UNIVERSITY OF SÃO PAULO
SÃO CARLOS SCHOOL OF ENGINEERING
DEPARTAMENT OF MECHANICAL ENGINEERING
GRADUATE PROGRAM IN MECHANICAL ENGINEERING

Leonardo José Consoni

**Adaptable System for Robotic
Telerehabilitation with Serious Games**

São Carlos
2017

Leonardo José Consoni

**Sistema Adaptável para Telereabilitação
Robótica com Jogos Sérios**

Versão corrigida

(A versão original se encontra disponível na EESC/USP que aloja
o Programa de Pós-Graduação de Engenharia Mecânica)

Dissertação apresentada ao Programa de Pós-Graduação
em Engenharia Mecânica da Escola de Engenharia de São
Carlos, para obtenção do título de Mestre em Ciências.

Área de Concentração: Dinâmica de Máquinas e Sistemas

Orientador: Prof. Dr. Adriano Almeida Gonçalves Siqueira

São Carlos

2017

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

C755s Consoni, Leonardo José
Sistema adaptável para telereabilitação robótica
com jogos sérios / Leonardo José Consoni; orientador
Adriano Almeida Gonçalves Siqueira. São Carlos, 2017.

Dissertação (Mestrado) - Programa de Pós-Graduação
em Engenharia Mecânica e Área de Concentração em
Dinâmica das Máquinas e Sistemas -- Escola de
Engenharia de São Carlos da Universidade de São Paulo,
2017.

1. Robótica. 2. Reabilitação robótica. 3. Jogos
sérios. 4. Teleoperação. 5. Telereabilitação. 6.
Controle robótico. 7. Internet. 8. Jogos multijogador.
I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **LEONARDO JOSÉ CONSONI**.

Título da dissertação: "Sistema adaptável para telereabilitação robótica com jogos sérios".

Data da defesa: 24/02/2017.

Comissão Julgadora:

Resultado:

Prof. Associado **Adriano Almeida Gonçalves Siqueira**
(Orientador)
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Prof. Titular **Marco Henrique Terra**
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Prof. Dr. **Hermano Igo Krebs**
(Instituto de Tecnologia de Massachusetts/MIT-EUA)

APROVADO

Coordenador do Programa de Pós-Graduação em Engenharia Mecânica:
Prof. Associado **Gherhardt Ribatski**

Presidente da Comissão de Pós-Graduação:
Prof. Associado **Luís Fernando Costa Alberto**

Acknowledgements

First of all, I would like to thank my family for all the support provided during most part of my life's journey, with special attention to my parents, who constantly gave up their personal interests in order to raise my sister, my brother and me under the best possible conditions.

I would like to express my gratitude to my supervisor Adriano Siqueira for the useful comments, remarks and engagement through the learning process of this master thesis, as well as for the trust placed in my work.

I am also grateful to Prof. Hermano Krebs for the opportunity to perform and show my work at MIT, during a short internship visit that added enormously to my research and experience in many aspects.

I would like to thank my colleagues from the Robotic Rehabilitation Laboratory, for all the technical help and collaborative work that they provided, enabling me to carry out my project.

I would also like to thank my friends for all the years of comradeship, accepting me and my frequent faults without lacking the honesty to point them to me, giving me the push to become a better human being.

I recognize as well the financial support provided by São Paulo Research Foundation (FAPESP), under grant 2014/25828-4, and Coordination for the Improvement of Higher Level Personnel (CAPES), in the form of scholarships, without which I would not be able to carry out this project.

Last but not least, I would like to thank all the people, known or anonymous, that directly or indirectly collaborated to the developments I went through over the course of my life, to whom I wish to give something in return someday.

*“Man tries to make for himself in the fashion that suits him best
a simplified and intelligible picture of the world;
he then tries to some extent to substitute this cosmos of his
for the world of experience, and thus to overcome it.
This is what the painter, the poet, the speculative philosopher,
and the natural scientist do, each in his own fashion.
Each makes this cosmos and its construction the pivot of his emotional life,
in order to find in this way the peace and security
which he cannot find in the narrow whirlpool of personal experience.”*
(Albert Einstein)

Resumo

Consoni, L. J. **Sistema Adaptável para Telereabilitação Robótica com Jogos Sérios**. 56 p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2017.

Nas últimas décadas, o aumento global nos casos de problemas de saúde neuromotores, devido ao envelhecimento da população, motivou um interesse crescente na pesquisa sobre melhoria de processos de reabilitação utilizando robótica. Os estudos realizados real-izados abriram possibilidade de se incluir outras tecnologias auxiliares na terapia física e ocupacional, como realidade virtual e interação remota entre pacientes e terapeutas. Apesar dos benefícios verificados ou potenciais da aplicação dessas técnicas, ainda há poucas iniciativas no sentido de padronizar ferramentas e metodologias para sua implementação e teste, o que poderia catalisar os avanços na área. Esse trabalho descreve a criação de uma plataforma computacional comum para estudos de Reabilitação Robótica, com suporte a utilização de Jogos Sérios e teleoperação. Uma revisão bibliográfica prévia ajudou a definir os requisitos e ferramentas de desenvolvimento adequadas para tal sistema multi-propósito. Seu projeto modular e configurável tem o intuito de permitir reutilização de componentes e sua fácil adaptação a diferentes tipos de terapia, mesmo não inicialmente planejadas, evitando duplicação de trabalho. Jogos multijogador envolvendo controle de força e posição são também criados para testar o sistema proposto em situações reais ou simuladas, de modo a demonstrar sua utilidade para aplicação. Detalhes de sua estrutura de operação, protocolos de comunicação e componentes são mostrados, destacando-se a flexibilidade oferecida. Testes simples de viabilidade com indivíduos saudáveis são realizados, a fim de demonstrar sua utilidade para aplicação e avaliação de estratégias de reabilitação robótica. Ao fim, os resultados obtidos até então são discutidos, e considerações sobre informações ainda faltantes e trabalhos futuros são feitas.

Palavras-chave: Robótica. Reabilitação robótica. Jogos Sérios. Teleoperação. Telereabilitação. Controle robótico. *Internet*. Jogos multijogador..

Abstract

Consoni, L. J. **Adaptable System for Robotic Telerehabilitation with Serious Games**. 56 p. Master Thesis – São Carlos School of Engineering, University of São Paulo, 2017.

Over the last decades, the worldwide increase in cases of neuromotor health issues, due to overall population aging, motivated a growing research interest in improving rehabilitation processes with robotics. The performed studies opened the possibility to include other auxiliary technologies in physical and occupational therapy, like virtual reality and remote interaction between patients and therapists. Although there are verified and potential benefits to rehabilitation provided by these techniques, there are still few attempts towards tooling and methodology standardization, which could lead to faster developments. This work describes the creation of a proposed common computational platform for robotic rehabilitation studies, with support for virtual games and telecommunication. A preceding literature review helped to determine the requirements and development tools for such multipurpose system. Its modular and configurable design aims to allow components reuse and adaptability to different robotic therapy cases, even ones not initially intended, preventing work duplication. Details about the system's structure, components and operation are shown, focusing its provided flexibility. Multiplayer games involving position and force control are also created to test the proposed system in real and simulated environment, in order to demonstrate its usability for application and evaluation of robotic rehabilitation strategies. At the end, the results obtained so far are discussed and considerations about missing points and future developments are made.

Keywords: Robotics. Robotic rehabilitation. Serious games. Teleoperation. Telerehabilitation. Robot control. Internet. Multiplayer games..

List of Figures

Figure 1	Examples of robotic therapy devices using different types of assistance-based control algorithms: (A) MIT-MANUS, (B) Lokomat, (C) HWARD, (D) T-WREX, and (F) Pneu-WREX (MARCHAL-CRESPO; REINKENS-MEYER, 2009)	2
Figure 2	Representation of potential uses for telerehabilitation technology: (a) group therapy and (b) remote treatment like in home care (CARIGNAN; KREBS, 2006)	3
Figure 3	Top-level representation of possible communication layouts for the proposed system. Black borders delimit different hardware and colored borders different processes.	4
Figure 4	Example of unilateral telerehabilitation application: a patient interacts locally with a virtual reality environment, while a therapist receives activity data or modify its parameters through an Internet connection (CARIGNAN; KREBS, 2006).	11
Figure 5	Representation of bilateral robotic teleoperation, showing forward and backward data channels, slave force feedback to master trajectory input, and the doubled delay effect (adapted from Niemeyer e Slotline (2004)).	12
Figure 6	Wave variables transformations and transmission in bilateral teleoperation (adapted from Niemeyer e Slotline (2004))	12
Figure 7	Diagram of example implementation for proposed detailed design, showing all possible components and communications, to be detailed separately further ahead. In a multi-user setup, like this bilateral telerehabilitation game, operation is spread over 3 interconnected client-server groups. (adapted from Consoni, Siqueira e Krebs (2017)).	15
Figure 8	Commercial <i>Anklebot</i> model available at <i>ReRob</i>	16
Figure 9	Active knee orthosis for gait rehabilitation (SANTOS; SIQUEIRA, 2014) .	17

Figure 10	Communication scheme in control of the knee orthosis actuator. The <i>EPOS</i> motor drive has built-in CAN interfaces, while the real-time computer makes use of a <i>National Instruments</i> PXI-8512/2 external board.	18
Figure 11	Diagram of <i>RobRehabControl</i> configuration and abstraction levels, showing how each controlled robot data and code is handled by the application (adapted from Consoni et al. (2016)).	19
Figure 12	Diagram of <i>Robot Server</i> and its possible types of client connections: black borders delimit processes running on different machines, small colored boxes represent the IP sockets opened on each connection endpoint, large boxes inside the server process indicate the independent threads creating and handling connections on demand (adapted from Consoni et al. (2016)).	21
Figure 13	Example of robotic manipulator where controllable joint coordinates and tool/end-effector ones are not the same (taken from web page: http://www.intechopen.com/source/html/25813/media/fig2.png) . . .	22
Figure 14	Diagram of data flux and transformations inside the real-time control application. For each joint actuator of a robot, a Kalman filter makes possible to combine signals from redundant sensors or estimate values for missing measurements, keeping always the same set of joint output variables. The right side arrows represent interactions with client applications performed through the <i>Robot Server</i> proxy.	23
Figure 15	Control GUI created with the <i>Kivy</i> framework for <i>Python</i> : buttons on the left side issue calls to send commands to connected <i>Robot Server</i> , central graphs and right sliders show data received from update messages.	25
Figure 16	UML diagrams of the main components and subsystems of the game framework: (a) top-level management of client or server mode selection; (b) client classes, showing robot axis handling; (c) server classes.	26
Figure 17	Graphical development environment of <i>Unity3D Editor</i>	27

