Development of an open access system for remote operation of robotic manipulators

Bruno Matheu Moreira Stefanuto - 52478

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Engenharia Industrial, in the scope of the double diploma programme with the Federal University of Technology - Paraná.

Work oriented by:

Prof. Paulo Leitão

Prof. Marcos Banheti Rabello Vallim

Bragança

May, 2023

Development of an open access system for remote operation of robotic manipulators

Bruno Matheu Moreira Stefanuto - 52478

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Engenharia Industrial, in the scope of the double diploma programme with the Federal University of Technology - Paraná.

Work oriented by:

Prof. Paulo Leitão

Prof. Marcos Banheti Rabello Vallim

Bragança

May, 2023

Dedication

To the last person reading this dissertation, thank you for keeping this research alive until this point ...

Acknowledgement

I would like to begin by expressing my gratitude to my family, as none of this would have been possible without them. Thank you to my mother, Jane, for inspiring and motivating me every day to move forward, for believing in education, and for teaching me to believe as well. Thank you to my father, João Carlos, for his example and daily effort, as much of it is reflected in this work. Thank you to my sister, Ludymila, for being my lifelong companion and for always being there for me. Thank you, Meg, for bringing joy to my days. It was only possible because I have all of you by my side.

I would also like to thank all my friends, from those I have known for a long time since my school days to those I recently made when I arrived at IPB, who have supported me equally in achieving another milestone. Thank you to each and every one of you; it is evident to me the difference your friendships have made in my life.

A heartfelt thank you to my supervisors for the time dedicated to me and for all the knowledge imparted. Rest assured that I have tried to make the most of every opportunity, and your guidance has had a profound impact on both my academic and personal life.

Lastly, I express my gratitude to UTFPR and IPB for the honor of studying at two magnificent institutions. Through their professors, staff, and colleagues, I have received valuable lessons in engineering and life. My sincerest thanks to all!

Abstract

Exploring the realms of research, training, and learning in the field of robotic systems poses obstacles for institutions lacking the necessary infrastructure. The significant investment required to acquire physical robotic systems often limits access and hinders progress in these areas. While robotic simulation platforms provide a virtual environment for experimentation, the potential of remote robotic environments surpasses this by enabling users to interact with real robotic systems during training and research activities. This way, users, including students and researchers, can engage in a virtual experience that transcends geographical boundaries, connecting them to real-world robotic systems though the Internet. By bridging the gap between virtual and physical worlds, remote environments offer a more practical and immersive experience, and open up new horizons for collaborative research and training. Democratizing access to these technologies means empower educational institutions and research centers to engage in practical and hands-on learning experiences. However, the implementation of remote robotic environments comes with its own set of technical challenges: communication, security, stability and access. In light of these challenges, a ROS-based system has been developed, providing open access with promising results (low delay and run-time visualization). This system enables remote control of robotic manipulators and has been successfully validated through the remote operation of a real UR3 manipulator.

Keywords: Remote operation, virtual and remote laboratories, robotic manipulators, ROS-based.

Resumo

Explorar as áreas de pesquisa, treinamento e aprendizado no campo de sistemas robóticos apresenta obstáculos para instituições que não possuem a infraestrutura necessária. O investimento significativo exigido para adquirir sistemas robóticos físicos muitas vezes limita o acesso e dificulta o progresso nessas áreas. Embora as plataformas de simulação robótica forneçam um ambiente virtual para experimentação, o potencial dos ambientes robóticos remotos vai além disso, permitindo que os usuários interajam com sistemas robóticos reais durante atividades de treinamento e pesquisa. Dessa forma, os usuários, incluindo estudantes e pesquisadores, podem participar de uma experiência virtual que transcende as fronteiras geográficas, conectando-os a sistemas robóticos do mundo real por meio da Internet. Ao estabelecer uma ponte entre os mundos virtual e físico, os ambientes remotos oferecem uma experiência mais prática e imersiva, abrindo novos horizontes para a pesquisa colaborativa e o treinamento. Democratizar o acesso a essas tecnologias significa capacitar instituições educacionais e centros de pesquisa a se envolverem em experiências práticas e de aprendizado prático. No entanto, a implementação de ambientes robóticos remotos traz consigo um conjunto próprio de desafios técnicos: comunicação, segurança, estabilidade e acesso. Diante desses desafios, foi desenvolvida uma plataforma baseada em ROS, oferecendo acesso aberto com resultados promissores (baixo *delay* e visualização em *run-time*). Essa plataforma possibilita o controle remoto de manipuladores robóticos e foi validada com sucesso por meio da operação remota de um manipulador UR3 real.

Palavras-chave: Operação remota, laboratórios virtuais e remotos, manipuladores robóticos, baseado em ROS

Contents

List of Tables

List of Figures

Acronyms

API Application Programming Interface.

BSD Berkeley Software Distribution.

CeDRI Research Centre in Digitalization and Intelligent Robotics.

CPU Central Processing Unit.

CSS Cascading Style Sheets.

DHCP Dynamic Host Configuration Protocol.

HRC Human-Robot Collaboration.

HTML HyperText Markup Language.

ICT Information and Communication Technologies.

IoT Internet of Things.

JSON JavaScript Object Notation.

LMS Learning Management System.

RAM Random Access Memory.

ROS Robot Operating System.

RWT Robot Web Tools.

STEM Science, Technology, Engineering and Mathematics.

TCP Transmission Control Protocol.

URDF Unified Robot Description Format.

UTP Unshielded Twisted Pair.

VPN Virtual Private Network.

VR Virtual Reality.

VRML Virtual Reality Modeling Language.

XML Extensible Markup Language.

Chapter 1

Introduction

1.1 Problem

The realm of robotics has undergone rapid expansion in recent years, with manipulator robots progressively assuming a more prominent role in various industries [1]. The advent of Industry 4.0 and the emergence of cyber-physical systems have consequently introduced collaborative robots into the market, resulting in an increased demand for skilled professionals capable of effectively operating these systems and the necessity to retrain the existing workforce accordingly [2]. However, educational institutions and research centers often face resource constraints, making it challenging for them to procure these systems and offer practical hands-on experience with robotic manipulators, develop teaching-oriented tasks, and experiment with innovative robotic approaches.

Recent advancements in Information and Communication Technologies (ICT) and robust communication infrastructures have prompted an upsurge of interest in the remote operation of manipulator robots [3], [4]. Notably, the utilization of remotely operated robots in medical surgeries has become a tangible reality [5], enabling the faithful replication of a surgeon's desired movements during an operation.

Remote operation entails employing physical robots, obviating the fidelity limitations associated with simulated environments, without requiring the operator to be physically

present. This addresses the challenge posed by the unavailability of real robots for local testing.

Existing software programs such as RobotStudio and RoboDK serve as sophisticated platforms for robot development and simulation. However, these programs demand prerequisite knowledge and entail steep learning curves. Moreover, they typically constitute commercial software packages that necessitate local installations, rendering them inaccessible to certain students and researchers.

1.2 Objectives

Against this backdrop, the objective of this dissertation is to propose an open, accessible, and secure solution for remotely operating robotic manipulators in educational settings. The proposed approach allows authorized users to remotely access and control robots in run-time through a web interface, which boasts a more user-friendly learning curve. Consequently, individuals can operate robotic manipulators from any location with an internet connection, thereby enhancing accessibility and convenience. This system must provide run-time access to visualizations of both real and simulated robots, allowing for safe actuation and movement control. Furthermore, the intuitive and user-friendly interface of this system distinguishes it from the complex and technical interfaces of RobotStudio and RoboDK, rendering it a valuable addition to the existing repertoire of tools and platforms in this domain. As such, this system plays a role in democratizing access to robotics education by virtue of its ease of access, intuitive commands, free usage, and generic architecture that facilitates replication, ultimately increasing its global availability.

Furthermore, this work aims contribute to the advancement of Science, Technology, Engineering and Mathematics (STEM) education, since it must provide a friendly and open tool for students and researchers in the field of robotics to expand their knowledge and further their studies.

The methodology to achieve the stated objectives involves carrying out the steps:

- Analysis of existing solutions and technologies related to remote robot operation;
- Definition of the requirements and specifications for the proposed system, taking into account the needs of potential users and the available resources to make it a free and open-access tool;
- Design and development of the open access and ROS-based system, using web application, that ensures secure and reliable communication;
- Implementation of run-time data visualization and control features, supporting trajectory motion planning and collision avoidance, and enabling users to monitor and manipulate the robot from a remote location;
- Testing and evaluation of the system with real and simulated robots, to assess its performance, robustness, and ease of use;
- Documentation of the developed system, including a detailed description of its architecture, algorithms, and user interface, to make it easily implementable by other laboratories.

The proposed approach has been subjected to experimental validation using a UR3 collaborative robot. Furthermore, the experimental tests conducted on this system have yielded operational and promising outcomes, characterized by negligible communication delay and run-time data exchange, in line with the initial project objectives.

1.3 Document Structure

This document consists of six chapters that present a review of the tasks and results achieved throughout the work's progression.

Chapter 1, which is the Introduction, outlines some of the primary challenges in the field of simulated environments for students and researchers. The chapter then presents

a proposal of solution to improve accessibility to robotics education through remote operation and the sharing of resources from research centers that possess the necessary technologies.

In Chapter 2, the current state of remote operation of robotic manipulators is analyzed and discussed, including the technologies and techniques employed, as well as the current methods of teaching STEM education. This chapter provides an overview of the existing solutions and practices in the field, helping to inform the design and development of the proposed system.

Chapter 3 rounds up the design and structure of the proposed system in this dissertation, including the components and modules that make up the system and how they interact with each other to achieve the desired outcomes.

Chapter 4 provides the steps taken in implementing the proposed system. This section outlines the procedures involved in designing and developing the system for remote operation of robotic manipulators. It explains the techniques used to ensure secure and reliable communication between users and the robot, as well as the processes involved in integrating run-time data visualization and control features.

In Chapter 5, the results of the tests performed on the proposed system are presented and analyzed. The outcomes of the system's performance, efficiency, and functionality are discussed and evaluated against the objectives and requirements defined in previous chapters.

Chapter 6 includes an overview of the successes and limitations of the work accomplished during the development of the dissertation, as well as recommendations for future improvements and possible new directions to be explored in future work.

Finally, the appendices contain the files for replication of the developed system and the expanded results of the tests performed.

Chapter 2

State of the Art

This chapter presents a review of the state of the art in remote operation of robots and STEM education, including a discussion of the technologies used, the methods applied, and the challenges encountered. The aim is to provide a overview of the current state of research in this field and to identify the key trends and future directions of research.

2.1 Remote Robot Operation

The increasing use and applications of robotics in various domains of daily life, including medical, domestic, space, and aquatic exploration, have been widely acknowledged. The concept of robotics has been present in popular culture since the end of the 1910s and the beginning of the 1920s, primarily as science fiction [6]. To popularize this concept, Asimov introduced a set of rules for fictional robots in his science fiction works, particularly his Robot and Foundation series. These rules require robots to prioritize human safety and well-being, and forbid them from causing or allowing harm to humans. The impact of these rules on the development of real-world robotics and artificial intelligence has been significant, with researchers using them as a starting point for ethical guidelines and safety protocols for robots [7].

With the evolution of mechatronics, control and automation engineering, computing, and related fields, robots have not only transitioned from science fiction to reality, but it

has also become possible to operate them remotely. The remote operation of robots can be described as the control of robots by operators located at a distance, using a combination of communication technologies (e.g., internet, satellite, radiofrequency) and control interfaces (e.g., teleoperation, supervisory control, autonomous control) [8]. According to Boboc et al. [9], due to the potential to enable robots to perform tasks in environments that are inaccessible or dangerous for human beings, remote operation is widely explored in the fields of healthcare, manufacturing, mining, and space/underwater exploration to enhance safety, productivity, and efficiency. To explore the potential of remote operation of robots and robotic manipulators, it is inevitable to navigate through these areas that have the highest number of research and, consequently, the greatest development of this technology.

According to Hirzinger et al. [10], the first remotely controlled robot in space was ROTEX, which operated on board a spacecraft (Spacelab D2 on shuttle Columbia) in April 1993. This robot performed a number of prototype tasks such as grasping, assembling and inspecting objects using different operational modes: preprogrammed, remotely controlled and supervised autonomous. This was a significant milestone for space robotics, as it showed the potential of using robots for complex and delicate tasks in orbit. However, remote operation also poses many challenges for space robotics, such as: communication delays (the distance between Earth and other celestial bodies causes significant time lags in sending and receiving signals, which can affect the performance and safety of robotic systems), limited bandwidth (the amount of data that can be transmitted and received by robotic systems is limited by factors such as power consumption, antenna size, interference etc) and human factors (the operators of robotic systems need to have adequate training, skills and situational awareness to control them effectively, they also need to cope with stress, fatigue and boredom that may arise from long-duration missions).

The first reports of remote operation of robotic systems in healthcare were of surgeons manipulating handles a few meters away from the patient while the robot replicated the doctors' hand movements. However, the case that set new precedents and expanded the

horizons of this kind of remote operation in the field of medical surgery was undoubtedly the first case of transatlantic surgery [11]. Zeus was a surgical robot system that was used to perform remote operations across long distances and one of its most famous applications was a transatlantic cholecystectomy (removal of gallbladder) performed by a French doctor on a patient in New York in 2001 [12], [13]. Some of the difficulties faced by this remote robotic surgery were similar to those faced by the first remote space exploration operations, including latency (delay in signal transmission), bandwidth (capacity of data transfer), reliability (risk of technical failure), security (protection from unauthorized access), and ethical (legal and moral) issues. Some of the advantages of remote robotic surgery include improved precision, reduced trauma, enhanced vision, and expanded access to expert care [13], [14].

The development of robots and remote operations was made possible through engineering sciences advancements, despite the challenges of latency and limited bandwidth. The emergence of ICT has complemented these advancements, facilitating the mitigation of these challenges by improving the robustness and speed of communication infrastructures [15].

Yaovaja and Klunngien [16] demonstrated the potential of 5G mobile technology to facilitate remote robot operation. The research points out the benefits of 5G, including low latency and stable connections, which can enhance the reliability of wireless communication for industrial robots, while also addressing the challenges of teleoperation, such as system instability and time lag. The utilization of 5G technology offers a significant improvement in latency and bandwidth, ten times greater than its 4G predecessor, marking a remarkable advancement in wireless network capabilities since the time of the initial remote operations.

The use of a 5G wireless network for laparoscopic telesurgery employing the “MicroHand” surgical robot system was also investigated, and the results revealed that the 5G network enables secure and seamless telesurgery, even at ultra-remote distances exceeding 3000km. The 5G network is expected to offer more pronounced benefits with the

transmission of multimodal data and 4K/8K high-definition video [17]. Thus, the advancement of ICT has enabled remote robot operation to expand further, transforming it from a Utopian concept to a more widely used tool. This trend is evidenced by Figure 2.1, which displays the Scopus database's search results for documents published between 1960 and 2022 using the query "(remote AND operation) OR teleoperation" (common terminologies used to describe remote robot control) in the title, abstract, or keyword fields.

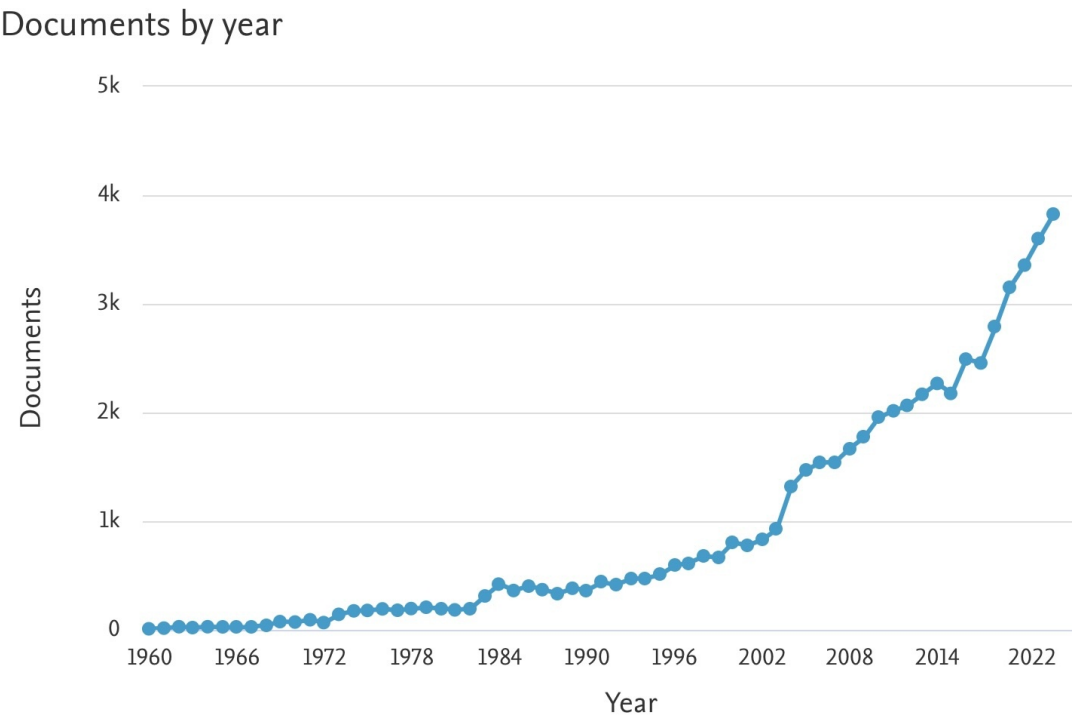

Figure 2.1: Evolution of papers related to remote operation over time (data taken from Scopus in March, 2023).

Through a comparison of an ICT survey [15] with results obtained from the Scopus database search, it is evident that remote robot operation has steadily progressed since the inception of the Third Industrial Revolution in 1969. The introduction of Ethernet-based networks in the 2000s led to a surge in the implementation of remote operation works, followed by a remarkable increase in production in this area from the 2010s to the present day. This progress has been driven by the development of faster mobile networks,

the Internet of Things (IoT), and Industry 4.0.

Despite significant advancements in the field, challenges remain regarding remote robot operation, including [18]–[24]:

- Uncertainties and time delays in the teleoperation system;
- Limitations of control algorithms;
- Difficulties in information perception;
- Weak human-robot interaction and perception;
- Limitations in synchronization, security, and transparency.

Luo, He and Yang [18] emphasize the importance of flexible and intelligent teleoperation systems that can handle increasingly complex and sophisticated tasks. Various control algorithms, robot learning modules, and interaction environment modules are proposed as potential solutions to address these challenges. Moreover, user-friendly human-robot interfaces and robot learning algorithms are suggested to improve weak human-robot interaction and perception.

2.2 Robotics Education

Robotics education is closely linked to STEM education, an educational approach that emphasizes Science, Technology, Engineering, and Mathematics. Robotics education is recognized as a promising domain of modern STEM education that provides students with opportunities to apply their knowledge to solve real-world problems. The aim of STEM education is to enhance the scientific aspect of curriculum by incorporating innovative technologies. Morze and Strutynska [25] point out that the development of robotics education is an important aspect of STEM education.

