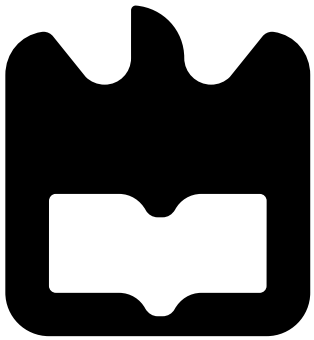
Universidade de Aveiro Departamento de Engenharia Mecânica
2012

Pedro Miguel Batista Cruz

Sistema de aquisição de dados por interface háptica
Haptic interface data acquisition system



Pedro Miguel Batista Cruz

Sistema de aquisição de dados por interface háptica
Haptic interface data acquisition system

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestrado em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro e de Filipe Miguel Teixeira Pereira da Silva, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

O júri / The jury

Presidente / President	**Prof. Doutor Jorge Augusto Fernandes Ferreira** Professor Auxiliar da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor António Manuel Ferreira Mendes Lopes** Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado da Universidade de Aveiro (orientador)
	Prof. Doutor Filipe Miguel Teixeira Pereira da Silva Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (co-orientador)

Agradecimentos / Acknowledgements

Em primeiro lugar, deixo aqui o meu sincero agradecimento ao Prof. Dr. Vítor Santos, o meu orientador neste trabalho. O seu contagiante entusiasmo pela ciência e a sua incansável luta pela robótica em Portugal foram um marco importante na minha vida, do qual nunca me esquecerei. Obrigado Professor!
Do mesmo modo agradeço ao Prof. Dr. Filipe Silva, o meu coorientador, pelo seu interesse e abertura, e cujo *know-how* foi bastante valioso.
Um agradecimento a todos aqueles com quem colaborei e me ajudaram, com especial foco para todos os membros, passados e atuais, do Laboratório de Automação e Robótica do Departamento de Engenharia Mecânica da Universidade de Aveiro. Agradeço ao Ricardo Godinho e ao António Festas, pela sua disponibilidade. Em especial destaque, quero deixar um grande agradecimento ao Jorge Almeida, ao Miguel Oliveira e ao Procópio Stein, esses dois grandes Doutores, por tudo o que aturaram durante este trabalho e pela excelente equipa de trabalho que formam.
Gostaria também de agradecer a cedência do *joystick* háptico ao Prof. Dr. Relvas e ao Prof. Dr. Simões, cuja disponibilidade foi exemplar.
Gostaria ainda de agradecer o apoio incondicional dos meus pais, do meu irmão, e de toda a minha família. Vocês são a massa crítica que deu origem a quem eu sou hoje e bússola para o que pretendo ser amanhã.
Deixo ainda um agradecimento a todos os que não mencionei, mas que no entanto sabem que não são menos importantes para mim.
Por fim, o meu maior agradecimento é dedicado à Magda Carmo. Sem ela, nada disto tinha sido conseguido e eu já teria com certeza dado em doido. O seu apoio foi crucial, e permitiu-me concretizar um dos meus sonhos de infância. Obrigado, amor!

Palavras-chave

Háptica, interface háptica, teleoperação, robótica humanoide, ensino tele-cinestésico, aprendizagem robótica.

Resumo

Neste trabalho é apresentada uma interface háptica com realimentação de força para a teleoperação de um robô humanoide e que aborda um novo conceito destinado à aprendizagem por demonstração em robôs, denominado de ensino tele-cinestésico. A interface desenvolvida pretende promover o ensino cinestésico num ambiente de tele-robótica enriquecido pela virtualização háptica do ambiente e restrições do robô. Os dados recolhidos através desta poderão então ser usados em aprendizagem por demonstração, uma abordagem poderosa que permite aprender padrões de movimento sem a necessidade de modelos dinâmicos complexos, mas que geralmente é apresentada com demonstrações que não são fornecidas tele-operando os robôs. Várias experiências são referidas onde o ensino cinestésico em aprendizagem robótica foi utilizado com um sucesso considerável, bem como novas metodologias e aplicações com aparelhos hápticos. Este trabalho foi realizado com base na plataforma proprietária de 27 graus-de-liberdade do Projeto Humanoide da Universidade de Aveiro (PHUA), definindo novas methodologias de comando em tele-operação, uma nova abordagem de *software* e ainda algumas alterações ao *hardware*. Um simulador de corpo inteiro do robô em *MATLAB SimMechanics* é apresentado que é capaz de determinar os requisitos dinâmicos de binário de cada junta para uma dada postura ou movimento, exemplificando com um movimento efectuado para subir um degrau. Irá mostrar algumas das potencialidades mas também algumas das limitações restritivas do *software*. Para testar esta nova abordagem tele-cinestésica são dados exemplos onde o utilizador pode desenvolver demonstrações interagindo fisicamente com o robô humanoide através de um joystick háptico *PHANToM*. Esta metodologia irá mostrar que permite uma interação natural para o ensino e perceção tele-robóticos, onde o utilizador fornece instruções e correções funcionais estando ciente da dinâmica do sistema e das suas capacidades e limitações físicas. Será mostrado que a abordagem consegue atingir um bom desempenho mesmo com operadores inexperientes ou não familiarizados com o sistema. Durante a interação háptica, a informação sensorial e as ordens que guiam a uma tarefa específica podem ser gravados e posteriormente utilizados para efeitos de aprendizagem.

Keywords Haptics, haptic interface, teleoperation, humanoid robotics, tele-kinesthetic teaching, robot learning.

Abstract In this work an haptic interface using force feedback for the teleoperation of a humanoid robot is presented, that approaches a new concept for robot learning by demonstration known as tele-kinesthethic teaching. This interface aims at promoting kinesthethic teaching in telerobotic environments enriched by the haptic virtualization of the robot's environment and restrictions. The data collected through this interface can later be in robot learning by demonstration, a powerful approach for learning motion patterns without complex dynamical models, but which is usually presented using demonstrations that are not provided by teleoperating the robots. Several experiments are referred where kinesthetic teaching for robot learning was used with considerable success, as well as other new methodologies and applications with haptic devices. This work was conducted on the proprietary 27 DOF University of Aveiro Humanoid Project (PHUA) robot, defining new wiring and software solutions, as well as a new teleoperation command methodology. A MATLAB SimMechanics full body robot simulator is presented that is able to determine dynamic joint torque requirements for a given robot movement or posture, exemplified with a step climbing application. It will show some of the potentialities but also some restricting limitations of the software. To test this new tele-kinesthetic approach, examples are shown where the user can provide demonstrations by physically interacting with the humanoid robot through a PHANToM haptic joystick. This methodology will show that it enables a natural interface for telerobotic teaching and sensing, in which the user provides functional guidance and corrections while being aware of the dynamics of the system and its physical capabilities and / or constraints. It will also be shown that the approach can have a good performance even with inexperienced or unfamiliarized operators. During haptic interaction, the sensory information and the commands guiding the execution of a specific task can be recorded and that data log from the human-robot interaction can be later used for learning purposes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Framework

The work presented in this dissertation has been developed within the framework of a new concept for robot learning by demonstration, combining teleoperation and kinesthethic teaching around an haptic interface using force feedback.

Robot learning by demonstration is a powerful approach in order to automate the tedious manual programming of robots, to learn locomotion patterns without complex dynamical models and to reduce the complexity of high dimensional search spaces [1; 2]. The demonstrations are typically provided by teleoperating the robot, or by vision and motion sensors recordings of the user doing the task. Recent progresses aim to provide more user-friendly interfaces, such as kinesthetic teaching [3; 4; 5].

The main focus of this work is centered on the University of Aveiro Humanoid Project (PHUA), that currently consists of a 27 degree-of-freedom (DOF) whole body small-sized humanoid platform based on an hybrid actuation system that combines servomotor powered joints with elastic elements, providing an energy storage/recover mechanism [6; 7].

The robot's anthropometric proportions and ranges served as inspiration towards a platform that permits walking with straight support legs by incorporating a compliant foot with a passive toe joint, bringing the robot's foot kinematics closer to the humans.

The control of full-body humanoid robots is an extremely complex problem, especially for tasks such as locomotion. This complexity arises from the many degrees-of-freedom involved, the lack of precise models, the non-closed form for robot control, the dependency on the environment conditions, the compliance of actuators, the variable stiffness of links, the backlash of transmissions or the noise in internal sensors, along with many others.

The same problems have been faced in the PHUA project. Although conceived with care and with many components previously simulated before effective construction, the platform suffers from many of the limitations mentioned above. Tasks like walking or stepping/squatting, or even manipulation or grasping, so natural in humans, become very difficult in these robots with all their mechanical and controlling limitations.

To tackle the pre-programmed control issues, the project took a new direction towards robot learning, but with a new approach: tele-kinesthetic teaching. Basically, it is teleoperation with an haptic interaction with the robot's environment and information from its state.

Haptic interaction can be defined as manipulation using the human sense of touch. The term "haptics" arises from the Greek root *haptikos*, meaning "able to grasp or perceive". This kind of interaction with computers implies the ability to use our natural sense of touch to feel and manipulate computed quantities [8].

The haptic interaction presented in this work is within the scope of an area known as computer haptics, "algorithms and software associated with generating and rendering the touch and feel of virtual objects" (analogous to computer graphics), but also slightly crossing into machine

haptics, defined as "the design, construction, and use of machines to replace or augment human touch" [9]. This approach uses the field of computer haptics to feedback the operators of the real world conditions through the use of machine haptics, which, in this case is not just an haptic interface device, but also the robot itself.

The field of haptic interaction and perception is a new way of human-machine interface, and one that may revolutionize the way we interact with machines, giving them a new "feel". The human haptic system, with its tactile, kinesthetic, and motor capabilities together with the associated cognitive processes, presents a uniquely bidirectional information channel to our brains that remains somewhat unexplored.

Given the increasing quantities and types of information that surrounds us, there is a critical need to explore new ways to interact with this ever-flowing new information. In order to be efficient in this interaction, it is essential that we utilize all of our sensorimotor capabilities.

In this work, an approach is suggested where the user provides demonstrations by physically interacting with the humanoid robot through an haptic interface [10]. The proposed methodology enables a natural interface for tele-kinesthetic teaching and sensing, in which the user provides functional guidance and corrections, while being aware about (i.e., able to "feel") the dynamics of the system, its physical capabilities and/or constraints.

The combination of haptic and visual (visuohaptic) information has been confirmed as a good memorizing mechanism for humans [11], which gives confidence that such an approach can perform well in demonstrations even with inexperienced operators.

Additionally, during the demonstration phase, the sensory information and the commands guiding the execution of a specific task can be recorded. All the data logged from the human-robot interaction can be later used for learning purposes.

This approach goes beyond previous research on teaching by demonstration that is unable to raise the level of bidirectional human-robot interaction. Instead, it refers to a deeper relationship between the user and the robot who share control to reach common goals using the same measures of outcome.

1.2 Objectives

The objective of this work is to present new programming and control concepts for humanoid robots by means of an haptic interface, specifically the development of a force feedback haptic control and sensing mechanism, aimed at the robot's perception/actuation data logging.

It is an objective to document current robot learning methodologies involving interactive kinesthetic learning and teleoperation. As this work is centered around humanoid platforms, it is also critical to document their current state and progresses.

As it was unsure if the platform could perform complex tasks such as walking, stepping or squatting, a dynamic model is proposed as an easy-to-deploy solution, using a toolbox for a commercial software, *MATLAB*.

It is also proposed the development of a user-friendly interface for robot control and an haptic interface, providing the means for expert data logging when used by inexperienced operators. This requires deep knowledge of the haptic device's programming library and the definition of a control structure for the platform. To ensure the operator perceives the current state of the robot, it is required to develop ways of translating that information into force feedback, through the haptic device. The collected data can then be used to obtain a model of the task, using data as the position and force information, among others.

Lastly, it was a primary objective to develop haptic tele-kinesthetic demonstrations that could show the potential of this work's approach.

1.3 State of the Art

1.3.1 Principles of Haptic Interaction

In the early 20th century, psychophysicists introduced the word "haptics" to label the subfield of their studies that addressed human touch-based perception and manipulation. In the 1970s and 1980s, significant research efforts in a completely different field, robotics, also began to focus on manipulation, perception and interaction by touch.

The idea of using touch as a means of communication grew in the late 1980s through the late 1990s [12; 13]. Since then it became popular and, in the late years, the number of published papers with the word "haptic" in them reached thousands a year.

The word "haptics" refers to the capability to sense a natural or synthetic mechanical environment through touch. Haptics also includes kinesthesia (or proprioception), the ability to perceive one's body position, movement and weight. It has become common to speak of the "haptic channel" to collectively designate the sensory and motor components of haptics. This is because certain anatomical parts (in particular the hand) are organs in which perceiving the world and acting upon it are activities that take place together. Tactile and kinesthetic channels work together to provide humans with means to perceive and act on their environment.

With the exception of input devices, such as the mouse, computer interaction relies on skills similar to those needed for using typewriters. Haptic interfaces, with a principle as exemplified in Figure 1.1, may be viewed as an approach to address this limitation. It is thus possible to classify haptics into the area of human-computer interfaces. Unlike traditional interfaces that provide visual and auditory information, haptic interfaces generate mechanical signals that stimulate human kinesthetic and touch channels [14].

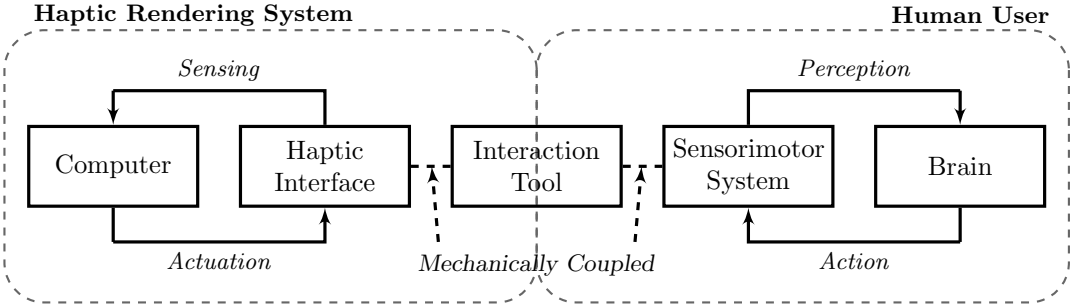


Figure 1.1: The principle of haptic interaction (adapted [15]).

These signals are synthesized through computer algorithms, in a process know as "*haptic rendering*", schematically represented on Figure 1.2. Haptic rendering algorithms compute the correct interaction forces between the haptic interface representation inside the virtual (or virtualized) environment and the objects populating the environment. Moreover, haptic rendering algorithms ensure that the haptic device correctly renders such forces on the human operator. Normally, haptic rendering is accompanied by simultaneous graphical rendering in what is more properly referred to as a visual/haptic interface. Unlike graphical rendering which can satisfy the eye at update rates of 30 frames per second or even less, haptic rendering must be done at rates around a kilohertz, the human kinesthetic sensing frequency minimum threshold, in order for the rendering to be seamless and natural [16].

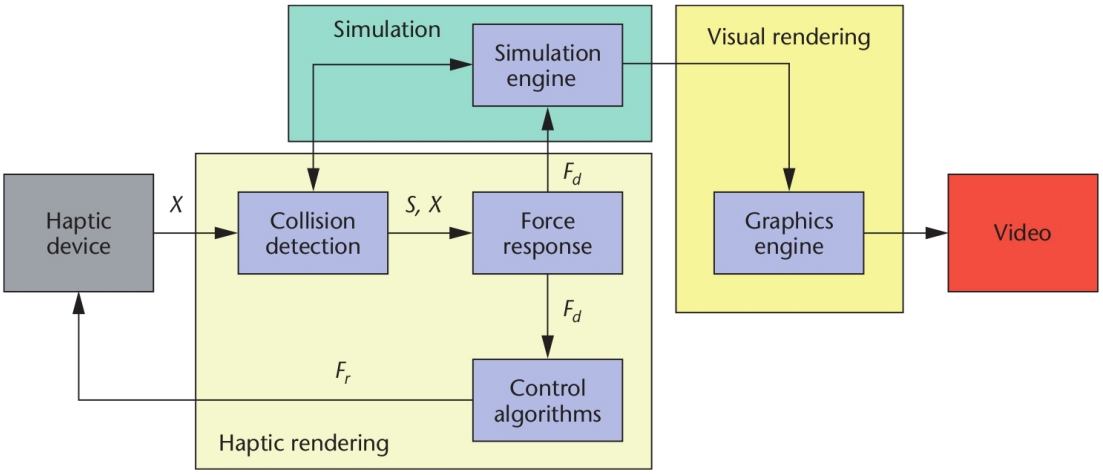


Figure 1.2: An example of haptic rendering architecture, split into three main blocks. Collision-detection algorithms provide information about contacts S occurring at position X and the objects in the virtual environment. Force-response algorithms return the ideal interaction force F_d while control algorithms return a force F_r to the user [16].

Nowadays, haptics can be found in a great variety of applications, such as video games, military war games, vehicle simulators, touchscreens, arts and creation, computer aided design/engineering/manufacturing (CAD/CAE/CAM), assisted surgery machines and, of course, robotics (telerobotics) [15].

In video games or touchscreens, haptic sensations are created by actuators, or motors, which create a vibration. Those vibrations are managed and controlled by embedded software, and integrated into device user interfaces and applications. The rumble effect in video game controllers or the touch vibration on smartphone dial pads are both examples of haptic effects [17].

For control or virtual reality applications, haptics goes beyond simple rumble or vibration signals. These signals are no other than energy, so for applications that require a deeper human-machine connection, force, torque, vibration and visual signals are combined. In this sense, we can say haptic interface devices behave like small robots that exchange mechanical energy with the user. The term *device-body interface* is frequent to highlight the physical connection between operator and device through which energy is exchanged. Although these interfaces can be in contact with any part of the operator's body, hand interfaces have been the most widely used and developed systems to date [16].

1.3.2 Haptic Interface Devices

Nowadays, a number of different haptic interface commercial and academic solutions coexist, for numerous applications. However, within the devices with relevance for the context of this work, a distinction between haptic interface devices must be introduced: their intrinsic mechanical behaviour.

On one hand, impedance based haptic devices simulate mechanical impedance, i.e., they read position and output force. Simpler to design and much cheaper to produce, these impedance-type architectures are the most common within both academic and commercial worlds.

On the other hand, admittance haptic devices simulate mechanical admittance, reading force and outputting position. Admittance-based devices, such as the 3DOF Haptic Master [18], represented i Figure 1.3, are generally used for applications requiring high forces in a large workspace. This device has a maximum exertable force of 250 N with a sensitivity of 0.01 N.

Additionally, haptic devices can be broadly classified into: exoskeletons and stationary devices; gloves and wearable devices; point-sources and specific task devices. Haptic exoskeletons/gloves are mainly admittance-based devices, not very frequent in telerobotics experiments. So from this point on, the term "haptic device" refers to point-source devices, for their preponderance in teleoperation and robotics.

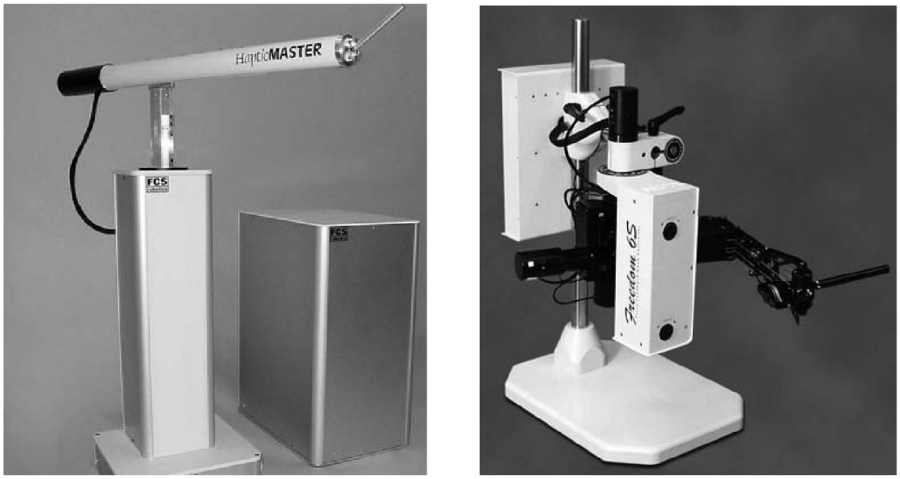


Figure 1.3: The FCS Control Systems admittance-based HapticMaster [18] (left) and the MTB Technologies impedance-based Freedom-6S [14] (right) devices.

Still in the 1990s, impedance-based commercial solutions began to emerge, some with intricate mechanics, such as the MTB Technologies Freedom-6S, in Figure 1.3, a 6DOF device (extensible to 7 *via* accessories) built with hybrid kinematics. It is characterized by a wide dynamic range and six-axis static and dynamic balancing. Its design was "wrist partitioned", i.e., the position and orientation stages each being parallel mechanisms. The position stage is directly driven and the orientation stage is driven through remotizing tendons with a differential tensioning technique that operates with low tension and hence low friction.

Today, commercial solutions for haptic interface devices exist in many different physical dimensions and specifications. One of the most popular is the desktop device, due to its compact form factor and portability. These solutions provide a force feedback system with affordable price, being very popular in the academia. The most common devices are the SensAble PHANToM OMNI and Desktop models, shown in Figure 1.4.

These are sometimes called "pen-grasping" joysticks, in the sense that the operator interacts with the device *via* a pen-like end-effector. With a workspace of around 27 dm^3 and force feedback magnitudes of 3.3 N and 7.4 N, respectively [19], these devices are well known for their capstan drive, which allows for their construction. They are mostly found in 3D CAD and 3D

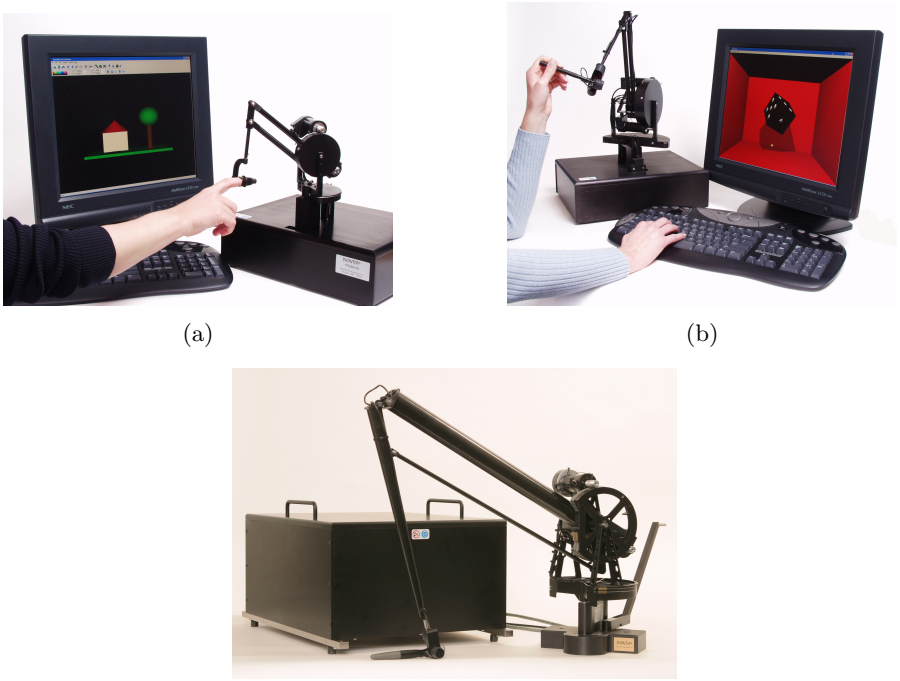
moulding applications, for the pen-like interface emulates the tool usage mechanisms, making the learning curve steeper.

Figure 1.4: The SensAble PHANToM OMNI (left) and PHANToM Desktop (right) haptic impedance-based desktop devices [20; 19].

Within the desktop solutions, there is a new subdivision based on a parallel configuration (Delta). These devices became popular for allowing 6DOF force feedback/position reading in a very compact mechanism, but sacrificing workspace dimensions. The Virtuose 6D Desktop from Haption is a 6DOF spherical joystick with 6DOF force feedback, within a volume of 120 mm in diameter and 35° in three directions at the center, with a resolution in position at about 2 mm. Its maximum exertable force is 10 N and its maximum exertable torque is 0.5 Nm [21]. Another example are the Force Dimension products, also based on a parallel kinematics mechanism. The Omega [22] and Delta [23] offer greater force feedback, but with a slightly smaller workspace. These devices are represented in Figure 1.5.

Figure 1.5: The Haption Virtuose 6D Desktop (left) , the Force Dimension Omega (middle) and Delta (right) haptic impedance-based desktop devices [21; 22; 23].

When bigger workspaces are required, these devices are not suitable. The Sensable PHANToM Premium line is the company's ultimate line of products. The PHANToM Premium 1.0, 1.5 and 3.0, in Figure 1.6, are shoulder pivoting devices with larger force feedback levels when compared to their desktop variants, but with the exact same mechanism type, allowing for 3DOF force feedback and a 6DOF position reading. The PHANToM Premium 3.0 and the 1.5 model both have a "HIGH FORCE" variant that is capable of rendering 6DOF forces/torques.



(a) (b)

(c)

Figure 1.6: The SensAble PHANToM Premium 1.0, 1.5 and 3.0 haptic impedance-based desktop devices [24].

Again from Haption, we can find two models with large workspaces. The Virtuose 6D35-45 offers force-feedback on all 6 degrees-of-freedom together with a large workspace (450 mm). Its maximum exertable force is 35 N and its maximum exertable torque is 3 Nm. Modular in design, it can be purchased as a 3DOF device and later upgraded to 6DOF [25]. The Virtuose 6D40-40 is a new generation master arm for teleoperation with force-feedback. It is very similar to the 6D35-45 model in workspace limitations (400 mm workspace), but allows for a much higher force/torque feedback. Its maximum exertable force is 100 N and its maximum exertable torque is 10 Nm [26]. Both models are shown in Figure 1.7.

Figure 1.7: The Haption Virtuose 6D35 (left) and 6D40 (right) haptic devices [25; 26].

New kinds of haptic devices have appeared in the later years. One of the most promising is the dual-pantograph arrangement device. These devices are 6DOF joysticks with full 6DOF force/torque feedback designed specifically to control industrial serial robots equipped with force sensors. The device is made combining two pantographs (Delta type actuators) driven by motors at its shoulders and another at the waist. The wand type end-effector is connected to both end points of each pantograph through universal joints (U-joints). The yaw about the wand axis gives its extra DOF. However, this kind of joysticks still possess very limited workspaces, so their applicability is still somewhat limited. Figure 1.8 depicts two examples of this technology.

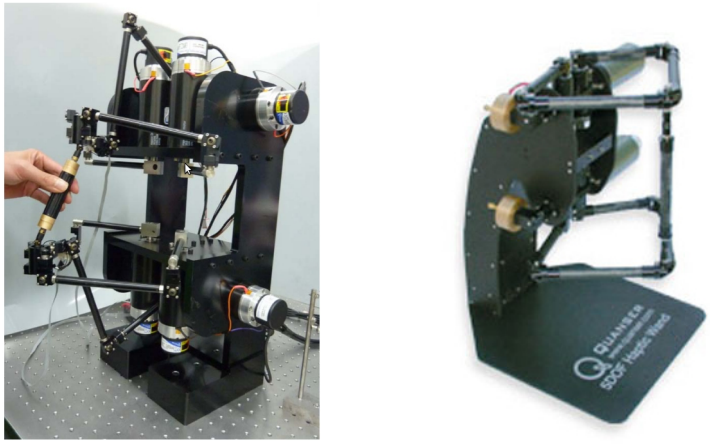


Figure 1.8: A double pantograph style haptic device proposal (left) [27] and the Quanser Haptic Wand (right) commercial haptic device [28].

Another area quickly growing is the dual-device operation, with a great focus in the medical industry. The Force Dimension Sigma, in Figure 1.9, is a dual joystick device that covers the natural range of motion of the human hand and it is compatible with bi-manual teleoperation console design. Along with a full gravity compensation and driftless calibration, it possesses a good level of forces, torques and a high precision on all its 7DOF [29], making it one the most accomplished master devices available.

Figure 1.9: The Force Dimension Sigma haptic device in a medical application [29].

Virtual environment (VE) manipulation/navigation is another field were dual configurations have great impact. The DLR (German Aerospace Center) bimanual haptic device, represented in Figure 1.10, is a good example of the development of this field [30]. The device is composed of two 6DOF DLR/KUKA Light Weight Robot (LWR) arms and an extra DOF at the handles. The LWRs have similar dimensions to the human arms and can be operated in torque and position control mode at an update rate of 1 kHz, for a full haptic rendering support. The two robot arms are mounted behind the user, so that the intersecting workspace of the robot and human arms becomes maximized. This system has a 150 N maximum exertable force generation. The haptic system is combined with a motion detect helmet and a haptic glove system for maximum VE immersion. Even though it is primary an impedance-based device, it is also prepared for admittance-based control and a hybrid mode, making it one the few functional devices with this capability. It is considered to be one of the most complex and complete VE haptic devices.

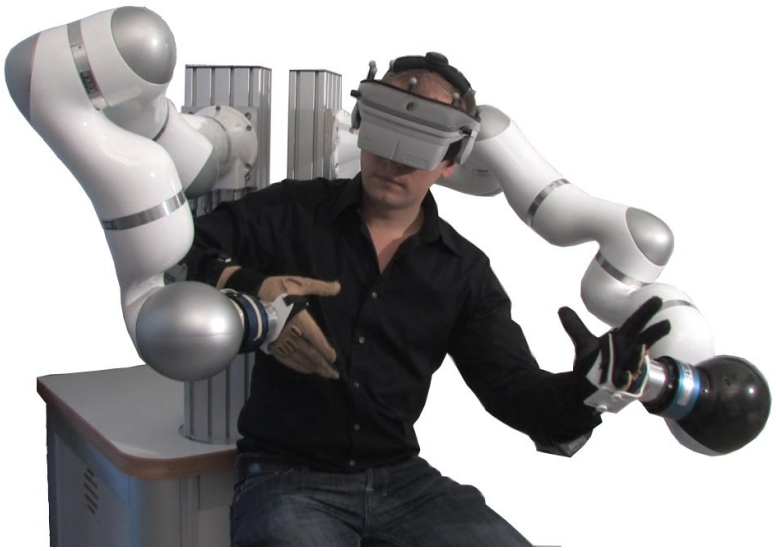


Figure 1.10: The DLR bimanual haptic device [30].

1.3.3 Teleoperation, Kinesthetic Teaching and Learning from Demonstration

The teleoperation of robots is currently a vast, yet still growing, scientific field, with countless publications on the matter. It is a concept that has grown to cover all fields of robotics and into the commercial/industrial worlds. Robot learning is another topic of great interest to the scientific community. New algorithms and techniques are proposed each year, with considerable improvements in every iteration. The use of haptic interfaces for human-machine learning processes study is also well accepted, as the field of haptic guidance is growing [31].

Teleoperation of robots benefits from the extension of their sensory capabilities with the use of haptic interfaces. Usually, telerobotic applications provide the remote user images and navigation sensor data. With additional force feedback, the remote control performance is improved. Small mobile robots have been successfully manipulated using an haptic interface, as shown on Figure 1.11, with the user being aware of the state changes in the robot's environment. Sensors in the car bumpers provide force feedback to the user [32].

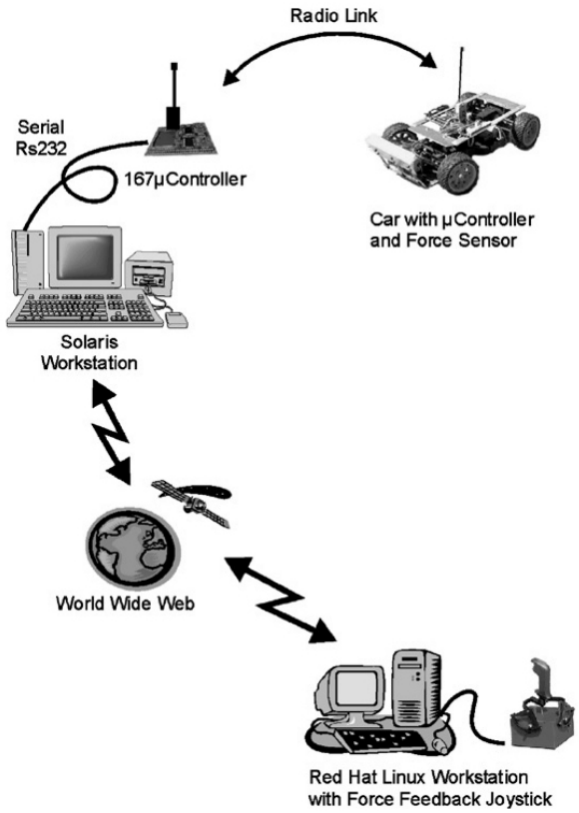


Figure 1.11: Haptic teleoperation of a mobile robot [32].

In robotics, learning from demonstration is also becoming common place. Rather than imitating human actions, the focus is to develop an intuitive and interactive method based on user demonstrations to create human-like, autonomous behaviour for a robot. New paradigms have emerged with the use of haptic systems and robot learning. Skill transfer mechanisms have gained a growing interest in the scientific community. For instance, one experiment constructs a reality-based haptic interaction system for knowledge transfer, by linking a human expert skill with robotic movement in real time, in an handwriting task [33]. The goal was to successfully recreate manipulation motion with a robotic system and transfer haptic forces to an untrained user with an haptic device.

In another experiment, illustrated in Figure 1.12, a human teacher uses an haptic control strategy to teach a humanoid robot collaborative tasks with a human partner [34]. The robot can learn through observation to perform a collaborative manipulation task which is first demonstrated by a user controlling the robot's hand via an haptic interface. Haptic communication reflects more than pure dynamic information on the task. It also includes communication patterns which result from both users, human partner and teleoperated robot, constantly adapting their hand motion to coordinate in time and space their respective motions.

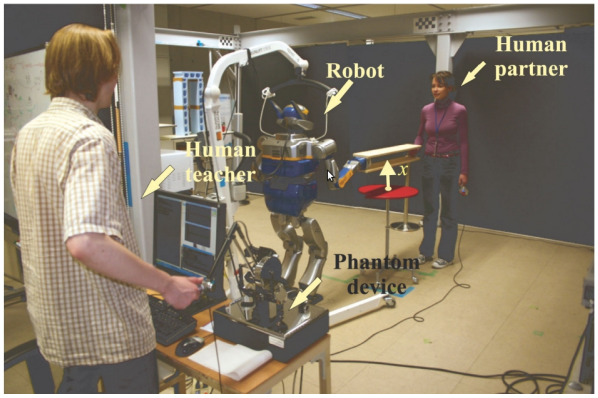


Figure 1.12: Teleoperating a humanoid robot to demonstrate and teach an object lifting task collaboratively with a human user [34].

Studies have shown that the use of force profiles along with position information improves the quality of the robot learning. In one experiment, using a Barret Technology *WAM* manipulator, kinesthetic teaching was used for position recording and an haptic interface was used to trace the force profile of the same tasks, such as ironing a piece of cloth or opening a door, as described in Figure 1.13. Feeding the force profile to the learning algorithm improved the robot's response when autonomously performing those tasks [4].

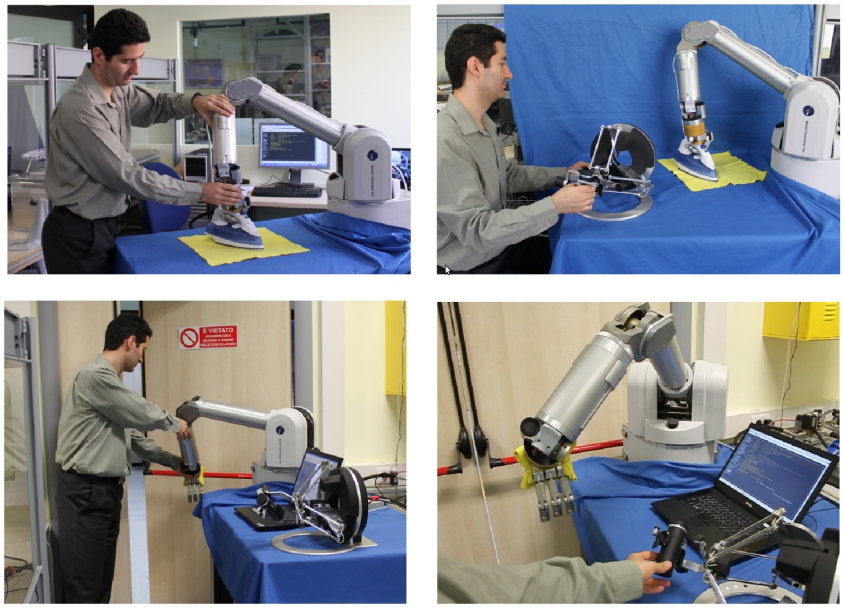
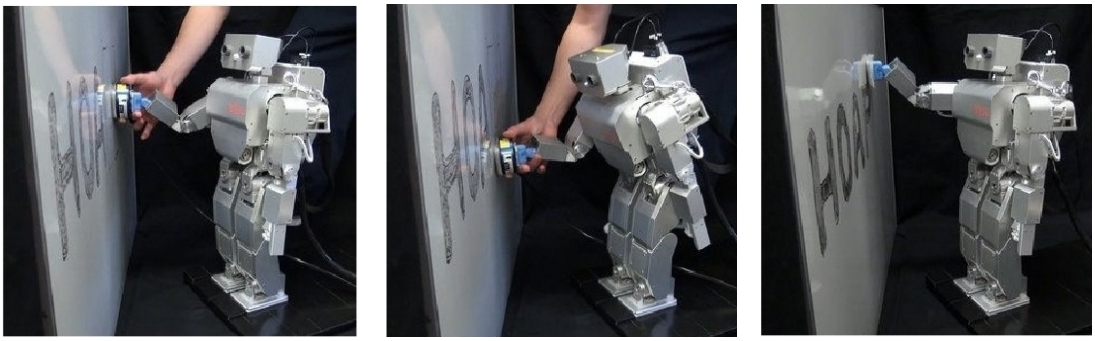


Figure 1.13: Retrieving position and force profiles from different tasks, such as ironing a piece of cloth or opening a door, for teaching purposes [4].

Teaching by kinesthetic manipulation with humanoid robots is also a topic of interest in the academia, with several new approaches successfully implemented at each passing year. One experiment placed a humanoid robot writing on a chalk board while standing up [35]. The humanoid robot upper body training for imitation learning is done via kinesthetic teaching, while the Reaction Null Space method is used for keeping the balance of the robot. During the user demonstration phase, as shown in Figure 1.14(a) and (b), along with the kinematic infromation logging, a force/torque sensor is used to record the exerted forces. During the reproduction, a hybrid position/force controller is used to apply the learned trajectories (position and force-wise) to the end-effector, as seen on Figure1.14(c).



(a) (b) (c)

Figure 1.14: Upper-body humanoid kinesthetic teaching while standing [35].

Learning from demonstration has also been successfully achieved using virtual environments. For instance, a serial robot arm was taught to catch a ball in mid-air by analysing data collected in a virtual world environment, as shown in Figure 1.15, compiled from user inputs [36].

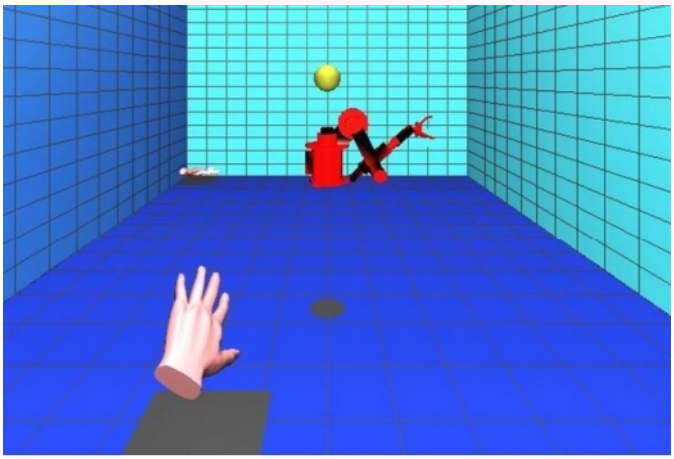


Figure 1.15: Virtual environment used for learning by demonstration with a serial arm catching a ball in mid-air [36].

This dissertation implements a keyword not often used: tele-kinesthetic teaching. This method refers to teleoperation/telerobotics with three-dimensional force feedback using an haptic interface. The forces reproduced are those of the controlled object or equivalent, and are not an input for the teleoperation. This is an important feature around this keyword. However, in literature, the most relevant work found around this keyword approximates the user's hand motion with force by synthesizing EMG signals that are transformed into 3D motion and then

applied to a hanging ball within a cubic frame [37], as shown in Figure 1.16. As the usage of this keyword in this experiment seems to be related to force transfer from human to machine, one must reinforce that the approach suggested in this work implements the opposite strategy.

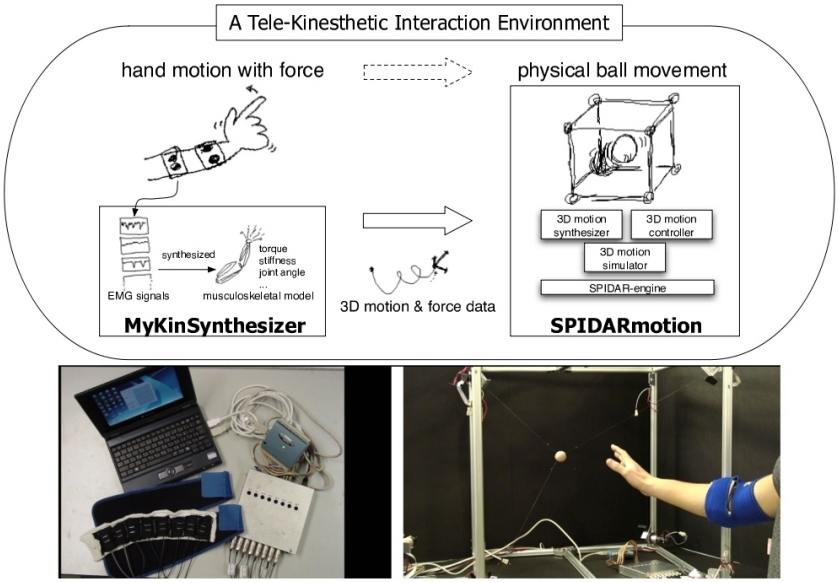


Figure 1.16: Another tele-kinesthetic interaction environment based on EMG signals [37].