Figure 18	Configuration and calibration screen for rehabilitation games: at the top left, it is possible to set IP addresses of <i>Robot Server</i> and game server processes; top right "Axis Select" box has a "Refresh" button that issues request of robots information from connected <i>Robot Server</i> , used to fill below drop-down widget for selection of axis to calibrated (including mouse and arrow keys); central drop-down selects the currently calibrated variable, whose value is displayed in the slider below; around the display there are buttons for setting reached minimum and maximum values, used in normalization, as well as for issuing offset and reset messages to robot control; buttons at the bottom allow choosing the game to be played.	27
Figure 19	DoublePong gameplay screen movable elements (CONSONI et al., 2016) .	29
Figure 20	Box Clash gameplay screen movable elements	30
Figure 21	Interface detail from the VPN software (<i>Hamachi</i>) used in tests, showing local (on top) and remote (below) client virtual IPv4/IPv6 addresses used for communication.	34
Figure 22	Diagram of test layout for unilateral rehabilitation game performed at <i>ReRob</i> (adapted from (CONSONI et al., 2016)). Green and blue colors are used to indicate robots and paddles correspondence.	35
Figure 23	Results for first 30 seconds of remote test without motion compensation	38
Figure 24	Results for first 30 seconds of remote test using motion smoothing . . .	38
Figure 25	Detail comparison between the two remote tests, showing of the effects of motion smoothing algorithm.	39
Figure 26	Results for first 25 seconds of simulation test with just client motion smoothing, where middle plot's green line shows absolute value of difference between local and remote paddle trajectories (RMS error for this period was around 2.84).	39
Figure 27	Results for first 25 seconds of simulation test using full motion compensation (RMS error for this period was around 2.71).	40
Figure 28	Effects of artificial latency raise for force exchange game with no stabilized communication	41
Figure 29	Stabilization effect of wave variables on force exchange game, making it tolerable to higher delays.	41

Figure 30	<i>Scilab/Xcos + Coselica</i> simulator for teleoperation: a sinoidal force signal of unitary amplitude enters the master system (single mass), and resulting position and velocity are transmitted to the slave side, passing through local and remote compensator superblocks, that could hold any implementation; a compliance control is applied to the slave system interacting with environment, and a feedback force signal is generated, making its way back to master; forward and backward signals are delayed by artificial latency superblocks (CONSONI; SIQUEIRA, 2016)	51
Figure 31	Results obtained for a 100 seconds simulation of the modeled teleoperation system: (a) unstable, with no delay compensation, as reference; (b) corrected by wave variables application. Blue line represent master measurements and red slave ones.	52
Figure 32	Results obtained for a 100 seconds simulation of the modeled teleoperation system: (a) unstable, with no delay compensation, as reference; (b) corrected by force saturation application. Blue line represent master measurements and red slave ones.	52

List of Tables

Table 1	Control variables units convention	20
Table 2	Byte arrangement for command messages sent to <i>Robot Server</i> from its clients	23
Table 3	Data format for axes/joints update messages (for input or output) exchanged between <i>Robot Server</i> and its clients	24
Table 4	Data arrangement for byte array messages exchanged by game clients and server. 4 custom floating point value slots are reserved for different uses by the implementation.	28

Abbreviations

AAN	Assistance-as-needed
SEA	Series Elastic Actuator
API	Application Programming Interface
ARM	Advanced RISC (Reduced Instruction Set Computer) Machine
ASCII	American Standard Code for Information Interchange
CAN	Controller Area Network
CPU	Central Processing Unit
DAQ	Data Acquisition
DC	Direct Current
DOF	Degree of Freedom
EMG	Electromyography
EPOS	Easy Positioning
GUI	Graphical User Interface
I/O	Input/Output
ID	Identifier
IP	Internet Protocol
IPC	Inter-Process Communication
ISO	International Organization for Standardization
JSON	Javascript Object Notation
LAN	Local Area Network
LGPL	GNU Lesser General Public License
OS	Operating System
PCI	Peripheral Component Interconnect
PCI4E	PCI for Encoder
PID	Proportional–integral–derivative
PXI	PCI Extensions for Instrumentation
PXIe	PCI Express Extensions for Instrumentation
RMS	Root Mean Square
ROS	Robot Operating System
SQL	Structured Query Language

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
XML	Extensible Markup Language
Wi-Fi	Wireless Fidelity

Contents

1	Introduction	1
1.1	Rehabilitation Robotics	1
1.2	Serious Games for Rehabilitation	2
1.3	Telerehabilitation	3
1.4	Problem Statement	3
1.5	Objectives	5
2	Literature Review	7
2.1	Adaptable Robot Control	7
2.1.1	Static adjustment/optimization	7
2.1.2	Dynamic adaption	8
2.2	Teleoperation Techniques for Rehabilitation	10
2.2.1	Unilateral telerehabilitation	11
2.2.2	Bilateral telerehabilitation	11
2.3	Serious Games Development	13
2.3.1	Competition vs. cooperation	14
2.4	Existing Solutions	14
3	Design and Implementation	15
3.1	Distributed Operation	15
3.1.1	Inter-process communication	16
3.2	Rehabilitation Hardware	16
3.2.1	Anklebot	16
3.2.2	Knee orthosis	17
3.3	Software Components	18
3.3.1	Real-time control application	18
3.3.2	Real-time control server	21
3.3.3	Real-time control data flow	21
3.3.4	Robot control messaging format	22
3.3.5	Control and visualization GUIs	24
3.3.6	Virtual telerehabilitation games	25

4	Testing Methodology	33
4.1	Unilateral Telerehabilitation Game	33
4.1.1	Remote Tests between USA and Brazil	34
4.1.2	Local Tests in Simulated Environment	34
4.2	Bilateral Telerehabilitation Game	35
5	Results and Discussion	37
5.1	Unilateral Telerehabilitation Game	37
5.1.1	Remote Tests between USA and Brazil	37
5.1.2	Local Tests in Simulated Environment	37
5.2	Bilateral Telerehabilitation Game	40
6	Conclusion	43
	Bibliography	45
	Appendices	49
	APPENDIX A Additional Simulations	51
	APPENDIX B Available Resources	55
	B.1 Open Code Repositories	55
	B.2 Documentation	56

Chapter 1

Introduction

The worldwide population aging, combined with proliferation with other risk factors (e.g. sedentarism and hypertension), lead to a growing number of people getting more susceptible to health issues that compromise motor ability, like orthopedic injuries and neurological disorders. Their effects have an impact not only in the affected individuals' quality of life, but also result in a significant economic burden for society, due to the loss of people's productive capacity and costs related to their medical treatment and rehabilitation (KREBS et al., 2008).

Among those illnesses, the Cerebrovascular Accident (CVA or stroke) makes about 15 million victims a year, of which a third are fatal, being one of the three major causes of death and disability in the world, besides generating expenses that were estimated in more than US\$ 50 million just for the USA during 2004 (World Health Organization, 2004). Moreover, although medical advances in first aid and subsequent treatments that allow an increasing survival rate for stroke victims, there is still no guarantee that this raising number of survivors will not suffer with motor and cognitive sequelae, which nowadays affect nearly half of them.

Consequently, there is a raising demand on rehabilitation services, costly in time and resources, from users of public and private health systems, already quite overburdened in many countries. Thus, one can perceive a real need for new and more efficient procedures that could replace or supplement conventional physical and occupational therapy, capable of recovering patients in a quicker and less costly way, mitigating the economic and social impact of the aforementioned diseases (KREBS et al., 2008).

1.1 Rehabilitation Robotics

Formerly used only as an assistive technology to compensate for physical limitations, robots usage in motor rehabilitation (Figure 1) have been progressively more studied in recent decades (KREBS et al., 2008; MARCHAL-CRESPO; REINKENSMEYER, 2009; CAO et al., 2014). Controllable end effectors, orthoses and exoskeletons replace the therapist in

sustaining and moving patient's limbs during a therapeutic exercise, acting as an interface between those two. The direct benefits provided by this intermediary include allowing a single professional to conduct the treatment session, during which more repetitions can be performed, as the robot does not tire. Additionally, patient's motion performance (effort, regularity, amplitude) can be assessed more objectively, through robot's sensors, and assistance or resistance forces can be applied, as needed, more precisely.

Figure 1: Examples of robotic therapy devices using different types of assistance-based control algorithms: (A) MIT-MANUS, (B) Lokomat, (C) HWARD, (D) T-WREX, and (F) Pneu-WREX (MARCHAL-CRESPO; REINKENSMEYER, 2009)

As the control and monitoring of a robotic device generally involve a computer software, its usage indirectly offers the possibility of registering treatment data over the course of multiple sessions, for systematical evaluation of patient's progress, and network communications, for integration with e.g. virtual reality game systems (GONÇALVES et al., 2014; ANDRADE et al., 2013) or home care through telerehabilitation (CARIGNAN; KREBS, 2006; LANINI et al., 2015).

1.2 Serious Games for Rehabilitation

Virtual environments in rehabilitation could help patients in fulfilling motor tasks: audiovisual and haptic (tactile) stimuli may present objectives and suggest actions in a way that is more intuitive to understand. In the case of a video-game, for instance, the virtual world and interface elements, as well as the robot, could react to player movements in a way that gives a meaningful updated feedback, keeping the person constantly aware of its performance in the activity to be performed. If this combination of incitement and online response is able to maintain the patient focused and entertained, it can also increase its motivation, one of the main factors in recovery (NOVAK; NAGLE; RIENER, 2014).

Games intended to make use of those characteristics for goals beyond player's enjoyment, like therapy, are classified as serious games. As any traditional multiplayer game, those can be used to stimulate human interaction through competitive or cooperative gameplay (ANDRADE et al., 2013).

1.3 Telerehabilitation

With availability of a direct link or Internet connection, two or more participants of robotic rehabilitation sessions, separated by long distances, could remotely interact visually and mechanically, considering that the computers responsible for robots control can also handle the network data transfers. That feature could be used by a therapist to provide home care to a subject due to mobility difficulties (e.g. severe impairment), or to allow multiple patients to engage group therapy (Figure 2). This usage of telecommunication technologies for medical purposes is defined as telemedicine and, which also covers telerehabilitation (CARIGNAN; KREBS, 2006).

Figure 2: Representation of potential uses for telerehabilitation technology: (a) group therapy and (b) remote treatment like in home care (CARIGNAN; KREBS, 2006)

Additionally, cameras with microphones, for audio and video transmissions, and even online multiplayer serious games, could be used as resources to enhance remote interaction and make the multi-user activity more appealing to patients.

