_{CM}. The use of a fixed marker allows to define a loop of reference frames E, C, M, R with two unknowns T_{EC} and T_{RM}, see Figure 4.3. Given two instants of time t_1 and t_2 with different robot's arm poses $T_{RE}^{t_i}$ and marker poses $T_{CM}^{t_i}$, we express the loop in the two instants of times and equate them brining us to the equation in the form $AX = XB$ with $X = T_{EC}$.

$$T_{MC}^{1} X T_{ER}^{1} = T_{MC}^{2} X T_{ER}^{2} \tag{4.20}$$

$$T_{CM}^{2} T_{MC}^{1} X = X T_{ER}^{1} T_{RE}^{2} \tag{4.21}$$

In comparison to the other calibration method, the hand-eye calibration requires the introduction of another input for supporting the correct execution of the algorithm. This is a reference frame that is fixed to the temporary frame M, called calibration anchor. In the example it corresponds to R.

The resolution of the equation via the algorithm [34] is not taking into account the noise of the measures, in particular for the robot the noise due to joint encoders affecting T_{RE} and the camera limitations for T_{CM}. A possible approach for taking into account noise is to employ the Sigma point transformation of the $SE(3)$ distribution that allows to perform a non-linear function [8]. This operation should be performed for every pair of poses of the equation $AX = BX$.

Algorithm 2 Hand-eye Calibration

```
1: procedure CALIBRATE
2:     Input: $T_{EC}$ calibration
3:     Input: $T_{CM}$ temporary
4:     Input: $R$ as calibration anchor
5:     repeat
6:         Trigger on new data $T_{CM}$
7:         Accumulate transformations $T_{RE}, T_{CM}$
8:     until convergence
9:     Output $T_{EC}$ as fixed and remove $T_{CM}$
10: end procedure
```

4.5 SENSOR FUSION

Sensor fusion is in general the combination of multiple sensor information providing the same information. It differs to the calibration process because calibration has the objective to compute a fixed transformation via multiple measures in time. The fusion operation comes straightforward in $SE(3)$ thanks to the probabilistic formulation discussed previously: contributions can be fused using the operations presented previously. We discuss in the following the mechanism used for the automatic computation of such fusion in the SRG.

The declaration of a fusion edge T_{AB} has the effect of computing all the paths between the two frames (A and B) and then, based on a user-specified update policy, the edge is updated using the information from the paths. The policy specifies when to trigger the function operation, meaning that it can be triggered on any update of the paths, a specific path, on a time basis or on request of the fused edge. As previously discussed the query mechanism uses the shortest path between two frames and, when the fusion edge is active, any further query will go through it, instead of passing by any of the other paths.

A simple but interesting example of fusion is provided by a surface covered with multiple fiducial markers M_i each placed in a fixed position with respect to the surface frame S (T_{SM_i}). We are interested in computing the fused pose of the camera C with respect to the surface that is T_{CS}. After declaring T_{CS} as sensor fusion, all the paths $P_i = (T_{CM_i}, T_{M_iS})$ are computed. When fusion is triggered each path's transformation is used for updating T_{CS}. Furthermore it is possible to exclude one of the paths if its covariance is to high in order to avoid soiling the final result.

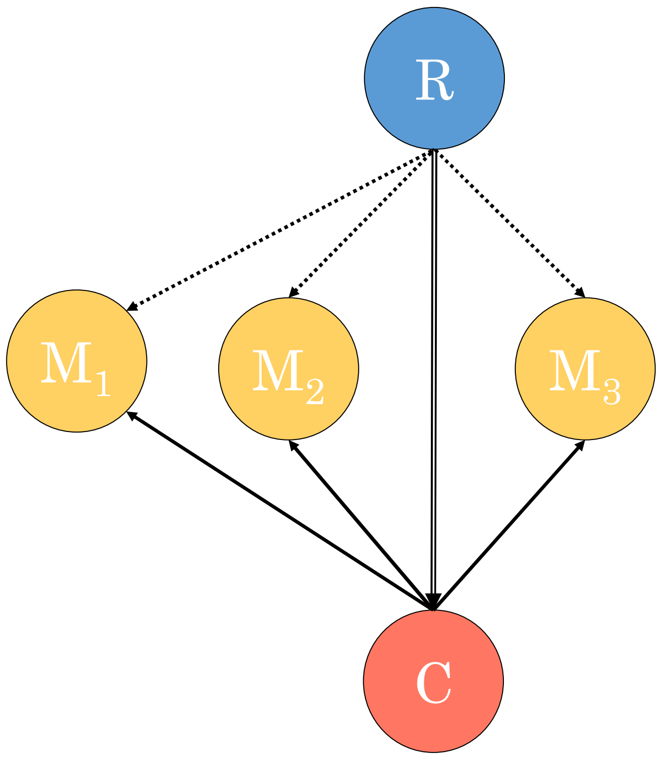


Figure 4.4: Example of a fusion scenario where a user wants to keep constant track of the position of a moving camera with respect to a robot. In the scene several markers are present whose pose with respect to the robot is known. At runtime the camera detect the markers and calculate $T^i_{RC} = T_{RM_i} * T_{M_iC}$ and compute the average transformation among all the i.

Figure 4.4 shows a scenario where a user wants to measure constantly the position of a moving camera with respect to the robot. To do so we place several markers close to the robot whose pose is known with respect to the robot reference system. The transformation robot-markers is assumed to be fixed, but this restriction can be relaxed if a system to constantly track marker pose is provided. When a marker is detected by the camera, the algorithm can compute its transformation with respect to the robot. In order to improve the quality of the result the system merges the results obtained from each of the visible markers by averaging them.

4.6 SOFTWARE IMPLEMENTATION

SRG was implemented as a C++ library called SRGViz. It supports two modalities: dynamic or static. In the first case the transformation graph is created on the fly, adding new frames or transformations every time a new update arrives. This approach is similar to the one used by ROS TF2, and has the advantage of speeding up the prototyping. On the other hand it doesn't allow for optimization, calibration and fusion. For this reason a static modality was introduced where the transformation graph needs to be specified at loading time via a JSON file. The file contains a list of all the nodes and all the transformations. Each transformation is associated with a set of properties, which are the initial value with the uncertainty, the temporal semantic (fixed, dynamic, temporary, calibration, or fusion), and the eventual constraint (single axis degree of freedom).

4.6.1 *Query Resolution*

The fact that the graph is static and the temporal semantic associated with each edge allow to perform important optimizations. For each query between two frames a breadth first search is performed to find a path between the two frames. The list of frames and transformations that constitutes the path is stored in a map with the queried frames as key. In addition, all consecutive fixed transformations are merged in a new single fixed transformation equal to their multiplication. This allows to reduce the number of matrix multiplications in successive queries of the same path. Furthermore, every time the same query is performed, instead of performing again the search, the framework returns the path from the query map. This is an important optimization, given that in an AR application for HRI the queries are always for the same 3D objects or robot's end-effector.

The library supports also all the operations described previously, as composition with uncertainty, and averaging of multiple certain poses.

4.6.2 *Calibration and Fusion*

To perform a helper-based calibration the user needs to specify the calibration edge (i.e. the edge for which we want to know the transformation), which must be fixed, and the temporary edge that is used to close the loop in the graph. For motion-based calibration, the user has also to specify the frame used as anchor. The library set up the graph so that, every time the temporary edge is updated, the algorithm 1 or 2 is triggered.

For fusion the approach is similar as the user specifies in the JSON file the fusion edge. The framework then computes all the disjoint paths that connect the source to the target frame of the fusion edge. The found resulting paths need to contain one temporary edge each, whose update is used to trigger the update of the fusion edge transformation.

4.6.3 *Interoperability*

SRGViz library supports single axis as it is specifically useful for robotics or human tracking in which the update value is a single joint value, and the 6DOF transformation is the resulting application of a Denavit-Hartenberg transformation or equivalent.

In addition SRGViz has a built-in support for URDF. SRGViz can, indeed, load any URDF file and combine it with its own JSON configuration file.

SRGViz comes with a stand-alone node for ROS that allows to manage any transformation efficiently. The SRGViz node is compatible with the TF2 system for updates and has its own API for queries, which uses TF2 types to be as compatible as possible.

SRGViz supports, of course, CoCo with a dedicated component. The component receives transformation updates via a port, and provides query resolution via an operation. This allows to do non-blocking updates for other components, and to have the result of the queries immediately.

4.6.4 *Diagnostics*

A fundamental aspect of the SRGViz management of transformations is diagnostics. First it is possible to serialize a graph using JSON, and to generate a graphical representation of the graph using Graphviz [42]. Secondly it is possible to access and visualize the graph using a Web page with a REST interface. The web page allows to view the overall graph structure, the real-time value of transformations, thanks to a WebSocket based continuous streaming, and to update any transformation.

This is specifically useful for interfacing SRGViz with other frameworks such as WebGL based frameworks or Unity.

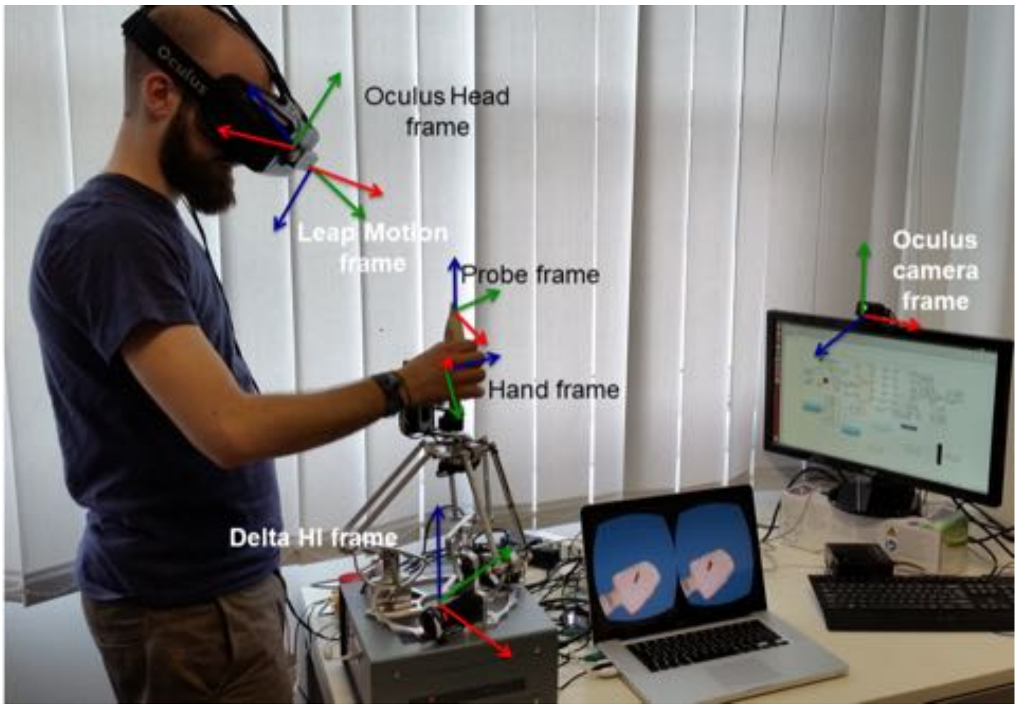


Figure 4.5: USG system setup annotated with the reference systems. On the left side the operator with the HI, on the right side the stereoscopic image displayed on the Oculus HMD.

4.7 REAL CASE SCENARIOS

4.7.1 *Calibration*

In the following we are going to present a real case scenarios where SRGViz has been proven successful, helping in handling complex setups and calibrations.

In this scenario the user interacts, using a haptic device, with a model of a human chest and belly in an immersive VE, wearing an Oculus Rift DK2. The purpose of the work was to create a VE to evaluate the efficacy of a haptic device to perform a remote ultrasonography examination. The complete setup together with all the reference frames involved is show in Figure 4.5.

An important aspect of the proposed setup is the co-location of the visual display in the HMD and the physical interaction with the Haptic Interface (HI). Co-location means that when the user moves his/her hand and touches the probe in the real world, it is perceived at the same location as displayed in the HMD. Hand's movements are rendered as an animated virtual hand, whose pose and fingers are obtained from a Leap Motion sensor.

The co-location is obtained by means of the absolute positioning provided by the HMD, and a calibration procedure that is discussed in the following. The process involves the following 7 reference systems:

- HI base and USG end-effector.
- Virtual geometry: comprising mannequin mesh and implicit surfaces.
- HMD tracking camera origin and head pose.
- Leap-Motion base (attached to the HMD) and hand pose.

The above reference systems are related by a set of transformations, some of which are dynamic and measured, others fixed and known, and others that need to be computed by the calibration process. The origin of the virtual world has been arbitrarily placed in the HI base, and the virtual geometry has been positioned in a way to be reachable by the HI end-effector. For the visual part the hand pose can be easily referred to the HMD tracking origin by means of the fixed transformation from the Leap Motion to the HMD local reference system.

The key action for achieving co-location is the calibration of the HMD reference system with respect to the HI base. Currently, the calibration is obtained as follows:

1. The USG probe is positioned in $(0, 0, z_0)$ and null rotation in the HI base frame.
2. The user places his hand horizontally on top of the probe rotated so that it is aligned with the probe. In this way the transformation between the hand frame and the probe frame is fixed and known.
3. When the above steps are done a key is pressed on the keyboard and the system acquires the poses of the hand and of the probe in their respective frames and computes the calibration matrix that allows to related the HI base frame with the HMD camera origin.

The transformation graph used for the calibration procedure and the execution is shown in Figure 4.6.

An alternative approach is based on the recognition of markers placed on the HMD tracking camera and the HI base.

In this example the required calibration procedure is neither standard nor immediate, nevertheless SRGViz is able to handle it easily and efficiently, proving its effectiveness, strength and flexibility.

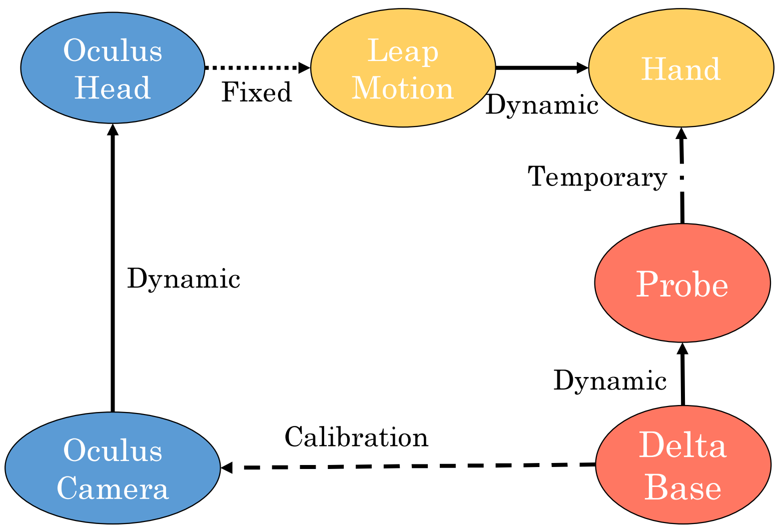


Figure 4.6: Transformation graph of the USG application.

4.7.2 *Fusion*

In this section we are going to present an application where SRGViz has been used to automatically provide fusion capabilities of transformations coming from multiple sources.

The application addresses the case of information sharing by a Baxter robot with an operator that is sharing the same workspace. The operator wears a HUD integrated in an industrial helmet, together with a stereo camera (see Figure 4.7). The robot sends to the user information regarding its intentions that are visualized in the HUD. Intentions can be the position where the robot is moving its arm, or the object that it is going to grasp.

In order to have a correct feedback the user and the robot need to be calibrated between each other. The problem is that the operator can move constantly, thus the calibration needs to be computed for each frame. This is done by placing a marker that is visible both from the robot and the operator's cameras. The markers positions are constantly fused providing a robust estimation of the pose of the robot with respect to the operator.

Figure 4.8 shows the reference frames involved in the applications.

4.8 CONCLUSION

In this section we presented the work done in the creation of a tool that, exploiting the state of the art of mathematical tools for 3D transformation,

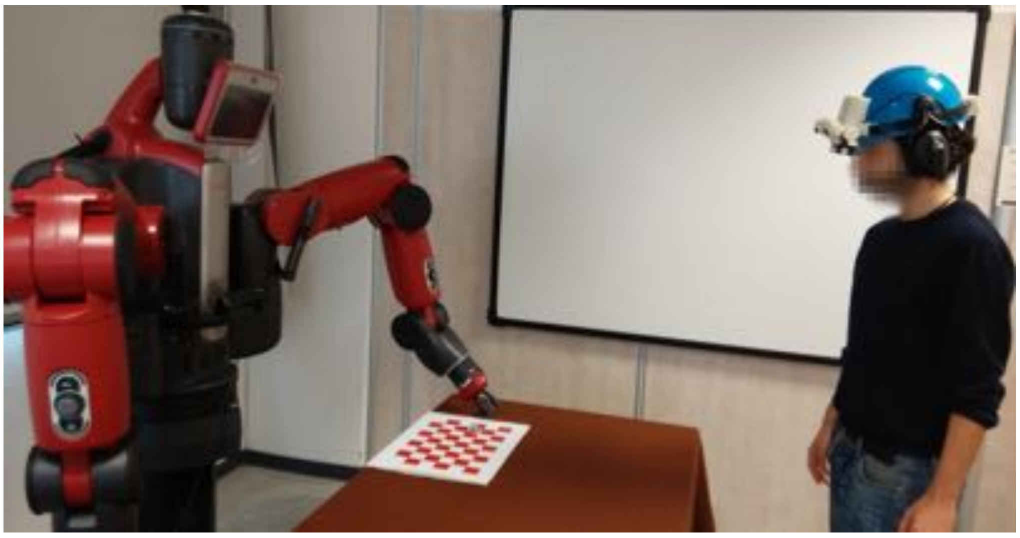


Figure 4.7: Experimental environment with Baxter and operator wearing the HUD.

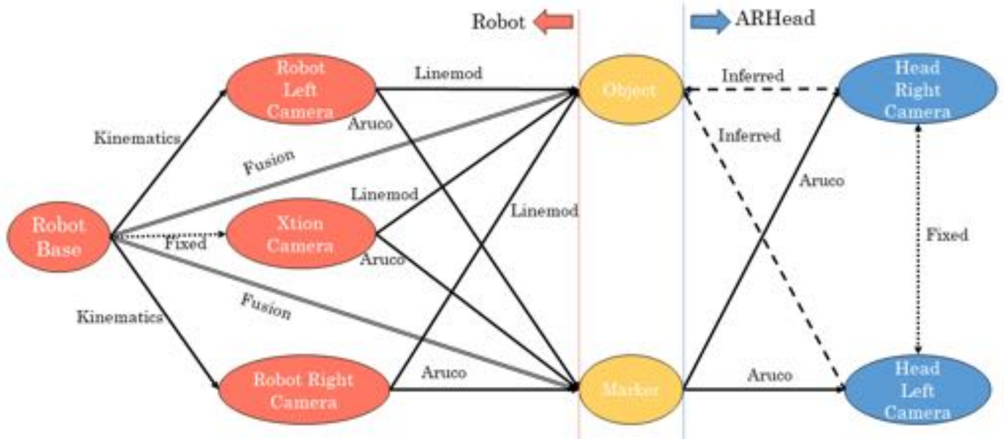


Figure 4.8: Experimental environment with Baxter and operator wearing the HUD.

allows to efficiently and easily manage transformation in AR applications, in particular when robotic devices are involved.

This work provides an improvement with respect to the current transformation graphs, like ROS TF2, as it combines uncertainty management and transformation averaging, with automatic calibration and fusion, all in a performing and flexible C++ framework.

The framework has been tested and used in many applications, some of which within the European Projects ReMeDi (grant number 610902) and Ramcip (grant number 643433), and the Italian Regional project TAUM.

4.9 PUBLICATIONS

Emanuele Ruffaldi and Filippo Brizzi. “Probabilistic Spatial Relationship Graph for Mixed-Reality Tele-Operation Systems”. In: *IEEE Transactions on Robotics* (2017). **Paper to be submitted**

Emanuele Ruffaldi, Filippo Brizzi, Alessandro Filippeschi, and Carlo Alerto Avizzano. “Co-located haptic interaction for virtual USG exploration”. In: *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*. IEEE. 2015, pp. 1548–1551

Emanuele Ruffaldi, Filippo Brizzi, Franco Tecchia, and Sandro Bacinelli. “Third point of view augmented reality for robot intentions visualization”. In: *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*. Springer International Publishing. 2016, pp. 471–478

5 TELEMEDICINE

Increased aging in the developed countries population will require make more medical examinations and treatment [116]. This will worsen the shortage of physicians that is already occurring.

Tele-medicine is actually the best solution for health care systems to keep a high standard of service quality and to serve those areas that lack hospitals and specialists. Tele-medicine services that are already available or under development in many of the WHO (World Health Organization) member countries are typically focused on sharing examination results among specialists such as in the case of tele-radiology, tele-pathology, tele-dermatology and tele-psychiatry [140]. However, tele-medicine services which rely on force feedback are still far from being ready for end-users. The readiness of tele-medicine services shifts from end-users to research level according to the importance of force feedback for the examination. In the last twenty years telemedicine was boosted by developments in the field of robotics, communication and visualization technologies, and it is currently an important research topic that is delivering solutions to our societies. There are several examples of how robotics aids telemedicine, allowing for solutions that have the potential to fill the gap between the required amount of examinations and physicians' capacity. Examples of such systems include endoscopy [132], sonography [4] and palpation [43], [29].

AR and VR are actively contributing to tele-medicine either as a tool for creating simulations for training and testing (VR), and as a support to overcome the limitations of current robotic devices (AR).

Tele-medicine is composed of two modules, the doctor's console, called Diagnostician User Interface (DiagUI), and the remote robotic device that interacts with the patient. The doctor uses the DiagUI to control the remote robot, which is equipped with the appropriate tools, and interact with the patient. In the following two sections we are going to present two DiagUI setups, implemented with the software tools presented in Chapters 3 and 4. The purpose of this work is to show how AR and VR can be used to create, test and validate a DiagUI for tele-medicine.

5.1 PALPATION

Palpation is an examination in which the doctor's fingers and palms interact with the patient's abdomen. The doctor explores the organs and the other tissues beneath the skin checking whether any abnormality is present, such as organs bigger than usual, nodules or other masses. Doctors adopt various techniques for abdominal palpation, varying the pushing location and the force.

5.1.1 *Related Work*

Robotic systems were developed in the recent years to simulate palpation [139], [127].

Inoue et al. [61] propose a simple device for abdomen palpation that is composed of two sheets held by two wooden boards. The motion of the board as well as the tension in the two sheets allow for varying the stiffness perceived by the user. Although the system proved to reliably simulate the abdomen wall, it does not allow for the simulation of abnormalities under the skin surface.

The HIRO system [43] features palpation of deformable tissue (breast palpation) based on a finite element model that allows to provide haptic feedback by means of a robot. Five thimbles are attached to the fingers of the robotic hand to provide force feedback to the user's fingers.

Coles et al. propose [29] a palpation and needle insertion system that integrates haptic feedback and virtual reality. The haptic feedback is provided by two Novint Falcon devices that actuate a tactile palpation end-effector. A camera placed on top of the user's hands tracks their motion. Finally, an AR representation of the scene is co-located with the haptic feedback device, and is displayed on a screen. The co-located haptic and visual feedback increase the fidelity of the system making it suitable for training. Currently, limitations of the system for abdomen palpation are the lack of representation of the real patient and the representation of abnormalities that are beneath the skin.

The approach proposed by Diez et al. [36] adopts also the encountered-type paradigm. They simulate the skin by means of a flat rubber sheet. On one side the user can touch the screen, whereas on the other side a robot provides haptic feedback according to a stiffness map that is computed depending on an assumed tissue and nodule stiffness. The authors propose an algorithm that switches from position to impedance control depending on the interaction of the user with the screen. The user's hand is tracked by means of a marker-based optical tracking system. In another work [142] the combination of visual and haptic cues is further pushed, as multiple point haptic feedback is featured. The image of the patient is used for the visual feedback and for creating a domain in the space in which organs and abnormalities are immersed. The haptic feedback is then based on a blobby objects rendering approach. The visual display is based on a 2D image of the patient that is deformed according to the doctor's hand depth in the direction perpendicular to the image.

5.1.2 *System Setup*

This section presents the architecture and the control strategy of a novel system for virtual remote examination. This approach combines visual and haptics cues and it allows the doctors to move freely their hands in space when they are out of the patient's body and to receive a force feedback only when they interact with the patient. An AR environment facilitates natural interaction with the system.

The AR environment is co-located with the real hand and HI, and the doctor's hand is tracked regardless of the contact with the patient. The doctor is always displayed, in the AR environment, with an avatar of his/her hand that is suitably located with respect to the patient. This system combines therefore the advantages of [29] (haptic rendering co-located with the AR environment) and of [36] (encountered-type interaction). Moreover, as for [142], in this approach the rendered force is the sum of two contributions: the force due to the indentation in the abdomen skin and the force due to internal structures. We progress from [142] approach as the first contribution is based not only on the doctor's current hand position, but also on an online scan of the patient's belly. Moreover, continuous tracking of the patient allows for a continuously updated 3D representation of the scene in the AR environment, in which the video stream of the patient and an avatar of the doctor's hand are displayed. This allows us to have a consistent haptic feedback even when the patient moves, thus handling real patients. In the future, this setting will allow us to extend the system to real remote examination, having a third (robotic) agent performing palpation guided by a doctor (see [89]). Therefore, this paradigm will be useful for

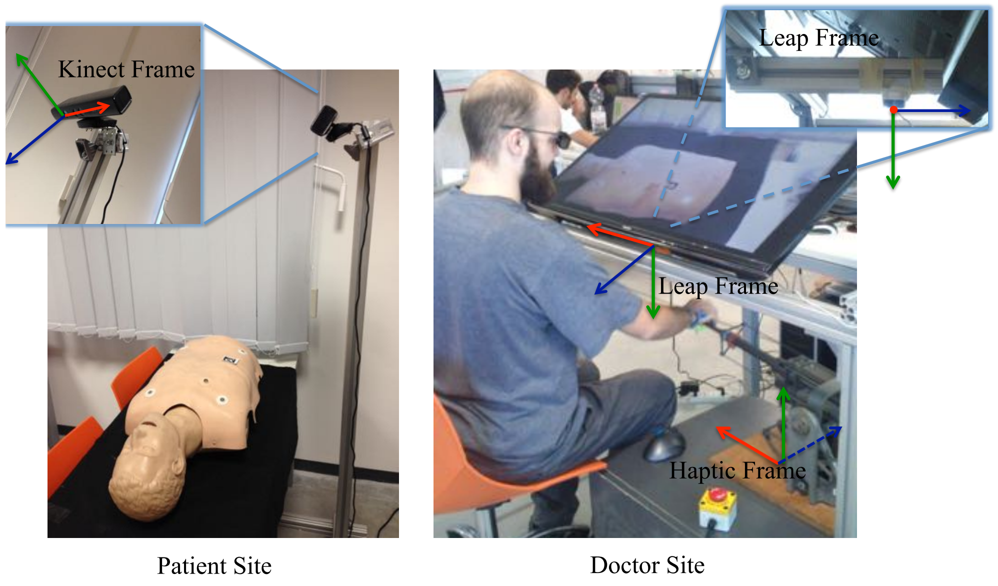


Figure 5.1: The virtual palpation system. The components are shown along with the reference frames in which the gathered data are represented.

training, for online (when combined with the robot) and offline remote examination.