Lastly, a new paradigm in kinesthetic teaching must be presented: industrial kinesthetic teaching. Presented in 2012, the Rethink Robotics Baxter, in Figure 1.17, is a dual-arm upper-body humanoid-like robot [38]. Composed of two 7DOF serial elastic actuated arms, an array of cameras and sensors, a quad-core computer running Linux, an LCD touchscreen for human-machine interface and interchangeable grippers, it is the first of its kind. This robot was built to be a low cost solution ($22.000 US) for industrial robotics, but its strength is not on high-precision manipulation. The Baxter possesses two buttons and a knob on each forearm used to navigate the robot user-interface and also to teach kinesthetic frames to the robot, demonstrating how to perform the tasks. This solution does not require a specialized human operator to perform the teaching and the LCD touchscreen provides a more user-friendly interface than the usual code input interfaces.

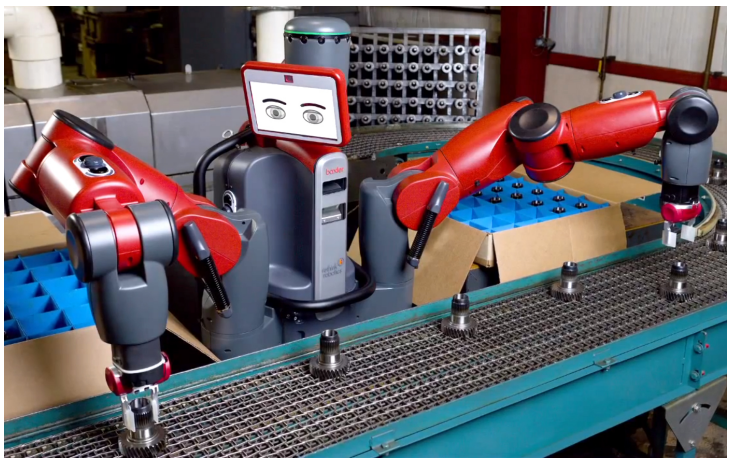


Figure 1.17: The Rethink Robotics Baxter industrial kinesthetic teaching robot.

1.3.4 University of Aveiro Humanoid Project (PHUA)

Started in 2003/2004, this project was the outcome of a joint venture between the Mechanical Engineering Department (DEM) and the Electronics, Telecommunications and Informatics Department (DETI) of the University of Aveiro. Dedicated to the development and integration of hardware and software components, the project made a major step forward with the design and development, in 2009, of a new small-size humanoid platform based on hybrid actuation, as depicted in Figure 1.18. The focus has been on the mechanical design and the selection of actuators and sensors [7; 39; 40; 41]. The humanoid robot's height is around 65 cm, and its weight is approximately 6 kg. It is equipped with an array of two 5000 mAh *LiPo* batteries on the back.

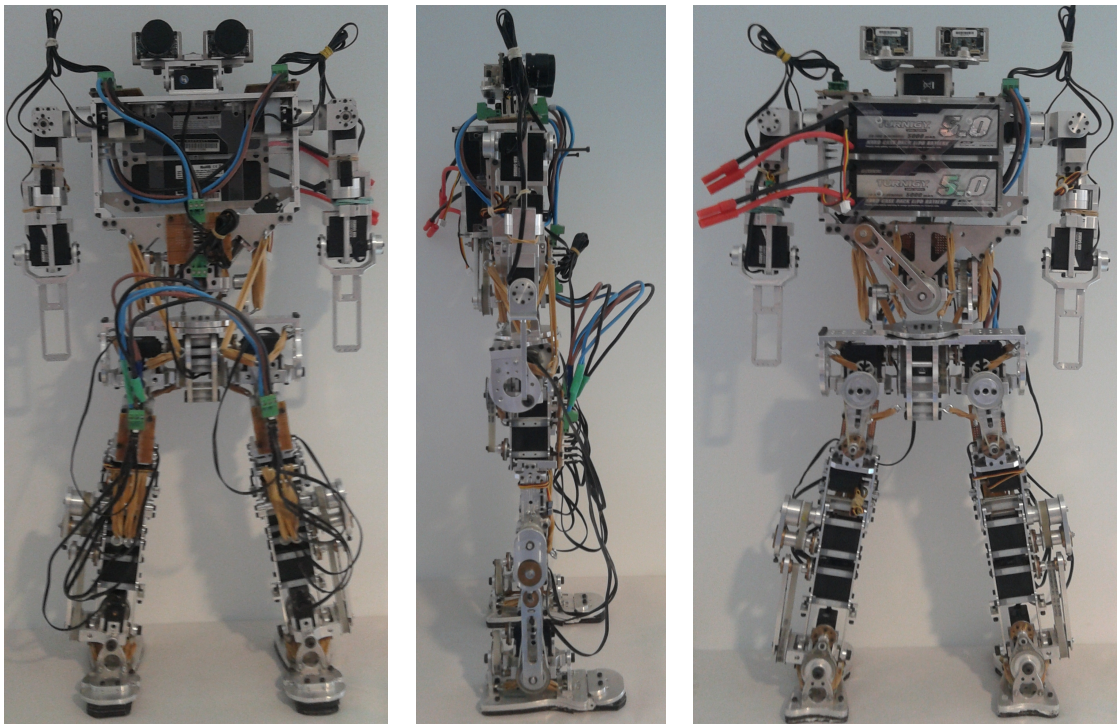


Figure 1.18: Current form of the PHUA robot platform, with the hybrid actuation system and the complete motor wiring installed.

This robot is a proprietary whole-body humanoid platform with a total of 25 active and 2 passive degrees-of-freedom, as shown in Figure 1.19. The joint actuation is composed of *HITEC®* analog and digital servomotors, as distributed in the Table 1.1, combined with elastic elements. The robot uses a belt and pulley adjustable transmission system on the legs and torso, providing torque to the joints with higher requirements.

As illustrated in Figure 1.20, the design of this platform accounts for a hybrid actuation system that combines active joints with adjustable elastic elements (rubber bands), providing a passive mechanism to store and recover energy. Mainly to enhance the system response and also to try lessen the actuator's torque demands, flexible rubber bands are attached between several pair of consecutive links in parallel with the servomotors.

The robot is also equipped with a set of 8 force sensors, 4 on each foot, and a *FireFly® FireWire* artificial vision system in the head [42]. This platform is also equipped with an onboard *PC-104* small form-factor computer.

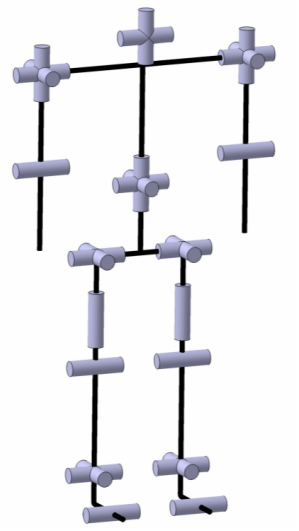


Figure 1.19: PHUA kinematic chains.

Table 1.1: PHUA platform's degrees-of-freedom and installed servomotors.

Joint	**Number of DOFs**	**Servomotor**
Toe	1(×2)	(none)
Ankle	2(×2)	*HSR-5980SG*
Knee	1(×2)	*HSR-5980SG*
Hip	3(×2)	*HSR-5980SG*
Trunk	3	*HSR-5980SG*(×2) *HSR-8498SG*(×1)
Shoulder	3(×2)	*HSR-5980SG* *HSR-5498SG* *HS-82MG*
Elbow	1(×2)	*HSR-5498SG*
Neck	2	*HSR-5498SG* *HS-5056MG*
Total	27	—

The robot's anthropometric proportions and ranges allow the platform to walk with straight support legs by incorporating a compliant foot with a passive toe joint, bringing the robot's foot kinematics closer to the humans and allowing for a more natural gait.

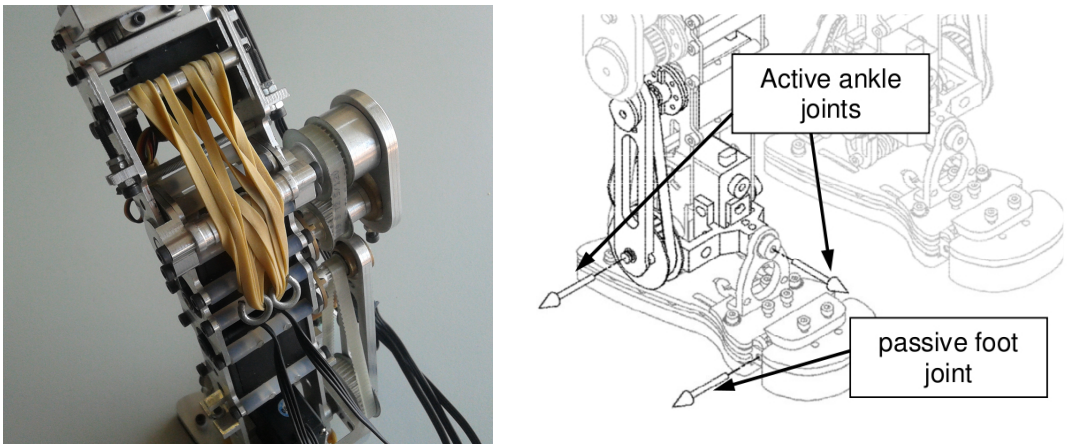


Figure 1.20: Elastic elements placed on the robot's joints and the passive toe joint.

Table 1.2 details the robot's joints anthropometric ranges and the total angular course, measured from the home position (in degrees), defined as the robot with all limbs at full stretch downwards, with the torso upright and a zero tilting in the head. At the moment there are unavailable DOF's due to reasons that will be detailed further in Chapter 3, and thus are not represented.

Table 1.2: PHUA robot degree-of-freedom range limitations.

DOF	**Minimum**	**Maximum**	**Total Range**
Head Tilt	0º	10º	10º
Shoulder Flexion/Extension	–45º	135º	180º
Shoulder Abduction/Adduction	0º	180º	180º
Elbow Flexion/Extension	0º	120º	120º
Torso Rotation	–90º	90º	180º
Torso Flexion/Extension	–15º	90º	105º
Torso Lateral Flexion/Extension	–90º	90º	180º
Ankle Inversion/Eversion	–30º	45º	75º
Ankle Flexion/Extension	–40º	20º	60º
Knee Flexion/Extension	0º	130º	130º
Hip Abduction/Adduction	–40º	45º	85º
Hip Flexion/Extension	–30º	120º	150º

Chapter 2

SimMechanics Simulator

The PHUA robot full body model simulator and results are presented in this chapter. This simulator is a MATLAB SimMechanics construction, built according to the actual robot mechanical and dimensional specifications, designed to assess the robot's hybrid actuation system and, if possible, study its behaviour. The model expects the joint course data input (position, velocity, acceleration and time vectors) and returns the calculated joint torques developed for the movement. It is also equipped to track the robot's center-of-gravity position, velocity and acceleration. To study the model and simulator's capabilities, a simple movement from a step climbing locomotion pattern is simulated, and its fundamental mathematical expressions are presented. The complete SimMechanics model is presented at Appendix A, for future reference.

2.1 SimMechanics Short Overview

MATLAB SimMechanics is a multibody simulation environment for mechanical systems, included within the Simscape suite. The multibody systems are modelled using blocks (representing bodies), joints, constraints, and force elements, whose equations of motion (for the complete mechanical system) are then formulated and solved.

All geometrical information, including masses, inertial properties, joints configurations, constraints, and three-dimensional geometry can be imported into the model bodies, or can also be given through CAD model importing. During (and after) calculations, a 3D animation allows the visualization of the resulting system dynamics.

The model's parametrizations can be done using regular MATLAB variables and expressions. SimMechanics allows to use other toolkit as complementary systems, as control systems from Simulink or electrical, hydraulic, or pneumatic systems from Simscape, and test them altogether in a single simulation environment.

One of the main key features of this MATLAB toolkit are its simulation modes for analysing motion and calculating forces/torques in a dynamical environment. This dynamical analysis allows to study certain intrinsic properties and behaviour of the mechanical constructions that would be strenuous to handle otherwise.

The SimMechanics model construction is very similar to the standard Simulink block-based approach, but is based on defining three-dimensional bodies and their respective mechanical connections. In this case, the robot bodies are all connected *via* rotational joints, actuated in position, velocity and acceleration, while the robot connection to the ground is rigid, through the tip of the robot's support leg foot. Joint actuation is made through an actuator block, and the required joint information is retrieved through a joint sensor block. An example of these constructions is given in Figure 2.1. The hybrid actuation system is installed on the knee and ankle joint systems using spring and damper blocks. The models can also account for a gravity acceleration vector, defined with an environment block.

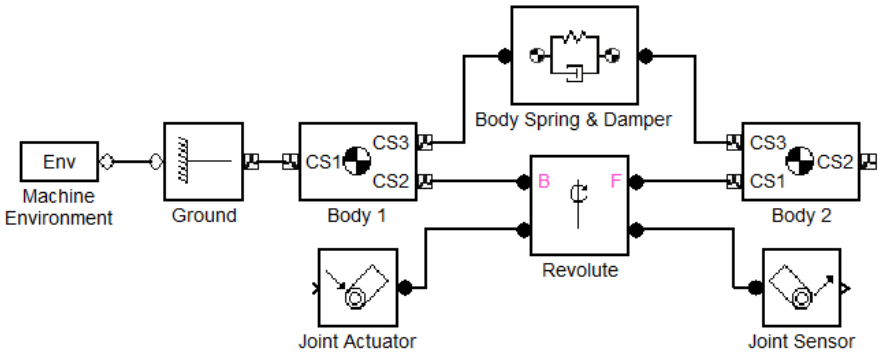


Figure 2.1: SimMechanics body to body construction example. joint blocks can be of rotational or prismatic joint types, allow for multiple DOF and different types combinations.

2.2 Model Construction

The SimMechanics toolkit had already been approached in past works, but only for the purpose of studying the elastic elements on the knee. Thus, in order to build all the bodies that compose the robot, the geometrical information, such as physical dimensions (lengths, centres of gravity), masses, and inertial matrices were taken from the robot CAD model, shown in Figure 2.2, in *CATIAv5* environment. The joint properties, such as actuation type and course limitations, as well as the elastic elements properties, were based on previous studies on the hybrid actuation system, the robot construction and other simulators [40; 39; 43].

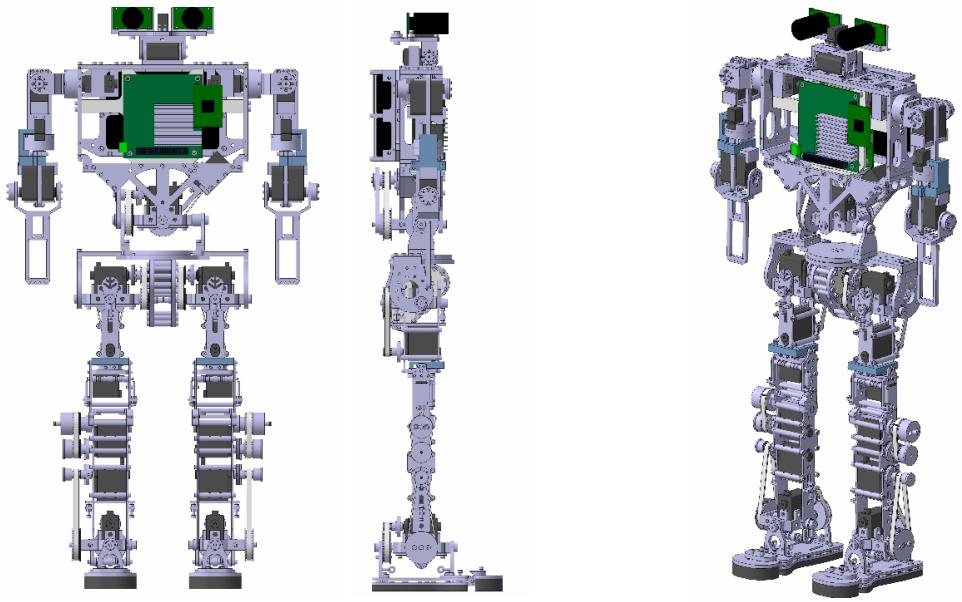


Figure 2.2: PHUA platform CAD design model.

It was determined that one of the most hampering limitations of the SimMechanics is the lack of a ground/floor interaction system, that is, a way for bodies to interact dynamically with blocks that are rigid to a static element. It is because of this limitation that the unactuated joints in the robot's feet are not represented in this model. As only the tip of the foot is linked rigidly to the ground block, the gravity acceleration made the foot rotate freely around the unactuated joint,

although the ground block occupied the space below them. Some strategies were attempted, like mirroring the foot's weight vector from the ground as a contact force, maintaining the foot and the floor plane together, but the toolkit is not prepared for this workaround and the results were unsatisfactory.

The SimMechanics model bodies are built from a reference frame to a destination frame, thus requiring at least two frames. Other frames can be defined in between, and can be given on any previous reference. This model was built from the ground up, defining bodies consecutively. In the legs, the model is built from the tip of the support leg's foot to the tip of the free leg's foot. The distinction between support and free leg is necessary because SimMechanics currently does not support parallel configurations. Table 2.1 details the geometrical data used to build all the robot body blocks, specifying their mass and centre of mass translation vector (and respective reference frame), as well as the respective inertia matrices, given in the frames of the centres of mass.

Table 2.1: SimMechanics PHUA robot model geometrical data used to build the body blocks (positive is for left limbs, negative is for right).

Body	**Mass** (kg)	**Center of Mass** (mm)	**Inertia Matrix** ($kg.m^{-2}$)
Feet	0.444	[±2.746 ; -18.300 ; 9.060] (ankle joints frame)	$\begin{bmatrix} 9.30\times10^{-4} & \pm2.20\times10^{-5} & \mp1.10\times10^{-5} \\ \pm2.20\times10^{-5} & 7.6\times10^{-4} & 1.50\times10^{-4} \\ \mp1.10\times10^{-5} & 1.50\times10^{-4} & 3.80\times10^{-4} \end{bmatrix}$
Legs	0.343	[±15.861 ; 88.519 ; 0.182] (ankle flexion joint frame)	$\begin{bmatrix} 5.10\times10^{-4} & \pm4.80\times10^{-5} & \mp6.50\times10^{-7} \\ \pm4.80\times10^{-5} & 2.30\times10^{-4} & \pm2.0\times10^{-5} \\ \mp6.50\times10^{-7} & \pm2.00\times10^{-5} & 6.80\times10^{-4} \end{bmatrix}$
Thighs	0.243	[±1.418 ; 101.284 ; -2.262] (knee flexion joint frame)	$\begin{bmatrix} 5.30\times10^{-4} & \pm1.10\times10^{-4} & -4.00\times10^{-6} \\ \pm1.10\times10^{-4} & 1.40\times10^{-4} & 4.60\times10^{-6} \\ -4.00\times10^{-6} & 4.60\times10^{-6} & 5.70\times10^{-4} \end{bmatrix}$
Pelvis	0.883	[0.000 ; -34.800 ; 8.110] (torso rotation joint frame)	$\begin{bmatrix} 6.39\times10^{-4} & 7.39\times10^{-8} & -5.60\times10^{-8} \\ 7.39\times10^{-8} & 0.002 & -6.80\times10^{-8} \\ -5.60\times10^{-8} & -6.80\times10^{-8} & 0.002 \end{bmatrix}$
Torso Base	0.089	[0.000 ; 13.411 ; 0.220] (torso joints frame)	$\begin{bmatrix} 3.65\times10^{-5} & -7.55\times10^{-14} & -3.83\times10^{-9} \\ -7.55\times10^{-14} & 1.23\times10^{-4} & 4.68\times10^{-7} \\ -3.83\times10^{-9} & 4.68\times10^{-7} & 1.22\times10^{-4} \end{bmatrix}$
Torso	1.849	[2.255 ; 100.867 ; -18.896] (torso joints frame)	$\begin{bmatrix} 0.007 & 8.48\times10^{-5} & -2.73\times10^{-5} \\ 8.48\times10^{-5} & 0.006 & 5.85\times10^{-4} \\ -2.73\times10^{-5} & 5.85\times10^{-4} & 0.01 \end{bmatrix}$
Shoulders	0.077	[±18.640 ; 0.000 ; -2.170] (shoulder abduction joint frame)	$\begin{bmatrix} 2.30\times10^{-5} & \mp2.00\times10^{-9} & 4.24\times10^{-6} \\ \mp2.00\times10^{-9} & 3.68\times10^{-5} & \pm2.00\times10^{-9} \\ 4.24\times10^{-6} & \pm2.00\times10^{-9} & 2.30\times10^{-5} \end{bmatrix}$
Arms	0.250	[±0.500 ; -62.457 ; 0.000] (shoulder flexion joint frame)	$\begin{bmatrix} 5.17\times10^{-4} & \pm6.14\times10^{-6} & \pm7.60\times10^{-8} \\ \pm6.14\times10^{-6} & 4.08\times10^{-5} & -6.70\times10^{-6} \\ \pm7.60\times10^{-8} & -6.70\times10^{-6} & 5.16\times10^{-4} \end{bmatrix}$
Forearms	0.048	[±3.418 ; -26.603 ; 0.000] (elbow flexion joint frame)	$\begin{bmatrix} 4.18\times10^{-5} & \pm4.34\times10^{-6} & 8.23\times10^{-12} \\ \pm4.34\times10^{-6} & 1.87\times10^{-5} & \mp8.37\times10^{-12} \\ 8.23\times10^{-12} & \mp8.37\times10^{-12} & 5.76\times10^{-5} \end{bmatrix}$
Head	0.333	[0.000 ; 22.400 ; -11.300] (head tilt joint frame)	$\begin{bmatrix} 1.95\times10^{-4} & 4.60\times10^{-7} & -1.60\times10^{-7} \\ 4.60\times10^{-7} & 2.48\times10^{-4} & -1.80\times10^{-5} \\ -1.60\times10^{-7} & -1.80\times10^{-5} & 2.75\times10^{-4} \end{bmatrix}$

The model was built to have as inputs each joint position, velocity and acceleration curves, and is equipped to measure and output the active/passive torques in all the joints. It was also implemented a mechanism that tracks the full body center of gravity position, velocity and acceleration curves, which is then represented using the three orthogonal lines (red, green and blue) that represent the COG frame, allowing to detect its movement even when it is occluded within the robot.

Also, the robot model bodies were created with redundant frames that help SimMechanics draw an approximate volume that enriches the graphical visualization of the model, shown in Figure 2.3. This volume is created with the outermost frame origins of each body, so the geometry is not exact, but it is also not used on any calculations.

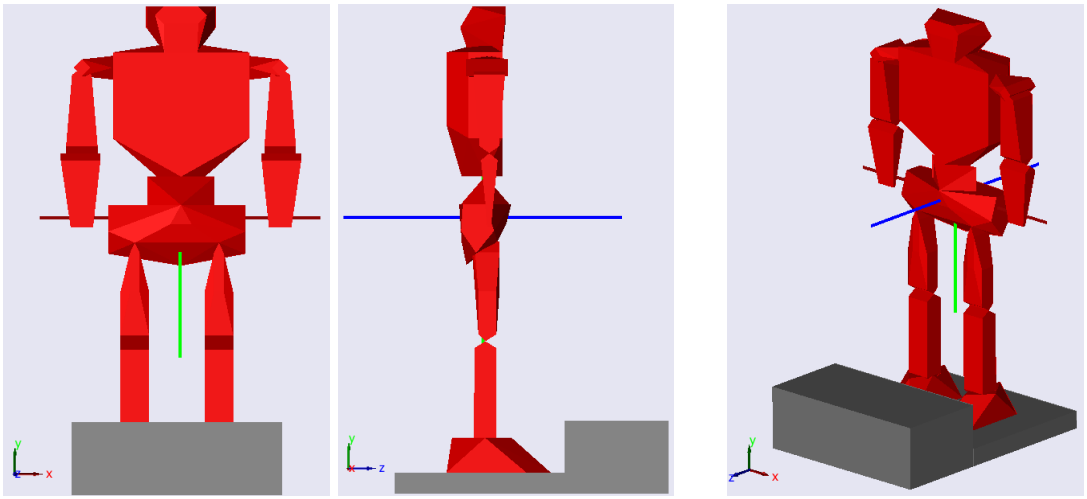
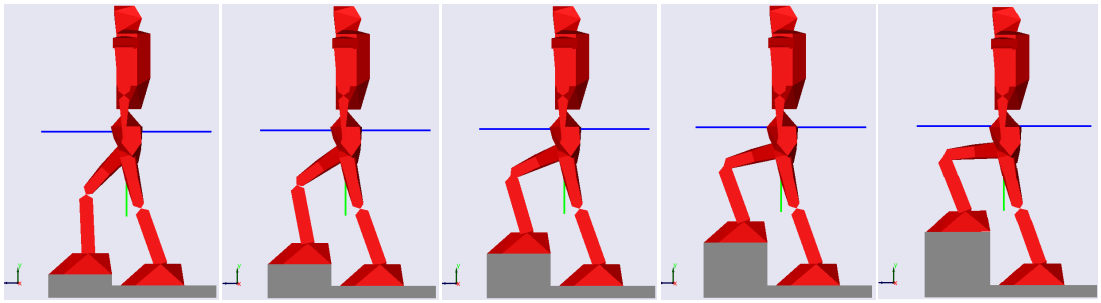


Figure 2.3: SimMechanics model graphical visualization. The robot is represented in red and the step and floor bodies in gray. The three orthogonal lines that represent the COG are also represented in their red-green-blue color order.

In preparation for the step climbing simulation, the body blocks for the step and floor plane were included in the model. All of the step's dimensions are pre-defined, with the exception of its height. The step height h is a parameter of the simulation, used to define the foot trajectories. Figure 2.4 shows a series of acceptable step heights with the robot on its final pose of the simulation.



(a) h=25mm (b) h=50mm (c) h=75mm (d) h=100mm (e) h=125mm

Figure 2.4: SimMechanics step heights visual comparison, ranging from 25 to 125mm.

2.3 Step Climbing

The definition of the step climbing joint position curves was based on an analysis on the humans stair ascent and descent natural joint courses [44]. The data is shown, in Figure 2.5, for the spans of the leg flexion joints of the ankle, knee and hip for the humans. Due to the SimMechanics floor plane interaction limitations, the whole step climbing movement can not be performed at once using the same model. The support leg becomes a free leg when the current free leg presses the step to project the body upwards. Therefore, this simulation only covers half of the step climbing movement.

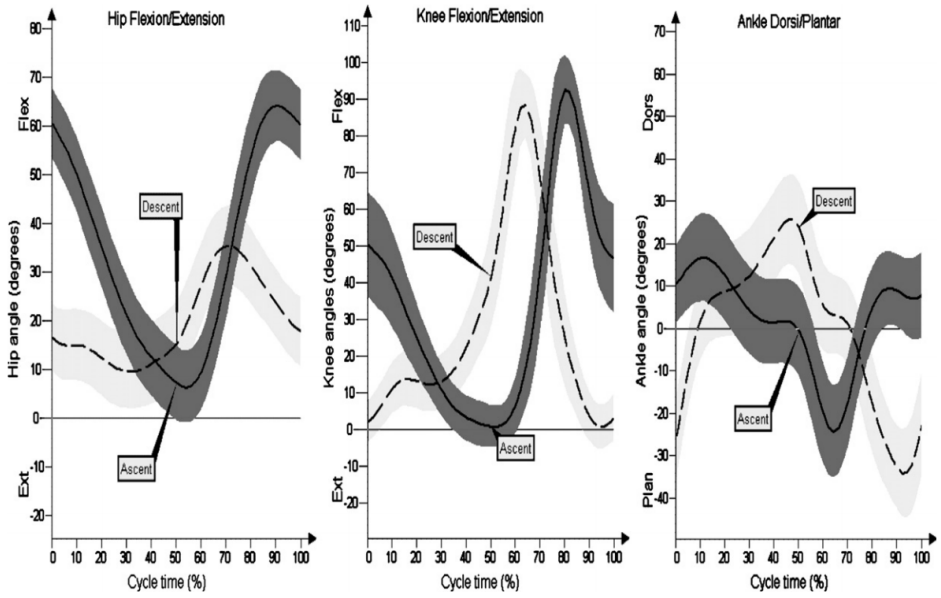
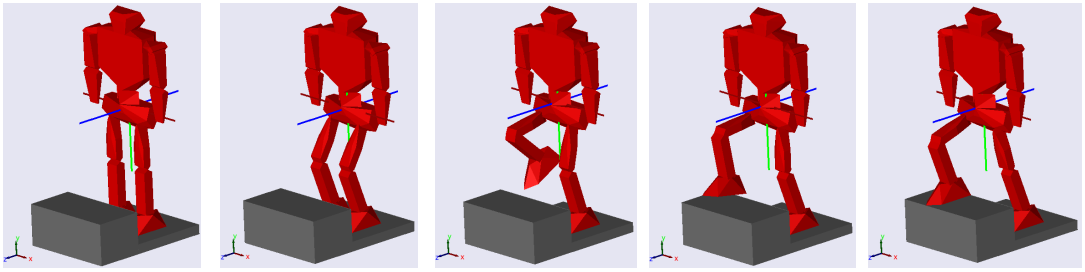


Figure 2.5: Mean sagittal plane angles for the human hip, knee, and ankle flexion joints during stair ascent and descent [44].

From the analysis of the human step climbing movement, a 4 stage approximate movement was designed for the robot, as shown in Figure 2.6. On the first stage the robot lowers the hip backwards to leave the singularity point of the upright posture and increase the joint ranges. The second and third stage represent the leg swing, where the robot will pull the thigh upwards and swing the leg and foot forward. The last stage is the beginning of the support leg switch stances, where the foot approaches and steps on the target step. Each stage duration was timed to approximately match the timings of the human natural stances.



(a) 0% (b) 35% (c) 56.7% (d) 78.3% (e) 100%

Figure 2.6: SimMechanics step climbing movement, with the correspondent time fractions, examplified for a 100mm step.

For the implementation of this simulation with the developed model, the joint course curves must be fed into the model. The limb joint space trajectories were translated from the Cartesian space using the standard inverse kinematics expressions for the RRR planar robot(three consecutive parallel rotational joints), shown on equations (2.1), (2.2) and (2.3), applied to both the support leg and the free leg, whereas on the support leg the end-effector is the hip and on the free leg the end-effector is the foot. The end-effector pose is determined by the Cartesian position Pw_x and Pw_y, and by the orientation ϕ. L_1 and L_2 stand for the limb lengths.

$$\theta_2 = \pm \arccos\left(\frac{Pw_x^2 + Pw_y^2 - L_1^2 - L_2^2}{2\,L_1\,L_2}\right) \tag{2.1}$$

$$\theta_1 = \arctan\left(\frac{Pw_y(L_1 + L_2\,\cos\,\theta_2\,) - Pw_x\,L_2\,sin\,\theta_2}{Pw_x(L_1 + L_2\,\cos\,\theta_2\,) + Pw_y\,L_2\,sin\,\theta_2}\right) \tag{2.2}$$

$$\theta_3 = \phi - \theta_1 - \theta_2 \tag{2.3}$$

The individual joint trajectory curves are then built using a 5^{th} degree polynomial, from the initial joint position θ_i to the final position θ_f. The time frame is defined by the initial and final time values, t_i and t_f. Parameters a, b and c (2.4) are used to create the time dependent curves for the joint position q, velocity v and acceleration a, represented in equations (2.5), (2.6) and (2.7).

$$a = -\frac{10\,(\theta_i - \theta_f)}{(t_f - t_i)^3}; \quad b = \frac{15\,(\theta_i - \theta_f)}{(t_f - t_i)^4}; \quad c = -\frac{6\,(\theta_i - \theta_f)}{(t_f - t_i)^5} \tag{2.4}$$

$$q = \theta_i + a\,t^3 + b\,t^4 + c\,t^5 \tag{2.5}$$

$$v = \dot{q} = 3\,a\,t^2 + 4\,b\,t^3 + 5\,c\,t^4 \tag{2.6}$$

$$a = \dot{v} = 6\,a\,t + 12\,b\,t^2 + 20\,c\,t^3 \tag{2.7}$$

Using a 5^{th} degree polynomial function for the position description ensures the acceleration curve is continuous in its domain. Figure 2.7 shows an example of the three curves used to define a joint movement.

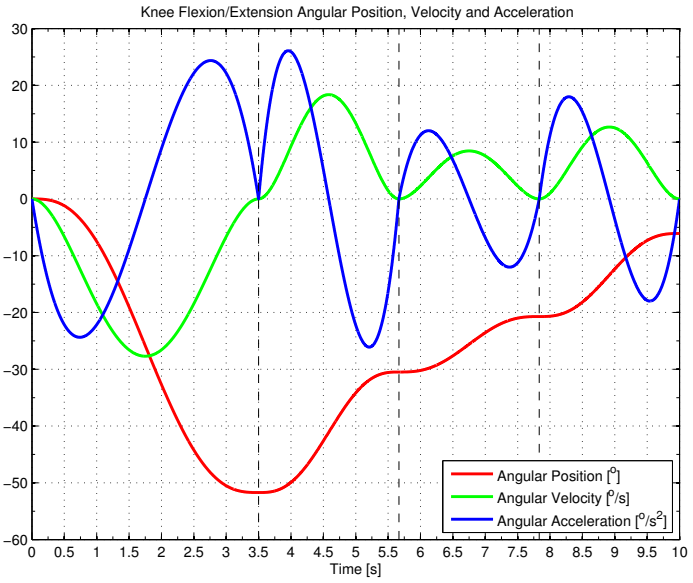


Figure 2.7: Example of a joint course description curve set, for the support leg knee in a 75mm step height 10 second climbing movement.

The model uses the joint information data sets to generate the robot motion and dynamical behaviour. As the model is parametrized by the step height and all of robot's 25 joints are analysed, the data produced is extensive. Figures 2.8 and 2.9 show two examples of torque curves that can be extracted from this simulator.

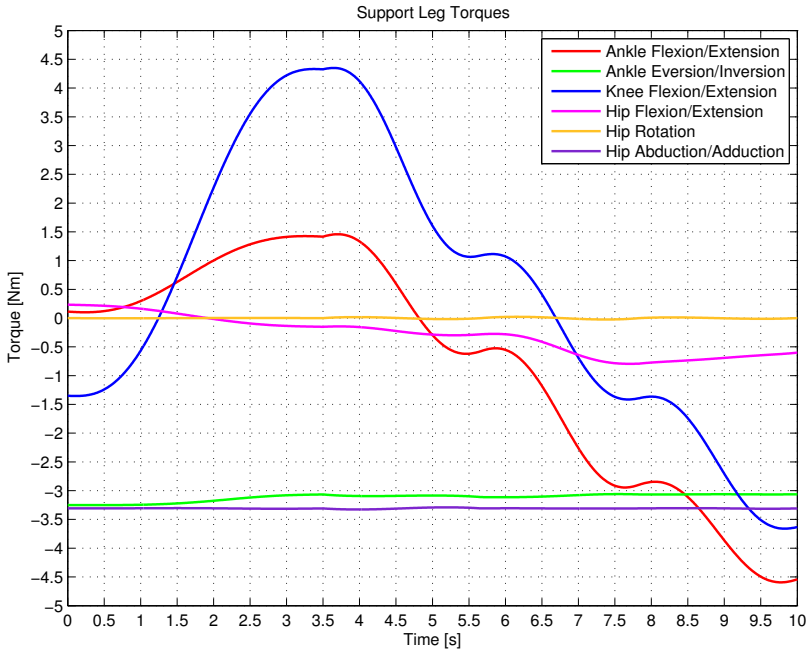


Figure 2.8: Example of a joint torque data set for the support leg in a 10 second movement with a step height of 50mm.

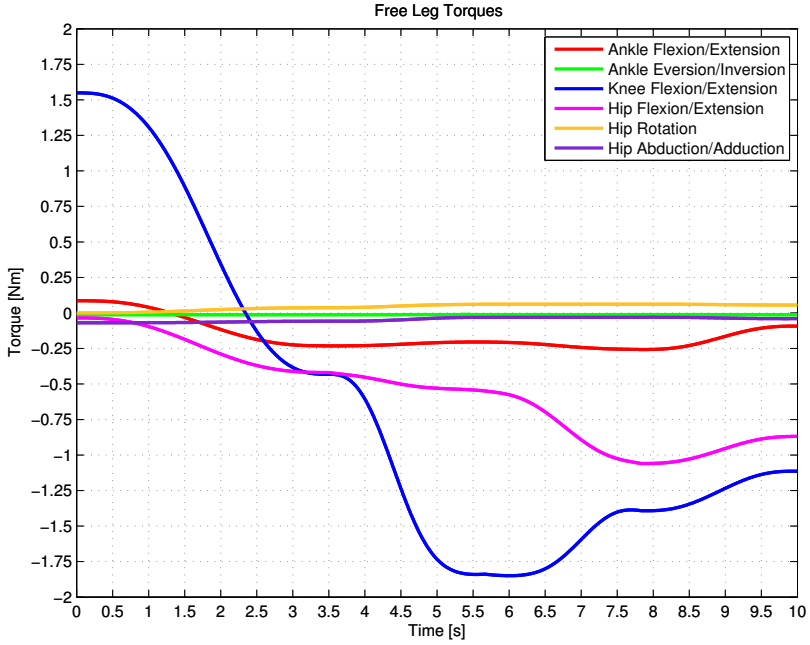


Figure 2.9: Example of a joint torque data set for the free leg in a 10 second movement with a step height of 50mm.

However, this model also calculates passive torque for the joints actuated with constant values, corresponding to the still torque the motor would have to produce to hold its position, as shown in the examples of Figures 2.10 and 2.11, for the same case as before.

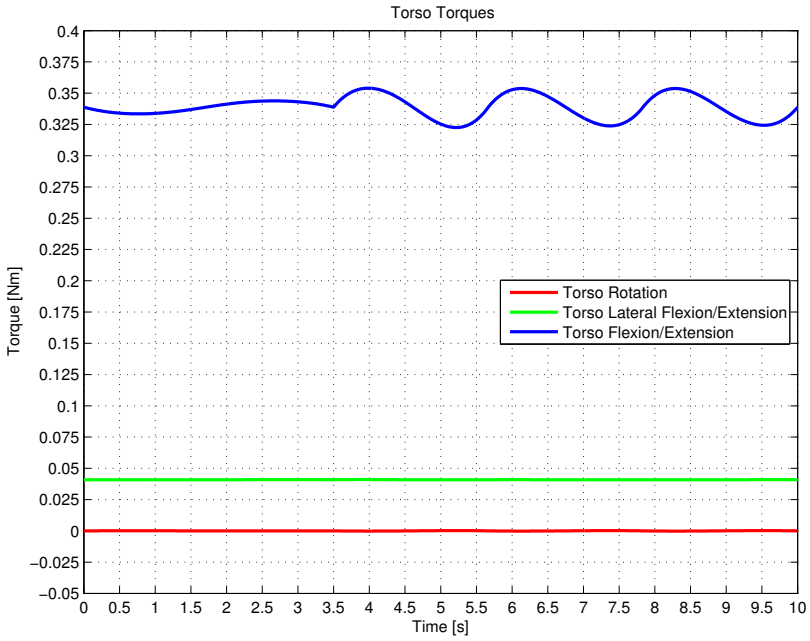


Figure 2.10: Example of a joint torque data set for the torso in a 10 second movement with a step height of 50mm.

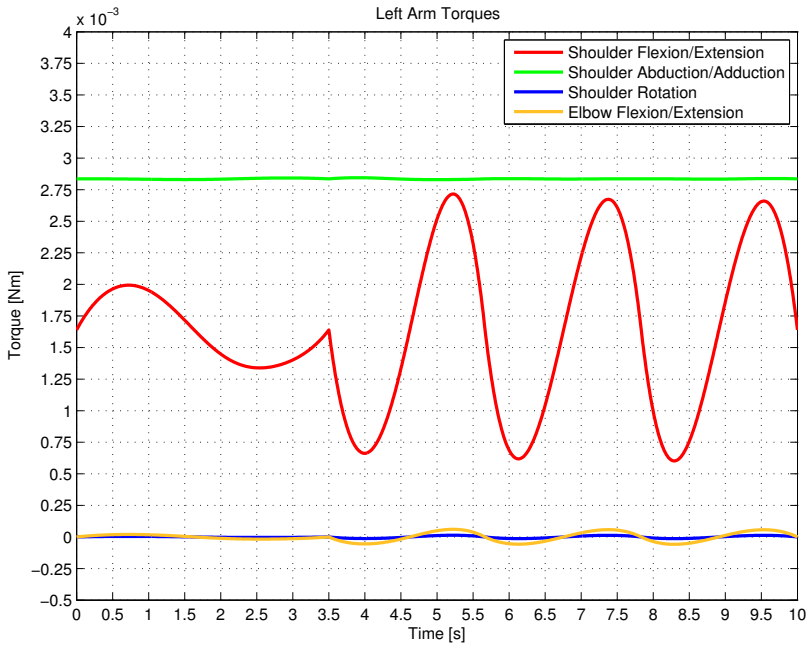


Figure 2.11: Example of a joint torque data set for the left arm in a 10 second movement with a step height of 50mm.

By examining the data collected for the knee and ankle joints using several simulations with different step heights, it was possible to verify that although the results for the ankle joints with the elastic elements in placed produced expectable results, the curves produced for the knee joint were not consistent to what was expected. It was expectable that the knee joint torque curves would show less energy spending while the knee was recovering from a given position, due to the torque gained from the elastic elements, just as it was seen for the ankle joints, but that was not the case.

The spring and damper block used to create the elastic elements of the robot (with a zeroed damping factor), simulates a rigid spring connected between two different bodies in space. On the knee, these connection points are placed on the leg and thigh, at approximately 30mm from the joint axis. When the knee is upright, the force vector from the spring is correctly applied to the bodies due to the spring being defined as pre-tensioned. Then, as the knee flexes, and as SimMechanics does not yet possess interaction mechanisms between bodies, the model solver sees the two end points of the spring coming closer, instead of farther. Naturally, neither the force direction nor its magnitude are well calculated by the algorithms and the joint torque influence that was expected from the hybrid actuation system is not verified in the resulting curves.

The force generation from the elastic elements in the knee is non-linear and very hard and time consuming to simulate using other Simulink blocks due to the closed form of the SimScape suite (in which SimMechanics is inserted). It was then decided that the simulation of the second half of the step climbing movement was not to be pursued because the results for the hybrid actuation of one of the most critical joint in that movement, the knee, could not be trusted.

Chapter 3

Experimental Setup

In this chapter, the primary hardware and software setup that was necessary for the implementation of the haptic demonstrations is presented, detailing the servomotor wiring scheme, mechanical modifications introduced and the chosen haptic joystick and programming libraries. When this work was initiated, the PHUA platform robot could not accommodate the desired implementation for haptic teleoperation and data logging. Some hardware issues were undefined and had to be resolved to allow a correct software implementation.

3.1 Servomotor Wiring and Communication

One challenge in this work was to come up with a solution for the robot actuation wiring and communication. Some solutions were already explored before [41; 45], based on distributed control architectures, but unfortunately none of them were installed as a final version. The implementation of a low-level hardware control system for the whole robot was not an objective of this dissertation, so another alternative had to be devised.