The Fourth Industrial Revolution and Industry 4.0 are bringing about significant changes to the way of living and working. These changes are driven by emerging technologies, i.e. robotics, artificial intelligence, the Internet of Things, among others. As these

technologies continue to advance, the demand for skilled professionals who can design, develop, and operate them is increasing proportionally [26]. This way, STEM education is becoming vital in fulfilling the growing demand for skilled professionals in the Industry 4.0. As emerging technologies and robots become more prevalent in various industries, the need for individuals with the necessary STEM skills will only continue to grow. Therefore, STEM education has gained increasing prominence worldwide as investing in this education is crucial for ensuring that individuals are prepared for the job market of the future and that countries remain competitive in the global economy [25]. However, many issues are still challenging in robotics education.

The challenges faced by universities in Japan in introducing undergraduate students to the basics of developing intelligent automated mechanisms, particularly in the field of robotics, are discussed by [27]. The proliferation of personal robotic platforms and their use in various applications in Japan pose a significant challenge for universities to provide undergraduate students with the necessary knowledge to design intelligent automated mechanisms. The current engineering curriculum lack practical and design components, which limits opportunities for students to improve their skills. This deficiency may have a substantial impact on the industry's ability to compete globally in the future by creating a shortage of skilled engineers.

In scientific discourse, there is much discussion about the integration of new technologies and robotics into science and engineering education. However, it is critical to recognize the budgetary and resource constraints of schools, universities, and research centers. Despite the potential benefits, the incorporation of new technology and robotics education into the curriculum remains limited in many educational settings. While there are initiatives to introduce innovative educational robots into classrooms, these efforts lack the necessary maturity to become widely accessible. To address this, it is essential to encourage information exchange and collaboration among different sectors to achieve the goal of widespread access to robotics education [28].

Currently, the way robotics education is traditionally conducted is viewed as problematic by several researchers. The most frequently cited challenges include the expenses

associated with purchasing and maintaining physical robots, the challenges of setting up and cleaning up after using it, the limited range of scenarios that can be presented by physical robots, and the requirement for physical presence in a classroom, which can be a hindrance to both students and educators [29]. Furthermore, for Correll and Rus [30], one of the main challenges in robotics education is the risk of cognitively overloading students with technology-intensive projects that have a steep learning curve.

In response to the high costs and limited resources associated with robotics education, educators and researchers often employ robotics simulations. Simulators offer numerous benefits in educational robotics, including promoting student engagement in robot development and programming, reducing costs and saving resources, facilitating the learning process, and enhancing accessibility while decreasing expenses. Nevertheless, simulators encounter limitations and challenges that need to be addressed, such as accurately modeling real-world conditions, providing limited interfaces for student interaction and collaboration, and possibly falling short of replicating the experience of working with actual robots [31]. Thus, an evolution of this alternative in education has come into focus: the virtual laboratories.

2.3 Virtual Laboratories

Virtual laboratories refer to computer-based simulation environments that enable users to conduct experiments and simulate scientific phenomena in a virtual setting [32]. These laboratories have been developed to address the physical infrastructure challenges faced by education, including robotics [33]. Research in this area has gained significant attention since the outbreak of the COVID-19 pandemic in 2020, with remote activities taking center stage due to social distancing measures. The evolution of this research can be tracked through the Scopus database using the search query "("virtual laboratory") OR ("remote laboratory")", as illustrated in Figure 2.2.

Gomes and Bogosyan [34] highlight several benefits of virtual laboratories in engineering education, including flexibility, cost savings, support for autonomous learning,

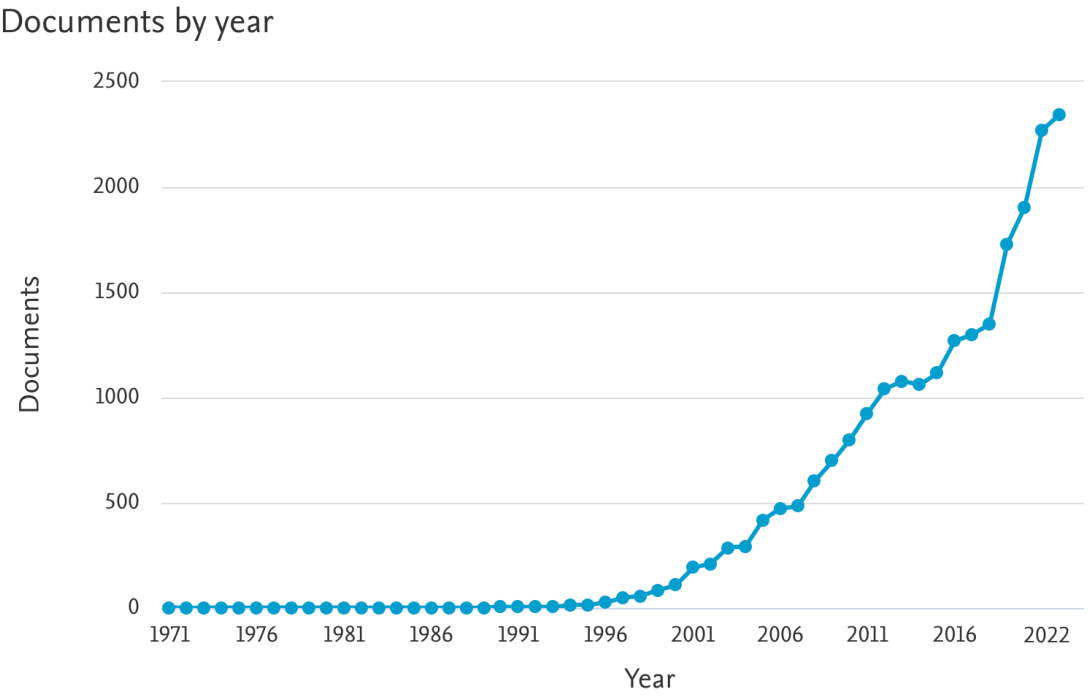

Figure 2.2: Evolution of papers related to virtual laboratories over time (data taken from Scopus in March, 2023).

increased interaction among remote users, access to diverse experiments and projects, and the potential to facilitate collaborations between institutions, teachers, students, and researchers worldwide. Additionally, virtual laboratories can address safety and security concerns related to the use of costly equipment at multiple levels. However, associated challenges are also recognized, such as the lack of standardization, which impedes modularity, portability, and scalability of solutions. Furthermore, facilitating easy access to remote laboratories from developing countries with limited computer access and constrained bandwidth is another challenge, along with the need to support new technologies that can offer the possibility of implementing new functionalities.

In 2003, Candelas et al. [35] developed a virtual laboratory for teaching robotics, which enabled simulation and teleoperation of a robot over the internet, despite the technological limitations of that time, e.g., less advanced operating systems and limited bandwidth. The system comprised a real robot arm, a web server, a database server, and a video server. The user interface, depicted in Figure 2.3, was a web page with a Java

applet and a Virtual Reality Modeling Language (VRML) window that displayed the simulated state of the robot. Additionally, the system included a database server for user information and test questions, and a web server for processing commands and queuing teleoperation requests. This system approaches with the concept proposed in this present work, with each respecting the available technologies of their respective times.

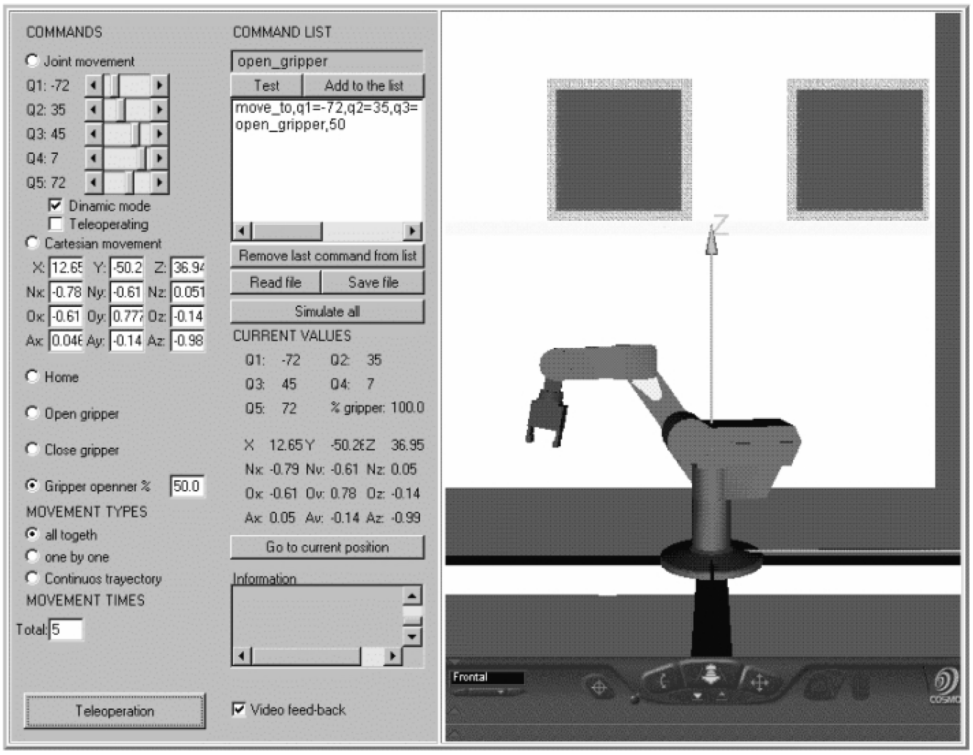

Figure 2.3: Web page available to the virtual laboratory user in 2003 [35].

Two different approaches have been used to develop virtual laboratories for teaching industrial robotics programming. In one approach [36], RobotStudio, ABB's commercial simulation software, is utilized to develop the virtual lab, allowing students to test basic concepts in robot programming. However, this approach requires the paid software to be installed on users' computers, which may not be accessible to everyone. In contrast, another approach [37] involves using open-source platforms like Robot Operating System (ROS) and MoveIt!. The virtual lab developed using this approach consists of two robot work cells with the same hardware setup and is designed to be remotely accessible, with a fully automated and efficient processing pipeline for experiments. This approach provides

an accessible option for students to learn industrial robotics programming.

Virtual laboratories can be classified into three types based on the level of interaction [38]. The first type involves interaction with a simulated or emulated model of the experiment through a computer interface. This type of virtual laboratory allows students to perform difficult, dangerous, or expensive experiments to conduct in a physical laboratory. The second type involves direct control of real devices and instrumentation equipment through a computer interface, often using virtual instrumentation or virtual reality environments. This type of virtual laboratory is useful in situations where remote access to equipment is necessary or when physical experiments are not feasible. Finally, hybrid virtual laboratories offer a combination of both simulated and real equipment, providing the benefits of both approaches [39].

A virtual laboratory architecture has been developed at Stevens Institute of Technology [40], which utilizes a client-server network approach to enable experiment execution, as shown in Figure 2.4. The laboratory is connected to the outside world through a Web server that hosts the process queue, input and output data files, and the graphical user interface. This server is networked to individual data acquisition PC terminals that control the experiments and report the results back to the Web server. The system has been designed to be modular, scalable, and compatible with existing communication standards. It has been piloted with students for mechanical and electrical experiments, resulting in highly positive feedback. Therefore, this architecture can serve as the foundation for a robotic system as well.

Liu and Wang [41] discuss the design of a remote Human-Robot Collaboration (HRC) system that enables a human operator to remotely lead-through an industrial robot with a local robot. The system is designed with four different operation modes, including a real robot controlling a real robot, a virtual robot controlling a real robot, a real robot controlling a virtual robot, and a virtual robot controlling a virtual robot. The remote robot control system is the core of the proposed remote HRC system, and it includes a cyber part and a physical part. The physical part of the remote robot control system monitors the real-time robot joints positions from the robot controller of the collaborative

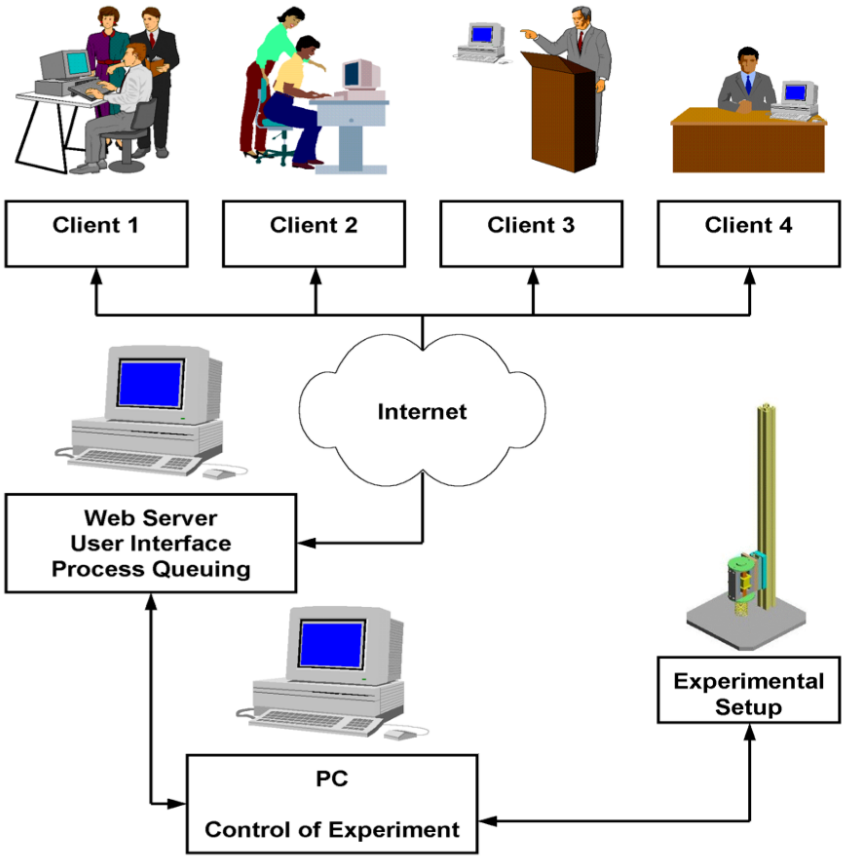

Figure 2.4: Basic architecture of a virtual laboratory (adapted from [40]).

robot, and the cyber part of the system calculates the forward kinematics of the collaborative robot with the joint positions and robot kinematics. The difficulties of remote robot operation are the need to guarantee the same geometrical motions for the collaborative robot and the remote robot during human lead-through, and the need to calculate the forward and inverse kinematics to adapt to the different robots.

In virtual laboratories, it is essential to consider the user's entry barrier and learning curve. When using advanced software that requires prior knowledge or is not user-friendly, the user may lose interest in learning, become frustrated, and even give up on this teaching method. According to Coleman et al. [42], the entry barrier refers to the time, effort, and knowledge required for a new user to integrate a software component with an arbitrary robot. This includes creating a virtual model of the robot's geometry and dynamics, customizing configuration files, choosing the fastest planning algorithm, and

tuning parameters. Additionally, as robotic software frameworks' capabilities increase, the setup difficulty and learning curve for new users also increase. A well-designed user interface, whether graphical or command-line, should be intuitive enough not to require documentation for most users.

For this proposal, the open-source Robot Web Tools (RWT) project [43] was developed, which focuses on enhancing interoperability and portability between heterogeneous robot systems, devices, and front-end user interfaces. The project employs the rosbridge protocol for ROS message topics in a client-server model that is suitable for wide area networks and human-robot interaction on a global scale using modern web browsers. The RWT includes client and visualization libraries that facilitate efficient development of front-end human-robot interfaces and more effective methods of transporting high-bandwidth topics for cloud robotics. The project has been successfully implemented on various robot platforms and utilized in a diverse range of applications, including virtual laboratories [44].

Using the RWT, Casan et al. [45] developed a system with different ROS configurations to construct open robotics courses for educational and training purposes. The programming courses were developed through Moodle, a Learning Management System (LMS) that allows educators to create online courses and manage virtual learning environments. Figure 2.5 illustrates an example activity developed in one of the open courses, highlighting the user interface created in Moodle through the use of RWT libraries.

Gravier et al. [46] reviewed the literature on modern virtual laboratories and identified challenges that need to be addressed to enhance their potential developments. These challenges include lack of reusability, interoperability, collaboration, and integration with LMS. These issues can hinder the integration of different remote lab systems and limit the learning experience for students and faculty. One potential solution is to formalize the software architecture to decrease integration time and required skills, which would allow for greater collaboration across institutions and disciplines. Integrating remote labs with LMS would also streamline the learning process and simplify faculty management of student progress. Additionally, the possibility of creating a geographically distributed

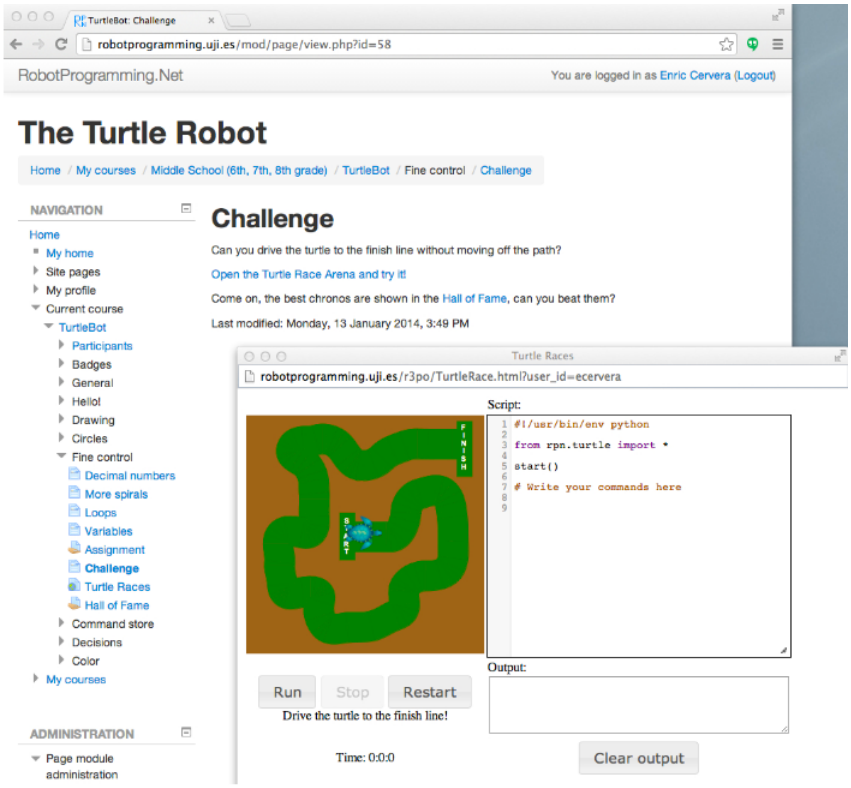

Figure 2.5: User interface using RWT libraries [45].

workbench that could be accessed from different locations is highlighted. This would enable students and teachers to integrate remote labs into their own learning environments. In conclusion, virtual labs have the potential to revolutionize engineering education, but addressing the challenges and limitations of current systems is crucial for realizing their full potential.