1.4 Problem Statement

Despite the potential of new therapy modalities enabled by robotics, and the growing interest in this research field, evidenced by the mentioned review articles, there is still few attempts to address some difficulties associated with these techniques:

- ❑ Physical and occupational therapy for neuromotor recovery, be it conventional or robot-assisted, must be adaptable to particularities of any treated individual, as patients may vary widely in anatomical characteristics and impairment level.
- ❑ Robot structures and controllers could be preadjusted for each user, but its performance evolution over the course of an activity (improvement from learning or degradation from fatigue) makes useful to have some strategy for dynamic adaption, based on predefined evaluation metrics. When there is a computational game tied to therapy, it is interesting that these criteria is somehow related to game score (or vice versa), making it part of the online feedback a player receives.
- ❑ In that last case, as the processing hardware more suited to robotic control is very distinct from one for electronic games (for instance, an embedded real-time system and a desktop with dedicated graphics card, respectively), as well as the tools for their development, its advantageous that those functionalities are implemented in different applications, that may run on separate machines. For telerehabilitation, the physical separation between applications is even unavoidable. Thereby, the proposed platform requires integration of at least on inter-process communication (IPC) mechanism that works across different computers (Figure 3).

Figure 3: Top-level representation of possible communication layouts for the proposed system. Black borders delimit different hardware and colored borders different processes.

- ❑ Any telecommunication technology (like the ones used in IPC) is bound to intrinsic delays, whose intensity variation may be non-deterministic, and susceptible to losses in data transfer (NIEMEYER; SLOTLIN, 2004). Those effects are affected by quality, type and distance of the network's medium of transmission. For simple remote visualization or one-directional control, a relatively large latency value is generally tolerable; but in online feedback control more intense lag may induce instabilities in a originally stable system. Unstable robot operation in direct contact with patients compromises safety, rendering the activity unfeasible, so there must be way to prevent delays or compensate for their effects.

1.5 Objectives

In this master's degree work, a created software platform for application of robot-aided therapy is proposed and detailed, to further expand works previously developed at the Robotic Rehabilitation Laboratory (ReRob) from São Carlos School of Engineering. It integrates adaptable robot control with virtual reality and telerehabilitation applications, differentiating itself from known solutions by providing defined interfaces for easy modification or replacement of those components.

As a proof of concept, a fully functional, although simple, implementation built for this work should be able to demonstrate:

- ❑ Enough flexibility to operate with different robotic devices and high/low level control algorithms without core changes.
- ❑ Handling of latency effects during a multiplayer telerehabilitation game match, under heavy communication lag constraints, keeping it mostly playable throughout the entire session.
- ❑ Ability to collect meaningful data about user's performance, that could aid a therapist's evaluation in a real patient treatment case.

Chapter 2

Literature Review

For dealing with distinct rehabilitation approaches, not totally related, the work hereby presented needs to be backed up by a broad review of the theoretical foundations. This chapter shows an attempt to recollect the most relevant references on current challenges and available solutions for each addressed issue.

It should be noted that it is outside the scope of this project to implement all the possible solutions for the proposed platform, but that knowledge is required in order to come up with a design capable of accommodating future developments.

2.1 Adaptable Robot Control

Industrial robots can be designed under the assumption that they will always move inside a restricted and unobstructed workspace. Thus, motion accuracy tends to be their major performance requirement, what is generally achieved by building these kinds of equipment with heavy and rigid mechanical parts, to minimize vibration, and applying simple PID (proportional–integral–derivative) controllers for desired motion tracking (HOGAN, 2014).

In contrast, devices designed for interaction with human subjects (for augmentation, assistance, evaluation, or rehabilitation) should not be built using the same structural or control principles of an industrial machine. For safety and usability reasons, rehab robots must be light and compliant, and are not required to enforce trajectory of a user during passive or active movements in different directions. However, conventional position control, even with force feedback, are not well suited for those cases (HOGAN, 2014).

2.1.1 Static adjustment/optimization

One well-known way to ensure stability of human-machine mechanical interactions involves modeling the entire as a single plant to be controlled, which requires knowledge about both actuator (robot) and load (person’s limb) behavior (HOGAN, 2014).

For a therapeutic robot shared among several patients, operation does not involve always the same interaction dynamics. Nevertheless, it is reasonable to assume constant values for mechanical properties (weights, lengths, inertia) of the robotic device and each patient's anatomy. Therefore, it is possible to set a per user control preadjustment of those parameters.

However, a dynamic and musculoskeletal human model, even for a single limb, gets too complex to be reliable enough for control (SETH et al., 2011), and its manual adjustment for each particular user would be a lengthy and error prone. That approach demands some kind of automated parameter optimization.

2.1.1.1 OpenSim

For supplying that need, Peña et al. (2017) uses *OpenSim*¹, a freely distributed software for biomechanical simulation. *OpenSim* ships with a set of generic human skeleton and muscle models that could be adjusted to specific individuals through an interactive application, that generates a properly scaled version by taking some subject's antropometric data as input (SETH et al., 2011; VIVIAN; REGGIANI; SARTORI, 2013). That new model, stored as a XML (Extensible Markup Language) file, can be loaded by *OpenSim* libraries, available for programs written in *C++*, *Java*, *Python* or *Matlab*, and then used to calculate patient's movement more accurately. However, one issue with applying *OpenSim* inside a control algorithm is its lack of proper support for online simulation, as the input values can not be updated directly

Fortunately, being open source software (available under the *Apache 2.0*² license), that functionality could be added to meet the specific needs of a project, but the changes will have to be maintained in a separate code tree or submitted upstream. That work could even be unnecessary, as it seems like real-time processing are already planned for next release³.

2.1.2 Dynamic adaption

Apart from their own anatomical characteristics, neuromotor rehabilitation patients present particularities regarding their behavior during exercises: over the course of a single therapy session, their motion performance can fluctuate significantly, due to unintentional reactions caused by neurological disorder (spasticity) and to the accentuated effects of fatigue on these people (KREBS et al., 2008; MARCHAL-CRESPO; REINKENSMEYER, 2009; GONÇALVES et al., 2014; CAO et al., 2014; HOGAN, 2014).

To deal with that increased variability, control strategies for therapeutic robots are mostly based on the *assistance-as-needed* (AAN) principle: it should help as little as

¹<http://opensim.stanford.edu/>

²<http://www.apache.org/licenses/LICENSE-2.0>

³<https://github.com/RealTimeBiomechanics/rtosim>

possible to ensure desired motion tracking, inciting the patient to complete that by itself. The matching between intentional muscle activation and actual limb movement stimulates reestablishment of nerve synapses, the lack of this combination prevents rehabilitation and leads to a slacking condition (MARCHAL-CRESPO; REINKENSMEYER, 2009; CAO et al., 2014; HOGAN, 2014).

2.1.2.1 Impedance control

One common way of applying the AAN principle is through impedance control. Originally proposed in Hogan (1985), this technique allows even a structurally rigid actuators to have a desired soft behavior, by modulating their apparent (virtual) stiffness and damping (MARCHAL-CRESPO; REINKENSMEYER, 2009). For this very reason, it has been largely applied on previous ReRob published works, regarding the development of compliant rehabilitation robots (GONÇALVES et al., 2014; ANDRADE et al., 2013; JARDIM; SIQUEIRA, 2013; SANTOS; SIQUEIRA, 2014).

Based on those studies, for a basic implementation of an impedance control algorithm, which is more directly related to the work hereby presented, it would be necessary to know, for each loop iteration, the current (measured) and desired values of position, velocity and interaction force for a given actuator. These data would then be internally used to define new values of virtual impedance and consequent motor output (Listing 2.1).

```
function RUN_CONTROL_STEP( current_position, current_position,
                           current_velocity, reference_velocity,
                           current_user_force, current_robot_force ):

    position_error = desired_position - current_position
    relative_velocity = current_velocity - reference_velocity

    virtual_stiffness = GET_STIFFNESS( position_error, current_user_force )
    virtual_damping = GET_DAMPING( position_error, current_user_force )

    interaction_force = PROCESS_DYNAMICS( current_robot_force,
                                          current_user_force )
    desired_force = - virtual_stiffness * position_error
                  - virtual_damping * relative_velocity

    return GET_MOTOR_OUTPUT( interaction_force, desired_force )
endfunction
```

Listing 2.1: Pseudocode describing a simplified algorithm for impedance modulation based on trajectory tracking error. The GET_STIFFNESS, GET_DAMPING, PROCESS_DYNAMICS and GET_MOTOR_OUTPUT functions could implement multiple control strategies

2.1.2.2 User's effort estimation

It can be seen that having an isolated measurement of patient's effort is desirable, but force sensors in the robot would only provide an interaction force signal. Available solution obtain that value indirectly, from inverse dynamics or muscular activation measurements through superficial electromyography (EMG) sensors (MARCHAL-CRESPO; REINKENSMEYER, 2009; CAO et al., 2014).

On both methods, the same *OpenSim* musculoskeletal model resulted for static optimization could be applied, as the software also provides articulation torques estimation through reaction forces or EMG signals input Seth et al. (2011), Sartori et al. (2015), Peña et al. (2017).

However, preview results from real-time *OpenSim* implementation, presented in (PIZZOLATO et al., 2016), show how the computational cost of that approach scales intensely with model-complexity, to the point that iteration times go much above the 5 milliseconds used in *ReRob* applications (SANTOS; SIQUEIRA, 2014). As an alternative, Peña et al. (2017) proposes using offline simulation results from a scaled and reliable *OpenSim* model to pre-train a neural network, whose runtime processing should be much cheaper.

2.2 Teleoperation Techniques for Rehabilitation

Delay compensation techniques for remote robot operation can also, by extension, be applied to the telerehabilitation case, as they are basically the same. The available literature contains various algorithms intended for this task, but many present some key disadvantage, like the need for full system modeling; reliance on accurate delay estimation, which is difficult to achieve; or stability loss under variable lag (jitter), characteristic of *Internet* communications (RODRÍGUEZ-SEDA; LEE; SPONG, 2009; KIM; CHANG; PARK, 2013; WANG; LI, 2013). Thus, methods that provide better overall performance, robustness and easier implementation were prioritized in this review.

There is something important to note about many articles in this field that deal with Internet connections: the emphasis given to the option for using UDP (User Datagram Protocol) instead of TCP (Transmission Control Protocol) for connections, despite the lack of data integrity guarantees from the former. The superior speed of UDP is more desirable, since most of the compensation algorithms suffer greater performance degradation due to raising latency than eventual network packet losses (RODRÍGUEZ-SEDA; LEE; SPONG, 2009; HALL, 2011).