The HITEC servomotors possess a proprietary protocol called *Hitec Multi-Protocol Interface* (HMI), that allows communication to be established in three different modes [41]:

- *Standard Pulse*
- *Extended Pulse*
- *Bidirectional Serial Interface*

The first two are *PWM* based communication protocols, requiring dedicated hardware. The *Bidirectional Serial Interface* is an RS-232 based communication, using a 7 byte command frame represented on Table 3.1, configured as the Table 3.2 shows. The last 2 bytes of the word contain the answer from the servomotor controller. Different answers from multiple servos on one communication line will result in a bus collision. This type of communication allows not just the position and velocity command of the servomotor, but also its internal programming.

Table 3.1: The HITEC *BSI* RS-232 7 byte word [41].

byte	1	2	3	4	5	6	7
Controller	Startbyte	Command	Param1	Param2	Checksum	0x00	0x00
Servomotor	High-Impedance					Return1	Return2

The checksum byte adds up the sum of bytes 1 through 4 to a multiple of 256, as shown on expression (3.1):

$$Checksum = 256 - (0x80 + Command + Param1 + Param2) \ \% \ 256 \tag{3.1}$$

Table 3.2: The HITEC *BSI* RS-232 configuration [41].

Parameter	**Value**
Baudrate	19200
Data Bits	8
Stopbits	2
Parity	(none)
Handshake	(none)

These servomotors perform their own motion control, having the low-level control already implemented in the Atmega micro-controller circuit inside their casing. The controller's trajectory planner generates a linear trajectory (constant angular velocity) starting from the last trajectory set point p_0 to the actual target position p_1, using the input position,defined in an integer range of 1800 positions [600-2400], and the velocity value, defined in an integer range from 1 to 255, it receives. The control loop then ensures the motor reaches the desired position by monitoring its error value. The planned trajectory parameters are stored in a different memory block from the one used for planning. This different block is read by the PD controller at every control loop to define the motor PWM pulse. This mechanism assures the trajectory can be changed at every loop without having to stop the motor, allowing for continuous operations.

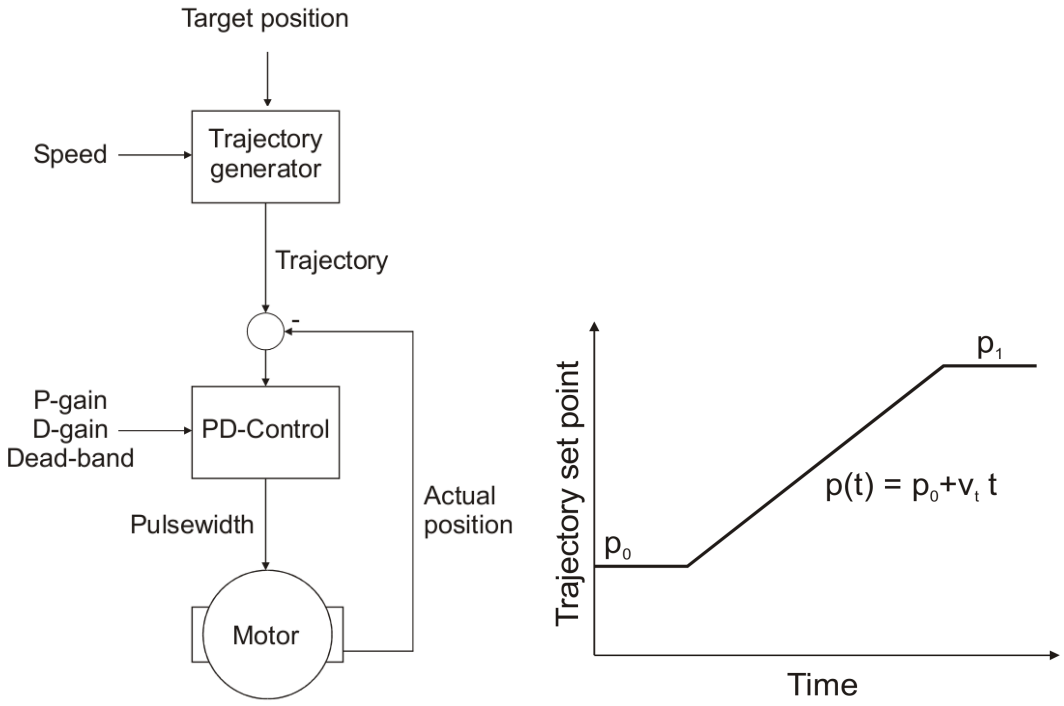


Figure 3.1: Servomotor controller motion control (left) and trajectory planning (right) (adapted [41]).

Closed-loop customized motor control is difficult with the current hardware. There was already a distributed control architecture strategy presented in a previous work [41] that tried to compensate this with local micro-controller units actuating small clusters of servomotors and communicating between them through a CAN type protocol.

It was decided that, having the possibility to define a unique identification number (ID) for each digital servomotor, the best solution was to implement a single direct line to connect all digital servomotors using the *BSI* protocol for communications. However, this had the incon-

venient of leaving out the analogue servomotors, since they require dedicated hardware. As it would be very time-consuming to make the hardware structure necessary to include those motors in the line, for this dissertation it was decided to drop those DOFs, and work with all the others. Further, using a digital protocol also meant that the low-level control of the servomotors was definitely set-aside for the work, and any commanding methodology had to take into account the controller's trajectory generator limitations.

The servomotor programming was conducted using the "HMI Servo Programmer" software, supplied by HITEC (for Windows 32bits). It defined the whole robot's ID number logic, depicted in Figure 3.2, as well as the reset of their internal controller to the factory defaults, so it could be tested for optimal performance. The numbering logic defines a range of numbers specific for each limb, so that the ID number quickly identifies the joint. With this method it is also possible to introduce new servomotors in the platform without major modifications to the control structure.

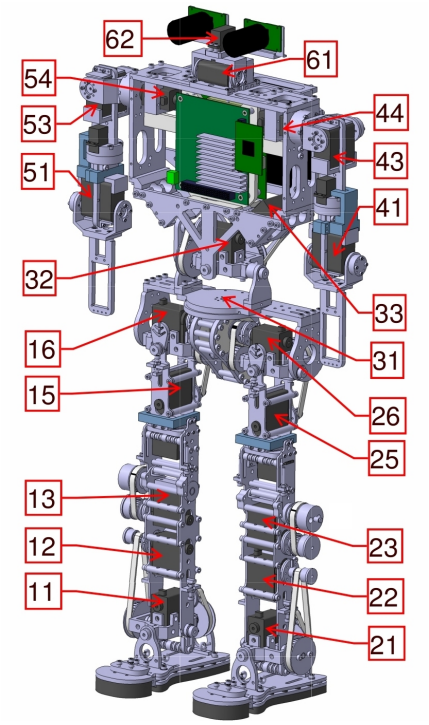


Figure 3.2: Servomotor ID numbering.

The servomotor internal controller can be configured in terms of proportional and derivative gains parameters (*P-Gain* and *D-Gain*) as well as their *Dead Zone* parameter. The proportional gain defines the speed and acceleration properties of the motor, and the derivative gain defines the servomotor's stopping behaviour. A small current consumption test showed that carrying heavy loads over great distances made the motors consume a considerable current. As advised by the manufacturer in the data sheets, the gain parameters were set to the their lowest setting in the controllers to avoid damage and wear throughout the course of the works. The *Dead Zone* parameter defines the inactive range of a stop point, or the "still tolerance". There was no noticeable change while varying this parameter, and it was decided to keep it to the lowest setting, again for servomotor preservation reasons.

To allow for the platform's mobility, this common line had to be installed on the robot and not restrain any DOF. Thus, it was decided to design small connector boards where the servomotors could be easily plugged/unplugged. These boards were then placed on each leg, on the trunk and on the shoulders, connecting all the servomotors together. These boards, as well as the connection diagram [41], are depicted on Figure 3.3.

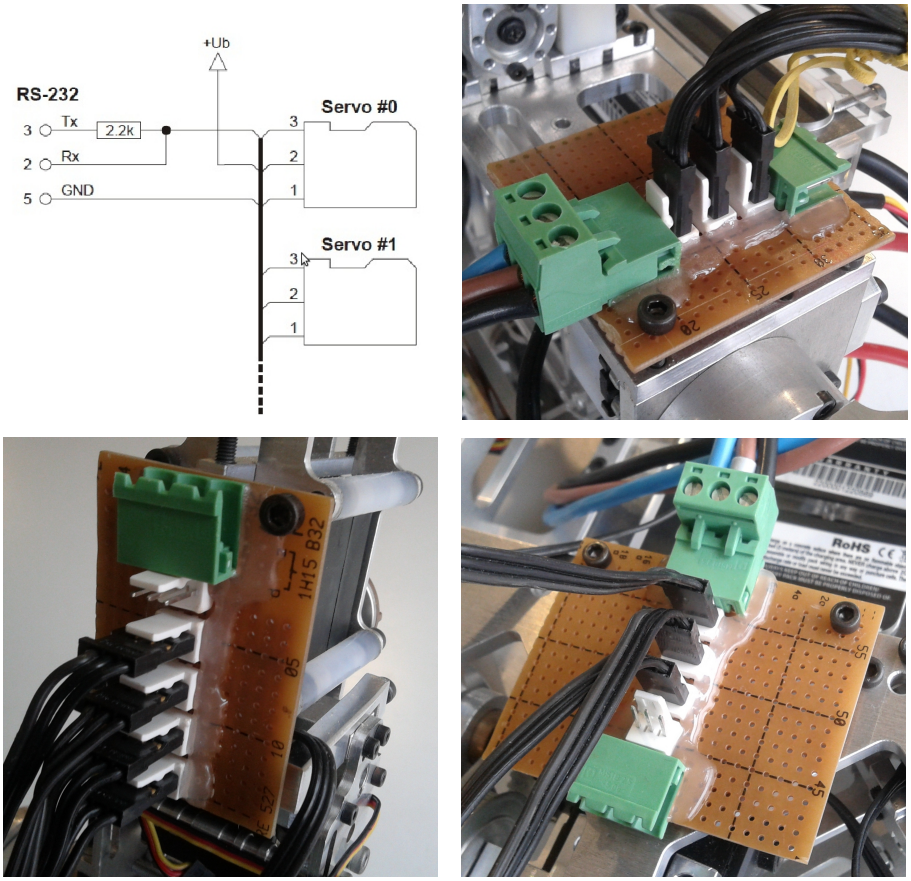


Figure 3.3: The servomotor line connection diagram [41] and connector boards.

Using the HITEC servomotor connection scheme, another small board was designed to connect the RS-232 connector and the power line to the robot's common line. These parts are represented in Figure 3.4.

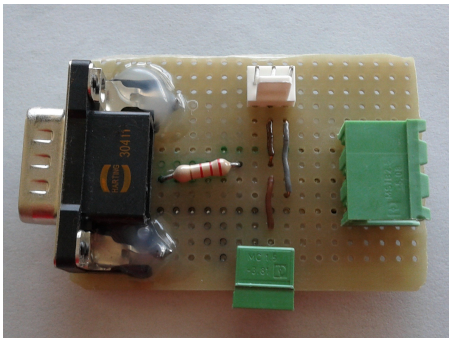


Figure 3.4: The designed servomotor communication converter board.

The common line allows for full robot control, with the exception of the analogue motors, and the connector board allows to separate the power line from the RS-232 communication line. It is also possible to control any kinematic chain separately, or a single servomotor, if necessary. The cables installed on the platform are not definitive and will require maintenance in the future, although perfectly fitting the purpose of testing.

3.2 Mechanical Modifications

Due to the wiring solution, the analogue servomotors on the arms and in the thighs were left unactuated. In order to stabilize them, it was decided to take one of two possible solutions: to power up the servos at zero position or to mechanically immobilize them. The later was the choice, given that it was unpredictable if the static torque of the analogue motors was enough to support the internal forces of the limbs while in operation, besides the fact that being powered up could eventually deteriorate them. So, the solution was to design small parts to restrain those DOFs and avoid possible problems. These parts, represented in Figure 3.5, were initially developed to be of low mass/dimensions, so that the limb's inertia matrices were not affected by them. However, the robot's dimensional restraints and the assembly/disassembly problems of the platform conditioned the design to its final geometry and position.

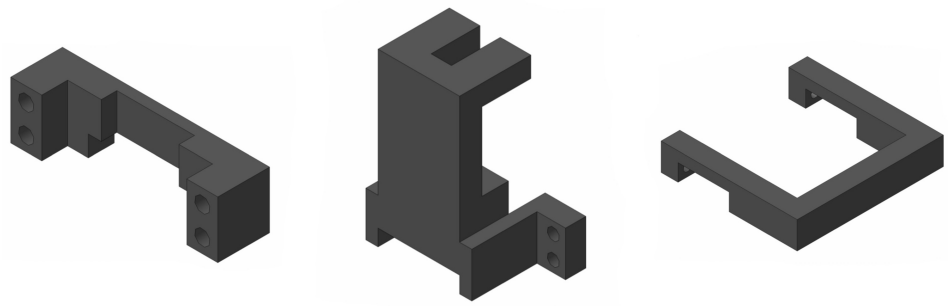
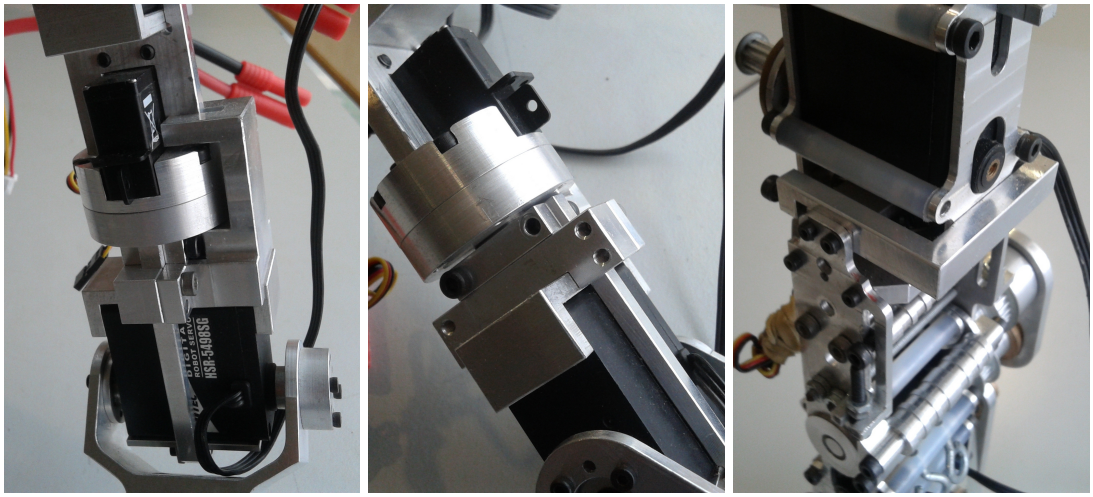


Figure 3.5: Designed parts for the servomotor fixation.

As the parts in the arms were developed with a considerable volume, the choice in material was made of aluminium. Although this material allowed for a small mass in the arm parts, it made the legs component frail around the bolt tightening phalanx. Even though the leg parts are slightly deformed, their mechanical performance is the same, and no problems are to be expected from al the parts while the robot is in operation. The mechanical drawings for the designed parts, shown Figure 3.6, are submitted on Appendix B.



(a) (b) (c)

Figure 3.6: Designed parts installed on the robot, for the arms on the left and middle, and for the legs on the right.

3.3 Elastic Elements Review

In the previous works on this platform [40; 41; 39; 45], there was no indication left about the mounting of the elastic elements of the ankle and hip joints elastic elements. Although there was some calculations regarding the mounting of the trunk and knee's rubber bands, no indication was given for any other joint or joint groups. The review of these mechanical elements was not an objective for the dissertation, but it quickly became clear that the experience gained over the course of this work was an interesting addition to the project's documentation.

The previous work and solutions on the mounting of these elements was not significantly altered, as the goal was to improve the current implementation. For that, the rubber band mounting on the knee flexion joint and on the trunk's flexion and lateral flexion joints was initially performed as previously defined.

On the robot's ankle flexion/extension and abduction/adduction joints, the mounting points were all previously defined, but were not tested. The placement of the rubber bands proved successful on maintaining the joint around it's home position, with the servomotors powered down, and with no apparent stress induced on them while in operation. Figure 3.7 shows the rubber bands mounting on the ankle.

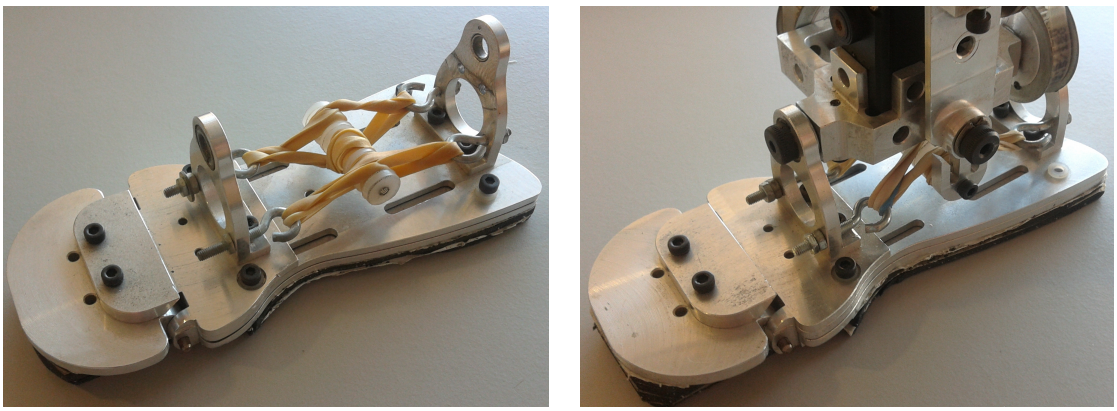


Figure 3.7: Elastic elements on the robot's ankle joints.

New mounting points had to be created to apply the rubber bands on the hip flexion/extension and abduction/adduction joints, in order to fulfil the goal of the hybrid actuation system. As there was very little geometric/dimensional margin for a mechanical modification, the solution was to remove some screws and replace them with metric screw hooks, similarly to what had been done before in trunk [39], as shown in Figure 3.8.

However, the results were not satisfying. It was observed that, when the tested courses started on a joint limit (where the elastic force is greater), the joints accelerated up to a point were the servomotor could no longer stop at its destination point, due to lack of stopping torque. Diminishing the number of rubber bands was not a good solution either, for their presence became redundant due to very low exerted force.

The torques developed on these joints make the hybridization very complex. A simple approach as the one taken is not sufficient to assure the rubber band's force vector is applied in favour of the energy minimization required. It is necessary to achieve a compromise between the rubber bands stiffness and the increase in joint torques.

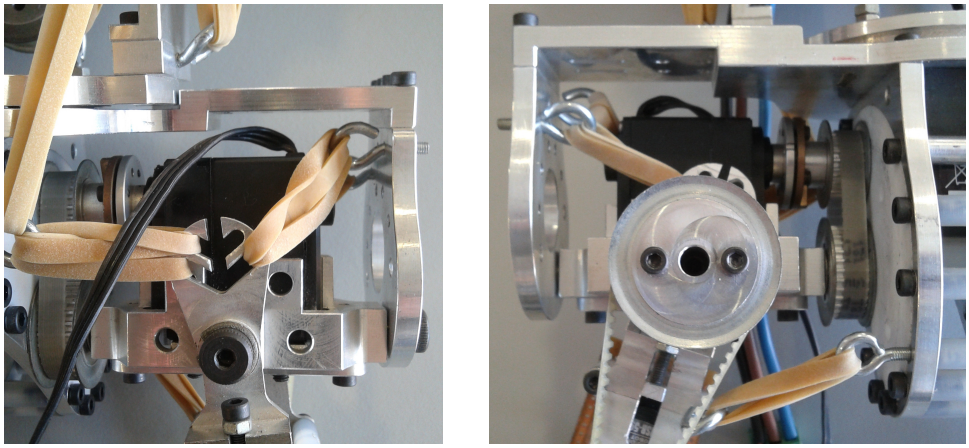


Figure 3.8: Elastic elements on the robot's hip joints, viewed from the front (left) and from the back (right).

As for the trunk joints, there was considerable amount of testing done, because of their critical nature on the robot's self balancing tasks, or even for it's operation while standing still. The first event of notice was the failure of the flexion/extension joint (forward/backward torso tilting). Due to the overweight of the upper-body (chest + head + arms), the servomotor assigned to this joint is severely under-dimensioned and fails to tilt the torso at any position or posture. The natural outcome is that the task of maintaining the torso upright falls completely on the elastic elements. The overweight problem was the result of unpredicted mechanical insertions to the upper-body [39] combined with a small margin in design tolerance after the design of the leg system [40].

In this sense, the solution determined was to over-supply the trunk's joints with rubber bands in such a way that they would remain almost locked. The trunk's lateral flexion motor has enough torque to move itself around 10º, at most upright postures, so the rubber bands must not restrict it in that range. As for the torso rotation joint, the low-end servomotor installed (see Table 1.1) fails to rotate the joint more than a mere 5ºdue to its lack of torque.

As the Figure 3.9 shows, the final solution is able to maintain the joints around the robot's home position, even with a small tilting. Unfortunately, rapid movements or big tilts are not possible with this strategy due to the intrinsic nature of elastic elements and the robot's upper-body overweight problem.

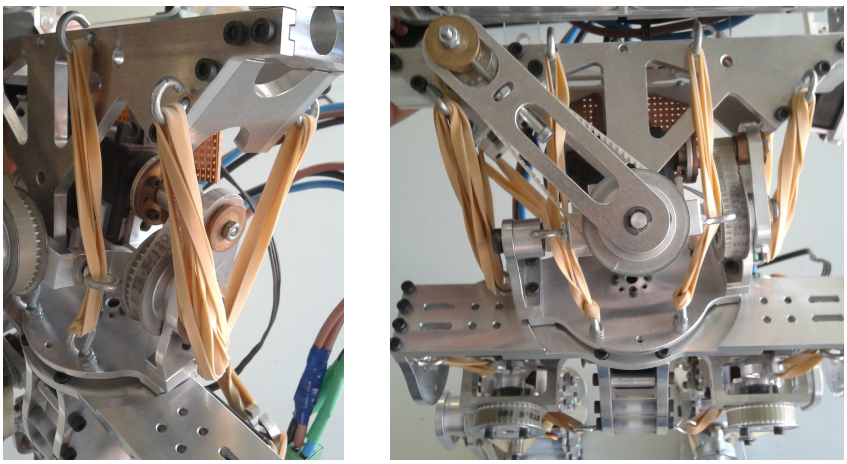


Figure 3.9: Elastic elements on the robot's trunk joints.

3.4 The Haptic Device

The haptic device used in this work is a SensAble PHANToM OMNI haptic joystick, as shown in Figure 3.10, which is a 6DOF impedance-based joystick capable of rendering three-dimensional force vectors at high frequencies. The OMNI model connects to the PC *via* IEEE-1394a (*FireWire*) ports and it can be connected in chain to another identical equipment. This equipment requires proprietary drivers to function properly, with specific requirements and dependencies that must be met. Therefore, its installation procedures are described in detail in Section C.2 for future reference.

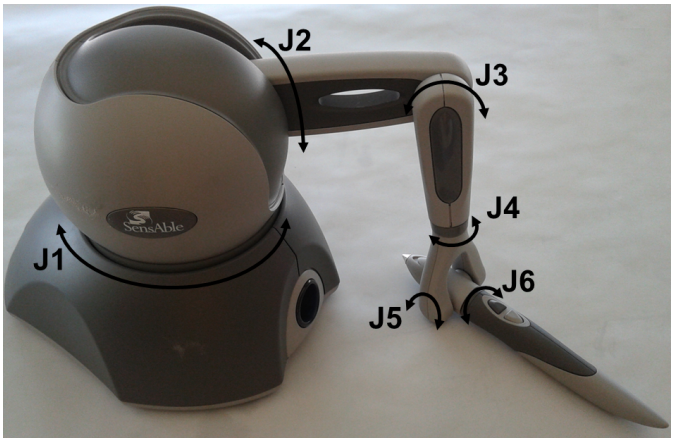


Figure 3.10: The PHANToM OMNI device used and its six degrees-of-freedom.

Although the device possesses 6DOF, from which it is possible to retrieve a correspondent three-dimensional Cartesian position, only three of these are active (actuated) joints, meaning it is only possible to render force vectors and not possible to render torques (directly) with the last 3DOF. This is mainly due to its mechanical construction. This device is built with a capstan drive actuating two out of three base degrees-of-freedom, as the Figure 3.11 illustrates, allowing for its compact form-factor and good force feedback level on its actuated workspace, coupled with high precision optical encoders. The last three DOFs are unactuated joints equipped with potentiometers, allowing for a compact mounting. A more extensive list of specifications is shown on Table 3.3.

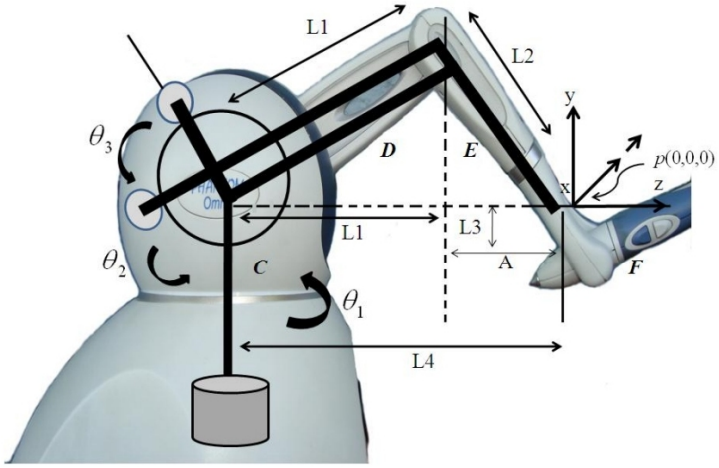


Figure 3.11: The PHANToM OMNI capstan drive mechanism illustration [46].

Before exploring the software solution offered by SensAble, the kinematics and manipulability of the device were consulted [46], to determine the possibility of developing a in-house control method for the device, avoiding proprietary libraries . However, it quickly became clear that implementing a such low-level control offered no real advantage to this work. The software supplied by SensAble, described ahead on Section 3.5, is well endowed with quickly deployable solutions to address the device's control.

Table 3.3: PHANToM OMNI device specifications [20].

Force Feedback Workspace	160 W X 120 H X 70 D [mm]
Footprint (Physical area the base of the device occupies)	168 W x 203 D [mm]
Weight (device only)	3 lb 15 oz (aprox. 1.786 kg)
Range of Motion	Hand movement pivoting at wrist
Nominal Position Resolution	450 dpi (aprox. 0.055 mm)
Backdrive Friction	0.26 N
Maximum Exertable Force (at nominal position)	3.3 N
Continuous Exertable Force (24 hrs.)	0.88 N
Stiffness	X axis : 1.26 N/mm Y axis : 2.31 N/mm Z axis : 1.02 N/mm
Inertia (apparent mass at tip)	45 g
Force Feedback	x, y, z
Position Sensing (with stylus gimbal)	- x, y, z (digital encoders) - roll, pitch, yaw (±5% linearity potentiometers)
Interface	IEEE-1394 FireWire port (6-pin to 6-pin)

3.5 OpenHaptics Toolkit

Developed in *C/C++*, the OpenHaptics toolkit is the proprietary programming solution offered by SensAble to build haptically enabled applications. Currently on version 3.0 and available for Linux, Windows and MacOS X operating systems, it includes [47; 48]:

- QuickHaptics micro API;
- Haptic Library API (HLAPI);
- Haptic Device API (HDAPI);
- PHANTOM Device Drivers (PDD);
- Utilities;
- Source code examples;
- Documentation.

The typical application structure under OpenHaptics is depicted in Figure 3.12. Its usual implementation points towards object virtualization and virtual environments (VE), usage for which it is exceptionally well prepared. Due to its long installation procedures, the relevant steps are described in detail in Section C.2 for future reference.

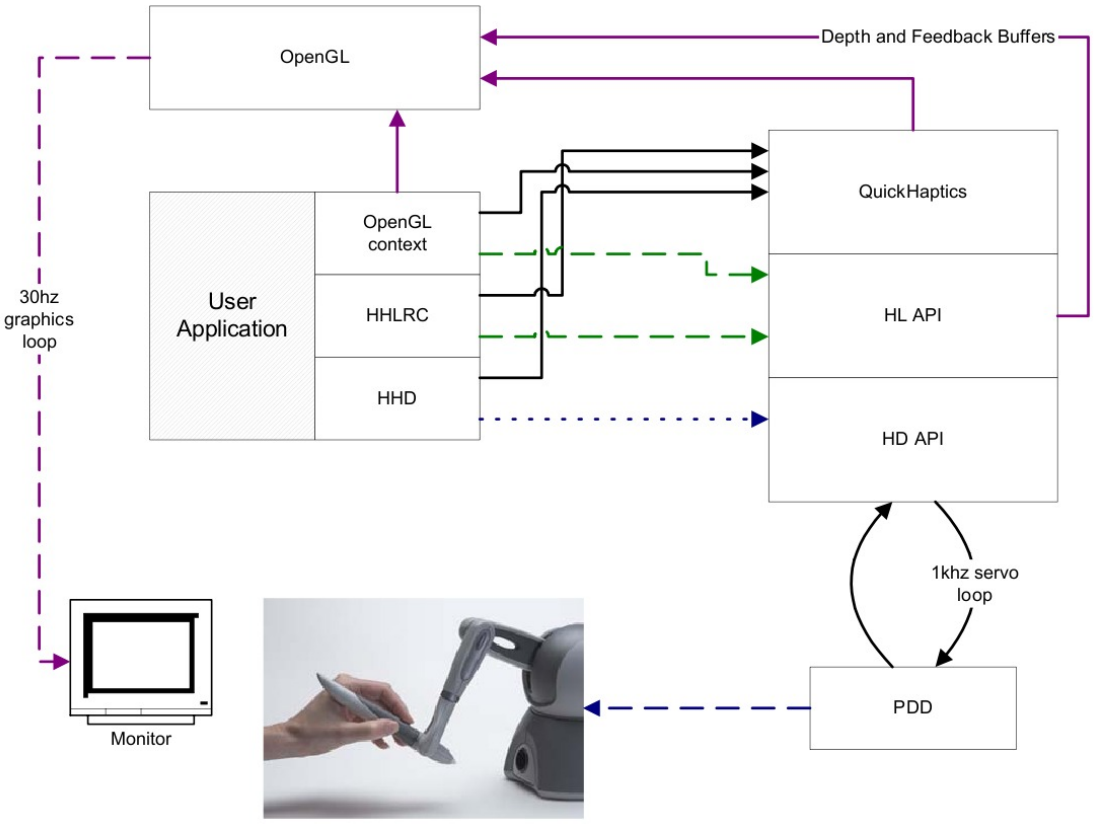


Figure 3.12: A typical OpenHaptics application structure [47].

Benefiting from the advances in this research field over the later years, these toolkits allow to ease programming by encapsulating the some of the steps common to all haptics/graphics applications in the *C++* classes of the QuickHaptics micro API, making use of the Standard Template Library (STL). By anticipating typically used scenarios and using built-in geometry

parsers and intelligent default parameters, it is possible to set up haptics/graphics scenes with a minimal amount of code.

The HLAPI is designed for high-level haptic rendering and it is designed to be familiar to OpenGL API programmers. It is targeted for advanced OpenGL developers who are less familiar with haptics programming, but desire to quickly and easily add haptics to existing graphical applications.

The HDAPI (the most explored in this work) is the foundational layer for haptics, providing low-level access to the haptic device. It is best suited for developers who are already familiar with haptic paradigms and have to send forces directly. It enables haptics programmers to render forces directly, offers control over configuring the runtime behavior of the drivers and provides convenient utility features and debugging aids. Experts can still use HLAPI and HDAPI in conjunction with QuickHaptics to take advantage of all SDKs.

The HDAPI requires the developer to manage direct force rendering for the haptic device, whereas HLAPI handles the computations of haptic rendering based on geometric primitives, transforms, and material properties. Direct force rendering with HDAPI requires efficient force rendering/collision detection algorithms and data structures coding. This is due to the high frequency of force refreshes required for stable closed-loop control of the haptic device.

The HLAPI prevents the developer from having to implement efficient force rendering algorithms and managing the synchronization of servo loop thread-safe data structures and state. The servo loop refers to the tight control loop used to calculate forces to send to the haptic device. In order to render stable haptic feedback, this loop must be executed at a consistent 1 kHz rate or higher. In order to maintain such a high update rate, the servo loop is generally executed in a separate, high priority thread. This thread is referred to as the servo loop thread [47].

Diagram 3.13 illustrates the relationship between the QuickHaptics micro API and the existing HD and HL layers of OpenHaptics.

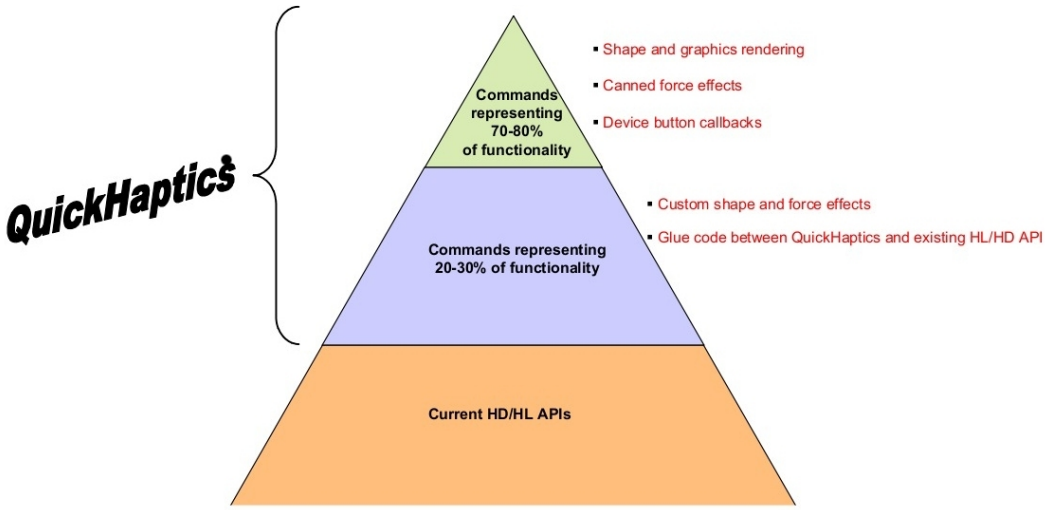


Figure 3.13: The OpenHaptics structure diagram [48].

3.6 *ROS* (Robot Operating System) Framework

Even though it was not in the original objectives, specific components of this work are running under the *Robot Operating System* (ROS) framework. Created in 2007, ROS started as an open-source programming platform for robots, designed for the development of the Stanford Artificial Intelligence Robot project (STAIR) [49]. The platform is now maintained by the open-source community and Willow Garage, responsible for the Personal Robots Program (PR/PR2 robots). ROS was created with the intent of fulfilling a series of objectives [50]:

- Build a lightweight and robust operating system;
- Ease modular software deployment;
- Build a free and open-source platform;
- Supply tools for code implementation as a configuration parameter server, message viewer, documentation auto-generation, etc ...;
- Multi-lingual (currently *C++*, *Python*, *Octave* and *LISP*);
- Support for *peer-to-peer* connections.

Designed to run under Unix/Linux, ROS eases the interaction of hardware and software, using a modular architecture. This architecture allows the creation and management of software modules and assures the communication between them in a very well structured and robust form. The platform is organized in *nodes*, *messages*, *topics* and *services* [50].

The *nodes* are processes that compute the system, being part of the software modules. With this modular architecture, these processes need to communicate among them. This communication is made by means of *messages*, which are data structures with the respective languages primitive data types which can be recorded in logs. An interesting fact about ROS is that the data in these logs can be later reproduced as if the real system is online, making it possible to record real world events and develop code for them in the laboratory with real event data playback information.

However, for a module to subscribe (or publish) a message, it must know its "name", called a *topic*. It is possible to have several ROS nodes that subscribe or publishe the same topics, or one node that does it for several topics, so in the ROS constitution we find *services*, which are synchronous transactions "watchers". They prevent the wrong node from getting the wrong message at the wrong time.

For humanoid robots, ROS hardware abstraction and low-level device control, are very handy tools, saving up the strenuous task of programming low-level functions for kinematics, dynamics, sensor/actuator communication, etc ...

ROS is an asset of invaluable worth in the programming of robots. Although it may not be the best solution overall, it is certainly the best documented due to the large online community and the distribution under a *BSD* license (permissive free software license). The ROS community creates and maintains software repositories for drivers and algorithms, making the software very flexible and malleable to a variety of systems.

The inclusion of ROS in the PHUA project will represent the next milestone completed. In that sense, after having tested some of preliminary versions of this work's systems, it was decided that using ROS would allow for a more efficient robot system construction and maintenance. The software developed in that sense is fully described in Chapter 5. The ROS tool "RViz" was used to create graphical visualizations of the haptic environments, the robot/joystick frames and other debugging constructions.

Chapter 4

Robot Kinematics and Interactive Command

Having defined the hardware setup, there is a need for mathematical models that define the two kinematic chains that will be used for the haptic demonstrations: the arm and the detached leg. The mathematical expressions for forward, inverse and differential kinematics expressions for these chains will be presented. Further, two distinct methodologies for robot command will be detailed. The position command methodologies actuates the robot joints by sending position commands to the servomotors through inverse kinematics and the velocity command methodology actuates the robot *via* the update of each joint instantaneous velocity value, using differential inverse kinematics.

4.1 Kinematic Models

In this work, two different kinematic chains were used to create a control structure needed to support the haptic demonstrations: the robot arm and the support leg (detached from the hip), as depicted in Figure 4.1. As the analogue servomotors were disconnected from the communication line and mechanically immobilized, their degrees-of-freedom were unused and had to be discarded.

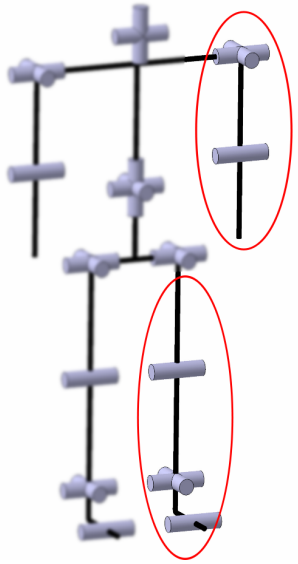


Figure 4.1: Kinematic chains used in the work.

For the arm, restricting the analogue servomotor removed one degree-of-freedom from the chain, from 4 to just 3DOF. This chain was then composed of the shoulder's flexion/extension and abduction/adduction joints and the elbow flexion/extension joint, starting in the shoulder and ending in the forearm end-point (wrist).

For the legs, as the robot disassembly procedures determine that they must be mechanically detached from the pelvis, it is impossible to maintain the hip's three degrees-of-freedom (2 digital plus 1 analogue) on the detached leg kinematic chain, leaving it with just 3DOF. These degrees-of-freedom are the ankle's flexion/extension and abduction/adduction joints and the knee flexion/extension joint. The chain starts in the ankle and ends in the thigh's end-point (hip abduction axis interface). It was not possible to include the hip's degrees-of-freedom in this kinematic chain because the thigh joints are assembled as a part of the robot's pelvic sub-assembly (with the exception of the analogue motor), which includes the other leg's hip joints and the robot's torso rotation joint.

Although only the left side chains are presented, this work is valid for both left and right side arms/legs. The forward and inverse kinematics of the right side limbs are calculated as if it was the left side kinematic model and the joint angles are then converted to their right side equivalent. This is possible due to symmetry in the reference frames of each chain.

Note: In this text, non-explicit references are made to specific recurrent mathematical expressions for equation/matrix shortening purposes. They are hereby summarized for future reference:

- ***rotx(), roty(), rotz() and trans()***
  These expressions refer to the elementary geometrical transformation matrices. The *rotx(), roty(), rotz()* expressions are associated with the x, y and z axis rotation matrices and *trans()* refers to the three-dimensional translation matrix.
- $\boldsymbol{C_1}$**,** $\boldsymbol{C_2}$**,** $\boldsymbol{C_{23}}$**, ... and** $\boldsymbol{S_1}$**,** $\boldsymbol{S_2}$**,** $\boldsymbol{S_{23}}$**, ...**
  These condensed expressions are reductions of the notation for the sine (*sin*) and cosine (*cos*) functions applied to a kinematic chain's degrees-of-freedom, as $cos(\theta_1)$, $sin(\theta_2)$, $sin(\theta_2 + \theta_3)$, etc....
- $\boldsymbol{^AT_B}$ **and** $\boldsymbol{(^AT_B)^{-1}}$
  This notation represents generic geometric transformation matrices. In this example, the transformation occurs from frame A to frame B in first case, while the second case represents the inverse transformation, going from frame B to frame A.

4.1.1 3DOF Arm

Forward Kinematics

The forward kinematics for the 3DOF left arm was set using the *Denavit-Hartenberg* method, with the parameters shown in Table 4.1. The arm is built with two links of lengths L_1 and L_2 and three joints, being a universal joint (θ_1, θ_2) at the shoulder and a rotational joint at the elbow (θ_3). Zero position (home configuration) was defined as the arm at full stretch downwards.

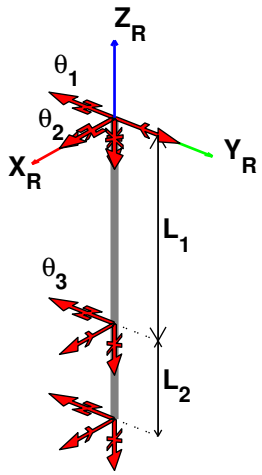


Figure 4.2: Arm (left side) frame system according to the *Denavit-Hartenberg* method.

Table 4.1: Denavit-Hartenberg table for the 3DOF arm.