2.4 Summary

Therefore, the subject of remote operation of robotic manipulators, in conjunction with robotics research, is highly relevant and has garnered significant interest within the academic community. As evident from the publication trends on the Scopus platform and the literature reviewed in this section, numerous approaches and implementations have been explored. However, a notable gap exists in terms of an open-access system that can

be easily shared across institutions, with the specific goal of promoting broader robotics education. Furthermore, such a system should feature a user-friendly interface that does not necessitate specialized training and offers a smoother learning curve for users.

Chapter 3

System Architecture

In this chapter, the proposed approach for a system designed to facilitate remote robot operation and exploit the benefits of virtual laboratories will be described. The system aims to enable students and researchers worldwide to benefit from both the accessibility and availability of a virtual laboratory and the realism of a physical laboratory. By merging these two domains, the system aims to enhance the learning experience and provide learners with a more comprehensive understanding of robotics.

The proposed approach's fundamental architecture, illustrated in Figure 3.1 , reveals a clear influence of the fundamental architecture of virtual laboratories presented in Figure 2.4.

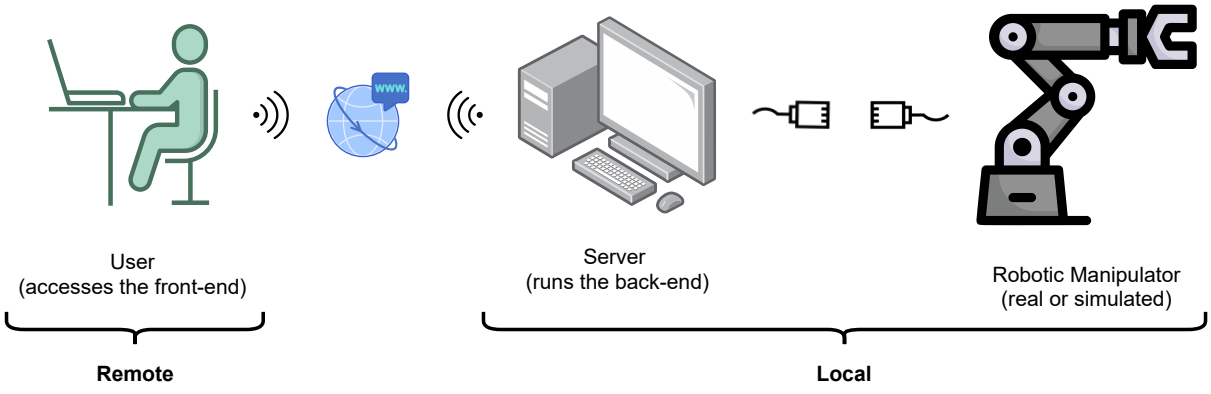

Figure 3.1: Fundamental architecture for the remote robot operation system.

The proposed architecture replaces a generic experimental setup with a robotic manipulator (real or simulated), consolidates the control of the object of study (i.e., the manipulator), and performs the processing of the user interface and the process queuing on a single server computer. This architecture enables remote users to connect via the Internet to the server, which provides the front-end and back-end of the proposed system, and connects to the robotic manipulator via the local network through cable connection.

In a detailed way, the complete architecture is illustrated in Figure 3.2, and the next sections of this chapter will describe each element that makes it up.

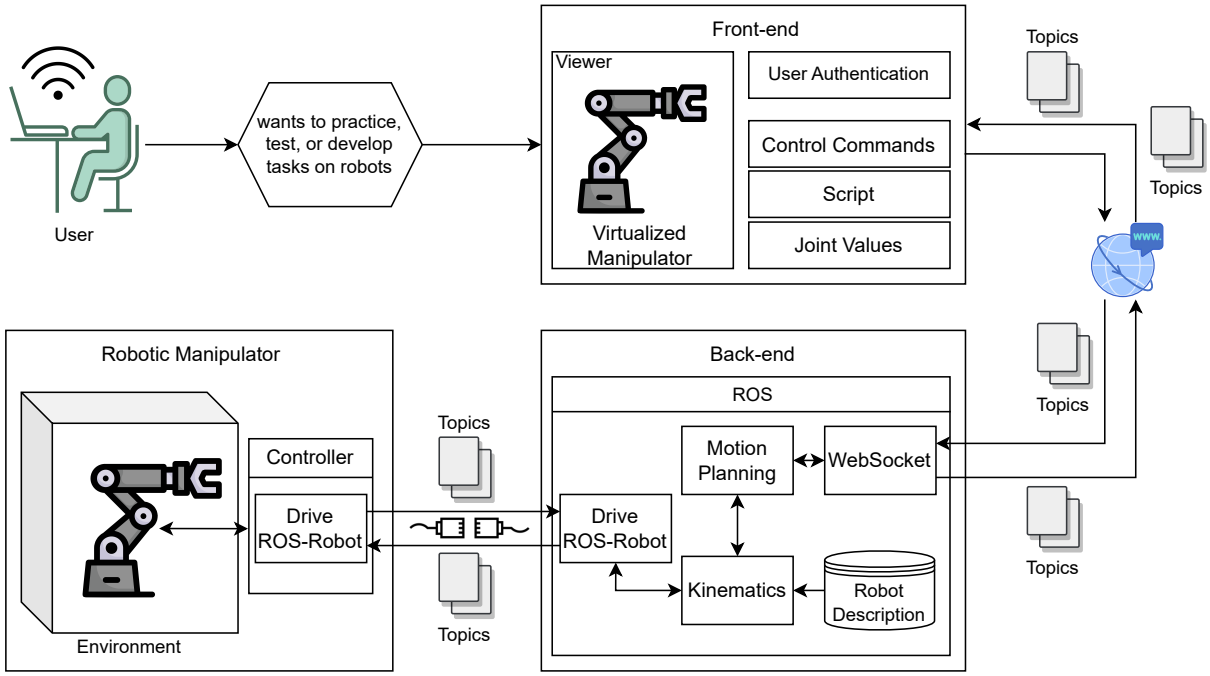

Figure 3.2: Detailed system architecture.

3.1 User Role

The target audience for the proposed system includes high school or technical students focused on robotics, undergraduate, master's, or doctoral students in engineering and related fields, as well as researchers, professors, and robotics enthusiasts. Additionally, the system can serve as a training tool for remote robot operation, allowing operators

and workers from other areas to adapt to the new realities of Industry 4.0 and the robotic world. As a result, the system must cater to a wide range of users, from those with little knowledge of robotics to experts in the field seeking a robot for experimental testing.

Thus, the specific applications that the system was designed for include research environments, such as conducting experiments in robotics, automation, or control systems; and in education and training, as it allows for remote operation and experimentation of robots, which could be particularly useful in areas where physical access to such equipment is limited. In general, it is important to consider the diversity of potential applications and research areas for the system and highlight its versatility and adaptability.

Tasks are specific activities or actions that need to be completed to achieve a particular goal or objective on the robot, varying according to the user's needs or as planned by the teacher/instructor. Some examples of tasks that can be performed in this system include performing pick-and-place tasks, training operators to control a robot in hazardous or inaccessible environments, experimenting with various path planning algorithms, conducting experiments to evaluate the performance of the robot, and training students to operate a robotic system for educational purposes.

The aim of maximizing accessibility for users worldwide was a key consideration in determining the hardware prerequisites necessary for utilizing the proposed system. Consequently, the development of a web-based application was deemed most suitable, allowing for access on a range of devices, i.e. personal computers, tablets, and smartphones. To access the web application, a stable network connection and VPN are necessary. Additionally, 3D graphics can be visualized in the interface provided that the web browser used has Web Graphics Library (WebGL) installed, a software application that facilitates the rendering of interactive 3D graphics, which is natively available in most browsers and runs on most operating systems.

Therefore, the target audience for this system is broad, encompassing individuals who are studying, researching, or working in the field of robotics, as illustrated in Figure 3.3. The use of this system by these users arises due to the unavailability of physical resources and the need to experiment or practice on robots, as discussed in Chapter 2.

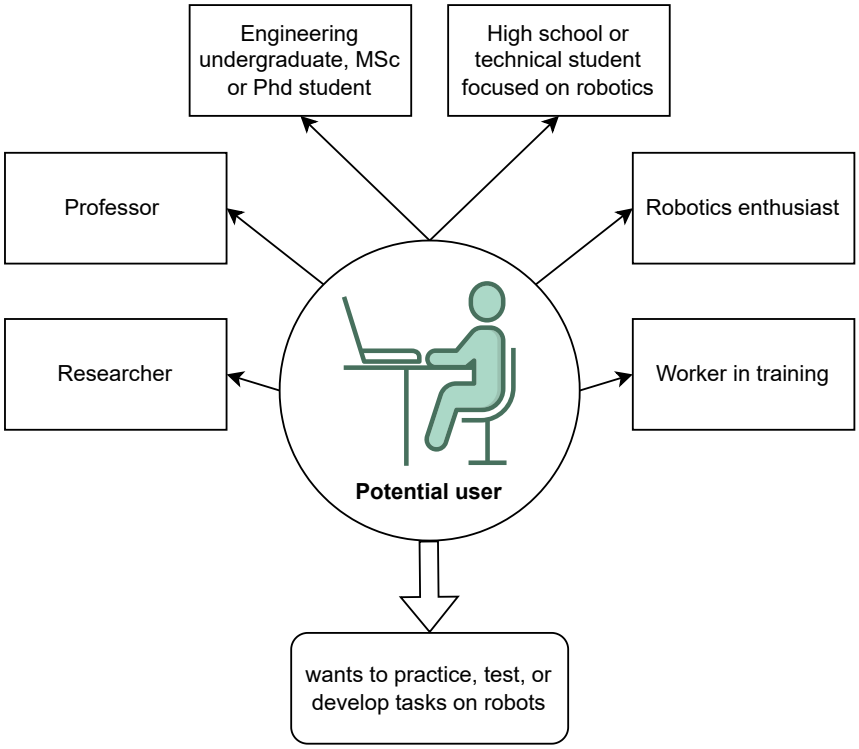

Figure 3.3: Potential user profile of the proposed system.

3.2 Computer Components

It is important to discuss the server-side components of the system, including both the front-end and back-end. The server plays a critical role in providing essential functions, such as establishing communication between the client, i.e. the user, establishing communication between the robot, keeping the system working, and performing complex computations.

3.2.1 Front-end

In web development, front-end refers to the client-side of an application or website that users interact with directly [47]. It consists of the user interface, which includes the layout, design, and visual elements such as buttons, menus, and forms. Popular front-end technologies may include HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript, along with frameworks and libraries such as React, Angular, and

Vue.

Considering the importance of providing a user-friendly interface for a virtual laboratory and remote robot operation, the front-end of the system is developed to connect the end-user with the system. All the information presented to the user when accessing the web application goes through the front-end. As discussed in Chapter 2, the user interface in such applications should be intuitive and simplified. To meet this requirement, a basic user interface framework has been developed to support the system, as shown in Figure 3.4.

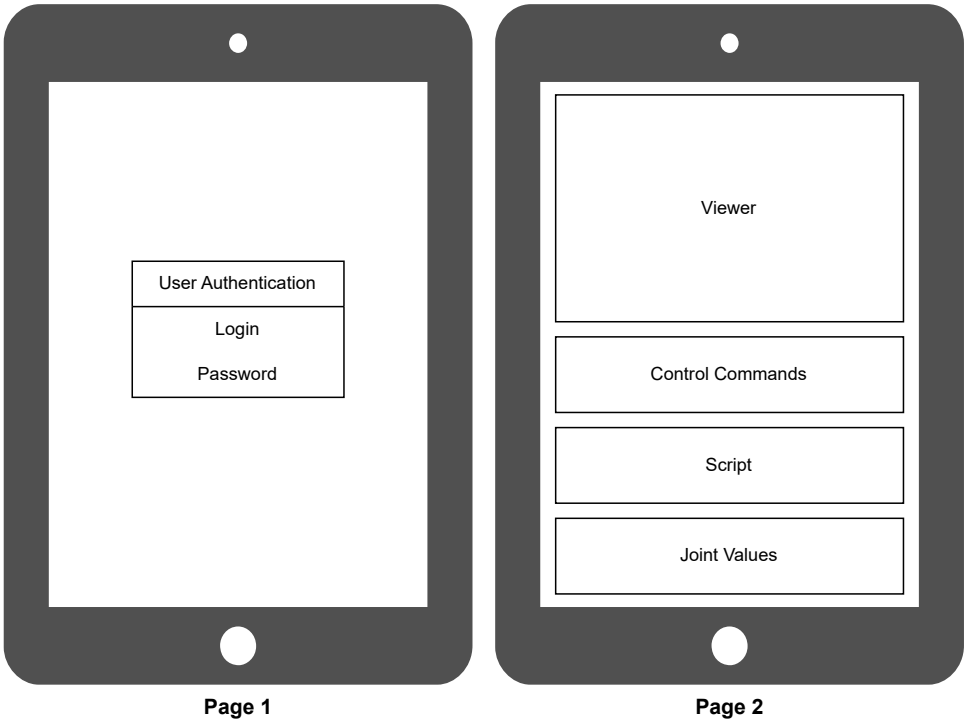

Figure 3.4: Basic user interface framework for remote robot operation.

Upon launching the application, the user is prompted to provide authentication by entering their login credentials, which are provided by the system provider. This serves as a crucial security measure to ensure access control is granted only to authorized users during a valid period. As a result, the user can only access the controls for the robotic manipulator using the provided login credentials while they remain valid.

The page displayed after authentication should provide a 3D view of the robot, i.e.,

a virtualized representation of the robot's current state and the representation of the desired target pose. This viewer contains an interactive marker where the user can make modifications from the current pose to the desired pose, either by changing the position and rotation in each x, y, or z axis independently or changing all axes freely at the same time.

The control commands consist of a collection of buttons and selectors that enable the user to execute or discard desired movements, as well as adjust parameters related to robot control, e.g., the planning path algorithm and parts group.

There is a dedicated section for script creation, where users can save points of interest that have been reached by the robot and create a script to return to these points at any time. These scripts are JavaScript Object Notation (JSON) files that contain the values of each joint necessary to reach the point(s). Users can download these scripts to their local computer and then upload them when they access the application again, which will enable them to resume their work from the previously saved state. It is expected that this functionality will be frequently used for pick-and-place tasks and for comparing path planning algorithms.

At last, the value of each joint of the robot, including the opening and closing value of the attached gripper, is displayed to allow the user to make desired modifications to the poses. This is in addition to the interactive marker which provides an alternative way to modify the pose, but based on joint values rather than by axis.

3.2.2 Back-end

The back-end is responsible for all the operations that occur behind the remote robot operation system. This includes receiving data from the front-end, processing information, and controlling the robot. As previously mentioned, the system aims to minimize the computational demands on the user by passing on only the minimum required information. Therefore, the computer hosting the back-end is responsible for performing most of the processing. This approach differs from other simulators and sophisticated software on

the market that often burden users with high computational demands. The Figure 3.5 illustrates the main components that integrate the system's back-end.

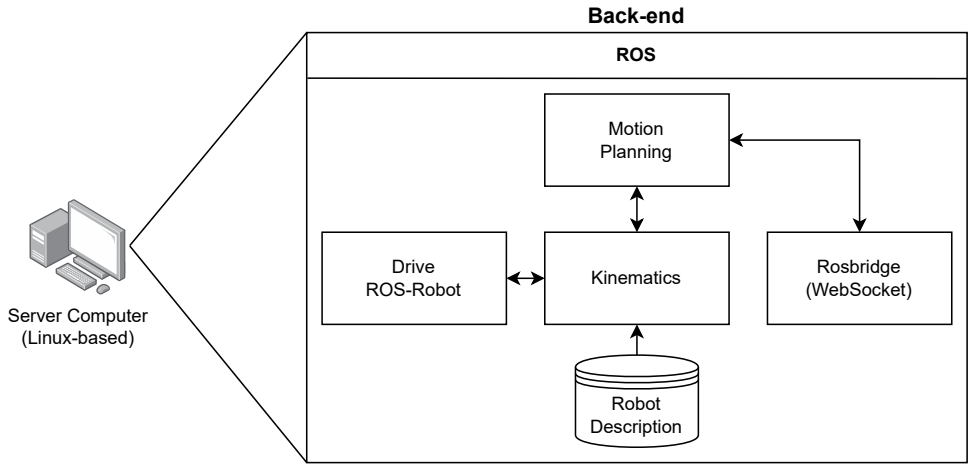

Figure 3.5: Main back-end components.

This Linux-based computer is equipped with ROS, an open-source middleware framework widely used for developing robotic systems. ROS offers a suite of libraries, tools, and conventions that assist software developers in creating intricate robotic systems by abstracting low-level hardware details and providing a common communication infrastructure between different components of the system. Although ROS is extensively used across academia, industry, and hobbyist communities for developing a diverse range of robotic applications, its slow learning curve and specific versions can present challenges for users without extensive experience or formal training in ROS [48]. Therefore, implementing the back-end based on ROS eases processing and communication with the robot, enabling users to operate the system with ease using the user-friendly front-end interface, without requiring any prior knowledge of ROS. Moreover, this approach ensures that the system can be accessed from various devices, not only limited to computers with ROS.

In order to control the robot, it is necessary to generate kinematic commands based on the fixed parameters of the robot, such as its number of Degrees of Freedom, joint limits, and working radius. A database containing robot description files within the server feeds the system with this information. However, before sending the kinematic commands to

the ROS-Robot drive, which exchanges information/topics between the robot and ROS, motion planning must be performed to formulate the appropriate trajectory with the correct kinematic commands.

However, motion planning for robotics is a difficult calculation since it requires determining a path that the robot can travel in its environment and fulfilling limitations such as joint limits, velocity and acceleration limits, and avoiding collisions with other objects. Additionally, the solution needs to take task completion time, efficiency, and safety into account. As a result, in order to efficiently develop viable and ideal courses for the robot to follow, motion planning involves the use of sophisticated mathematical techniques, algorithms, and heuristics [49]. To accomplish this, the server computer must be equipped with specific software that can carry out motion planning and produce kinematic commands that can be transmitted to the robot via the ROS-Robot drivers.

Lastly, the server also needs to include the WebSocket computer communication, which facilitates two-way communication channels over a single Transmission Control Protocol (TCP) connection. This allows web servers to push data to clients (such as a web browser) without being requested, and clients to send data to the server at any time. Since the system is built on ROS, the Rosbridge protocol must be utilized. Rosbridge enables communication with ROS via a WebSocket connection, which enables clients to interact with a ROS system from any programming language that supports WebSockets, such as JavaScript, Python, or C [50]. Clients can use Rosbridge to send and receive messages, call services, and receive notifications about changes in the ROS system. Thus, this communication protocol allows for interaction between the back-end and front-end, server and client, as will be further discussed later on.

3.2.3 Front-end and Back-end Interaction

In a ROS-based system, all interactions and communications between services and platforms occur through ROS topics. These topics are a publish/subscribe communication mechanism in ROS that enable topics to exchange messages with one another. Topics

can send and receive data, which allows for flexible and scalable communication between different components of a robotic system.

As observed, the back-end utilizes the WebSocket-based Rosbridge protocol to communicate with the front-end and establish a connection between the server and the client who is accessing the system via a web-based application, as illustrated in Figure 3.6.