5.1.2.1 *The palpation system*

The virtual palpation system that we developed is shown in Figure 5.1. The system includes two locations: the patient's site (PS) and the doctor's site (DS). In the PS the patient lies on a table while he/she is tracked by an RGB-D sensor (Microsoft Kinect® version 1). A computer manages the video and depth stream to the DS. In the DS, the doctor wears 3D glasses and sits in front of a frame that holds a 3D screen positioned horizontally. Under the screen the doctor's hand interacts with a high performance HI [15] while his/her hand are tracked by means of a Leap Motion mounted between the bottom of the screen and the HI. This solution has a sufficient tracking workspace and does not interfere with the contact of the hand with the HI. In the DS a computer (PC-1) manages the video and depth stream coming from the PS. A second computer (PC-2) embeds the Matlab® XPC Target application that runs the control of the HI. Currently, the end-effector of the HI is a ball (6 cm diameter) that is used to handle the contact with either the fingers or the hand's palm. All the computers involved are Intel PC (Core™ i7 4770R 3.2 GHz, 8 GB RAM, embedded GPU) running Ubuntu Linux. The HI is a 3 DoFs robotic interface (see Figure 5.1). The workspace is a sphere sector that spans between 0.4 and

0.8 m from the center of rotation and include a barrel's rotation of $[-20, 20]$ deg and $[-40, 40]$ deg about the first two DoFs axes. The worst end-effector position resolution is 0.13 μm whereas the maximum continuous and peak force are 4N and 10N. Other HI could be used, given that they match the features of this one.

Thanks to the aforementioned components, the doctor interacts with a virtual model of the patient receiving a visual and a haptic feedback. The patient virtual model is based on a point cloud representation of the chest and abdomen surface obtained from the RGB-D sensor. When the doctor's hand indents the skin the virtual force that is needed for the indentation is calculated. Moreover, this setup allows us to add components within the patient's virtual body such as organs and abnormalities, and to calculate the virtual force generated by interacting with these elements. The overall virtual force is then used to provide the doctor with a haptic feedback. When the doctor's hand is out of the body the HI end-effector is not in contact but still follows the doctor's hand. At the same time the doctor is provided with a 3D AR representation of the PS in which an avatar of the doctor's hand is superimposed to the PS scene.

5.1.2.2 *Software Setup*

The system is made of several CoCo components acting either in the DS or the PS. In the PS, the video and depth streams from the RGB-D sensor are compressed and sent via TCP to the DS. In DS PC-1 receives the information via TCP from the PS and runs the following modules:

- **Visualization**, that provides the AR feedback based on the PS video stream and the doctor's hand position.
- **Haptic rendering**, that exploits the point cloud from the PS and the doctor's hand position.
- **Communication** with the patient site computer, the Leap Motion, and the XPC target computer for the HI control.

PC-2 receives data from PC-1 and runs the encountered haptic control that regulates both the haptic feedback and the doctor's hand position. Figure 5.2 shows the CoCo components running in PC-1 in DS, together with the connections and the different rates at which each module runs depending on the different computational burden of each task.

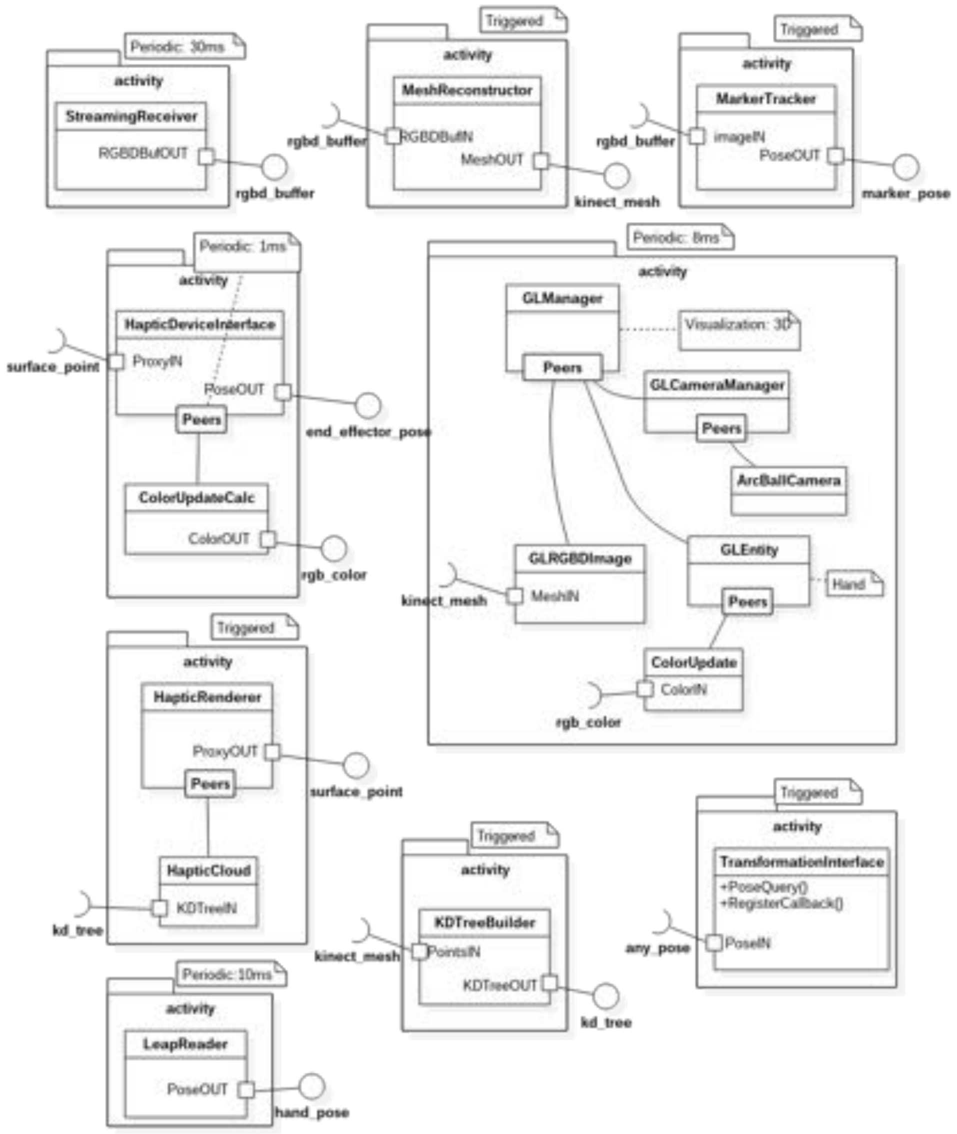
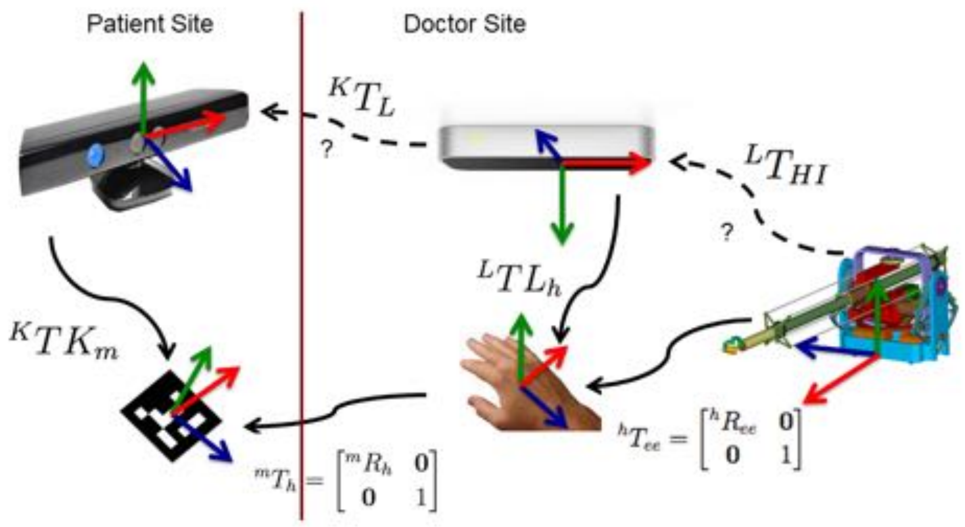


Figure 5.2: The system architecture. Each block corresponds to a CoCo component or peer. Ports with the same name are connected to each other (e.g. RGBDBufOUT is connected to RGBDBufIN). For each component is also specified the execution policy. The TransformationInterface component embed the SRGViz library presented in Chapter 4.



Patient Site
Doctor Site
^KT_L
?
$^LT_{HI}$
?
^LTL_h
^KTK_m
$^mT_h = \begin{bmatrix} ^mR_h & 0 \\ 0 & 1 \end{bmatrix}$
$^hT_{ee} = \begin{bmatrix} ^hR_{ee} & 0 \\ 0 & 1 \end{bmatrix}$

Figure 5.3: All the reference frames involved in the applications. Dotted lines represent calibration transformations.

HAPTIC RENDERING The haptic rendering is performed using the CoCo component described in Section 3.4.2. The module takes in input the mesh of the mannequin from the PS and the hand position from the Leap Motion. The mesh is obtained from the reconstruction of the depth image obtained from the Kinect in the PS, streamed to the DS and updated at every frame. The haptic rendering module outputs the position of the proxy on the surface of the mannequin, which, together with the hand pose, is sent, via UDP, to the low level haptic control.

REFERENCE SYSTEMS AND CALIBRATION In the following $^A\mathbf{pB}_c$ means that the position $\mathbf{p}$ has been obtained by the device $\mathbf{B}$, is associated to the object c and is written in the frame A. If a variable has been inferred it has no capitol letters in its name. Each frame axis direction is associated to the unit vectors $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ for the x, y and z axes respectively. The variables available for the module are:

- $^L\mathbf{pL}_h$ and ^LTL_h i.e. the hand position and pose homogeneous matrix obtained from the Leap motion.
- $^L\mathbf{p}_p$ i.e. the proxy position from the collision detection.
- $^H\mathbf{pH}_E$ i.e. the end-effector position provided by the HI.

Figure 5.3 reports the frames of each hardware/software component in PS and DS. The relationships among reference frames that are needed for

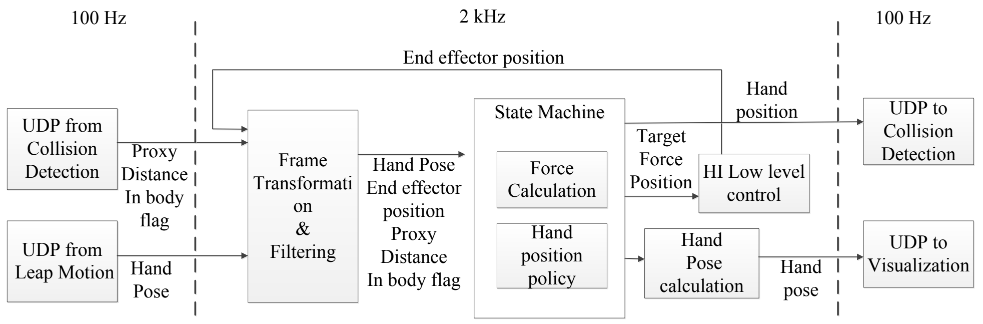


Figure 5.4: The encountered haptic module along with its main components.

a consistent co-location of haptic and the visual feedback (dotted edges) are obtained through a calibration procedure. The calibration procedure consists of putting the hand on the end-effector where we desire to have the contact with the skin. ${}^{L}\mathbf{pL}_h$ and ${}^{H}\mathbf{pH}_E$ are recorded in this configuration to obtain ${}^{L}\mathbf{pL}_h^0$ and ${}^{H}\mathbf{pH}_E^0$ that are used as offsets for the following position transformations

- Leap to Haptic frame (L2H)

$$ {}^{H}\mathbf{p} = R_L^H({}^{L}\mathbf{p} -{}^{L}\mathbf{pL}_h^0) +{}^{H}\mathbf{pH}_E^0 \tag{5.1} $$

- Haptic to Leap frame (H2L)

$$ {}^{L}\mathbf{p} = R_H^L({}^{H}\mathbf{p} -{}^{H}\mathbf{pH}_E) +{}^{L}\mathbf{pL}_h^0 \tag{5.2} $$

where

$$ R_L^H = R_H^L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{5.3} $$

The reference frames and the calibration are handled using the SRGViz library, presented in Chapter 4, and embedded into a CoCo component.

ENCOUNTERED HAPTIC CONTROL The low level control manages the encountered protocol and computes the forces based on the indentation of the hand with respect to the proxy position (see Figure 5.4). It is implemented as a Matlab Simulink model that runs in external modality in the PC-2 XPC target at 2kHz frequency.

Hand pose, proxy position and a boolean variable (i_b) that is true when the doctor's hand is within the body are available at 100 Hz from CoCo, via UDP. These variables are firstly converted to 2kHz frequency and then

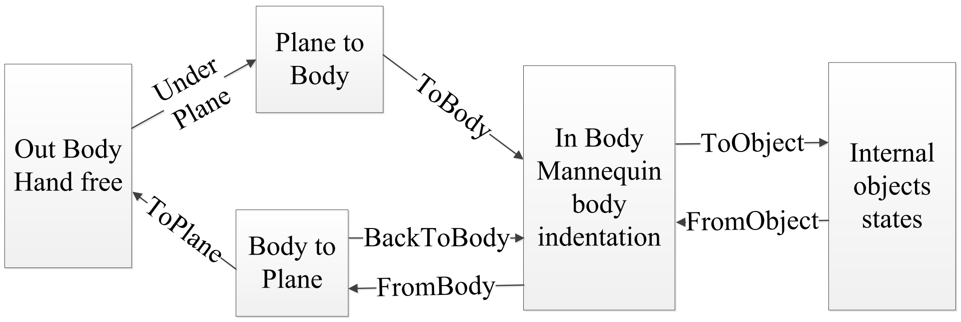


Figure 5.5: The state machine that regulates the haptic feedback and the doctor's hand position.

transformed in the HI frame. ${}^{H}\mathbf{pH}_E$ is available from the HI sensors at 2kHz. Being based on ${}^{L}\mathbf{pH}_h$, i_b showed to be stable enough to be used without filtering. A Kalman filter was applied to ${}^{L}\mathbf{pL}_h$ and ${}^{L}\mathbf{p}_p$ to have a smoother signal. Given the low update rate of these variables, we exploited the previous estimates of these variables to predict the variable evolution in the 20 time steps between two updates.

The module returns the doctor hand position ${}^{L}\mathbf{pH}_h$ that is used for the collision detection and the doctor's hand pose ${}^{L}TH_h$ that is used by the visualization module.

$$ {}^{L}TH_h = \begin{bmatrix} {}^{L}RL_h & {}^{L}\mathbf{pH}_h \\ \mathbf{0} & 1 \end{bmatrix} \tag{5.4} $$

Moreover it embeds the low level control of the HI that features position and force control given a desired position of the end-effector and a desired force $\mathbf{F}$ to be displayed at the end-effector. The force control is an open loop control that provides the correct currents to the motor. The position control is a proportional-derivative controller based on ${}^{H}\mathbf{pH}_E$.

We managed the transition between being out of the body and within the body by means of a finite state machine that is shown in Figure 5.5, and each state of the machine is labeled as s_f.

In the *Out Body* condition the HI is in position control. The target hand position is

$$ {}^{H}\mathbf{p}_t =^{H} \mathbf{p}_p + [0\,\delta_o\,0]^T + \Delta^{H}\mathbf{pH}_E \tag{5.5} $$

where $\delta_o = 1\,\text{mm}$ is a small vertical offset that guarantees undesired transitions to the *In body*. $\Delta^{H}\mathbf{pH}_E =^{H} \mathbf{pH}_E -^{H} \mathbf{p}_p$ is calculated during the last sample before the transition from the *In body* state and allows us to avoid discontinuities in the position control of the HI just after the transition.

The target force $\mathbf{F}$ is set to zero during this phase. Finally, in this state ${}^{L}TH_h ={}^{L} TL_h$. In the *In body* state the HI is in force control. This is obtained by setting ${}^{H}\mathbf{p}_t ={}^{H} \mathbf{pH}_E$. We define ${}^{H}\mathbf{p}_p^0$ as the proxy position just after the system enters the *In body* state. Hence ${}^{H}\mathbf{p}_p^0$ is the hand's first contact point on the undeformed skin. The target force $\mathbf{F}$ is then calculated depending on the indentation vector $\mathbf{d}$ with respect to the first contact point. This condition represents well the fact that during palpation the perceived force depends on how the skin has been stretched and indented with respect to the initial contact point (the hand does not slide on the skin)

$$\mathbf{F} = [k_x(\mathbf{d}_x)k_y(\mathbf{d}_y)k_z(\mathbf{d}_z)]^T\mathbf{d} \tag{5.6}$$

where

$$k_i(d_i) = \alpha(d_i^2 + d_i) \quad \text{i = x,y,z} \tag{5.7}$$

and

$$\mathbf{d} ={}^{H} \mathbf{pH}_E -{}^{H} \mathbf{p}_p^0 \tag{5.8}$$

This approach can be easily extended by continuously updating the proxy as in the algorithm presented in [113] where $\mathbf{d} ={}^{H} \mathbf{pH}_E -{}^{H} \mathbf{p}_p$ and k_i calculated according to equation 5.7. In case a collision occurs with other objects within the body, the force due to the collision can be calculated according to one of the previous two modalities and it is managed by means of further states. The overall target force is then calculated as the sum of the force due to the indentation $\mathbf{F}$ and the collision force with each of the objects. Currently a sphere and a cylinder are available as objects to be inserted in the body.

Since the transition between the *In body* state and the *Out Body* state may cause discontinuities in the hand representation we sum the difference between the ${}^{L}\mathbf{pH}_h$ estimations occurring at the state transition. ${}^{L}RH_h$ is extracted from ${}^{L}TL_h$. Whereas the transitions among states within the body are simply dependent on the hand position with respect to the inner objects, the transitions between the *In body* state and the *Out Body* state may cause discontinuities in the visualization, instability in the control of the HI and undesirable loops between the two aforementioned states. To prevent these effects, we set a timer to stay at least 0.3 s in each state and we added two intermediate states called *Body to Plane* and *Plane to Body* (see Figure 5.5). The first avoids to have loops between *In body* and *Out Body* and allows for a smooth leaving of *In body*. The *FromBody* transition occurs when the collision detection module sets the in-body flag i_b to zero and the distance from the surface d_s is $d_S < d_{thr}$. In the *Body to Plane* state there's no position control but a force is applied to the end-effector in order to go farther from the body. When the height of the plane out of the body is reached, the transition *ToPlane* occurs. In the *Out Body* state there is no force applied to the end-effector, and a position control forces the end-effector to move on a plane above the body according to ${}^{H}\mathbf{pL}_h$ x and z components. When the user's hand pushes the end-effector towards the body, moving it below the plane, the *Under Plane* transition takes place. Then, in the *Plane to Body* state there is neither position nor force control, and the end-effector waits to be pushed towards the body. Finally, we allow the users to go from the *Body to Plane* back to the *In Body* state in case they start pushing again towards the body before leaving the intermediate state.

VISUALIZATION The visualization module (see Section 3.4.3) displays on the 3D screen the remote scene as a 3D mesh, created from the point cloud provided by the Kinect sensor, augmenting it with a virtual hand model controlled by Leap Motion.

The mannequin and therefore its 3D representation don't deform as the hand penetrates their surface causing the virtual hand to disappear beneath the mannequin mesh as soon as the indentation exceeds few centimeters. To avoid this inconvenience the position of the hand is moved, during the palpation, according to the position of the proxy, while the orientation is still the one provided by the Leap Motion. We set a zero-hold policy when the hand is not tracked.

To improve the alignment between the visual and the haptic feedback the texture of the virtual hand gradually shifts its color towards a red shade linearly with the depth of the end effector inside of the surface.

5.1.3 *Experiment Setup*

Two healthy volunteers tested the system for a preliminary assessment of its usability. They carried out an experiment aimed at checking that forces are correctly displayed, thus allowing the participant to naturally interact with the body and to identify structures within the body. These goals are necessary steps to allow the final user to perform a correct diagnosis. The protocol is composed of four trials, in the first they had to indent the patient's skin in different points, thus verifying whether the system provides

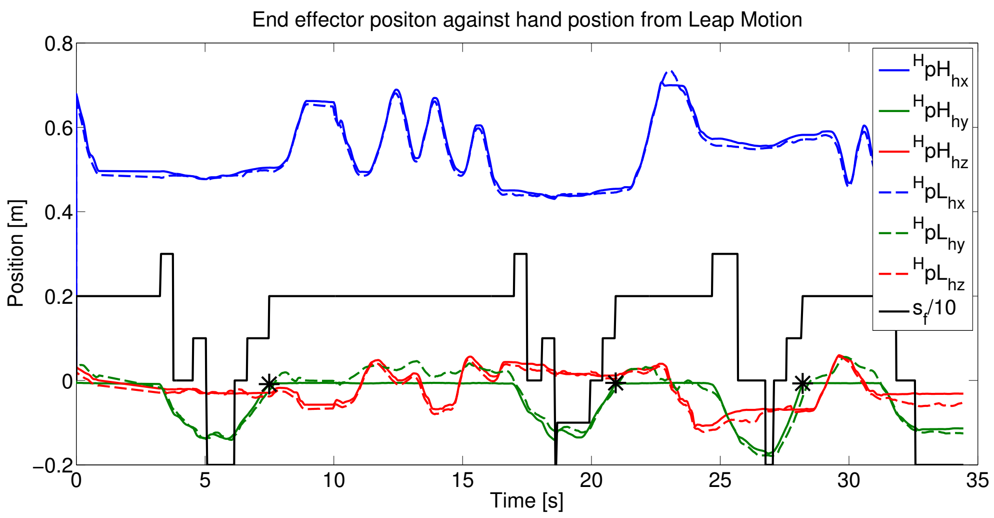


Figure 5.6: First trial. Comparison of the end-effector position against the leap motion estimation of the hand position depending on the state s_f.

a natural interaction when switching between contact and non-contact with the patient. In the second trial, the volunteers had to enter the patient's body at a specific point and try to move within the patient's body. In the third trial, the volunteers were asked to interact with a cylinder (radius $r_c = 0.03\,\mathrm{m}$, height $h_c = 0.05\,\mathrm{m}$) lying 0.03 m under the skin. The cylinder's stiffness was set to 800 N/m in order to facilitate the volunteers identifying it. In the fourth trial they were left free to interact with the virtual patient looking for abnormalities within the body. In all the trials the collision detection module used $R = 0.1$ m and $N = 10$ for the radius search of the k-d tree, whose leaves include at most 10 points, and an influence radius of 0.1 m in the implicit surface algorithm. We gathered a 300k point cloud from the Kinect and all the k-d tree leaves were checked.

5.1.4 *Results*

We report here the results that we obtained. In the following figures, we show only explanatory examples, but the conclusions are supported by all the data that we gathered. In the following figures the possible states s_f are: $s_f = 0$ i.e. *In body*; $s_f = 1$ i.e. *Body to Plane*; $s_f = 2$ i.e. *Out body*; $s_f = 3$ i.e. *Plane to Body*; $s_f = -3$ i.e. *Cylinder Upper Area*; $s_f = -2$ i.e. *Around the Cylinder*; $s_f = -1$ i.e. *Cylinder Lateral Area*. The first trial confirmed that the system manages the transition between *Out Body* and *In Body* conditions preserving a natural interaction. Figure 5.6 shows that there are no jumps in the position of the end-effector, meaning that

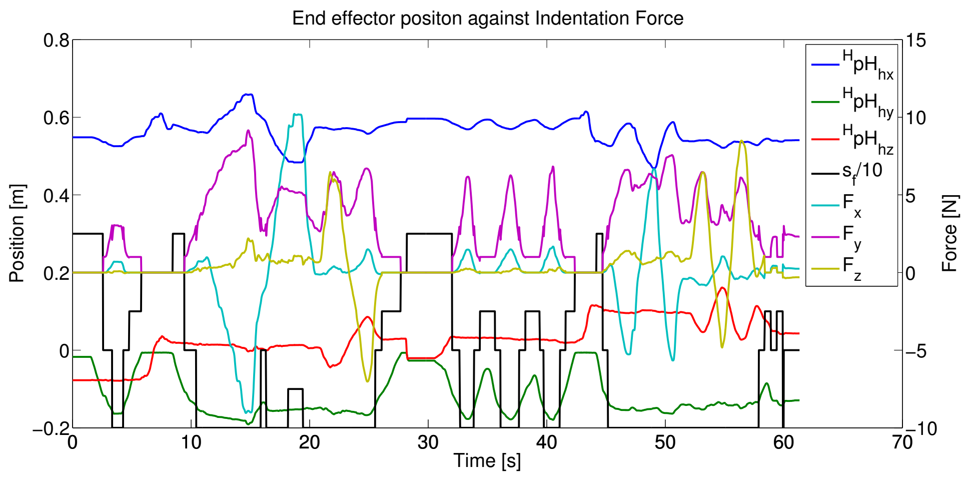


Figure 5.7: Second trial. End-effector position against indentation force.

the transition is fluid. When the user's hand is outside of the body the end-effector correctly tracks the hand's position being on a plane just over the body (see the stars in Figure 5.6). The position of the hand to be used in the AR environment, instead, moves in 3D according to ${}^{L}TL_{h}$.

The second trial shows that the force is correctly displayed to the user in all the directions during the indentation. Figure 5.7 shows the force evolution according to the indentation and the force transition when switching the condition of the interaction (i.e. between inside and outside of the body).

The third trial result shows that the volunteers were able to perceive the cylinder within the user body. Figure 5.8 shows the trajectories along with the forces that were perceived during the contact with the cylinder's upper and lateral area.

Figure 5.9 shows an example of interaction of the user with the environment when the goal was searching a cylinder. The system correctly performs in every s_f state, allowing the user to change the location of the exploration during the non-contact phases, to perceive the resistance of the abdomen tissue in the indentation phases and to perceive the force due the interaction with the cylinder.