Link	**l**	**d**	$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$
1	0	0	θ_1	$-\pi/2$
2	L_1	0	θ_2	$\pi/2$
3	L_2	0	θ_3	0

As the Denavit-Hartenberg notation places the primary axis (Z_0) aligned with the first joint, the end-effector coordinates of the chain will be given under that frame. In order to correct them, a transformation matrix, defined as equation (4.1) shows, must be pre-multiplied so that the those coordinates are rotated back into the world reference frame (Z_R). This reference frame was placed in the respective shoulder, so there would be no need for a translation transformation in-between.

$$^{R}T_0 = rotx(\frac{\pi}{2}) \times rotz(\frac{-\pi}{2}) \tag{4.1}$$

Applying the Denavit-Hartenberg algorithm with the parameters in Table 4.1, and pre-multiplying by transformation (4.1), it was possible to obtain the robot shoulder-to-wrist forward transformation matrix ($^{Shoulder}T_{Wrist}$), equation (4.2) , in the world frame.

$$^{S}T_W = \begin{bmatrix} C_1S_3 + C_2C_3S_1 & C_1C_3 - C_2S_1S_3 & S_1S_2 & L_1C_2S_1 + L_2(C_1S_3 + C_2C_3S_1) \\ C_3S_2 & -S_2S_3 & -C_2 & S_2(L_1 + L_2C_3) \\ S_1S_3 - C_1C_2C_3 & C_3S_1 + C_1C_2S_3 & -C_1S_2 & L_2(S_1S_3 - C_1C_2C_3) - L_1C_1C_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Through the elements in this matrix it is possible to retrieve the forward kinematics expressions for the robot wrist cartesian position, in the shoulder frame, shown in equation (4.3).

$$\begin{aligned} X_{arm} &= {}^{S}T_W(1,4) = L_1C_2S_1 + L_2(C_1S_3 + C_2C_3S_1) \\ Y_{arm} &= {}^{S}T_W(2,4) = S_2(L_1 + L_2C_3) \\ Z_{arm} &= {}^{S}T_W(3,4) = L_2(S_1S_3 - C_1C_2C_3) - L_1C_1C_2 \end{aligned} \tag{4.3}$$

The expressions for RPY (*Roll*, *Pitch* and *Yaw*) angles can also be retrieved from the transformation matrix 4.2, by comparing to the theoretical $RPY(\phi, \theta, \psi)$ matrix, in equation (4.4).

$$\begin{aligned} RPY(\phi,\theta,\psi) &= trans(p_x,p_y,p_z) \times rotz(\phi) \times roty(\theta) \times rotx(\psi) \\ &= \begin{bmatrix} C_\phi C_\theta & -S_\phi C_\psi + C_\phi S_\theta S_\psi & S_\phi S_\psi + C_\phi S_\theta C_\psi & p_x \\ S_\phi C_\theta & C_\phi C_\psi + S_\phi S_\theta S_\psi & -C_\phi S_\psi + S_\phi S_\theta C_\psi & p_y \\ -S_\theta & C_\theta C_\psi & C_\theta S_\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4.4}$$

With a small mathematical reordering, the outcome are the expressions for $Roll\,(\phi)$, $Pitch\,(\theta)$ and $Yaw\,(\psi)$.

$$\phi = \arctan\left(\frac{{}^S T_W(2,1)}{{}^S T_W(1,1)}\right) \Leftrightarrow \phi = \arctan\left(\frac{C_3 S_2}{C_1 S_3 + C_2 C_3 S_1}\right) \tag{4.5}$$

$$\theta = \arctan\left(\frac{-{}^S T_W(3,1)}{{}^S T_W(1,1) C_\phi + {}^S T_W(2,1) S_\phi}\right) \Leftrightarrow \theta = \arctan\left(\frac{C_1 C_2 C_3 - S_1 S_3}{(C_1 S_3 + C_2 C_3 S_1) C_\phi + C_3 S_2 S_\phi}\right) \tag{4.6}$$

$$\psi = \arctan\left(\frac{{}^S T_W(3,2)}{{}^S T_W(3,3)}\right) \Leftrightarrow \psi = \arctan\left(\frac{C_3 S_1 + C_1 C_2 S_3}{-C_1 S_2}\right) \tag{4.7}$$

Inverse Kinematics

Closed form expressions for inverse kinematics are determined by combining and re-arranging the forward kinematics equations and isolating the desired joint angle variables.

Equation (4.8) shows the the expression for the elbow flexion/extension joint angle (θ_3) as a function of the end-effector space coordinates, which is achieved by squaring and adding the forward kinematics expressions for X, $Y$and Z coordinates (4.3).

$$\begin{aligned} X_{arm}^2 + Y_{arm}^2 + Z_{arm}^2 &= {L_1}^2 + {L_2}^2 + 2C_3 L_1 L_2 \Leftrightarrow \\ &\Leftrightarrow \theta_3 = \pm \arccos\left(\frac{X_{arm}^2 + Y_{arm}^2 + Z_{arm}^2 - {L_1}^2 - {L_2}^2}{2L_1 L_2}\right) \end{aligned} \tag{4.8}$$

By combining and re-arranging expressions for X and Z, the expression for the shoulder flexion/extension joint angle (θ_1) can be obtained as a function of the arm's cartesian position and the elbow flexion/extension joint angle (θ_3), as equation (4.9) shows.

$$\begin{cases} \dfrac{X_{arm} - L_2 C_1 S_3}{C_2 S_1} = L_1 + L_2 C_3 \\ \dfrac{Z_{arm} - L_2 S_1 S_3}{-C_2 C_1} = L_1 + L_2 C_3 \end{cases} \Leftrightarrow S_1 Z_{arm} + C_1 X_{arm} = L_2 S_3 \Leftrightarrow$$

$$\Leftrightarrow \theta_1 = 2\arctan\left[\frac{Z_{arm} \pm \sqrt{X_{arm}^2 + Z_{arm}^2 - (L_2 S_3)^2}}{X_{arm} + L_2 S_3}\right] \tag{4.9}$$

For the shoulder abduction/adduction joint angle (θ_2), expressions for Y and Z coordinates are manipulated to achieve equation (4.10), showing the joint angle value also as function of the end-effector cartesian position and the two previous joint angles.

$$\begin{cases} S_2 = \dfrac{Y_{arm}}{L_1 + L_2 C_3} \\ C_2 = \dfrac{-C_1\,(L_1 + L_2 C_3)}{Z_{arm} - L_2 S_1 S_3} \end{cases} \Leftrightarrow \theta_2 = \arctan\left[\frac{Y_{arm}(Z_{arm} - L_2 S_1 S_3)}{-C_1(L_1 + L_2 C_3)^2}\right] \tag{4.10}$$

There are other solutions for this chain's inverse kinematics expressions. However, the use of the arctangent function (arctan()) is beneficial for programming purposes, allowing the use of the *atan2(y,x)* function, which is an enhanced version of the original, auto-detecting the quadrant of the angle and returning the result in the interval $]-\pi, \pi]$. The angle is positive for counter-clockwise angles (upper half-plane, $y > 0$), and negative for clockwise angles (lower half-plane, $y < 0$).

Differential Kinematics

For differential kinematics the Jacobian matrix (J matrix) and its inverse must be defined, so that equation (4.11) is solvable. This expression allows to determine the end-effector cartesian position increments ($\Delta \boldsymbol{r}$) given it's current posture and joint angular increments ($\Delta \boldsymbol{q}$). Similarly, the inverse is true if the J matrix admits an inverse (J^{-1}). It is possible to deduce an analogous expression, as equation (4.12) shows, in where the position increment vector represents the three-dimensional velocity vector ($\dot{\boldsymbol{r}}$) of the end-effector, and the result of the operation with the J^{-1} matrix is the joint angular velocities ($\dot{\boldsymbol{q}}$) for that posture.

$$\Delta \boldsymbol{r} = J(\boldsymbol{q})\Delta \boldsymbol{q} \Leftrightarrow \Delta \boldsymbol{q} = J^{-1}(\boldsymbol{q})\Delta \boldsymbol{r} \tag{4.11}$$

$$\dot{\boldsymbol{q}} = J^{-1}(\boldsymbol{q})\dot{\boldsymbol{r}} \tag{4.12}$$

For forward differential kinematics the J_{arm} matrix is then defined as expression (4.13) shows, using the forward kinematics expressions (4.3) presented before.

$$J_{arm}(q) = \begin{bmatrix} \dfrac{\partial X_{arm}}{\partial \theta_1} & \dfrac{\partial X_{arm}}{\partial \theta_2} & \dfrac{\partial X_{arm}}{\partial \theta_3} \\ \dfrac{\partial Y_{arm}}{\partial \theta_1} & \dfrac{\partial Y_{arm}}{\partial \theta_2} & \dfrac{\partial Y_{arm}}{\partial \theta_3} \\ \dfrac{\partial Z_{arm}}{\partial \theta_1} & \dfrac{\partial Z_{arm}}{\partial \theta_2} & \dfrac{\partial Z_{arm}}{\partial \theta_3} \end{bmatrix} =$$

$$= \begin{bmatrix} C_1(L_1C_2 + L_2C_2C_3) - L_2S_1S_3 & -S_2S_1(L_1 + L_2C_3) & L_2(C_3C_1 - S_3S_1C_2) \\ 0 & C_2(L_1 + L_2C_3) & -L_2S_2S_3 \\ L_2C_1S_3 + S_1C_2(L_1 + L_2C_3) & C_1S_2(L_1 + L_2C_3) & L_2(C_3S_1 + S_3C_1C_2) \end{bmatrix} \tag{4.13}$$

For the inverse differential kinematics, the analytical inverse of this matrix was not determined and J_{arm}^{-1} is calculated numerically. This allows for faster algorithms and real time kinematics calculations. By monitoring the value of the matrix determinant for both J_{arm} and J_{arm}^{-1} matrices, it is possible to detect singularity points and other Jacobian limitations.

4.1.2 3DOF Support Leg

As the 3DOF support leg kinematics were already explored by another work in the past [45], it was decided to re-use that same formulation for this work, only introducing minor modifications to suit the nomenclature presented in this text.

Forward Kinematics

The forward kinematics for the 3DOF support legs was also based on the *Denavit-Hartenberg* method, with the frame placement as Figure 4.3 shows and with the parameters described on Table 4.2.The figure depicts a right leg, but both sides were implemented. The leg chain is built with two links of lengths L_1 and L_2, plus the distance to the ground plane L_0, and with three joints, a universal joint (θ_1, θ_2) at the ankle and a rotational joint at the knee (θ_3). Home configuration was defined as the leg at full stretch upwards.

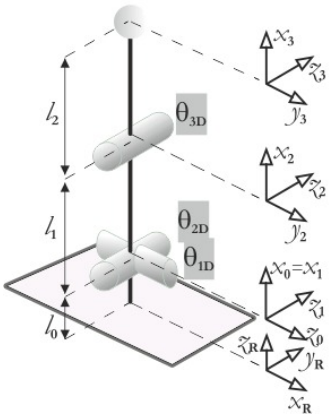


Figure 4.3: Support leg (right side) frame system acording to the *Denavit-Hartenberg* method [45].

Table 4.2: *Denavit-Hartenberg* parameter table for the 3DOF support leg (left side is positive).

Link	**l**	**d**	$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$
1	0	0	θ_1	$\pm\pi/2$
2	L_1	0	θ_2	0
3	L_2	0	θ_3	0

Again, the Denavit-Hartenberg notation places the primary axis (Z_0) aligned with the first joint. In this chain, it was useful to place the world reference frame (Z_R) at the bottom of the robot's feet, so the transformation matrix (4.14) was pre-multiplied to rotate and translate the chain's end-effector coordinates into that frame.

$$ {}^{R}T_0 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & L_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.14} $$

Applying Denavit-Hartenberg's algorithm to Table 4.2, and pre-multiplying by transformation (4.14), it was possible to obtain the robot foot-to-hip forward transformation matrix (${}^{Foot}T_{Hip}$), expression (4.15), in the world frame.

$$ {}^{F}T_{H} = \begin{bmatrix} \pm S_{23} & \pm C_{23} & 0 & \pm L_1 S_2 \pm L_2 S_{23} \\ -S_1 C_{23} & S_1 S_{23} & \pm C_1 & -L_1 S_1 C_2 - L_2 S_1 C_{23} \\ C_1 C_{23} & -C_1 S_{23} & \pm S_1 & L_0 + L_1 C_1 C_2 + L_2 C_1 C_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15} $$

From this matrix we can retrieve the forward kinematics expressions for the cartesian position in the reference frame Z_R, shown in equations (4.16), (4.17) and (4.18).

$$ X_{leg} = {}^{F}T_{H}(1,4) = \pm L_1 S_2 \pm L_2 S_{23} \tag{4.16} $$

$$ Y_{leg} = {}^{F}T_{H}(2,4) = -L_1 S_1 C_2 - L_2 S_1 C_{23} \tag{4.17} $$

$$ Z_{leg} = {}^{F}T_{H}(3,4) = L_0 + L_1 C_1 C_2 + L_2 C_1 C_{23} \tag{4.18} $$

The expressions for *Roll*, *Pitch* and *Yaw* angles can also be retrieved from the transformation matrix 4.15, comparing it to the $RPY(\phi, \theta, \psi)$ matrix. The result are expressions (4.19), (4.20) and (4.21).

$$ \phi = \arctan\left(\frac{{}^{F}T_{H}(2,1)}{{}^{F}T_{H}(1,1)}\right) \Leftrightarrow \phi = \arctan\left(\frac{-S_1 C_{23}}{\pm S_{23}}\right) \tag{4.19} $$

$$ \theta = \arctan\left(\frac{-{}^{F}T_{H}(3,1)}{{}^{F}T_{H}(1,1)C_\phi + {}^{F}T_{H}(2,1)S_\phi}\right) \Leftrightarrow \theta = \arctan\left(\frac{-C_1 C_{23}}{\pm S_{23} C_\phi - S_1 C_{23} S_\phi}\right) \tag{4.20} $$

$$ \psi = \arctan\left(\frac{{}^{F}T_{H}(3,2)}{{}^{F}T_{H}(3,3)}\right) \Leftrightarrow \psi = \arctan\left(\frac{-C_1 S_{23}}{\pm S_1}\right) \tag{4.21} $$

Inverse Kinematics

The closed form inverse kinematics expressions are again determined combining and re-arranging the forward kinematics equations and isolating the desired joint angle variables. This solution is given for all joint angles in the form of the solution in θ to the generic expression $k_1 \sin\theta + k_2 \cos\theta = k_3$, in equation (4.22), where k_1, k_2 and k_3 known constants.

$$ \theta = \arctan\left(\frac{k_1}{k_2}\right) \pm \arctan\left(\frac{\sqrt{{k_1}^2 + {k_2}^2 - {k_3}^2}}{k_3}\right), \quad \text{if } \left({k_1}^2 + {k_2}^2\right) \geq {k_3}^2 \tag{4.22} $$

The inverse kinematics parameters are then presented on Table 4.3. Parameter A assumes whether the value $(Z_{leg} - L_0)/C_1$ or $-Y_{leg}/S_1$ according to the quadrant to avoid divisions by zero. Like before, this solution takes the advantage of the use of arctan() functions, improving the resulting code robustness to numerical uncertainties.

Table 4.3: The 3DOF support leg inverse kinematics solution.

$\boldsymbol{\theta}_i$	$\boldsymbol{k}_1$	$\boldsymbol{k}_2$	$\boldsymbol{k}_3$
θ_1	$Z_{leg} - L_0$	$\mp Y_{leg}$	0
θ_2	$\pm 2L_1 X$	$2L_1 A$	$X_{leg}^2 + A^2 + {L_1}^2 - {L_2}^2$
θ_3	0	$2L_1 L_2$	$X_{leg}^2 + A^2 - {L_1}^2 - {L_2}^2$

Differential Kinematics

For forward differential kinematics the J_{leg} matrix was defined as expression (4.23) shows, using the forward kinematics expressions (4.16), (4.17) and (4.18), presented before. For the inverse differential kinematics, the analytical inverse of this matrix was not determined and J_{leg}^{-1} is calculated numerically.

$$J_{leg}(q) = \begin{bmatrix} \frac{\partial X_{leg}}{\partial \theta_1} & \frac{\partial X_{leg}}{\partial \theta_2} & \frac{\partial X_{leg}}{\partial \theta_3} \\ \frac{\partial Y_{leg}}{\partial \theta_1} & \frac{\partial Y_{leg}}{\partial \theta_2} & \frac{\partial Y_{leg}}{\partial \theta_3} \\ \frac{\partial Z_{leg}}{\partial \theta_1} & \frac{\partial Z_{leg}}{\partial \theta_2} & \frac{\partial Z_{leg}}{\partial \theta_3} \end{bmatrix} = \tag{4.23}$$

$$= \begin{bmatrix} 0 & L_1C_2 + L_2C_{23} & L_2C_{23} \\ -L_1C_1C_2 - L_2C_1C_{23} & L_1S_1S_2 + L_2S_1S_{23} & L_2S_1S_{23} \\ -L_1S_1C_2 - L_2S_1C_{23} & -L_1C_1S_2 - L_2C_1S_{23} & -L_2C_1S_{23} \end{bmatrix}$$

Center of Gravity

The expressions for the support leg center of gravity COG, matrix (4.24), and its Jacobian matrix J_{legCOG} (4.25) can be written as follows:

$$COG(q) = \frac{1}{M_t} \begin{bmatrix} m_0CM_{0x} + m_1r_1S_2 + m_2(L_1S_2 + r_2S_{23}) \\ -m_1(r_1S_1C_2 + CM_{1y}C_1) - m_2(L_1S_1C_2 + r_2S_1C_{23}) \\ m_0CM_{0z} + m_1(L_0 + r_1C_1C_2 + CM_{1y}S_1) + m_2(L_0 + L_1C_1C_2 + r_2C_1C_{23}) \end{bmatrix} \tag{4.24}$$

$$J_{legCOG}(q) = \frac{1}{M_t} \begin{bmatrix} 0 & \begin{matrix} m_1r_1C_2+ \\ +m_2(L_1C_2 + r_2C_{23}) \end{matrix} & m_2r_2C_{23} \\ \begin{matrix} -m_1(r_1C_1C_2 + CM_{1y}S_1)- \\ -m_2(L_1C_1C_2 + r_2C_1C_{23}) \end{matrix} & \begin{matrix} m_1r_1S_1S_2+ \\ +m_2(L_1S_1S_2 + r_2S_1S_{23}) \end{matrix} & m_2r_2S_1S_{23} \\ \begin{matrix} -m_1(r_1S_1C_2 - CM_{1y}C_1)- \\ -m_2(L_1S_1C_2 + r_2S_1C_{23}) \end{matrix} & \begin{matrix} m_1r_1C_1S_2- \\ -m_2(L_1C_1S_2 + r_2C_1S_{23}) \end{matrix} & -m_2r_2C_1S_{23} \end{bmatrix} \tag{4.25}$$

For these calculations, the information on the robot's foot, leg and thigh (m_0, m_1, m_2) and respective centers of mass positions (CM_0, CM_1, CM_2) are required, as well as the robot's leg and thigh lengths (r_1, r_2).

4.2 Command Methodologies

After defining the servomotor communication strategy, on Section 3.1, methodologies for teleoperation were defined using the mathematical support presented earlier in this Chapter. However, some hardware and software limitations are common to both methodologies presented.

The servomotors have a total course of 180º(±90º) and read their position using a linear potentiometer. The potentiometer's analogue value is read by the integrated circuit, converted into the discrete integer range, and then stored in the volatile memory. It is this value that is returned by the controller when requested, and as the position digital range is of 1800 digital position value, the nominal position resolution can be defined to a theoretical 0.1º.

It was determined that the angular position that the controller responds in the position query has a small error that is not constant neither equal in all servomotors. Except for controller predefined positions, such as 0º, 90ºor 180ºpositions (center and extreme positions, configurable), the angle that the servomotors respond does not match the physical angle measured in the motor (which strangely matches the position command sent). This error is different for each servomotor, including identical models, making it impossible to define an error compensating strategy that suits all cases. The only possible solution for this problem was to define joint specific error correction algorithms, but due to unrelated mechanical complications that could lead to servomotor replacements, it was decided to blindly abide by the motors read value for robot joint state scanning.

Commanding the robot through an RS-232 protocol requires that the commanding unit keeps track of the position and velocity information from the servomotor common line to determine the robot joint state. As previously concluded [45], this message overflow can significantly increase the servomotor response time, as the internal controller is programmed to prioritize "move" commands over "read" orders, so that the control loop is not disturbed. This is another limitation that needs to be taken into account when implementing teleoperation command strategies with the PHUA robot.

So, as the low-level control is in closed-form over the communication line, the PC side of the command structure had to be defined. As shown before, inverse kinematics of both kinematic chains each offer two different solutions.

The first command methodology implemented was in *Position*, meaning that it gave use to the inverse kinematics expressions for the joint angles presented before. Because the servomotors define their own velocity planning, the computations on the PC side are lighter, but the method does not allow the user to specify robot end-effector velocity vectors.

As the J matrices and the differential inverse kinematics were defined, it was decided to implement a control structure that could define the robot's end-effector velocity vector at each loop. The *Velocity* command methodology is more intense in terms of servomotor communication, scanning the robot joint state at each loop in order to correctly calculate the joint course increments that ultimately define the end-effector velocity and Cartesian navigation.

4.2.1 Commanding in Position

The position command defined in this work is a closed loop between the PHANToM joystick and the robot, serving as the intermediary between the user and the world, as explained in the diagram in Figure 4.4. This method defines a constant angular speed for the servomotor line, and sends only position commands.

The coordinate mapping in this loop is relative, rather than absolute, mainly due to the morphological differences between the robot's kinematic chains workspaces and the joystick's own workspace. With this mechanism, the control can be initiated on any robot posture without workspace restrictions, because the control loops are always initiated relatively to the current end-effector position.

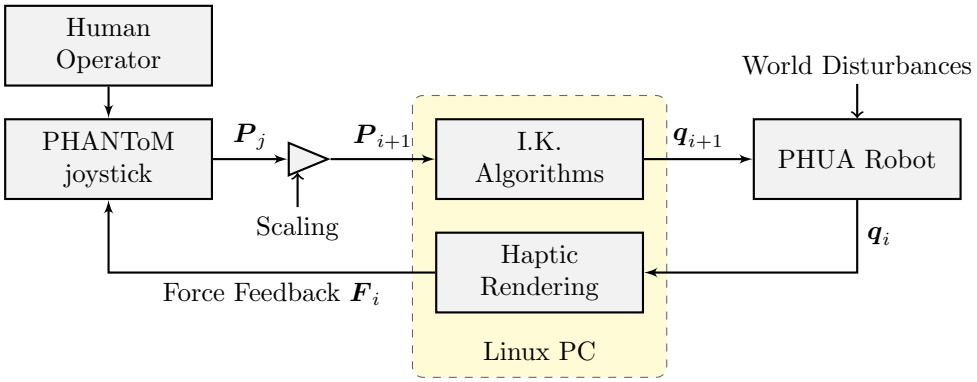


Figure 4.4: Position command diagram, from the user to world interaction.

When the loop is initiated, current robot state $\boldsymbol{q}_i$ and the joystick absolute position are known. Using the robot's joint configuration and the forward kinematics expressions, the current end-effector position is calculated. This information is used as the reference for the loop, meaning that the joystick information is retrieved relatively to that first position and added to the calculated end-effector position, building the vector $\boldsymbol{P}_j$. Then, this increment is adjusted according to a scaling factor that is used to control the precision and sensitivity of the control. This factor controls the workspace scaling of the joystick, meaning that the user can "zoom in", refining the control, but increasing the sensitivity. The vector $\boldsymbol{P}_{i+1}$ contains the next end-effector position to be executed by the robot.

The inverse kinematics equations will then produce the next set of joint angles to send to the robot, $\boldsymbol{q}_{i+1}$. If the communication is successful, the robot will feedback its current joint state $\boldsymbol{q}_i$. Due to the low update frequency possible with the communication protocol implemented, a fallback routine was built to use the last sent joint angle vector if the communication slows down or is unsuccessful.

The force generator consists of algorithms that use the robot current state to generate a force vector $\boldsymbol{F}_i$ to be reproduced by the PHANToM device, interactively, as the user is controlling the robot. These forces are varied in nature and affect the user joystick navigation and effectively influence the position that will be sent to the robot in the next iteration of the loop. Normally, these signals would come from the world, provided by sensors installed on the platform. As those sensors are not currently installed, the solution is to use simple models to generate those forces, which will be described in detail in Chapter 5.

None of the 3DOF manipulators used in this work allow redundancies at the third joint (elbow or knee joints), due to not being mechanically possible to execute negative angle values, and so the inverse kinematics solutions were restricted to "elbow-down" postures, for both kinematic models.

This command strategy using inverse kinematics has the advantage of direct coordinate mapping, but also singularity point avoidance mechanisms. It is possible to build algorithms where singularities do not represent a significant limitation, at the cost of a difficult end-effector velocity control. This methodology is able to keep track of both the robot joint state and the joystick's Cartesian position during execution.

This point by point methodology of command combined with this kind of digital servomotors defines the robot end-effector velocity by the number of points per second sent. If the servomotor angular velocities are set to the maximum value, the motors will reach the sent positions very quickly. So, if the user navigation velocity vectors are adequate, and the servomotors are parametrized to move without overshoot, the robot will be able to match those end-effector velocities perfectly. It was determined, by testing with several operators, that the velocity matching is seamless and any potential desynchronization issues never appeared or posed a problem to the users.

Although the PHANToM software and drivers communicate with the device at a very high frequency (≈1kHz), the communication with the robot servomotor line is very slow (≈25Hz) and depends on the number of servomotors present. To maintain the integrity of the rendered signals, the haptic force generator cannot wait for the robot state to update. Thus, the rendered force vector components are only updated after the robot state is updated (and the end-effector estimated position is corrected), but the force rendering mechanism is continuous and the software ensures it is executed at the highest possible frequency.

Frequencies of around 100Hz in position command can be achieved if the force vector update does not wait to confirm the end-effector Cartesian position. If the algorithm uses the previous sent joint configuration as the current state, trusting the robot to have performed it correctly and using the forward kinematics to estimate the current end-effector position, the only messages sent to the servomotor common line are the position commands for the respective motors.

4.2.2 Commanding in Velocity

The velocity command implemented in this work is very similar to the position commanding methodology, except in that the information retrieved from the joystick is its three-dimensional velocity vector, as the diagram in Figure 4.5 depicts. The commands sent to the servomotors are also different, with position and velocity commands sent at every iteration. This control method was only applied to the robot arms, since that for the support leg the control had different goals.

With this methodology there is no direct coordinate mapping between the robot's end-effector and the joystick position. The information taken from the joystick is its end-point velocity vector at every iteration.

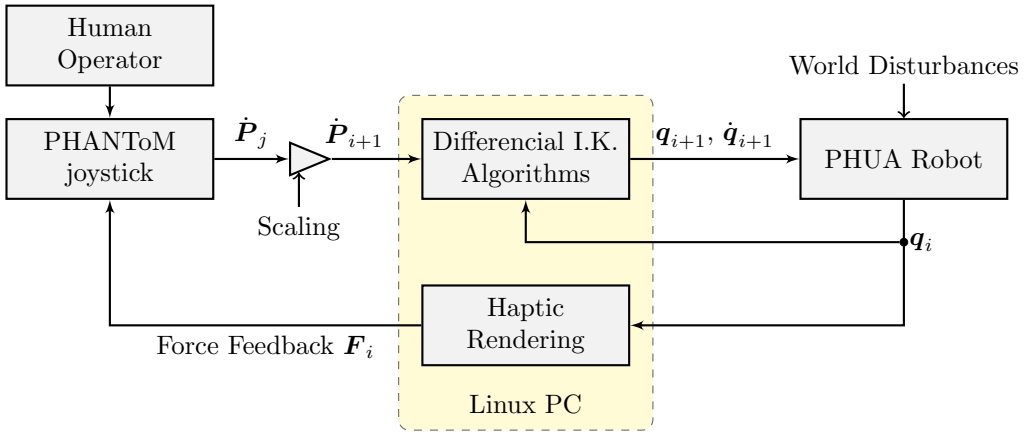


Figure 4.5: Velocity command diagram, from the user to world interaction.

End-point velocity vectors are calculated from the joystick workspace navigation over time, and sent forward at every iteration. As the previous case, a scaling factor is introduced to define sensitivity, but in this case it does not define the workspace coordinate scale, but the velocity vector magnitude scaling. It was expected to retrieve the velocity information directly from the haptic joystick, but there was an unidentified hardware problem with the device and the velocity vector retrieved was not reliable. To counter this issue and proceed with the work, the velocity vector is estimated by monitoring the raw Cartesian position increments of the joystick, time stamped by a timer set on the *PC*. This information is then accumulated and mean filtered during run time. The size of this accumulation is adjustable in the code, and can be used to damp this velocity estimation.

Like before, when the loop is initiated, current robot state $\boldsymbol{q}_i$ is known and updated at every iteration. The current joint angle values are used to update the differential inverse kinematics algorithms, that use the velocity information $\dot{\boldsymbol{P}}_{i+1}$ from the joystick to create the next set of

joint angles and joint angular instantaneous velocities, $\boldsymbol{q}_{i+1}$ and $\dot{\boldsymbol{q}}_{i+1}$, to be sent to the servo line.

This control method actuates the robot by continuously varying the joint velocity values. The sign of each joint velocity determines the direction of the servo rotation, achieved by sending position commands to the joint limits. A zero value for a joint velocity means that joint will receive a command containing the same position angle value it fed back in the previous iteration.

Inverse differential kinematics has the clear advantage of allowing all manipulator configurations to take place in a natural, somewhat unbounded way. The only drawback are the singularity points, which require special dedication by the algorithms so that the user experience is not hindered by a mathematical limitation. In these points, the forward J matrix determinant becomes zeroed and thus that matrix cannot be inverted. Further, as the end-effector approaches these points, the J matrix determinant reaches values that are computationally valid for inverting the matrix, but produce unrealistically high velocity values, generating very high accelerations in the motors that induce unnecessary stress and wear. To avoid these points, the determinant of the Jacobian matrix is watched to prevent singularities and high magnitude joint velocity values.

The forces generated with this command method are similarly generated as in the position command case. The force generator uses the forward kinematics to track the robot end-effector Cartesian position, using the joint values feedback $\boldsymbol{q}_i$. Because the rendered force vector is only updated at the rate of the robot servomotor line update frequency, very erroneous results are produced. The system keeps evaluating postures that the robot has already executed several loops before, due to the joints being constantly moving (towards their limit values). The user feels obstacles and signals that are no longer a limitation of the robot in the real world, but may have been in the past.

The only solution to this problem is to lower the velocity vectors magnitude that are sent to the robot, meaning that either the operator must restrain himself from executing high velocity trajectories, or the system must crop the generated vectors to reasonable magnitude values. However, neither case is acceptable. Cropping the velocity vectors by software is the same as dampening the interaction between user and robot, producing a very unrealistic robot command. As the force rendering around unreachable or singularity points produces high forces, and therefore high accelerations, at a very high update frequency, the human operator is not capable of a rapid suitable response and generates high velocity vectors unwillingly.

Furthermore, the trajectory planner in the servomotor's controller is not designed to generate motions using a methodology based on velocity instead of position values. It is difficult to ensure the servomotor stops at the right position when the Jacobian matrix fails, because due to both the slow communications and the internal controller's giving priority to the position reaching instead of the velocity adaptations.

This command methodology was then deprecated for the demonstrations robot command, because it was considered that the position commanding approach produced better results. Also, the differential inverse kinematics issues with singularities would also require a different approach to allow the reachable workspace to match that which is achieved with the position commanding methodology.

Chapter 5

Tele-Kinesthetic Interaction and Haptic Demonstrations

In this chapter, the software application to support the three distinct haptic demonstrations developed is presented. The application is internally separated into threads, to take advantage of all available processing power and haptic joystick's libraries, and possesses a custom built graphical user interface. The haptic demonstrations detail three different approaches for haptic signals integration on teleoperation events and also on enriched data logging applications. The workspace limits demonstrations tries to ensure the operator does not perform incorrect motions by avoiding workspace limitations, virtualized as haptic objects. An object interaction demonstration placed the teleoperated robot interacting with a real world object. A leg balancing demonstration tries to show the usefulness of haptic signalling in balance applications.

5.1 Application Developed

5.1.1 Overview

As stated before, the PHANToM haptic device is the primary interface for robot control. However, early tests quickly showed that the simple shell terminal information output of the application could not provide the level of flexibility required for keeping track of the robot state in real time, while maintaining high frequency haptic rendering information output. It was then decided that a user-friendly front-end had to be created to improve the robot command and teleoperation experience, the overall efficiency and debug effectiveness while testing.

This interface eventually evolved into a more mature approach. A *C/C++* Linux application was designed to improve user-friendliness, to maximize information output and provide a certain level of hardware abstraction, all trying to maintain execution speed, critical for haptic rendering.

The application is multi-threaded, in an effort to minimize potential execution speed reductions from heavy kinematics or haptics algorithms. To ease the robot manipulation it is equipped with user-friendly menus for kinematic chain selection, robot joint state information output, shortcuts with the joystick buttons and safety routines. The application is also able to perform joint-by-joint robot control, inverse kinematics testing, individual joint velocity and position definition, low-level servomotor control, joystick calibrations, etc...

The code was developed to be portable to other platforms and languages, in the case it will ever be needed. There was an effort to create a *Doxygen*-based documentation that would be helpful in understanding the code structure, the APIs used and all the functions developed.

Many libraries used in the creation of this application are not regularly distributed with the usual Linux distributions. On Appendix C are hardware and software related instructions for all

dependencies of the application.

At the start of this work, it was uncertain if the *ROS* framework was to be introduced into the platform, and the application was not built to support it. Even though in the end it became a functional *ROS* node, some of its functionalities were unfortunately not implemented.

5.1.2 Application Structure

The need to easily export the code to other formats and the need for very high execution frequencies led to the design of a compartmentalized application, divided among three separate threads: Main thread, HD/HLAPI thread and the Graphical Interface thread, as shown in the diagram in Figure 5.1. The choice of multi-threading over "forking" was due to the high memory consumption of some of the libraries that were used, but also to take full advantage of the OpenHaptics libraries which already had this as a pre-requisite.

Further, multi-threading, specially on multi-core/multi-processor systems, is able to use latent *CPU* time associated to idle processes, boosting the computational power accessible to the applications. Old systems however, as the *PC-104* computer that is a part of the robot, will not be able to take the full advantage from this, but the code should be compilable nonetheless. As most computational systems today possess multi-threading capabilities, it should not be an issue in the future.

As the the diagram in Figure 5.1 illustrates, all three threads communicate between them through a shared data structure which, combined with thread-safe coding, provides an elegant information flow structure. Each individual thread is designed to process specific individual tasks, in so that no overlapping occurs.

This structure allows the code to easily be ported or expanded. The same logic applied here can work with *ROS* nodes, or even by introducing other threads. Most of the base functions developed can easily be recycled into new applications.

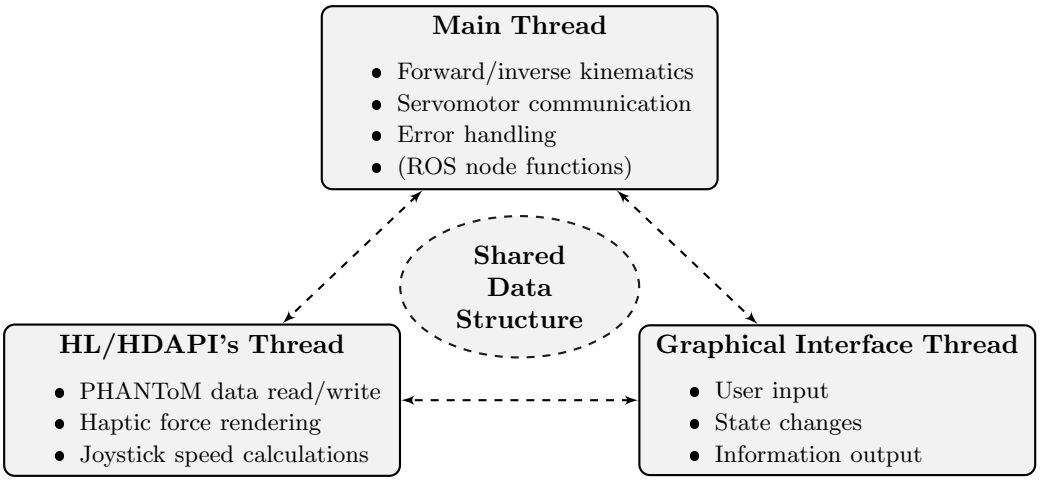


Figure 5.1: Developed application structure diagram, showing the three separate threads communicating through a shared data structure.

Main Thread

The main thread is where the application's main loop resides, as well as where all the other threads are launched, the shared data structure is instantiated and the *ROS* node and message publishers are initiated.

Before the application's main loop starts, a series of routines take place that verify calling arguments, hardware state and communications. The program can be called using different arguments, or none at all.

When no argument is passed to the application, the program execution will abruptly stop if any of the threads fail to launch, the robot communication is determined to be impossible or some other unspecified error occurs. Alternatively, the argument "- -debug" causes the application to ignore these hardware verification procedures and attempt to load all functionalities without terminating execution. It is mainly used to test the interface callbacks, or joystick functions when the robot is not operational.

If the joystick is not detected by the PDD (*PHANToM Device Drivers*), the HDAPI does not allow to create none of the hardware-dedicated *C/C++* classes to manage and communicate with the device, as well as its dedicated thread (the *Servo Loop Thread*), detailed ahead.

If any motor in the robot's servomotor communication line does not respond, robot joint state scanning and robot commanding are disabled. All of the kinematics functions work as usual, but the robot message orders are not actually sent.

However, the graphic interface is not critical. If it fails to load, or its thread cannot be properly launched, program execution continues normally (although now it can only be stopped by sending "kill" signals through the operating system).

Although the communication with the robot is done *via* RS-232, the computer available for this work had no serial port so a *USB* converter had to be used. In some Linux kernel versions, the "ttyUSB" object number associated with the same physical port can change, thus making the application programming a bit more challenging. To counter this inconvenience, the application can be called with the path to the correct object, overriding the default object usage, "/dev/ttyUSB0". For future versions, automatic *USB* port scanning may be the best solution.

Calling the program with the argument "- -help" will print to the console a patch of text with usage options and instructions. This argument was initial designed just to test user-friendliness, but it was later maintained and merged into the main code.

After all the hardware verifications are cleared, the application will start its *ROS* node. This node publishes the PHANToM wrist and pen tip frame transforms, force vectors created and the haptic object data. The publish frequency is given by the main loop execution frequency, which in turn is variable with the amount of calculations it must perform (apart from *CPU* related issues).

The main loop is where the application begins. The loop waits for the user to change states, whether through the joystick buttons or though the interface. While it loops, the joystick, the robot state, the error state and all exit parameters are updated. Inside the main loop is another nested loop that runs the robot control and respective inverse kinematics.

The control loops whenever the operator moves the joystick more than a user-configurable value. It was determined that the joystick's precision was too high for the low communication frequency of the servo line. While trying to stand still, the operator is unable to stabilize the joystick while keeping the position fluctuations below the nominal position resolution. Due to roundings in the algorithms and the servomotor's position reading error, the robot kinematic chains could not reach such low resolution in position, and these small joystick fluctuations in position did not affect the end-effector position. At every loop, inverse kinematics calculations gave approximately the same value, which are then converted to the servomotor digital position range and sent to the robot, overflowing the communication line with consecutive position commands that the servos had already received. This waste of *CPU* time was countered by introducing a condition for triggering the robot control, defined by a minimum position increment the operator must perform.

The exit parameter forces the loop to terminate and perform a series of clean-up routines along with forcing all other threads to close correctly. If all goes well, the application returns the system macro "EXIT_SUCCESS" (or "EXIT_FAILURE" otherwise).

HL/HDAPI's

The thread assigned to haptics is a specification of the OpenHaptics suite. In an attempt to ensure the 1kHz update frequency goal for haptic rendering, both API's required the so called *Servo Loop Thread*, which is, basically, a high priority thread designed to exponentiate the low-level interaction between the libraries and the PHANToM hardware.

This thread opens an infinite loop structure, with a built-in execution cue for user functions, that can be assigned to run consecutively in an ordered, prioritized and thread-safe manner. Although these functions have pre-defined prototypes and recommended structures, their content, return value and execution order are user-defined. They are conveniently called "callbacks", a nomenclature that derives from the background of these API's, the technical specifications of mechanisms on virtual environment creation and rendering.

These callbacks can be set to run synchronous or asynchronously. Synchronous functions are thrown in the execution queue, and the main program execution is halted until that function emits its return value. With asynchronous functions the program execution is not halted, allowing the callbacks to run in loop at the maximum possible frequency. Freeing *CPU* cycles and avoid inefficient processing power spending allows this mechanism to have functions with impressively low execution times.

The execution order in the queue is defined by setting the priority (maximum, default or minimum) property of these callbacks, and is managed by a "scheduler" . This scheduler is responsible for appending functions to the queue at each loop of the thread. This mechanism allows for new callbacks to be appended whenever the user requires, synchronous or asynchronously. The loop thread does not depend on the scheduler, as it may be running with a void queue if the scheduler is shut down, but the scheduler can only be launched if the thread is open.

In this work, both types of functions were used. There are three asynchronous functions defined, for distinct uses, that run in loop throughout normal program execution, and two synchronous functions, that are placed in the queue whenever they are required.

- **Main device update** *(Asynchronous, Maximum priority)*
  This function is responsible for the haptic device data extraction, running at the highest possible frequency. It collects data from the joystick, such as position, button states, joystick joint angles, motor temperatures,frame transformations, etc. . . , and inserts that information into an exclusive dedicated data structure. It is also responsible for the device error state handling.

- **Force feedback** *(Asynchronous, Default priority)*
  This function is responsible for rendering force vectors. It is basically a watcher function, that keeps track of the selected kinematic model, looking at the haptic demonstration chosen and generating the correspondent force vector. It also determines if the previously sent force vector was correctly executed by the device.

- **Joystick calibration** *(Asynchronous, Minimum priority)*
  This function checks the joystick internal calibration parameters. If the user decides to calibrate the device this function acts accordingly, but only if required. It was designed for this particular PHANToM device, and will not work with other models.

- **Copy device data** *(Synchronous, Default priority)*
  This function updates the joystick data on the application shared data structure. It is added to the queue whenever it is needed, copying the data from the dedicated structure for continuous device data extraction. It is also responsible for the ${}^{Joystick}T_{World}$ frame transformation using the matrix (5.1), which is applied to the joystick coordinate vector,

rotating it into the world orientation.

$$^{J}T_{W} = rotz\,(\pi/2) \times rotx\,(\pi/2) \tag{5.1}$$

It is not possible to retrieve the joystick pen tip coordinates directly from the device, as this model is only equipped to work with the wrist point. These devices are meant to be used in virtual reality environments, where the physical distance from the wrist to the pen tip frame is irrelevant, and usually used as a configurable parameter to refine the *VE* experience. To suit this purpose, the device outputs a single transformation matrix that contains the wrist point 3DOF position and its 6DOF orientation components. This information on orientation was then used to determine the pen tip frame, by defining the transformation as a translation, whose corresponding translation vector represents an approximate physical distance (manually measured) from the wrist to the tip. The pen tip coordinates are then updated in the shared data structure, and the transformations will be later published as *ROS* messages in the main thread.