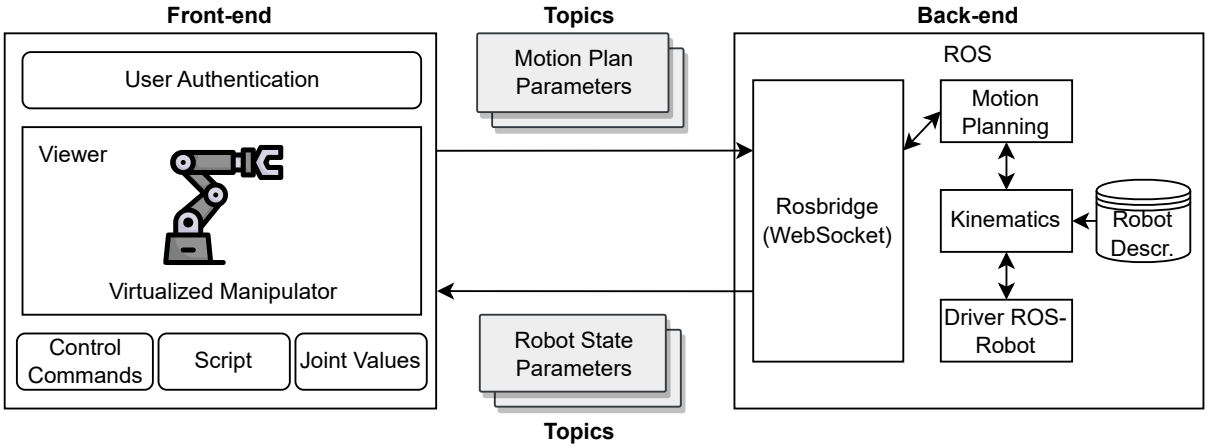

Figure 3.6: Front-end and Back-end Interaction.

The main topics exchanged between the back-end and front-end pertain to crucial parameters for the remote operation system. The back-end receives from the front-end parameters related to motion planning, including user inputs such as joint values at the target position and the chosen trajectory optimization algorithm. Meanwhile, the front-end receives parameters from the back-end related to the current state of the robot for visualization and rendering of the virtualized environment.

The parameters of the motion plan are essential to compose the Motion Plan Request, a message that triggers the motion planning program. This message is composed of several parameters, including start state, goal constraints, path constraints, trajectory constraints, planner ID, group name, number of planning attempts, allowed planning time, max velocity scaling factor, and max acceleration scaling factor. It is this message that links the user inputs to the program that will calculate the trajectory and control the robot.

On the other hand, the parameters related to the current state of the robot are included

in the Robot State message, which contains joint state and attached collision objects. This message is critical in providing the front-end with the ability to represent the robot's current position in a virtualized form.

3.3 Robotic Manipulator

By adopting the concept of hybrid virtual laboratories, the developed system can establish connections with real robots to leverage the benefits of working in real environments, such as higher fidelity than simulations; and, alternatively, the system can connect with simulated robots to reap the advantages of simulation, including greater availability and resource savings.

Therefore, the system offers two distinct methods of connection: the first involves communication with a real robot, where kinematic commands and the Robot State are transmitted through ROS-Robot drivers. To establish this connection, the robot must be compatible with ROS, and the necessary drivers must be installed on both the robot and the server computer. Presently, over 126 models of robotic manipulators from leading brands such as Universal Robots, Franka Emika, ABB, and Fanuc are compatible with ROS and can be integrated into the system [51]. The second method involves using simulated robots, which are executed on the server computer using ROS-integrated software, enabling automatic communication to occur within the computer itself.

For both real and simulated robots, the surrounding environment must be described to the motion planning software in order to prevent collisions with objects or benches. This safety precaution ensures the system functions properly, conserves resources, and enables obstacle avoidance testing.

Chapter 4

Development of the system

The development of a remote robot operation system for robotics education involves multiple complex stages and challenges. This chapter presents the main steps taken to create, design, and implement the proposed system. The aim is to provide a clear description of the development process, including the key decisions, design choices, and technical solutions that led to the final system.

Initially, the educational needs and requirements in the field of robotics were thoroughly analyzed. Through a literature review in Chapter 2, the main goals, constraints, and functionalities that the system should provide were identified, along with the target users and scenarios of use.

Based on this analysis, a high-level architecture and components of the system were defined and presented in the previous chapter. The architecture addressed the primary challenges of remote robot operation, such as communication protocols, bandwidth limitations, and safety concerns, while providing a user-friendly and flexible interface for educational purposes.

To implement the system, an incremental approach was followed, beginning with the development of basic functionalities and gradually adding more complex features and refinements.

4.1 Server Computer and Robot Setup

The server computer used in this phase of the work is equipped with an Intel Core i7-7500U processor and 8 GB of Random Access Memory (RAM), providing sufficient resources for executing the processes described in the system architecture. To ensure compatibility and stability, it was installed the Ubuntu 18.04.6 LTS (Bionic Beaver) and ROS Melodic, the version of ROS compatible with this operating system [52]. The selection of these components was based on their compatibility with the software used, availability of technical support, and the operating system's stability and security.

The manipulator robot available for implementation in the system is the Universal Robots UR3, which is present at Research Centre in Digitalization and Intelligent Robotics (CeDRI). According to the manufacturer's specifications, the UR3 is a small and versatile collaborative robot weighing 11 kg with a payload capacity of 3 kg. It boasts six degrees of freedom and a maximum reach of 500 mm. Furthermore, the robot has a built-in safety system. Due to its features, the UR3 is a popular choice for various applications, including assembly, pick-and-place operations, and quality control.

4.2 Communication Between Robot and Computer

The first step in the implementation process involved ensuring that the UR3 robot could be controlled via ROS from a computer. This was achieved by identifying the corresponding ROS Driver for the UR3 model, typically available on Git and provided by the manufacturer [53]. The Universal Robots ROS Driver is a software package that facilitates the integration of UR robotic arms, including CB3 models such as UR3, and e-Series models, with ROS. It acts as a bridge between the UR robot and ROS, enabling communication and control of the robot using ROS messages, services, and actions.

The driver provides the communication for controlling various aspects of the UR robot, including arm movement, gripper control, and joint control. This enables ROS nodes to send commands to the robot, such as setting joint angles and controlling end-effector

movements. Additionally, the driver monitors the state of the UR robot, including joint angles, tool position, and robot status, and publishes this information as ROS topics. This allows ROS nodes to receive real-time feedback from the robot, providing information on the current robot state.

To ensure seamless communication, it is essential that both the robot and the computer are on the same network. For improved speed and stability of the connection, it is recommended to use a wired connection. To this end, both the robot and the computer were connected to the local network using fixed IP addresses. This prevents any loss of communication due to IP address updates in Dynamic Host Configuration Protocol (DHCP) mode.

Another package that facilitates the integration of the robot with the computer is the Robot Description, typically provided by the manufacturer [54]. This package provides Unified Robot Description Format (URDF) files, which are an Extensible Markup Language (XML) based format used in the ROS ecosystem to define the kinematic and visual properties of a robot. An URDF archive includes information such as robot geometry, joint properties, and visual representations of the robot's physical structure. URDF files are used for robot modeling and simulation, as well as for visualization and control of robots in ROS-based applications, e.g., RViz or Gazebo.

This way, it is possible to visualize the robot in its current state using the default ROS 3D visualizer, RViz, as illustrated in Figure 4.1. The launch file included in the Robot Description package initializes RViz with the robot's specific configurations, and the information exchange through the ROS-Robot Driver begins immediately to display the physical robot's current state.

4.3 Controlling the Robot via Computer

In order to control the robot using the computer, addressing the motion planning challenge discussed in Chapter 3 is essential. As part of the system architecture, the server computer needs to be equipped with a program capable of performing complex motion planning

calculations involving advanced mathematical techniques, algorithms, and heuristics.

Therefore, the MoveIt! software was selected as it is a widely adopted motion planning framework for ROS [55]. This software offers a set of tools, libraries, and Application Programming Interface (API) for motion planning, kinematics, collision checking, and control of robotic systems. It enables robots to autonomously plan and execute motion trajectories while avoiding obstacles and ensuring collision-free paths. MoveIt! is released under the Berkeley Software Distribution (BSD) license, which allows for free usage in industrial, commercial, and research applications.

As MoveIt! is widely adopted, the Robot Description package comes with a pre-configured launch file that initializes MoveIt! with the necessary settings for robot control, including constraint handling, self-collision detection, default poses, and other relevant configurations.

Therefore, with the successful installation and configuration of the visualization and motion planning software, it is possible to switch the robot to external control mode, i.e. enabling remote operation. This mode can be activated using a programming script provided in the ROS-Robot Driver package.

As a result, the UR3 robot's start state could be visualized from the computer, desired modifications could be made to the goal state, and motion planning parameters such as maximum planning time, maximum planning attempts, velocity, and acceleration could be configured, as shown in Figure 4.1.

For the purpose of ensuring operational safety, an object representing the workbench where the robot is positioned was incorporated to prevent potential collisions. Upon executing the command in the control interface, kinematic topics are transmitted to the robot via ROS-Robot Drive, resulting in the execution of the desired motion and effective control of the robot through the computer.

With the planned connection in the system architecture established between the real robot and the server computer, along with the successful transfer of kinematic parameter and robot state topics to enable system operation, the development of the front and back-end will be done.

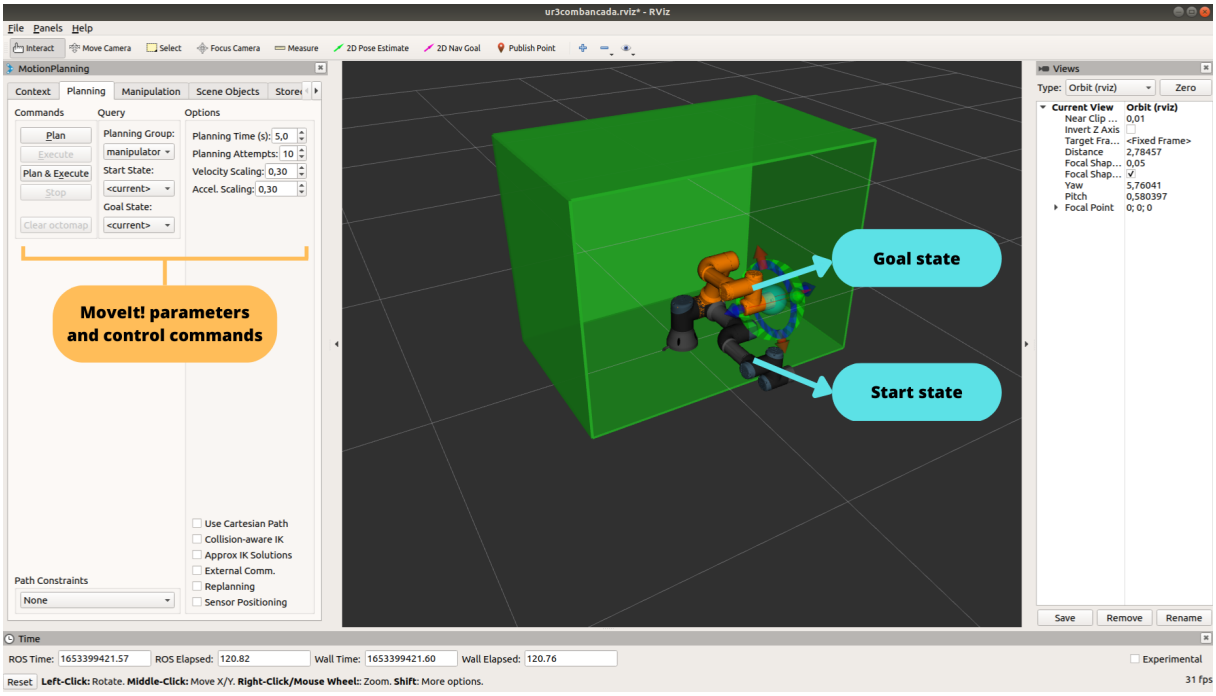

Figure 4.1: RViz software performing the visualization of the external control.

4.4 Building a Web Viewer

The front-end was developed using HTML and JavaScript, specifically designed to meet the system requirements (described in the objectives of this dissertation) and align with the proposed architecture. The primary goal of this initial version was to create a web-based 3D viewer, inspired by RViz software, that would allow for remote operation in a similar manner as the robot-computer connection.

During the initial phase of development, similar proposals for web-based 3D ROS visualizers were found in the conducted research. However, none of them explicitly aimed to enable remote control of a robotic manipulator. Two notable examples were RVizWEB [56] and Zethus [57]. Given that these projects are open-source, a comprehensive feasibility study was conducted to explore the potential integration of MoveIt! with these platforms to add remote operation functionality for robots through the web. Nevertheless, some challenges were identified during this preliminary study:

- These tools already have well-defined user interfaces for their respective functions.

Integrating MoveIt! within these existing interfaces could potentially compromise their intuitiveness and deviate from the primary objective of this work, which is to create a user-friendly application;

- Transforming such projects into a remote operation system would render some functionalities, which are primarily focused on object visualization and rendering, unusable, thus deviating from the original goals of the developers;

- Adapting these projects to fulfill the proposed function of this work would likely bring more challenges than benefits.

In spite of the challenges outlined previously, an analysis was carried out on the commonly employed libraries in these web-based ROS projects, with special focus on those developed by RWT as mentioned in Chapter 2. Among these, a suite of JavaScript libraries known as "Visualization RWT" emerged as particularly relevant for this work, as it is specifically designed to provide web-based visualization capabilities for ROS applications.

The "Visualization RWT" library suite combines ROS nodes to create interactive and dynamic visualizations of robotic systems, including robots, sensors, and their environments, in a 3D space. This aligns with the initial plan of this work in constructing a web-based visualizer inspired by RViz, as the libraries provide a set of tools and functionalities for rendering ROS data, visualizing robot models, displaying sensor data, and enabling real-time interaction with the 3D scene using web browsers.

In fact, one of the examples of application of these libraries is the integration of MoveIt! with the web-based visualizer, which would be in complete alignment with the proposal of this work. However, the RWT project has been discontinued for some years, and has not received proper updates, resulting in some lost functionalities due to incompatibilities, data loss, and malfunction.

The installation of the requisite packages to execute "Visualization RWT" was performed in accordance with the primer provided by the developer in 2014 [58]. This approach ensured that all the necessary dependencies were installed correctly, and any

execution failures should be attributed to the lack of updates from the developer, rather than missing dependencies.

When running the example HTML code provided on a web browser, only the basic functions such as buttons and titles continued to work, as shown in Figure 4.2, while the functionalities dependent on the RWT libraries did not appear or function properly. According to the developers, the example was supposed to generate a page as described in Figure 4.3.

Figure 4.2: The state of the example web page with outdated RWT libraries.

Based on the comparison between the two web-pages, it is evident that the tool has become non-functional, as the 3D visualization and robot parameters are not displayed. Nevertheless, despite this limitation, it is worthwhile to address compatibility and programming issues through updates, as the “Visualization RWT” libraries hold significant potential for remote operation of robotic manipulators through web application.

As part of the first version, each library was downloaded individually to the local host

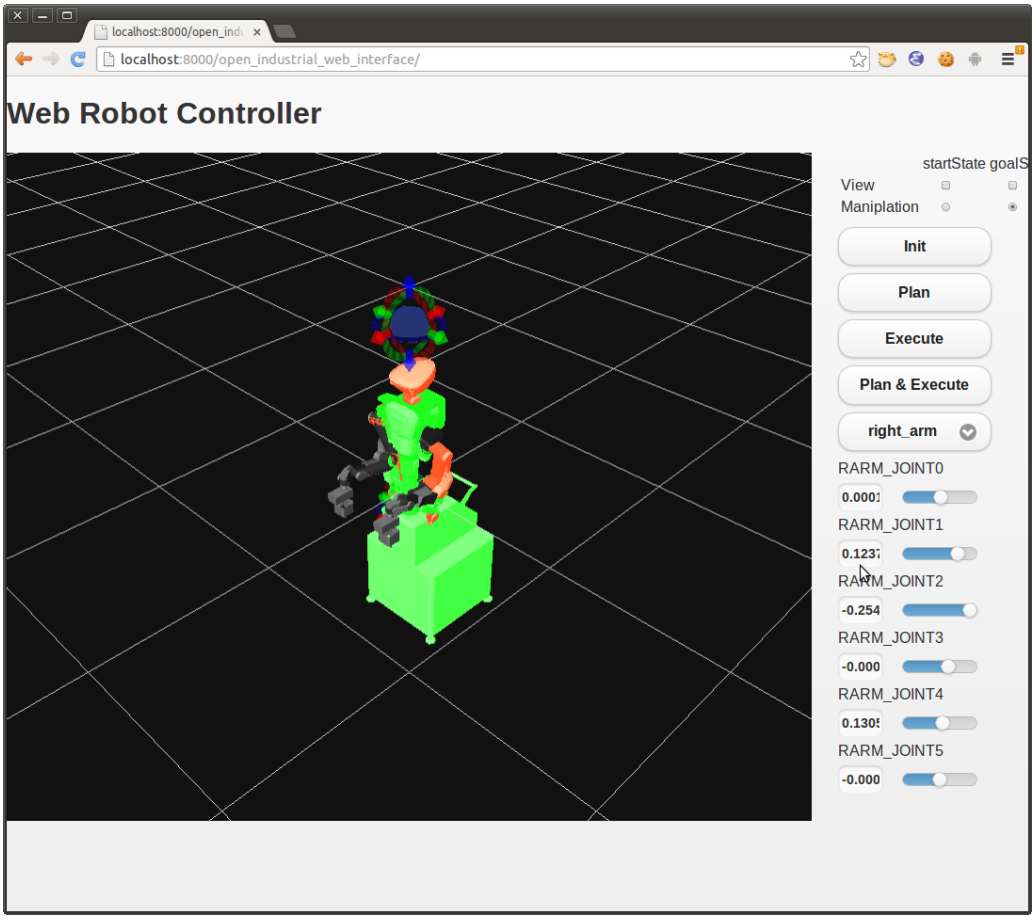

Figure 4.3: How the example web page should work [58].

due to the unavailability of public links for most of the libraries developed by RWT. This prevented errors when the web page attempted to locate these libraries in the HTML code. To address this issue, a search was conducted to locate each library in other projects that had utilized them, and they were subsequently downloaded and stored in a designated folder on the local computer. These libraries were then included along with the HTML index file. Detailed information about the libraries used and added in the original project can be found in Table 4.1.

As a result, the HTML code was adjusted to enable local retrieval of most libraries, while preserving those that continued to offer support through public links. Despite these efforts to address the “404 - not found” errors associated with the libraries, the 3D viewer remained inaccessible on the web page, suggesting the existence of another obstacle

Table 4.1: List of libraries used with brief descriptions.