5.1.5 *Discussion*

The stiffness values were selected in these preliminary tests to ease the exploration of the belly. Increasing these values does not introduce any technical difficulty and more extensive tests will be carried out to make the system more specific for palpation. However, this first bench of tests

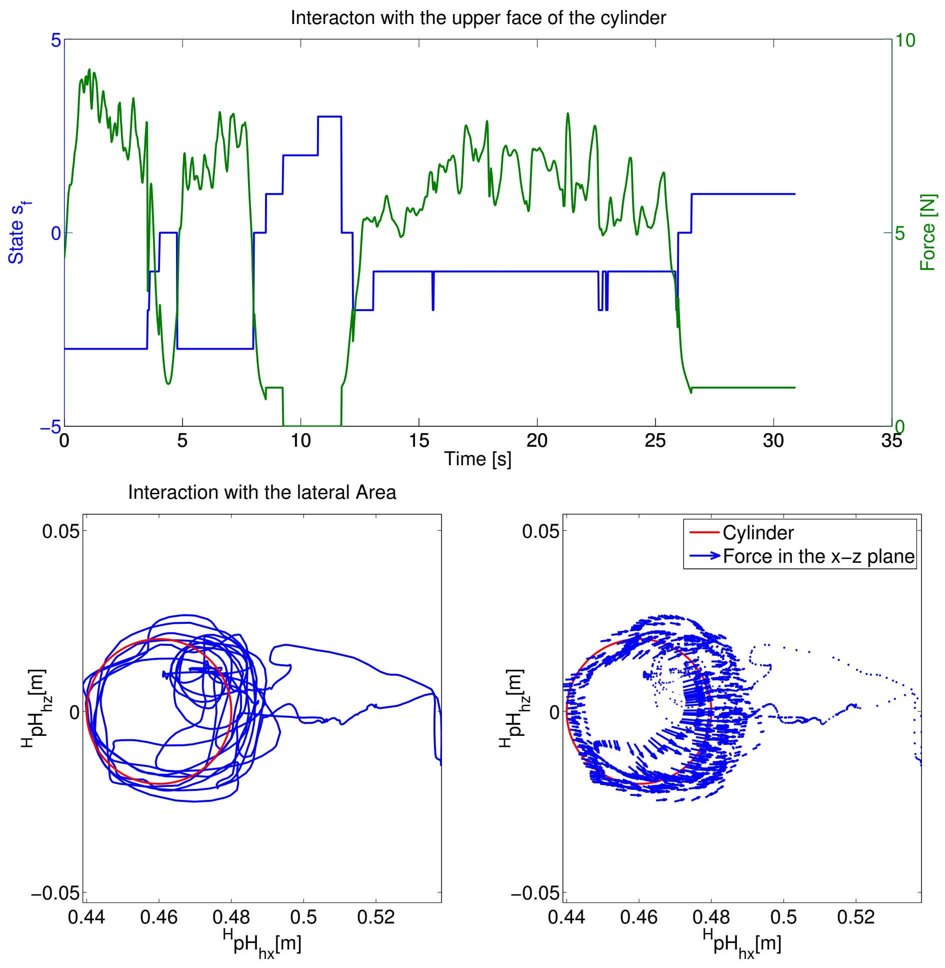


Figure 5.8: Third trial. The upper figure shows the interaction with the upper area (see s_f). The lower figures shows the interaction with the lateral area.

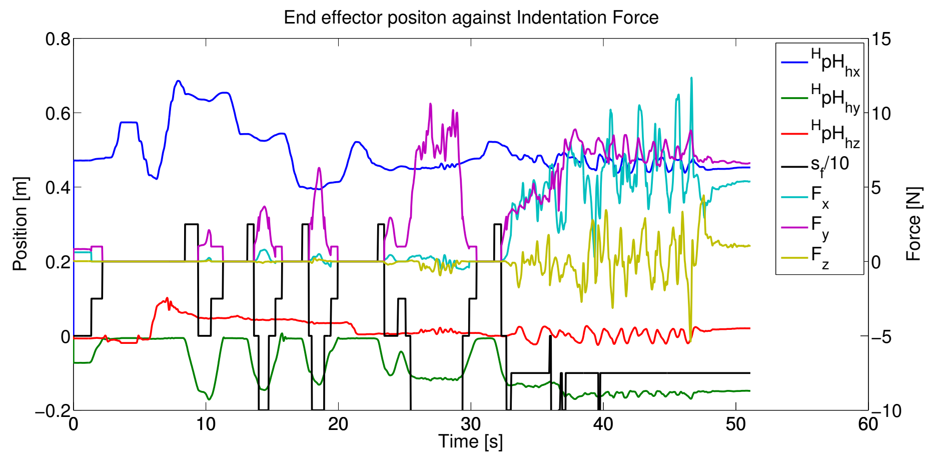


Figure 5.9: Fourth trial. The search presents some of the elements of each of the previous trials: we note transitions form inside to outside with free hand and we not force exertion in different locations to find the cylinder.

demonstrate the usability and the capabilities of the system. We also recognize that a more reliable hand tracker could avoid the need for two intermediate states between the *In Body* and the *Out Body* states. We highlight, however, how the interaction with the system turns out to be very natural and fluid despite these states. The volunteers required indeed a short training to be able to use the system. We also mention here the nice trade off in the volunteers' strategy that we noted when searching for the cylinder in the fourth trial. The users firstly try to find the cylinder from inside the body, thus experiencing increasing values of indentation force. When the indentation force was too high to keep searching, they exited from the body to start from another indentation point. This resembles what happens in palpation, in which the exploration is rather local around the first contact point with the skin. This aspect will be further investigated. Although there were not issues related to the resolution of the point cloud, we plan to test our system with the Kinect One sensor, in order to check possible improvements. In the present system Kinect was used because the newer sensor's drivers were not available for the Linux OS.

5.2 ULTRASONOGRAPHY

Ultrasonography (USG) is one of the most important examinations in order to decide if a patient needs to be directed to a specialist. The continuous progress of HIs and VR and AR let recent tele-ultrasonography (tele-USG) systems include a DiagUI that features both visual and force feedback. The fidelity and usability of these systems depend strongly on the features of such DiagUI. However, tele-USG systems are often evaluated as a whole, and it is impossible to assess how specific DiagUI features influence the sonographer's performance. A specific evaluation of force and visual feedback is fundamental to assess how a DiagUI contributes to the effectiveness of a tele-USG system. First, it is important to evaluate how a DiagUI is perceived per se before it is integrated in a complete system. Second, a validated DiagUI allows to set up training environments in which novice sonographer can speed-up the development of their skills. Finally, this kind of evaluation enables specific improvement of the DiagUI, thus speeding up the process of making a tele-USG system accepted by specialists. This problem has been tackled within the European project ReMeDi [89] that aims at achieving a complete tele-examination system for the cardio and abdominal USG, palpation, and auscultation. This system features a DiagUI, which includes state of the art technologies for both the HI and the visualization system.

This section presents a method to evaluate the DiagUI usability in terms of both sonographers' performance in accomplishing USG-like tasks and their subjective perception of the interface. In particular, experts' perception of the DiagUI refers to the usability in a real USG task and its pleasantness in the everyday work. This evaluation can be replicated for other DiagUIs to allow for a comparison before the integration of a whole tele-examination system.

5.2.1 *Related Work*

Tele-USG systems currently focus more on the correct transmission of images and control of the robot at the patient site than on the features of the DiagUI. However, DiagUIs of many tele-USG systems feature both visual and force feedback. For example, in [111] and [2] the system includes a 6 DoFs robot composed of an orientable pantograph and an end-effector that allows for 3D positioning of the probe and a reasonable decoupling of translational and rotational DoFs. A DiagUI equipped with either a 6 DoFs power mouse or a 6 DoFs joystick-like HI [110] allows the doctor to select a feature on the graphical user interface (GUI) and the robot controller acts so that the desired feature location is achieved. Pierrot et al. [94] use an industrial 6 DoFs arm provided with a force feedback to carry out USG examinations. In this system, the sonographer can either grab the probe and place it manually or s/he can run an examination routine by means of a GUI. More recently a complete tele-USG system was developed within the European project OTELO [35]. The system includes a 6 DoFs robot at the patient's site and a 6 DoFs HI at the expert site. The DiagUI of such system also features a telepresence module which includes a VE where the patient is displayed, a USG image transmission and an audio/video conference tool [24]. In this line Arbeille et al. [4] developed a tele-USG system which works over a satellite link to make available echography examination for astronauts (TERESA project [133]). In their system, the patient and the expert sites are linked by a videoconference system. At the patient's site a non-specialist operator places a robot (ESTELE) on the patient. The robot holds the probe whose pose is remotely controlled by a sonographer who uses a probe dummy and exploits the visual feedback from the videoconference system. In the TER tele-USG system [135], a new slave robot at the patient site decouples the gross positioning of the probe from its pose fine refinement on the patient's body. The sonographer at the expert site uses video, audio, and haptic feedback to conduct the examination. In particular, a videoconference system is complemented by a 6 DoFs (3 actuated) HI (PHANToM from SensAble Technologies Inc.). The HI displays the contact force between the probe and the patient thanks to a force sensor mounted on the slave robot end-effector. The PROSIT 4 DoFs robot [87] is used by Krupa et al. in [71] to develop a portable and kinematically suitable robotic interface at the patient's site. This system includes a teleoperation loop between this robot and a HI at the doctor's site. The authors propose to have at the doctor's site alternatively a purely passive probe dummy, a passively actuated probe dummy or a fully actuated probe dummy as master of the teleoperation loop. More systems (see [1] for a review) were developed for robotic USG examination, some of them (e.g. [83] for tele-USG) were developed as sonographer's UI.

With respect to these systems, our DiagUI features several advantages. First, the translational and rotational DoFs of the probe are completely decoupled and the interface can display forces up to 40 N in any direction. Although only 3 DoFs are actuated, thanks to the selected design it is simple to actuate the three rotational DoFs. Second, the aforementioned systems focus on achieving good remote control and videoconference system, but they do not eviscerate the problem of mapping the sonographer's interface workspace to the workspace needed to explore the patient's body. Finally, they use 2D visualization and do not investigate what visualization systems are more suitable or simply preferred by experts. Since our DiagUI features a 3D AR environment which can be displayed both on a screen or in an HMD, considerable improvements lie also in the visualization.

To the author's knowledge, none of the DiagUIs of the aforementioned tele-USG systems was tested alone, whereas the respective tele-USG systems were tested in clinically relevant settings. In [111] and [2] teleoperated control and visual servoing are tested to demonstrate the validity of the system. The navigation problem was solved by means of a position-velocity controller. Since the sonographer operated at the patient site, the visualization system played a minor role and it simply consisted of a GUI shown on a 2D screen. The evaluation of the whole tele-USG system is also proposed in [35] and [24] with a focus on the USG image processing. The main drawback of the OTELO system [35] is the delay between the motion of the expert and the actual motion of the robotic arm. A solution is proposed in [24]:

visual or force feedback are used to prevent the sonographer moving too fast, thus allowing the robot to correctly follow the sonographer's motion. This work proposes a first implementation of a 3D representation of the patient's site, witnessing a perceived importance of giving the sonographer the possibility to navigate the patient's site in a 3D VE. Arbeille et al. in [5] compare three different tele-USG methods: moving the probe at the patient's site by means of a robotic arm, and using an echograph with a motorized probe and remote guidance, in which an operator at the patient's site handles the probe according to the remote expert's instructions. This work reports the results of 340 examinations showing that for deep organs examinations, a motorized system allows the sonographer to position the probe more accurately. However, no further investigations were carried out on visualization or navigation modalities.

In a first phase, a USG examination requires the sonographer to place the probe on the rib cage. After that a suitable point on the rib cage is reached, the examination is driven by the USG images and this makes the orientation of the probe crucial. Although the successful positioning of the probe is easily achieved in traditional USG, it becomes more difficult in teleUSG. Therefore, we propose an evaluation of the sonographer DiagUI that excludes factors introduced by a complete USG examination. Evaluating this DiagUI during the whole USG task would indeed hide specific issues of positioning the probe and being able to perceive what is underneath the skin. The proposed evaluation investigates different navigation and visualization modalities in order to select their best combination. First, we propose a comparison of 2D and 3D visualizations of the patient's site, which was not done in the previously described systems. Moreover, we exploit two off-the-shelf devices for visualization, namely 3D LCD screen and HMD. Second, we target the navigation modality.

A rate control allows some of the aforementioned systems to use small HIs to operate in a large workspace. However, a pure mapping of the UI end-effector translation into an operated arm velocity is detrimental for the probe position accuracy [38]. Moreover, it requires the user to easily reach the neutral configuration of the HI to stop the slave arm. On the other side, a position/position control requires the DiagUI workspace be scaled (i.e. "scaling" [50]). The main advantage of the scaling technique is that a single smooth movement allows the user to reach any point of the patient. However, the main drawback is a loss of resolution that may be detrimental for the accuracy. Moreover, the proper scaling factor may depend on the patient's body size. Alternative solutions, known as "clutching" techniques (e.g. [65] also found as "indexing" in [30]) allow the user to shift the area of the patient which is explored. In [30] the authors exploit the dominance of visual on haptic feedback to introduce small deviations of the visual cue in order to translate the part of the VE (which they call virtual workspace) that the user is exploring. Instead, Dominjon et al. present and validate in [37, 38], the "bubble" technique. This technique features a hybrid rate/position control thus deploying the speed advantage of a pure rate control and the accuracy advantage of a position control. Similarly, in [9] Barrio et al. modify the bubble technique with a dead zone around the bubble. Finally, Song et al. [123] extend the clutching technique in the case of 3D HIs by setting a plane orthogonal to the line of sight which discriminates clutched and declutched conditions.

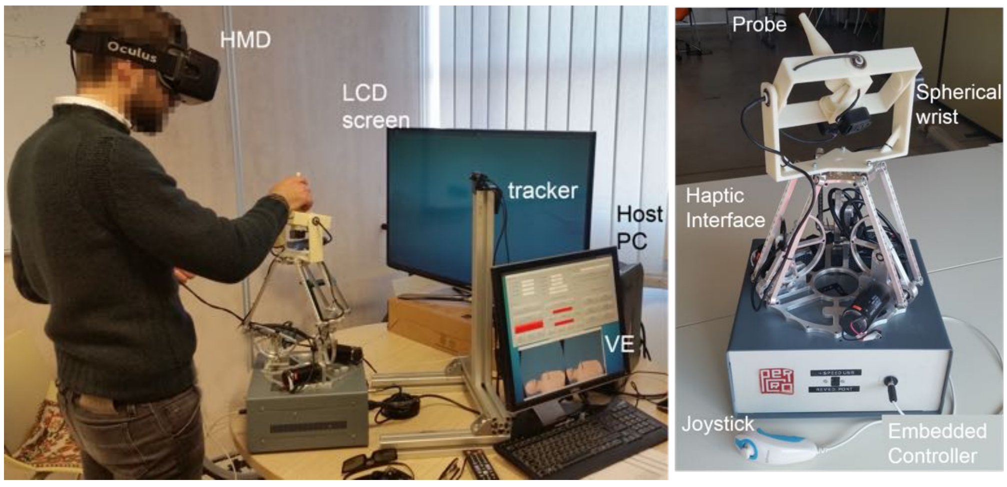
We selected scaling and bubble techniques because in our opinion they are the most appropriate for tele-USG. The indexing technique described in [30] seems indeed more suitable when small shifts of the virtual workspace are needed, whereas the Z-clutching technique seems to perform similarly to the bubble technique [123].

5.2.2 *System Setup*

5.2.2.1 *Hardware Setup*

The DiagUI is composed of two parts, one including the visualization tools and the other the HI (Figure 5.10). Visualization tools include a 3D 40" LCD monitor (resolution 1920x1080) with 3D active glasses and the Oculus Rift DK2 HMD (Oculus, CA) that features: 960x1080 per eye resolution, refresh rate 60Hz, 100 deg FOV, head orientation and position tracked with respect to a fixed frame by fusing a near infrared optical camera sensor (60Hz frequency)and an IMU (inertial measurement unit, 1kHz frequency). A Leap Motion sensor (Leap Motion Inc, CA) mounted on the frontal part of the HMD allows for hand tracking (100Hz).

The haptic feedback is provided by a custom Delta-like [95] HI (Figure 5.10 b) with a 3 DoFs translational motion of the end-effector whose workspace includes a cylinder (diameter 0.26m and height 0.12m). The device can display up to 40 N force in any direction. A 3DoFs, non-actuated, spherical wrist (Figure 5.10 a) is mounted on top of this device and hosts an ABS printed dummy of a real USG probe which was obtained by 3D



(a) (b)

Figure 5.10: (a) A sonographer during the experiments. He wears the HMD and navigates the VE using the HI in front of him. The interaction with the device is performed handling a copy of a real USG probe. In front of him the 3D screen used as alternative to the HMD. (b) The HI of the DiagUI.

scanning of a real probe. The wrist is equipped with three encoders to measure out the probe's orientation. The hardware includes a joystick as shown in Figure 5.10 b.

The firmware of the HI is embedded in a custom board based on an ARM Cortex-M4 32 bits STM32F407VGT6 micro controller that manages encoders' signals, joystick's input, motor drivers and the communication with external devices. The high-level control, the haptic rendering and the VE rendering are executed on a Dell Alienware x51 computer featuring a quad-core Intel Core i7-4790K, 16GB of RAM and a GPU Nvidia GeForce GTX 670 running Ubuntu Linux 14.04.

5.2.2.2 *Software Setup*

The software used to run and control the overall system is composed of three elements: low-level, medium-level and high-level software. The low-level software is implemented as a Matlab (Natick, MA) Simulink model, which is compiled and downloaded into the embedded device board. The low-level code executes at 1 kHz the algorithms that control the motor drivers, reads sensors and computes kinematics and gravity compensation of the device [6]. Due to the small measured friction and the low velocity of the device during USG examinations, neither friction nor dynamics compensation are

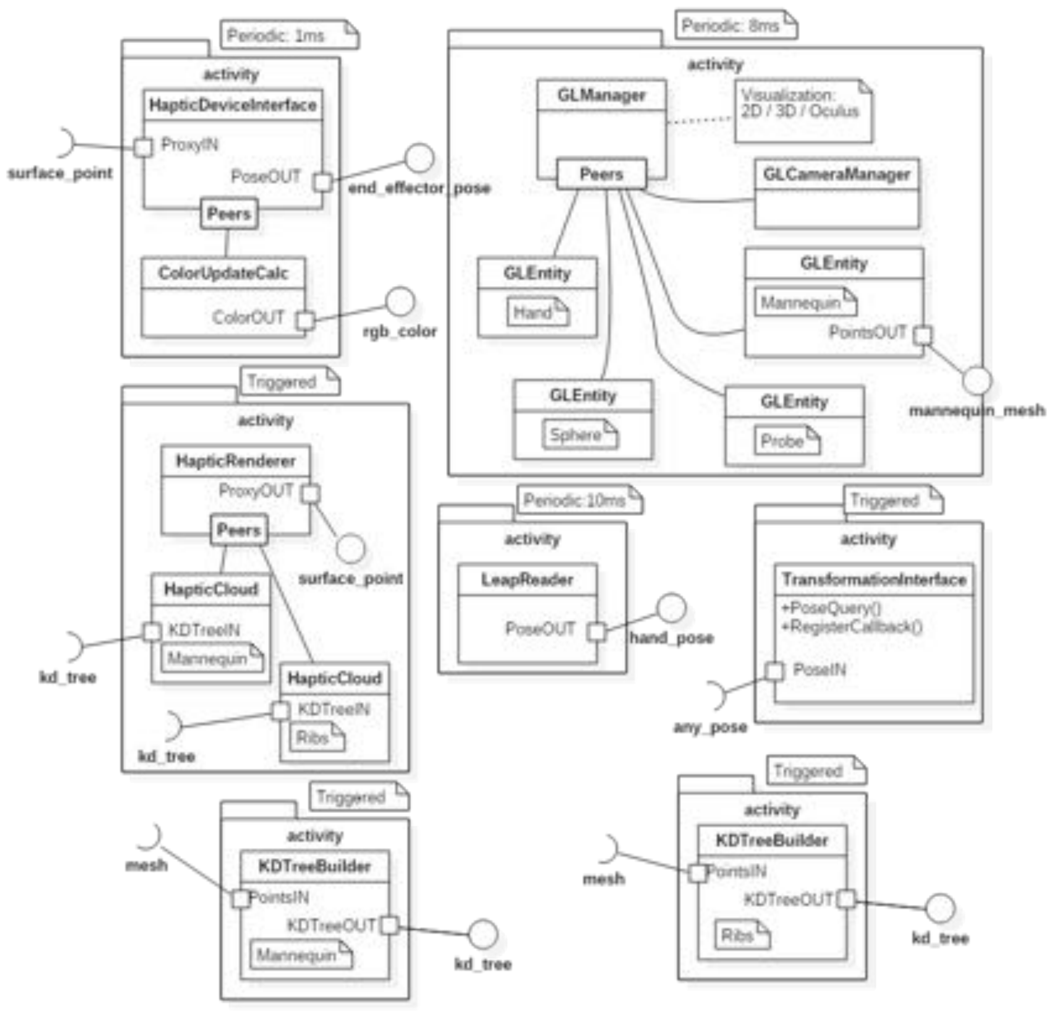


Figure 5.11: CoCo components, with the connections and the execution policies.

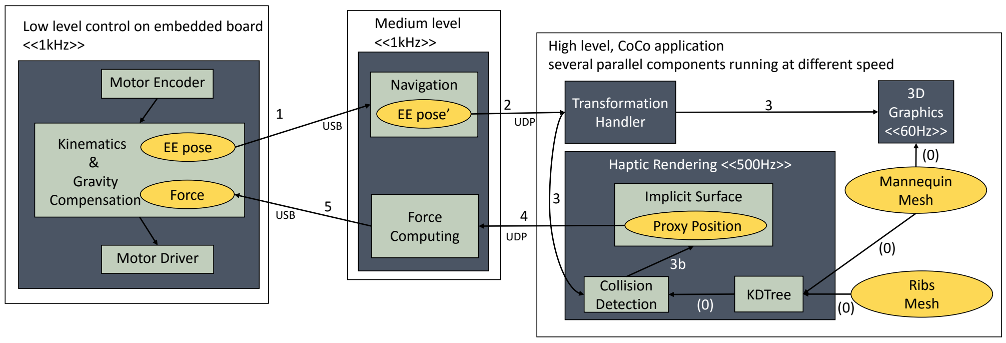


Figure 5.12: Structure of the software with the communication channels. Number on edges represents the order of message passing between components for each control cycle. (0) means that the message is sent only in initialization phase. Communication between high level components is done through shared-memory.

implemented.

The medium-level software is implemented as a Matlab Simulink model, but it is executed in the host computer and communicates with the low-level control via High-Speed USB (low-latency and 3Mbit/s effective throughput). Medium-level controls the navigation modalities and it computes the force to be displayed by the haptic device. The information needed to compute the force comes from the high-level software which is connected via local UDP and runs at 1kHz.

The high-level software is implemented in C++ using the CoCo framework (see Section 3). The high-level software is composed of several CoCo components running in parallel. Figure 5.11 shows all the components involved with their connections and execution policies. In particular, the high-level software is in charge of graphically rendering the VE for the different devices (3D screen or HMD), and to execute the collision detection algorithm between the probe and the virtual patient that is used by the haptic rendering module to calculate the final force. All the reference frames, transformations and calibrations are managed by a component embedding SRGViz (Chapter 4). The calibration procedure needed to use the haptic device with the HMD was explained in Chapter 4.7. The graphics loop runs at 60Hz while the collision detection module at 1kHz.

Figure 5.12 shows the overall software's structure together with the communication pattern. At each iteration, the medium-level software asks the end-effector position to the low-level control via USB, adjusts it based on the navigation modality and sends it to the high-level code dedicated component. In the high-level software the pose is adjusted to the correct reference frame and passed to the graphics component that uses it to display the virtual probe and to the haptic rendering module that computes the proxy's position on the patient surface. The proxy is sent via UDP to the force module in the medium-level software, which computes the force and sends it to the low-level code that drives the motors.

INTERACTION To use the device decoupled from a remote robot we designed a VE that contains a geometrical mesh of the patient's body created from the 3D scanning of a medical mannequin and the 3D representation of a real USG probe. In addition, we created a mesh of the rib cage according to the human anatomy and we placed into the virtual chest. This mesh is used to simulate the interaction with ribs, which is a key part in a USG examination. The rib cage is modeled by means of a set of toruses which are fused together. The toruses were placed and sized to replicate a rib cage between fourth and seventh ribs, i.e. perceiving ribs whose transversal section varies from 15 to 20 mm and whose distance varies in the range 15-20 mm.

There are two aspects that mostly characterize the interaction of a user with this interface. The first one is the haptic rendering, which regards in our case the algorithm chosen to calculate the force feedback based on the probe's and patient's position. The second one is about how to handle the differences between the device's workspace and the needed one. This problem affects the navigation technique.

HAPTIC RENDERING The force rendering is compute by the CoCo component described in Section 3.4.2. In order to simulate the interaction with the human chest comprising skin and ribs with different stiffness values, we have employed the composition of two separate 3 DoFs proxies each associated with a different geometry. The first layer represents the skin ($k = 300N/m$) whose geometrical model has been obtained by 3D scanning a mannequin, and the second inner layer represents the ribs ($k = 1000N/m$) whose geometry is obtained from the fusion of toruses via constructive solid geometry. The force is saturated at 15N in each direction to prevent damages in case of temporary communication issues.

NAVIGATION The horizontal workspace of the device is smaller than the normal extension of a human trunk (0.26m diameter against ∼[0.70m x

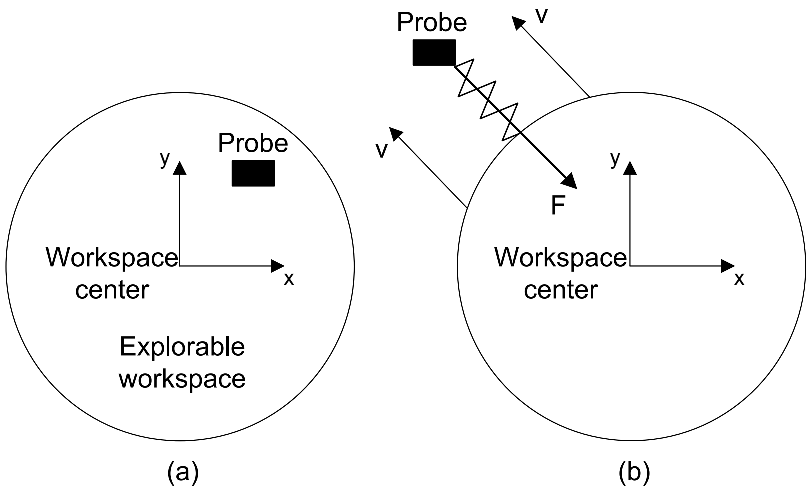


Figure 5.13: Bubble navigation. (a) When the DiagUI end-effector is in the central zone, it can be moved freely in the explorable workspace and its position is sent to the remote environment. (b) When the end-effector exits the central zone, the center of the workspace is translated in the direction of the end-effector.