- **Retrieve device parameters** *(Synchronous, Default priority)*
  This function runs only once, prior to the main loop. It is used to retrieve four of the device's nominal parameters for force rendering, that will later be used to establish upper and lower bounds for the force generator algorithms. These parameters are the nominal maximum stiffness and damping (N/mm) values, and the nominal maximum force (N) values renderable instantly and continuously.

Graphical User Interface (GUI)

The graphical user interface, as stated before, was a non-scheduled need that emerged from the early tests with the robot and the joystick. It soon became clear that the format of information output had to be a little more complex than the simple *ASCII* text possible with the terminal, but at the same time become user-friendly to the operator.

This interface was built using GTK+ (*Gimp ToolKit*), in *C/C++* languages. The early versions of this interface were dedicated to show the simple robot joint angle values. When both arms control was implemented, the interface was altered to allow changing the kinematic model interactively, without the need to recompile a new application. From that point on, several new graphical additions were successively introduced to improve overall user-friendliness and the user overview of the robot state. To test these changes, the interface was submitted to unfamiliarized users for input on the user-friendliness and access to information. The joystick buttons shortcuts are a clear example of this interaction's usefulness, because they were introduced as recommendation from one of those unfamiliarized operators.

The interface main page, shown in Figure 5.2, is the primary interface for the user when teleoperating the robot. The left side of the window is dedicated to robot command and state, and outputs the current joint angle values, updated during runtime. According to the kinematic model selected (through the dropdown menu or the joystick button 2) the interface also outputs the correspondent end-effector Cartesian positions. It is also possible to switch control methods and define servomotor velocity values. On the bottom right side, the parameters for control resolution and workspace scaling can be specified. On the top right side, the haptic data is shown according to the demonstration selected and a set of bars outputs the force vector components every time they are sent to the device.

On the bottom of the page lay buttons for closing the application, robot state updating or setting the robot to its home position. Starting/stopping the robot control loop *via* the interface button is equivalent to using the joystick button 1.

The status bar shows the communications status and frequencies throughout the application.

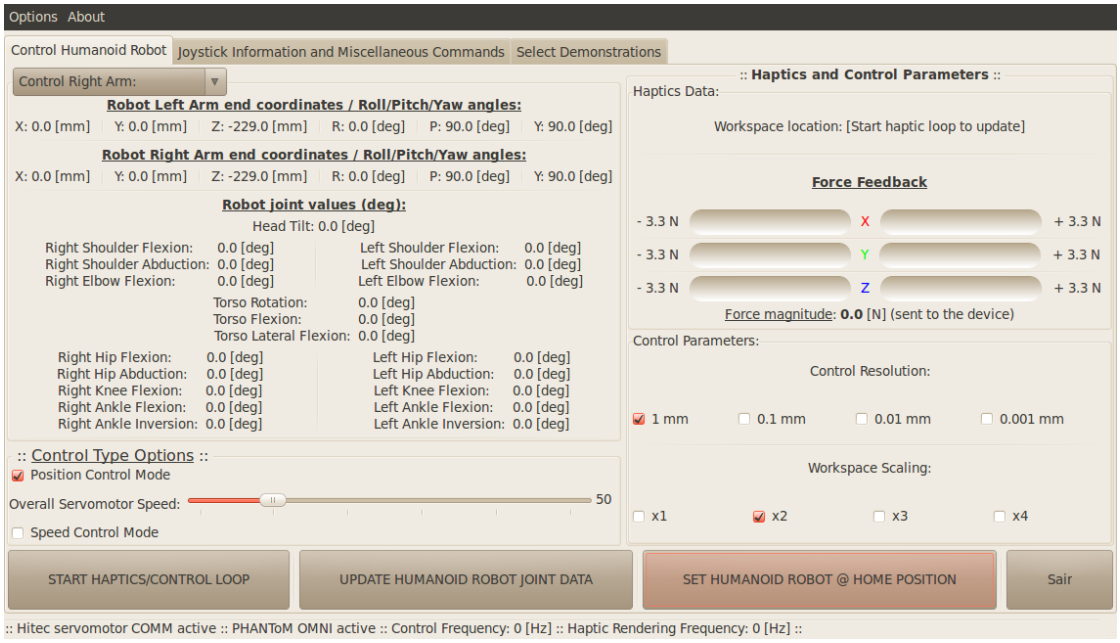


Figure 5.2: The graphical user interface main page, dedicated to the robot control and haptic feedback information output.

The interface miscellaneous page, shown in Figure 5.3, contains, on the left, the individual robot joint control and inverse kinematics testings, and on the right the extracted joystick information output, updated at approximately 22 *fps* (*frames per second*). The device calibration can also be tested using the dedicated button. On the bottom lay the buttons for active/passive servomotor commands.

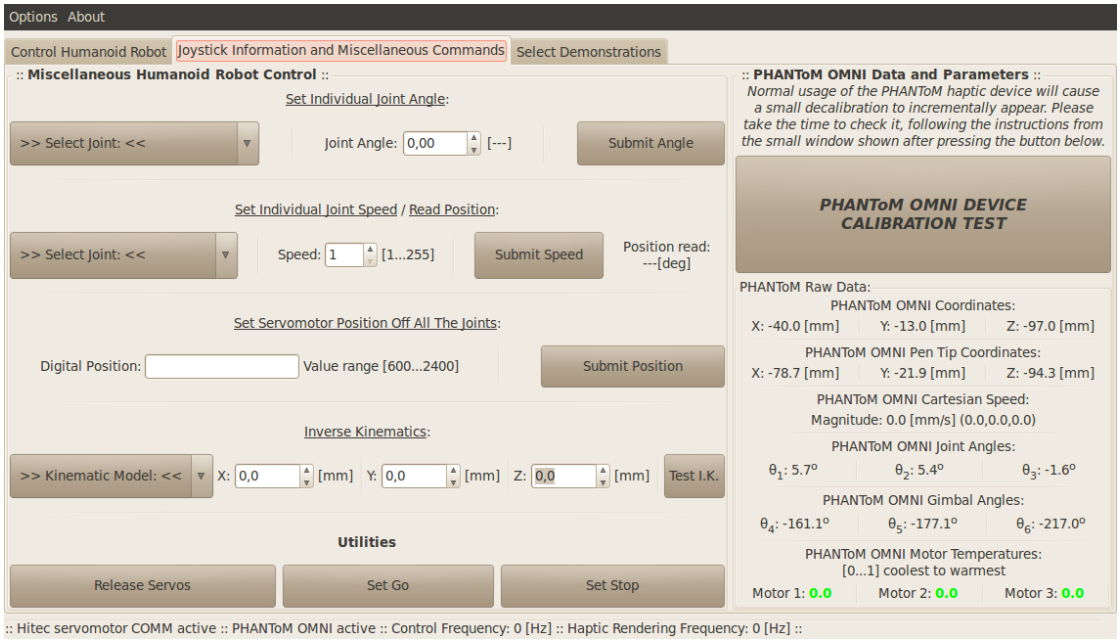


Figure 5.3: The graphical user interface miscellaneous page, dedicated to the robot individual joint control and the device information output.

The interface demonstrations page, shown in Figure 5.4, is dedicated to the choice and description of the demonstrations created. These demonstrations will be thoroughly detailed on Section 5.2 of this chapter.

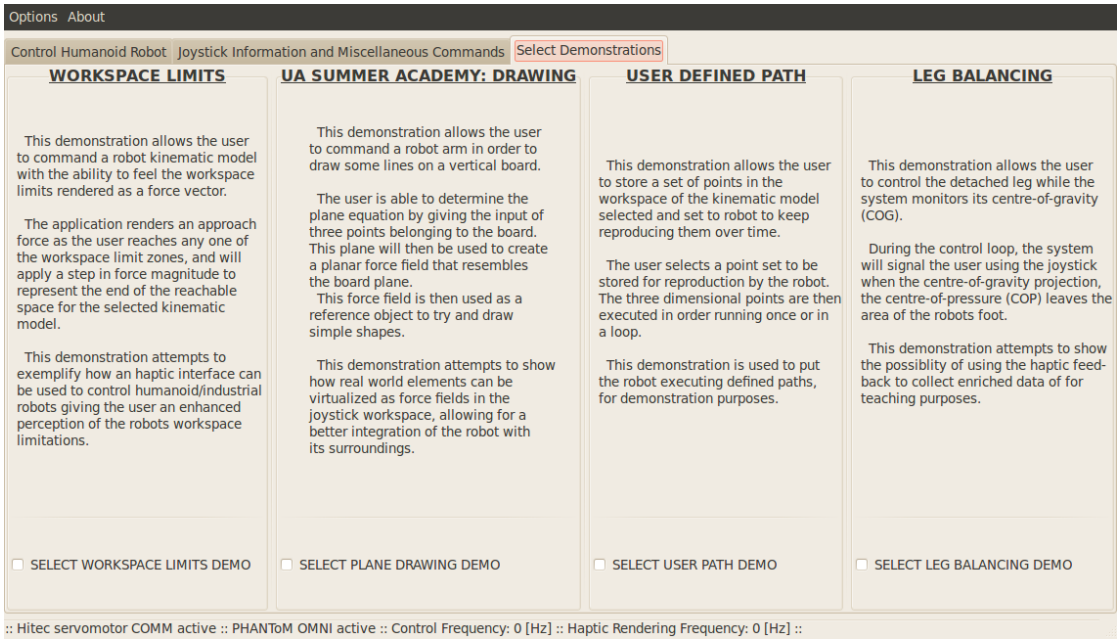


Figure 5.4: The graphical user interface demonstrations page, where the user can select/deselect an haptic demonstration.

Shared Data Structure

The shared data structure is a *C/C++ struct* variable composed of several other sub-structures, variables and classes. It is shared throughout the whole application using C/C++ *pointers*. Some of its sub-structures are exclusive to a single thread or to a pair threads that communicate between them. It suits the same purpose of shared memory blocks in "forked" applications.

This structure is roughly composed of:

- **Servomotor communication class**
  This class is responsible for establishing and maintaining communications with HITEC servomotors connected *via* RS-232 through a *USB* adaptor. This class allows to easily send orders to the servos, such as position and velocity commands, define internal controller parameters or define active/passive motor states. It is shared between the main and GUI threads.

- **Robot state structure**
  This structure holds all of the robot's joint angle values, end-effector positions and orientations and both legs centres-of-gravity. It is shared by all the threads.

- **GTK widgets structure**
  This structure is exclusive to the GUI thread, and contains a set of GTK widget pointers that need to be shared throughout the interface to ease the development.

- **Device state data structure**
  This structure holds all the haptic device data and parameters that are extracted with

the HDAPI. It contains the joystick button states (pressed/clicked), position, joint angle values, motor temperatures, frame transformations, error states and other internal device parameters. It is shared by all the threads and some of its components are published by the *ROS* node.

- **Haptics data structure**
  This structure holds all the haptic data of the application and it is used mainly by the demonstration algorithms. It is shared throughout the application and some of its components are also published by the *ROS* node.

- **General parameters structure**
  This structure contains a set of general parameters that the application internally needs to share, such as update rates, application state variables, etc... It is mainly used for main thread/graphical thread communication.

- **Humanoid functions class**
  This class was designed to hold all the robot geometrical information, like segment lengths and masses or joint angle limitations. It contains several functions used to work with kinematics (direct/inverse/differential) and convert joint angles to servomotor digital range values (and *vice-versa*).

- **Miscellaneous**
  There are other variables, structures and classes shared throughout the application. They were not compartmentalized properly since their respective code is not yet definitive or simply because there is no category for them. Examples are the graphical thread "mutex" (anti-deadlock object), the HDAPI device and context classes, application miscellaneous state variables, etc...

5.2 Haptic Demonstrations

Due to the robot currently lacking sensors that allow to retrieve real world information, and thus work with it to force feedback the system, the haptic demonstration possibilities become somewhat limited.

The solution found was to synthesise the external forces, using as much robot information as possible, which in this case was the robot joint angle and velocity states, which by turn allow forward kinematics to provide all the rest. This synthesis is a mechanism long presented and validated in the academia [51].

The rendered force vectors originate attractor and/or repulsive force fields, perceived as haptic objects that the user can feel through the haptic joystick. Due to the high frequencies involved, the user can perceive these fields as objects, by correctly defining the stiffness parameters.

5.2.1 Workspace Limitations

This demonstration implements a strategy for generating force vectors based in workspace limitations [52]. While the user is controlling the robot using the position control method, the system is monitoring the respective kinematic model end-effector Cartesian position and renders force vectors to signal the operator when the end-effector is nearing a workspace limitation.

A 3DOF kinematic chain, built with a universal joint, has a workspace composed of intersecting spherical regions.

In the arms, the minimum and maximum reach points radius, relative to the shoulder, define the workspace inner and outer spheres, as illustrated in Figure 5.5. As the shoulder flexion/extension joint is limited to a motion range of -45^{o} to 135^{o} (measured from the home position), two other smaller spheres appear centred on the elbow when that shoulder joint is

at its limits, but only one of them represents a real workspace limit due to the "elbow down" postures limitations. In the arm, an extra limitation was introduced representing the shoulder sagittal plane. The robots arms are not capable of transposing their shoulder sagittal plane, for their abduction/adduction joint have a motion range of 0º to 180º.

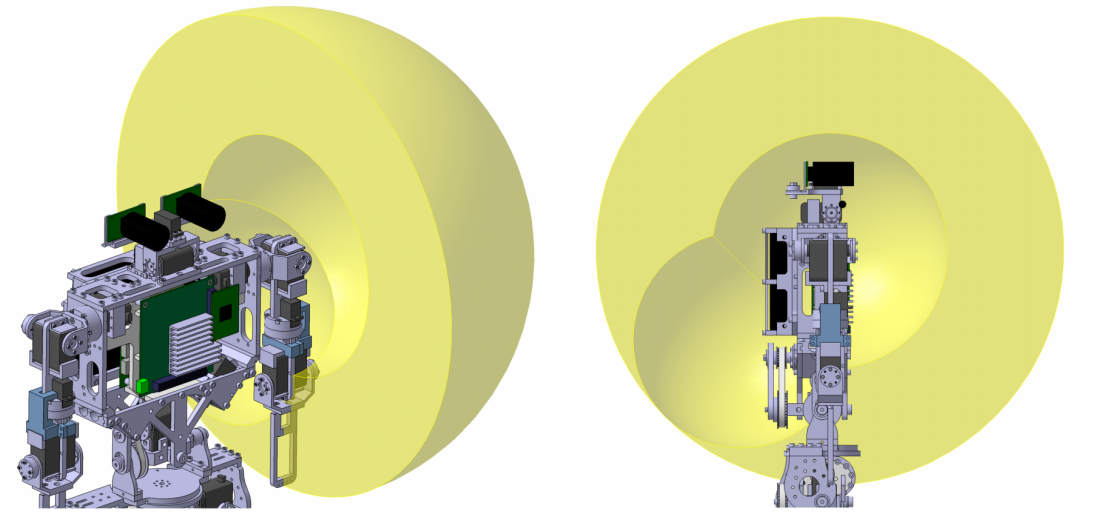


Figure 5.5: Workspace example for the robot arm kinematic chain, resulting from the the two possible postures.

The workspace of the detached leg, as illustrated in Figure 5.6, is also defined by intersecting spherical regions, but with the introduction of two toroidal surfaces that define the reachable space when the flexion/extension joint of the ankle is at its minimum or maximum positions.

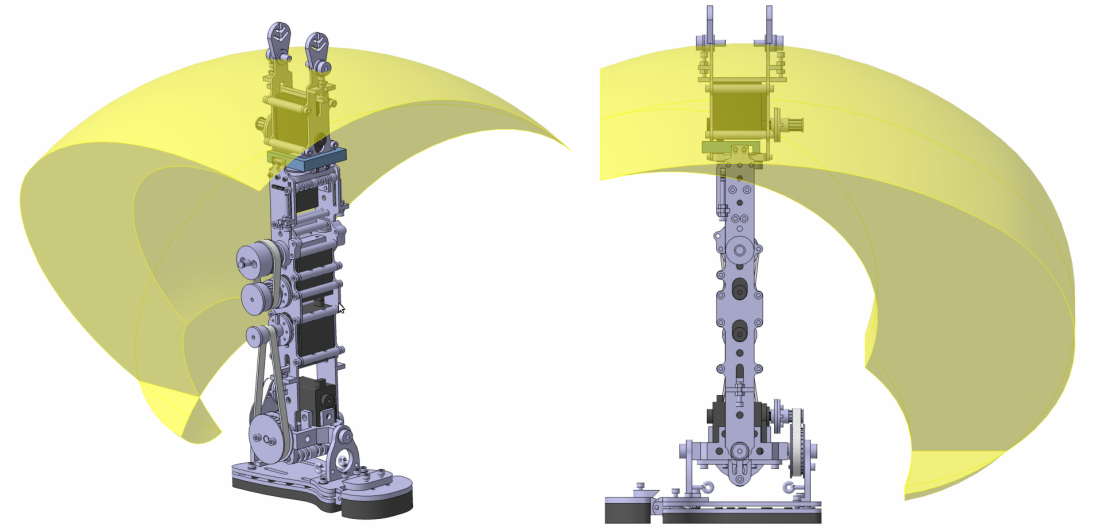


Figure 5.6: Workspace example for the robot detached leg kinematic chain.

When a control loop starts, the robot end-effector position $\boldsymbol{E}$, given in the kinematic chain base frame (and adjusted by the scaling k_{scale}), is subtracted to the current device position $\boldsymbol{P}j_i$, as equation (5.2) shows, to determine the robot workspace spheres centre position $\boldsymbol{C}$, in the joysticks workspace.

$$\boldsymbol{C} = \boldsymbol{P}j_i - \left(\frac{\boldsymbol{E}}{k_{scale}}\right) \tag{5.2}$$

This centre position allows to create the haptic spheres independently of the end-effector position and the joystick position when the control loop starts.

While the control loop is underway, the end-effector position is monitored using equation (5.3), by calculating the radius r of the current joystick position $\boldsymbol{P}_j$ in relation to the sphere centre position $\boldsymbol{C}$, in the joystick workspace. The distance to the arms saggital plane is determined using the same equation but adjusted for the Y axis components.

$$r = \|\boldsymbol{P}_j - \boldsymbol{C}\| \tag{5.3}$$

Force rendering takes place at two distinct zones in the workspace. The joystick position is in a closed loop with the human operator *via* the force vector rendering. Defining a rigid force barrier (haptic object), thresholding the Cartesian navigation, becomes unstable at high frequencies due to the human inability to adequately react to high frequency disturbances induced by inappropriate force curve derivative planning.

So, an approach zone $d_{overlap\,zone}$ (in millimetres) was defined were the user is signalled of the proximity to the limit, thus avoid a rapid and steep increment in force rendering when the end-effector position is at the edge of that limit. This zone's length is programmable in the code, but can be associated with an element in the interface, if required.

When the end-effector is within this zone, the user is signalled with a non-linear force vector, whose magnitude is defined by equation (5.4) in terms of the distance to the limit x, and parametrized by expressions (5.5), (5.6), (5.7), (5.8) and (5.9) with the variable stiffness k.

$$F(x) = a \times x^3 + b \times x^2 + c \times x + d, \qquad x \in]0\ ;\ d_{overlap\,zone}] \tag{5.4}$$

$$a = 2 \times \frac{F_{max}}{d^3_{overlap\,zone}} \tag{5.5}$$

$$b = -3 \times \frac{F_{max}}{d^2_{overlap\,zone}} \tag{5.6}$$

$$c = 0 \tag{5.7}$$

$$d = F_{max} \tag{5.8}$$

$$F_{max} = k \times d_{overlap\,zone} \tag{5.9}$$

If the user tries to leave the workspace and moves inward or outward from the reach limits, a force step is added to the maximum force of the approach zone using the expression (5.10), signalling the user that those points are unreachable. Due to its greater magnitude, it helps drive the user's hand towards a navigable workspace zone. The parameter k_2 defines the greater stiffness of this zone.

$$F(x) = F_{max} + k_2 \times (-x), \qquad x \in]-\infty\ ;\ 0] \quad (N) \tag{5.10}$$

Using a modified Hooke's Law expression allows to avoid a steep increase in force magnitude and the subsequent instability it may cause.

The graphical representation of this polynomial formulation for all the scaling factors ($\times 1$, $\times 2$, $\times 3$ and $\times 4$) is shown in Figure 5.7, exemplified for an approach zone of 30mm and a maximum renderable force of 3.3N.

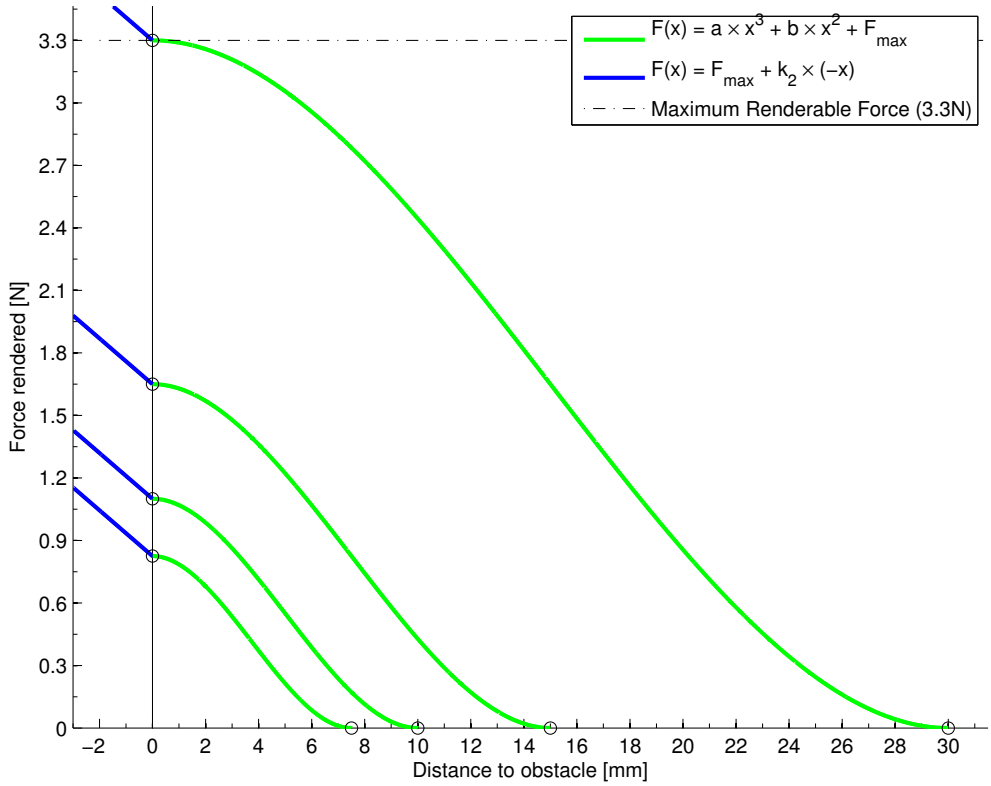


Figure 5.7: Polynomial formulation for the haptic force generation.

The direction of the force vector sent $\boldsymbol{F}$ depends on whether the end-effector is near the minimum (r_{min}) or maximum (r_{max}) reach limits or normally navigating the workspace, as the expression (5.11) shows. Inward or outward direction is defined by calculating the current position radius versor and manipulating its signal.

If no other forces are rendered, a gravity compensating force is sent to the device, whose magnitude is determined by the joysticks apparent mass at the tip m_{tip} and the gravity acceleration vector $\boldsymbol{g}$. The gravity compensation was implemented as a safety measure, after letting the unfamiliarized operators test the system. If the users lost control of the haptic device, due to force generation instability or incorrect hand postures, the closed loop between the joystick position and the end-effector made the robot follow the downward trajectories, and execute random movements that could compromise the robot and the operators integrity.

$$\boldsymbol{F} = \begin{cases} F(x) \times \left(\frac{\boldsymbol{P}_j - \boldsymbol{C}}{r}\right) & \text{if} \quad r \geq (r_{min} + d_{overlap\,zone}) \\ m_{tip} \times \boldsymbol{g} & \text{if} \quad (r_{min} + d_{overlap\,zone}) \leq r \leq (r_{max} - d_{overlap\,zone}) \\ -F(x) \times \left(\frac{\boldsymbol{P}_j - \boldsymbol{C}}{r}\right) & \text{if} \quad r \leq (r_{max} - d_{overlap\,zone}) \end{cases} \tag{5.11}$$

This formulation is useful for progressively signalling the operator of his proximity to the workspace limits, but can be unstable around the limit value. As the decision criteria for establishing if the end-effector is in or out of the workspace is purely mathematical, the small

fluctuations of the user's hand can cause unstable force generation around the workspace limit due to increasing/decreasing the force magnitude at high frequency.

In an effort to make the force rendering seamless to the user, the method was modified to allow force vectors to be calculated without the system knowing the current robot state. It became clear that the position control was sufficiently reliable to have the rendered forces generated according to the next position sent, instead of the previous one. The force generator was placed in a closed loop with the joystick by ignoring the position readout of the robots joints and, allowing for a very fast force rendering scheme with a small de-synchronization that is not noticeable by the operator.

Figure 5.8 shows an example of an RViz-based visualization of the haptic environment created with this methodology. It is possible to view the generated force vector direction and approximate magnitude during runtime. The workspace limits are represented with *ROS* markers.

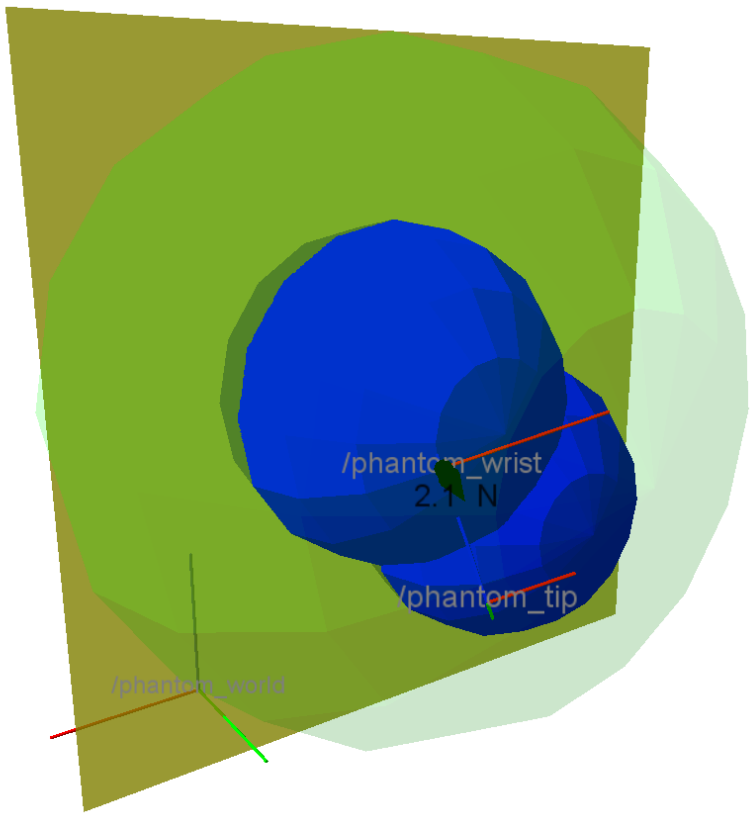


Figure 5.8: Workspace limits demonstration exemplification for the left arm, developed under ROS and RViz. The smaller spheres appear in blue and the outer sphere is shown in faded green, while the yellow plane represents the arm saggital transposition limit. The rendered force value and illustration vector appear in black.

5.2.2 Haptic Object Interaction

One of the advantages of haptic feedback is the possibility of interaction with real world objects. Even without direct sensory data from the robot, a strategy was devised to use real world objects and interact with them *via* the teleoperated robot.

An experiment was proposed for the University of Aveiro Summer Academy activities, because this event represented a unique opportunity to test the system with inexperienced and unfamiliarized operators. The experiment, as shown in Figure 5.9, demonstrated how the system

could be used to teleoperate the robot while attempting to draw line segments on a transparent board.

The marker was placed on the robot's left forearm, approximately 2cm further than the wrist point (end-effector). The transparent board was placed between the operator and the robot, in an attempt to remove the operator's depth perception. If the user has a visual perception of the task, the visuohaptic signals will prevail over tactile-based haptics, and the goal here was to try to understand if it was possible to simply rely on the device to perform the task.

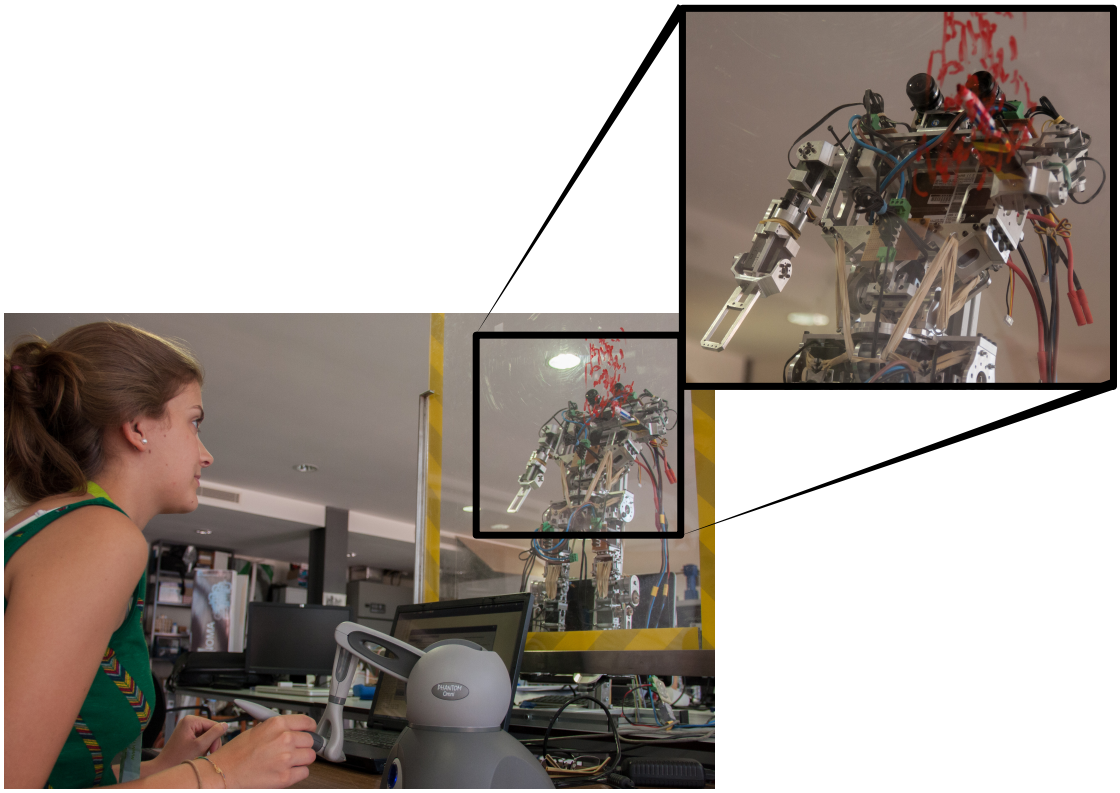


Figure 5.9: Inexperienced operator interacting with a virtualized object while teleoperating the PHUA robot, over the course of the Summer Academy activities.

As there was no hardware to provide the board plane features, or the robot-to-plane transformation matrix, its virtualization had to be accomplished from the haptic device point-of-view, ignoring any possible changes in the robot/plane relative position.

The board plane definition in the device's workspace was made by inputting three points, which by turn meant the user had to blindly press the marker against the board three times to retrieve the set of points $\boldsymbol{P_1}$, $\boldsymbol{P_2}$ and $\boldsymbol{P_3}$. These points were then used to determined the vector pair $\boldsymbol{u}_p$ and $\boldsymbol{v}_p$, that in turn can be used to determine the plane normal vector $\boldsymbol{n}_p$ as the cross product of the previous, as expressions (5.12), (5.13) and (5.14) show.

$$\boldsymbol{u}_p = \boldsymbol{P_2} - \boldsymbol{P_1} \tag{5.12}$$

$$\boldsymbol{v}_p = \boldsymbol{P_3} - \boldsymbol{P_1} \tag{5.13}$$

$$\boldsymbol{n}_p = \boldsymbol{u}_p \times \boldsymbol{v}_p \tag{5.14}$$

This allowed to determine the distance s from the current joystick position (X,Y,Z) to the virtual plane defined by the pair $\boldsymbol{P_1}$ and $\boldsymbol{n}_p$ using expression (5.15), defined with the components of vector $\boldsymbol{n}_p$, and with parameter d determined by the dot product between the negative $\boldsymbol{P_1}$ and the vector $\boldsymbol{n}_p$ components, in equation (5.16).

$$s = \frac{n_x \times X + n_y \times Y + n_z \times Z + d}{\sqrt{{n_x}^2 + {n_y}^2 + {n_z}^2}} \tag{5.15}$$

$$d = -\boldsymbol{P_1} \cdot \boldsymbol{n}_p \tag{5.16}$$

Knowing the distance to the virtualized board plane, it is possible to create an haptic object representing the plane, using the same formulation as in the workspace limits demonstration. However, as the experiment was conducted with inexperienced operators, activating the workspace limitation signalling became clearly necessary.

In order to create a distinction between signals from a workspace limitation and those coming from the targeted planar object, another formulation for the force generation was developed. Such distinction may be required to diminish the operator dependence on the visuohaptic information set, providing a pure haptic tool for object interaction.

Another objective with this new equation set, shown in expression (5.17), was to avoid the instability around the limit of the object interaction still verified with the polynomial formulation of the workspace demonstration. Instability on these zones was hindering to a good task performance, for the end-effector was affected by the force peaks and did not allow to correctly navigate over the surface.

The force magnitude $F(x)$ is again given in function of the distance to the object (s), but now with one additional region d_{limit} after the approach signalling zone $d_{overlap\,zone}$. This new region is modelled by a tanh() type function, so that the force peaks appear within a safety margin of the limit, that together with the position uncertainty from the robot's end-effector allow for a better navigation over the objects surface. Parameters γ, k and k_2 are used to adjust the stiffness of the regions without modifying the functions continuity.

$$F(s) = \begin{cases} 0 & \text{if} \quad s \in [d_{overlap\,zone} \quad +\infty[\\ k \times \gamma \times s & \text{if} \quad s \in [d_{limit} \quad d_{overlap\,zone}[\\ \alpha \times \tanh(\beta + s) + \theta & \text{if} \quad s \in]0 \quad d_{limit}[\\ F_{max} + k_2 \times (-s) & \text{if} \quad s \in]-\infty \quad 0] \end{cases} \tag{5.17}$$

Expressions for β, α and θ parameters are shown on equations (5.21), (5.22) and (5.23), obtained after a series of mathematical manipulations and substitutions, as expressions (5.18), (5.19) and (5.20) resume. These substitutions are only performed for code and representation shortening, and have no mathematical meaning of note.

$$A = \mathrm{e}^{d_{limit}}, \qquad B = \mathrm{e}^{2\,d_{limit}}, \qquad C = \mathrm{e}^{3\,d_{limit}}, \qquad R = \gamma\,k\,(d_{limit} - d_{overlap\,zone}) \tag{5.18}$$

$$D = \sqrt{(A+1)\,(F_{max}+R)\,(F_{max} - 2\,A + F_{max}\,A + R + R\,A + 2)} \tag{5.19}$$

$$E = \ln\left(-\frac{A - C + F_{max}\,A - A\,D + F_{max}\,C + B\,D + R\,A + R\,C}{2\,F_{max}\,A - B + 2\,R\,A + 1}\right) \tag{5.20}$$

$$\beta = \frac{E}{2} \tag{5.21}$$

$$\alpha = \cosh\left(\beta - \frac{d_{limit}}{2}\right)^2 \tag{5.22}$$

$$\theta = F_{max} + \frac{\tanh(\beta)}{\tanh\left(\beta - \frac{d_{limit}}{2}\right)^2 - 1} \tag{5.23}$$

Graphically, this formulation is represented as Figure 5.10 shows, for a maximum renderable force F_{max} of 3.3N, a γ factor of 10 and an approach zone $d_{overlap\,zone}$ of 30mm, and for all the scaling factors ($\times 1$, $\times 2$, $\times 3$ and $\times 4$). The limit zone d_{limit} length is proportional to the approach zone length by a factor of 25%.

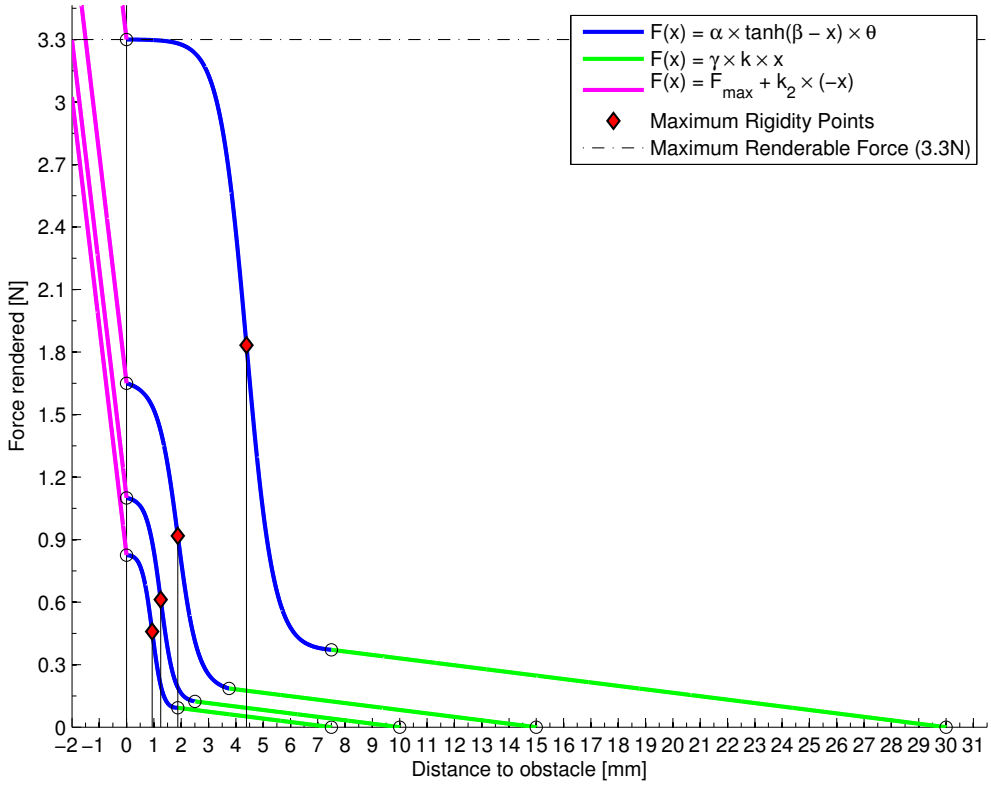


Figure 5.10: Alternate formulation for the haptic force generation.

The force vector $\boldsymbol{F}$ that is sent to the device is then obtained using the versor of the normal vector $\boldsymbol{n}_p$ and the force value $F(s)$ calculated before, as shown on expression (5.24).

$$\boldsymbol{F} = F(s) \times \hat{\boldsymbol{n}}_p \quad (N) \tag{5.24}$$

5.2.3 Support Leg Balancing

The usual means for humanoid balancing/locomotion is through their centre-of-pressure (COP), measured on the feet by pressure sensors or a pressure plate. The COP can be viewed as the dynamical projection of the robot's centre-of-gravity on the feet, and is used widely to track and predict the dynamic behaviour of humanoid robots.

This demonstration is meant to show the potential for haptic signalling in balancing applications, which are fundamentally different from the object interaction or navigation signalling applications. In a balancing situation, and as the operator's goal is to retrieve data sets for learning algorithms, the force rendering cannot overwhelm the user operation, but it must interfere in the control in such a way that the user knows the posture is unachievable.

Unfortunately, at the time, this platform did not have its respective force sensors and dedicated electronics installed on the feet, making it impossible to use that information for force rendering. The alternative devised was to simulate the COP navigation by monitoring the robot's centre-of-gravity (COG), measuring the travelled distance (x and y coordinates) from the home position and detecting the foot area edge crossing.

The force feedback signals of the force vector $\boldsymbol{F}$ are determined directly through the distance to the robot's foot edges, using expression (5.25), which is also illustrated on Figure 5.11.

$$\boldsymbol{F} = \begin{bmatrix} e^{(-x+1)} \\ e^{(-y+1)} \\ 0 \end{bmatrix} \quad (N) \tag{5.25}$$

The graphical representation shows how the force generation for the progression of the distance s, comparing it to the maximum renderable force and constant gravity compensation force levels.

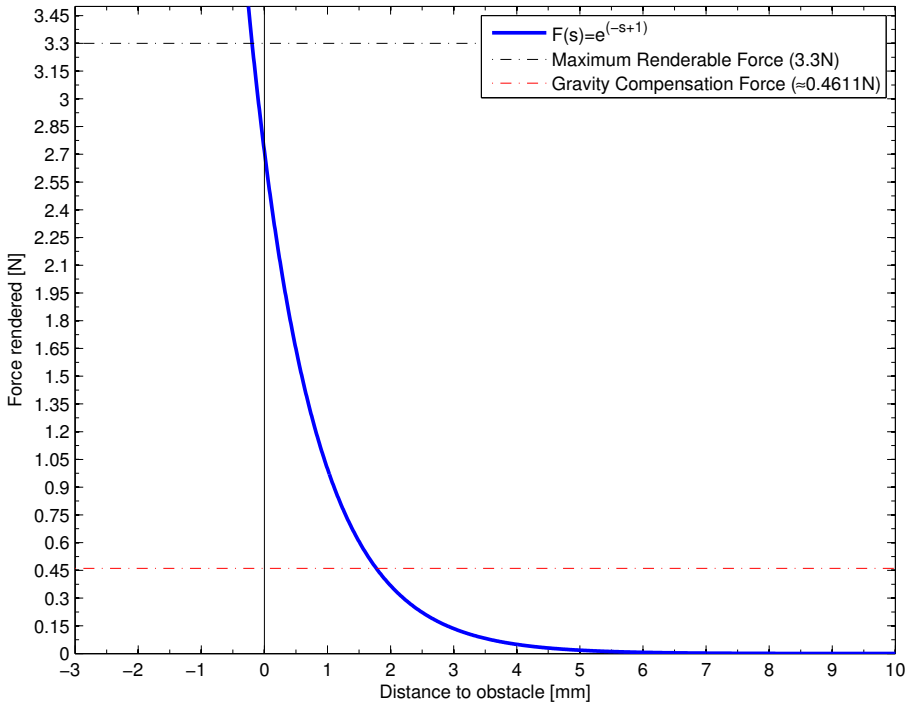
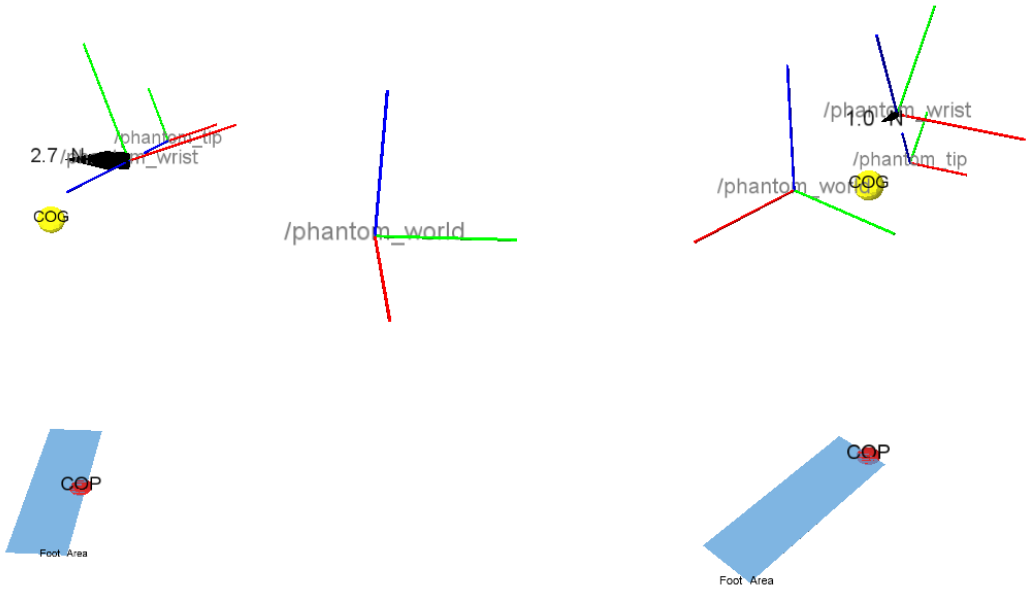


Figure 5.11: Leg balancing force graphical representation, as a function of the distance to the foot edges.