Library Name	**Description**
jQuery U	Styles for applying pre-designed themes to UI elements
Bootstrap	Responsive styles for creating visually appealing, mobile-friendly web pages
Font Awesome	Icon font toolkit for adding scalable vector icons to HTML elements
Vue	Building user interfaces and single-page applications
EventEmitter2	Implementing the EventEmitter pattern for event-driven programming
RosLib	Communicating with ROS in web applications
BootBox	Creating programmatic, customizable modal dialogs using Bootstrap
Three	Creating interactive 3D visualizations in web applications
ColladaLoader	Loading 3D models in Collada format
STLLoader	Loading 3D models in STL format
ColladaLoader2	Updated version of ColladaLoader for loading Collada format 3D models in Three
ROS 3D	Rendering 3D visualizations in web applications using ROS
MJPEGcanvas	Rendering MJPEG video streams in HTML5 canvas
Keyboard Teleop	Controlling robots using keyboard inputs in web applications
File Saver	Saving files on the client-side in web applications
Popper	Positioning popovers and tooltips used in conjunction with Bootstrap
App	Custom JavaScript file for implementing application-specific logic in a web application

hindering its rendering.

The main code responsible for enabling communication between user inputs, such as buttons, interactive markers, and selectors generated by the HTML code, and the ROS topics to be sent or received via Rosbridge by the server computer is referred to as “Basic.js”. This JavaScript code connects all the front-end functionalities to a specific topic and generates visualizations based on the parameters received. Originally, the “Basic.js” needed to be downloaded locally and located on the server computer during web application startup. However, in this work, it was integrated into a script in the HTML code to avoid the need to save it locally and indicate the directory path for file search during web application initialization.

The main viewer, which was not displaying due to a code error, is generated by "Basic.js". As a result, the investigation has centered around this block, which has been included as a script within the HTML index. The viewer is created using the ROS3D library with measurements determined by the user's screen dimensions.

The problem was identified in this code snippet, as the width and height parameters required for ROS3D to display the viewer were causing a formatting error and preventing its generation. To address this issue, the code was modified to the excerpt described in Listing 4.1, which not only rectified the problem but also optimized the dimensions to adapt to varying screen sizes. This way, the main viewer was successfully displayed on the web page, as illustrated in Figure 4.4.

```
1   // Create the main viewer.
2   var width = parseInt($("#app1").css("width"));
3   if (width >= 768) width = parseInt(width * 0.75);
4   else if (width >= 576) width = parseInt(width / 2);
5   console.log(width);
6   var height = Math.max(
7       $(document).height(),
8       $(window).height(),
9       /* For opera: */
10      document.documentElement.clientHeight
11  );
12  if (width < 576) height = parseInt(height / 2);
13  var viewer = new ROS3D.Viewer({
14      divID : 'urdf',
15      width : width,
16      height : height,
17      antialias : true,
18  });
```

Listing 4.1: Updated code of the viewer creation.

To display the virtualized robot model in the viewer, it was necessary to locate the

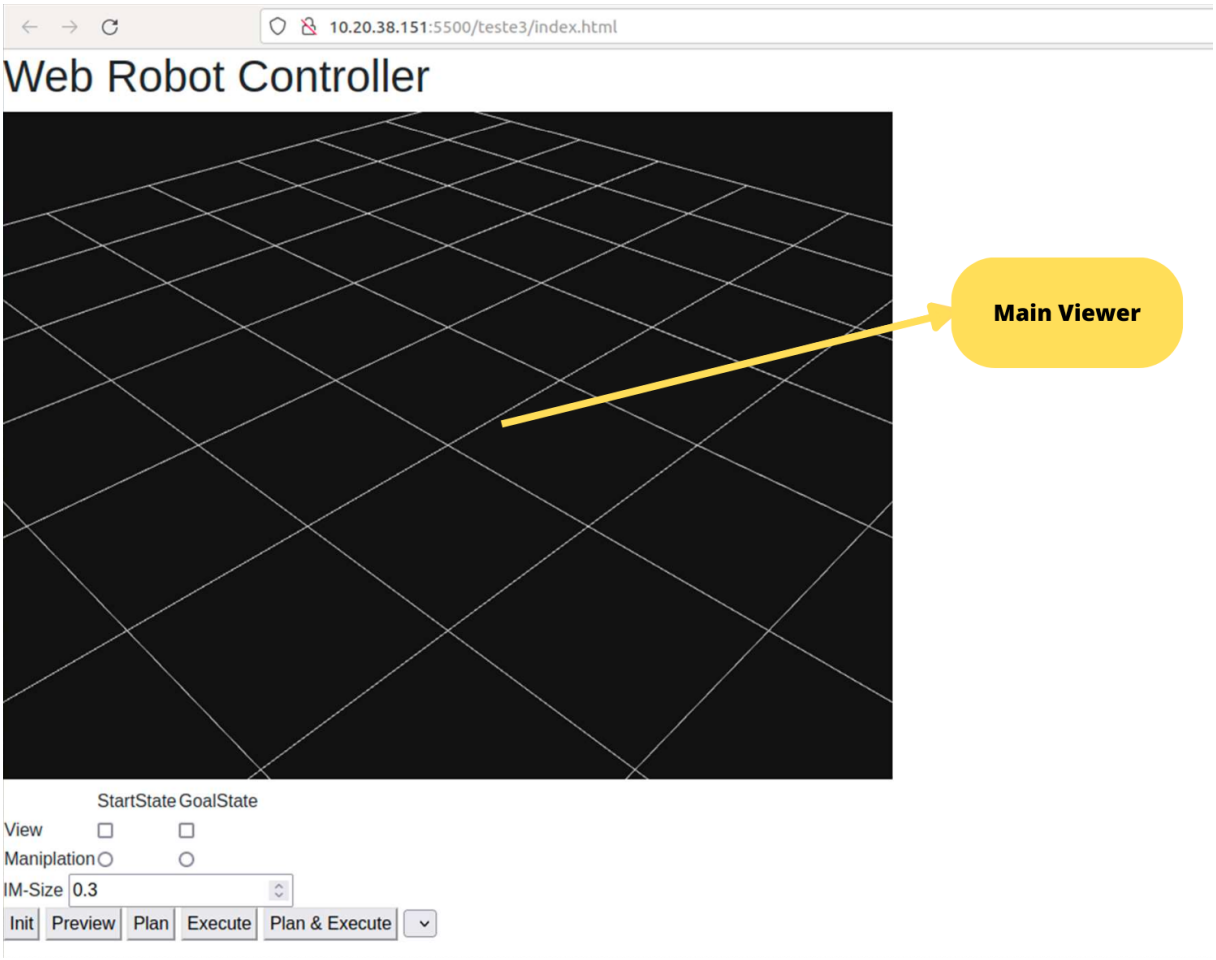

Figure 4.4: The state of the example web page with some updates and fixes.

Robot Description database constructed during the development phase described in Section 4.2. As previously mentioned, the Robot Description packages contain the visual properties of the robot model, enabling its display in the web application. Thus, to perform this search and display the start state and goal state of the robot, as with RViz, the modifications resulted in the code snippet in Listing 4.2. Consequently, upon system initialization, the JavaScript script within the HTML index performs a search for the Robot Description used in the database, retrieving the requisite parameters to exhibit the virtualized robot manipulator, according to Figure 4.5.

Thus, the goal of the first version to construct a web-based 3D viewer, inspired by the RViz software, has been accomplished, allowing for further progress in implementing this

viewer for remote operation of robotic manipulators.

```
    //Display GoalState
    goalState = new ROS3D.UrdfClient({
        ros : virtual_ros,
        tfPrefix : 'goal',
        color : 0xff3000,
        tfClient : tfClient,
        hidden : true,
        param : 'robot_description',
        rootObject : viewer.scene,
        loader : ROS3D.COLLADA_LOADER
    });
    //Display RealState same as StartState
    var urdfClient = new ROS3D.UrdfClient({
        ros : real_ros,
        tfClient : tfClient,
        param : 'robot_description',
        rootObject : viewer.scene,
        loader : ROS3D.COLLADA_LOADER
    });
```

Listing 4.2: Displaying the virtualized robot via Robot Description.

4.5 Integration with MoveIt!

In order to integrate MoveIt! with the components that make up the back-end of the system, it is necessary to use "Basic.js", as previously mentioned, due to its role in programming the connection of front-end elements with the ROS topics to be sent or received. As described in the system architecture, the front-end and back-end exchange ROS topics, which serve as the parameters for forming the Motion Plan Request and Robot State.

The Motion Plan Request is responsible for searching for the parameters that must or can be provided by the user and that will be taken into account for MoveIt! to perform the motion planning. Accordingly, based on the user inputs, e.g., pressing a button

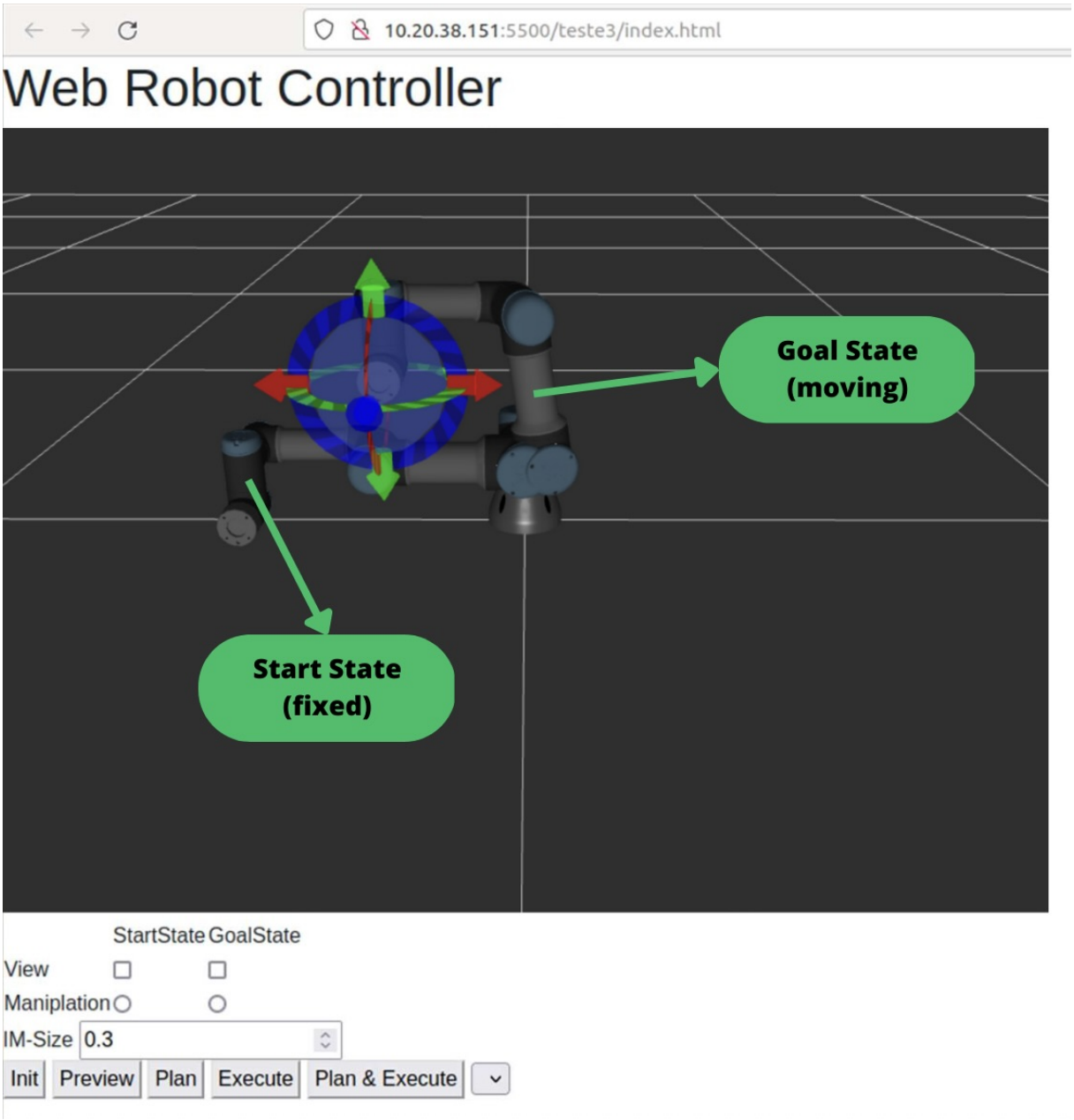

Figure 4.5: Visualization of the robot states in the web application.

or modifying the goal state, the script within the HTML associates these actions with specific topics that will be published to the server computer. The mandatory parameters, such as joint position and move group, were already linked to a topic in the original Visualization RWT project, and some optional parameters were implemented following the same structure to enhance the system's functionality. One such additional feature is

the selector that allows the user to choose which motion planning algorithm to use and compare the differences between them.

Within the HTML code, a select element was added with some typical algorithm options. As seen in Listing 4.3, this select element has a unique ID and stores the value of the algorithm selected by the user. In the JavaScript script, a topic called "planner/update" was created with String-type messages that contain the selected algorithm's value. A function was created to receive this value, include it in the message, and publish it to the corresponding topic. As a result, the server computer receives this message and, with MoveIt! already running, recognizes it as one of the parameters for forming the Motion Planning Request. The system then executes it in the next motion planning cycle.

```
1    planner_pub = new ROSLIB.Topic({
2        ros: joint_ros,
3        name: '/planner/update',
4        messageType: 'std_msgs/String'
5    });
6    [...]
7    function set_planner_callback(selectObject) {
8        var planner = selectObject.value;
9        var msg = new ROSLIB.Message({
10           data : planner
11       });
12       planner_pub.publish(msg);
13   }
14   [...]
15   <select name="set_planner" id="set_planner" onchange="
     set_planner_callback(this)">
16       <option value='RRTConnect'>--Select one--</option>
17       <option value="SBL">SBL</option>
18       <option value="EST">EST</option>
19   [...]
20   </select>
```

Listing 4.3: Linking HTML elements to ROS topics.

The message exchange is facilitated through Rosbridge, which establishes a connection with the ROS Master upon launch. By default, the Master is located on the localhost's port 9090, and the bridge also connects to the web client when the web application is accessed. The ROS Master's major function is to make it possible for individual ROS topic to find one another, tracking publishers and subscribers to topics and services. Peer-to-peer communication is possible once the nodes have located one another.

Once the robot is initialized, the Robot State (service that publishes the state of a robot) is automatically transmitted. Thus, during server launch, the topics necessary for generating the run-time visualization of the remote operation system are specified and sent to the application.

Therefore, the communication was established between the front-end and the back-end. MoveIt! receives the user's input data and sends kinematic commands to the robot in the same manner as at the beginning of the implementation of this work. Meanwhile, the web application receives the necessary data for executing the run-time visualization of the system.

4.6 Enhancement of Functionalities

The final version implementation focused on enhancing user-friendliness of the web application by incorporating visual improvements and adding useful functionalities beneficial to both the host and the user.

The font style and buttons that used HTML standards were replaced with those from the Bootstrap and Font Awesome libraries, which offer a wide range of visual elements to enhance the application's appearance and interactivity. The number of buttons and selectors was reduced and optimized to avoid confusion among users or redundancy in functionality (e.g., init and preview). Moreover, the main page was split into two columns, with the viewer on one side and the command center on the other, to better accommodate horizontally-oriented screens and enable remote operation without requiring the user to scroll the page to access hidden elements due to screen size limitations. Figure 4.6 provides

a comparison between the first and the final version.

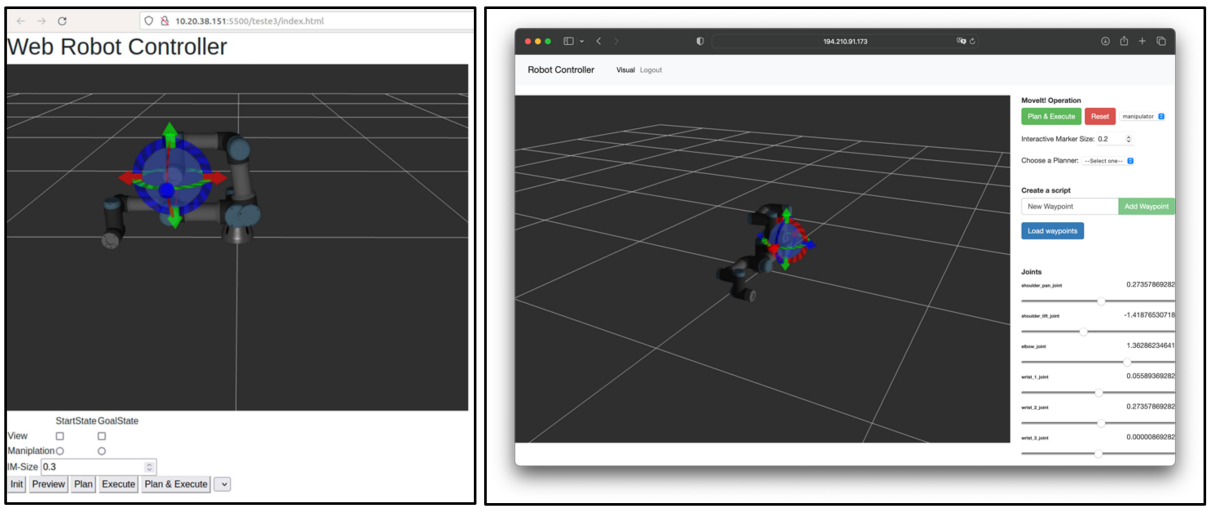

Figure 4.6: On the right the first version and on the left the final version.

To expand the functionality and attract more users, a dedicated section for script creation was implemented, as specified in the system architecture. The HTML code was updated to include a division where users can enter a name and save the current position of the robot (i.e., the value of each joint) in JSON format for later retrieval. The waypoints were created by capturing the user-specified name, the joint names, and their corresponding position values, as shown in the code snippet in Listing 4.4. The File Saver and App libraries were utilized to allow the user to download a file containing all the created waypoints in JSON format, which can be uploaded later to resume work from a previous session.

```
1    var new_wp = {};
2    new_wp.name = wp_name;
3    new_wp.joints = joint_states.name;
4    new_wp.position = joint_states.position;
5    waypoints.push(new_wp);
6    createWaypointButtons();
```

Listing 4.4: Creating the waypoints JSON file.

A Python code was developed to be executed when the system is connected to the

UR3 robot present in CeDRI. The code describes to MoveIt! the dimensions and positions of the worktable in which the robot is placed, thereby preventing collisions between the robot and its environment. This enhances the safety of remote operations and reduces the risk of damage or loss to the laboratory during the operation with the system developed. These dimensions and positions can easily be adapted to other benches and scenarios.

Finally, a significant functionality was added to enhance system security for the host. A login page, was created to provide control over authorized and unauthorized access, as specified in the system architecture for user authentication. The login was implemented using a script outside of HTML, employing the Vue and App libraries. Usernames and passwords are set by the host and cannot be modified by the user, ensuring access is limited to the host's established time-frame.

Chapter 5

Experiments and Results

In this chapter, the tests conducted on the developed system regarding performance, efficiency, and functionality are presented. The main objective is to provide a concise description of each test methodology, present the obtained results, and establish a comprehensive analysis and discussion based on the expected or required outcomes.