0.45m]) causing many parts of the body to be unreachable. We excluded the possibility of using a bigger device with larger workspace because it could be perceived as cumbersome and difficult to be moved. Scaling and bubble navigation techniques were therefore selected and evaluated. The "scaling" technique (SN), consists of scaling the position of the device up in the remote or virtual environment.

The second implemented solution is the "bubble" technique (BN), in which the workspace in the virtual/remote environment is translated to make every point reachable without losing resolution. To do this we define a vertical cylinder, smaller than the device's workspace, in which the end-effector moves freely with a 1:1 movement mapping (see Figure 5.13 a). When the user brings the end-effector out of this space and moves it towards the end of the physical workspace, the workspace translates towards the end-effector and its velocity is proportional to the distance of the end-effector to the surface of the cylinder. At the same time, the user perceives a force opposite to the workspace velocity and proportional to the distance of the end-effector from the cylinder, as if a spring was connected from the end-effector to a massless moving cylinder (see Figure 5.13 b).

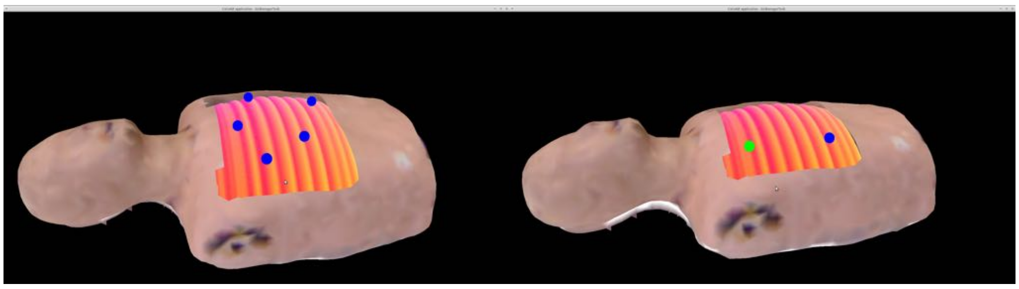
Our implementation differs slightly from the standard algorithm as it uses a cylinder as workspace delimiter instead of a sphere. This change was introduced because we need only horizontal displacements of the workspace.

When compared to the SN, BN may require more time to bring the probe in the desired position when moving between distant points.

LATENCY ESTIMATION One important aspect that may affect users' performance is the latency between user's input (e.g. user's hand position) and DiagUI's outputs (e.g. force rendering). Theoretically, at time t, given the position of the end-effector, it is necessary to compute the force F that the user should perceive. However, since rendering of F is mediated by the DiagUI, F is rendered to the user at a time $t' > t$, thus introducing a latency $\lambda_2 = t' - t$ which we aim to estimate.

The whole DiagUI works as a loop (see Figure 5.12) in which information is propagated to give the user feedback. This loop can be arbitrarily inspected to track the flow of information and to estimate the time the information needs to become feedback to the user. Based on this idea, we made two measurements of latency (λ_1 and λ_3) to set limits to λ_2, i.e. $\lambda_1 < \lambda_2 < \lambda_3$. First, we measured the loop latency λ_1 between the generation of a position signal in the low-level firmware to the associated force value as computed by the medium-level and high-level software components. Figure 5.12 shows the steps of this loop. λ_1 comprises the round trip time (RTT) of the communication between the embedded board and the host computer via USB. However, it does not take into account the time needed by the firmware and device control loop to move the end-effector, therefore it is a lower bound for λ_2. The measurement has been performed by tracking a timestamp generated by the low-level control and computing the closed loop latency inside the low-level control itself. Given a session of 157 s, i.e. 73527 samples, we obtained for λ_1 5.99 ms on average with 1.43 ms standard deviation.

In order to set an upper bound to λ_2, we control the robot to follow a sinusoidal path in position control. The position control is a PID (proportional-integrative-derivative) that has been implemented in the medium-level software. A sinusoid along the Z axis (amplitude 0.02 m, frequency 1.5 Hz) is generated in the medium-level software, sent to the high-level software and received back in the medium-level software to serve as target for the position control. The PID was tuned to have the same delay in the ascending and descending intervals of the position target. The latency λ_3 was then calculated as the phase between the target sinusoid and the actual position of the end-effector. This method includes the delay introduced by the position control, which actually depends on the chosen control algorithm. Since


(a) Experiment 1 and 2 (b) Experiment 3

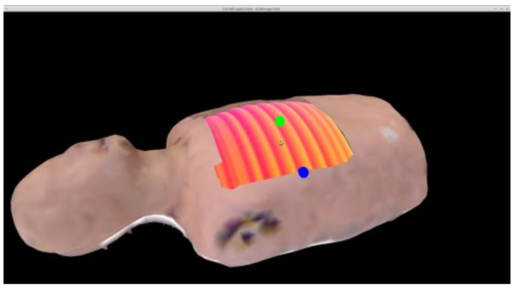

(c) Experiment 4

Figure 5.14: Sphere positions in the four experiments. the chest is removed to show the ribs. (a) The five spheres used in experiment one and two. (b) Two spheres of experiment three along with ribs in between spheres. (c) Spheres of experiment four and their placement on a ribs sulcus. In (b) and (c) the blue sphere is the task starting point.

the motors' current loop that makes the interface render the target force F to the user is much faster (5 kHz see [136]) than this position controller, λ_3 provides an upper bound of λ_2. Latency λ_3 was recorded during seven sessions (60s each). We obtained that λ_3 was 27.1 ms on average, standard deviation being 1.35 ms, which is below the limit found in the literature for producing an effect on performance (see [64] and [63]).

5.2.3 *Experiment Setup*

The goal of our study was to assess the usability of the interface and preferences of the experts related to the navigation techniques and the visualization modalities. Therefore, we formulated the following two hypotheses and related sub-hypotheses: **H1** : the DiagUI allows for accurate positioning of the probe. **H1-1** : navigation modality influences accuracy in positioning the probe over the patient. **H1-2** : visualization modality influences accuracy in positioning the probe over the patient. **H2** : the DiagUI allows the sonographer to detect the ribs of a patient. **H2-1** : navigation modality affects accuracy in counting the ribs of a patient. **H2-2** : navigation modality affects accuracy in following the sulcus between two ribs. These hypotheses are tied to tasks that are crucial for tele-USG, i.e. being able to navigate the patient's site environment and being able to perceive parts of the human body. We set up four experiments, which are described in section 5.2.3.1 and 5.2.3.2 (see Figure 5.10 a for the setup), to verify these hypotheses.

5.2.3.1 *Experiments 1 and 2*

These two experiments target hypothesis H1. Experts were asked to put the tip of the probe in a virtual sphere which was placed on the surface of the virtual mannequin. They were asked to be as accurate as possible, whereas no instruction was given about speed. The radius of the sphere was selected to be 1cm as this distance is comparable to the space between two ribs. Once the expert believed to have achieved the goal, s/he pressed a button of the joystick to mark the position of the end-effector and to move to the next sphere over the virtual patient. This was repeated for five sphere positions, whose selection is a compromise between the positions of the echocardiography basic acoustic windows and the need of spanning the whole thorax of the virtual patient, to make the navigation task more significant. The first experiment targeted hypothesis H1-1. Experts repeated the five spheres navigation task six times. The visualization modality was 3D on the LCD screen while participants wore glasses (3D-LCD). In the first three trials experts navigated the VE with the BN, and after with SN. The second experiment targeted hypothesis H1-2, i.e. to assess visualization modality. In this experiment, experts repeated the task nine times. Navigation was set to BN for all the trials whereas visualization varied: in the first three trials, the 3D VE was represented on the LCD screen in 2D (2D-LCD) by switching off screen's 3D modality. In trials 4 to 6, visualization modality was 3D-LCD, whereas in the latter trials the VE was displayed in 3D in the HMD (3D-HMD).

Accuracy in reaching the target points was the main performance measure. For each target point, error e was defined as the Euclidean distance between the marked end-effector point and the target sphere center. We also calculated the total distance and the total completion time for each trial as variables related to the expert's behavior during the task. In both experiments, outcomes of trials carried out in the same experimental condition were averaged.

5.2.3.2 *Experiments 3 and 4*

Experiments 3 and 4 targeted hypothesis H2. Given that hypothesis H2 deals with force feedback, we assumed that this is not affected by visualization, whereas it may be influenced by the navigation technique. Therefore, the visualization was always a 3D-LCD whereas navigation modality could be either BN or SN.

In the third experiment two spheres were shown in the VE, aligned to a plane which is parallel to patient's sagittal plane, placed on the skin and separated by five ribs (see Figure 5.14 b). Experts were asked to count the ribs between the two spheres taking advantage of the visual and the force feedback, i.e. looking at the screen and pushing against virtual patient's body to perceive the ribs. Expert were asked to focus on the rib count. Indeed, they were encouraged to iterate multiple times between the two spheres in case they were not sure of the result. Experts were asked to press the joystick's button when aligned to the starting sphere and ready to begin exploration and to press it a second time at the destination sphere when they were ready to tell the result. Each participant carried out six trials, the first three in BN condition, the latter in SN. The number of ribs between the two spheres was always five, but the participants had no clue about this. In the third experiment, the metrics of the performance was the number of ribs that was reported. We also checked that exerted forces were comparable to forces which are usually recorded in USG examination.

In the fourth experiment two spheres were displayed on the mannequin, but this time they were aligned to a transversal plane of the virtual patient. They were placed in a ribs sulcus so that an almost circular trajectory which is 0.177m long could be traced along the sulcus between the two spheres (see Figure 5.14 c). Experts were asked to follow the ribs sulcus moving from one sphere to the other. They were instructed to stay as close as possible to the sulcus bottom, without concerning about the completion time. Differently from experiment 3, they could not go back and forth to retry in case they realized they exited from the sulcus. This task is especially meaningful for USG. Indeed, once the probe is correctly placed on the point to be examined, sonographers exploit the support of the ribs to refine the pose of the probe to get the best window. As for experiment three six trials were carried out, the first three in BN modality, the latter in SN. Performance assessment was done as follows: we defined a "correct zone" along the target trajectory. This zone is delimited by a cylinder whose external surface contains the sulcus' bottom line and whose axis is parallel to the virtual patient's craniocaudal axis. Since the skin and the ribs are elastic it is possible to move the probe slightly under the surface. Therefore, when the probe is kept in the bottom of the sulcus the probe is in the correct zone. Two performance metrics were then established: the first, namely e_p, is the percentage of samples in which the probe was outside the sulcus' bottom line with respect to the total number of task's samples; the second, namely e_d, is the average distance covered outside the correct zone. This latter measure is normalized with respect to the theoretical trajectory's length. In both experiments, outcomes of trials carried out in the same experimental condition were averaged.

5.2.3.3 *Participants and experimental protocol*

Twelve experts (doctors and sonographers, average age 38.5, 9 men, 3 women), participated in the experiments. Prior to the beginning of the experiments, participants received and read a written description of the experiments and were asked to sign a consent form. They were also asked to complete a pre-experiment questionnaire. We then explained to them how the system works with a focus on the bubble navigation modality as it may not be intuitive, especially for users that are not familiar with haptic devices. Participants were then instructed about the goals of each experiment as described in 5.2.3.1 and 5.2.3.2. In the following familiarization phase, that lasted 5 to 10 minutes, participants could explore freely the VE using the haptic device and switch between the different visualization and navigation modalities. In addition to the haptic device, experts held the joystick in their non-dominant hand. The joystick served to mark specific phases of the experiments i.e. switching target point in experiments 1 and 2, marking the beginning and the end of the exploration in experiments 3 and 4. Before each experiment experts were reminded about the instructions.

Finally, they were asked to fill a questionnaire that allowed us to gather their preferences. The questionnaire was composed of 17 questions (see Figure 5.19 for the detailed questions). In questions one to eight, experts had to rate navigation modalities in relation to the task that they carried out. Questions 9 to 11 asked the same related to visualization modality. Questions 11 to 15 asked the expert to rate four aspects of the system's usability. Finally, the last two questions investigated expert's preferences: question 16 asked what combination of navigation and visualization modalities experts preferred to use, whereas question 17 asked which combination they consider more usable to carry out USG examination.

5.2.4 *Results*

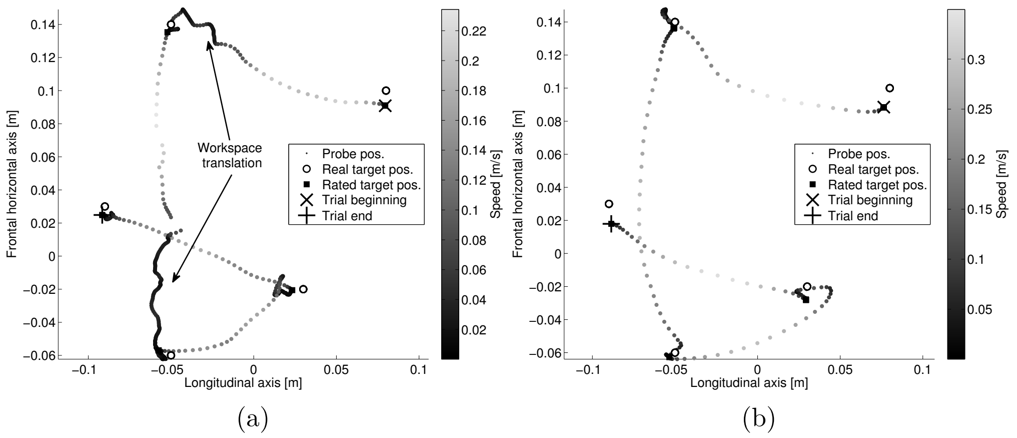


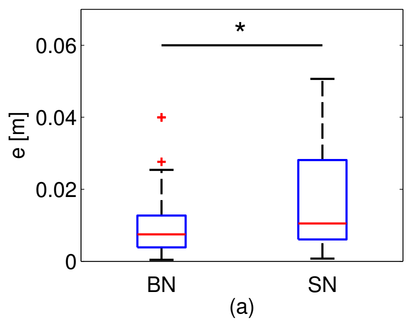
Figure 5.15: Scatter of end-effector positions during the task in representative Experiment 1 trials, position markers are coloured according to the speed. (a) Experiment 1 trial in BN condition. (b) Experiment 1 trial in SN condition.

This section reports the results of the experiments. In the following, boxplots report median as red lines, 25th ($q1$) and 75th ($q3$) percentiles in blue along with whiskers within 1.5 times $q3 - q1$ in black. Outliers are shown as red crosses.

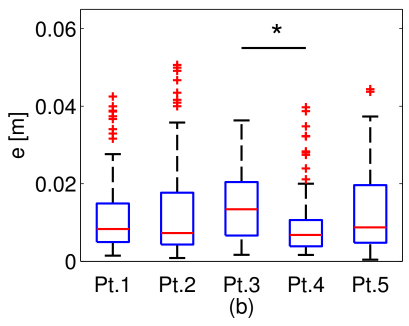
5.2.4.1 *Experiment 1*

First we show two representative trials in which navigation was set to BN and to SN (Figure 5.15 a and b respectively). The figure shows a typical behaviour, with some elongations beyond the targets in SN and with "pauses" between some of the points in BN conditions.

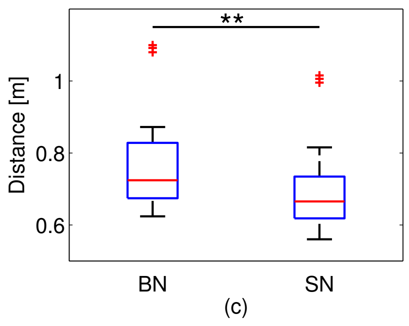
Experiment 1 has a factorial within-subjects design whose factors are navigation and point. Navigation has two levels whereas point has 5 levels, i.e. the task target points. We carried out a statistical analysis to check if outcome variables, namely error, distance and completion time were significantly influenced by such factors. Figure 5.16 a shows the error e in experiment 1. In general, e is smaller than 15 mm, often even smaller than the sphere radius. A better performance in BN condition is apparent



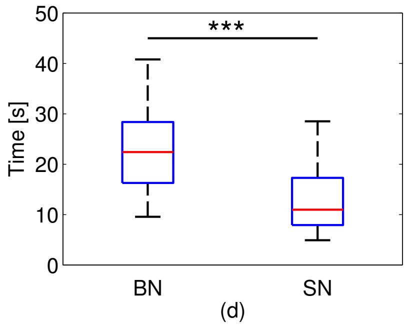
(a)



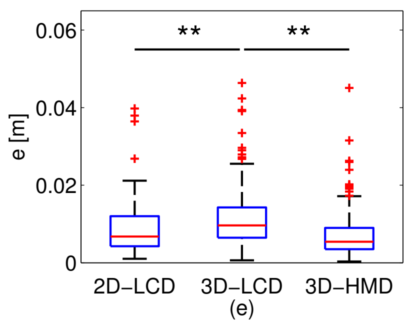
(b)



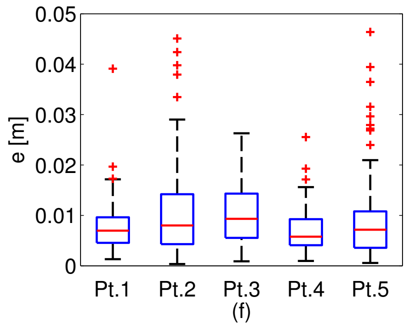
(c)



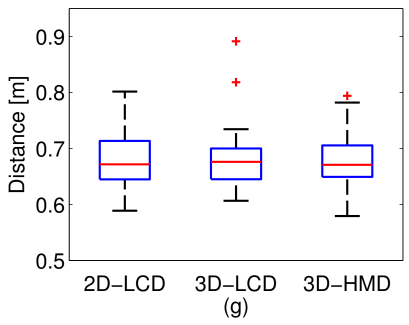
(d)



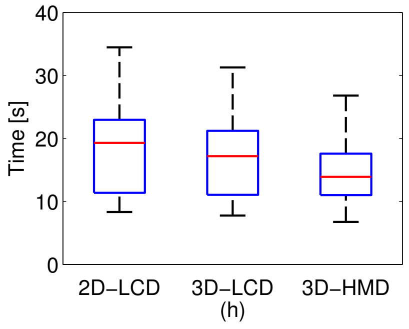
(e)



(f)



(g)



(h)

Figure 5.16: Error, distance and completion time results of experiments 1 ((a) to (d)) and 2 ((e) to (h)). (a) and (e): error e by navigation condition. (b) and (f): error e by target point, data were aggregated regardless of visualization and navigation modality. (c) and (g): distance covered per trial by navigation condition. (d) and (h) Elapsed time per trial by navigation condition.

but further analyses were carried out to confirm the result. Since e is a positive measure and the goal of the experiment was to minimize it, it was not normally distributed. This prevented from using t-tests and ANOVA to analyze it. However, the log-transformation of error measurement e is normally distributed, hence we used $log(e)$ in a repeated measures two-way ANOVA. All subsets of our data passed Shapiro-Wilk test [119] for normality and Mauchly test [76] for homogeneity of variance ($\chi^2(9) = 12.82$, $p = 0.177$ for point factor, $\chi^2(9) = 8.39$ $p = 0.502$ for interaction). Both navigation ($F(1, 11) = 5, 421$, $p = 0.04$) and point ($F(4, 44) = 2.675$, $P = 0.044$) were significant at $\alpha = 0.05$ level, whereas their interaction was not ($F(4, 44) = 1.238$, $p = 0.309$). A posthoc test with Bonferroni correction confirmed that accuracy was better under BN condition (see also Figure 5.16 a) than in SN ($p = 0.04$). The post hoc analysis shows that the only significant difference occurs between points 3 and 4 (see Figure 5.16 b), being point 4 only reached with a smaller error.

Similarly to variable e, the log-transformed of distance and completion time were normally distributed (we used the Shapiro-Wilk test to check for it). A paired t-test was carried out for each of the two variables. Distance covered in BN was significantly bigger than in the SN condition ($p = 0.002$), namely $0.81 \pm 0.14\,\text{m}$ and $0.73 \pm 0.13\,\text{m}$ for BN and SN respectively. To judge these values, we report that the theoretical minimum distance that is required to reach the five points in the given sequence is 0.46 m. A similar result was obtained for completion time: trials executed in BN condition required significantly more time than in SN ($p < 0.001$), namely $22.3 \pm 7.7\,\text{s}$ and $12.7 \pm 6.2\,\text{s}$ for BN and SN respectively. Figure 5.16 c and d plots show distance and completion time grouped by navigation condition.

5.2.4.2 *Experiment 2*

Experiment 2 has also a factorial within-subjects design in which the two factors are visualization and point. Visualization has three levels whereas point has 5 levels. We carried out a statistical analysis to check effects of such factors on outcome variables, namely error, distance and completion time.

Figure 5.16 e shows the error e in experiment 2. It is generally smaller than the sphere radius. A better performance in 2D-screen and HMD conditions is apparent but further analyses were carried out to confirm it. Likewise, experiment 1, we used $log(e)$ in a repeated measures two-way ANOVA to test the effects of the factors. All subsets of our data passed the

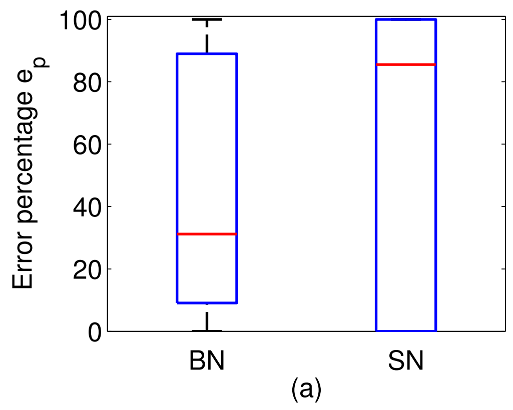
Shapiro-Wilk test for normality. Mauchly test for homogeneity of variance was passed except for interaction ($\chi^2(2) = 2.27$, $p = 0.321$ for visualization factor, $\chi^2(9) = 10.45$, $p = 0.324$ for point factor, $\chi^2(35) = 53.87$ $p = 0.044$ $\epsilon = 0.47$ for interaction). Results show that visualization ($F(2,20) = 8.523$, $p = 0.002$) has a significant effect on error, whereas point ($F(4,40) = 2.29$, $p = 0.077$) and interaction were not significant ($F(8,80) = 0.77$, $p = 0.630$). The Bonferroni posthoc test confirmed that accuracy was better under 2D-LCD and HMD conditions (see Figure 5.16 e) than in 3D-LCD (2D-LCD against 3D-LCD $p = 0.009$, HMD against 3D-LCD $p = 0.007$, 2D-LCD against HMD $p = 0.862$).

Distance and completion time were log-transformed to obtain normally distributed data, which was checked by means of the Shapiro-Wilk test. Mauchly test for variance sphericity was passed ($\chi^2(2) = 5.40$, $p = 0.067$ for distance factor, $\chi^2(2) = 3.03$, $p = 0.220$ for completion time factor). A repeated measures one-way ANOVA showed that visualization does not have a significant effect on covered distance ($F(2,20) = 0.593, p = 0.562$). However, a low power was observed for this test (0.135). ANOVA results show that visualization influences completion time ($F(2,20) = 6.06, p = 0.009$). After Bonferroni's correction for multiple comparisons, no significant differences among visualization modalities were left. However, completion time difference between 2D-LCD and HMD approached significance ($p = 0.055$). Figure 5.16 g and h plots show distance and completion time grouped by visualization condition.

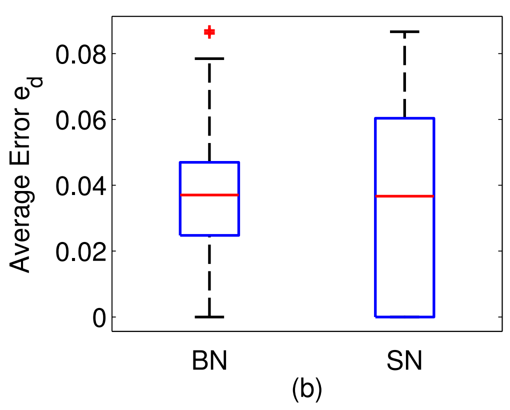
5.2.4.3 *Experiments 3 and 4*

Experiment 3 outcome is a binary variable as the experts could either be wrong or right regarding the number of ribs. The overall percentage of correct responses was 92.4%. Moreover, some experts who made mistakes reported that they could not judge whether to include the first and/or the last rib that they perceived. The percentage of correct answers grouped by navigation condition was 90.9% and 93.9% for BN and SN respectively. A McNemar test resulted in a non-statistically significant difference ($as.\, p = 1$) between BN and SN.

Results of experiment 4 are shown in Figure 5.17. Visual inspection of the plots suggests that percentage of error e_p was similar for the two navigation conditions, whereas BN seemed to be related to a smaller distance e_d out of the sulcus with respect to SN. A Wilcoxon test confirmed that the effect of navigation on e_p was negligible ($as.\, p = 0.386$).

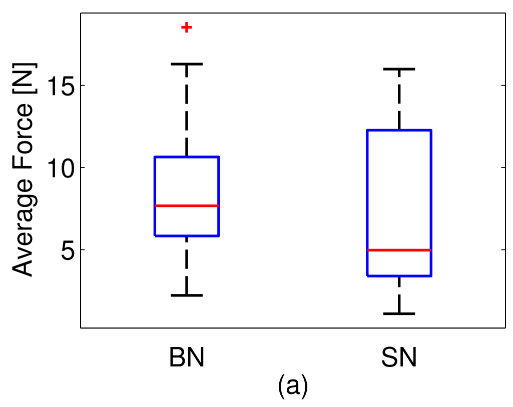


(a)

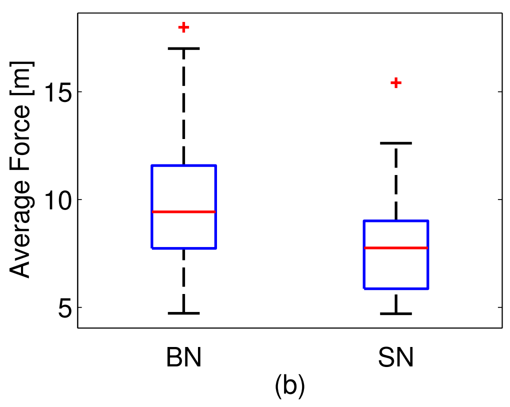


(b)

Figure 5.17: Percentage of error and average error of experiment 4. (a) Error percentage for each navigation condition. (b) Average error for each navigation condition.



(a)



(b)

Figure 5.18: Highest forces during experiments 3 and 4 trials. (a) Experiment 3. (b) Experiment 4.