To better understand the use cases for each compensation method, it is useful to follow the distinction made by Carignan e Krebs (2006) between two broad classes of telerehabilitation: unilateral and bilateral.

2.2.1 Unilateral telerehabilitation

According to (CARIGNAN; KREBS, 2006), the unilateral case involves just one-way communication, with a single robot user (patient) transmitting motion data over the network to observers (mainly therapists evaluating the session), which implies no delay-induced instability. Nonetheless, local force feedback could still happen, as it does not close a remote control loop (Figure 4).

Figure 4: Example of unilateral telerehabilitation application: a patient interacts locally with a virtual reality environment, while a therapist receives activity data or modify its parameters through an Internet connection (CARIGNAN; KREBS, 2006).

2.2.2 Bilateral telerehabilitation

In bilateral teleoperation or telerehabilitation scenarios, each data transmission channel from the user controlled device (master) to a remote one (slave, in actual contact with the target environment) has a corresponding returning channel, transporting online synchronization and feedback data from interaction with the remote system (Figure 5). That is the case to which most of the proposed compensation algorithms apply, because raising latency, as already stated, becomes a real concern.

2.2.2.1 Wave variables

Based on passivation theory and scattering transformations, wave variables are an encoding and decoding scheme for velocity ($\dot{\mathbf{x}}$) and force ($\mathbf{F}$) values transmitted over network in bilateral teleoperation (Equation 1), that ensure stability for any system, even if unknown (i.e. no modeling needed), under fixed latency of arbitrary intensity.

Figure 5: Representation of bilateral robotic teleoperation, showing forward and backward data channels, slave force feedback to master trajectory input, and the doubled delay effect (adapted from Niemeyer e Slotline (2004)).

Assuming some additional constraints, communication stability could be also guaranteed for discrete updates (NIEMEYER; SLOTLIN, 2004; RODRÍGUEZ-SEDA; LEE; SPONG, 2009), as it happens in this work.

$$u = \frac{b\dot{x} + F}{\sqrt{2b}} \quad v = \frac{b\dot{x} - F}{\sqrt{2b}} \quad (1)$$

The variables $\mathbf{u}$ and $\mathbf{v}$ are a complementary pair, denoting the forward (output) and backward (input) waves, respectively, for a master controller, and the exact opposite for its slave (Figure 6). In practice, by combining effort and flow values, wave variables work as virtual energy exchanges between corresponding actuators, establishing a trade-off, modulated by the wave impedance $\mathbf{b}$: as communication delays get larger, the master becomes stiffer to user input, in order to keep it stable; when latency falls, controller compliance raises, turning synchronization more transparent.

Figure 6: Wave variables transformations and transmission in bilateral teleoperation (adapted from Niemeyer e Slotline (2004))

The matching of velocities does not prevent actuators position to drift apart over time, due to packet losses or integration inaccuracies. To fix that, subsequent versions of the originally proposed algorithm started to integrate transmission of wave integrals, encoding position $\mathbf{x}$ and momentum $\mathbf{p}$ values (Equation 2). Theory ensures that trajectory tracking by the slave using these position setpoints will not violate system's intended passivity (NIEMEYER; SLOTLIN, 2004; RODRÍGUEZ-SEDA; LEE; SPONG, 2009).

$$U(t) = \int_0^t u d\tau = \frac{bx + p}{\sqrt{2b}} \quad V(t) = \int_0^t v d\tau = \frac{bx - p}{\sqrt{2b}} \quad (2)$$

2.2.2.2 Small-gain criteria

Remote control techniques based on passivity may impose large damping to a system under more severe latency situations, as a stabilization strategy, giving the operator an erroneous perception about the remote environment. As an attempt to improve upon those state-of-the-art solutions, Atashzar, Polushin e Patel (2012) proposes a new method based on simple slave force feedback f_s saturation, simplified in Equation 3.

$$f_s^* = \begin{cases} f_s & \text{if } f_s \leq f_s^{MAX} \\ \frac{f_s}{|f_s|} \cdot f_s^{MAX} & \text{otherwise} \end{cases}, \quad \text{with } f_s^{MAX}(t) \leq \|Z_p(s)\|_{L_1} \cdot \sup_{\tau \in [t-T, t]} |v_s(\tau)| \quad (3)$$

Where f_s^* is the interaction response actually delivered to the operator, which only differs from the original feedback f_s when the latter goes beyond a dynamic safe threshold f_s^{MAX} , optimizing communication transparency. The force limit depends on the lower bound of patient's impedance Z_p and varies with the greatest received value of slave velocity v_s inside a defined update time windows of duration $\mathbf{T}$.

Atashzar, Polushin e Patel (2012) claims that this is enough to ensure teleoperation safety, for any finite delay values, if the whole system matches a defined small-gain stability criteria, what is sustained through simulations. Lanini et al. (2015) endorses that theory by demonstrating it mathematically.

One notable disadvantage of this method, relative to wave variables, resides in its requirement for a model of human limb stiffness and damping, without which the slave feedback limitation could not be calculated.

2.3 Serious Games Development

Research on previous serious games applications in robotic rehabilitation can help to define common requirements when looking for a proper game development tools. From Novak et al. (2014), one can imply that features that are appealing in video-games targeted at pure entertainment could not be essential or even desirable for serious games. Large amounts of graphical and sound effects that enhance commercial titles, for instance, could distract or disturb the player during task execution. This issue is aggravated in the case of neurological rehabilitation like post CVA treatment, where the patient's cognitive impairment makes any excess of provided information lead to confusion and consequent inability to fulfill the virtual goals.

Therefore, it is not demanded that employed tools offer the most bleeding edge game technologies, and factors like ease of use and availability of learning material could be more relevant in deciding for one. Even so, Cowan e Kapralos (2014) verifies that well-

know modern game engines like *Unity3D*⁴ and *Unreal*⁵ end up being the most picked by serious games developers.

2.3.1 Competition vs. cooperation

Besides underlying infrastructure it is necessary to comprehend the effects of different types of gameplay, in order to design the game around specific tasks and kinds of players. Online gameplay is particularly interesting, as it could provide a virtual remote environment for telerehabilitation.

Following the terminology adopted in (JARRASSÉ; CHARALAMBOUS; BURDET, 2012), multiplayer games between patients can be either competitive or cooperative for divided (unilateral channels) tasks, and competitive or collaborative for interactive (bilateral channels) ones. As shown in Novak et al. (2014), Novak, Nagle e Riener (2014), a proper matchup between subject's personality and gameplay mode has the potential to increase enjoyment and participation in two-player games over local area network (LAN), but there is no mention to cases when player can not see each other.

2.4 Existing Solutions

It is also important to review published works that potentially already address the issues planned to be tackled. This project would be more justified if no similar solution was available so far.

At first sight, the idea of a flexible software for robotic control resembles what is found in the Robot Operating System (ROS) project⁶. However, it would be unfit for use here due to incompatibility with real-time processing, multi-robot functionality, lack of support for operating systems other than modern *Unix* versions and ability to only write client applications in *C++*, *Python* and *Lisp* languages⁷, making integration more difficult. Those and other issues are recognized by the developers and became the motivation for a redesign in ROS2⁸, still in development.

For serious games, most of the articles show no concern for standardizing a reusable framework, even less with the resources sought. As an exception, a dedicated and feature-packed platform for this purpose was already proposed in Conconi et al. (2008), but it seems like nowadays the project is no longer maintained and its code sources are not accessible.

⁴<https://unity3d.com/pt>

⁵<https://www.unrealengine.com/what-is-unreal-engine-4>

⁶<http://www.ros.org/>

⁷<http://wiki.ros.org/ROS/Introduction>

⁸http://design.ros2.org/articles/why_ros2.html

Chapter 3

Design and Implementation

In this chapter, a extensive review of the most important design patterns, software tools and hardware applied in the project is made. Many of these resources originate from previous *ReRob* developments, used directly or further enhanced during this work.

3.1 Distributed Operation

As already mentioned, the rehabilitation platform is made of multiple software components, running on separate, even distant, hardware units, and providing different functionality to the whole system, like robotic actuators control, graphical user interfaces (GUIs), interactive virtual reality, among others. That modularity and distributed operation leaded to a client-server design (Figure 7), capable of integrating a variable number of communicating processes, inserted or removed on demand.

Figure 7: Diagram of example implementation for proposed detailed design, showing all possible components and communications, to be detailed separately further ahead. In a multi-user setup, like this bilateral telerehabilitation game, operation is spread over 3 interconnected client-server groups. (adapted from Consoni, Siqueira e Krebs (2017)).

3.1.1 Inter-process communication

The chosen IPC method was Internet Protocol (IP) based connections, as they are capable of inter computer data transfer, over Ethernet or Wi-Fi, and supported in the majority of operating systems (OS) and programming languages through the BSD sockets interface. For applications necessarily executed on the same machine, the overhead of socket calls is avoided with IPC over shared memory buffers, using a code layer that abstracts different OS-specific memory mapping implementations.

3.2 Rehabilitation Hardware

This section describes the rehabilitation robots, available at ReRob, employed for tests during the platform development. The sensing and actuation features of all these equipment allow setting effector position, velocity or force, enabling implementation of at least basic impedance controllers.

3.2.1 Anklebot

The first therapy device, Anklebot (Figure 8), provides active control over user's movement in two degrees of freedom (DOFs), along sagittal (plantar-dorsiflexion) and frontal (inversion-eversion) planes of the ankle. These ankle movements are produced or facilitated by combining two linear actuators mounted in parallel. The robot is intrinsically backdriveable, which means that its gear can be manually moved even when turned off (ROY et al., 2009; FORRESTER et al., 2011).

Figure 8: Commercial *Anklebot* model available at *ReRob*

Its control is performed by an industrial computer, running *Ubuntu Linux* 5.10 OS, whose kernel was modified with *Xenomai*¹ patches, which add hard real-time capabilities to the system. Physical connection between controller and robot happens through *PCI4E*² and *PowerDAQ*³ boards, for *Gurley R119* position encoders measurement and *Kollmorgen RBE(H)* motors actuation, respectively (ROY et al., 2009).

3.2.2 Knee orthosis

An orthosis for knee rehabilitation (Figure 9), developed at ReRob, was also used in this study. The robotic design is based on a custom-made rotary series elastic actuator (SEA). The SEA is a common type of flexible effector where interaction torques (with the device user) can be measured by the deformation of a spring component. The motor gear offers no mechanical backdrive due to high reduction ratios (150:1), so that it can be only moved when a compliant control is active (SANTOS; SIQUEIRA, 2014).