5.1 Communication Between Robot and Computer

During the initial implementation phase, the first tests of this work were conducted to evaluate the communication between the robot and the server computer. Two important aspects need to be highlighted: the use of an Unshielded Twisted Pair (UTP) cable for connecting the robot and the computer on the local network, and the configuration of a fixed IP address instead of using DHCP mode.

To assess the establishment of communication, a ping test was performed. The expected outcomes aimed to surpass the recommendations set by network experts, taking into account the wired and local nature of the connection. Specifically, latency values below 10 milliseconds, jitter lower than 15 milliseconds, and packet loss rates below 2% were anticipated.

By executing the ping command from the computer's terminal with the robot's IP address 30 times at different times and days of the week to test different network usage

scenarios, the obtained results were analyzed to evaluate network performance. Latency measurements provided insights into real-time network performance, while jitter values characterized the network's stability over time.

According to the test illustrated in Figure 5.1, the average latency value of 0.132 milliseconds indicated excellent communication speed, while the jitter value of 0.018 milliseconds indicated good network stability. Additionally, the packet loss rate of 0% demonstrated that all transmitted packets were successfully received. These results confirm the correct establishment of the network between the two devices, allowing the exchange of necessary topics and messages for remote operation.

```
bruno@Ubuntu-Bruno:~$ ping 10.20.38.11
PING 10.20.38.11 (10.20.38.11) 56(84) bytes of data.
64 bytes from 10.20.38.11: icmp_seq=1 ttl=64 time=0.144 ms
64 bytes from 10.20.38.11: icmp_seq=2 ttl=64 time=0.144 ms
64 bytes from 10.20.38.11: icmp_seq=3 ttl=64 time=0.125 ms
64 bytes from 10.20.38.11: icmp_seq=4 ttl=64 time=0.167 ms
64 bytes from 10.20.38.11: icmp_seq=5 ttl=64 time=0.124 ms
64 bytes from 10.20.38.11: icmp_seq=6 ttl=64 time=0.111 ms
64 bytes from 10.20.38.11: icmp_seq=7 ttl=64 time=0.145 ms
64 bytes from 10.20.38.11: icmp_seq=8 ttl=64 time=0.125 ms
64 bytes from 10.20.38.11: icmp_seq=9 ttl=64 time=0.116 ms
64 bytes from 10.20.38.11: icmp_seq=10 ttl=64 time=0.122 ms
^C
--- 10.20.38.11 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9194ms
rtt min/avg/max/mdev = 0.111/0.132/0.167/0.018 ms
bruno@Ubuntu-Bruno:~$
```

Figure 5.1: Ping test between the computer and robot.

After successful validation, the next step involved conducting actual remote operation tests using only the server computer and the robot. As detailed in Chapter 4, this initial phase utilized two well-established software packages in ROS, namely MoveIt! and RViz, which are widely recognized for their capabilities in motion planning and operation visualization. In addition to verifying the installation steps and ROS topic exchange between the two devices, the obtained values from the upcoming tests will serve as a benchmark for comparison with the values obtained by the system developed in this work.

Initially, a test was performed to ensure the correct installation of all dependencies,

drivers, and software components, and to verify the successful communication between ROS and the physical robot. To execute this test, the UR3 robot was initiated with the external control program, while RViz and MoveIt! were launched on the server computer with the appropriate configurations for this specific robot manipulator. The expected outcome was that any inputs provided through RViz would be processed by MoveIt!'s motion planning capabilities, enabling robotic operation through the computer.

The visualization in RViz confirmed the successful transmission of the Robot State topic from the robot to the computer. To verify this, a straightforward test was performed by manually manipulating the robot using the button that enables the joints to move freely under external force. As a result, the new initial state was accurately displayed in RViz, reflecting the robot's updated position.

To test the transmission of the Motion Parameters topic from the computer to the robot, the MoveIt! interface in RViz was used to set some parameters of the Motion Plan Request. These parameters included a maximum planning time of 5 seconds, a speed of 50%, and the default planner algorithm (RRT#). The goal state was randomly changed multiple times, and the MoveIt! was executed. The results showed that in all valid positions, the physical robot successfully reached the specified goal state using the kinematic commands provided by MoveIt!, as illustrated in Figure 5.2. However, in cases where there was a collision detection or the position was outside the robot's range, the execution reported a failure. These findings confirmed the robustness of the message transmission from the computer to the robot and demonstrated the capability of controlling the robot through the computer using these software tools.

With the successful establishment and testing of communication, the attention turned to establishing reference values for future performance comparison of the developed system. To conduct this test, the previous setup was maintained, ensuring that the robot and server computer remained connected to the local network via a cable. Fixed start and goal states were defined and described in Table 5.1, while their graphical representation can be found in Figure 5.3. These predefined states were chosen to ensure consistency and enable comparability in future tests. The movement from the start state to the goal

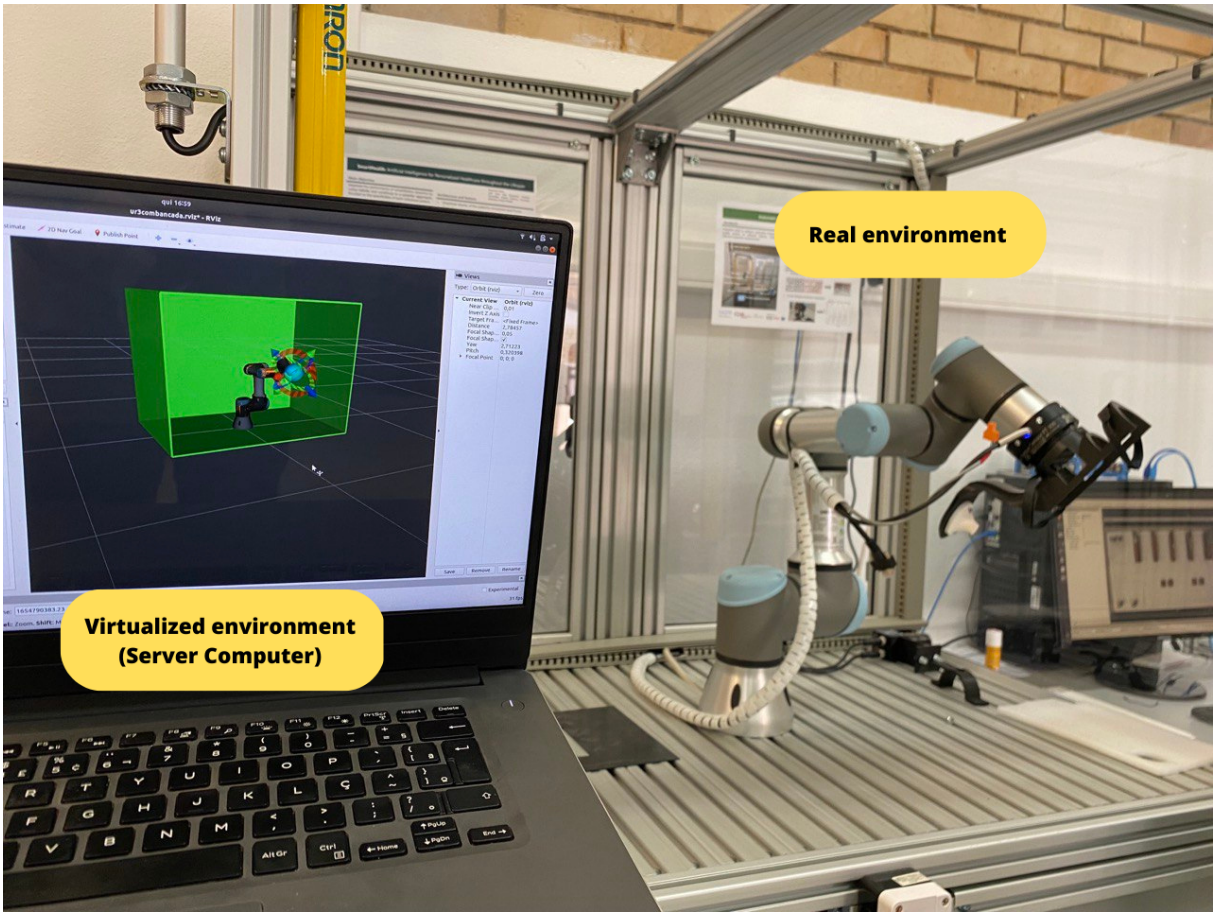

Figure 5.2: Real robot reaching the goal state of the virtualized robot.

state was repeated a total of 30 times, and the resulting execution times were recorded (see Appendix B).

The average execution time for the 30 movements was 4.13 seconds, with a standard deviation of 6.8 milliseconds. These values serve as the benchmark for execution time in validated software. It is expected that the tests performed with the developed system will achieve execution times that are comparable to or approximate these values.

5.2 Communication Between Robot and User

The communication tests performed on the developed system in conjunction with the robot aimed to evaluate the execution time, performance, and validation of the proposed

Table 5.1: Values of each joint for the fixed start and goal states of the movement performed in the tests.

Joints	**Start State (rad)**	**Goal State (rad)**
Shoulder Pan	3.1527023241568957	1.5818839590503053
Shoulder Lift	-1.6334594117795955	-2.2308669896427142
Elbow	1.6617515310048647	2.4116209521937453
Wrist 1	-1.6237434264053565	-2.4113348158936576
Wrist 2	-1.560491223527432	-1.5584844970696974
Wrist 3	1.5823676848805528	0.024211865946034195

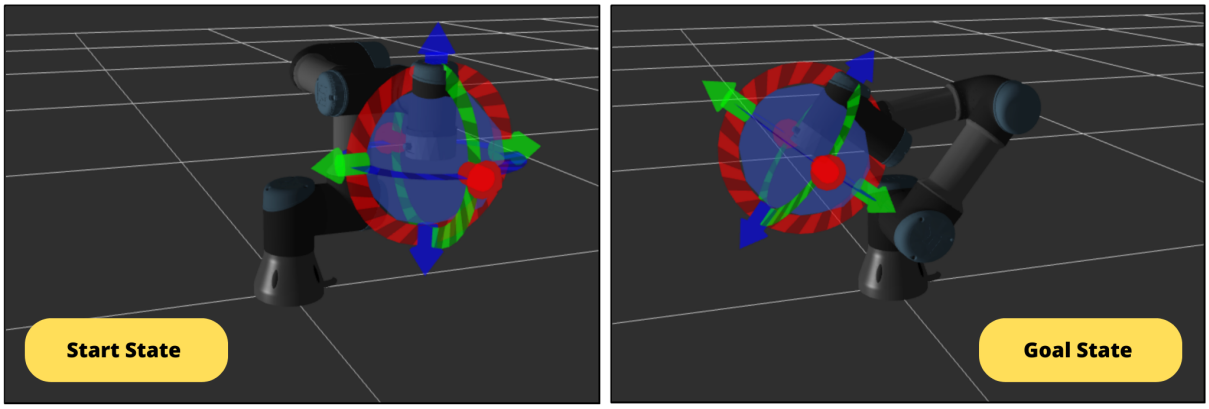

Figure 5.3: Graphical representation of the fixed start and goal states of the movement performed in the tests.

solution. These tests involved the operational functionality of the physical robot, the server computer, and the user's computer, ensuring comprehensive testing of all components within the proposed architecture.

The execution time test was performed under two different conditions: one where the user's computer was connected to the same local network as the server computer and the robot, and another where the user's computer was connected via Virtual Private Network (VPN). The robot's movement replicated the same start state and goal state from the previous test, and the test was repeated 30 times. The results of these tests are presented in Appendix B.

The execution time test, which previously considered only the processing time of MoveIt!, kinematic command transfer to the robot, and movement to the goal state, now also includes the time required for sending parameters to construct the Motion Plan

Request between the computers. With a local connection, the average execution time was 4.41 seconds, with a standard deviation of 14.2 milliseconds. With a VPN connection, the average execution time was 4.92 seconds, with a standard deviation of 21.8 milliseconds.

Table 5.2 presents a comparison between the established reference, which involves direct remote control of the robot through the server computer, and the recent tests incorporating the user's computer, both in a local connection and via VPN. The results demonstrate a minimal difference in execution time, with a variation of less than 0.8 seconds. Importantly, this negligible difference remains imperceptible to the end user and does not introduce substantial delays in communication or remote operation, once the control of the movement of the joints is still done by the robot's computer while the system sends commands to be executed by it. Consequently, it can be inferred that the proposed architecture and the developed system perform effectively, delivering execution times comparable to a direct computer-robot connection, even when inputs originate from an external computer.

Table 5.2: Average execution time comparing connections.

	Direct Connection	**Local Network**	**VPN Connection**
Avg. Execution Time	4.13 s	4.41 s	4.92 s
Standard Deviation	0.007 s	0.014 s	0.022 s

In terms of system performance, three tests were conducted to evaluate different aspects. The first test focused on computational resource utilization during system operation. The second test compared the performance of three commonly used web browsers to determine if system performance is influenced by this parameter. Finally, the third test evaluated the run-time 3D visualization performance.

To evaluate the utilization of computational resources, the experiment utilized the same setup described in Chapter 4, comprising a computer equipped with an Intel Core i7-7500U processor and 8GB of RAM. Figure 5.4 illustrates the activity monitor of this computer functioning as the server, displaying its status prior to executing the system and while the system is running.

Upon analyzing the activity monitor graphs, it becomes apparent that the Central

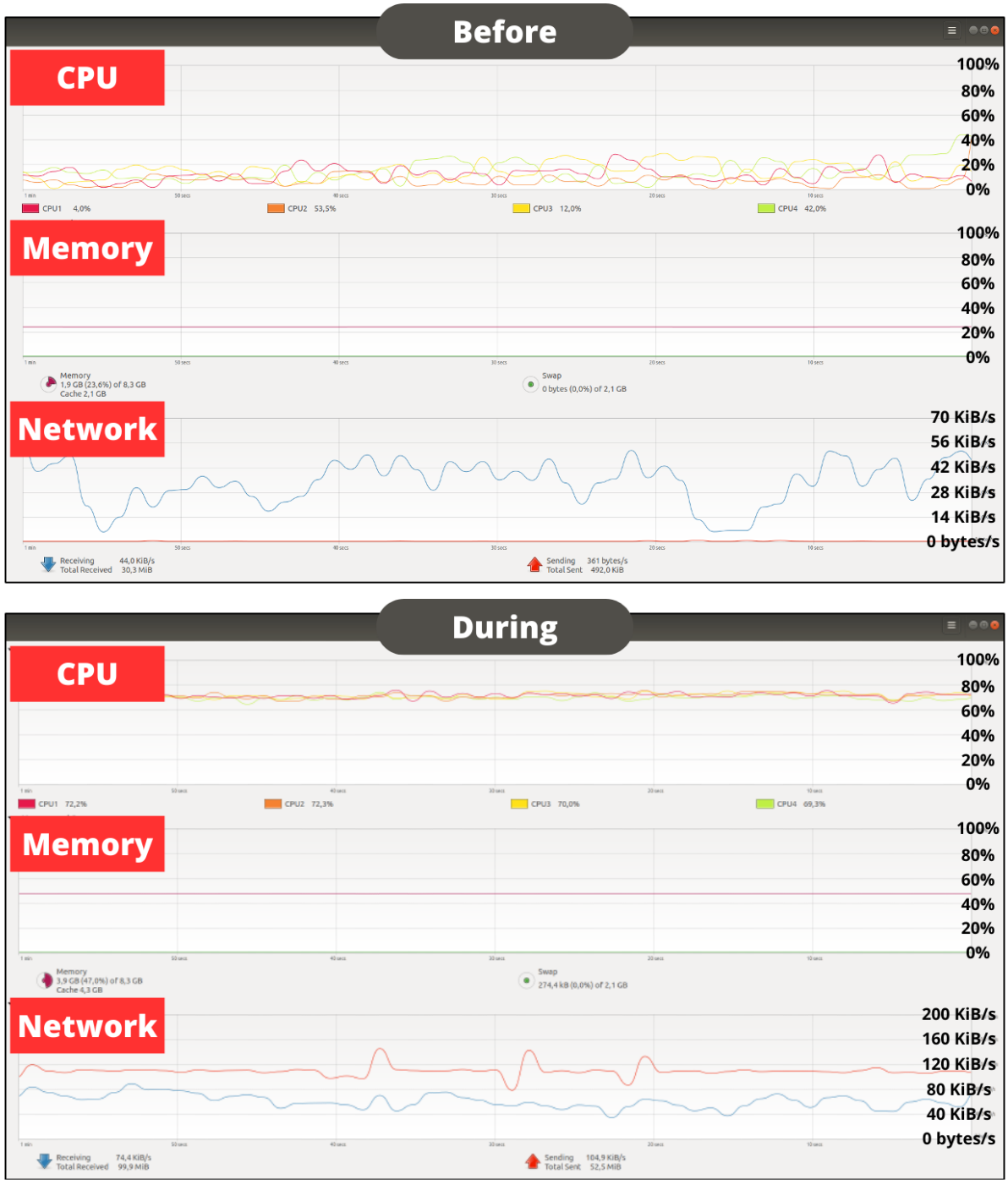

Figure 5.4: Comparison of computer resource usage before and during system execution.

Processing Unit (CPU) usage maintained an average of 30% prior to executing the remote operation system, which subsequently increased to 70% during execution. Similarly,

the RAM usage experienced an increase from 24% to 47%. Notably, the network usage demonstrated significant changes as well, with the download rate escalating from 44.0 kB/s to 74.4 kB/s, while the upload rate surged from 361 B/s to 104.9 kB/s. This way, the hardware setup for the server computer is enough for running the developed system.

In this evaluation, it was also examined the utilization of computational resources by the user. One of the primary requirements was to ensure that the server computer carried the majority of the processing load while placing minimal demands on the user's computer. The intention was to create a system that could be inclusive and accommodate computers with lower computational capabilities. This test involved utilizing the same computer from the last test as the user, accessing the system through the Google Chrome browser. Through the browser's task manager, during system execution, the CPU usage was measured at approximately 27%, with a corresponding memory usage of around 57 MB.

These results demonstrate that the system successfully fulfilled the requirement of minimizing resource consumption on the user's computer, ensuring that it only needed to handle the necessary tasks of displaying graphics and running the web browser. These findings also validate the effectiveness of the system's architecture, with the bulk of the processing load concentrated on the server computer. This design choice guarantees a more inclusive experience for users with varying computational resources, as the demanding computational tasks are offloaded to the server, allowing even less powerful computers to effectively participate in the system.

The following test was conducted to examine whether the user's choice of web browser influenced the performance of the system. In the previous test, Google Chrome was randomly selected as the browser, and now it is essential to evaluate whether using different browsers affects the remote operation. To investigate this, three popular browsers were selected, namely Google Chrome, Mozilla Firefox, and Microsoft Edge.

During this test, the same start state and goal state used in the execution time test were repeated 20 times for each browser. The objective was to measure the average execution time and its standard deviation. It was anticipated that the system would

remain largely unaffected by the choice of browser. However, minor variations in the execution time were deemed acceptable due to inherent characteristics specific to each browser, such as memory usage, operating system optimization, and processing speed. The results obtained from this experiment are presented in Table 5.3 (see Appendix B for complete results).

Table 5.3: Average execution time comparing browsers.