This approach is mathematically valid for static purposes, but incorrect for dynamic behaviours, and so this demonstration is limited to very low control speeds. As the robot state update can only be done at low frequencies, due to the high response time of the servomotors, estimating trustworthy COG velocity vectors and their corresponding differential information is therefore ruled out.

Figure 5.12 shows a visualization of the haptic environment created and the COG/COP workspace navigation representations in RViz environment while teleoperating in a support leg balance situation. The workspace haptic limits representations can also be added to further enrich the data sets.



(a) (b)

Figure 5.12: Detached leg balancing haptic environment examples, developed under ROS and RViz. The COG and COP spheres appear in yellow and red, while the light blue plane represents the foot sole area. The rendered force value and illustration vector appear in black.

Chapter 6

Conclusions and Future Work

The final conclusions of this dissertation will be detailed in this chapter, detailing the success or unsuccess of the initially proposed objectives. For future works, a wide range of proposals are detailed, to suggest future approaches to haptic rendering applications, enriched data logging, software solutions and hardware modification suggestions.

6.1 Conclusions

The main objective of this dissertation was to develop a force feedback haptic control and sensing mechanism, aiming at robot perception / actuation data logging for learning purposes.

The proposed approach can effectively bridge together the robot teleoperation and kinesthetic teaching fields if the world and robot current state are correctly fed back to the user. With the introduction of more dedicated hardware, this approach will still maintain its validity, and naturally allow for a deeper and truer user immersion in haptic control.

It was also proposed the development of a robot SimMechanics simulator, so it would suit the purpose of a predictive and testing tool for locomotion and other intricate humanoid tasks. Although mechanically correct, it was verified that the SimMechanics simulator was not able to describe the dynamic behaviour of the hybrid actuation system, and requires an enormous effort to correctly build a humanoid robot as the PHUA robot.

It was found during the course of this work that the SimMechanics suite possesses yet another limitation for the simulation of the PHUA robot platform. Although the simulation of the dynamical behaviour of the ankle joints with the elastic elements in placed produced the expected results, the same cannot be said about the knee joint. When the knee flexes, the model solver sees the two end points of the spring becoming closer to each other, and so it assumes the spring is being pressed instead of pulled, producing erroneous results for the hybrid actuation system analysis.

Also, as this MATLAB toolkit does not provide any tool for calculating kinematics or perform any kind of trajectory planning within the models, it falls to the user to generate and feed that information into them, increasing the overall computational cost of the simulation (associated with MATLAB scripting) and diminishing the simulator's user-friendliness.

The servomotor wiring was deemed suitable, and it served the purpose without failing. The cable gauge chosen was enough to handle all testing runs without issues of note, and should hold future demands sufficiently well. The connector boards developed also performed well, without overheating. The locations chosen for the installation of these boards was also satisfactory, minimizing cable bending issues in the legs and not restricting the arms and torso's workspaces.

Concerning the mechanical parts for the analogue servomotor's immobilization, their performance was also deemed satisfactory. The parts installed on the arms were correctly machined, and hold the arm with a single pair of bolts. In the legs, however, although the servomotor immobilization was successful, the gripping mechanism is too thin and bends just with the gripping

force from the bolt tightening. The part design was severely restricted by the robot geometry, and the choice of aluminium material was not ideal.

The introduction of new elastic elements to the structure was only significant in immobilizing the torso rotation joint servomotor. There were no visible changes in the servomotor actuation by the introduction of rubber bands in the ankle or hip joints. A coherent and valid test was not possible, but it is recommended nonetheless.

The position control method developed was considered successful. Although it fails to specify end-effector velocity vectors, the velocity can be controlled indirectly by the number of joint angle vectors fed per second. On the velocity control method, due to the velocity measuring problem, the method for the determination of the pen tip velocity is not as exact as the internal joystick controller would be, and due to the mean filtering, the calculated vector is slightly desynchronized from the current one and dampens accelerations. These limitations hinder the control by limiting it to low / medium velocity values and acceleration vectors. However, the velocity control method allows to navigate the entire workspace without adding a significant computational weight to the system, hinting towards the possibility of implementing an hybrid control method using inverse differential kinematics with a position incremented vector adjusted by a standard velocity magnitude.

On the PHANToM joystick, the low-level interaction implemented was considered to be very complete, but heavy on calculations and pre-processing routines, dropping the overall performance and thus the computing frequencies. On the testing computer, with an hardware as described on section C.1, the heaviest haptic rendering demonstration registered frequencies of approximately 500Hz, below the optimum value of 1kHz but still indistinguishable from the user's perspective if the incrementation of the vector's magnitude is controlled. The joystick data gathering routines alone could be performed at frequencies well above 1.5kHz while controlling the robot.

It was verified that the device can be further explored to enhance the operator's perception of the haptic environment. Although the device does not allow to render torques directly, it may be possible to have that effect emulated if the joystick control frequency is high enough. As the forces are applied on the wrist point, it would be possible to consecutively apply perpendicular forces vectors to the pen tip that would be felt by the user as applied torques, which would be useful for rendering inertial signals (forces and torques), for example.

The haptic demonstrations presented fitted the purpose of showing the potential of the approach but do not represent a final version. They are meant to be the precursors of those that will be used for the actual data logging, destined for the future of the platform.

On the workspace limitations demonstration, using custom force generation algorithms was successful for testing purposes, but these formulations are limited in user perception for a fully functional solution. The force generation expressions shown on this work represented an effort to emulate the solutions already offered by the OpenHaptics toolkit, but failed to meet their complexity and sharpened realism.

The same can be said for the transparent board drawing attempt. The force algorithms failed to offer a real object perception to an experienced user, but were satisfactory for the inexperienced users. The familiarization with these interfaces leads to a refined perception of the haptic object's geometry and other subliminal properties, like surface stiffness and texture.

The leg balancing demonstration presented is scarce in content due to the hardware limitations of the platform, but it never meant to represent an actual approach to the balancing problem. It was showed that rendering a force signal is a useful way to provide the user with a perception-based tool that complements the visual information obtained during the balancing data logging stage.

For all haptic-based human-robot interfaces, refining the underlying properties of the haptic objects and signals will ultimately be needed to enhance the user's perception of the virtualized robot world. Object states will require more advanced haptic signals, so that the user can learn to distinguish and prioritize them correctly when data logging, giving rise to the possibility that after the leaning process the robot does that distinction as well.

6.2 Future work

The work in this dissertation was the first of its kind in the PHUA project, therefore probing a number of fields that can now be thoroughly explored. Further, the experience acquired over the course the work allowed to gain a more incisive look on the platform's current issues of note. Future work recommendations are then hereby presented, subdivided by areas of interest, without ever forgetting the project's original goal of enriched data logging for robot learning, on walking and locomotion tasks.

6.2.1 Simulation

For future (dynamic) robot simulations, the *V-REP* software (`www.v-rep.eu`) is strongly recommended. This software has a built-in support for *ROS* nodes, automatic kinematics calculations, dynamic simulators, among several other characteristics.

If the *SimMechanics* simulator is deemed appropriate, a different approach must be taken concerning the hybrid actuation. As is, the model is capable of simulating approximate results for the dynamic behaviour of the directly mounted servomotor limbs, as the arms. For the hybrid joint, that is not possible, and is recommended a study on the matter, looking into other *Simulink* blocks that can be used to emulate the elastic elements impact on the motor's torque.

6.2.2 Software

The full conversion of this work into the *ROS* framework, allowing for a better development and integration of all the system in the platform, is highly recommend for the future of this project. Any work developed from that point on will largely benefit from all the *ROS* embedded tools.

The developed application can be easily ported to *ROS* and maintain it's full functionality, if required. The three thread logic can be expanded into *ROS* nodes, and the shared data structure can easily become a *ROS* message structure. In this sense, the most significant module to develop would be the robot base module, which takes care of the robot actuation and publishes the robot state messages. This would represent the stepping stone of the *ROS* integration for the project. The joystick base module would also be an important step, along with Ethernet node-to-node communication, for it would mean that dual-device setups would be possible. With the current *PHANToM* driver version, that is not possible.

The *ROS* visualizing software *Rviz* allows for a rapid deployment of visual information that is crucial to the development of any software solution for the platform. *ROS*'s sensor integration will be a major fact for its introduction into the platform. In the future, inertial sensors in the structure, force sensors in the feet and a vision system in the head will allow measuring the centres of pressure for the whole system and will be later used to provide force feedback into the haptic tele-kinesthetic interface, so that the operator is able to "feel" the robot's dynamics, improving the data logged towards the goal of walking.

Unfortunately, there is no open-source solution for the low-level pc-to-device communication, other than the OpenHaptics suite, through the *HDAPI*. Even within *ROS* there is no alternative yet. The *phantom_omni* package uses the *HDAPI* to implement a node that publishes a limited amount of information, but still does not provide the same level of hardware interaction as what was achieved in this work. However, exploring and improving this package may be suitable for the projects long term goals.

6.2.3 Haptics

It is clear that haptics and haptic interfaces are the future heading of the PHUA humanoid robot towards the goal of autonomous balancing and locomotion. The presented haptic demonstrations were exploratory of the validity of the approach, but also of the existing programming solutions.

The force rendering schemes presented in this work were not able to provide a "real feel" to the demonstrations, but this can be solved by recurring to the haptic object algorithms that the OpenHaptics suite already possesses. These algorithms are recommended for correctly defining and refining the haptic surfaces and stiffnesses, for these aspects will not be the focus of future works, but merely means to an end. Alternatively, the open-source *H3DAPI* libraries may be recommended for virtual environment creation. An aspect of notice in the OpenHaptics suite is their lack of support for Linux-based programming, which hindered this work heavily.

6.2.4 Command

The velocity command methodology presented on this work was not able to establish a coordinate mapping between the joystick and the end-effector positions. The mechanism used, retrieving the joystick end point velocity vectors indiscriminately throughout the workspace, does not make it possible to do so.

In order to solve this issue, a new strategy was conceived that, due to time limitations, was unfortunately not implemented. It consisted of building what was named an haptic three-dimensional analogue velocity controller. This controller was built with a static haptic rigid sphere, with an anchor-point at its center, and filled on the inside by an attractor force field whose force magnitude is proportional to the travelled radius. The sphere surface represented a maximum velocity magnitude, and the anchor point in the middle represents a zero velocity value. Navigating inside this sphere means creating three-dimensional velocity vectors, scaled in magnitude by the travelled radius (felt as the force magnitude). This method allows the creation of dampened velocity vectors, with the user having complete control over their direction and magnitude. As the sphere is static inside the joystick workspace, there is no danger of reaching the joystick physical workspace limits, and thus leaving a margin for the development for an force feedback methodology for the velocity command of the PHUA robot.

6.2.5 Hardware

On the mechanical aspects, it is still critical the review of the robot's belt and pulley transmission system. Some reductions, as the one on the ankle flexion joint, are too high for the type of belts available, causing the belt's polyurethane body to fail while the robot is in operation. These timing belts pass all the static criteria, and are able to hold the robot in position, but are not able to withstand the robot's dynamic torques for long. In that sense, it is again proposed a revision on the transmission systems of the platform, with a complete survey on available solutions for the belts, as performed on all past and current works on the platform.

The rubber bands that were used as elastic elements do not withstand long exposures to ultra-violet light and thus fail to achieve a satisfactory performance in terms of durability. As the platform still possesses some assembly / disassembly constraints, it becomes gruelling to keep replacing parts in the robot after a work session or minor testing run, or even after a few days idle. It is recommended the development of a conditioning or clustering method for the rubber bands, allowing them be easily replaced in bulk. This may allow to keep the low-cost and easiness of access of the rubber-bands as elastic elements.

Ultimately, the robot's torso joint system and structure will require mechanical modifications due to its overweight. It became clear during the course of this work that the leg motors are not capable of performing a wide variety of movements necessary for walking purposes, even though they succeed on maintaining static postures. On squatting, for instance, if the joint velocities are not well defined, the robot fails to return from the squat posture due to lack of torque in the ankle and hip joints.

Using the leg and hip system separated from the rest of the robot may also be a solution for gait training if this problem is not solved. The drawback is that the sensors in the feet will not record actual usable data for the whole assembled robot applications. Moving the battery set from the back to inside the robot torso may help reduce the upper body overweight issues,

because it will move the robot's centre-of-gravity forward in the sagittal plane, thus modifying torque requirements in the legs.

If all else fails, reviewing the actuators may be a good endeavour, although of a more complex nature. There are solutions in the market for servomotor actuation without some of the restrictions of the current setup, and that would require small but possible mechanical modifications. Higher torques on the lower limbs would allow for wider operation ranges in full body motion patterns, and refined communication protocols could largely improve robot joint state scanning frequencies. These changes would undoubtedly mean the whole project would need to be reviewed, which would undoubtedly mean the development of a new robot. The cost-benefit relation of such a radical enforcement would have to be well studied and weight all the platform's current limitations versus the current potentialities and long term goals of the project.

Finally, a note is left regarding the robot power supply. Although dangerous by nature, the *LiPo* batteries are superb for the robot power supply. However, it was determined that while in operation, they can drain very quickly and without previous notice, lowering their output voltage to very low, harmful, levels. As there is currently no mechanism to warn the operator of such event, it is proposed the installation of a small, affordable buzzer. These devices, which the battery manufacturers also supply, emit a loud buzz (sound signal) warning when the connected power source's voltage drops below a safety value. Furthermore, as the servomotors have specific power supply specifications, some buzzer models can be programmed to warn at specified levels, i. e., the servomotors minimum supply voltage.

During the course of this work, a 12V/20A DC power supply was acquired to serve the purpose of a tabletop power supply for workbench testing runs, in an effort to avoid unnecessary battery wear. Unfortunately, due to a shortage of DC-DC converters on the supplier that goal was not achieved. In that sense, the completion of this workbench power supply could prove to be an invaluable tool for future works, and it is thus recommended.

Bibliography

[1] A. Billard, S. Callinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), New York: Springer, 2008.

[2] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, no. 57(5), pp. 469–483, 2009.

[3] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *EEE Trans. on Systems, Man and Cybernetics, Part B*, no. 37(2), pp. 286–298, 2007.

[4] P. Kormushev, S. Calinon, and D. G. Caldwell, "Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input," *Advanced Robotics*, no. 25(5), pp. 581–603, 2011.

[5] D. Lee and C. Ott, "Incremental kinesthetic teaching of motion primitives using the motion refinement tube," *Autonomous Robot*, no. 31, pp. 115–131, 2011.

[6] A. Cardoso, L. Gomes, N. Pereira, M. Silva, V. Santos, and F. Silva, "Desenvolvimento de uma plataforma humanoide autónoma de custo limitado: Componentes e soluções tecnológicas," *Revista Robótica*, no. 63, pp. 14–23, 2006.

[7] V. Santos, R. Moreira, M. Ribeiro, and F. Silva, "Development of a hybrid humanoid platform and incorporation of the passive actuators," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, pp. 679–684, 2010.

[8] R. Hollis, "Haptics," in *Berkshire Encyclopedia of Human-Computer Interaction* (W. Bainbridge, ed.), pp. 311–316, Berkshire Publishing Group, 2004.

[9] M. A. Srinivasan, *What is Haptics?* Massachusetts Institute of Technology: Laboratory for Human and Machine Haptics: The Touch Lab, 2012.

[10] P. Cruz, V. Santos, and F. Silva, "Tele-kinesthetic teaching of a humanoid robot with haptic data acquisition," in *Intelligent Robots and Systems (IROS): Learning and Interaction in Haptic Robots Workshop*, (Vilamoura, Portugal), 2012.

[11] D. Morris, H. Tan, F. Barbagli, T. Chang, and K. Salisbury, "Haptic feedback enhances force skill learning," in *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint*, pp. 21 –26, 2007.

[12] C. E. Sherrick, "Touch as a communicative sense: Introduction," *The Journal of the Acoustical Society of America*, vol. 77, no. 1, pp. 218–219, 1985.

[13] J. C. Craig and G. B. Rollman, "Somesthesis," *Annual Review of Psychology*, vol. 50, no. 1, pp. 305–331, 1999.

[14] V. Hayward, O. Astley, M. Cruz-Hernandez, D. Grant, and G. Robles-De-La-Torre, "Haptic interfaces and devices," *Sensor Review*, vol. 24, no. 1, pp. 16–29, 2004.

[15] V. Fulkar, M. Shivramwar, and A. Alkari, "Applications of haptics technology in advance robotics," in *2010 International Conference on Emerging Trends in Robotics and Communication Technologies (INTERACT)*, pp. 273 –277, 2010.

[16] K. Salisbury, F. Conti, and F. Barbagli, "Haptic rendering: introductory concepts," *IEEE Computer Graphics and Applications*, vol. 24, pp. 24 – 32, Apr. 2004.

[17] Immersion Corporation, *Haptics in Touch Screen Hand-Held Devices*. Apr. 2012.

[18] R. Van der Linde, P. Lammertse, E. Frederiksen, and B. Ruiter, *The HapticMaster, a new high-performance haptic interface*. The Netherlands: FCS Control Systems, 2012.

[19] Sensable Technologies, *Specifications for the PHANToM Desktop and PHANToM Omni haptic devices*. Jan. 2009.

[20] *PHANToM OMNI User's Guide*. Boston, USA: SensAble Technologies, 2008.

[21] Haption, *Virtuose 6D Desktop Data Sheet*. France: Engineering Systems Technologies (EST), 2012.

[22] Force Dimension, *Omega 6 haptic device data sheet*. 2012.

[23] Force Dimension, *Delta 6 haptic device data sheet*. 2012.

[24] Sensable Technologies, *Specifications comparison for the PHANToM Premium 1.0, 1.5, 1.5 High Force, and 3.0 Haptic Devices*. Jan. 2009.

[25] Haption, *Virtuose 6D 6D35-45 Data Sheet*. France: Engineering Systems Technologies (EST), 2012.

[26] Haption, *Virtuose 6D 6D40-40 Data Sheet*. France: Engineering Systems Technologies (EST), 2012.

[27] M. H. Vu and U. J. Na, "A new 6-DOF haptic device for teleoperation of 6-DOF serial robots," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, pp. 3510 –3523, Nov. 2011.

[28] Quanser, *5-DOF Haptic Wand Data Sheet*. 2012.

[29] Force Dimension, *Sigma 7 haptic device data sheet*. 2012.

[30] T. Hulin, K. Hertkorn, P. Kremer, S. Schatzle, J. Artigas, M. Sagardia, F. Zacharias, and C. Preusche, "The DLR bimanual haptic device with optimized workspace," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3441 –3442, May 2011.

[31] D. Feygin, M. Keehner, and R. Tendick, "Haptic guidance: experimental evaluation of a haptic training method for a perceptual motor skill," in *10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2002. HAPTICS 2002. Proceedings*, pp. 40 –47, 2002.

[32] O. J. Rösch, K. Schilling, and H. Roth, "Haptic interfaces for the remote control of mobile robots," *Control Engineering Practice*, vol. 10, pp. 1309–1313, Nov. 2002.

[33] C. H. Park, J. W. Yoo, and A. M. Howard, "Transfer of skills between human operators through haptic training with robot coordination," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, (Anchorage, Alaska, USA), pp. 229–235, IEEE, May 2010.

[34] S. Calinon, P. Evrard, E. Gribovskaya, A. Billard, and A. Kheddar, "Learning collaborative manipulation tasks by demonstration using a haptic interface," in *International Conference on Advanced Robotics, 2009. ICAR 2009*, pp. 1 –6, June 2009.

[35] P. Kormushev, D. Nenchev, S. Calinon, and D. Caldwell, "Upper-body kinesthetic teaching of a free-standing humanoid robot," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3970 –3975, May 2011.

[36] L. Hamon, P. Lucidarme, E. Richard, and P. Richard, "Virtual reality and programming by demonstration: Teaching a robot to grasp a dynamic object by the generalization of human demonstrations," *Presence: Teleoperators and Virtual Environments*, vol. 20, no. 3, pp. 241–253, 2011.

[37] Y. Koike, K. Nakakoji, and Y. Yamamoto, "Tele-kinesthetic interaction: using hand muscles to interact with a tangible 3D object," in *ACM SIGGRAPH 2006 Emerging technologies*, SIGGRAPH '06, (New York, NY, USA), ACM, 2006.

[38] E. Guizzo and E. Ackerman, "The rise of the robot worker," *IEEE Spectrum International*, vol. 49, pp. 28–35, Oct. 2012.

[39] R. Godinho, *Desenvolvimento do Tronco e Braços de uma Plataforma Humanoide Híbrida*. Master's thesis, University of Aveiro, 2011.

[40] R. Sabino, *Estrutura Híbrida de Locomoção para um Robô Humanoide*. Master's thesis, University of Aveiro, 2009.

[41] M. Ribeiro, *Desenvolvimento dos Sistemas Sensorial e Motor Para Um Robô Humanoide*. Master's thesis, University of Aveiro, 2010.

[42] V. Santos, R. Moreira, and F. Silva, "Hybrid actuation and distributed control for a small-size humanoid robot," in *Introduction to Modern Robotics* (D. Chugo and S. Yokoda, eds.), iConcept Press, 2011.

[43] L. Wang, Y. Zhao, M. O. Tokhi, and S. C. Gharooni, "Modeling and simulation of humanoid stair climbing," in *Climbing and Walking Robots* (M. O. Tokhi, G. S. Virk, and M. A. Hossain, eds.), pp. 73–80, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[44] A. Protopapadaki, W. I. Drechsler, M. C. Cramp, F. J. Coutts, and O. M. Scott, "Hip, knee, ankle kinematics and kinetics during stair ascent and descent in healthy young individuals," *Clinical Biomechanics*, vol. 22, pp. 203–210, Feb. 2007.

[45] W. Lage, *Algoritmos de Controlo do Movimento para um Robô Humanoide*. Master's thesis, University of Aveiro, 2011.

[46] A. Silva, O. Ramirez, V. Vega, and J. Oliver, "PHANToM OMNI haptic device: Kinematic and manipulability," in *Electronics, Robotics and Automotive Mechanics Conference, 2009. CERMA '09.*, pp. 193 –198, Sept. 2009.

[47] *Programmer's Guide*. OpenHaptics Toolkit version 3.0, Boston, USA: SensAble Technologies, 2008.

[48] *API Reference Manual*. OpenHaptics Toolkit version 3.0, Boston, USA: SensAble Technologies, 2008.

[49] M. Quigley, E. Berger, and A. Y. Ng, "STAIR: hardware and software architecture," in *AAAI 2007 Robotics Workshop.*, 2007.

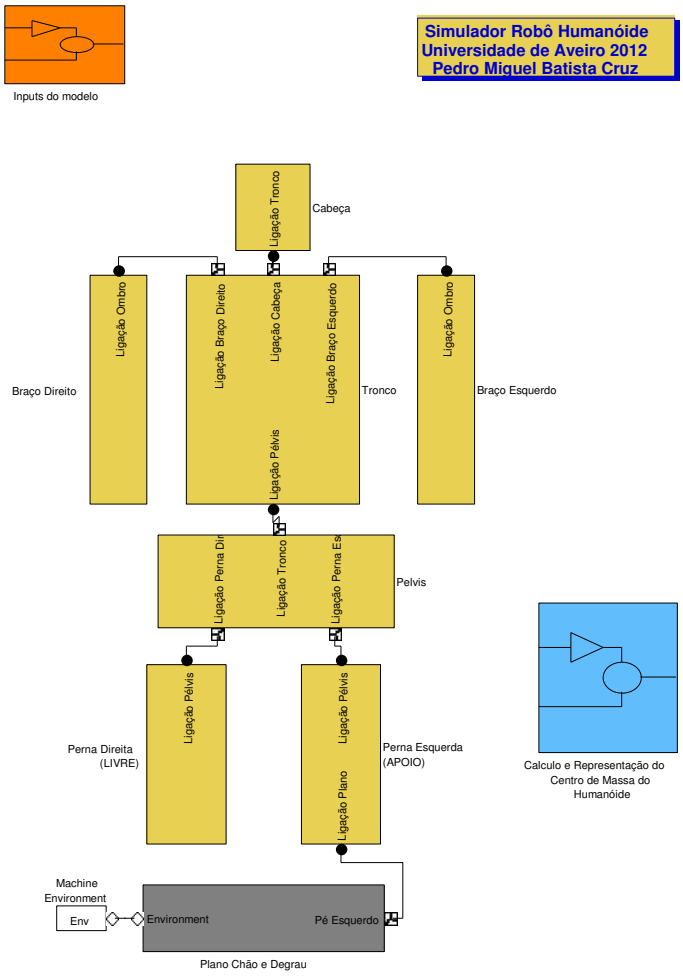
[50] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and N. Andrew, "ROS: an open-source robot operating system," in *2009 International Conference on Robotics and Automation (ICRA)*, 2009.

[51] J. Salisbury and M. Srinivasan, "Phantom-based haptic interaction with virtual objects," *IEEE Computer Graphics and Applications*, vol. 17, no. 5, pp. 6 – 10, 1997.

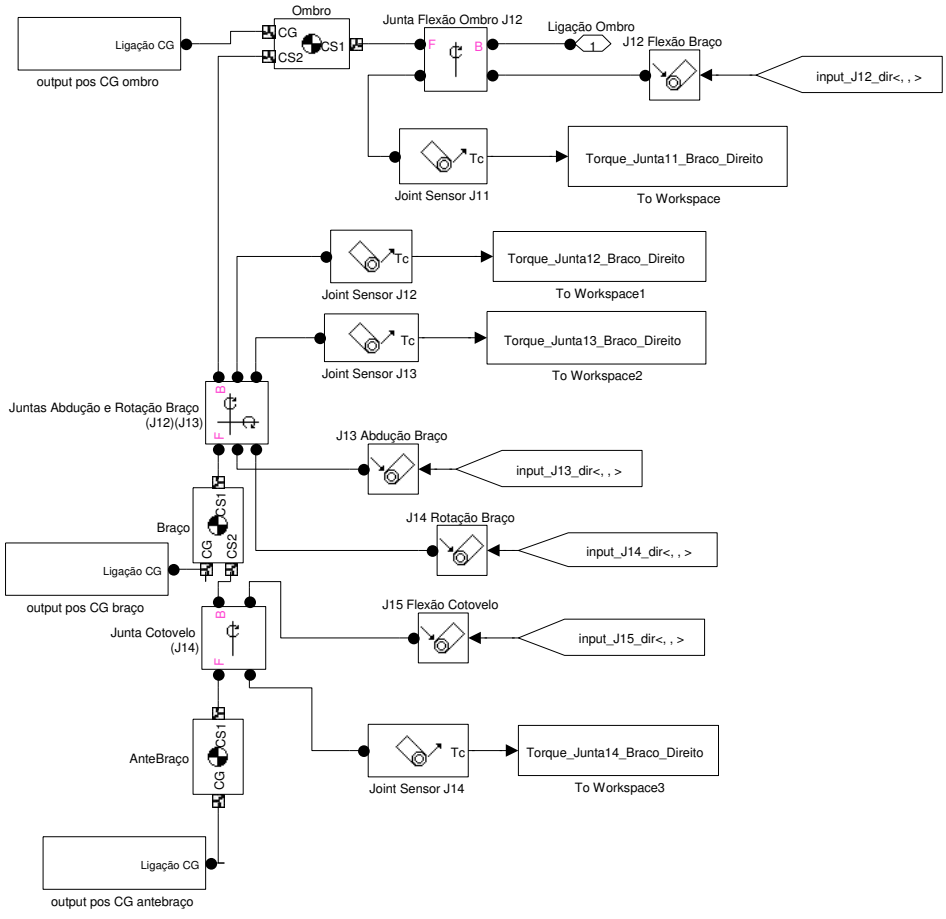
[52] C. Peña, R. Aracil, and R. Saltaren, "Teleoperation of a robot using a haptic device with different kinematics," in *Haptics: Perception, Devices and Scenarios* (M. Ferre, ed.), vol. 5024 of *Lecture Notes in Computer Science*, pp. 181–186, Springer Berlin / Heidelberg, 2008.

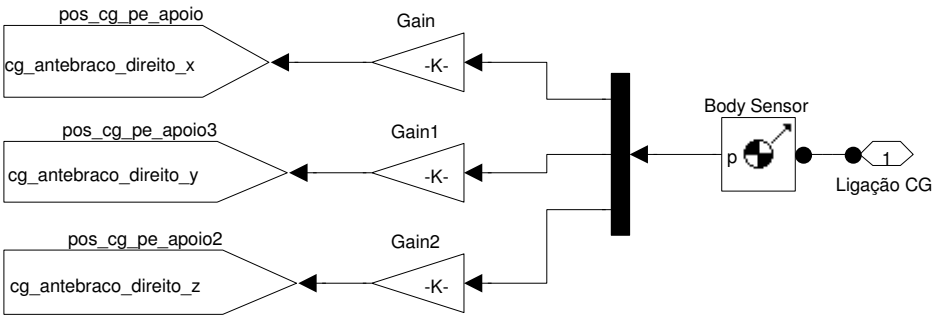
Appendix A

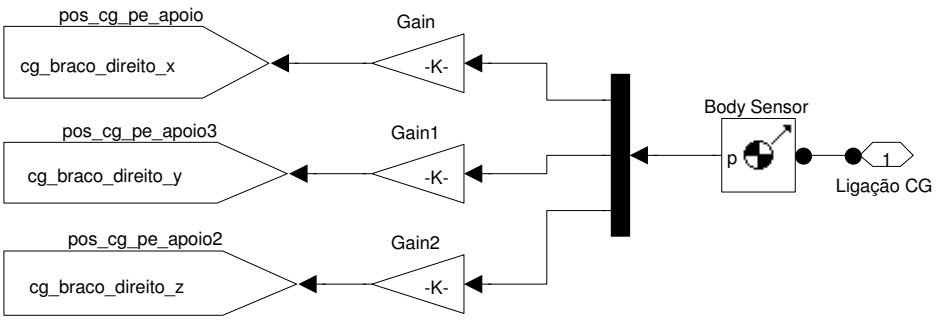
SimMechanics Model

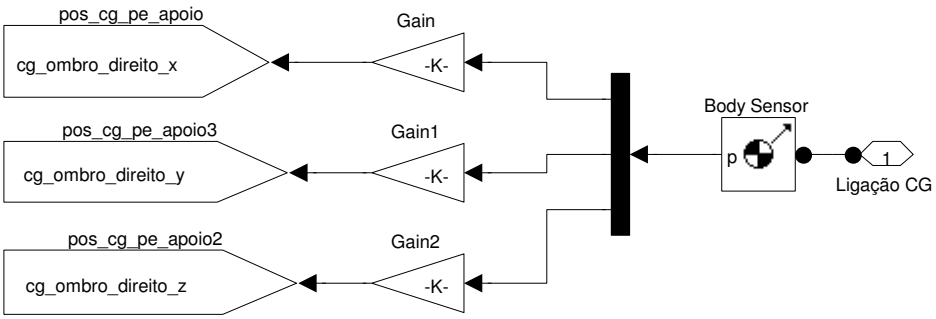


Braço Direito

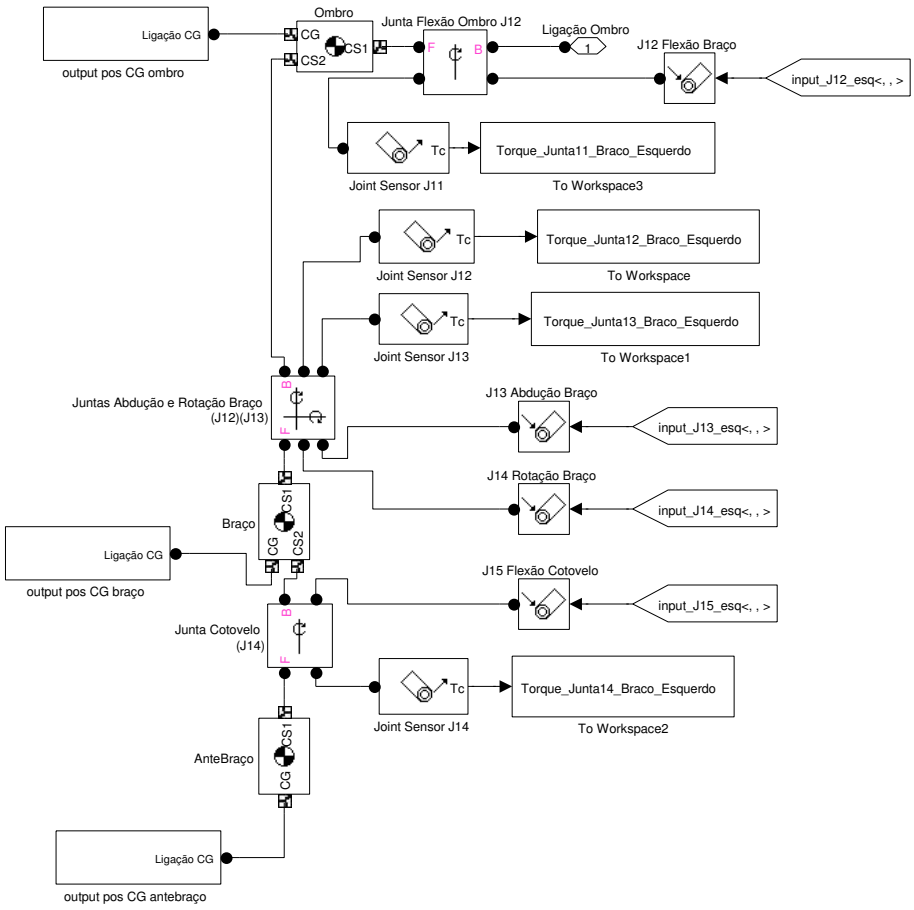


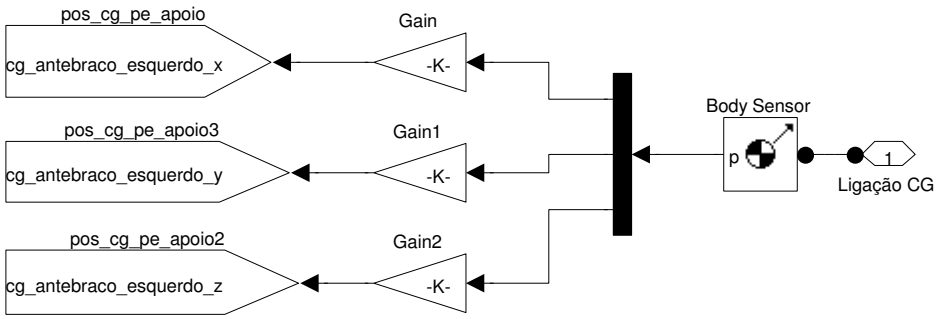

pos_cg_pe_apoio
cg_antebraco_direito_x
Gain
-K-
pos_cg_pe_apoio3
cg_antebraco_direito_y
Gain1
-K-
pos_cg_pe_apoio2
cg_antebraco_direito_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

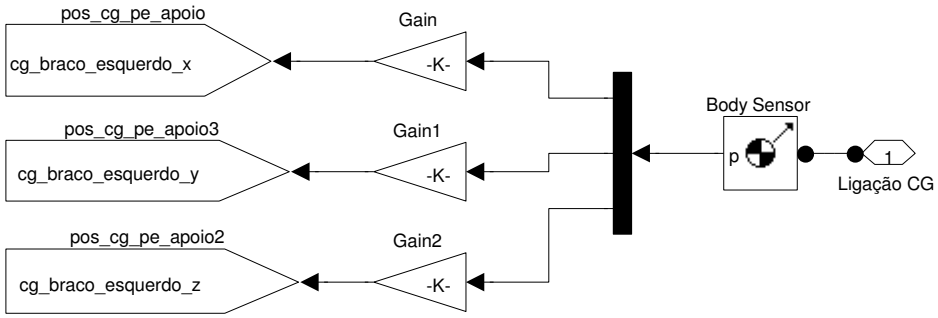

pos_cg_pe_apoio
cg_braco_direito_x
Gain
-K-
pos_cg_pe_apoio3
cg_braco_direito_y
Gain1
-K-
pos_cg_pe_apoio2
cg_braco_direito_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

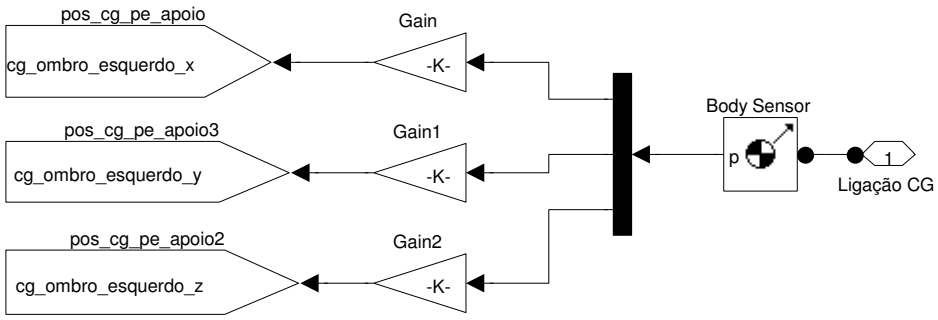


Braço Esquerdo

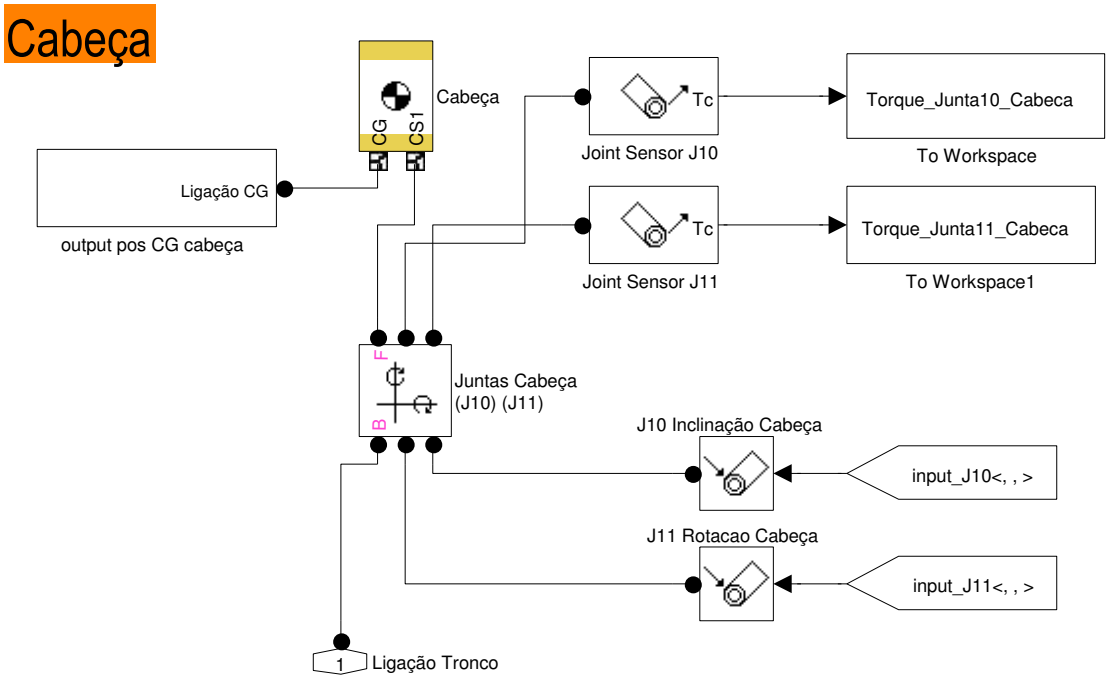


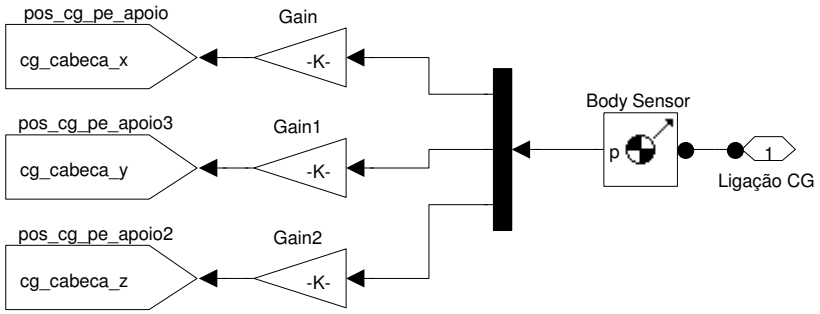

pos_cg_pe_apoio
cg_antebraco_esquerdo_x
Gain
-K-
pos_cg_pe_apoio3
cg_antebraco_esquerdo_y
Gain1
-K-
pos_cg_pe_apoio2
cg_antebraco_esquerdo_z
Gain2
-K-
Body Sensor
p
1
Ligação CG


pos_cg_pe_apoio
cg_braco_esquerdo_x
Gain
-K-
pos_cg_pe_apoio3
cg_braco_esquerdo_y
Gain1
-K-
pos_cg_pe_apoio2
cg_braco_esquerdo_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