Figure 9: Active knee orthosis for gait rehabilitation (SANTOS; SIQUEIRA, 2014)

That actuator is also built with *Maxon Motor* direct current (DC) motor, motor drive and encoders. Control software is processed on a *National Instruments PXIe-8115* embedded computer, running the DOS-compatible *Phar Lap ETS*⁴ real-time OS. The controller hardware is connected by a CAN (Controller Area Network) network to the *EPOS* (Easy Positioning) motor interface, and communication is performed using CANOpen⁵ protocol, as outlined in Figure 10.

¹<http://xenomai.org/>

²<http://usdigital.com/products/interfaces/pc>

³<http://www.ueidaq.com/data-acquisition-accessories/powerdaq>

⁴<https://www.intervalzero.com/products/ets/>

⁵<http://www.can-cia.org/can-knowledge/>

Figure 10: Communication scheme in control of the knee orthosis actuator. The *EPOS* motor drive has built-in CAN interfaces, while the real-time computer makes use of a *National Instruments* PXI-8512/2 external board.

3.3 Software Components

This section describes in more detail the software pieces composing the system architecture presented in Figure 7:

3.3.1 Real-time control application

The actual robot control process, named *RobRehabControl*, is written in *C* programming language (*ISO C99* standard), due to performance requirements, restrictions of the used real-time platforms, and available support from libraries and APIs (application programming interfaces) employed. To avoid blocking by other operations and ensure correct timing (5 millisecond per step), each device control algorithm is executed in its own independent thread, which also allows the application to take advantage of multi-core CPUs (computer processing units).

As there are many different mechanical designs, hardware interfaces and control schemes that could be used for rehabilitation robots, the usual approach involves rewriting the control software to handle each combination. However, in order to effectively carry out the intended objective of making a universal platform, enabling code reuse, *RobRehabControl* splits generic core functionality from specific implementation details. Through a provided multi-layer configuration structure (Figure 11) and module API, the application allows addition of new functionality without modification of its internals.

From top to bottom levels, the full control application is composed by:

- ❑ A hash table of generic robot instances, filled according to a provided list of identifiers (IDs), with each object defined by:

Figure 11: Diagram of *RobRehabControl* configuration and abstraction levels, showing how each controlled robot data and code is handled by the application (adapted from Consoni et al. (2016)).

- Robot configuration parameters
- High-level control implementation module
- A list of generic actuators list, each one, in turn, complemented by:
 - * Actuator configuration parameters
 - * Low-level control implementation module
 - * A list of generic sensors, each one, in turn, completed by:
 - Sensor configuration parameters
 - Data acquisition/input implementation module
 - * Generic actuation motor, complemented by:
 - Motor configuration parameters
 - Signal output implementation module

3.3.1.1 Interchangeable modules (plug-ins)

Sensor and motor boards I/O (input/output), low-level (single actuator) and high-level (whole robot) control algorithms are the functionalities which can vary too much to support by simply changing parameters. Being able to write to write optimized implementations directly in code seems more useful for that purpose.

To enable such feature while keeping its genericism, *RobRehabControl* supports dynamic loading of code precompiled as libraries (like DLLs on *Windows* OS). By following

defined plug-in APIs, one can build its own device-specific code to be loaded and called by the core control application during runtime.

In order to keep consistency in data transformations across user developed plug-ins and allow interoperation between them, the convention in Table 1 is adopted for measurement (output) and setpoint (input) variables:

Control Variabe	Translational Unit	Rotational Unit
Position	meter (m)	radian (rad)
Velocity	m / second (m/s)	radian / second (rad/s)
Force/Torque	Newton (N)	Newton · meter (N · m)
Acceleration	meter / second ² (m/s ²)	radian / second ² (rad/s ²)
Stiffness	Newton / meter (N/m)	Newton · meter / radian (N·m/rad)
Damping	N x s / m	N · m · s / rad

Table 1: Control variables units convention

3.3.1.2 File-based configuration

The definition of plug-ins to be loaded and additional device control parameters (to handle minor variations) is made by configuration files written in *JSON* (JavaScript Object Notation) format⁶ (Listing 3.1), loaded at execution time.

```
{
  "sensors": [
    { "input_variable": "POSITION",
      "config": "joint_position" },
    { "input_variable": "VELOCITY",
      "config": "joint_velocity" },
    { "input_variable": "FORCE",
      "config": "joint_torque" }
  ],
  "motor": {
    "output_variable": "VELOCITY",
    "config": "joint_motor"
  },
  "controller": "ForcePIControl"
}

{
  "output_interface":
  {
    "type": "NICANEPOS",
    "config": "1",
    "channel": 1
  },
  "output_gain":
  {
    "multiplier": 1,
    "divisor": 1
  },
}
```

Listing 3.1: Actuator and motor configuration

The configuration system is also applied to another program's parts, to provide more flexibility. The JSON file parser itself is implemented as a plug-in, selected based on command-line arguments passed to *RobRehabControl* on its initialization. Thus, possible future changes to other storage formats, like XML files, or even SQL (Structured Query Language) databases, would be facilitated.

⁶<http://www.json.org/>

3.3.2 Real-time control server

Running on the same real-time system as *RobRehabControl* and also written in C, another program, *Robot Server*, acts as a proxy between the control logic and client applications. From one side, it takes requests and sends information to remote clients through IP sockets, using multithreading to handle multiple connections independently. From the other side, it exchanges that control data with *RobRehabControl*, its effective producer/consumer, on a single memory-mapped buffer.

Depending on the type of data transmitted through IP communication channels, different underlying protocols are used, considering their intrinsic characteristics (Figure 12). TCP, slower and more reliable, is used for critical information sent only a few times, like initialization and state change commands; UDP, faster and more error prone, works for continuous updates, when lower latency matters most and the loss of a message would be quickly compensated by a following one.

Figure 12: Diagram of *Robot Server* and its possible types of client connections: black borders delimit processes running on different machines, small colored boxes represent the IP sockets opened on each connection endpoint, large boxes inside the server process indicate the independent threads creating and handling connections on demand (adapted from Consoni et al. (2016)).

3.3.3 Real-time control data flow

From the point of view of a *Robot Server* client application operating a robot, or just visualizing its data, the positions, velocities and forces of interest does not necessarily match the ones measured in or followed by the actuators. That happens, for instance, when trying to control the Cartesian (x,y) position of an end-effector tool in a 2-link robotic arm with motors and sensors on each revolution joint (Figure 13).

Figure 13: Example of robotic manipulator where controllable joint coordinates and tool/end-effector ones are not the same (taken from web page: <http://www.intechopen.com/source/html/25813/media/fig2.png>)

Making the required conversions from one coordinate system to another would demand from developers knowledge about specific robot's kinematics and dynamics, violating the network interface abstraction. Besides, that would complexify a frontend implementation with operations not related to user interaction. It seems reasonable that such task should be performed by the real-time controller, as it already runs robot-specific code.

For clients, then, there are two sets of variables accessible for each controllable device DOF: the original actuated joint values and the result of their transformation, here called "axis" values, as an analogy with joystick/gamepad input axes. The real-time higher-level control pass is responsible for performing conversions from joints to axes measurements, and from axes to joints control setpoints. The process also involves joints stiffness and damping estimation and modulation for axis impedance control (Figure 14).

3.3.4 Robot control messaging format

Client applications communicating with *Robot Server* through IP connections must use fixed-size 512 bytes messages, as data packets below 1 kilobyte are very likely to be transmitted without fragmentation (HALL, 2011). Depending on the type of message sent or received, a specific data format is defined.

Messages requesting state changes or information about one or more robot are sent by clients to the server over TCP sockets, on server port 50000, as a byte array with data organized as shown in Table 2.

Figure 14: Diagram of data flux and transformations inside the real-time control application. For each joint actuator of a robot, a Kalman filter makes possible to combine signals from redundant sensors or estimate values for missing measurements, keeping always the same set of joint output variables. The right side arrows represent interactions with client applications performed through the *Robot Server* proxy.

1 byte	1 byte	1 byte	1 byte	1 byte	...
number of commands	robot index 1	corresponding command code	robot index 2	corresponding command code	...

Table 2: Byte arrangement for command messages sent to *Robot Server* from its clients

Currently available command codes, their values and roles are the following:

- ❑ **ENABLE** (0x01): turn actuators on and make control active;
- ❑ **DISABLE** (0x02): turn actuators off and disable control;
- ❑ **RESET** (0x03): clear errors and calibration values;
- ❑ **OPERATE** (0x04): set the robot control to normal operation;
- ❑ **OFFSET** (0x05): set the robot control to joints reference acquisition;
- ❑ **CALIBRATE** (0x06): set the robot control to joints amplitude measurement;
- ❑ **PREPROCESS** (0x07): call implementation-specific post-calibration routines for the robot control.

The same channel could be used to send more complex messages, in string format (character array with null terminator). Currently, only one message of this type is supported, but more are planned (like setting user name for personalized logs). **Robots Info** message is divided in two parts:

- ❑ Client Request: Request information about all available robots, axes and joints, performed by sending a single NULL (0x00) byte to *Robot Server*
- ❑ Server Reply: Information returned to requesting client. Serialized ASCII string representation in the format specified by currently used file I/O implementation. In the default JSON format, it would resemble Listing 3.2:

```
{"robots":["robot_1"],"axes":["r1_x","r1_y"],"joints":["r1_0","r1_1"]}
```

Listing 3.2: Example of JSON-formatted information for a single robot with 2 DOFs

On the other hand, measurements and setpoints are transported by UDP messages, using ports 50001 for axes and 50002 for joints. Arrays of DOF indexes and related single precision (4 bytes) floating-point values are serialized to and deserialized from these network packets following a predefined data format (Table 3).

1 byte	1 b	4 b	4 b	4 b	4 b	4 b	4 b	1 b	...
n° of DOFs	DOF index 1	pos.	vel.	force	accel.	stiff.	damp.	DOF index 2	...

Table 3: Data format for axes/joints update messages (for input or output) exchanged between *Robot Server* and its clients

When working with inter computer communication involving explicit memory buffer copies, like this one, a thing to be taken into consideration is the byte ordering for numeric values (integer and floating point) used in each system. Some hardware architectures like *x86* use *Little-Endian*, storing least and most significant bytes in reverse memory order, relative to systems employing *Big-Endian* (HALL, 2011).

In practice that issue has little impact, since two of the predominant architectures, namely *x86* and ARM, support *Little-Endian* format. By respecting that restriction and complying with other encoding conventions like IEEE 754 for floating point (guaranteed for *ISO C99* compliant compilers), there is no need for data conversions overhead.