	Google Chrome	Mozilla Firefox	Microsoft Edge
Avg. Execution Time	4.42 s	4.44 s	4.41 s
Standard Deviation	0.010 s	0.008 s	0.009 s

The results indicate that the choice of web browser does not have a substantial impact on the system's performance in terms of execution time. The maximum difference observed, which amounted to only 30 milliseconds in average execution time, is virtually imperceptible to the end user. These findings provide strong evidence that the system remains robust and unaffected by the browser selected for accessing it. The only requirement is that the browser supports WebGL to ensure accurate visualization generation. As a result, users are free to choose their preferred browser without concerns about significant performance disparities, offering them flexibility and convenience.

Finally, the last test in this phase aimed to evaluate the performance of the 3D viewer integrated into the remote operation application. The run-time visualization of the system plays a crucial role in enabling the operation, allowing users to verify the robot's movement, track its trajectory, and observe its current state. This aspect is highlighted as one of the system requirements in the system architecture.

For this test, the same fixed start and goal states from Table 5.1 were used, and some timestamps were recorded during the robot's movement to identify any potential delays in the visualization. The expectation was that the timestamps of the real robot's position and its corresponding visualization would be closely aligned, ensuring minimal discrepancies that would be practically imperceptible to the user. The results of the timestamps captured at some positions during the movement are illustrated in Figure 5.5.

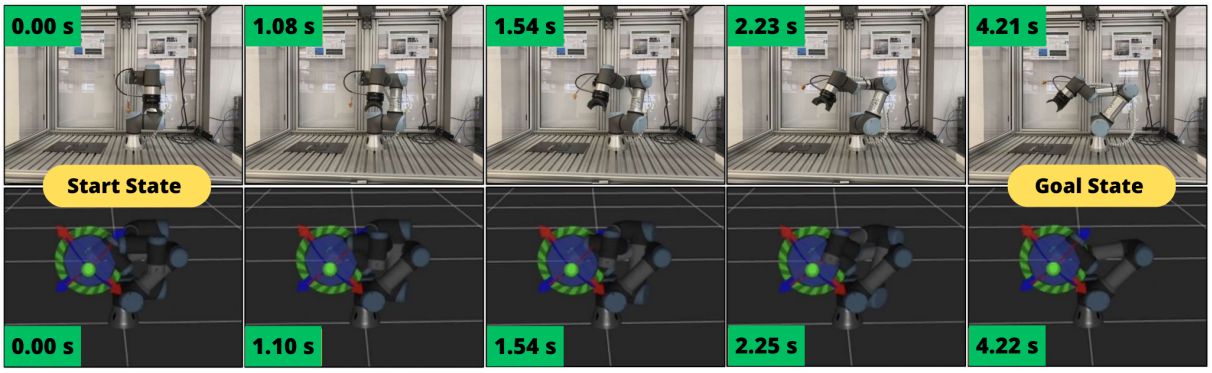

Figure 5.5: Robot trajectory and timestamps during the run-time test.

Based on these results, it was noted that the delay between the real movement and the visualization is less than 30 milliseconds, rendering it imperceptible to the user and meeting the run-time operation requirement. It should be highlighted that in slower connections, there is a slight reduction in the refresh rate of the visualization. Nonetheless, even under these circumstances, the system continues to function in run-time, enabling seamless remote operation of the robotic manipulator.

5.3 Communication Between Simulation and User

As previously stated in Chapter 3, the developed system was designed to support hybrid usage, meaning that it could connect to either a real robot (as demonstrated in the previous tests) or a simulated robot (to be evaluated in this section). The Gazebo software, a widely used simulation tool within the ROS framework, was selected for simulating the robotic manipulator. To verify the system's consistent performance across different scenarios, an operational test was conducted, ensuring its seamless functionality in both real and simulated environments.

In this test, the system was executed in a way that replicated the process used when connected to the real robot, with the exception of the connection. The back-end was executed on the server computer, which also ran the UR3 simulation in Gazebo, while a separate computer was used to access the web application as a user. Random movements

were executed, and it was verified whether it was possible for the user to control the simulation on the server via the application.

The results of this test demonstrated successful remote control and operation of the simulated robot, with all user inputs accurately replicated in the server computer's simulation. This outcome was expected, as the simulator provided the necessary topics (Robot State) for MoveIt! and web visualization, and received kinematic topics (Move Plan) to execute movements, mirroring the behavior observed when connected to a real robot. Consequently, no modifications were needed in the developed system to accommodate simulated or real robots.

Furthermore, it is worth emphasizing that there were no noticeable differences in execution time. The operation proceeded seamlessly, resembling the experience of controlling a physical robot, with the server computer handling the processing load while maintaining low resource utilization on the user's computer. This reinforces the system's versatility and efficiency in facilitating remote robotic operations.

Still using the simulated environment, additional tests were conducted to assess the system's versatility. Apart from the UR3 robot, which was selected based on its physical availability for this work, other robots were integrated into the system through simulation, including UR5, Denso VS-060, and Franka Emika Panda. The system has the capability to operate any robot supported by MoveIt! and equipped with the necessary Robot Description packages provided by the respective manufacturers. Figure 5.6 showcases the successful remote operation of a simulated UR5 robot, where control commands from the user's computer are transmitted to the server computer for execution.

The final simulation test focused on end-effector control. Previous tests evaluated only the control of the robot's individual joints, performing movements within the main group that constitutes the robot. The end-effector, which must also be described in the Robot Description package for visualization and control, resides in a separate group that can be selected using the group selector in the web application. The end-effector control test utilized a simulated UR5 robot equipped with a Robotiq gripper, as depicted in Figure 5.7.

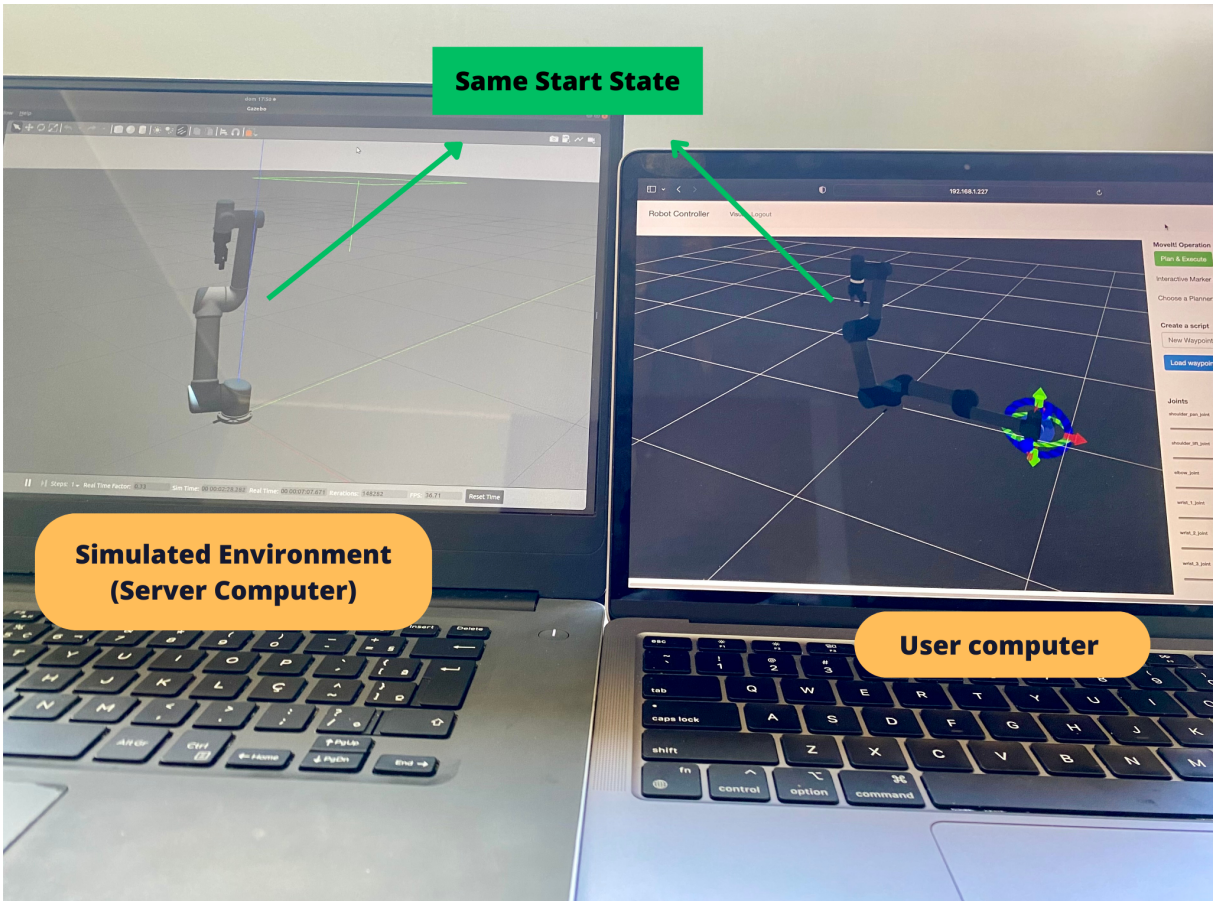

Figure 5.6: User operating the robot simulated by the server computer.

Selecting a new group with the group selector in the web application allows the interactive marker and the joint value sliders to exclusively control the joints within that specific group. As a result, the system enabled satisfactory control of the gripper, as well as the control achieved for other parts of the robotic manipulator.

5.4 Functionality Tests

The subsequent tests were conducted to assess other functionalities provided by the web application. These include the system access functionality (credentials, simultaneous access, and access from different devices) and the extra features functionality (waypoints and planners).

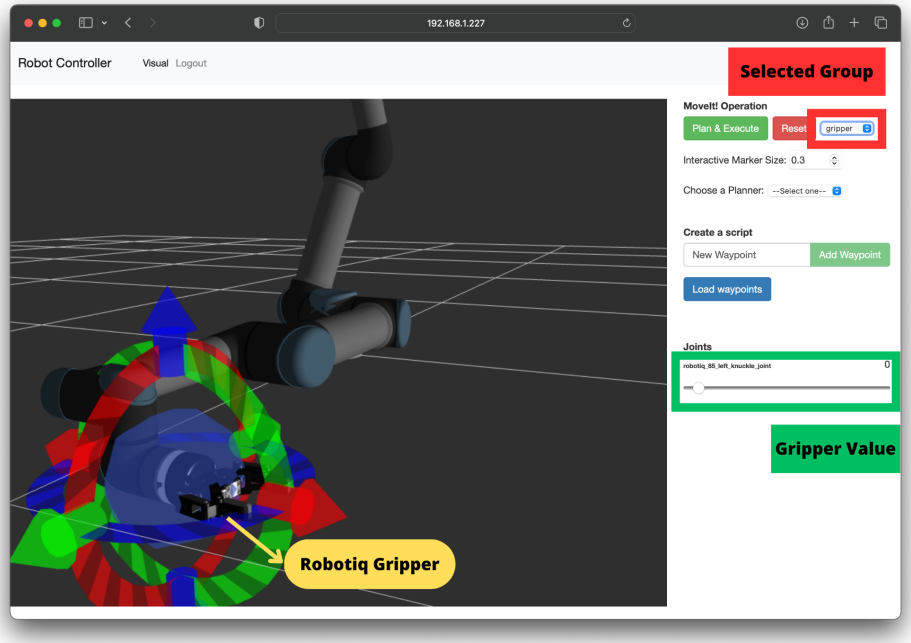

Figure 5.7: End-effector control test.

The utilization of credentials for system access was implemented as a security measure to prevent unauthorized or unknown entry. To evaluate this functionality, various access scenarios were simulated, such as incorrect passwords, incorrect usernames, missing fields, and valid credentials. As illustrated in Figure 5.8, any incompatible access credentials were promptly rejected, accompanied by an error message, thereby denying system entry. Furthermore, upon successful login, the server computer's terminal generates a notification indicating that a client has subscribed to the Rosbridge topics.

Regarding simultaneous access by multiple users, a test was conducted with four devices accessing the system. It is important to highlight that a single back-end can only control a single robotic manipulator, thus preventing the concurrent control of a real robot and a simulation, or multiple simulations. Consequently, when multiple users access the system simultaneously, they share access to the same robot. The credential functionality for access control already provides security measures for simultaneous access. However, in certain scenarios, multiple accesses are desired, such as during robotics classes or co-working sessions. Having this in mind, the test with four devices simultaneously ensured

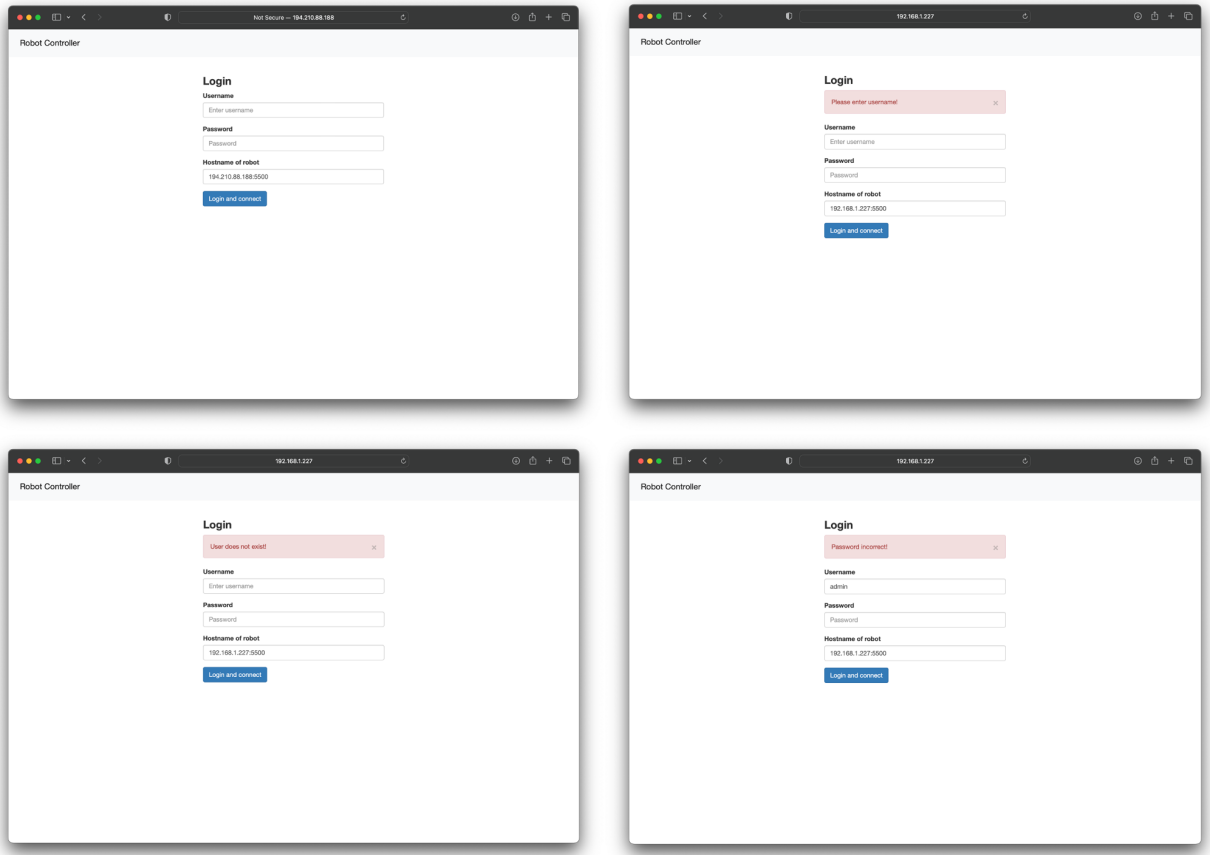

Figure 5.8: Testing different cases of unauthorized access.

that the system is mirrored for all logged-in users, allowing everyone to contribute by modifying parameters, which are shared with everyone. This approach ensures that only one set of instructions is sent to the server computer at a time, preventing any potential damage to the manipulator, and it fosters collaborative work. Therefore, this serves as an additional security measure.

Finally, the aspect of accessing the system from different devices is considered. During the previous performance tests, the system was accessed multiple times from computers with different operating systems, such as Windows, MacOS, and Ubuntu. However, this particular test focuses on assessing access from devices beyond computers, specifically smartphones. It is anticipated that the system can be effectively operated from any device that meets the architectural requirements, encompassing not only computers but also smartphones and tablets.

As a result, it was observed that the system could be operated through a smartphone in a manner similar to the computer, although some challenges were expected due to the smaller screen size. Figure 5.9 presents a visual comparison between two screenshots, one captured on a computer and the other on a smartphone.

Figure 5.9: Web application accessed from a computer and a smartphone.

The final set of tests focused on exploring additional features that enhance the system's functionality, making it more engaging and appealing to users. Specifically, the waypoints feature was tested to ensure the proper operation of its add, save, and load functions. During the test, the system was accessed, and the remote operation of the UR3 robot was initiated. Two waypoints were added, and movements between them were executed. To preserve the progress made during the session, the waypoints were saved before logging out, resulting in the download of a JSON file. In subsequent access to the system, the saved file was successfully loaded, allowing for the restoration of the previously defined waypoints. Figure 5.10 provides a visual representation of the waypoints added in the web application and demonstrates the organization of the downloaded file. This functionality proved to be valuable in creating movement scripts between specific points (e.g., A-B-A or A-B-C), while the save and load features ensured the preservation of progress between different user sessions.

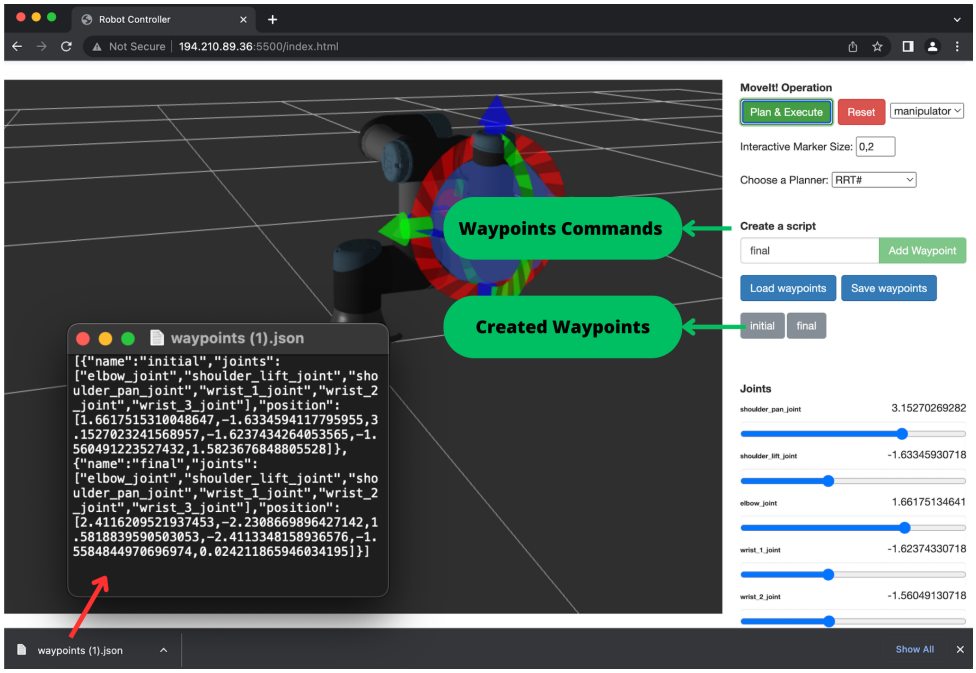

Figure 5.10: Creating and downloading waypoints.