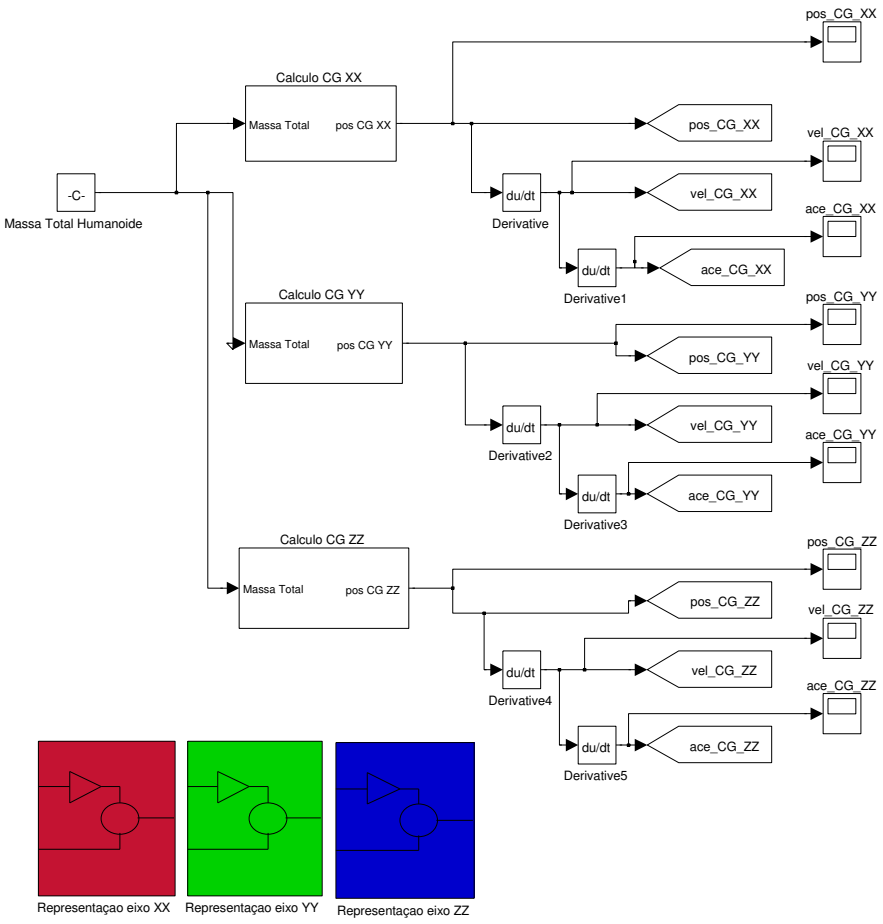

pos_cg_pe_apoio
cg_ombro_esquerdo_x
Gain
-K-
pos_cg_pe_apoio3
cg_ombro_esquerdo_y
Gain1
-K-
pos_cg_pe_apoio2
cg_ombro_esquerdo_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

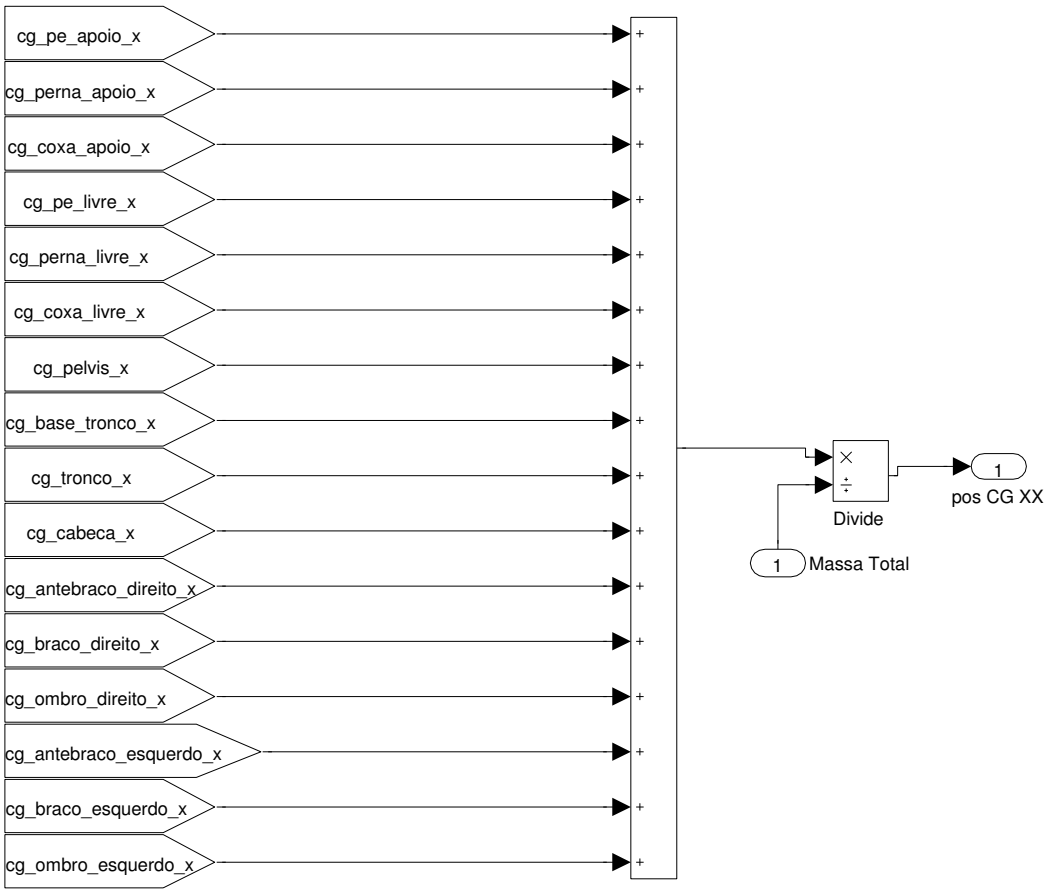
Cabeça

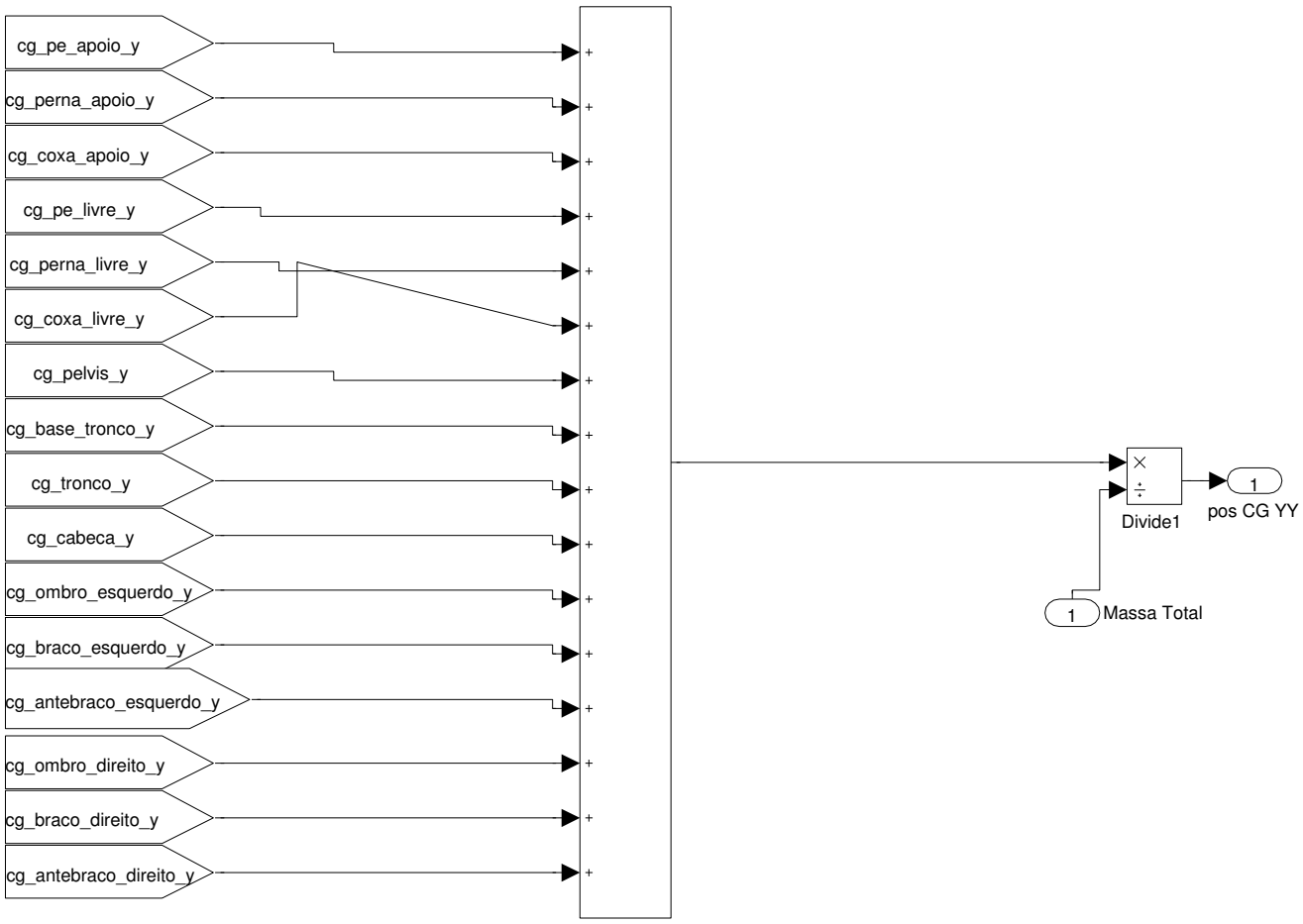

Cabeça
CG
CS1
Ligação CG
output pos CG cabeça
Tc
Joint Sensor J10
Torque_Junta10_Cabeca
To Workspace
Tc
Joint Sensor J11
Torque_Junta11_Cabeca
To Workspace1
F
B
Juntas Cabeça
(J10) (J11)
J10 Inclinação Cabeça
input_J10<, , >
J11 Rotacao Cabeça
input_J11<, , >
1
Ligação Tronco

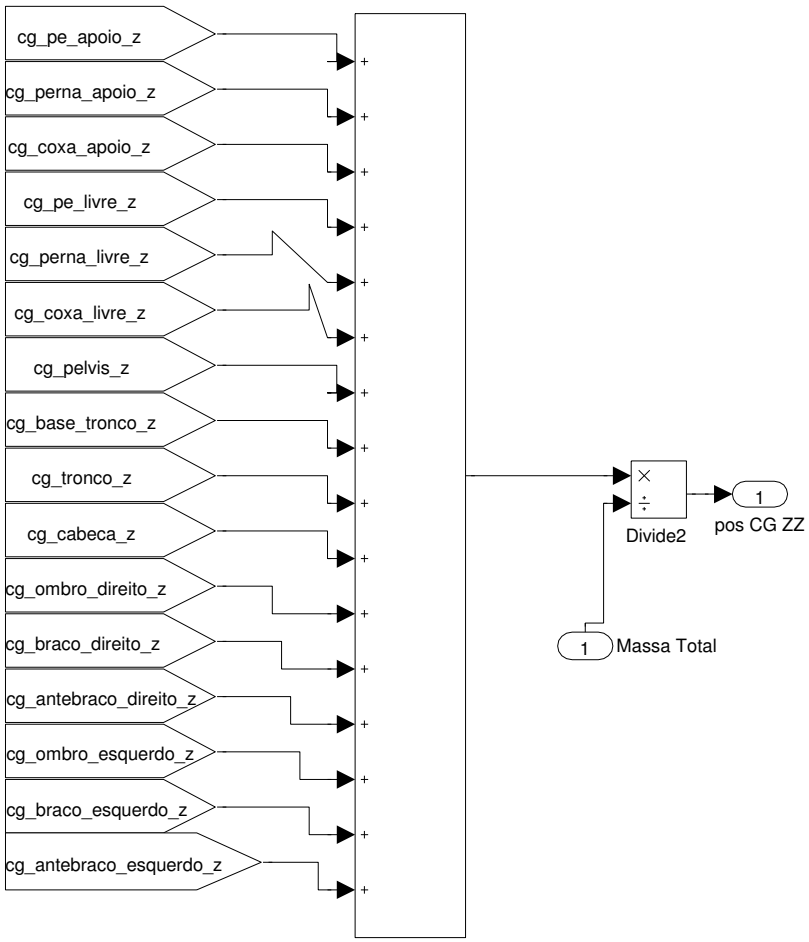


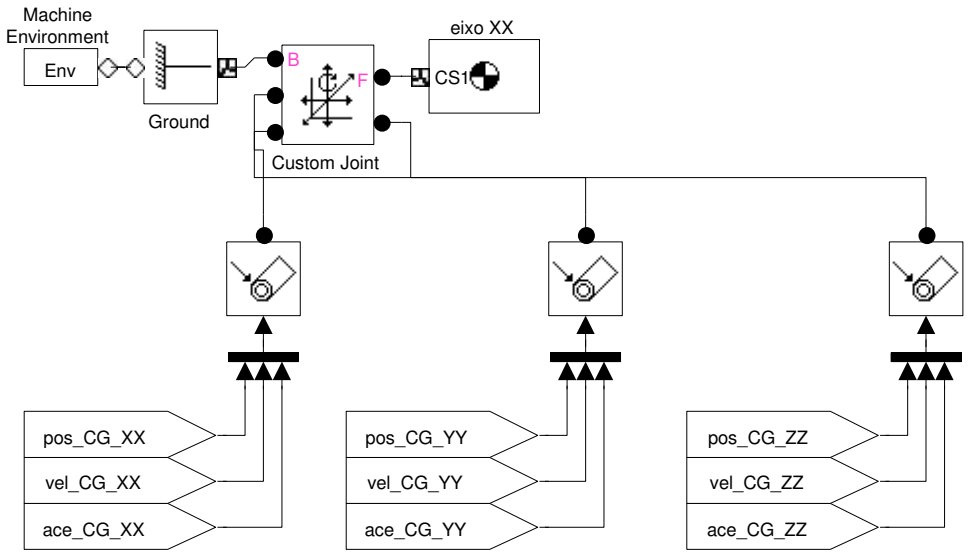
Cálculo do centro de gravidade (COG)

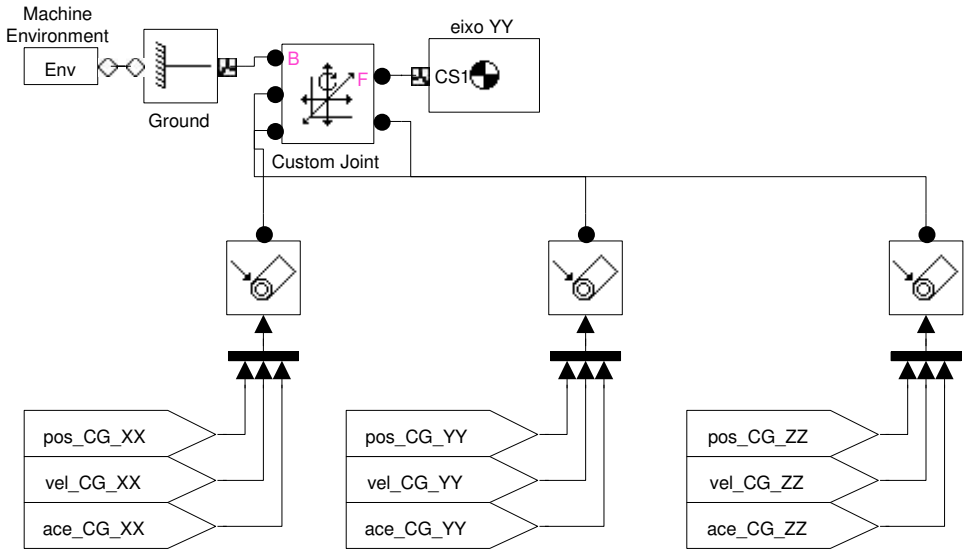


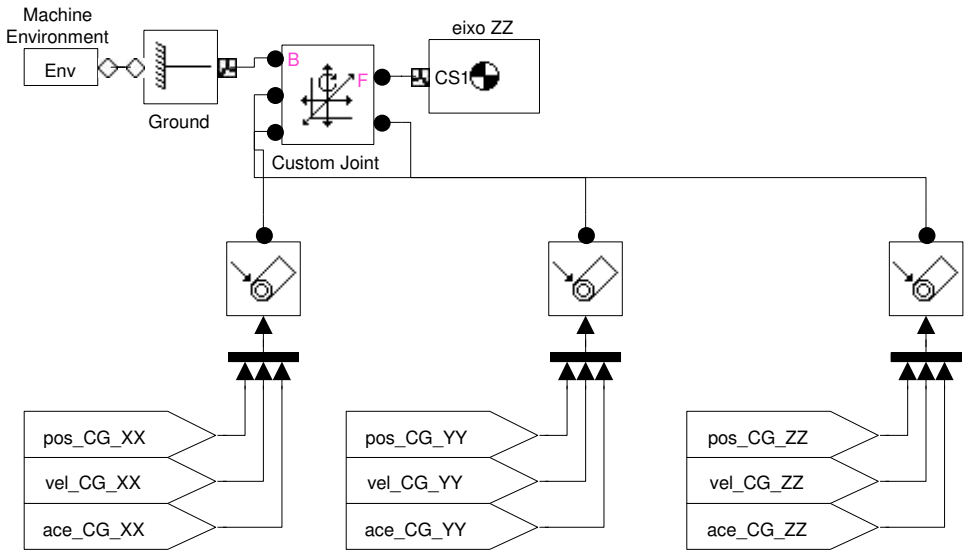

cg_pe_apoio_x
cg_perna_apoio_x
cg_coxa_apoio_x
cg_pe_livre_x
cg_perna_livre_x
cg_coxa_livre_x
cg_pelvis_x
cg_base_tronco_x
cg_tronco_x
cg_cabeca_x
cg_antebraco_direito_x
cg_braco_direito_x
cg_ombro_direito_x
cg_antebraco_esquerdo_x
cg_braco_esquerdo_x
cg_ombro_esquerdo_x
Divide
1
pos CG XX
1
Massa Total


cg_pe_apoio_y
cg_perna_apoio_y
cg_coxa_apoio_y
cg_pe_livre_y
cg_perna_livre_y
cg_coxa_livre_y
cg_pelvis_y
cg_base_tronco_y
cg_tronco_y
cg_cabeca_y
cg_ombro_esquerdo_y
cg_braco_esquerdo_y
cg_antebraco_esquerdo_y
cg_ombro_direito_y
cg_braco_direito_y
cg_antebraco_direito_y
Divide1
1
pos CG YY
1
Massa Total

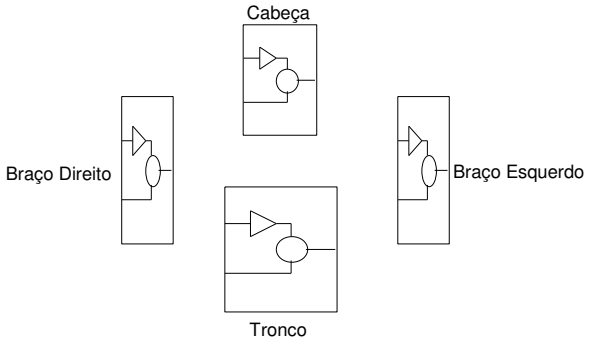

cg_pe_apoio_z
cg_perna_apoio_z
cg_coxa_apoio_z
cg_pe_livre_z
cg_perna_livre_z
cg_coxa_livre_z
cg_pelvis_z
cg_base_tronco_z
cg_tronco_z
cg_cabeca_z
cg_ombro_direito_z
cg_braco_direito_z
cg_antebraco_direito_z
cg_ombro_esquerdo_z
cg_braco_esquerdo_z
cg_antebraco_esquerdo_z
Divide2
1
pos CG ZZ
1
Massa Total

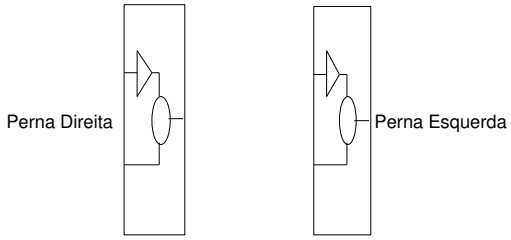

Machine
Environment
Env
Ground
B
F
eixo XX
CS1
Custom Joint
pos_CG_XX
vel_CG_XX
ace_CG_XX
pos_CG_YY
vel_CG_YY
ace_CG_YY
pos_CG_ZZ
vel_CG_ZZ
ace_CG_ZZ


Machine
Environment
Env
Ground
B
F
eixo YY
CS1
Custom Joint
pos_CG_XX
vel_CG_XX
ace_CG_XX
pos_CG_YY
vel_CG_YY
ace_CG_YY
pos_CG_ZZ
vel_CG_ZZ
ace_CG_ZZ

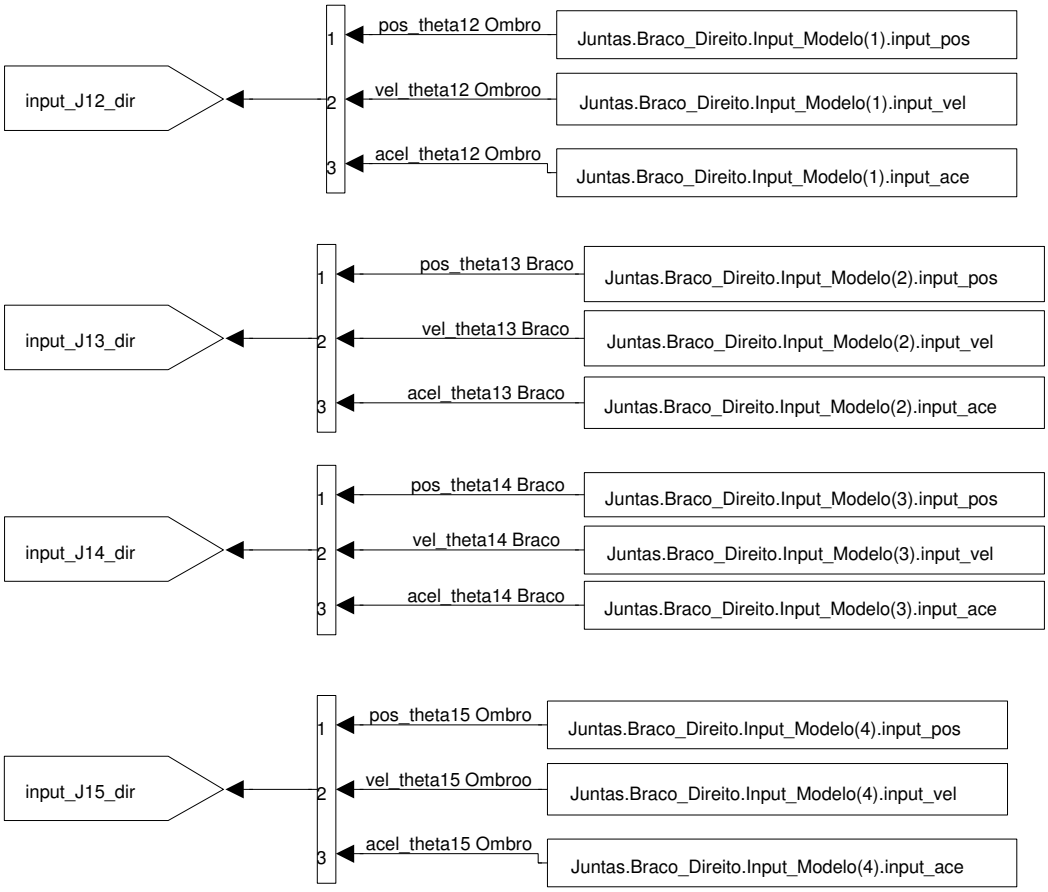

Machine
Environment
Env
Ground
B
F
eixo ZZ
CS1
Custom Joint
pos_CG_XX
vel_CG_XX
ace_CG_XX
pos_CG_YY
vel_CG_YY
ace_CG_YY
pos_CG_ZZ
vel_CG_ZZ
ace_CG_ZZ

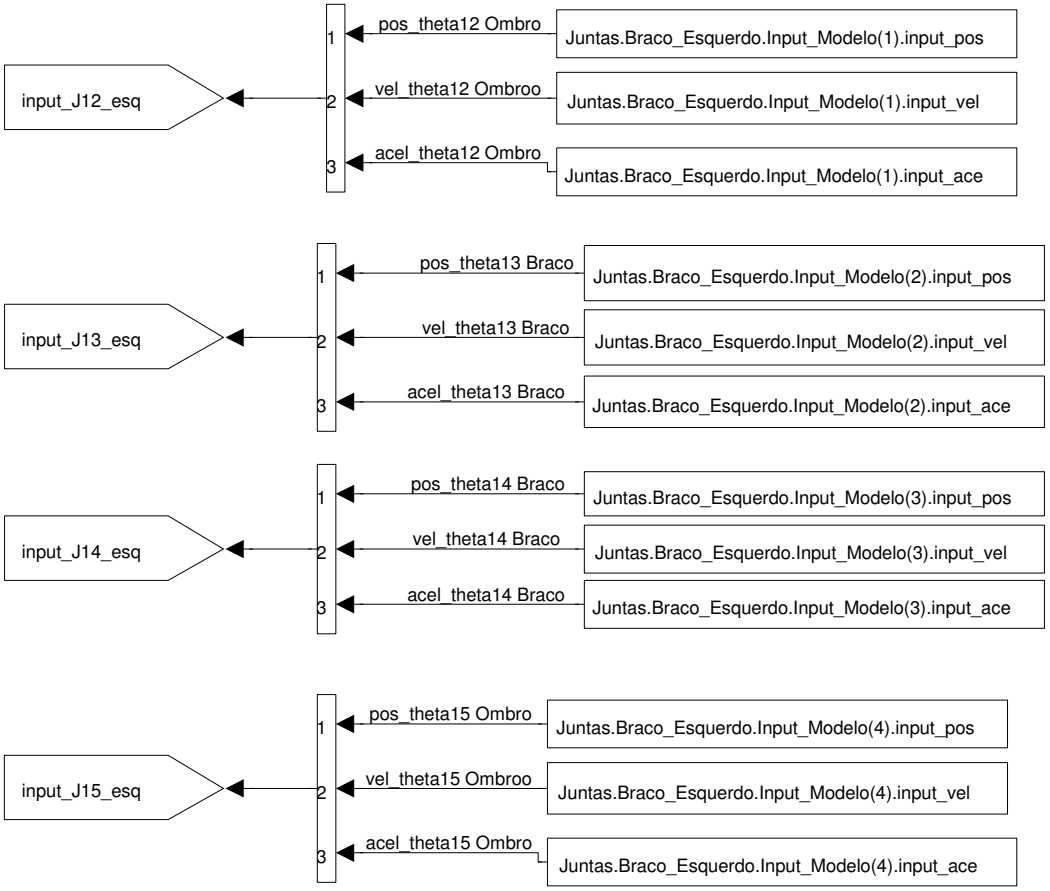
Dados de entrada do modelo

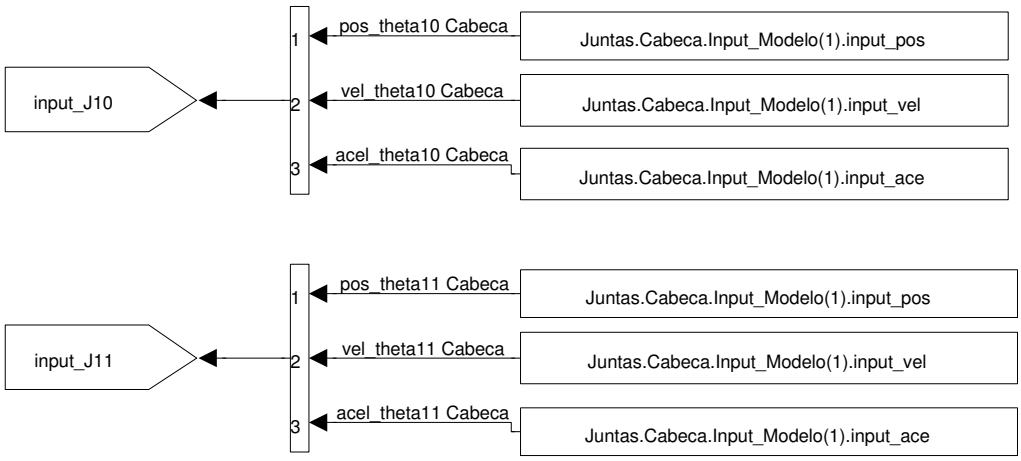


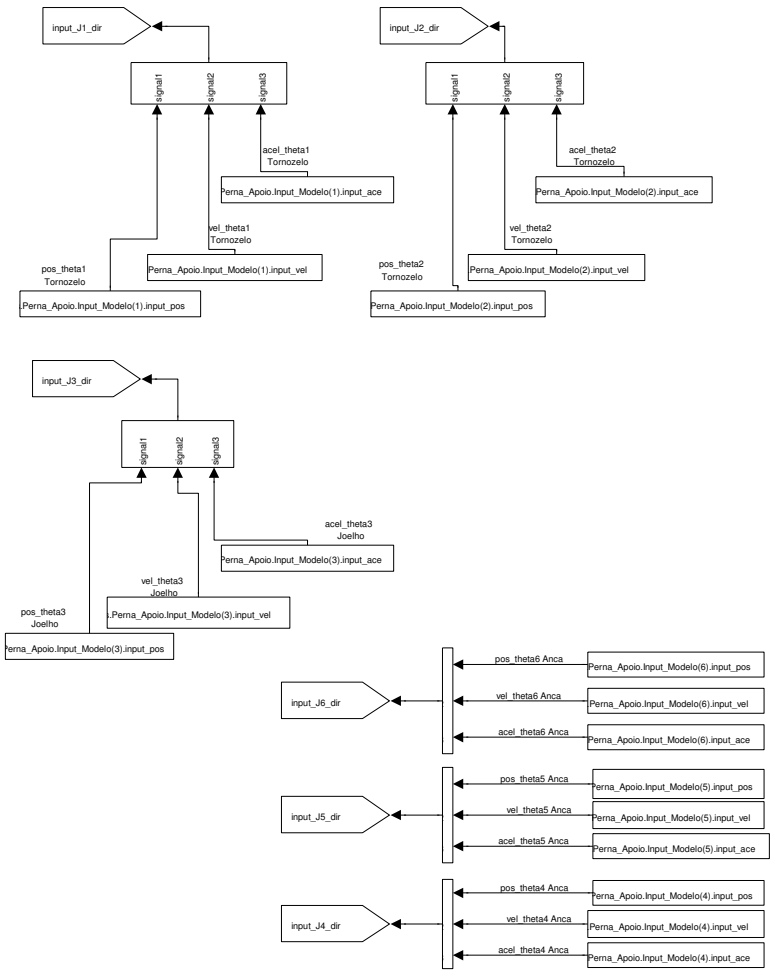


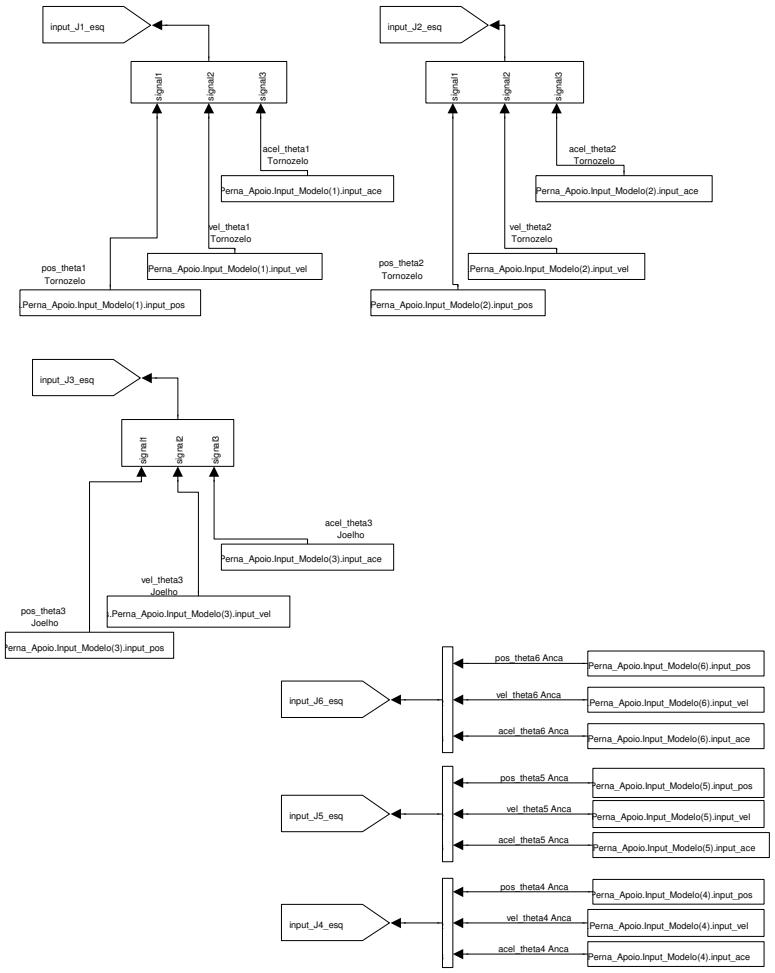
Para mudar as variáveis que o modelo procura no workspace, entrar em cada uma das máscaras e alterar o parâmetro devido nos blocos "From Workspace".

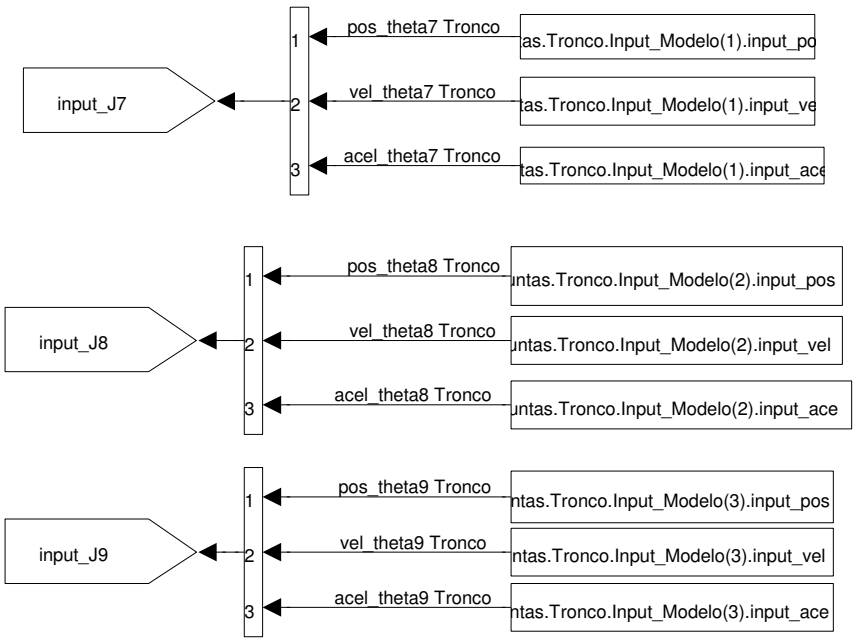



input_J12_esq
1
2
3
pos_theta12 Ombro
vel_theta12 Ombroo
acel_theta12 Ombro
Juntas.Braco_Esquerdo.Input_Modelo(1).input_pos
Juntas.Braco_Esquerdo.Input_Modelo(1).input_vel
Juntas.Braco_Esquerdo.Input_Modelo(1).input_ace
input_J13_esq
1
2
3
pos_theta13 Braco
vel_theta13 Braco
acel_theta13 Braco
Juntas.Braco_Esquerdo.Input_Modelo(2).input_pos
Juntas.Braco_Esquerdo.Input_Modelo(2).input_vel
Juntas.Braco_Esquerdo.Input_Modelo(2).input_ace
input_J14_esq
1
2
3
pos_theta14 Braco
vel_theta14 Braco
acel_theta14 Braco
Juntas.Braco_Esquerdo.Input_Modelo(3).input_pos
Juntas.Braco_Esquerdo.Input_Modelo(3).input_vel
Juntas.Braco_Esquerdo.Input_Modelo(3).input_ace
input_J15_esq
1
2
3
pos_theta15 Ombro
vel_theta15 Ombroo
acel_theta15 Ombro
Juntas.Braco_Esquerdo.Input_Modelo(4).input_pos
Juntas.Braco_Esquerdo.Input_Modelo(4).input_vel
Juntas.Braco_Esquerdo.Input_Modelo(4).input_ace

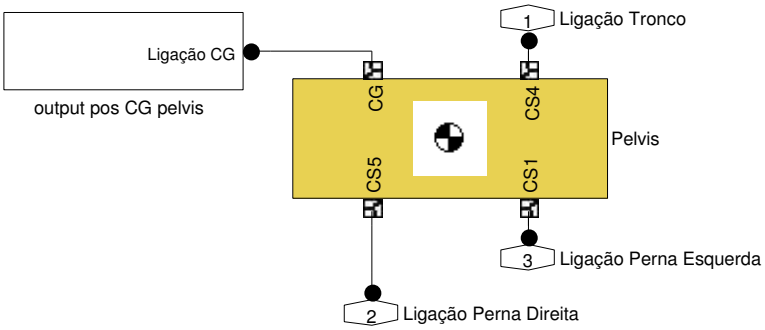

input_J10
1
2
3
pos_theta10 Cabeca
vel_theta10 Cabeca
acel_theta10 Cabeca
Juntas.Cabeca.Input_Modelo(1).input_pos
Juntas.Cabeca.Input_Modelo(1).input_vel
Juntas.Cabeca.Input_Modelo(1).input_ace
input_J11
1
2
3
pos_theta11 Cabeca
vel_theta11 Cabeca
acel_theta11 Cabeca
Juntas.Cabeca.Input_Modelo(1).input_pos
Juntas.Cabeca.Input_Modelo(1).input_vel
Juntas.Cabeca.Input_Modelo(1).input_ace

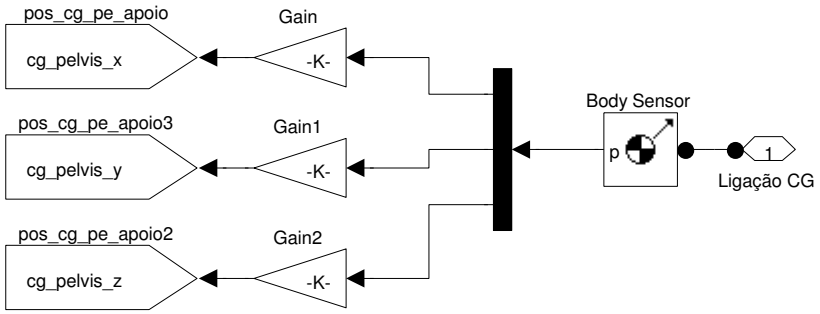

input_J1_dir
signal1
signal2
signal3
acel_theta1
Tornozelo
Perna_Apoio.Input_Modelo(1).input_ace
vel_theta1
Tornozelo
Perna_Apoio.Input_Modelo(1).input_vel
pos_theta1
Tornozelo
Perna_Apoio.Input_Modelo(1).input_pos
input_J2_dir
signal1
signal2
signal3
acel_theta2
Tornozelo
Perna_Apoio.Input_Modelo(2).input_ace
vel_theta2
Tornozelo
Perna_Apoio.Input_Modelo(2).input_vel
pos_theta2
Tornozelo
Perna_Apoio.Input_Modelo(2).input_pos
input_J3_dir
signal1
signal2
signal3
acel_theta3
Joelho
Perna_Apoio.Input_Modelo(3).input_ace
vel_theta3
Joelho
Perna_Apoio.Input_Modelo(3).input_vel
pos_theta3
Joelho
Perna_Apoio.Input_Modelo(3).input_pos
input_J6_dir
pos_theta6 Anca
Perna_Apoio.Input_Modelo(6).input_pos
vel_theta6 Anca
Perna_Apoio.Input_Modelo(6).input_vel
acel_theta6 Anca
Perna_Apoio.Input_Modelo(6).input_ace
input_J5_dir
pos_theta5 Anca
Perna_Apoio.Input_Modelo(5).input_pos
vel_theta5 Anca
Perna_Apoio.Input_Modelo(5).input_vel
acel_theta5 Anca
Perna_Apoio.Input_Modelo(5).input_ace
input_J4_dir
pos_theta4 Anca
Perna_Apoio.Input_Modelo(4).input_pos
vel_theta4 Anca
Perna_Apoio.Input_Modelo(4).input_vel
acel_theta4 Anca
Perna_Apoio.Input_Modelo(4).input_ace