3.3.5 Control and visualization GUIs

User interfaces for online robot control or data visualization, like the one in Figure 15) are normally created with some specialized toolkit, like *TK* or *Qt*. It is intended that they comply with *Robot Server*'s data channels and message structure for remote communication, but if the used real-time system presents graphical capabilities, it would be possible for those GUIs to talk directly to *RobRehabControl* by mapping its shared memory buffer.

Figure 15: Control GUI created with the *Kivy* framework for *Python*: buttons on the left side issue calls to send commands to connected *Robot Server*, central graphs and right sliders show data received from update messages.

3.3.6 Virtual telerehabilitation games

One of the essential parts of the telerehabilitation platform is the video-game through which the users interact. In this work, a common and reusable framework (Figure 16) was developed first, and then used to create two games with different purposes for testing.

3.3.6.1 Multiplayer game framework

The tool of choice for game development tool was the *Unity3D* editor (Figure 17), a widespread solution around which a large community has grown over the years, making it easy to find learning resources and example implementations for some functionality. Its underlying engine (runtime) provides a series of premade options for graphics, sound, input handling and networking, as well as multiplatform support (including mobile). The behavior of the interface and gameplay elements are controlled by scripts, coded in C# language (MESSAOUDI; SIMON; KSENTINI, 2015).

For the common multiplayer game framework developed, a master server scheme was adopted, where the server (Figure 16c) is simply a unique instance of the implemented game itself with different behavior (e.g. without graphics), promoting authoritative or semi-authoritative synchronization across the connected client (Figure 16b) instances, with whom players interact using the rehabilitation robots.

Upon loading any application based on this framework, a configuration interface (Figure 18) is presented to the player. It allows parameters such as the network address, input, type of device and calibration measurements (range, offset) to be set. Following initial set-up, the gameplay screen is displayed, which allows initialization of the activity.

Figure 16: UML diagrams of the main components and subsystems of the game framework: (a) top-level management of client or server mode selection; (b) client classes, showing robot axis handling; (c) server classes.

Figure 17: Graphical development environment of *Unity3D Editor*

Figure 18: Configuration and calibration screen for rehabilitation games: at the top left, it is possible to set IP addresses of *Robot Server* and game server processes; top right "Axis Select" box has a "Refresh" button that issues request of robots information from connected *Robot Server*, used to fill below drop-down widget for selection of axis to calibrated (including mouse and arrow keys); central drop-down selects the currently calibrated variable, whose value is displayed in the slider below; around the display there are buttons for setting reached minimum and maximum values, used in normalization, as well as for issuing offset and reset messages to robot control; buttons at the bottom allow choosing the game to be played.

3.3.6.2 Network messaging

Thanks to Unity3D’s recently released *Transport Layer API*⁷, communication channels built on top of UDP sockets with different Quality of Service (QoS) requirements⁸, regarding reliability and ordering, can be easily established. Also, continuously updated stats about network performance could be automatically obtained from the available *Transport Layer* methods.

Using that interface, messages of game objects state synchronization are exchanged between clients and server in format similar to robot axis updates (Table 4). Server port 50004 is used and packets are limited to 512 bytes in order to avoid some UDP fragmentation issues (BELLIS, 2010).

4 bytes	1 byte	1 byte	4 bytes	4 bytes	4 bytes	4 bytes	1 byte	...
array length	object ID 1	variable ID 1	custom value 1	custom value 2	custom value 3	custom value 4	object ID 2	...

Table 4: Data arrangement for byte array messages exchanged by game clients and server. 4 custom floating point value slots are reserved for different uses by the implementation.

3.3.6.3 Delay effects prevention

Trying to find a fixed network update interval that optimizes communication is not an easy solution: bounding it to the *Unity3D*’s physics engine updates, that by default happen every 20 milliseconds, produces an usually excessive number of messages, that could easily lead to traffic congestion, raising lag to unfeasible amounts; simply increasing it, on the other hand, to avoid filling network buffers, compromises elements movement synchronization between game instances during changes of direction or collisions.

Thus, a primary adaptive strategy is implemented: clients send messages updating the state values (like position, velocity, force) of their locally controlled objects only when they vary by a minimum threshold (5% of maximum motion range).

3.3.6.4 Unilateral telerehabilitation game

The first developed game (Figure 19), named *Double Pong*, consists of a two-dimensional variant of the classical Pong arcade game (PASQUAL; CAURIN; SIQUEIRA, 2016). It presents two pairs of diametrically opposed paddles, each pair controlled by a single user. The view-port rotates so the user gains visual feedback of the movement controlling the vertical paddles. A vertical slider at the left edge of the screen acts as a visual cue, prompting the required position of the paddles for successful completion of the task.

⁷<https://docs.unity3d.com/Manual/UNetUsingTransport.html>

⁸<https://docs.unity3d.com/ScriptReference/Networking.QosType.html>

Figure 19: DoublePong gameplay screen movable elements (CONSONI et al., 2016)

According to the researched literature definitions, *Double Pong* brings to players a divisible task, that could be interpreted either as competitive or cooperative. For instance, if one player's score count increases each time the other fails to intercept the ball, they are in a competition; if both scores increase when the ball bounces on a paddle, cooperation is encouraged.

The lack of physical interaction between objects controlled by different players conforms to the principle of unilateral teleoperation and, by extension, telerehabilitation (CARIGNAN; KREBS, 2006). As the communication among clients and server involves only position data, without force feedback, synchronization across those can be improved with basic motion compensation.

Network messages containing movement data of locally controlled objects are sent by each game instance to update their remote counterparts. When a delayed message arrives at server, a prediction p_{pred} of the actual current object position is performed (Eq. 4) before the physics step, based on last received position p_{rec} , velocity v_{rec} and acceleration a_{rec} values, as well as on delay estimation $\hat{\tau}_{lag}$. This corrected value is then passed along with v_{rec} to the opposite client.

$$p_{pred} = p_{rec} + v_{rec} * \hat{\tau}_{lag} + a_{rec} * \hat{\tau}_{lag}^2 \quad (4)$$

When latest p_{pred} and v_{rec} values arrive on client side, positions of remotely controlled objects are not directly set. Instead, a compensated velocity v_{comp} is calculated for a smoother transition, accounting the drift between p_{pred} and current local position p_{curr} , weighted by a factor α (unitary for now). Until the next message arrives, this value is used to estimate p_{curr} for every physics update interval Δt_{upd} , following a simple linear extrapolation (Equation 5).

$$p_{curr} = p_{curr} + v_{comp} \cdot \Delta t_{upd}, \quad \text{with } v_{comp} = v_{rec} + \alpha \cdot (p_{pred} - p_{curr}) \quad (5)$$

The game client also calculates position and stiffness setpoints and sends them back to robot control for assistance regulation, according to the implemented algorithm.

3.3.6.5 Bilateral telerehabilitation game

The second game, named *Box Clash*, was made to evaluate bilateral operation performance in interactive game tasks. It has a simplified layout (Figure 20), constituted by two player controlled boxes connected by a spring element, causing a continuous force exchange between them. Analogously to *Double Pong*, local players always view their input affecting the bottom box. As the spring allows the players to push or pull each other, this game could be used for either competitive (different targets) or collaborative (same target) activities.

Figure 20: Box Clash gameplay screen movable elements

For now, wave variables are the only stabilization algorithm implemented for multi-player synchronization, as described force feedback saturation needs stiffness estimations still unavailable (left for another projects). For the wave controller, based on the simplest teleoperator design proposed in Niemeyer e Slotline (2004), the feedback loop starts with calculation of local velocity $\dot{x}_m$ for player's box (*Unity3D*'s rigid body), resulted from the application of user input force F_{input} in the absence of spring reaction, as in Equation 6.

$$\dot{x}_m = physics(m_{body}, F_{input} - B \cdot \dot{x}_m + F_m), \quad \text{with } F_m(t) = F_s(t - \tau_{lag}(t)) \quad (6)$$

Where m_{body} is the virtual body mass and $\mathbf{B}$ is a dragging factor to prevent early destabilization. As the velocity values considered for this damping effect are absolute (global space), it could be calculated locally. F_m is the delayed by τ_{lag} value of F_s reaction, returned from the remote environment (server virtual world), initially set at zero. The obtained velocity value $\dot{x}_m$ is then used to encode the outgoing wave variable u_m (Equation 7).

$$u_m = b\dot{x}_m + \sqrt{2b} \cdot v_m, \quad \text{with } v_m(t) = v_s(t - \tau_{lag}(t)) \quad (7)$$

Being v_m the late version of incoming wave value v_s . Upon arriving at the server, where bodies are just kinematic (not affected by forces), velocity $\dot{x}_s$ of each box is updated, based on the time-delayed wave variable u_s (Equation 8).

$$\dot{x}_s = \frac{\sqrt{2b} \cdot u_s + F_s}{b}, \quad \text{with } u_s(t) = u_m(t - \tau_{lag}(t)) \quad (8)$$

Where the reaction force F_s is the one exerted on each body by the deformation of the virtual spring element of stiffness $\mathbf{K}$ and initial length l_0 . Its value is used to encode the returning wave v_s (Equation 9), closing the control loop.

$$v_s = u_s - \sqrt{\frac{F_s}{b}}, \quad \text{with } F_s = -K(x_s - x_s^{other} - l_0) \quad (9)$$

Also, to help patient in reaching desired box positions, *Box Clash* client also sets the same position and stiffness setpoints, that are sent back to *Robot Server* along with F_s values, so that the player can experience the force interactions.

Chapter 4

Testing Methodology

To determine the feasibility and quality of the proposed approach, we conducted experimental tests to evaluate the system's performance in the presence of communication delays. Both games were played by healthy subjects while the following network data were collected:

- ❑ Connection stats provided by Unity3D's API:
 - Round-trip time (RTT): the interval from sending a message to receiving its response, whose half value is also used as delay estimation here;
 - I/O overhead: time spent on send/receive calls;
 - Packet loss: number of failures in delivering data through the unreliable underlying protocol (UDP).
- ❑ Network timestamps appended to every transmitted message for calculating communication delays;
- ❑ Position, velocities and force values for synchronized objects on each client.

These variables were registered for each user and indexed based on time (relative to a common clock reference established when clients connected the server).

A missing factor in this evaluation is how the robotic assistance affects player performance, due both to not involving impaired subjects and to the lack of a proper player estimation implementation, that would be needed for regulating actuator output properly.

4.1 Unilateral Telerehabilitation Game

For the motion compensation strategy, tests on *Double Pong* were performed separately for server prediction Equation 4 and client smoothing Equation 5 passes. Due to the aforementioned robot assistance limitation, the robots used were left with passive

behavior: control turned off for *Anklebot* devices and zero impedance control for knee orthosis.