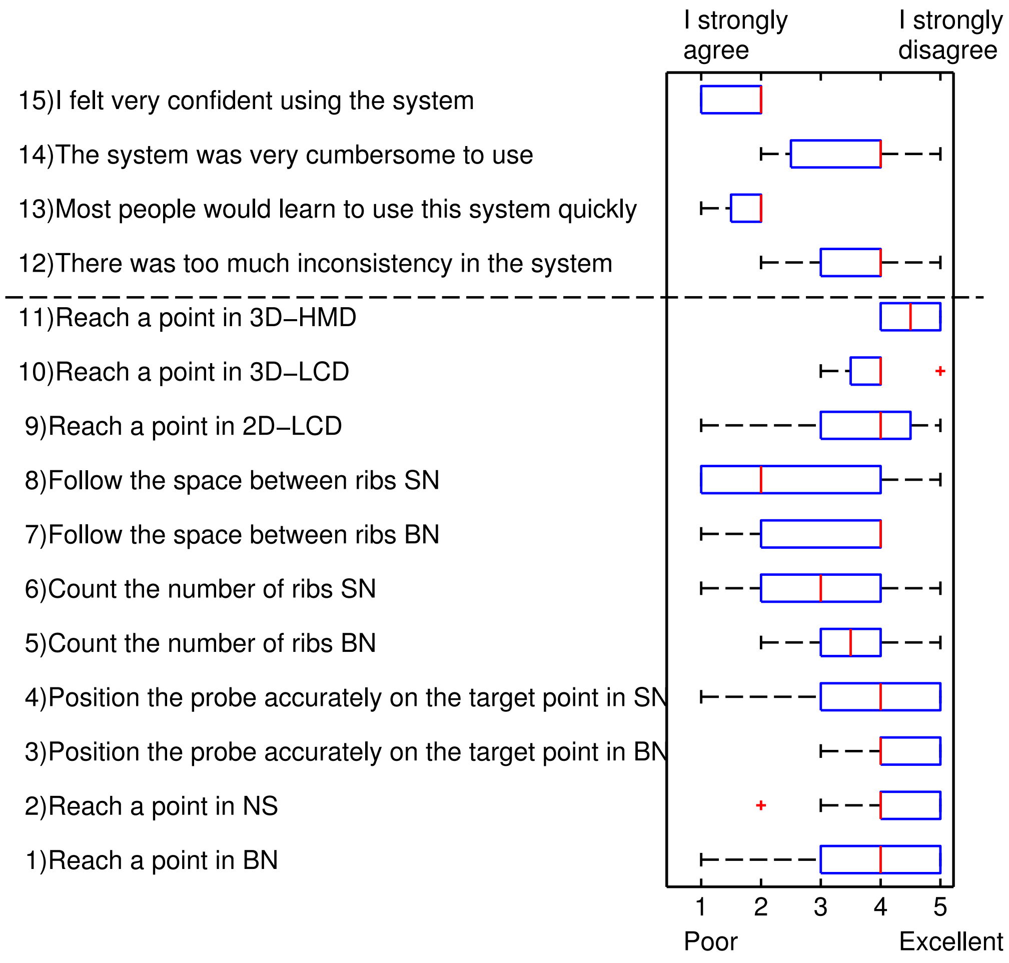


Figure 5.19: Questions 1-15 of the post-experiment questionnaire along with answers' statistics. Navigation and visualization scores (questions 1 to 11) range from 1, i.e "poor" to 5, i.e. "excellent". Usability scores (questions 12 to 15) range from 1, i.e "I strongly agree" to 5, i.e. "I strongly disagree".

As a further assessment of the interface, we report the forces that were experienced during the experiments 3 and 4, which are the only ones that required to interact with the virtual patient's body similarly to a USG examination. Figure 5.18 shows forces recorded during the experiments.

5.2.4.4 *Questionnaire*

This section reports the answers to the questionnaire. In particular, Figure 5.19 shows the first 15 questions that were formulated along with the scores provided as answers. Questions 1 to 8 were analyzed in couples to compare navigation techniques. In general, minor differences between BN and SN occurred. However, we noticed a slight preference towards SN for reaching a point, whereas BN reaches higher scores for accurately position

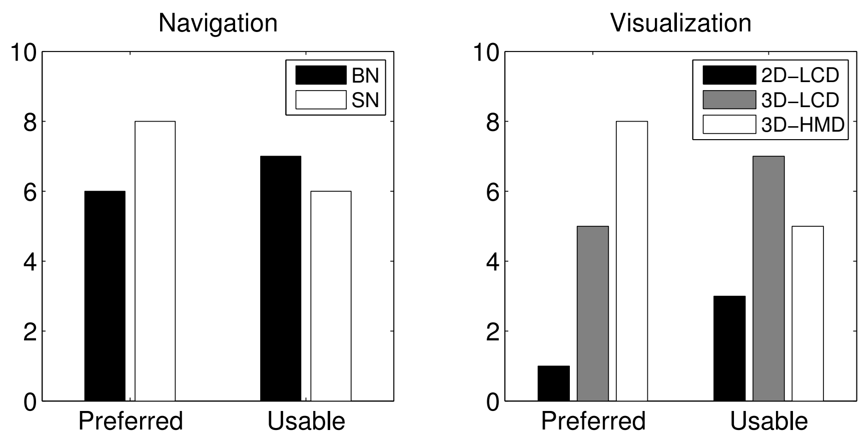


Figure 5.20: Preferences reported by experts. Bars shows the amount of preferences per condition. “Preferred” and “Usable” refer to questions 16 and 17 respectively.

the probe. Looking at the tasks involving haptics (questions 5 to 8), we see that BN is slightly preferred to SN for rib counting, whereas it has much higher scores than SN for following the sulcus between two ribs. Since differences were small, we did not perform further analysis on questions 1 to 6, whereas we performed a Wilcoxon signed-rank test to compare answers to questions 7 and 8. Although a high difference in the median of the answers (4 against 2), the test did not elicit a statistically significant difference ($Z = -1.594$, $p = 0.111$). Questions 9 to 11 allow us to compare the visualization modalities. Figure 5.19 shows high and comparable scores of LCD representation of the patient’s site, whereas 3D-HMD seems to be preferred with respect to the other conditions. A Friedman test showed that this difference is not statistically significant at 5% level ($\chi^2(2) = 5.615$, $p = 0.06$).

From questions 16 and 17 we obtained the results reported in Figure 5.20. We remark that experts were allowed to express multiple preferences. Although there is not a strong preference towards BN or SN, we notice that experts preferred to use the system in SN modality but they consider BN to be more usable in a USG examination. A similar pattern occurs for visualization: experts preferred to use the head mounted display, but they think that a 3D visualization on an LCD is better to do a USG examination. In general, results show a strong preference towards 3D visualization modalities.

5.2.5 *Discussion*

In the results of navigation experiments 1 and 2 (see Figure 5.16 a, b, e, f), we note that the median error e that the experts committed in positioning the probe is almost always below 1 cm, which is less than the sphere's radius. This result supports a positive answer to hypothesis H1: the DiagUI allows for accurate positioning of the probe in terms of requirements of a USG examination.

We notice also, when considering BN+3D-LCD experimental condition of experiments 1 and 2, that e does not vary significantly, thus excluding possible training effects on performance. Experiments 1 and 2 gave us answers in relation to hypotheses H1-1 and H1-2. Figure 5.16 a shows that navigation condition has an influence on the accuracy that experts achieve. Results confirm findings of Dominjon et al. [38] as BN makes experts improve the accuracy. Bigger differences were found in the total distance that experts had to cover to complete the task and in the total completion time. Figure 5.16 c and d show that SN technique makes the user complete the task faster and with a smaller distance covered. Although being in contrast with results in [38], these results support the qualitative observations that we made after a visual inspection of the trajectories (see examples in Figure 5.15). In SN a co-articulation of motion is indeed apparent, and despite small elongations beyond the target points, the total covered distance is less than in BN thanks to a smoother motion. A smoother and faster motion is likely to be the cause of the preference that the experts expressed towards SN (see Figure 5.20 navigation plot). Moreover, experts reported (Figure 5.19 questions 1 and 2) a preference towards SN for reaching a point. However, experts have perceived a better control in positioning in BN condition, as supported by quantitative results (Figure 5.16), as they rated BN better than SN to accurately position the probe (Figure 5.19) and they considered BN more usable than SN (Figure 5.20 navigation plot). Although the error e was small in each visualization condition, the analysis of visualization conditions resulted in an unexpected deterioration of performance when using 3D visualization on a flat screen. This result might have been due to a misleading representation of the third dimension on the flat screen.

Performance did not differ significantly between 2D-LCD and 3D-HMD conditions, in agreement with the variable results reported in [114]. Differently from navigation, visualization condition influenced significantly neither the covered distance nor completion time (though a main effect on completion time approached statistical significance). Anyway, Figure 5.16 h shows that completion time is lower with 3D visualization and in particular with HMD, thus suggesting a more comfortable interaction with 3D visualization. Questionnaire results confirm this suggestion as experts preferred 3D technologies and in particular HMD to navigate the VE (Figure 5.19 questions 9 to 11 and section 5.2.4.4). Interestingly, from the questionnaires (Figure 5.20) emerged that experts perceived 2D visualization neither as preferable nor as usable. However, in 2D-LCD condition they generally performed as good as or even better than in other conditions. We suppose that, in 3D conditions, the comfort and the pleasure they experienced and that they reported us made them feel that they performed better.

Experiment 1 and 2 also highlighted that the error was not the same for the five points (see Figure 5.16 b and f), being 3 the point with the highest error and 4 the point with the lowest. This difference achieved statistical significance in experiment 1. The large error in point 3 may be due to the long trajectory that links point 2 and point 3. Moreover, this trajectory was almost perpendicular to the screen, thus favoring mistakes especially with LCD screen. This hypothesis would also explain that the smallest errors occurred in 3D-HMD condition.

Hypothesis H2 and related sub-hypotheses are then discussed. Experiment 3 showed that experts could recognize the correct number of ribs in most of the case regardless the navigation condition, which does not influence significantly the percentage of correct responses. The questionnaire confirms a substantial equivalence of the two navigation conditions with a small preference towards BN (see Figure 5.19 questions 5 and 6). Experiment 4 shows that experts could successfully follow the sulcus between two ribs. The high error percentage (Figure 5.17) is due to the strict criterion that we set to highlight performance differences between BN and SN. Indeed, the average distance from the sulcus does not exceed 9% of the sulcus' length and is below 4% on average. Therefore, we can answer positively to H2. This experiment (see Figure 5.17) along with answers to questions 7 and 8 of the questionnaire (see Figure 5.19) suggest us that navigation modality plays a minor role in this task, in which haptic feedback is dominant. Therefore, we can draw conclusions with respect to H2-1 and H2-2, i.e. navigation modality does not influence performance in USG examination-like tasks in which haptic feedback is more important than visual feedback.

The analysis of forces during experiments 3 and 4 shows that the maximum forces that we obtained are consistent with the literature. Indeed, Salcuedan et al. in [111] reports forces up to 10 N, Vieyres et al. in [134] found $5-20$ N as a suitable range, whereas Gilbertson et al. measured up to 20 N in [55]. This result supports our hypothesis that the selected tasks replicate USG examination conditions in terms of haptic feedback. We noticed also, that in both tasks involving force feedback experts exerted higher forces in BN condition.

Finally, experts reported positive opinions about the system (Figure 5.19 questions 12 to 15). Although they used the system for less than 30 minutes, they felt very confident using it. Accordingly, they reported that it is easy to learn using the system. A few experts found some inconsistency in the system mostly due to the absence of ecographer's images. Some users reported that the system is cumbersome, thus supporting our design decision of using a haptic device whose workspace is smaller than patient's body.

5.3 CONCLUSION

In this chapter we have seen how AR can contribute to creation of a telemedicine DiagUI for palpation, and how a VE can be used to validate a DiagUI for USG examination. In both cases AR and VR have allowed to test the doctor's interfaces without the need to couple them with expensive robots. AR and VR allow researchers to develop and test new consoles in isolation, reducing costs and time. In addition this allows to create general interfaces that can be coupled with several remote robots, provided that they agree on the data they exchange.

The USG DiagUI was tested with real doctor that have proven it feasible to be used for real examination. In addition it showed that it is possible to create robotic interfaces with a controlled low learning curve, even for people not familiar with robotic devices or even simple joysticks.

5.4 PUBLICATIONS

Emanuele Ruffaldi, Alessandro Filippeschi, Filippo Brizzi, Juan Manuel Jacinto, and Carlo Alberto Avizzano. "Encountered haptic augmented reality interface for remote examination". In: *3D User Interfaces (3DUI),*

2015 IEEE Symposium on. IEEE. 2015, pp. 179–180

Alessandro Filippeschi, Filippo Brizzi, Emanuele Ruffaldi, Juan Manuel Jacinto, and Carlo Alberto Avizzano. “Encountered-type haptic interface for virtual interaction with real objects based on implicit surface haptic rendering for remote palpation”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE. 2015, pp. 5904–5909

Emanuele Ruffaldi, Filippo Brizzi, Alessandro Filippeschi, and Carlo Alerto Avizzano. “Co-located haptic interaction for virtual USG exploration”. In: *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE.* IEEE. 2015, pp. 1548–1551

Alessandro Filippeschi, Filippo Brizzi, Emanuele Ruffaldi, Juan Manuel Jiacinto Villegas, and Carlo Alberto Avizzano. “Diagnostician Haptic Interface Study for Tele-Echography Examination Feasibility”. In: *IEEE Transactions on Human Machine System* (2017). **Paper resubmitted as new**

6 TELEOPERATION IN AN INDUSTRIAL SCENARIO

The recent advances in robotics are characterized by an impressive evolution of humanoid robots that culminated in the DARPA Robotics Challenge in June, 2015. Humanoid robots are designed to operate in human environments with minimal adaptations. This type of robot can be used, particularly, in hazardous scenarios, such as rescue missions and industrial manufacturing. Despite humanoid robots' capability to autonomously perform a gamut of tasks, those robots still need human guidance when executing complex tasks, typically by means of teleoperation. In the particular case of teleoperation through body-based interfaces, humanoid robots have proved to be ideal, thanks to the capability of straightforward mapping between the human operator's motion and the robot's motion [124, 27, 79].

Nevertheless, there are several aspects that render teleoperation a non-trivial task, such as delay and high latency in the end-to-end communication, visualization issues of remote environment, and difficulties in identifying the right objects to interact with, as also in judging the objects' distances from the robot end-effector [86]. Producing a teleoperation system that can provide the operator with the same quality and quantity of visual information as what would be possible when the operator is physically present in the remote place, is an extremely complex task. The less complex methods utilize a static camera and a monitor [14, 56] to provide a monovision feedback, where the perception of distances is constrained by the lack of parallax. Such constraints are usually overcome through the learning process of the operator. When the teleoperation scenario requires an advanced form of visual feedback, either multi-modal feedbacks are integrated [122] or the operator is "cheated" by utilizing some advanced visualization technique to increase his/her illusion of being in the remote place [46, 52]. Usually complex visual feedback requires implementation of a virtual version of the remote environment. But, using of virtualized environments leads to perceptual issues, such as the perception of egocentric distances [99], and loss of visual acuity and contrast [74], which have been empirically proved to influence the action capabilities of the body [73].

AR has been proved [81] to be a viable solution to overcome visual feedback limitations by providing additional information to the operator.

AR solutions for teleoperation can be divided into two categories, which are mutually complementary: embodiment enhancement and virtual fixture.

Embodiment enhancement refers to the capability of making the operator feel that s/he is in the remote environment, and s/he is the robot [144]. Such capability is obtained by placing the camera on the head of the robot and showing the received image stream to the operator by means of an HMD. The vision problem associated with this camera-based approach is generically considered trivial, although it is often one of the major causes of poor performance in teleoperation tasks. The motivation behind using this method is enabling the operator to see the current position of the end-effector by eliminating the problems caused by the limited field of view and low resolution of the cameras, and the bulky size of the robots, which obstructs the scene's view.

The second category of AR solutions employs virtual fixtures [100] to help the operator in overcoming the difficulties of perceiving the remote environment, and to guide him or her in accomplishing the tasks. Virtual fixtures refer to virtual images or objects, overlaid on the remote scene, to provide the operator with visual cues that highlight the points of interest and useful information for accomplishing the given task [115]. Virtual fixtures have been proved to be capable of speeding up the execution of a teleoperation task, especially in high latency scenarios [141]. They also serve as an effective tool in teleoperation by providing sensory substitution, particularly in perceiving a force feedback, in the absence of haptic devices [70].

Although AR has been successfully applied to reduce the rate of task error in many robotic teleoperation scenarios, no studies were carried out so far to assess the impact of multiple AR features.

This work contributes to fulfill this requirement by studying the effects of AR in a generic industrial assembly scenario. In particular, this work addresses the use of both task-related and non-task related features and their combination, by quantifying, the features' effects not only on the task performance, but also on the sense of tele-presence and embodiment for the operator. The study shows that specific findings are associated with different types of operators' expertise in AR/VR and gaming.

6.1 SYSTEM SETUP

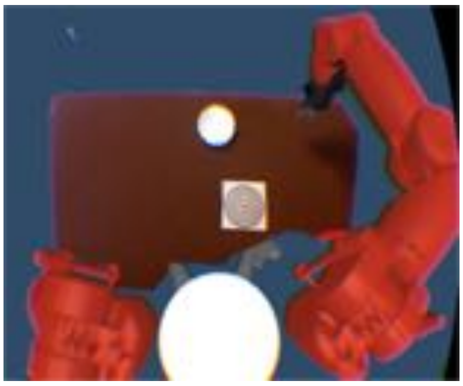
The fundamental question addressed in this work is how different types of AR features impact the operator's experience and performance. The other issues addressed here include the following: 1) understanding the extent to which excessive visual information can be detrimental and 2) evaluating the non-task specific AR features. The focus of this work is, therefore, on exploring the effects of 3D AR feedback on the presence, embodiment and ease of task execution, while accounting for the possible effects of participants' expertise. Effort was made to minimize the learning effect that could arise during the execution of the experimental trials.

All these questions were sought to be answered by undertaking an experiment that simulates an industrial assembly scenario; together with telemedicine, this scenario forms the main application field for AR. The task involves a pick-and-place operation by teleoperating a robot, seen from an egocentric point of view. The task execution is parametrized in terms of completion time and placement accuracy. Furthermore, the operators' hand trajectories were analyzed, together with their subjective sense of telepresence and embodiment in different conditions. Also, the combinations of the chosen features were evaluated.

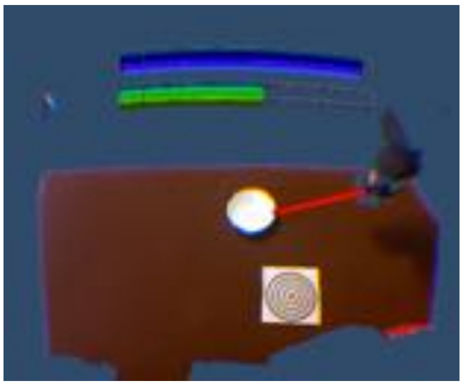
6.1.1 *Augmented Reality*

To create accurate and effective AR feedback, it is necessary to extract from the remote environment as much information as possible, exploiting all the available sensors. In the scenario presented in this work the available sensors were an RGB-D camera and the robot's joints encoders. Exploiting the state of the art computer vision algorithms, it is possible to track the pose of the target objects to be manipulated. The calibration between the robot and the camera allows for co-locating the robot and the remote environment by obtaining their exact relative poses. This information can be used as an AR feature, because it is not directly inferable by humans owing to the difficulty in perceiving distances accurately in a virtual scenario [85].

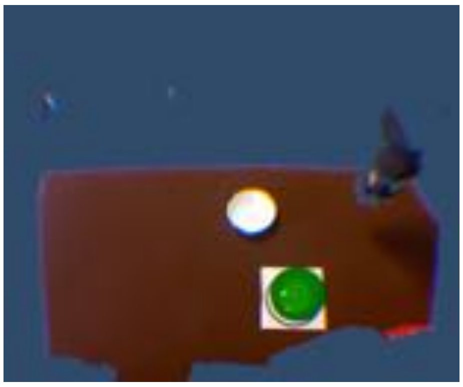
Different types of augmented information were implemented. The features can be categorized into two classes: **Embodiment** and **Visual Virtual Fixtures**. These two categories of AR features were chosen, because they are considered to be possibly the most informative ones to aid task execution.



(a) **Embodiment.** Image of the 1:1 scale model of the remote robot animated by the real robot's movements. The overlaid model allows to see the position of the robot limbs even if not in the real camera view space. The picture shows the precise overlay of the 3D model of the robot with the point cloud from the 3D camera.



(b) **Manipulation information.** Blue bar: gripper closure. Green bar: distance from target. Red beam: shortest trajectory from the robot's end-effector to the bowl's grasp point.



(c) **Task information.** In the target pose a green 3D mesh of the task object (bowl) is placed. The placement of the object succeeds if the real bowl and the green one perfectly match.

Figure 6.1: Three different groups of AR features (a, b & c) used in the experiment.

Table 6.1: The implemented visual cues and the corresponding classes and groups.

Class	Group	Feature
Embodiment	-	3D Robot Model
Virtual Fixture	Manipulation Information	Trajectory to the object grasping point (3D beam) - Distance from the target object (color bar) - Gripper closure (color bar) - Mesh of the object to be grasped
Virtual Fixture	Task Information	Objects Target Poses

The embodiment class comprises information that can help the operator in improving the overall sensation of embodiment and illusion of presence. In the setup used for this work, the embodiment class comprises two features. The first feature enables the operator to explore the remote environment by changing the virtual viewpoint with head and body movements. The second and novel feature is a virtual 1:1 scale model of the remote robot that is animated by the real robot's movements (see Figure 6.1 a). This feature allows the operator to see the position of the robot's end-effector when it is not visible in the camera's field of view; besides, it helps the operator in understanding the position of the robot in the remote environment.

The chosen visual virtual fixtures provide different types of information, which can be classified into two main sub-groups, according to the information they deliver: **Manipulation Information** and **Task Information**. The fixtures that belong to the Manipulation group (see Figure 6.1 b) deliver additional information, relating to object manipulation, such as distances from the object's grasp points (green bar) and robot grip closure (blue bar). They also include a red 3D beam representing, in real-time, the optimal trajectory between the robot end-effector and the closest grasp point of the object. The visual virtual fixtures that belong to the Task group are used to highlight information relating to the task execution. They are characterized by a green 1:1 3D mesh of the real object, placed in the task's target pose (see Figure 6.1 c). AR features that deliver information, relating to the task execution, have been demonstrated to significantly reduce error rates in assembly tasks [128].

6.1.2 *Implementation*

The AR feedback component was developed using CoCo (see Chapter 3), and SRGViz (see Chapter 4). CoCo component *MeshReconstructor* is used to create a 3D mesh from camera streams and to augment it with additional information. In particular, the first module receives and decompresses the video and depth streams from the camera (*StreamingReceiver*). The RGB channel is streamed (*StreamingSerever*) using H.264 compression (435kbps on average) and the depth channel with zn16 compression from OpenNI2 (21Mbps on average). The second module (*MeshReconstructor*) reconstructs a point cloud with interpolated points from the decompressed buffers. This is done to enhance the quality of the mesh of the virtual scene. The third module *GLManager* does the graphic rendering for the augmented scene. The end-to-end latency of the visualization is 89ms, as computed after synchronizing the robot and graphics computers with the Precision Time Protocol (PTP) [32]. Sections 3.4.1 and 3.4.3 provide additional information on the implementation of these CoCo components. The CoCo application is run as ROS node so to be able to communicate with the robot. Figure 6.2 shows the overall system architecture.

On the operator's side, a wearable device captures the movements of the operator's upper limb and his pinch grip position through inertial sensors and a custom haptic device [57]. The raw sensor data was sent, via wireless, to the main computational unit, where several ROS nodes reconstruct the operator's motion and combine it with the grip position to generate a control signal for the teleoperated robot (Baxter Robot, Rethink Robotics, Boston, Massachusetts, USA). On the remote robot side, the environment in which the robot was acting was captured through a Kinect 360 camera (Microsoft, Redmond, Washington, USA) placed on the top of the Baxter's head, as shown in Figure 6.5. The camera was not actuated and its field of view was fixed with respect of the robot's pose. From the captured point cloud, a virtual scene was created, which was augmented by the main computational unit with AR information coming from the ROS Control node. The visual appearance of the robot model used in the embodiment feedback is based on the URDF of the Baxter robot. The 3D AR scene was sent as a visual feedback to the operator's side and visualized with an HMD (Oculus Rift DK2, Facebook, Menlo Park, California, USA). Further details about the motion reconstruction algorithms, the HI and the control architecture can be found in [92, 57].

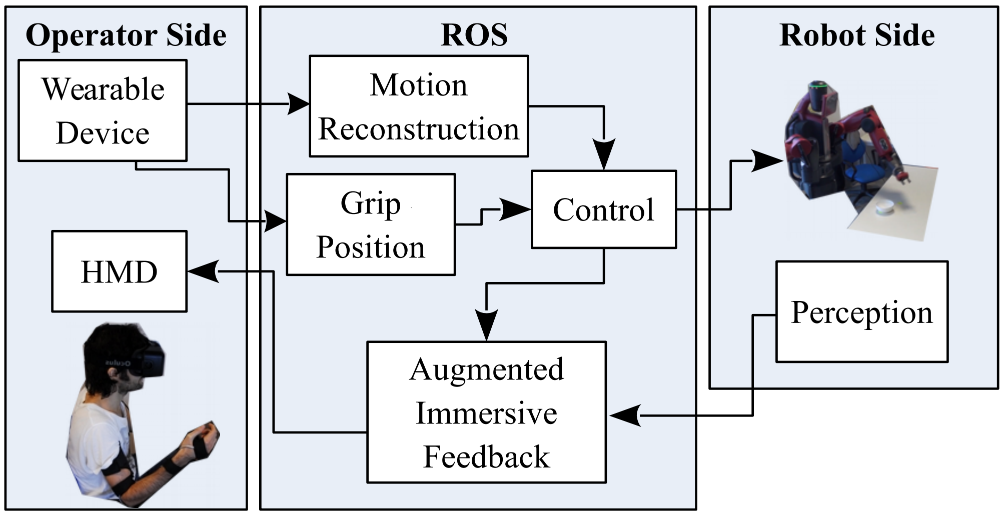


Figure 6.2: The system's setup. On the operator's side, the upper limb motion and the grip position were captured through a wearable interface. Operator's motion was reconstructed and used together with the grip position, to generate a control signal for the teleoperated robot. The remote scene was captured and virtualized; AR information was added to the virtual scenario, which can be visualized on the operator's side as a 3D visual feedback.