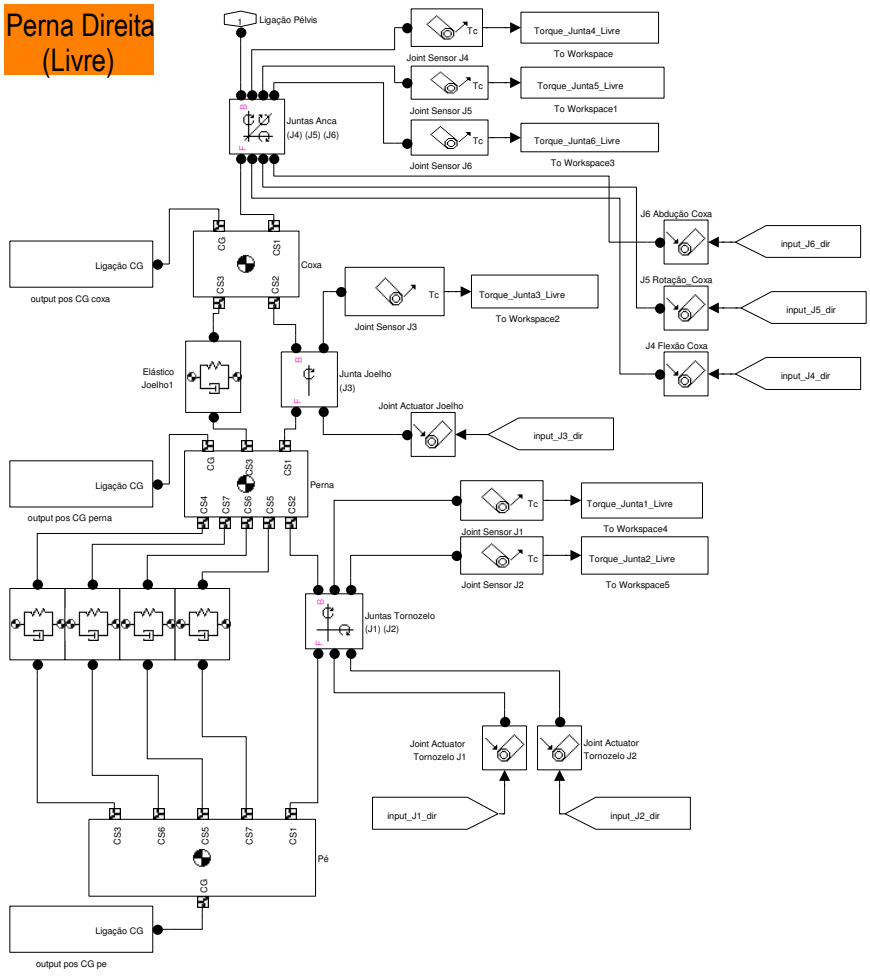

input_J1_esq
signal1
signal2
signal3
acel_theta1
Tornozelo
Perna_Apoio.Input_Modelo(1).input_ace
vel_theta1
Tornozelo
Perna_Apoio.Input_Modelo(1).input_vel
pos_theta1
Tornozelo
Perna_Apoio.Input_Modelo(1).input_pos
input_J2_esq
signal1
signal2
signal3
acel_theta2
Tornozelo
Perna_Apoio.Input_Modelo(2).input_ace
vel_theta2
Tornozelo
Perna_Apoio.Input_Modelo(2).input_vel
pos_theta2
Tornozelo
Perna_Apoio.Input_Modelo(2).input_pos
input_J3_esq
signal1
signal2
signal3
acel_theta3
Joelho
Perna_Apoio.Input_Modelo(3).input_ace
vel_theta3
Joelho
Perna_Apoio.Input_Modelo(3).input_vel
pos_theta3
Joelho
Perna_Apoio.Input_Modelo(3).input_pos
input_J6_esq
pos_theta6 Anca
Perna_Apoio.Input_Modelo(6).input_pos
vel_theta6 Anca
Perna_Apoio.Input_Modelo(6).input_vel
acel_theta6 Anca
Perna_Apoio.Input_Modelo(6).input_ace
input_J5_esq
pos_theta5 Anca
Perna_Apoio.Input_Modelo(5).input_pos
vel_theta5 Anca
Perna_Apoio.Input_Modelo(5).input_vel
acel_theta5 Anca
Perna_Apoio.Input_Modelo(5).input_ace
input_J4_esq
pos_theta4 Anca
Perna_Apoio.Input_Modelo(4).input_pos
vel_theta4 Anca
Perna_Apoio.Input_Modelo(4).input_vel
acel_theta4 Anca
Perna_Apoio.Input_Modelo(4).input_ace

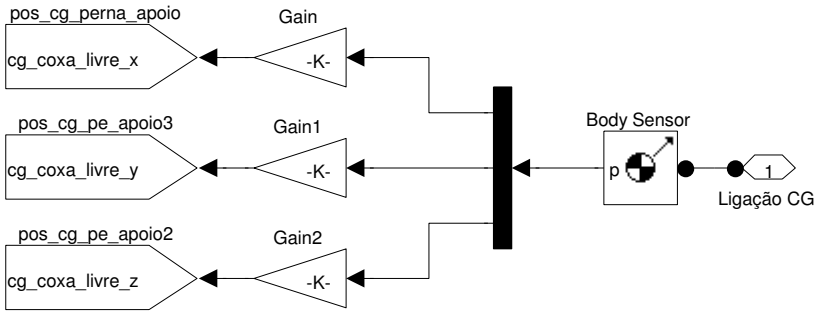

input_J7
1
2
3
pos_theta7 Tronco
vel_theta7 Tronco
acel_theta7 Tronco
as.Tronco.Input_Modelo(1).input_po
tas.Tronco.Input_Modelo(1).input_ve
tas.Tronco.Input_Modelo(1).input_ace
input_J8
pos_theta8 Tronco
vel_theta8 Tronco
acel_theta8 Tronco
untas.Tronco.Input_Modelo(2).input_pos
untas.Tronco.Input_Modelo(2).input_vel
untas.Tronco.Input_Modelo(2).input_ace
input_J9
pos_theta9 Tronco
vel_theta9 Tronco
acel_theta9 Tronco
ntas.Tronco.Input_Modelo(3).input_pos
ntas.Tronco.Input_Modelo(3).input_vel
ntas.Tronco.Input_Modelo(3).input_ace

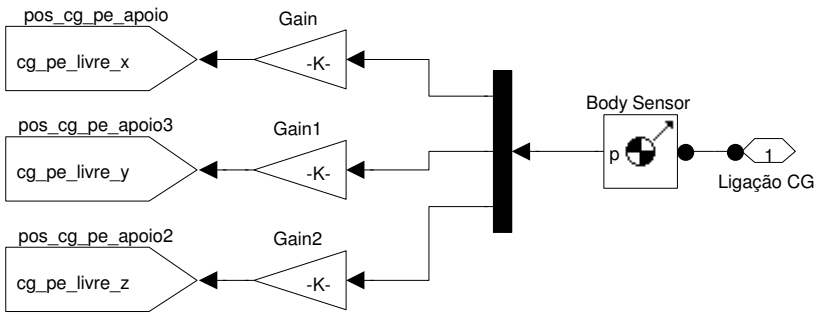
Pélvis

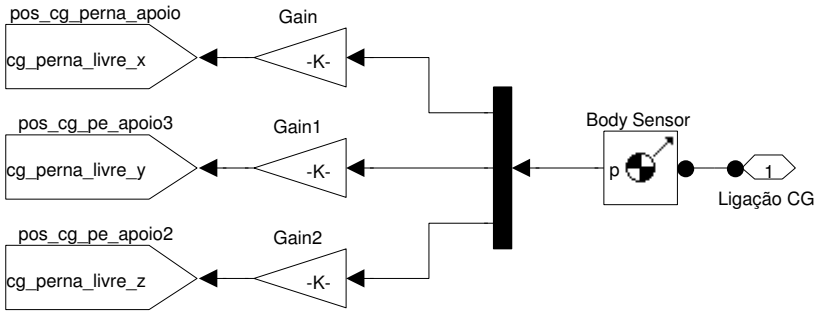

Ligação CG
output pos CG pelvis
CG
CS4
CS5
CS1
Pelvis
1
Ligação Tronco
3
Ligação Perna Esquerda
2
Ligação Perna Direita

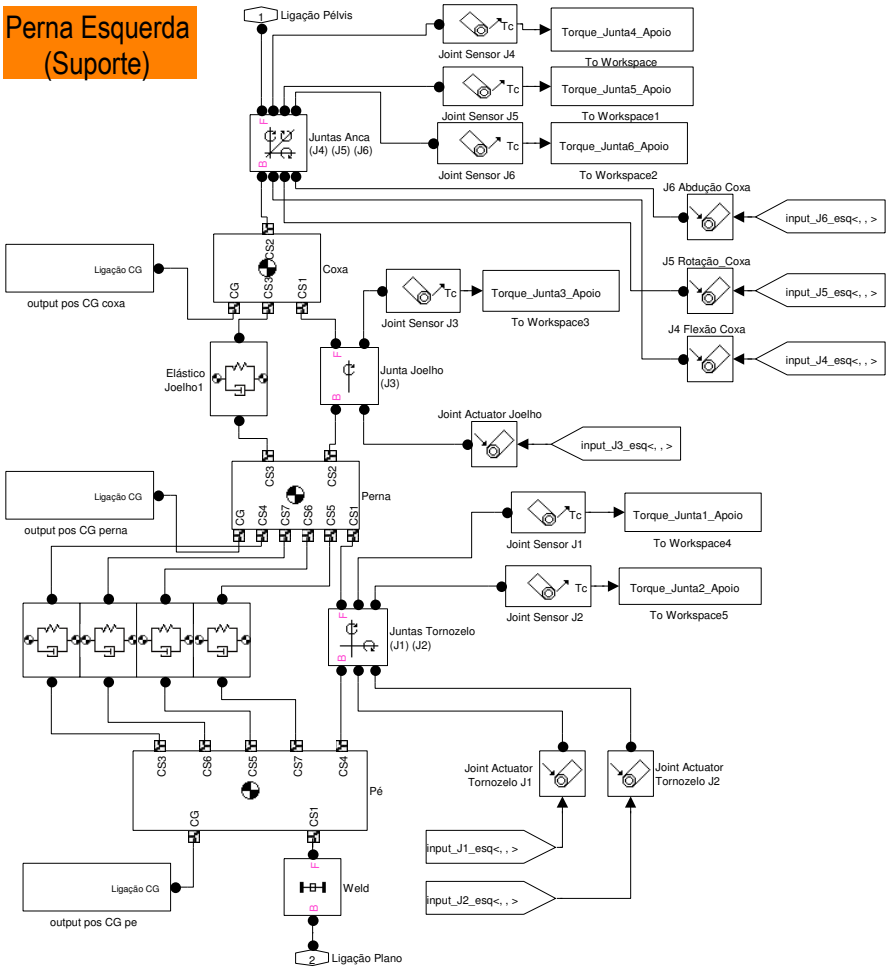

pos_cg_pe_apoio
cg_pelvis_x
Gain
-K-
pos_cg_pe_apoio3
cg_pelvis_y
Gain1
-K-
pos_cg_pe_apoio2
cg_pelvis_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

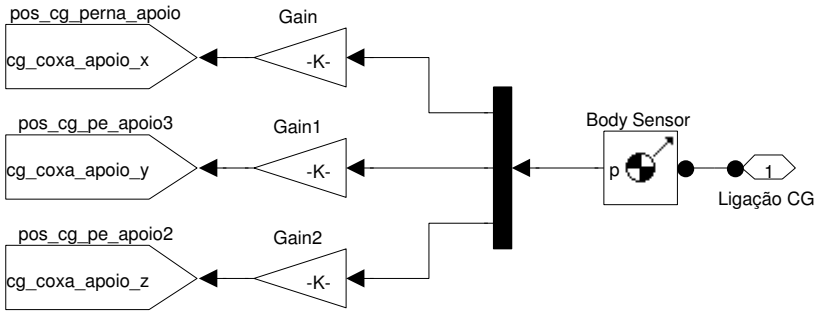

Perna Direita (Livre)
Ligação Pélvis
Juntas Anca (J4) (J5) (J6)
Joint Sensor J4
Joint Sensor J5
Joint Sensor J6
Torque_Junta4_Livre
Torque_Junta5_Livre
Torque_Junta6_Livre
To Workspace
To Workspace1
To Workspace3
J6 Abdução Coxa
J5 Rotação_Coxa
J4 Flexão Coxa
input_J6_dir
input_J5_dir
input_J4_dir
Ligação CG
output pos CG coxa
Coxa
Joint Sensor J3
Torque_Junta3_Livre
To Workspace2
Elástico Joelho1
Junta Joelho (J3)
Joint Actuator Joelho
input_J3_dir
Ligação CG
output pos CG perna
Perna
Joint Sensor J1
Joint Sensor J2
Torque_Junta1_Livre
Torque_Junta2_Livre
To Workspace4
To Workspace5
Juntas Tornozelo (J1) (J2)
Joint Actuator Tornozelo J1
Joint Actuator Tornozelo J2
input_J1_dir
input_J2_dir
Pé
Ligação CG
output pos CG pe

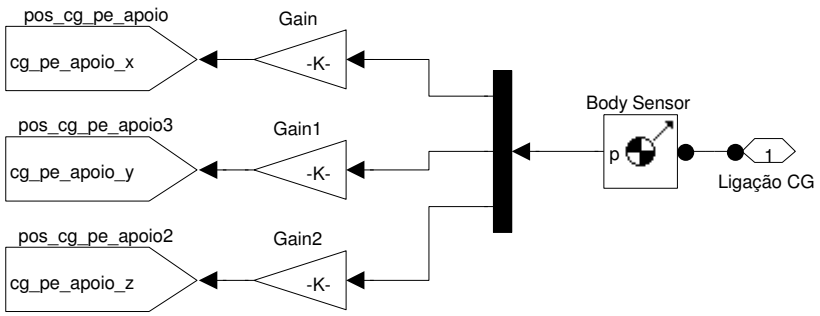

pos_cg_perna_apoio
cg_coxa_livre_x
Gain
-K-
pos_cg_pe_apoio3
cg_coxa_livre_y
Gain1
-K-
pos_cg_pe_apoio2
cg_coxa_livre_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

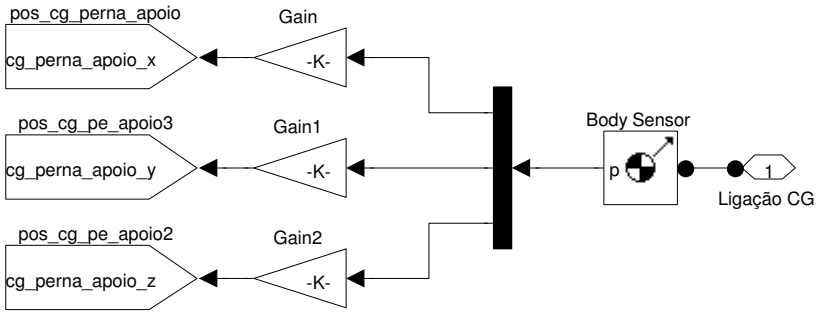

pos_cg_pe_apoio
cg_pe_livre_x
Gain
-K-
pos_cg_pe_apoio3
cg_pe_livre_y
Gain1
-K-
pos_cg_pe_apoio2
cg_pe_livre_z
Gain2
-K-
Body Sensor
p
1
Ligação CG


pos_cg_perna_apoio
cg_perna_livre_x
Gain
-K-
pos_cg_pe_apoio3
cg_perna_livre_y
Gain1
-K-
pos_cg_pe_apoio2
cg_perna_livre_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

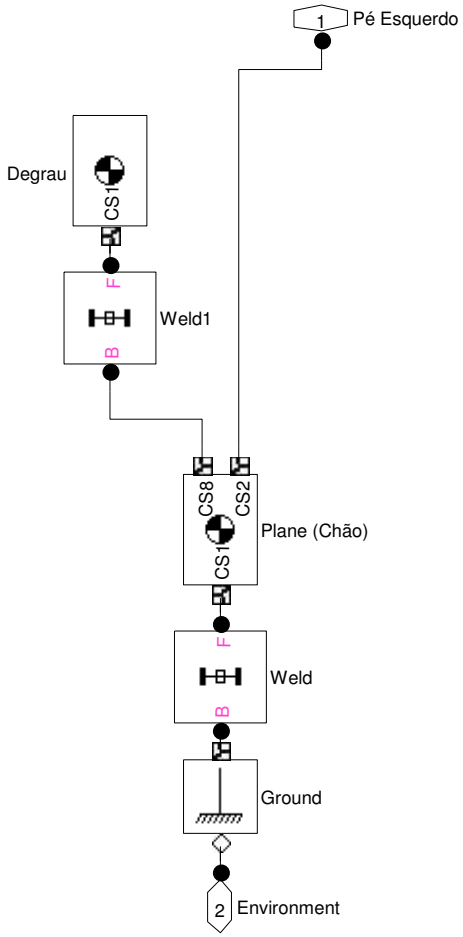

Perna Esquerda (Suporte)
1
Ligação Pélvis
Joint Sensor J4
Torque_Junta4_Apoio
To Workspace
Joint Sensor J5
Torque_Junta5_Apoio
To Workspace1
Juntas Anca (J4) (J5) (J6)
Joint Sensor J6
Torque_Junta6_Apoio
To Workspace2
J6 Abdução Coxa
input_J6_esq<, , >
J5 Rotação_Coxa
input_J5_esq<, , >
J4 Flexão Coxa
input_J4_esq<, , >
Ligação CG
output pos CG coxa
Coxa
Joint Sensor J3
Torque_Junta3_Apoio
To Workspace3
Elástico Joelho1
Junta Joelho (J3)
Joint Actuator Joelho
input_J3_esq<, , >
Ligação CG
output pos CG perna
Perna
Joint Sensor J1
Torque_Junta1_Apoio
To Workspace4
Joint Sensor J2
Torque_Junta2_Apoio
To Workspace5
Juntas Tornozelo (J1) (J2)
Joint Actuator Tornozelo J1
Joint Actuator Tornozelo J2
Pé
input_J1_esq<, , >
input_J2_esq<, , >
Ligação CG
output pos CG pe
Weld
2
Ligação Plano


pos_cg_perna_apoio
cg_coxa_apoio_x
Gain
-K-
pos_cg_pe_apoio3
cg_coxa_apoio_y
Gain1
-K-
pos_cg_pe_apoio2
cg_coxa_apoio_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

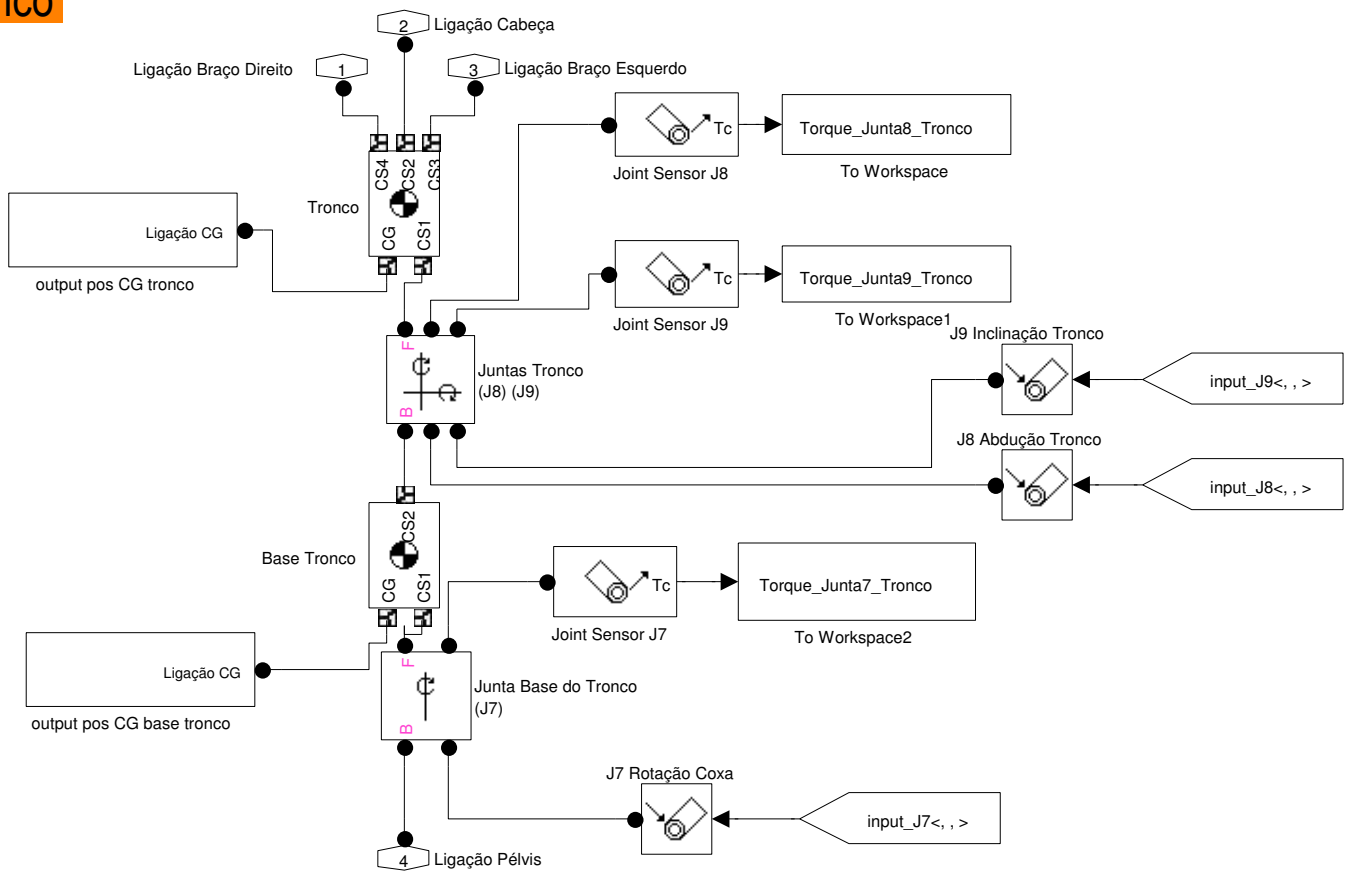

pos_cg_pe_apoio
cg_pe_apoio_x
Gain
-K-
pos_cg_pe_apoio3
cg_pe_apoio_y
Gain1
-K-
pos_cg_pe_apoio2
cg_pe_apoio_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

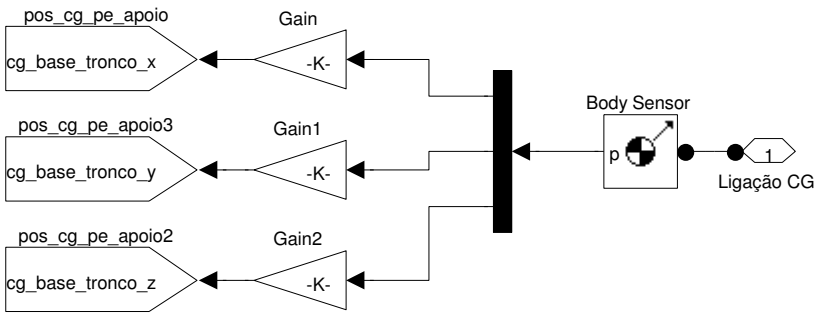


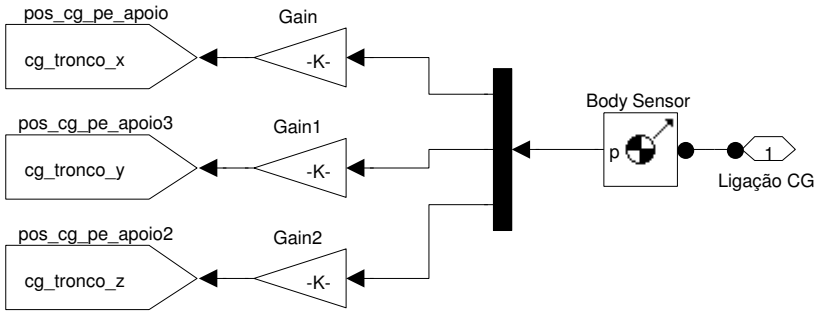
Plano do chão



Tronco

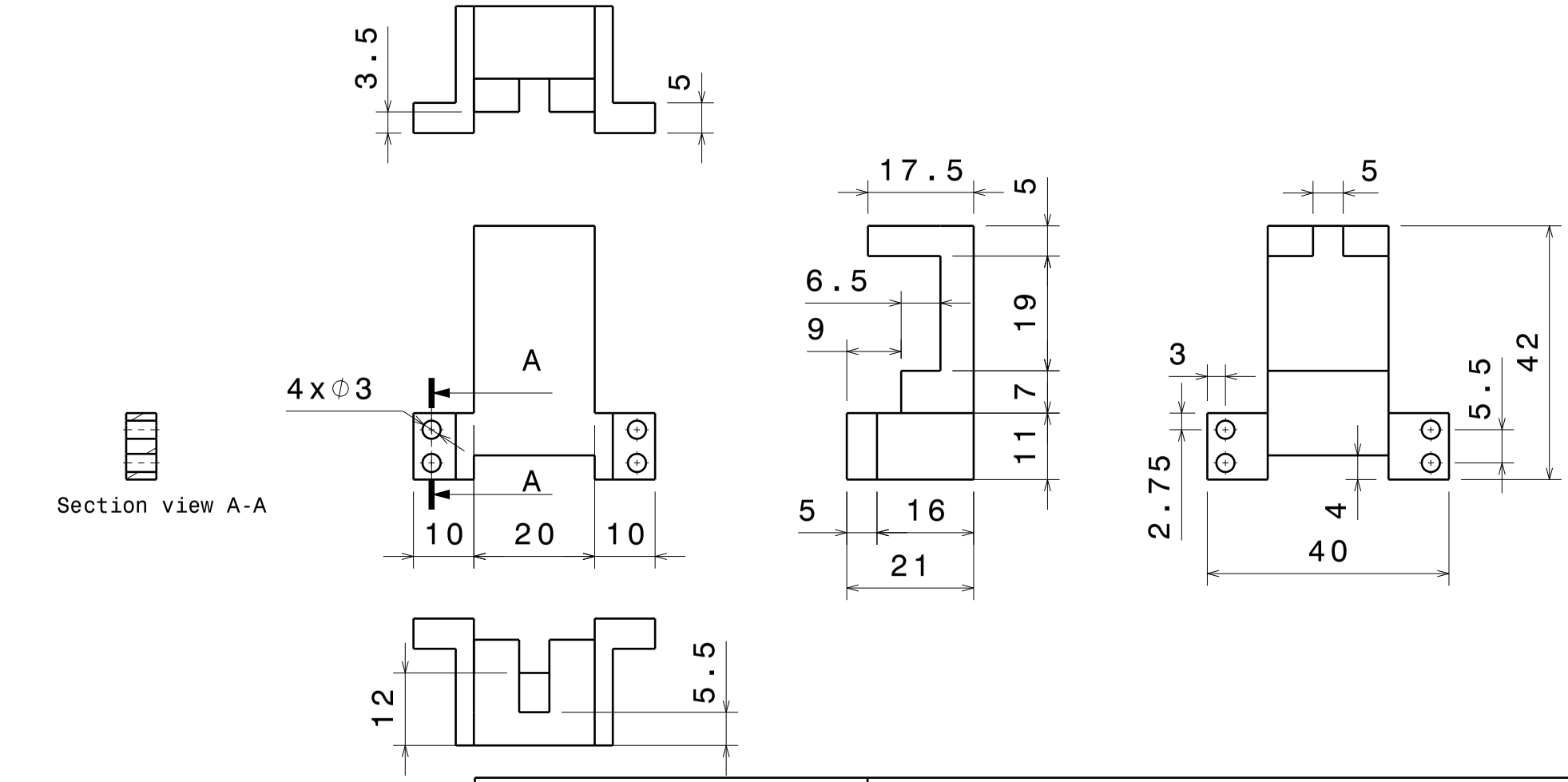

2
Ligação Cabeça
Ligação Braço Direito
1
3
Ligação Braço Esquerdo
CS4
CS2
CS3
Tronco
CG
CS1
Ligação CG
output pos CG tronco
Tc
Torque_Junta8_Tronco
Joint Sensor J8
To Workspace
Tc
Torque_Junta9_Tronco
Joint Sensor J9
To Workspace1
F
B
Juntas Tronco
(J8) (J9)
J9 Inclinação Tronco
input_J9<, , >
J8 Abdução Tronco
input_J8<, , >
CS2
Base Tronco
CG
CS1
Tc
Torque_Junta7_Tronco
Joint Sensor J7
To Workspace2
Ligação CG
output pos CG base tronco
F
B
Junta Base do Tronco
(J7)
J7 Rotação Coxa
input_J7<, , >
4
Ligação Pélvis

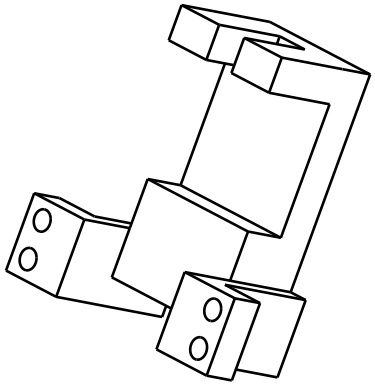

pos_cg_pe_apoio
cg_base_tronco_x
Gain
-K-
pos_cg_pe_apoio3
cg_base_tronco_y
Gain1
-K-
pos_cg_pe_apoio2
cg_base_tronco_z
Gain2
-K-
Body Sensor
p
1
Ligação CG


pos_cg_pe_apoio
cg_tronco_x
Gain
-K-
pos_cg_pe_apoio3
cg_tronco_y
Gain1
-K-
pos_cg_pe_apoio2
cg_tronco_z
Gain2
-K-
Body Sensor
p
1
Ligação CG

Appendix B

Mechanical Drawings



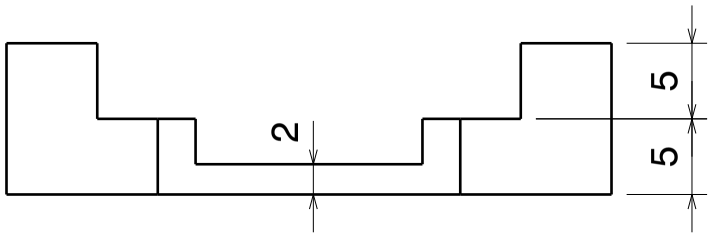


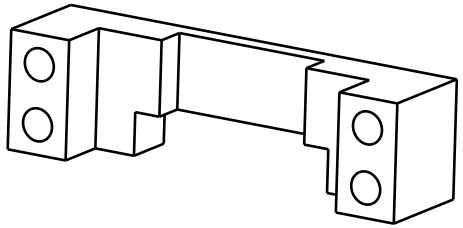
Material: Alumínio.

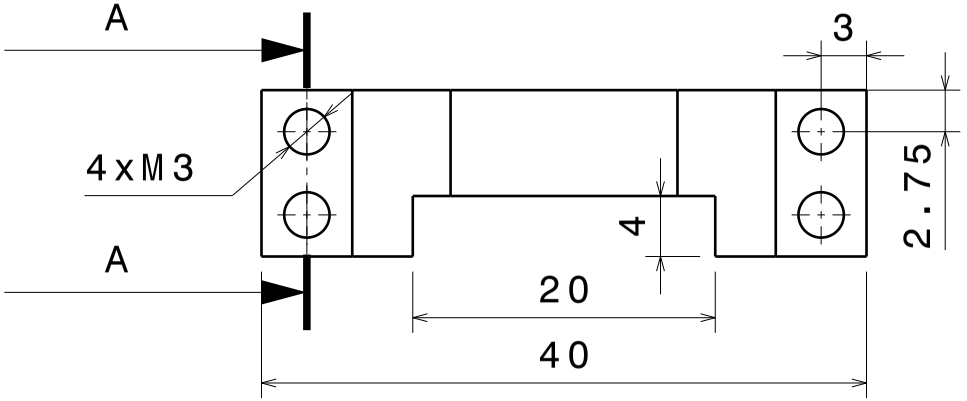
Quantidade: 2 unidades

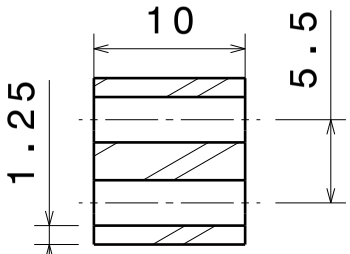
Projecto Humanoide
Universidade de Aveiro

DRAWN BY: Pedro Cruz	DATE: 17-04-2012	DRAWING TITLE		
CHECKED BY: XXX	DATE: XXX	SIZE: A4	DRAWING NUMBER: fixador_rotacao_braco_esquerdo_1	REV: X
DESIGNED BY: XXX	DATE: XXX	SCALE: 1:1	WEIGHT(kg): 0,02	SHEET: 1/1

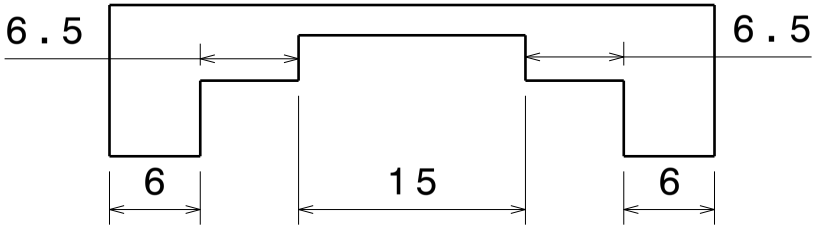








Section view A-A

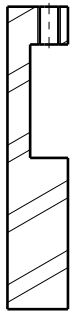


Material: Alumínio.

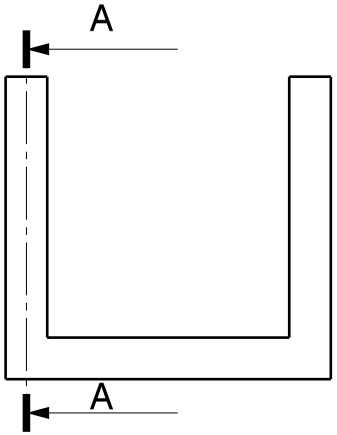
Quantidade: 2 Unidades

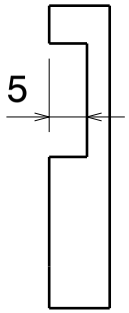
Projecto Humanoide
Universidade de Aveiro

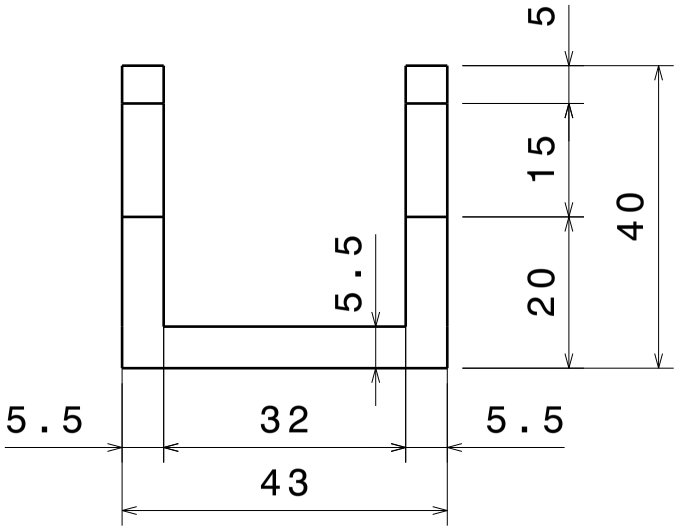
DRAWN BY: Pedro Cruz	DATE: 17-04-2012	DRAWING TITLE		
CHECKED BY: XXX	DATE: XXX	SIZE: A4	DRAWING NUMBER: fixador_rotacao_braco_esquerdo_2	REV: X
DESIGNED BY: XXX	DATE: XXX	SCALE: 2:1	WEIGHT(kg): 0,01	SHEET: 1/1

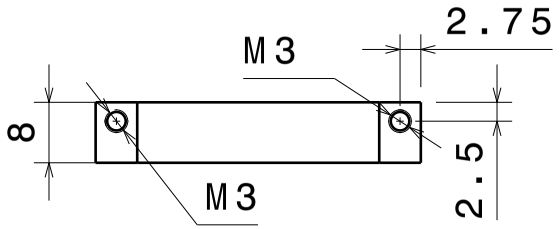


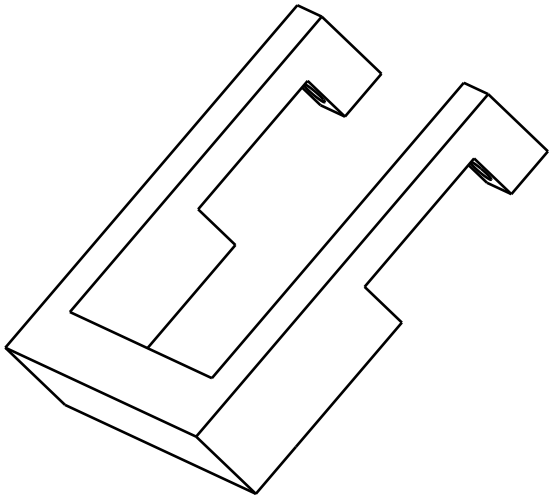
Section view A-A











Material: Alumínio.

Quantidade: 2 unidades.

Projecto Humanoide
Universidade de Aveiro

DRAWN BY: Pedro Cruz	DATE: 16-04-2012	DRAWING TITLE			
CHECKED BY: XXX	DATE: XXX	SIZE: A4	DRAWING NUMBER: fixador_rotacao_coxa		REV: X
DESIGNED BY: XXX	DATE: XXX	SCALE: 1:1	WEIGHT(kg): 0,01		SHEET: 1/1

Appendix C

Software Notes

C.1 Computer Hardware Configuration

This work was conducted on a Hewlett-Packard Compaq nx9420 notebook with the following specifications:

- **Hardware**
  - Intel Corp. T2400 @1.83GHz processor (32 bits)
  - 4GB (2+2) SODIMM DDR2 Synchronous 667 MHz RAM memory
  - ATI Technologies Radeon Mobility X1600 graphics card
  - Fujitsu MJA2500B 465GiB (500GB) SATA hard drive
  - Texas Instruments PCIxx12 OHCI Compliant IEEE-1394 Host Controller (internal)
- **System**
  - Ubuntu Linux version 10.04 LTS
  - Linux kernel version 2.6.32-43-generic-pae
  - ROS Fuerte version

C.2 PHANToM Device Drivers and OpenHaptics

Installation of the PHANToM device software is hereby described for future reference.

Note: As Ubuntu is a Debian-based Linux distribution, software installation files come frequently in the form of *.deb* packages. Other Linux distributions have similar packages, but the instructions for installation and setup may differ. Beware of this fact beforehand, for this text is meant for Debian-like systems.

Also, system component dependencies, as the *libc* libraries, the *gcc* and *g++* compilers or the *raw1394 (FireWire)* module libraries, will not be explicitly requested to be updated on automatic configuration scripts. However, keep in mind that distribution-dependent repositories not always provide the latest version for system components, even non kernel-dependent software. Core components update may be recommended to advanced users to improve execution speeds and memory management.

C.2.1 SensAble Software Pre-requisites

- **SensAble Developer Support Center (DSC) access credentials**:

  Place a request for credential generation at the SensAble Developer Support Center website (`dsc.sensable.com`). As the software components are proprietary (libraries and drivers), these credentials will be needed to download all the software and documentation, as well to give access to the online discussion forum.

- **Necessary libraries**:

  According to the respective Linux distribution, install all the dependencies and requisites mentioned on the file "HW_userguide_Linux.pdf". As stated there, all the dependencies must be satisfied before attempting to install the PHANToM drivers.

- **Mesa3D libraries**:

  In the previous *.pdf* file, the MesaLib7.4.2 library appears as a pre-requisite. However, this version was flagged on Ubuntu 10.04 for its numerous bugs, so the installed version was 7.5.2. This version is recommended, along with the MesaGLUT associated package, and is available at `www.mesa3d.org`.

  For a better system integration, the library should be compiled as follows (*sudo* is optional if the working subdirectory is within the */home* directory):

  1. Extract the downloaded file and run the package configuration in a console:
     sudo ./configure --enable-motif
     (The configuration report detects both critical and optional dependencies. If any critical library fails, the Linux distribution's repositories should have a version available. The *motif* package linking may not be required for every system.)
  2. Compile the libraries with the command:
     sudo make
  3. Install with the command:
     sudo make install
  4. Check if the location of installed libraries is in directory */usr/lib*:
     sudo locate libGlw
     sudo locate libGLUT

C.2.2 SensAble Software Installation

The installation of the OpenHaptics v3.0 suite must only be executed after the correct installation of the PHANToM drivers. To install both, navigate to the directory of the *.deb* files (or extract the compressed version) downloaded and run consecutively in a console ("*" represents the version downloaded):

sudo dpkg -i phantomdevicedrivers_*.-*_i386.deb*

sudo dpkg -i openhaptics_*.-*_i386.deb*

Alternatively, a *GUI* application for the *.deb* package manager *dpkg* can also be used. But installing the *.deb* file through the GUI may not work. If running amd64-based systems or other 64bit system, *dpkg* requires a different argument list, as exemplified:

dpkg -i –force-architecture /path/to/file.deb

After the installation is complete, device communication must be tested. These libraries are equipped with two small applications, installed by default with the drivers. The *PHANToM-Configuration* is an application for managing the connected hardware, defining the default device and allowing for dual-device setup configurations. The *PHANToMTest* is a small application designed to test the device's capabilities. This library, drivers, and applications were not designed for systems running in any other than the English language. For that reason, normal execution of these programs, via command line calls, is not possible. The only way these applications can run is by calling their configuration routines, which are no other than scripts that change the environment language just for the program's execution.

The configuration application is called by running:

sudo /usr/sbin/runPHANToMConfiguration

Select the correct PHANToM device and press "OK". Next, to test the device:

sudo /usr/sbin/runPHANToMTest

The PHANToM drivers do not have access to the FireWire communication port object, so it falls to the user the responsibility to probe it and change the access rules. To workaround these issues, a small *shell* file was created with the lines:

```
#!/bin/bash
modprobe raw1394
chmod 777 /dev/raw1394
```

This script is to be run before any call for device applications. Alternatively, these commands can be added to the *shell*'s startup script or to the system's startup routines.

The OpenHaptics toolkit deploys compilable examples to demonstrate both the device and the libraries capabilities. If correctly installed, the compilable source code should be installed on the directory */usr/share/3DTouch/examples*. All the provided documentation is also installed in the hard-drive on the directory */usr/share/3DTouch/doc*.

Note: Some examples require the creation of an environment variable *3DTOUCH_BASE* that is supposed to point at a directory. However, the Ubuntu Linux *bash* does not allow to initiate an environment variable started by a number. The most common workaround for this case is to use the *tcsh shell*, an old *C-shell* still supported on Linux, and run the following commands:

sudo tcsh
sudo setenv 3DTOUCH_BASE /usr/share/3DTouch

The examples can be run inside the *tcsh* terminal, or outside back in the *bash shell*, exiting with the command "*exit*".

C.3 Adicional Software

C.3.1 Robot Operating System (ROS)

In this work, the *ROS* version installed was *Fuerte*. The installable Linux version is system dependent, but the detailed instructions can be found on `http://www.ros.org`. Here it can also be found tutorials, code examples, discussion forums, etc...

C.3.2 *Eigen* Library

The *Eigen* library is a C++ template library for linear algebra, with matrices, vectors, numerical solvers, and other related algorithms, and is provided together with *ROS*. Should it be needed for another purpose, the compilable source code can be found on `http://eigen.tuxfamily.org`.

C.3.3 *Armadillo* Library

Armadillo is an open-source *C++* linear algebra library with a syntax that is deliberately similar to Matlab. It supports integer, floating point and complex numbers, and has a built trigonometric and statistics functions. Various matrix decompositions are provided through optional integration with *LAPACK*, or one of its high performance drop-in replacements (such as the multi-threaded *MKL* or *ACML* libraries). Primarily developed at NICTA (Australia), it has contributions from around the world.

To install the library, instructions are as follows (*sudo* is optional if the working subdirectory is within the */home* directory):

1. Install *ROS* as detailed above. This will install a series of dependencies that will be needed to compile this library.
2. Install CMake:

   sudo apt-get install cmake
3. Install BLAS, LAPACK and ATLAS libraries:

   sudo apt-get install libblas-dev

   sudo apt-get install liblapack-dev

   sudo apt-get install libatlas-base-dev libatlas-headers
4. Download and unpack the source code from `http://arma.sourceforge.net`.
5. Open a command line and change into the directory that was created by unpacking the downloaded archive:

   sudo cmake .

   (CMake should detect all necessary dependencies and warn if any fails.)
6. Compile the library with the command:

   sudo make
7. Install with the command:

   sudo make install