4.1.1 Remote Tests between USA and Brazil

As part of a partnership project between Massachusetts Institute of Technology (MIT) and São Carlos School of Engineering (EESC-USP), sponsored by São Paulo Research Foundation (FAPESP) under grant 2015/50376-2 and MIT-Brazil Misti Program grant 2160632, this work was involved in a study of robotic telerehabilitation over long distances. During it, *Double Pong* matches were performed between users, wearing *Anklebot* models, located at MIT's *Newman Laboratory* (Cambridge, MA, USA) and ReRob (São Carlos, SP, Brazil). The sites were separated by approximately 7,554 km (4,694 mi), which translates to a minimum communication lag in the order of 37 milliseconds¹ (74 ms for RTT, which at least amounts to two messages delay).

To avoid the need for a public domain to host the game server, a virtual private network (VPN) tool (Figure 21) was used, providing a direct link over Internet. For simplicity, the game server was run on the same computer that processed one of the user's clients. Thus, transmission lag between them was considerably less significant, and delay effects were analyzed for the the physically separated client. The first experiment was conducted with the *Anklebot* robot at both *Newman Laboratory* and *ReRob* sites.

Figure 21: Interface detail from the VPN software (*Hamachi*) used in tests, showing local (on top) and remote (below) client virtual IPv4/IPv6 addresses used for communication.

4.1.2 Local Tests in Simulated Environment

Even with the long distances involved during the first experiment, the initial results revealed that the high quality network infrastructure available at both Universities did not represent a “worst case scenario” for transmission delay. Further tests were conducted in

¹<http://www.timbercon.com/time-delay-of-light-in-fiber-calculator/>

São Carlos to simulate poor network operation conditions. With all applications running on a single machine (local match), an IP packet filter was used to delay every message in 300 ms, causing a net effect of 600 ms for RTT. That is a little above what is considered a tolerable latency limit for online gaming in general (SUZNJEVIC; SALDANA, 2015).

As there was only a single Anklebot system in São Carlos, we were also able to test the system's ability to handle different types of robotic hardware. We employed the knee orthosis described in section 3.2.2, controlled by the same software, except for device-specific plug-ins (Figure 22).

Figure 22: Diagram of test layout for unilateral rehabilitation game performed at *ReRob* (adapted from (CONSONI et al., 2016)). Green and blue colors are used to indicate robots and paddles correspondence.

4.2 Bilateral Telerehabilitation Game

For *Box Clash*, only the tests in simulated delay environment were run, and for safety reasons (immature implementation due to time constraints) no robot was employed, just keyboard input. Firstly, for comparison, game communications exchanged force and velocity data directly in a match under no artificial latency, in order to adjust physical variables with minimally stable values. After this, employing the obtained parameters, lag was raised until the system synchronization reached instability. Wave variables were then applied to data transmissions to verify their stabilization effect, with a wave impedance fixed in 10, already proven an usable value in Consoni e Siqueira (2016).

Chapter 5

Results and Discussion

Stored results were plotted relative to synchronized time, for the first 30 or more seconds of synchronization, using *Scilab* software. I/O overhead was negligible and no packet data was lost, hence these values were eliminated from the data analysis.

5.1 Unilateral Telerehabilitation Game

5.1.1 Remote Tests between USA and Brazil

Figure 23 shows results for tests between USA and Brazil without any motion compensation. The top plot shows the delay curves estimated from difference in timestamps (green) and half of RTT (black). Middle plot exhibits the original positions, in engine space meters, of the vertical paddles from the side of the client closer to server (blue) compared to its reproduction on the other client (red). The bottom plot does the same for the server generated ball X trajectory.

The proposed movement smoothing (client side) is activated on a second run, producing the results seen on Figure 24. Note that on both runs, there was a good match across clients, partly due to relatively small delays (around 90 ms), but paddle synchronization deviates a little more due to taking both clients latency into account. It seems like half of the RTT value provided by Unity3D's API is roughly equivalent to the average of delays, proving to be a sufficient estimate for following results. The effect of our proposed scheme can be observed on Figure 25, which shows a much smoother trace.

5.1.2 Local Tests in Simulated Environment

The same measurements were taken for the local test with simulated delay, except for time sample delay calculation being removed and absolute error (green line) being added for paddle trajectories. Behavior resulting from applying artificial large network delays can be seen in Figs. 26 and 27, which show, the response with just client-side smoothing and full proposed movement correction (prediction + smoothing), respectively.

Figure 23: Results for first 30 seconds of remote test without motion compensation

Figure 24: Results for first 30 seconds of remote test using motion smoothing

Figure 25: Detail comparison between the two remote tests, showing of the effects of motion smoothing algorithm.

Figure 26: Results for first 25 seconds of simulation test with just client motion smoothing, where middle plot's green line shows absolute value of difference between local and remote paddle trajectories (RMS error for this period was around 2.84).

Figure 27: Results for first 25 seconds of simulation test using full motion compensation (RMS error for this period was around 2.71).

As expected, communication delay remained around 300 ms for each communication channel. Consequently, for paddle synchronization across clients, the delay adds up to a 600 ms gap from blue to the red line at certain plot points. In comparison, the ball movement data (generated by the server) reaches both users at practically the same time with improved trajectory matching. When running the game with server-side motion prediction, local and remote paddle positions were kept closer, as shown by the slight amplitude reduction on the error curve. However, there is a noticeable overshoot in compensated values during direction changes, and the delay was still perceived during gameplay, which highlights the limitations of working with lack of actual network data. Even though, as in this case paddle movements have upper and lower constraints, the activity was able to be completed without significant problems.

5.2 Bilateral Telerehabilitation Game

After some trial and error, virtual environment parameters were fixed in 10 Kg for rigid body masses, 5 N/m for spring stiffness and 0.1 N·s/m for drag effect damping. With those values, the system remained stable during operation under zero delay conditions, and started showing unstable behavior once the artificial lag reached 50 ms (Figure 28). At this point, wave variables transformations were enabled and their performance was compared (Figure 29) to the direct transmissions of velocity and force.

Figure 28: Effects of artificial latency raise for force exchange game with no stabilized communication

Figure 29: Stabilization effect of wave variables on force exchange game, making it tolerable to higher delays.

Although theory claims that wave variables are able to stabilize a teleoperated system under any amount of communication latency, the attempted implementation raised the delay toleration (before instability) to just over 100 milliseconds, which is already an improvement, but leaves room for further improvement. More detailed investigation and experiments would be necessary to determine the actual causes of this inconsistency between mathematical model and the observed results. Also, a more suitable approach for defining initial virtual physics parameters could also be developed, as the one achieved manually here didn't offer much stability margin to begin with.

As only a simple wave controller method was applied to the bilateral experiment, due time constraints, it would be much interesting to compare, in the future, the results obtained here with more elaborated wave variables approaches, like impedance matching between both communication endpoints (NIEMEYER; SLOTTLINE, 2004) or dynamic wave impedance adaption (RODRÍGUEZ-SEDA, 2015). Besides, even performance of completely different strategies, like the feedback force saturation proposed in Atashzar, Polushin e Patel (2012), could be verified and used as a comparison parameter.

Chapter 6

Conclusion

In this master thesis, the proposal, development and initial evaluation of a multipurpose platform for robotic rehabilitation were presented in detail. The system's ability to be adapted to different types of therapy activities, robotic control, virtual reality applications and telerehabilitation modes is one of its main goals, and a considerable effort was made for fitting it to requirements gathered through related literature survey and from knowledge directly obtained over the course of this study.

Many issues regarding an effective implementation were discussed and addressed: need for splitting software processing across multiple computers; organization of generic control code intended to work with completely distinct devices; well structured but not overly complex design of inter-process communication channels and message formats; proper definition of roles for real-time and GUI components; latency compensation techniques for unilateral and bilateral telerehabilitation video-games; among others.

A minimum set of applications, implementing all the proposed platform's interfaces, was built in order to test the idea, to some extent, in a real case. ReRob's robots and other hardware were employed in combination with the developed software for completing a functional system. Multiplayer virtual games controlled by those rehabilitation robots were played over LAN and Internet, under variable communication delays, with different degrees of success, but ultimately proving the feasibility of integrating all the aforementioned approaches, at least according to the initial objectives.

Considering its coverage, this project still offers many opportunities for better understanding of its performance bottlenecks and for incremental enhancement. It can also be promoted for integration with concurrent studies developed at ReRob, for which the proposed platform can serve as a unified research tool, accelerating advancements and justifying even more the work invested in its development.

Bibliography

ANDRADE, K. D. O. et al. Rehabilitation robotics and serious games: An initial architecture for simultaneous players. **ISSNIP Biosignals and Biorobotics Conference**, 2013. 2013. ISSN 23267771.

ATASHZAR, S. F.; POLUSHIN, I. G.; PATEL, R. V. Networked teleoperation with non-passive environment: Application to tele-rehabilitation. In: **IEEE International Conference on Intelligent Robots and Systems**. 2012. p. 5125–5130. ISBN 9781467317375. ISSN 21530858.

BELLIS, R. **RFC 5966 - DNS Transport over TCP - Implementation Requirements**. 2010. Disponível em: <<https://tools.ietf.org/html/rfc5966>>.

CAO, J. et al. Control strategies for effective robot assisted gait rehabilitation: the state of art and future prospects. **Medical Engineering & Physics**, 2014. v. 36, n. 12, p. 1555–66, 2014. ISSN 1873-4030.

CARIGNAN, C. R.; KREBS, H. I. Telerehabilitation robotics: Bright lights, big future? **The Journal of Rehabilitation Research and Development**, 2006. v. 43, n. 5, p. 695, 2006. ISSN 0748-7711.

CONCONI, A. et al. PlayMancer: A Serious Gaming 3D Environment. In: **International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution**. : IEEE, 2008. p. 111–117. ISBN 978-0-7695-3406-0.

CONSONI, L. J. et al. A robotic telerehabilitation game system for multiplayer activities. In: **IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics**. Singapore: IEEE, 2016. v. 2016-July, p. 798–803. ISBN 9781509032877. ISSN 21551774.

CONSONI, L. J.; SIQUEIRA, A. A. G. Comparativo de algoritmos de estabilização em sistemas de telereabilitação robótica bilateral. In: **I Simpósio do Programa de Pós-Graduação em Engenharia Mecânica da EESC-USP**. São Carlos: , 2016.

CONSONI, L. J.; SIQUEIRA, A. A. G.; KREBS, H. I. Compensating for Telecommunication Delays during Robotic Telerehabilitation. In: **15th IEEE Conference on Rehabilitation Robotics**. London: , 2017.

COWAN, B.; KAPRALOS, B. A Survey of Frameworks and Game Engines for Serious Game Development. In: **IEEE International Conference on Advanced Learning Technologies**. : IEEE, 2014. p. 662–664. ISBN 978-1-4799-4038-7.