The functionality of selecting the optimization algorithm for motion planning, as depicted in Figure 5.11, was tested by evaluating the processing time of MoveIt! until a feasible trajectory was generated. A maximum time limit of 10 seconds was set, and the same movement, as described in Table 5.1, was performed from the start state to the goal state. The processing times for each algorithm are presented in Table B.3. This feature proved valuable as it allows users to experiment and identify the most suitable algorithm for a specific motion. Among the algorithms tested in this experiment, the default "RRT#" algorithm used by MoveIt! achieved the best performance (i.e. fewer number of states/iterations until reaching the solution) for the specific movement.

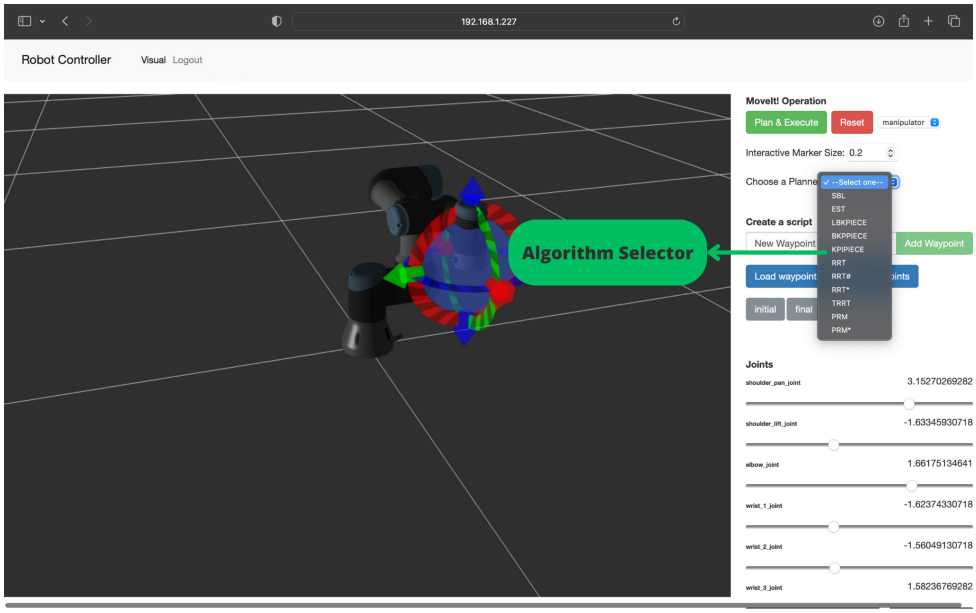

Figure 5.11: Selection of motion planning optimization algorithms.

Chapter 6

Conclusion and Future Work

This research endeavors to address the issue of restricted accessibility to robotics and the escalating need for skilled professionals in the field. The primary objective was to conceive a solution for remotely operating robotic manipulators, setting it apart from prevailing simulators and platforms available in the market. Notably, these conventional options frequently impose steep learning curves, demand substantial computational resources, and are commercially oriented.

In order to broaden the study, research, and development in this field and cater to a wide range of interested individuals, this work employed exclusively open-source resources for the construction of the system. The chosen resources include the ROS framework, MoveIt!, and the RWT Project. By relying on open-source tools, this approach ensures accessibility and encourages collaboration within the robotics community. Furthermore, from the user's perspective, the system only necessitates access to a web application, thereby minimizing the execution prerequisites and simplifying the overall user experience.

The integration of ROS played a crucial role in this project, enabling the integration of numerous robots and facilitating robust communication capabilities. Nonetheless, one persistent challenge faced by both robotics students and educators pertains to the steep learning curve associated with ROS. With its multiple versions and peculiarities, navigating the intricacies of ROS can impede or demotivate teaching and research efforts. In light of this, the web application was designed to enable users to interact with ROS

effortlessly, unaware of its presence in the background. By automatically connecting user inputs to ROS topics and messages, the application obviates the need for prior knowledge or explicit ROS-related actions.

MoveIt! has demonstrated remarkable efficacy in executing motion planning tasks, which involve substantial computational processing and are performed externally to the user's computer, specifically on the system's host computer.

The RWT Project offered some functionalities that played a role in the system's functioning. These included establishing a client-server connection and developing a front-end with run-time process visualization. Leveraging these capabilities, a user-friendly interface was created, greatly facilitating the overall system development process.

The outcomes of performance, efficiency, and functionality tests have yielded promising results, showcasing imperceptible communication delays, seamless run-time visualization, and the provision of practical and user-friendly features.

In this way, the system can be used in classrooms, research centers, laboratories, and even home offices to practice robotics, for example, pick-and-place operations, analysis of trajectory, analysis of the number of Degrees of Freedom, and collision avoidance.

Therefore, the predefined objectives and requirements of this project have been accomplished, and the system's replicability is facilitated by its generic architecture and compatibility with diverse robot types. Consequently, the system effectively achieves its primary goal of enhancing accessibility to the field of robotics.

Future work is directed towards addressing security concerns, implementing new functionalities, and conducting impactful research.

In the realm of security, it is necessary to enhance and incorporate cyber-security parameters to safeguard against malicious attacks. By doing so, it would be feasible to host the back-end on an Internet-accessible server, facilitating system access without the requirement of a VPN and allowing the utilization of a public IP.

The new functionalities of the system aim to enhance user engagement and attract new users by introducing innovative features. These enhancements include the ability for users to define and incorporate custom scenarios, enabling interaction with and avoidance

of objects and obstacles within those scenarios. Moreover, users can have the flexibility to integrate their own algorithms (e.g., code in Python or C) into the system, enabling advanced robotic manipulator operations beyond the basic point-to-point waypoint movements. Additionally, the system can be extended to incorporate sensor data acquisition from the user, promoting enhanced HRC through inputs from some devices such as joysticks, Kinect, and Virtual Reality (VR) interfaces.

In the realm of research, it is anticipated that the impact on STEM education and research centers, which embrace the utilization of this developed system, will be examined. This assessment aims to validate the system's functionalities and collect additional suggestions to further enhance the value of remote operation of robotic manipulators for educational and research endeavors.

Bibliography

[1] C. Yang, J. Luo, C. Liu, M. Li, and S.-L. Dai, "Haptics electromyography perception and learning enhanced intelligence for teleoperated robot," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1512–1521, 2018.

[2] J. A. Garcıa-Esteban, L. Piardi, P. Leitão, B. Curto, and V. Moreno, "An interaction strategy for safe human co-working with industrial collaborative robots," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, IEEE, 2021, pp. 585–590.

[3] A. W. Barbosa, "Sistema remoto para controle de robôs móveis via web," Ph.D. dissertation, Universidade de São Paulo, 2016.

[4] S. Sirouspour, "Modeling and control of cooperative teleoperation systems," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1220–1225, 2005.

[5] N. Shahzad, T. Chawla, and T. Gala, "Telesurgery prospects in delivering healthcare in remote areas," *The Journal of the Pakistan Medical Association*, vol. 69, no. Supl. 1, S69, 2019.

[6] N. G. Hockstein, C. Gourin, R. Faust, and D. J. Terris, "A history of robots: From science fiction to surgical robotics," *Journal of robotic surgery*, vol. 1, pp. 113–118, 2007.

[7] R. Clarke, "Asimov's laws of robotics: Implications for information technology-part i," *Computer*, vol. 26, no. 12, pp. 53–61, 1993.

[8] I. Abeykoon and X. Feng, "A forensic investigation of the robot operating system," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, 2017, pp. 851–857.

[9] R. Boboc, H. Moga, and D. Talaba, "A review of current applications in teleoperation of mobile robots," *Bulletin of the Transilvania University of Brasov. Engineering Sciences. Series I*, vol. 5, no. 2, p. 9, 2012.

[10] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl, "Rotex-the first remotely controlled robot in space," in *Proceedings of the 1994 IEEE international conference on robotics and automation*, IEEE, 1994, pp. 2604–2611.

[11] M. Ghodoussi, S. E. Butner, and Y. Wang, "Robotic surgery-the transatlantic case," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 2, 2002, pp. 1882–1888.

[12] H.-X. Zhou, Y.-H. Guo, X.-F. Yu, *et al.*, "Clinical characteristics of remote zeus robot-assisted laparoscopic cholecystectomy: A report of 40 cases," *World journal of gastroenterology: WJG*, vol. 12, no. 16, p. 2606, 2006.

[13] J. Marescaux, J. Leroy, M. Gagner, *et al.*, "Transatlantic robot-assisted telesurgery," *Nature*, vol. 413, no. 6854, pp. 379–380, 2001.

[14] J. Marescaux, J. Leroy, F. Rubino, *et al.*, "Transcontinental robot-assisted remote telesurgery: Feasibility and potential applications," *Annals of surgery*, vol. 235, no. 4, pp. 487–492, 2002.

[15] G. Aceto, V. Persico, and A. Pescapé, "A survey on information and communication technologies for industry 4.0: State-of-the-art, taxonomies, perspectives, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3467–3501, 2019.

[16] K. Yaovaja and J. Klunngien, "Teleoperation of an industrial robot using a non-standalone 5g mobile network," in *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, IEEE, 2019, pp. 320–323.

[17] J. Zheng, Y. Wang, J. Zhang, *et al.*, "5g ultra-remote robot-assisted laparoscopic surgery in china," *Surgical Endoscopy*, vol. 34, pp. 5172–5180, 2020.

[18] J. Luo, W. He, and C. Yang, "Combined perception, control, and learning for teleoperation: Key technologies, applications, and challenges," *Cognitive Computation and Systems*, vol. 2, no. 2, pp. 33–43, 2020.

[19] P. M. Kebria, A. Khosravi, S. Nahavandi, P. Shi, and R. Alizadehsani, "Robust adaptive control scheme for teleoperation systems with delay and uncertainties," *IEEE transactions on cybernetics*, vol. 50, no. 7, pp. 3243–3253, 2019.

[20] Y. Su, X. Chen, T. Zhou, C. Pretty, and G. Chase, "Mixed reality-integrated 3d/2d vision mapping for intuitive teleoperation of mobile manipulator," *Robotics and Computer-Integrated Manufacturing*, vol. 77, p. 102 332, 2022.

[21] W.-t. Law, K.-s. Li, K.-w. Fan, W.-h. Ko, T. Mo, and C.-k. Poon, "Two-way human-robot interaction in 5g tele-operation," in *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, IEEE, 2022, pp. 1198–1199.

[22] A. A. Ateya, A. Muthanna, A. Vybornova, *et al.*, "Model mediation to overcome light limitations—toward a secure tactile internet system," *Journal of Sensor and Actuator Networks*, vol. 8, no. 1, p. 6, 2019.

[23] U. I. Atmaca, C. Maple, G. Epiphaniou, *et al.*, "Challenges in threat modelling of new space systems: A teleoperation use-case," *Advances in Space Research*, vol. 70, no. 8, pp. 2208–2226, 2022.

[24] H. Takanashi, A. Kosugi, K. Teranishi, T. Mizuya, K. Abe, and K. Kogiso, "Cyber-secure teleoperation with encrypted four-channel bilateral control," *arXiv preprint arXiv:2302.13709*, 2023.

[25] N. V. Morze and O. V. Strutynska, "Model of the competences in educational robotics," in *Proceedings of the symposium on advances in educational technology, aet*, 2020.

[26] A. A. Shahroom and N. Hussin, "Industrial revolution 4.0 and education," *International Journal of Academic Research in Business and Social Sciences*, vol. 8, no. 9, pp. 314–319, 2018.

[27] J. Solis and A. Takanishi, "Practical issues on robotic education and challenges towards roboethics education," in *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, IEEE, 2009, pp. 561–565.

[28] L. Farcas and O. Caltun, "Challenges and perspectives in education for new technologies and robotics," in *Proc. of Fifth Int. Conf. on Adult Education*, 2018, pp. 679–686.

[29] G. Stein and A. Lédeczi, "Enabling collaborative distance robotics education for novice programmers," in *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2021, pp. 1–5.

[30] N. Correll and D. Rus, "Peer-to-peer learning in robotics education: Lessons from a challenge project class," *Computers in Education Journal*, vol. 1, no. 3, pp. 60–66, 2010.

[31] S. Tselegkaridis and T. Sapounidis, "Simulators in educational robotics: A review," *Education Sciences*, vol. 11, no. 1, p. 11, 2021.

[32] J. Badillo, G. Londino-Smolar, and P. Savvides, "7 things you should know about virtual labs," 2020.

[33] V. Potkonjak, M. Gardner, V. Callaghan, *et al.*, "Virtual laboratories for education in science, technology, and engineering: A review," *Computers & Education*, vol. 95, pp. 309–327, 2016.

[34] L. Gomes and S. Bogosyan, "Current trends in remote laboratories," *IEEE Transactions on industrial electronics*, vol. 56, no. 12, pp. 4744–4756, 2009.

[35] F. A. Candelas, S. T. Puente, F. Torres, F. G. Ortiz, P. Gil, and J. Pomares, "A virtual laboratory for teaching robotics," *complexity*, vol. 1, no. 10, 2003.

[36] P. Abreu, M. R. Barbosa, and A. M. Lopes, "Virtual experiment for teaching robot programming," in *2014 11th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, IEEE, 2014, pp. 395–396.

[37] W. Wiedmeyer, M. Mende, D. Hartmann, R. Bischoff, C. Ledermann, and T. Kroger, "Robotics education and research at scale: A remotely accessible robotics development platform," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 3679–3685.

[38] M. Hernández-de-Menéndez, A. Vallejo Guevara, and R. Morales-Menendez, "Virtual reality laboratories: A review of experiences," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 13, pp. 947–966, 2019.

[39] J. Jurc, M. Sterbak, and M. Kontsek, "Virtual laboratories and their usage in university environment," in *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, IEEE, 2020, pp. 260–265.

[40] S. K. Esche, C. Chassapis, J. W. Nazalewicz, D. J. Hromin, *et al.*, "An architecture for multi-user remote laboratories," *Dynamics (with a typical class size of 20 students)*, vol. 5, no. 6, 2003.

[41] H. Liu and L. Wang, "Remote human–robot collaboration: A cyber–physical system application for hazard manufacturing environment," *Journal of manufacturing systems*, vol. 54, pp. 24–34, 2020.

[42] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[43] *Robot web tools*. [Online]. Available: `https://robotwebtools.github.io/`.

[44] R. Toris, J. Kammerl, D. V. Lu, *et al.*, "Robot web tools: Efficient messaging for cloud robotics," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2015, pp. 4530–4537.

[45] G. A. Casan, E. Cervera, A. A. Moughlbay, J. Alemany, and P. Martinet, "Ros-based online robot programming for remote education and training," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 6101–6106.

[46] C. Gravier, J. Fayolle, B. Bayard, M. Ates, and J. Lardon, "State of the art about remote laboratories paradigms–foundations of ongoing mutations.," *International Journal of Online Engineering*, vol. 4, no. 1, 2008.

[47] M. Arruat, L. Fernandez, S. Jackson, *et al.*, "Front-end software architecture," in *ICALEPCS*, vol. 7, 2007, p. 310.

[48] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.

[49] S. M. La Valle, "Motion planning," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 108–118, 2011.

[50] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: Ros for non-ros users," in *Robotics Research: The 15th International Symposium ISRR*, Springer, 2017, pp. 493–504.

[51] [Online]. Available: `https://moveit.ros.org/robots/`.

[52] [Online]. Available: `http://wiki.ros.org/melodic`.

[53] UniversalRobots, *Universal robots ros driver*. [Online]. Available: `https://github.com/UniversalRobots/Universal_Robots_ROS_Driver`.

[54] [Online]. Available: `http://wiki.ros.org/ur_description`.

[55] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[56] Osrf, *Rvizweb: Rviz on the browser*. [Online]. Available: `https://github.com/osrf/rvizweb`.

[57] Rapyuta-Robotics, *Zethus: Realtime robot data visualization in the browser*. [Online]. Available: `https://github.com/rapyuta-robotics/zethus`.

[58] R. W. Tools, *Usage of rwt_moveit*. [Online]. Available: `http://docs.ros.org/en/indigo/api/rwt_moveit/html/index.html`.

Appendix A

External Links

A primer containing installation and usage instructions, along with all the necessary and developed files in this work, can be accessed through this link.

The video showcasing the execution of the movement, which was utilized in multiple tests, can be accessed through this link.

Appendix B

Expanded Results

Table B.1: Expanded runtime test result per browser.

Google Chrome (s)	**Mozilla Firefox (s)**	**Microsoft Edge (s)**
4.43	4.45	4.41
4.42	4.45	4.42
4.42	4.45	4.42
4.43	4.44	4.41
4.42	4.44	4.41
4.42	4.45	4.42
4.41	4.44	4.42
4.41	4.44	4.41
4.41	4.43	4.42
4.42	4.43	4.41
4.41	4.43	4.42
4.41	4.44	4.42
4.44	4.44	4.41
4.42	4.44	4.41
4.40	4.45	4.41
4.41	4.45	4.40
4.43	4.44	4.40
4.41	4.44	4.40
4.40	4.43	4.40
4.42	4.43	4.39

Table B.2: Expanded runtime test result per connection.

Direct Connection (s)	**Local Network (s)**	**VPN Connection (s)**
4.12	4.43	4.94
4.13	4.42	4.95
4.13	4.43	4.93
4.14	4.42	4.93
4.13	4.41	4.93
4.13	4.40	4.94
4.12	4.41	4.92
4.13	4.41	4.90
4.13	4.40	4.89
4.13	4.39	4.90
4.12	4.39	4.93
4.12	4.40	4.87
4.13	4.40	4.89
4.14	4.42	4.89
4.14	4.41	4.92
4.13	4.40	4.93
4.13	4.40	4.95
4.13	4.41	4.93
4.12	4.42	4.92
4.12	4.44	4.90
4.13	4.44	4.91
4.13	4.42	4.90
4.14	4.40	4.88
4.14	4.41	4.88
4.13	4.43	4.90
4.13	4.41	4.92
4.12	4.40	4.93
4.13	4.42	4.94
4.12	4.44	4.92
4.12	4.42	4.93

Table B.3: Comparison between trajectory optimization algorithms.

Planner ID	**Processing Time**	**States to Solution**	**Result**
SBL	>10	-	Failed
EST	0.003929	7	Succeeded
LBKPIECE	>10	-	Failed
BKPPIECE	>10	-	Failed
KPIPIECE	>10	-	Failed
RRT	0.026085	33	Succeeded
RRT#	0.011241	5	Succeeded
RRT*	10.002444	3866	Succeeded
TRRT	0.002908	28	Succeeded
PRM	10.003154	3217	Succeeded
PRM*	10.003888	1929	Succeeded