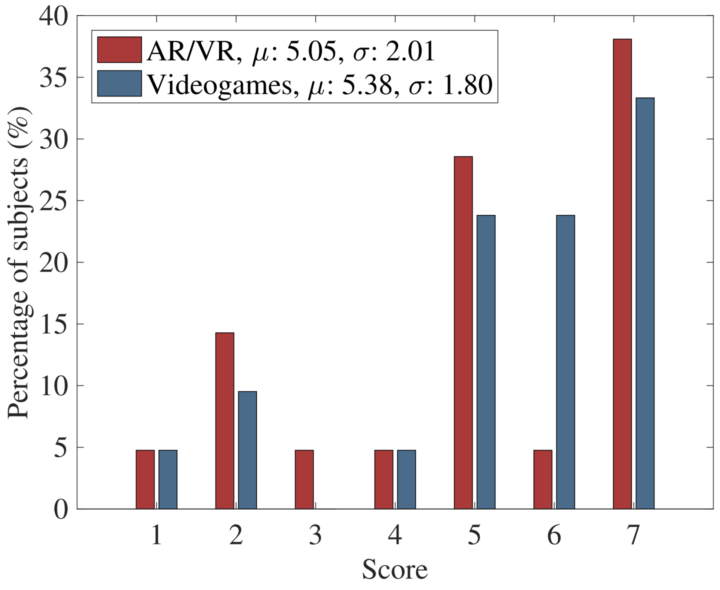
6.2 EXPERIMENT SETUP

6.2.1 *Participants*

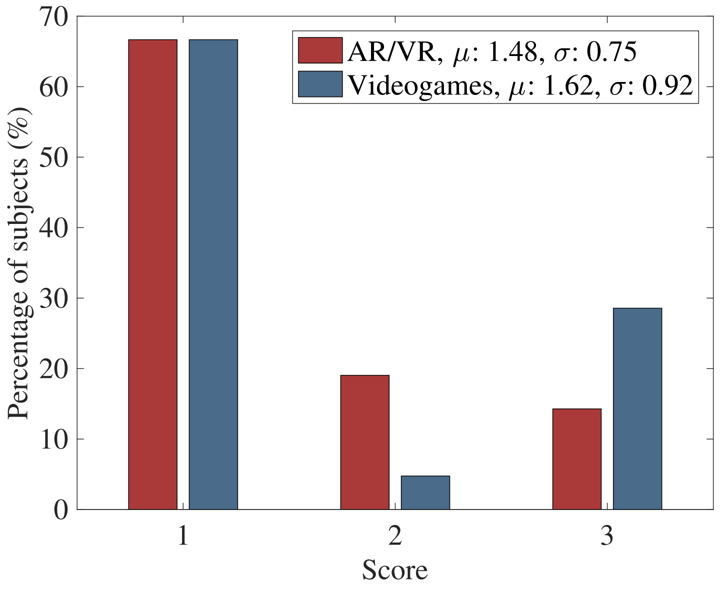
Twenty-two operators (16 male, 6 females), aged between 23 and 40 years, all right-handed, participated in the study, after giving their informed consent for participation. Their familiarity with AR and VR systems was assessed through a written questionnaire, in which they were asked to state their experience with AR/VR and in video games on a Likert scale (1 to 7), and in terms of number of hours they spend on them per week. For this, they were given the option of choosing between three answers: less than one hour, between two and five hours, and more than five hours. Their answers to the questionnaire are reported in Figure 6.3.

6.2.2 *Experimental Protocol*

The participants reported to the laboratory for a single experimental session. Prior to the commencement of the session, they were asked to read a written description of the experiment and then fill-in a questionnaire with answers



(a) Questionnaire results of experience assessment



(b) Questionnaire results of time spent

Figure 6.3: The results from the questionnaires regarding (a) the expertise and (b) the hours spent weekly, interacting with AR/VR environments and in playing video games. The answers were to be given on the Likert Scale 1 to 7 for the expertise, and 1 for less than one hour, 2 for two to five hours, and 3 for more than five hours, time spent interacting weekly with AR/VR and in playing video games.

that enable assessment of their expertise. After mounting the wearable interface, the participants were asked to perform a series of familiarization trials, followed by experimental trials that involved performing remotely a pick-and-place task of a given object by teleoperating the Baxter robot, simulating the classical step of an industrial assembly task. The object chosen for this task was a plastic bowl, which measured 16 cm in diameter and 7cm in height. The reason behind choosing the bowl was it had radial symmetry that helped in grasping without introducing any rotational error while placing it. To normalize the task execution among the participants, the starting and target positions of the bowl on the table were fixed. A post-experiment questionnaire was used to assess the effects of the AR on the participants' experience during the experimental trials.

6.2.2.1 *Familiarization*

The familiarization trials were conducted for allowing the participants to practice sufficiently so that they can perform the task correctly in its entirety and minimize the effects of their learning during the experimental trials. In particular, the participants had to perform the task of each trial within the time frame given by the experimenters. This was done to ensure prediction of the motion of the remote robot end-effector with respect to the participant's upper limb motion, control of the robot over the full manipulation workspace and proper grasp and release of the remote object with the remote robot gripper. The familiarization trials were performed for approximately five minutes. During the first trial, the participants were allowed to share the working space with the robot (i.e., without wearing the HMD) and teleoperate it during several pick-and-place tasks. During the second familiarization trial, the participants had to teleoperate the robot, using the visual feedback (i.e., wearing the HMD) without AR information, thus visualizing only the virtual remote scene. During familiarization, the target object was the same as the one in the experiment, but the pick-and-place locations were randomized. The participants performed the trials until they and the investigators were confident that the task could be performed in its entirety. Overall, the participants required 3-4 repetitions of the pick-and-place trial to become proficient in executing the aforementioned aspects of the task.

6.2.2.2 *Experimental Trials*

The participants were required to perform the pick-and-place task of grasping and moving the object as accurately as possible to its target pose. They executed the task under five different modalities of visual feedback: 1) no AR information; 2) only AR information relating to the Manipulation class, is visualized in the feedback; 3) only AR information, related to the Embodiment group, is visualized; 4) only AR information, relating to the Task group, is visualized; 5) full AR feedback (all the features relating to Embodiment, Manipulation and Task are activated simultaneously). Figure 6.4 shows the remote scenario for the task, wherein the starting point (the cross on the table) and the target point (the circular marker) are visible, and the robot is grasping the target object. On the left is the virtual scene, as displayed to the participants and on the right, the real environment.

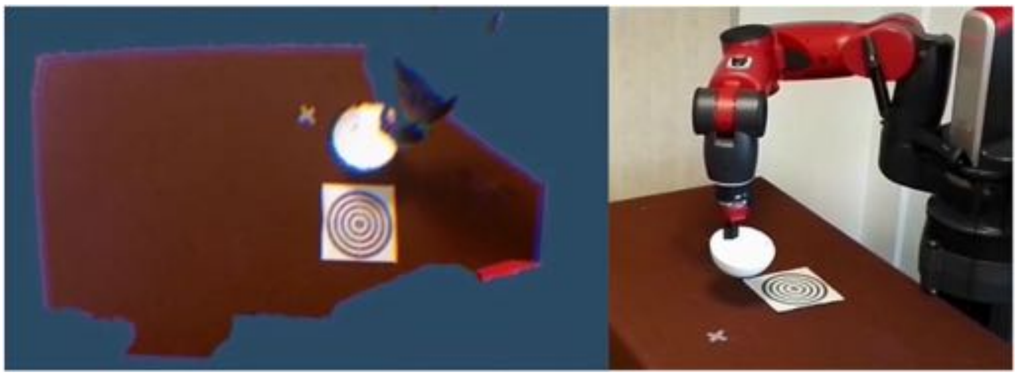


Figure 6.4: The virtual and the real remote environments captured simultaneously. On the left is the 3D image, captured from the camera and rendered in the HMD, and on the right is the same image from the real environment. In both the images, it is possible to see the robot grasping the target object (bowl), the starting position of the object (cross marker) and the target position (circular marker).

To minimize the learning effect, the trial order was randomized. To simulate the effect of non-co-location between the robot and the operator, the operator was acoustically isolated from the environment and was asked to teleoperate the robot over the Internet. For full experimental setup, see Figure 6.5.

To assess the effects of various types of AR feedback, several variables, relating to task execution performance and task execution modality, were monitored. The task completion time and the accuracy of each placement of the object were used as metrics for defining the execution performance.

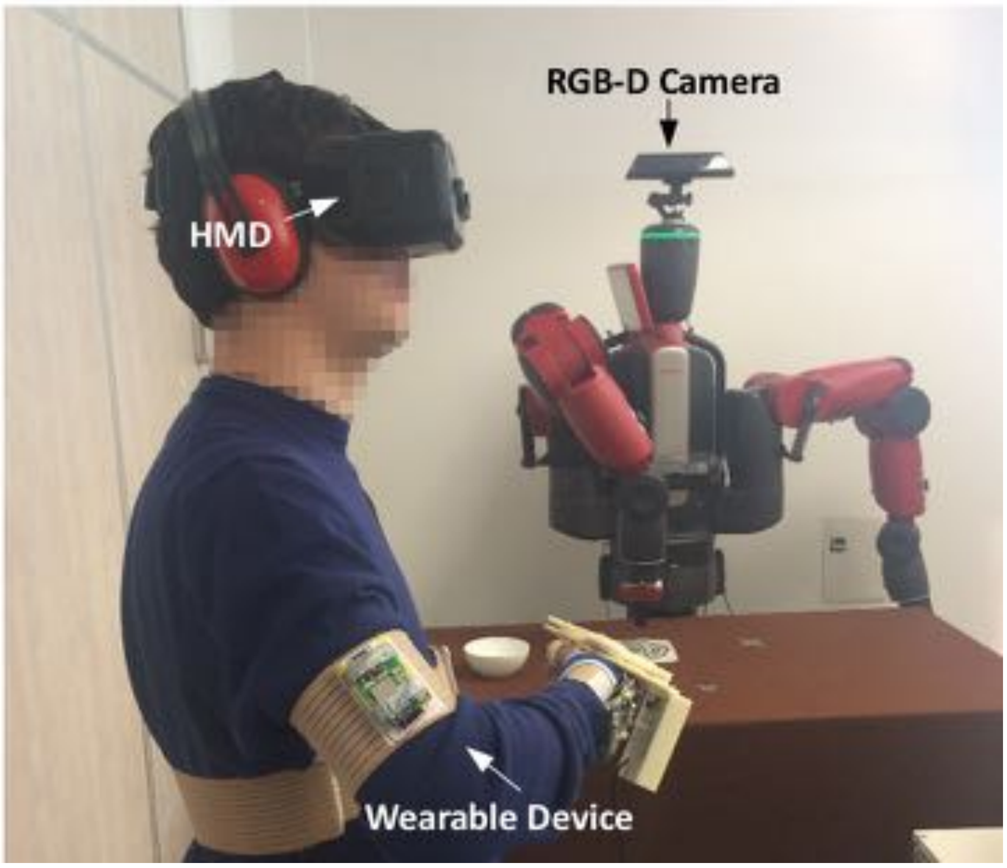


Figure 6.5: The full experimental setup. The participant teleoperate the robot over the Internet and is acoustically isolated from the environment. The robot-side environment is captured with an RGB-D camera and sent to the operator as a 3D visual feedback through an HMD.

The accuracy was expressed in terms of the Euclidean distance between the actual position and the target position; no orientation error was possible in measuring this metric, because the object had radial symmetry.

Before the commencement of each task, the robot arm was moved to a predefined rest position. For computing the completion time of the task, the timer was so configured that it started as soon as the operator started moving the arm to grasp the object, and stopped once the operator released the object and the system tracked its pose.

The trajectories of eight operators' right hands were also recorded at 100Hz, which denotes the sampling frequency of the inertial sensors (MPU9150, Invensense San Jose, California, USA) mounted on the wearable device.

6.2.3 *Data Analysis*

Analysis of the order effect was carried out to ensure that no learning effect occurred, between successive trials. The data relating to completion time and placement accuracy was grouped according to their trial order numbers, and then compared, using one-way ANOVA. To quantify the differences in the placement errors and execution times between the five AR feedback modalities, all the distributions were first checked for normality, using the Lilliefors test. Student's t-test was used to test for the differences in mean, when the resulting data was normally distributed, and Mann–Whitney U-test, when the resulting data was non-normally distributed.

The smoothness of the participant's right hand trajectory in the main motion plane (x-y) was used as an indicator of skillfulness in the task execution [120]. The effects of different AR feedbacks, during the task, on the smoothness of the trajectories were of particular interest.

The smoothness of the trajectory was defined as the normalized jerk ($\hat{J}$), a metric commonly used to determine smoothness:

$$\hat{J} = \sqrt{1/2 \int_t j^2(t) \frac{D^5}{\mathrm{L}^2} dt}. \tag{6.1}$$

where D is the duration and L is the length of the trajectory. Low normalized jerk is indicative of smooth trajectory, and a high jerk of a less-smooth trajectory.

Owing to the complexity of the setup, execution anomalies (e.g., unexpected occlusion, grasp problems) may arise, which can drastically increase the values of the variables of interest. With this in view and considering the possible impact on the statistical analysis, it was decided to eliminate, from each visualization modality, the participants with the maximum and minimum performance scores, resulting in 20 participants per visualization modality.

An additional analysis was performed, to take into account the effects of training and personal experience, which are known to strongly affect the performance of teleoperation tasks. This analysis was done by dividing the participants into two groups according to their self-reported experience in AR/VR and video games. The reported average hours spent weekly by participants, in interacting with AR/VR environment and in playing video games, were used to decide whether to insert them into the expert group (at least two hours weekly) or non-expert group (less than two hours weekly).

Table 6.2: All the measured times (in seconds) and errors (in meters) are divided per participant, for each modality. Table also reports the competency of each participant, plus the tested conditions order.

	(1)		(2)		(3)		(4)		(5)			
ID	No AR		Manipulation AR		Embodiment AR		Task AR		Full AR		Order	Expertise
	Time	Error	Time	Error	Time	Error	Time	Error	Time	Error		
1	56.0	0.0168	40.7	0.0132	32.4	0.0292	40.0	0.0251	37.6	0.0129	12345	None
2	54.5	0.0369	27.4	0.0227	118.8	0.0204	14.7	0.0124	48.1	0.0085	23451	AVR & Games
3	23.6	0.0103	24.8	0.0400	29.9	0.0351	22.4	0.0513	11.7	0.0206	34512	AVR
4	32.9	0.0144	54.5	0.0166	33.0	0.0243	35.1	0.0057	44.5	0.0597	45123	None
5	129.4	0.1041	31.7	0.0294	50.6	0.0250	33.3	0.0416	47.0	0.0232	51234	None
6	10.4	0.0408	13.1	0.0369	15.8	0.0847	32.8	0.0277	55.3	0.0667	54321	Games
7	26.4	0.0218	70.2	0.0074	33.3	0.0548	57.0	0.0903	15.4	0.0190	25134	None
8	125.4	0.0559	91.5	0.0163	43.9	0.0389	28.0	0.0195	28.5	0.0641	13524	AVR
9	81.8	0.0144	63.1	0.0914	39.5	0.0329	67.3	0.0495	32.3	0.0026	43215	None
10	38.1	0.0079	98.0	0.0327	44.8	0.0290	72.0	0.0325	30.9	0.0151	12543	None
11	28.2	0.0263	137.9	0.0293	53.0	0.0248	32.7	0.0285	34.4	0.0136	34251	AVR
12	47.4	0.0671	23.3	0.0180	20.8	0.0141	31.1	0.0253	24.7	0.0100	23154	None
13	25.0	0.0193	36.9	0.0389	31.8	0.0392	33.0	0.0175	19.7	0.0116	32514	Games
14	34.8	0.0694	16.5	0.0049	16.9	0.0146	16.3	0.0143	24.9	0.0285	21534	Games
15	38.5	0.0220	10.4	0.0183	25.9	0.0124	30.2	0.0064	32.8	0.0129	45132	None
16	32.8	0.0287	31.9	0.0200	32.0	0.0125	40.2	0.0256	24.9	0.0155	32154	None
17	37.9	0.0259	20.4	0.0248	33.8	0.0191	46.2	0.0068	25.5	0.0193	31245	None
18	30.9	0.0162	25.3	0.0113	60.4	0.0091	57.2	0.0148	41.1	0.0092	45123	AVR & Games
19	50.4	0.0244	41.6	0.0054	46.9	0.0147	96.1	0.0190	32.0	0.0205	43125	None
20	19.7	0.0236	27.5	0.0115	15.8	0.0194	16.1	0.0279	10.6	0.0051	23154	AVR & Games
21	30.6	0.0872	20.3	0.0134	27.9	0.0504	18.9	0.0239	18.0	0.0257	32154	None
22	20.7	0.0357	18.0	0.0114	27.6	0.0119	24.3	0.0236	27.0	0.0110	51432	AVR & Games

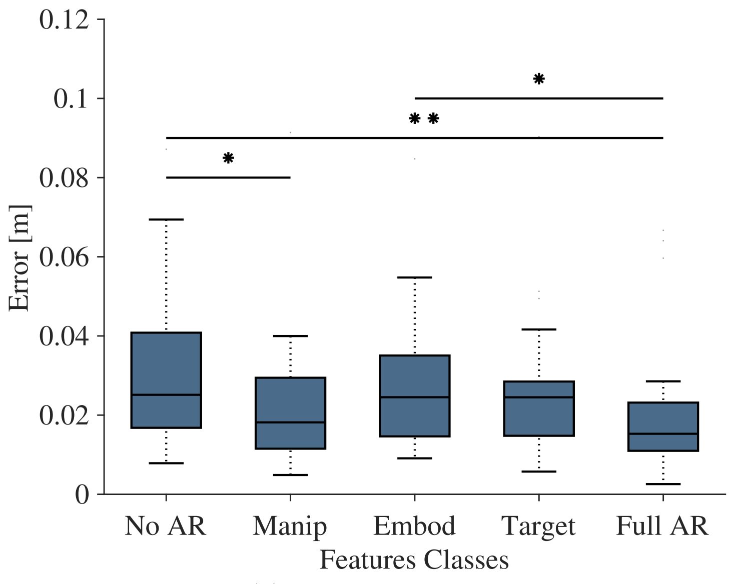
For each condition, the participants with the maximum and minimum scores were removed from the two groups (*expert* and *non-expert*), creating thereby 18 participants for each AR modality.

Each test was considered significant at 95% confidence level ($p \leq 0.05$). Statistical analysis and the trajectories' analysis were performed using algorithms implemented in Matlab R2015b (Mathworks Inc., Natick, Massachusetts, USA).

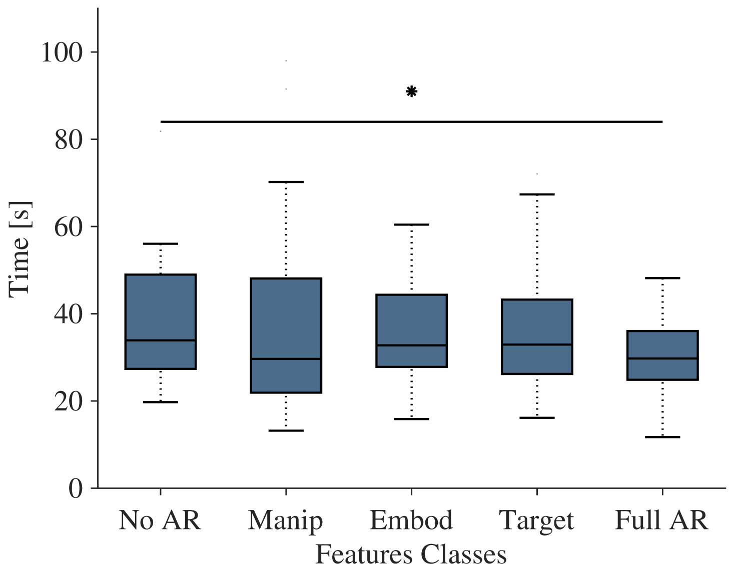
6.3 RESULTS

6.3.1 *Order Effect Analysis*

The results show no statistical difference either in completion time ($p = 0.3187$) or in accuracy ($p = 0.3688$) among different trials. It is, therefore, plausible to assume statistical significance although not all possible



(a) Placement Error



(b) Execution Time

Figure 6.6: Results for the variables, relating to task execution without considering expertise (20 participants per boxplot): (a) Accuracy for different AR feedback modalities, together with significance results. (b) Execution time for different AR feedback modalities, together with the significance results.

combinations of visualization modalities were tested.

6.3.2 *Task Execution*

Significant group differences were found in the parameters relating to task execution (placement accuracy and execution time) among the five modali-

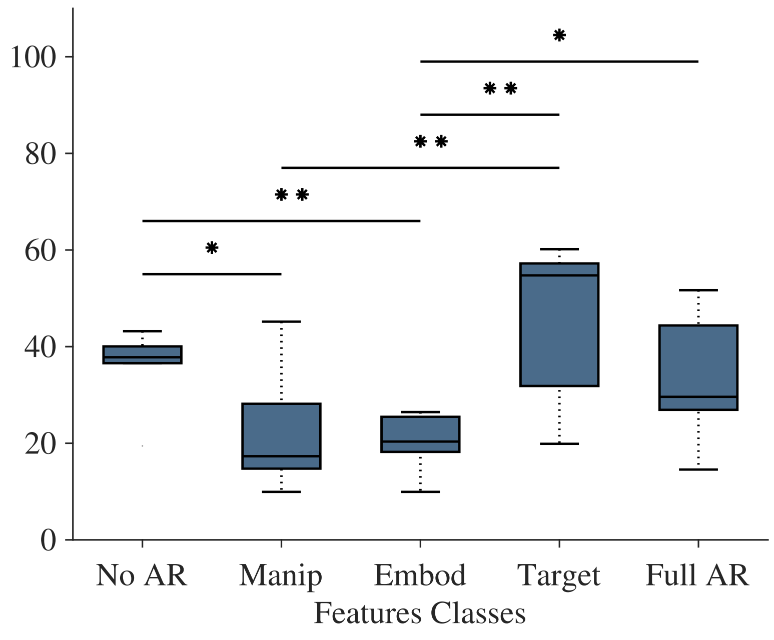
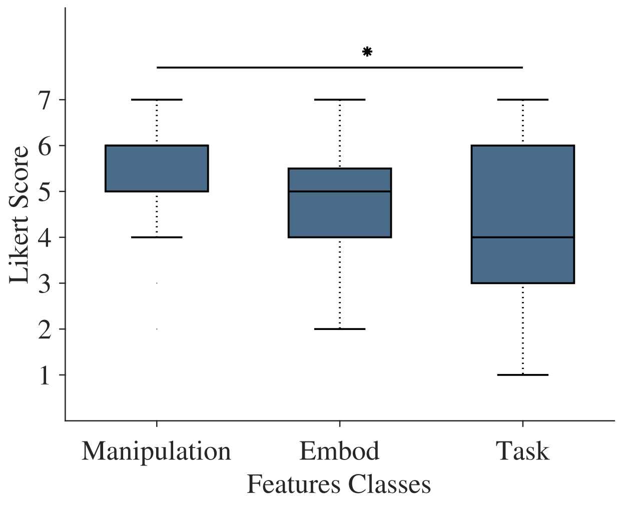


Figure 6.7: Results for the smoothness of the participants' hand trajectory. The values of the normalized jerk for the different AR feedback modalities together with the significance results are shown. Each boxplot depicts the statistics of the 8 participants for which this information is reliably available.

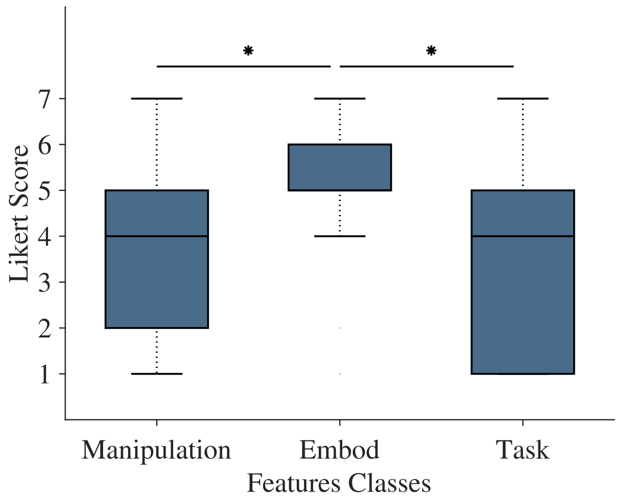
ties of AR feedback, when the analysis is performed on all the 20 participants without accounting expertise. Lilliefors test shows that both Execution Time and Accuracy results are normally distributed. The accuracy obtained during the task execution, with full AR feedback, is significantly higher ($p = 0.0058$) than the one obtained during the task execution with no AR feedback, as also the one obtained with the AR features relating to embodiment ($p = 0.0344$). The accuracy obtained during the task execution with AR features, relating to manipulation, is significantly higher ($p = 0.046$) than the one obtained with no AR feedback. The time required to complete the task with full AR feedback is significantly lower ($p = 0.04$) than the time required with no AR feedback. These statistics, together with the significance results, are shown in Figure 6.6. Figure 6.6 a shows the results for accuracy, and 6.6 b the results for the execution time ($^{*}p \leq 0.05$, $^{**}p < 0.01$).