FORRESTER, L. W. et al. Ankle training with a robotic device improves hemiparetic gait after a stroke. **Neurorehabilitation and neural repair**, 2011. v. 25, n. 4, p. 369–77, 2011. ISSN 1552-6844.

GONÇALVES, A. C. B. F. et al. Serious games for assessment and rehabilitation of ankle movements. In: **IEEE 3rd International Conference on Serious Games and Applications for Health**. Rio de Janeiro: IEEE, 2014. p. 1–6. ISBN 9781479948239.

HALL, B. **Beej's Guide to Network Programming**. : Jorgensen Publishing, 2011. 142 p.

HOGAN, N. Impedance Control: An Approach to Manipulation: Part I—Theory. **Journal of Dynamic Systems, Measurement, and Control**, 1985. v. 107, n. 1, p. 1, 1985. ISSN 00220434.

_____. Robot-Aided Neuro-Recovery. **Mechanical Engineering**, 2014. v. 136, n. 9, p. S3–S5, 2014.

JARDIM, B.; SIQUEIRA, A. a. G. Development and analysis of series elastic actuators for impedance control of an active ankle-foot orthosis. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, 2013. v. 36, n. 3, p. 501–510, 2013. ISSN 1678-5878.

JARRASSÉ, N.; CHARALAMBOUS, T.; BURDET, E. A Framework to Describe, Analyze and Generate Interactive Motor Behaviors. **PLoS ONE**, 2012. Public Library of Science, v. 7, n. 11, 1 2012. ISSN 19326203.

KIM, J.; CHANG, P. H.; PARK, H. S. Two-channel transparency-optimized control architectures in bilateral teleoperation with time delay. **IEEE Transactions on Control Systems Technology**, 2013. v. 21, n. 1, p. 40–51, 2013. ISSN 10636536.

KREBS, H. I. et al. A paradigm shift for rehabilitation robotics. **IEEE Engineering in Medicine and Biology Magazine**, 2008. v. 27, n. 4, p. 61–70, 2008. ISSN 07395175.

LANINI, J. et al. Teleoperation of two six-degree-of-freedom arm rehabilitation exoskeletons. In: **IEEE International Conference on Rehabilitation Robotics**. Singapore: , 2015. p. 514–519. ISBN 9781479918072. ISSN 19457901.

MARCHAL-CRESPO, L.; REINKENSMEYER, D. J. Review of control strategies for robotic movement training after neurologic injury. **Journal of Neuroengineering and Rehabilitation**, 2009. v. 6, n. 1, p. 20, 2009. ISSN 1743-0003.

MESSAOUDI, F.; SIMON, G.; KSENTINI, A. Dissecting games engines: The case of Unity3D. In: **International Workshop on Network and Systems Support for Games (NetGames)**. : IEEE, 2015. v. 2016-Janua, p. 1–6. ISBN 978-1-5090-0068-5. ISSN 21568146.

NIEMEYER, G.; SLOTLIN, J.-J. E. Telemanipulation with Time Delays. **The International Journal of Robotics Research**, 2004. v. 23, n. 9, p. 873–890, 2004. ISSN 0278-3649.

NOVAK, D. et al. Increasing motivation in robot-aided arm rehabilitation with competitive and cooperative gameplay. **Journal of neuroengineering and rehabilitation**, 2014. v. 11, p. 64, 2014. ISSN 1743-0003.

- NOVAK, D.; NAGLE, A.; RIENER, R. Can two-player games increase motivation in rehabilitation robotics? **Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14**, 2014. p. 447–454, 2014. ISSN 21672148.
- PASQUAL, T. B.; CAURIN, G. A. d. P.; SIQUEIRA, A. A. G. Serious Game Development for Ankle Rehabilitation Aiming at User Experience. In: **IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob 2016)**. 2016.
- PEÑA, G. G. et al. Neural Network-based EMG-driven Model for Patient Torque Estimation Applied to Rehabilitation Robotics. In: **The 20th World Congress of the International Federation of Automatic Control**. Toulouse: , 2017.
- PIZZOLATO, C. et al. Real-time inverse kinematics and inverse dynamics for lower limb applications using OpenSim. **Computer Methods in Biomechanics and Biomedical Engineering**, 2016. Taylor & Francis, n. November, p. 1–10, 2016. ISSN 1025-5842.
- RODRÍGUEZ-SEDA, E. J. Passive Transparency Compensation for Bilateral Teleoperators with Communication Delays. 2015. v. 2015, 2015.
- RODRÍGUEZ-SEDA, E. J.; LEE, D.; SPONG, M. W. Experimental comparison study of control architectures for bilateral teleoperators. **IEEE Transactions on Robotics**, 2009. v. 25, n. 6, p. 1304–1318, 2009. ISSN 15523098.
- ROY, A. et al. Robot-Aided Neurorehabilitation: A Novel Robot for Ankle Rehabilitation. **IEEE Trans Robotics**, 2009. v. 25, n. 3, p. 569–582, 2009. ISSN 1552-3098.
- SANTOS, W. M. dos; SIQUEIRA, A. a. G. Robust Torque Control Based on H-infinity criterion. In: **IEEE International Conference on Biomedical Robotics and Biomechatronics**. 2014. p. 644–649. ISBN 9781479931279.
- SARTORI, M. et al. Modeling and simulating the neuromuscular mechanisms regulating ankle and knee joint stiffness during human locomotion. **Journal of Neurophysiology**, 2015. v. 114, n. 4, p. 2509–2527, 2015. ISSN 0022-3077.
- SETH, A. et al. OpenSim: a musculoskeletal modeling and simulation framework for in silico investigations and exchange. **Procedia IUTAM**, 2011. v. 2, p. 212–232, 1 2011. ISSN 2210-9838.
- SUZNJEVIC, M. U. o. Z.; SALDANA, J. U. o. Z. **Delay Limits for Real-Time Services**. 2015. Disponível em: <<https://tools.ietf.org/id/draft-suznjevic-dispatch-delay-limits-00.html>>.
- VIVIAN, M.; REGGIANI, M.; SARTORI, M. Experimentally-based optimization of contact parameters in dynamics simulation of humanoid robots. In: **IEEE International Conference on Robotics and Automation**. 2013. p. 1643–1648. ISBN 9781467356411. ISSN 10504729.
- WANG, X.; LI, J. A tele-rehabilitation system with bilateral haptic feedback to both the therapist and the patient via time-delay environment. **World Haptics Conference**, 2013. p. 331–334, 2013.

World Health Organization. **Global burden of stroke**. 2004. 50 - 51 p.

Appendices

APPENDIX A

Additional Simulations

Before the actual tests with bilateral teleoperation algorithms, simulations of a complete virtual model (Figure 30), created in *Scilab/Xcos* software with *Coselica*¹ toolbox, were performed. Their focus was mainly in performance comparison between wave variables and small-gain based force saturation, and also serve this work as a compensation for the lack of real tests on the latter method.

Figure 30: *Scilab/Xcos* + *Coselica* simulator for teleoperation: a sinoidal force signal of unitary amplitude enters the master system (single mass), and resulting position and velocity are transmitted to the slave side, passing through local and remote compensator superblocks, that could hold any implementation; a compliance control is applied to the slave system interacting with environment, and a feedback force signal is generated, making its way back to master; forward and backward signals are delayed by artificial latency superblocks (CONSONI; SIQUEIRA, 2016)

As in the experiments described in chapter 4, artificial delay of the model with no compensation applied was raised to the point that it became unstable (Figs. 31a,32a). From this point, the simulations were re-executed separately for implementations of the two compensators, and the same results were plotted again and compared to the original as in Figs. 31,32, respectively, for wave variables and force saturation.

¹<https://atoms.scilab.org/toolboxes/coselica>

Figure 31: Results obtained for a 100 seconds simulation of the modeled teleoperation system: (a) unstable, with no delay compensation, as reference; (b) corrected by wave variables application. Blue line represent master measurements and red slave ones.

Figure 32: Results obtained for a 100 seconds simulation of the modeled teleoperation system: (a) unstable, with no delay compensation, as reference; (b) corrected by force saturation application. Blue line represent master measurements and red slave ones.

As seen, both techniques are capable of stabilizing an unstable system, with their respective, somehow expected, drawbacks: wave variables introduce considerable damping effect to movement, making synchronization less transparent; force saturation avoids that issue, but does not provide the same efficiency in matching local and remote trajectories.

The process of implementation for this simulator and data obtained from it resulted in an article Consoni e Siqueira (2016) published in a local university conference.

APPENDIX B

Available Resources

This appendix covers results from this work that were made available in online *Web* pages to any interested person.

B.1 Open Code Repositories

It would not make sense to push for a universal software solution without making it easily accessible. With that in mind, there was a concern for making the produced code open source, whenever possible, without risking author’s intellectual property. Thus, most of the work developed here was published to *GitHub* repositories, and distributed under the LGPL license.

Actually, produced software was split in 3 project repositories, in order to facilitate code reuse:

- ❑ *Platform Utils*¹: multiplatform abstractions for low-level resources, such as thread management, memory sharing and asynchronous IP connections;
- ❑ *Robot Control Library*²: utilities and abstractions for generic and configurable robot control, built on top of *Platform Utils*;
- ❑ *RobRehabSystem*³: set of described robot control and server applications, built on top of *Robot Control Library* and *Platform Utils*;
- ❑ *Unity Rehab Game Framework*⁴: the *Unity3D* framework for rehabilitation games compatible with *RobRehabSystem*, containing the two games used in this work (*Double Pong* and *Box Clash*) as example projects.

¹<https://github.com/Bitiquinho/Platform-Utils>

²<https://github.com/Bitiquinho/Robot-Control-Library>

³<https://github.com/Bitiquinho/RobRehabSystem>

⁴<https://github.com/Bitiquinho/Unity-Rehab-Game-Framework>

B.2 Documentation

Apart from simply disclosing platform's code, some time was also dedicated for writing proper documentation about the software. *Web* pages describing utilization of available methods, data structures and configuration options, as well as the rationale behind them, were generate using *Doxygen*⁵ for *Platform Utils*⁶ and *Robot Control Library*⁷. Also, a wiki⁸ explaining more top-level information for developers was published for *RobRehabSystem* using *GitHub* Project Pages.

⁵<http://www.stack.nl/~dimitri/doxygen/index.html>

⁶<https://bitiquinho.github.io/Platform-Utils/annotated.html>

⁷<https://bitiquinho.github.io/Robot-Control-Library/annotated.html>

⁸<https://github.com/Bitiquinho/RobRehabSystem/wiki>