6.3.3 *Trajectories Analysis*

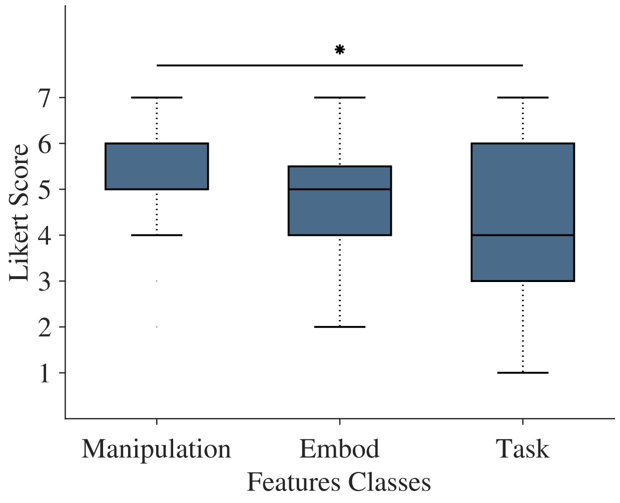
Significant group differences are found in the smoothness of the participant's hand trajectory between the five modalities of AR feedback. Particularly,



(a) Sense of presence

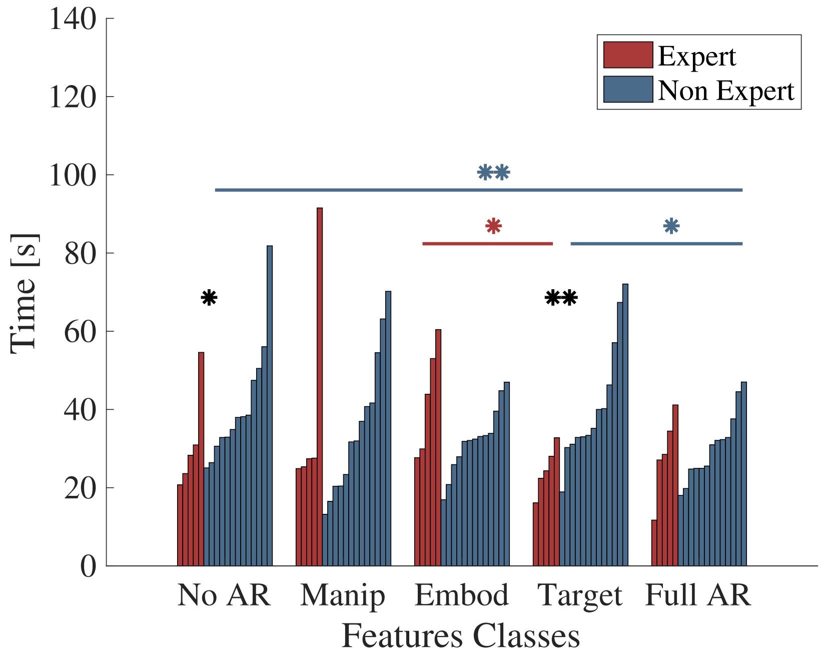


(b) Sense of embodiment

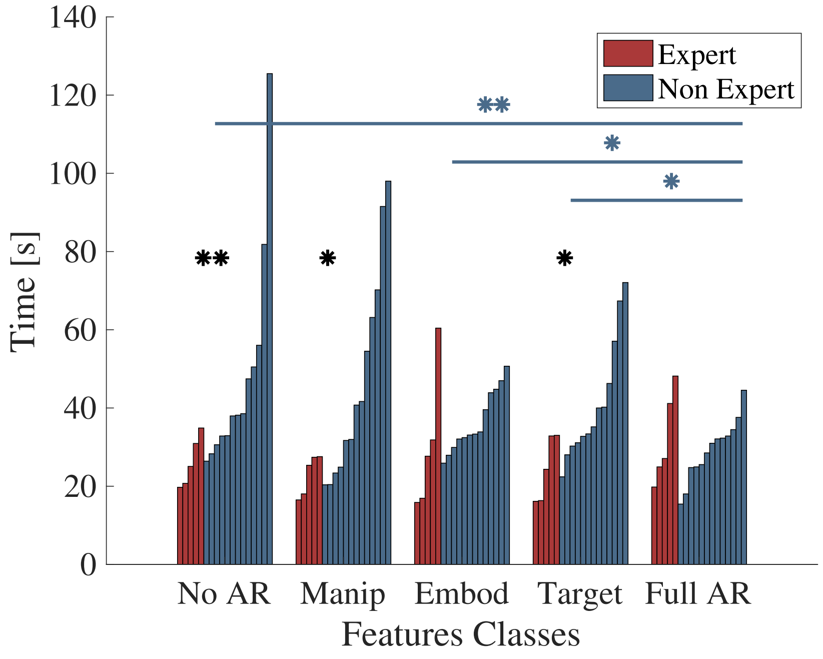


(c) Assistance for the task

Figure 6.8: The reported effects of different AR classes on (a) the sense of presence, (b) embodiment and (c) the ease of executing the task. The values from the questionnaires results (from 1 to 7), together with the significance results, are shown here. Each boxplot depicts the statistics for all the 22 participants.



(a) Expertise in AR/VR



(b) Expertise in Video Games

Figure 6.9: Task execution time of the expert and non-expert groups for (a) AR/VR and (b) video games, based on the participants' partitioning, after removing the outliers from each group. Each bar is associated with a participant, with 18 participants per condition. Significance of the comparison between experts and non-experts for each AR condition is shown with black asterisks. Colored asterisks represent statistical significance of the comparison of different conditions, the expertise group being equal. The bars were sorted, based on execution time, to improve visualization.

the normalized jerk on the x-y plane, obtained during the task execution with AR features relating to embodiment, is significantly lower ($p = 0.002$) than the one on the x-y plane, obtained with no AR feedback, with AR feedback relating to the task information ($p = 0.002$), or with full AR feedback ($p = 0.029$). The normalized jerk on the x-y plane, obtained during task execution with AR features relating to manipulation is significantly lower ($p = 0.027$) than the one on the x-y plane, obtained during the task execution with no AR feedback or with AR feedback, relating to the task information ($p = 0.009$). These statistics, together with the significance results, are shown in Figure 6.7 ($^{*}p \leq 0.05,^{**} p < 0.01$).

6.3.4 *Questionnaires Results*

Significant group differences are found in the reported effects of different AR feedback classes on the participant's sense of acting in the remote environment, of embodiment and on the ease of task execution. Particularly, the AR features relating to manipulation, have a significantly stronger effect ($p = 0.024$) on providing the sense of presence in the remote environment, as compared to that of the task-related AR feedback. The virtual robot model has a significantly stronger effect on enhancing the illusion of the embodiment towards the remote robot, as compared to that of the manipulation-related features ($p = 0.044$) or the task-related features ($p = 0.02$). The manipulation-related features are more effective than the task-related features in facilitating task execution ($p = 0.0138$). These statistics, together with significance results, are shown in Figure 6.8 ($^{*}p \leq 0.05,^{**} p < 0.01$).

6.3.5 *Expertise Analysis*

The partitioning of the 22 participants, based on their expertise, resulted in 7 expert participants and 15 non-expert participants in AR/VR and in 7 expert participants and 15 non-expert participants in Video Games. These two expertise types have 4 participants in common.

The expertise types were analyzed separately and then compared in terms of expert vs non-expert groups. As discussed in Section 6.2.3, the analysis was performed over 18 participants, after removing the outliers from both expert and non-expert sub-groups for every condition.

Figure 6.9 shows the results of the four groups' completion time as a bar chart. Each column is associated with a participant and is grouped, based on visualization modality and expertise. To improve visualization, the bars in the chart are sorted group-wise, without altering the data semantic.

6.3.5.1 *AR/VR*

The expert group completed the task in significantly less time than the non-expert group during the task execution with no AR feedback ($p = 0.05$) or with AR features, relating to the task information ($p = 0.003$). In terms of the execution time, significant differences are found among the AR/VR games non-expert groups with different AR feedback modalities. In particular, the time required to complete the task with full AR feedback is significantly less than the time required with no AR feedback ($p = 0.008$) or with the AR features, relating to the task information ($p = 0.015$).

Significant group differences are found, in terms of the execution time, among the AR/VR expert groups with different AR feedback modalities. In particular, the time required to complete the task with the AR features relating to task information is significantly lesser ($p = 0.027$) than the time required to complete the task with the AR features relating to embodiment.

Figure 6.9 a shows the results, as also their significance, for comparison of the execution time between AR/VR experts and non-experts ($^{*}p \leq 0.05$, $^{**}p < 0.01$).

6.3.5.2 *Video Games*

The expert group completed the task in significantly less time than the time taken by the non-expert group for the task execution with no AR feedback ($p = 0.007$), with AR features relating to manipulation ($p = 0.022$) or with AR features relating to task information ($p = 0.017$).

Significant group differences are found in terms of the execution time, among the video games non-expert groups with different AR feedback modalities. In particular, the time required to complete the task with full AR feedback is significantly less than the time required no AR feedback ($p = 0.003$), with AR features related to embodiment ($p = 0.02$) or with AR features relating to task information ($p = 0.017$). In terms of accuracy, no significant group differences are found between the two groups.

Figure 6.9 b shows the results, as also their significance, for comparison of the execution time between video games experts and non-experts ($^{*}p \leq 0.05$, $^{**}p < 0.01$).

6.4 DISCUSSION

The experiment revealed some differences between the AR feedback modalities, which possibly answer the research questions of this work. The AR effects were quantified in terms of placement error of the manipulated object, task completion time and smoothness of the hand trajectory.

6.4.1 *AR Effects*

Overall, the participants showed, using the full AR feedback, significant improvement in execution, in terms of both placement error and completion time, as compared to their execution without AR feedback. According to the participants' judgment, the robot model is the most effective AR feature in improving the illusion of embodiment. Information like the target object position, the robot end-effector position or the distance between the robot end-effector and the target position can be lost during the execution, due to technological constraints (i.e, low resolution, non-optimality of the camera positioning and view point), and according to the experiment's result, all the features that focus in compensating this loss tend to lead to a more skillful and sure execution. The motivation for this result may depend on the autonomic responses connected to the improved sense of presence and improved sense of embodiment.

Interestingly, the AR features relating to embodiment gave better execution results with smoother trajectories than the full AR feedback (Figure 6.7). This possibly indicates that high degree of additional information added to the scene may lead to an execution that could be more accurate, but more unsure from motion view point. This is probably because the operator could have taken into account too much information in one go, during task execution.

The features delivering information about the task execution seem to have negligible impact on task performance. Moreover, compared to the manipulation-related features, these features contributed less to the sense of presence and to task execution, according to the participants' judgment.

This result is not in line with the finding of previous workers about the use of task-related features, which are usually found to improve the task performance [128]. However, the scenario they considered did not regard the teleoperation of a robot. These results may also be partially influenced by the chosen task, which does not have a particularly complex or articulated execution. The fact that execution with task-related features is comparable to execution without AR features is further strengthened when the results are evaluated considering the previous expertise of the participants. Both the groups, who are considered experts in AR/VR, as well as in video games, performed better than their non-expert counterparts in both no AR and only task related information modalities. The performance of non-experts under these two modalities is worse (in terms of completion time) than their execution performance with full AR feedback. The completion time of non-experts in AR/VR is higher when they used no AR information than when they used the full AR feedback using task-related features compared to the full AR feedback. The completion time of non-experts in video games is higher when they used no AR information than when they used the full AR feedback; it is similarly higher when they used task-related features than when they used the full AR feedback.

6.4.2 *Expertise Effects*

Interestingly, the expert gamers completed the task in lesser time than did the non-expert gamers, even when the AR feedback relating to manipulation information was available. This may be because the video gamers are more proficient at utilizing and relying on synthetic visual cues. This difference is lost with full AR feedback, which can improve the non-expert groups' performance, and thus level out the difference between experts' and non-experts' performances.

Overall, the expert groups were less affected than the non-expert groups by the presence and by the modality of the AR feedback. This finding is in line with that of Gwilliam et al. [58]. The difference between the effects of AR for the expert gamers is not statistically significant, but that in the completion time between the executions with embodiment-related features and task-related features for the experts in AR/VR is statistically significant. This could be because the experts are more proficient at performing tasks in a virtual scenario by virtue of their more skillfulness. It is plausible to assume that the experts already developed skills relating to virtual scenarios, such as judging distances and finding effective strategies of motion. In this regard, AR emerges as a useful tool to assist the first time operators and less skilled operators in executing assembly tasks.

6.5 CONCLUSION

Although the performance of teleoperation tasks with complex visual feedback is strongly affected by the operator's skill and expertise, additional information delivered with AR seems to help in reducing the gap between the performance of expert and non-expert operators. Therefore, AR could help in shortening the learning curve, so that the operators become proficient in the teleoperation setup and can thus perform better with just a short familiarization program with the system. These results may be attributed to the increased sense of presence and embodiment, which benefit from the additional information that can be lost because of technological constraints, but recovered and delivered through AR. The practical implication of these results is that, AR feedback, limited to task specific information, can be useful in supporting expert operators' activity, and full AR feedback in supporting non-expert operators' activity.

Further studies are necessary to fully assess the effects of task-related AR features on task performance. Those studies may require experimental trials with more complex and articulated tasks, which possibly involve multiple execution steps. The outcome of the present work would hopefully facilitate future studies about the effect of AR on the learning curve of teleoperation tasks.

6.6 PUBLICATIONS

Lorenzo Peppoloni, Filippo Brizzi, Carlo Alberto Avizzano, and Emanuele Ruffaldi. "Immersive ros-integrated framework for robot teleoperation". In: *3D User Interfaces (3DUI), 2015 IEEE Symposium on.* IEEE. 2015, pp. 177–178

Lorenzo Peppoloni, Filippo Brizzi, Emanuele Ruffaldi, and Carlo Alberto Avizzano. "Augmented reality-aided tele-presence system for robot manipulation in industrial manufacturing". In: *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology.* ACM. 2015, pp. 237–240

F. Brizzi, L. Peppoloni, A. Graziano, E. D. Stefano, C. A. Avizzano, and E. Ruffaldi. “Effects of Augmented Reality on the Performance of Teleoperated Industrial Assembly Tasks in a Robotic Embodiment”. In: *IEEE Transactions on Human-Machine Systems* PP.99 (2018), pp. 1–10. ISSN: 2168-2291. DOI: `10.1109/THMS.2017.2782490`

7 CONCLUSIONS

The presented work investigated the challenges faced while developing virtual and augmented reality applications for human machine interaction scenarios. The main technological challenges faced when developing AR and VR applications for HRI can be grouped in three areas: components orchestration, reference systems management and choice of the best graphical feedback and of the user interface.

The first purpose of this work was to investigate the requirements of AR/VR applications in telemedicine and general HRI. In particular, we wanted to understand what was already available in the literature, and what, instead, needed to be studied and developed.

An AR application is composed of several components running in parallel and exchanging data. These components ranges from devices' drivers and tracking algorithms, to graphics rendering and physic simulation. More tasks are needed when coupling AR with HRI, like haptic rendering or telecommunication. For most of these tasks it is possible to find a mature state of the art that provides reliable software and algorithms. What we found missing was a tool that reliably and efficiently allowed to connect together and orchestrate all these components (**C1**).

To overcome these limitations we investigated the main existent robotic middleware frameworks, namely ROS and Orocos, to understand which were their strengths and weakness. From this analysis came out that there was deficiency in the available platforms. ROS is easy to use, has fast prototyping capabilities and a huge community that shares code and applications. It allows to create distributed application, and has strong and reliable serialization. On the other side ROS doesn't allow to optimize for performance, as there are no easy way to improve or optimize communications between tasks that always rely on TCP/IP. This becomes a huge problem when dealing with images that can be very big, or with haptic data that has high frame rate. The Orocos framework instead supports hard real-time execution of tasks, but only in special Linux kernels, and it is not as flexible as ROS. AR applications need only soft real-time guarantees, making Orocos an overkilling solution (**C2**).

The middleware we needed was something in between, which is capable to provide the flexibility of ROS and as performance oriented as Orocos. Starting from this consideration we designed a new C++ framework that could fill the missing requirements we needed. The design then led to the implementation of the CoCo library. CoCo provides support for the implementation of independent components that can communicate and run according to several policies. The library is optimize for performance, while providing easy and fast prototyping. It is implemented exploiting the latest features of C++ and is very lightweight, allowing the integration with any third-party software (**C3**). CoCo allows for fast reconfigurability, portability and abstraction as ROS, while it exploits the shared-memory multi-thread approach of Orocos to provide an efficient communication system. It has been tested in several applications, also within the European projects ReMeDi (grant number 610902) and Ramcip (grant number 643433), and the Italian Regional project TAUM. In all these cases CoCo has proven the ability to handle many concurrent components and to interface with third-party software and devices. Given the strong importance of ROS in the robotic community, CoCo has a built in support for ROS that allows it to work with any pre-existing software built for the robotic framework.

The second challenge regards the management of transformations and reference systems. CoCo was not enough to create complex AR applications, in particular when binded to robotic devices, as this introduces the additional complexity of managing reference systems in an unified way. Several software exists to handle transformation graphs, calibration of difference sensors and management of uncertainty in measurements. But, as for the middleware, no library allows to manage all of these problems at the same time. For this reason we have implemented a new library, called SRGViz, that exploits the state of the art for each single problem in an unified approach. The library does not only allow to query and update a transformation graph, but also provides support for calibration and fusion. This greatly simplifies the setup of an AR application and the registration of cameras with robotic devices. The library allows to calibrate multiple sensors on the fly while executing the final application. This saves a lot of time because users don't need to launch multiple programs to have their application running. The library is also very lightweight, allowing it to be embedded in any application. In addition, it provides support for ROS data types and transformation file format, so to be fully compliant and integrable with any existing software (**C4**).

The last challenge to face was to understand how to use the CoCo and SRGViz library to create the best possible AR application that could provide beneficial feedback to the users. The tackled scenarios were telemedicine and teleoperation in industrial scenarios. These two scenarios represent the ones where AR can have the greatest impact in HRI, and are also the most studied. In the case of tele-medicine we had the specific task of creating two user interfaces that could allows doctors to control a remote robots to perform palpation and USG examinations.

The approach chosen for palpation was to create a DiagUI composed of an horizontal screen, displaying the remote scene, and a haptic device placed below it. An encountered type control is used for the haptic device, so to allow doctors to move their hands freely. The hands are tracked by a Leap Motion device and their pose is used to control the haptic device position. AR is used to display a synthetic hand in the video feedback, as doctors cannot see their hands. In addition, given that we didn't have access to a remote robot to perform the palpation, the evaluation of the setup was performed doing haptic rendering on the remote image of the patient coming from a depth camera. CoCo has been used to implement the streaming, the AR, the haptic feedback and the devices drivers. SRGViz helped for the transformation management and the calibration of all the sensors (**C5**).

For the USG examination setup, instead we implemented a VE to test and evaluate the haptic device chosen to perform the teleoperation. The VE was needed as we didn't have access to the remote robot. The VE contained a mesh of a mannequin obtained from 3D scanning, together with a 3D model of the human ribs. The ribs where used to calculate the interaction between the virtual probe and the mannequin. The setup was tested with doctors to evaluate various visualization modalities, as 2D screen, 3D screen and HMD, and also different navigation approaches for the device (**C5**).

The work on the teleoperation application for industrial tasks was carried out to analyze the use of AR to improve task accomplishment, reduce the learning curve and reduce the effort of the operators. The chosen task was the teleoperation of a humanoid robot through a wearable suit that tracks the user's arm movements. The user needed to perform a pick and place operation of a given object, while viewing the remote scene via an HMD displaying the images coming from a Kinect camera mounted on the robot's head. We proposed three types of AR features and we tested them in isolation and all together to evaluate their effectiveness. The features were both task-related and non-task related. Our purpose was also to evaluate how different types of AR features impact the operator's experience and performance. In addition, we wanted to understand the extent to which excessive visual information can be detrimental, and evaluating the non-task specific AR features. The results from the evaluation show that AR effects strongly depend on the operator's skill and expertise. However, AR allows to reduce the gap between the performance of expert and non-expert operators. Therefore, AR could help in shortening the learning curve allowing operators to perform better with just a short familiarization phase with the system. These results may be attributed to the increased sense of presence and embodiment, which benefit from the additional information that can be lost because of technological constraints, but recovered and delivered through AR. The practical implication of these results is that, AR feedback, limited to task specific information, can be useful in supporting expert operators' activity, and full AR feedback in supporting non-expert operators ' activity. Further studies are necessary to fully assess the effects of task-related AR features on task performance. Those studies may require experimental trials with more complex and articulated tasks, which possibly involve multiple execution steps. The outcome of the present work will hopefully facilitate future studies about the effect of AR on the learning curve of teleoperation tasks (**C6**).

The contribution of this thesis represents an attempt to further improve the development of AR application for HRI scenarios, in particular for telemedicine and teleoperation in industrial task. The underlying idea is to relieve the burden of low level complexity from the developer, so that they can focus on the study of the specific components. On top of that, we also started to analyze exemplary scenarios, in order to understand which are the requirements and the guidelines that can be followed when implementing AR for HRI.

Appendix

A LIST OF PUBLICATIONS

CONFERENCE PAPERS

- Lorenzo Peppoloni, Filippo Brizzi, Carlo Alberto Avizzano, and Emanuele Ruffaldi. “Immersive ros-integrated framework for robot teleoperation”. In: *3D User Interfaces (3DUI), 2015 IEEE Symposium on.* IEEE. 2015, pp. 177–178
- Emanuele Ruffaldi, Alessandro Filippeschi, Filippo Brizzi, Juan Manuel Jacinto, and Carlo Alberto Avizzano. “Encountered haptic augmented reality interface for remote examination”. In: *3D User Interfaces (3DUI), 2015 IEEE Symposium on.* IEEE. 2015, pp. 179–180
- Emanuele Ruffaldi, Filippo Brizzi, Alessandro Filippeschi, and Carlo Alerto Avizzano. “Co-located haptic interaction for virtual USG exploration”. In: *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE.* IEEE. 2015, pp. 1548–1551
- Lorenzo Peppoloni, Filippo Brizzi, Emanuele Ruffaldi, and Carlo Alberto Avizzano. “Augmented reality-aided tele-presence system for robot manipulation in industrial manufacturing”. In: *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology.* ACM. 2015, pp. 237–240
- Alessandro Filippeschi, Filippo Brizzi, Emanuele Ruffaldi, Juan Manuel Jacinto, and Carlo Alberto Avizzano. “Encountered-type haptic interface for virtual interaction with real objects based on implicit surface haptic rendering for remote palpation”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE. 2015, pp. 5904–5909
- Emanuele Ruffaldi and Filippo Brizzi. “Coco-a framework for multicore visuo-haptics in mixed reality”. In: *International Conference on Augmented Reality, Virtual Reality and Computer Graphics.* Springer International Publishing. 2016, pp. 339–357

- Emanuele Ruffaldi, Filippo Brizzi, Franco Tecchia, and Sandro Bacinelli. "Third point of view augmented reality for robot intentions visualization". In: *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*. Springer International Publishing. 2016, pp. 471–478

- Emanuele Ruffaldi, Filippo Brizzi, Giacomo Dabisias, and Giorgio Buttazzo. "SOMA: an OpenMP toolchain for multicore partitioning". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM. 2016, pp. 1231–1237

JOURNAL PAPERS

- Emanuele Ruffaldi and Filippo Brizzi. "Probabilistic Spatial Relationship Graph for Mixed-Reality Tele-Operation Systems". In: *IEEE Transactions on Robotics* (2017). **Paper to be submitted**

- F. Brizzi, L. Peppoloni, A. Graziano, E. D. Stefano, C. A. Avizzano, and E. Ruffaldi. "Effects of Augmented Reality on the Performance of Teleoperated Industrial Assembly Tasks in a Robotic Embodiment". In: *IEEE Transactions on Human-Machine Systems* PP.99 (2018), pp. 1–10. ISSN: 2168-2291. DOI: 10.1109/THMS.2017.2782490

- Alessandro Filippeschi, Filippo Brizzi, Emanuele Ruffaldi, Juan Manuel Jiacinto Villegas, and Carlo Alberto Avizzano. "Diagnostician Haptic Interface Study for Tele-Echography Examination Feasibility". In: *IEEE Transactions on Human Machine System* (2017). **Paper resubmitted as new**

BIBLIOGRAPHY

[1] Alireza Abbasi Moshaii and Farshid Najafi. “A review of robotic mechanisms for ultrasound examinations”. In: *Industrial Robot: An International Journal* (2014).

[2] Purang Abolmaesumi et al. “Image-guided control of a robot for medical ultrasound”. In: *TRO* 18.1 (2002), pp. 11–23.

[3] Jérémie Allard, Valérie Gouranton, Loïck Lecointre, Sébastien Limet, Emmanuel Melin, Bruno Raffin, and Sophie Robert. “FlowVR: a middleware for large scale virtual reality applications”. In: *Euro-par 2004 Parallel Processing*. Springer. 2004, pp. 497–505.

[4] Philippe Arbeille, Arnaud Capri, Jean Ayoub, et al. “Use of a robotic arm to perform remote abdominal telesonography”. In: *American Journal of Roentgenology* 188.4 (2007), W317–W322.

[5] Philippe Arbeille et al. “Tele-Operated Echography and Remote Guidance for Performing Tele-Echography on Geographically Isolated Patients”. In: *Journal of Clinical Medicine* 5.6 (2016), p. 58.

[6] Carlo Alberto Avizzano, Alessandro Filippeschi, Juan Manuel Jacinto Villegas, and Emanuele Ruffaldi. “An Optimal Geometric Model for Clavels Delta Robot”. In: *IEEE EMS*. 2015, pp. 232–237.

[7] Ronald T Azuma. “A survey of augmented reality”. In: *Presence: Teleoperators and virtual environments* 6.4 (1997), pp. 355–385.

[8] Timothy D Barfoot and Paul T Furgale. “Associating uncertainty with three-dimensional poses for use in estimation problems”. In: *IEEE Transactions on Robotics* 30.3 (2014), pp. 679–693.

[9] Jorge Barrio et al. “A remote handling rate-position controller for telemanipulating in a large workspace”. In: *Fusion Engineering and Design* 89.1 (2014), pp. 25–28.

[10] Cagatay Basdogan, Alper Kiraz, Ibrahim Bukusoglu, Aydin Varol, and Sultan Doğanay. “Haptic guidance for improved task performance in steering microparticles with optical tweezers”. In: *Optics Express* 15.18 (2007), pp. 11616–11621.

[11] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riss, Christian Sandor, and Martin Wagner. “Design of a component-based augmented reality framework”. In: *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*. IEEE. 2001, pp. 45–54.

[12] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “Surf: Speeded up robust features”. In: *Computer vision–ECCV 2006* (2006), pp. 404–417.

[13] Johannes Behr, Ulrich Bockholt, and Dieter Fellner. “Instantreality—A Framework for Industrial Augmented and Virtual Reality Applications”. In: *Virtual Reality & Augmented Reality in Industry.* Springer, 2011, pp. 91–99.

[14] Antal K Bejczy, Won S Kim, and Steven C Venema. “The phantom robot: predictive displays for teleoperation with time delay”. In: *IEEE ICRA*. 1990, pp. 546–551.

[15] Massimo Bergamasco, Carlo Alberto Avizzano, Antonio Frisoli, Emanuele Ruffaldi, and Simone Marcheschi. “Design and validation of a complete haptic system for manipulative tasks”. In: *Advanced Robotics* 20.3 (2006), pp. 367–389.

[16] Massimo Bergamasco, Antonio Frisoli, and Carlo Avizzano. “Exoskeletons as man-machine interface systems for teleoperation and interaction in virtual environments”. In: *Advances in Telerobotics* (2007), pp. 61–76.

[17] Oliver Bimber and Ramesh Raskar. *Spatial augmented reality: merging real and virtual worlds.* CRC press, 2005.

[18] Jose-Luis Blanco. “A tutorial on se (3) transformation parameterizations and on-manifold optimization”. In: *University of Malaga, Tech. Rep* 3 (2010).

[19] Stanley J Bolanowski Jr, George A Gescheider, Ronald T Verrillo, and Christin M Checkosky. “Four channels mediate the mechanical aspects of touch”. In: *The Journal of the Acoustical society of America* 84.5 (1988), pp. 1680–1694.

[20] F. Brizzi, L. Peppoloni, A. Graziano, E. D. Stefano, C. A. Avizzano, and E. Ruffaldi. “Effects of Augmented Reality on the Performance of Teleoperated Industrial Assembly Tasks in a Robotic Embodiment”. In: *IEEE Transactions on Human-Machine Systems* PP.99 (2018), pp. 1–10. ISSN: 2168-2291. DOI: 10.1109/THMS.2017.2782490.

[21] Frederick Brooks Jr, U Marcus Brown, Tuscaloosa Chris Burbeck, Nat Durlach, MIT Steve Ellis, James Lackner, Warren Robinett, Mandayam Srinivasan, MIT Ivan Sutherland, Dick Urban, et al. “Research directions in virtual environments”. In: *Computer Graphics* 26.3 (1992), p. 153.

[22] Herman Bruyninckx. “Open robot control software: the OROCOS project”. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on.* Vol. 3. IEEE. 2001, pp. 2523–2528.

[23] Dave Burns and Robert Osfield. “Tutorial: open scene graph A: introduction tutorial: open scene graph B: examples and applications”. In: *Virtual Reality, 2004. Proceedings. IEEE.* IEEE. 2004, pp. 265–265.

[24] Cristina Cañero, Nikolaos Thomos, George A Triantafyllidis, George C Litos, and Michael Gerassimos Strintzis. “Mobile tele-echography: user interface design”. In: *IEEE Trans Inf Technol Biomed* 9.1 (2005), pp. 44–49.

[25] Marcello Carrozzino, Franco Tecchia, Sandro Bacinelli, Carlo Cappelletti, and Massimo Bergamasco. “Lowering the development time of multimodal interactive application: the real-life experience of the XVR project”. In: *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology.* ACM. 2005, pp. 270–273.

[26] Thomas P Caudell and David W Mizell. “Augmented reality: An application of heads-up display technology to manual manufacturing processes”. In: *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on.* Vol. 2. IEEE. 1992, pp. 659–669.

[27] Sokho Chang, Jungtae Kim, Insup Kim, et al. “KIST teleoperation system for humanoid robot”. In: *IEEE IROS.* Vol. 2. 1999.

[28] Enylton Machado Coelho, SJ Julier, and B Maclntyre. “OSGAR: A scene graph with uncertain transformations”. In: *Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on.* IEEE. 2004, pp. 6–15.

[29] Timothy R Coles, Nigel W John, Derek Gould, and Darwin G Caldwell. “Integrating haptics with augmented reality in a femoral palpation and needle insertion training simulation”. In: *IEEE Transactions on Haptics* 4.3 (2011), pp. 199–209.

[30] Francois Conti and Oussama Khatib. “Spanning large workspaces using small haptic devices”. In: *World Haptics 2005.* IEEE. 2005, pp. 183–188.

[31] Francois Conti, D Morris, F Barbagli, and C Sewell. “CHAI 3D”. In: *Online: http://www. chai3d. org* (2006).

[32] Kendall Correll, Nick Barendt, and Michael Branicky. “Design considerations for software only implementations of the IEEE 1588 precision time protocol”. In: *Conference on IEEE.* Vol. 1588. 2005, pp. 11–15.

[33] Carolina Cruz-Neira, Daniel J Sandin, Thomas A DeFanti, Robert V Kenyon, and John C Hart. “The CAVE: audio visual experience automatic virtual environment”. In: *Communications of the ACM* 35.6 (1992), pp. 64–73.

[34] Konstantinos Daniilidis. “Hand-eye calibration using dual quaternions”. In: *The International Journal of Robotics Research* 18.3 (1999), pp. 286–298.

[35] Cécile Delgorge, Courrèges, et al. “A tele-operated mobile ultrasound scanner using a light-weight robot”. In: *Information Technology in Biomedicine, IEEE Transactions on* 9.1 (2005), pp. 50–58.

[36] Sergio Portolés Diez, Emmanuel B Vander Poorten, Gianni Borghesan, and Dominiek Reynaerts. “Towards palpation in virtual reality by an encountered-type haptic screen”. In: *International Conference on Human Haptic Sensing and Touch Enabled Computer Applications*. Springer. 2014, pp. 257–265.

[37] Lionel Dominjon et al. “The” Bubble” technique: interacting with large virtual environments using haptic devices with limited workspace”. In: *World Haptics*. IEEE. 2005.

[38] Lionel Dominjon et al. “A comparison of three techniques to interact in large virtual environments using haptic devices with limited workspace”. In: *Advances in Computer Graphics*. Springer, 2006, pp. 288–299.

[39] U. Eck and C. Sandor. “HARP: A framework for visuo-haptic augmented reality”. In: *Virtual Reality (VR), 2013 IEEE*. 2013, pp. 145–146. DOI: 10.1109/VR.2013.6549404.

[40] Ulrich Eck, Frieder Pankratz, Christian Sandor, Gudrun Klinker, and Hamid Laga. “Comprehensive workspace calibration for visuo-haptic augmented reality”. In: *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*. IEEE. 2014, pp. 123–128.

[41] Erik Einhorn, Tim Langner, Ronny Stricker, Christian Martin, and Horst-Michael Gross. “Mira-middleware for robotic applications”. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 2591–2598.

[42] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. “Graphviz—open source graph drawing tools”. In: *International Symposium on Graph Drawing*. Springer. 2001, pp. 483–484.

[43] Takahiro Endo, Haruhisa Kawasaki, et al. “Five-fingered haptic interface robot: HIRO III”. In: *IEEE Worldhaptics*. 2009, pp. 458–463.

[44] Farzam Farbiz, Zhou Hao Yu, Corey Manders, and Waqas Ahmad. “An electrical muscle stimulation haptic feedback for mixed reality tennis game”. In: *ACM SIGGRAPH 2007 posters*. ACM. 2007, p. 140.

[45] Steven Feiner, Blair Macintyre, and Dorée Seligmann. “Knowledge-based augmented reality”. In: *Communications of the ACM* 36.7 (1993), pp. 53–62.

[46] Charith Lasantha Fernando, Masahiro Furukawa, et al. "Design of TELESAR V for transferring bodily consciousness in telexistence". In: *IEEE IROS*. 2012.

[47] P Figueroa, W Bischof, P Boulanger, H Hoover, and R Taylor. "InTml: A Dataflow Oriented Development System for Virtual Reality Applications". In: *Presence* 17.5 (2008), pp. 492–511. ISSN: 1054-7460. DOI: `10.1162/pres.17.5.492`.

[48] Alessandro Filippeschi, Filippo Brizzi, Emanuele Ruffaldi, Juan Manuel Jacinto, and Carlo Alberto Avizzano. "Encountered-type haptic interface for virtual interaction with real objects based on implicit surface haptic rendering for remote palpation". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 5904–5909.

[49] Alessandro Filippeschi, Filippo Brizzi, Emanuele Ruffaldi, Juan Manuel Jiacinto Villegas, and Carlo Alberto Avizzano. "Diagnostician Haptic Interface Study for Tele-Echography Examination Feasibility". In: *IEEE Transactions on Human Machine System* (2017). **Paper resubmitted as new**.

[50] Andrew Fischer and Judy M Vance. "Phantom haptic device implemented in a projection screen virtual environment". In: *Workshop on Virtual environments*. ACM. 2003.

[51] Tully Foote. "tf: The transform library". In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. 2013, pp. 1–6. DOI: `10.1109/TePRA.2013.6556373`.

[52] Lars Fritsche, Felix Unverzag, Jan Peters, and Roberto Calandra. "First-person tele-operation of a humanoid robot". In: *IEEE Humanoid*. 2015, pp. 997–1002.

[53] Henry Fuchs, Mark A Livingston, Ramesh Raskar, Kurtis Keller, Jessica R Crawford, Paul Rademacher, Samuel H Drake, Anthony A Meyer, et al. "Augmented reality visualization for laparoscopic surgery". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 1998, pp. 934–943.

[54] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292.

[55] Matthew W Gilbertson and Brian W Anthony. "An ergonomic, instrumented ultrasound probe for 6-axis force/torque measurement". In: *EMBC*. IEEE.

[56] Ken Goldberg, Michael Mascha, Steve Gentner, Nick Rothenberg, Carl Sutter, and Jeff Wiegley. "Desktop teleoperation via the world wide web". In: *IEEE ICRA*. Vol. 1. 1995.

[57] Alessandro Graziano, Paolo Tripicchio, Emanuele Ruffaldi, and Carlo Alberto Avizzano. "A haptic datasuit for controlling humanoid robots". In: *ISR, 47th International Symposium on Robotics*. 2016.

[58] James C Gwilliam, Mohsen Mahvash, Balazs Vagvolgyi, et al. "Effects of haptic and graphical force feedback on teleoperated palpation". In: *IEEE ICRA*. 2009.

[59] Matthias Harders, Gérald Bianchi, Benjamin Knoerlein, and Gábor Székely. "Calibration, registration, and synchronization for high precision augmented reality haptics". In: *IEEE Transactions on Visualization and Computer Graphics* 15.1 (2009), pp. 138–149.

[60] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz Schröder. "Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds". In: *Information Fusion* 14.1 (2013), pp. 57–77.

[61] Kenji Inoue, Kentaro Ujiie, and Suwoong Lee. "Development of haptic devices using flexible sheets for virtual training of abdominal palpation". In: *Advanced Robotics* 28.20 (2014), pp. 1331–1341.

[62] Juan Manuel Jacinto-Villegas, Massimo Satler, Alessandro Filippeschi, Massimo Bergamasco, Matteo Ragaglia, Alfredo Argiolas, Marta Niccolini, and Carlo Alberto Avizzano. "A Novel Wearable Haptic Controller for Teleoperating Robotic Platforms". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2072–2079.

[63] Caroline Jay, Mashhuda Glencross, and Roger Hubbold. "Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment". In: *TOCHI* (2007).

[64] Caroline Jay and Roger Hubbold. "Does performance in the face of delayed sensory feedback change with practice". In: *Proceedings of the Eurohaptics conference, Springer, New York*. 2004, pp. 530–533.

[65] Edwin G Johnsen and William R Corliss. *Human factors applications in teleoperator design and operation*. Wiley-Interscience New York, 1971.

[66] Matt Jones and Gary Marsden. "Mobile interaction design". In: (2006).

[67] Hirokazu Kato. "ARToolKit". In: *http://www. hitl. washington. edu/artoolkit/* (1999).

[68] Peter Keitler, Daniel Pustka, Manuel Huber, Florian Echtler, and Gudrun Klinker. “Management of tracking for mixed and augmented reality systems”. In: *The Engineering of Mixed Reality Systems*. Springer, 2010, pp. 251–273.

[69] Christian Kerl, Jurgen Sturm, and Daniel Cremers. “Dense visual SLAM for RGB-D cameras”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 2100–2106.

[70] Masaya Kitagawa, Daniell Dokko, Allison M Okamura, and David D Yuh. “Effect of sensory substitution on suture-manipulation forces for robotic surgical systems”. In: *The Journal of thoracic and cardiovascular surgery* (2005).

[71] Alexandre Krupa et al. “Robotized Tele-Echography: an Assisting Visibility Tool to Support Expert Diagnostic”. In: *IEEE Systems Journal* (2014), pp. 1–10.

[72] Yuichi Kurita, Takaaki Ishikawa, and Toshio Tsuji. “Stiffness display by muscle contraction via electric muscle stimulation”. In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 1014–1019.

[73] Sally A Linkenauger, Heinrich H Bülthoff, and Betty J Mohler. “Virtual armś reach influences perceived distances but only after experience reaching”. In: *Neuropsychologia* (2015).

[74] Mark A Livingston et al. “Basic perception in head-worn augmented reality displays”. In: *Human factors in augmented reality environments*. Springer, 2013.

[75] Asa MacWilliams, Thomas Reicher, Gudrun Klinker, and Bernd Bruegge. “Design Patterns for Augmented Reality Systems.” In: *MIXER*. 2004.

[76] John W Mauchly. “Significance test for sphericity of a normal n-variate distribution”. In: *The Annals of Mathematical Statistics* 11.2 (1940), pp. 204–209.

[77] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. “YARP: yet another robot platform”. In: *International Journal of Advanced Robotic Systems* 3.1 (2006), p. 8.

[78] Paul Milgram and Fumio Kishino. “A taxonomy of mixed reality visual displays”. In: *IEICE TRANSACTIONS on Information and Systems* 77.12 (1994), pp. 1321–1329.

[79] Nathan Miller, Odest Chadwicke Jenkins, et al. “Motion capture from inertial sensing for untethered humanoid teleoperation”. In: *IEEE Humanoid Robots*. Vol. 2. 2004.

[80] Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. “Middleware for robotics: A survey”. In: *Robotics, Automation and Mechatronics, 2008 IEEE Conference on.* Ieee. 2008, pp. 736–742.

[81] Giovanni Mosiello, Andrey Kiselev, and Amy Loutfi. “Using Augmented Reality to Improve Usability of the User Interface for Driving a Telepresence Robot”. In: *Paladyn, Journal of Behavioral Robotics* 4.3 (2013), pp. 174–181.

[82] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. “ORB-SLAM: a versatile and accurate monocular SLAM system”. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163.

[83] Farshid Najafi and Nariman Sepehri. “Design and prototyping of a force-reflecting hand-controller for ultrasound imaging”. In: *Journal of Mechanisms and Robotics* (2011).

[84] Joseph Newman, Martin Wagner, Martin Bauer, Asa MacWilliams, Thomas Pintaric, Dagmar Beyer, Daniel Pustka, Franz Strasser, Dieter Schmalstieg, and Gudrun Klinker. “Ubiquitous tracking for augmented reality”. In: *Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on.* IEEE. 2004, pp. 192–201.

[85] Adrian KT Ng, Leith KY Chan, and Henry YK Lau. “Depth perception in virtual environment: The effects of immersive system and freedom of movement”. In: *International Conference on Virtual, Augmented and Mixed Reality.* Springer. 2016, pp. 173–183.

[86] Verena Nitsch and Berthold Farber. “A meta-analysis of the effects of haptic interfaces on task performance with teleoperation systems”. In: *Haptics, IEEE Transactions on* 6 (2013).

[87] Laurence Nouaille, Natalie Smith-Guérin, Gérard Poisson, and Philippe Arbeille. “Optimization of a 4 dof tele-echography robot”. In: *IROS.* IEEE. 2010, pp. 3501–3506.

[88] Lina M Paz, Pedro Piniés, Juan D Tardós, and José Neira. “Large-scale 6-DOF SLAM with stereo-in-hand”. In: *IEEE transactions on robotics* 24.5 (2008), pp. 946–957.

[89] Angelika Peer and et al. Buss. “Towards a remote medical diagnostician for medical examination”. In: *NextMed MMVR21.* 2014.

[90] Lorenzo Peppoloni, Filippo Brizzi, Carlo Alberto Avizzano, and Emanuele Ruffaldi. “Immersive ros-integrated framework for robot teleoperation”. In: *3D User Interfaces (3DUI), 2015 IEEE Symposium on.* IEEE. 2015, pp. 177–178.

[91] Lorenzo Peppoloni, Filippo Brizzi, Emanuele Ruffaldi, and Carlo Alberto Avizzano. “Augmented reality-aided tele-presence system for robot manipulation in industrial manufacturing”. In: *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology.* ACM. 2015, pp. 237–240.

[92] Lorenzo Peppoloni, Alessandro Filippeschi, Emanuele Ruffaldi, and Carlo Alberto Avizzano. “A novel 7 degrees of freedom model for upper limb kinematic reconstruction based on wearable sensors”. In: *IEEE IROS.* 2013.

[93] Max Pfeiffer, Stefan Schneegaß, and Florian Alt. “Supporting interaction in public space with electrical muscle stimulation”. In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication.* ACM. 2013, pp. 5–8.

[94] François Pierrot, Etienne Dombre, Eric Dégoulange, Loïc Urbain, Pierre Caron, Sylvie Boudet, Jérome Gariépy, and Jean-Louis Mégnien. “Hippocrate: a safe robot arm for medical applications with force feedback”. In: *Medical Image Analysis* (1999).

[95] François Pierrot, C Reynaud, and Alain Fournier. “DELTA: a simple and efficient parallel robot”. In: *Robotica* 8.2 (1990), pp. 105–109.

[96] Daniel Pustka, Martin Huber, Manuel Bauer, and Gudrun Klinker. “Spatial relationship patterns: Elements of reusable tracking and calibration systems”. In: *Mixed and Augmented Reality, 2006. ISMAR 2006. IEEE/ACM International Symposium on.* IEEE. 2006, pp. 88–97.

[97] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software.* Vol. 3. 3.2. 2009, p. 5.

[98] Ramesh Raskar, Greg Welch, Kok-Lim Low, and Deepak Bandyopadhyay. “Shader lamps: Animating real objects with image-based illumination”. In: *Rendering Techniques 2001.* Springer, 2001, pp. 89–102.

[99] Rebekka S Renner, Boris M Velichkovsky, and Jens R Helmert. “The perception of egocentric distances in virtual environments-a review”. In: *ACM Computing Surveys* 46 (2013).

[100] Louis B Rosenberg. “Virtual fixtures: Perceptual tools for telerobotic manipulation”. In: *IEEE VR.* 1993, pp. 76–82.

[101] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. “ORB: An efficient alternative to SIFT or SURF”. In: *Computer Vision (ICCV), 2011 IEEE international conference on.* IEEE. 2011, pp. 2564–2571.

[102] E. Ruffaldi, D. Morris, F. Barbagli, K. Salisbury, and M. Bergamasco. “Voxel-Based Haptic Rendering Using Implicit Sphere Trees”. In: *Haptic interfaces for virtual environment and teleoperator systems, 2008. haptics 2008. symposium on.* 2008, pp. 319–325. DOI: `10.1109/HAPTICS.2008.4479964`.

[103] Emanuele Ruffaldi and Filippo Brizzi. “Coco-a framework for multicore visuo-haptics in mixed reality”. In: *International Conference on Augmented Reality, Virtual Reality and Computer Graphics.* Springer International Publishing. 2016, pp. 339–357.

[104] Emanuele Ruffaldi and Filippo Brizzi. “Probabilistic Spatial Relationship Graph for Mixed-Reality Tele-Operation Systems”. In: *IEEE Transactions on Robotics* (2017). **Paper to be submitted**.

[105] Emanuele Ruffaldi, Filippo Brizzi, Giacomo Dabisias, and Giorgio Buttazzo. “SOMA: an OpenMP toolchain for multicore partitioning”. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing.* ACM. 2016, pp. 1231–1237.

[106] Emanuele Ruffaldi, Filippo Brizzi, Alessandro Filippeschi, and Carlo Alerto Avizzano. “Co-located haptic interaction for virtual USG exploration”. In: *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE.* IEEE. 2015, pp. 1548–1551.

[107] Emanuele Ruffaldi, Filippo Brizzi, Franco Tecchia, and Sandro Bacinelli. “Third point of view augmented reality for robot intentions visualization”. In: *International Conference on Augmented Reality, Virtual Reality and Computer Graphics.* Springer International Publishing. 2016, pp. 471–478.

[108] Emanuele Ruffaldi, Alessandro Filippeschi, Filippo Brizzi, Juan Manuel Jacinto, and Carlo Alberto Avizzano. “Encountered haptic augmented reality interface for remote examination”. In: *3D User Interfaces (3DUI), 2015 IEEE Symposium on.* IEEE. 2015, pp. 179–180.

[109] Diego C Ruspini, Krasimir Kolarov, and Oussama Khatib. “The haptic display of complex graphical environments”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques.* ACM Press/Addison-Wesley Publishing Co. 1997, pp. 345–352.

[110] SE Salcudean and NR Parker. “6-dof desk-top voice-coil joystick”. In: *International Mechanical Engineering Congress and Exposition.* 1997.

[111] SE Salcudean et al. “Robot-assisted diagnostic ultrasound–design and feasibility experiments”. In: *Int Conf on Medical Image Computing.* Springer. 1999.

[112] Kenneth Salisbury, Francois Conti, and Federico Barbagli. "Haptic rendering: introductory concepts". In: *Computer Graphics and Applications, IEEE* 24.2 (2004), pp. 24–32.

[113] Kenneth Salisbury and Christopher Tarr. "Haptic rendering of surfaces defined by implicit functions". In: *ASME Dynamic Systems and Control Division*. Vol. 61. 1997, pp. 61–67.

[114] Beatriz Sousa Santos et al. "Head-mounted display versus desktop for 3D navigation in virtual reality: a user study". In: *Multimedia Tools and Applications* (2009).

[115] Craig P Sayers, Richard P Paul, Louis L Whitcomb, and Dana R Yoerger. "Teleprogramming for subsea teleoperation using acoustic communication". In: *IEEE Journal of Oceanic Engineering* 23.1 (1998), pp. 60–71.

[116] Richard M Scheffler, Jenny X Liu, Yohannes Kinfu, and Mario R Dal Poz. "Forecasting the global shortage of physicians: an economic-and needs-based approach". In: *Bulletin of the World Health Organization* 86.7 (2008), 516–523B.

[117] Stefano Scheggi, Francesco Chinello, and Domenico Prattichizzo. "Vibrotactile haptic feedback for human-robot interaction in leader-follower tasks". In: *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*. ACM. 2012, p. 51.

[118] JM Selig. *Geometrical methods in robotics*. Springer-Verlag New York, Inc., 1996.

[119] Samuel Sanford Shapiro and Martin B Wilk. "An analysis of variance test for normality (complete samples)". In: *Biometrika* 52.3/4 (1965), pp. 591–611.

[120] Lior Shmuelof, John W Krakauer, and Pietro Mazzoni. "How is a motor skill learned? Change and invariance at the levels of task success and trajectory control". In: *Journal of neurophysiology* 108.2 (2012), pp. 578–594.

[121] Ken Shoemake. "Animating rotation with quaternion curves". In: *ACM SIGGRAPH computer graphics*. Vol. 19. 3. ACM. 1985, pp. 245–254.

[122] Tobias Sielhorst, Tobias Obst, Rainer Burgkart, Robert Riener, and Nassir Navab. "An augmented reality delivery simulator for medical training". In: *International Workshop on Augmented Environments for Medical Imaging-MICCAI Satellite Workshop*. Vol. 141. 2004.

[123] Deok-Jae Song et al. "Z-Clutching: Interaction Technique for Navigating 3D Virtual Environment Using a Generic Haptic Device". In: *JCSE* 10.1 (2016), pp. 32–38.

[124] Weibo Song, Xianjiu Guo, et al. “Teleoperation humanoid robot control system based on kinect sensor”. In: *IEEE Intelligent Human-Machine Systems and Cybernetics*. Vol. 2. 2012.

[125] Tiago Boldt Sousa. “Dataflow programming concept, languages and applications”. In: *Doctoral Symposium on Informatics Engineering*. Vol. 130. 2012.

[126] John A Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*. Vol. 460. Springer Science & Business Media, 2012.

[127] Ali Talasaz and Rajnikant V Patel. “Remote palpation to localize tumors in robot-assisted minimally invasive approach”. In: *IEEE ICRA*. 2012, pp. 3719–3724.

[128] Arthur Tang, Charles Owen, Frank Biocca, and Weimin Mou. “Comparative effectiveness of augmented reality in object assembly”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2003, pp. 73–80.

[129] Roger Y Tsai and Reimar K Lenz. “A new technique for fully autonomous and efficient 3D robotics hand/eye calibration”. In: *IEEE Transactions on robotics and automation* 5.3 (1989), pp. 345–358.

[130] Stefan Tuchschmid, Markus Grassi, D Bachofen, P Früh, Markus Thaler, Gábor Székely, and Matthias Harders. “A flexible framework for highly-modular surgical simulation systems”. In: *International Symposium on Biomedical Simulation*. Springer. 2006, pp. 84–92.

[131] Sebastian Ullrich and Torsten Kuhlen. “Haptic palpation for medical simulation in virtual environments”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.4 (2012), pp. 617–625.

[132] Pietro Valdastri, Massimiliano Simi, and Robert J Webster III. “Advanced technologies for gastrointestinal scopy”. In: *Annual review of biomedical engineering* 14 (2012), pp. 397–429.

[133] Pierre Vieyres, Gérard Poisson, Fabien Courrèges, Olivier Mérigeaux, and Philippe Arbeille. “The TERESA project: from space research to ground tele-echography”. In: *Industrial robot: an international journal* 30.1 (2003), pp. 77–82.

[134] Pierre Vieyres et al. “A tele-operated robotic system for mobile tele-echography: The OTELO project”. In: *M-Health*. Springer, 2006, pp. 461–473.

[135] Adriana Vilchis, Jocelyne Troccaz, Philippe Cinquin, Kohji Masuda, and Franck Pellissier. “A new robot architecture for tele-echography”. In: *IEEE TRO* (2003).

[136] Juan Manuel Jacinto Villegas, Carlo Alberto Avizzano, Emanuele Ruffaldi, and Massimo Bergamasco. “A Low Cost Open-Controller for Interactive Robotic System”. In: *IEEE EMS*. 2015, pp. 462–468.

[137] Yunfeng Wang and Gregory S Chirikjian. “Nonparametric second-order theory of error propagation on motion groups”. In: *The International journal of robotics research* 27.11-12 (2008), pp. 1258–1273.

[138] INGRID Wigerstad-Lossing, Gunnar Grimby, TORSTEN Jonsson, Bengt Morelli, LARS Peterson, and Per Renström. “Effects of electrical muscle stimulation combined with voluntary contractions after knee ligament surgery.” In: *Medicine and science in sports and exercise* 20.1 (1988), pp. 93–98.

[139] Robert L Williams II, Mayank Srivastava, et al. “The virtual haptic back for palpatory training”. In: *Prof of the 6th int. conference on Multimodal interfaces*. ACM. 2004, pp. 191–197.

[140] World Health Organization. *Telemedicine: opportunities and developments in Member States: report on the second global survey on eHealth*. 2010.

[141] Tian Xia, Simon Léonard, et al. “Augmented reality environment with virtual fixtures for robotic telemanipulation in space”. In: *IEEE IROS*. 2012.

[142] Shamima Yasmin and Alexei Sourin. “Virtual palpation for medical training in cyberworlds”. In: *Cyberworlds (CW), 2012 International Conference on*. IEEE. 2012, pp. 207–214.

[143] Yongqiang Ye and Peter X Liu. “Improving haptic feedback fidelity in wave-variable-based teleoperation orientated to telemedical applications”. In: *IEEE Transactions on Instrumentation and Measurement* 58.8 (2009), pp. 2847–2855.

[144] Pavel Zahorik and Rick L Jenison. “Presence as being-in-the-world”. In: *Presence* 7.1 (1998), pp. 78–89.

INSTITUTE
OF COMMUNICATION
INFORMATION
AND PERCEPTION
TECHNOLOGIES

Sant'Anna
School of Advanced Studies – Pisa